photo by unsplash user @tuvaloland

# CMP784

## DEEP LEARNING

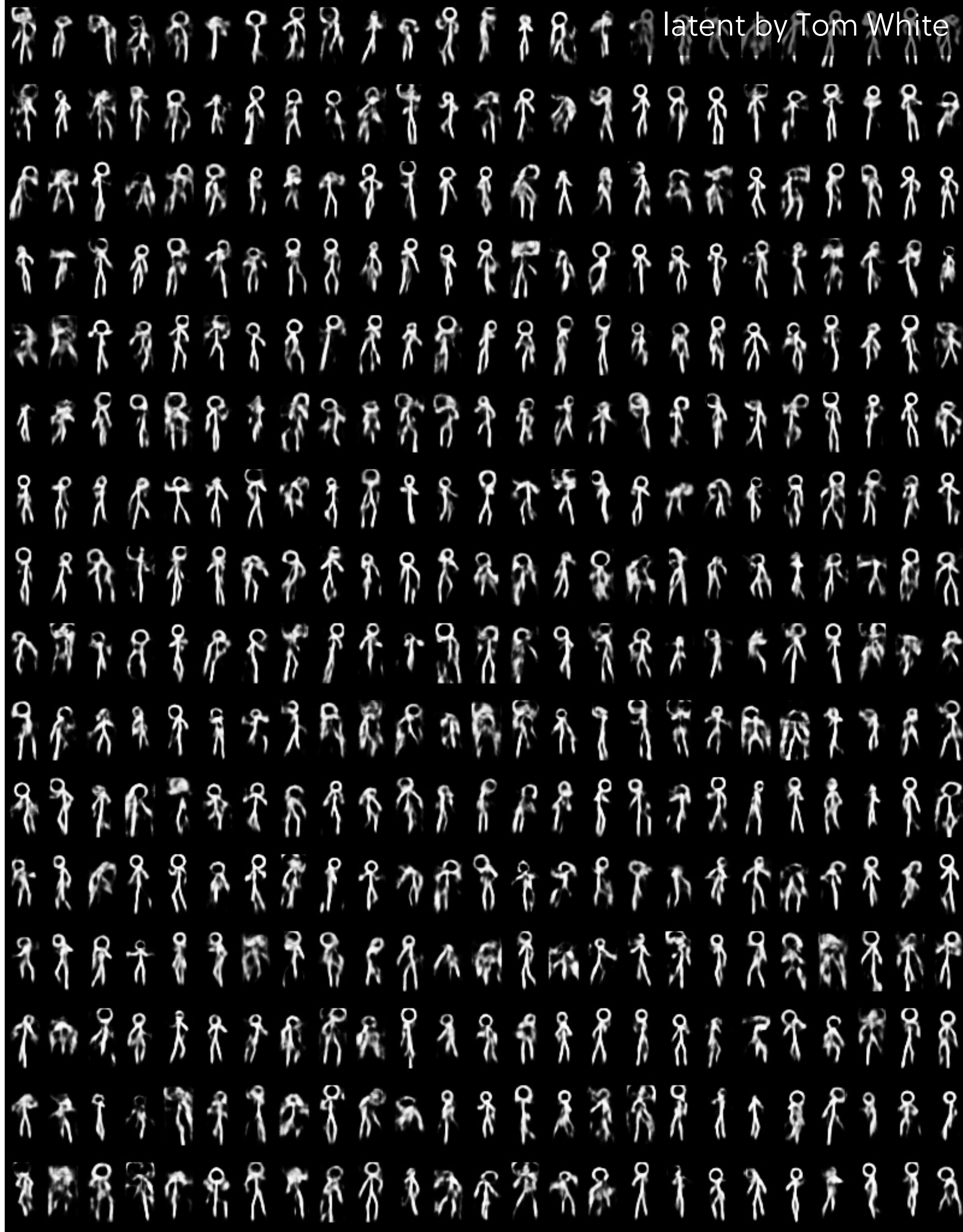
Lecture #12 – Self-Supervised Learning

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Erkut Erdem // Hacettepe University // Fall 2021

# Previously on CMP784

- Motivation for Variational Autoencoders (VAEs)
- Mechanics of VAEs
- Separatibility of VAEs
- Training of VAEs
- Evaluating representations
- Vector Quantized Variational Autoencoders (VQ-VAEs)



## Lecture Overview

- Predictive / Self-supervised learning
- Self-supervised learning in NLP
- Self-supervised learning in vision

**Disclaimer:** Much of the material and slides for this lecture were borrowed from

—Andrej Risteski's CMU 10707 class

—Jimmy Ba's UToronto CSC413/2516 class

—Fei-Fei Li, Ranjay Krishna, Danfei Xu’s CS231n class

# Unsupervised Learning

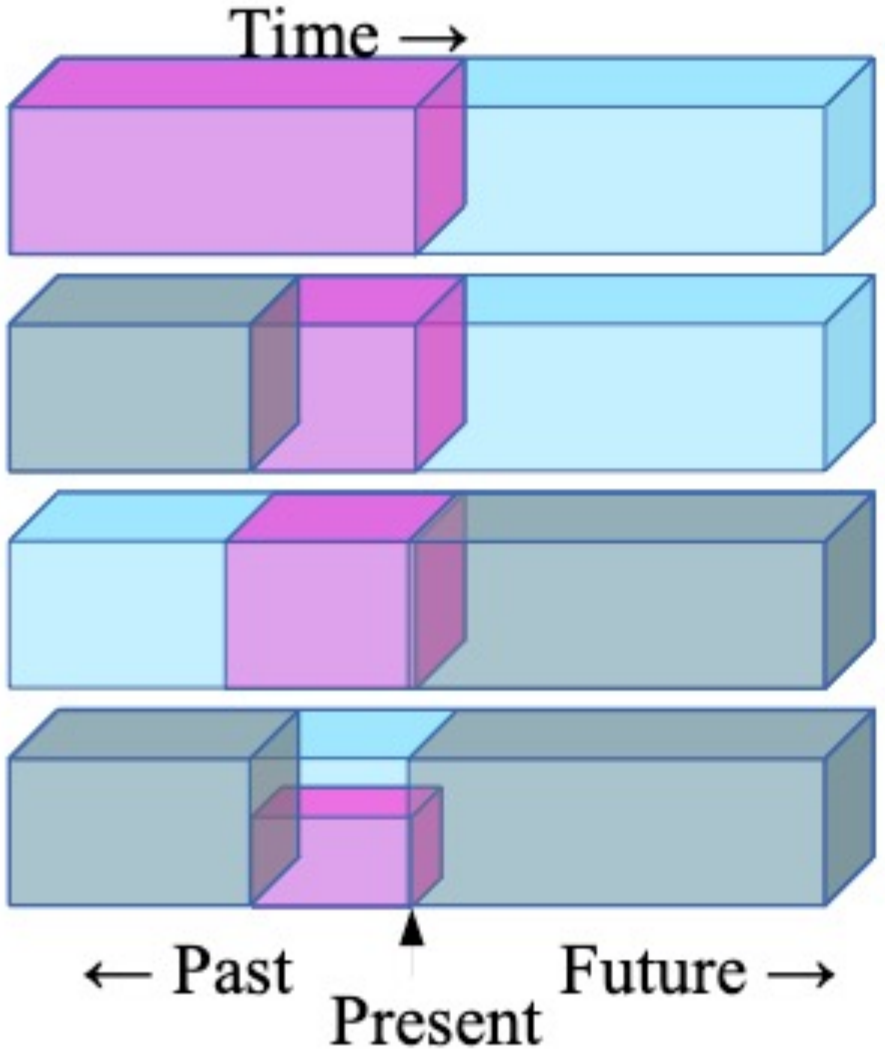
- Learning from data **without** labels.
- What can we hope to do:
  - **Task A:** Fit a parametrized structure (e.g. clustering, low-dimensional subspace, manifold) to data to reveal something meaningful about data **(Structure learning)**
  - **Task B:** Learn a (parametrized) distribution close to data generating distribution. **(Distribution learning)**
  - **Task C:** Learn a (parametrized) distribution that implicitly reveals an "embedding"/"representation" of data for downstream tasks. **(Representation/feature learning)**
- Entangled! The "structure" and "distribution" often reveals an embedding.

## Self-Supervised/Predictive Learning

- Given unlabeled data, design supervised tasks that induce a good representation for downstream tasks.
- No good mathematical formalization, but the intuition is to “force” the predictor used in the task to learn something “semantically meaningful” about the data.

# Self-Supervised/Predictive Learning

- ▶ Predict any part of the input from any other part.
- ▶ Predict the future from the past.
- ▶ Predict the future from the recent past.
- ▶ Predict the past from the present.
- ▶ Predict the top from the bottom.
- ▶ Predict the occluded from the visible
- ▶ **Pretend there is a part of the input you don't know and predict that.**



## How Much Information Does the Machine Need to Predict?

Y LeCun

- "Pure" Reinforcement Learning (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- Supervised Learning (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- Unsupervised/Predictive Learning (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

- LeCun's original cake analogy slide, presented at his keynote speech in NIPS 2016.

## How Much Information is the Machine Given during Learning?

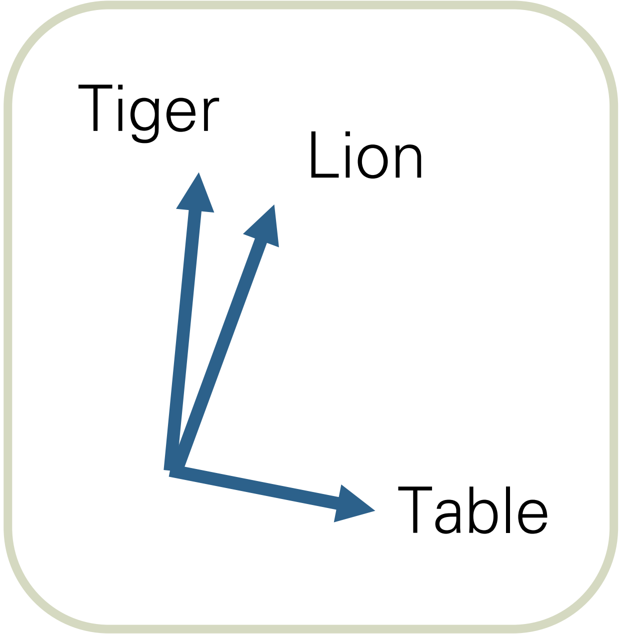
- **“Pure” Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- Updated version at (ISSCC 2019, where he replaced “**unsupervised learning**” with “**self-supervised learning**”.

## Self-Supervised Learning in NLP

## Word Embeddings

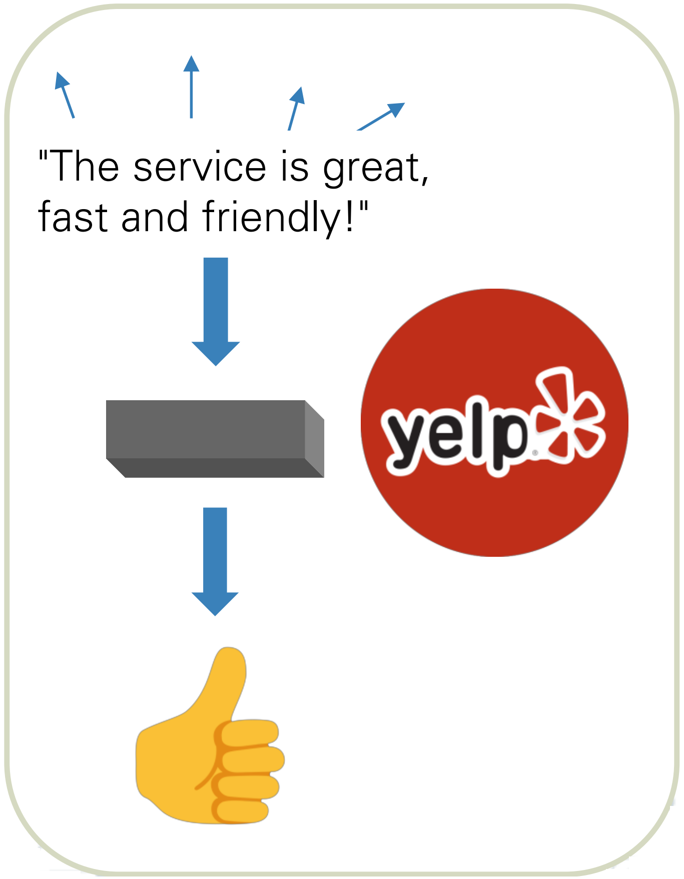
- Semantically meaningful **vector representations** of words



<u>Example</u>: Inner product (possibly scaled, i.e. cosine similarity) correlates with word similarity.

# Word Embeddings

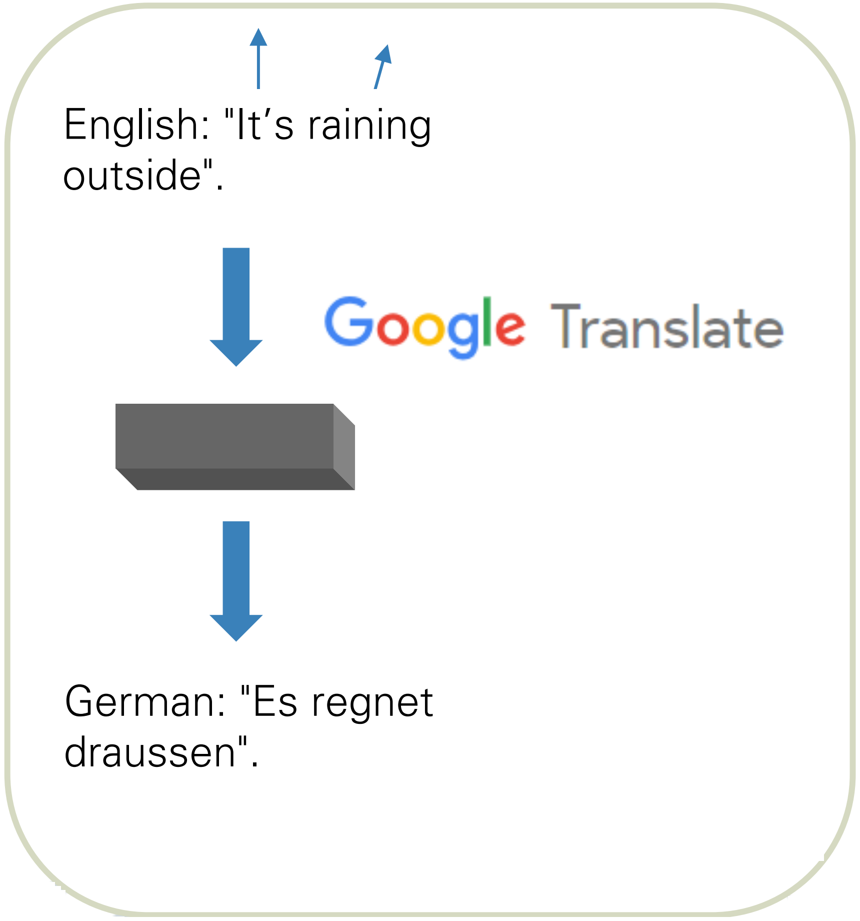
- Semantically meaningful **vector representations** of words



Example: Can use embeddings to do sentiment classification by training a simple (e.g. linear) classifier

# Word Embeddings

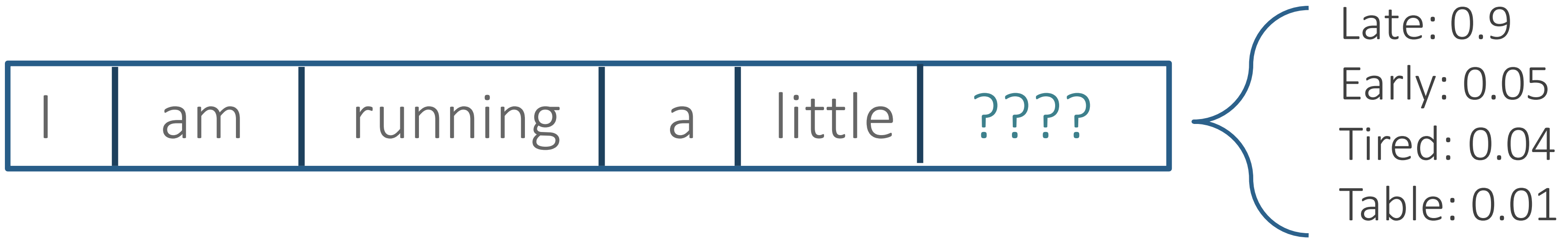
- Semantically meaningful **vector representations** of words


English: "It's raining outside".
Google Translate
German: "Es regnet draussen".

Example: Can train a "simple" network that if fed word embeddings for two languages, can effectively translate.

# Word Embeddings via Predictive Learning

- **Basic task:** predict the next word, given a few previous ones.



In other words, optimize for

$$\max_{\theta} \sum_{t} \log p_{\theta} \left( x_t \middle| x_{t-1}, x_{t-2}, \ldots, x_{t-L} \right)$$

# Word Embeddings via Predictive Learning

- **Basic task:** predict the next word, given a few previous ones.

$$\max_{\theta} \sum_{t} \log p_{\theta}\left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L}\right)$$

Inspired by classical assumptions in NLP that the underlying distribution is Markov – that is, $x_t$ only depends on the previous few words.

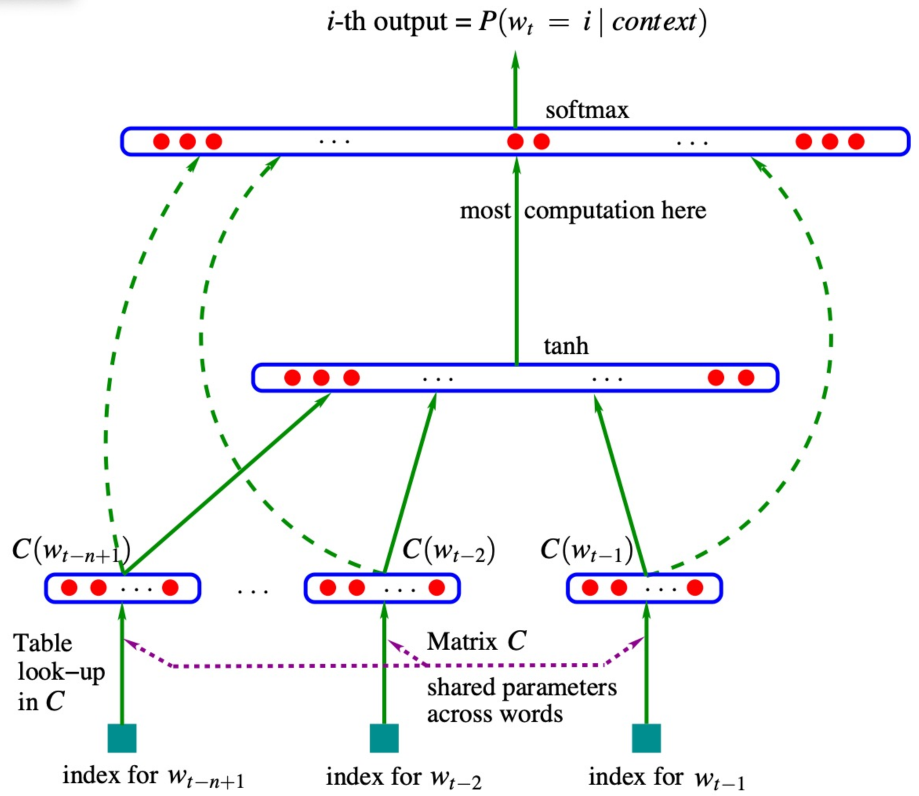
(Of course, this is violated if you wish to model long texts like paragraphs / books.)

**The main issue:** The trivial way of parametrizing $p_{\theta}\left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L}\right)$ is a “lookup table” with $V^L$ entries.

# Word Embeddings via Predictive Learning

- **Basic task:** predict the next word, given a few previous ones.

$$\max_{\theta} \sum_{t} \log p_{\theta}\left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L}\right)$$



[Bengio-Ducharme-Vincent-Janvin '2003]: A neural parametrization of the above probabilities.

**Main ingredients:**

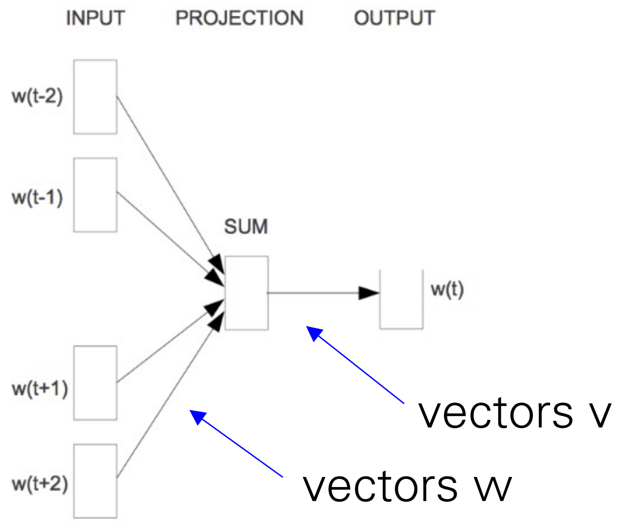
- Embeddings: A word embedding $C(w)$ for all words $w$ in dictionary.
- Non-linear transforms: Potentially deep network taking as inputs i, $C(x_{t-1})$, $C(x_{t-2})$,…,$C(x_{t-L})$, and outputting some vector o. Can be recurrent net too.
- Softmax: Softmax distribution for $x_t$ with parameters given by o.

## Word Embeddings via Predictive Learning

- **Related:** predict middle word in a sentence, given surrounding ones.

$$\max_{\theta} \sum_{t} \log p_{\theta} \left( x_t | x_{t-L}, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{t+L} \right)$$

**CBOW (Continuous Bag of Words):** proposed by Mikolov et al. '13



Parametrization is chosen s.t.

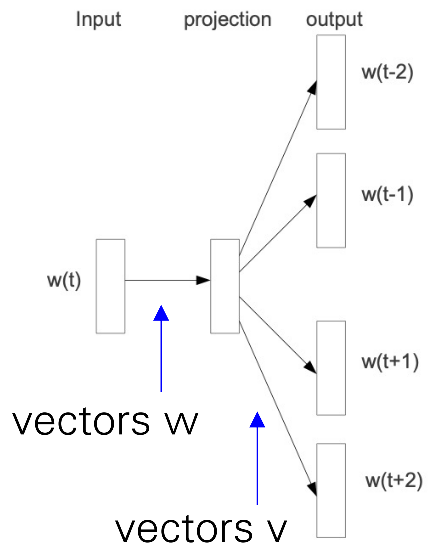
$$p_{\theta} \left( x_t | x_{t-L}, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{t+L} \right) \propto \exp \left( v_{x_t}, \textstyle\sum_{i=t-L}^{t+L} w_{t_i} \right)$$

# Word Embeddings via Predictive Learning

- **Related:** predict middle word in a sentence, given surrounding ones.

$$\max_\theta \sum_t \sum_{i=t-L, i\neq t}^{t+L} \log p_\theta\left(x_i | x_t\right)$$

**Skip-Gram:** also proposed by Mikolov et al. '13



Parametrization is chosen s.t. $p_\theta\left(x_i | x_t\right) \propto \exp\left(v_{x_i}, w_{x_t}\right)$

In practice, lots of other tricks are tacked on to deal with the slowest part of training: the softmax distribution (partition function sums over entire vocabulary).

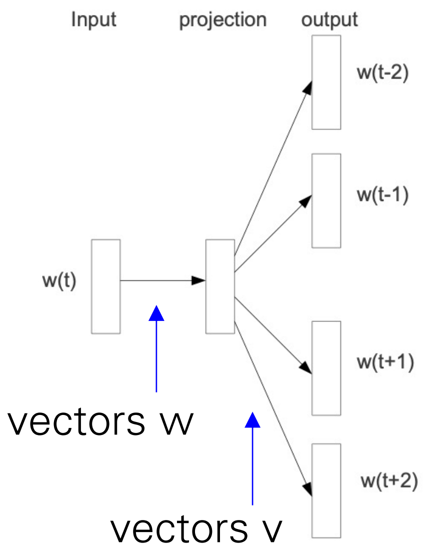
Common ones are negative sampling, hierarchical softmax, etc.

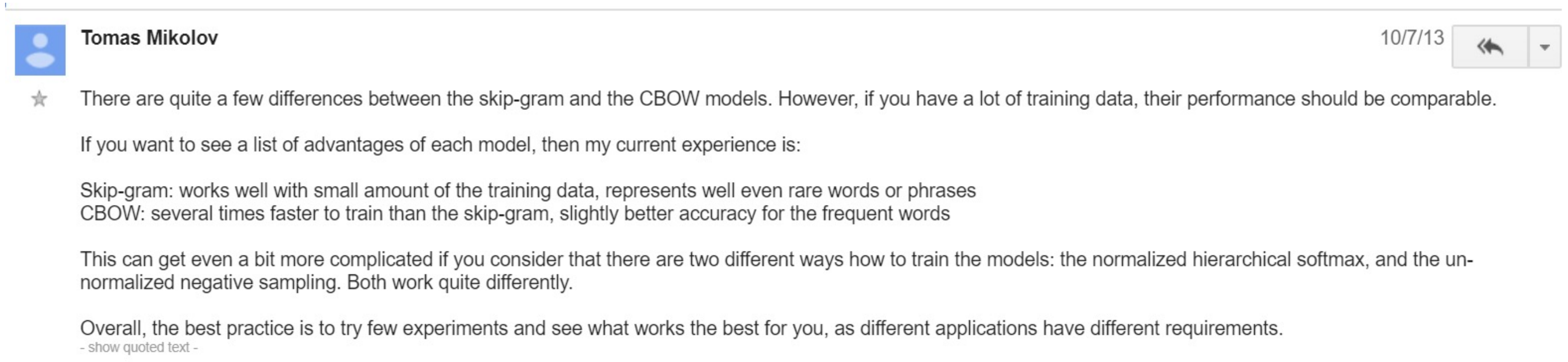
# Word Embeddings via Predictive Learning

- **Related:** predict <u>middle</u> word in a sentence, given <u>surrounding</u> ones.

$$\max_\theta \sum_t \sum_{i=t-L, i\neq t}^{t+L} \log p_\theta \left(x_i | x_t\right)$$

**Skip-Gram:** also proposed by Mikolov et al. '13




Tomas Mikolov 10/7/13

There are quite a few differences between the skip-gram and the CBOW models. However, if you have a lot of training data, their performance should be comparable.

