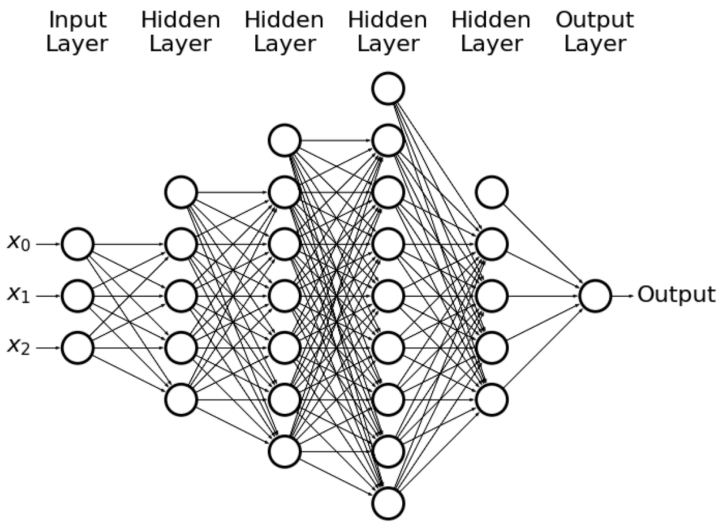
Convolutional neural networks

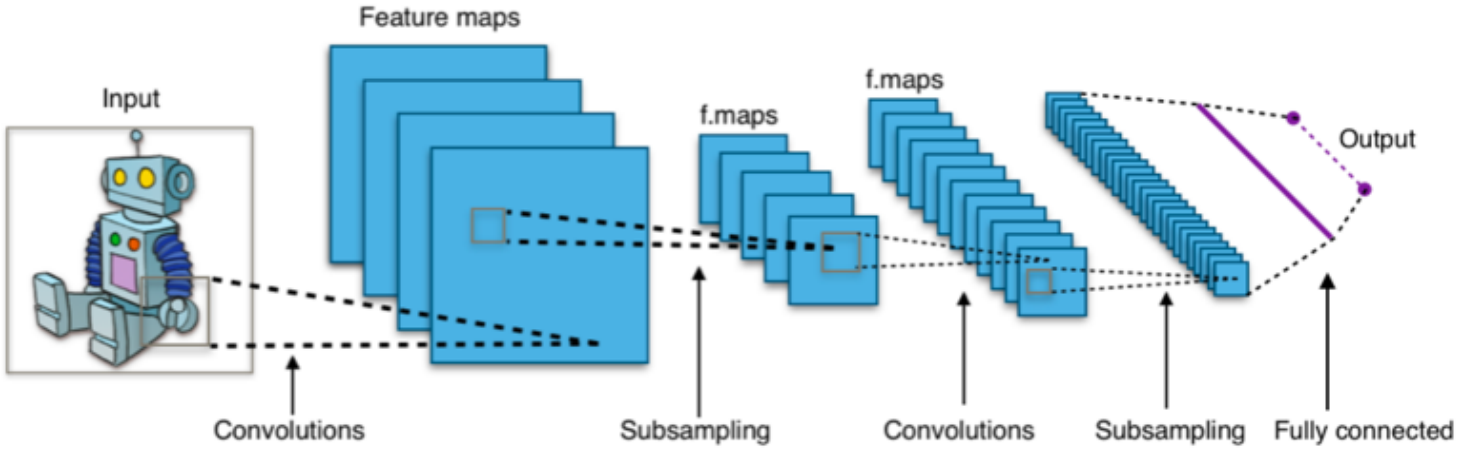
Materials for this section:

- https://cs231n.github.io/convolutional-networks
- simple review: https://arxiv.org/abs/1901.06032
- https://arxiv.org/pdf/1603.07285.pdf

Convolutional neural networks



vs.



- fully-connected multi-layer perceptron uses (weights) matrix **multiplication** when passing between layers

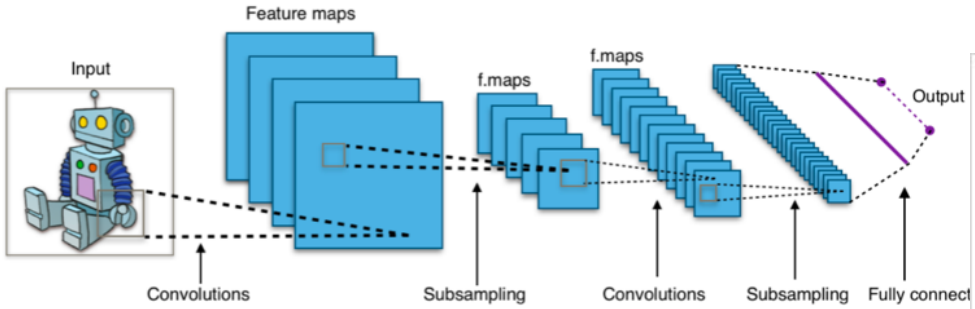
- *convolutional networks* use **convolution** operation in place of matrix multiplication in at least one of their layers
- other operation used: pooling (subsampling), dropout (regularization)

Convolutional neural network = ConvNet = CNN

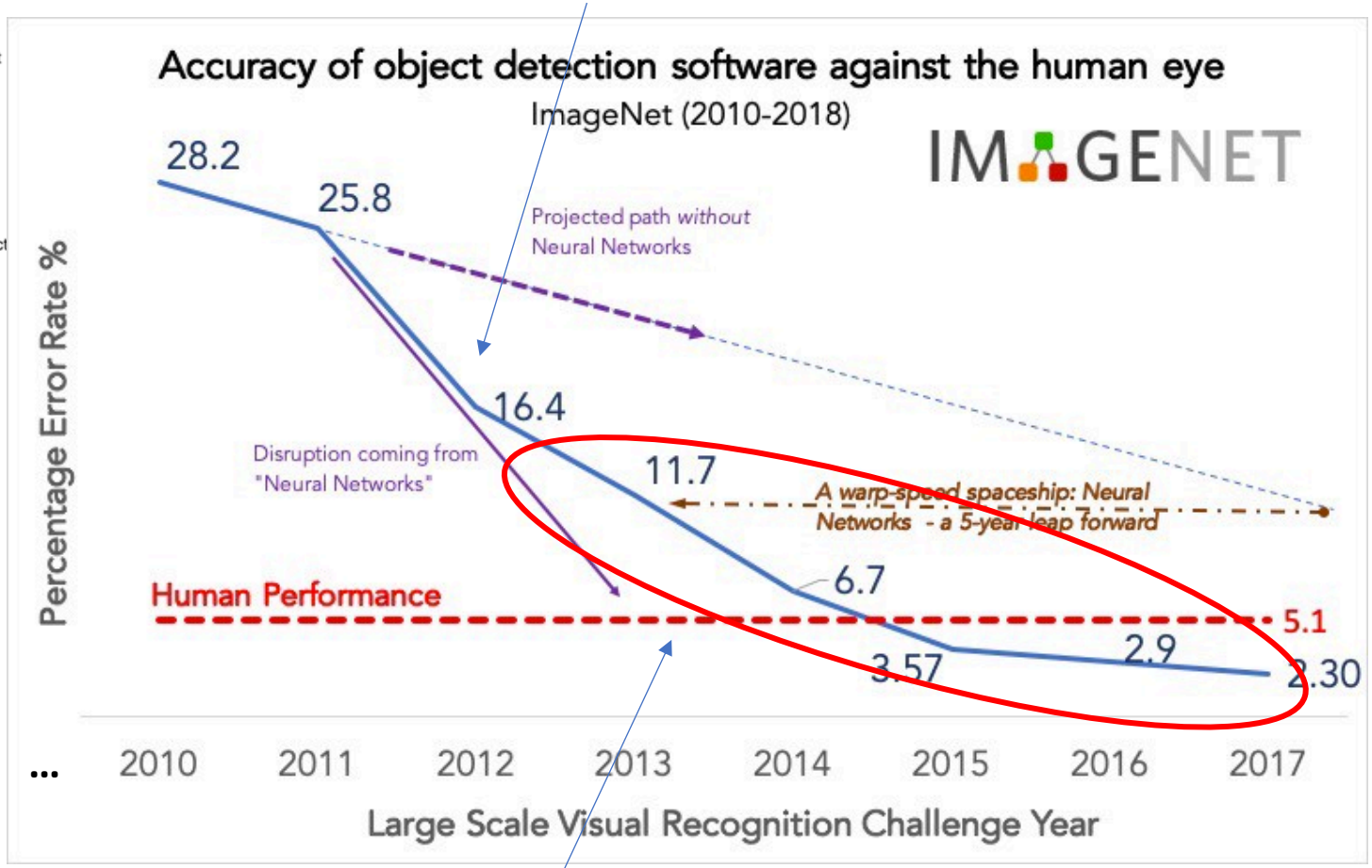
- CNNs start deep-learning revolution (AlexNet, 2012)
- widely used in *computer-vision*
- for processing data that has 1D, 2D, … grid topology

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

Convolutional neural networks



AlexNet CNN



- CNNs start deep-learning revolution: ImageNet competition
- are the most significant NN architecture type nowadays, also being a part of much complicated architectures

