Deep Learning

Contrastive & Self-Supervised Learning

or how to overcome the need for huge amounts of labels

Contrastive & Self-Supervised Learning

or how to overcome the need for huge amounts of labels

A large class of contemporary ML methods rely on **human provided-labels** or rewards as the only form of learning signal used during the training process.

This over-reliance on direct-**semantic** supervision has several perils:

- The underlying **data has a much richer structure than what sparse labels could provide**. Thus, purely supervised learning algorithms often require large numbers of samples to learn from, and converge to brittle solutions.
- It leads to **task-specific solutions**, rather than knowledge that can be repurposed.

https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html

Contrastive & Self-Supervised Learning

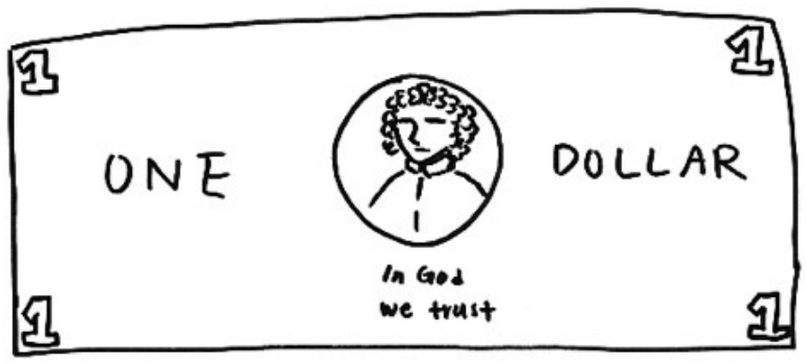
or how to overcome the need for huge amounts of labels

https://jrzech.medium.com/what-are-radiological-deep-learning-models-actually-learning-f97a546c5b98

Contrastive & Self-Supervised Learning

Self-Supervised Learning provides a promising alternative **to learn powerful representations**, where the data itself provides the supervision for a learning algorithm.

Consider this experiment conducted where subjects were asked to draw a picture of the dollar bill as detailed as possible. Here is a drawing ‘from memory’:



And here is the drawing subsequently maded with a dollar bill present:

https://aeon.co/essays/your-brain-does-not-process-information-and-it-is-not-a-computer

Contrastive & Self-Supervised Learning

Despite having seen a dollar bill countless number of times, we don’t retain a full representation of it.

In fact, we really only retain enough features of the bill **to distinguish it from other objects.** This can be extended to objects that have no name or to individuals.

Caricatures of people you know are recognized more quickly and accurately than life-like drawings or photographs.

https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html

Contrastive & Self-Supervised Learning

Similarly, can we build **representation learning** algorithms (**not relying in semantic labels**) that don't concentrate on pixel-level details, and only encode high-level features sufficient enough to distinguish different objects?

Considering the amount of **unlabelled data** (e.g. free text, all the images on the Internet) is substantially more than a limited number of human curated labelled datasets, it is kinda wasteful not to use them. However, **unsupervised learning** is not easy and usually works much less **efficiently** than supervised learning.

https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html

Contrastive & Self-Supervised Learning

What if we can get **labels for free** for unlabelled data and train unsupervised dataset in a supervised manner?

We can achieve this by framing an **auxilary supervised learning task** in a special form to predict only **a subset of information of the target using the rest**.

This is known as **self-supervised learning**.

https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Contrastive & Self-Supervised Learning

EXAMPLE

The default task for a language model is to **predict the next word given the past sequence**.

BERT adds two other auxiliary tasks and both rely on self-generated labels:

Task 1: Mask language model (MLM)

> From Wikipedia: "A cloze test (also cloze deletion test) is an exercise, test, or assessment consisting of a portion of language with certain items, words, or signs removed (cloze text), where the participant is asked to replace the missing language item. ... The exercise was first described by W.L. Taylor in 1953."

It is unsurprising to believe that a representation that learns the context around a word rather than just after the word is able to better capture its meaning, both syntactically and semantically. BERT encourages the model to do so by training on the *"mask language model" task*:

1. Randomly mask 15% of tokens in each sequence. Because if we only replace masked tokens with a special placeholder `[MASK]`, the special token would never be encountered during fine-tuning. Hence, BERT employed several heuristic tricks:
    - (a) with 80% probability, replace the chosen words with `[MASK]`;
    - (b) with 10% probability, replace with a random word;
    - (c) with 10% probability, keep it the same.
2. The model only predicts the missing words, but it has no information on which words have been replaced or which words should be predicted. The output size is only 15% of the input size.

Contrastive & Self-Supervised Learning

EXAMPLE

The default task for a language model is to predict the next word given the past sequence. BERT adds two other auxiliary tasks and both rely on self-generated labels:

Task 2: Next sentence prediction

Motivated by the fact that many downstream tasks involve the understanding of relationships between sentences (i.e., QA, NLI), BERT added another auxiliary task on training a *binary classifier* for telling whether one sentence is the next sentence of the other:

1. Sample sentence pairs (A, B) so that:
    - (a) 50% of the time, B follows A;
    - (b) 50% of the time, B does not follow A.
2. The model processes both sentences and output a binary label indicating whether B is the next sentence of A.

The training data for both auxiliary tasks above can be trivially generated from any monolingual corpus.

Hence the scale of training is unbounded.

