Topics:

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

• HW2 / PS2 grade out. Please submit your regrade request by the end of this week
• HW4 / PS4 out. Due Nov 14th. Grade Period ends 16th.
• **Start the coding part NOW** --- it takes some time to run GAN / diffusion model training on Colab GPUs.
• Milestone Report & Video due Nov 7th. **NO GRACE PERIOD**
Denoising Diffusion: Image to Noise and Back

VAE

\[ x \rightarrow q(z|x) \rightarrow z \rightarrow p(x|z) \rightarrow \tilde{x} \]

Denoising Diffusion

\[ x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_T \rightarrow \ldots \rightarrow \tilde{x}_2 \rightarrow \tilde{x}_1 \rightarrow \tilde{x}_0 \]

Generative Adversarial Networks (GANs)

\[ z \rightarrow G(x|z) \rightarrow \tilde{x} \]
The Denoising Diffusion Process

The “forward diffusion” process:
add Gaussian noise each step

image from dataset

The “denoising diffusion” process:
generate an image from noise by
denoising the gaussian noises

$x_0 \rightarrow x_1 \rightarrow \cdots \rightarrow x_{T-1} \rightarrow x_T$
The Denoising (Decoding) Process

The learned denoising process

\[ x_0 \leftrightarrow x_1 \leftrightarrow \cdots \leftrightarrow x_T \]

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

**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:** \( \arg\min_\theta D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)) \)

What does it look like?

\[ q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \mu_q(t), \Sigma_q(t)) \]

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

Recall: Gaussian reparameterization trick

The “ground truth” noise that brought \( x_{t-1} \) to \( x_t \).
The Denoising (Decoding) Process

The learned denoising process:

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

**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:**

\[ \arg\min_\theta D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) \]

What does it look like?

\[ q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \mu_q(t), \Sigma_q(t)) \]

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

\[ \arg\min_\theta D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) = \arg\min_\theta \mathcal{W} ||\mu_q(t) - \mu_\theta(x_t, t)|| \]

Should be variance-dependent, but constant works better in practice.
The Denoising Diffusion Algorithm

**Algorithm 1** Training

1: repeat
2: \( x_0 \sim q(x_0) \)
3: \( t \sim \text{Uniform}\{1, \ldots, T\} \)
4: \( \epsilon \sim \mathcal{N}(0, I) \)
5: Take gradient descent step on
   \[ \nabla_{\theta} \| \epsilon - \epsilon(\sqrt{\alpha_t}x_0 + \sqrt{1 - \alpha_t}\epsilon, t) \|^2 \]
6: until converged

\( \mathcal{N}(0, I) \) → \( \epsilon \) → \( L = \| \epsilon - \tilde{\epsilon} \|^2 \) → \( \tilde{\epsilon} \) → Compute regression loss

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

**Algorithm 1 Training**

1: repeat
2: \( x_0 \sim q(x_0) \)
3: \( t \sim \text{Uniform}\{1, \ldots, T\} \)
4: \( \epsilon \sim \mathcal{N}(0, I) \)
5: Take gradient descent step on
   \[ \nabla_{\theta} \| \epsilon - \epsilon_{\theta}(\sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon, t) \|^2 \]
6: until converged

**Algorithm 2 Sampling**

1: \( x_T \sim \mathcal{N}(0, I) \)
2: for \( t = T, \ldots, 1 \) do
3: \( z \sim \mathcal{N}(0, I) \) if \( t > 1 \), else \( z = 0 \)
4: \( x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_{\theta}(x_t, t) \right) + \sigma_t z \)
5: end for
6: return \( x_0 \)
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
GANs: Learning to play a two-party game

VAE

\[ x \rightarrow q(z|x) \rightarrow z \rightarrow p(x|z) \rightarrow \tilde{x} \]

Denoising Diffusion

\[ x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_T \rightarrow \tilde{x}_2 \rightarrow \tilde{x}_1 \rightarrow \tilde{x}_0 \]

Generative Adversarial Networks (GANs)

\[ z \rightarrow G(x|z) \rightarrow \tilde{x} \]
Generative Adversarial Networks

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

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.

Input: Random noise

Output: Sample from training distribution

z

Generator Network
Generative Adversarial Networks

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 $\rightarrow$ can’t learn by reconstructing training images

Output: Sample from training distribution

Generator Network

Input: Random noise

$z$

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

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

Input: Random noise

Objective: generated images should look “real”
Generative Adversarial Networks

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

Discriminator Network

Real? Fake?

gradient
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

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

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

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

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

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

- Discriminator outputs likelihood in (0,1) of real image
- Discriminator output for real data \( x \)
- Discriminator output for generated fake data \( G_{\theta_g}(z) \)
- Classify all real images as real
- Classify all generated images as fake
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:

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

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

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

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

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

- 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)

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

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

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

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!