If you want to see a list of advantages of each model, then my current experience is:

Skip-gram: works well with small amount of the training data, represents well even rare words or phrases
CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words

This can get even a bit more complicated if you consider that there are two different ways how to train the models: the normalized hierarchical softmax, and the un-normalized negative sampling. Both work quite differently.

Overall, the best practice is to try few experiments and see what works the best for you, as different applications have different requirements.
- show quoted text -

# Evaluating Word Embeddings

- First variant (predict next word, given previous ones) can be used as a **generative model** for text. (Also called language model.) The other ones cannot.
- In former case, a natural measure is the cross-entropy

$$-\mathbb{E}_{x_1,x_2,\ldots,x_T} \log p_\theta \left(x_{\leq T}\right) = \mathbb{E}_{x_1,x_2,\ldots,x_T} \sum_t \log p_\theta \left(x_t | x_{<t}\right)$$

- For convenience, we often take exponential of this (called perplexity)
- If we do not have a generative model, we have to use indirect means.

# Evaluating Word Embeddings

- **Intrinsic tasks:** Test performance of word embeddings on tasks measuring their "semantic" properties. Examples include solving "which is the most similar word" queries, analogy queries (i.e. "man is to woman as king is to ??"
- **Extrinsic tasks:** How well can we "finetune" the word embeddings to solve some (supervised) downstream task. "Finetune" usually means train a (relatively small) feedforward network. Examples of such tasks include:
  - Part-of-Speech Tagging (determine whether a word is noun/verb/...),
  - Named Entity Recognition (recognizing named entities like persons, places) – e.g. label a sentence as Picasso[person] died in France[country], many others.

## Semantic Similarity

- **Observation**: similar words tend to have larger (renormalized) inner products (also called cosine similarity).
- Precisely, if we look at the word embeddings for words i,j
  $\left\langle \frac{w_i}{||w_i||}, \frac{w_j}{||w_j||} \right\rangle = \cos(w_i, w_j)$ tends to be larger for similar words i,j

  Example: the nearest neighbors to "Frog" look like

0. frog
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

3. litoria

4. leptodactylidae

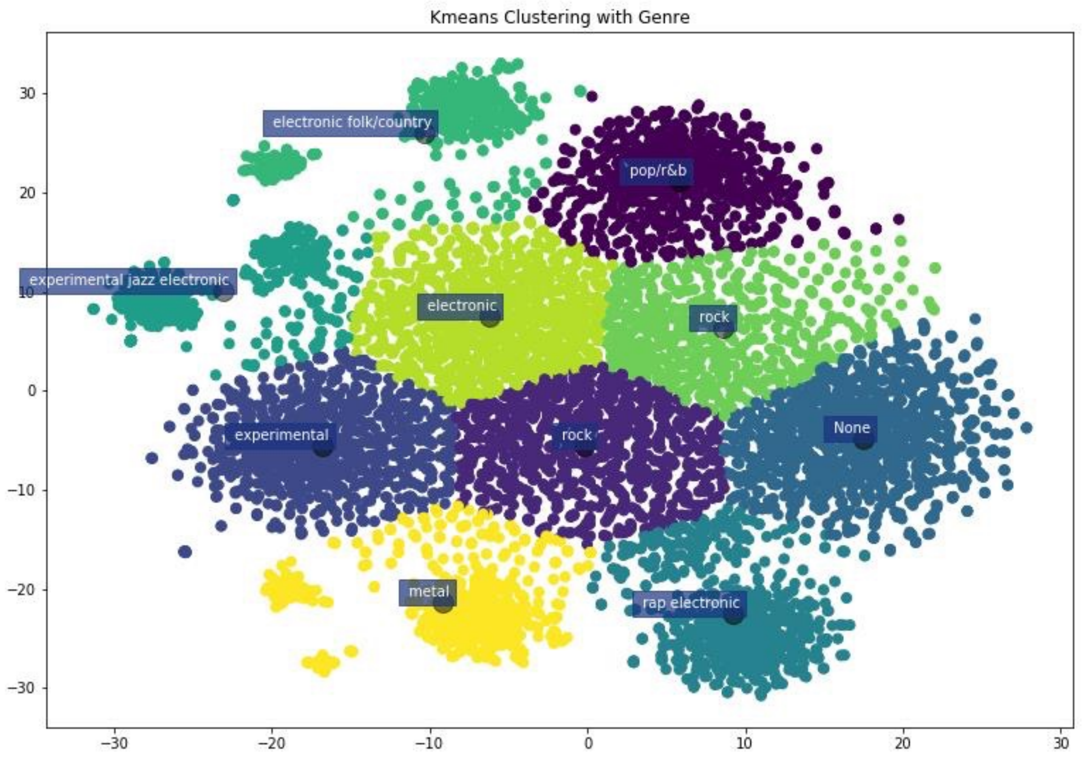
5. rana

7. eleutherodactylus

- To solve semantic similarity query like "which is the most similar word to", output the word with the highest cosine similarity.

# Semantic Clustering

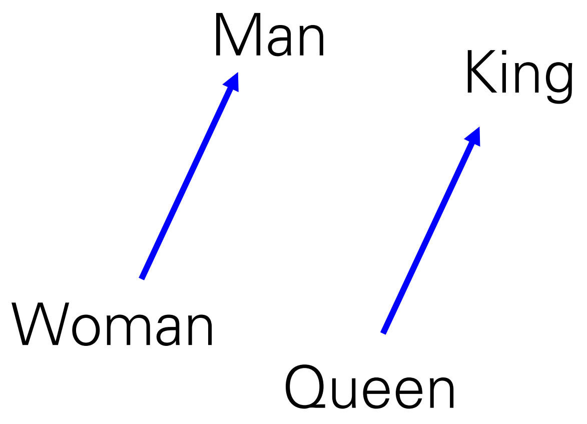
- Consequence: clustering word embeddings should give “semantically” relevant clusters.



t-SNE projection of word embeddings for artists (clustered by genre). Image from https://medium.com/free-code-camp/learn-tensorflow-the- word2vec-model-and-the-tsne-algorithm-using-rock-bands-97c99b5dcb3a

# Analogies

- **Observation**: You can solve analogy queries by linear algebra.



Precisely, $w$ = queen will be the solution to:

$$\operatorname{argmin}_w \|v_w - v_{\text{king}} - (v_{\text{woman}} - v_{\text{man}})\|^2$$

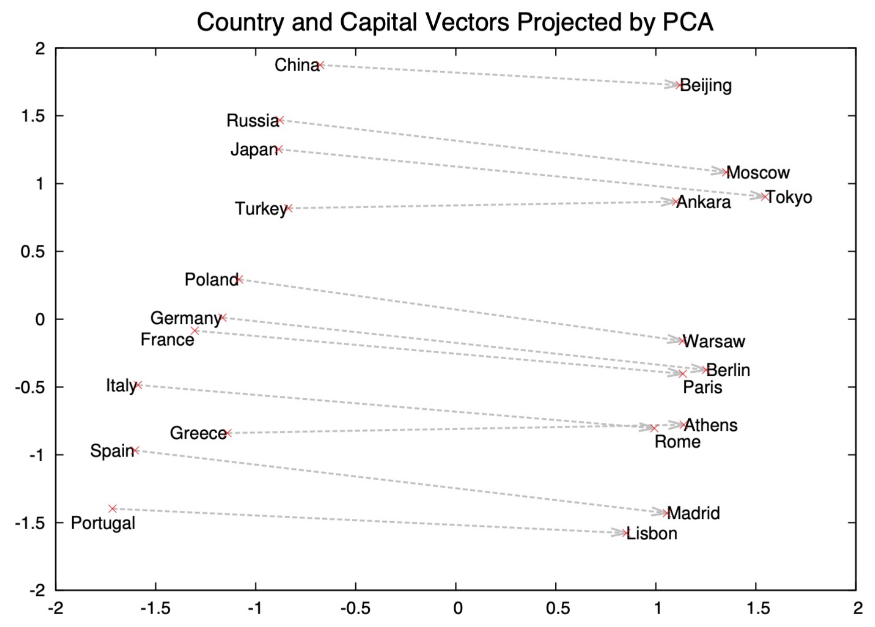


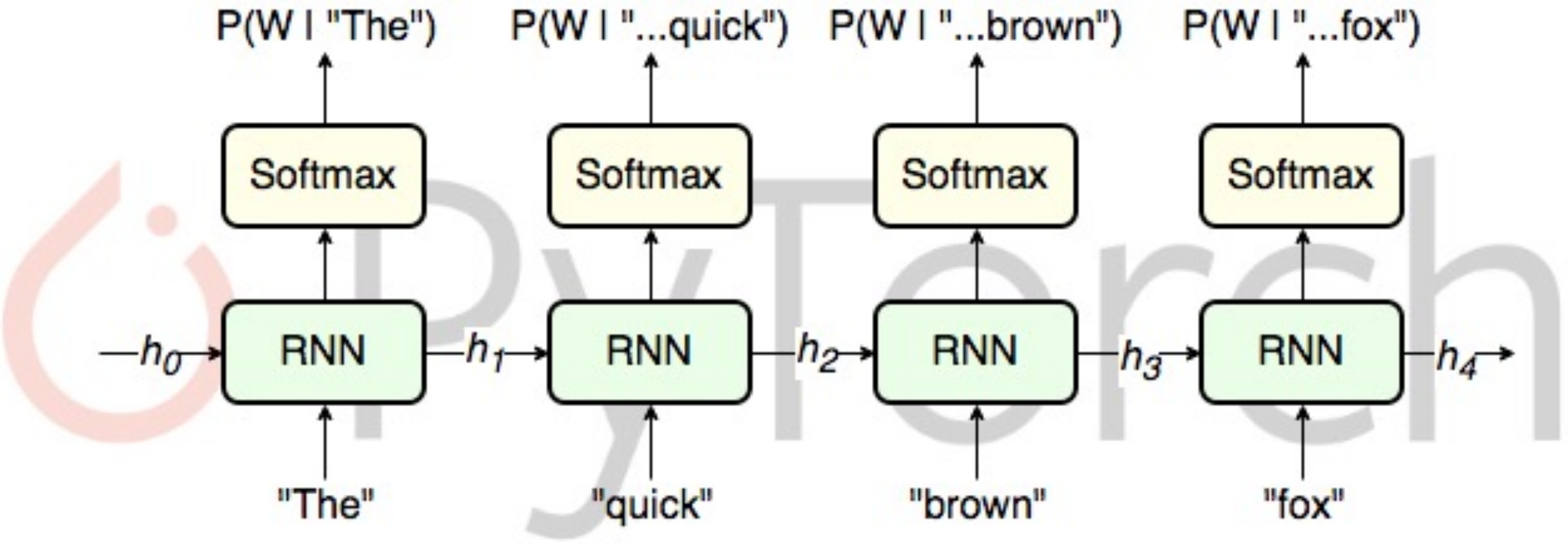
Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

## Language Models (LMs)

- A statistical model that assigns probabilities to the words in a sentences.
- **Most commonly:** Given previous words, what should the next one be?
- **Neural language model:** Model the probability of words given others using neural networks.

# Recurrent Architectures for LM

- We can use recurrent architectures.
- LSTM, GRU ...
- Great for variable length inputs, like sentences.



# Recurrent Architectures for LM

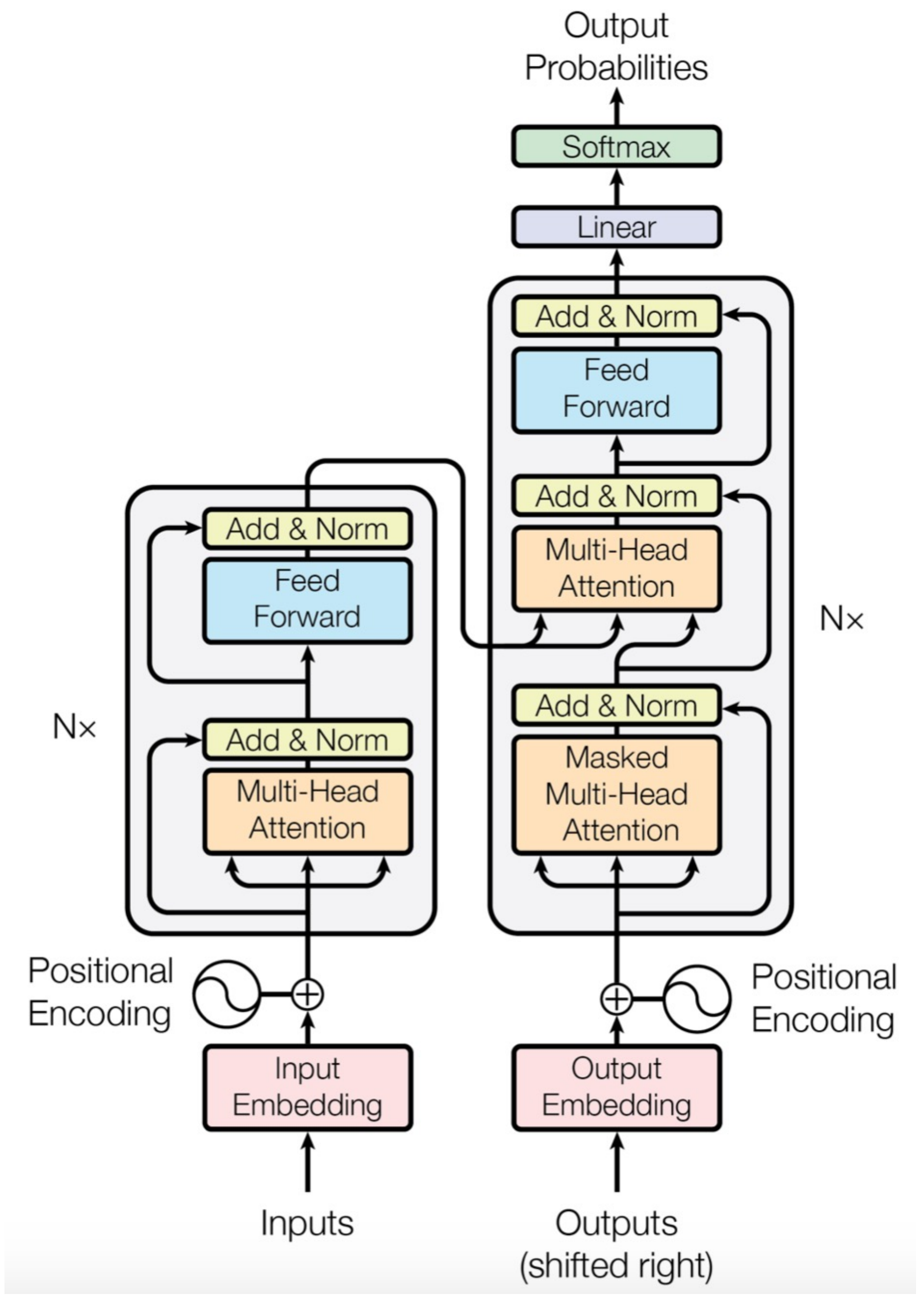
- What are some of the problems with recurrent architectures?
  - Not parallelizable across instances.
  - Cannot model long dependences.
  - Optimization difficulties (vanishing gradients).

- Attention to the rescue!

# Transformers

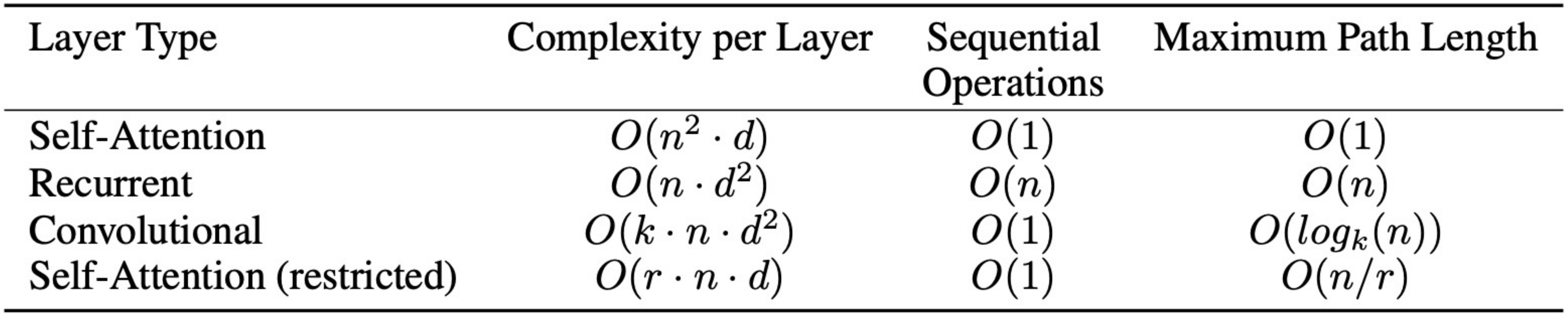
Properties of the transformer architecture:

- Fully feed forward.
- Equivariance properties of scaled dot product attention (important):
  - How does the output change if we permute the order of queries? (equivariance)
  - How does the output change if we permute the key-value pairs in unison? (invariance)



# Performance Comparison

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. $n$ is the sequence length, $d$ is the representation dimension, $k$ is the kernel size of convolutions and $r$ the size of the neighborhood in restricted self-attention.



| Layer Type | Complexity per Layer | Sequential Operations | Maximum Path Length |
|---|---|---|---|
| Self-Attention | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| Recurrent | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| Convolutional | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(log_k(n))$ |
| Self-Attention (restricted) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

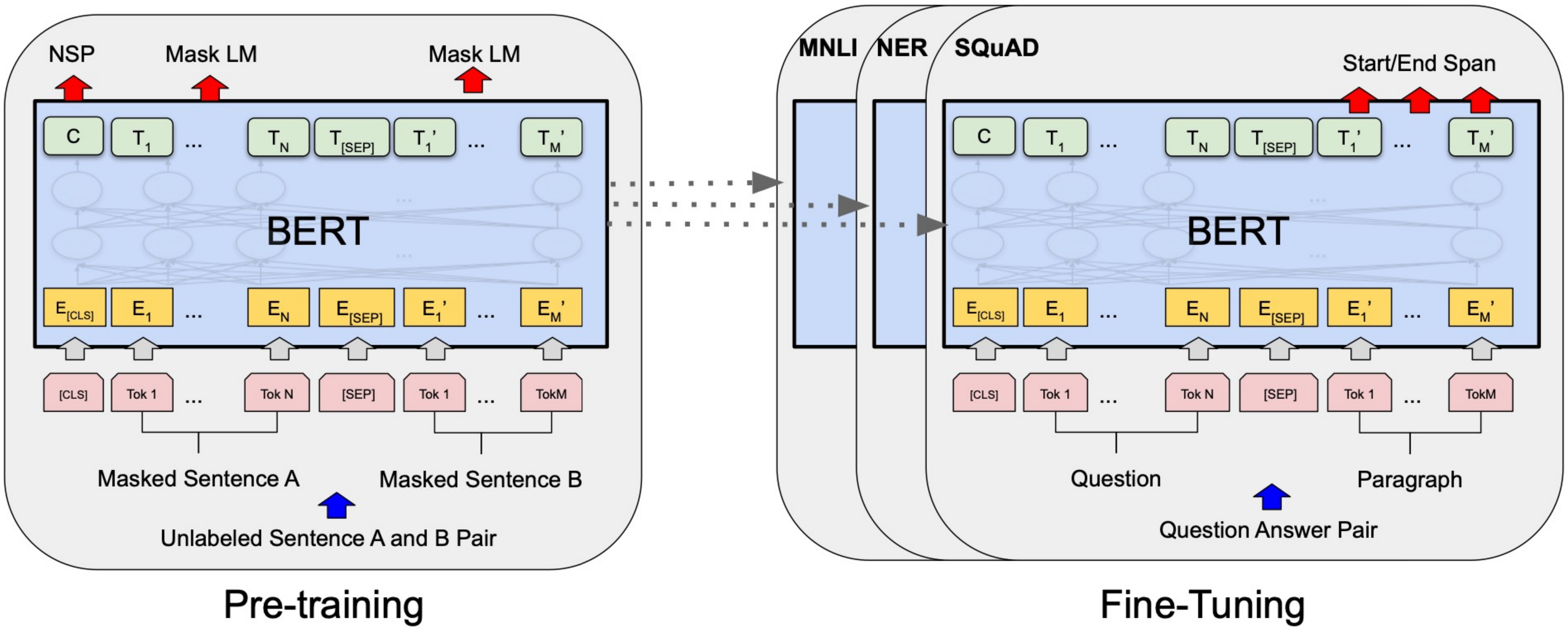
## Pretraining Language Models

- Can we use large amounts of text data to pretrain language models?
- Considerations:
  - ▶ How can we fuse both left-right and right-left context?
  - ▶ How can we facilitate non-trivial interactions between input tokens?
- Previous approaches:
  - ▶ ELMO (Peters. et. al., 2017): Bidirectional, but shallow.
  - ▶ GPT (Radford et. al., 2018): Deep, but unidirectional.
  - ▶ BERT (Devlin et. al., 2018): Deep and bidirectional!

## BERT Workflow

- The BERT workflow includes:
  - ► Pretrain on generic, self-supervised tasks, using large amounts of data (like all of Wikipedia)
  - ► Fine-tune on specific tasks with limited, labelled data.
- The pretraining tasks (will talk about this in more detail later):
  - ► Masked Language Modelling (to learn contextualized token representations)
  - ► Next Sentence Prediction (summary vector for the whole input)

# BERT Architecture



## BERT Architecture

**Properties:**

- Two input sequences.
  - ► Many NLP tasks have two inputs (question answering, paraphrase detection, entailment detection etc. )
- Computes embeddings
  - ► Both token, position and segment embeddings.
  - ► Special start and separation tokens.
- Architecture
  - ► Basically the same as transformer encoder.
- Outputs:
  - ► Contextualized token representations.
  - ► Special tokens for context.

# BERT Embeddings

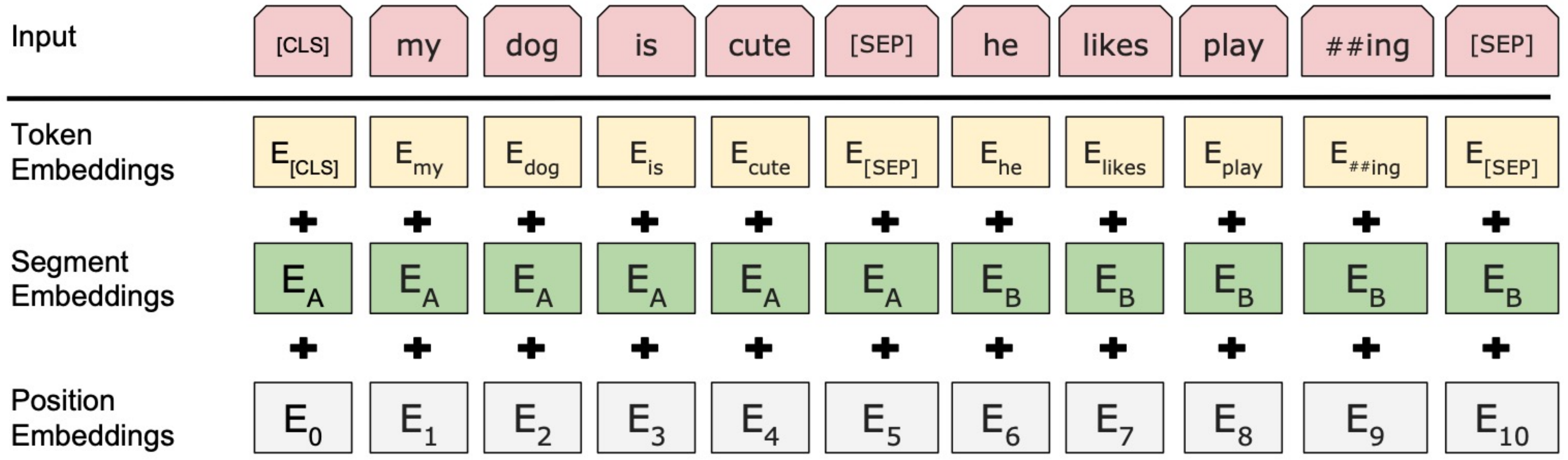

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ $E_{my}$ $E_{dog}$ $E_{is}$ $E_{cute}$ $E_{[SEP]}$ $E_{he}$ $E_{likes}$ $E_{play}$ $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
$E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_B$ $E_B$ $E_B$ $E_B$ $E_B$
Position Embeddings
$E_0$ $E_1$ $E_2$ $E_3$ $E_4$ $E_5$ $E_6$ $E_7$ $E_8$ $E_9$ $E_{10}$

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

- How we tokenize the inputs is very important!
- BERT uses the WordPiece tokenizer (Wu et. al. 2016)

## (Aside) Tokenizers

- Tokenizers have to balance the following:
  - Being comprehensive (rare words? translation to different languages)
  - Total number of tokens
  - How semantically meaningful each token is.
- This is an activate area of research.

## Pretraining tasks

- Masked Language Modelling, i.e. Cloze Task (Taylor, 1953)
- Next sentence prediction

## Masked Language Modelling

- Mask 15% of the input tokens. (i.e. replace with a dummy masking token)
- Run the model, obtain the embeddings for the masked tokens.
- Using these embeddings, try to predict the missing token.
- "I love to eat peanut ___ and jam. " Can you guess what's missing?