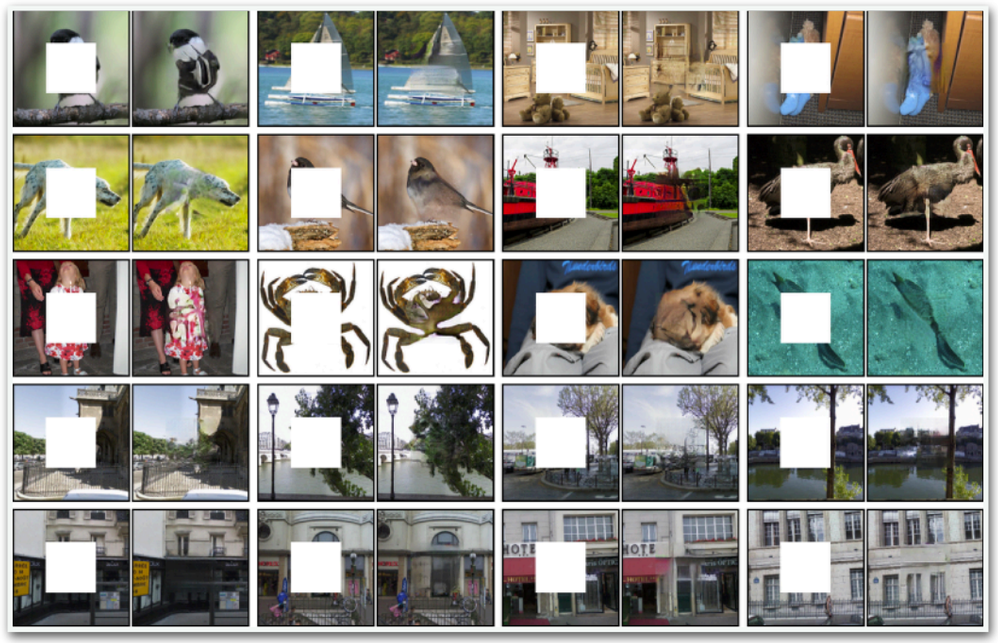
https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Many ideas have been proposed for self-supervised representation learning on **images**.

A common workflow is to train a model on one or multiple auxilliary tasks with unlabelled images and then **use one intermediate feature layer of this model to feed** a multinomial logistic regression classifier on image classification.

The final classification accuracy quantifies how good the learned representation is.

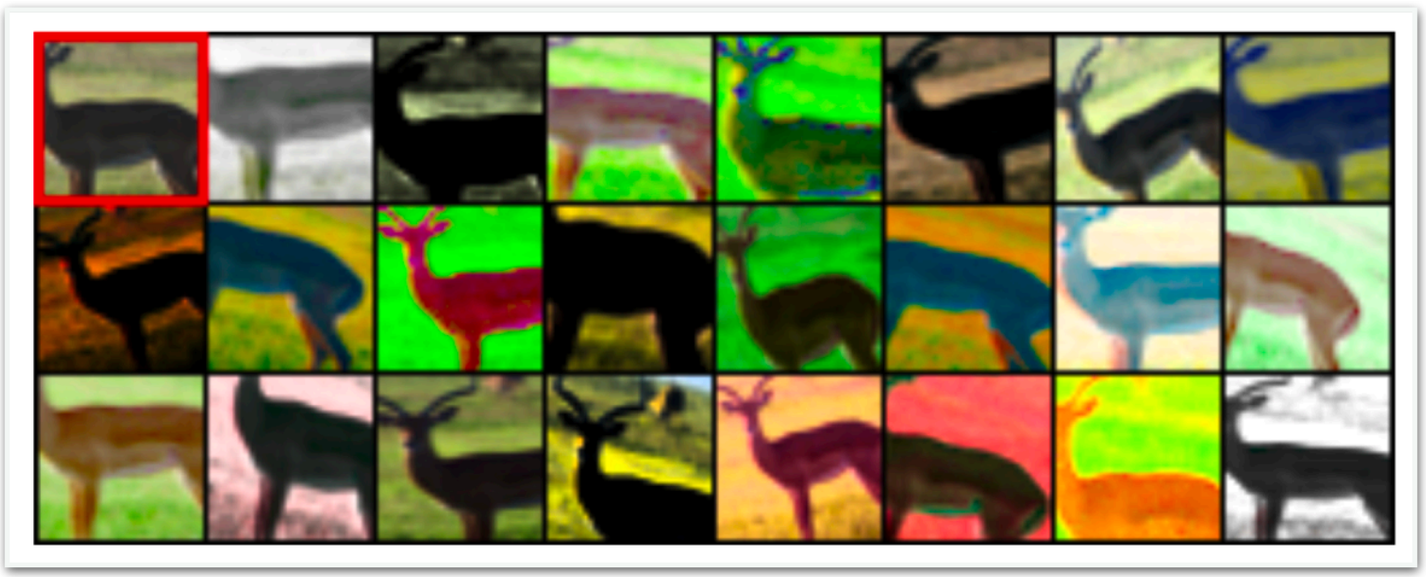


https://people.eecs.berkeley.edu/~pathak/context_encoder/

Image-based Self-Supervised Learning

Distortion

1. Sample N patches of size 32 × 32 pixels from different images at varying positions and scales, only from regions containing considerable gradients as those areas cover edges and tend to contain objects or parts of objects. They are “exemplary” patches.
2. Each patch is distorted by applying a variety of random transformations (i.e., translation, rotation, scaling, etc.). All the resulting distorted patches are considered to belong to the same surrogate class.
3. The pretext task is to discriminate between a set of surrogate classes. We can arbitrarily create as many surrogate classes as we want.

