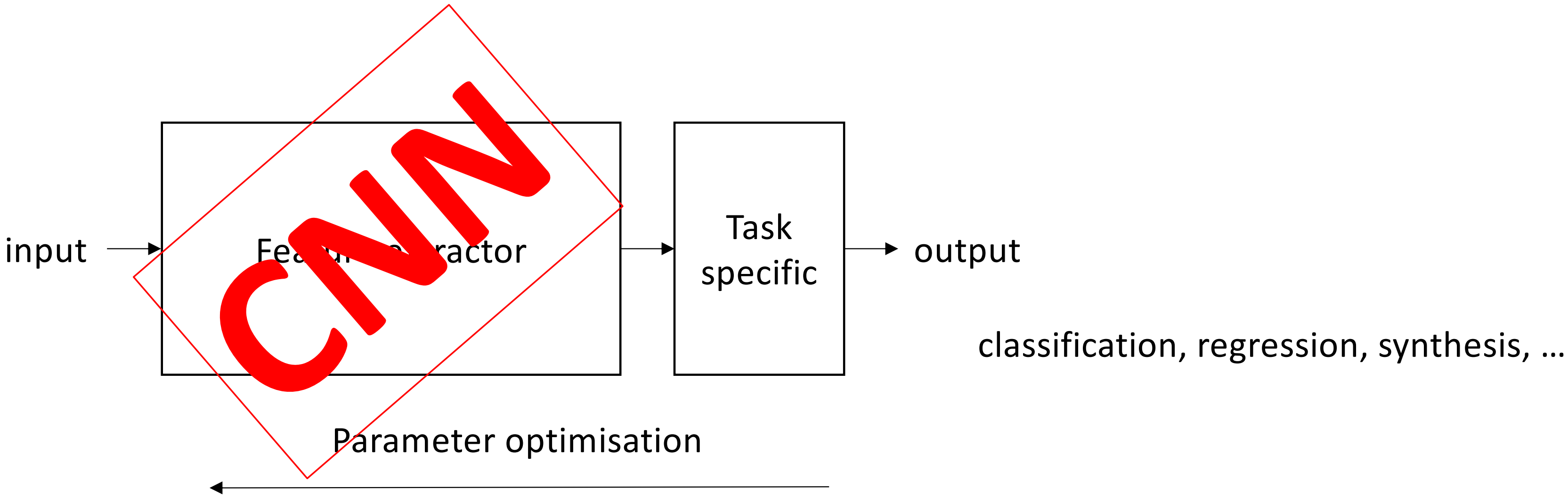
Deep Learning

Bernhard Kainz

Motivation

- Deep learning is **popular** because it works (often).
  - Big promise: just collect enough data and label it, then you get a magic black-box predictor that can predict any correlations at the click of a button. (only supervised setting really works well)
- Deep learning and Big data = **big money** = highly competitive and sometimes poisonous working environment.
- Deep learning can be **dangerous**, e.g. deep fakes, adversarial attacks, etc.

Fundamental learning system



*CNN = convolutional neural network

Success stories

Self driving cars: https://youtu.be/zRnSmw1i_DQ

Conversational AI: https://youtu.be/Xw-zxQSEzqo
https://youtu.be/jH-6-ZIgmKY https://chat.openai.com/

Deep fakes: https://youtu.be/gLoI9hAX9dw

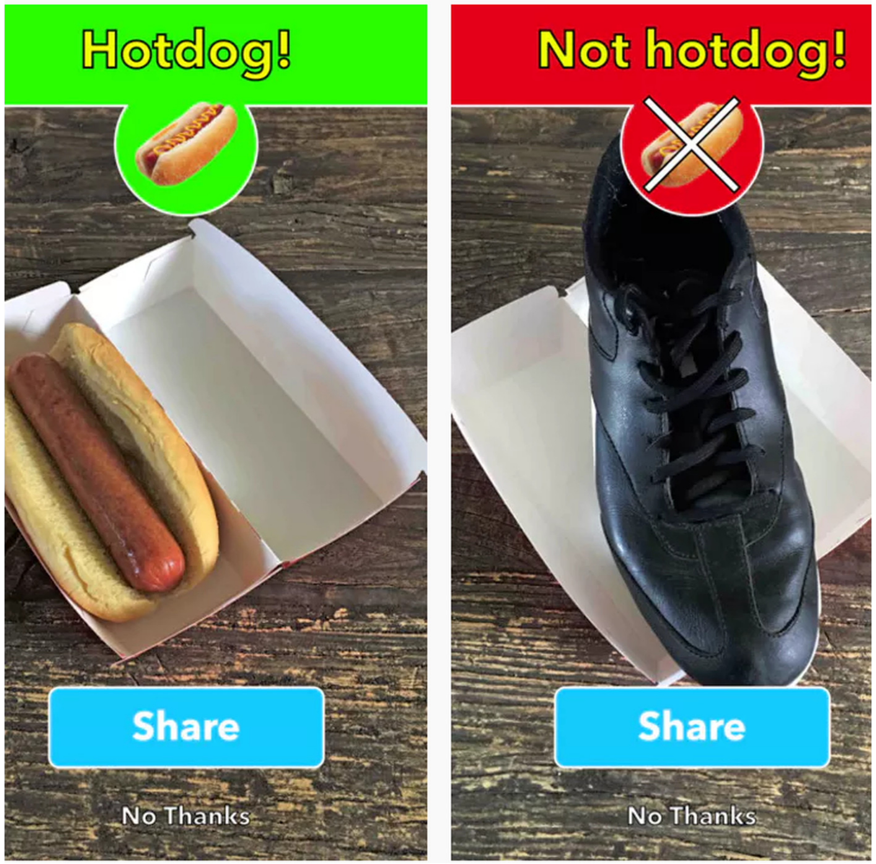
Neural rendering: https://www.matthewtancik.com/nerf

Image colourization: https://youtu.be/mUXpxxyThr8

Image captioning: https://youtu.be/8BFzu9m52sc

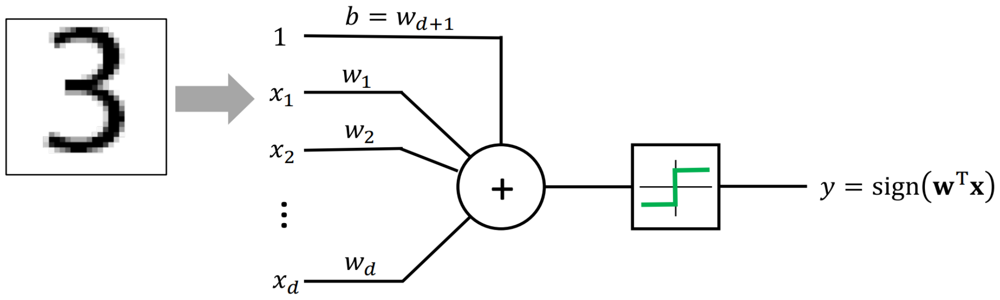
Automated diagnosis: http://ratchet.lucidifai.com/

Protein discovery: https://alphafold.ebi.ac.uk/



HBO and *Silicon Valley* *engadget.com*

Why did neural networks fail in image analysis?



Stack a 32x32x3 RGB image into a 3072x1 vector

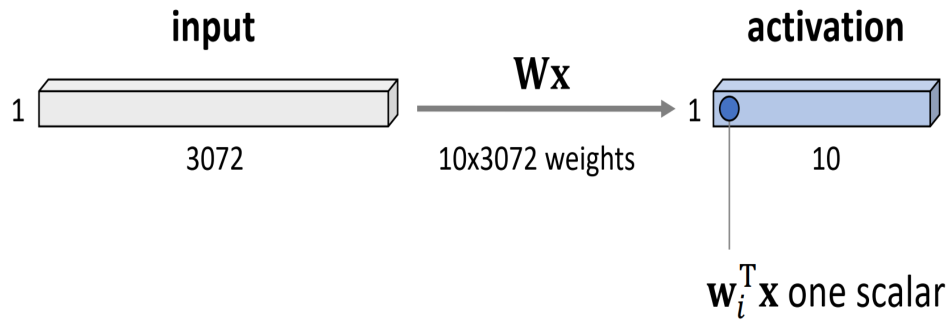


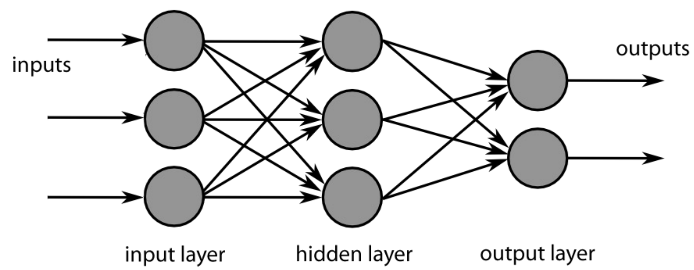
Figure: adapted from Fei Fei et al.

