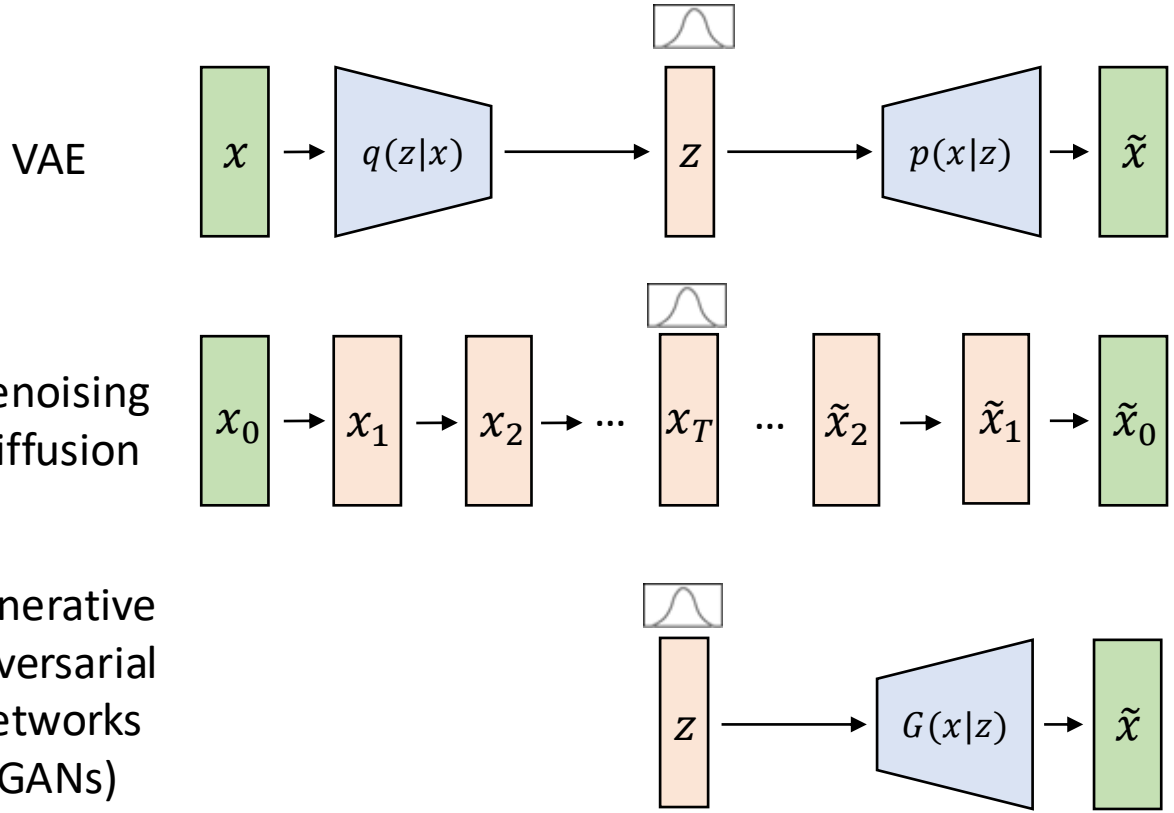
# CS 4644/7643: Lecture 16
# Danfei Xu

Topics:

- Generative Adversarial Networks
- Self-supervised Learning
  - **Pretext task from image transformation**
  - **Contrastive learning**

# Denoising Diffusion: Image to Noise and Back

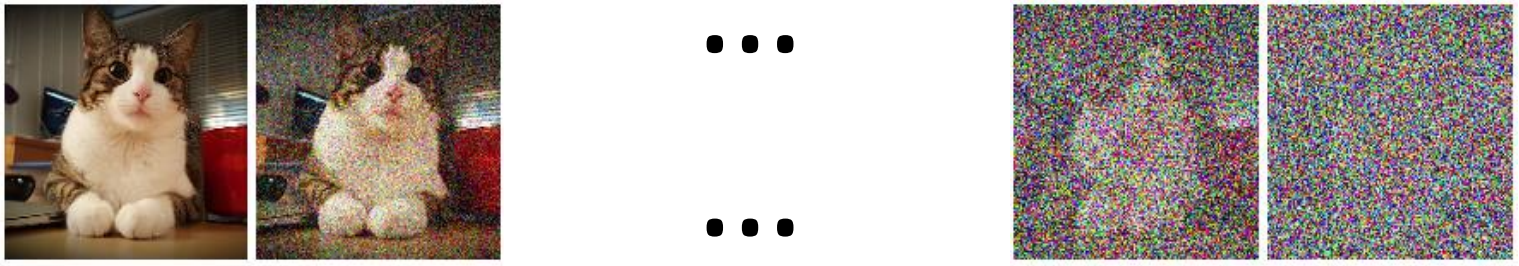


# The Denoising Diffusion Process

image from dataset

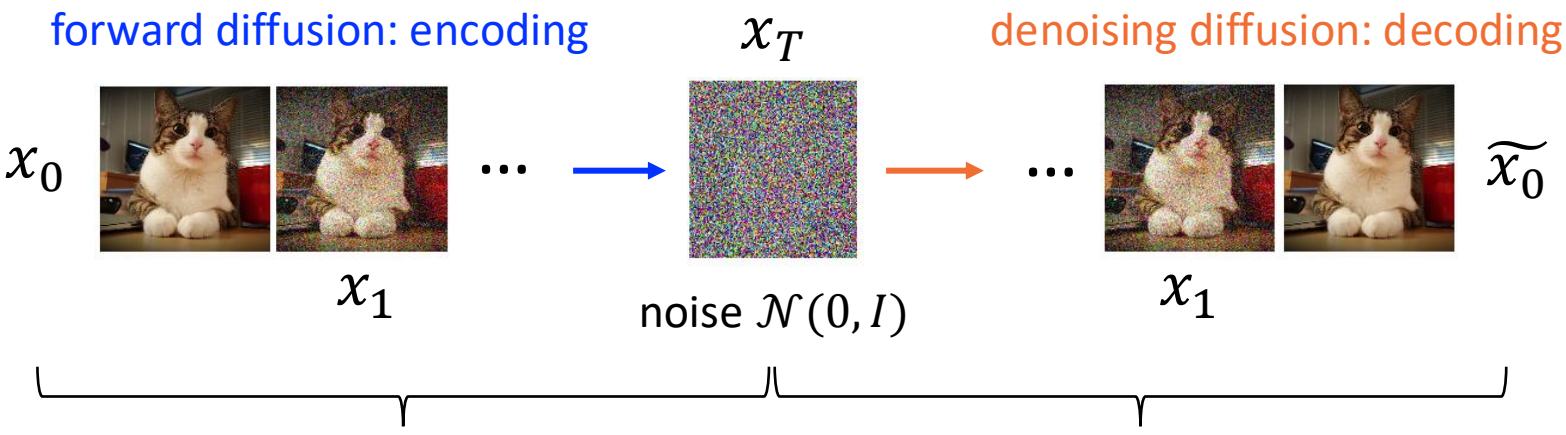
The "forward diffusion" process: add Gaussian noise each step

noise $\mathcal{N}(0, I)$




The "denoising diffusion" process: generate an image from noise by *denoising* the gaussian noises

# Connection to VAEs




Known / predefined:

$q(x_{1:T}|x_0)$

Unknown / learned:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

Similar to VAEs, use the denoising decoding process to generate new images.

# The Diffusion (Encoding) Process

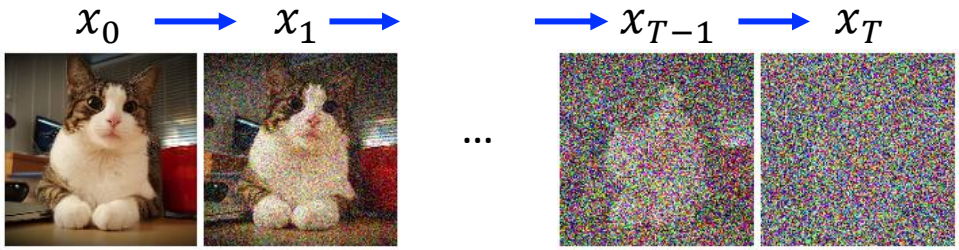
The **known** forward process $x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_T$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$ Probability Chain Rule (Markov Chain)

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; (1-\beta_t)x_{t-1}, \beta_t I)$$ Conditional Gaussian

$\beta_t$ is the *variance schedule* at the diffusion step $t$

$0 < \beta_1 < \beta_2 < \cdots < \beta_T < 1$, typical value range $[0.0001, 0.02]$, with $T = 1000$



# The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$ Probability Chain Rule (Markov Chain)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$ Conditional Gaussian

Want to learn time-dependent mean

Assume fixed / known variance (simplification)

How do we form a learning objective?

# The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

**High-level intuition**: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

**The learning objective**: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

What does it look like? $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right)$

$$\mu_q(t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right), \qquad \epsilon \sim \mathcal{N}(0, I)$$

Recall: Gaussian reparameterization trick

The "ground truth" noise that brought $x_{t-1}$ to $x_t$

# The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

**High-level intuition**: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

**The learning objective**: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

What does it look like? $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right)$

Assuming identical variance $\Sigma_q(t)$, we have:

$$\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) = \text{argmin}_\theta w||\mu_q(t) - \mu_\theta(x_t, t)||$$

Should be variance-dependent, but constant works better in practice

$p(x) = \int p(x|z)p(z)dz$ Intractable to estimate!

$$\log p(x) = \mathrm{E}_q\left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] + D_{KL}(q(z|x)||p(z|x))$$

$$\geq \mathrm{E}_q\left[\log \frac{p(x|Z)p(z)}{q(z|x)}\right]$$ Evidence Lower Bound (ELBO)

$$\log p(x_0) \geq \mathrm{E}_q\left[\log \frac{p(x_0|x_{1:T})p(x_{1:T})}{q(x_{1:T}|x_0)}\right] \qquad x = x_0,\ z = x_{1:T}$$

$$= \mathrm{E}_q\left[\log \frac{p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}{\prod_{t=1}^{T} q(x_t|x_{t-1})}\right]$$

… (derivation omitted, see Sohl-Dickstein *et al.,* 2015 Appendix B)

$$= -\mathrm{E}_q[D_{KL}(q(x_T|x_0)||p(x_T))] - \sum_{t=2}^{T} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) + \log p_\theta(x_0|x_1)$$

Maximize the agreement between the predicted reverse diffusion distribution $p_\theta$ and the "ground truth" reverse diffusion distribution $q$

Deep Unsupervised Learning using Nonequilibrium Thermodynamics, Sohl-Dickstein *et al.*, 2015

# Learning the Denoising Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma(t))$$

Conditional Gaussian

Learning objective: $\text{argmin}_\theta ||\mu_q(t) - \mu_\theta(x_t, t)||$

$$\mu_q(t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right), \qquad \epsilon \sim \mathcal{N}(0, I)$$

known during inference

Unknown during inference

Recall: this is the "ground truth" noise that brought $x_0$ to $x_t$

Idea: just learn $\epsilon$ with $\epsilon_\theta(x_t, t)$!

# Learning the Denoising Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma(t))$$ Conditional Gaussian

Simplified learning objective: $\text{argmin}_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||$

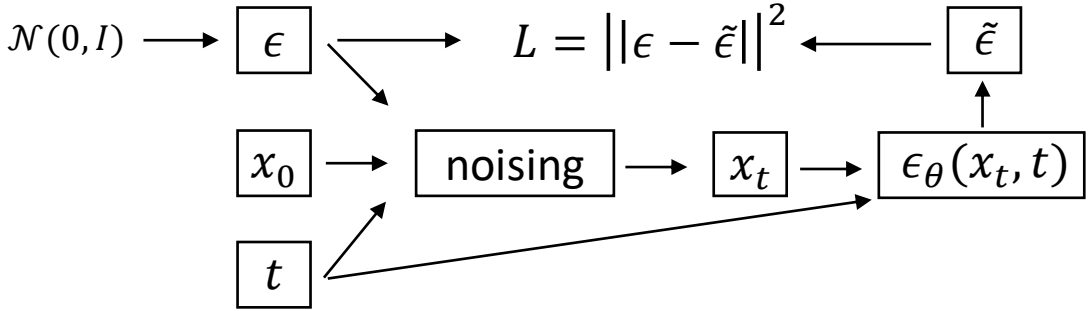
Inference time: $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon_\theta(x_t, t)\right)$

Predicted “denoising noise”

# The Denoising Diffusion Algorithm

**Algorithm 1** Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged




Compute regression loss

The Denoising Diffusion Probabilistic Models, Ho *et al.*, 2020

# The Denoising Diffusion Algorithm

**Algorithm 1** Training

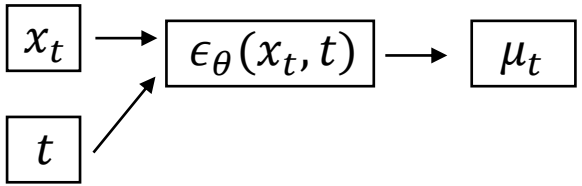
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

**Algorithm 2** Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

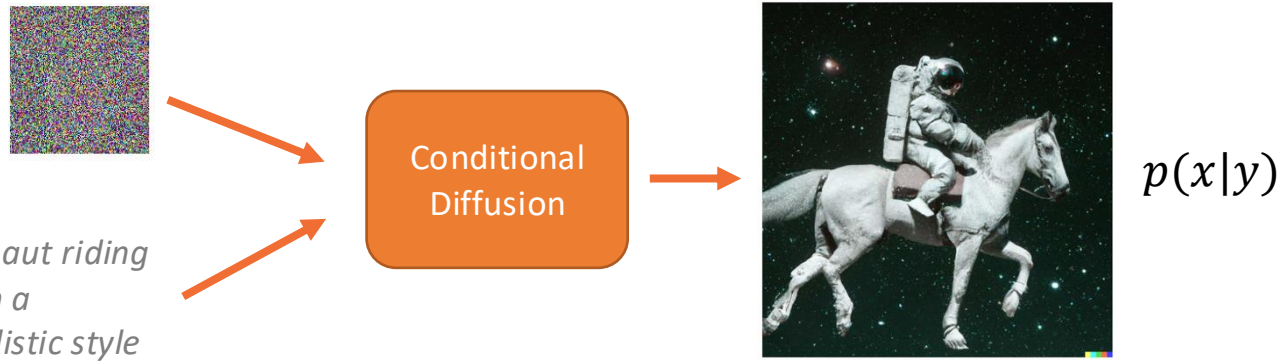
$\sigma_t = \sqrt{\beta}_t$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu(t), \Sigma(t))$$




The Denoising Diffusion Probabilistic Models, Ho *et al.*, 2020

# Conditional Diffusion Models

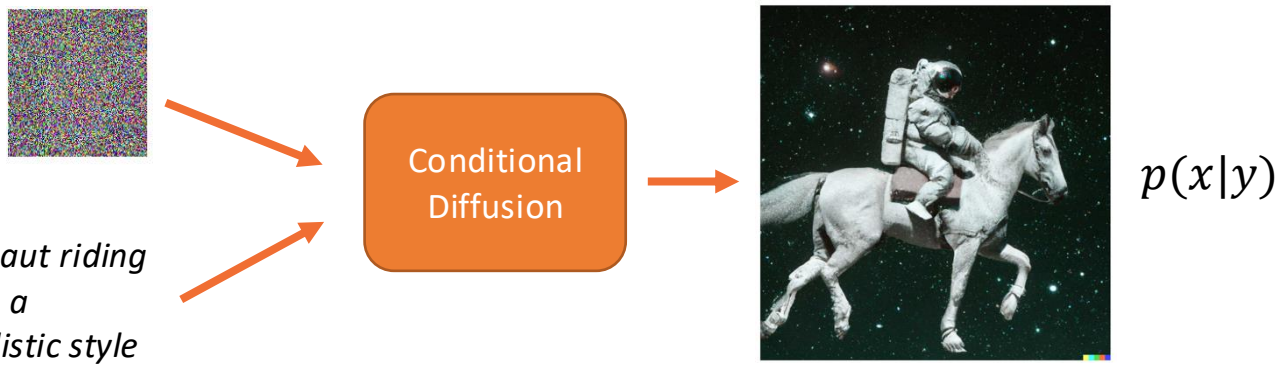



Simple idea: just condition the model on some text labels $y$!

$$\epsilon_\theta(x_t, y, t)$$

Problem: Very blurry generation

# Classifier-free Guided Diffusion




**Classifier-free Guided Diffusion**: estimate the gradient of the classifier model with conditional diffusion models!

$$\nabla_{x_t} \log f_\varphi(y|x_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}(\epsilon_\theta(x_t, t, y) - \epsilon_\theta(x_t, t))$$

$$\bar{\epsilon}_\theta(x_t, t, y) = (w+1)\epsilon_\theta(x_t, t, y) - w\epsilon_\theta(x_t, t)$$

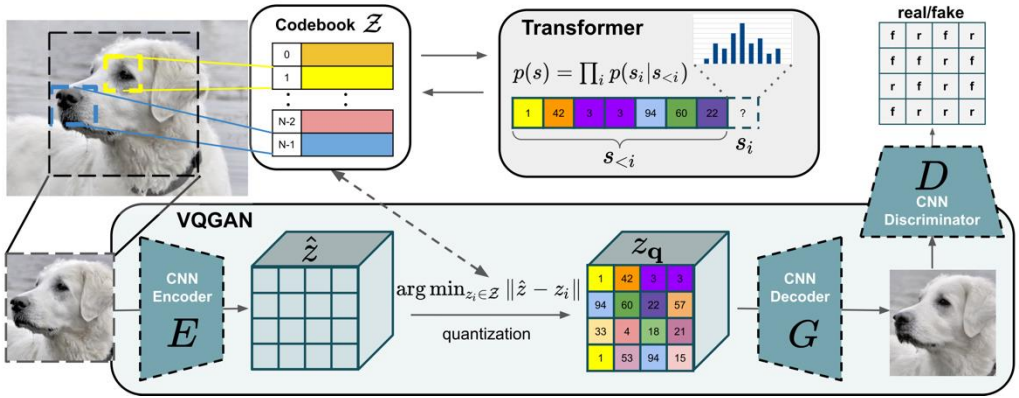
Linearly combine denoisers from an unconditional distribution and a conditional distribution

Ho and Salimans, 2022

# Latent-space Diffusion

Problem: Hard to learn diffusion process on high-resolution images

Solution: learn a low-dimensional latent space using a ViT-based autoencoder and *do diffusion on the latent space*!

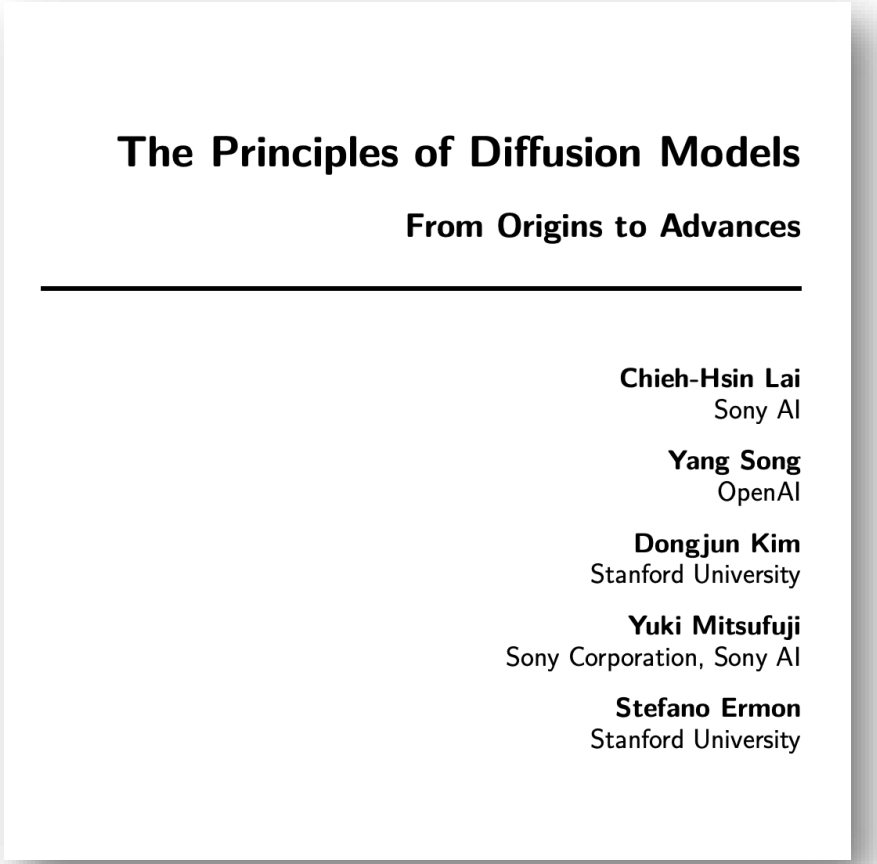



The latent space autoencoder

Esser and Rombach *et al.*, 2021

# Hot off arXiv!

https://www.arxiv.org/pdf/2510.21890



**The Principles of Diffusion Models**

**From Origins to Advances**

**Chieh-Hsin Lai**
Sony AI

**Yang Song**
OpenAI

**Dongjun Kim**
Stanford University

**Yuki Mitsufuji**
Sony Corporation, Sony AI

**Stefano Ermon**
Stanford University

# Summary

- Denoising Diffusion model is a type of generative model that learns the process of “denoising” a known noise source (Gaussian).
- We can construct a learning problem by deriving the evidence lower bound (ELBO) of the denoising process.
- The learning objective is to minimize the KL divergence between the “ground truth” and the learned denoising distribution.
- A simplified learning objective is to estimate the noise of the forward diffusion process.
- The diffusion process can be guided to generate targeted samples.
- Can be applied to many different domains. Same underlying principle.
- Very hot topic!