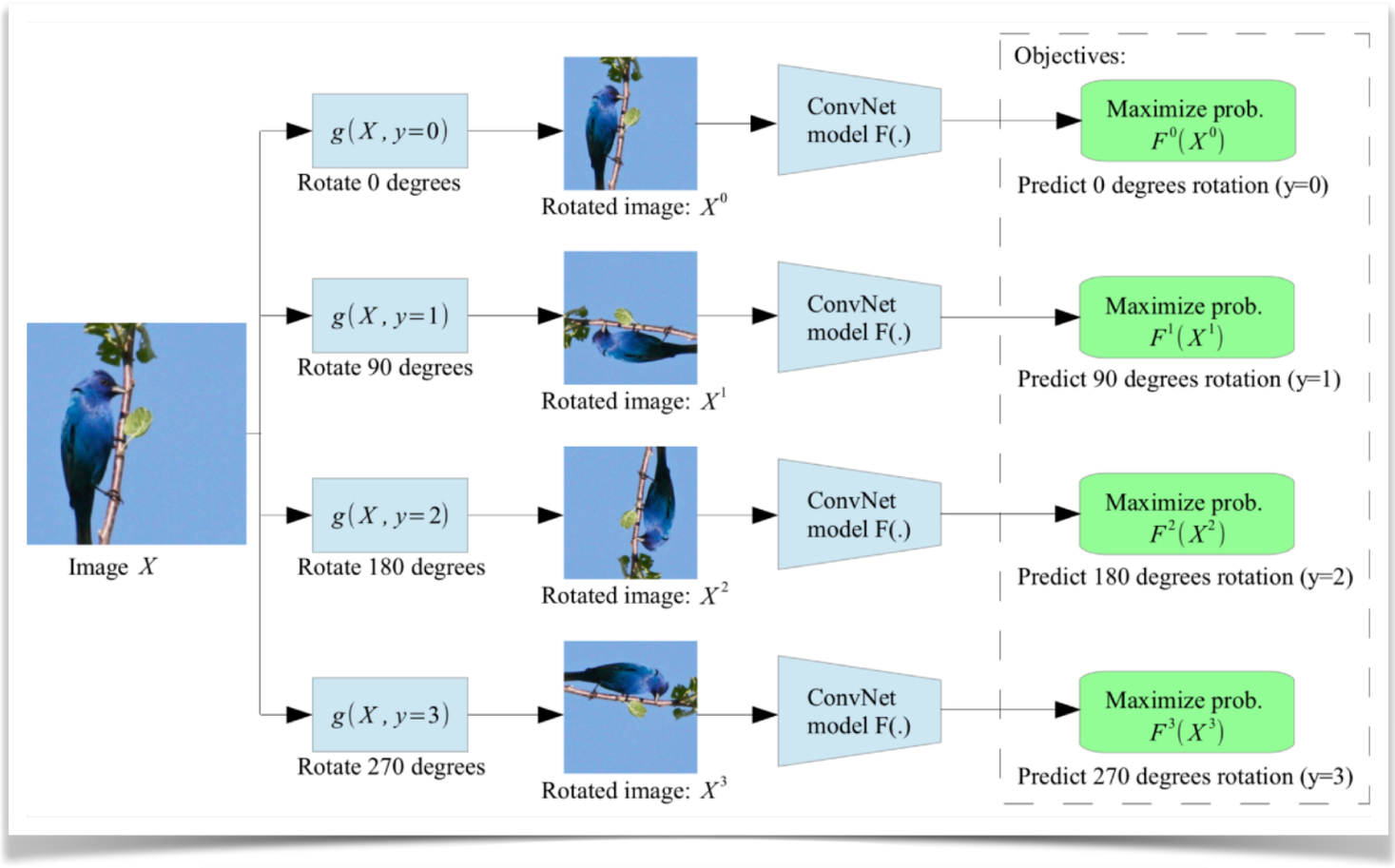


https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Rotation

Each input image is first rotated by a multiple of 90 degrees at random, corresponding to [0,90,180,270]. The model is trained to predict which rotation has been applied, thus a 4-class classification problem.



In order to identify the same image with different rotations, **the model has to learn to recognize high level object parts**, such as heads, noses, and eyes, and the relative positions of these parts, rather than local patterns.

https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Patches

> Predicting the **relative position** between two random patches from one image. A model needs to understand the spatial context of objects in order to tell the relative position between parts.

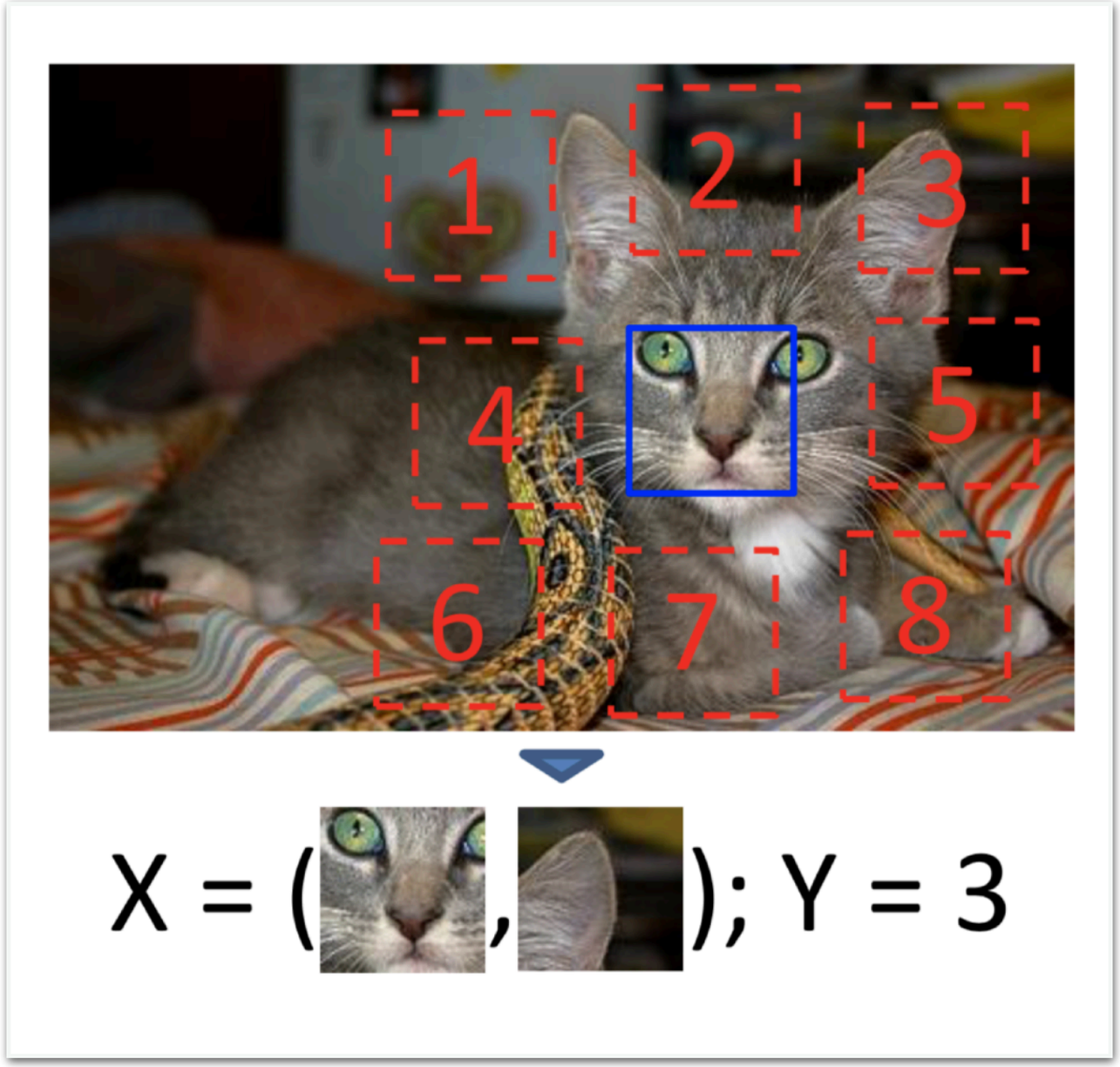
The training patches are sampled in the following way:

