

MAI4CAREU

Master programmes in Artificial
Intelligence 4 Careers in Europe



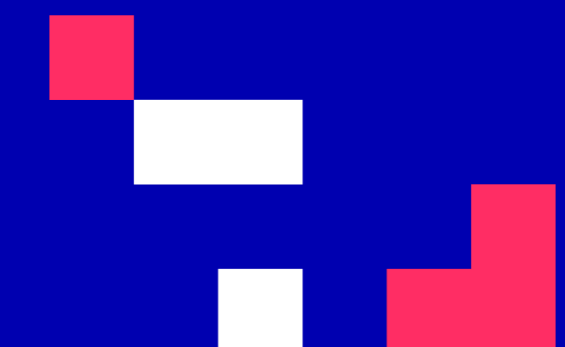
University
of Cyprus

University of Cyprus

MAI645 - Machine Learning for Graphics and Computer Vision

Andreas Aristidou, PhD

Spring Semester 2023



Video Understanding, Generative Models, & Self-Supervised Learning

These notes are based on the work of Fei-Fei Li, Jiajun Wu, Ruohan Gao,
CS231 - Deep Learning for Computer Vision

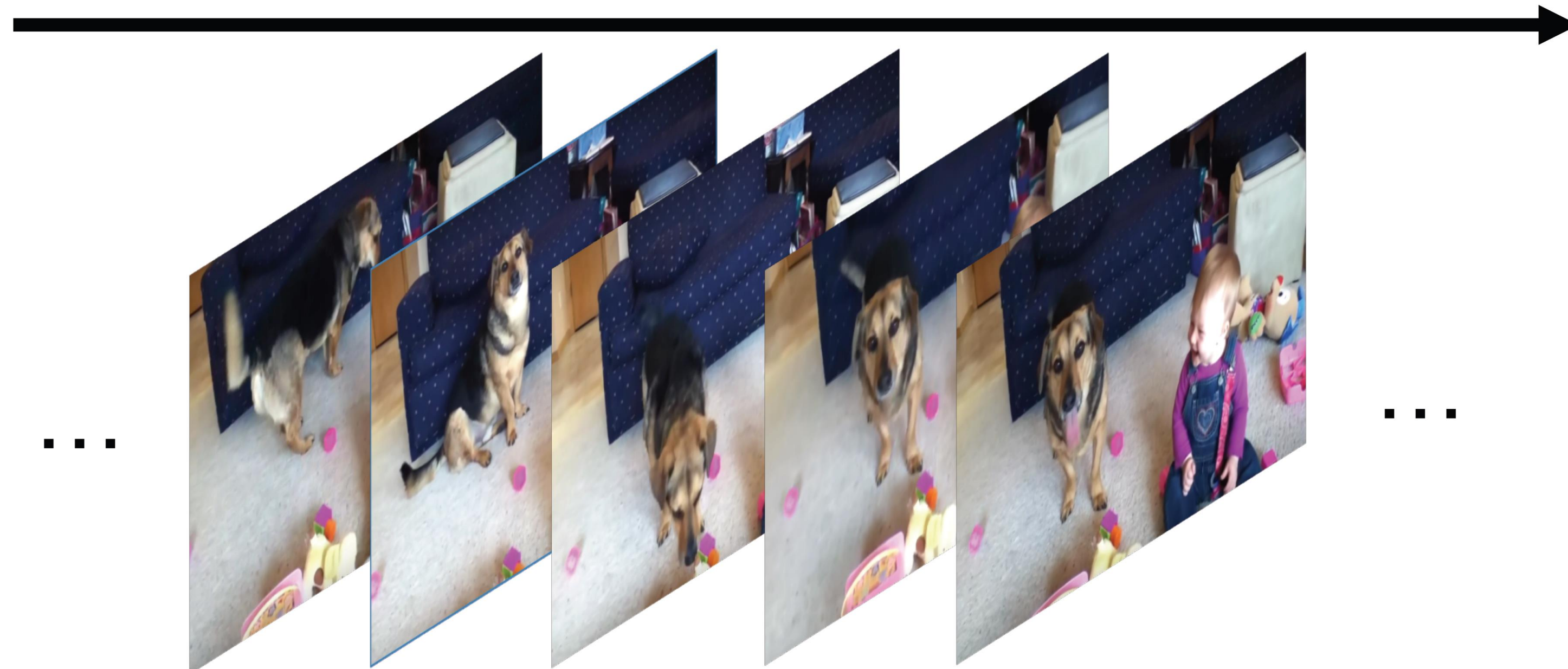


Video Understanding

A video is a sequence of images

4D tensor: $T \times 3 \times H \times W$

(or $3 \times T \times H \times W$)



Video Understanding



Images: Recognize **objects**



Dog
Cat
Fish
Truck



Videos: Recognize **actions**



Swimming
Running
Jumping
Eating
Standing

Video Understanding

Problem: Videos are big!



Input video:
 $T \times 3 \times H \times W$

Videos are ~30 frames per second (fps)

Size of uncompressed video

(3 bytes per pixel):

SD (640 x 480): ~**1.5 GB per minute**

HD (1920 x 1080): ~**10 GB per minute**

Solution: Train on short clips:

low fps and low spatial resolution e.g. $T = 16$, $H=W=112$ (3.2 seconds at 5 fps, 588 KB)

Video Understanding: *Training on Clips*

Raw video: Long, high FPS



Training: Train model to classify short **clips** with low FPS



Testing: Run model on different clips, average predictions

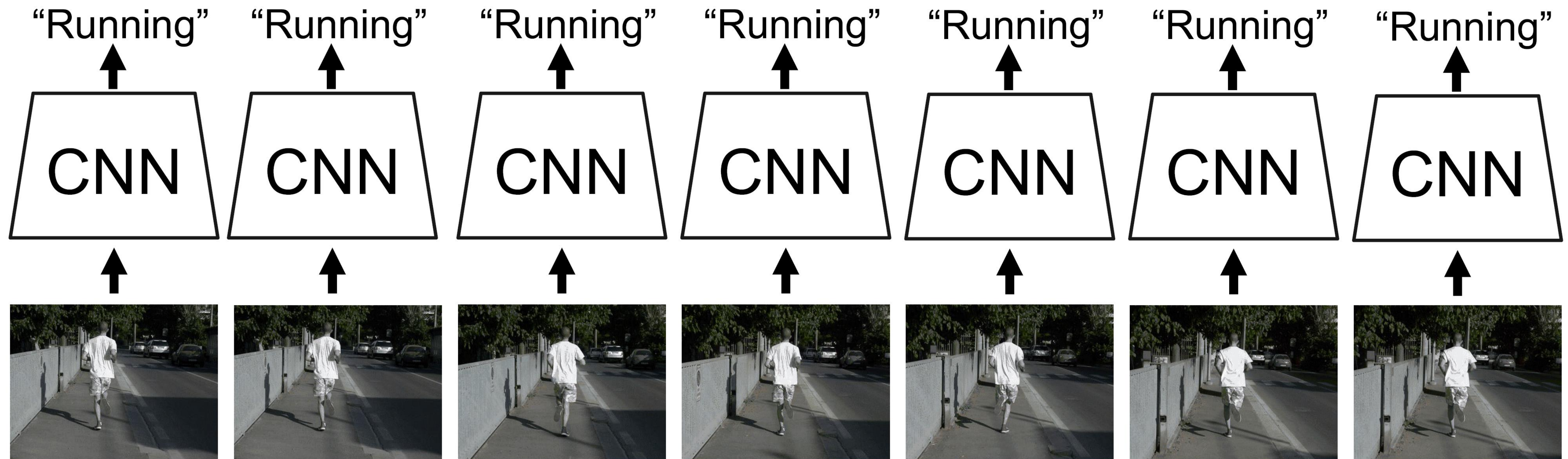


Video Classification: *Single-Frame CNN*

Simple idea: train normal 2D CNN to classify video frames independently!

(Average predicted probs at test-time)

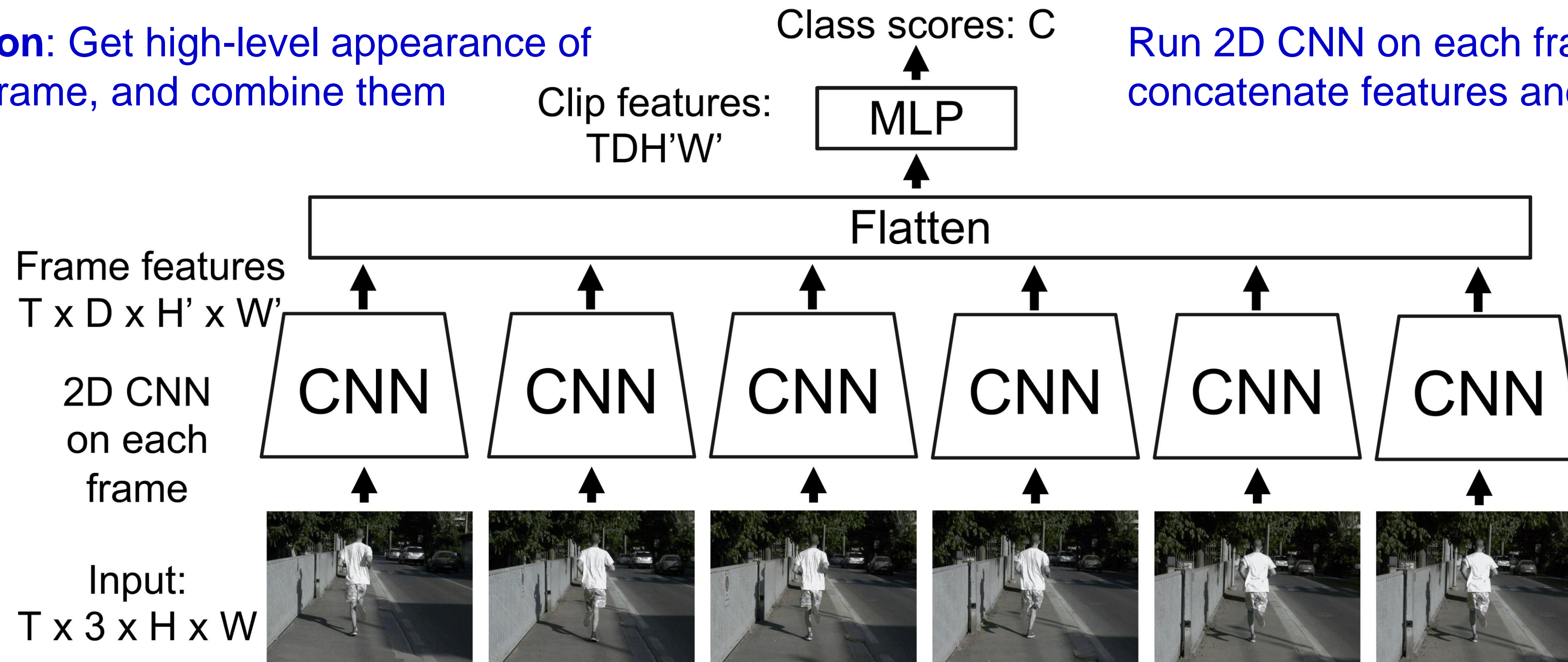
Often a **very** strong baseline for video classification



Video Classification: *Late Fusion (with FC layers)*

Intuition: Get high-level appearance of each frame, and combine them

Run 2D CNN on each frame, concatenate features and feed to MLP

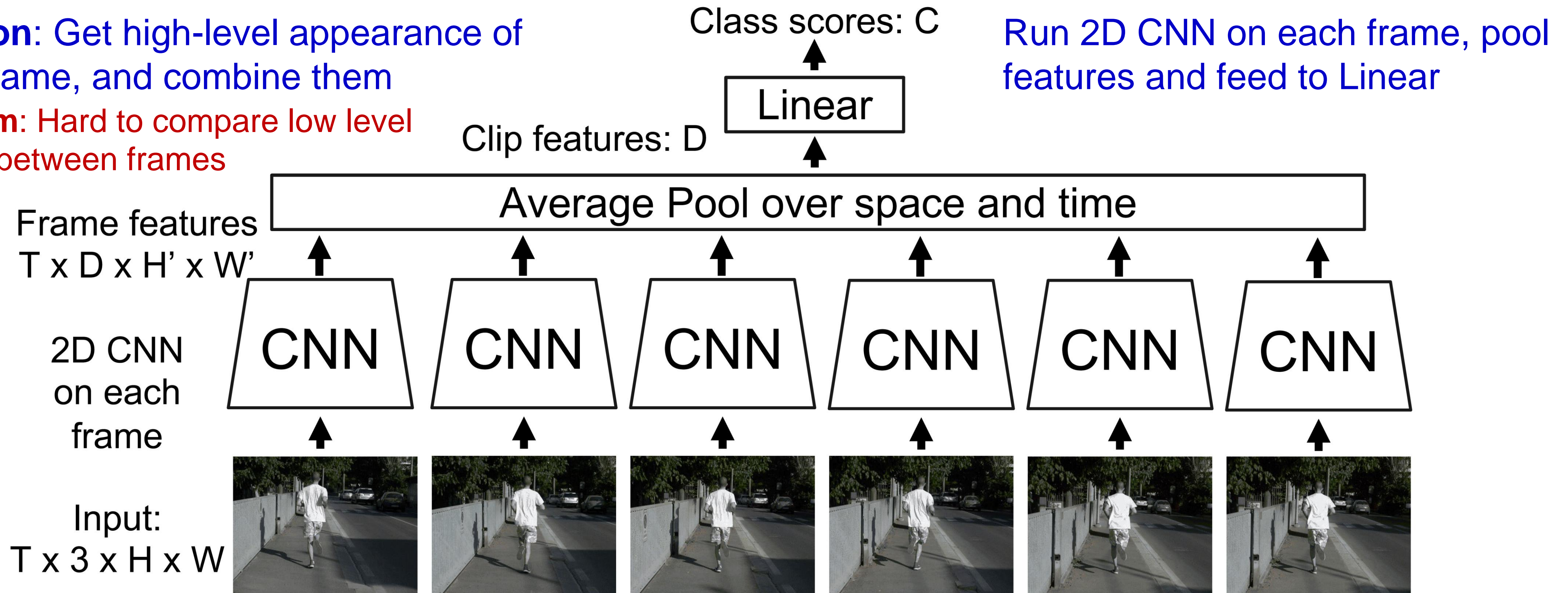


Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: *Late Fusion (with pooling)*

Intuition: Get high-level appearance of each frame, and combine them

Problem: Hard to compare low level motion between frames



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: *Early Fusion*

Intuition: Compare frames with very first conv layer, after that normal 2D CNN

Problem: One layer of temporal processing may not be enough!

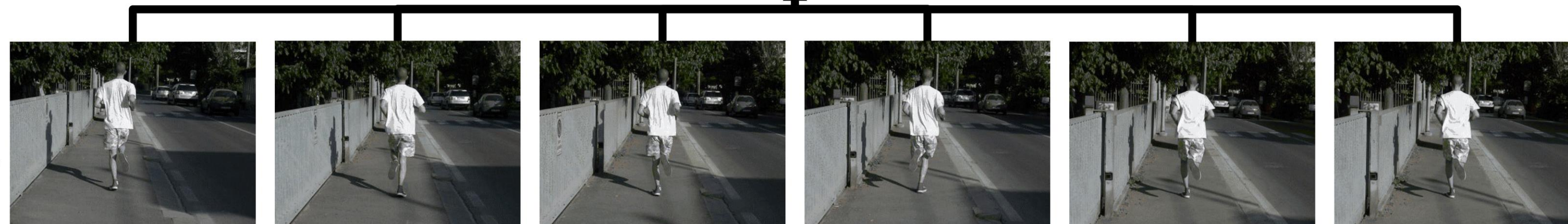
First 2D convolution collapses all temporal information:

Input: $3T \times H \times W$

Output: $D \times H \times W$

Reshape:
 $3T \times H \times W$

Input:
 $T \times 3 \times H \times W$



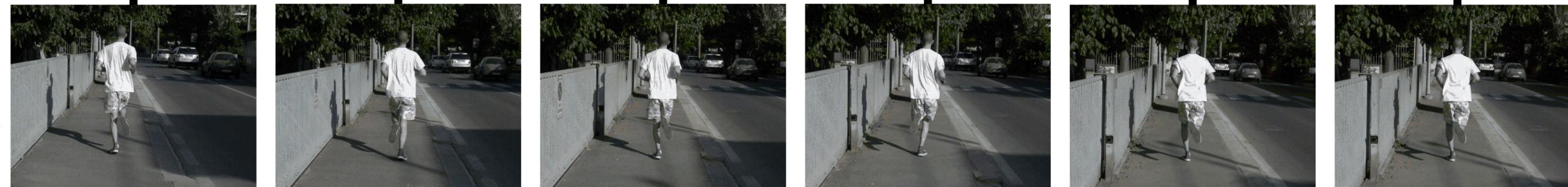
Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: 3D CNN

Intuition: Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

Each layer in the network is a 4D tensor: $D \times T \times H \times W$
Use 3D conv and 3D pooling operations

Input:
 $3 \times T \times H \times W$



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Video Classification: 3D CNN

Early Fusion vs Late Fusion vs 3D CNN

	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Late Fusion	Input	3 x 20 x 64 x 64	
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64
Early Fusion	Input	3 x 20 x 64 x 64	
	Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64
3D CNN	Input	3 x 20 x 64 x 64	
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

What is the difference?

Build slowly in space,
All-at-once in time at end

Build slowly in space,
All-at-once in time at star

Build slowly in space,
Build slowly in time
"Slow Fusion"

(Small example architectures, in practice much bigger)



C3D: *The VGG of 3D CNNs*

3D CNN that uses all 3x3x3 conv and 2x2x2 pooling (except Pool1 which is 1x2x2)

Released model pretrained on Sports-1M: Many people used this as a video feature extractor

3D CNN that uses all 3x3x3 conv and 2x2x2 pooling (except Pool1 which is 1x2x2)

Problem: 3x3x3 conv is very expensive!

AlexNet: 0.7 GFLOP

VGG-16: 13.6 GFLOP

C3D: 39.5 GFLOP (2.9x VGG!)

Tran et al, "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015

Measuring Motion: *Optical Flow*

Image at frame t

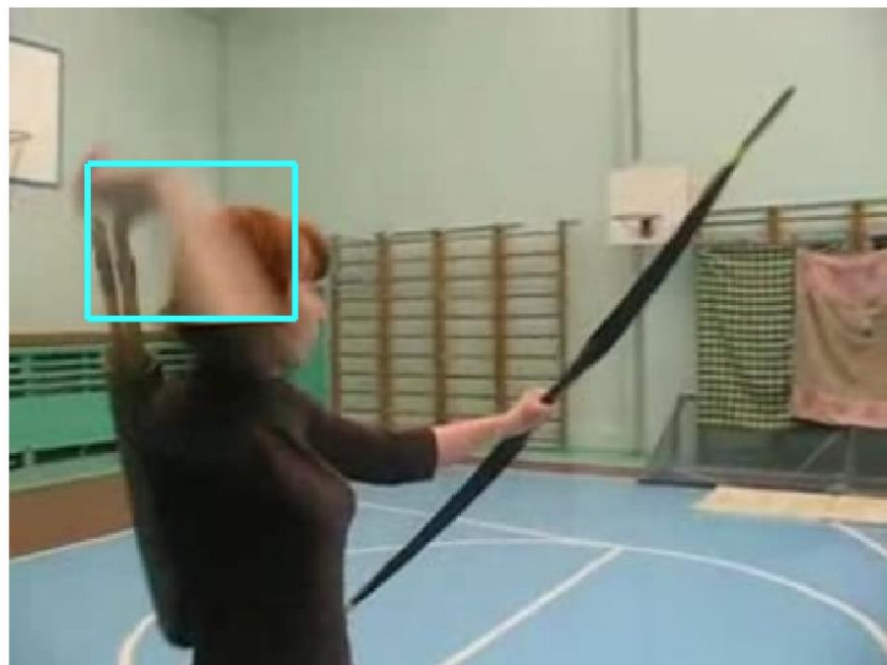
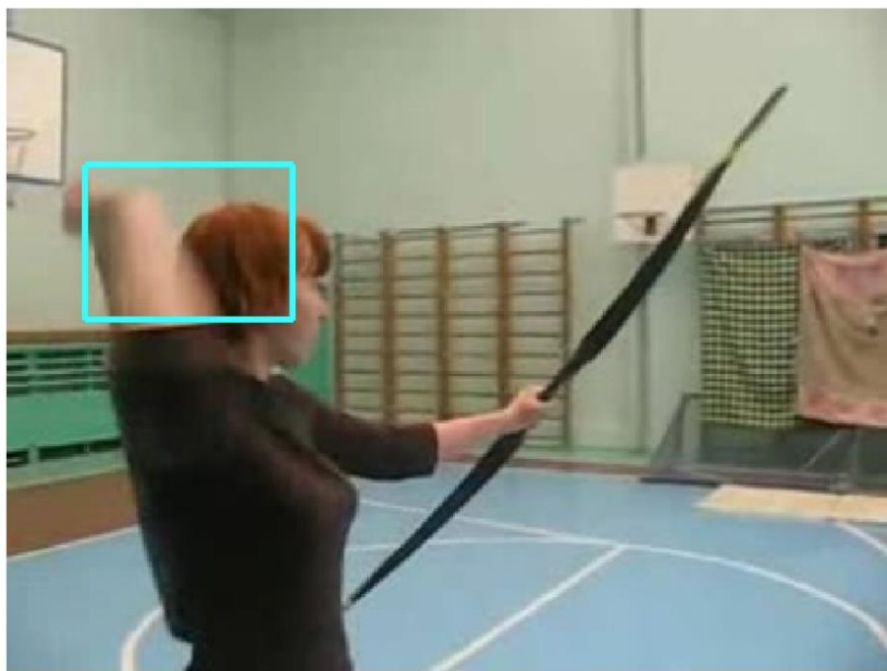
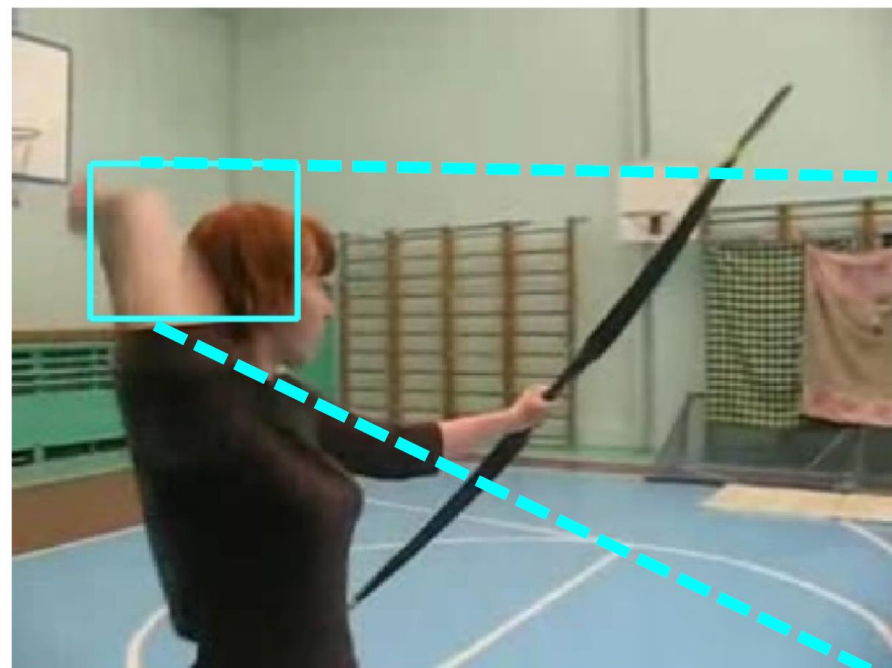


Image at frame $t+1$

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Measuring Motion: *Optical Flow*