Universal Approximator

- Let $\varphi(\cdot)$ be a non-constant, bounded and monotonically increasing function
- For any $\epsilon > 0$ and any continuous function defined on a compact subset of $\mathbb{R}^m$, there exists and integer N, real constants $v_i b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}$ where $i = 1, \ldots, N$, such that

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} v_i \varphi(\boldsymbol{w}_i^T \boldsymbol{x} + b_i) \; with \; |F(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon$$

We can approximate *any* function with just one hidden layer with a sensible actitation function!
In practice ϵ very large and curse of dimensionally!
Solution: break up problem in many smaller problems (layers)



The curse of dimensionality

Curse of dimensionality

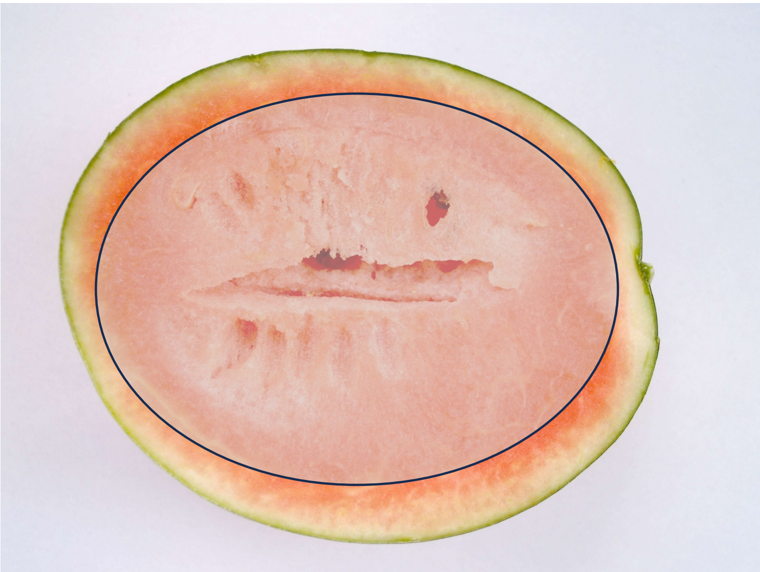
As the number of features or dimensions grows,
the amount of data we need to generalise accurately grows exponentially!

To approximate a (Lipschitz) continuous function $f\colon \mathbb{R}^d \to \mathbb{R}$
with ϵ accuracy one needs $O(\epsilon^{-d})$ samples

https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

Intuition

- Let’s analyze a Pizza
- And a water melon
- Shrink by α



Intuition

- In n dimension the n-dimensional volume of the interior will be α^n times the volume of the original shape.
- The volume of the rind relative to the original volume therefore is

$$1 - \alpha^n$$

- As a function of α its rate of growth is

$$d(1 - \alpha^n) = -n\alpha^{n-1} d\alpha$$

- Beginning with no shrinking (α=1) and noting α is *decreasing* (dα is negative), we find the initial rate of growth of the rind equals n.
- This shows that the volume of the rind initially grows much faster -- n times faster -- than the rate at which the object is being shrunk.
- in higher dimensions, relatively tiny changes in distance translate to much larger changes in volume.

Intuition

- If the salami is uniformly spread out over a high dimensional pizza
  - What proportion of the salami is near the boundary?
  - i.e. how much should we shrink the pizza to e.g. make it half of its volume, say half length like half-life of radioactive elements
  - The half-length is α, solve

$$\alpha^n = \frac{1}{2};\ \alpha = 2^{-1/n} = e^{-(\log 2)/n} \approx 1 - \frac{\log 2}{n} \approx 1 - \frac{0.7}{n}$$

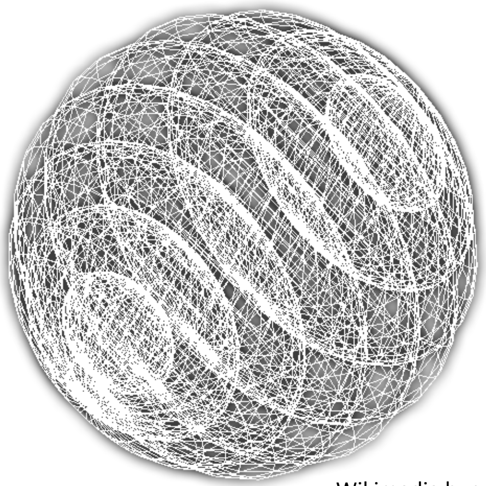
- 2D Pizza: half-length is 1−0.35
  - *half of the area of a pizza (n=2) lies within (approximately)* 35/2 = 18% *of its diameter from the boundary.*
- *3D Pizza:* half-length is 1−0.23
  - *half the volume lies within* 12% *of its diameter from its boundary.*

- **In very large dimensions the half-length is very close to 1**
  - *n=350 dimensions it is greater than 98%*
  - *Thus, expect half of any 350-dimensional pizza's salami to lie within 1% of its diameter from its boundary*

Intuition

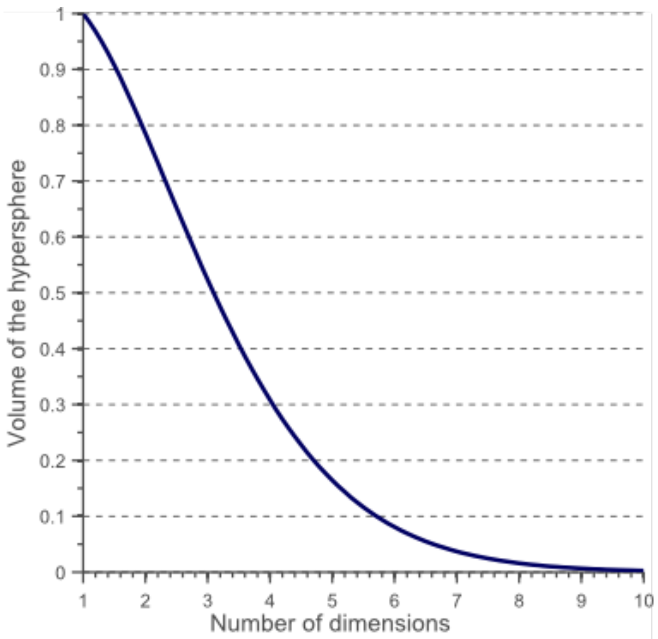
- Without strong clustering, in higher dimensions n we can expect most Euclidean distances between observations in a dataset to be very nearly the same and to be very close to the diameter of the region in which they are enclosed. "Very close" means on the order of $1/n$.

Intuition



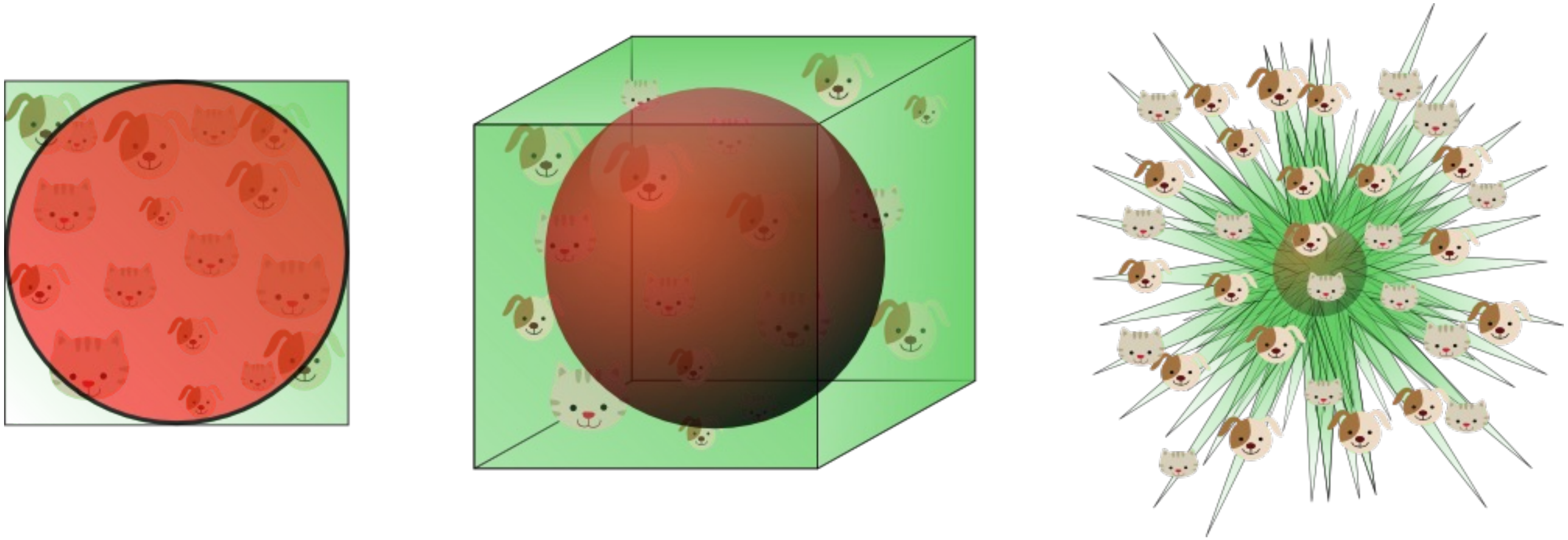
Wikimedia hypersphere

$$V_{sphere}(d) = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)2^d} \backsim O(c^{-d})$$



https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

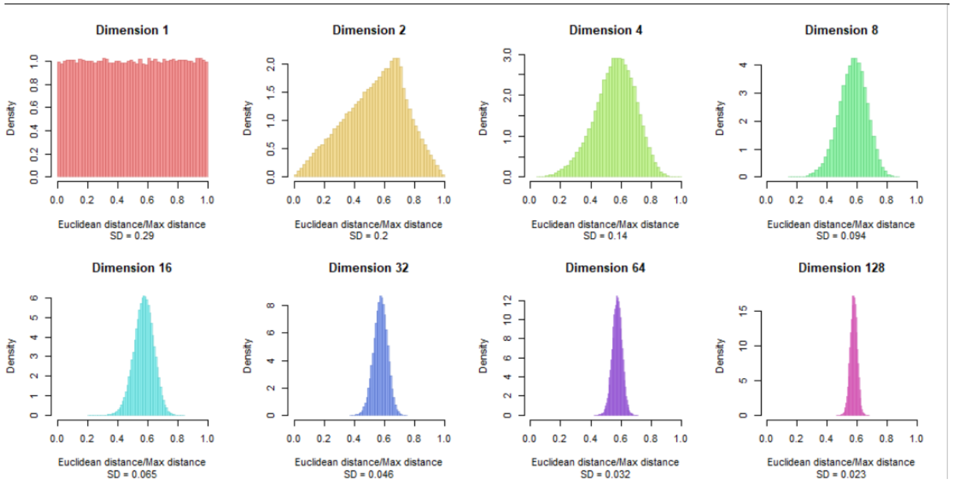
The higher dimensional the feature space the more training samples will be in the corners of the hypercube, thus generalisation suffers.



https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/
https://stats.stackexchange.com/questions/451027/mathematical-demonstration-of-the-distance-concentration-in-high-dimensions

Intuition

- Unit cube is asymmetric.
- To remove the asymmetry, roll the interval around into a loop where the beginning point 0 meets the end point 1: *d*-torus in *n* dimensions
- Plot distribution of normalized distance between different samples in different dimensional space
- This normalization has centered the histograms near 0.58
- around any given point on a high-dimensional torus nearly all other points on the torus are nearly the same distance away!