1. Randomly sample the first patch without any reference to image content.
2. Considering that the first patch is placed in the middle of a 3x3 grid, and the second patch is sampled from its 8 neighboring locations around it.
3. To avoid the model only catching **low-level trivial signals**, such as connecting a straight line across boundary or matching local patterns, additional noise is introduced by:
    - Add gaps between patches
    - Small jitters
    - Randomly downsample some patches to as little as 100 total pixels, and then upsampling it, to build robustness to pixelation.
    - **Shift green and magenta toward gray or randomly drop 2 of 3 color channels.** Chromatic Aberration

https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Patches



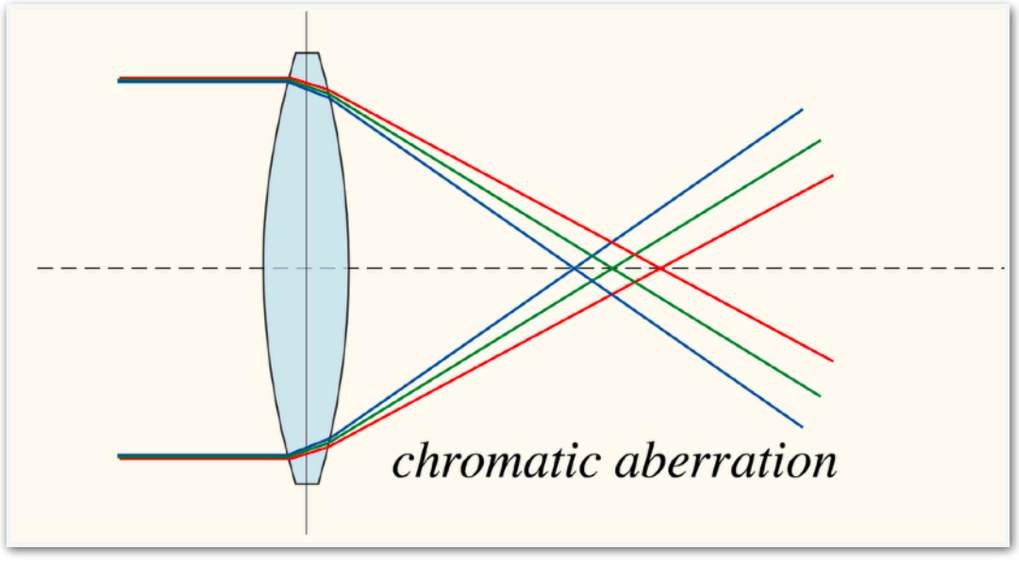
https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Chromatic Aberration

Other than trivial signals like boundary patterns or textures continuing, another interesting and a bit surprising trivial solution was found, called “chromatic aberration”.

It is triggered by different focal lengths of lights at different wavelengths passing through the lens. In the process, there might exist small offsets between color channels. Hence, the model can learn to tell the relative position by simply comparing how green and magenta are separated differently in two patches. This is a trivial solution and has nothing to do with the image content.



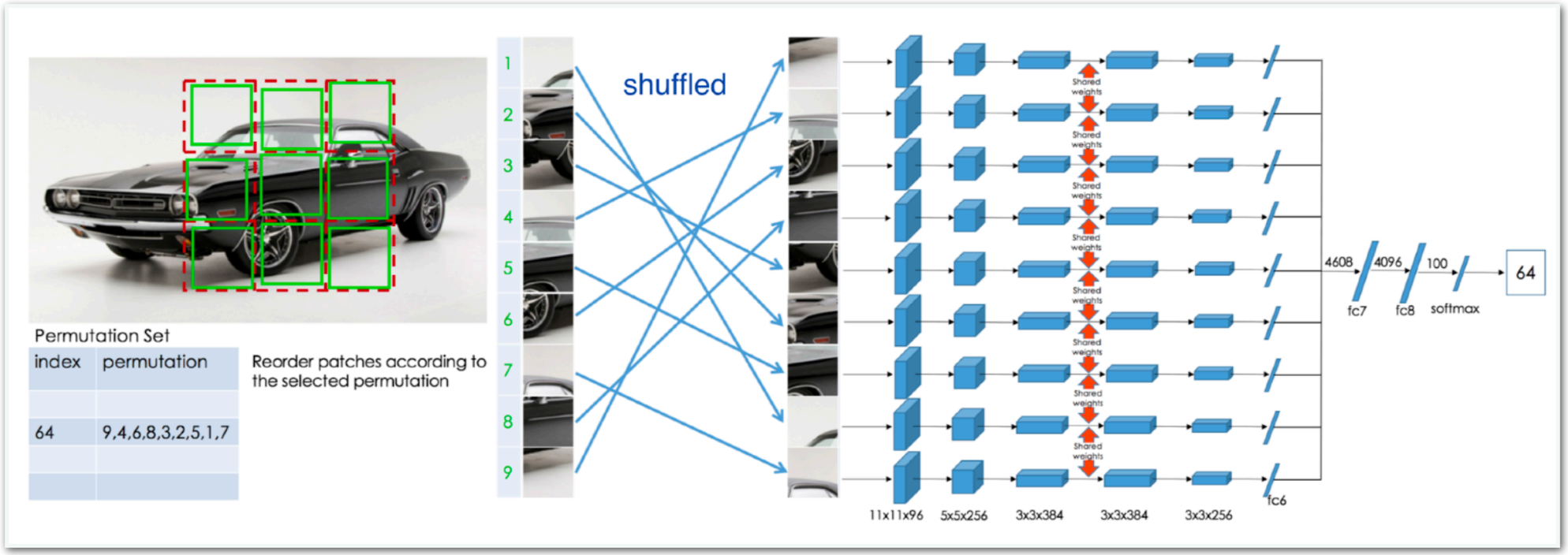
https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Jigsaw puzzle