# Taxonomy of Generative Models

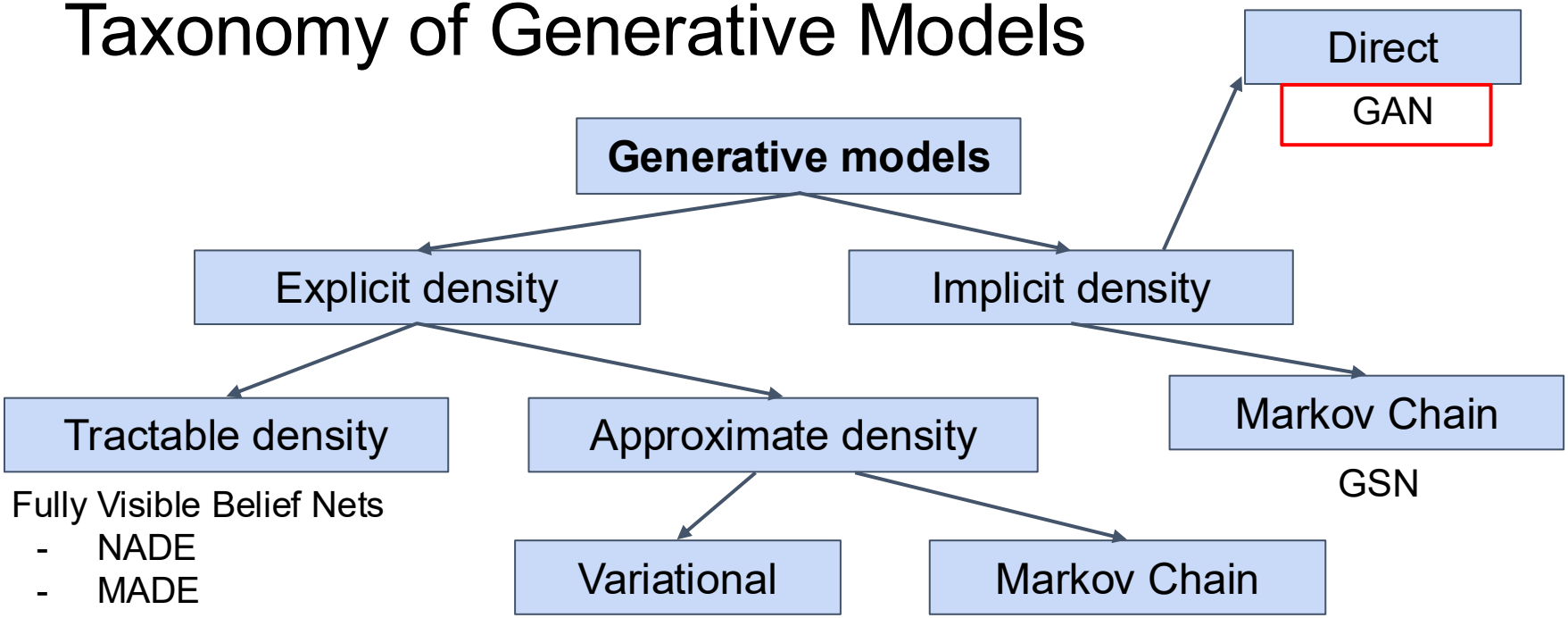
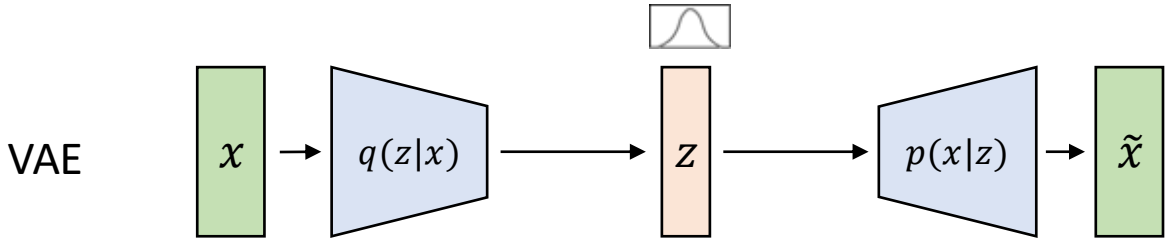
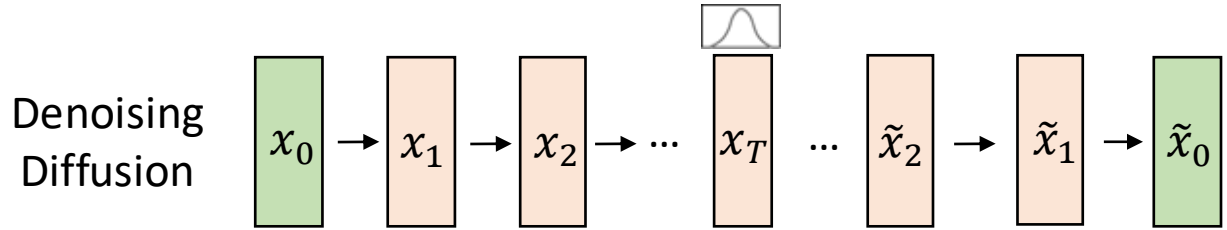


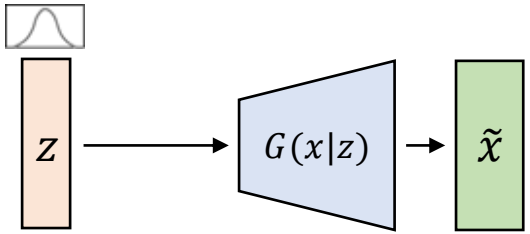

Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# GANs: Learning generate samples directly




Generative
Adversarial
Networks
(GANs)



# Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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

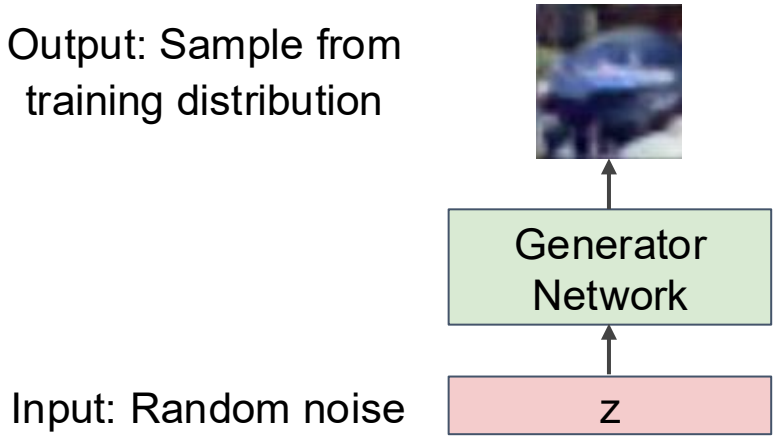
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

# Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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.



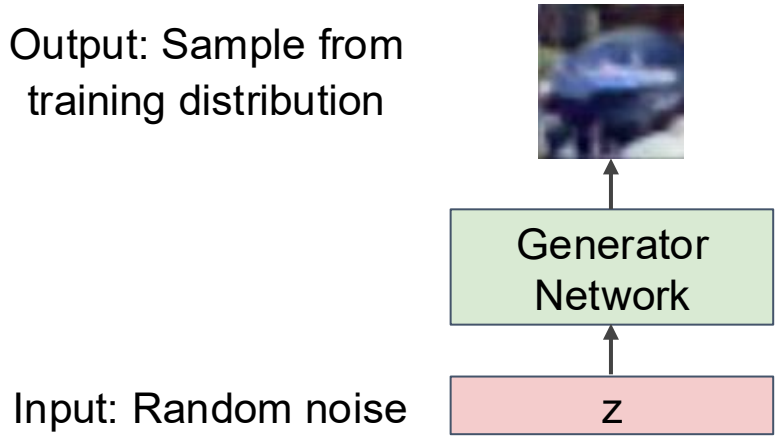
# Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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.

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



# Generative Adversarial Networks

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

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

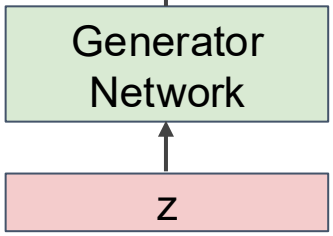
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

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

Output: Sample from training distribution



**Objective**: generated images should look "real"




Input: Random noise

# Generative Adversarial Networks

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

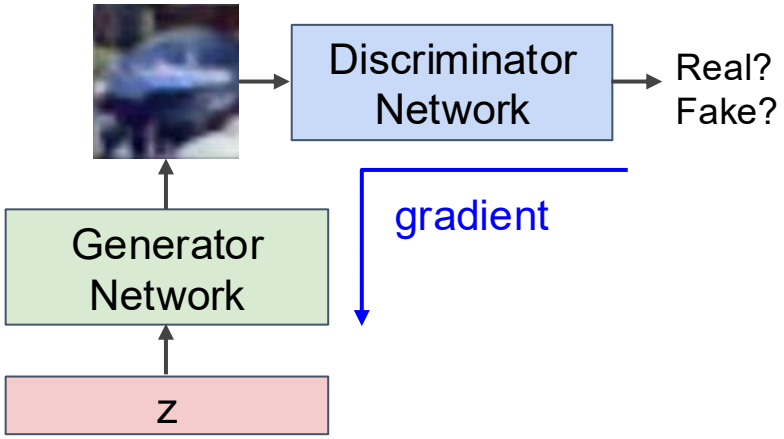
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

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

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

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

Output: Sample from training distribution




Input: Random noise

# Training GANs: Two-player game

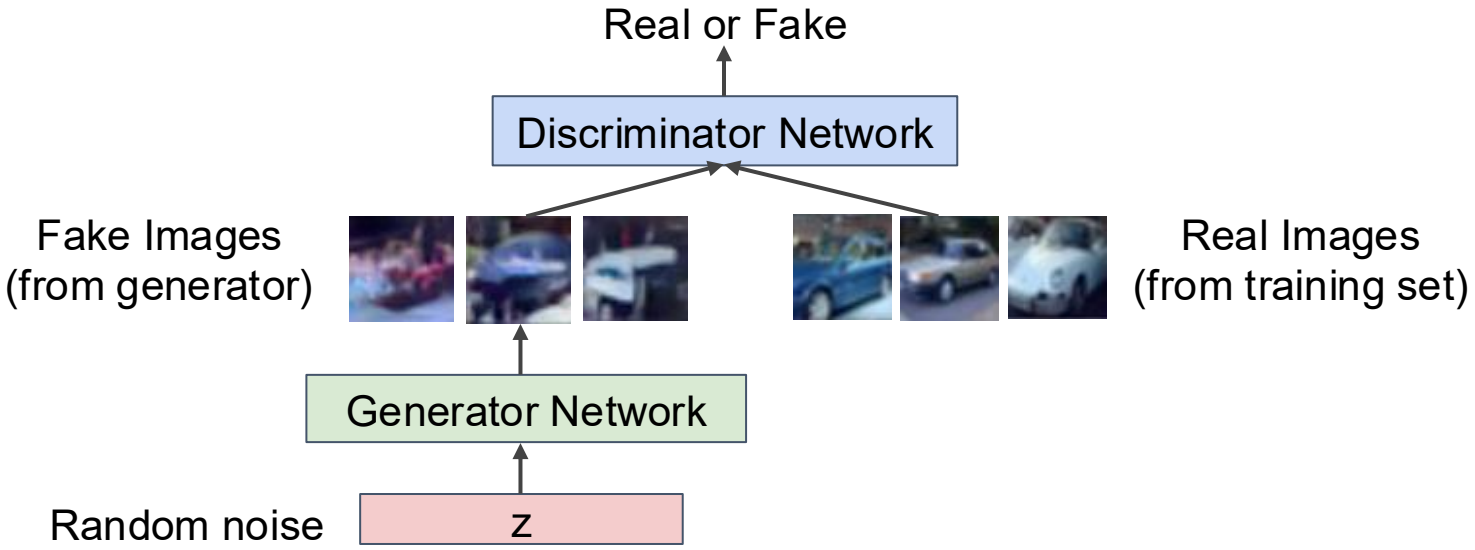
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

**Discriminator network**: try to distinguish between real and fake images
**Generator network**: try to fool the discriminator by generating real-looking images

# Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

**Discriminator network**: try to distinguish between real and fake images
**Generator network**: try to fool the discriminator by generating real-looking images

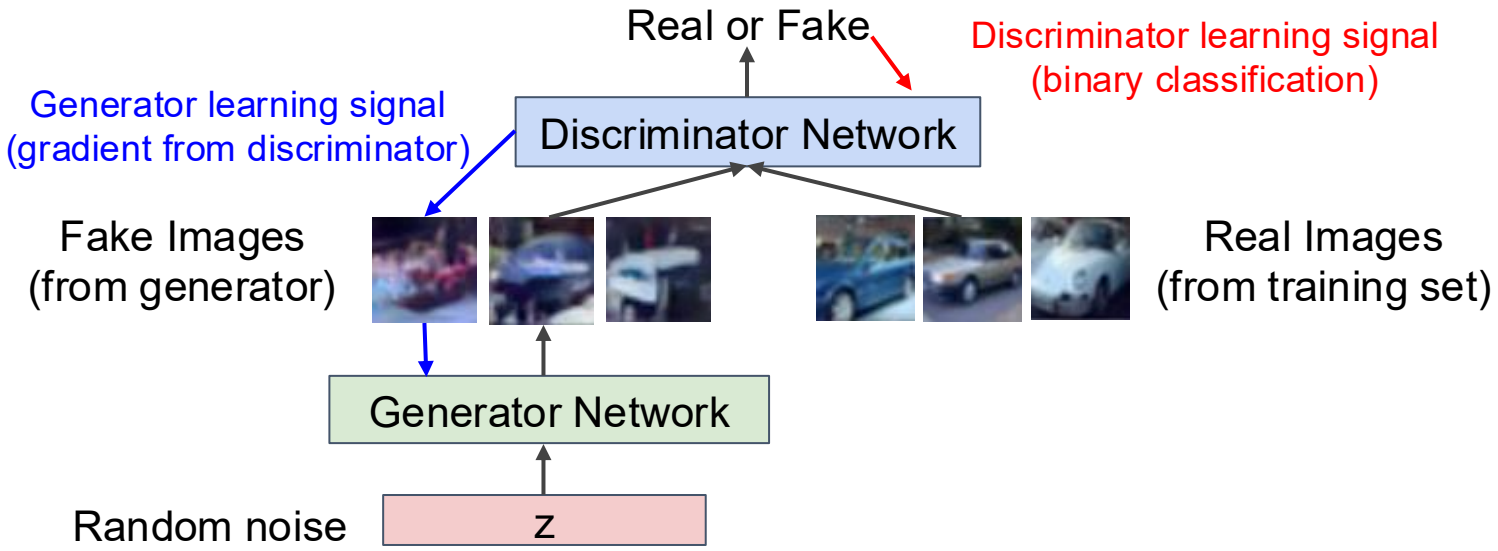



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

# Training GANs: Two-player game

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

**Discriminator network**: try to distinguish between real and fake images
**Generator network**: try to fool the discriminator by generating real-looking images




Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

# Training GANs: Two-player game

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

**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:

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

Generator objective

Discriminator objective

# Training GANs: Two-player game

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

**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

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

Discriminator output for real data x

training data x

Classify all real images as real

# Training GANs: Two-player game

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

**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

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

- $\max_{\theta_d}$: Maximize discriminator
- $\mathbb{E}_{x \sim p_{data}}$: training data x
- $D_{\theta_d}(x)$: Discriminator output for real data x
- $\mathbb{E}_{z \sim p(z)}$: noise z
- $D_{\theta_d}(G_{\theta_g}(z))$: Discriminator output for generated fake data G(z)
- $\min_{\theta_g}$: Minimize generator

Classify all real images as real

Classify all generated images as fake

# Training GANs: Two-player game

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

**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**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

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

Generator: learn to fool discriminator. Minimize $\log(1 - p_{\theta_d}(x_{gen}))$

# Training GANs: Two-player game

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

**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**

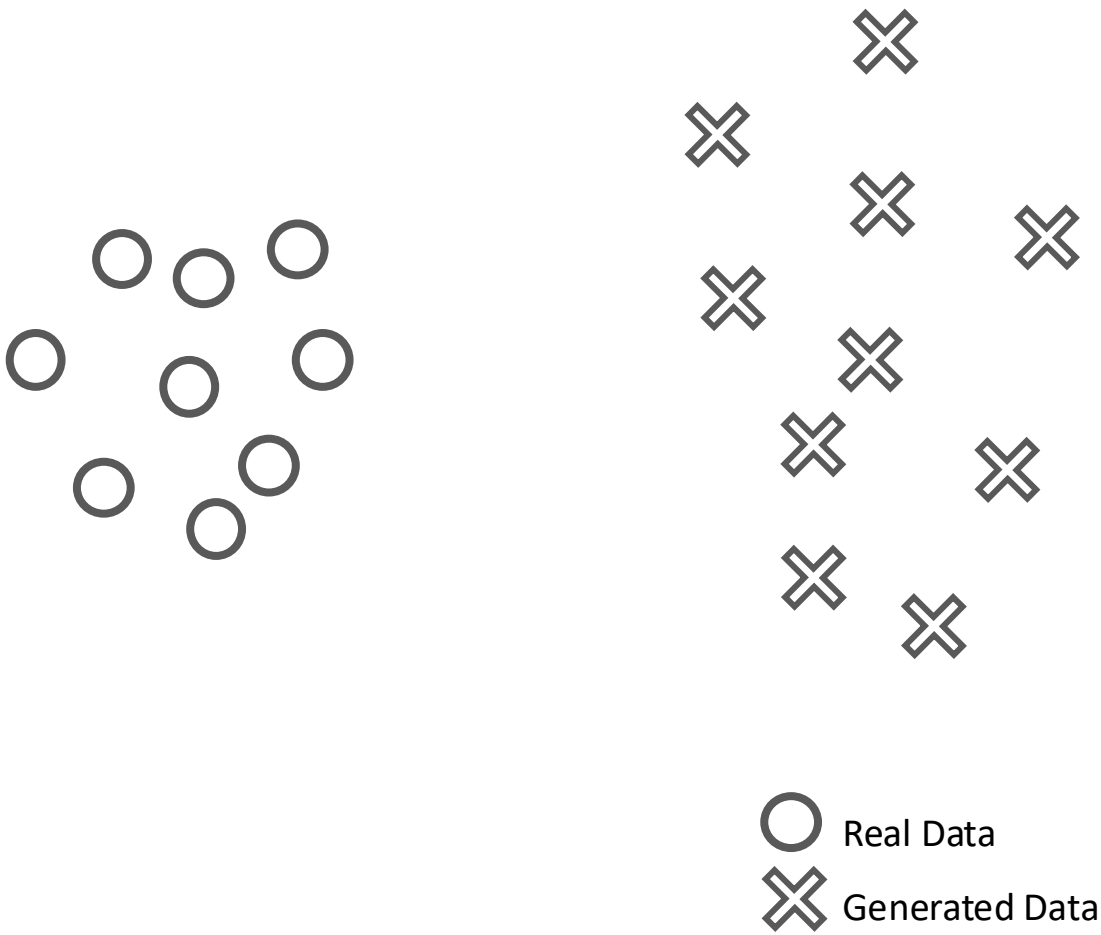
Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

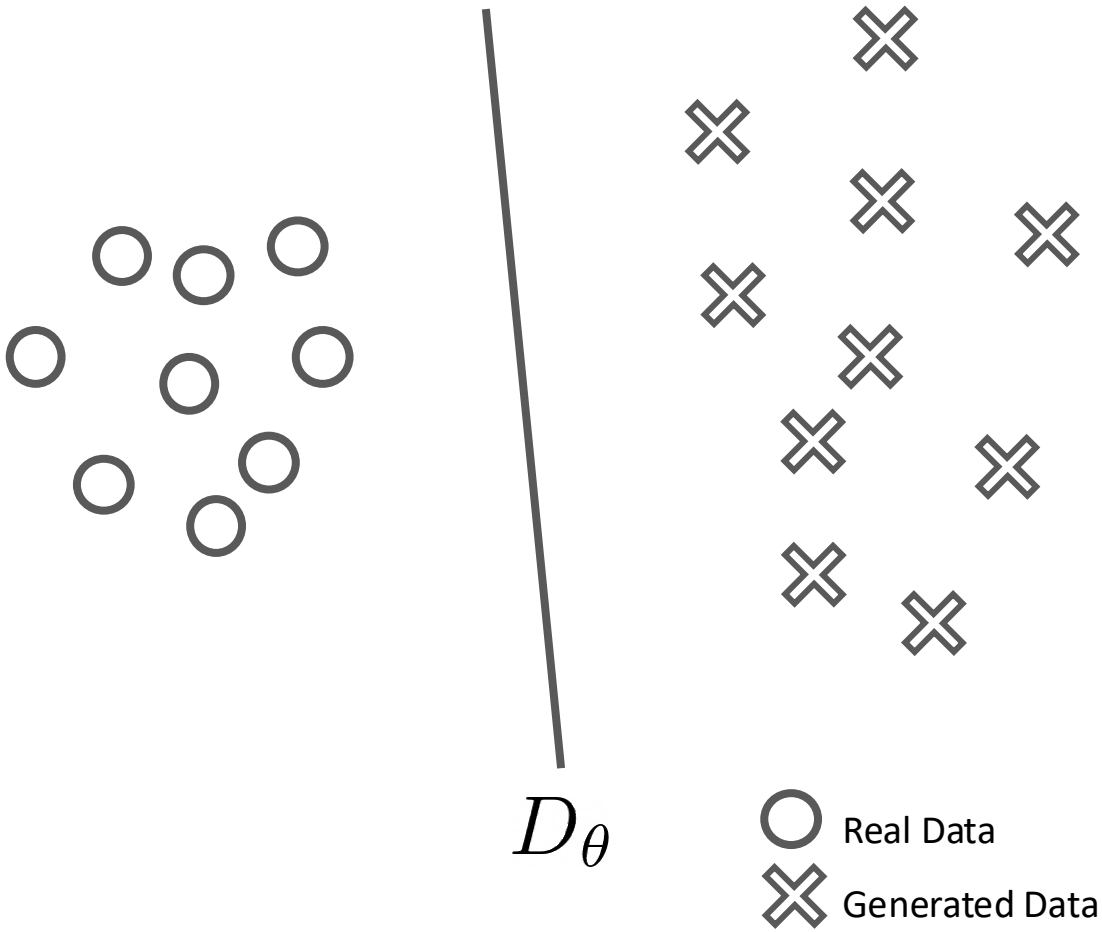
$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

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

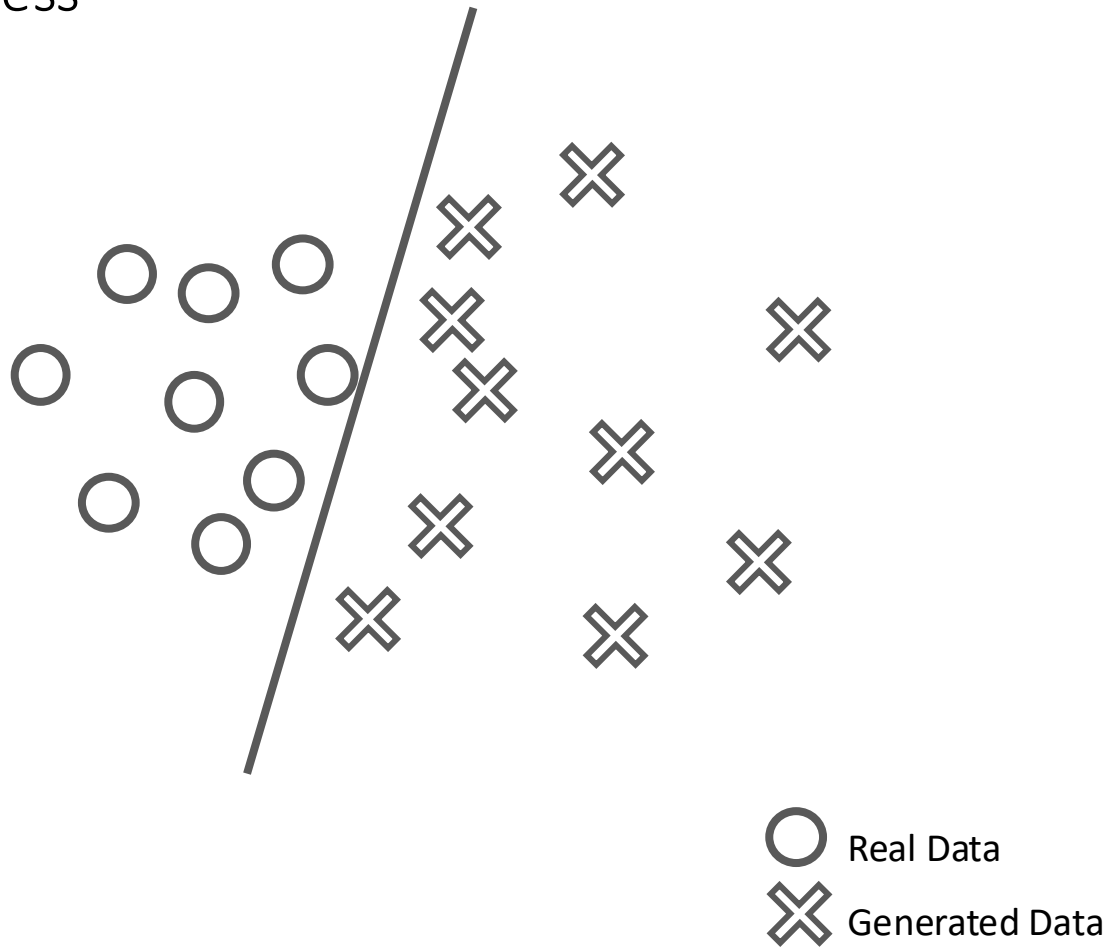
# GAN Learning Process



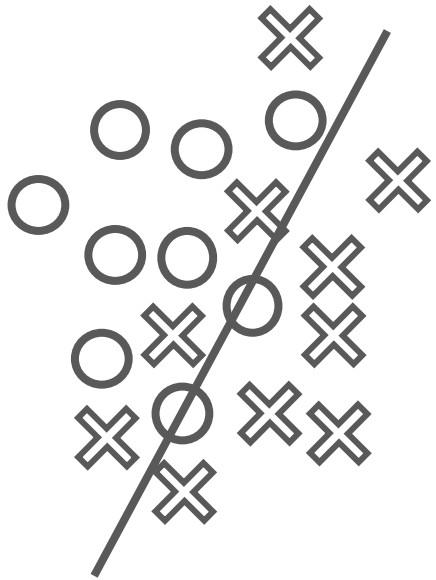
# GAN Learning Process



# GAN Learning Process



# GAN Learning Process



Real Data

Generated Data

# Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

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

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

# Training GANs: Two-player game

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

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

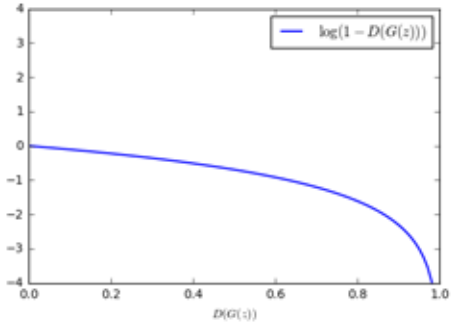
$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

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!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



# Training GANs: Two-player game

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

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

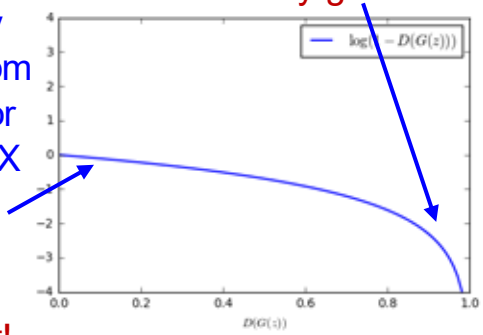
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

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

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



# Training GANs: Two-player game

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

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:

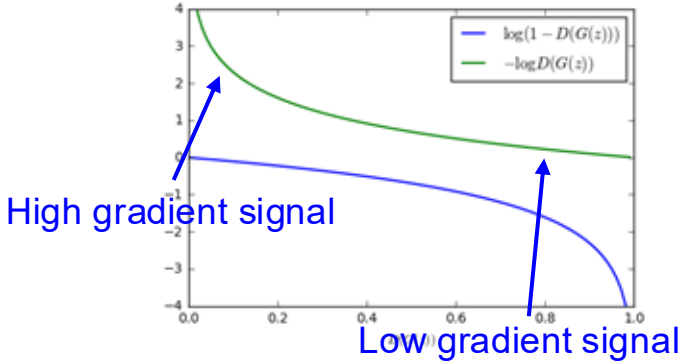
1. **Gradient ascent** on discriminator

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

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.