Image at frame t



Optical flow gives a displacement field F between images I_t and I_{t+1}

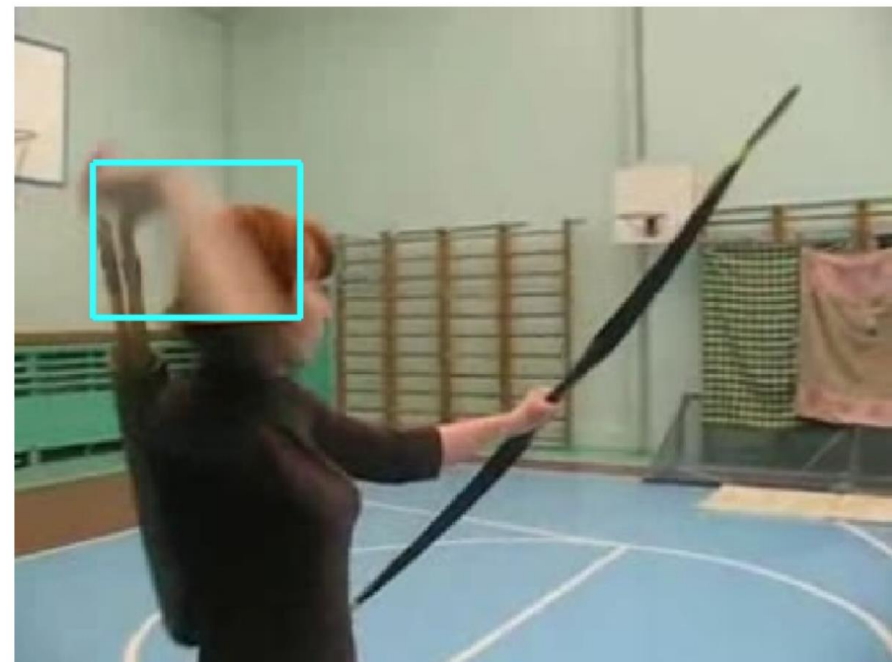
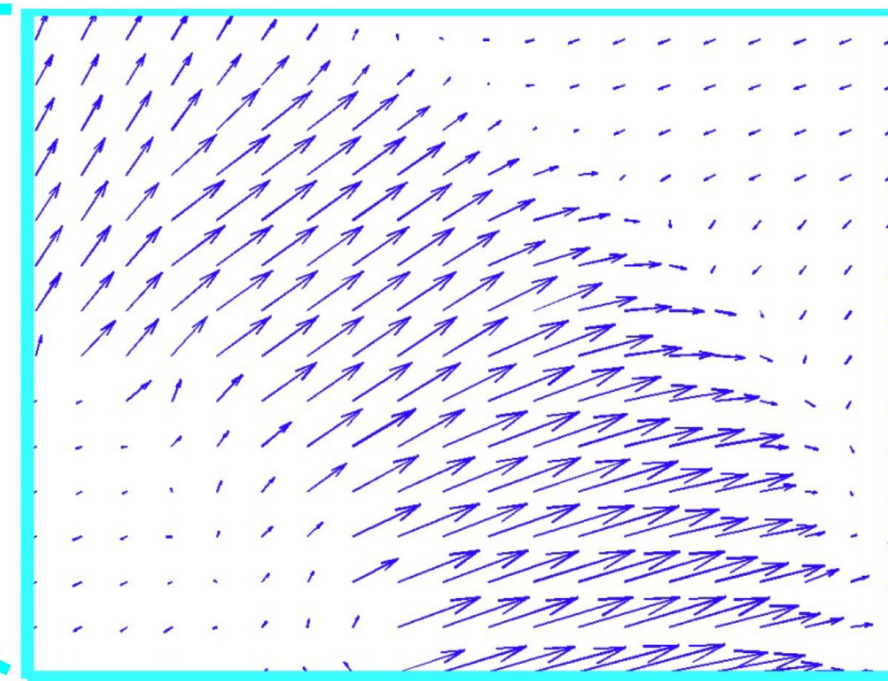
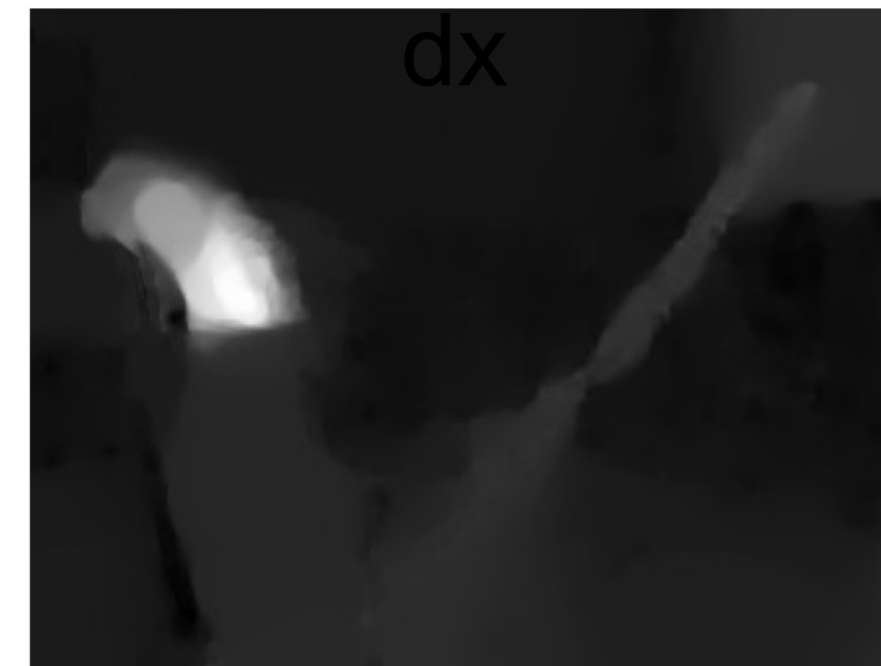


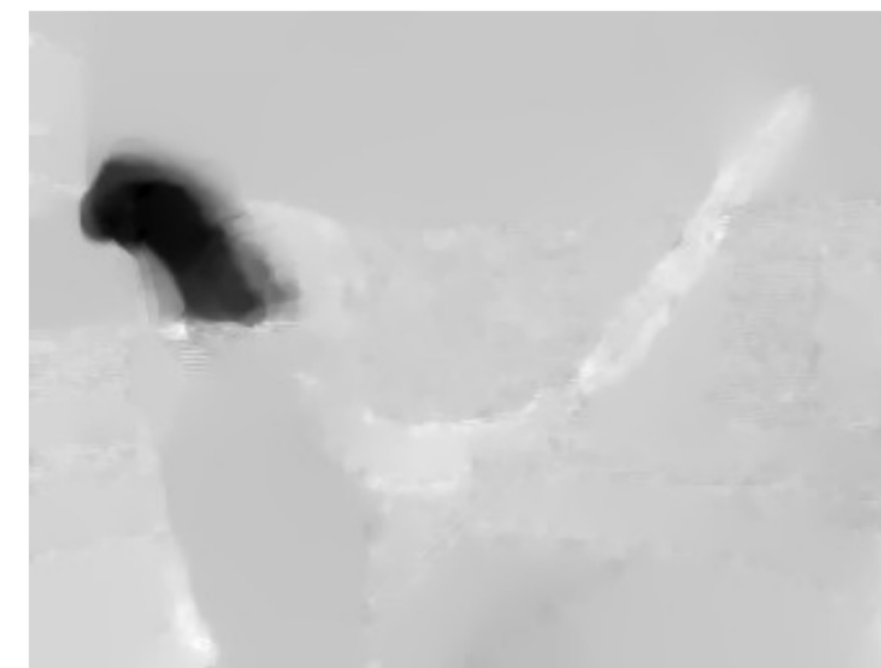
Image at frame t+1

Tells where each pixel will move in the next frame:
 $F(x, y) = (dx, dy)$
 $I_{t+1}(x+dx, y+dy) = I_t(x, y)$

Horizontal flow



Optical Flow highlights **local motion**



Vertical Flow dy

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

More on Video understanding

Modeling long-term temporal structure: So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?

Process local features using recurrent network (e.g. LSTM) - Many to one or Many to many

Sometimes don't backprop to CNN to save memory; pretrain and use it as a feature extractor

Problem: RNNs are slow for long sequences (can't be parallelized)

Spatio-Temporal Self-Attention

- Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014
- Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
- Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015
- Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016
- Wang et al, "Non-local neural networks", CVPR 2018

Temporal Action Localization

Given a long untrimmed video sequence, identify frames corresponding to different actions

Running



Jumping

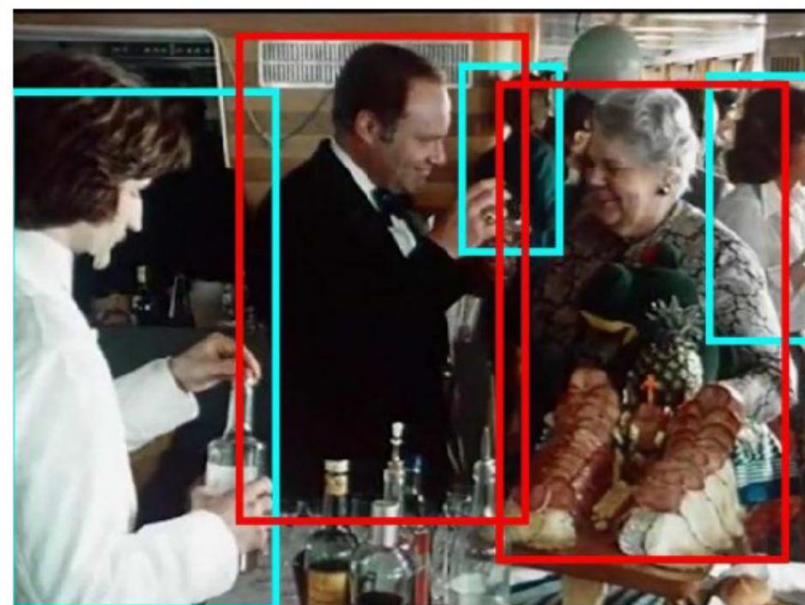


Can use architecture similar to Faster R-CNN: first generate temporal proposals then classify

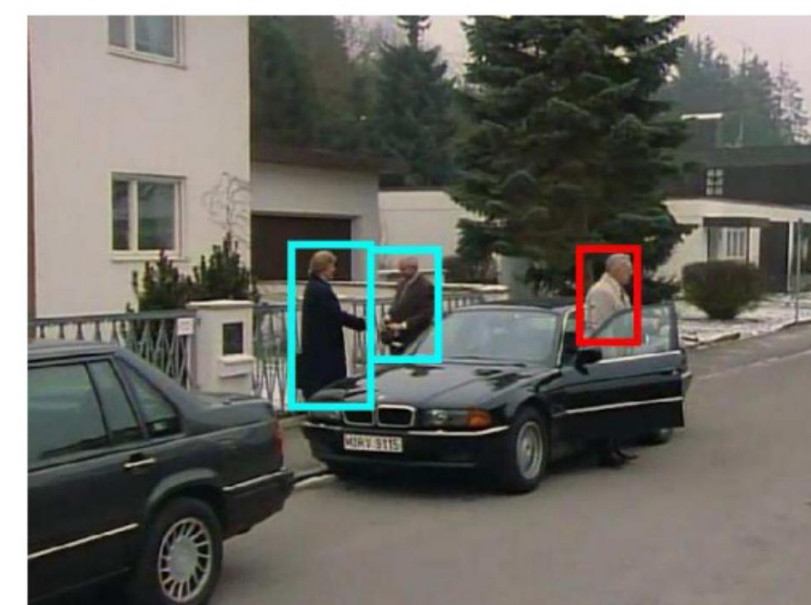
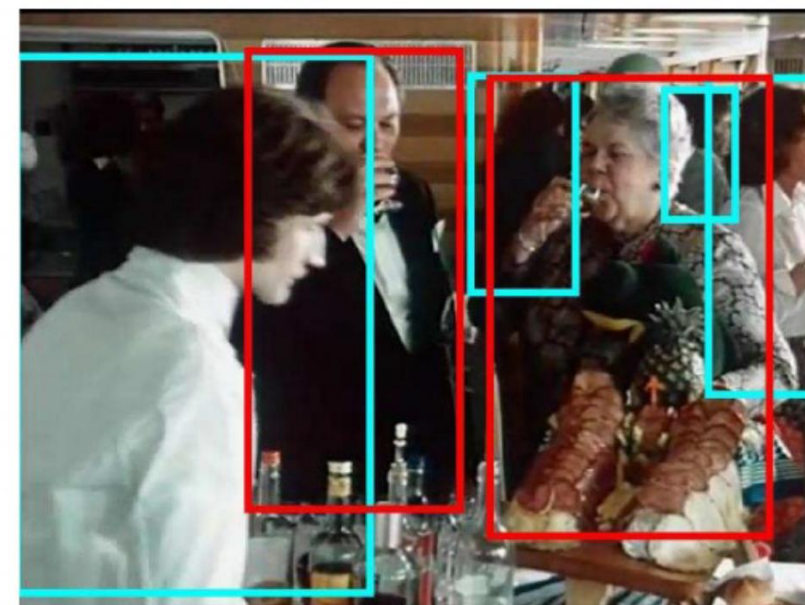
Chao et al, "Rethinking the Faster R-CNN Architecture for Temporal Action Localization", CVPR 2018

Temporal Action Localization

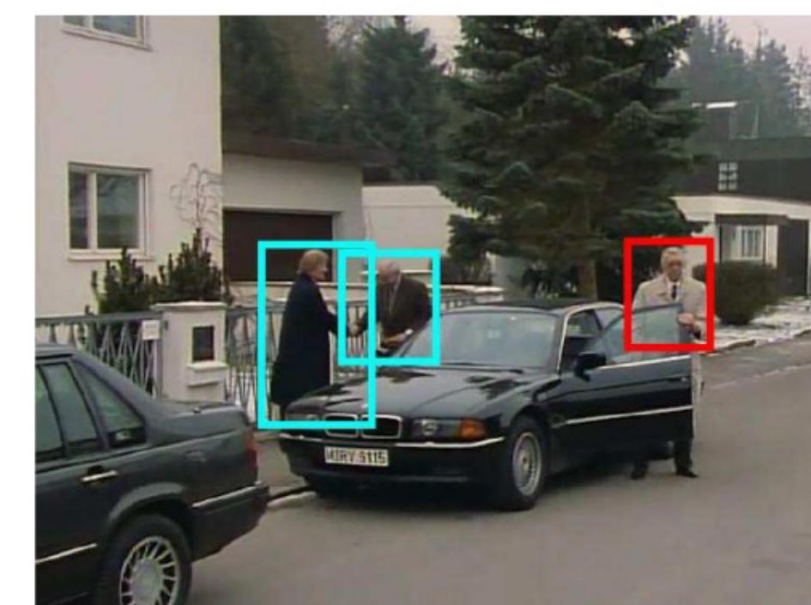
Given a long untrimmed video, detect all the people in both space and time and classify the activities they are performing.



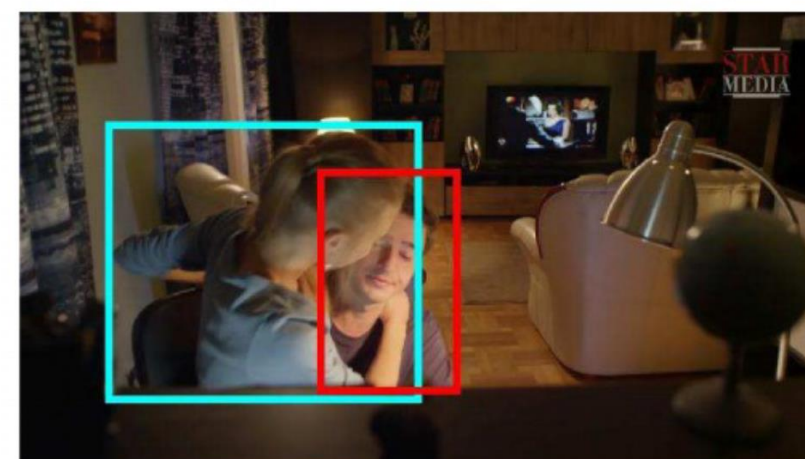
clink glass → drink



open → close



grab (a person) → hug



look at phone → answer phone



Gu et al, "AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions", CVPR 2018



Generative Models



Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

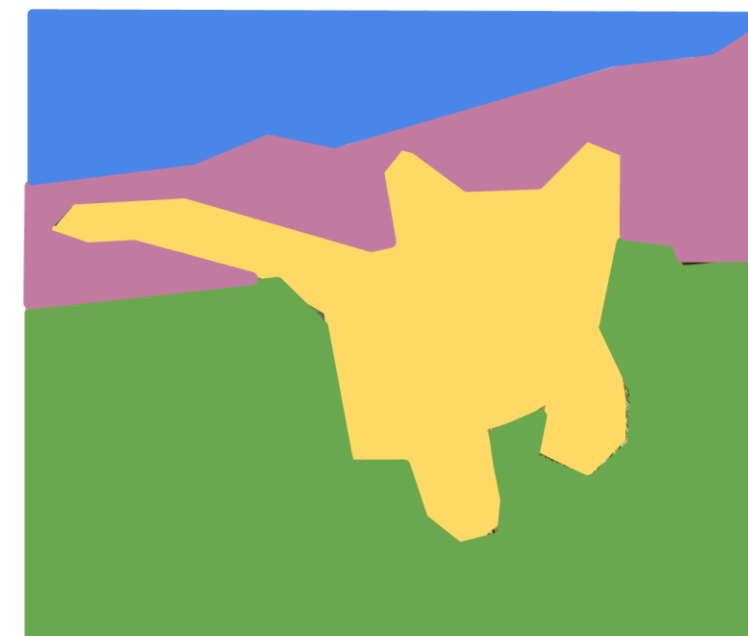
Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.



→ Cat

Classification



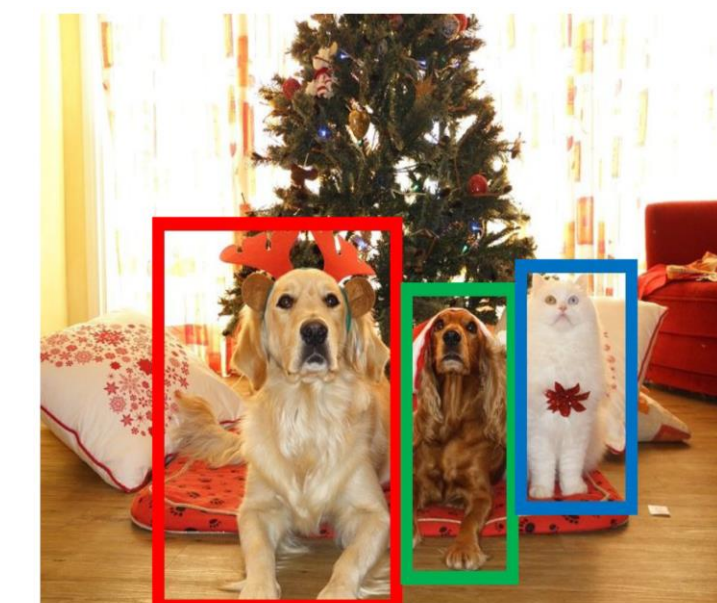
GRASS, CAT,
TREE, SKY

Semantic Segmentation



A cat sitting on a suitcase on the floor

Image captioning



DOG, DOG, CAT

Object Detection

Supervised vs Unsupervised Learning

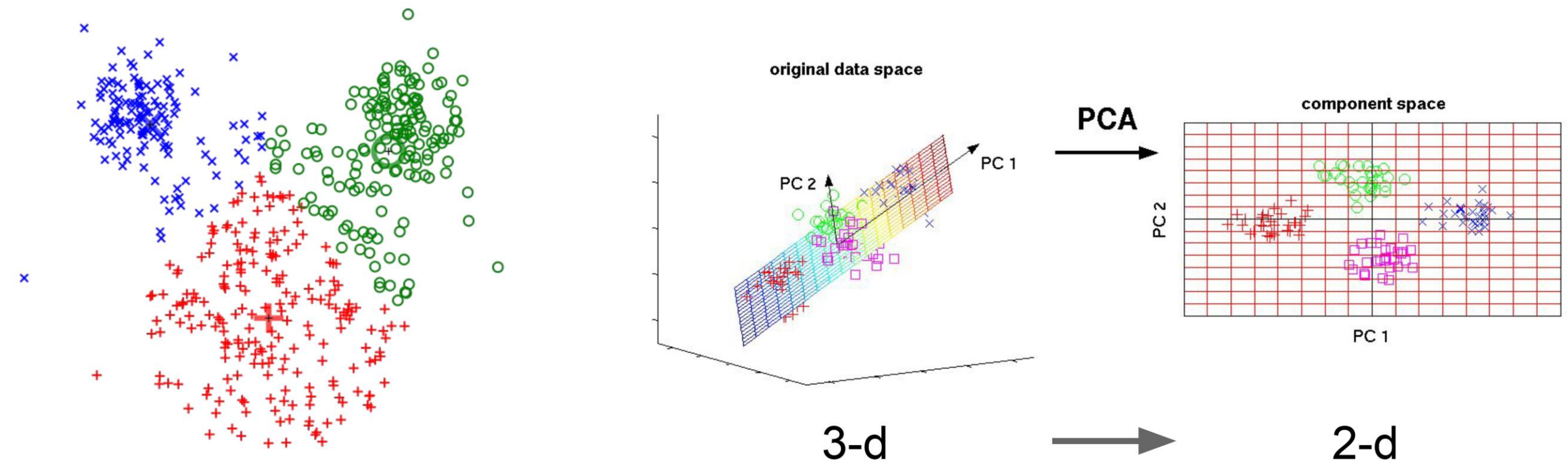
Unsupervised Learning

Data: X

Just data, no labels

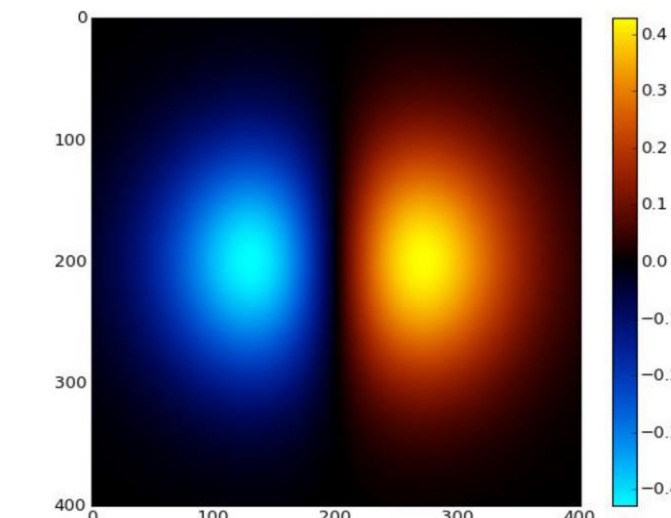
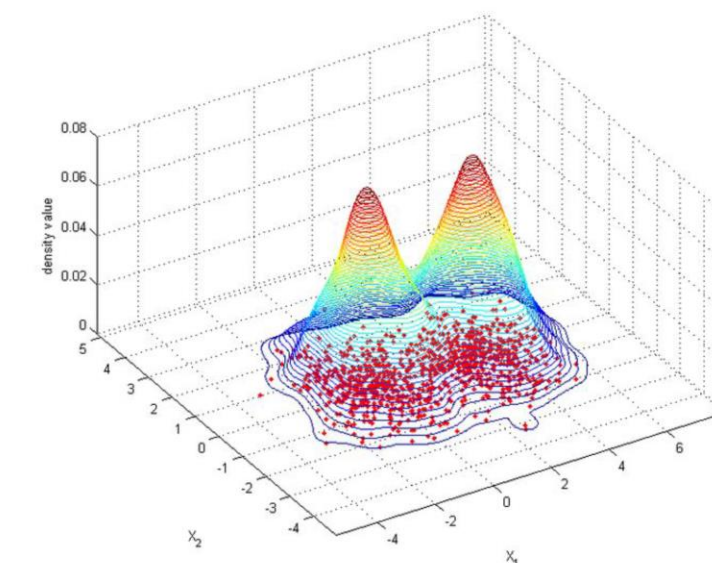
Goal: Learn some underlying hidden structure of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



K-means clustering

Principal Component Analysis (Dimensionality reduction)



2-d density estimation

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

Data: x

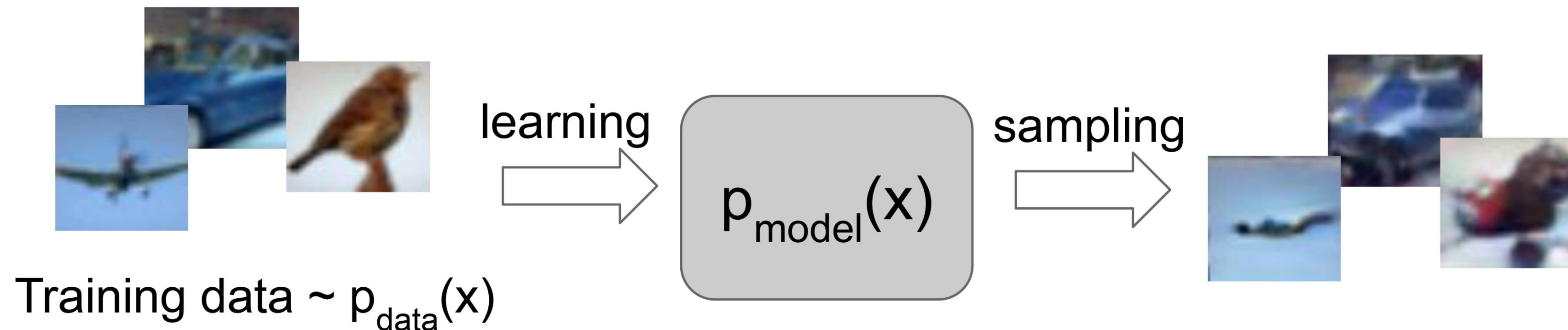
Just data, no labels

Goal: Learn some underlying hidden structure of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Generative Modeling

Generative models: Given training data, generate new samples from same distribution.



Objectives:

1. Learn $p_{\text{model}}(x)$ that approximates $p_{\text{data}}(x)$
2. **Sampling new x from $p_{\text{model}}(x)$**

Formulate as density estimation problems:

- **Explicit density estimation:** explicitly define and solve for $p_{\text{model}}(x)$
- **Implicit density estimation:** learn model that can sample from $p_{\text{model}}(x)$ **without explicitly defining it.**

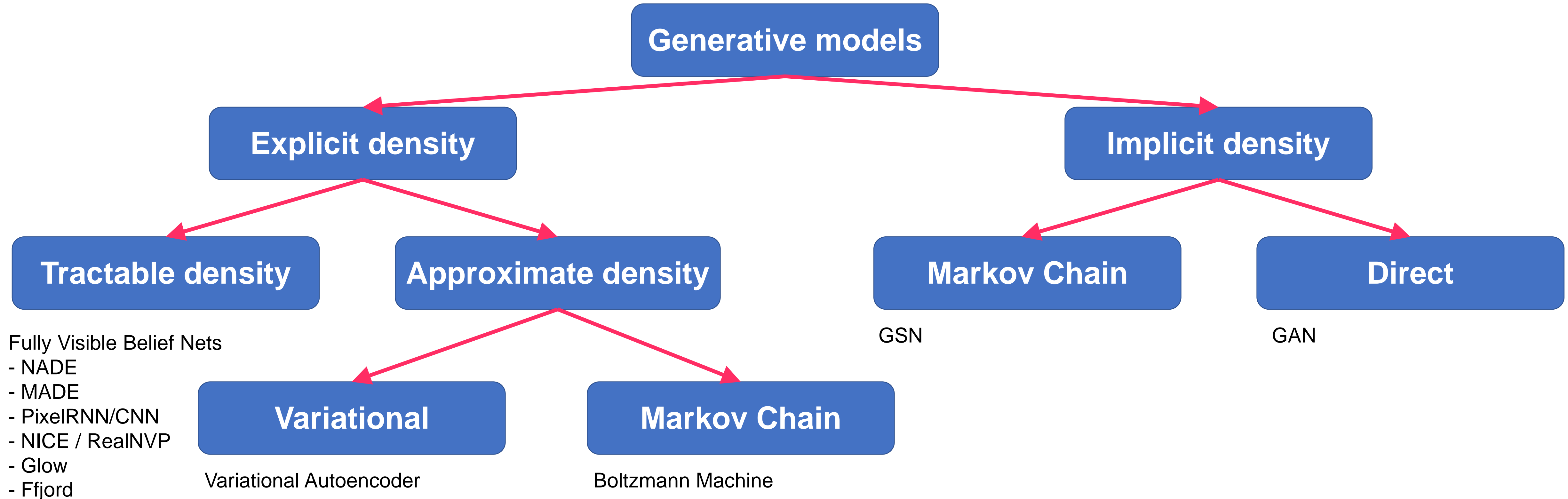
Generative Modeling

Generative models are a class of machine learning models that are designed to generate new data samples that are similar to a given set of training data. The **objective** of these models is to learn the underlying probability distribution of the training data, so that they can generate new samples that are statistically similar to the training data. This is useful for a wide range of applications, including image and text generation, data augmentation, and anomaly detection.