The model is trained to place 9 shuffled patches back to the original locations.

A convolutional network processes each patch independently with shared weights and outputs a probability vector per patch index out of a predefined set of permutations. To control the difficulty of jigsaw puzzles, the paper proposed to shuffle patches according to a predefined permutation set and configured the model to predict a probability vector over all the indices in the set.

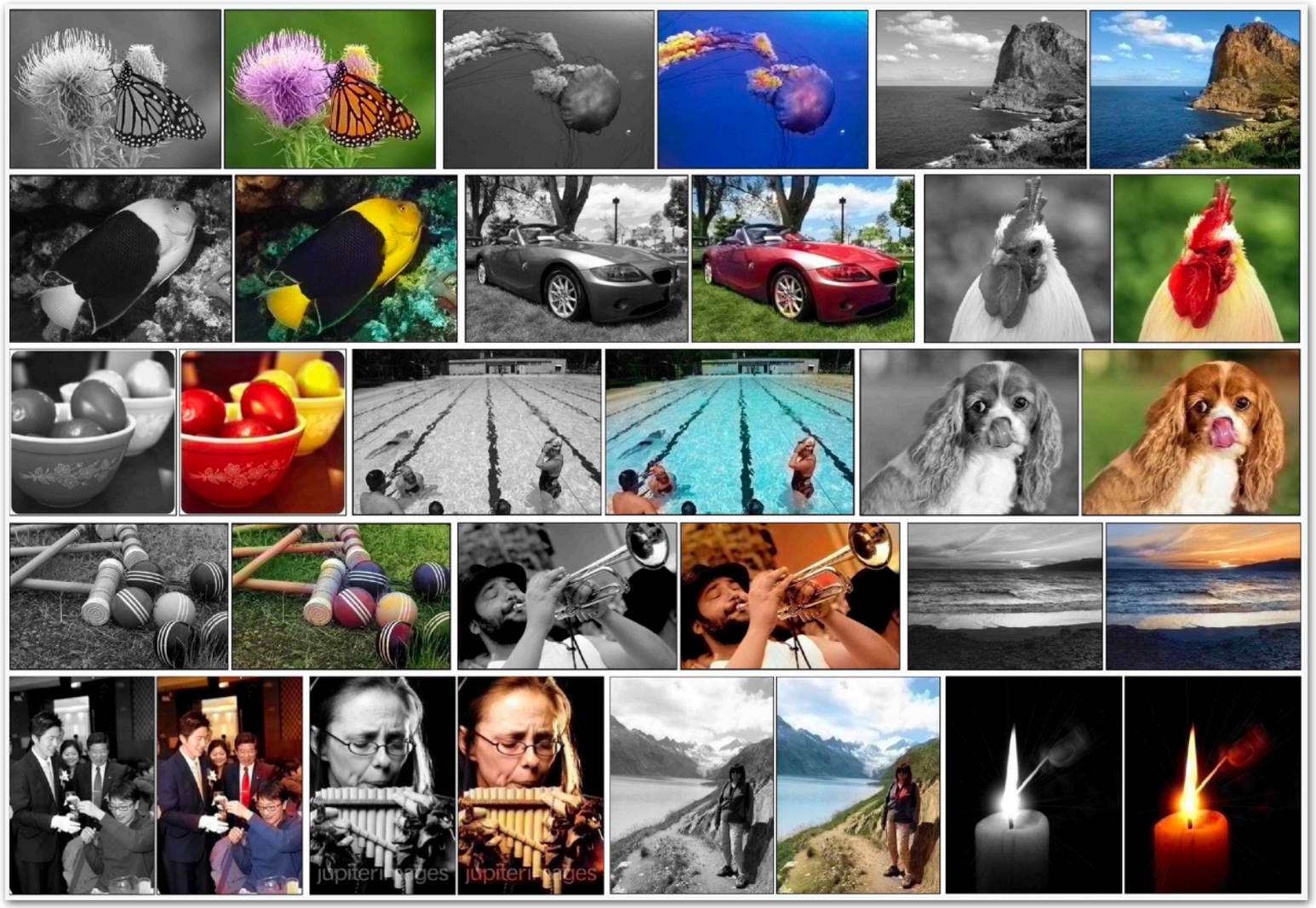


https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Image-based Self-Supervised Learning

Colorization

Colorization can be used as a powerful self-supervised task: a model is trained to color a grayscale input image; precisely the task is to map this image to a distribution over quantized color value outputs



Contrastive Methods

Contrastive methods, as the name implies, learn **representations** by contrasting positive and negative examples.

- Contrastive methods trained on unlabelled image data and evaluated with a linear classifier now surpass the accuracy of supervised AlexNet.
- Contrastive pre-training on images successfully transfers to other downstream tasks and outperforms the supervised pre-training counterparts.