# Training GANs: Two-player game

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

## Putting it together: GAN training algorithm

**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:

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

**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 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**

# Training GANs: Two-player game

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

## Putting it together: GAN training algorithm

**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:

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

Update discriminator

  **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 ascending its stochastic gradient (improved objective):

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

Update generator

**end for**

Some find k=1 more stable, others use k > 1, no best rule.

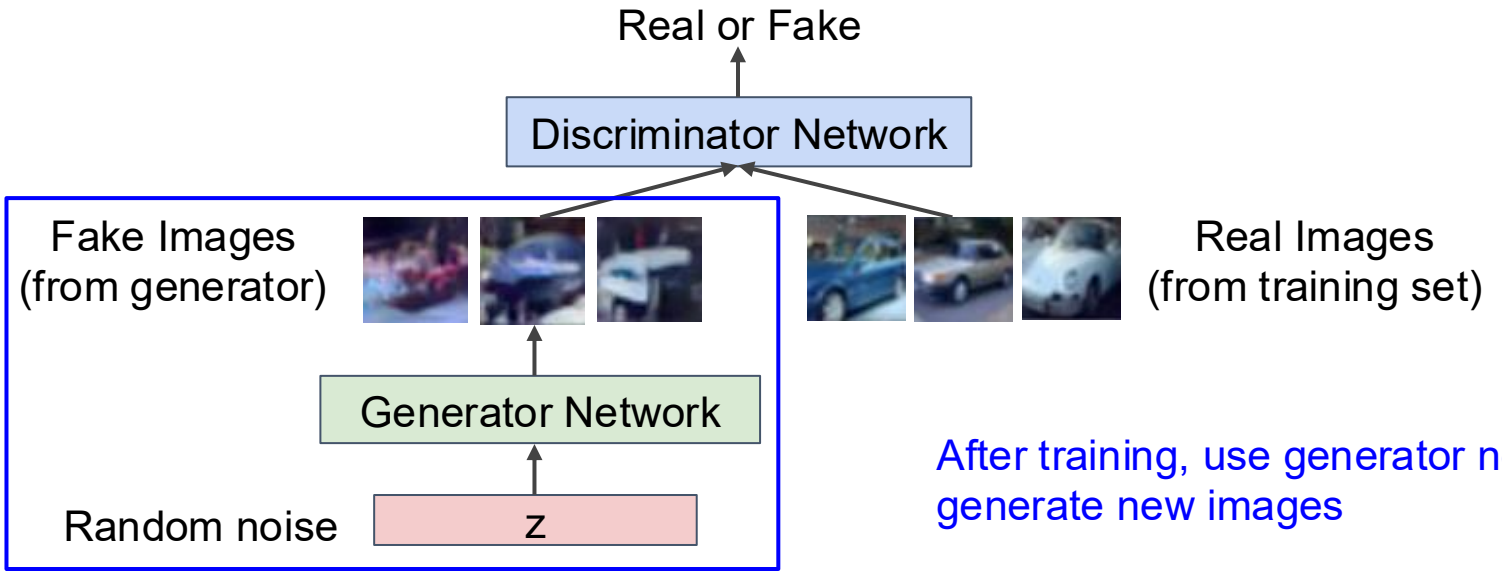
Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

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

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

**Generator network**: try to fool the discriminator by generating real-looking images
**Discriminator network**: try to distinguish between real and fake images




After training, use generator network to generate new images

Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

# Generative Adversarial Nets

## Generated samples





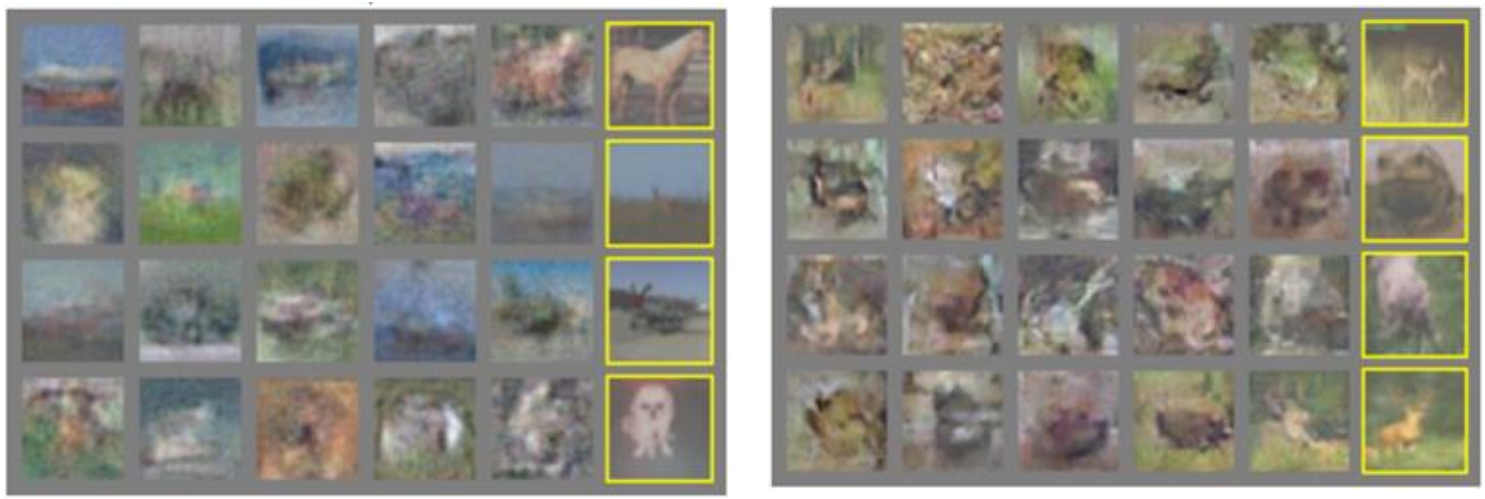
Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

# Generative Adversarial Nets

## Generated samples (CIFAR-10)



Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

# Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

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.

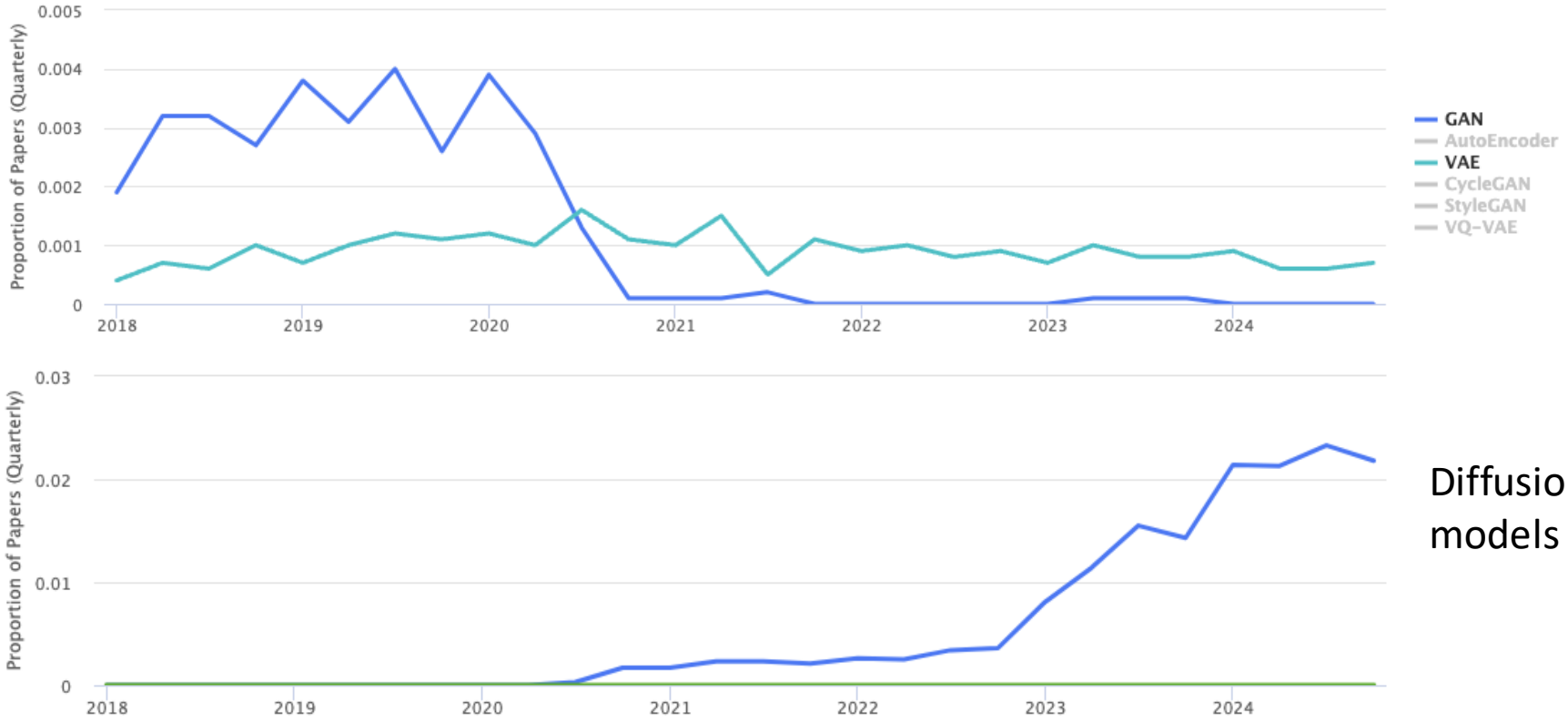
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

# 2019: BigGAN



Brock et al., 2019

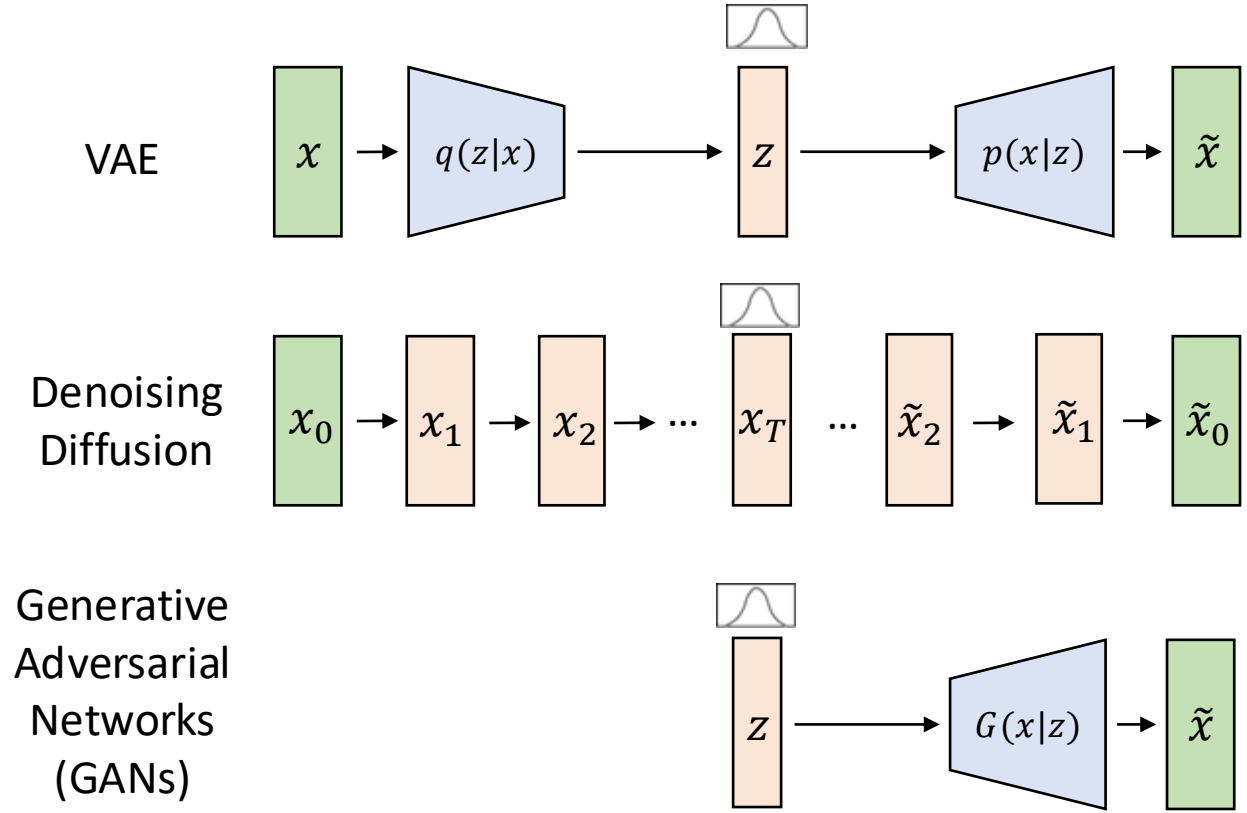
# GANs were popular …




Diffusion models

Source: https://paperswithcode.com

# Deep Generative Models



# Generative Models: Closing Thoughts

- Learn without supervision = ability to leverage large, raw dataset
- Realism: Generate plausible samples given dataset
- Diversity: Generate diverse samples (avoid mode collapse)
- Controllability: Generate based on instruction / conditioning
- Healthy combination of theory and deep learning magic
- Generative Modeling is extremely hot in 2025. More will come …

## Supervised Learning

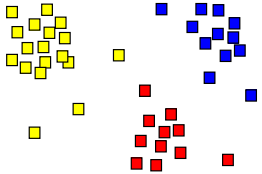
- Train Input: $\{X, Y\}$
- Learning output: $f : X \rightarrow Y$, $P(y|x)$
- e.g. classification



Sheep
Dog
Cat
Lion
Giraffe

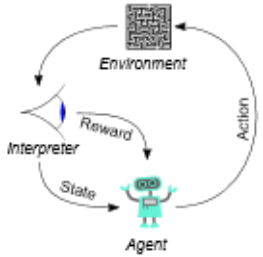
## Unsupervised Learning

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



## Reinforcement Learning

- Evaluative feedback in the form of **reward**
- No supervision on the right action

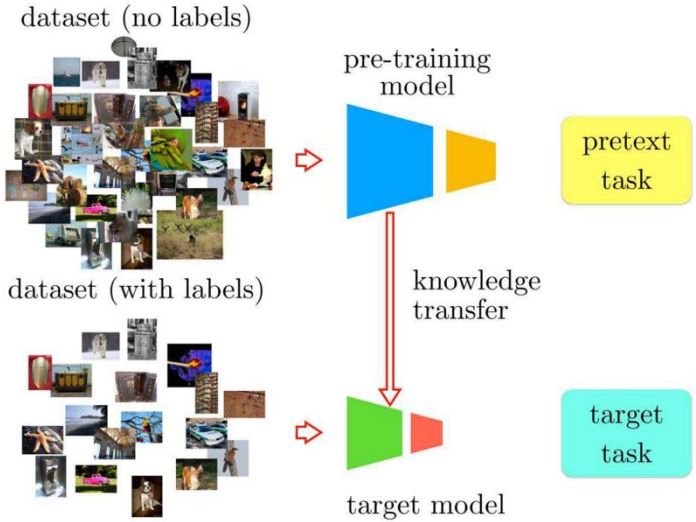



**Self-Supervised Learning:**
Create your own supervision

# Self-supervised Learning

In short: still supervised learning, with two important distinctions:

1. Learn from labels generated *autonomously* instead of human annotations.
2. The goal is to learn *good representations* for *other target tasks.*




Source: Noroozi *et al.*, 2018

# Self-supervised pretext tasks

Example: learn to predict image transformations / complete corrupted images

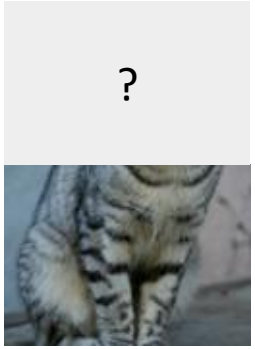
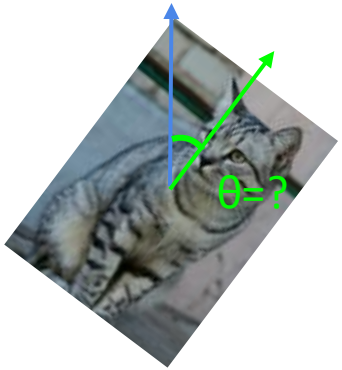



image completion




rotation prediction



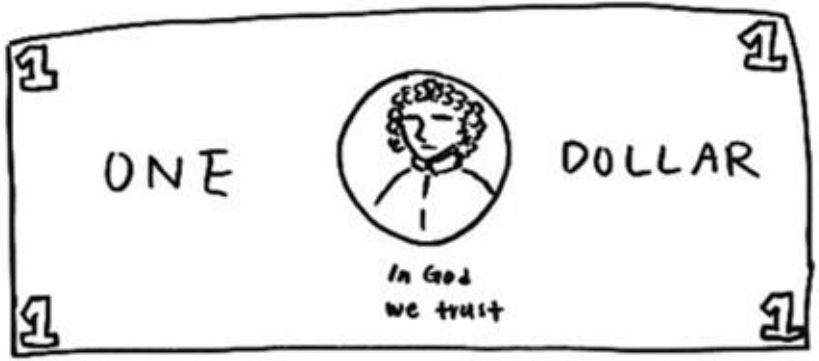
“jigsaw puzzle”



colorization

1. Solving the pretext tasks allow the model to learn good features.
2. We can automatically generate labels for the pretext tasks.

# Generative vs. Self-supervised Learning







Left: Drawing of a dollar bill from memory. Right: Drawing subsequently made with a dollar bill present. Image source: Epstein, 2016

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

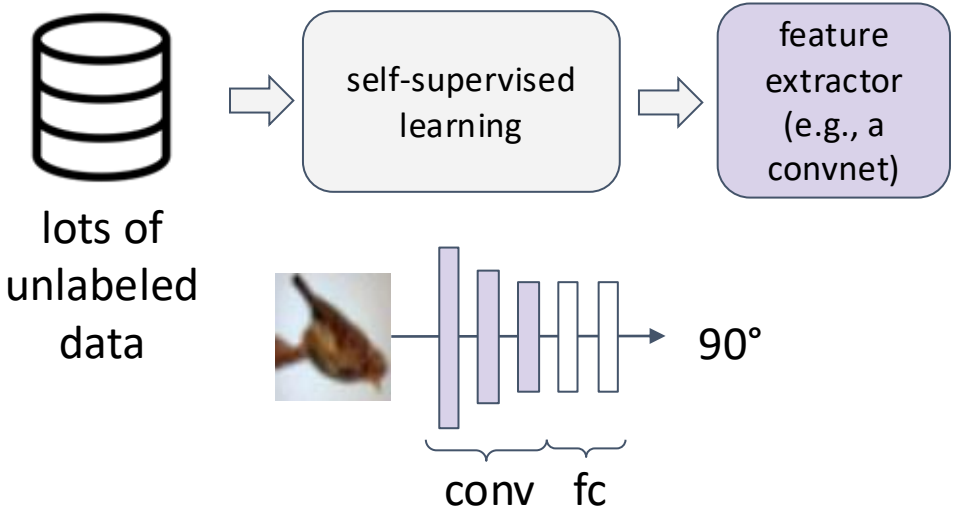
Source: Anand, 2020

# How to evaluate a self-supervised learning method?

We usually don’t care about the performance of the self-supervised learning task, e.g., we don’t care if the model learns to predict image rotation perfectly.

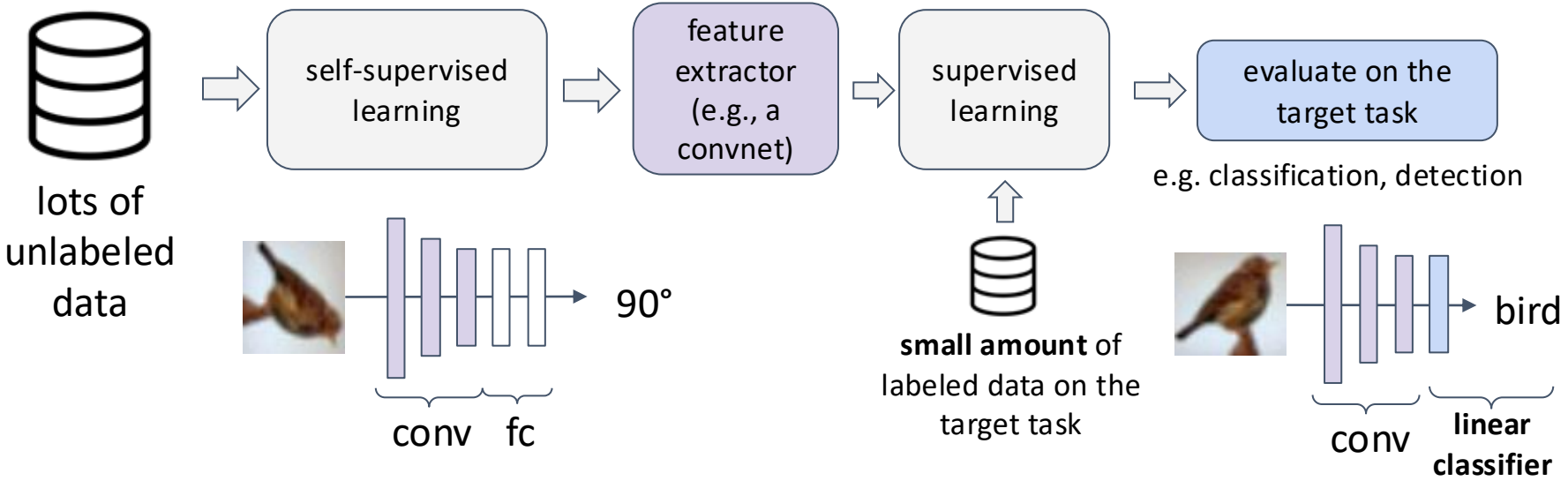
Evaluate the learned feature encoders on downstream *target tasks*

# How to evaluate a self-supervised learning method?




1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

# How to evaluate a self-supervised learning method?



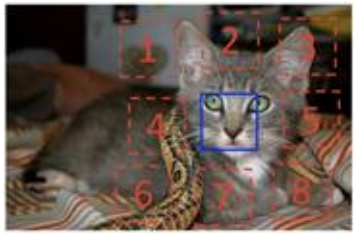

1. Learn good feature extractors from self-supervised pretext tasks, e.g., predicting image rotations

2. Attach a shallow network on the feature extractor; train the shallow network on the target task with small amount of labeled data

# Broader picture

**Today's lecture**

computer vision



Doersch et al., 2015

language modeling



**Language Models are Few-Shot Learners**

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry

Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan

Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter

Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray

Benjamin Chess Jack Clark Christopher Berner

Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

OpenAI

**Abstract**

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model, and test its performance in the few-shot setting. For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. At the same time, we also identify some datasets where GPT-3's few-shot learning still struggles, as well as some datasets where GPT-3 faces methodological issues related to training on large web corpora. Finally, we find that GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans. We discuss broader societal impacts of this finding and of GPT-3 in general.