- **This procedure forces the model to encode context information in the features of all of the tokens.**

## Next Sentence Prediction

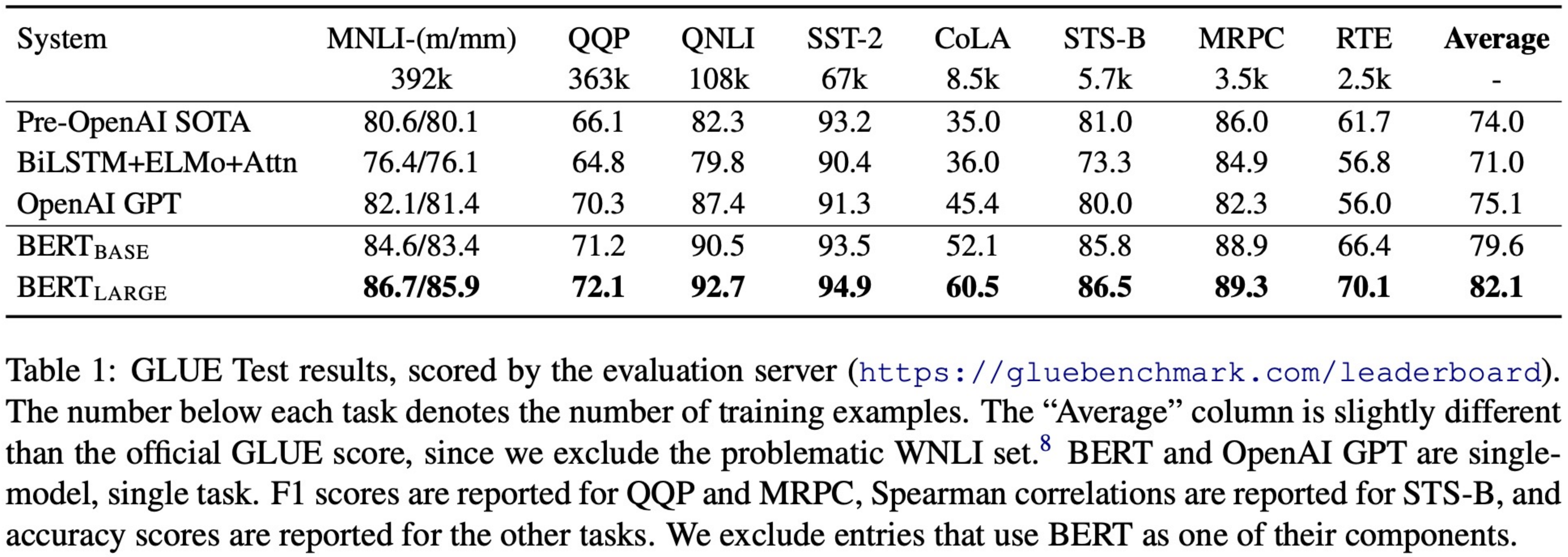
- Goal is to summarize the complete context (i.e. the two segments) in a single feature vector.
- Procedure for generating data
  - ► Pick a sentence from the training corpus and feed it as ”segment A”.
  - ► With 50% probability, pick the following sentence and feed that as ”segment B”.
  - ► With 50% probability, pick the a random sentence and feed it as ”segment B”.
- Using the features for the context token, predict whether segment B is the following sentence of segment A.
- Turns out to be a very effective pretraining technique!

# Fine Tuning

**Procedure**:

- Add a final layer on top of BERT representations.
- Train the whole network on the fine-tuning dataset.
- Pre-training time: In the order of days on TPUs.
- Fine tuning task: Takes only a few hours max.

## Fine Tuning



| System | MNLI-(m/mm) | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| | 392k | 363k | 108k | 67k | 8.5k | 5.7k | 3.5k | 2.5k | - |
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.8 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| BERT$_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| BERT$_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **92.7** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **82.1** |

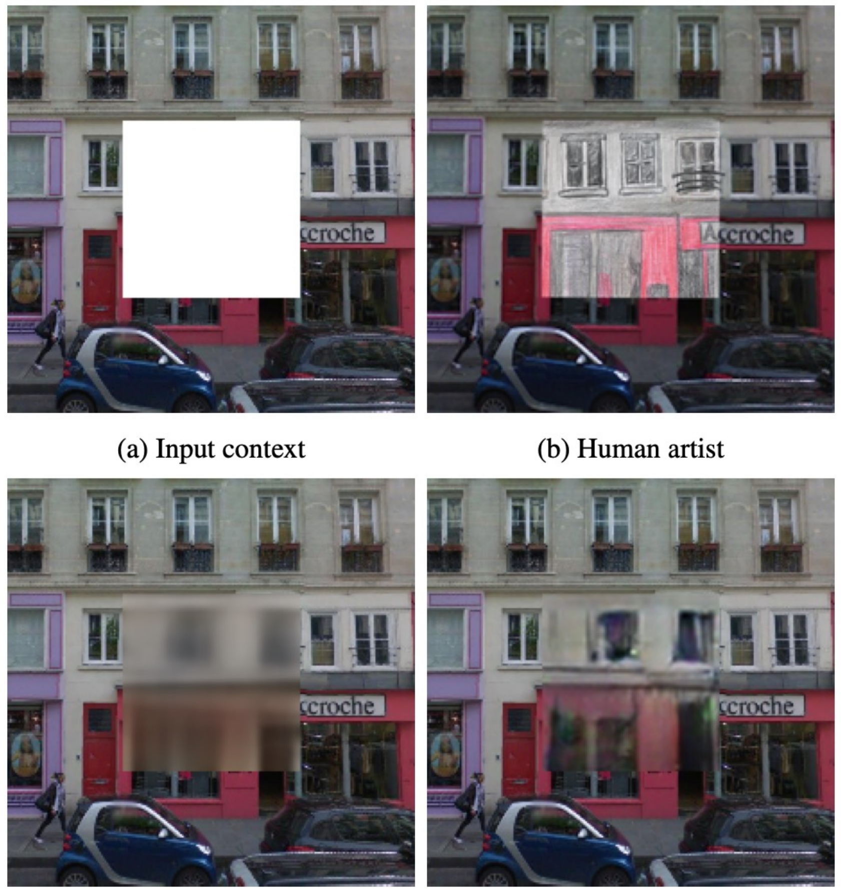
Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

## Self-Supervised Learning in Vision

# Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting



(a) Input context (b) Human artist

(c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

# Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

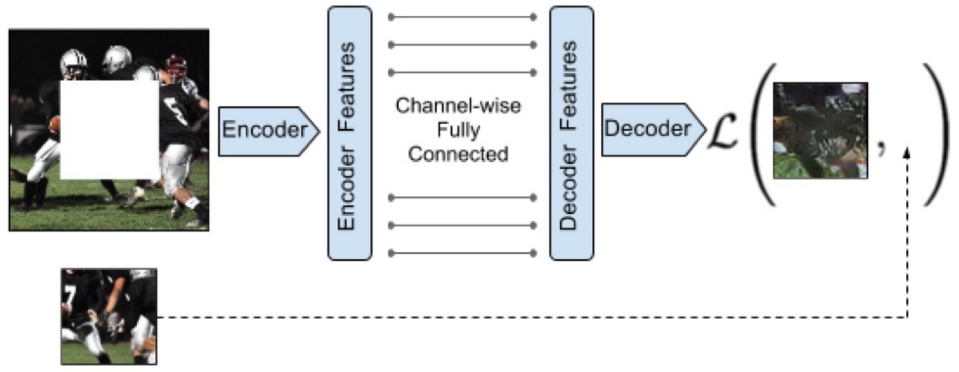


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

Architecture:
An encoder E takes a part of image, constructs a representation.

A decoder D takes representation, tries to reconstruct missing part.

- **Much** trickier than in NLP:
  As we have seen, meaningful losses for vision are much more difficult to design.
  Choice of region to mask out is much more impactful.

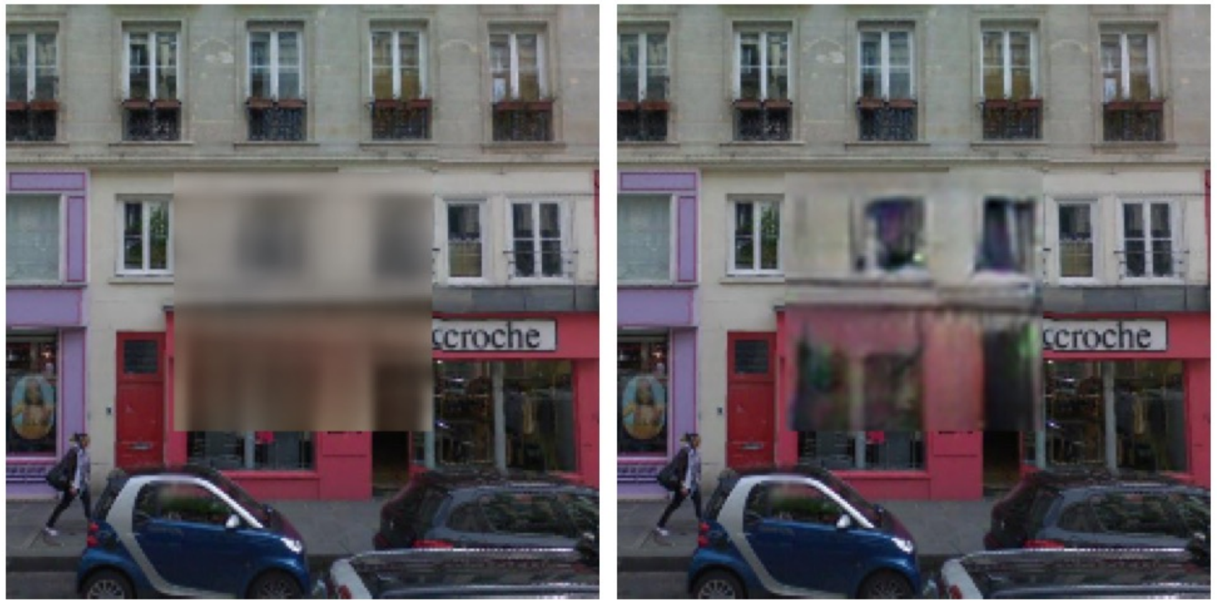
# Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

- If reconstruction loss is $L_2$: tendency to produce blurry images.

  Remember: one of the usefulness of GANs is to provide a better loss for images.



(c) Context Encoder ($L2$ loss)    (d) Context Encoder ($L2$ + Adversarial loss)

# Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. ’16: Context Encoders: Feature Learning by Inpainting

- If reconstruction loss is $L_2$: tendency to produce blurry images.
- Remember: one of the usefulness of GANs is to provide a better loss for images.

Composition of encoder+decoder

Mask

$$\mathcal{L}_{rec}(x) = \|\hat{M} \odot (x - F((1-\hat{M}) \odot x))\|_2^2,$$

DC-GAN objective

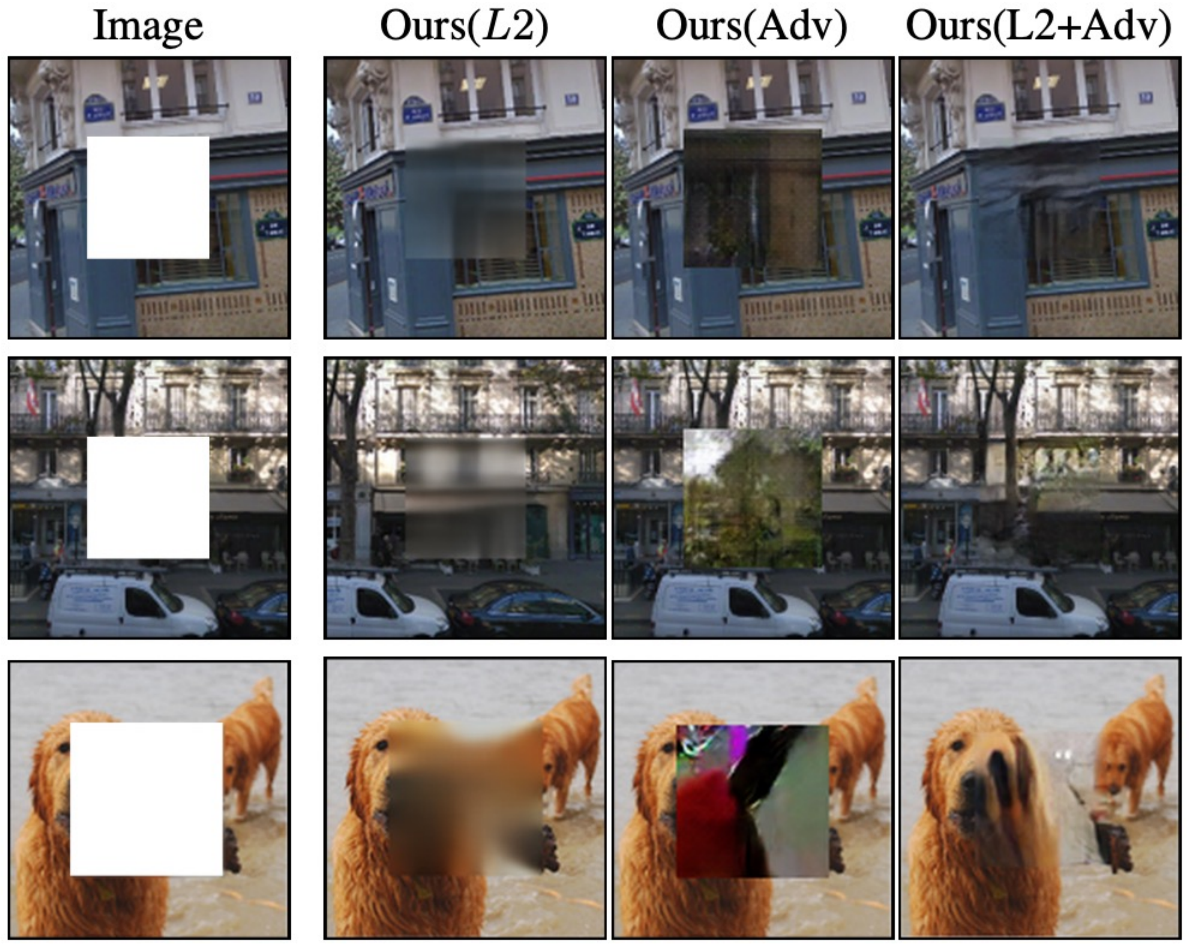
$$\begin{aligned}\mathcal{L}_{adv} = \max_{D} \quad & \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) \\ & + \log(1 - D(F((1-\hat{M}) \odot x)))],\end{aligned}$$

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}.$$

# Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. '16: Context Encoders: Feature Learning by Inpainting



# Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

- How to choose the region?

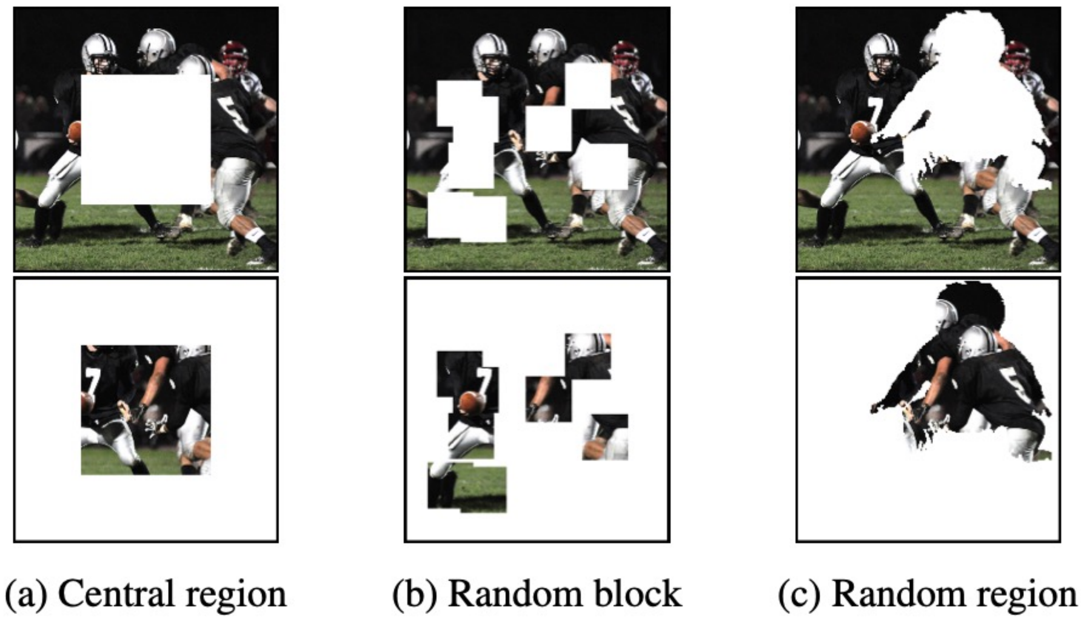


Figure 3: An example of image $x$ with our different region masks $\hat{M}$ applied, as described in Section 3.3.

Task should be "solvable", but not "too easy".

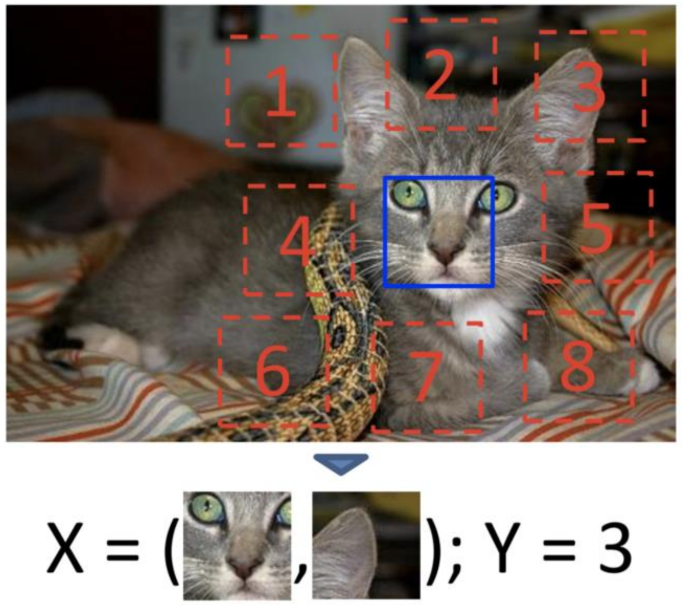
- Fixed (central region): tends to produce less generalizeable representations
- Random blocks: slightly better, but square borders still hurt.
- Random silhouette (fully random doesn't make sense – prediction task is too ill-defined) – even better!

# Jigsaw puzzles

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Doersch et al. '15: Unsupervised Visual Representation Learning by Context Prediction

- **Task**: Predict ordering of two randomly chosen pieces from the image.



**Representation**: penultimate layer of a neural net used to solve task.

**Intuition**: understanding relative positioning of pieces of an image requires some understanding of how images are composed.

# Jigsaw puzzles

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Doersch et al. '15: Unsupervised Visual Representation Learning by Context Prediction

- **Quite finnicky**: one needs to make sure the predictor cannot take any obvious "shortcuts".

  - Boundary texture continuity is a big clue: include gaps in tiles.
  - Long lines spanning tiles are a clue: jitter location of tiles.
  - Chromatic aberration (some cameras tend to focus different wavelengths at different position – e.g. green shifts towards center of image): randomly drop 2 of the 3 channels

# Predicting rotations

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

- **Task**: predict one of 4 possible rotations of an image.

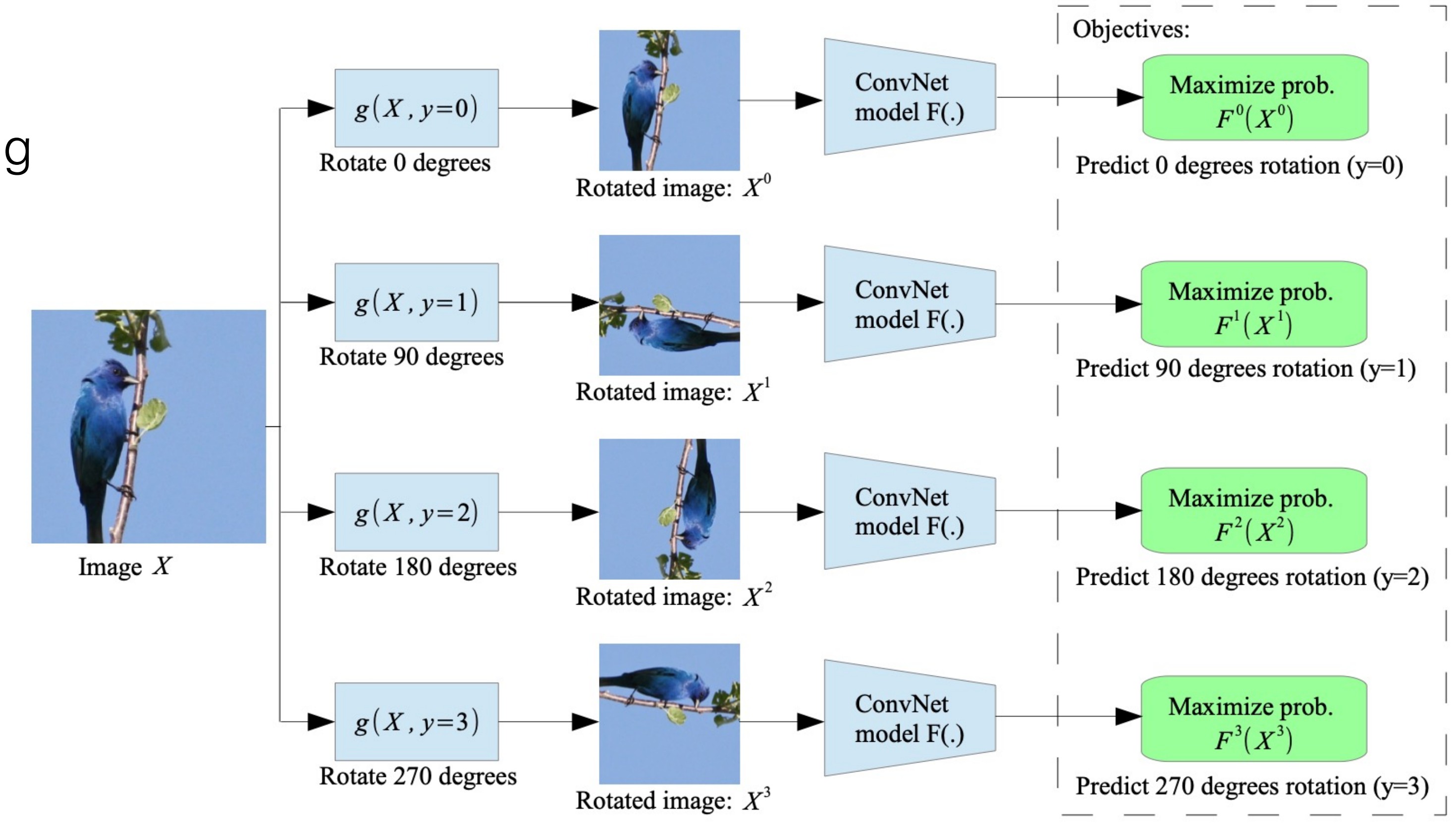


Image $X$
$g(X, y=0)$
Rotate 0 degrees
Rotated image: $X^0$
$g(X, y=1)$
Rotate 90 degrees
Rotated image: $X^1$
$g(X, y=2)$
Rotate 180 degrees
Rotated image: $X^2$
$g(X, y=3)$
Rotate 270 degrees
Rotated image: $X^3$
ConvNet model F(.)
Objectives:
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation $y$ predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation $y^*$.

# Predicting rotations

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

- **Task**: predict one of 4 possible rotations of an image.
  - **Representation**: penultimate layer of a neural net used to solve task.
  - **Intuition**: a rotation is a global transformation. ConvNets are much better at capturing local transformations (as convolutions are local), so there is no obvious way to "cheat".

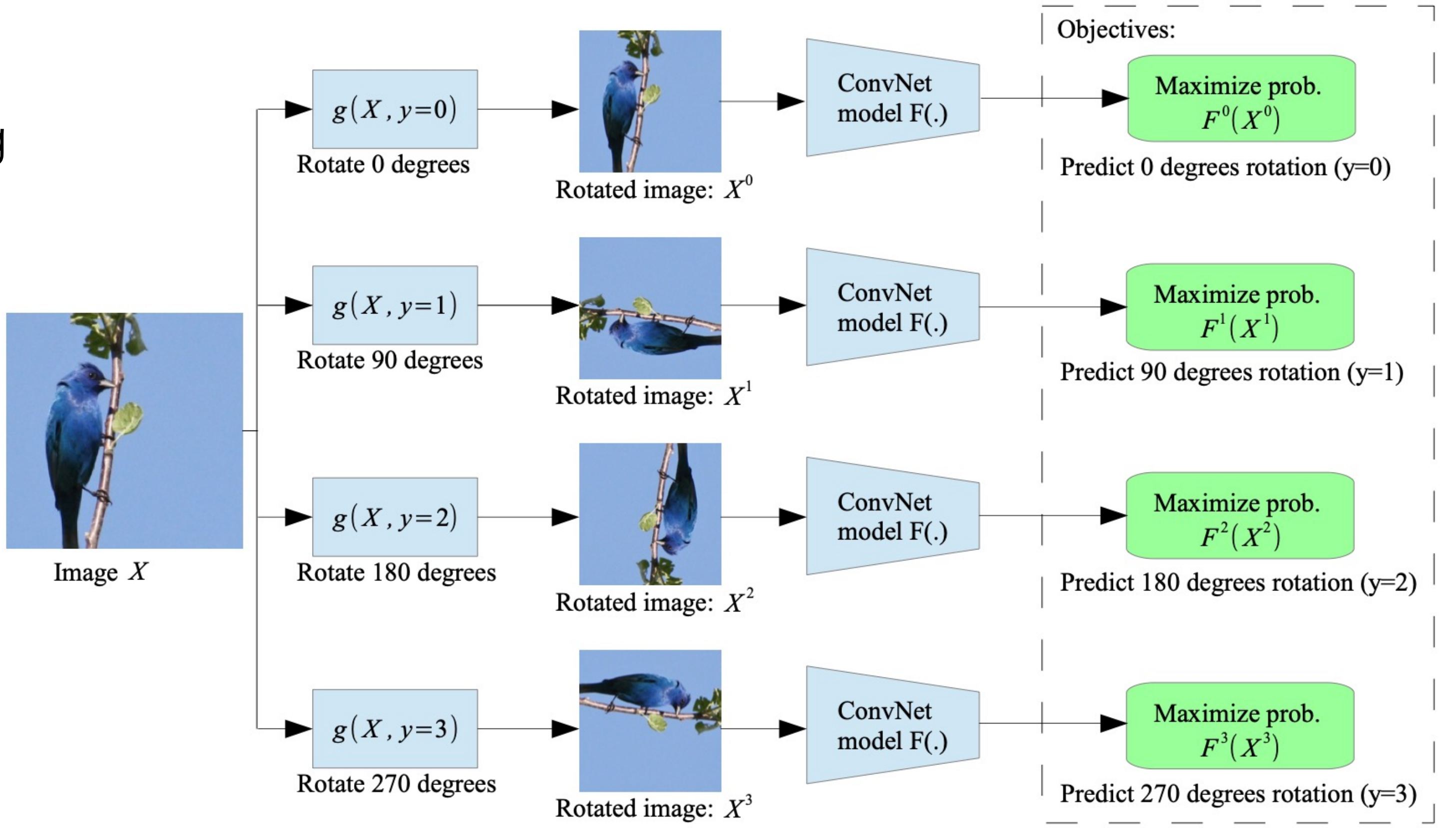


Image $X$
$g(X, y=0)$
Rotate 0 degrees
$g(X, y=1)$
Rotate 90 degrees
$g(X, y=2)$
Rotate 180 degrees
$g(X, y=3)$
Rotate 270 degrees
Rotated image: $X^0$
Rotated image: $X^1$
Rotated image: $X^2$
Rotated image: $X^3$
ConvNet model F(.)
Objectives:
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation $y$ predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation $y^*$.