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

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!
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.
Training GANs: Two-player game

Putting it together: GAN training algorithm

\begin{align*}
\text{for number of training iterations do} \\
\quad \text{for } k \text{ steps do} \\
\quad \quad \bullet \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z).
\quad \bullet \text{Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{\text{data}}(x).
\quad \bullet \text{Update the discriminator by ascending its stochastic gradient:}
\quad \quad \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]
\quad \text{end for}
\quad \bullet \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z).
\quad \bullet \text{Update the generator by ascending its stochastic gradient (improved objective):}
\quad \quad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))
\quad \text{end for}
\end{align*}
Training GANs: Two-player game

Putting it together: GAN training algorithm

Update generator

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!

Training GANs: Two-player game

**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.
Generative Adversarial Nets

Generated samples

Nearest neighbor from training set

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

Generated samples (CIFAR-10)

Nearest neighbor from training set

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

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.

2019: BigGAN

Brock et al., 2019
Deep Generative Models

**VAE**

\[ x \rightarrow q(z|x) \rightarrow z \rightarrow p(x|z) \rightarrow \tilde{x} \]

**Denoising Diffusion**

\[ x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_T \rightarrow \tilde{x}_2 \rightarrow \tilde{x}_1 \rightarrow \tilde{x}_0 \]

**Generative Adversarial Networks (GANs)**

\[ z \rightarrow G(x|z) \rightarrow \tilde{x} \]
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 Model is extremely hot in 2023. More will come …
Supervised Learning
- Train Input: \( \{X, Y\} \)
- Learning output: \( f : X \rightarrow Y, P(y|x) \)
- e.g. classification

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

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


Learning to generate pixel-level details is often unnecessary; learn high-level semantic 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**
- Dense Object Net (Florence and Manuelli et al., 2018)

**Language modeling**
- Wavenet (van den Oord et al., 2016)
- GPT3 (Brown, Mann, Ryder, Subbiah et al., 2020)

**Speech synthesis**
- Today’s lecture

**Robot / reinforcement learning**
- Doersch et al., 2015
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
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

Image $X$ is rotated by 0, 90, 180, or 270 degrees, resulting in rotated images $X^0$, $X^1$, $X^2$, and $X^3$, respectively.

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

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

- **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**.

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

Example:

Question 1:

Question 2:

(Image source: 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].

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

“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

Grayscale image: $L$ channel

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

Color information: $ab$ channels

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

Source: Richard Zhang / Phillip Isola
Pretext task: image coloring

Grayscale image: $L$ channel

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

$F$

Concatenate $(L, ab)$ channels

$(X, \hat{Y})$

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: image coloring

Source: Richard Zhang / Phillip Isola
Pretext task: image coloring

Source: Richard Zhang / Phillip Isola
Pretext task: video coloring

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

- Reference frame
- How should I color these frames?

Source: [Vondrick et al., 2018](#)
Pretext task: video coloring

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

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

\[ A_{ij} = \frac{\exp(\mathbf{f}_i^T \mathbf{f}_j)}{\sum_k \exp(\mathbf{f}_k^T \mathbf{f}_j)} \]

Source: Vondrick et al., 2018
Learning to color videos

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

Attention map on the reference frame

\[ y_j = \sum_i A_{ij} c_i \]

Predicted color = weighted sum of the reference color

Source: Vondrick et al., 2018
Learning to color videos

\[
A_{ij} = \frac{\exp(f_i^T f_j)}{\sum_k \exp(f_k^T f_j)}
\]

\[
y_j = \sum_i A_{ij} c_i
\]

\[
\min_{\theta} \sum_j \mathcal{L}(y_j, c_j)
\]

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 to 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).
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 to 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?

same object
A more general pretext task?

same object

different object
Contrastive Representation Learning

attract

repel
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

attract

repel
Contrastive Representation Learning

$x^+$ reference
$x^+$ positive
$x^-$ negative
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:

\[
L = -\mathbb{E}_x \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))}} \right]
\]
A formulation of contrastive learning

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

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

This seems familiar …
A formulation of contrastive learning

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

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

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 \( h(\cdot) \) 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: \textit{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.