GPT3 (Brown, Mann, Ryder, Subbiah et al., 2020)

speech synthesis




Wavenet (van den Oord et al., 2016)

robot / reinforcement learning



Dense Object Net (Florence and Manuelli et al., 2018)

...

# Today's Agenda

**Pretext tasks from image transformations**

- Rotation, inpainting, rearrangement, coloring

**Contrastive representation learning**

- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC

# Today's Agenda

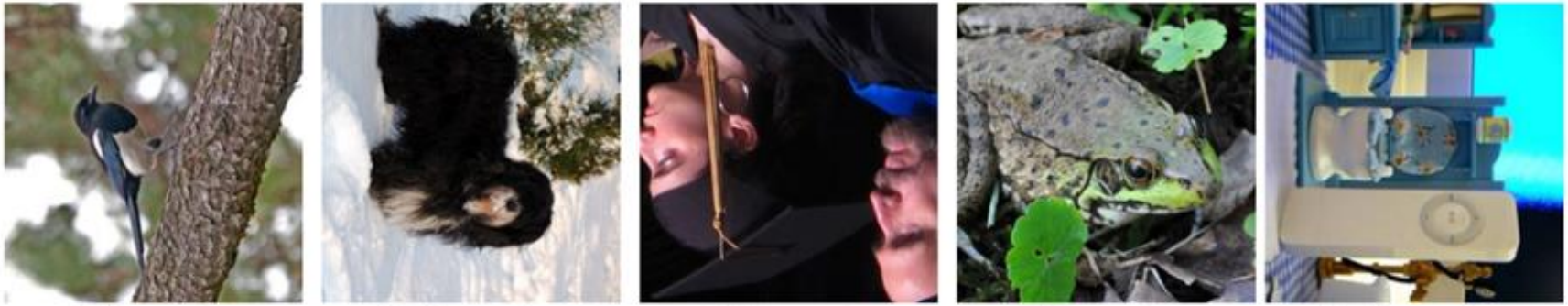
**Pretext tasks from image transformations**

- Rotation, inpainting, rearrangement, coloring

**Contrastive representation learning**

- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC

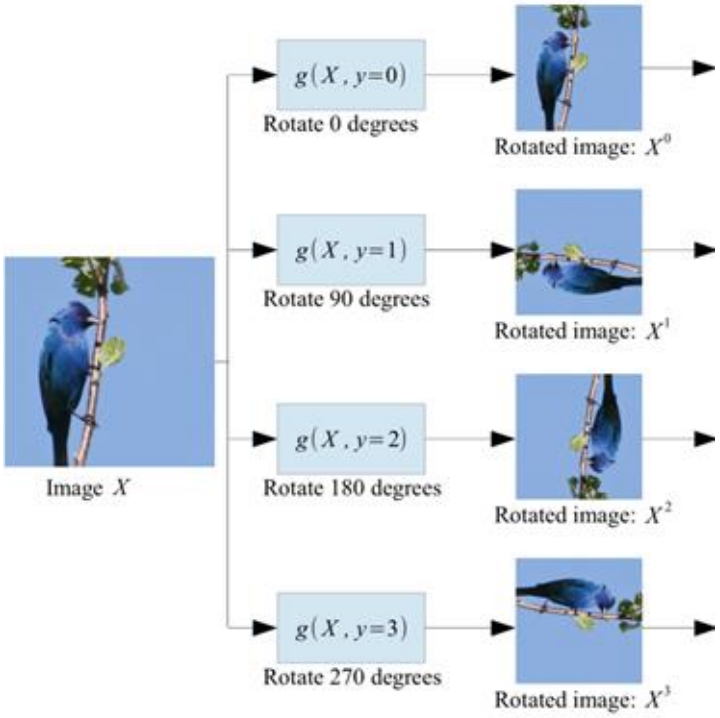
# Pretext task: predict rotations




**Hypothesis**: a model could recognize the correct rotation of an object only if it has the "visual commonsense" of what the object should look like unperturbed.

(Image source: Gidaris et al. 2018)

# Pretext task: predict rotations

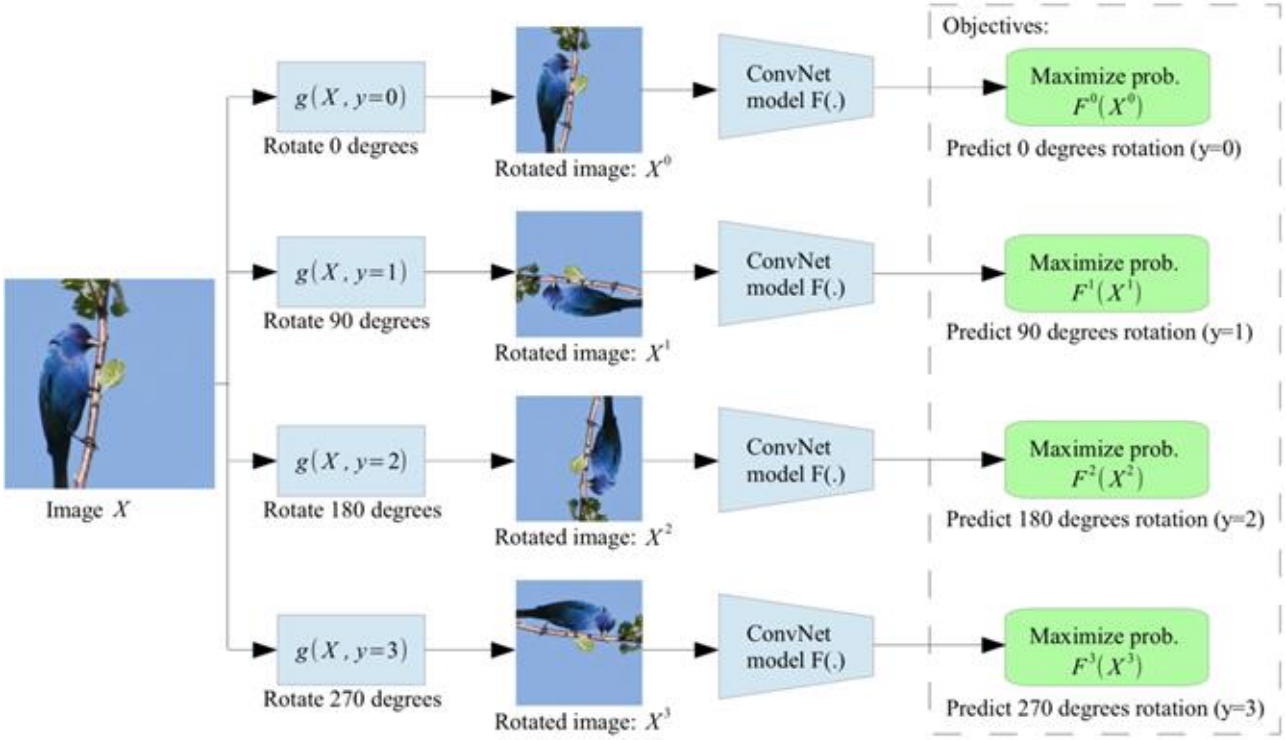



Self-supervised learning by rotating the entire input images.

The model learns to predict which rotation is applied (4-way classification)

(Image source: Gidaris et al. 2018)

# Pretext task: predict rotations

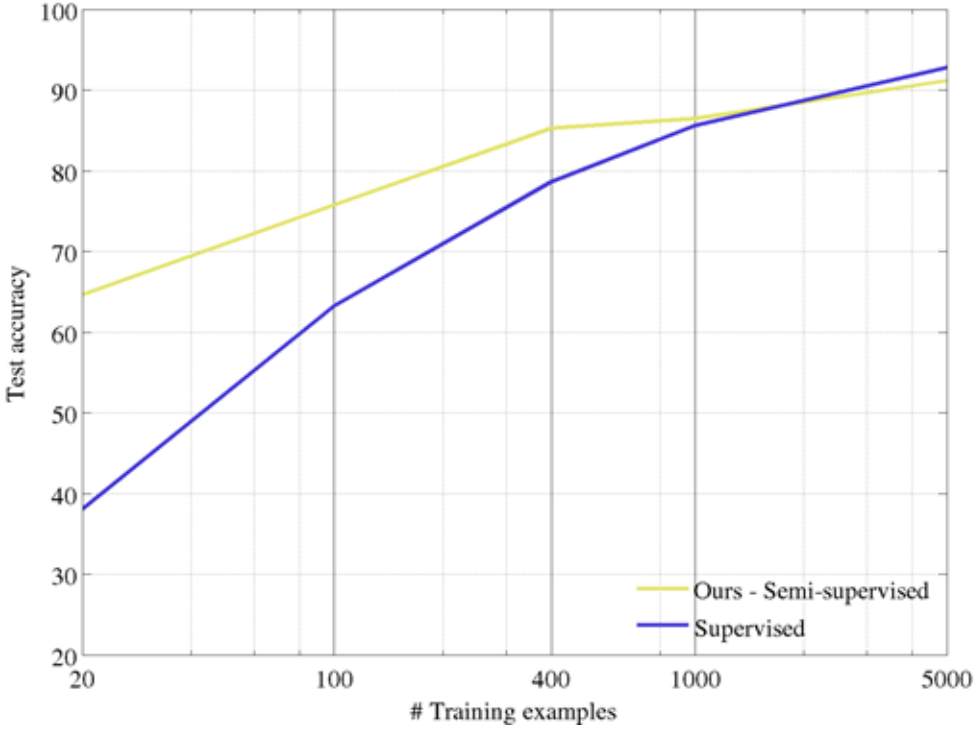



Self-supervised learning by rotating the entire input images.

The model learns to predict which rotation is applied (4-way classification)

(Image source: Gidaris et al. 2018)

# Evaluation on semi-supervised learning




Self-supervised learning on **CIFAR10** (entire training set).

Freeze conv1 + conv2
Learn **conv3 + linear** layers with subset of labeled CIFAR10 data (classification).

(Image source: Gidaris et al. 2018)

# Transfer learned features to supervised learning

| | Classification (%mAP) | | Detection (%mAP) | Segmentation (%mIoU) |
|---|---|---|---|---|
| Trained layers | fc6-8 | all | all | all |
| ImageNet labels | 78.9 | 79.9 | 56.8 | 48.0 |
| Random | | 53.3 | 43.4 | 19.8 |
| Random rescaled Krähenbühl et al. (2015) | 39.2 | 56.6 | 45.6 | 32.6 |
| Egomotion (Agrawal et al., 2015) | 31.0 | 54.2 | 43.9 | |
| Context Encoders (Pathak et al., 2016b) | 34.6 | 56.5 | 44.5 | 29.7 |
| Tracking (Wang & Gupta, 2015) | 55.6 | 63.1 | 47.4 | |
| Context (Doersch et al., 2015) | 55.1 | 65.3 | 51.1 | |
| Colorization (Zhang et al., 2016a) | 61.5 | 65.6 | 46.9 | 35.6 |
| BIGAN (Donahue et al., 2016) | 52.3 | 60.1 | 46.9 | 34.9 |
| Jigsaw Puzzles (Noroozi & Favaro, 2016) | - | 67.6 | 53.2 | 37.6 |
| NAT (Bojanowski & Joulin, 2017) | 56.7 | 65.3 | 49.4 | |
| Split-Brain (Zhang et al., 2016b) | 63.0 | 67.1 | 46.7 | 36.0 |
| ColorProxy (Larsson et al., 2017) | | 65.9 | | 38.4 |
| Counting (Noroozi et al., 2017) | - | 67.7 | 51.4 | 36.6 |
| (Ours) RotNet | **70.87** | **72.97** | **54.4** | **39.1** |

Pretrained with full ImageNet supervision

No pretraining

Self-supervised learning on **ImageNet** (entire training set) with AlexNet.

Finetune on labeled data from **Pascal VOC 2007**.

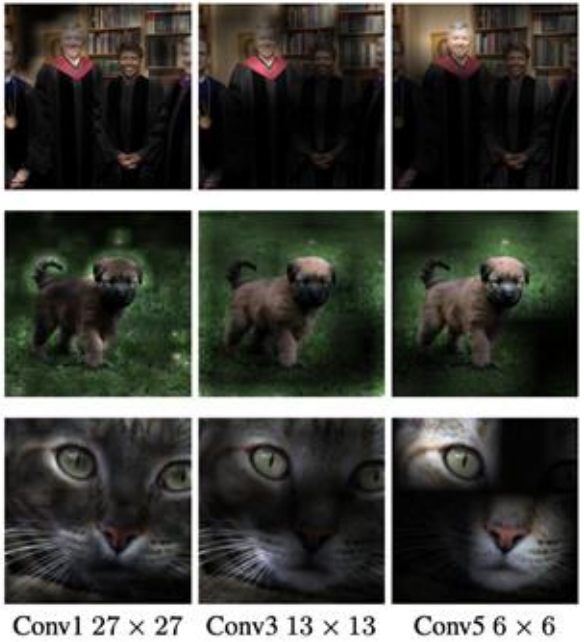
Self-supervised learning with rotation prediction

source: Gidaris et al. 2018

# Visualize learned visual attentions



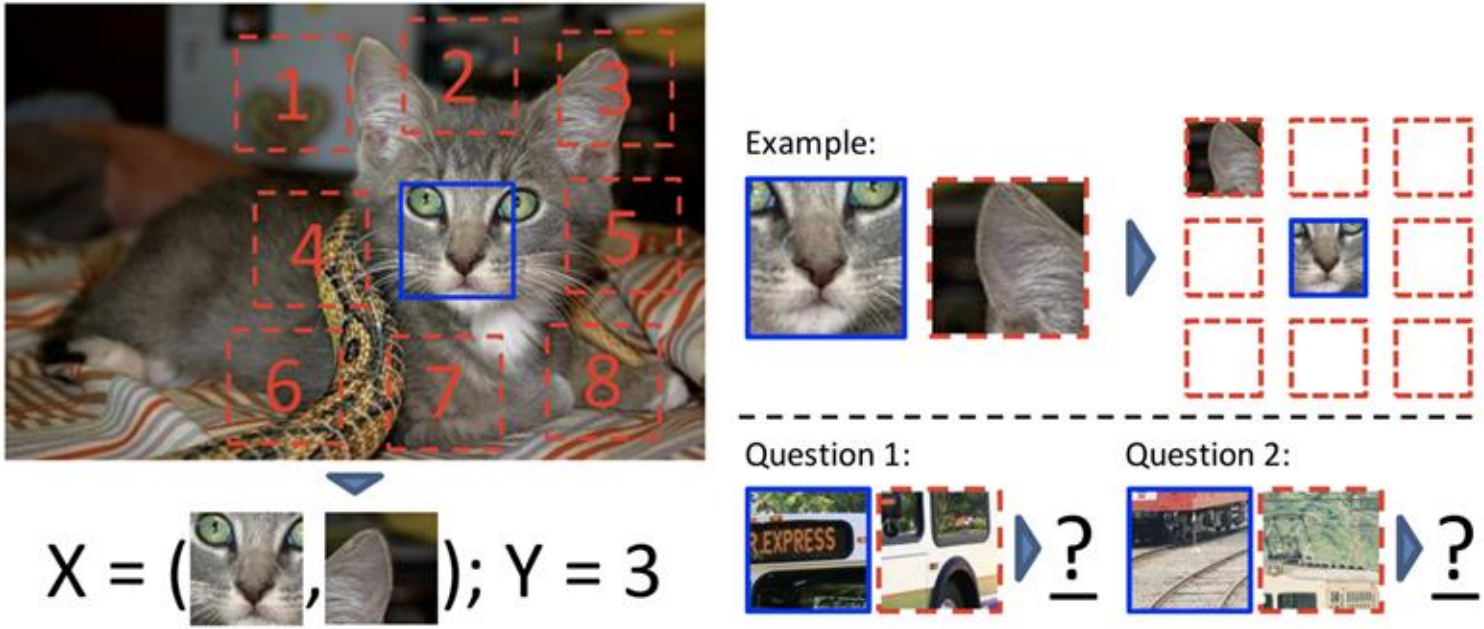

(a) **Attention maps of supervised model**




(b) **Attention maps of our self-supervised model**

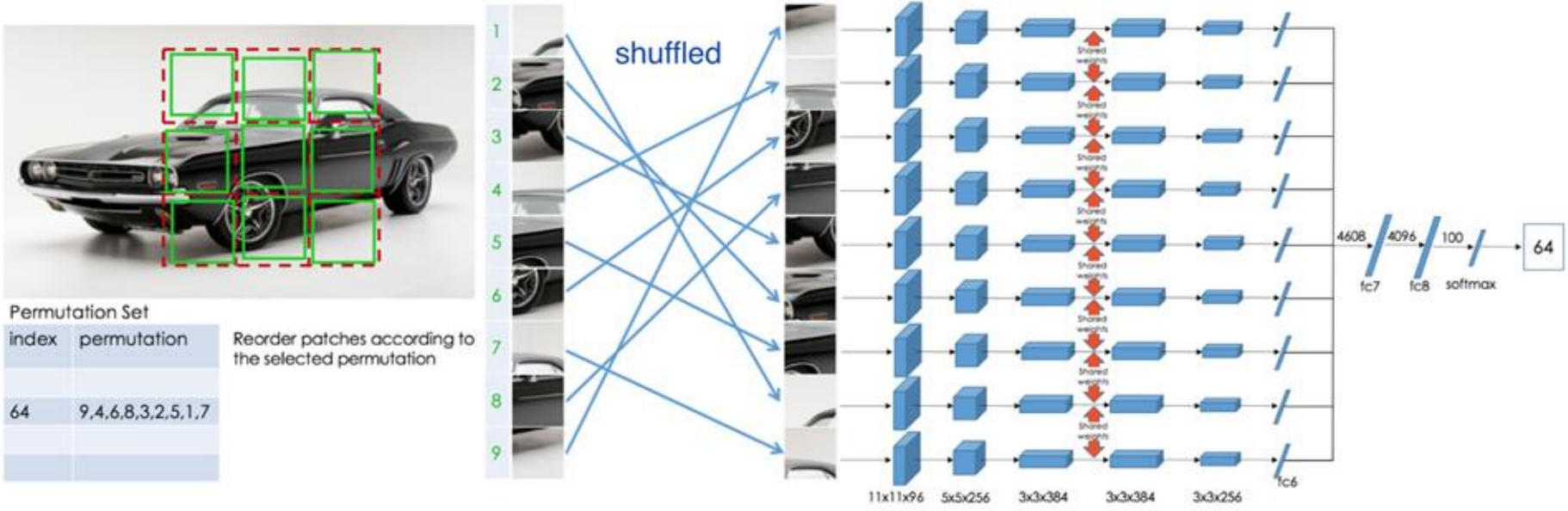
(Image source: Gidaris et al. 2018)

# Pretext task: predict relative patch locations




(Image source: [Doersch et al., 2015](Doersch et al., 2015))

# Pretext task: solving “jigsaw puzzles”




(Image source: [Noroozi & Favaro, 2016])

# Transfer learned features to supervised learning

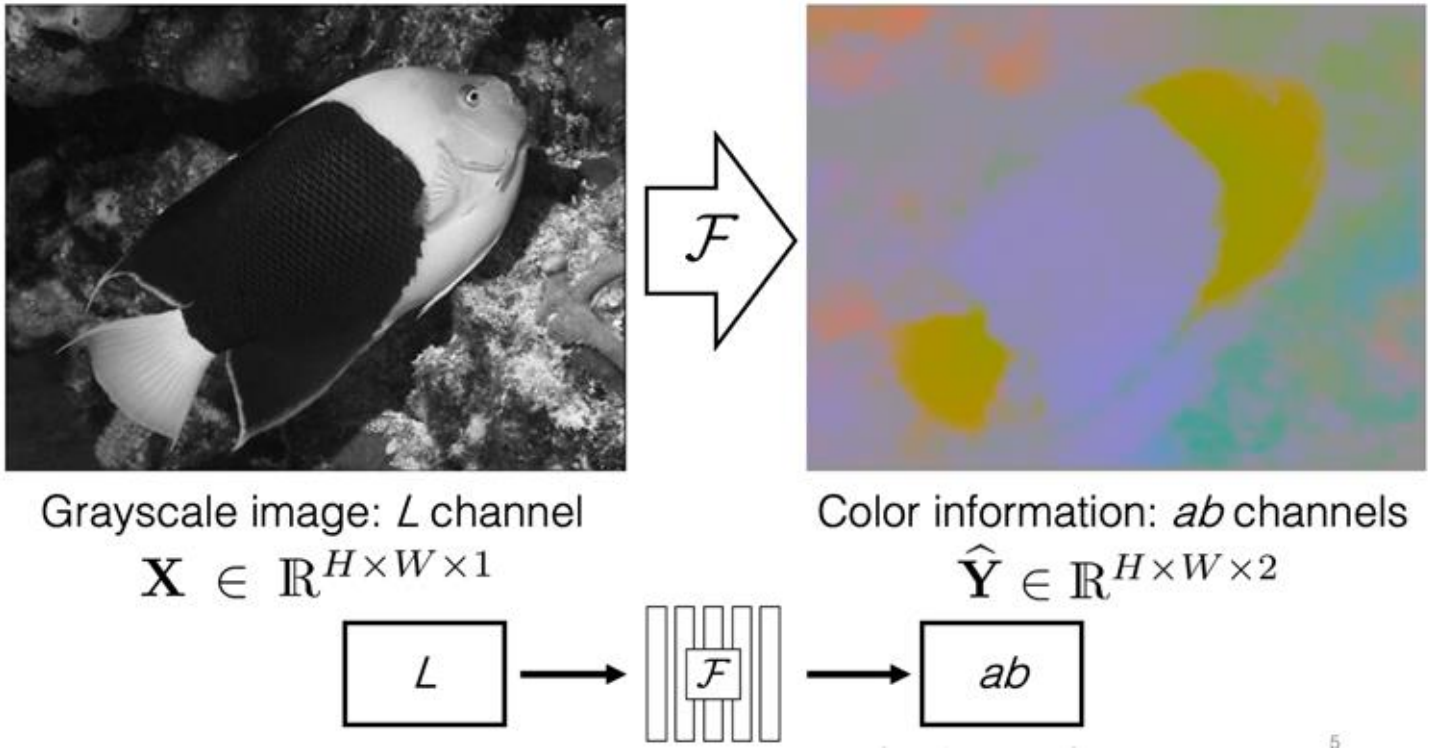
Table 1: Results on PASCAL VOC 2007 Detection and Classification. The results of the other methods are taken from Pathak *et al.* [30].

| Method | Pretraining time | Supervision | Classification | Detection | Segmentation |
|---|---|---|---|---|---|
| Krizhevsky*et al.* [25] | 3 days | 1000 class labels | **78.2%** | **56.8%** | **48.0%** |
| Wang and Gupta[39] | 1 week | motion | 58.4% | 44.0% | - |
| Doersch *et al.* [10] | 4 weeks | context | 55.3% | 46.6% | - |
| Pathak *et al.* [30] | 14 hours | context | 56.5% | 44.5% | 29.7% |
| Ours | 2.5 days | context | **67.6%** | **53.2%** | **37.6%** |

"Ours" is feature learned from solving image Jigsaw puzzles (Noroozi & Favaro, 2016). Doersch et al. is the method with relative patch location