# Predicting rotations

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

- **Task**: predict one of 4 possible rotations of an image.
  - Less finicky to get right: no obvious artifacts the model can make use of to cheat.
  - The 90 deg. rotations also don't introduce any additional artifacts due to discretization.

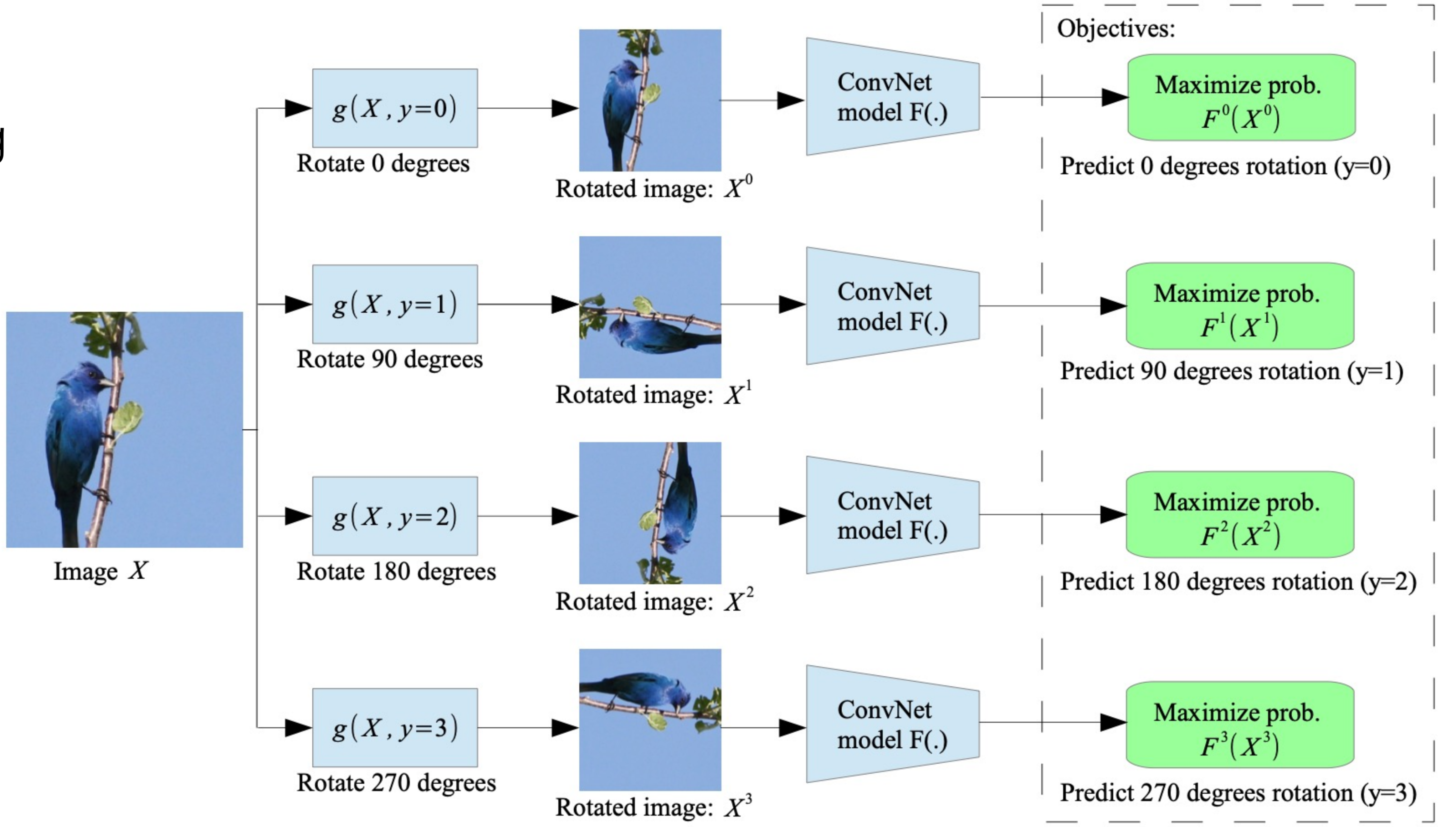
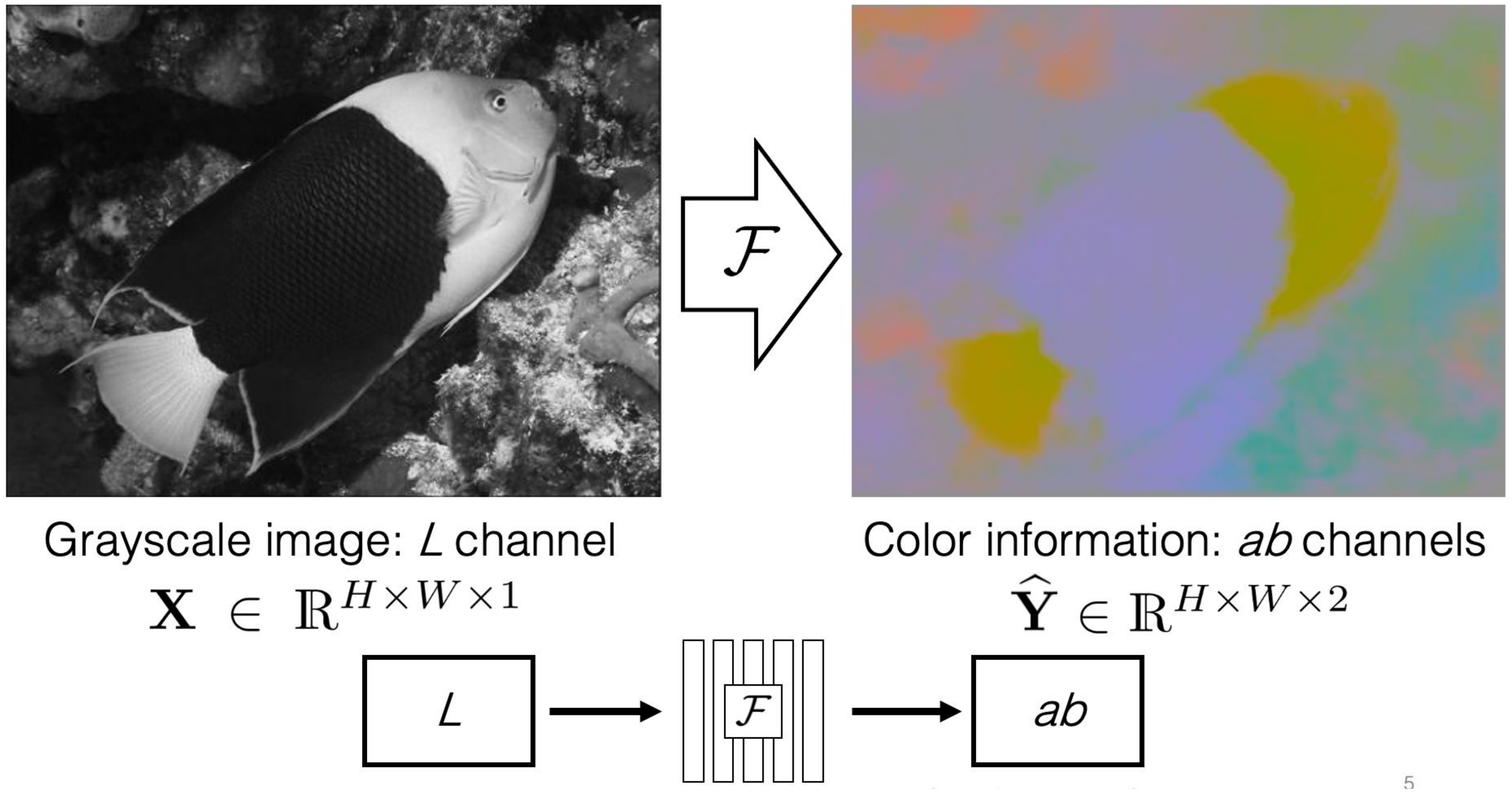


Image $X$
$g(X, y=0)$
Rotate 0 degrees
Rotated image: $X^0$
$g(X, y=1)$
Rotate 90 degrees
Rotated image: $X^1$
$g(X, y=2)$
Rotate 180 degrees
Rotated image: $X^2$
$g(X, y=3)$
Rotate 270 degrees
Rotated image: $X^3$
ConvNet model F(.)
Objectives:
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

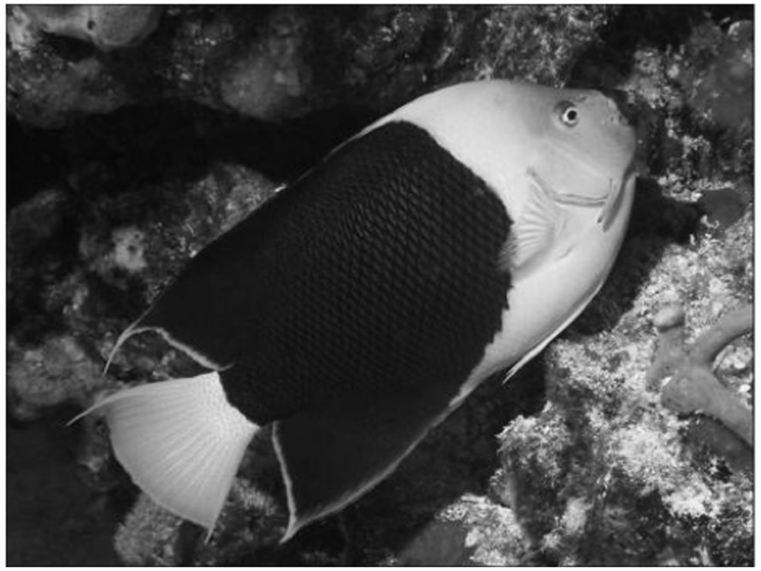
Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation $y$ predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation $y^*$.

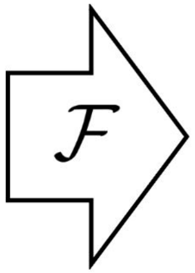
# Image coloring

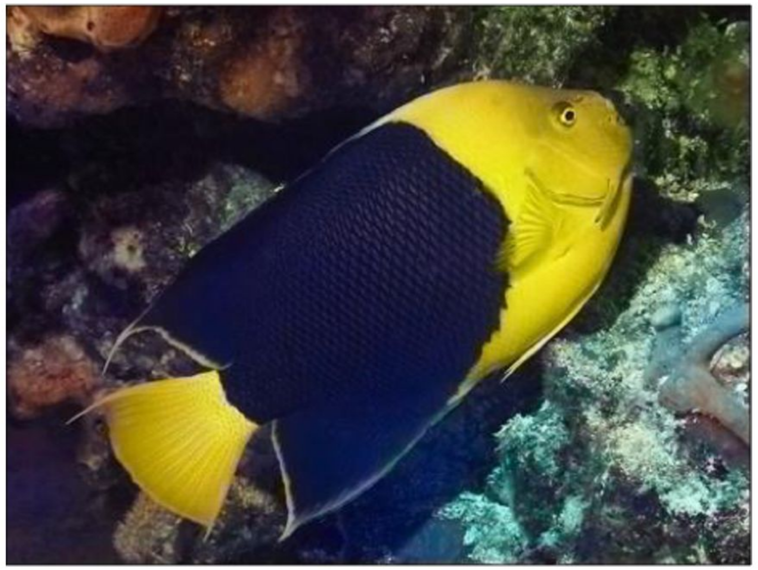

$\mathcal{F}$
Grayscale image: *L* channel
$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$
Color information: *ab* channels
$\widehat{\mathbf{Y}} \in \mathbb{R}^{H \times W \times 2}$
*L*
$\mathcal{F}$
*ab*
5