More formally, for any data point x, contrastive methods aim to learn an **encoder** f such that:

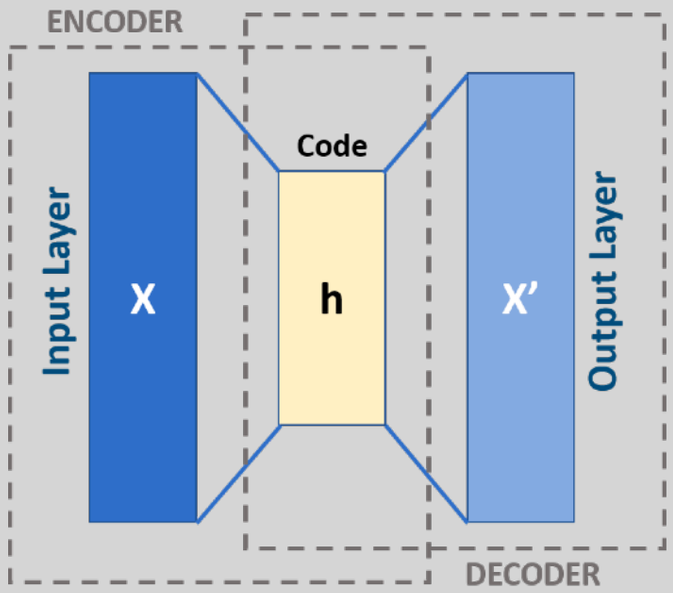
$$score(f(x), f(x^+)) >> score(f(x), f(x^-))$$

- here x^+ is data point similar or congruent to x, referred to as a *positive* sample.
- x^- is a data point dissimilar to x, referred to as a *negative* sample.
- the score function is a metric that measures the similarity between two features.

x is commonly referred to as an "**anchor**" data point.

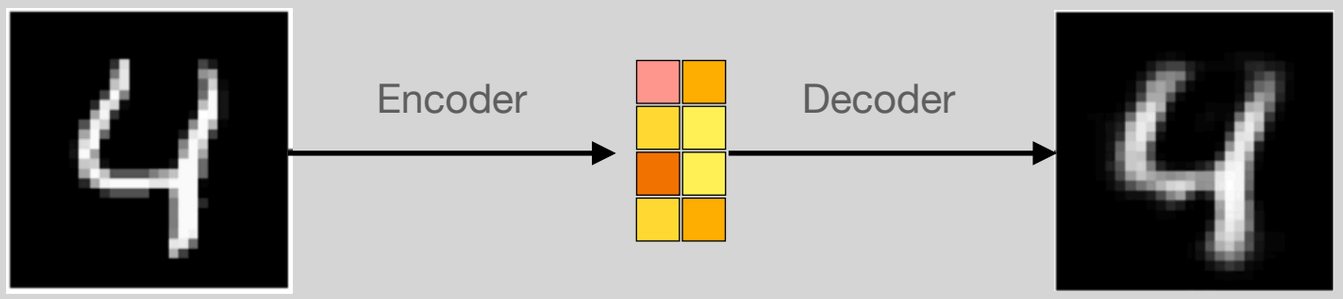
What is an encoder?

An **autoencoder** is a neural network that learns to copy its input to its output. It has an internal (*hidden*) layer that describes a **code** used to represent the input, and it is constituted by two main parts: an **encoder** that maps the input into the code, and a decoder that maps the code to a reconstruction of the original input.



To build an autoencoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target.

https://en.wikipedia.org/wiki/Autoencoder



What is an encoder?