1. **Input**: batch size $N$, constant $\tau$, structure of $f$, $g$, $\mathcal{T}$.
2. **For** sampled minibatch $\{x_k\}_{k=1}^N$ do
   1. **For all** $k \in \{1, \ldots, N\}$ do
      1. Draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
      2. \# the first augmentation
         - $\bar{x}_{2k-1} = t(x_k)$
         - $h_{2k-1} = f(\bar{x}_{2k-1})$ \# representation
         - $z_{2k-1} = g(h_{2k-1})$ \# projection
      3. \# the second augmentation
         - $\bar{x}_{2k} = t'(x_k)$
         - $h_{2k} = f(\bar{x}_{2k})$ \# representation
         - $z_{2k} = g(h_{2k})$ \# projection
   3. **For all** $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ do
      4. $s_{i,j} = z_i^\top z_j / (\|z_i\| \|z_j\|)$ \# pairwise similarity
   4. **Define** $\ell(i,j)$ as $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} 1_{[k \neq i]} \exp(s_{i,k}/\tau)}$
   5. $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
   6. Update networks $f$ and $g$ to minimize $\mathcal{L}$
5. **End for**
6. **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 $\{x_k\}_{k=1}^N$ do |
| for all $k \in \{1, \ldots, N\}$ do |
| draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$ |
| # the first augmentation |
| $\tilde{x}_{2k-1} = t(x_k)$ |
| $h_{2k-1} = f(\tilde{x}_{2k-1})$ # representation |
| $z_{2k-1} = g(h_{2k-1})$ # projection |
| # the second augmentation |
| $\tilde{x}_{2k} = t'(x_k)$ |
| $h_{2k} = f(\tilde{x}_{2k})$ # representation |
| $z_{2k} = g(h_{2k})$ # projection |
| end for |
| for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ do |
| $s_{i,j} = z_i^T z_j / (\|z_i\| \|z_j\|)$ # 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**

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 $x^-$

Algorithm 1 SimCLR’s main learning algorithm.

```plaintext
Algorithm 1 SimCLR’s main learning algorithm.

input: batch size $N$, constant $\tau$, structure of $f, g, \mathcal{T}$.

for sampled minibatch $\{x_k\}_{k=1}^N$ do

for all $k \in \{1, \ldots, N\}$ do

draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$

# the first augmentation

$\tilde{x}_{2k-1} = t(x_k)$

$h_{2k-1} = f(\tilde{x}_{2k-1})$ # representation

$z_{2k-1} = g(h_{2k-1})$ # projection

# the second augmentation

$\tilde{x}_{2k} = t'(x_k)$

$h_{2k} = f(\tilde{x}_{2k})$ # representation

$z_{2k} = g(h_{2k})$ # projection

end for

for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ do

$s_{i,j} = z_i^\top z_j / (\|z_i\| \|z_j\|)$ # pairwise similarity

end for

define $\ell(i,j)$ as

$\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N, i \neq k} \exp(s_{i,k}/\tau)}$

$L = \frac{1}{2N} \sum_{k=1}^N [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$

update networks $f$ and $g$ to minimize $L$

end for

return encoder network $f(\cdot)$, and throw away $g(\cdot)$
```

Source: Chen et al., 2020
SimCLR: mini-batch training

list of positive pairs

Each 2k and 2k + 1 element is a positive pair

$z \in \mathbb{R}^{2N \times D}$

$s_{i,j} = \frac{z_i^T z_j}{\|z_i\| \|z_j\|}$

“Affinity matrix”
SimCLR: mini-batch training

Each 2k and 2k + 1 element is a positive pair

\[ s_{i,j} = \frac{z_i^T z_j}{||z_i|| \cdot ||z_j||} \]

“Affinity matrix”

Each row is a classification label
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Architecture</th>
<th>Label fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1%</td>
</tr>
<tr>
<td>Supervised baseline</td>
<td>ResNet-50</td>
<td>48.4</td>
</tr>
<tr>
<td>Methods using other label-propagation:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pseudo-label</td>
<td>ResNet-50</td>
<td>51.6</td>
</tr>
<tr>
<td>VAT+Entropy Min.</td>
<td>ResNet-50</td>
<td>47.0</td>
</tr>
<tr>
<td>UDA (w. RandAug)</td>
<td>ResNet-50</td>
<td>-</td>
</tr>
<tr>
<td>FixMatch (w. RandAug)</td>
<td>ResNet-50</td>
<td>-</td>
</tr>
<tr>
<td>S4L (Rot+VAT+En. M.)</td>
<td>ResNet-50 (4×)</td>
<td>-</td>
</tr>
<tr>
<td>Methods using representation learning only:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InstDisc</td>
<td>ResNet-50</td>
<td>39.2</td>
</tr>
<tr>
<td>BigBiGAN</td>
<td>RevNet-50 (4×)</td>
<td>55.2</td>
</tr>
<tr>
<td>PIRL</td>
<td>ResNet-50</td>
<td>57.2</td>
</tr>
<tr>
<td>CPC v2</td>
<td>ResNet-161(*)</td>
<td>77.9</td>
</tr>
<tr>
<td>SimCLR (ours)</td>
<td>ResNet-50</td>
<td>75.5</td>
</tr>
<tr>
<td>SimCLR (ours)</td>
<td>ResNet-50 (2×)</td>
<td>83.0</td>
</tr>
<tr>
<td>SimCLR (ours)</td>
<td>ResNet-50 (4×)</td>
<td><strong>85.8</strong></td>
</tr>
</tbody>
</table>

*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 $g(\cdot)$, more information can be preserved in the $h$ representation space

Source: Chen et al., 2020
SimCLR design choices: large batch size

Large training batch size is crucial for SimCLR!