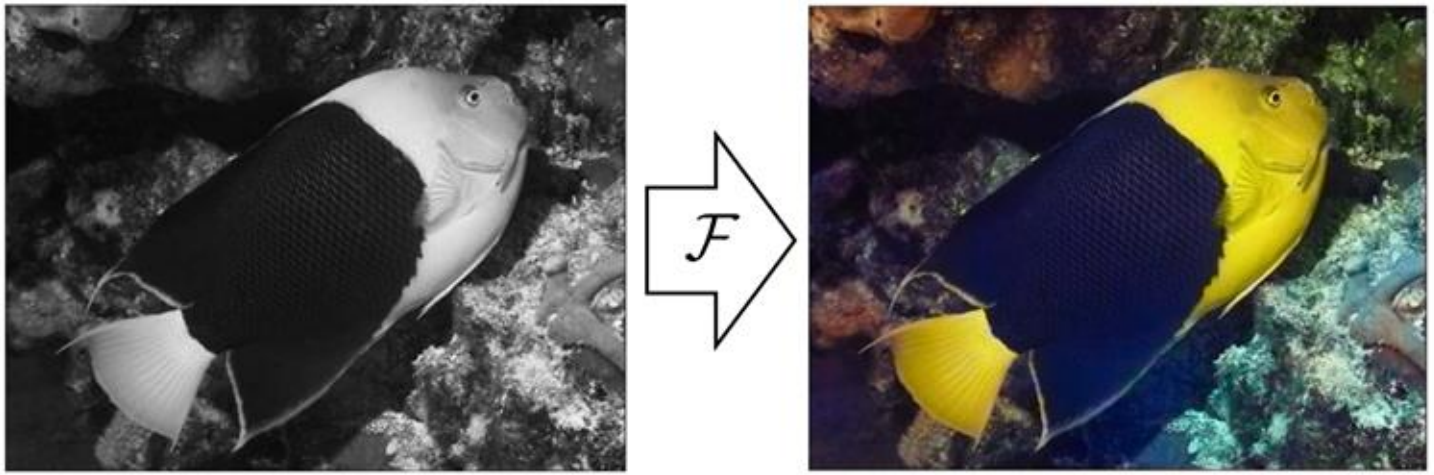
(source: Noroozi & Favaro, 2016)

# Pretext task: image coloring




Source: Richard Zhang / Phillip Isola

# Pretext task: image coloring

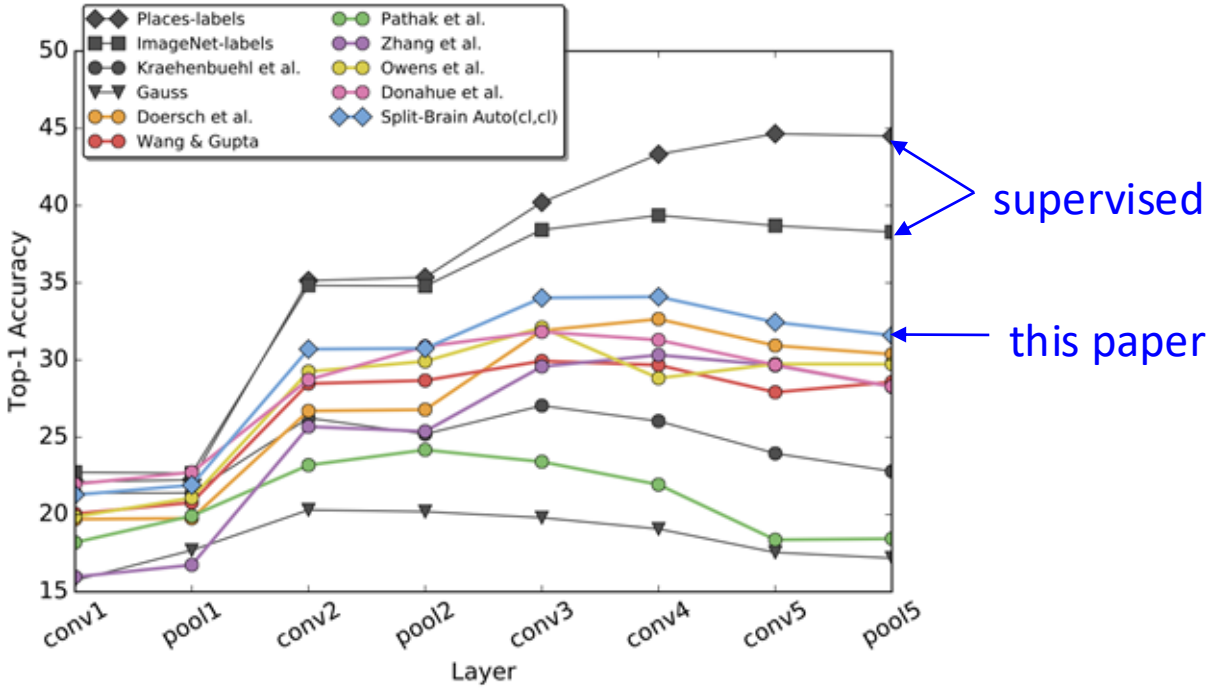






Source: Richard Zhang / Phillip Isola

# Transfer learned features to supervised learning




Self-supervised learning on **ImageNet** (entire training set).

Use *concatenated features* from $F_1$ and $F_2$

Labeled data is from the **Places** (Zhou 2016).

Source: Zhang et al., 2017

# Pretext task: video coloring

**Idea**: model the *temporal coherence* of colors in videos

reference frame

how should I color these frames?



t = 0



t = 1



t = 2



t = 3

...

Source: [Vondrick et al., 2018]

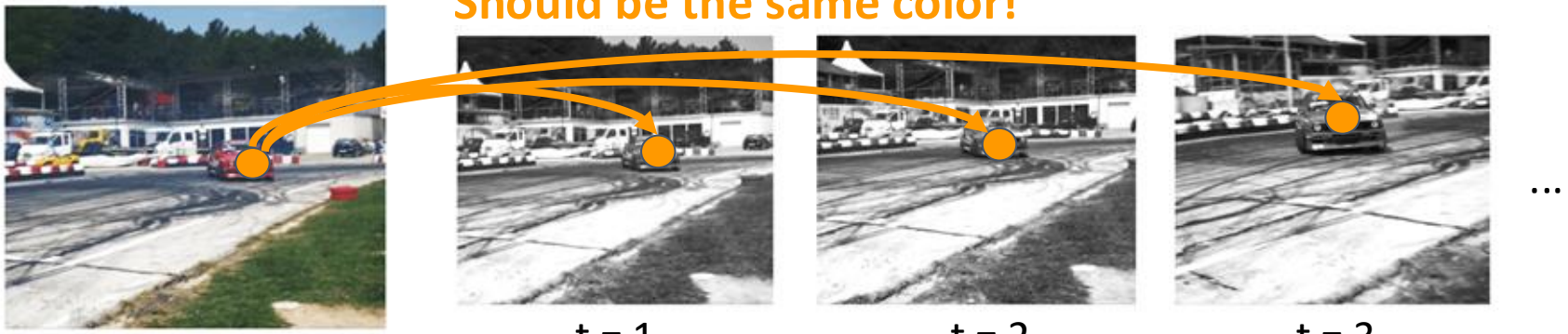
# Pretext task: video coloring

**Idea**: model the *temporal coherence* of colors in videos

reference frame

how should I color these frames?

**Should be the same color!**



t = 0

t = 1

t = 2

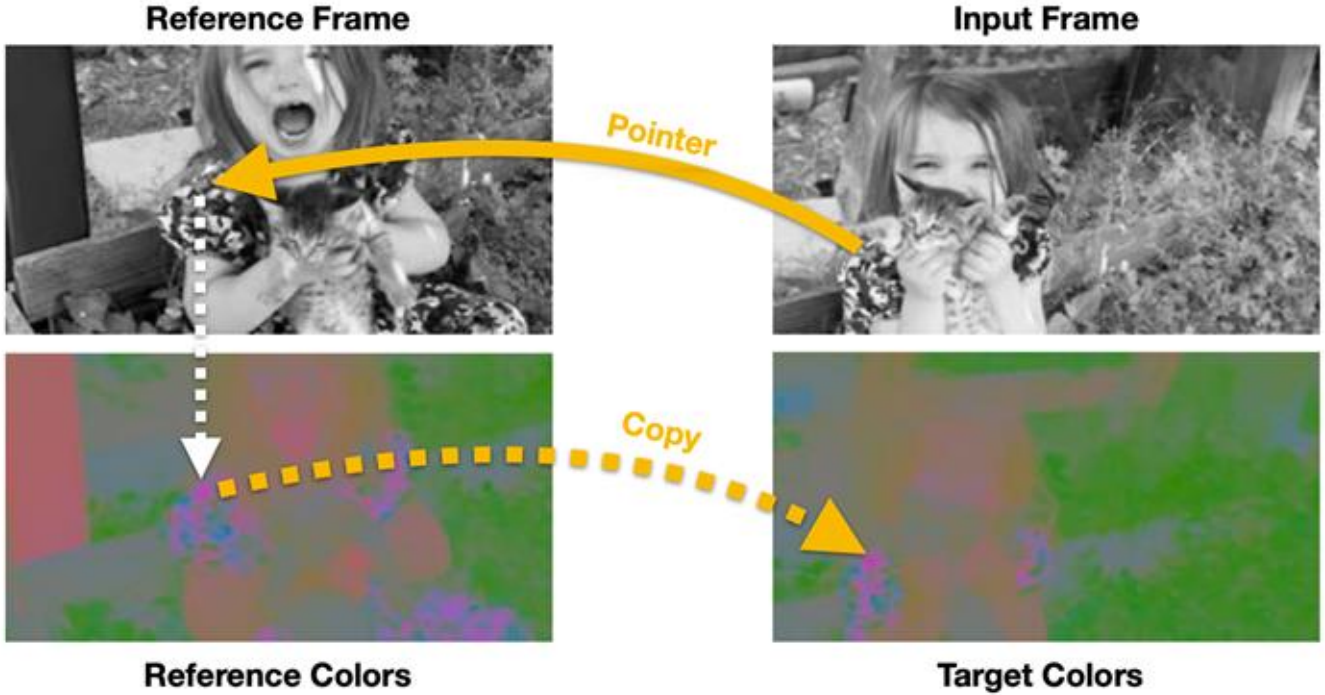
t = 3

...

**Hypothesis**: learning to color video frames should allow model to learn to track regions or objects without labels!

Source: Vondrick et al., 2018

# Learning to color videos



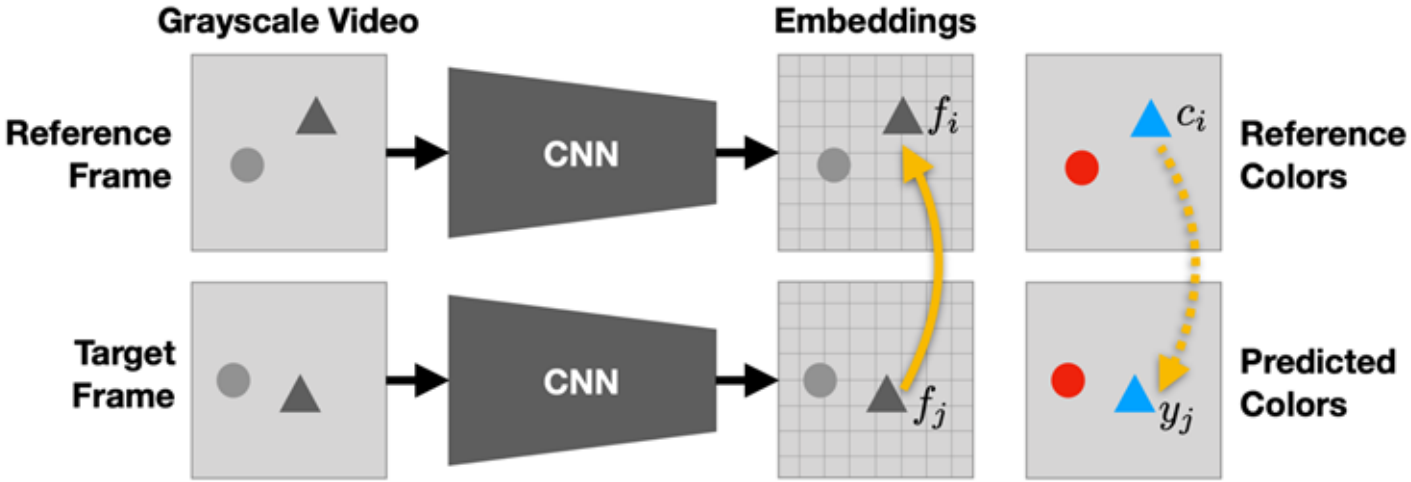

**Learning objective:**

Establish mappings between reference and target frames in a learned feature space.

Use the mapping as “pointers” to copy the correct color (LAB).

Source: Vondrick et al., 2018

# Learning to color videos

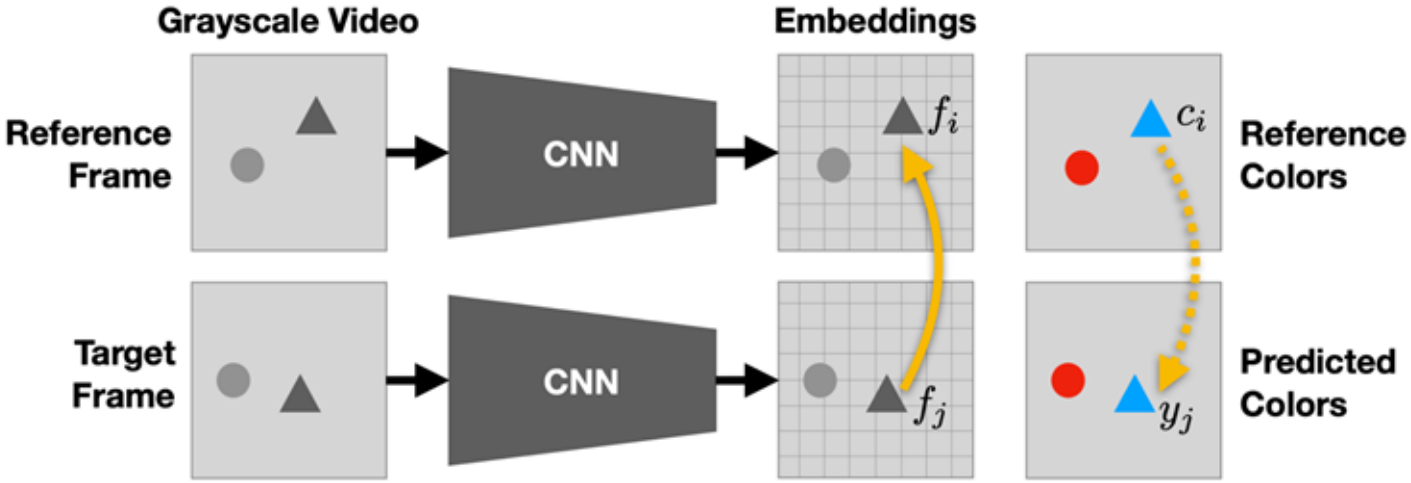



attention map on the reference frame

$$A_{ij} = \frac{\exp\left(f_i^T f_j\right)}{\sum_k \exp\left(f_k^T f_j\right)}$$

Source: [Vondrick et al., 2018]

# Learning to color videos




attention map on the reference frame

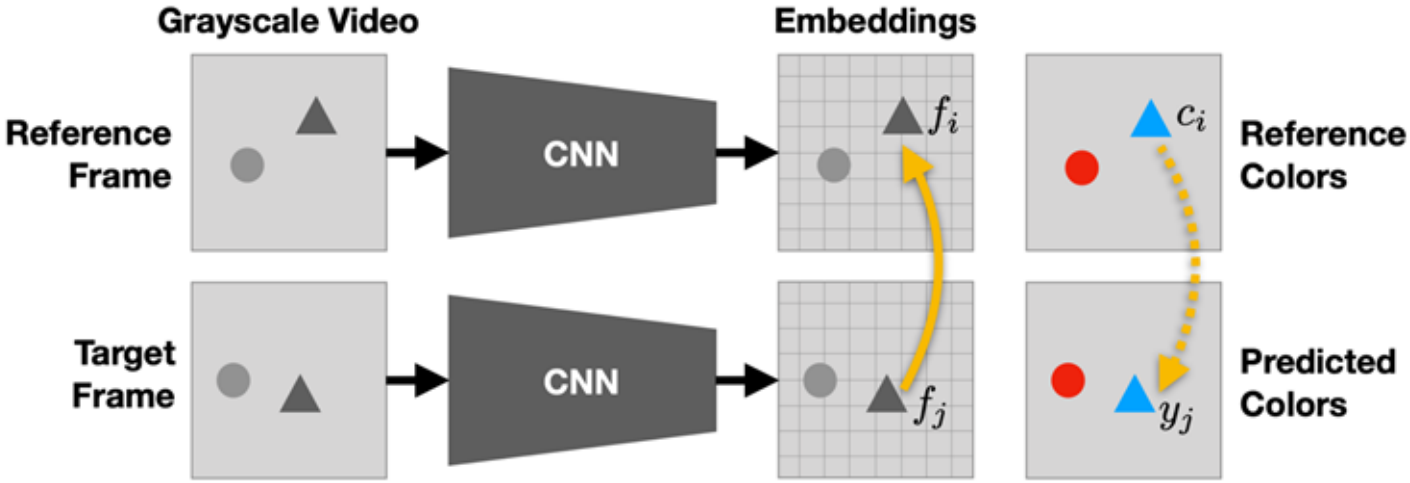
$$A_{ij} = \frac{\exp\left(f_i^T f_j\right)}{\sum_k \exp\left(f_k^T f_j\right)}$$

predicted color = weighted sum of the reference color

$$y_j = \sum_i A_{ij} c_i$$

Source: [Vondrick et al., 2018]

# Learning to color videos




attention map on the reference frame

$$A_{ij} = \frac{\exp\left(f_i^T f_j\right)}{\sum_k \exp\left(f_k^T f_j\right)}$$

predicted color = weighted sum of the reference color

$$y_j = \sum_i A_{ij} c_i$$

loss between predicted color and ground truth color

$$\min_\theta \sum_j \mathcal{L}\left(y_j, c_j\right)$$

Source: [Vondrick et al., 2018]

# Colorizing videos (qualitative)

reference frame



target frames (gray)



predicted color



Source: [Google AI blog post]

# Colorizing videos (qualitative)

reference frame



target frames (gray)



predicted color



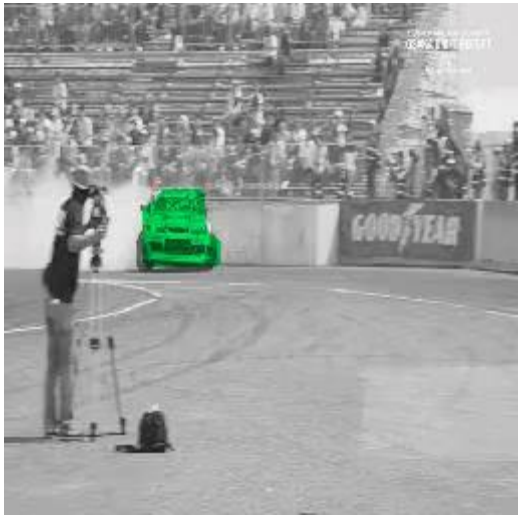
Source: [Google AI blog post]

# Tracking emerges from colorization

Propagate segmentation masks using learned attention







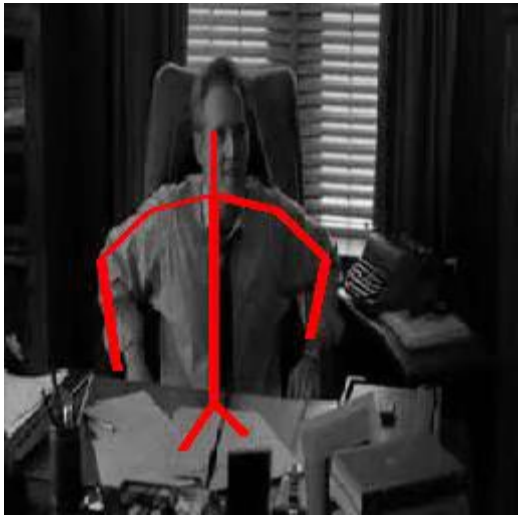

Source: [Google AI blog post]

# Tracking emerges from colorization

Propagate pose keypoints using learned attention







Source: [Google AI blog post]

# Summary: pretext tasks from image transformations

- Pretext tasks focus on “visual common sense”, e.g., predict rotations, inpainting, rearrangement, and colorization.
- The models are forced learn good features about natural images, e.g., semantic representation of an object category, in order to solve the pretext tasks.
- We don’t care about the performance of these pretext tasks, but rather how useful the learned features are for downstream tasks (classification, detection, segmentation).
- Problems: 1) coming up with individual pretext tasks is tedious, and 2) the learned representations may not be general.