Curse of dimensionality

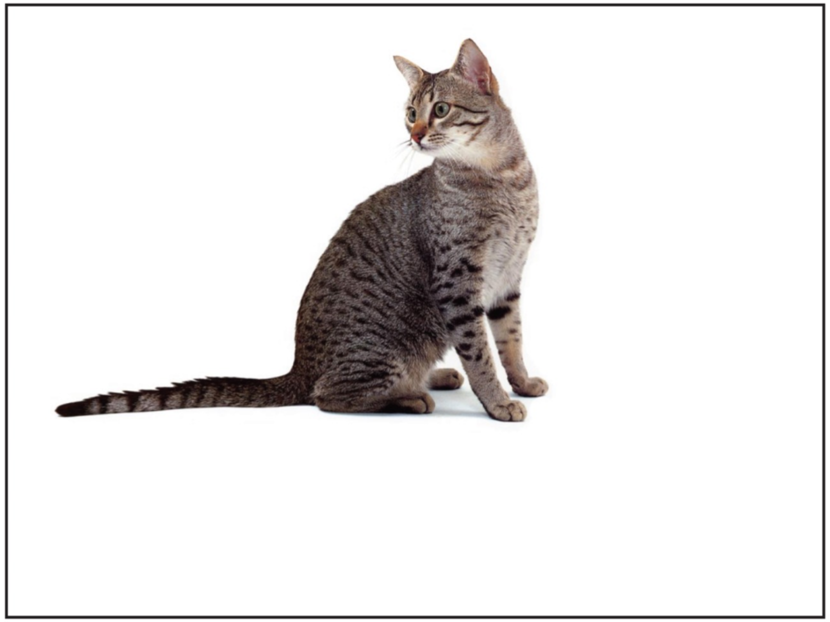
To approximate a (Lipschitz) continuous function $f\colon \mathbb{R}^d \rightarrow \mathbb{R}$
with ϵ accuracy one needs $O(\epsilon^{-d})$ samples

Input image resolution = 12 Mpixel * 3 channels = 36M elements
With $\epsilon \sim 0.1$, we need $10^{36000000}$ samples to approximate this function space
(10^{78} to 10^{82} atoms in the known, observable universe)

Invariance and Equivariance

Invariance and equivariance

- Shift invariance



Predictor: 'cat'

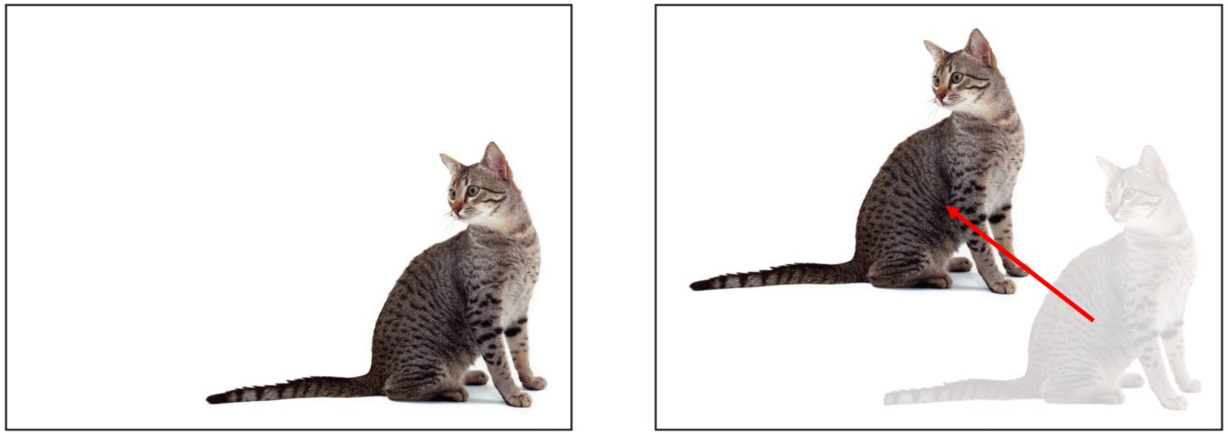
Invariance and equivariance

- Shift invariance

Predictor: 'cat'

Shift invariance

Input $\mathbf{x}$ **Shifted input** $S_v\mathbf{x}$

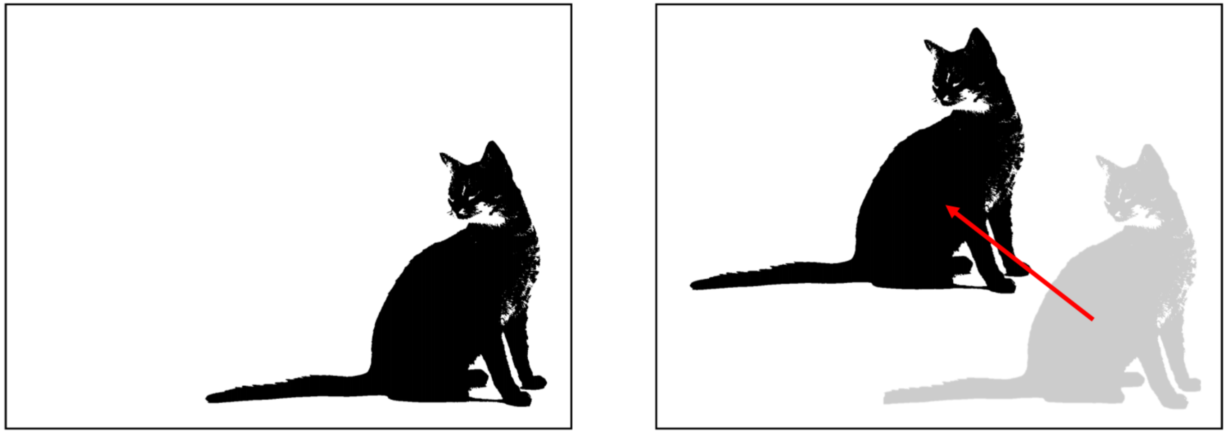


Output $f(\mathbf{x}) = 1$ **Output** $f(S_v\mathbf{x}) = 1$

- **'Cat detector'** $f: \mathbb{R}^d \rightarrow \mathbb{R}$

Shift equivariance

Input $\mathbf{x}$ | **Shifted input $S_v\mathbf{x}$**



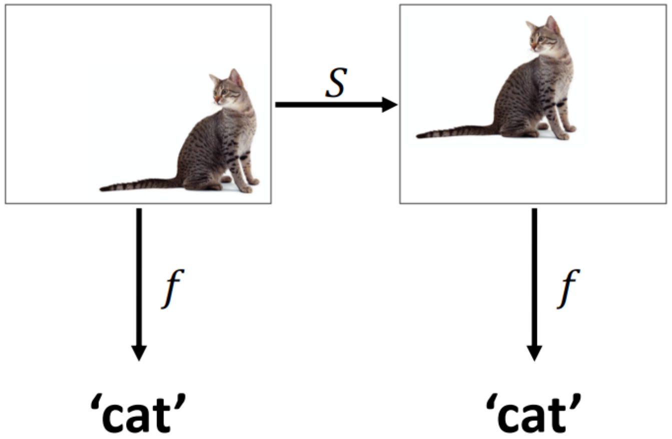
Output $f_i(\mathbf{x}) = \begin{cases} 1 & \text{pixel } i \ \in \text{cat} \\ 0 & \text{otherwise} \end{cases}$

Output $f_i(S_v\mathbf{x}) = \begin{cases} 1 & \text{pixel } i \ \in \text{cat} \\ 0 & \text{otherwise} \end{cases}$

- **'Cat segmentor' f**: $\mathbb{R}^d \rightarrow \mathbb{R}^d$
- **Shift operator** $S_v: \mathbb{R}^d \rightarrow \mathbb{R}^d$ shifting the image by vector $\boldsymbol{v}$

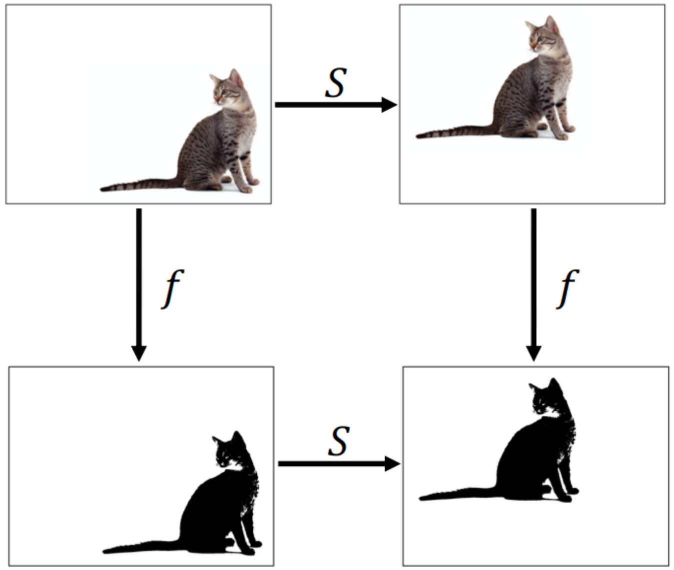
Invariance vs equivariance

Invariance



Output is the same

Equivariance



Output is shifted like the input

Inductive bias/assumptions

- First principle: translation invariance
  - a shift in the input should simply lead to a shift in the hidden representation
- second principle: locality
  - we believe that we should not have to look very far away from any location (i,j) in order to glean relevant information to assess what this area contains

translation invariance and locality – sliding window

Subimage $\hat{I}_{i,j} = I[i : i + m, j : j + n]$

Image I

- Correlation

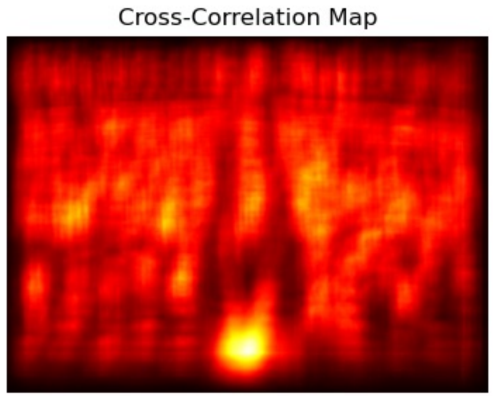
$$C_{i,j} = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \hat{I}_{i,j}(x, y) \cdot K(x, y)$$