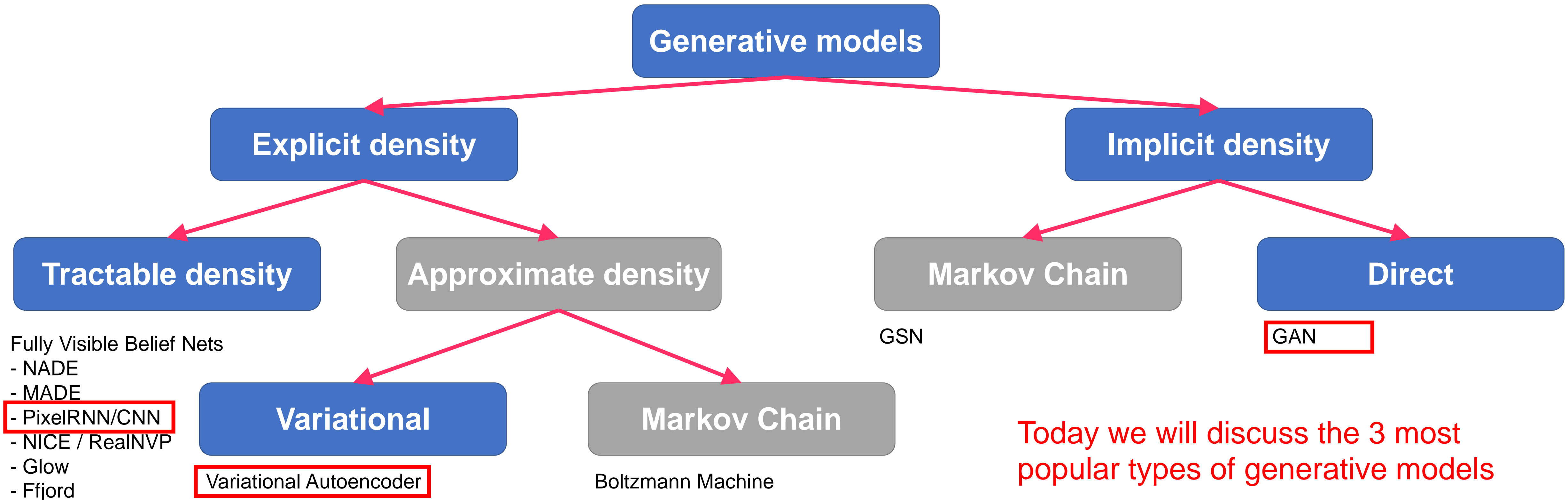
Generative models work by modeling the probability distribution of the training data. They learn to generate new samples by sampling from this probability distribution. There are several types of generative models, including:

- **Autoregressive models:** These models generate data by modeling the conditional probability of each data point given the previous data points.
- **Variational Autoencoders (VAEs):** These models learn to encode data into a lower-dimensional latent space, and then generate new data by sampling from the latent space and decoding it back into the original space.
- **Generative Adversarial Networks (GANs):** These models consist of two neural networks: a generator network and a discriminator network. The generator network generates new data samples, while the discriminator network tries to distinguish between the generated samples and the real samples. The two networks are trained together in a game-like setting, where the generator tries to generate samples that fool the discriminator, and the discriminator tries to correctly classify the samples.
- **Boltzmann Machines:** These models are a type of probabilistic graphical model that learns the joint probability distribution of the training data. They can be used to generate new samples by sampling from the learned distribution.

Taxonomy of Generative Models



Taxonomy of Generative Models



Today we will discuss the 3 most popular types of generative models



Taxonomy of Generative Models

Explicit density models: An explicit density model, also known as a parametric model, specifies a mathematical formula or a set of parameters that directly define the probability density function (PDF) of the data. This means that given a set of parameters, the model can generate new data points with a high degree of accuracy. Examples of explicit density models include Gaussian mixture models and linear regression models.

Implicit density models: An implicit density model, also known as a non-parametric model, does not explicitly specify the PDF. Instead, it defines a generative process that can sample data points from the underlying distribution. This means that implicit density models can capture complex, multi-modal distributions that may be difficult to represent with explicit density models. Examples of implicit density models include generative adversarial networks (GANs) and variational autoencoders (VAEs).

PixelRNN and PixelCNN

PixelRNN (Pixel Recurrent Neural Network) is a type of autoregressive model that generates images by modeling the conditional probability of each pixel given the previous pixels. The model is typically trained on a dataset of images, and during training, it learns to predict the value of each pixel given the values of the previous pixels in the same row and column. The model is trained using maximum likelihood estimation, and the learned parameters are used to generate new images by sampling from the learned distribution.

PixelCNN (Pixel Convolutional Neural Network) is another type of autoregressive model that generates images by modeling the conditional probability of each pixel given the previous pixels. However, unlike PixelRNN, which uses recurrent neural networks to model the conditional probabilities, PixelCNN uses convolutional neural networks. Specifically, it uses masked convolutions to ensure that each pixel only depends on the previous pixels in the same row and column. Like PixelRNN, PixelCNN is trained using maximum likelihood estimation, and the learned parameters are used to generate new images by sampling from the learned distribution.

Both PixelRNN and PixelCNN have been shown to be effective at generating high-quality images, and they have been used in a variety of applications, including image synthesis, data compression, and image inpainting. However, they can be computationally expensive to train, especially on large datasets, and they may not scale well to high-resolution images.

PixelRNN and PixelCNN

Fully visible belief network (FVBN)

Explicit density model:

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

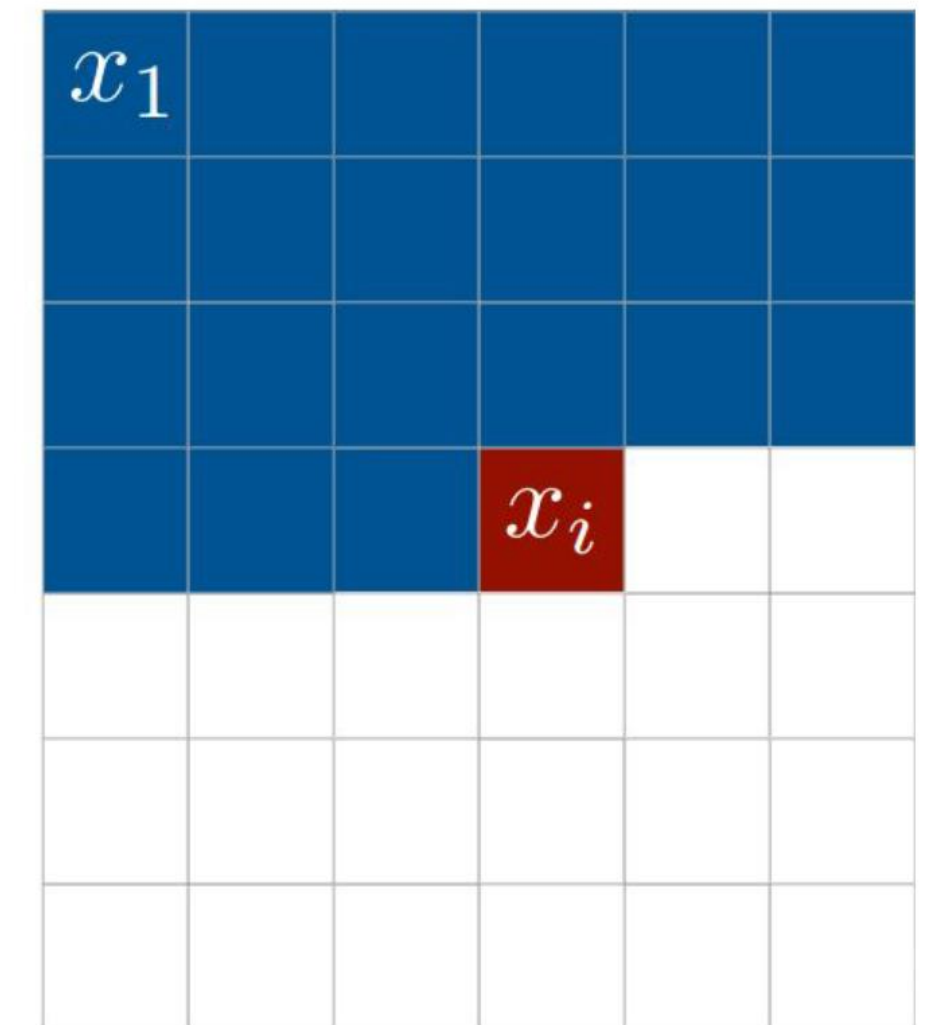
$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

↑

Likelihood of image x

↑

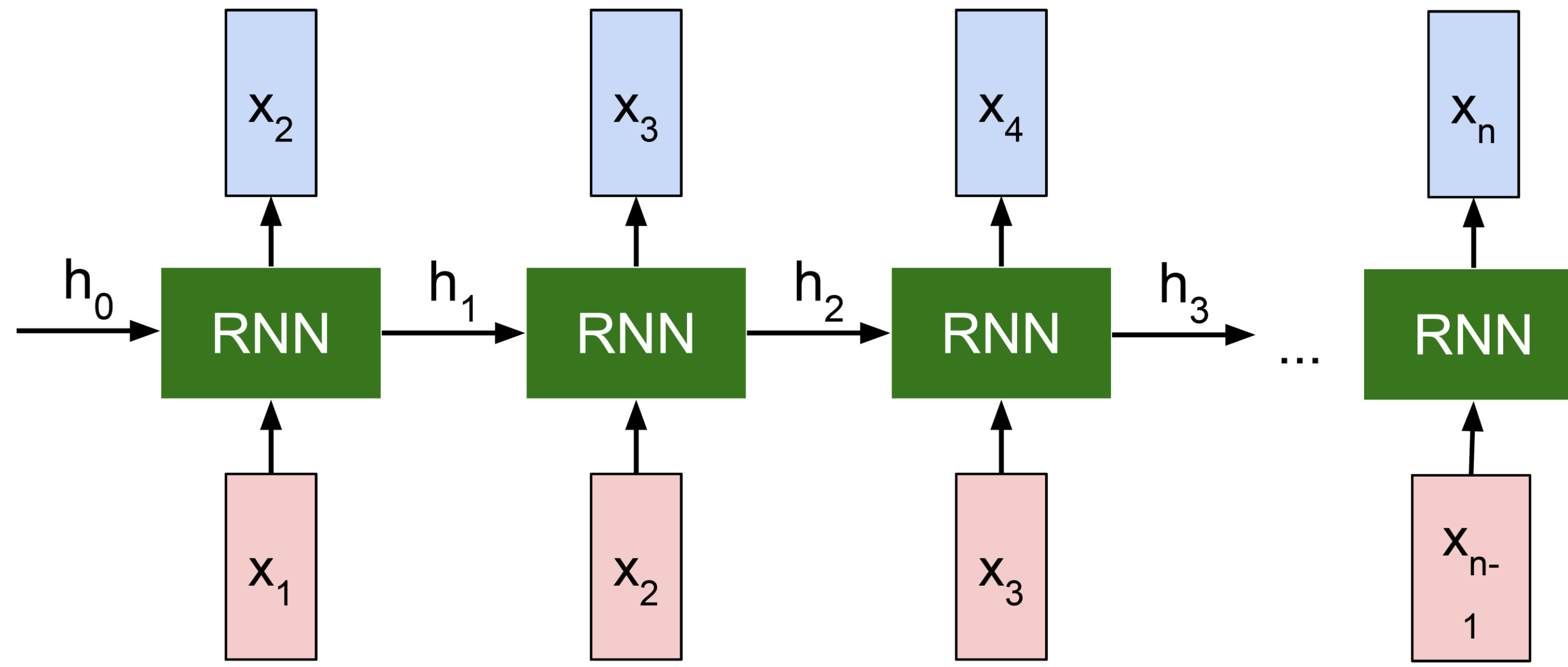
Probability of i 'th pixel value given all previous pixels



Then maximize likelihood of training data

Complex distribution over pixel values
=> Express using a neural network!

PixelRNN



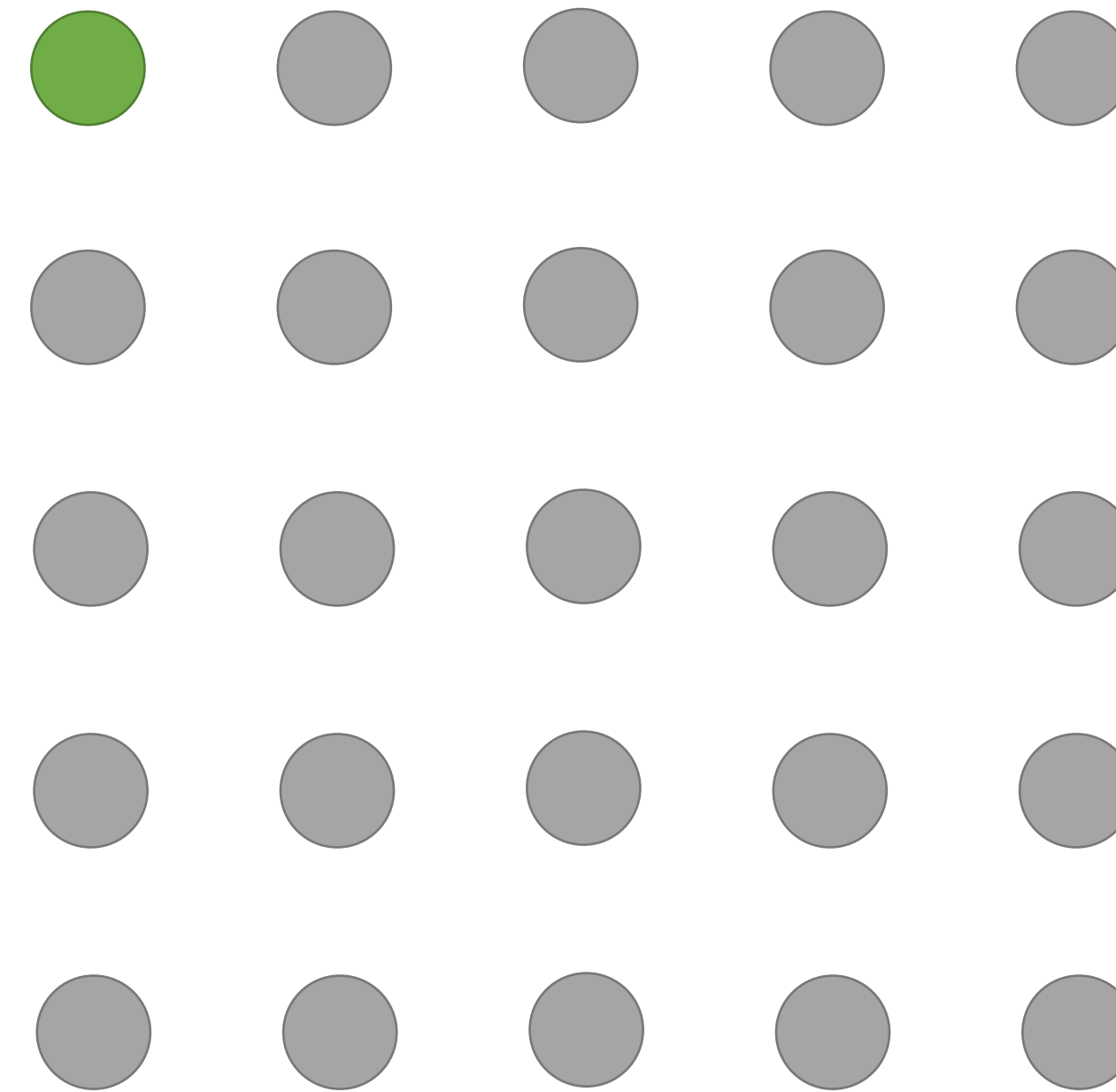
$$p(x_i | x_1, \dots, x_{i-1})$$

Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu 2016, Pixel Recurrent Neural Networks

PixelRNN

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)

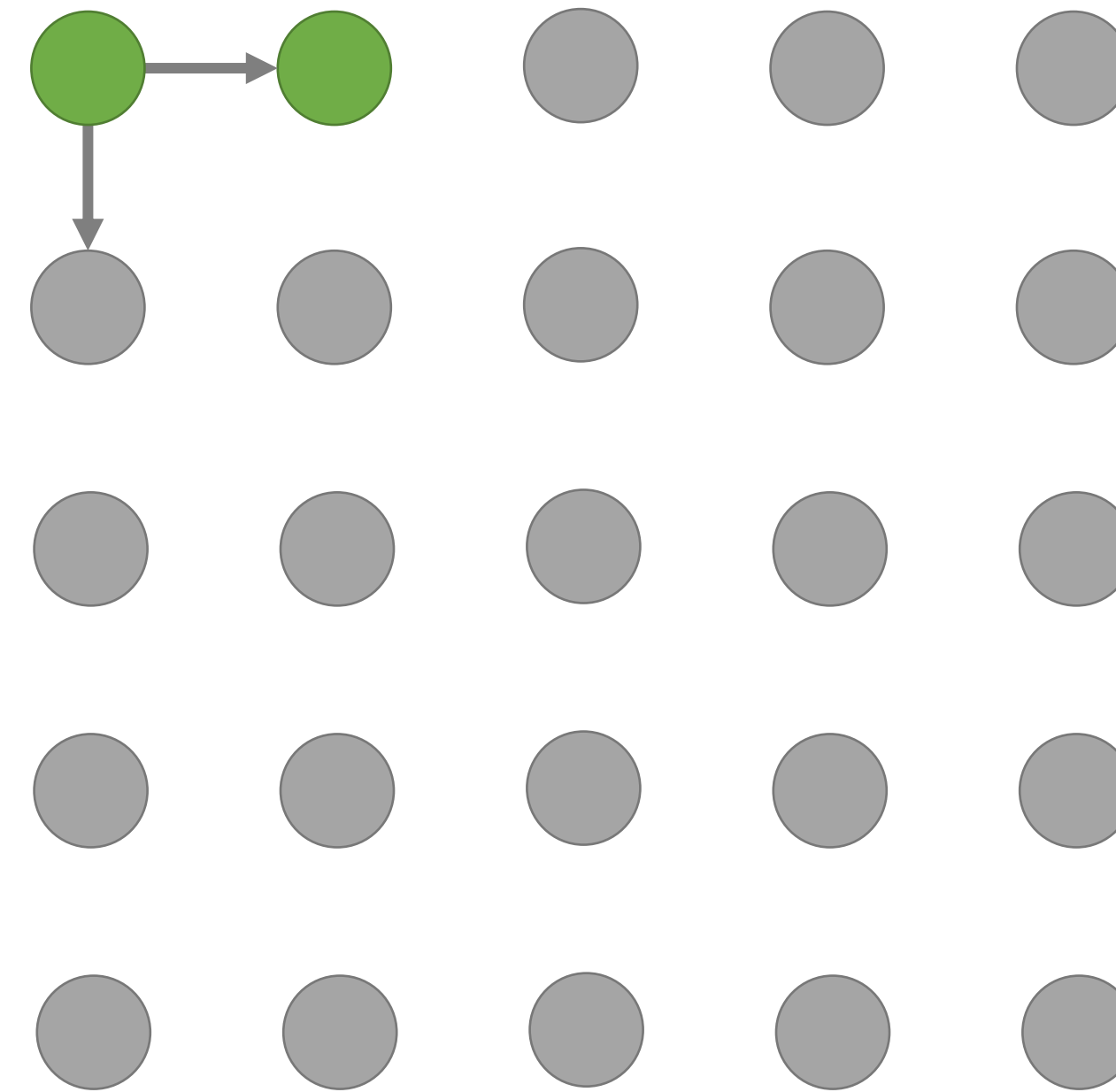


Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu 2016, Pixel Recurrent Neural Networks

PixelRNN

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)

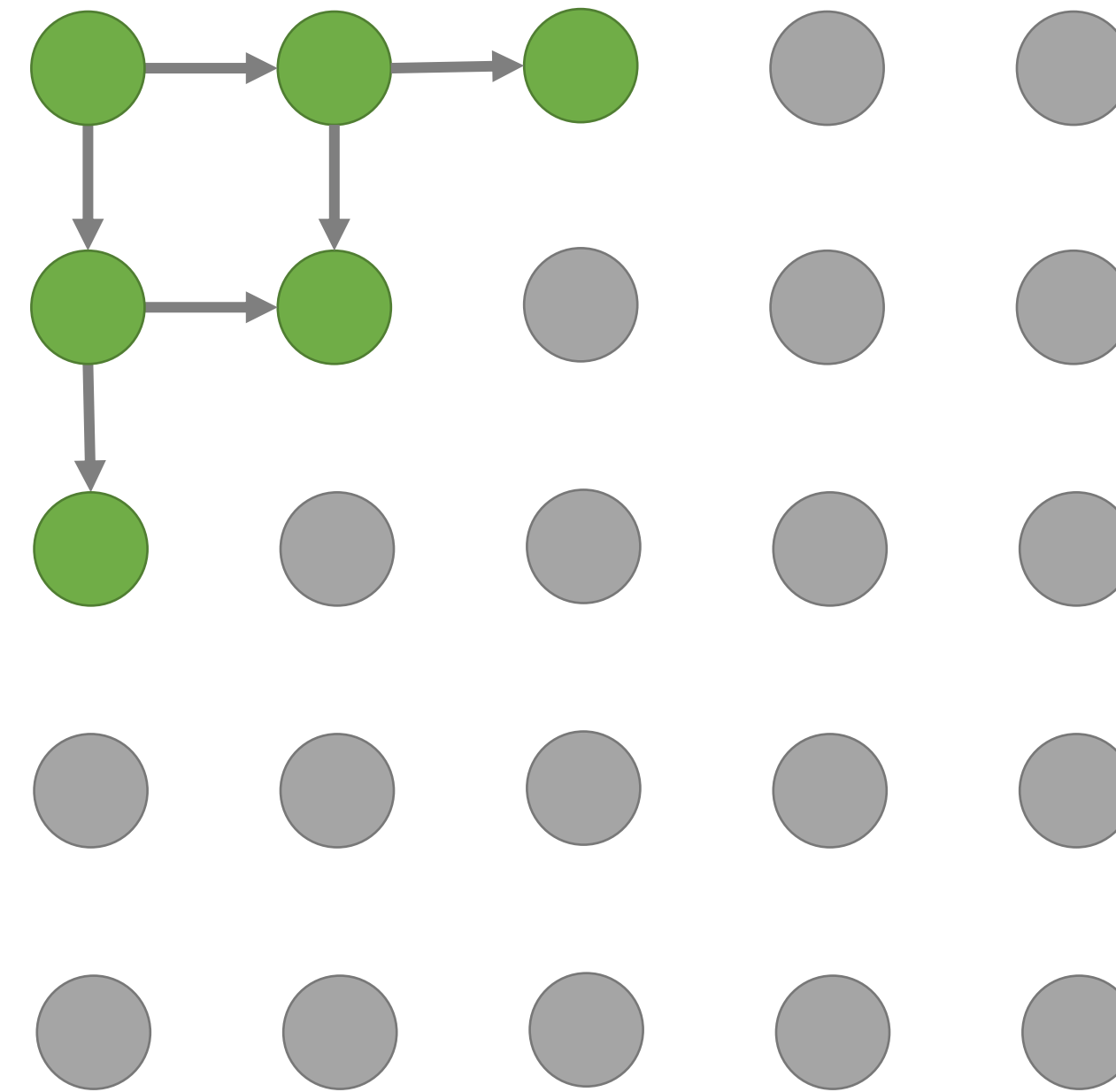


Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu 2016, Pixel Recurrent Neural Networks

PixelRNN

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



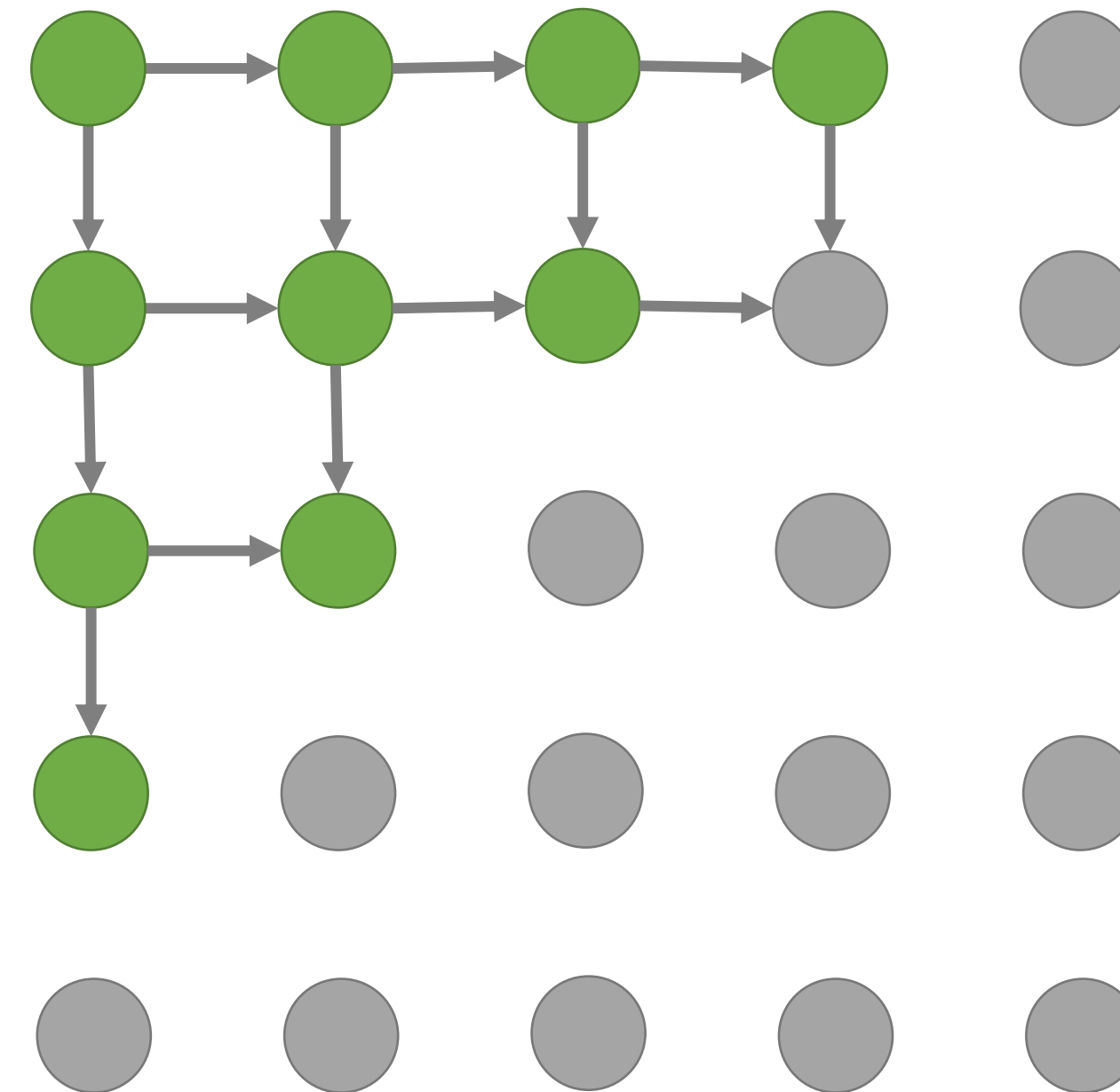
Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu 2016, Pixel Recurrent Neural Networks

PixelRNN

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!



Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu 2016, Pixel Recurrent Neural Networks

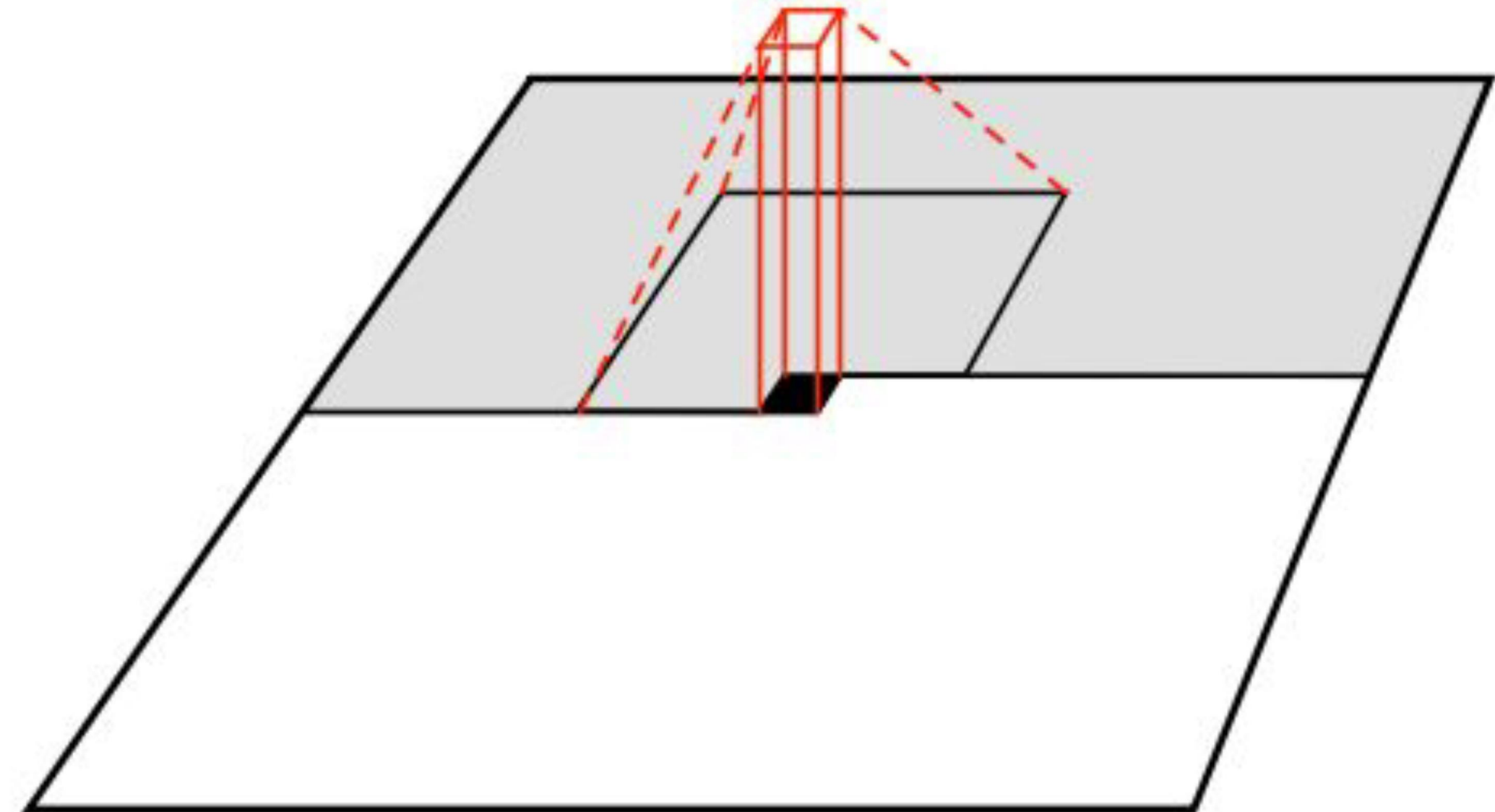
PixelCNN

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN (can parallelize convolutions since context region values known from training images)

Generation is still slow: For a 32x32 image, we need to do forward passes of the network 1024 times for a single image



Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, Koray Kavukcuoglu. 2016. Conditional Image Generation with PixelCNN Decoders

PixelRNN and PixelCNN

Pros:

- Can explicitly compute likelihood $p(x)$
- Easy to optimize
- Good samples

Con:

- Sequential generation => slow

Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc...

See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017
(PixelCNN++)



Variational Autoencoders (VAE)



Variational Autoencoders (VAE)

VAEs (Variational Autoencoders) are generative models that can learn to generate new data similar to the training data. They combine the ideas of an autoencoder and a probabilistic latent variable model.

A VAE consists of two main components: an **encoder** and a **decoder**. The encoder takes an input data point and maps it to a latent space representation, which is a lower-dimensional representation of the input data. The decoder then takes this latent representation and generates a new data point that is similar to the input data.

The key idea behind VAEs is to learn a probabilistic model that can generate data points from the latent space representation. This is achieved by introducing a probabilistic distribution over the latent space, typically a multivariate Gaussian distribution, and training the model to minimize the difference between the generated data points and the training data.

Variational Autoencoders (VAE)

VAEs (Variational Autoencoders) are generative models that can learn to generate new data similar to the training data. They combine the ideas of an autoencoder and a probabilistic latent variable model.

During training, VAEs use a technique called the "reparameterization trick" to sample from the latent space distribution and to ensure that the model can be efficiently trained using backpropagation. This allows VAEs to learn the parameters of the latent space distribution that best capture the underlying structure of the data.

Once trained, VAEs can be used to generate new data points by sampling from the learned latent space distribution and passing the sample through the decoder network. VAEs have been successfully applied in a wide range of applications, including image and speech synthesis, anomaly detection, and data compression.

Variational Autoencoders (VAE)

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

Why latent z ?

No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

Autoencoders

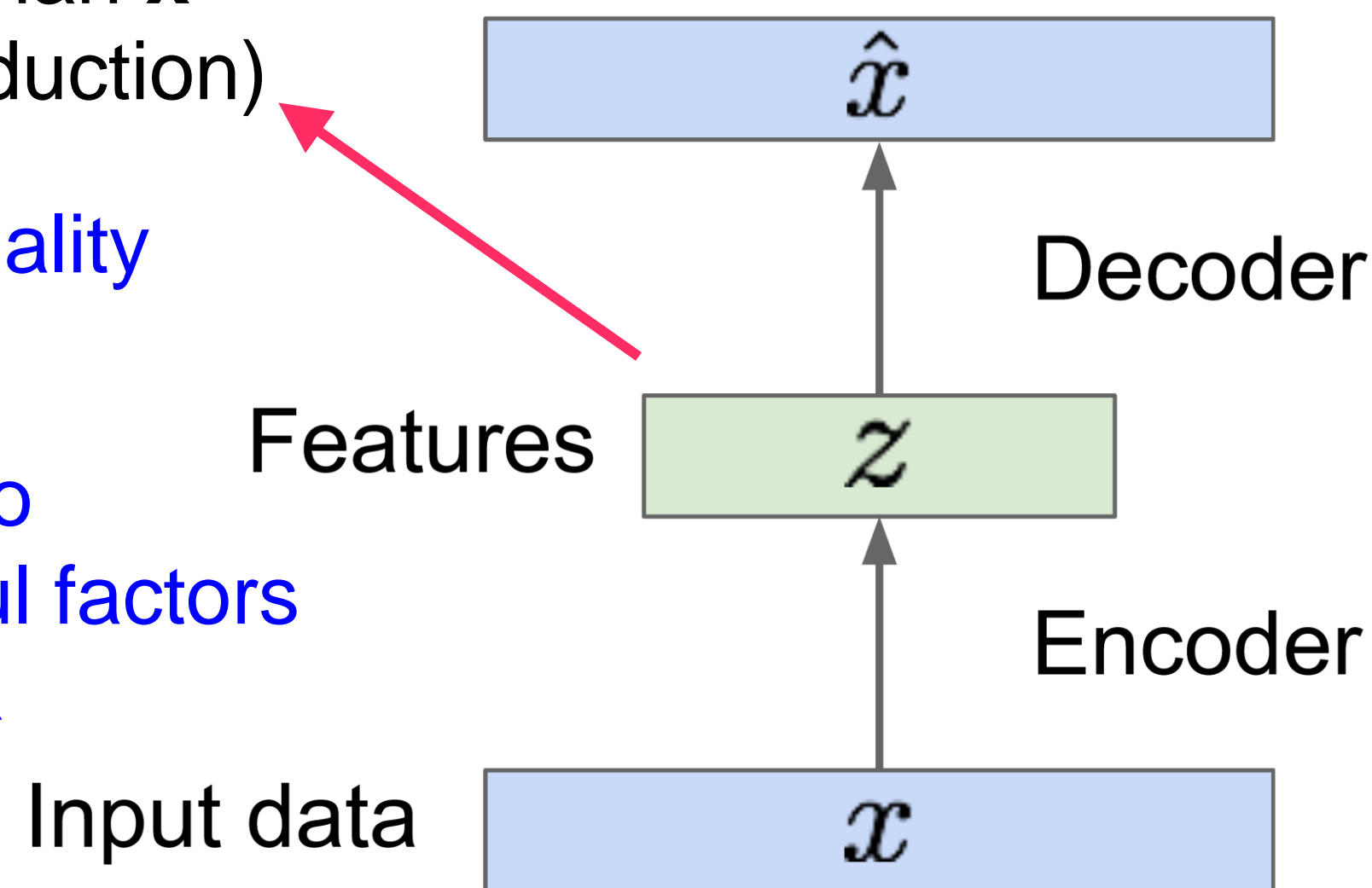
Some background first on autoencoders:

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

\mathbf{z} usually smaller than \mathbf{x}
(dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data



How to learn this feature representation?

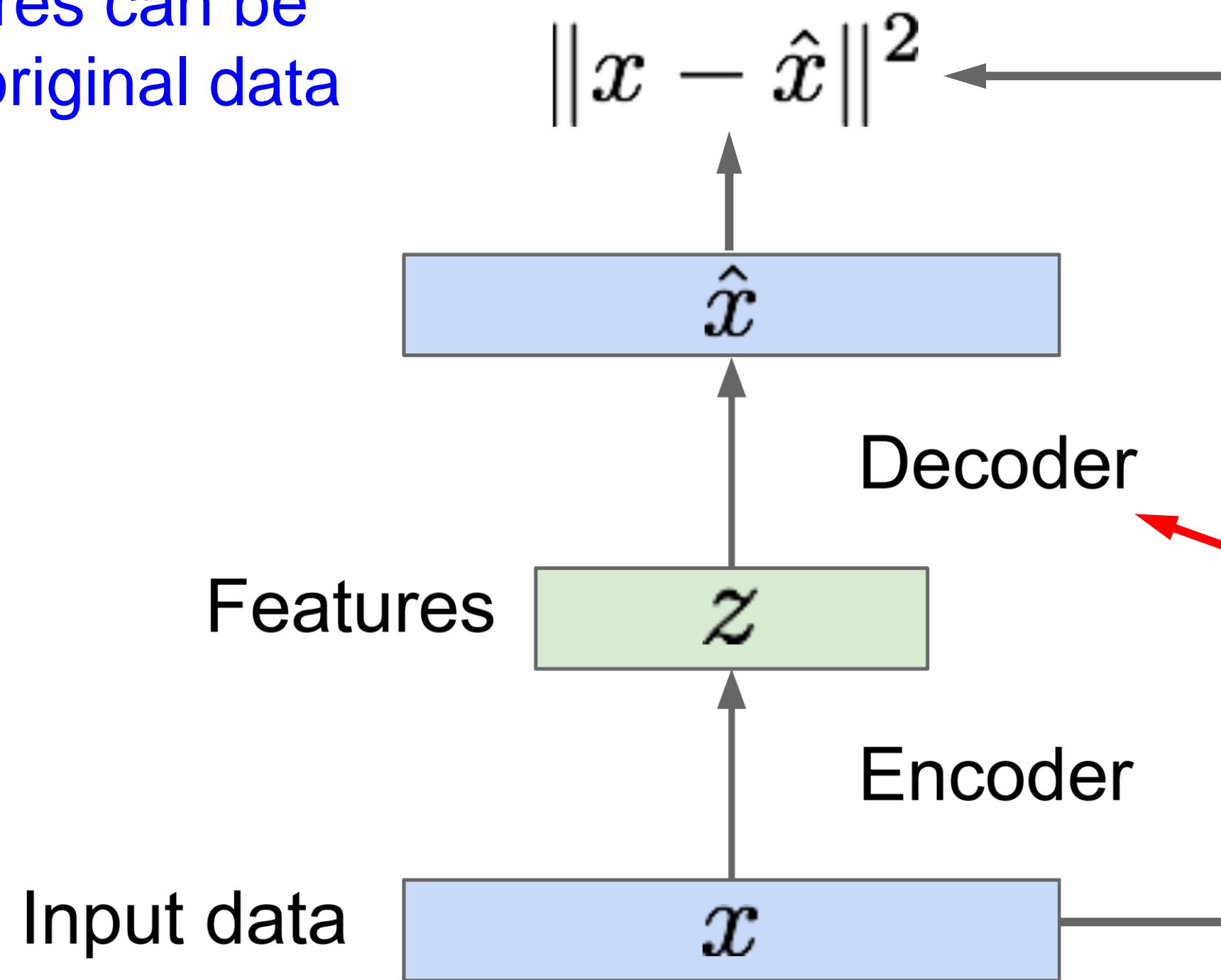
Train such that features can be used to reconstruct original data "Autoencoding"
- encoding input itself

Autoencoders

Train such that features can be used to reconstruct original data

L2 Loss function:

Doesn't use labels!



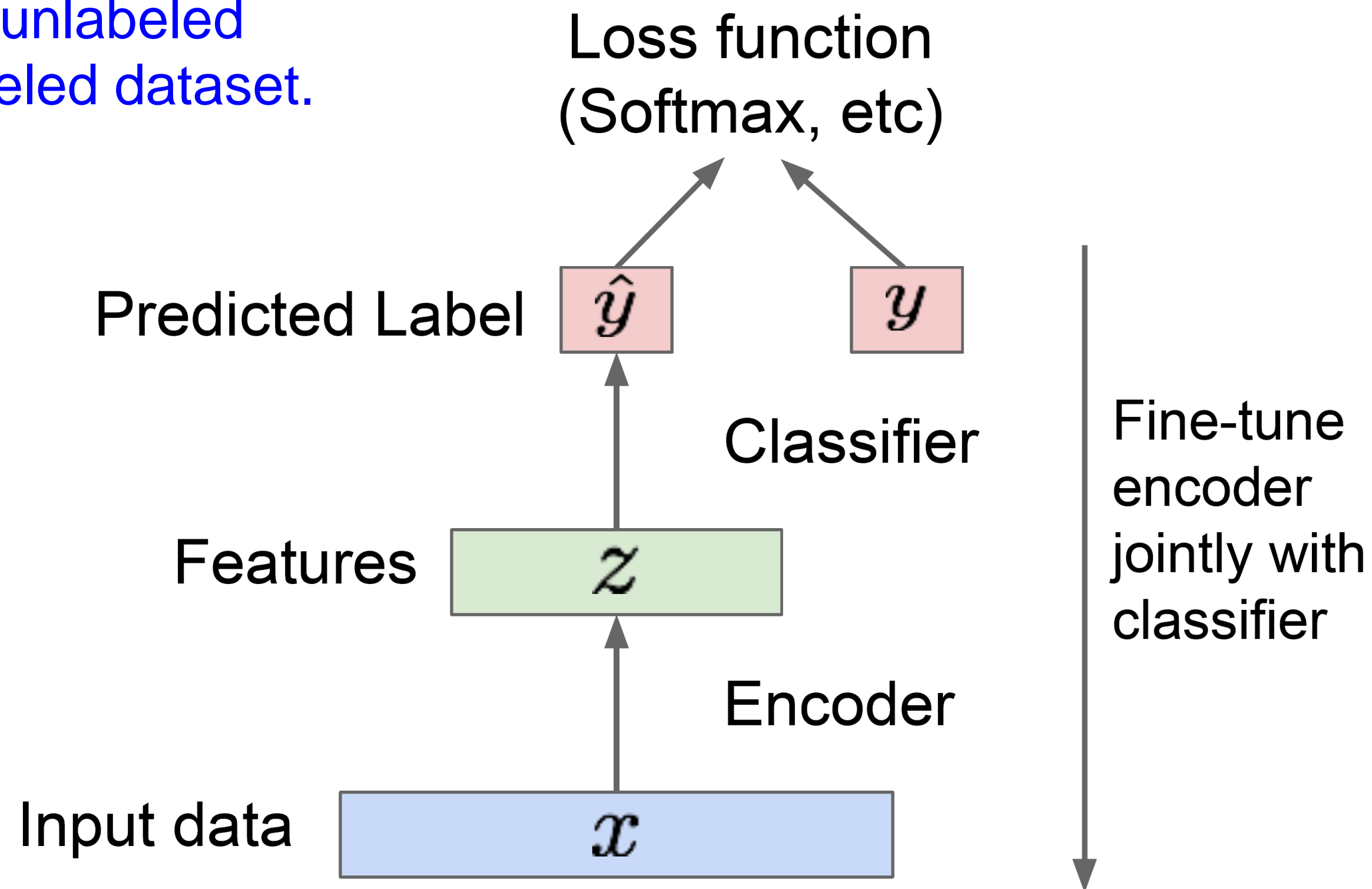
After training, throw away decoder



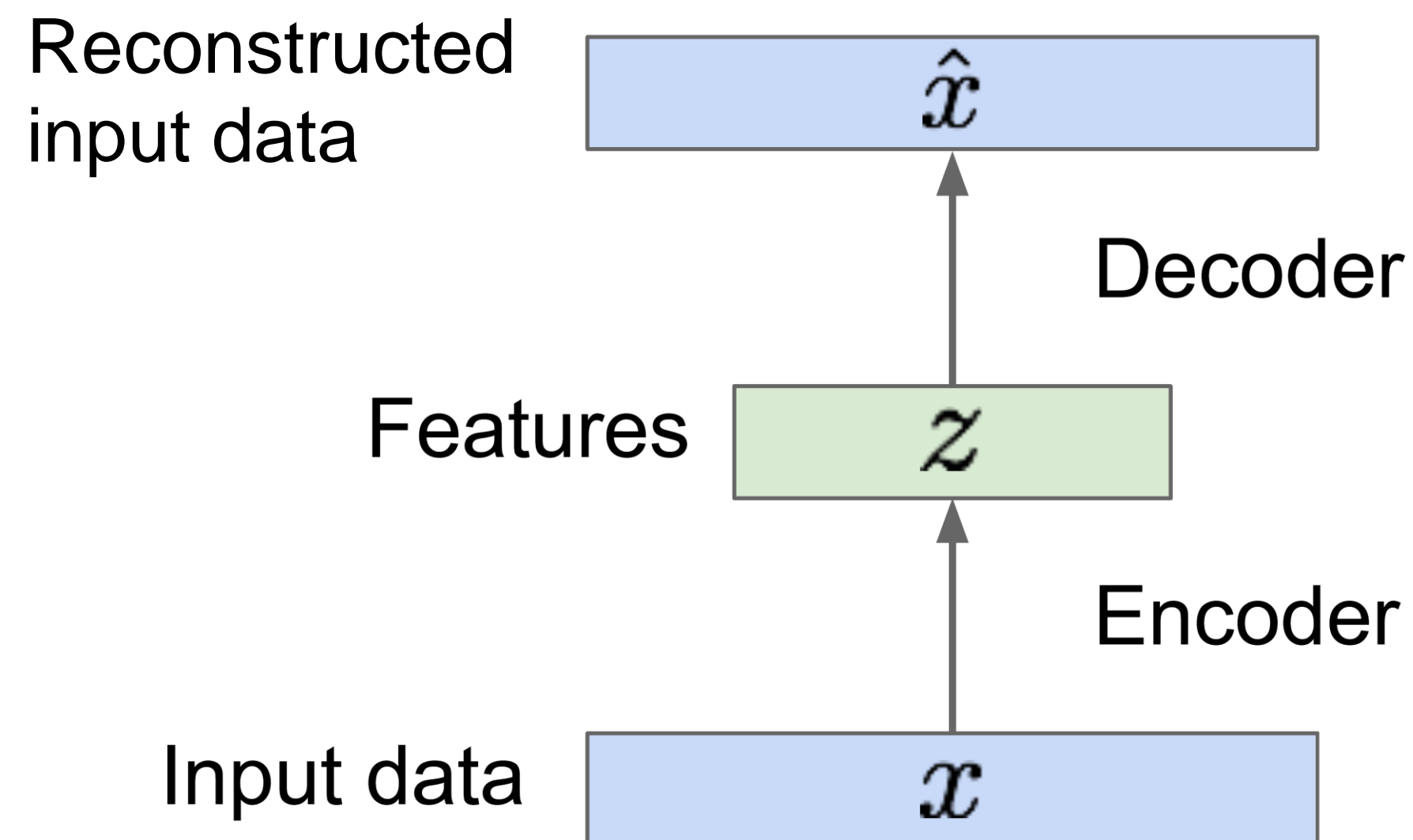
Autoencoders

Transfer from large, unlabeled dataset to small, labeled dataset.

Encoder can be used to initialize a **supervised** model



Autoencoders



Autoencoders can reconstruct data, and can learn features to initialize a supervised model. Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of z .

How do we make autoencoder a **generative model**?

Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

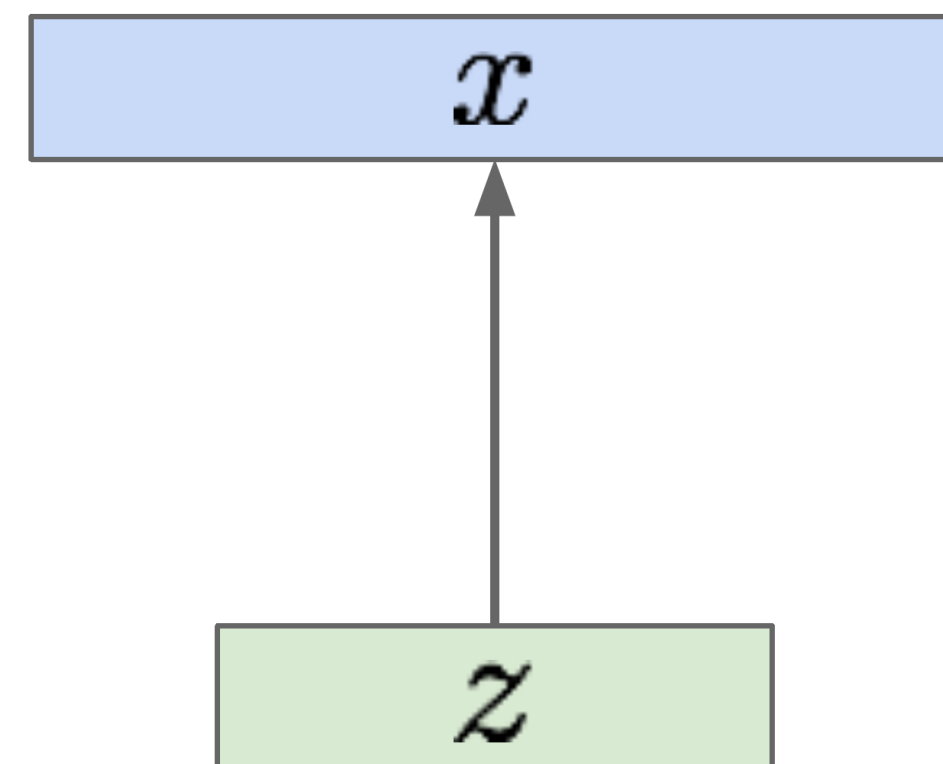
Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from the distribution of unobserved (latent) representation \mathbf{z}

Sample from true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



Intuition (remember from autoencoders!): \mathbf{x} is an image, \mathbf{z} is latent factors used to generate \mathbf{x} : attributes, orientation, etc.

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

We want to estimate the true parameters θ^* of this generative model given training data x .