Yann LeCun (1989) used back-propagation to learn **shared** convolution kernel

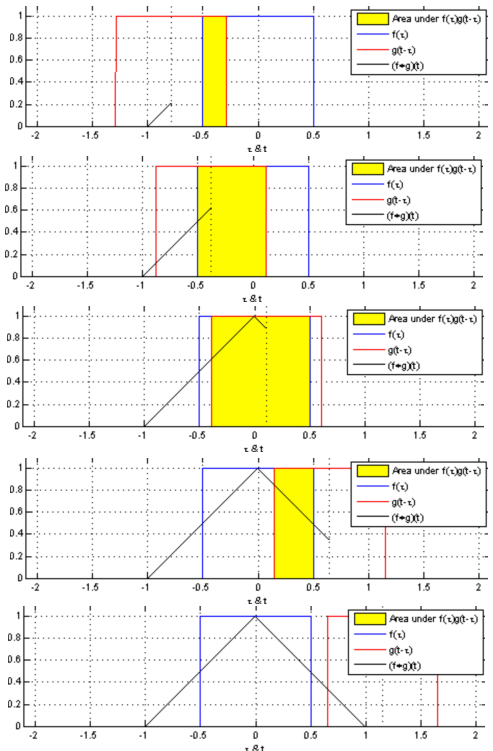
deep-learning era

http://image-net.org/

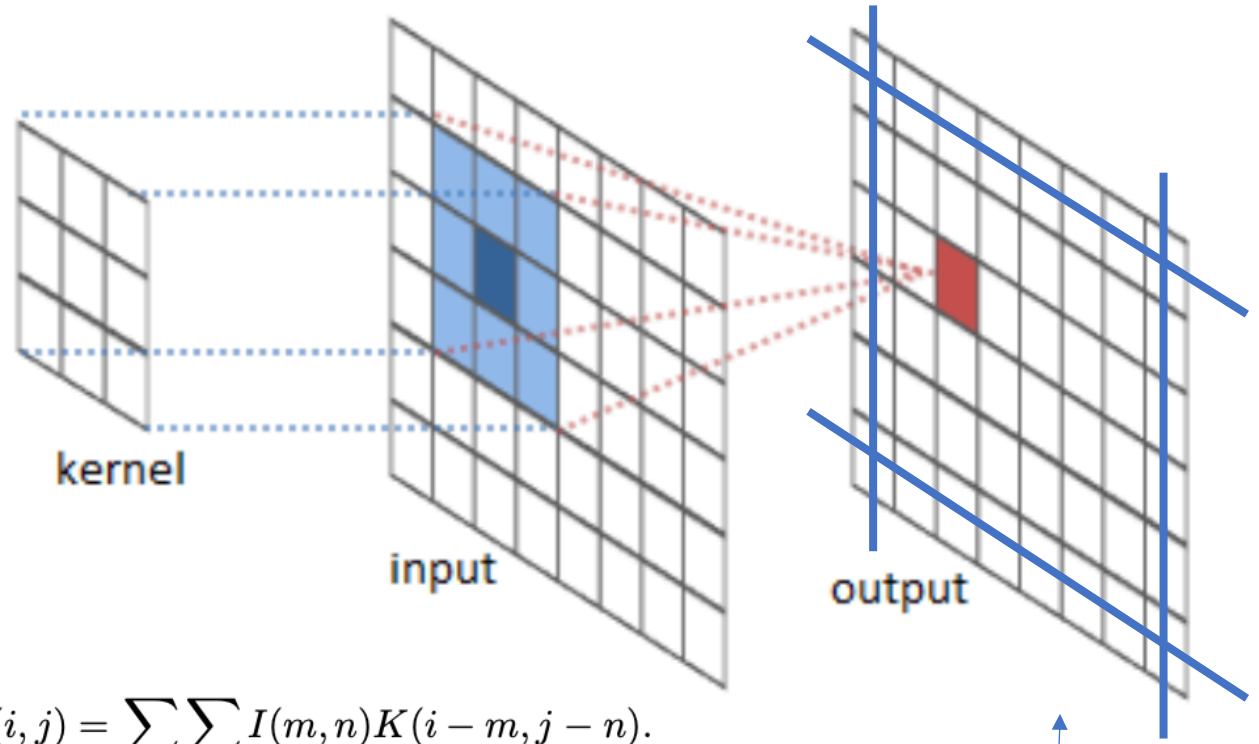
Convolutional neural networks

Convolution in an operation where we multiply input by moving weight (kernel) function an obtaining subsequent outputs

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$



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



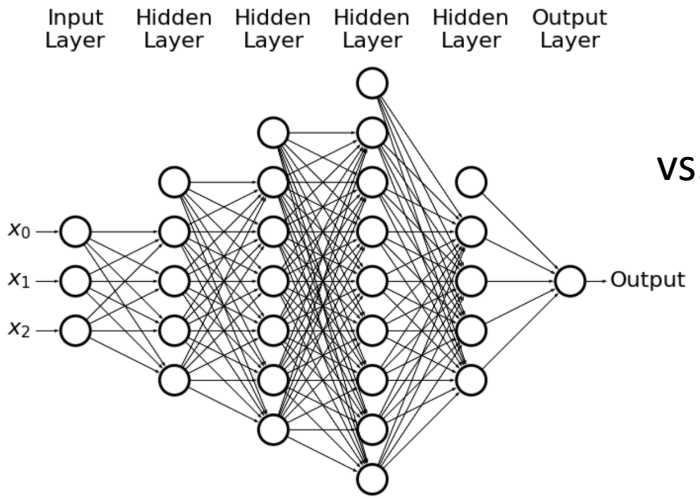
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

- To get 1d or 2d **discrete** convolutions we move kernel over input grid, e.g. image,
- kernel is usually much smaller, e.g. 3x3, than input image, e.g. 512x512 pixels,
- and obtain reduced side output (510x510 in our example).

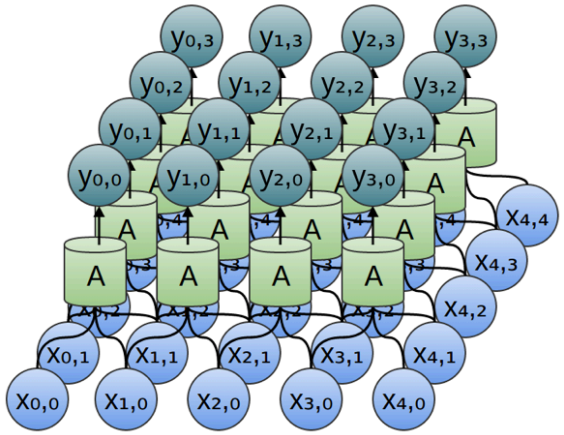
Convolutional neural networks

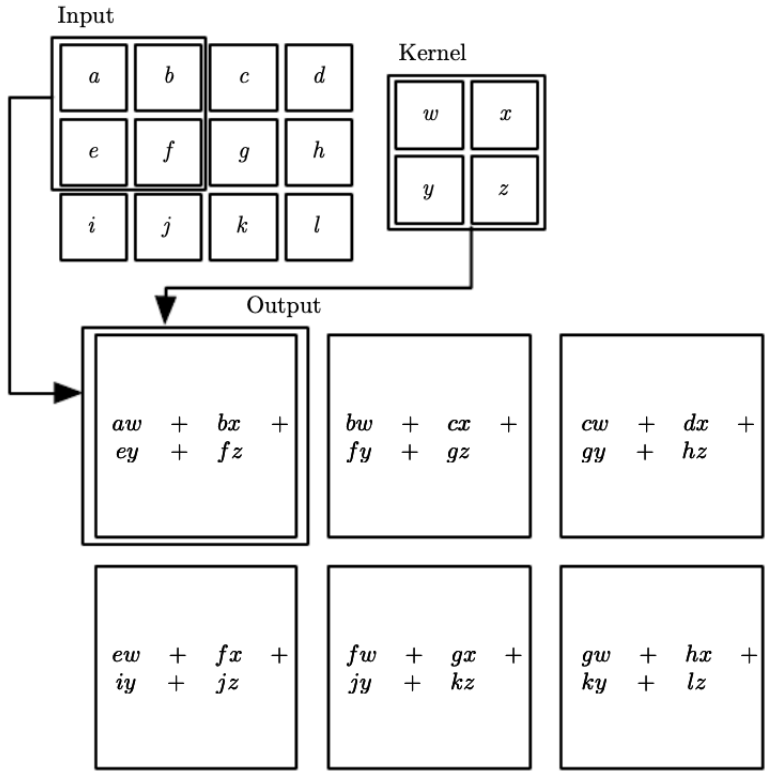
Idea why to use convolutions: Sparse-connectivity

- CNNs are regularized version of fully-connected MLPs
- A single element in the feature map is connected to only a small patch of pixels.
- This is very different from connecting to the whole input image, in the case of multi-layer perceptrons.



vs.





- CNNS are also much faster to train than fully MLPs.