Source: Richard Zhang / Philip Isola

# Image coloring





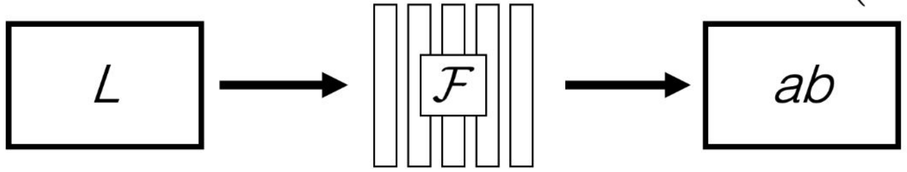


Grayscale image: *L* channel

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

Concatenate (*L*,*ab*) channels

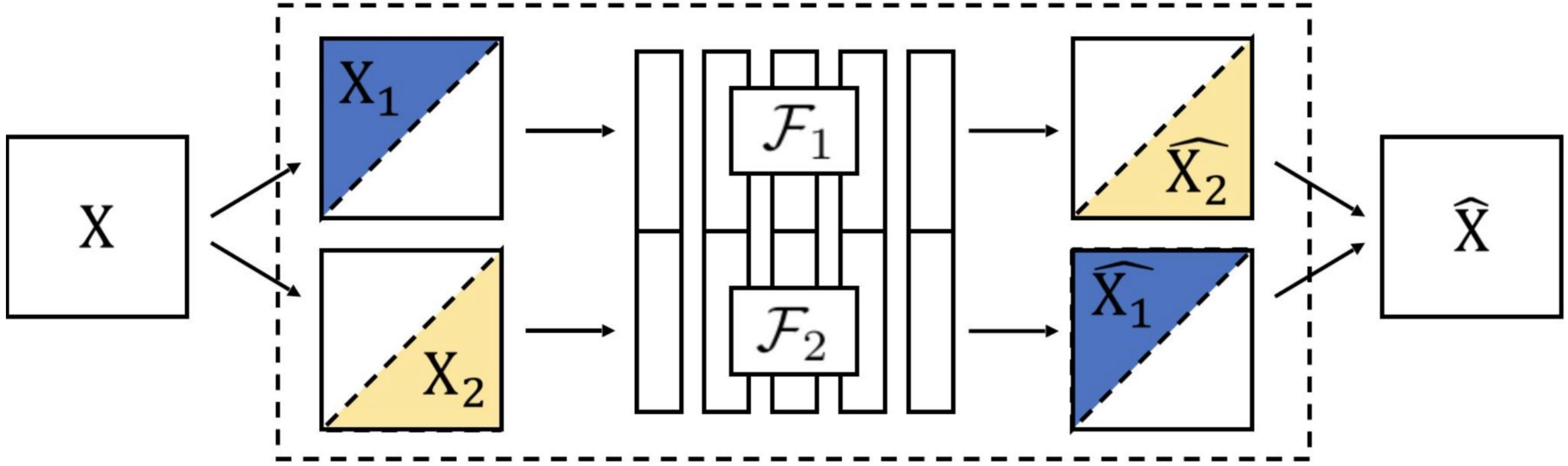
$(\mathbf{X}, \widehat{\mathbf{Y}})$



Source: Richard Zhang / Philip Isola

# Learning features from colorization: Split-brain Autoencoder

- **Idea:** cross-channel predictions

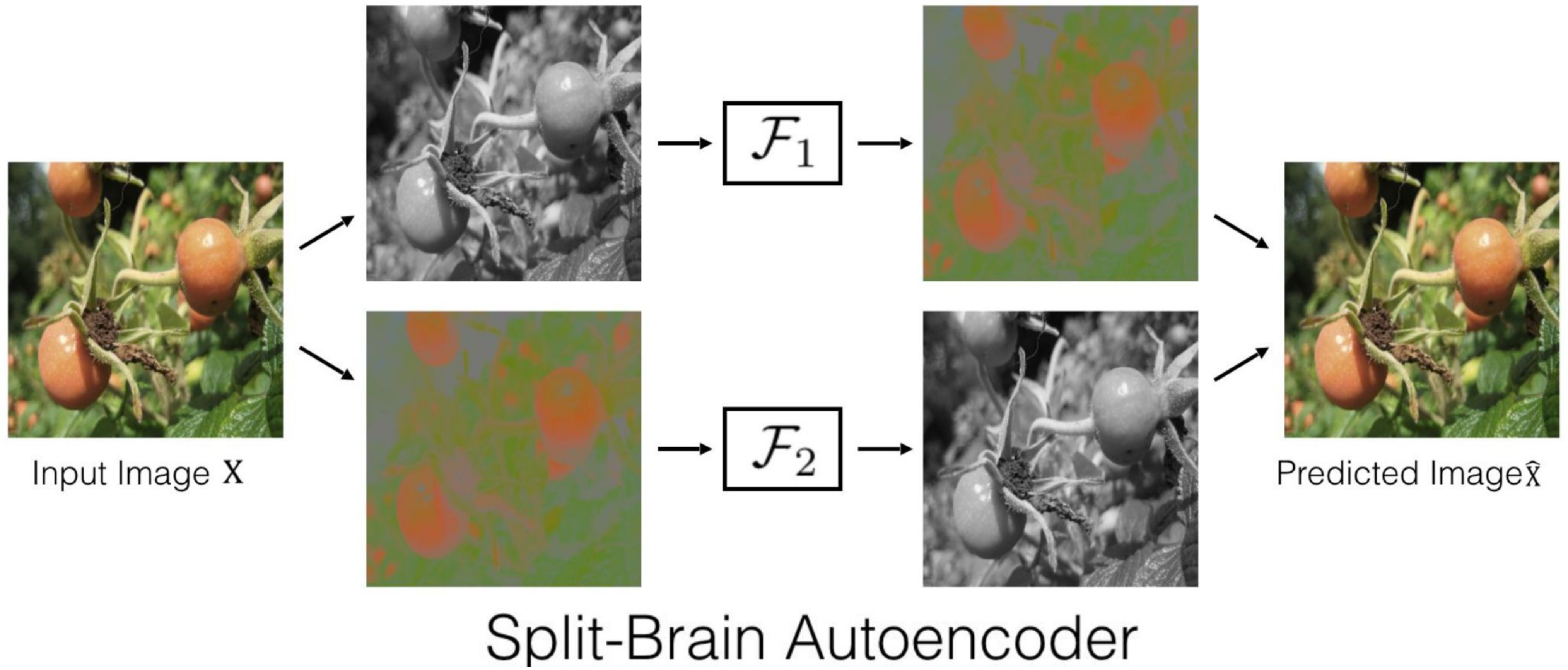


Split-Brain Autoencoder

Source: Richard Zhang / Philip Isola

# Learning features from colorization: Split-brain Autoencoder

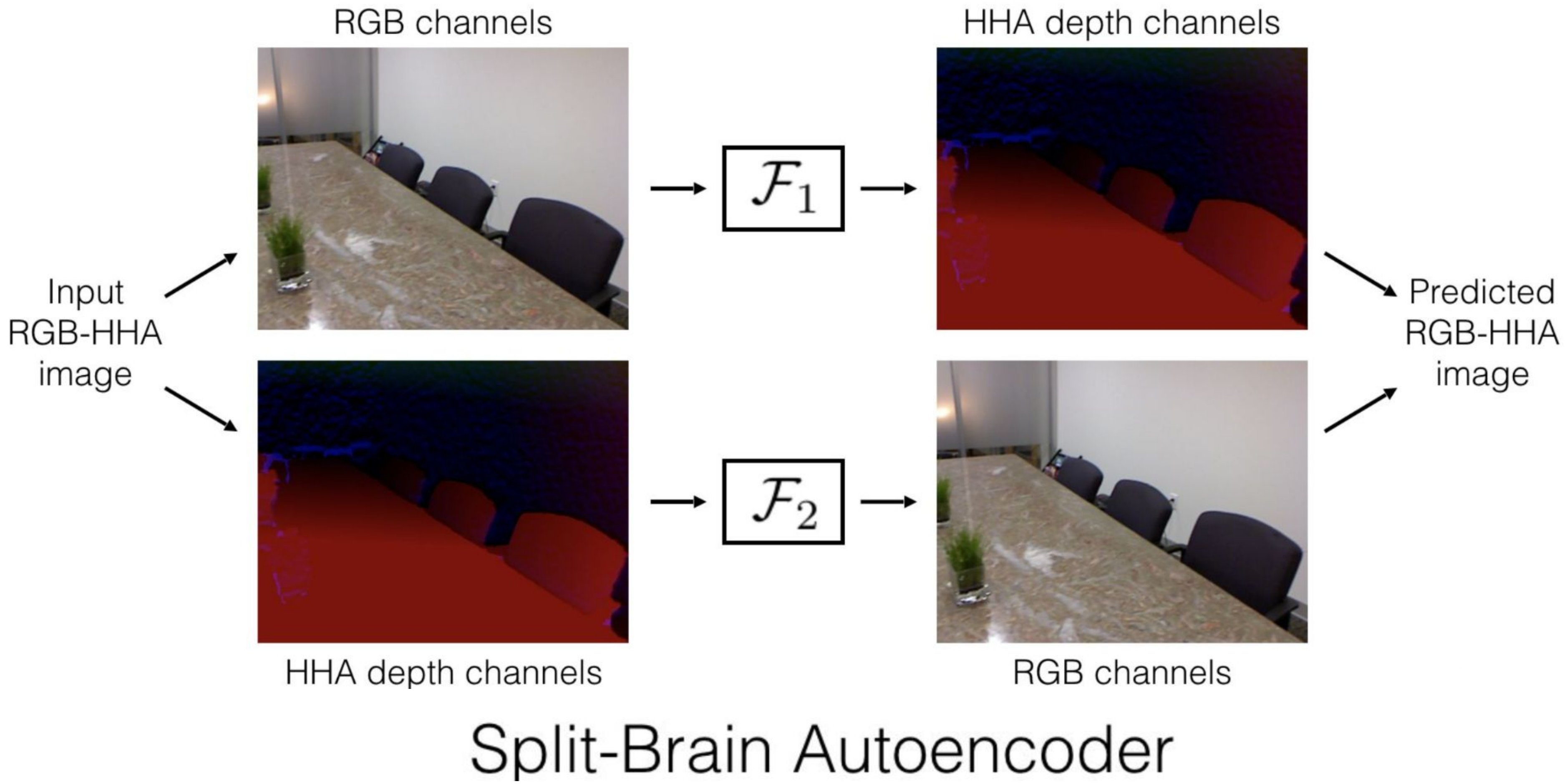
- **Idea:** cross-channel predictions



Source: Richard Zhang / Philip Isola

# Learning features from colorization: Split-brain Autoencoder

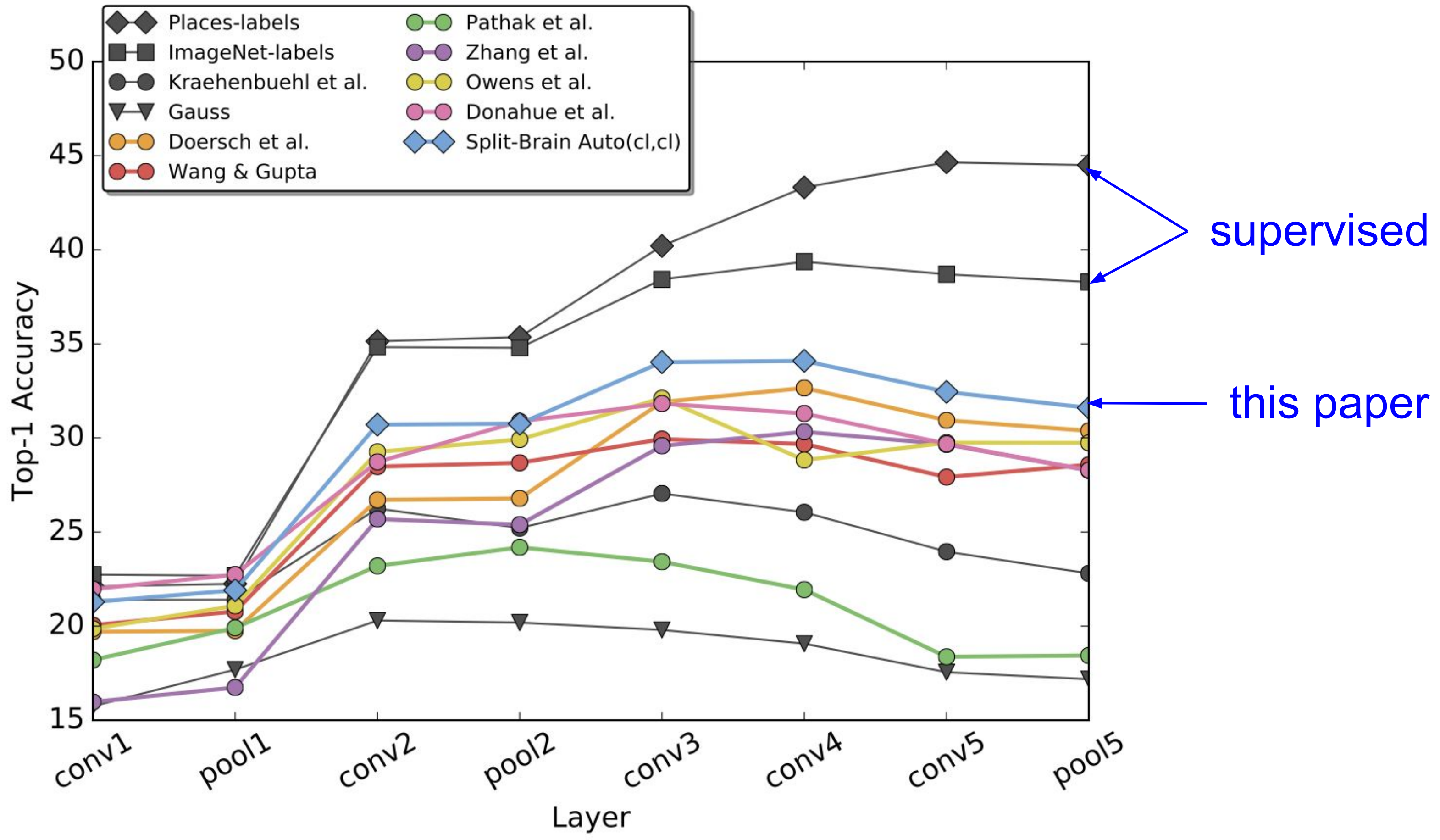
- **Idea:** cross-channel predictions



Source: Richard Zhang / Philip Isola

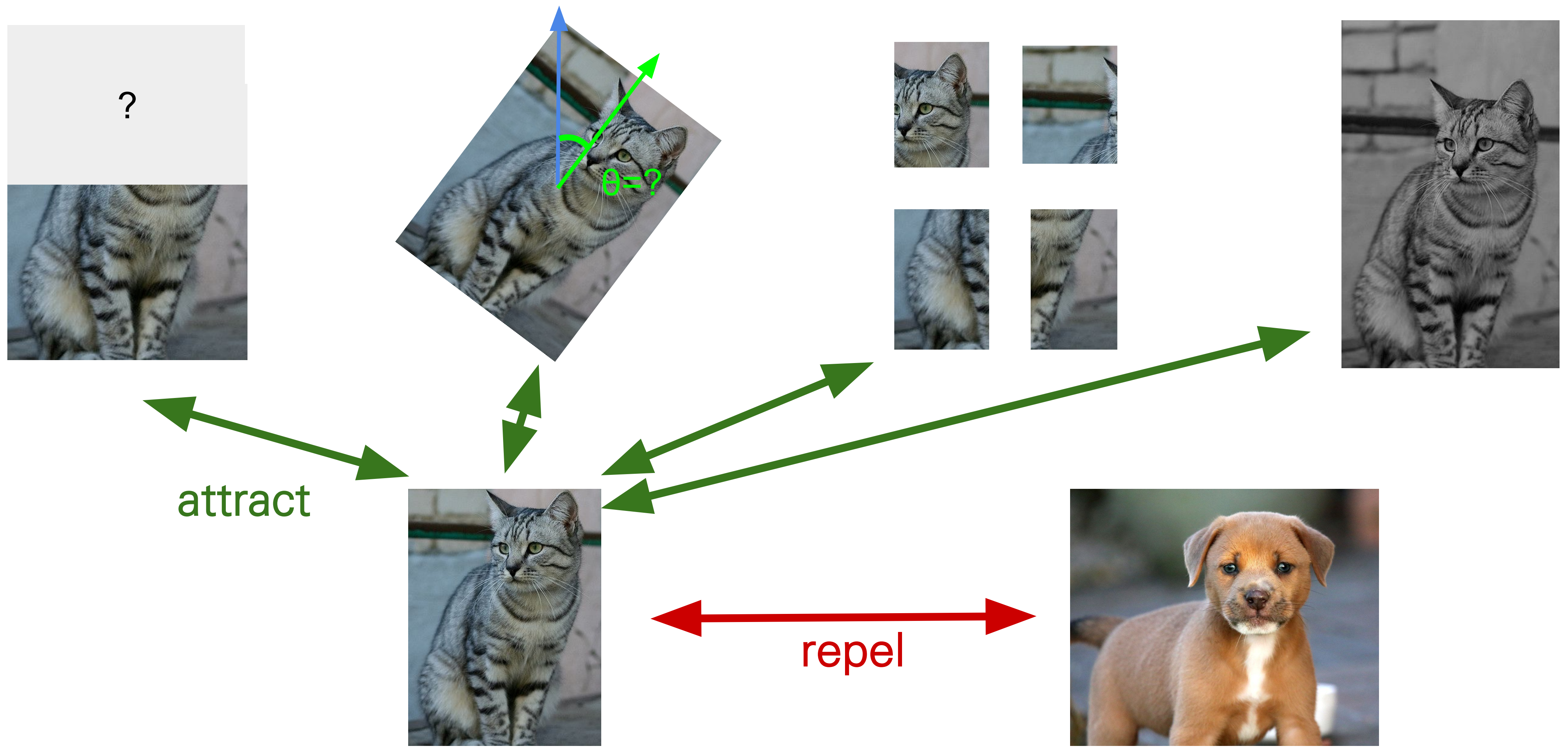
# Split-brain Autoencoder: Transfer learned features to supervised learning

- 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).

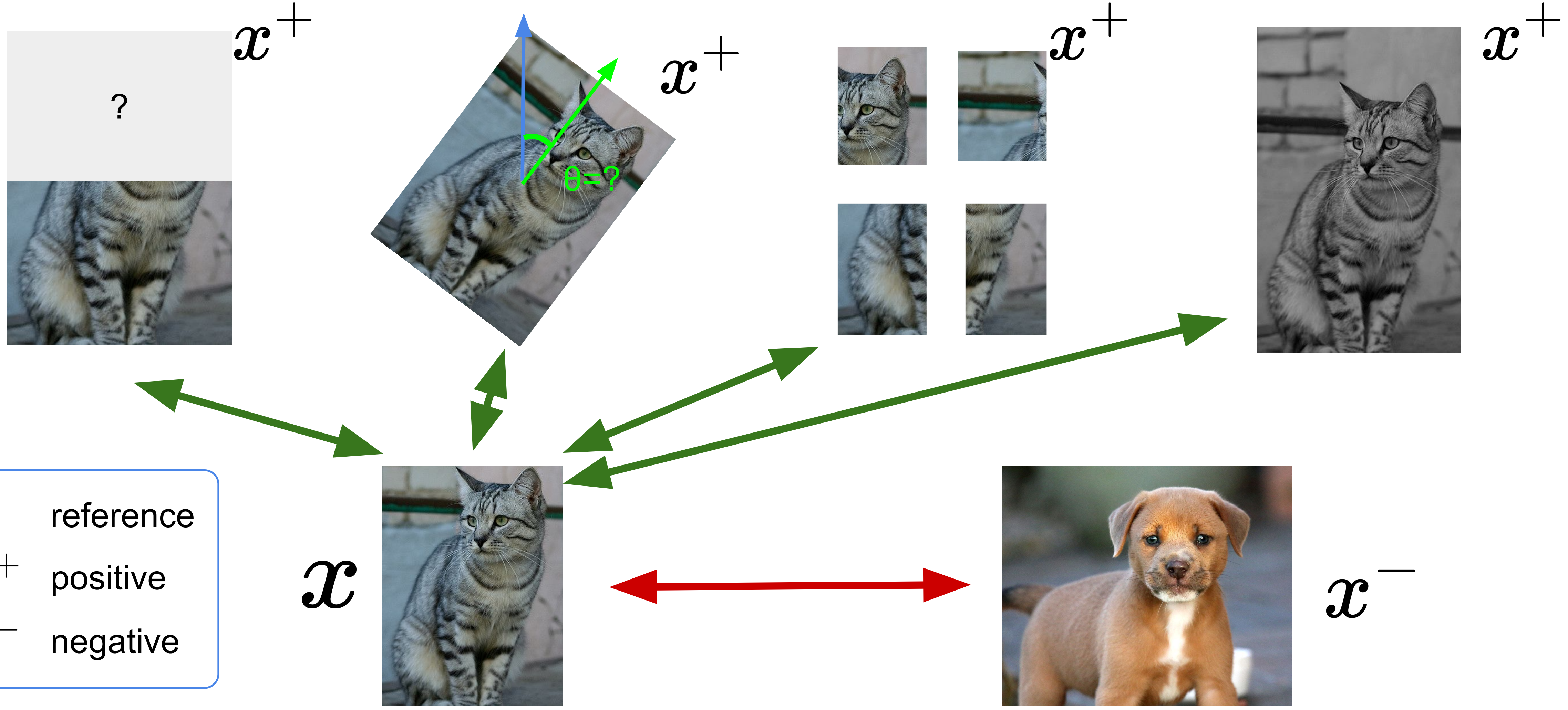


Source: Zhang et al., 2017

# Contrastive Representation Learning



# 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

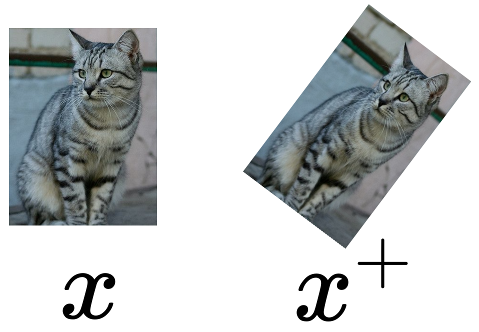
- Loss function given 1 positive sample and N - 1 negative samples:

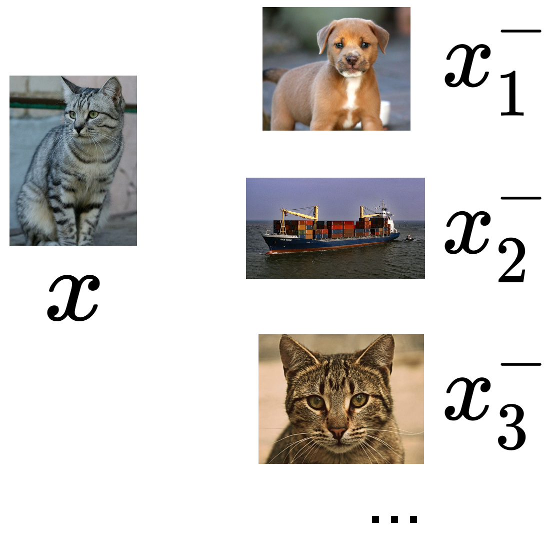
$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-))} \right]$$

# A formulation of contrastive learning

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

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





# A formulation of contrastive learning

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

$$L = -\mathbb{E}_X\left[\log \frac{\overline{\exp(s(f(x), f(x^+)))}}{\underbrace{\exp(s(f(x), f(x^+)))}_{\text{score for the positive pair}} + \underbrace{\sum_{j=1}^{N-1}\exp(s(f(x), f(x_j^-)))}_{\text{score for the N-1 negative pairs}}}\right]$$

- This seems familiar..

## A formulation of contrastive learning

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

$$L = -\mathbb{E}_X \left[ \log \frac{\overline{\exp(s(f(x), f(x^+)))}}{\underbrace{\exp(s(f(x), f(x^+)))}_{\text{score for the positive pair}} + \underbrace{\sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))}_{\text{score for the N-1 negative pairs}}} \right]$$

- 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:

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

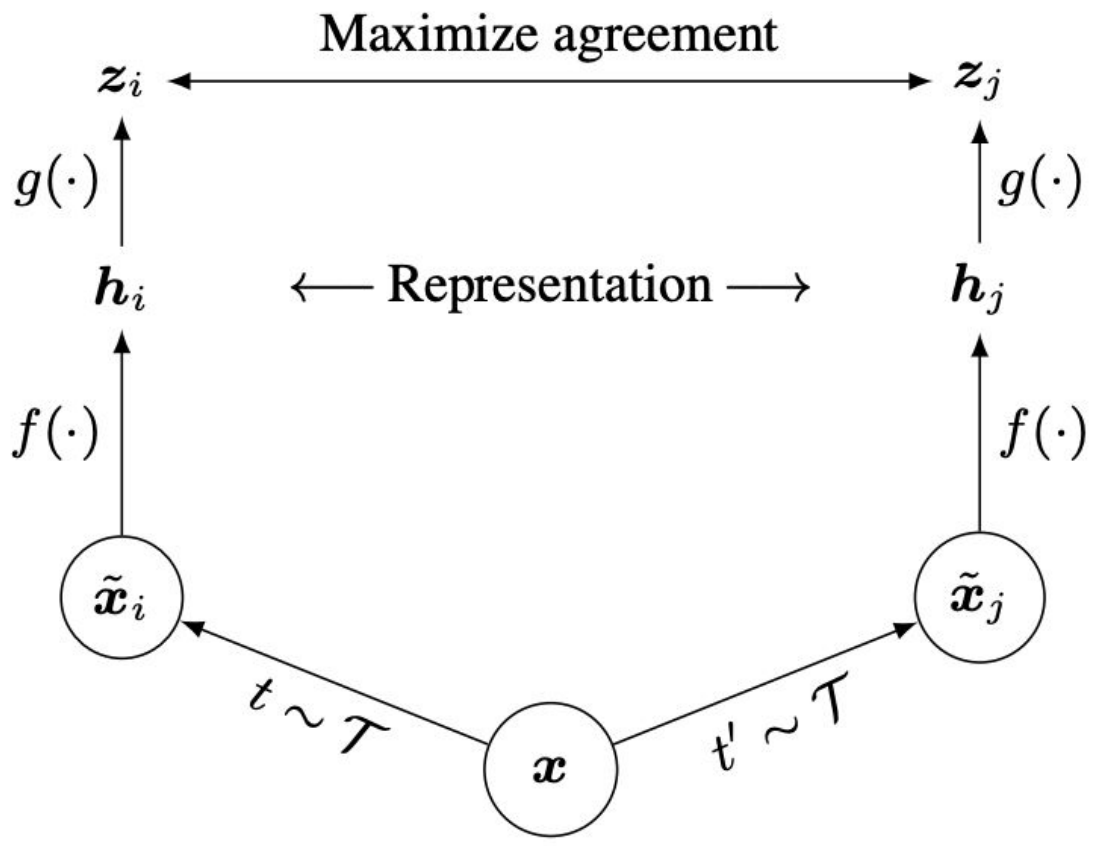
- Commonly known as the InfoNCE loss (van den Oord et al., 2018)
  A lower bound on the mutual information between f(x) and f(x$^+$)

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

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

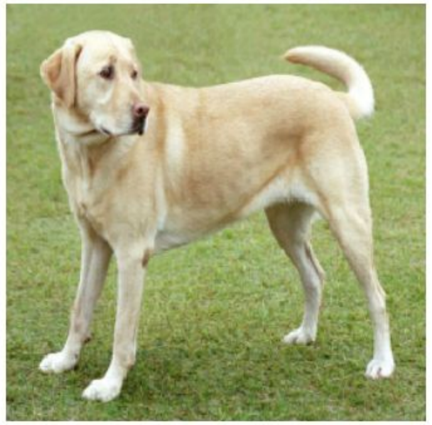
# SimCLR: A Simple Framework for Contrastive Learning

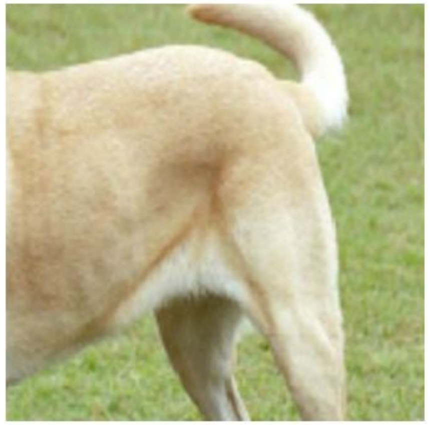
- Cosine similarity as the score function: $$s(u, v) = \frac{u^T v}{||u||||v||}$$
- Use a projection network h(·) to project features to a space where contrastive learning is applied.
- Generate positive samples through data augmentation:
  - random cropping, random color distortion, and random blur.

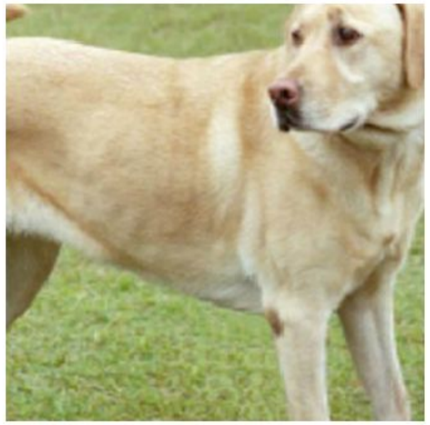


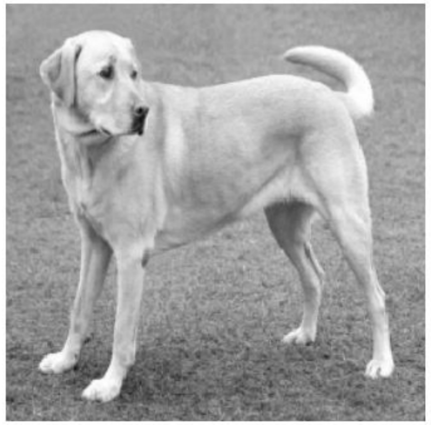
Source: Chen et al., 2020

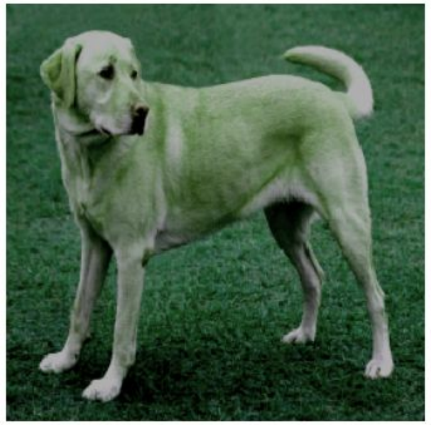
# SimCLR: Generating positive samples from data augmentation

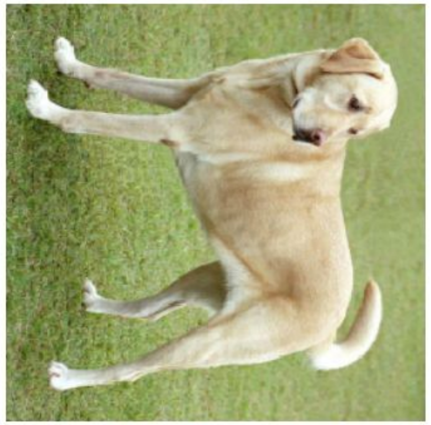

(a) Original

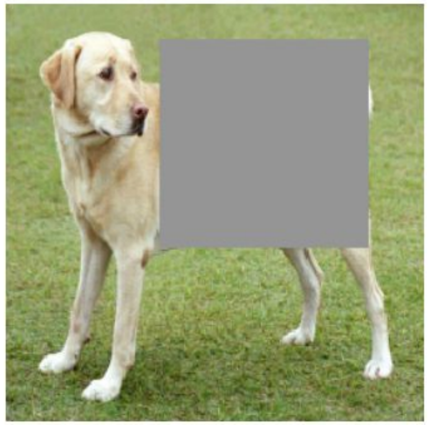

(b) Crop and resize

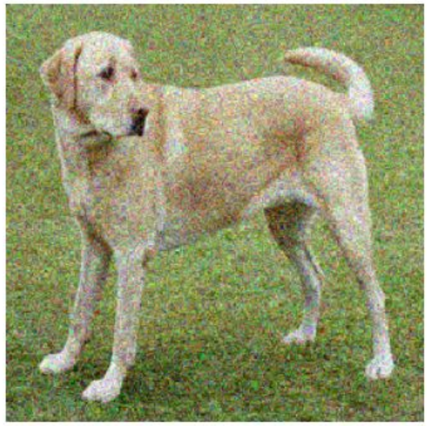

(c) Crop, resize (and flip)

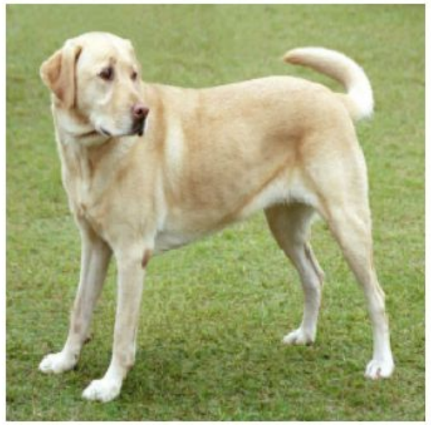

(d) Color distort. (drop)

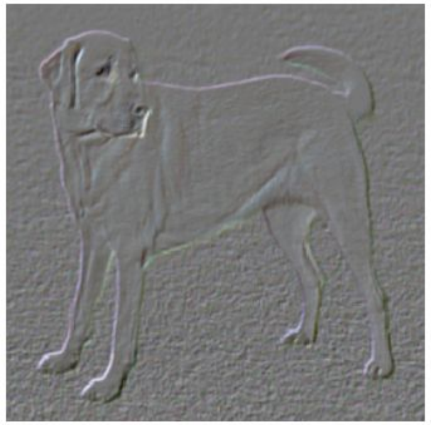

(e) Color distort. (jitter)