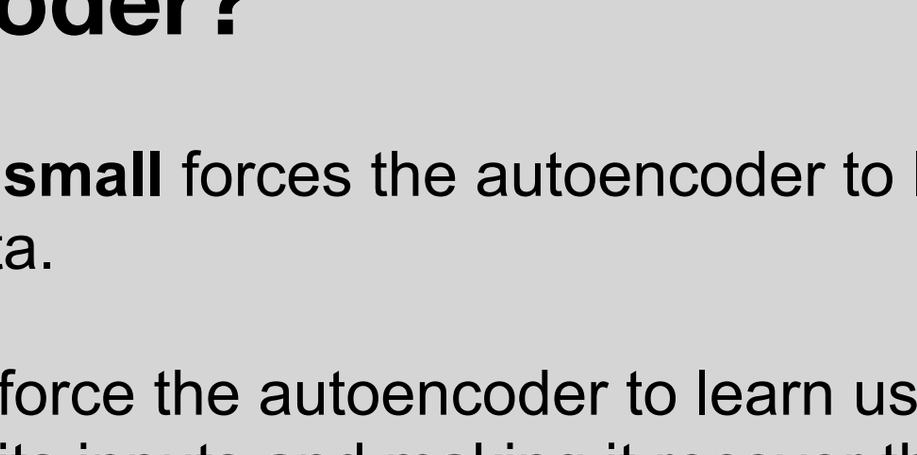
```
input_img = tf.keras.layers.Input(shape=(28, 28, 1))

x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
x = tf.keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)
x = tf.keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = tf.keras.layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = tf.keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
decoded = tf.keras.layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = tf.keras.models.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

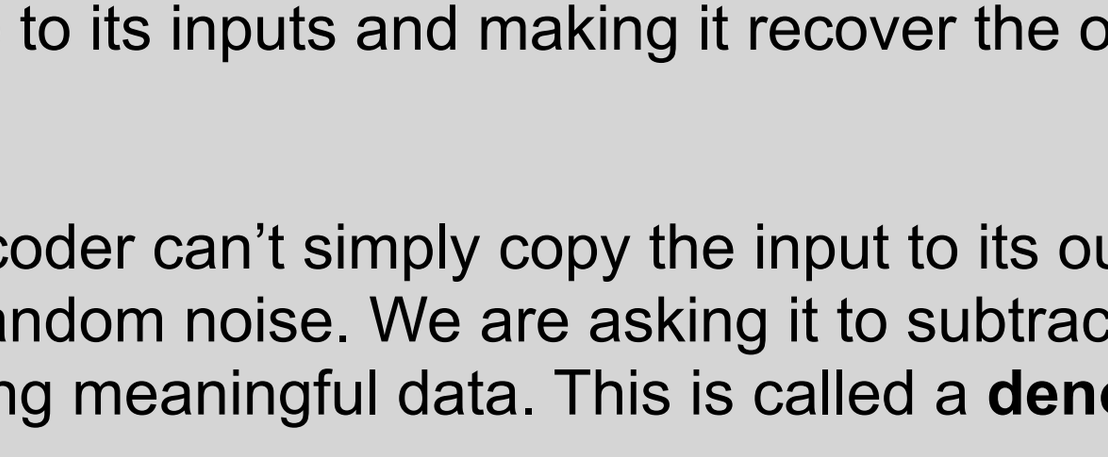
What is an encoder?

Keeping the **code layer small** forces the autoencoder to learn an intelligent representation of the data.

There is another way to force the autoencoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data.

This way the autoencoder can’t simply copy the input to its output because the input also contains random noise. We are asking it to subtract the noise and produce the underlying meaningful data. This is called a **denoising autoencoder**.





https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Contrastive Methods

$$score(f(x), f(x^+)) >> score(f(x), f(x^-))$$

To optimize for this property, we can construct a softmax classifier that classifies positive and negative samples correctly.

This should encourage the score function to assign large values to positive examples and small values to negative examples:

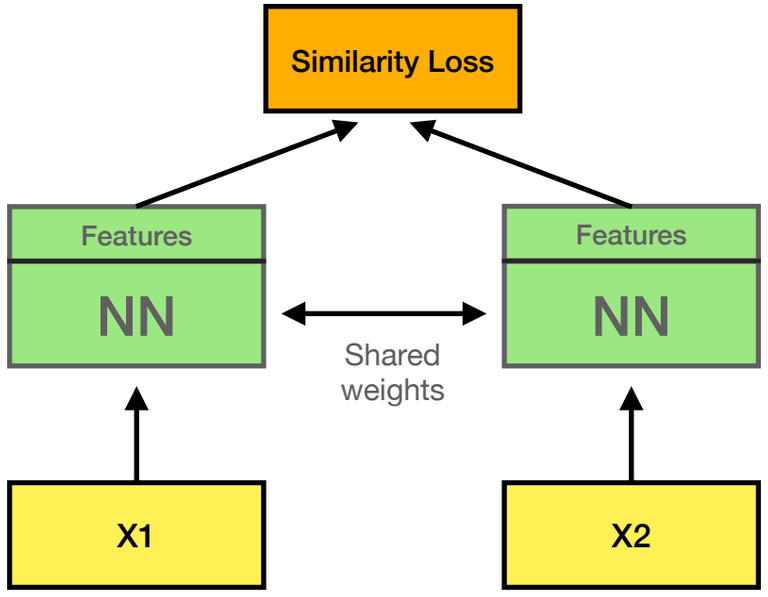