# Pretext tasks from image transformations

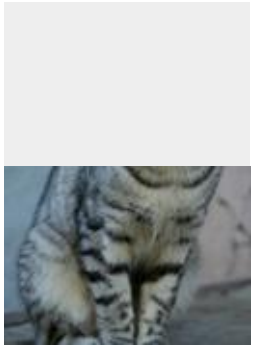


image completion



rotation prediction



“jigsaw puzzle”

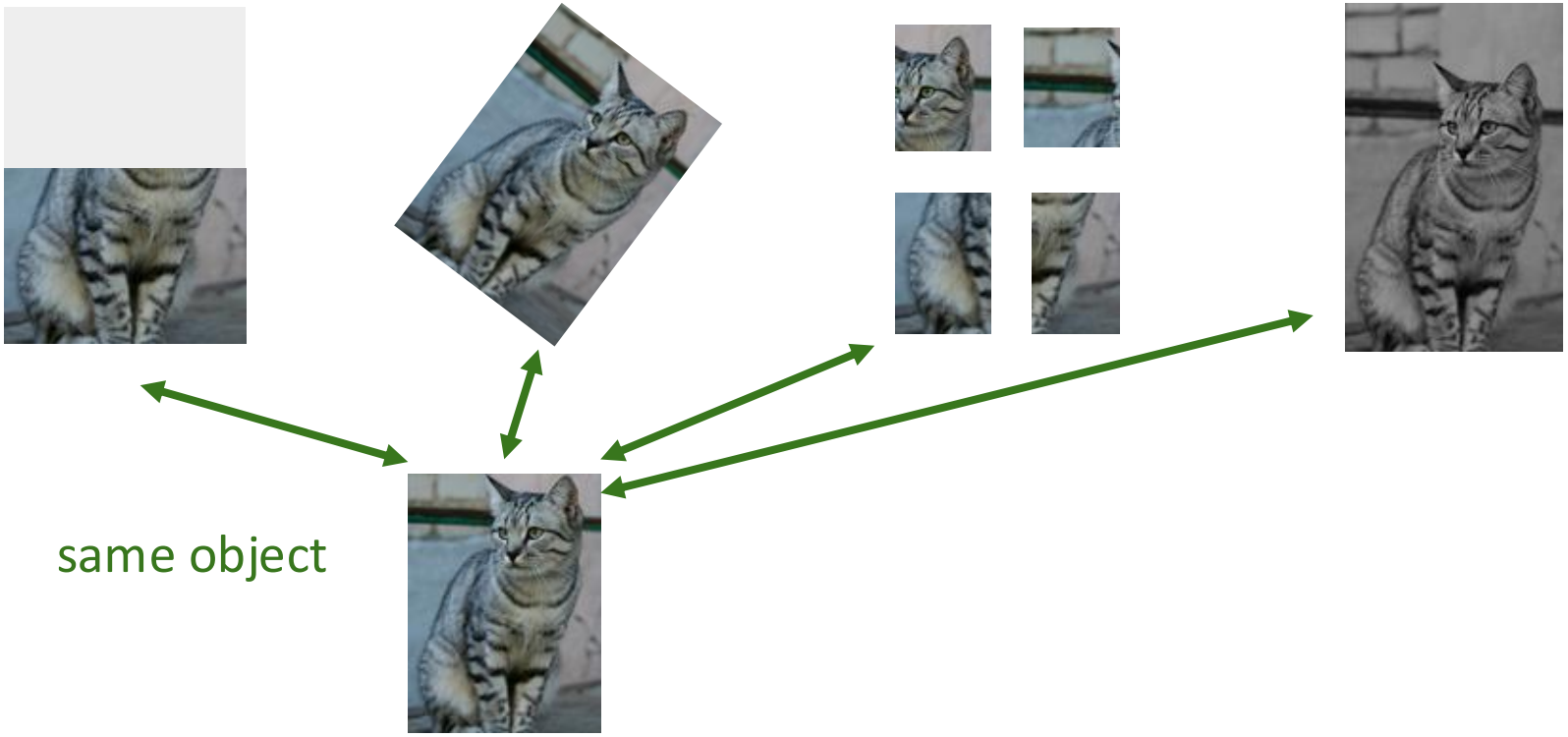


colorization

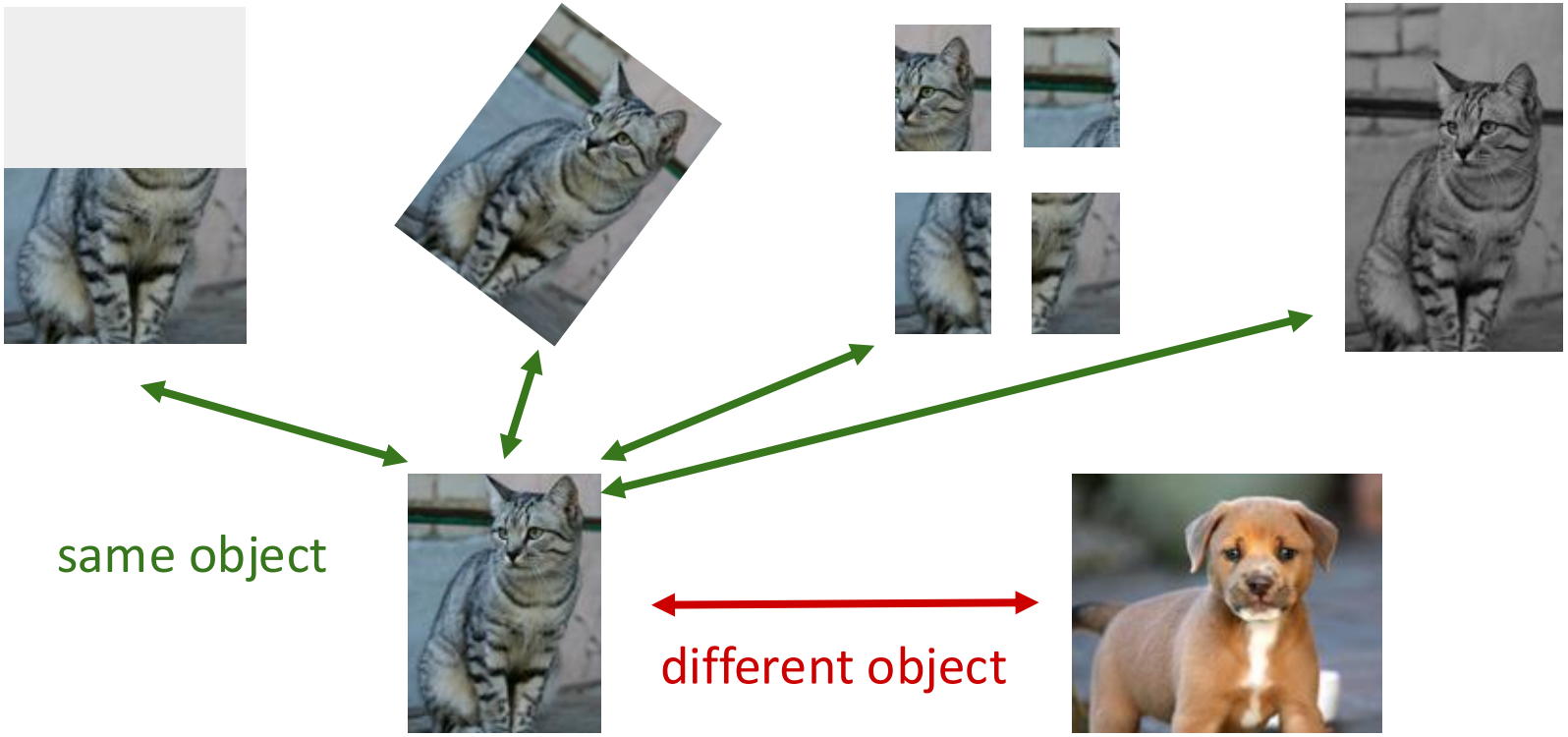
Learned representations may be tied to a specific pretext task!

Can we come up with a more general pretext task?

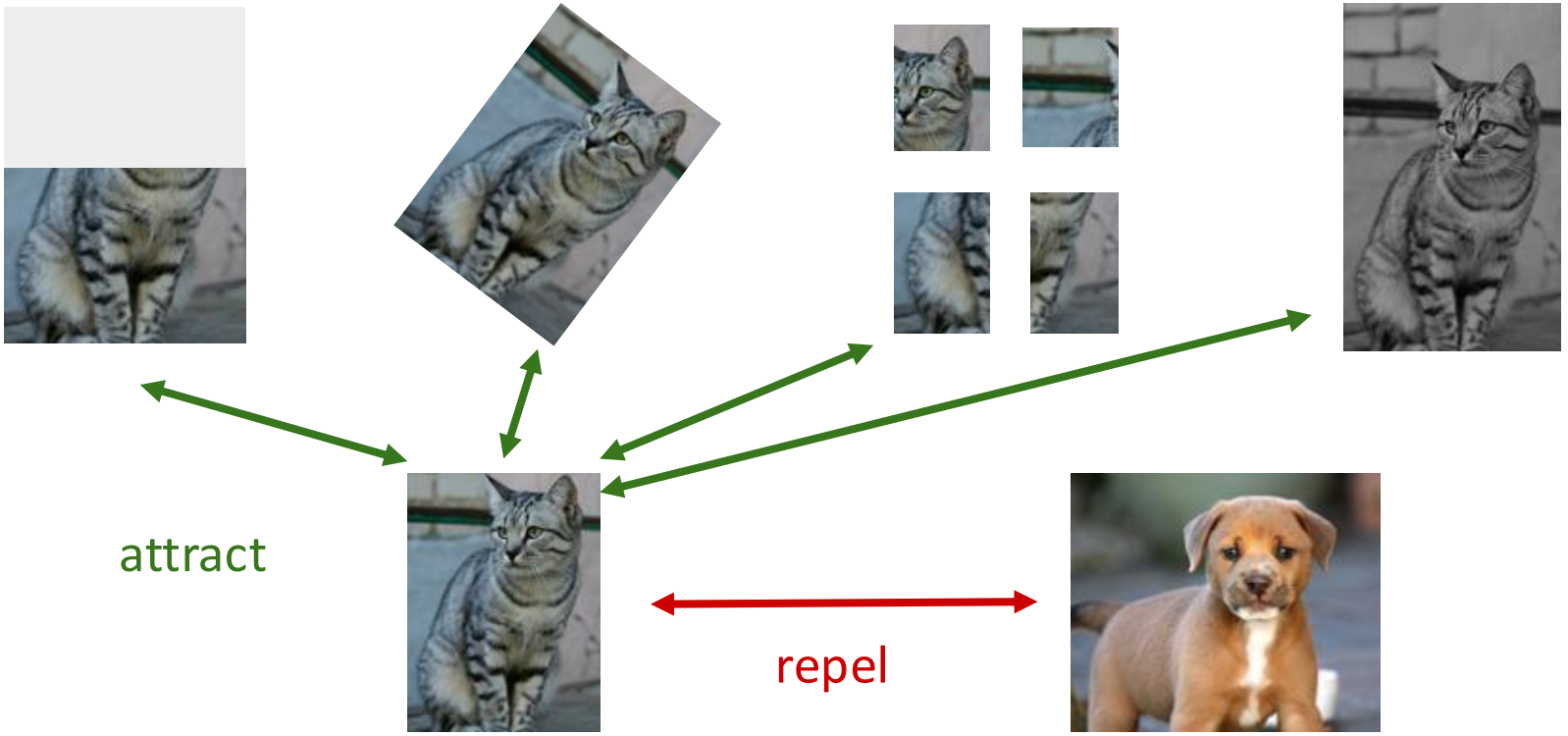
# A more general pretext task?



# A more general pretext task?



# Contrastive Representation Learning



# Today's Agenda

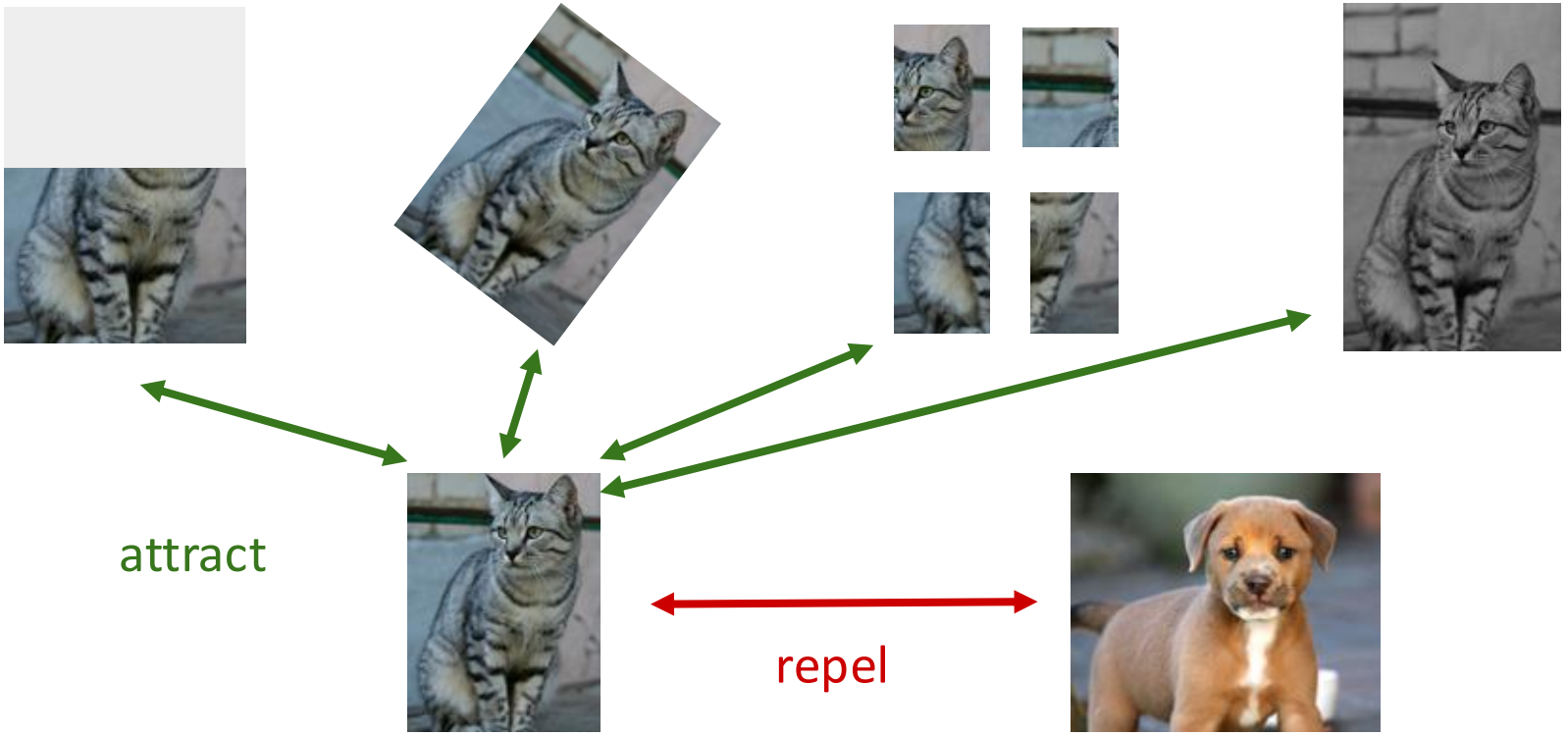
**Pretext tasks from image transformations**

- Rotation, inpainting, rearrangement, coloring

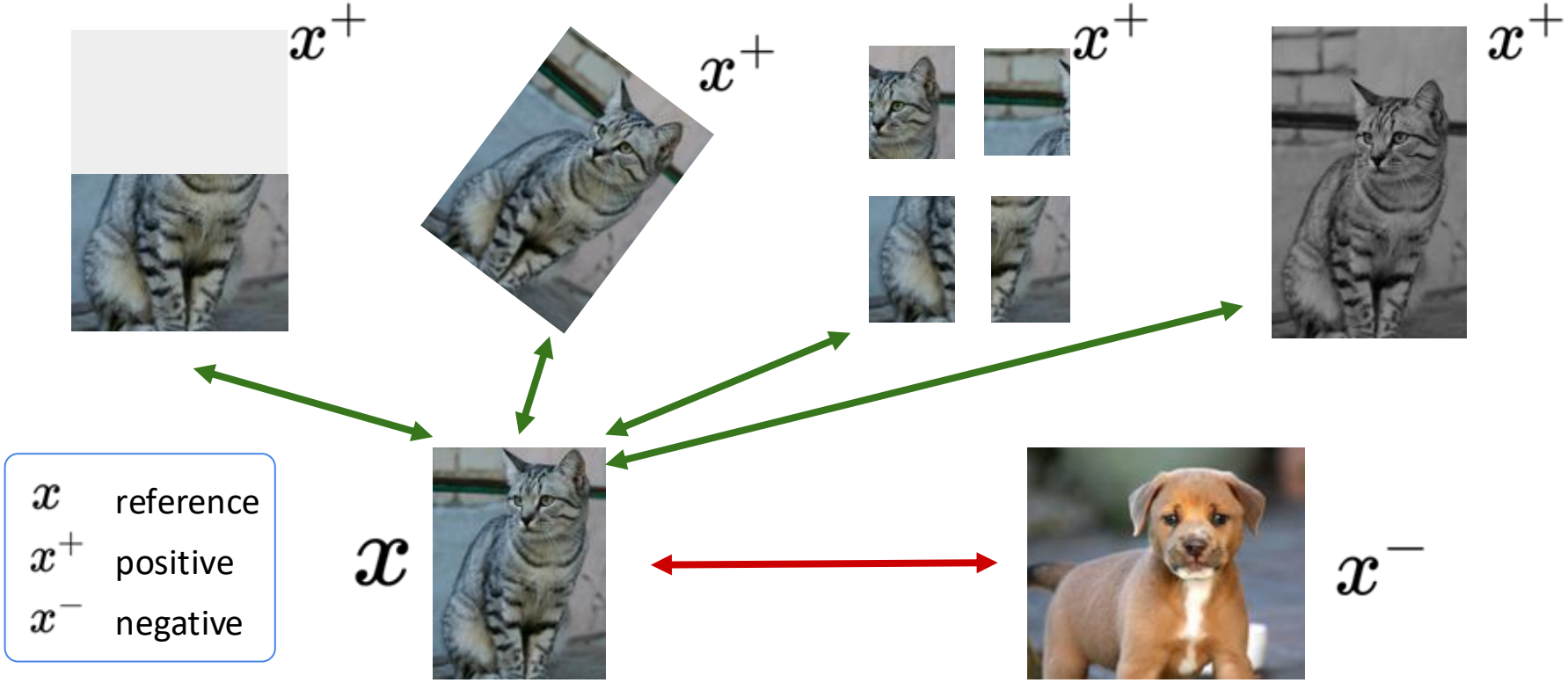
**Contrastive representation learning**

- Intuition and formulation
- Instance contrastive learning: SimCLR and MOCO
- Sequence contrastive learning: CPC

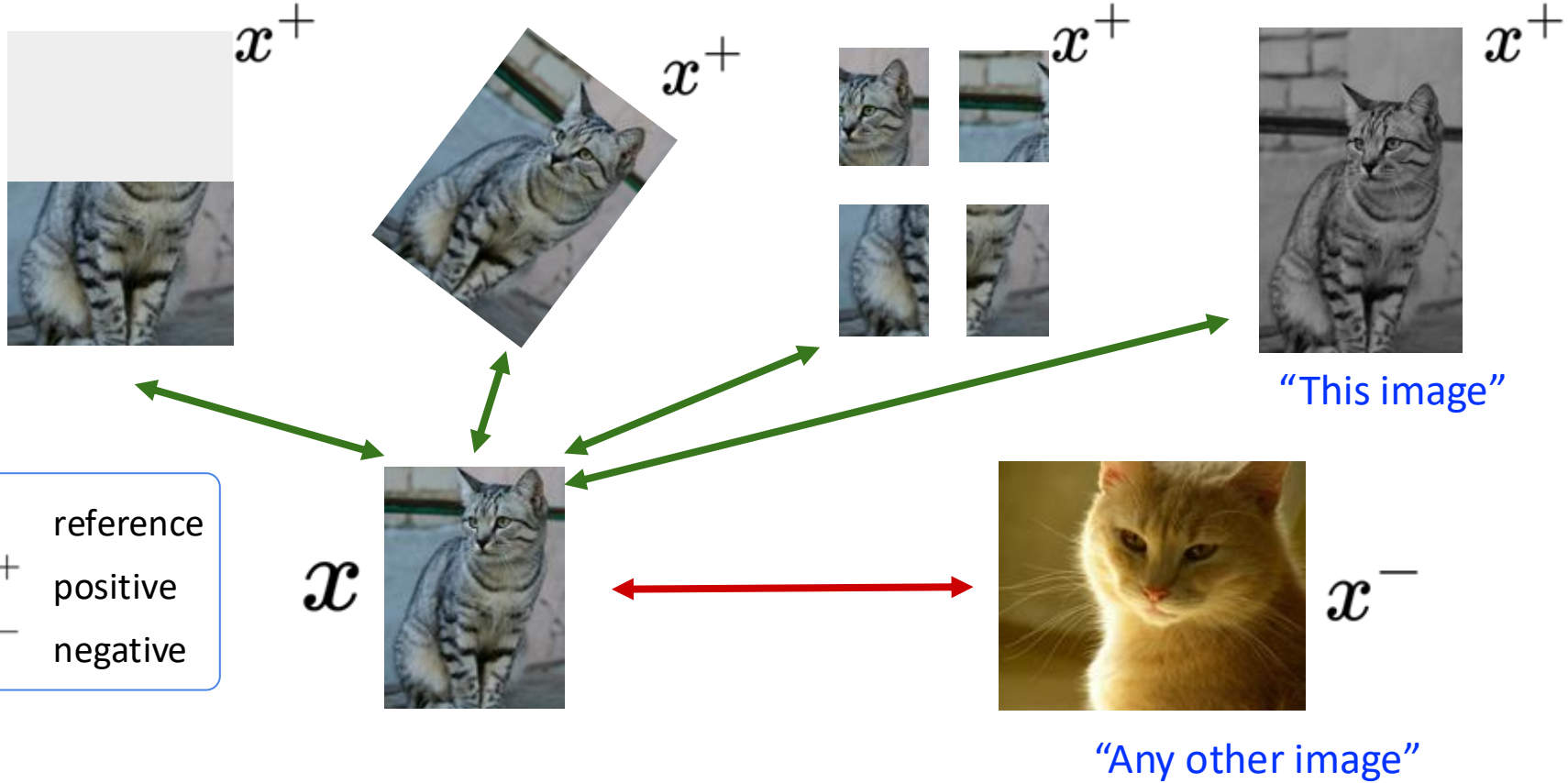
# Contrastive Representation Learning



# Contrastive Representation Learning



# Contrastive Representation Learning



# A formulation of contrastive learning

What we want:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

*x*: reference sample; $x^+$ positive sample; $x^-$ negative sample

Given a chosen score function, we aim to learn an **encoder function** *f* that yields high score for positive pairs ($x$, $x^+$) and low scores for negative pairs ($x$, $x^-$).

# A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

# A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$



$x$



$x^+$



$x$



$x_1^-$



$x_2^-$



$x_3^-$

...

# A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

score for the positive pair

score for the N-1 negative pairs

This seems familiar …

# A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

score for the positive pair

score for the N-1 negative pairs

This seems familiar …

Cross entropy loss for a N-way softmax classifier!

I.e., learn to find the positive sample from the N samples

# A formulation of contrastive learning

Loss function given 1 positive sample and N - 1 negative samples:

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

Commonly known as the InfoNCE loss (van den Oord et al., 2018)

A *lower bound* on the mutual information between *f(x)* and $f(x^+)$

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

The larger the negative sample size (*N*), the tighter the bound

Detailed derivation: Poole et al., 2019

# SimCLR: A Simple Framework for Contrastive Learning

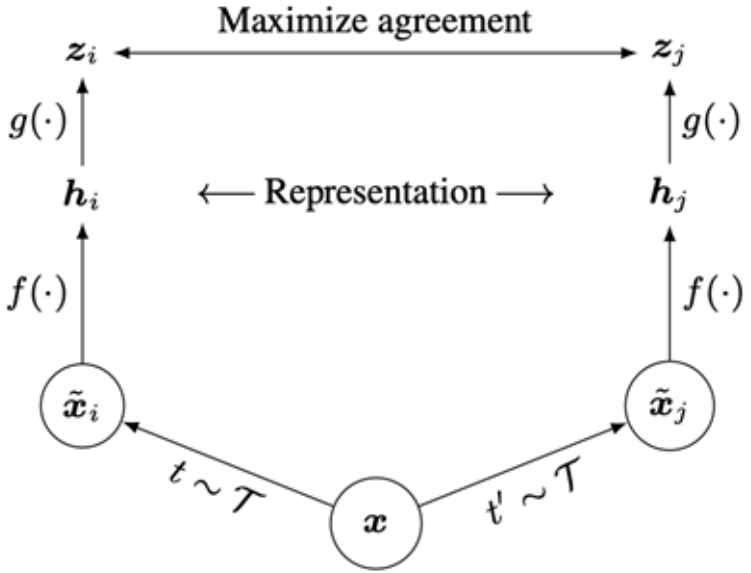
Cosine similarity as the score function:

$$s(u, v) = \frac{u^T v}{||u||||v||}$$

Use a projection network ***h(·)*** to project features to a space where contrastive learning is applied

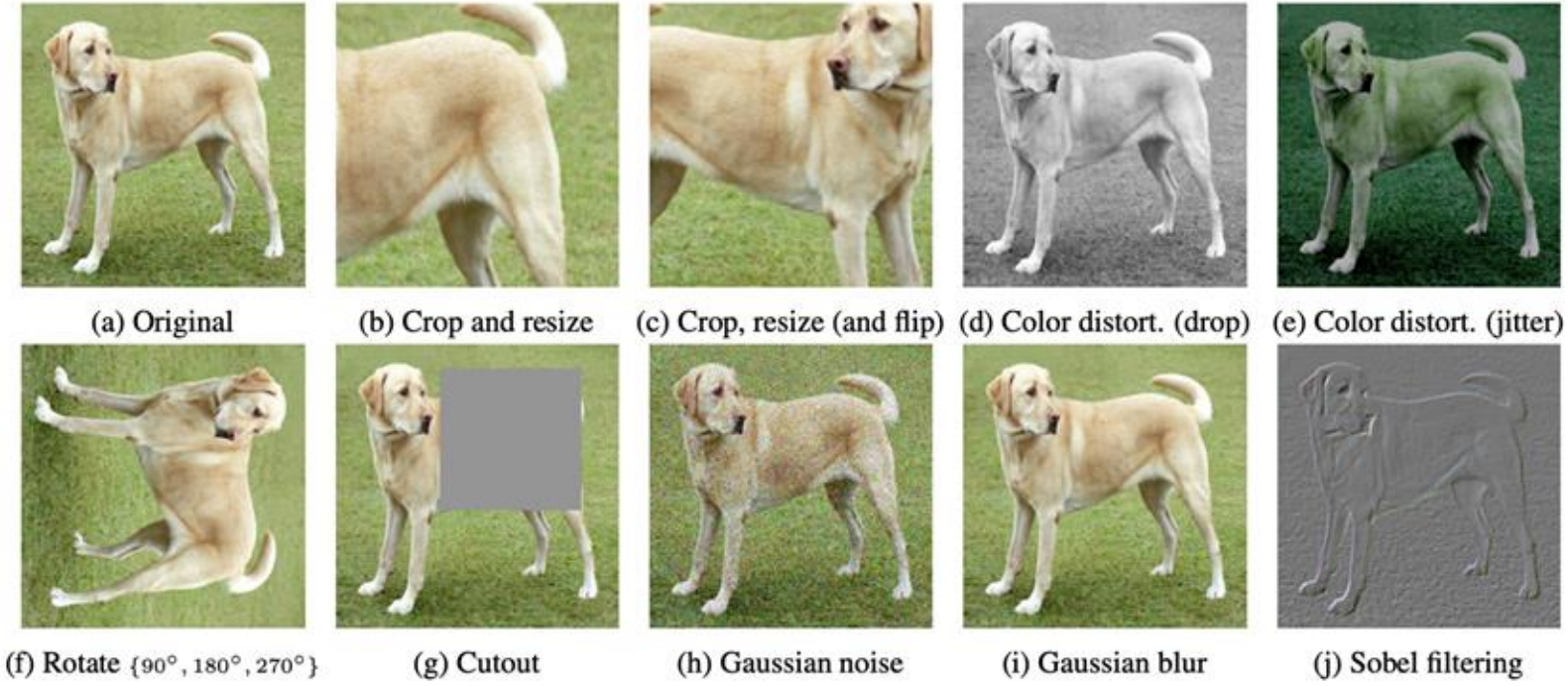
Generate positive samples through data augmentation:

- random cropping, random color distortion, and random blur.