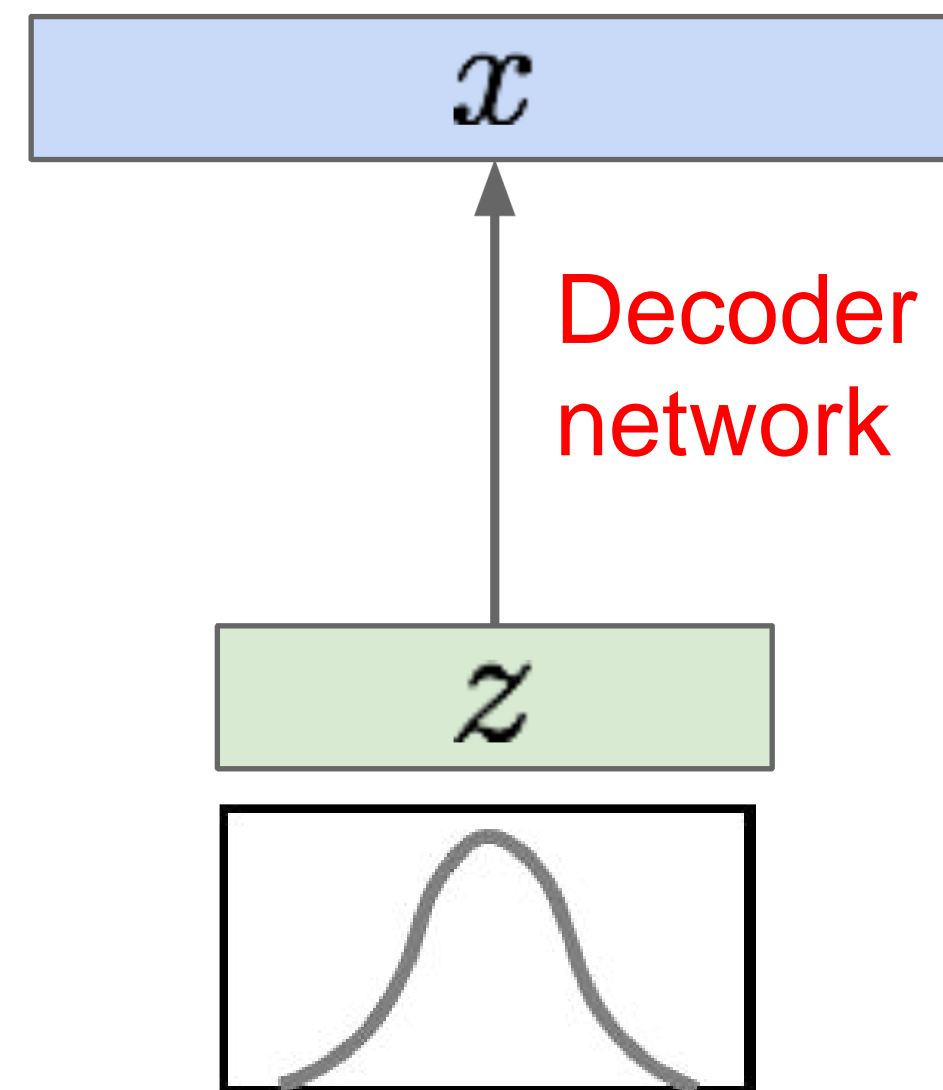
How should we represent this model?

Sample from true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



Choose prior $p(z)$ to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional $p(x|z)$ is complex (generates image) => represent with neural network

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014



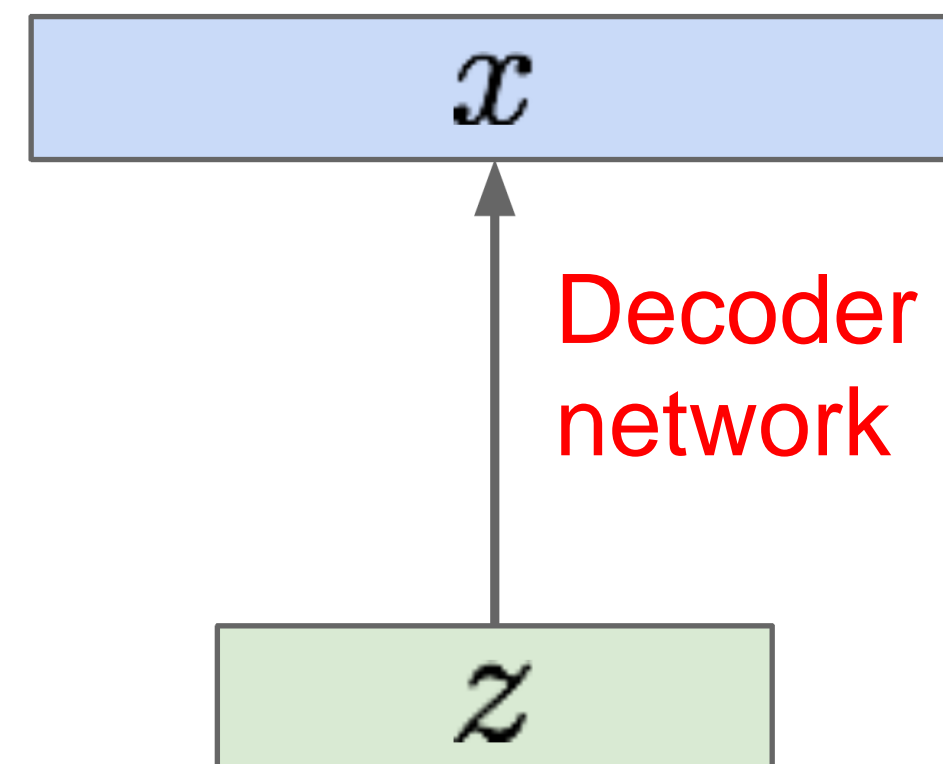
Variational Autoencoders

Sample from true conditional

$$p_{\theta^*}(x | z^{(i)})$$

Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters θ^* of this generative model given training data x .

How to train the model?

Learn model parameters to maximize likelihood of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

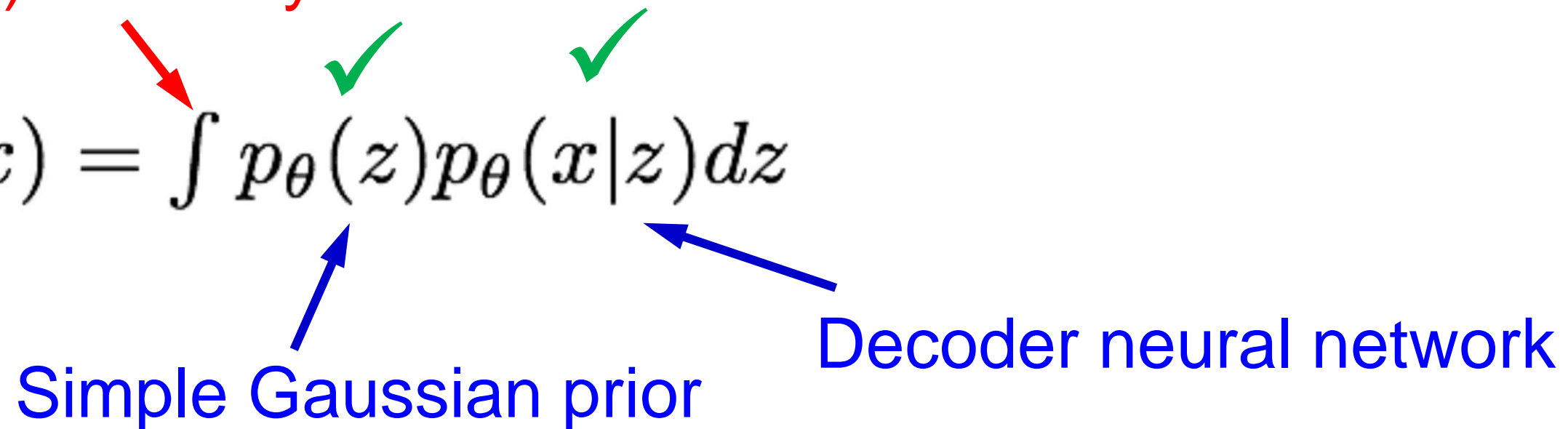
Q: What is the problem with this?
Intractable!

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: *Intractability*

Intractable to compute $p(x|z)$ for every z !

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$



$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^k p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z)$$

Monte Carlo estimation is too high variance

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: *Intractability*

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$

Posterior density: $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

↙ Intractable data likelihood

Solution: In addition to modeling $p_{\theta}(x|z)$, learn $q_{\phi}(z|x)$ that approximates the true posterior $p_{\theta}(z|x)$.

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data x

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

We want to maximize the data likelihood

$$\begin{aligned}
 \log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[\log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)} \text{ Does not depend on } z) \\
 &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\
 &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\
 &= \mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms}) \\
 &= \mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))
 \end{aligned}$$

Decoder network gives $p_{\theta}(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_{\theta}(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always ≥ 0 .

Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[\log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}$$

We want to maximize the data likelihood

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution close to prior

Tractable lower bound which we can take gradient of and optimize! ($p_{\theta}(x|z)$ differentiable, KL term differentiable)

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - \boxed{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}$$

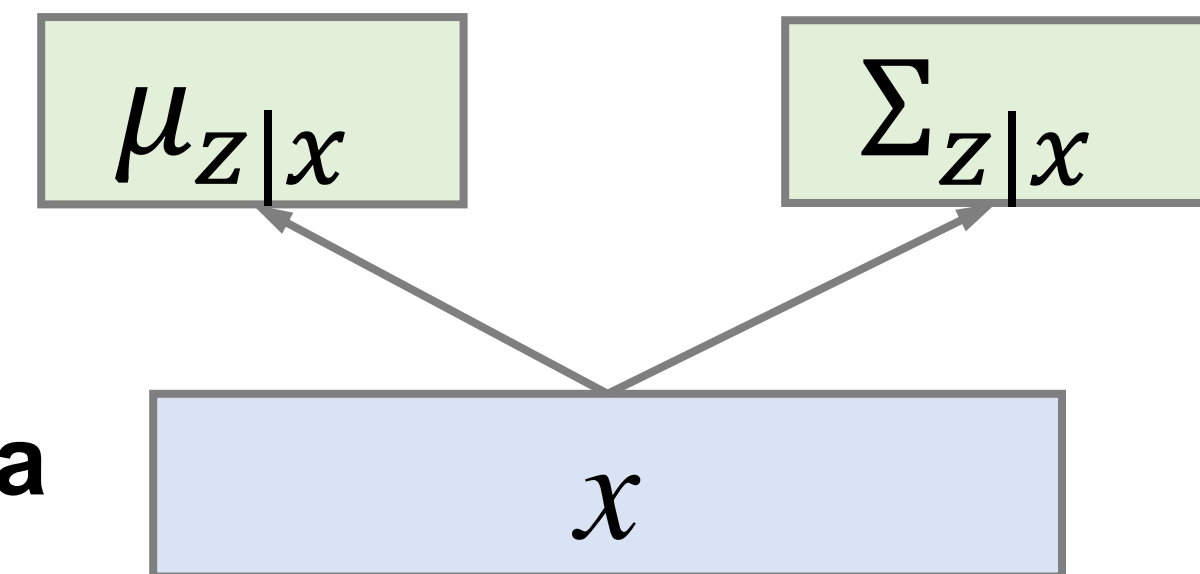
Let's look at computing the KL divergence between the estimated posterior and the prior given some data

Make approximate posterior distribution close to prior

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution

Encoder network
 $q_\phi(z|x)$



Input Data



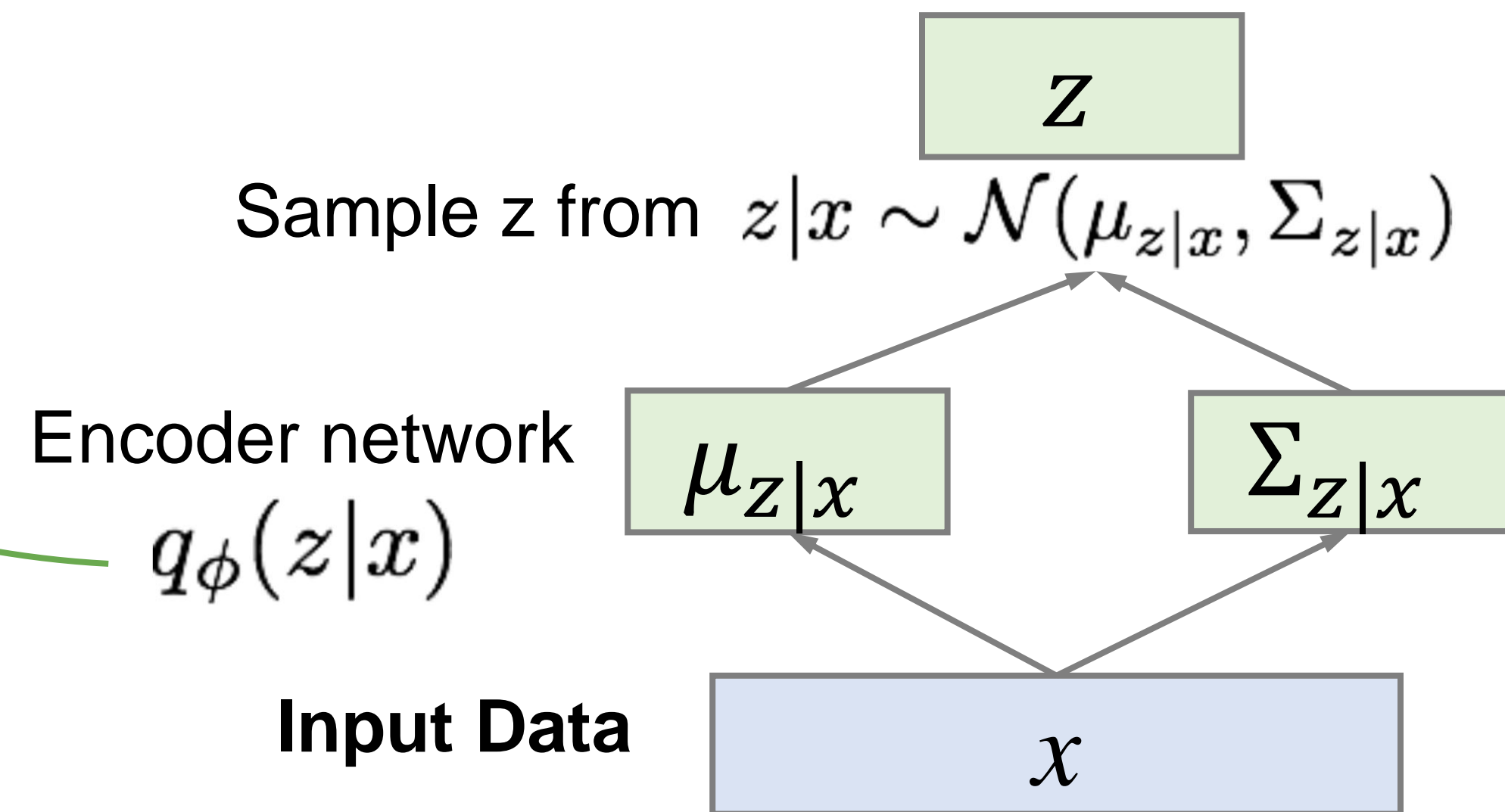
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

Not part of the computation graph!



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Reparameterization trick to make sampling differentiable:

Sample $\epsilon \sim \mathcal{N}(0, I)$ Input to the graph

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$

Part of computation graph

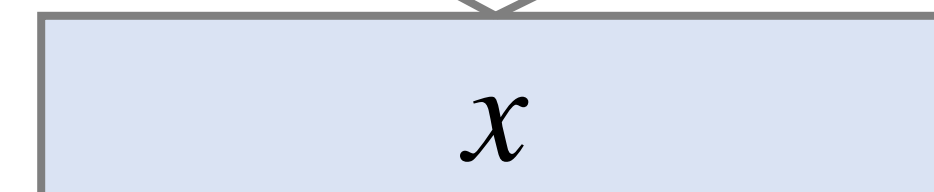


Sample z from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

Encoder network $q_\phi(z|x)$



Input Data

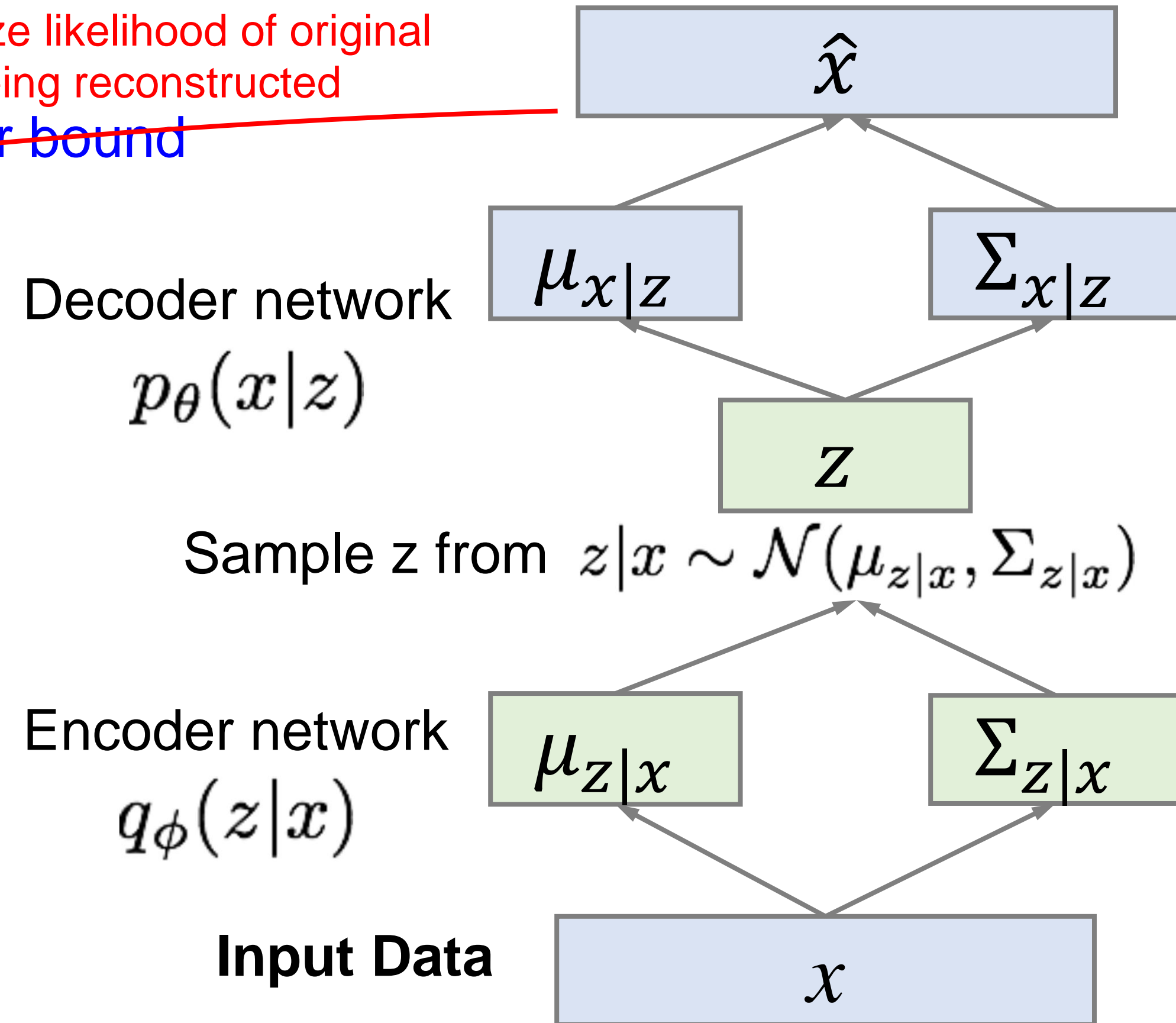


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

Maximize likelihood of original input being reconstructed

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

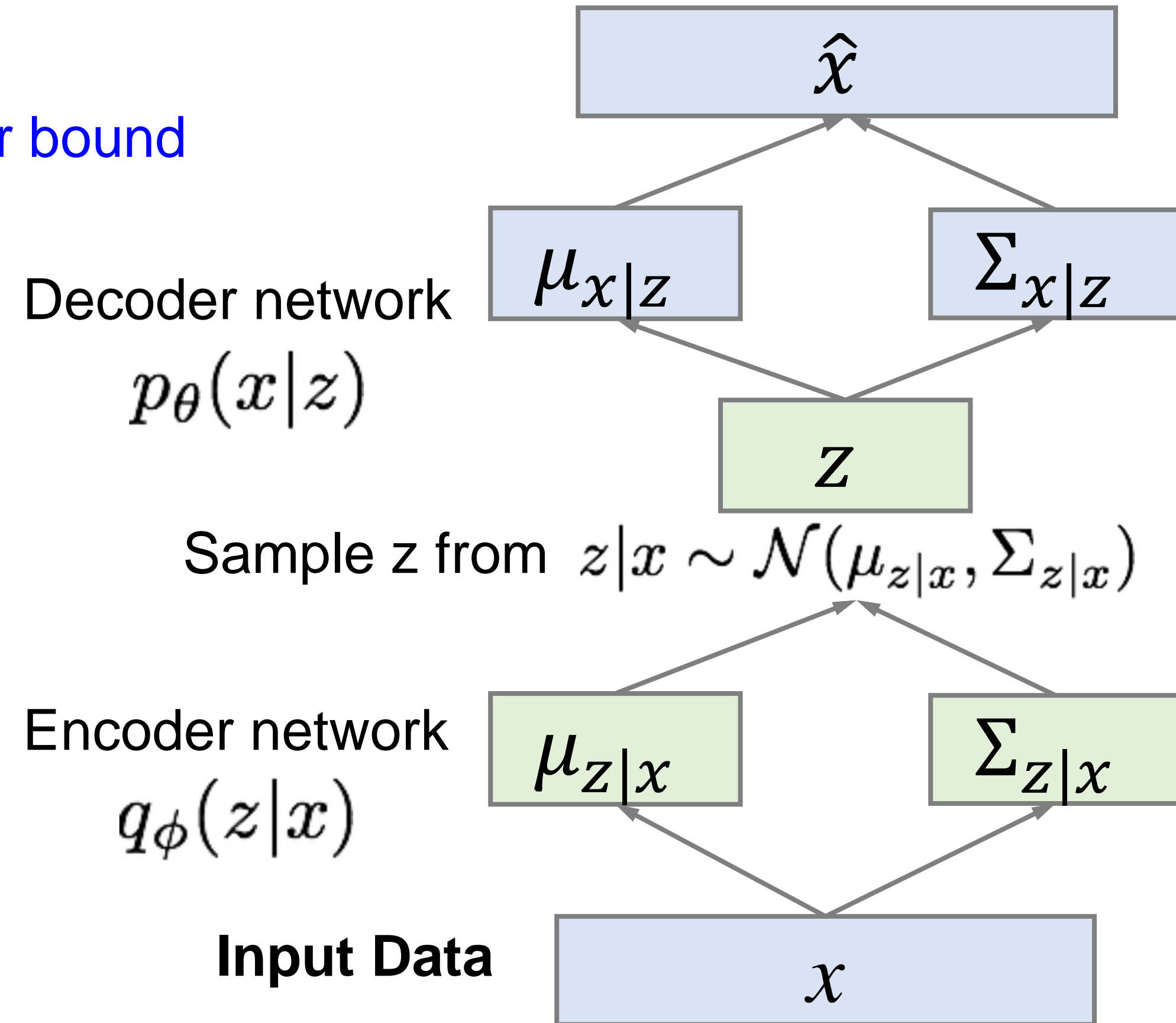


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

For every minibatch of input data: compute this forward pass, and then backprop!