Kernel (template) k

George

translation invariance and locality – sliding window

Subimage $\hat{I}_{i,j} = I[i{:}\,i+m, j{:}\,j+n]$



- Correlation

$$C_{i,j} = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \hat{I}_{i,j}(x,y) \cdot K(x,y)$$

Kernel (template) k

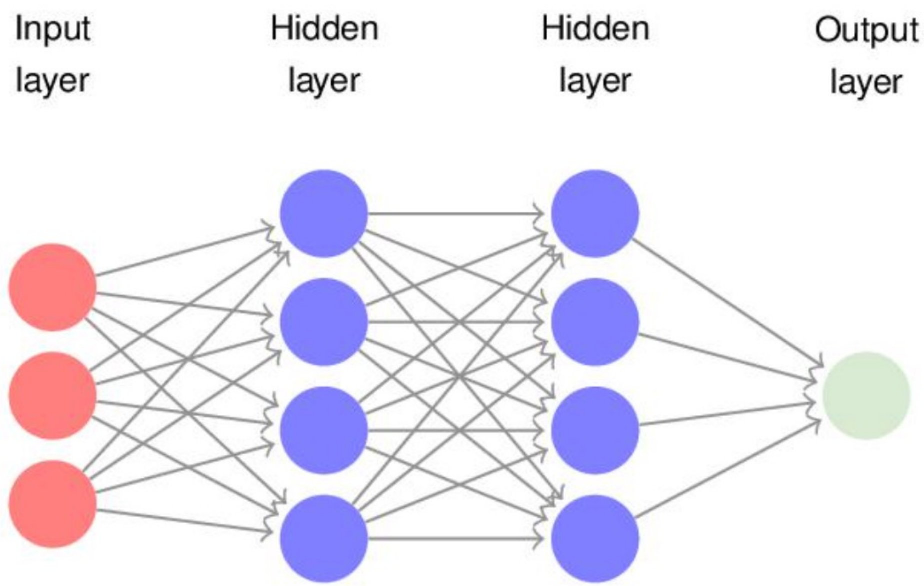
George

Image I

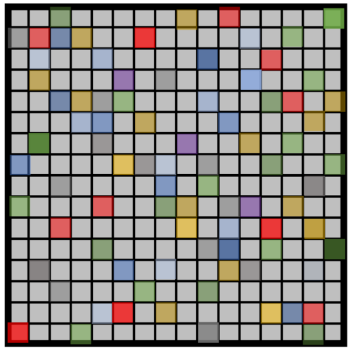
Convolutions

Fully connected neural networks

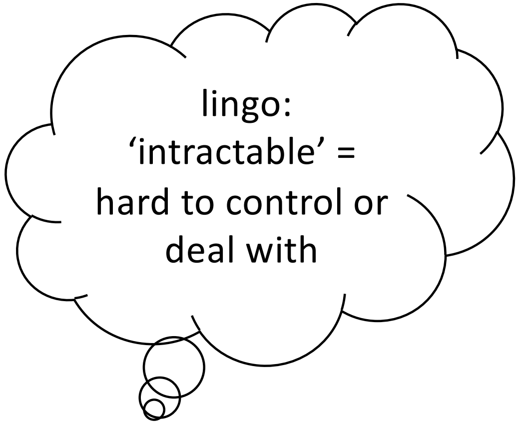
- Each input is connected to each node
- Can represent any kind of (linear) relationship between inputs

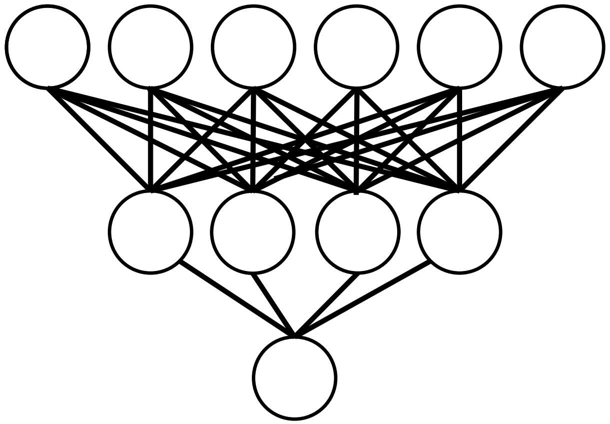


Fully connected neural networks



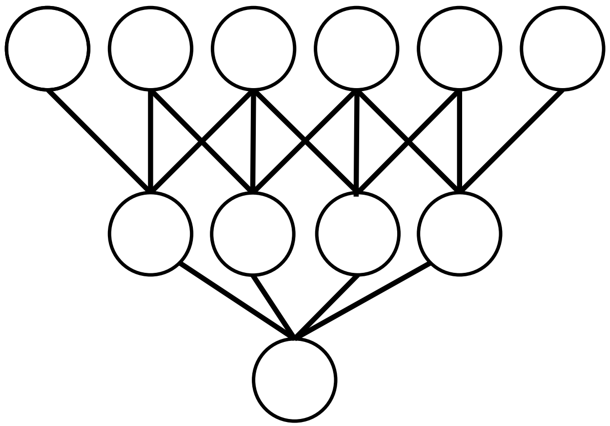
$$y_j = w_{j,1}x_1 + \cdots + w_{j,n}x_n$$

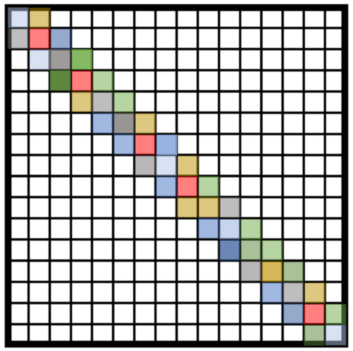


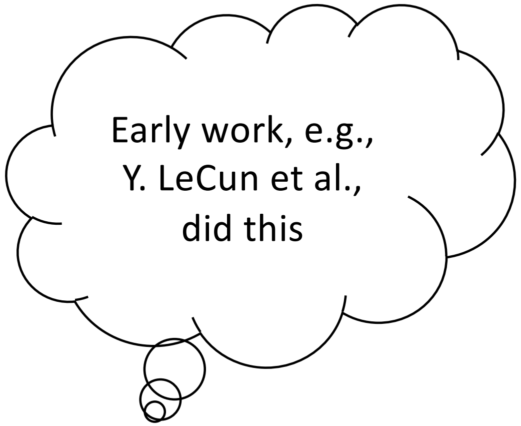


n^2 parameters, e.g., $36M^2$ parameters!

Sparsely connected neural networks



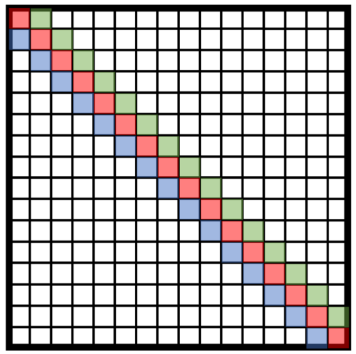




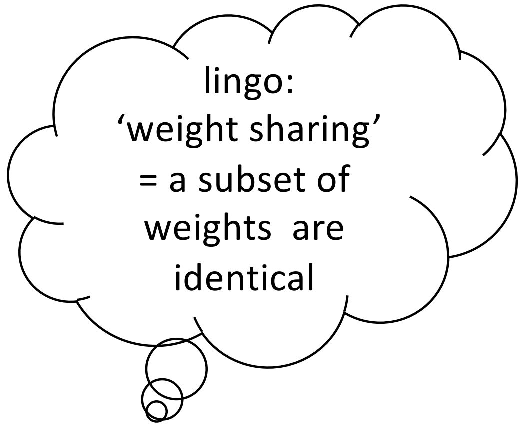
$$y_j = w_{j,i-1}x_{i-1} + w_{j,i}x_i + w_{j,i+1}x_{i+1}$$

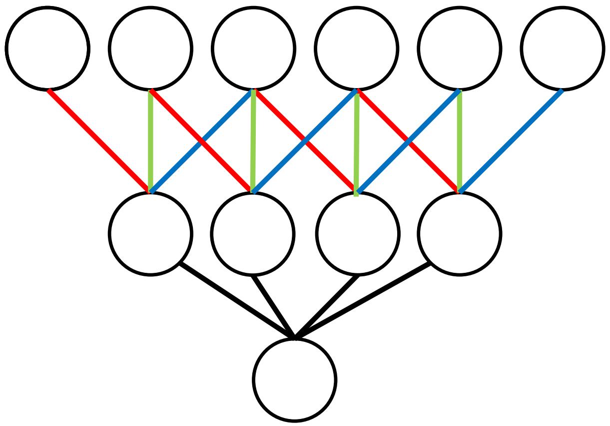
Each input neuron is connected to a small number k of hidden neurons.
Sparse connections: k*n parameters, e.g., 3*36M parameters!