Source: [Chen et al., 2020]

# SimCLR: generating positive samples from data augmentation




Source: [Chen et al., 2020]

# SimCLR

Generate a positive pair by sampling data augmentation functions

**Algorithm 1** SimCLR's main learning algorithm.

**input:** batch size $N$, constant $\tau$, structure of $f, g, \mathcal{T}$.
**for** sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  **for all** $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  **end for**
  **for all** $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  **end for**
  **define** $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks $f$ and $g$ to minimize $\mathcal{L}$
**end for**
**return** encoder network $f(\cdot)$, and throw away $g(\cdot)$

Source: Chen et al., 2020

# SimCLR

**Algorithm 1** SimCLR's main learning algorithm.

**input:** batch size $N$, constant $\tau$, structure of $f, g, \mathcal{T}$.
**for** sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  **for all** $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  **end for**
  **for all** $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  **end for**
  **define** $\ell(i, j)$ **as** $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks $f$ and $g$ to minimize $\mathcal{L}$
**end for**
**return** encoder network $f(\cdot)$, and throw away $g(\cdot)$

Generate a positive pair by sampling data augmentation functions

InfoNCE loss:
Use all non-positive samples in the batch as $x^-$

Source: Chen et al., 2020

# SimCLR

**Algorithm 1** SimCLR's main learning algorithm.

**input:** batch size $N$, constant $\tau$, structure of $f, g, \mathcal{T}$.
**for** sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  **for all** $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  **end for**
  **for all** $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  **end for**
  **define** $\ell(i, j)$ **as** $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks $f$ and $g$ to minimize $\mathcal{L}$
**end for**
**return** encoder network $f(\cdot)$, and throw away $g(\cdot)$

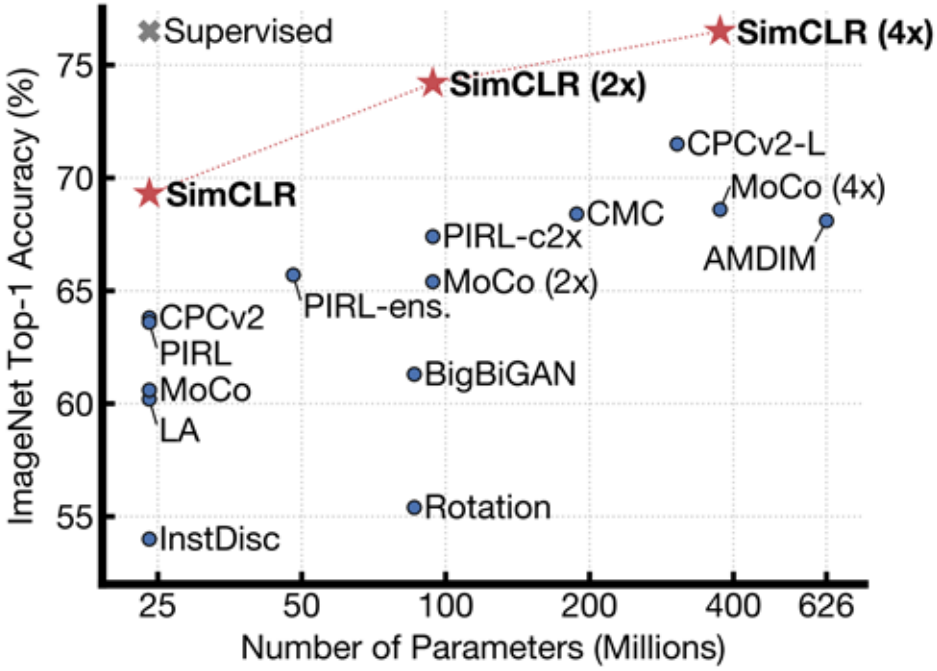
Generate a positive pair by sampling data augmentation functions

Iterate through and use each of the 2N sample as reference, compute average loss

InfoNCE loss:
Use all non-positive samples in the batch as $x^-$

Source: Chen et al., 2020

# Training linear classifier on SimCLR features




Train feature encoder on **ImageNet** (entire training set) using SimCLR.

Freeze feature encoder, train a linear classifier on top with labeled data.

Source: [Chen et al., 2020]

# Semi-supervised learning on SimCLR features

| Method | Architecture | Label fraction (Top 5) 1% | 10% |
|---|---|---|---|
| Supervised baseline | ResNet-50 | 48.4 | 80.4 |
| *Methods using other label-propagation:* | | | |
| Pseudo-label | ResNet-50 | 51.6 | 82.4 |
| VAT+Entropy Min. | ResNet-50 | 47.0 | 83.4 |
| UDA (w. RandAug) | ResNet-50 | - | 88.5 |
| FixMatch (w. RandAug) | ResNet-50 | - | 89.1 |
| S4L (Rot+VAT+En. M.) | ResNet-50 (4×) | - | 91.2 |
| *Methods using representation learning only:* | | | |
| InstDisc | ResNet-50 | 39.2 | 77.4 |
| BigBiGAN | RevNet-50 (4×) | 55.2 | 78.8 |
| PIRL | ResNet-50 | 57.2 | 83.8 |
| CPC v2 | ResNet-161(*) | 77.9 | 91.2 |
| SimCLR (ours) | ResNet-50 | 75.5 | 87.8 |
| SimCLR (ours) | ResNet-50 (2×) | 83.0 | 91.2 |
| SimCLR (ours) | ResNet-50 (4×) | **85.8** | **92.6** |

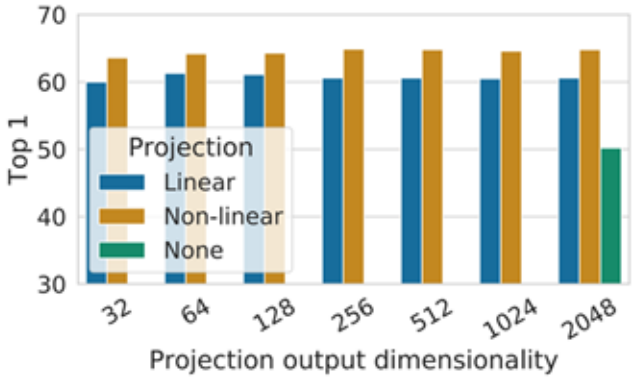
*Table 7.* ImageNet accuracy of models trained with few labels.

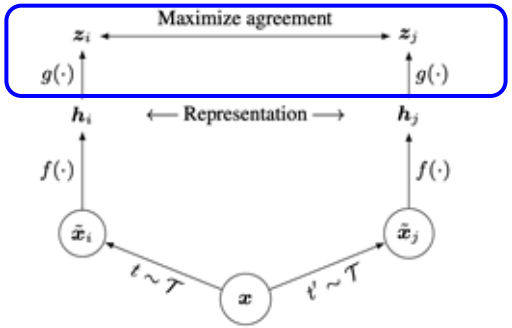
Train feature encoder on **ImageNet** (entire training set) using SimCLR.

**Finetune** the encoder with 1% / 10% of labeled data on ImageNet.

Source: [Chen et al., 2020]

# SimCLR design choices: projection head




Linear / non-linear projection heads improve representation learning.

A possible explanation:

- contrastive learning objective may discard useful information for downstream tasks
- representation space ***z*** is trained to be invariant to data transformation.
- by leveraging the projection head **g(·)**, more information can be preserved in the **h** representation space

Source: [Chen et al., 2020]

# SimCLR design choices: large batch size

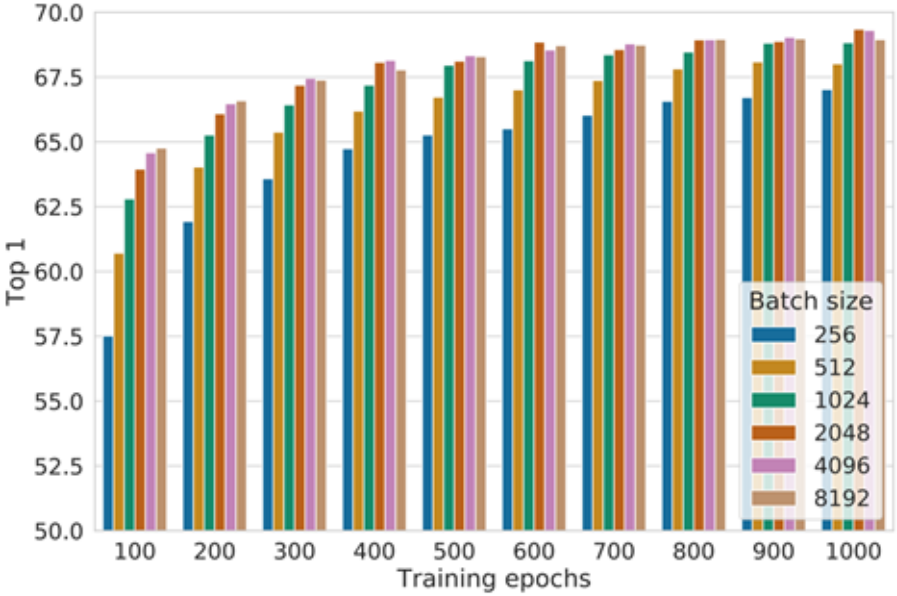


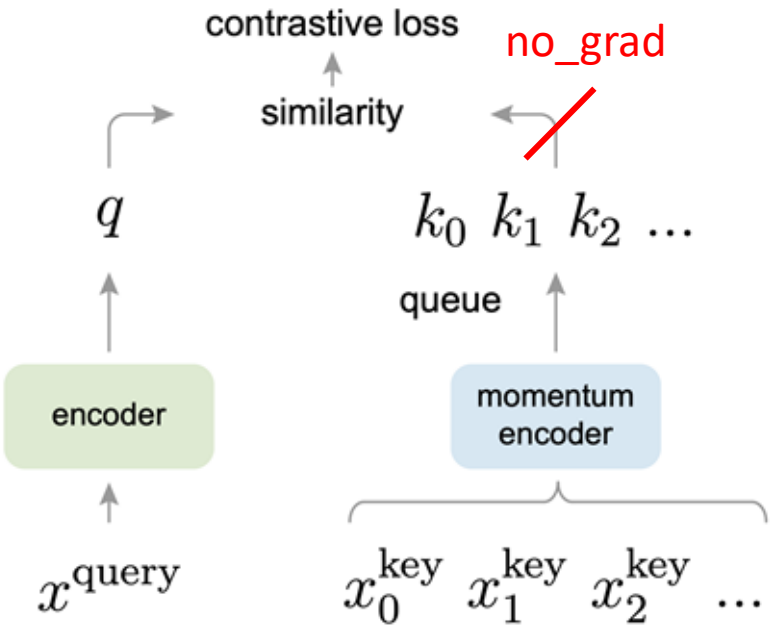

*Figure 9.* Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.[10]

Large training batch size is crucial for SimCLR!

Large batch size causes large memory footprint during backpropagation: requires distributed training on TPUs (ImageNet experiments)

Source: Chen et al., 2020

# Momentum Contrastive Learning (MoCo)

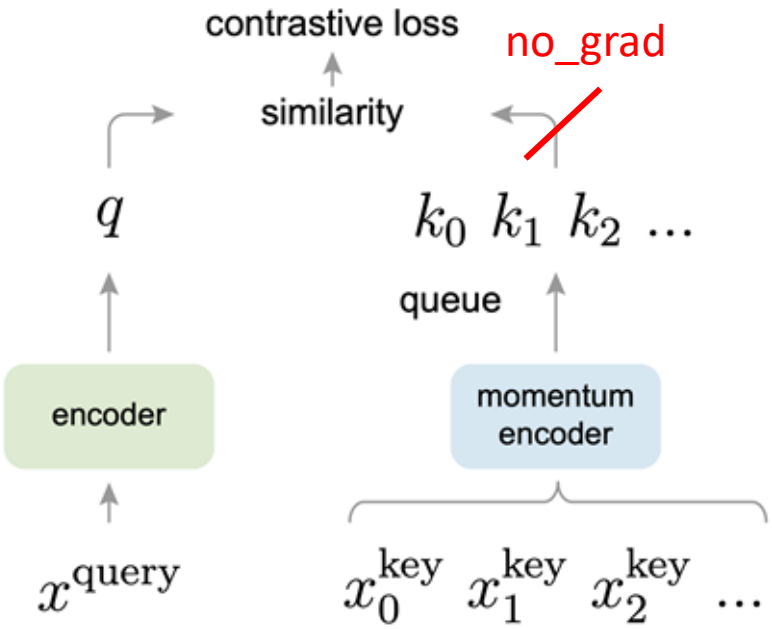



**Key differences to SimCLR:**

- Keep a running queue of keys (negative samples).
- Compute gradients and update the encoder only through the queries.
- Decouple min-batch size with the number of keys: can support a large number of negative samples.

Source: He et al., 2020

# Momentum Contrastive Learning (MoCo)




**Key differences to SimCLR:**

- Keep a running queue of keys (negative samples).
- Compute gradients and update the encoder only through the queries.
- Decouple min-batch size with the number of keys: can support a large number of negative samples.
- The key encoder is slowly progressing through the momentum update rules:

$$\theta_{\text{k}} \leftarrow m\theta_{\text{k}} + (1-m)\theta_{\text{q}}$$

Source: He et al., 2020

# MoCo

Generate a positive pair by sampling data augmentation functions

No gradient through the positive sample

**Algorithm 1** Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
   x_q = aug(x) # a randomly augmented version
   x_k = aug(x) # another randomly augmented version

   q = f_q.forward(x_q) # queries: NxC
   k = f_k.forward(x_k) # keys: NxC
   k = k.detach() # no gradient to keys