Convolutional neural networks

- Please note, that the output size (W_{out} x H_{out}) is determined by the input size (W_{in} x H_{in}) and the kernel size (W_f x H_f):

$$W_{out} = W_{in} - W_f + 1$$
$$H_{out} = H_{in} - H_f + 1$$

Padding

- Padding (zero padding) - adding zeros around an image

```
                | 0 0 0 0 0 0 0 |
| 0 1 0 1 1 |   | 0 0 1 0 1 1 0 |
| 1 1 0 1 0 |   | 0 1 1 0 1 0 0 |
| 1 0 1 0 1 | -> | 0 1 0 1 0 1 0 |
| 1 1 0 1 0 |   | 0 1 1 0 1 0 0 |
| 0 0 1 1 0 |   | 0 0 0 1 1 0 0 |
                | 0 0 0 0 0 0 0 |
```

- The output size (W_{out} x H_{out}) is determined by the input size (W_{in} x H_{in}), the kernel size (W_f x H_f), and the padding size (P)

$

$$W_{out} = W_{in} - W_f + 2P + 1$$
$$H_{out} = H_{in} - H_f + 2P + 1$$

Sometimes downsizing is unwanted, and information on the edges is of the same importance

But sometimes downsampling is desired (e.g. too high resolution)

Stride

- Stride controls the movement of the filter

```
Stride = 1

| [0 1 0] 1 1 |   | 0 [1 0 1] 1 |
| [1 1 0] 1 0 |   | 1 [1 0 1] 0 |
| [1 0 1] 0 1 | -> | 1 [0 1 0] 1 | -> ...
|  1 1 0  1 0 |   | 1  1 0 1  0 |
|  0 0 1  1 0 |   | 0  0 1 1  0 |

Stride = 2

| [0 1 0] 1 1 |   | 0 1 [0 1 1] |
| [1 1 0] 1 0 |   | 1 1 [0 1 0] |
| [1 0 1] 0 1 | -> | 1 0 [1 0 1] | -> ...
|  1 1 0  1 0 |   | 1 1  0 1 0  |
|  0 0 1  1 0 |   | 0 0  1 1 0  |
```

- The output size (W_{out} x H_{out}) is determined by the input size (W_{in} x H_{in}), the kernel size (W_f x H_f), the padding size (P), and the stride size (W_s x H_s)

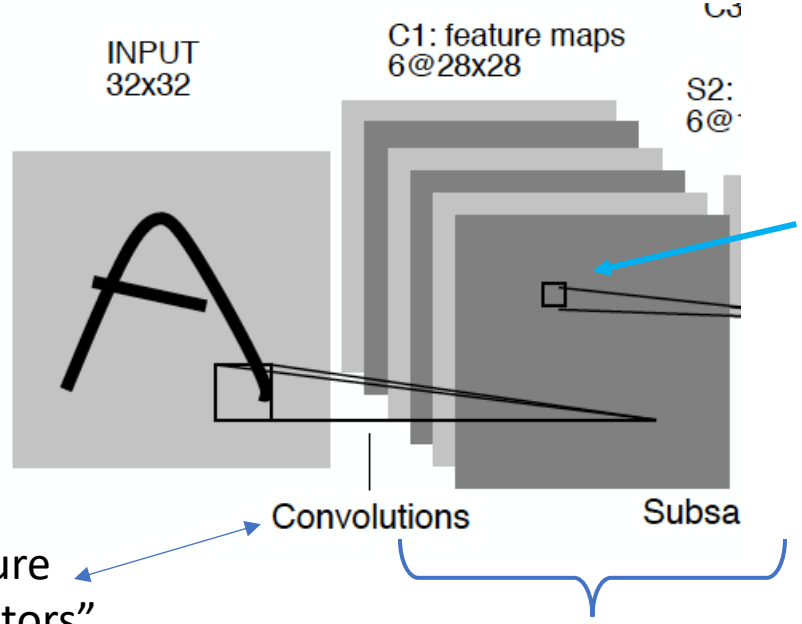
$$W_{out} = \frac{W_{in} - W_f + 2P}{W_s} + 1$$
$$H_{out} = \frac{H_{in} - H_f + 2P}{H_s} + 1$$

source: tomaszgolan.github.io/introduction_to_machine_learning

CNNs: feature maps

The idea is to build hierarchy of concepts using multiple layers each consisting feature maps:

- we need to extract **features** from images, time series, etc.
- each kernel extract different feature, producing so-called *feature map*



Points on given map shares the **same** kernel

feature "extractors"

multiple feature maps, each with its own kernel forms single layer in CNN

example kernels

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

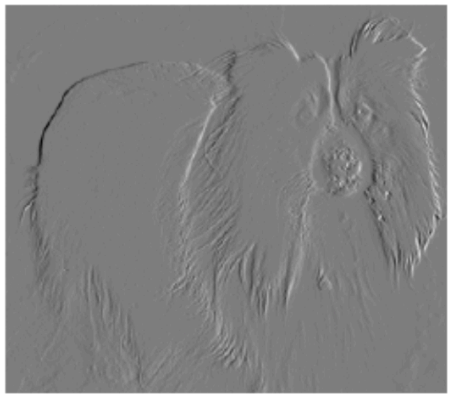
3 x 3

=

-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

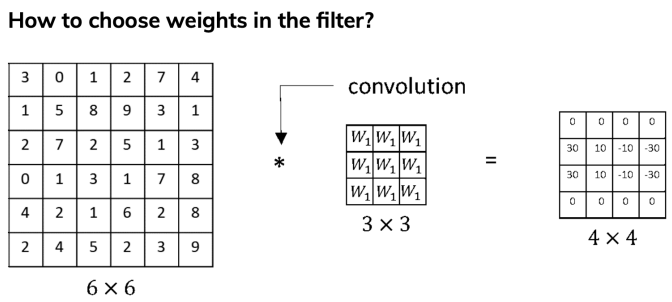
4 x 4

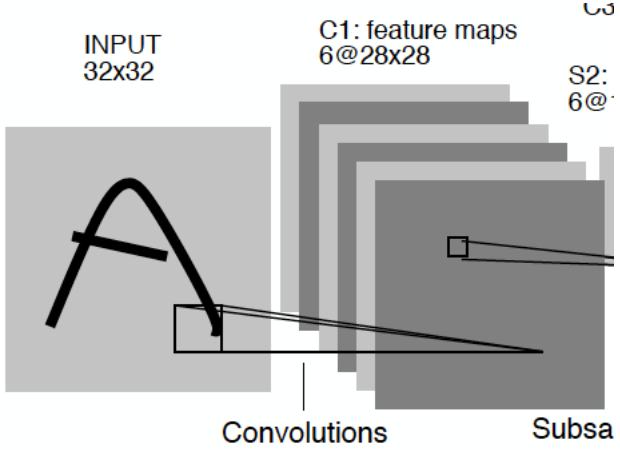
e.g. vertical edge detection kernel



CNNs: learning kernels

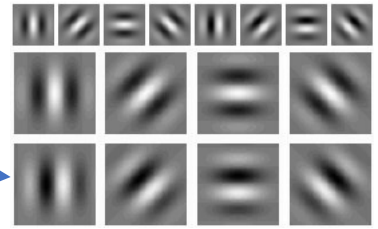
The coolest thing is that network learns kernel types automatically by itself!

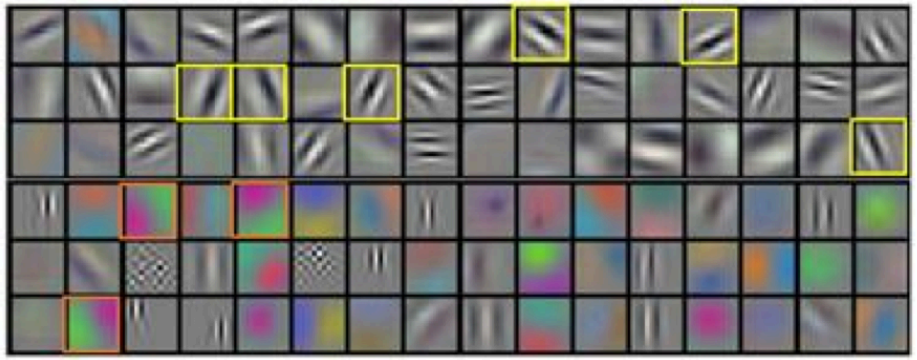


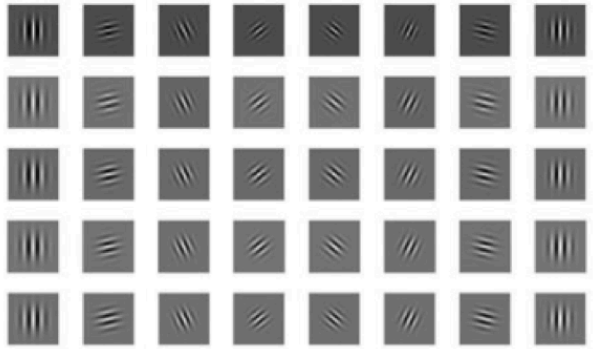


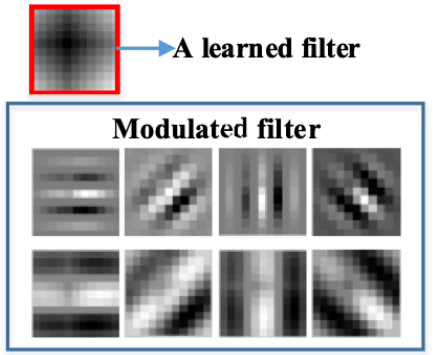
CNNs: Feature extractors using kernels learned by itself

AlexNet learnt by itself filters that resemble *Gabor filters* known from classical image processing, https://arxiv.org/pdf/1705.01450.pdf



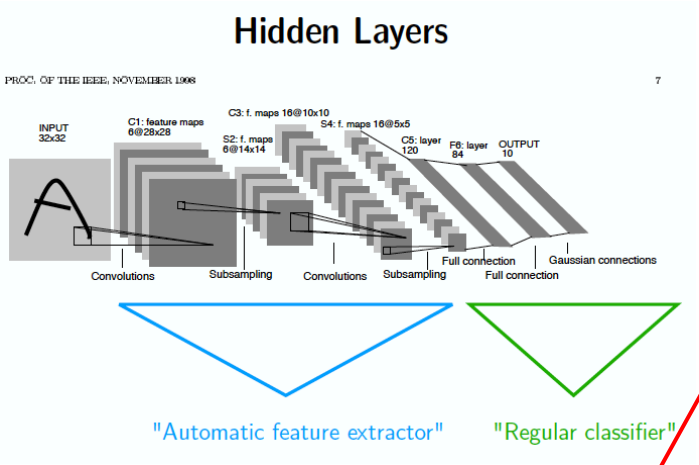






CNN full architecture

Let's go deeper into full CNN



- feature maps in between neighbor convolutional layers are usually fully connected
- feature map is a basic unit of CNNs

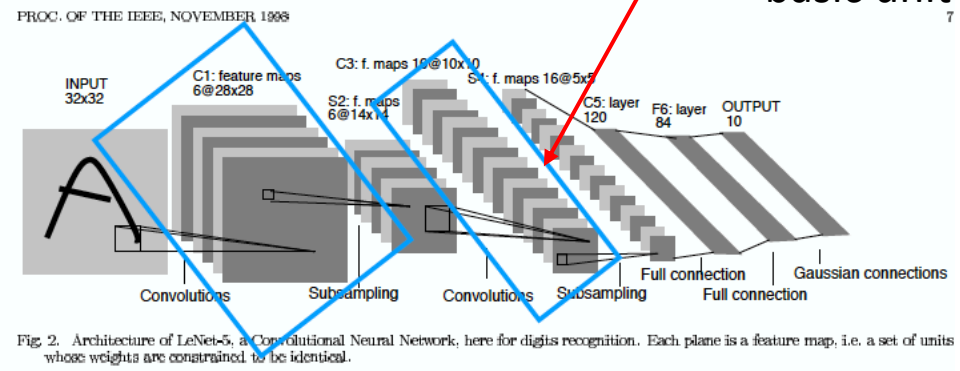


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Each "bunch" of feature maps represents one hidden layer in the neural network.