Weight sharing neural networks



$$y_j = w_{-1}x_{i-1} + w_0 x_i + w_{+1}x_{i+1}$$





Each input neuron is connected to a small number k of hidden neurons and weights are shared

Shared weights (position independent): k parameters, e.g. 3 parameters!

Convolution

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \;\; for\; f, g : [0, \infty) \to \mathbb{R}$$

Correlation

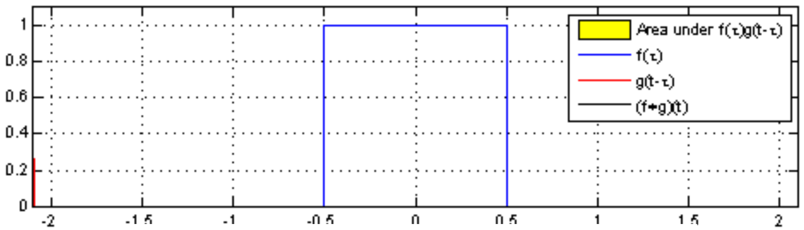
$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t + \tau)d\tau \;\; for\; f, g : [0, \infty) \to \mathbb{R}$$

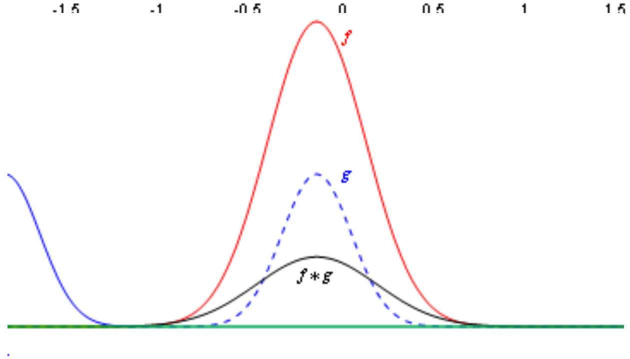
Convolution discrete version

- Given array u_t and w_t, their convolution is a function s_t

$$s_t = \sum_{a=-\infty}^{+\infty} u_a w_{t-a}$$

- When either u_t and w_t are not defined, they are assumed to be 0





wikipedia.org

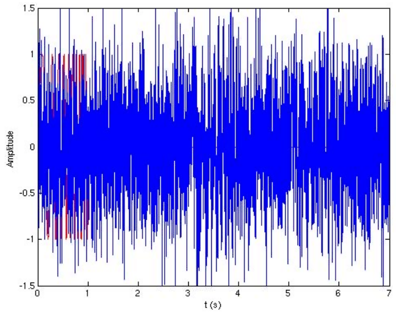
Network output (continuous):

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau \;\; for\ f,g : [0,\infty) \rightarrow \mathbb{R}$$

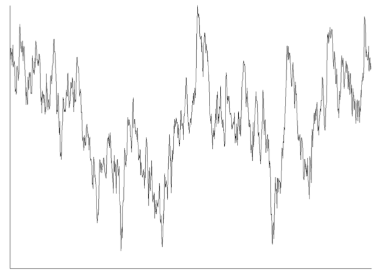
Some features of convolution are similar to <u>cross-correlation</u>: for real-valued functions, of a continuous or discrete variable, it differs from cross-correlation only in that either $f(x)$ or $g(x)$ is reflected about the y-axis; thus it is a cross-correlation of $f(x)$ and $g(-x)$, or $f(-x)$ and $g(x)$.

Why not simply input = output for this feature detector?

Signals in the wild:



Features in the wild:



Watch:
https://www.youtube.com/watch?v=N-zd-T17uiE
https://www.youtube.com/watch?v=IaSGqQa5O-M

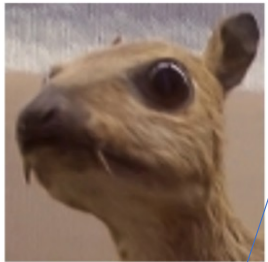
Properties of convolutions

- Commutativity, $f \ * g = g \ * f$
- Associativity, $f * (g * h) = (f * g) * h$
- Distributivity, $f * (g + h) = (f * g) + (g * h)$
- Associativity with scalar multiplication, $a(f * g) \ = (af) * g$

Why Convolutions for pattern-matching?

- *Historical Reasons:* The operation in CNNs resembles the discrete 2D convolution operation, even though it's technically cross-correlation. The term "convolution" in CNNs has stuck due to historical reasons and convention. Computational advantages for large kernels with FFT. Mathematical advantages for probability distributions.
- *Flipped Kernels:* In some contexts, before applying the convolution operation, the kernel is flipped both horizontally and vertically. Once flipped, applying cross-correlation will be equivalent to applying convolution with the original kernel. However, **in CNNs, the kernels are learned, so it doesn't matter if they are flipped or not**; **the network will learn the appropriate values during training**.
- Regardless of whether true convolution (with kernel flipping) or cross-correlation is used, the result of training will be the same. The network will adjust its weights based on the feedback from the loss during backpropagation. Thus, for the purpose of training neural networks, the distinction between the two becomes largely irrelevant.
- *Implementation*: In deep learning frameworks like TensorFlow or PyTorch, the operation performed in the convolutional layers is actually cross-correlation. However, they still use the term "convolution" due to convention.

Examples of 2D image filters



(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

Remember: in CNNs all learned through backpropagation, dependant on the task!

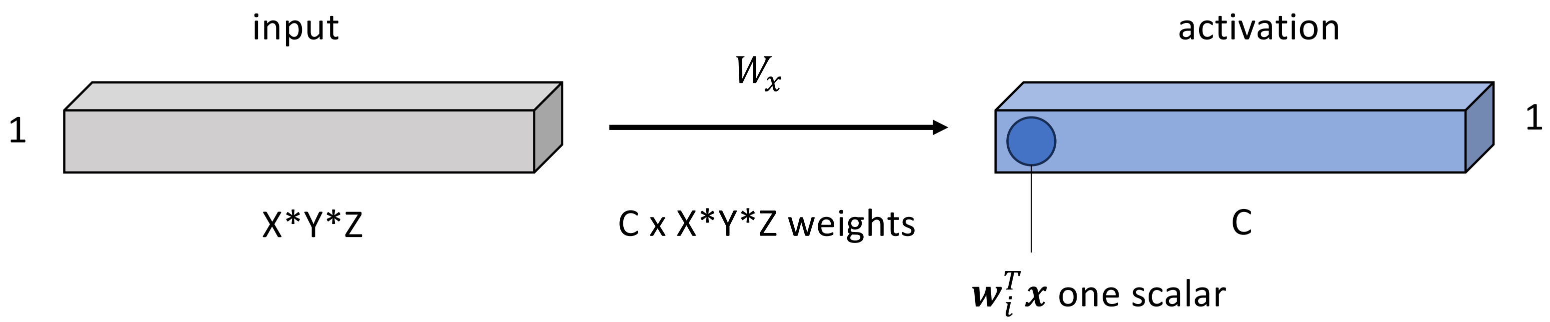
Slide credit: Smola, Li 2019

CNN building blocks

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer
- Flatten Layer
- Dropout
- Batch Normalization
- Activation Function
- Loss Function
- Optimizer

Input Tensor conventional NNs

Instead of stacking a [X,Y,Z] RGB image into a X*Y*Z x 1 vector for a conventional NN categorising in C classes



- we have priors about the data!

First principle: translation invariance

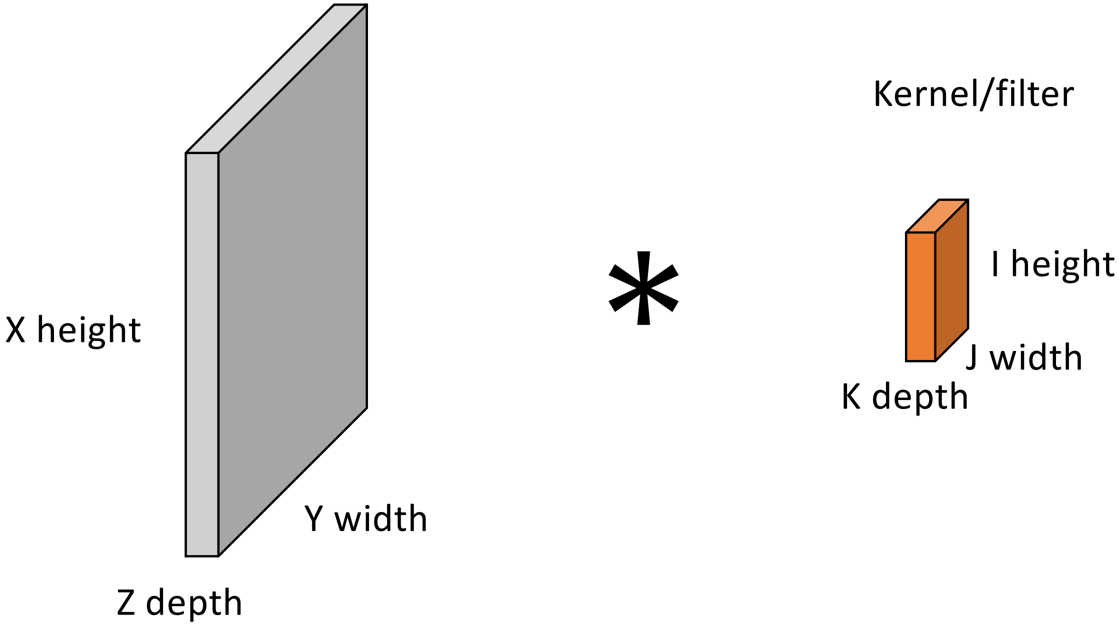
a shift in the input should simply lead to a shift in the hidden representation

second principle: locality

we believe that we should not have to look very far away from any location (i,j) in order to glean relevant information to assess what this area contains