Here, we are using the dot product or cosine distance as the score function.

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

The denominator terms consist of one positive, and *N*–1 negative samples.

Contrastive Loss

One of the firsts contrastive models were **siamese networks**.

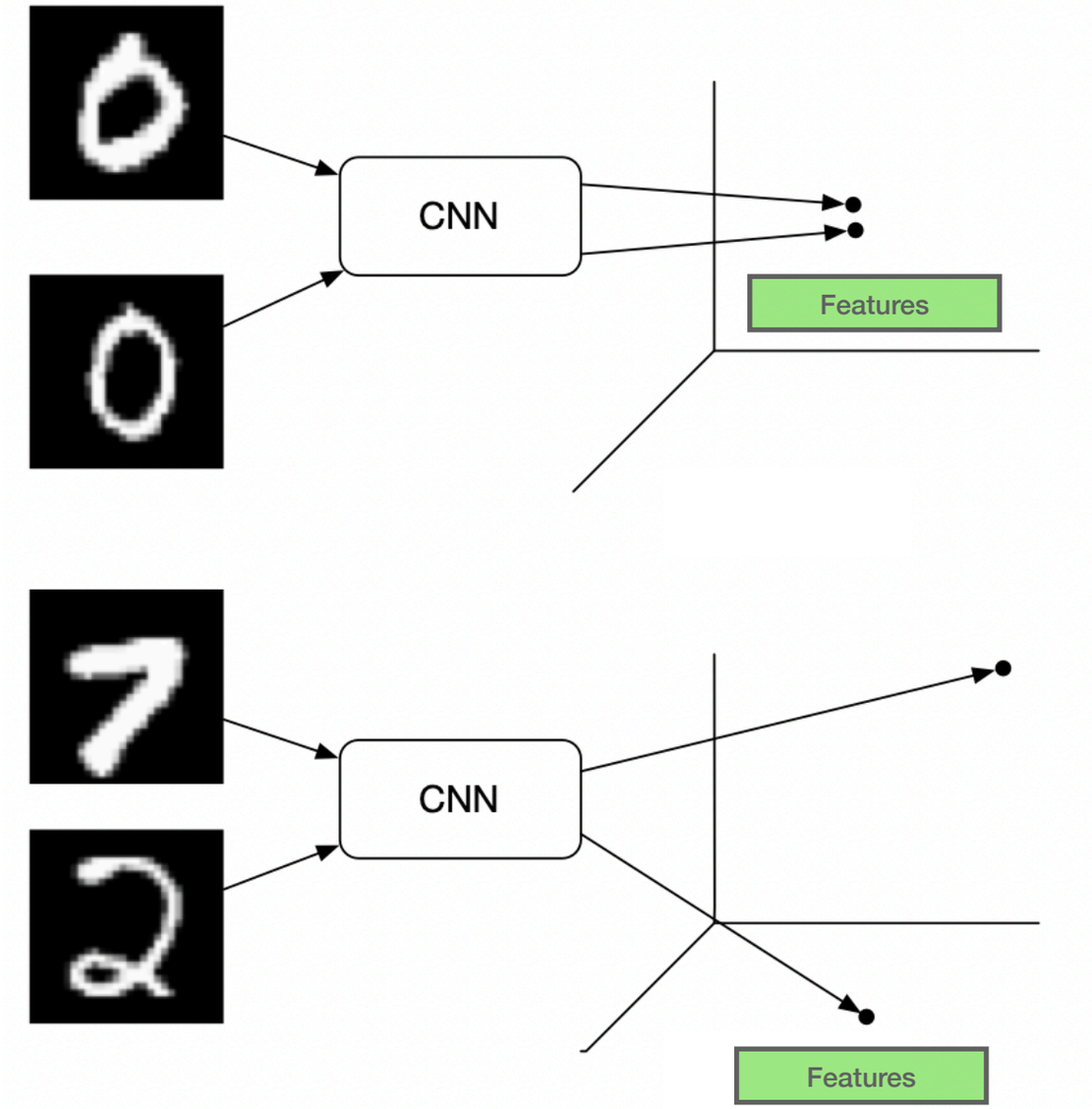


A siamese network is often shown as two different encoding networks that share weights, but in reality the same network is just used twice before doing backpropagation.

When training a siamese network, 2 or more inputs are encoded and the output features are compared. This comparison can be done in a number of ways. Some of the comparisons are triplet loss and contrastive loss. They used the **pairwise ranking loss**:

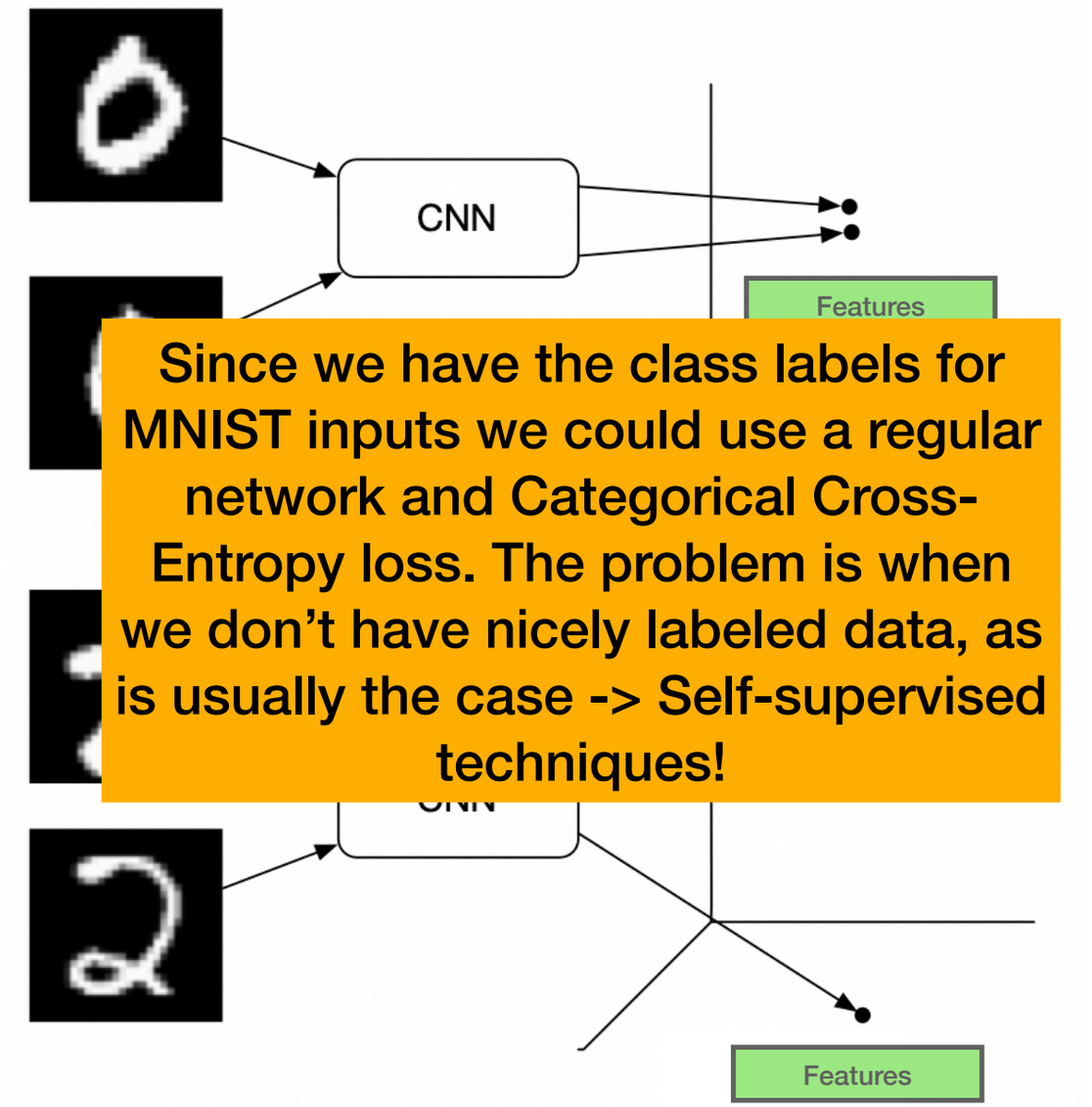
$$L(x_0, x_1, y) = y \left\| f(x_0) - f(x_1) \right\| + (1 - y) max(0, m - \left\| f(x_0) - f(x_1) \right\|)$$

Contrastive Loss



Let’s look at an example where we want to make features out of MNIST numbers.
Each image of an MNIST number should encode into a vector that is close to vectors from images of the same class. Conversely different numbers should encode into vectors that are far from each other.

Contrastive Loss



Let’s look at an example where we want to make features out of MNIST numbers.
Each image of an MNIST number should encode into a vector that is close to vectors from images of the same class. Conversely different numbers should encode into vectors that are far from each other.