Convolutional Neural Networks

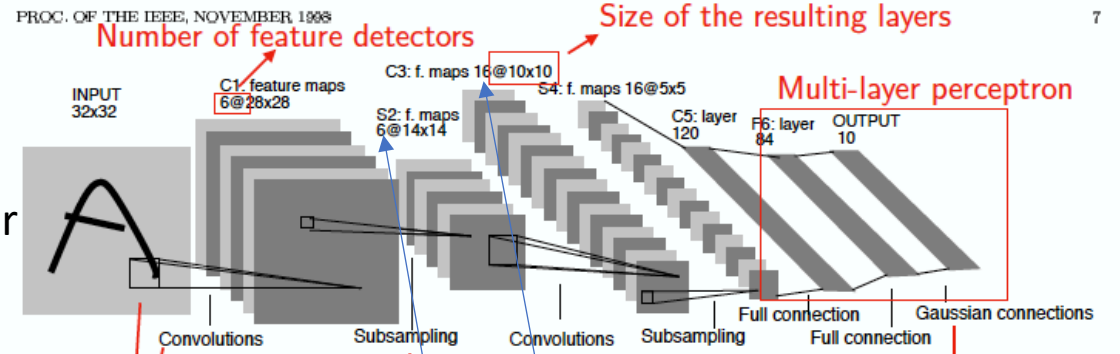


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

nowadays called "pooling"

basically a fully-connected layer + MSE loss (nowadays better to use fc-layer + softmax + cross entropy

"Feature detectors" (weight matrices) that are being reused ("weight sharing") => also called "kernel" or "filter"

CONV2D

N *M*

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

*N*M* kernels

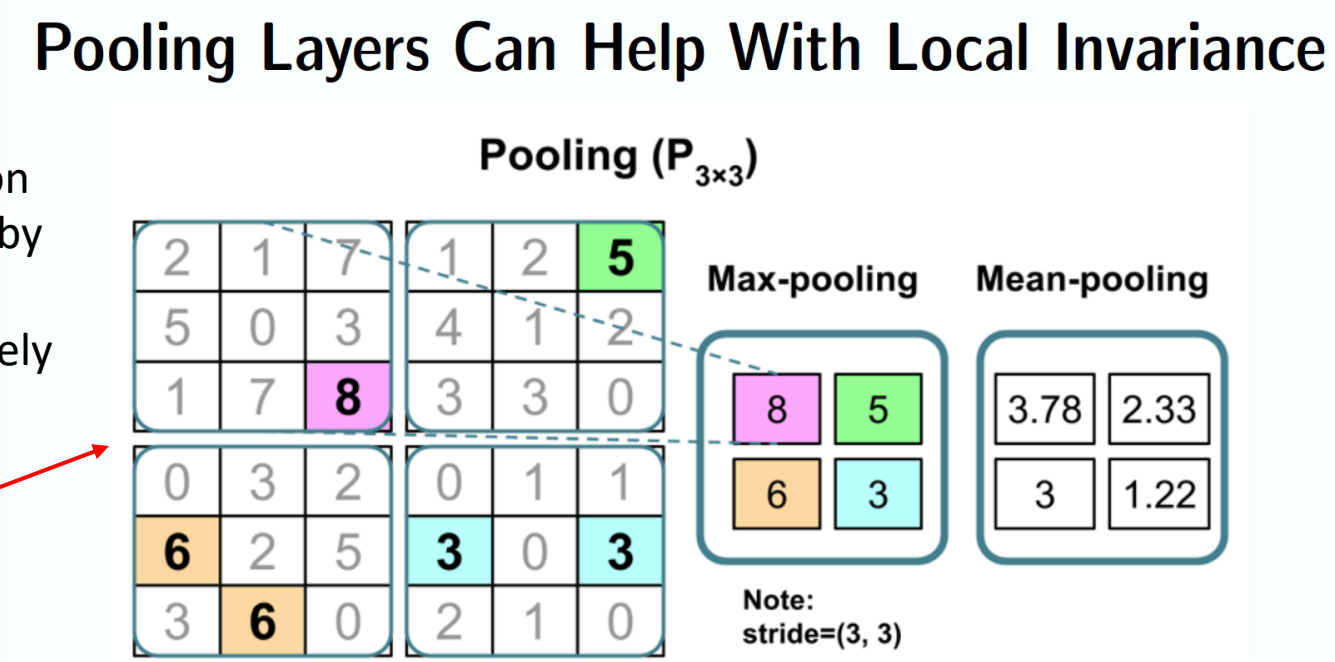
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

https://github.com/rasbt/stat453-deep-learning-ss20

CNN components: pooling

Pooling operation

- A pooling function replaces the given output activation map with map composed of **local** statistic of the nearby values
- Pooling helps to make the representation approximately **invariant** to small translations of the input.



e.g. here pool size is of 3x3 and stride between pools equals 3

The pooling layer reduces the number of parameters and calculations in the network. This improves the efficiency of the network and avoids overfitting.

Note that typically pooling layers do not have any learnable parameters

CNN components: dropout and batch-normalization

Another type of regularizations used in CNNs is **dropout**:

- frequently we apply dropout to fully-connected-layers part
- but sometimes also after pooling layers

Moreover, to stabilize and speed-up the training process we apply **batch-normalization** layers:

- the idea is to make activations units (in the given layer) gaussian-like by scaling their input x_i (from earlier layer)
- normalization is performed using averages for entire mini-batch: mini-batch mean and mini-batch variance (N is a batch size):

The mean and the variance are calculated:
- for given mini-batch (when training)
- for the whole training dataset (inference)

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

- and normalize inputs

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

each input x_i contains
- activations (MLP)
- or complete set of feature maps (conv layers)

from previous layer

Note1: SGD (training) will still work since $\hat{x}_i$ is a differentiable function
Note2: when training CNNs we process whole mini-batch at once
(input tensor is of size $N \times C \times W \times H$, for batch of N color images each of $W \times H$ size)

Labs

Autoencoders and GANs

Materials for this section:

- https://www.deeplearningbook.org/contents/autoencoders.html
- https://www.youtube.com/watch?v=BUNl0To1IVw
- https://arxiv.org/pdf/1703.10593.pdf
- https://arxiv.org/pdf/1611.07004.pdf

Autoencoders

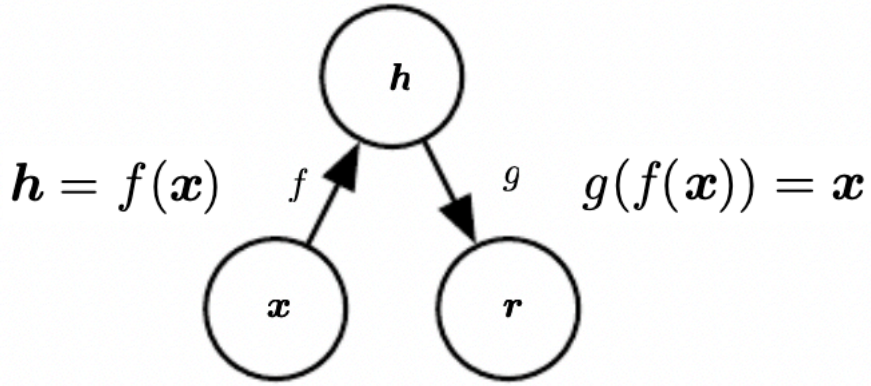
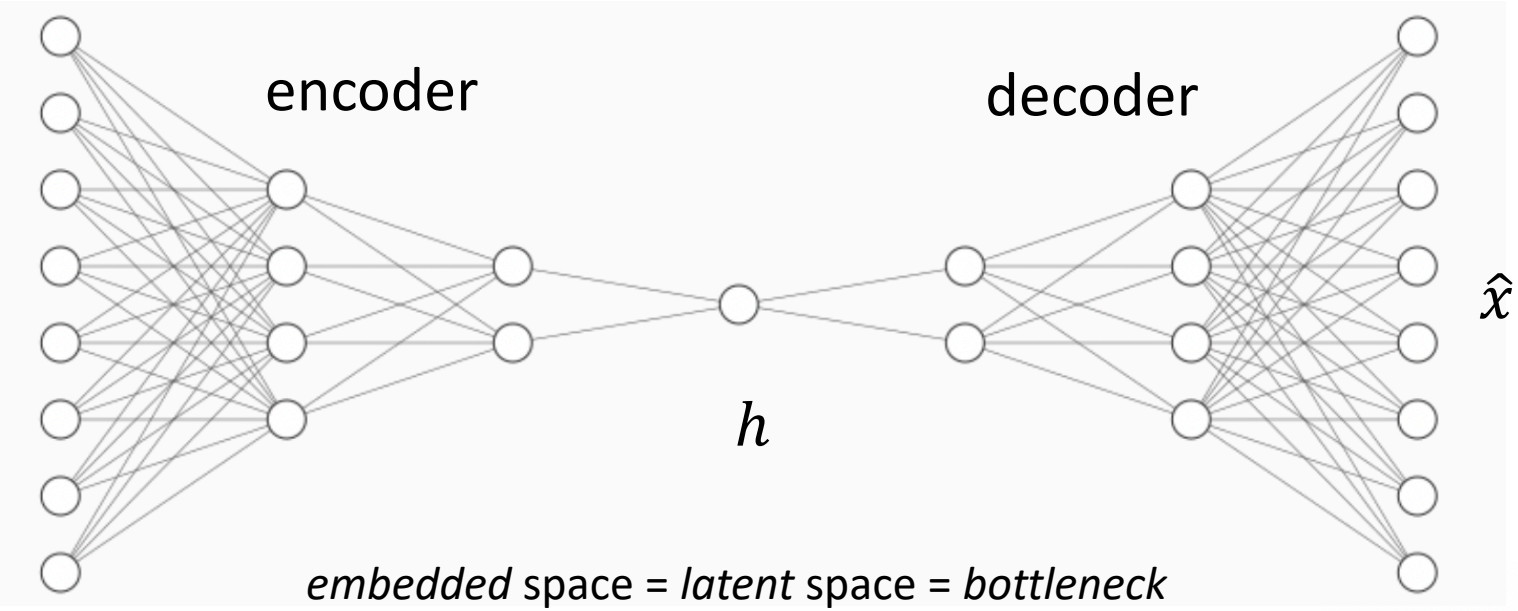