(f) Rotate $\{90^\circ, 180^\circ, 270^\circ\}$


(g) Cutout


(h) Gaussian noise

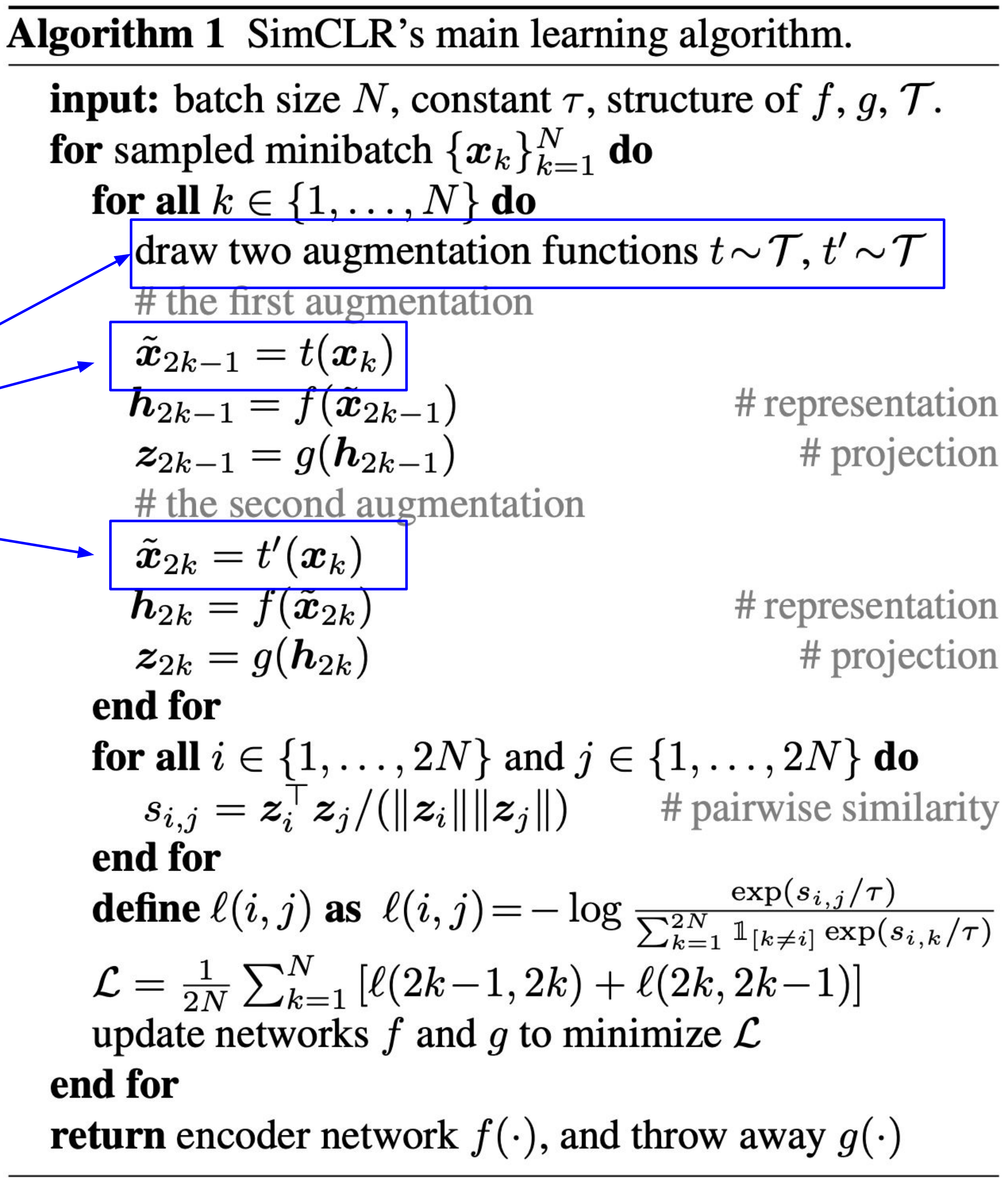

(i) Gaussian blur


(j) Sobel filtering

Source: Chen et al., 2020

# SimCLR: Generating positive samples from data augmentation

Generate a positive pair by sampling data augmentation functions

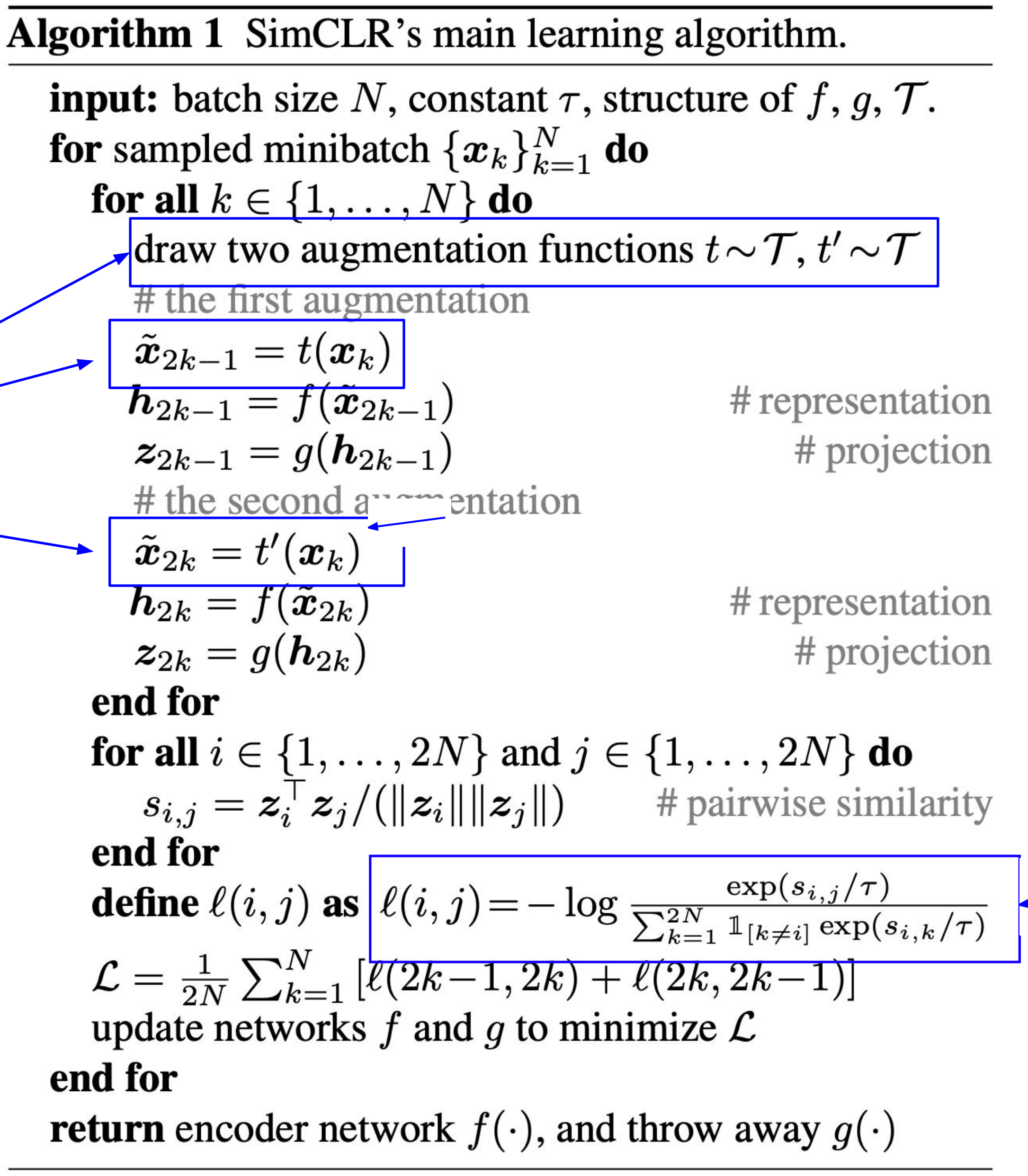


**Algorithm 1** SimCLR's main learning algorithm.

```
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
        $\boxed{\text{draw two augmentation functions } t \sim \mathcal{T}, t' \sim \mathcal{T}}$
        # the first augmentation
        $\boxed{\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)}$
        $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$                # representation
        $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$                # projection
        # the second augmentation
        $\boxed{\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)}$
        $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$                # 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: Generating positive samples from data augmentation



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

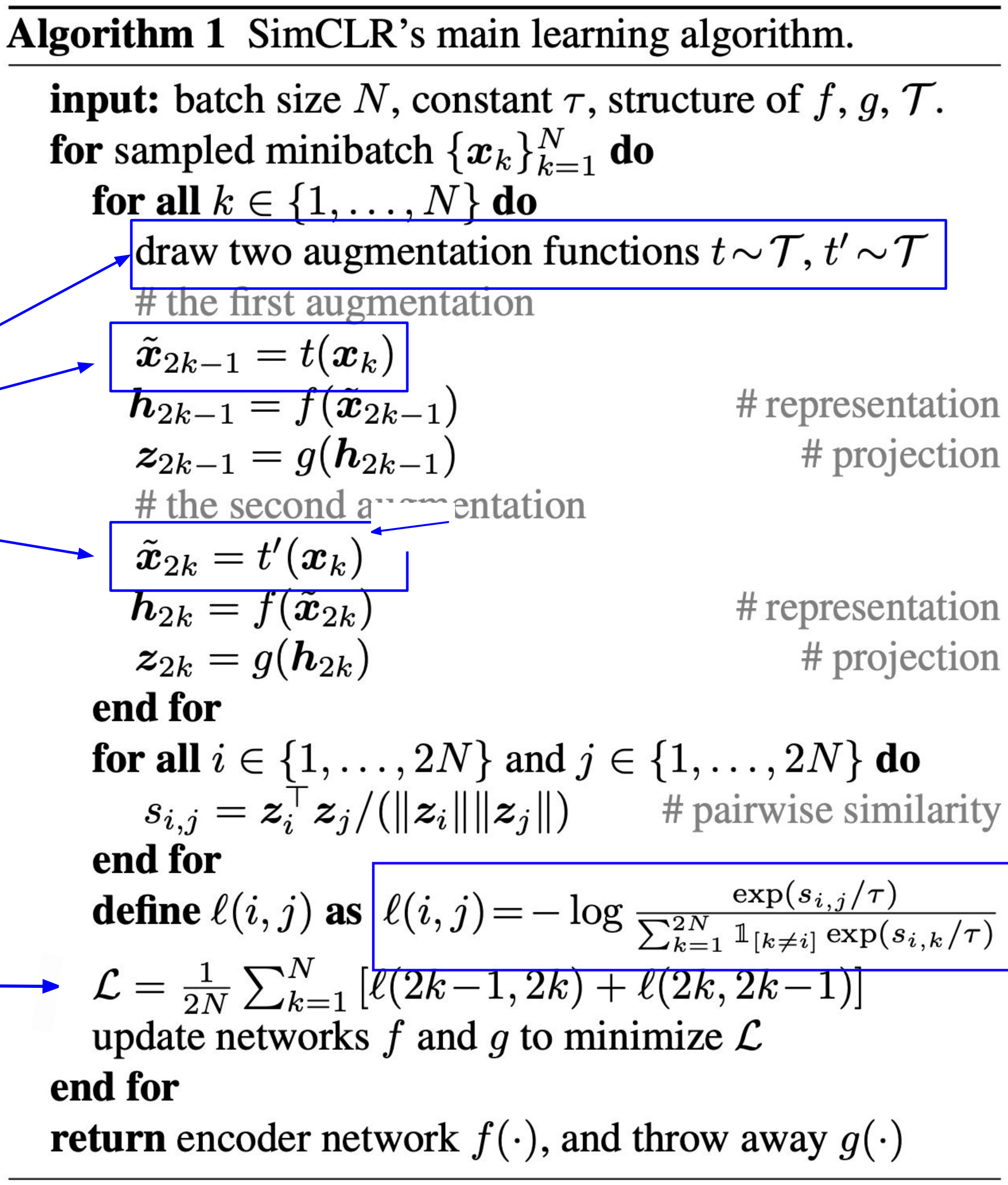
```
input: batch size $N$, constant $\tau$, structure of $f$, $g$, $\mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ do
  for all $k \in \{1, \ldots, N\}$ do
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$                # 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: Generating positive samples from data augmentation



**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

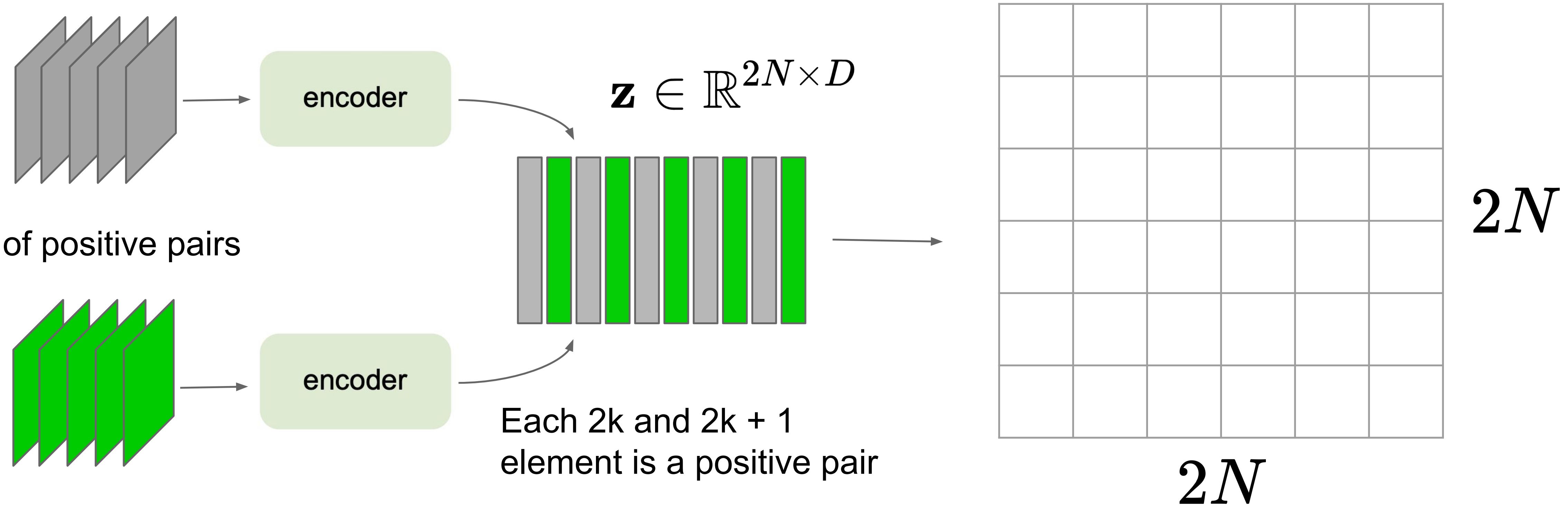
InfoNCE loss: Use all non-positive samples in the batch as x⁻

Source: Chen et al., 2020

# SimCLR: mini-batch training

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

"Affinity matrix"

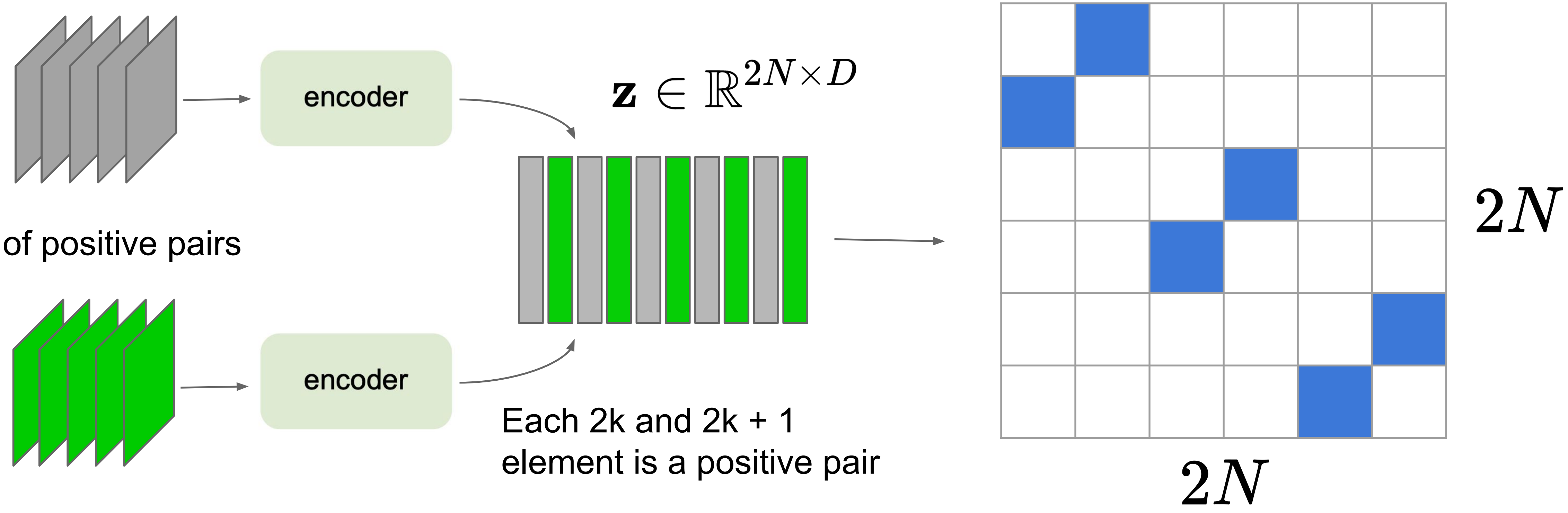


Each 2k and 2k + 1 element is a positive pair

# SimCLR: mini-batch training

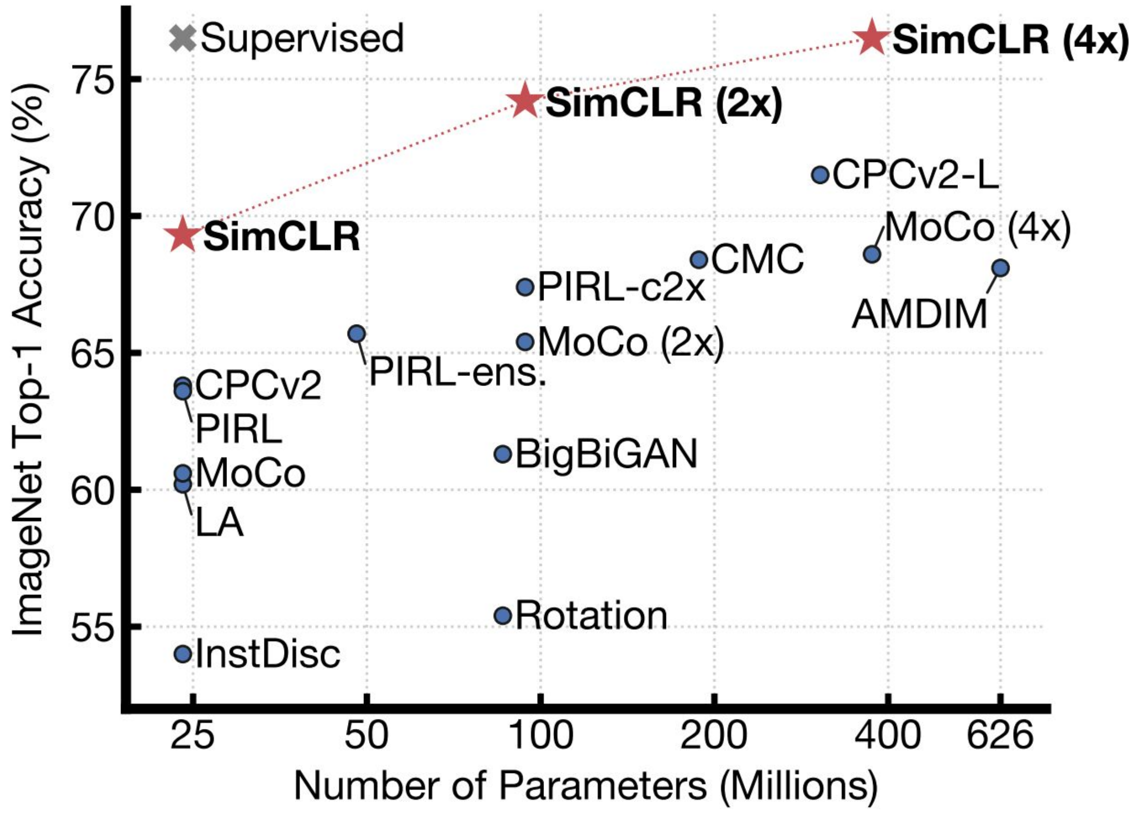
$$s_{i,j} = \frac{z_i^T z_j}{||z_i|| \, ||z_j||}$$

"Affinity matrix"


encoder
encoder
list of positive pairs
$\mathbf{z} \in \mathbb{R}^{2N \times D}$
Each 2k and 2k + 1 element is a positive pair
$2N$
$2N$

🟦 = classification label for each row

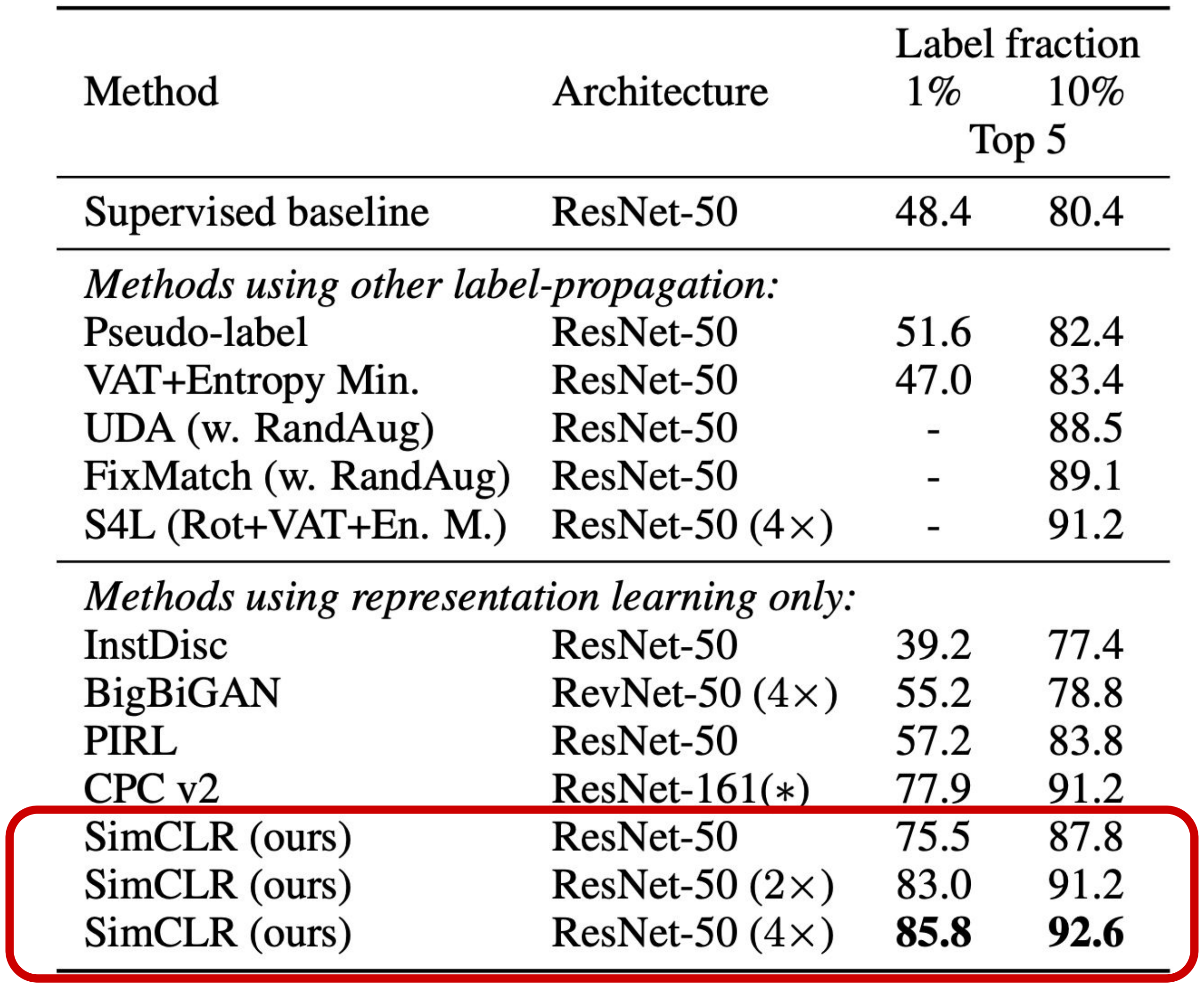
# Training linear classifier on SimCLR features



- 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

# Training linear classifier on SimCLR features



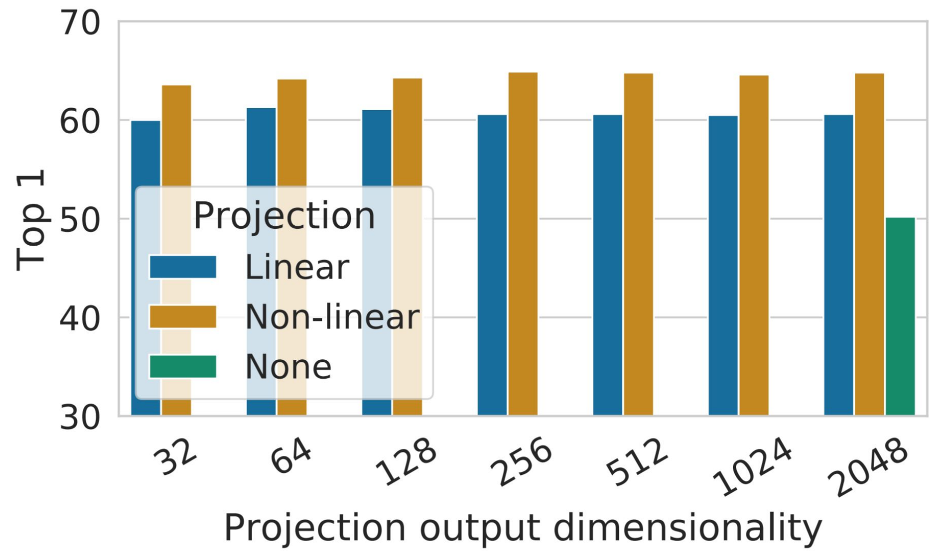
| Method | Architecture | Label fraction 1% | Label fraction 10% |
|---|---|---|---|
| | | Top 5 | |
| Supervised baseline | ResNet-50 | 48.4 | 80.4 |
| *Methods using other label-propagation:* | | | |
| Pseudo-label | ResNet-50 | 51.6 | 82.4 |
| VAT+Entropy Min. | ResNet-50 | 47.0 | 83.4 |
| UDA (w. 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** |

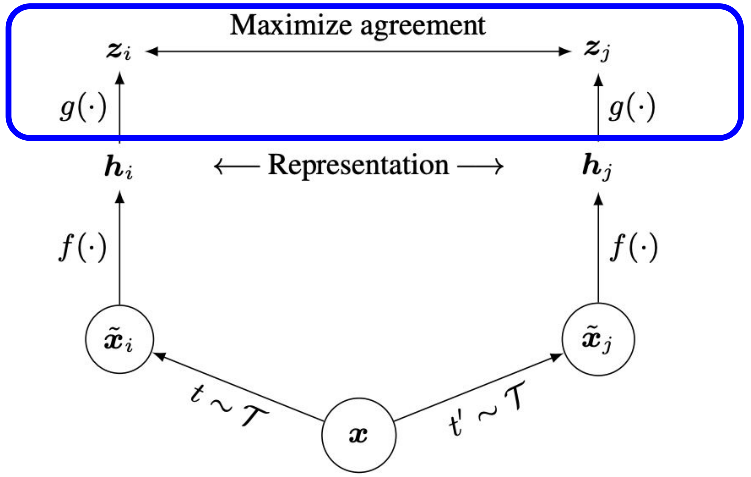
*Table 7.* ImageNet accuracy of models trained with few labels.

- 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

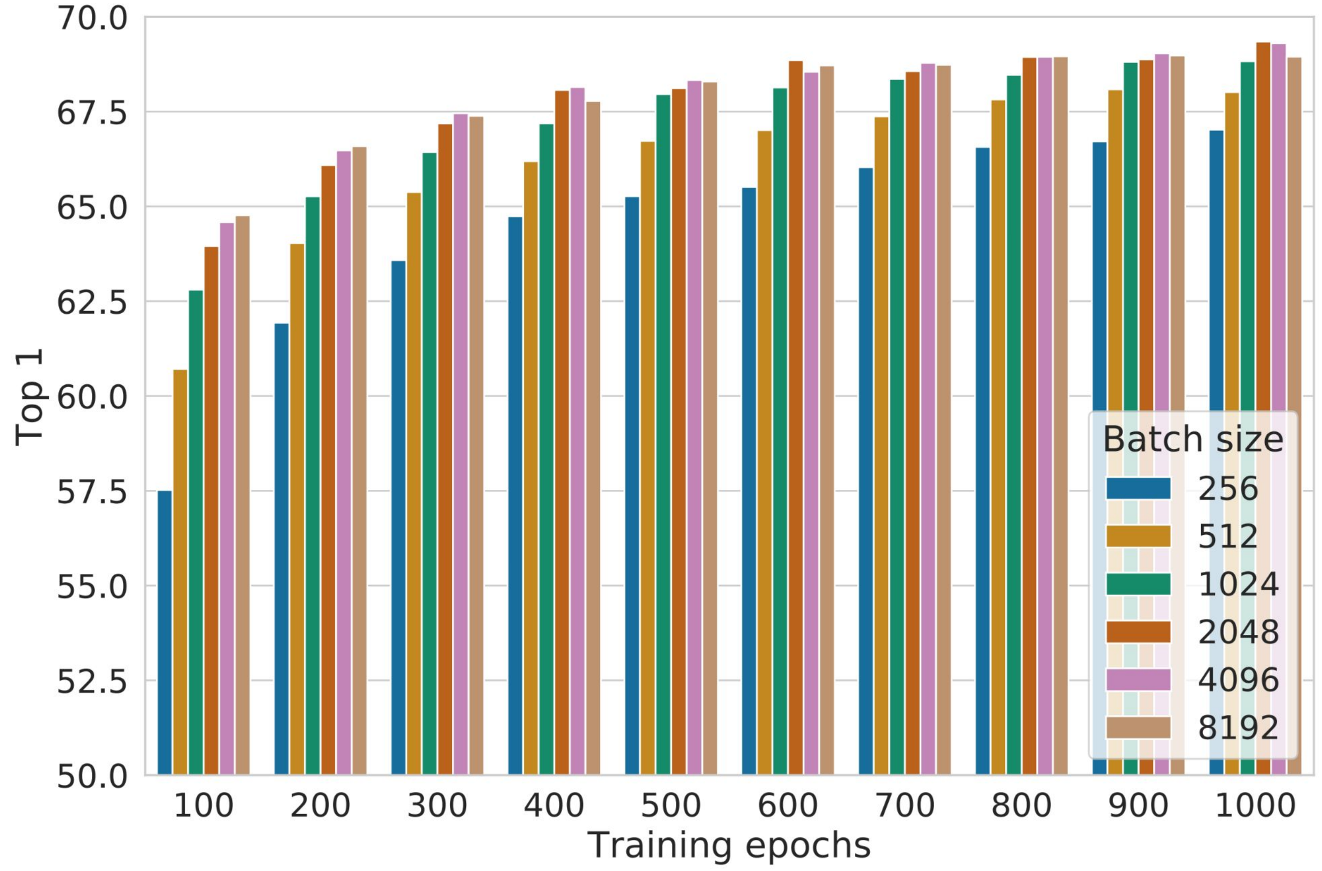
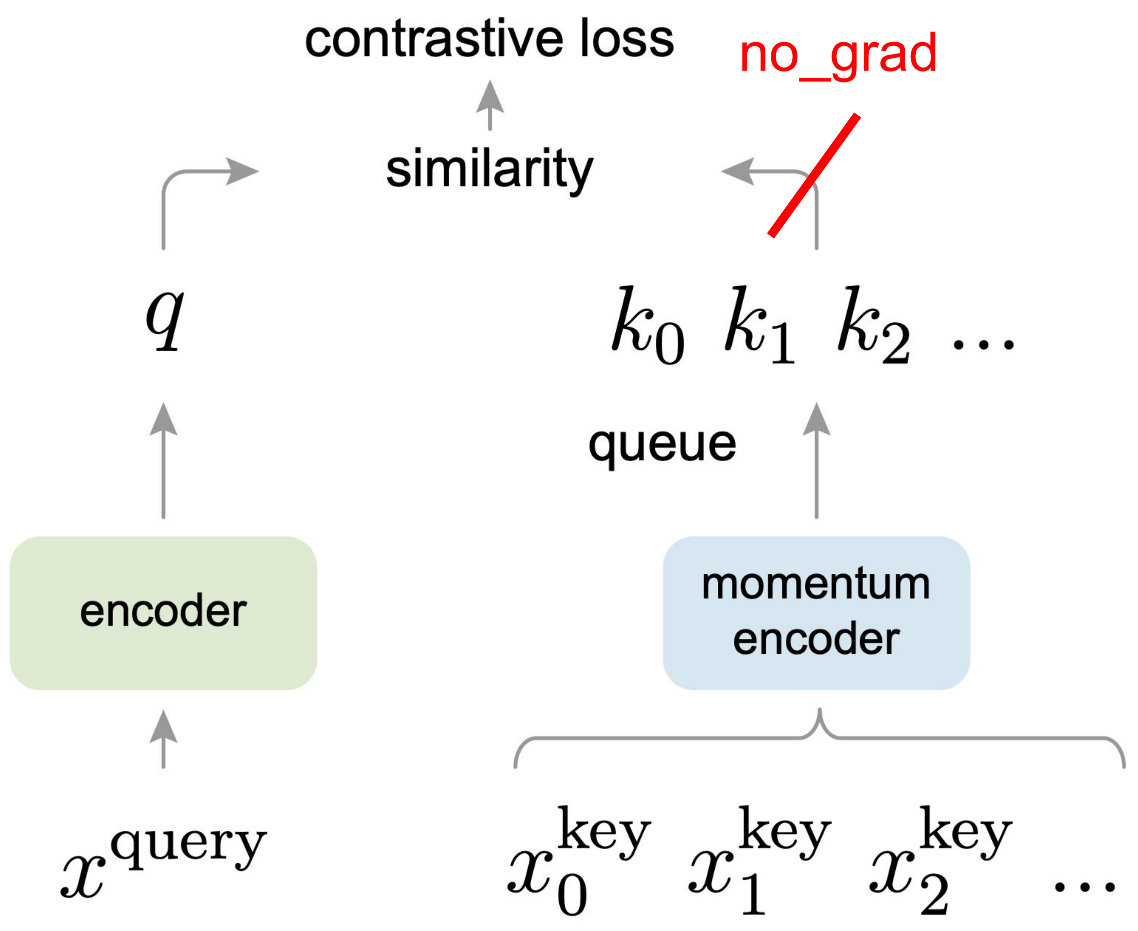