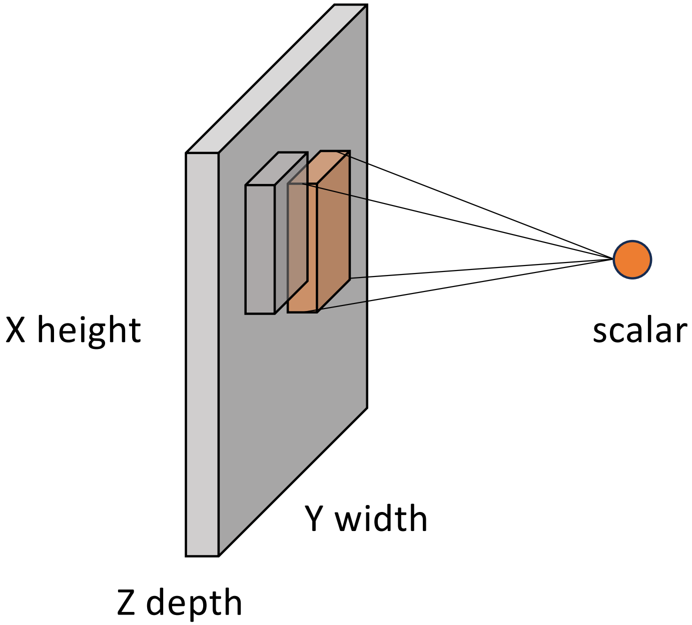
Kernel

We keep locality as a [X,Y,Z] $*$ [I,J,K] convolution



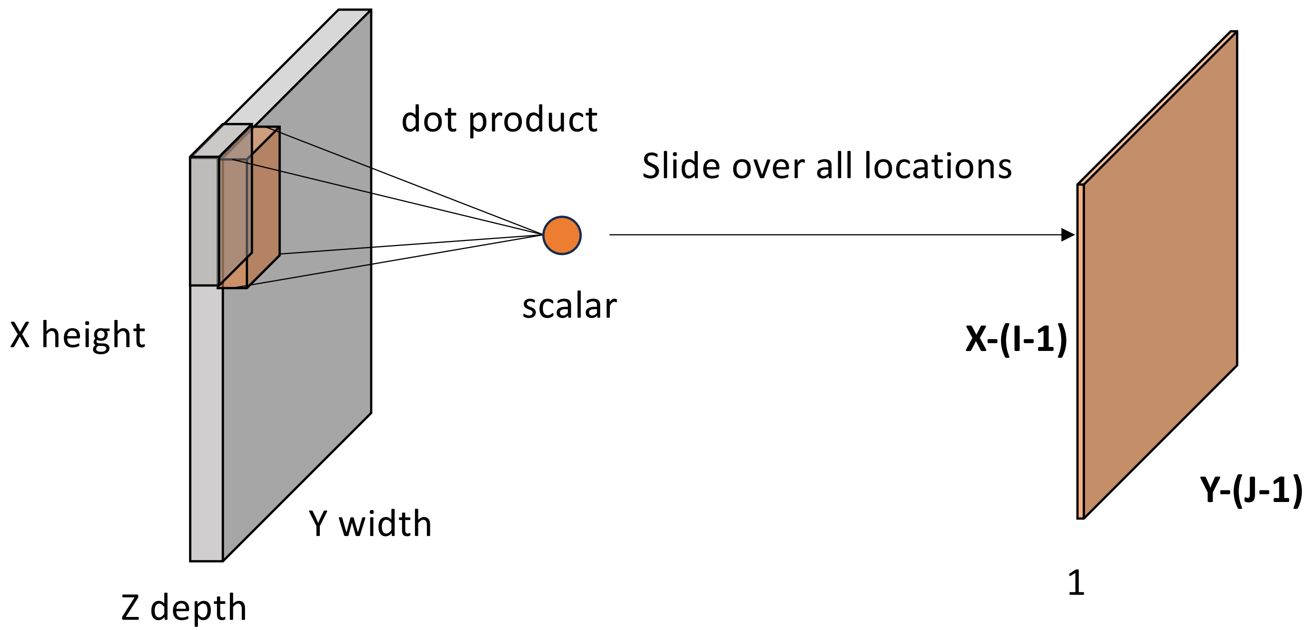
Convolution

In practice: dot product between the kernel and each image patch

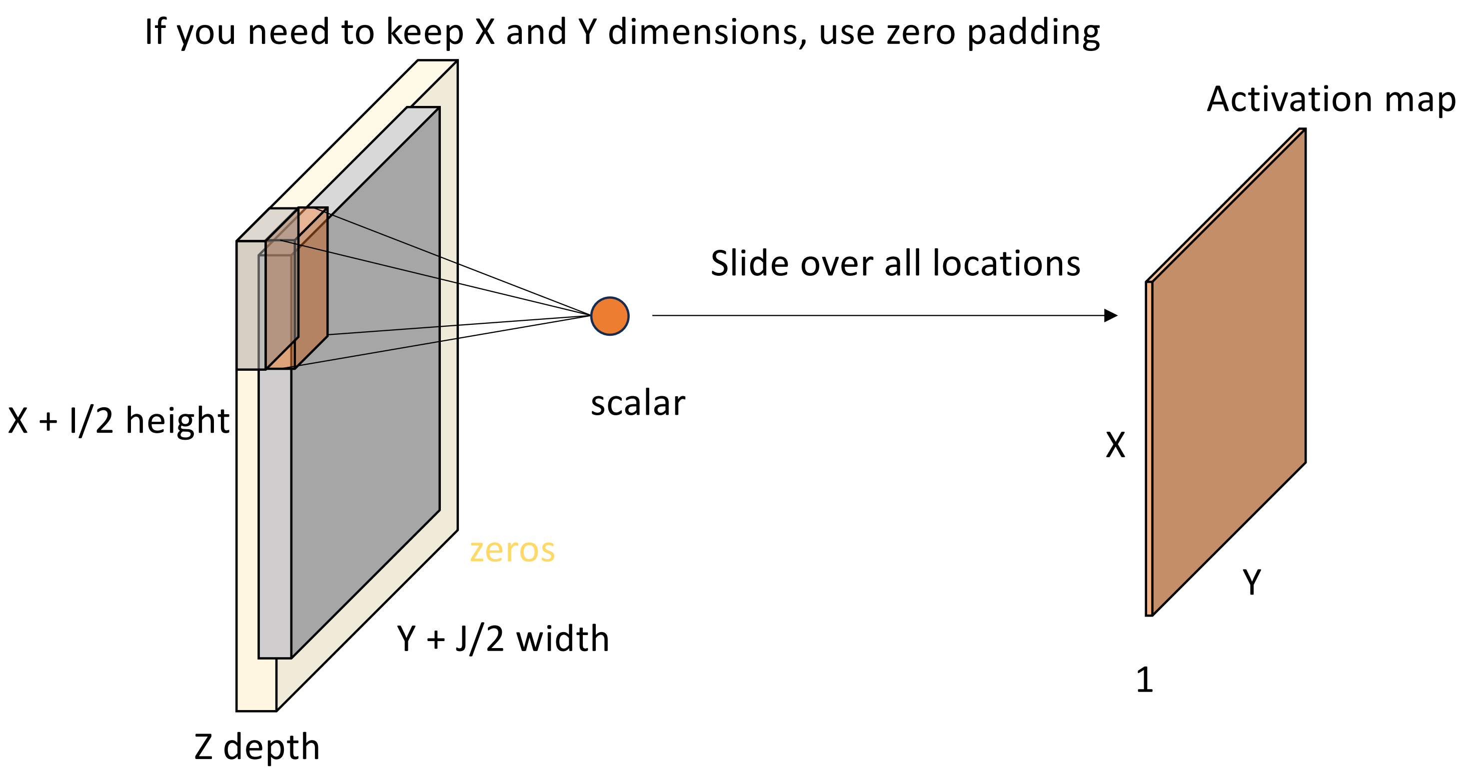


Convolution

In practice: dot product between the kernel and each image patch

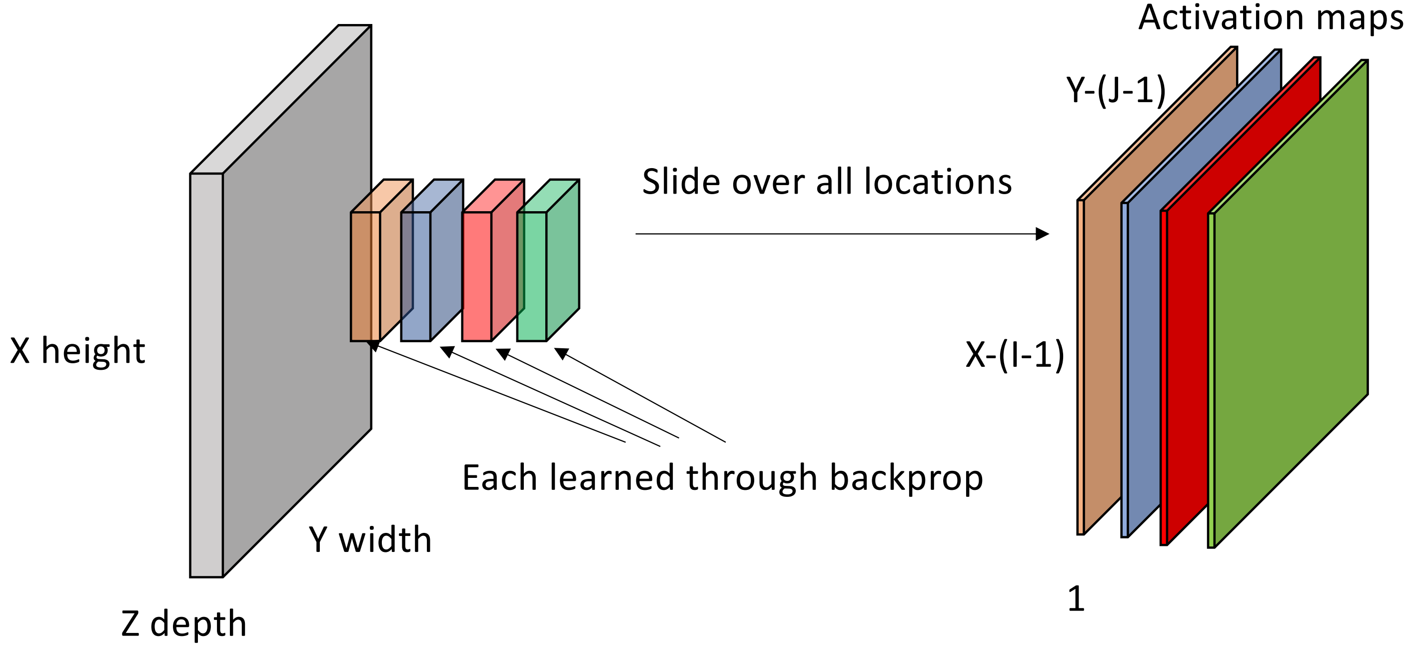


Input Tensor



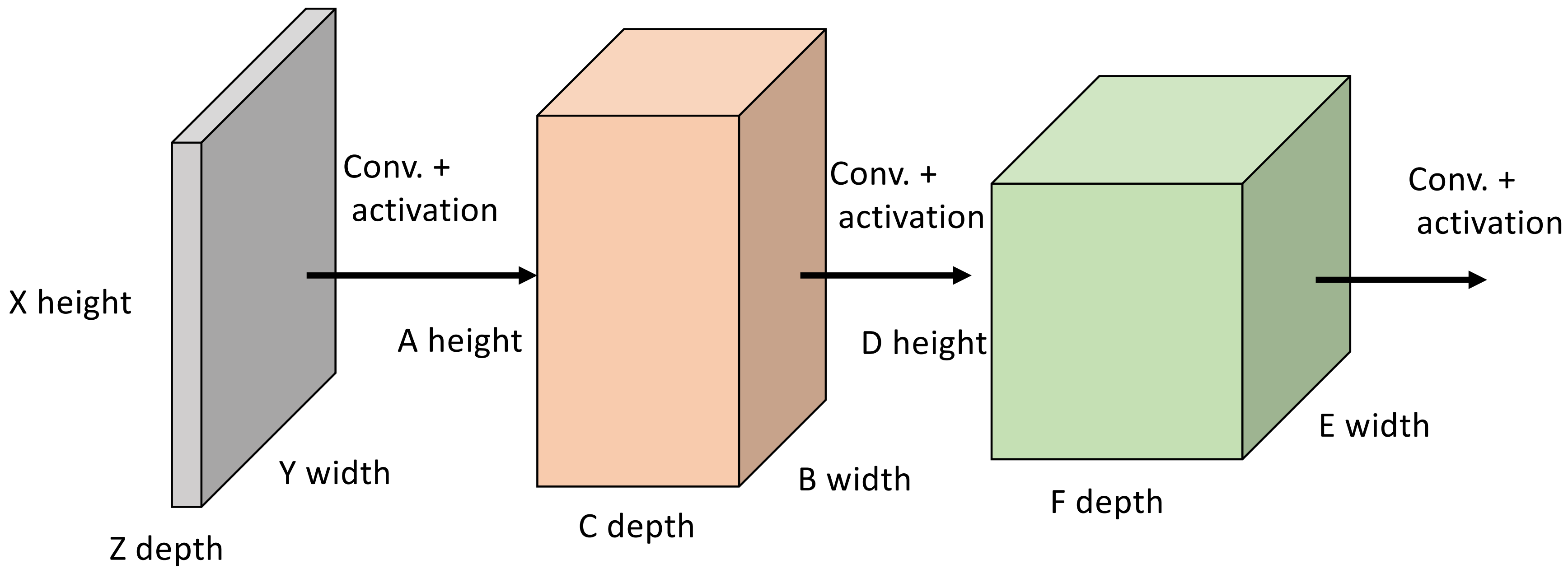
Feature extraction

Convolutional layers can learn as many kernels as you like (of the same dimension)



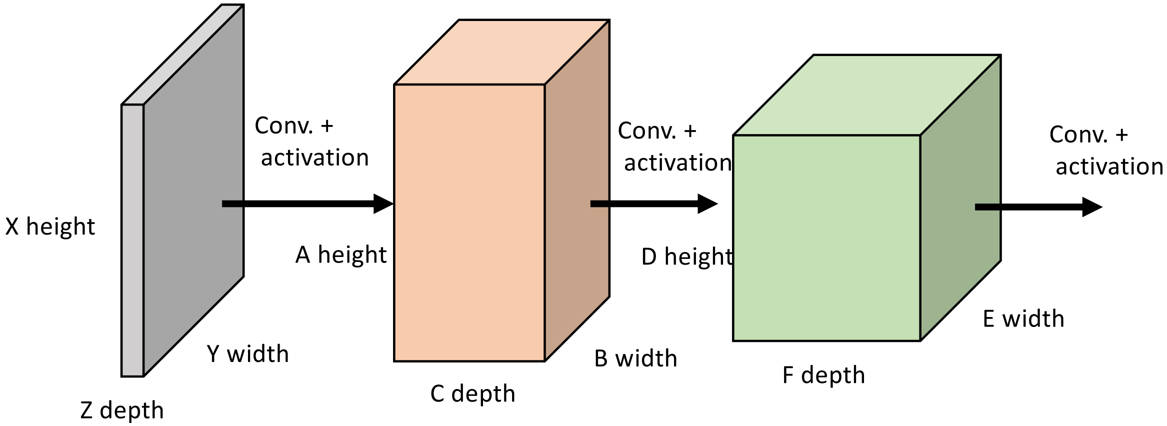
CNN