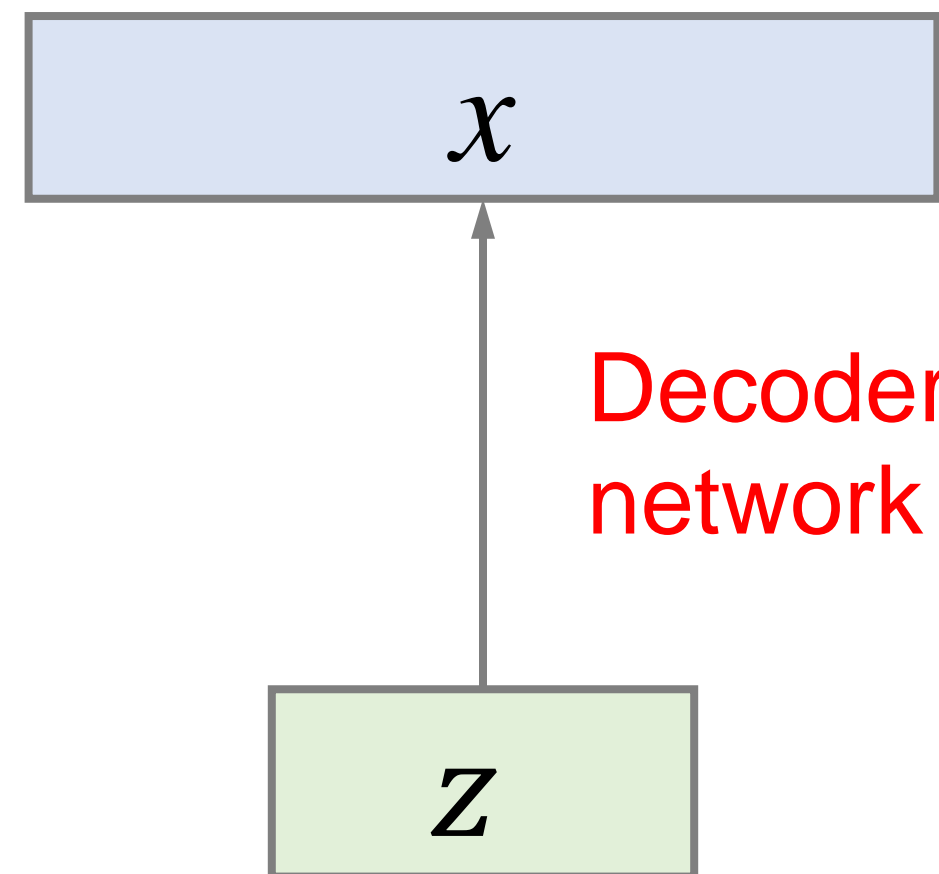


Variational Autoencoders: *Generating Data!*

Our assumption about data generation process

Sample from true conditional

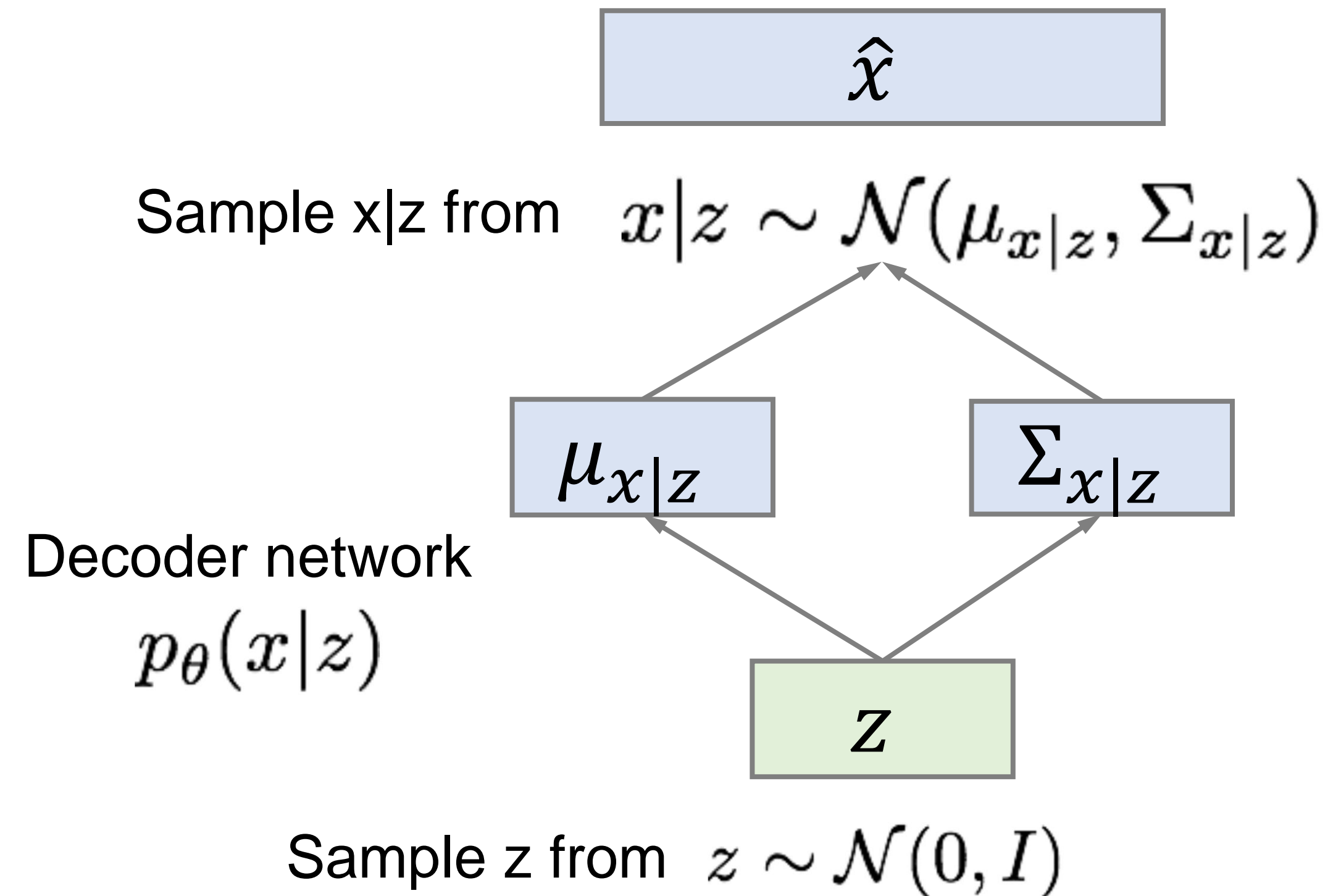
$$p_{\theta^*}(x | z^{(i)})$$



Sample from true prior

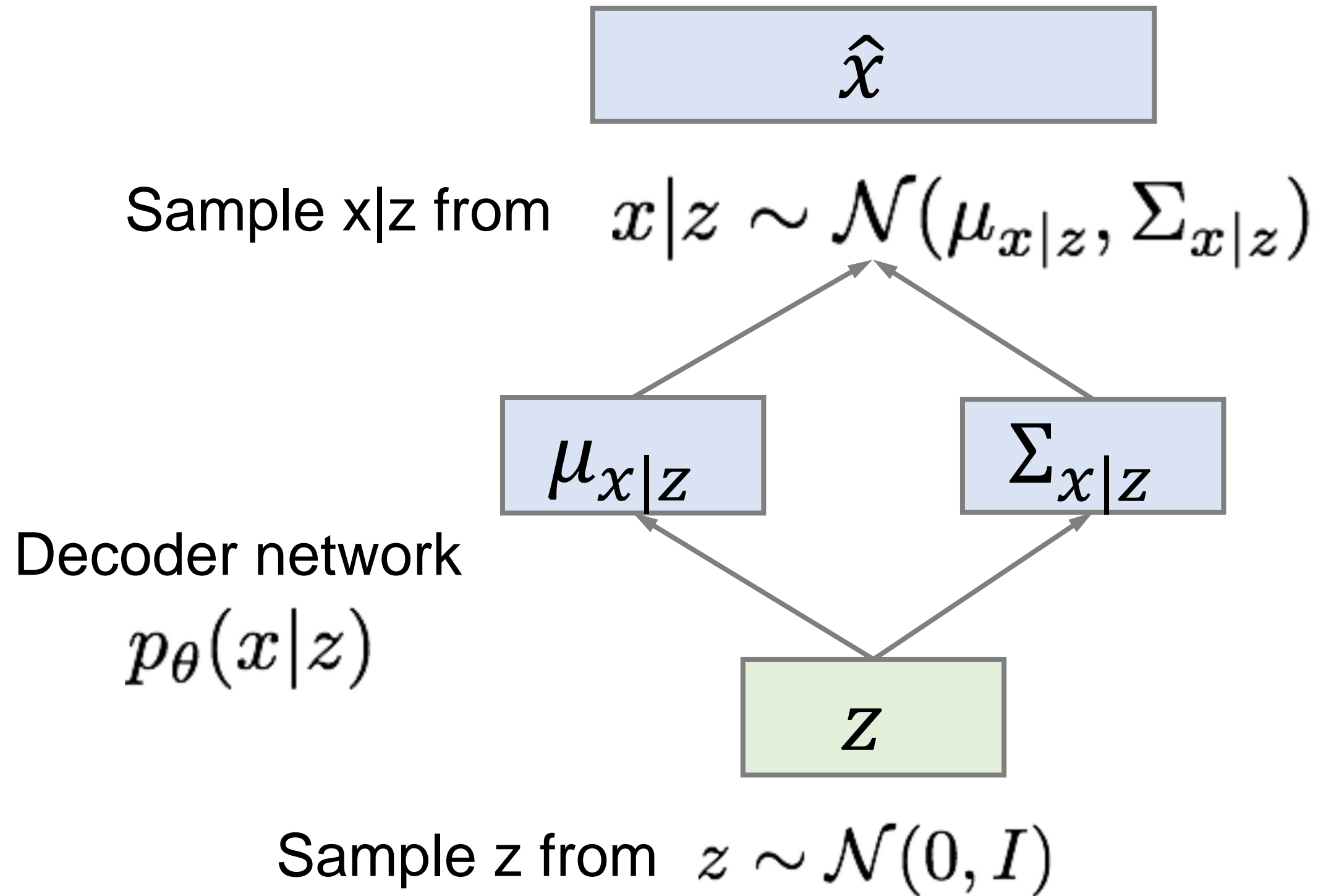
$$z^{(i)} \sim p_{\theta^*}(z)$$

Now given a trained VAE:
use decoder network & sample z from prior!

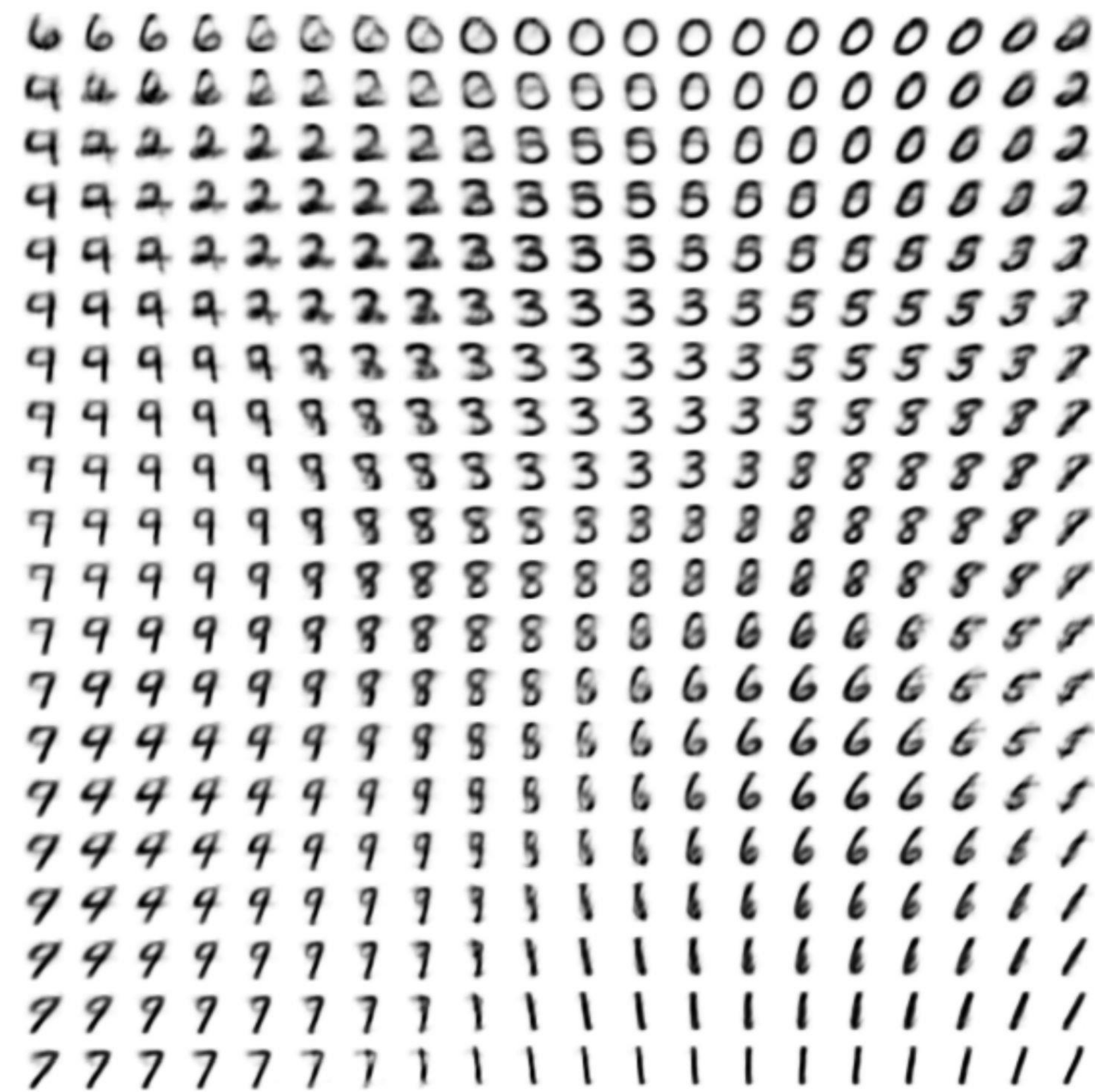


Variational Autoencoders: *Generating Data!*

Use decoder network. Now sample z from prior!



Data manifold for 2-d z



Variational Autoencoders: *Generating Data!*

Diagonal prior on \mathbf{z}
 => independent latent variables

Different dimensions of \mathbf{z}
 encode interpretable factors
 of variation

Also good feature representation that
 can be computed using $q_\phi(\mathbf{z}|\mathbf{x})!$

Degree of smile
 Vary \mathbf{z}_1



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

Advantages of VAEs:

1. Generative model: VAEs are generative models, which means they can learn to generate new samples of data. This makes them useful in applications such as image and speech processing, where it is often difficult to collect and label large amounts of data.
2. Continuous latent space: VAEs learn a continuous latent space, which allows for interpolation between samples. This means that VAEs can generate new samples that are similar to existing ones, but with some variations.
3. Regularization: VAEs have built-in regularization that encourages the learned representations to be smooth and well-behaved. This can help prevent overfitting and improve generalization to new data.

Variational Autoencoders

Disadvantages of VAEs:

1. Blurry images: VAEs tend to generate blurry images, especially when compared to other generative models like Generative Adversarial Networks (GANs). This is because VAEs use a reconstruction loss to ensure that the generated samples are similar to the original data, but this loss function does not capture high-frequency details.
2. Difficulty with high-dimensional data: VAEs can have difficulty with high-dimensional data, as the latent space may become too complex to learn. This can lead to poor performance and slow training times.
3. Lack of diversity: VAEs can sometimes generate samples that lack diversity, as they tend to learn the average characteristics of the training data. This can be addressed by using techniques such as data augmentation or modifying the loss function.

Overall, VAEs are a powerful and versatile type of generative model that have both advantages and disadvantages. Their ability to learn a continuous latent space and their built-in regularization make them useful in many applications, but they may struggle with high-dimensional data and generate blurry or less diverse samples compared to other generative models.

Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.



Generative Adversarial Networks (GANs)



Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) is a class of deep learning models that is used for generative tasks. GANs consist of two neural networks: a generator and a discriminator.

The **generator** network takes in random noise as input and produces a sample that mimics the training data. The **discriminator** network takes in both the generated samples and the real training data and tries to distinguish between them. The generator network tries to produce samples that are more and more similar to the real data, while the discriminator network tries to correctly classify the samples as either real or fake.

During training, the two networks are trained in an adversarial manner, with the generator trying to fool the discriminator into thinking its generated samples are real, and the discriminator trying to correctly classify the samples. As the two networks compete against each other, the generator learns to generate more realistic samples and the discriminator becomes better at distinguishing between real and fake samples.

GANs have been used for a wide range of tasks, such as image and video synthesis, style transfer, super-resolution, and data augmentation. GANs have the ability to produce high-quality and diverse samples that resemble the real data, making them a powerful tool for generative tasks. However, GANs can be challenging to train, and can suffer from issues such as mode collapse (where the generator produces a limited set of similar samples) and instability during training.

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?
GANs: not modeling any explicit density function!

Generative Adversarial Networks (GANs)

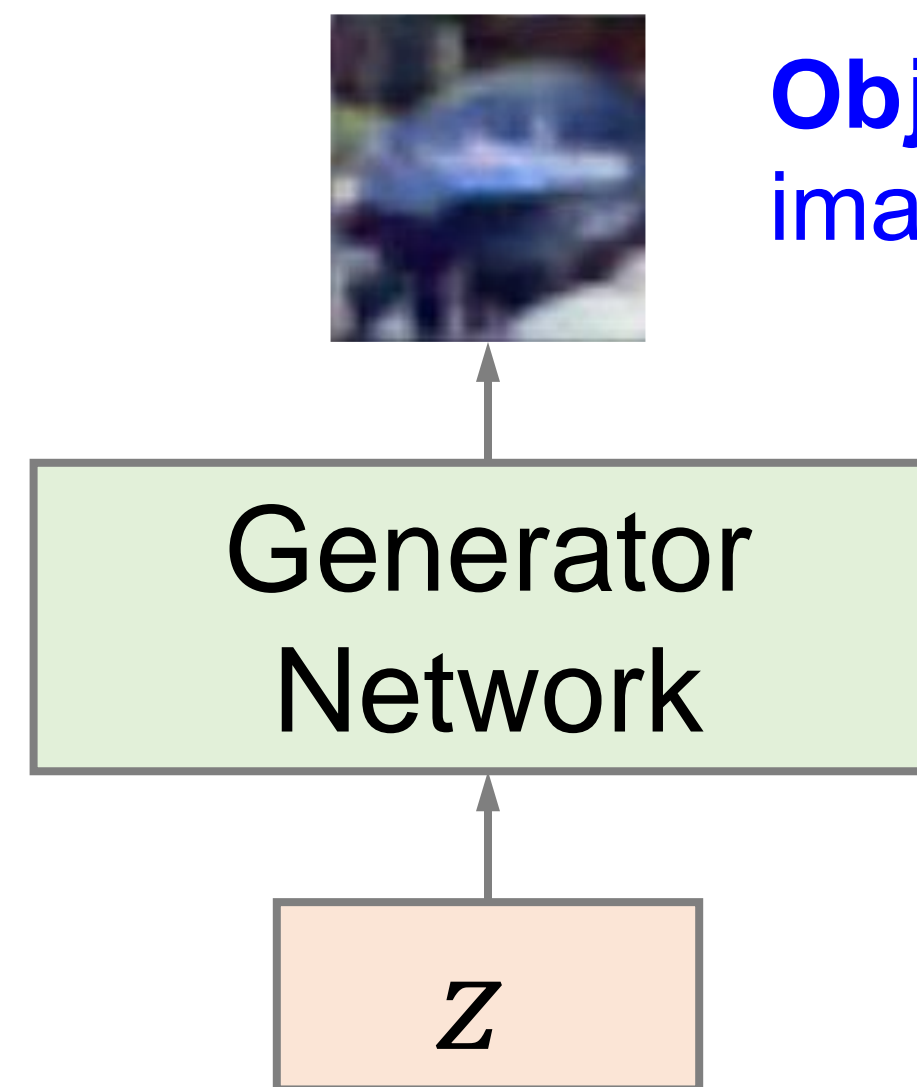
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Output: Sample from training distribution

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Input: Random noise



Objective: generated images should look "real"

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014



Generative Adversarial Networks (GANs)

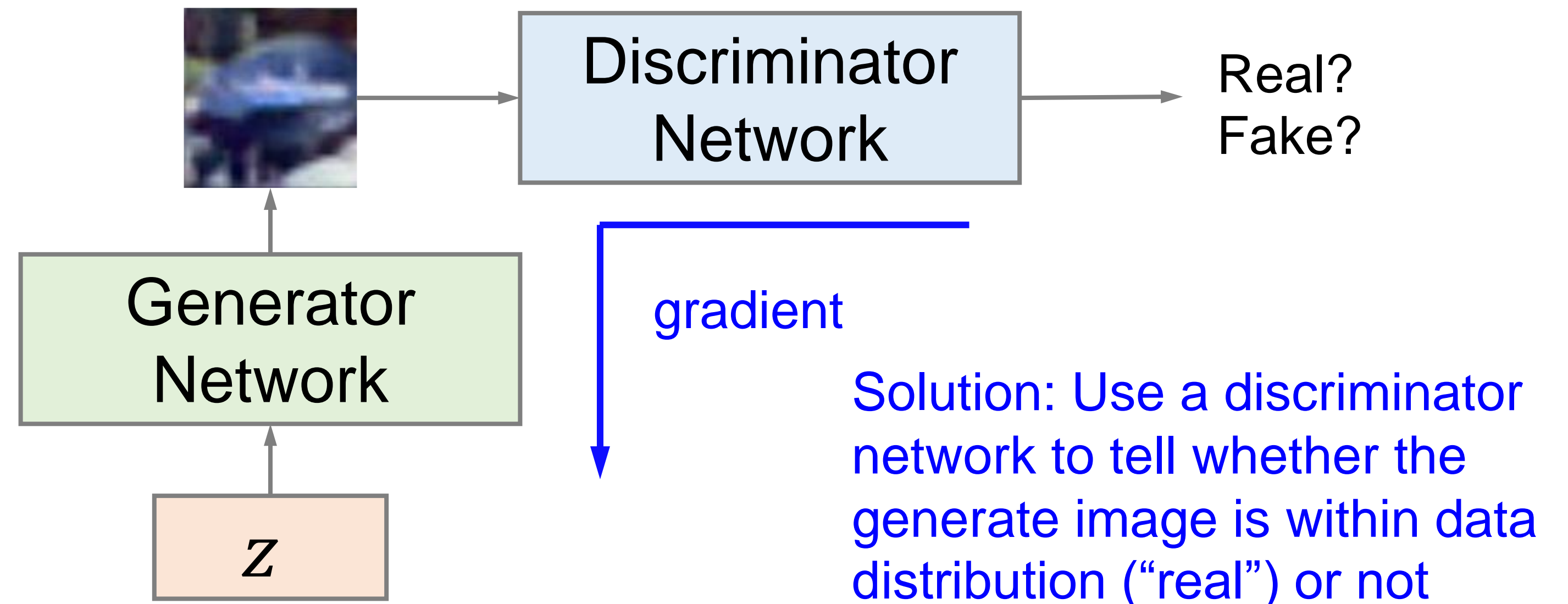
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Output: Sample from training distribution

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Input: Random noise

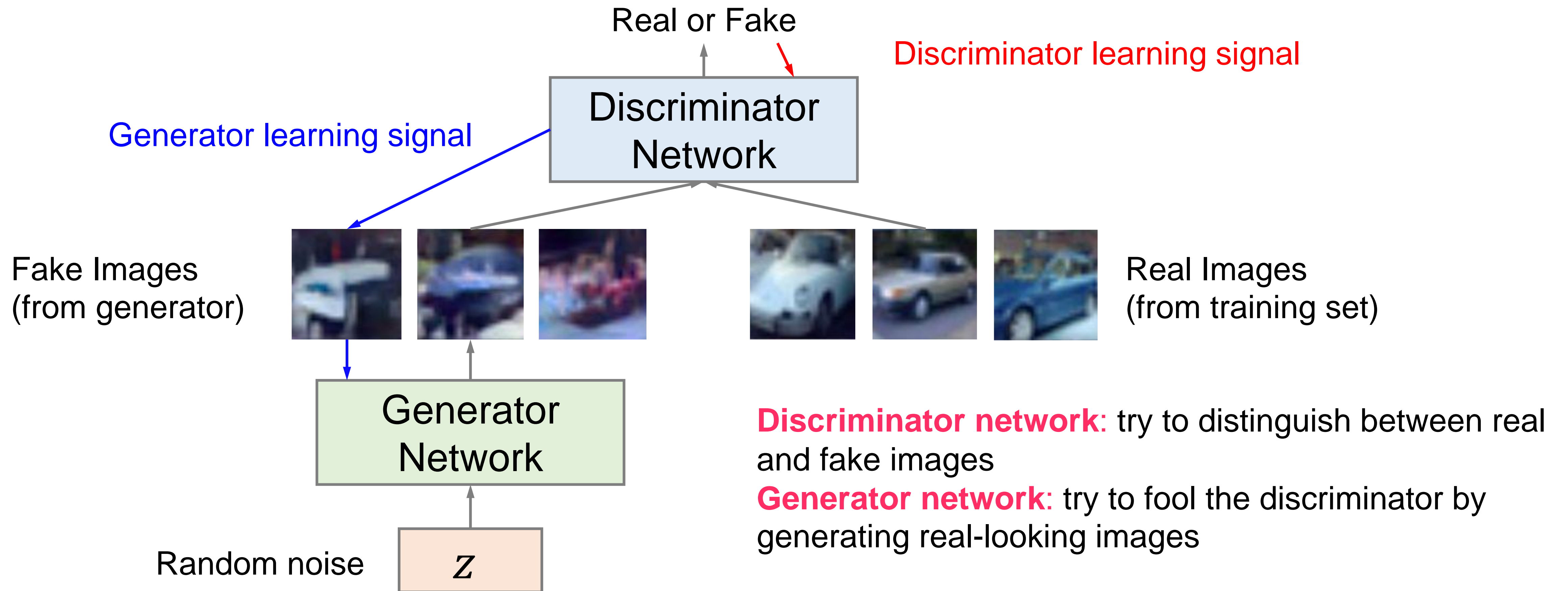


Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014



Training GANs: *Two-player game*



Training GANs: *Two-player game*

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function: Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \underbrace{\log D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$

Generator objective
Discriminator objective

- Discriminator (θ_d) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)



Training GANs: *Two-player game*

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. Gradient ascent on discriminator

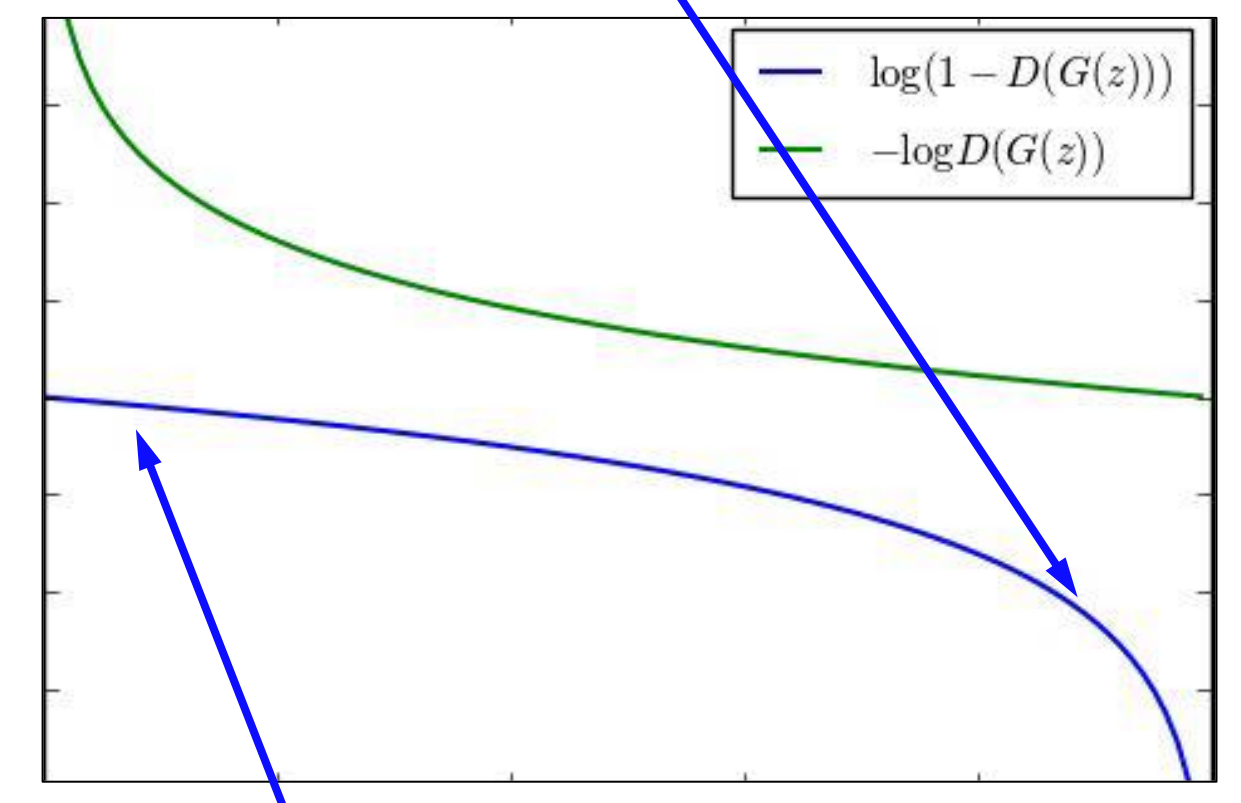
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good



When sample is likely fake, want to learn from it to improve generator

But gradient in this region is relatively flat!