*Figure 9.* Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.[10]

- 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)

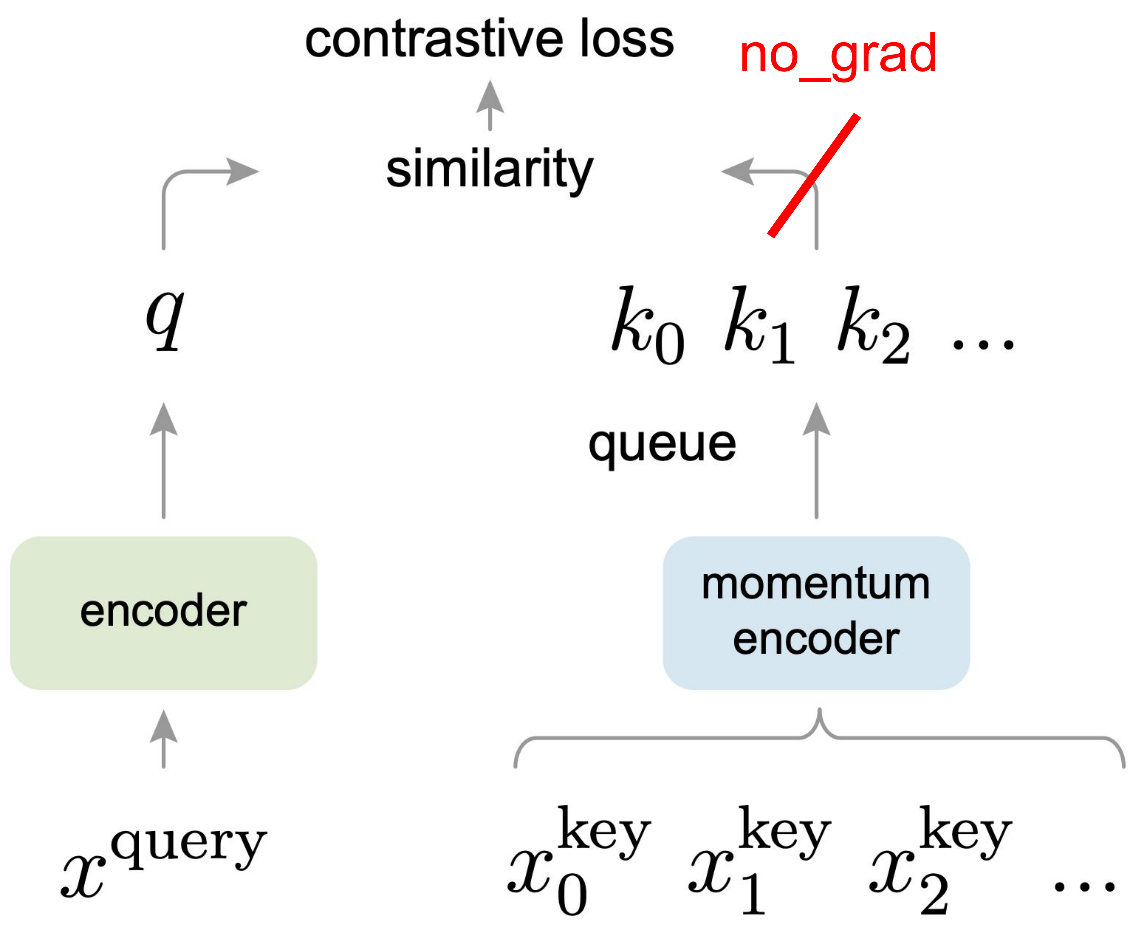


contrastive loss
no_grad
similarity
$q$
$k_0$ $k_1$ $k_2$ $\dots$
queue
encoder
momentum encoder
$x^{\text{query}}$
$x_0^{\text{key}}$ $x_1^{\text{key}}$ $x_2^{\text{key}}$ $\dots$

- 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)

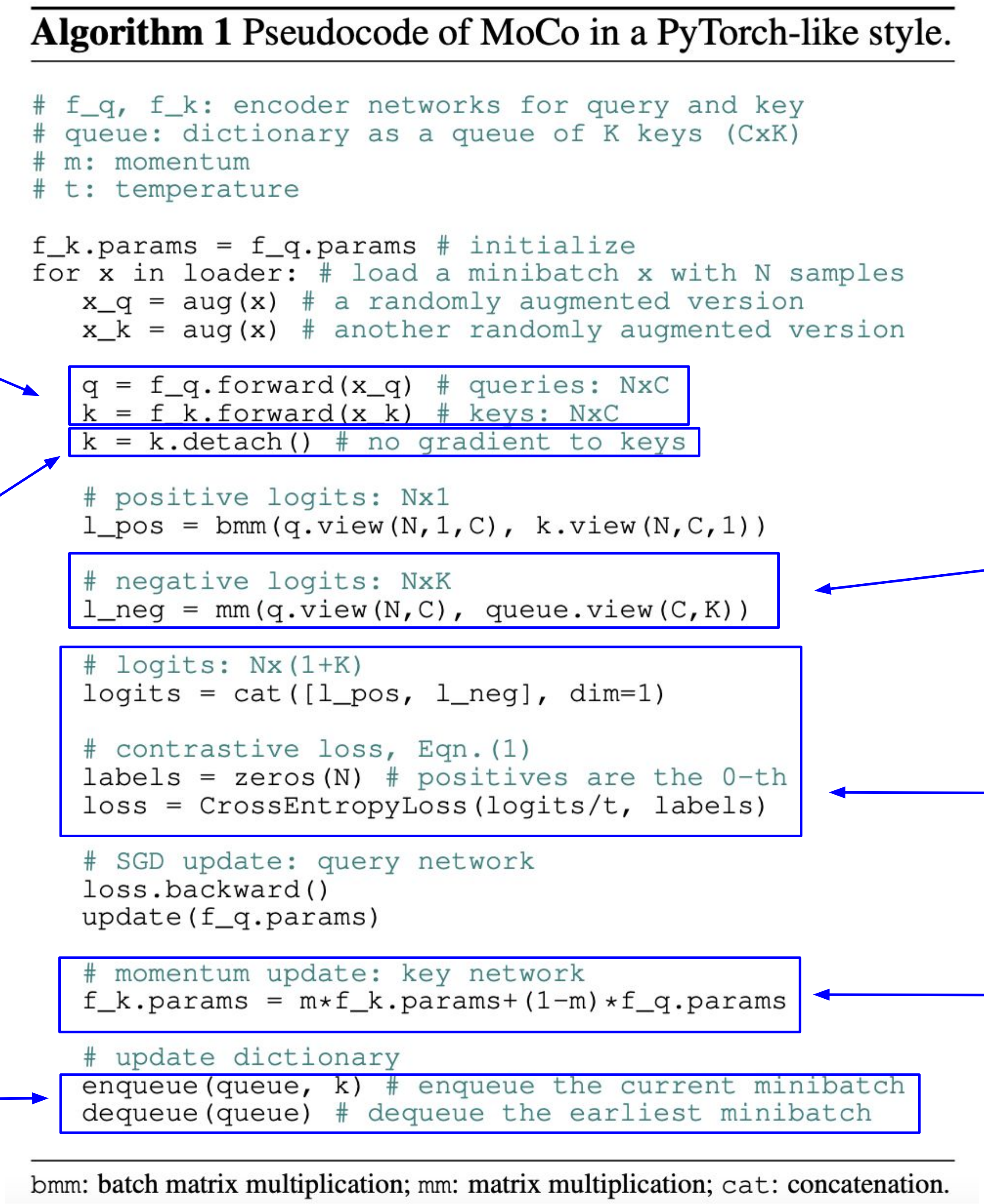


- Key differences to SimCLR:
  - Keep a running queue of keys (negative samples).
  - Compute gradients and update the encoder only through the queries.
  - Decouple min-batch size with the number of keys: can support a large number of negative samples.
  - The key encoder is slowly progressing through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

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

Source: He et al., 2020

# MoCo



**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.

Generate a positive pair by sampling data augmentation functions

No gradient through the positive sample

Use the running queue of keys as the negative samples

InfoNCE loss

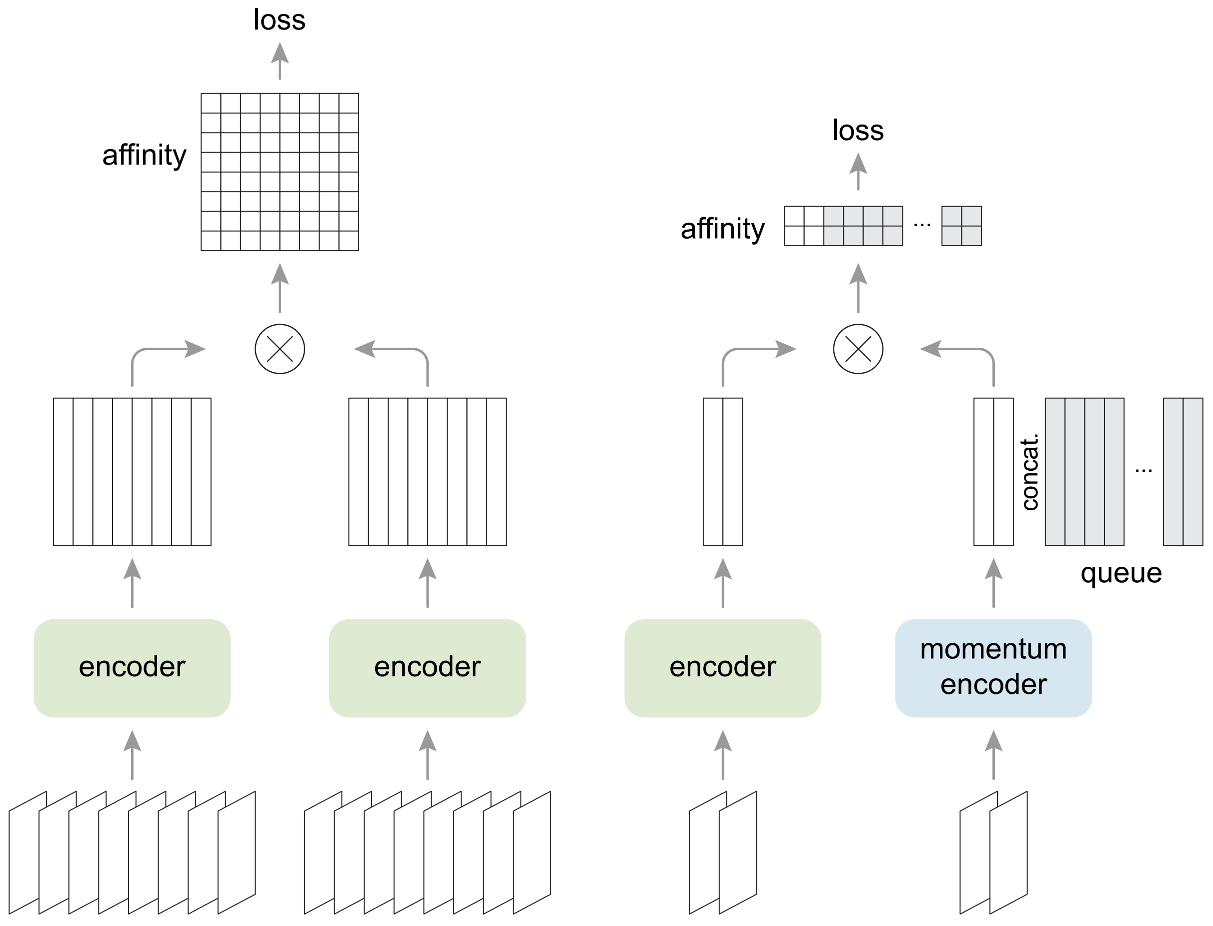
Update f_k through momentum

Update the FIFO negative sample queue

Source: He et al., 2020

# MoCo v2

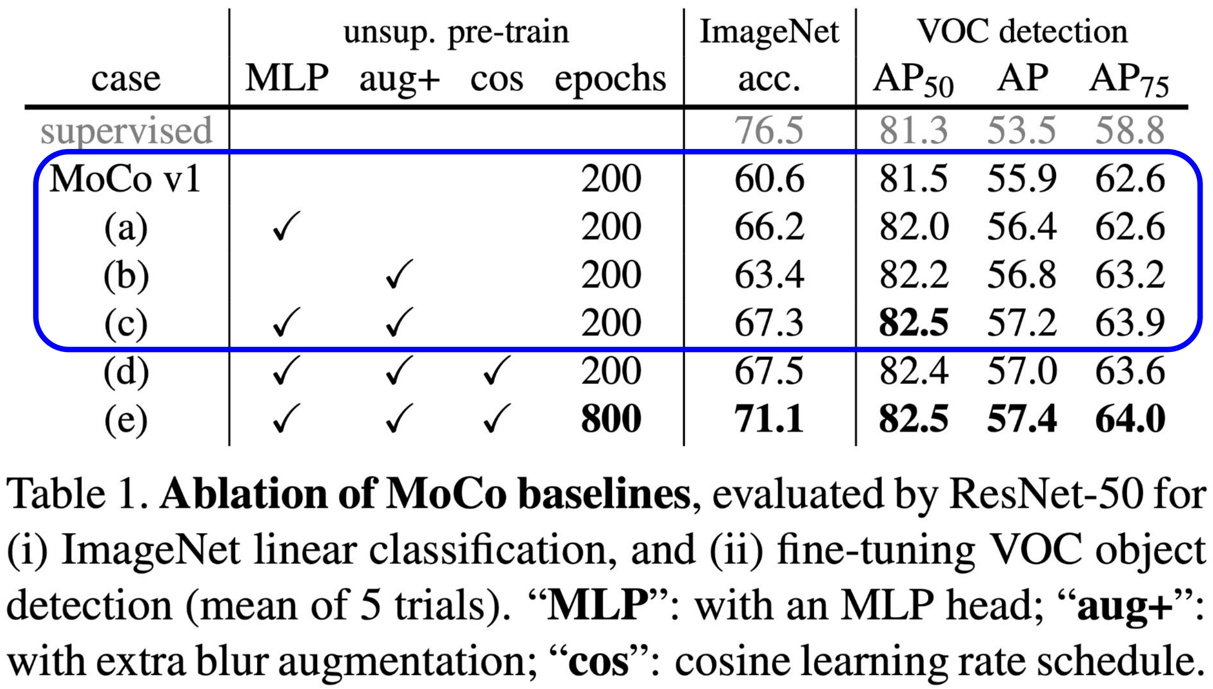
- A **hybrid of ideas** from SimCLR and MoCo:
- **From SimCLR:** non-linear projection head and strong data augmentation.
- **From MoCo:** momentum-updated queues that allow training on a large number of negative samples (no TPU required!).



(a) end-to-end (b) Momentum Contrast

Source: Chen et al., 2020

# Momentum Contrastive Learning (MoCo)



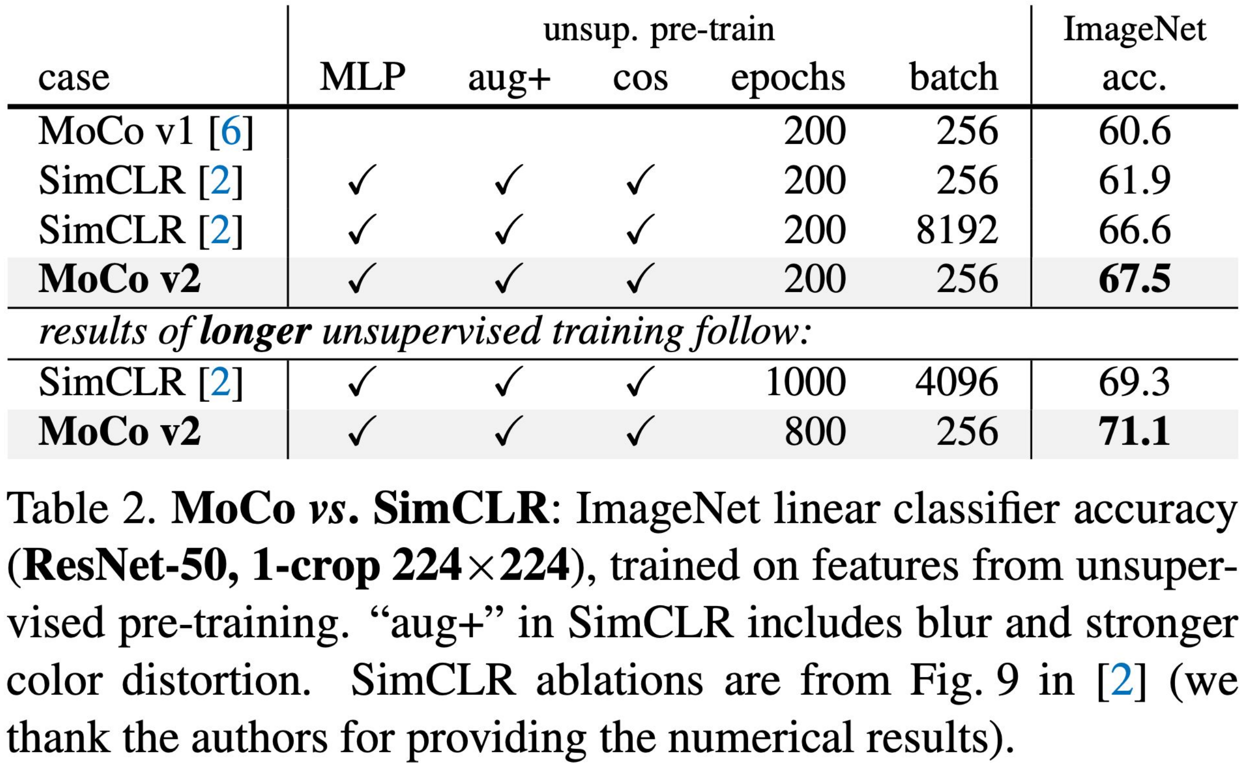
| case | unsup. pre-train | | | | ImageNet | VOC detection | | |
|---|---|---|---|---|---|---|---|---|
| | MLP | aug+ | cos | epochs | acc. | $AP_{50}$ | AP | $AP_{75}$ |
| supervised | | | | | 76.5 | 81.3 | 53.5 | 58.8 |
| MoCo v1 | | | | 200 | 60.6 | 81.5 | 55.9 | 62.6 |
| (a) | ✓ | | | 200 | 66.2 | 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

# Momentum Contrastive Learning (MoCo)



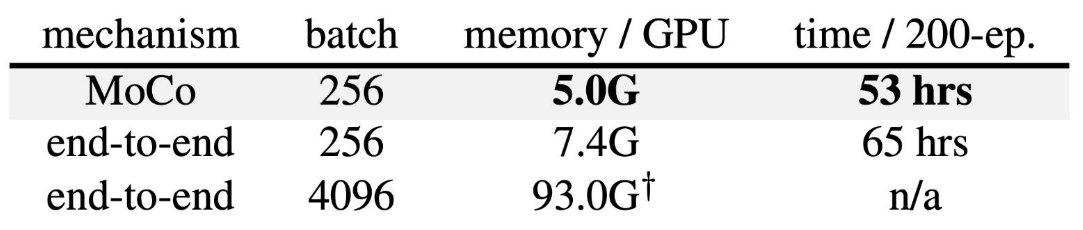
| | unsup. pre-train | | | | | ImageNet |
|---|---|---|---|---|---|---|
| case | MLP | aug+ | cos | epochs | batch | acc. |
| MoCo v1 [6] | | | | 200 | 256 | 60.6 |
| SimCLR [2] | ✓ | ✓ | ✓ | 200 | 256 | 61.9 |
| SimCLR [2] | ✓ | ✓ | ✓ | 200 | 8192 | 66.6 |
| **MoCo v2** | ✓ | ✓ | ✓ | 200 | 256 | **67.5** |
| *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

# Momentum Contrastive Learning (MoCo)



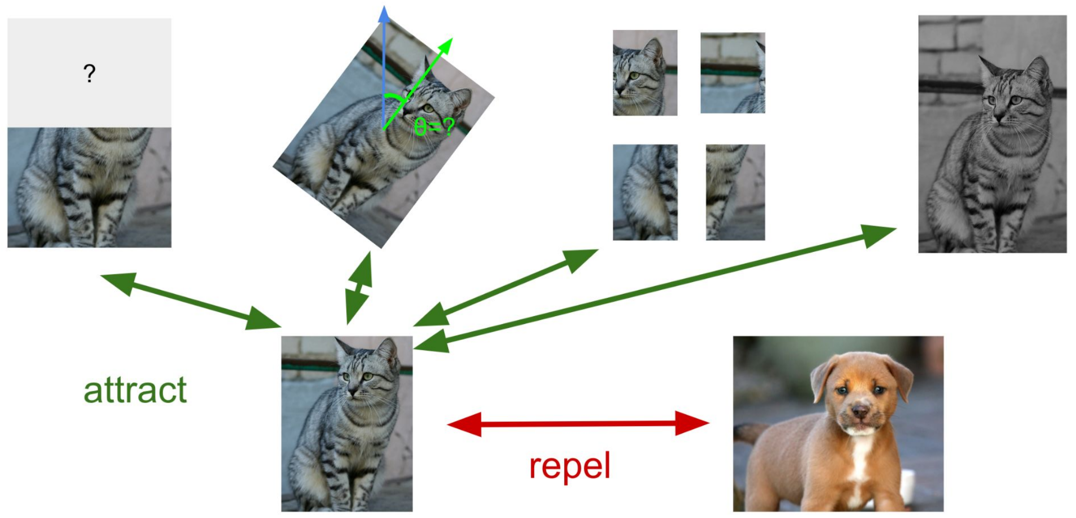
| mechanism | batch | memory / GPU | time / 200-ep. |
|---|---|---|---|
| MoCo | 256 | **5.0G** | **53 hrs** |
| end-to-end | 256 | 7.4G | 65 hrs |
| end-to-end | 4096 | 93.0G[†] | n/a |

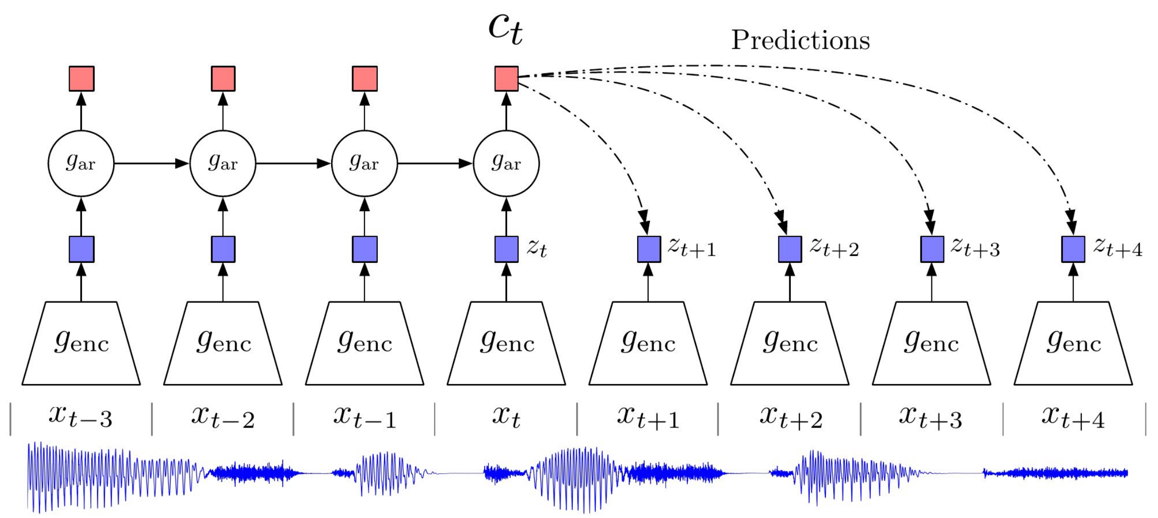
Table 3. **Memory and time cost** in 8 V100 16G GPUs, implemented in PyTorch. [†]: based on our estimation.