   # positive logits: Nx1
   l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

   # negative logits: NxK
   l_neg = mm(q.view(N,C), queue.view(C,K))

   # logits: Nx(1+K)
   logits = cat([l_pos, l_neg], dim=1)

   # contrastive loss, Eqn.(1)
   labels = zeros(N) # positives are the 0-th
   loss = CrossEntropyLoss(logits/t, labels)

   # SGD update: query network
   loss.backward()
   update(f_q.params)

   # momentum update: key network
   f_k.params = m*f_k.params+(1-m)*f_q.params

   # update dictionary
   enqueue(queue, k) # enqueue the current minibatch
   dequeue(queue) # dequeue the earliest minibatch
```

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.

Use the running queue of keys as the negative samples

InfoNCE loss

Update f_k through momentum

Update the FIFO negative sample queue

Source: [He et al., 2020]

# “MoCo V2”

**Improved Baselines with Momentum Contrastive Learning**

Xinlei Chen Haoqi Fan Ross Girshick Kaiming He
Facebook AI Research (FAIR)

A hybrid of ideas from SimCLR and MoCo:

- **From SimCLR**: non-linear projection head and strong data augmentation.
- **From MoCo**: momentum-updated queues that allow training on a large number of negative samples (no TPU required!).

Source: Chen et al., 2020

# MoCo vs. SimCLR vs. MoCo V2

| case | unsup. pre-train MLP | aug+ | cos | epochs | ImageNet acc. | VOC detection $AP_{50}$ | AP | $AP_{75}$ |
|---|---|---|---|---|---|---|---|---|
| supervised | | | | | 76.5 | 81.3 | 53.5 | 58.8 |
| MoCo v1 | | | | 200 | 60.6 | 81.5 | 55.9 | 62.6 |
| (a) | ✓ | | | 200 | 66.2 | 82.0 | 56.4 | 62.6 |
| (b) | | ✓ | | 200 | 63.4 | 82.2 | 56.8 | 63.2 |
| (c) | ✓ | ✓ | | 200 | 67.3 | **82.5** | 57.2 | 63.9 |
| (d) | ✓ | ✓ | ✓ | 200 | 67.5 | 82.4 | 57.0 | 63.6 |
| (e) | ✓ | ✓ | ✓ | **800** | **71.1** | **82.5** | **57.4** | **64.0** |

Table 1. **Ablation of MoCo baselines**, evaluated by ResNet-50 for (i) ImageNet linear classification, and (ii) fine-tuning VOC object detection (mean of 5 trials). "**MLP**": with an MLP head; "**aug+**": with extra blur augmentation; "**cos**": cosine learning rate schedule.

**Key takeaways:**

- Non-linear projection head and strong data augmentation are crucial for contrastive learning.

Source: [Chen et al., 2020]

# MoCo vs. SimCLR vs. MoCo V2

| case | unsup. pre-train MLP | aug+ | cos | epochs | batch | ImageNet acc. |
|---|---|---|---|---|---|---|
| MoCo v1 [6] | | | | 200 | 256 | 60.6 |
| SimCLR [2] | ✓ | ✓ | ✓ | 200 | 256 | 61.9 |
| SimCLR [2] | ✓ | ✓ | ✓ | 200 | 8192 | 66.6 |
| **MoCo v2** | ✓ | ✓ | ✓ | 200 | 256 | **67.5** |
| *results of **longer** unsupervised training follow:* | | | | | | |
| SimCLR [2] | ✓ | ✓ | ✓ | 1000 | 4096 | 69.3 |
| **MoCo v2** | ✓ | ✓ | ✓ | 800 | 256 | **71.1** |

Table 2. **MoCo *vs.* SimCLR**: ImageNet linear classifier accuracy (**ResNet-50, 1-crop 224×224**), trained on features from unsupervised pre-training. "aug+" in SimCLR includes blur and stronger color distortion. SimCLR ablations are from Fig. 9 in [2] (we thank the authors for providing the numerical results).

**Key takeaways:**

- Non-linear projection head and strong data augmentation are crucial for contrastive learning.
- Decoupling mini-batch size with negative sample size allows MoCo-V2 to outperform SimCLR with smaller batch size (256 vs. 8192).

Source: Chen et al., 2020

# MoCo vs. SimCLR vs. MoCo V2

| mechanism | batch | memory / GPU | time / 200-ep. |
|---|---|---|---|
| MoCo | 256 | **5.0G** | **53 hrs** |
| end-to-end | 256 | 7.4G | 65 hrs |
| end-to-end | 4096 | 93.0G† | n/a |

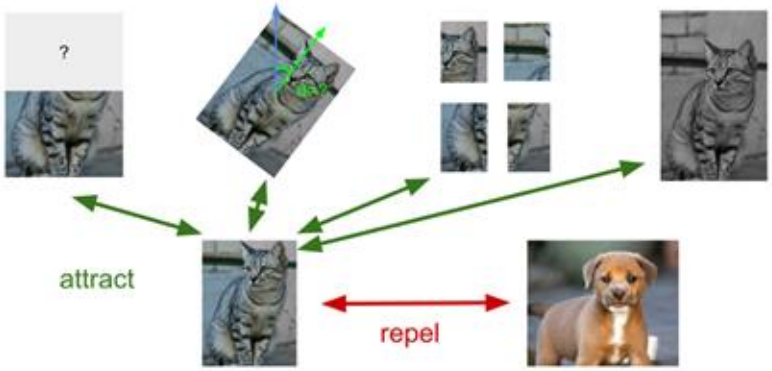
Table 3. **Memory and time cost** in 8 V100 16G GPUs, implemented in PyTorch. †: based on our estimation.

**Key takeaways:**

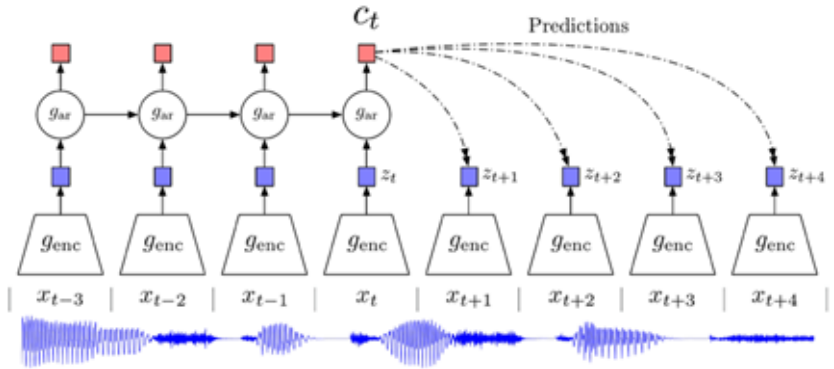
- Non-linear projection head and strong data augmentation are crucial for contrastive learning.
- Decoupling mini-batch size with negative sample size allows MoCo-V2 to outperform SimCLR with smaller batch size (256 vs. 8192).
- … all with much smaller memory footprint! ("end-to-end" means SimCLR here)

Source: Chen et al., 2020

# Instance vs. Sequence Contrastive Learning



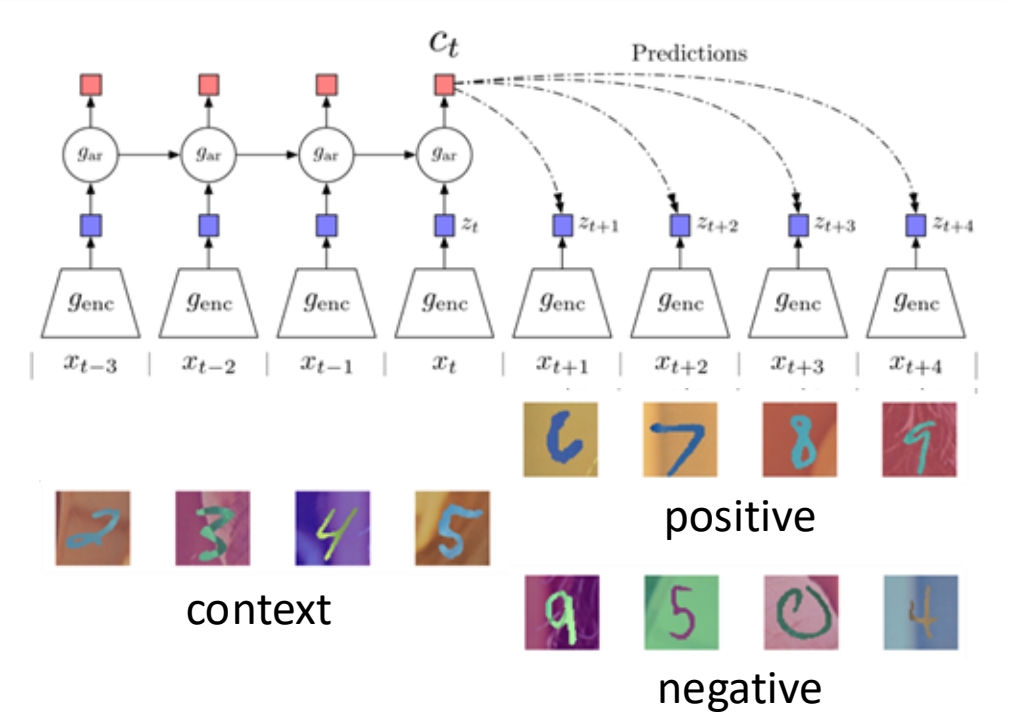

**Instance-level contrastive learning**: contrastive learning based on positive & negative instances.
Examples: SimCLR, MoCo




Source: van den Oord et al., 2018

**Sequence-level contrastive learning**: contrastive learning based on sequential / temporal orders.
Example: **Contrastive Predictive Coding (CPC)**

# Contrastive Predictive Coding (CPC)




**Contrastive**: contrast between "right" and "wrong" sequences using contrastive learning.

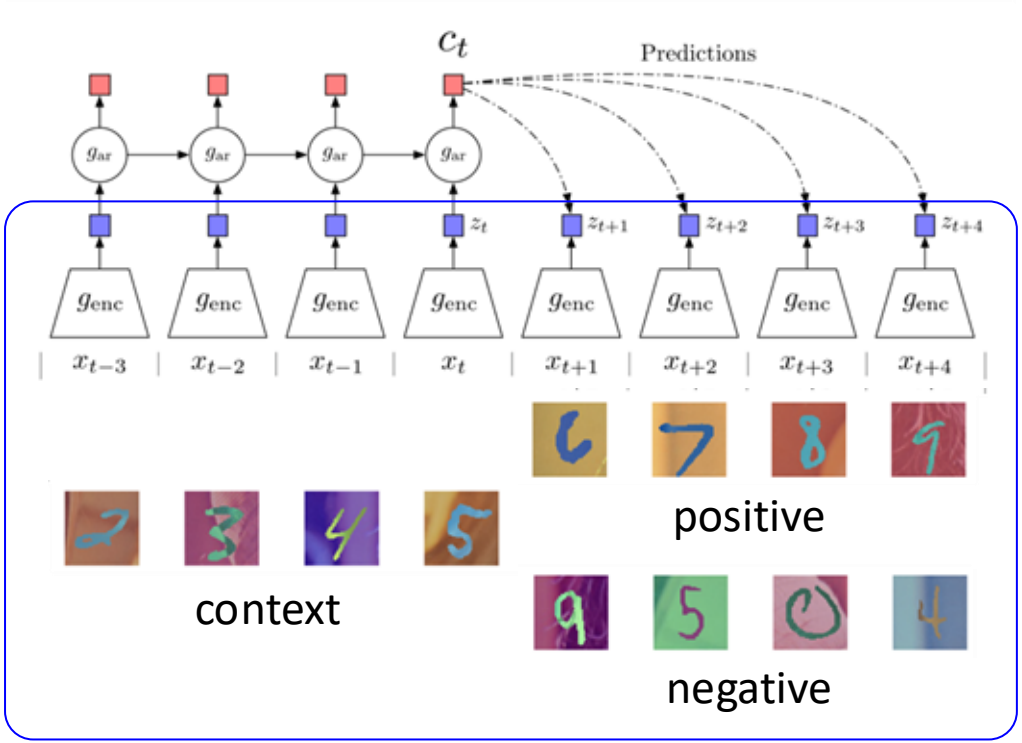
**Predictive**: the model has to predict future patterns given the current context.

**Coding**: the model learns useful feature vectors, or "code", for downstream tasks, similar to other self-supervised methods.

Figure source

Source: van den Oord et al., 2018,

# Contrastive Predictive Coding (CPC)

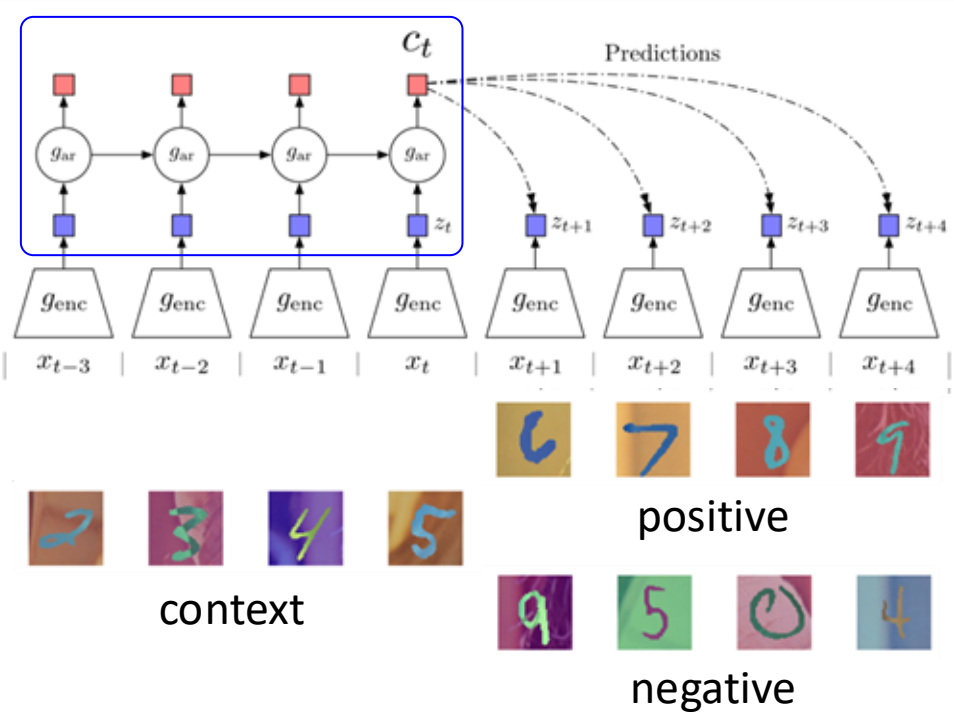



1. Encode all samples in a sequence into vectors $z_t = g_{enc}(x_t)$

Figure source

Source: van den Oord et al., 2018,

# Contrastive Predictive Coding (CPC)



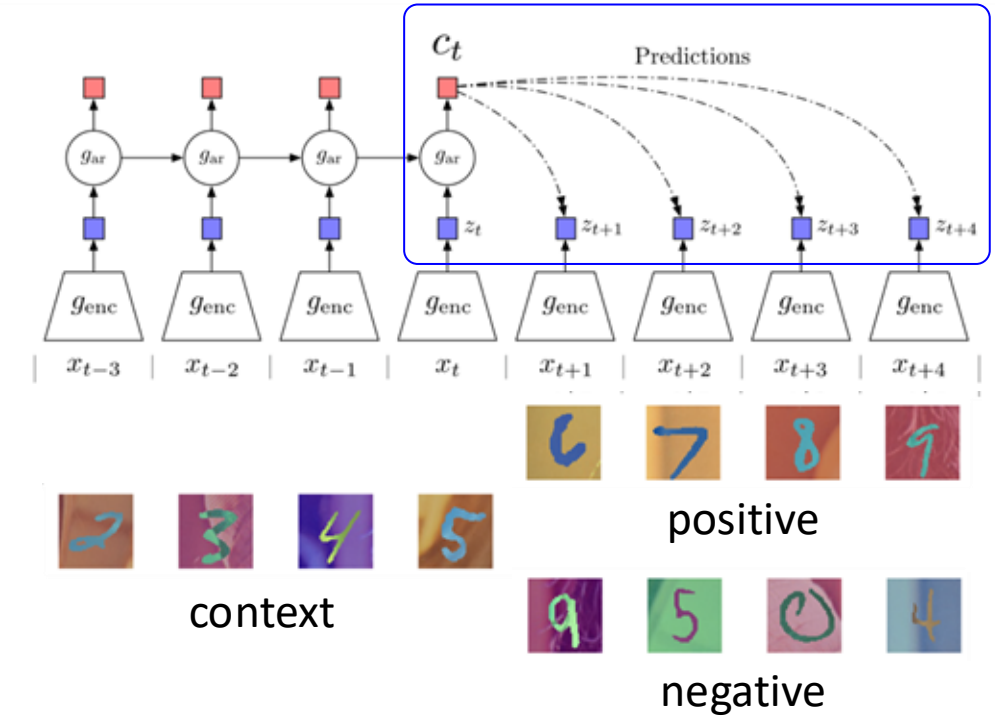

1. Encode all samples in a sequence into vectors $z_t = g_{enc}(x_t)$

2. Summarize context (e.g., half of a sequence) into a context code $c_t$ using an auto-regressive model ($g_{ar}$).

Figure source

Source: van den Oord et al., 2018,

# Contrastive Predictive Coding (CPC)




1. Encode all samples in a sequence into vectors $z_t = g_{enc}(x_t)$

2. Summarize context (e.g., half of a sequence) into a context code $c_t$ using an auto-regressive model ($g_{ar}$).

3. Compute InfoNCE loss between the context $c_t$ and future code $z_{t+k}$ using the following time-dependent score function:

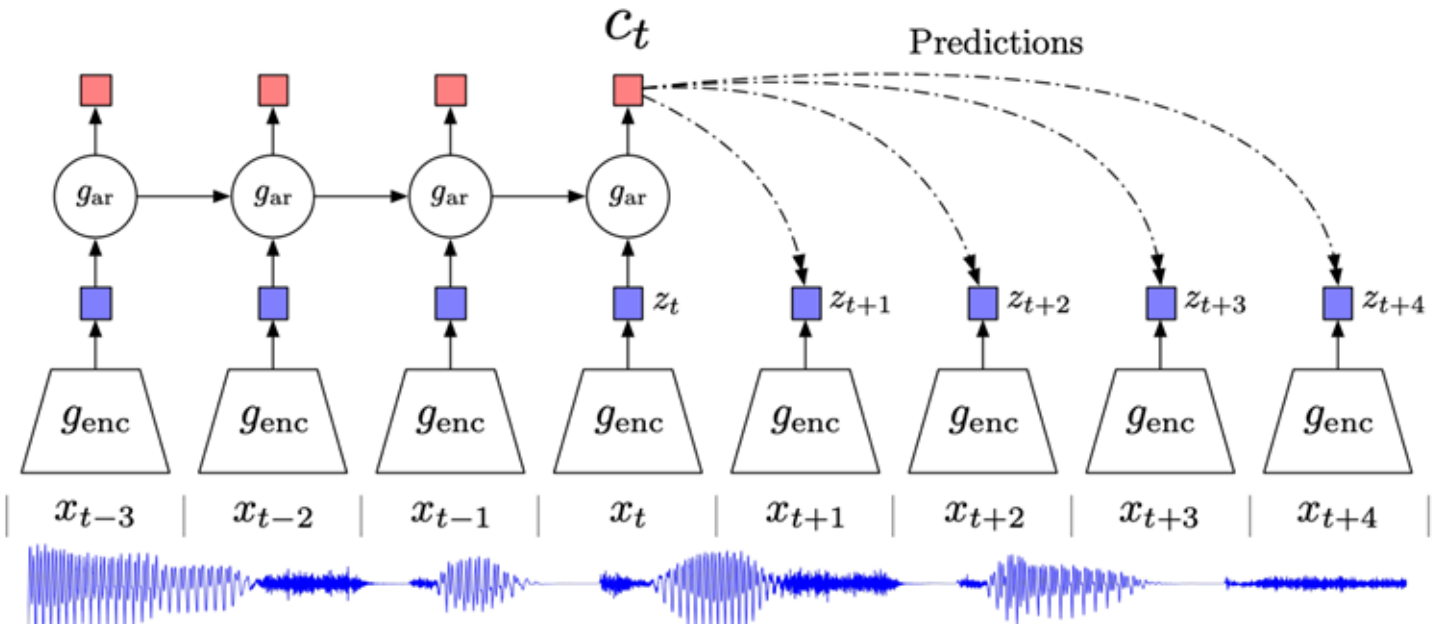
$$s_k(z_{t+k}, c_t) = z_{t+k}^T W_k c_t$$

, where $W_k$ is a trainable matrix.

Figure source

Source: van den Oord et al., 2018,

# CPC example: modeling audio sequences




Source: [van den Oord et al., 2018](), 

# CPC example: modeling audio sequences

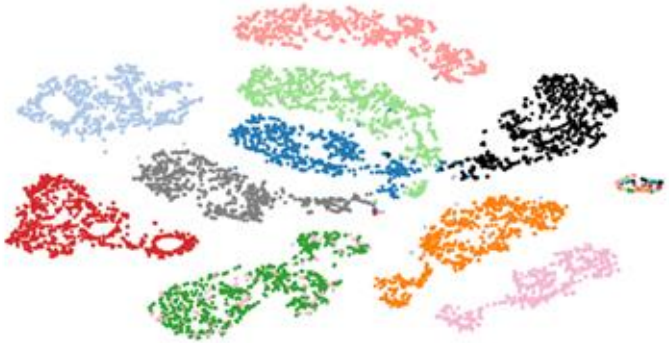


Figure 2: t-SNE visualization of audio (speech) representations for a subset of 10 speakers (out of 251). Every color represents a different speaker.

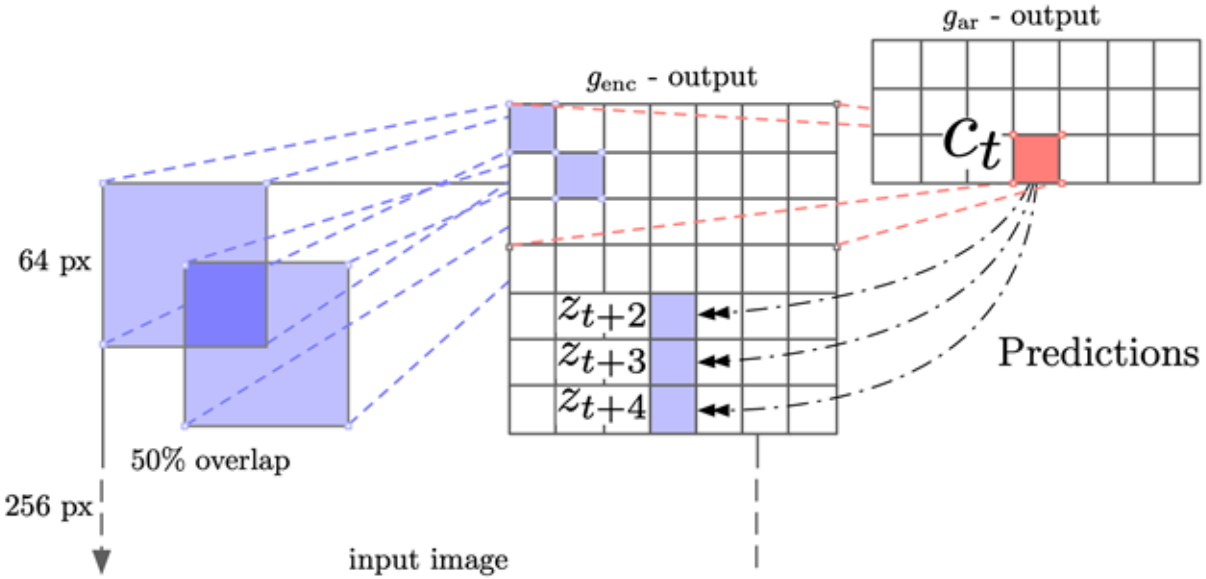
| Method | ACC |
|---|---|
| **Phone classification** | |
| Random initialization | 27.6 |
| MFCC features | 39.7 |
| CPC | 64.6 |
| Supervised | 74.6 |
| **Speaker classification** | |
| Random initialization | 1.87 |
| MFCC features | 17.6 |
| CPC | 97.4 |
| Supervised | 98.5 |

Linear classification on trained representations (LibriSpeech dataset)

Source: van den Oord et al., 2018,

# CPC example: modeling visual context

**Idea**: split image into patches, model rows of patches from top to bottom as a sequence. I.e., use top rows as context to predict bottom rows.



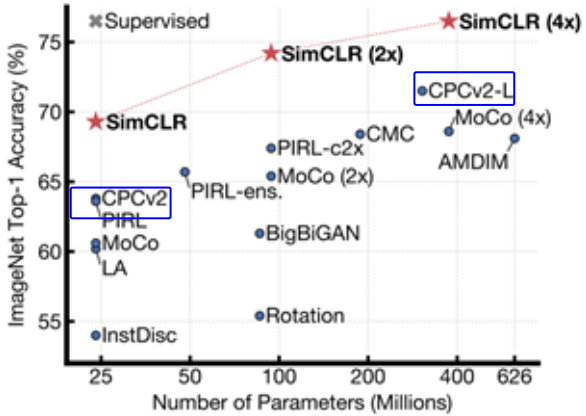

Source: [van den Oord et al., 2018](), 

# CPC example: modeling visual context

| Method | Top-1 ACC |
|---|---|
| **Using AlexNet conv5** | |
| Video [28] | 29.8 |
| Relative Position [11] | 30.4 |
| BiGan [35] | 34.8 |
| Colorization [10] | 35.2 |
| Jigsaw [29] * | 38.1 |
| **Using ResNet-V2** | |
| Motion Segmentation [36] | 27.6 |
| Exemplar [36] | 31.5 |
| Relative Position [36] | 36.2 |
| Colorization [36] | 39.6 |
| **CPC** | **48.7** |

Table 3: ImageNet top-1 unsupervised classification results. *Jigsaw is not directly comparable to the other AlexNet results because of architectural differences.

- Compares favorably with other pretext task-based self-supervised learning method.
- Doesn't do as well compared to newer instance-based contrastive learning methods on image feature learning.




Source: [van den Oord et al., 2018](), 

# Summary: Contrastive Representation Learning

A general formulation for contrastive learning:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

InfoNCE loss: N-way classification among positive and negative samples

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

Commonly known as the InfoNCE loss (van den Oord et al., 2018)

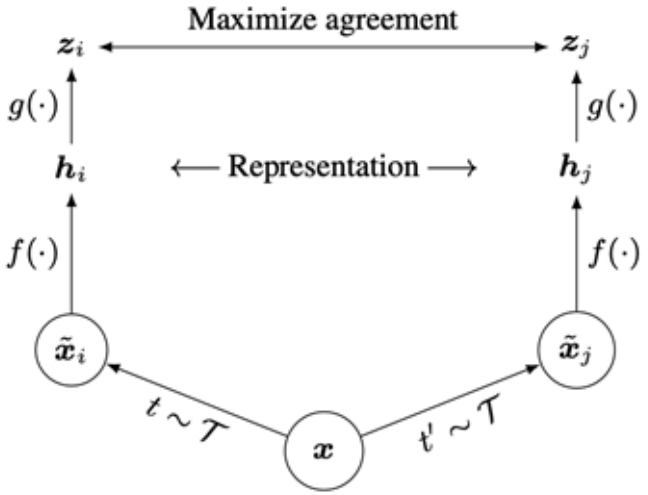
A *lower bound* on the mutual information between *f(x)* and $f(x^+)$

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

# Summary: Contrastive Representation Learning

**SimCLR**: a simple framework for contrastive representation learning

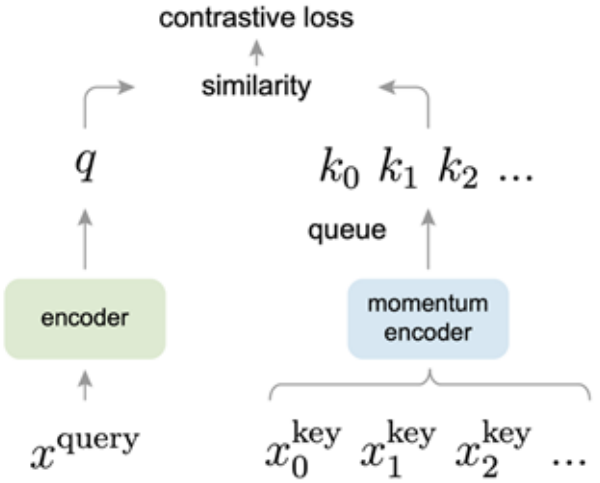
- **Key ideas**: non-linear projection head to allow flexible representation learning
- Simple to implement, effective in learning visual representation
- Requires large training batch size to be effective; large memory footprint



# Summary: Contrastive Representation Learning

**MoCo** (v1, v2): contrastive learning using momentum sample encoder

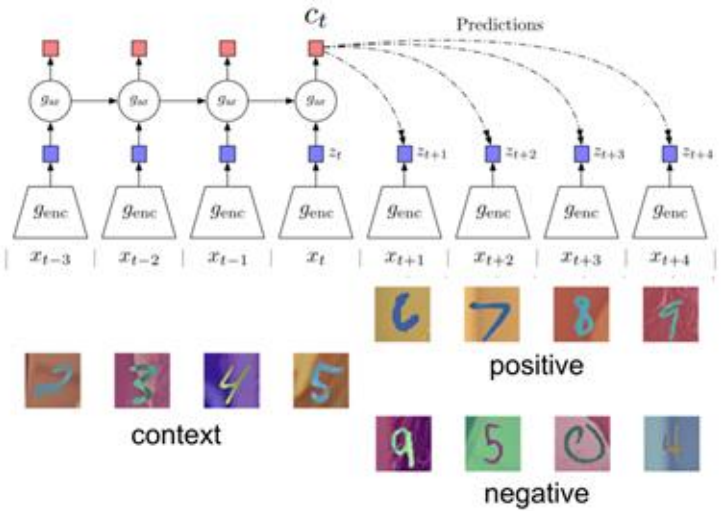
- Decouples negative sample size from minibatch size; allows large batch training without TPU
- MoCo-v2 combines the key ideas from SimCLR, i.e., nonlinear projection head, strong data augmentation, with momentum contrastive learning



# Summary: Contrastive Representation Learning

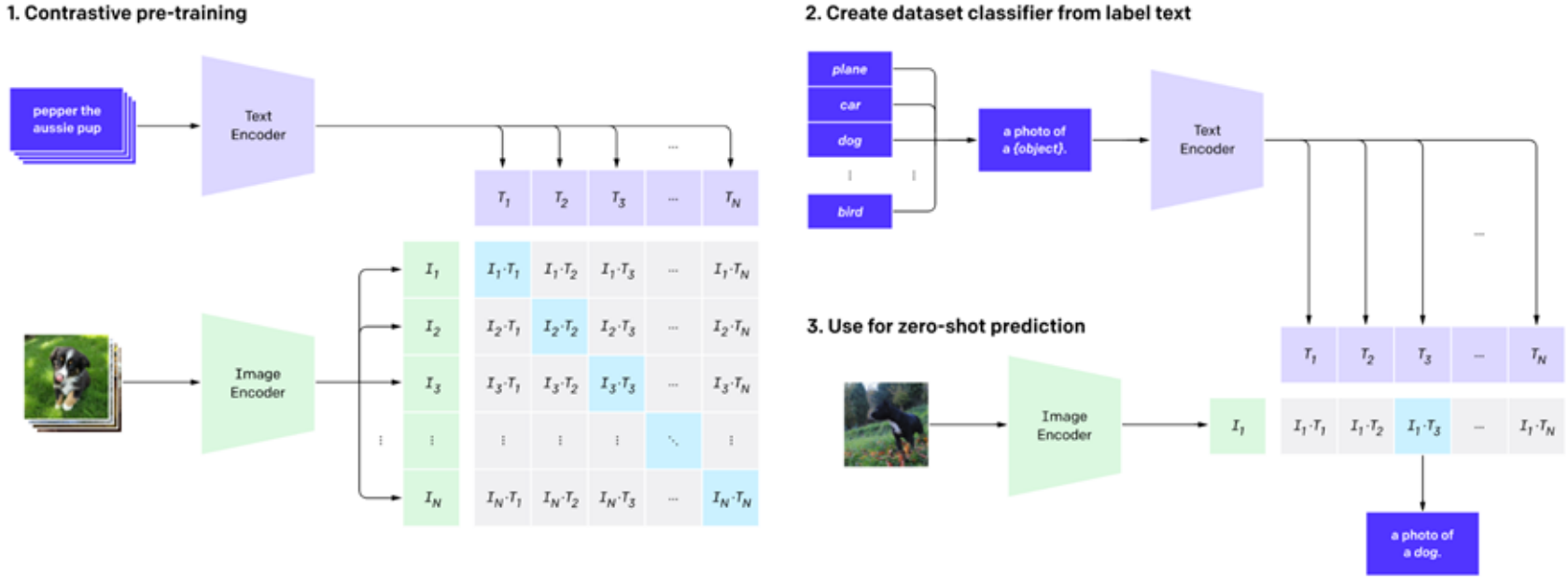
**CPC**: sequence-level contrastive learning

- Contrast “right” sequence with “wrong” sequence.
- InfoNCE loss with a time-dependent score function.
- Can be applied to a variety of learning problems, but not as effective in learning image representations compared to instance-level methods.



# Other examples

Contrastive learning between image and natural language sentences




CLIP (*Contrastive Language–Image Pre-training*) Radford *et al.*, 2021