Training GANs: *Two-player game*

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

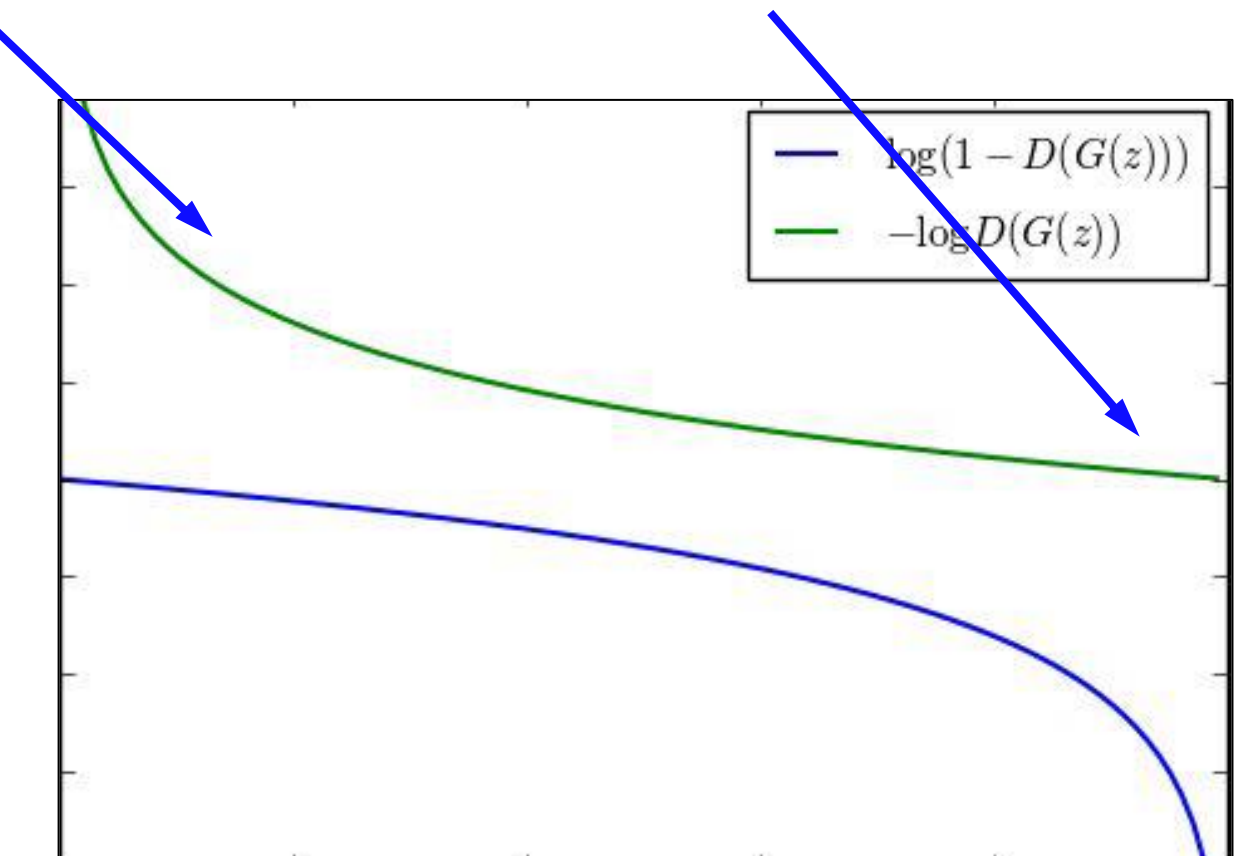
2. Instead: **Gradient ascent** on generator, **different objective**

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong. Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

High gradient signal

Low gradient signal



Training GANs: *Two-player game*

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

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

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

end for

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

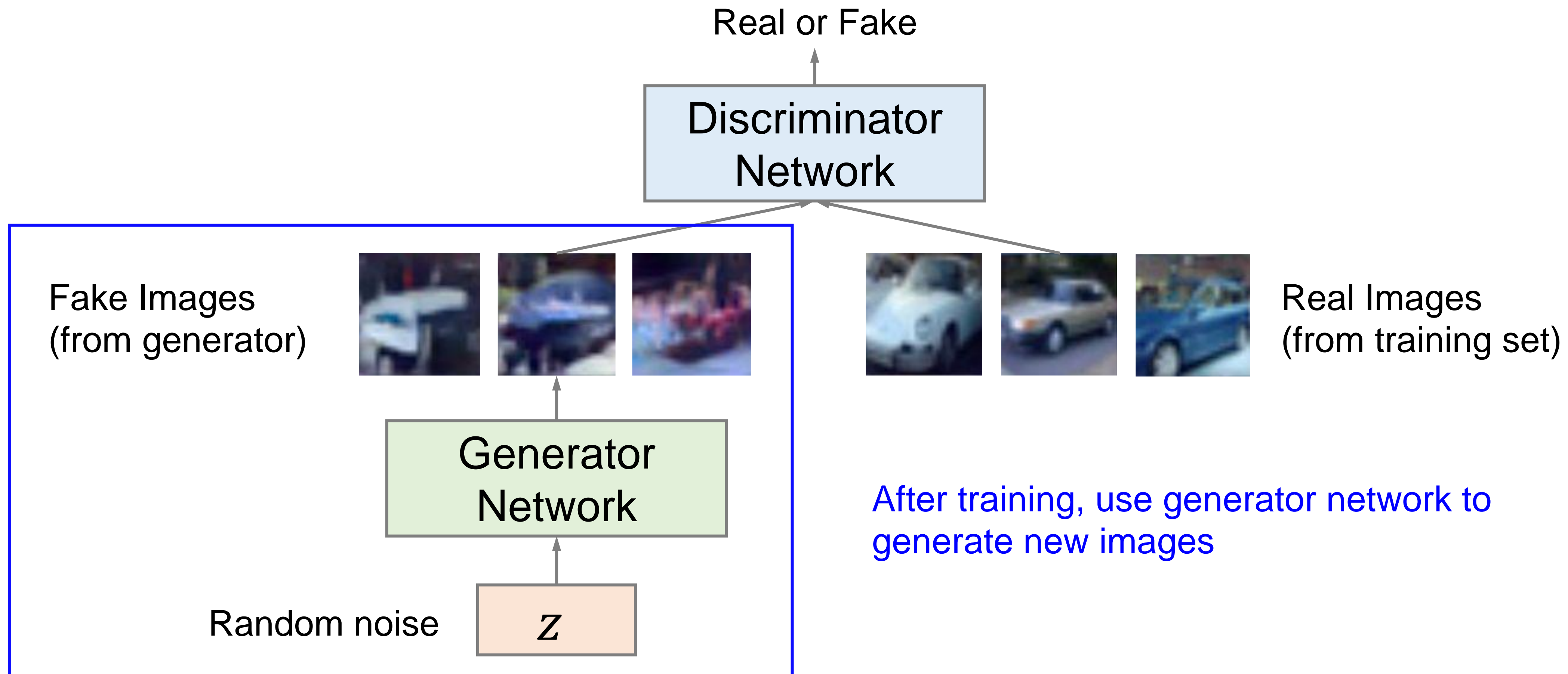
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)

Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: *Two-player game*



2017: Explosion of GANs

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

See also: <https://github.com/soumith/ganhacks> for tips and tricks for trainings GANs

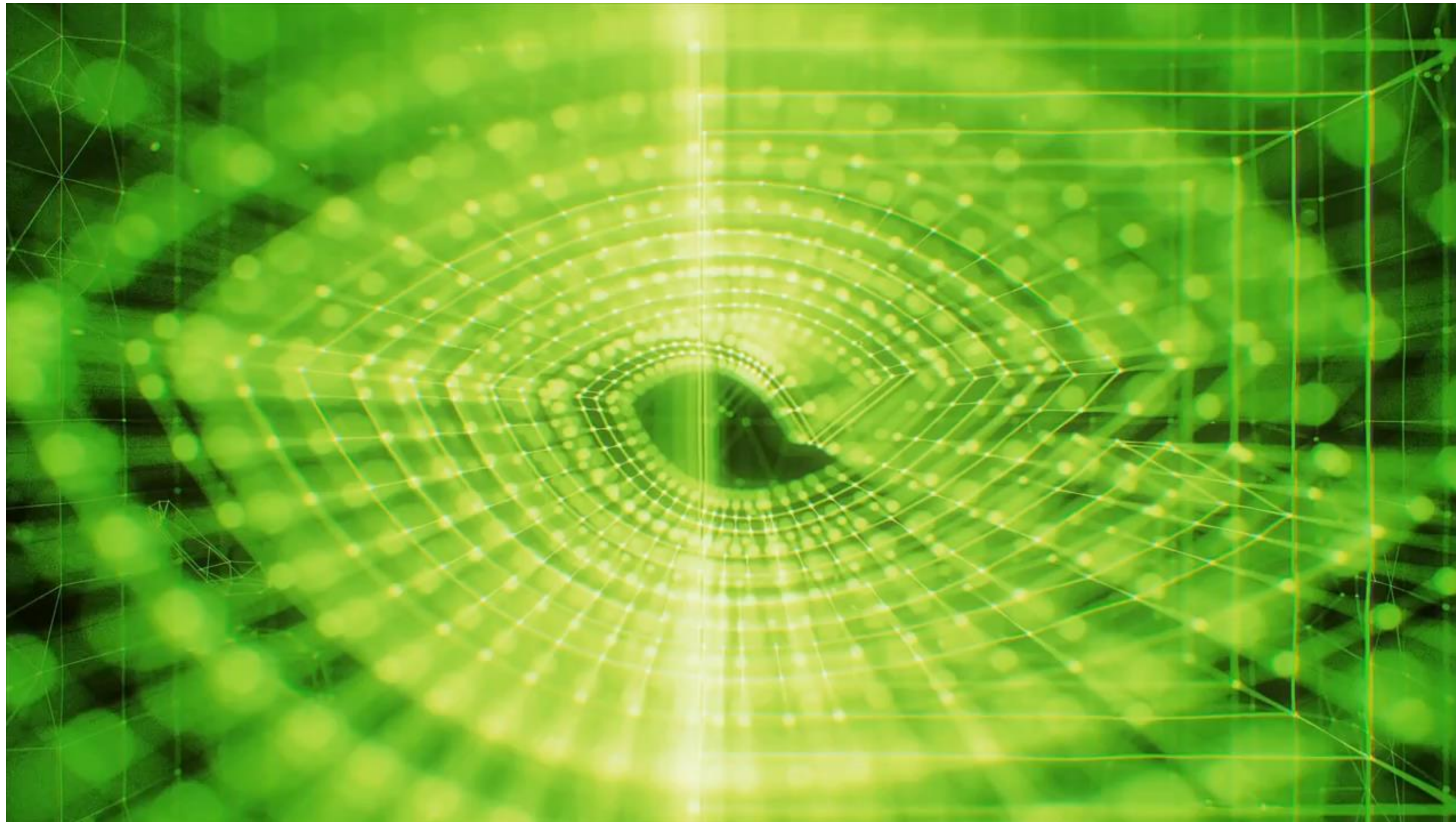
2017: Explosion of GANs



LSGAN, Zhu 2017.

Wasserstein GAN, Arjovsky 2017. Improved Wasserstein GAN, Gulrajani 2017

2017: Explosion of GANs

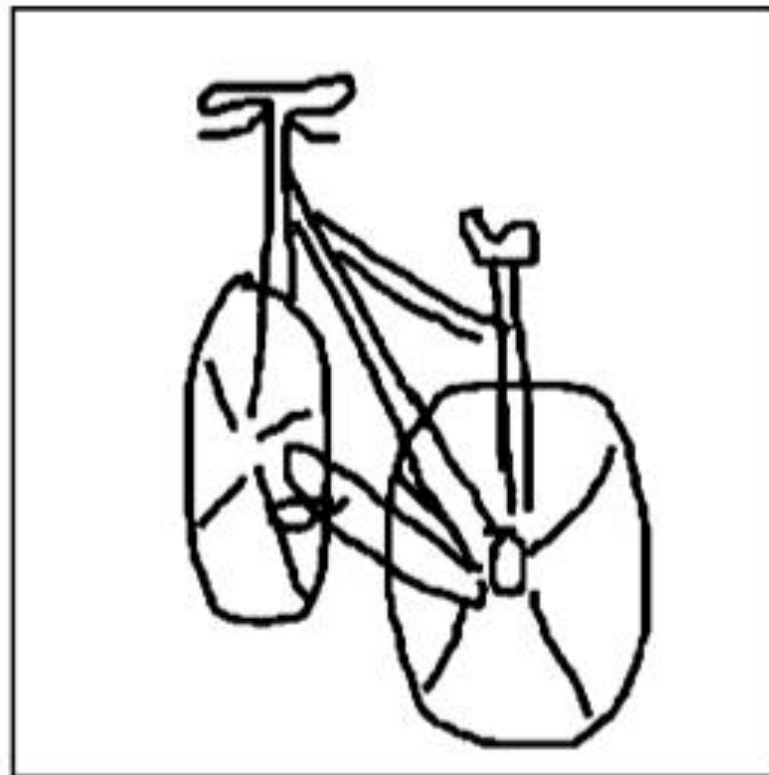


Generative style transfer

<https://thispersondoesnotexist.com/>

2017: Explosion of GANs

Input Sketch



“A photo of a bicycle”



“An origami bicycle”



“A bicycle in a snowy weather”



“A macro photo of a toy bicycle”

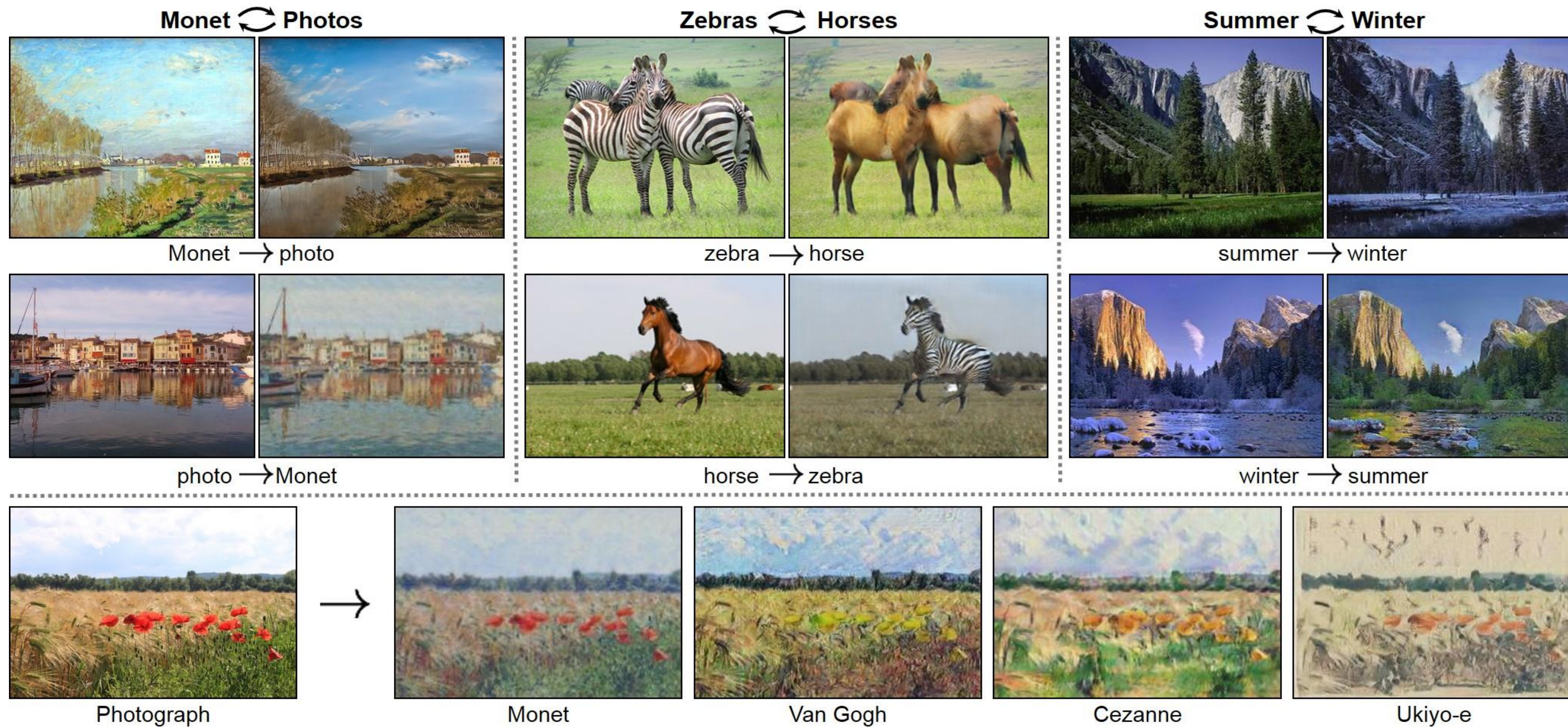


“A bicycle made of wood”



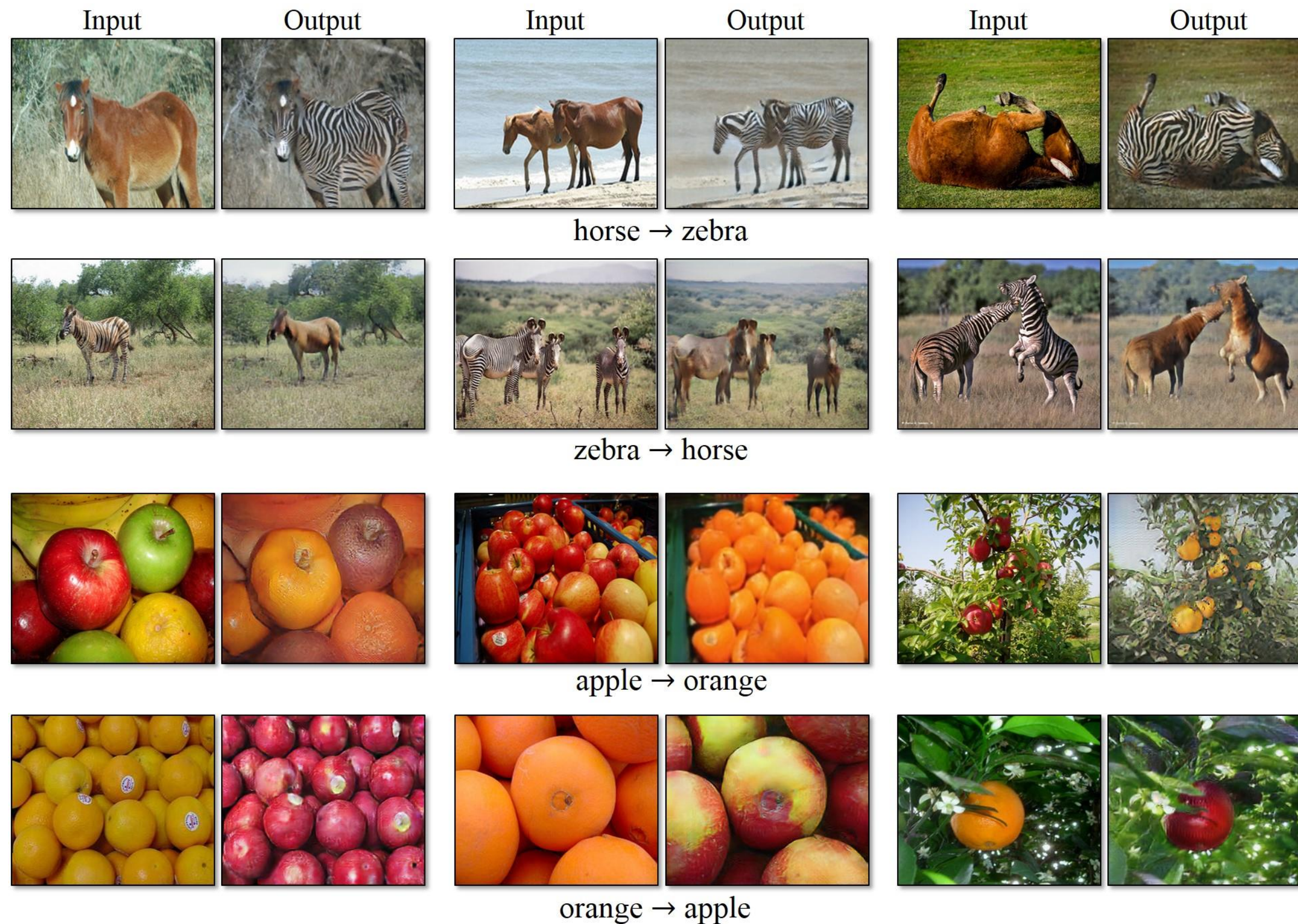
Sketch-Guided Text-to-Image

2017: Explosion of GANs

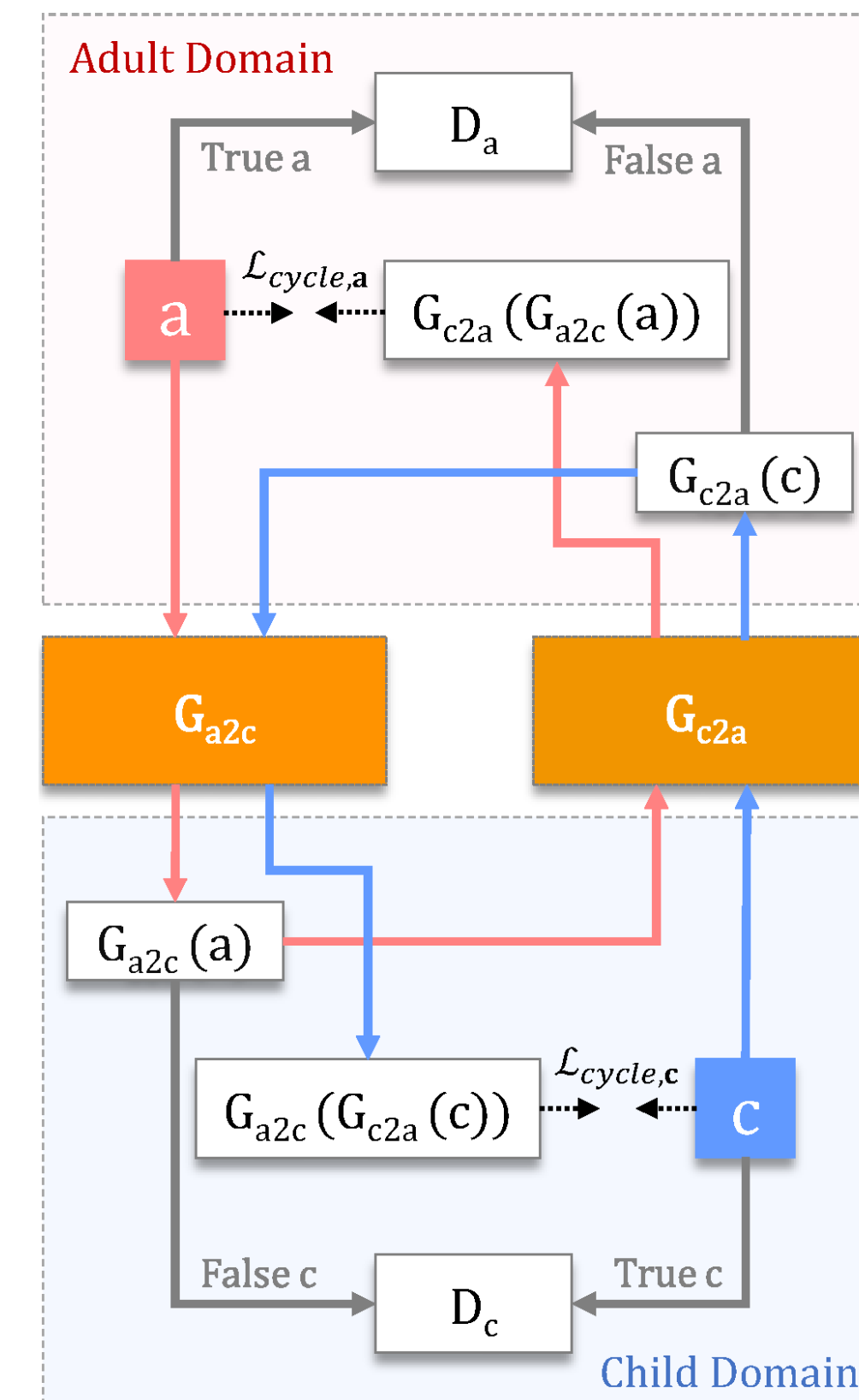


CycleGAN. Zhu et al. 2017.

2017: Explosion of GANs



CycleGAN. Zhu et al. 2017.



Adult2Child. Dong et al. 2019.