- 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





Source: van den Oord et al., 2018

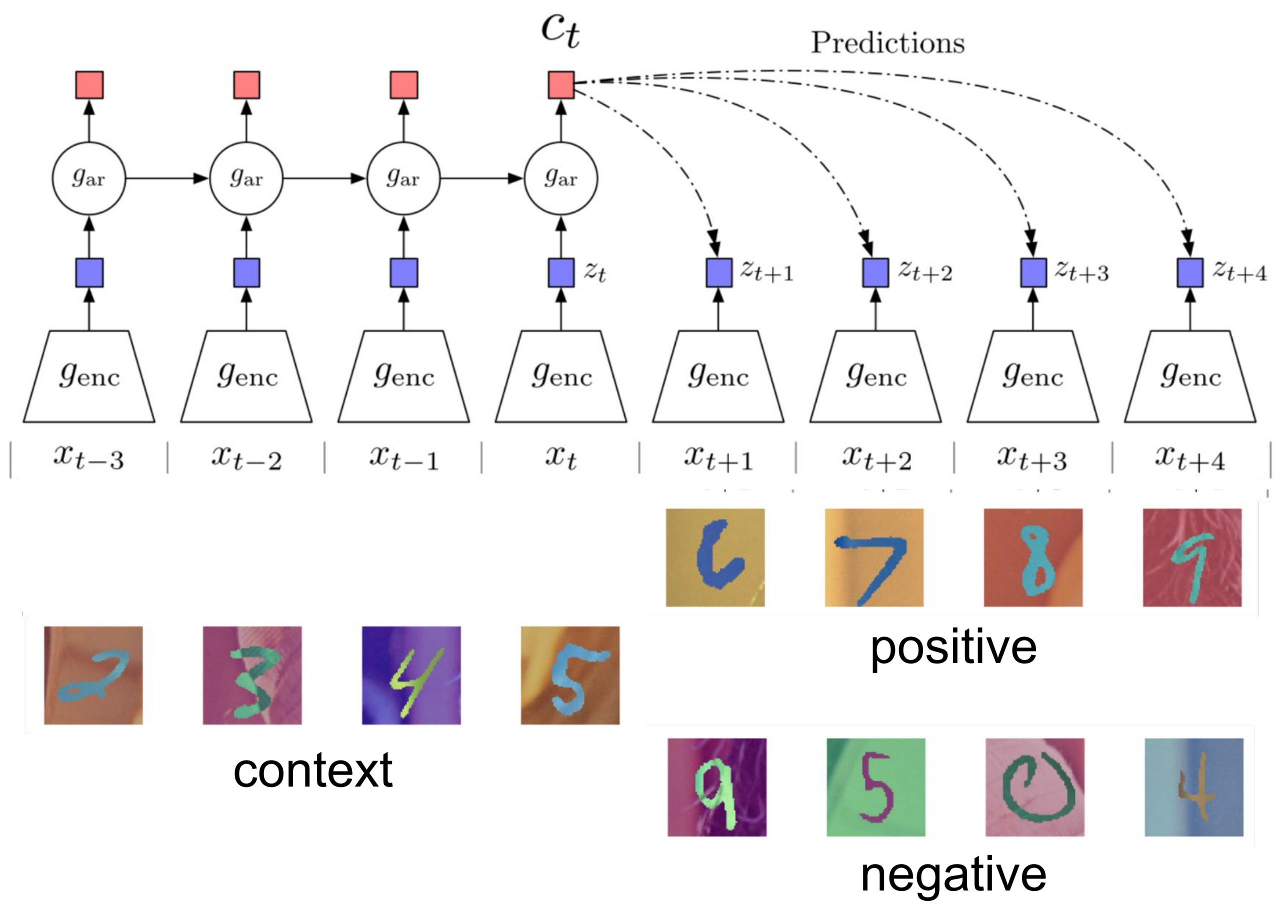
**Instance-level contrastive learning:**
contrastive learning based on
positive & negative instances.

Examples: **SimCLR, MoCo, MoCo v2**

**Sequence-level contrastive learning:**
contrastive learning based on
sequential / temporal orders.

Example: **Contrastive Predictive Coding (CPC)**

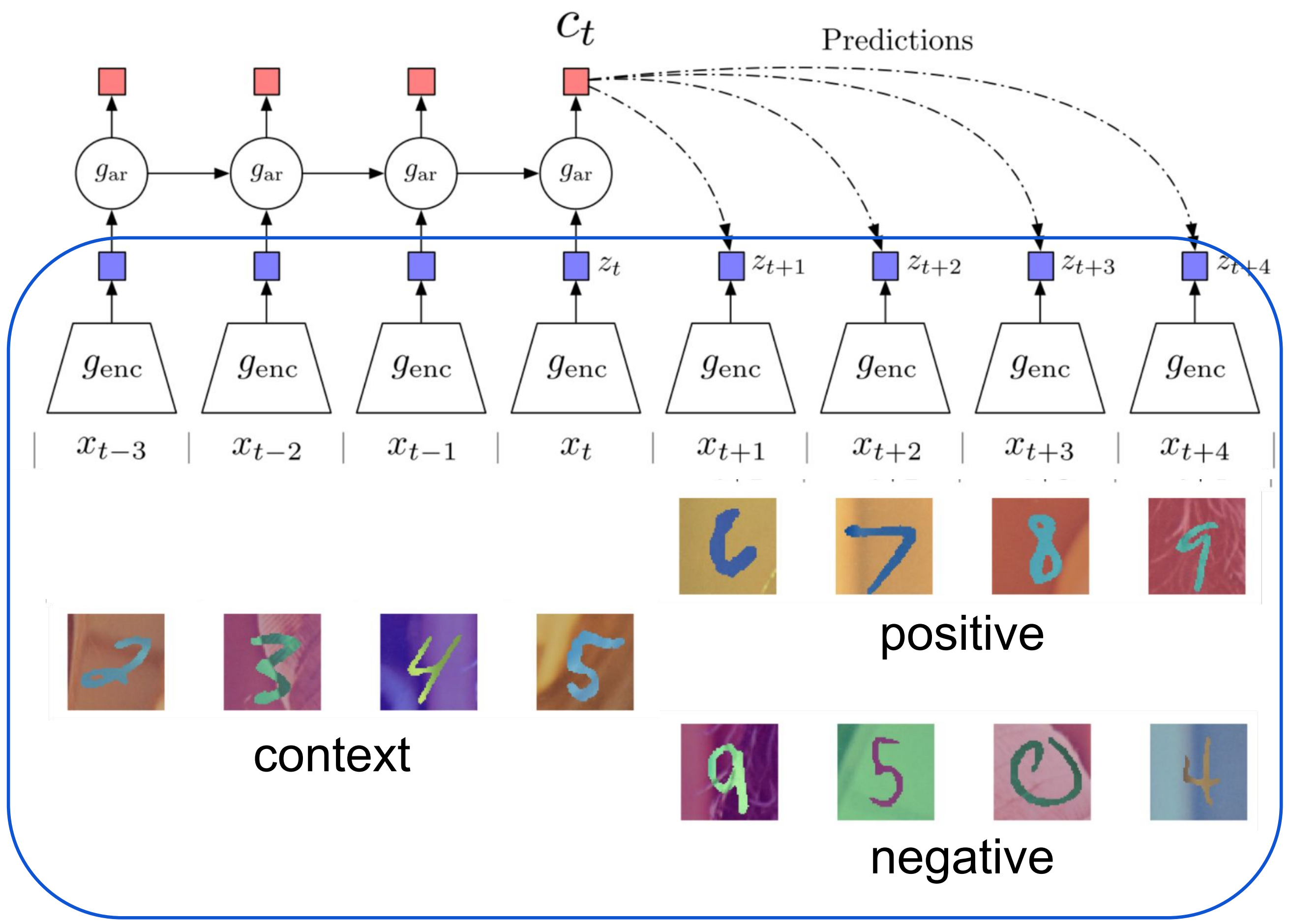
# Contrastive Predictive Coding (CPC)



- **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.

Source: van den Oord et al., 2018

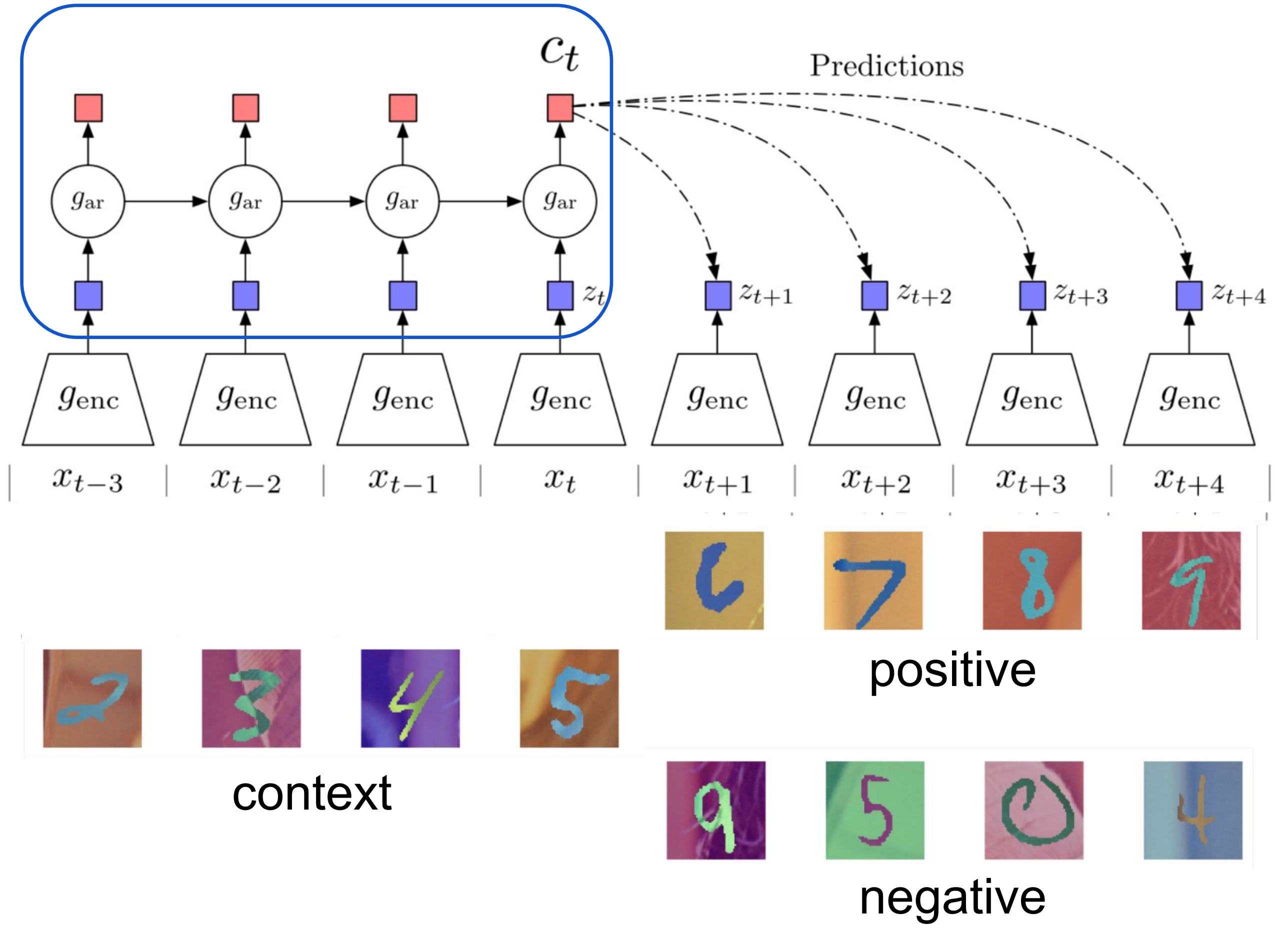
# Contrastive Predictive Coding (CPC)



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

Source: van den Oord et al., 2018

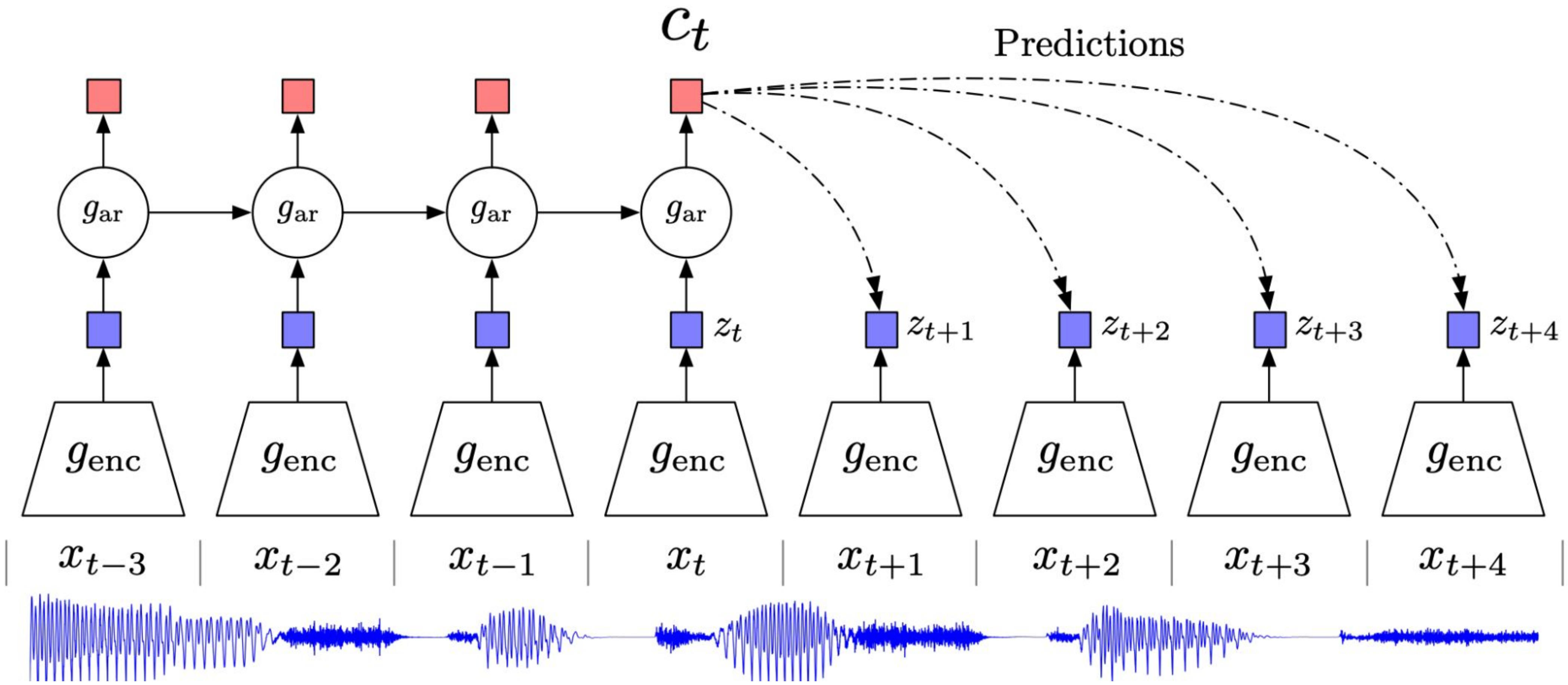
# Contrastive Predictive Coding (CPC)



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 ct using an auto-regressive model ($g_{ar}$).
   The original paper uses GRU-RNN here.

Source: van den Oord et al., 2018

# CPC example: modeling audio sequences



Source: van den Oord et al., 2018

# CPC example: modeling audio sequences

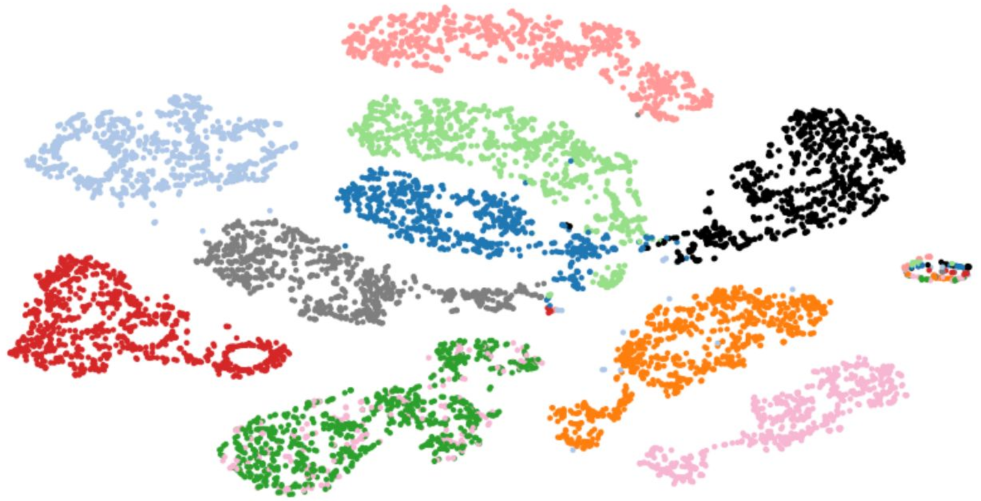


Figure 2: t-SNE visualization of audio (speech) representations for a subset of 10 speakers (out of 251). Every color represents a different speaker.

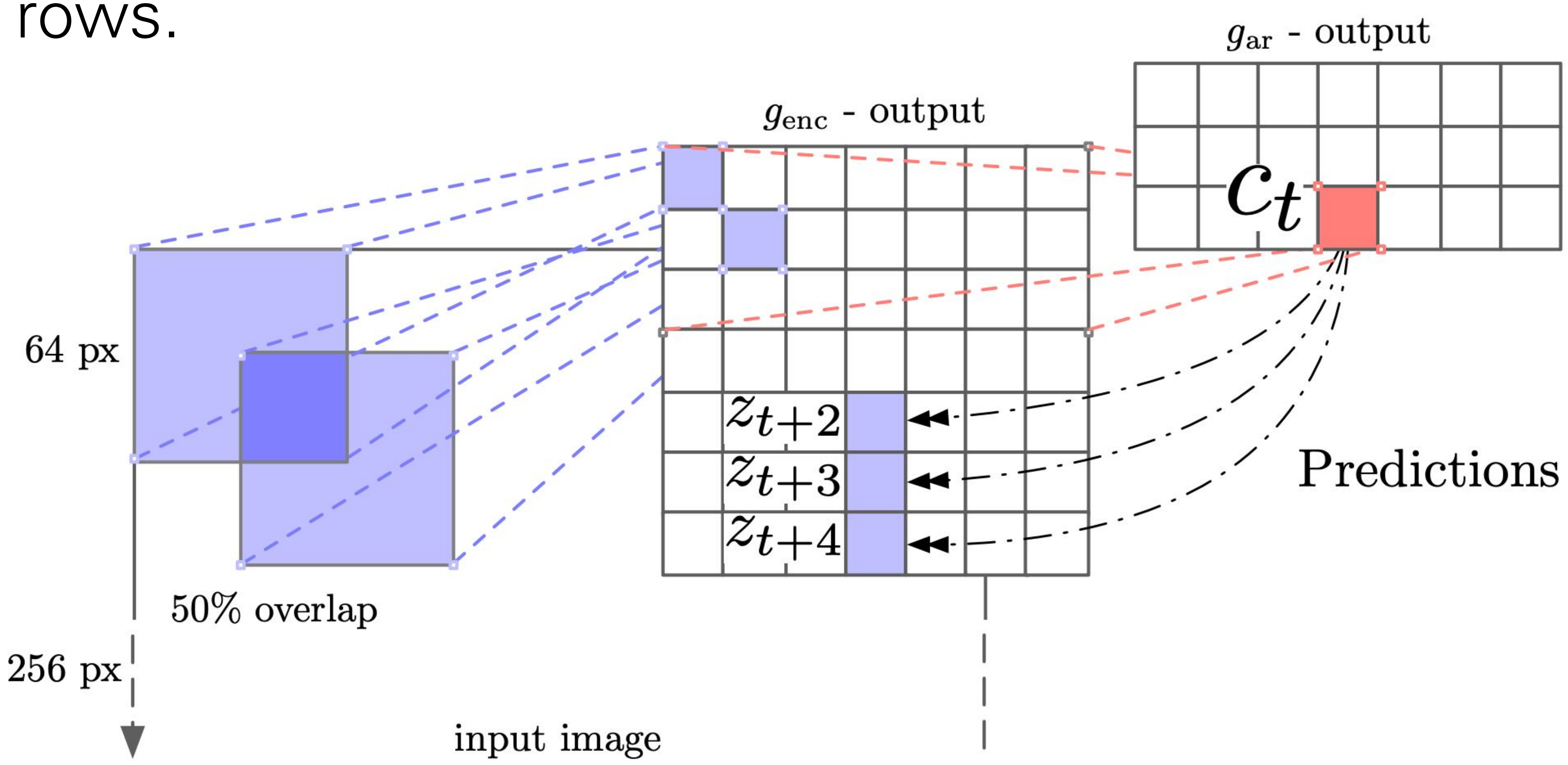
| Method | ACC |
|---|---|
| **Phone classification** | |
| Random initialization | 27.6 |
| MFCC features | 39.7 |
| CPC | 64.6 |
| Supervised | 74.6 |
| **Speaker classification** | |
| Random initialization | 1.87 |
| MFCC features | 17.6 |
| CPC | 97.4 |
| Supervised | 98.5 |

Linear classification on trained representations (LibriSpeech dataset)

Source: van den Oord et al., 2018

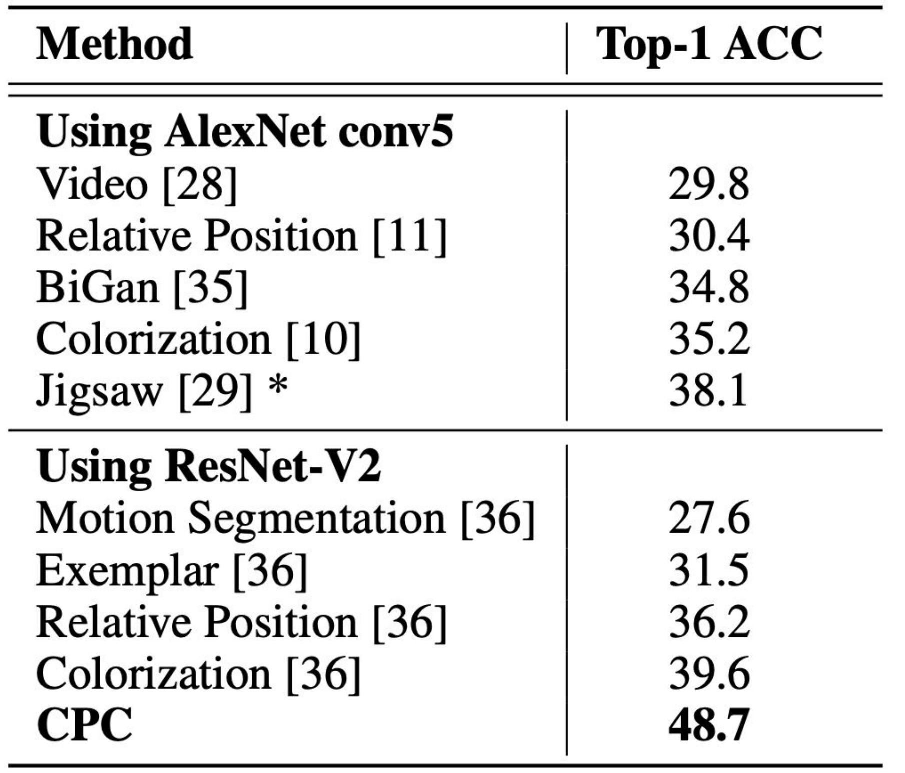
## CPC example: modeling audio sequences

- **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

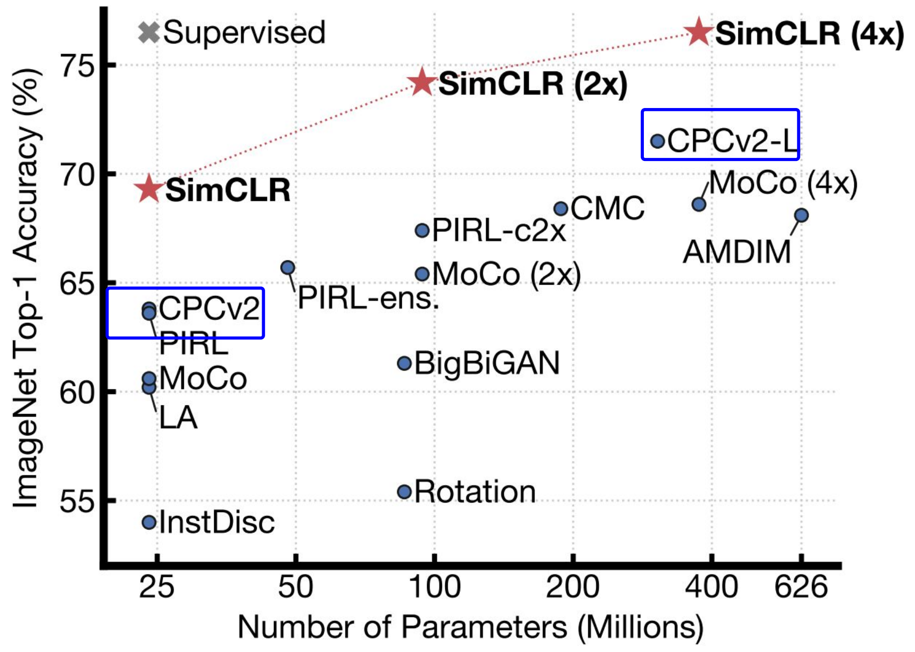
# CPC example: modeling audio sequences



| 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