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

Figure 9. Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.\textsuperscript{10}

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:
    \[ \theta_k \gets m\theta_k + (1 - m)\theta_q \]

Source: He et al., 2020
Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```python
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (C x K)
# 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
```

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

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

<table>
<thead>
<tr>
<th>case</th>
<th>unsup. pre-train</th>
<th>ImageNet acc.</th>
<th>VOC detection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MLP</td>
<td>aug+</td>
<td>cos</td>
</tr>
<tr>
<td>supervised</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoCo v1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(c)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(d)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(e)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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.

Source: Chen et al., 2020
MoCo vs. SimCLR vs. MoCo V2

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).

<table>
<thead>
<tr>
<th>case</th>
<th>MLP</th>
<th>aug+</th>
<th>cos</th>
<th>epochs</th>
<th>batch</th>
<th>ImageNet acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoCo v1 [6]</td>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>256</td>
<td>60.6</td>
</tr>
<tr>
<td>SimCLR [2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>200</td>
<td>256</td>
<td>61.9</td>
</tr>
<tr>
<td>SimCLR [2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>200</td>
<td>8192</td>
<td>66.6</td>
</tr>
<tr>
<td>MoCo v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>200</td>
<td>256</td>
<td>67.5</td>
</tr>
</tbody>
</table>

Results of longer unsupervised training follow:

<table>
<thead>
<tr>
<th>case</th>
<th>MLP</th>
<th>aug+</th>
<th>cos</th>
<th>epochs</th>
<th>batch</th>
<th>ImageNet acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimCLR [2]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1000</td>
<td>4096</td>
<td>69.3</td>
</tr>
<tr>
<td>MoCo v2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>800</td>
<td>256</td>
<td>71.1</td>
</tr>
</tbody>
</table>

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).

Source: Chen et al., 2020
## MoCo vs. SimCLR vs. MoCo V2

<table>
<thead>
<tr>
<th>mechanism</th>
<th>batch</th>
<th>memory / GPU</th>
<th>time / 200-ep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoCo</td>
<td>256</td>
<td>5.0G</td>
<td>53 hrs</td>
</tr>
<tr>
<td>end-to-end</td>
<td>256</td>
<td>7.4G</td>
<td>65 hrs</td>
</tr>
<tr>
<td>end-to-end</td>
<td>4096</td>
<td>93.0G†</td>
<td>n/a</td>
</tr>
</tbody>
</table>

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

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

Source: van den Oord et al., 2018
**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 $z_t = g_{enc}(x_t)$

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 ($g_{ar}$).

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.

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Phone classification</strong></td>
<td></td>
</tr>
<tr>
<td>Random initialization</td>
<td>27.6</td>
</tr>
<tr>
<td>MFCC features</td>
<td>39.7</td>
</tr>
<tr>
<td>CPC</td>
<td>64.6</td>
</tr>
<tr>
<td>Supervised</td>
<td>74.6</td>
</tr>
<tr>
<td><strong>Speaker classification</strong></td>
<td></td>
</tr>
<tr>
<td>Random initialization</td>
<td>1.87</td>
</tr>
<tr>
<td>MFCC features</td>
<td>17.6</td>
</tr>
<tr>
<td>CPC</td>
<td>97.4</td>
</tr>
<tr>
<td>Supervised</td>
<td>98.5</td>
</tr>
</tbody>
</table>

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.
CPC example: modeling visual context

- 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using AlexNet conv5</td>
<td>29.8</td>
</tr>
<tr>
<td>Video [28]</td>
<td></td>
</tr>
<tr>
<td>BiGan [35]</td>
<td>34.8</td>
</tr>
<tr>
<td>Colorization [10]</td>
<td>35.2</td>
</tr>
<tr>
<td>Jigsaw [29] *</td>
<td>38.1</td>
</tr>
</tbody>
</table>

| 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.

Source: van den Oord et al., 2018,
Summary: Contrastive Representation Learning

A general formulation for contrastive learning:

\[
\text{score}(f(x), f(x^+)) \gg \text{score}(f(x), f(x^-))
\]

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
Other examples

Contrastive learning on pixel-wise feature descriptors

Dense Object Net, Florence et al., 2018
Other examples

Dense Object Net, Florence et al., 2018
Next Lecture: Large Vision and Language Model