Figure 14.1: The general structure of an autoencoder, mapping an input $\boldsymbol{x}$ to an output (called reconstruction) $\boldsymbol{r}$ through an internal representation or code $\boldsymbol{h}$. The autoencoder has two components: the encoder f (mapping $\boldsymbol{x}$ to $\boldsymbol{h}$) and the decoder g (mapping $\boldsymbol{h}$ to $\boldsymbol{r}$).

- typically dim(h) << dim(x)
- autoencoders are unsupervised learning models

www.deeplearningbook.org/contents/autoencoders.html

Autoencoders



embedded space = *latent* space = *bottleneck*

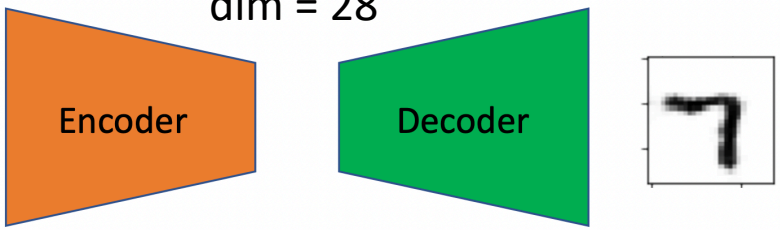
Typically, *latent space* h have a smaller dimension than input x

- An autoencoder whose code dimension is less than the input dimension is called **undercomplete**

$$\mathcal{L}(x, \hat{x}) = \frac{1}{N} \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2$$

input: 28x28

latent space
dim = 28

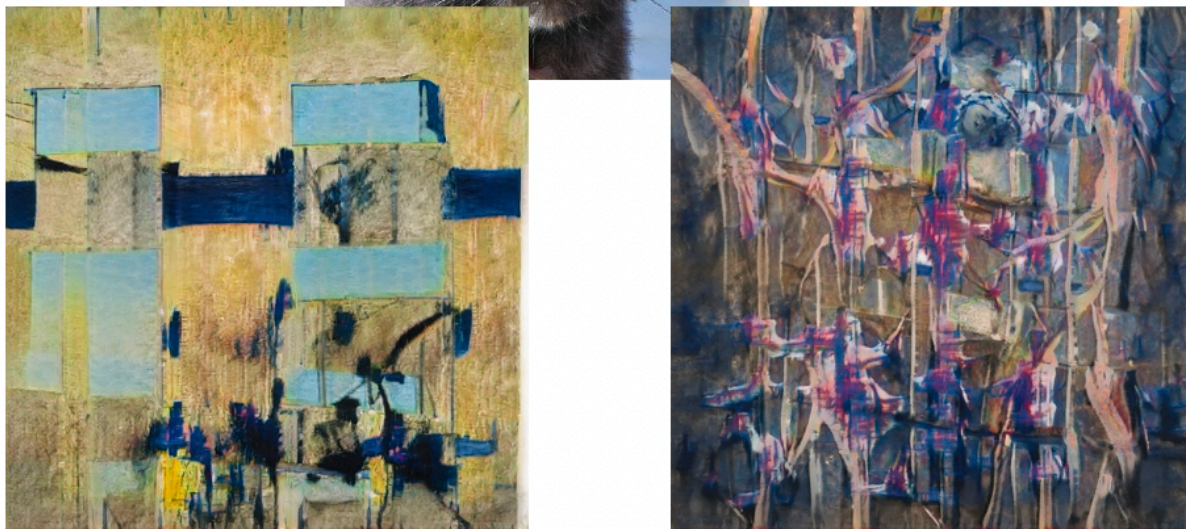


sebastianraschka.com/pdf/lecture-notes/stat479ss19/L15_autoencoders.pdf

Generative adversarial networks

https://thiscatdoesnotexist.com

https://thisartworkdoesnotexist.com/

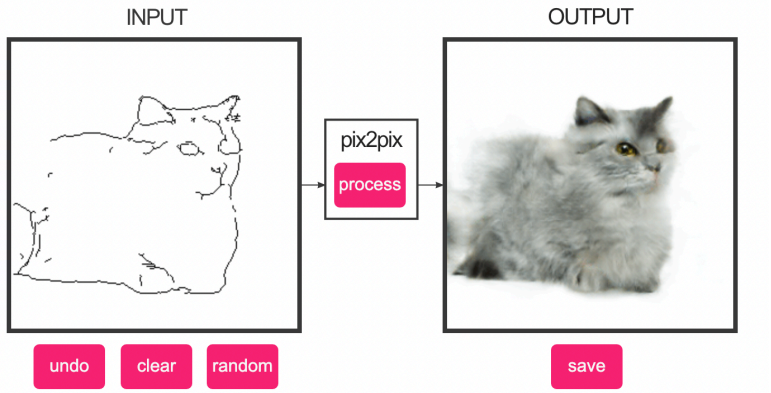


Generative adversarial networks

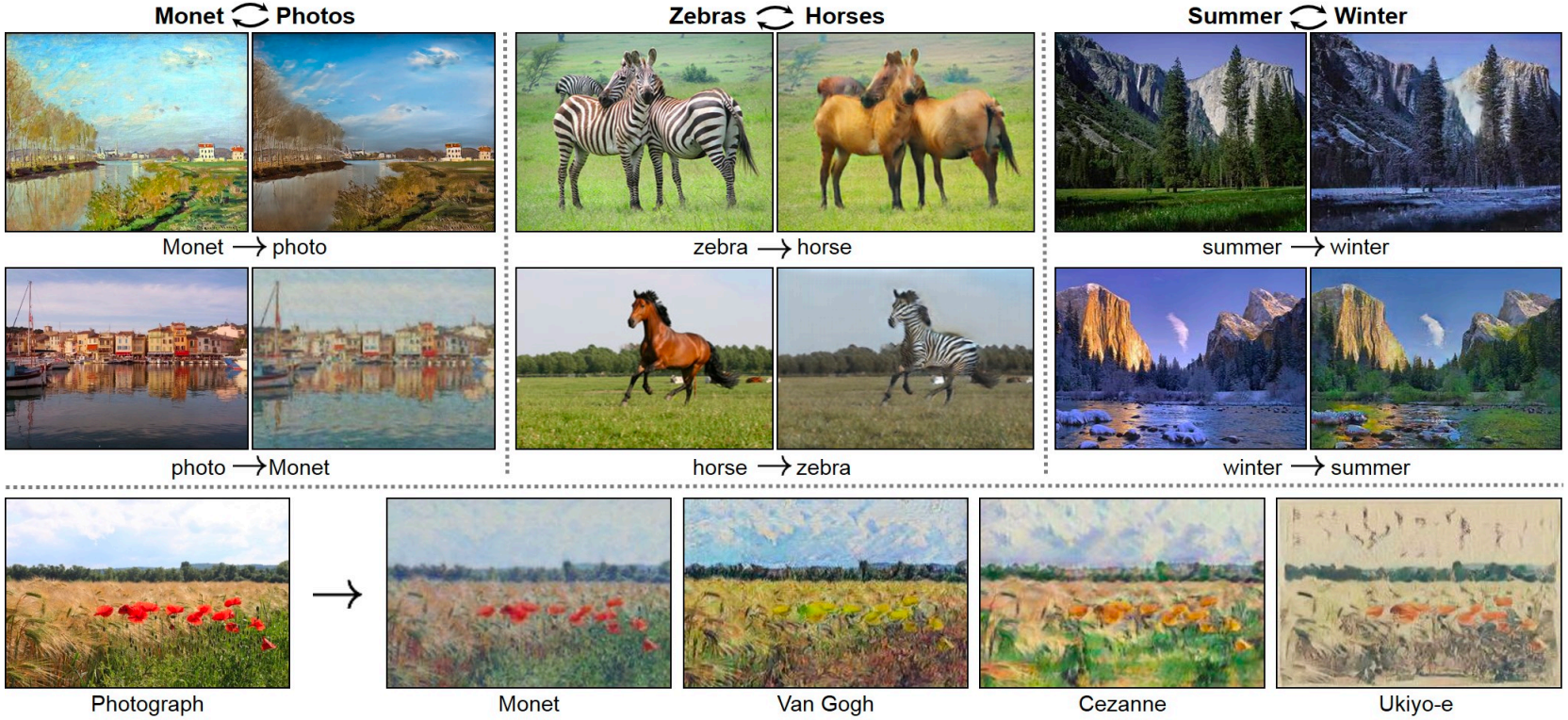
Another famous examples:

- cycle GANs
- pix2pix

github.com/junyanz/pytorch-CycleGAN-and-pix2pix



https://affinelayer.com/pixsrv/

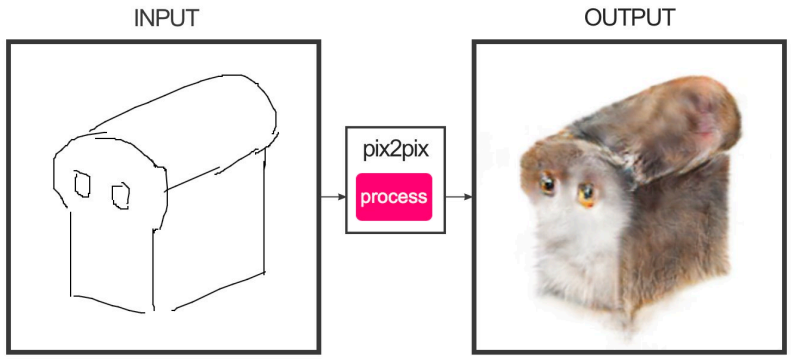


Generative adversarial networks

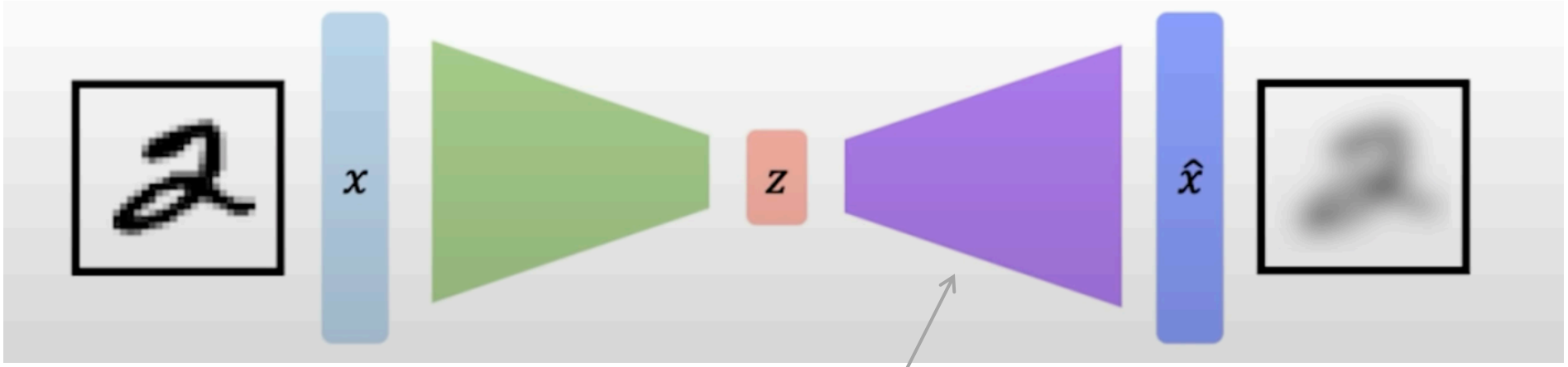
GAN = letting two neural networks **compete** with each other to **generate** new data

- The original purpose is to generate new data
- Classically for generating new images, but applicable to wide range of domains
- Learns the training set distribution and can generate new images that have never been seen before
- In contrast to e.g., autoregressive models or RNNs (generating one word at a time), GANs generate the whole output all at once

Original GAN paper has about 45k citations
https://arxiv.org/abs/1406.2661

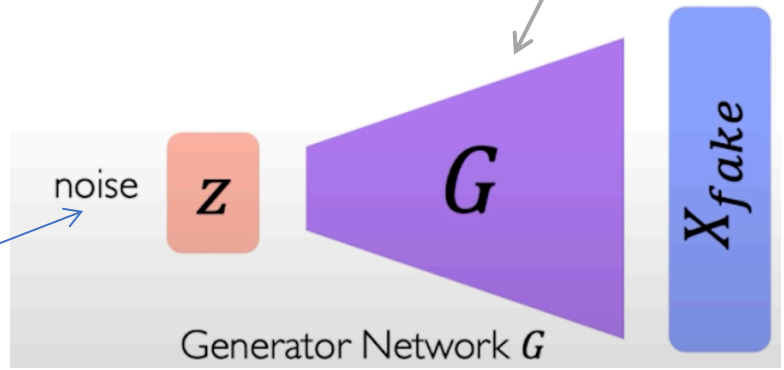


Generative adversarial networks



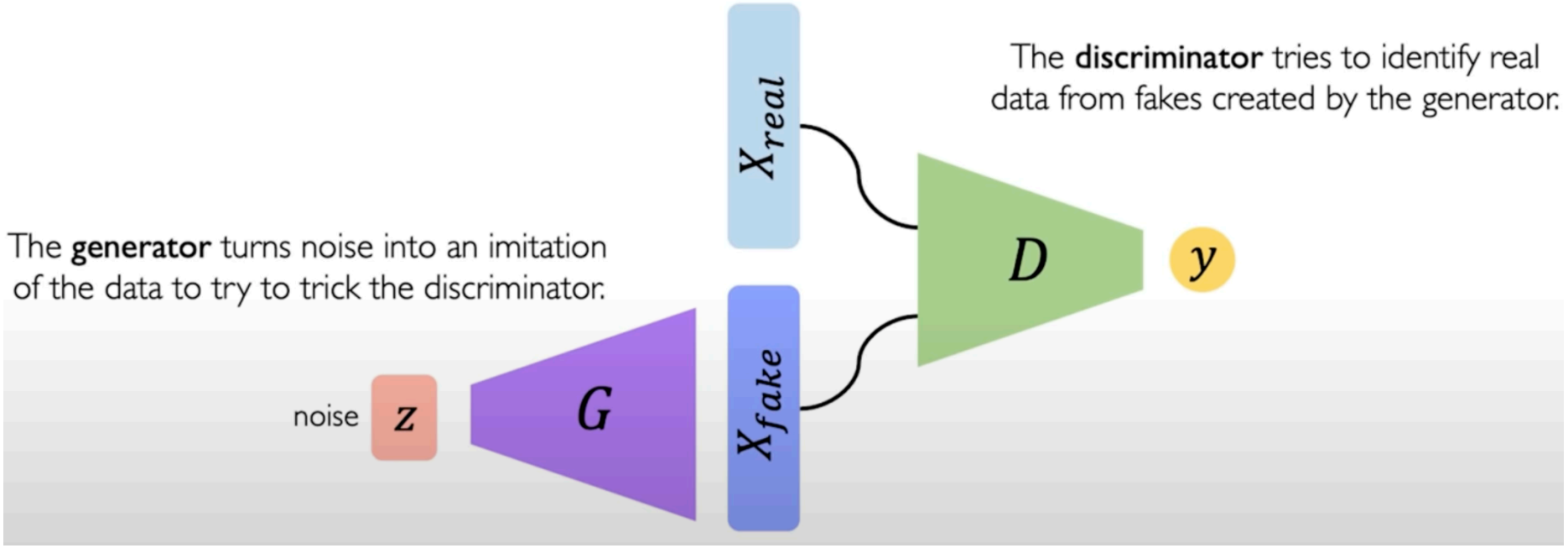
well organized latent space from which we can sample -> Variational Autoencoders

add noise to latent space



youtube.com/watch?v=BUNl0To1IVw

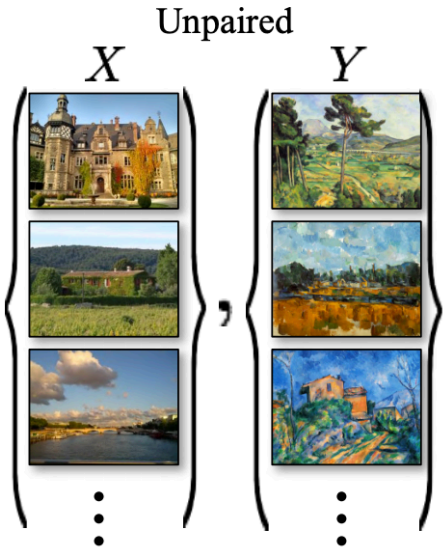
Generative adversarial networks



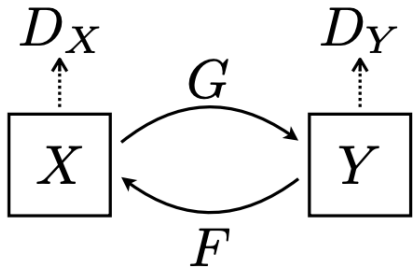
youtube.com/watch?v=BUNl0To1IVw

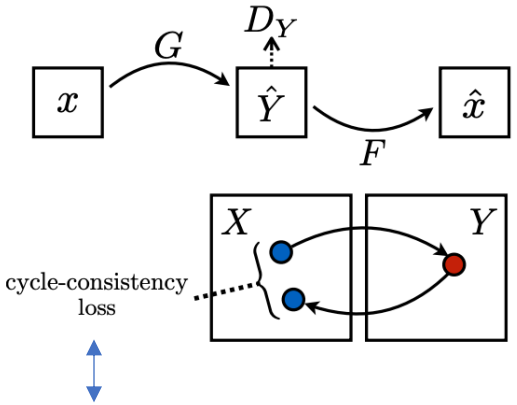
Cycle GANs

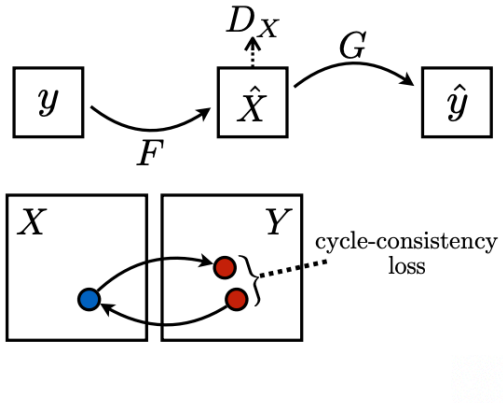
Cycle GAN
-> unpaired Image-to-Image translation