CNN = sequence of convolutional layers interleaved with activation functions



Parameters

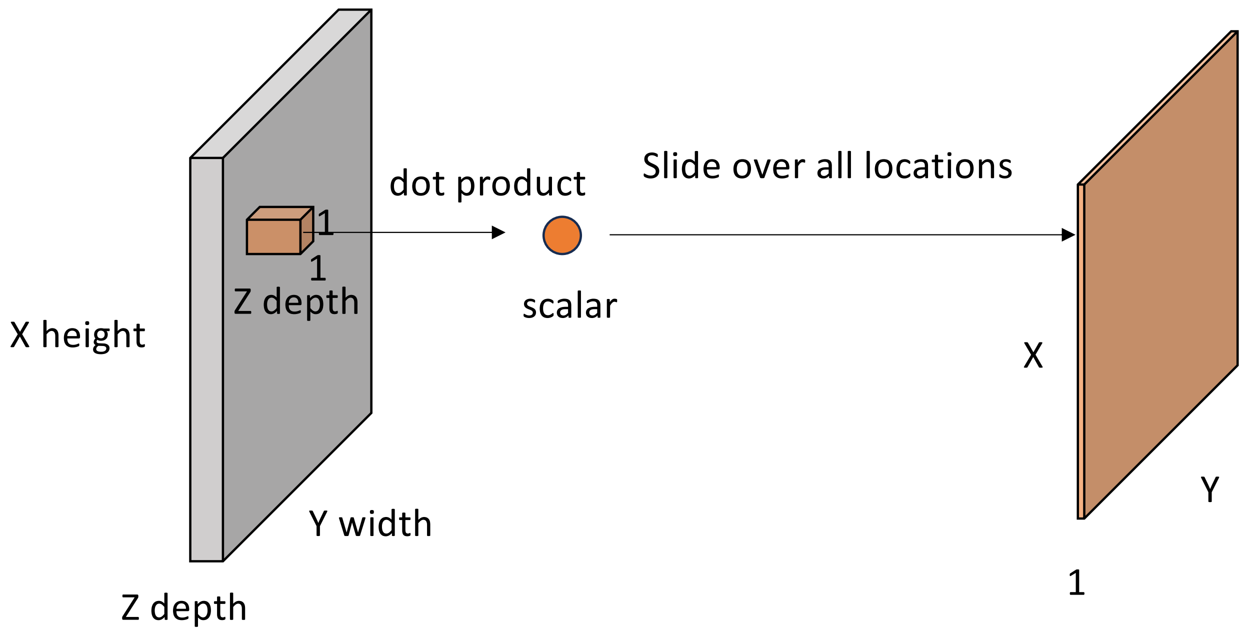
CNN = sequence of convolutional layers interleaved with activation functions



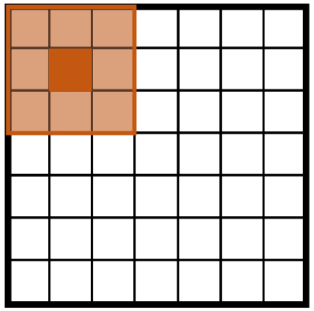
Each filter: I*J*K + 1 (bias) parameters to learn

1x1 Convolution – reduce depth/NN across depth

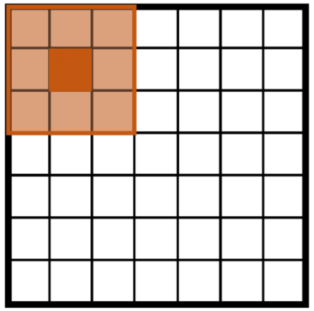
Learn to aggregate many channels into one



Padding and strides



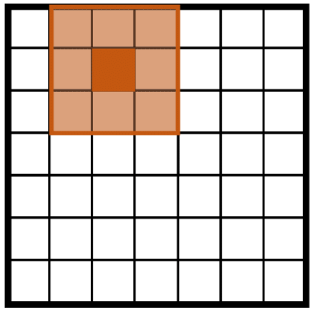
7x7 input
3x3 filter
stride 1
no padding



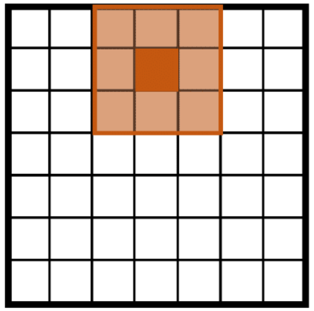
7x7 input
3x3 filter
stride 2
no padding

Figure: adapted from Fei Fei et al.