2019: BigGAN



Brock et al., 2019

Deep Learning Today: *Timeline of images generated by artificial intelligence*

2014



Goodfellow et al. (2014) – Generative Adversarial Networks

2015



Radford, Metz, and Chintala (2015) – Unsupervised Representation Learning with Deep Convolutional GANs

2016



Liu and Tuzel (2016) – Coupled GANs

2017



Karras et al. (2017) – Progressive Growing of GANs for Improved Quality, Stability, and Variation

2018



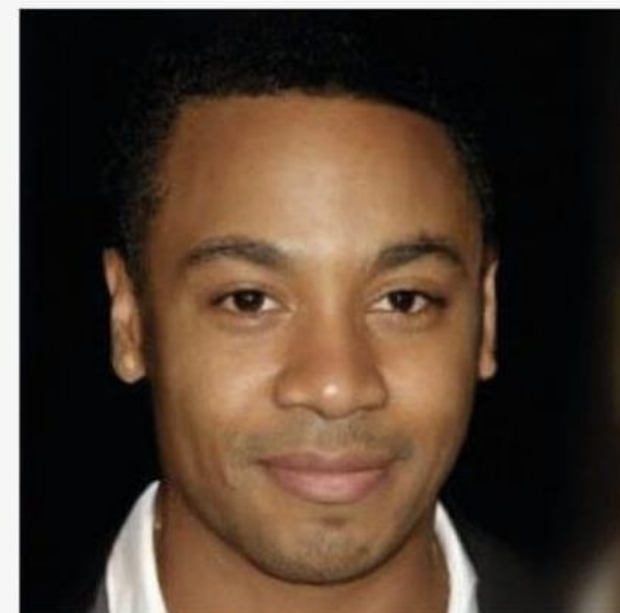
Karras, Laine, and Aila (2018) – A Style-Based Generator Architecture for Generative Adversarial Networks

2019



Karras et al. (2019) – Analyzing and Improving the Image Quality of StyleGAN

2020



Ho, Jain, & Abbeel (2020) – Denoising Diffusion Probabilistic Models

2021 Image generated with the prompt: "a couple of people are sitting on a wood bench"



Ramesh et al. (2021) – Zero-Shot Text-to-Image Generation (OpenAI's DALL-E 1)

2022 Image generated with the prompt: "A Pomeranian is sitting on the King's throne wearing a crown. Two tiger soldiers are standing next to the throne."



Saharia et al. (2022) – Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding (Google's Imagen)

Summary: GANs

Advantages of GANs:

1. **High-quality samples:** GANs are known for their ability to produce high-quality, diverse and realistic samples that resemble the real data. This is especially true for image and video synthesis, where GANs can produce photo-realistic images and videos.
2. **Unsupervised learning:** GANs can learn to generate new samples of data without requiring labeled training data. This makes them useful in applications where obtaining large amounts of labeled data is difficult or impossible.
3. **Data augmentation:** GANs can be used to generate additional training data, which can be useful in improving the performance of other machine learning models.

Summary: GANs

Disadvantages of GANs:

1. **Difficult to train:** GANs can be challenging to train, especially for large and complex datasets. The training process requires careful tuning of hyperparameters and can be unstable.
2. **Mode collapse:** GANs can suffer from mode collapse, where the generator produces a limited set of similar samples, rather than diverse and realistic samples.
3. **Evaluation:** It can be difficult to evaluate the quality of the generated samples, as there is no objective measure for how similar they should be to the real data.
4. **Limited to the training data:** GANs can only generate samples that are similar to the training data. This means that they may not be able to generate novel or creative samples that are different from the training data.

Overall, GANs are a powerful and versatile type of generative model that have both advantages and disadvantages. Their ability to generate high-quality and diverse samples without requiring labeled data makes them useful in many applications, but they can be challenging to train and evaluate, and may suffer from mode collapse

Summary: GANs

Don't work with an explicit density function

Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

- Trickier / more unstable to train
- Can't solve inference queries such as $p(x)$, $p(z|x)$

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications



Self-Supervised Learning



Self-supervised Learning

Self-supervised learning is a type of machine learning that involves training a model on a pretext task without the need for explicit supervision. In self-supervised learning, the model is trained to predict certain properties or relationships within the input data, such as predicting the next word in a sentence, filling in a missing part of an image, or predicting the rotation of an image.

The key idea behind self-supervised learning is to use the inherent structure or redundancy in the input data to provide the supervision signal for training. By using these pretext tasks, the model learns useful representations of the input data that can be used for downstream tasks.

Self-supervised Learning

One **advantage** of self-supervised learning is that it does not require large amounts of labeled data, which can be expensive and time-consuming to obtain. Instead, self-supervised learning can leverage large amounts of readily available unlabeled data to learn useful representations. Additionally, self-supervised learning can help to address the problem of domain shift, where the distribution of the test data differs from that of the training data, by learning representations that are more robust to changes in the input data distribution.

However, one **limitation** of self-supervised learning is that the quality of the learned representations is highly dependent on the quality of the pretext task. If the pretext task is not well-designed, the learned representations may not be useful for downstream tasks. Additionally, self-supervised learning can be computationally expensive, especially for large datasets or complex models.

Generative vs. Self-supervised Learning

- Both aim to learn from data without manual label annotation.
- Generative learning aims to model data distribution $p_{data}(x)$, e.g., generating realistic images.
- Self-supervised learning methods solve “pretext” tasks that produce good features for downstream tasks.
 - Learn with supervised learning objectives, e.g., classification, regression.
 - Labels of these pretext tasks are generated automatically

Generative vs. Self-supervised Learning



Left: Drawing of a dollar bill from memory. Right: Drawing subsequently made with a dollar bill present.

Learning to generate pixel-level details is often unnecessary; learn high-level semantic features with pretext tasks instead

Self-supervised Learning

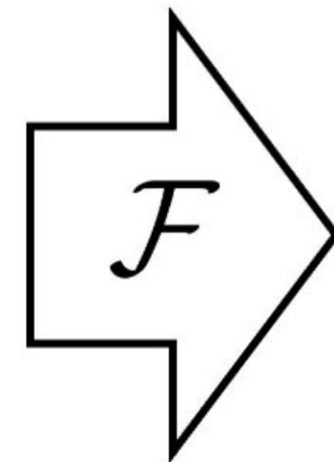
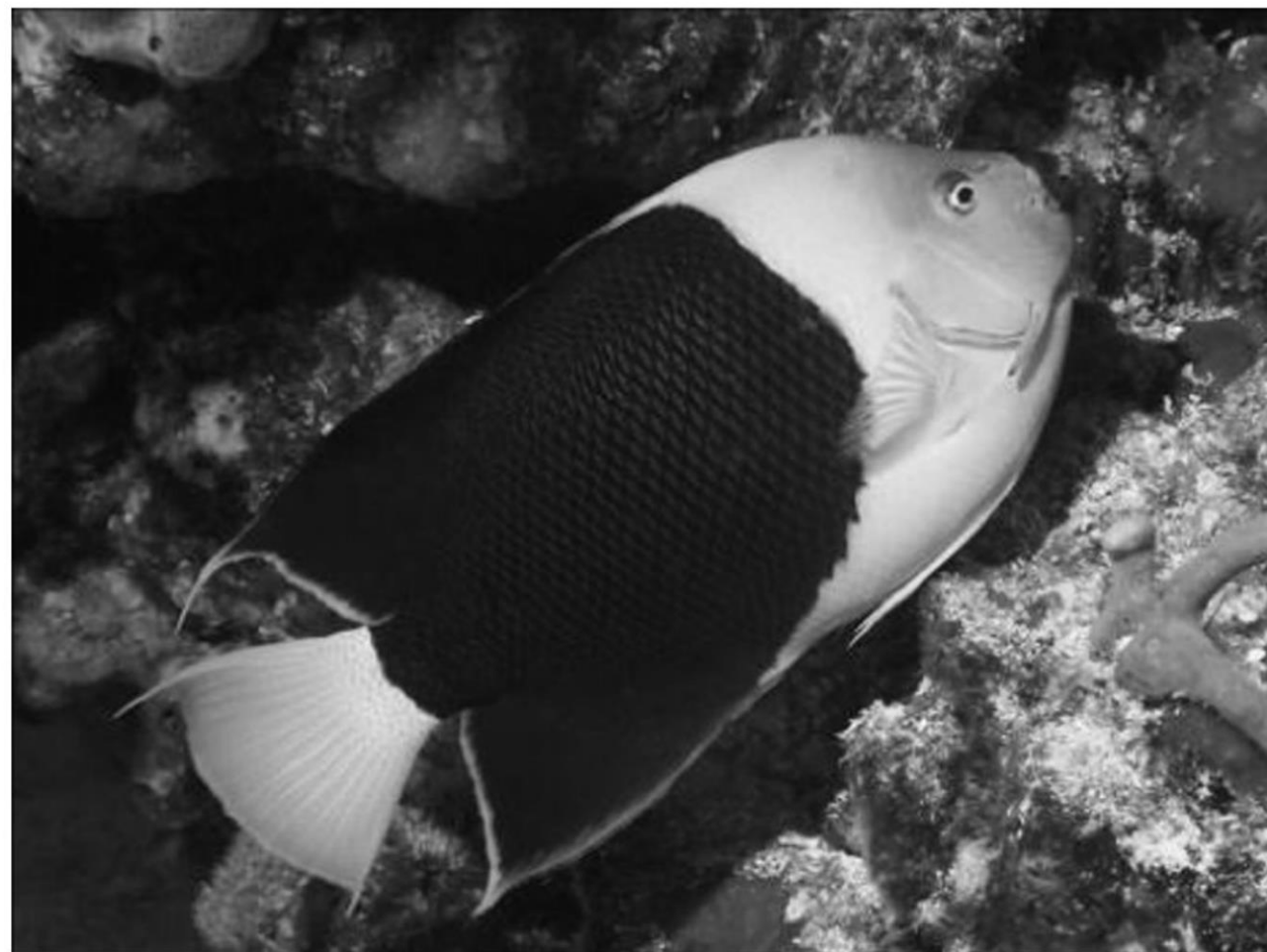
Pretext task: predict missing pixels (inpainting)



Context Encoders: Feature Learning by Inpainting (Pathak et al., 2016)

Self-supervised Learning

Pretext task: image coloring

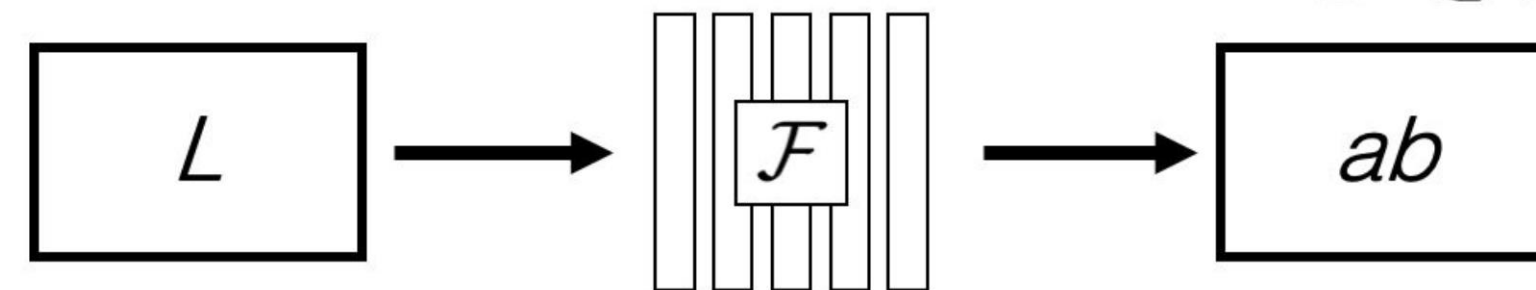


Grayscale image: L channel

$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

Color information: ab channels

$$\hat{\mathbf{Y}} \in \mathbb{R}^{H \times W \times 2}$$

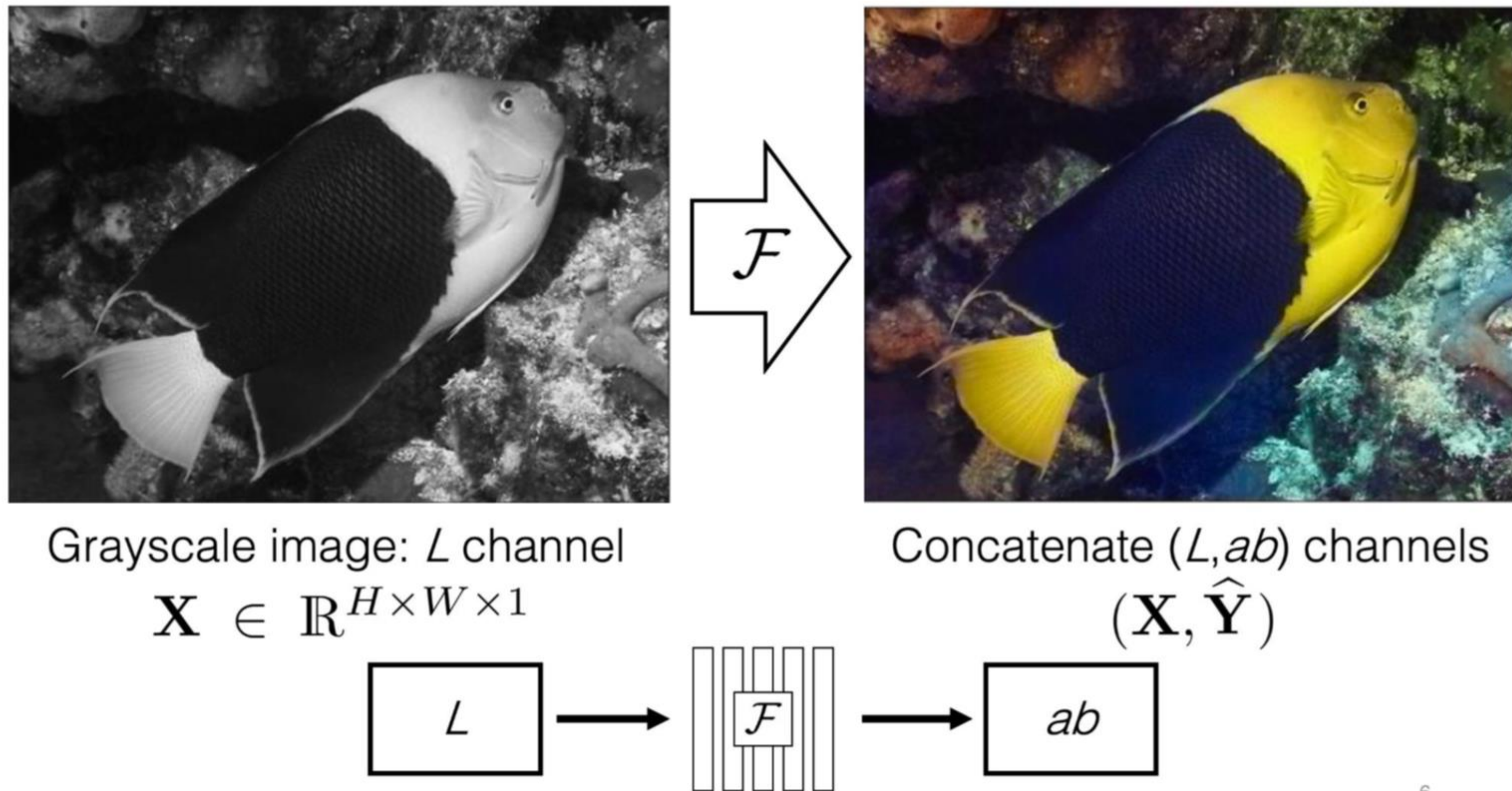


5



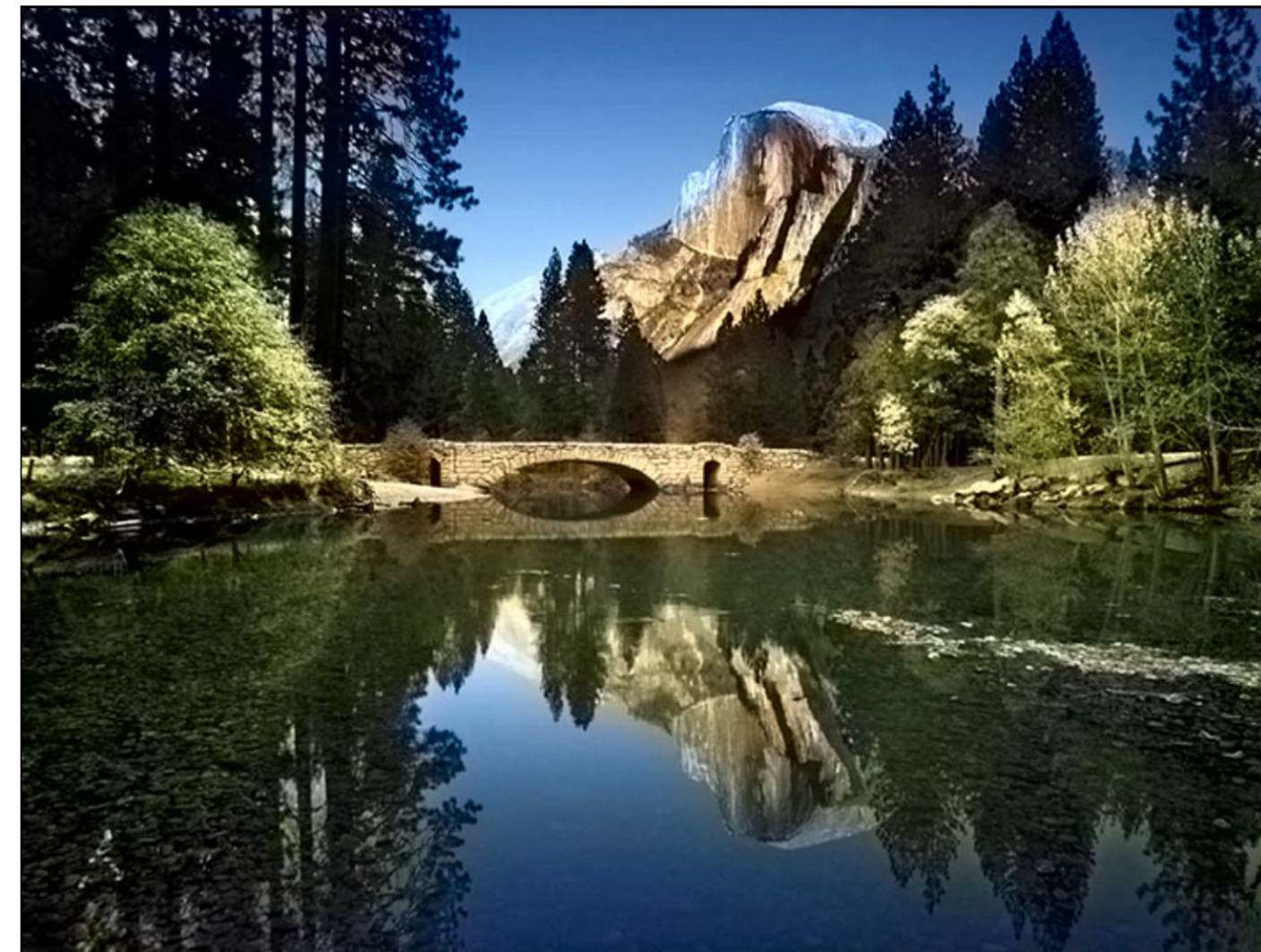
Self-supervised Learning

Pretext task: image coloring



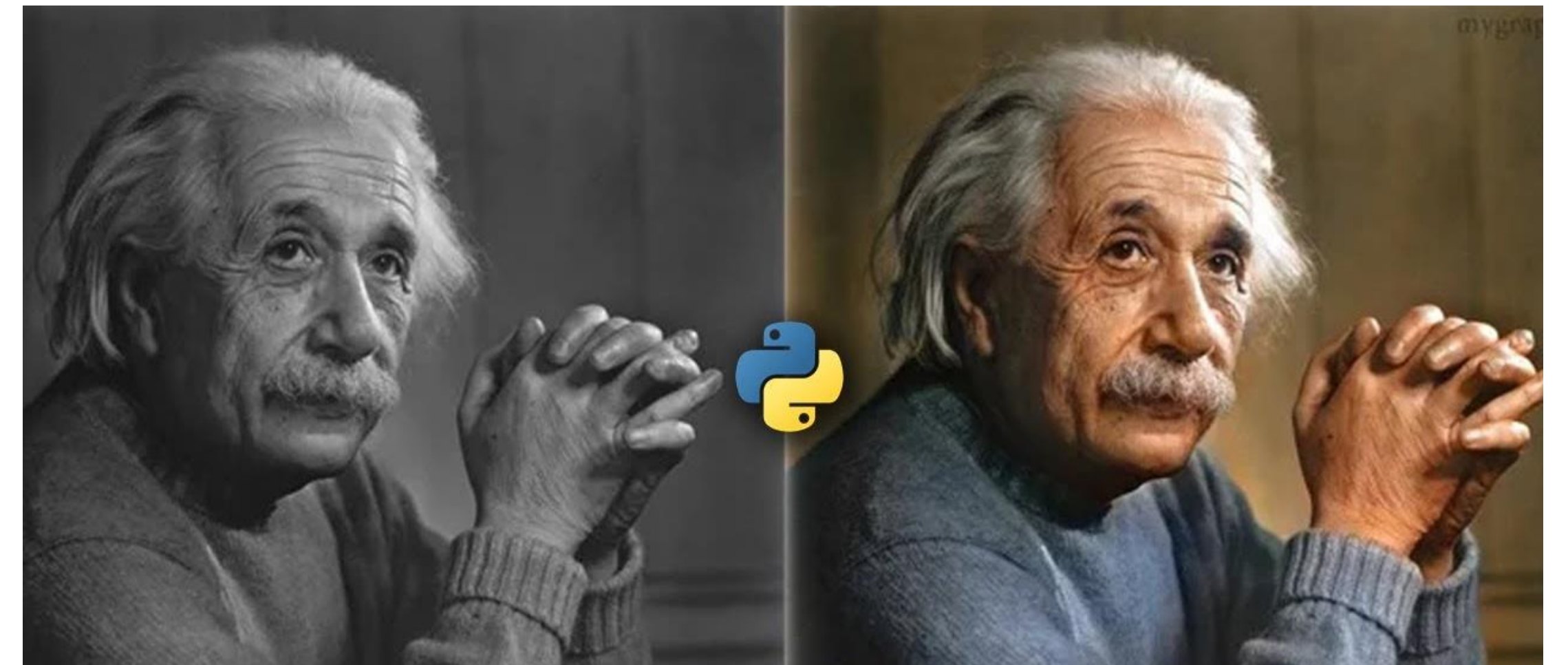
Self-supervised Learning

Pretext task: image coloring

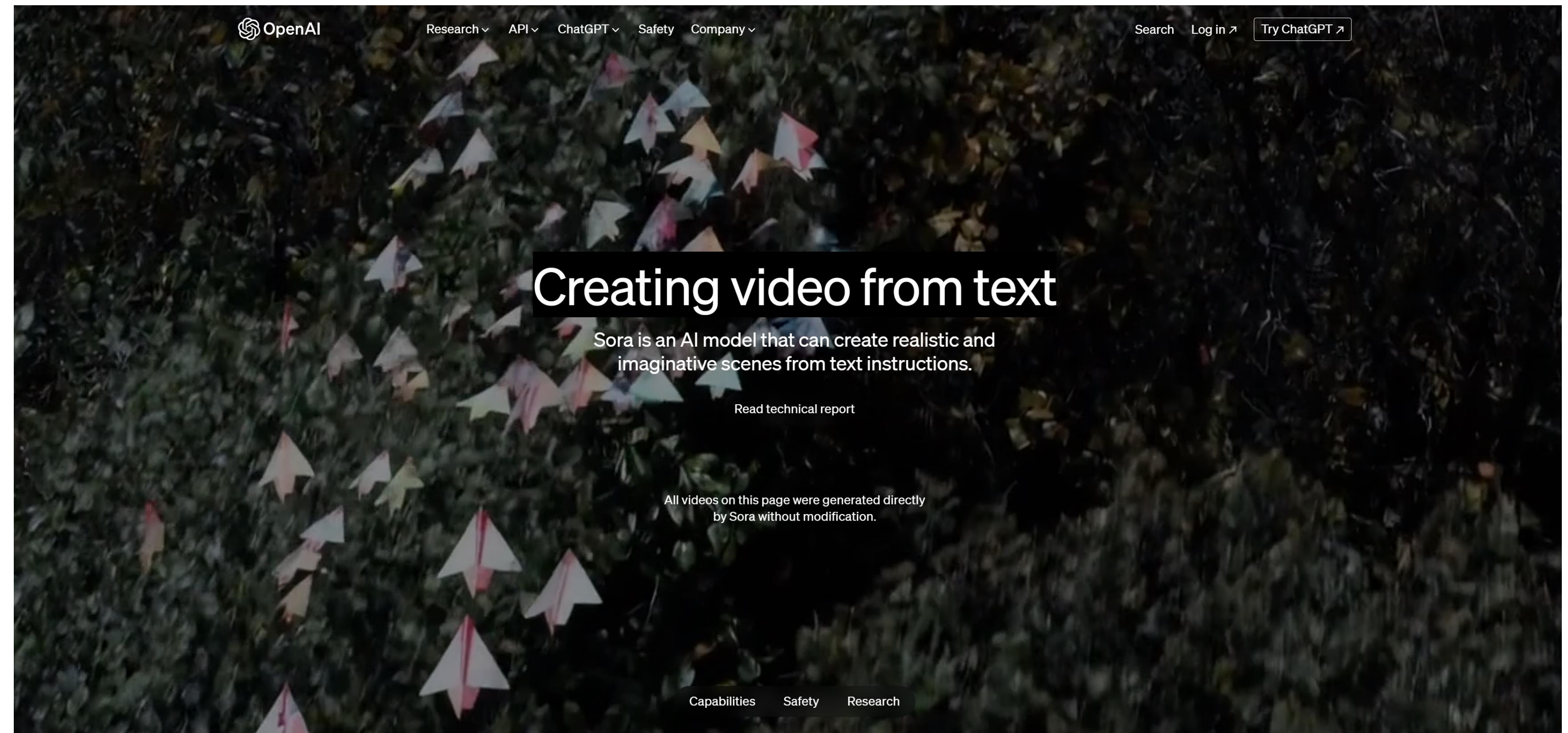


Self-supervised Learning

Pretext task: image coloring



SORA: Creating video from text



<https://openai.com/sora>





Thank you!

See you next week