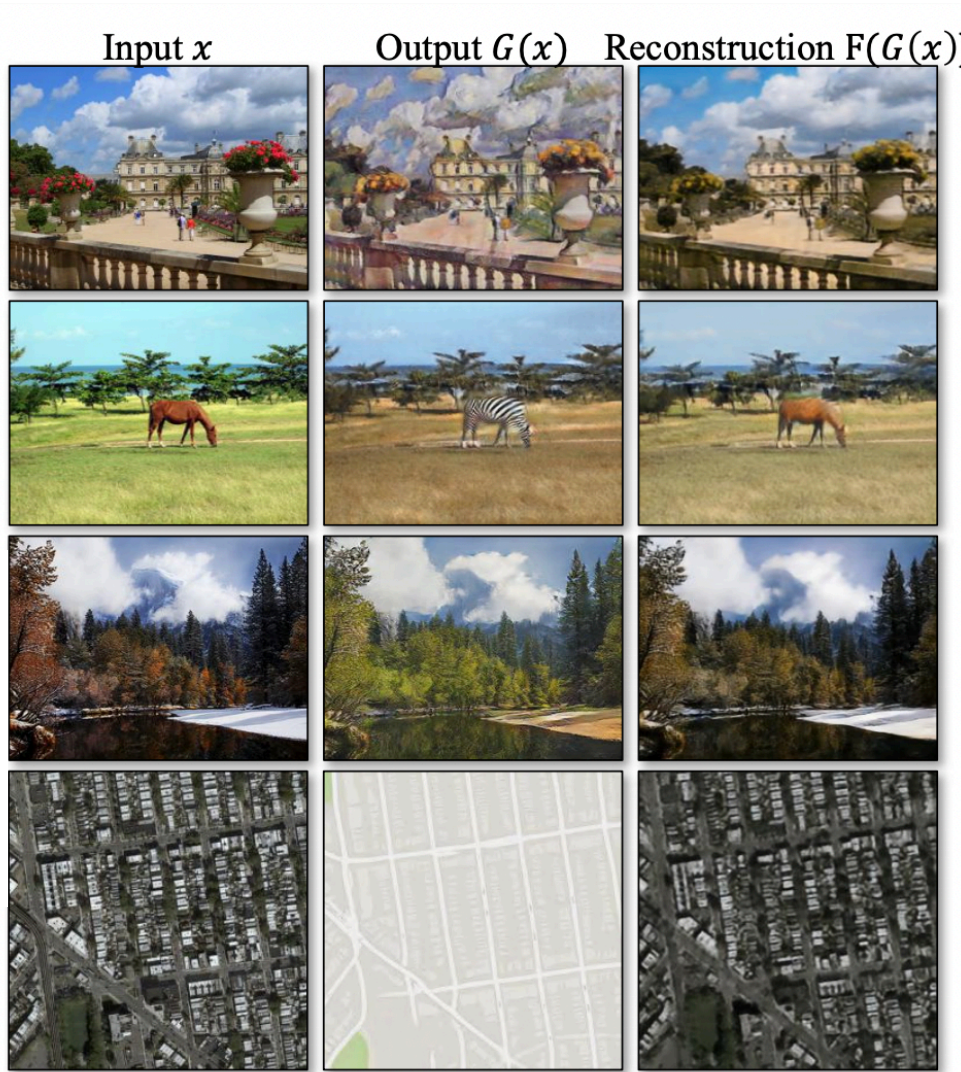
https://arxiv.org/pdf/1703.10593.pdf





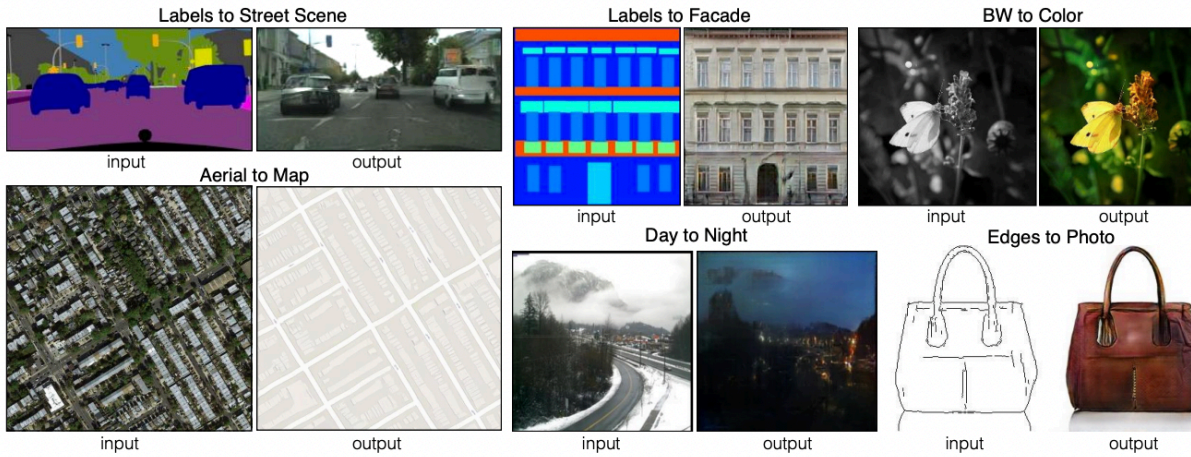


$$x \rightarrow G(x) \rightarrow F(G(x)) \approx x$$

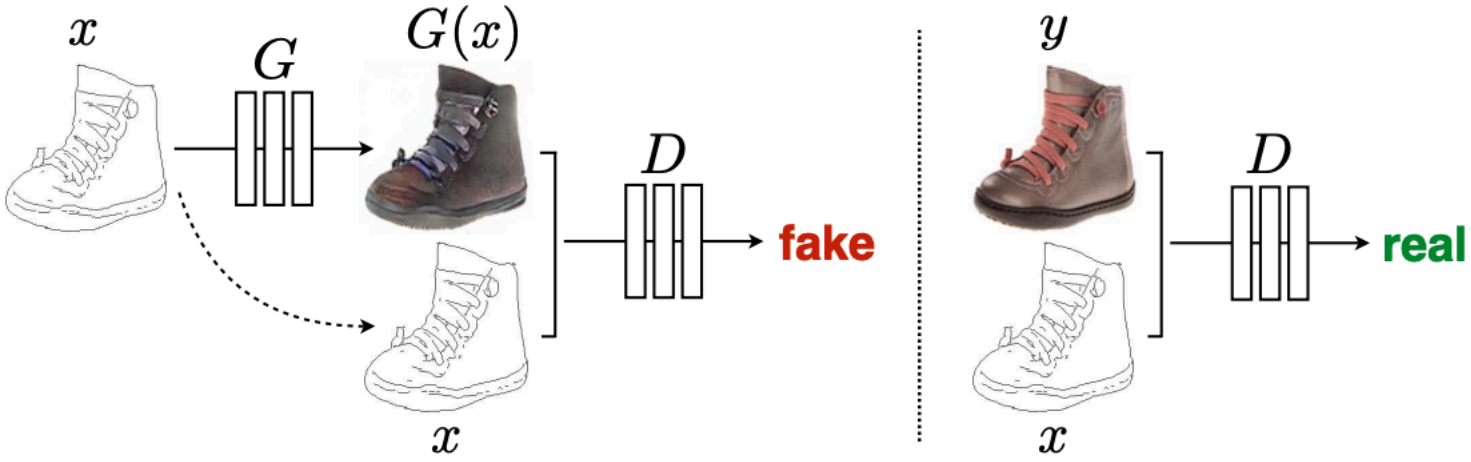


Conditional GANs

Conditional GAN
-> e.g. pix2pix model



Unlike an unconditional GAN, in CGANs both the generator and discriminator observe the input edge map:

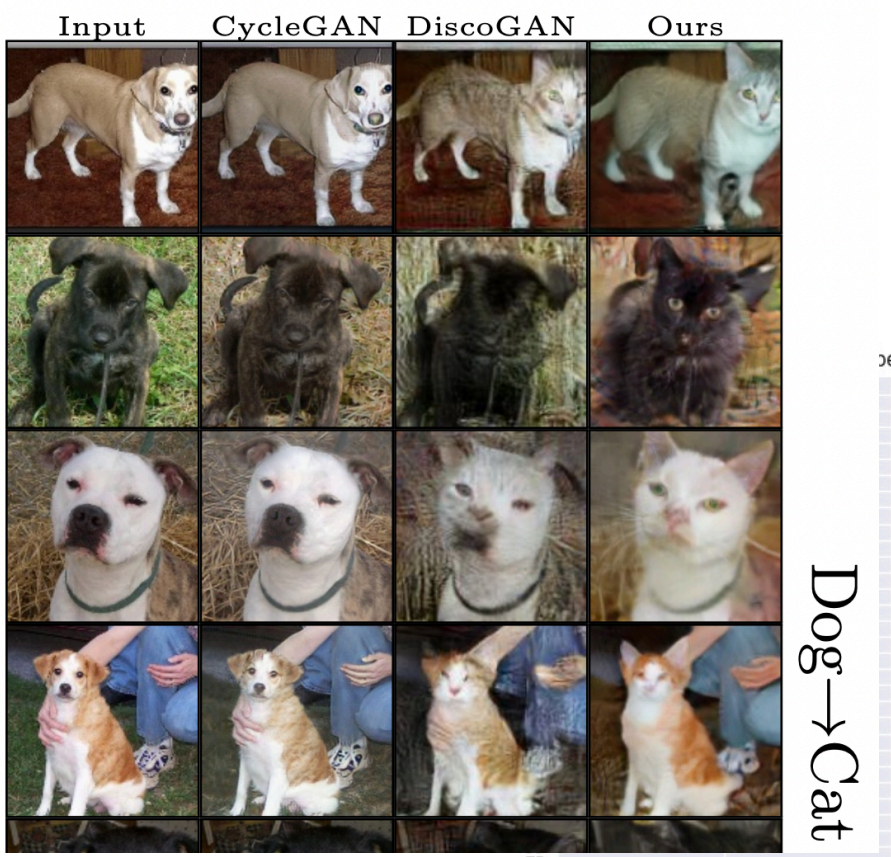


https://arxiv.org/pdf/1611.07004.pdf https://affinelayer.com/pixsrv/

Generative adversarial networks zoo

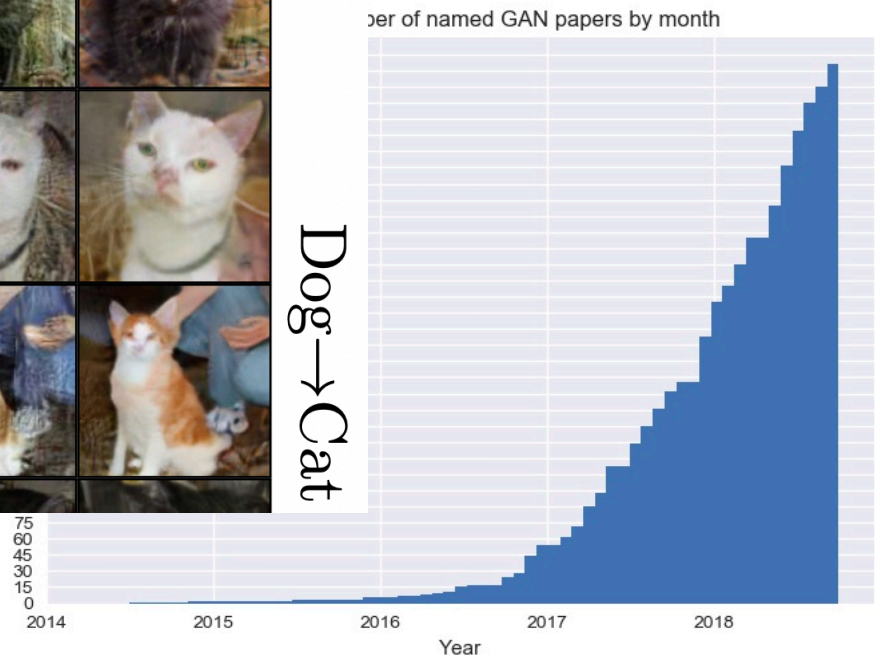
Many more flavors of GANs

github.com/hindupuravinash/the-gan-zoo



GANimorph

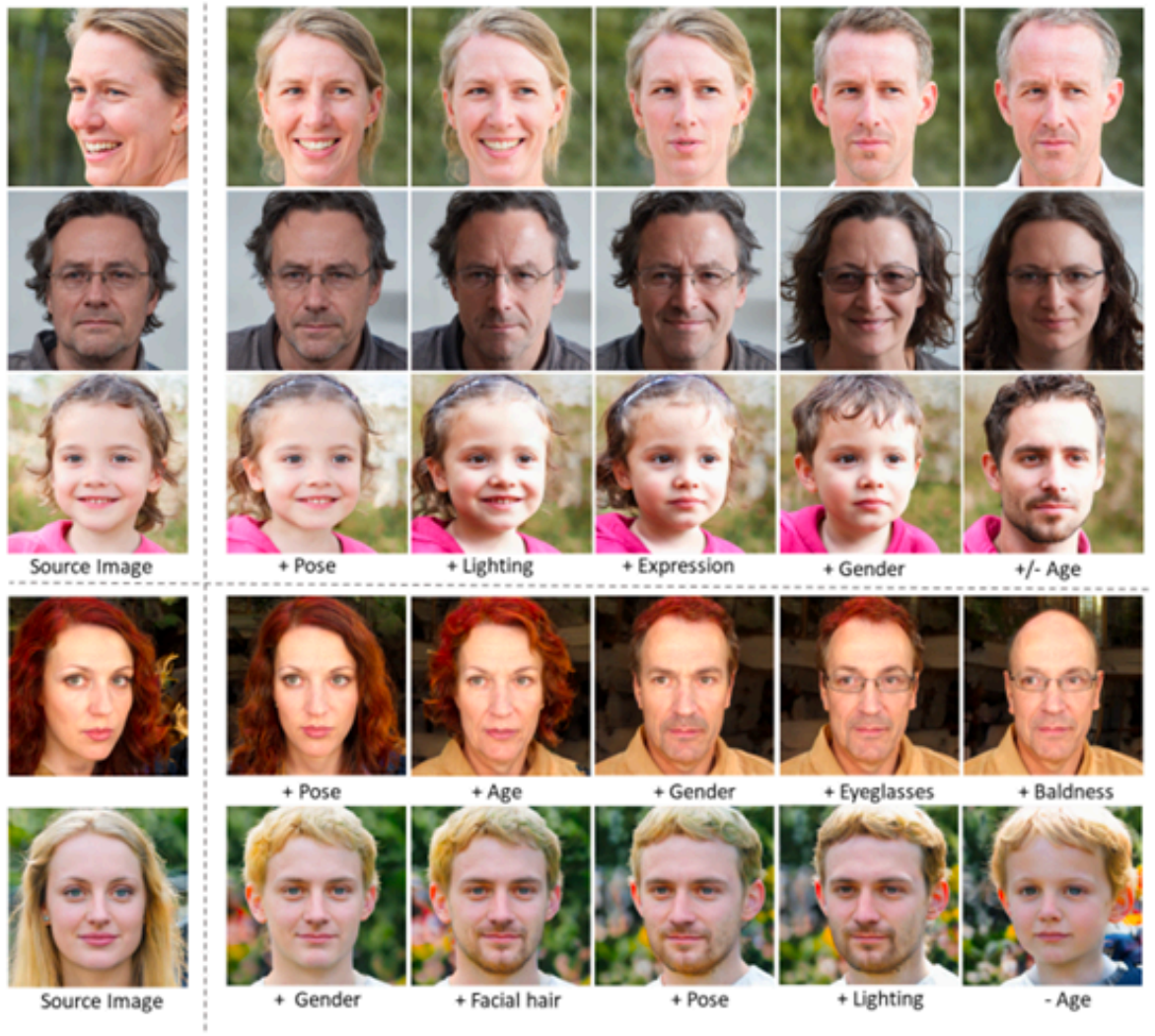
> 500 @ 2019



- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling (github)
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (github)
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations
- 3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning (github)
- ABC-GAN - ABC-GAN: Adaptive Blur and Control for improved training stability of Generative Adversarial Networks (github)
- ABC-GAN - GANs for LIFE: Generative Adversarial Networks for Likelihood Free Inference
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- ACGAN - Coverless Information Hiding Based on Generative adversarial networks
- acGAN - On-line Adaptative Curriculum Learning for GANs
- ACtuAL - ACtuAL: Actor-Critic Under Adversarial Learning
- AdaGAN - AdaGAN: Boosting Generative Models
- Adaptive GAN - Customizing an Adversarial Example Generator with Class-Conditional GANs
- AdvEntuRe - AdvEntuRe: Adversarial Training for Textual Entailment with Knowledge-Guided Examples
- AdvGAN - Generating adversarial examples with adversarial networks
- AE-GAN - AE-GAN: adversarial eliminating with GAN
- AE-OT - Latent Space Optimal Transport for Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AF-DCGAN - AF-DCGAN: Amplitude Feature Deep Convolutional GAN for Fingerprint Construction in Indoor Localization System
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AIM - Generating Informative and Diverse Conversational Responses via Adversarial Information Maximization
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference (github)
- AlignGAN - AlignGAN: Learning to Align Cross-Domain Images with Conditional Generative Adversarial Networks
- AlphaGAN - AlphaGAN: Generative adversarial networks for natural image matting
- AM-GAN - Activation Maximization Generative Adversarial Nets

Editing GANs latent space

How to edit images with GANs?



Deep Learning Workshops

Next meeting

- Wednesday, October 12th, 17:00 - 19:00 (theory)
- Monday, October 17th, 17:00 - 19:00 (theory + applications),
  - CNN, GANs
  - how to train simple neural network: MLP, CNN, GAN
- **Projects: subsequent Wednesdays**
  - **in-person: A1 wyb. Wyspiańskiego 27, room 219, 10:00-12:00 or 14:00-16:00**
  - **or online: zoom**