Padding and strides



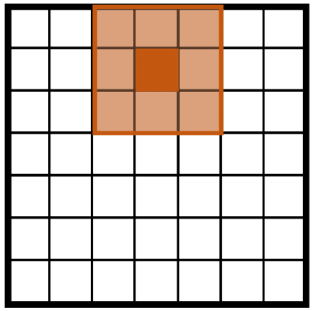
7x7 input
3x3 filter
stride 1
no padding



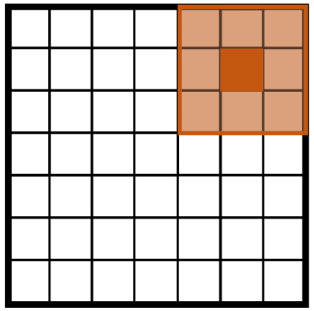
7x7 input
3x3 filter
stride 2
no padding

Figure: adapted from Fei Fei et al.

Padding and strides



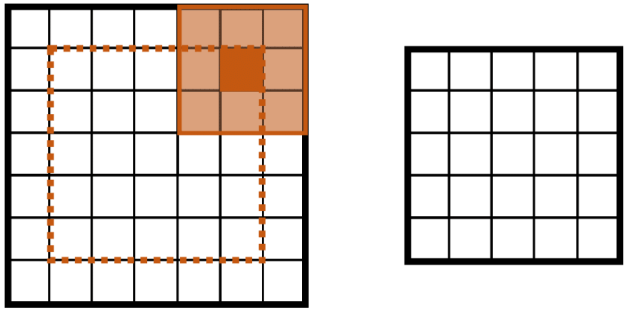
7x7 input
3x3 filter
stride 1
no padding



7x7 input
3x3 filter
stride 2
no padding

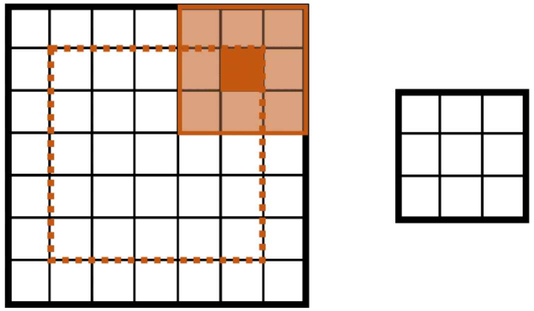
Figure: adapted from Fei Fei et al.

Padding and strides



7x7 input
3x3 filter
stride 1
no padding

5x5 output

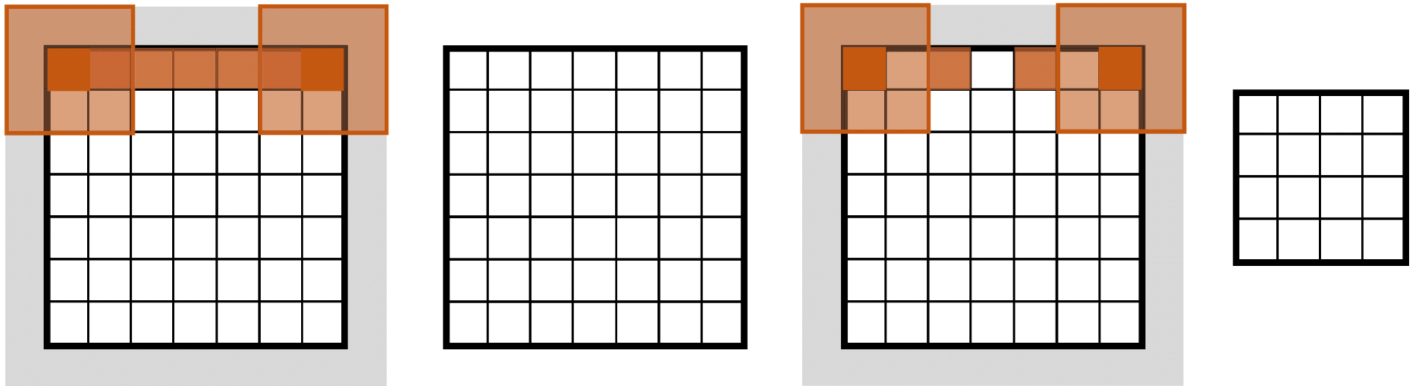


7x7 input
3x3 filter
stride 2
no padding

3x3 output

Figure: adapted from Fei Fei et al.

Padding and strides



7x7 input
3x3 filter
stride 1
zero padding

7x7 output

7x7 input
3x3 filter
stride 2
zero padding

4x4 output

Figure: adapted from Fei Fei et al.

Computational complexity

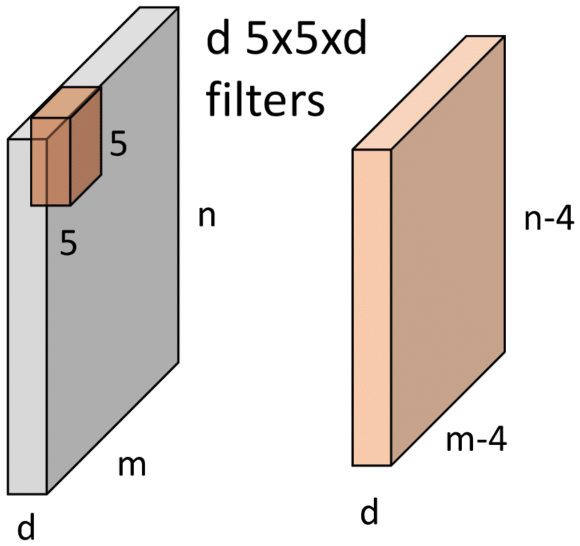


Figure: adapted from Fei Fei et al.

Factorized convolution

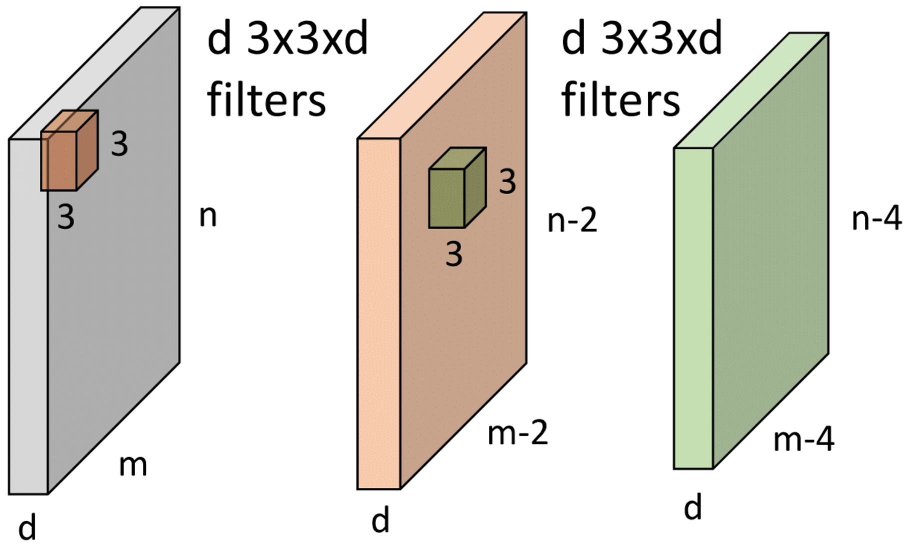
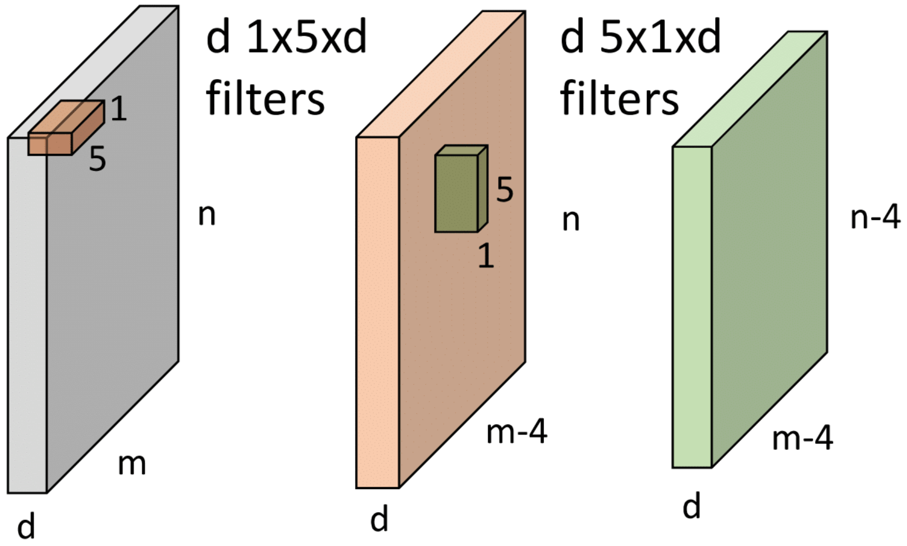


Figure: adapted from Fei Fei et al.

Separable convolution



e.g. m=n=32, d=3
32x28x3x5x3+1 +
28x28x3x5x3+1 =
75602 ops

vs. 5x5
28x28x3x5x5x3+1=
176401 ops

Figure: adapted from Fei Fei et al.

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

Figure: adapted from Fei Fei et al.

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	

Figure: adapted from Fei Fei et al.

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	8

Figure: adapted from Fei Fei et al.

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	8
5	

Figure: adapted from Fei Fei et al.

Pooling

- Permutation-invariant aggregation+downsampling (typically max or avg)
- Reduces resolution
- Hierarchical features
- Contributes to approximate shift/deformation invariance

6	1	2	4
1	6	7	8
3	5	2	0
1	2	3	4

6	8
5	4

Figure: adapted from Fei Fei et al.

Pooling

- Applied to each channel separately

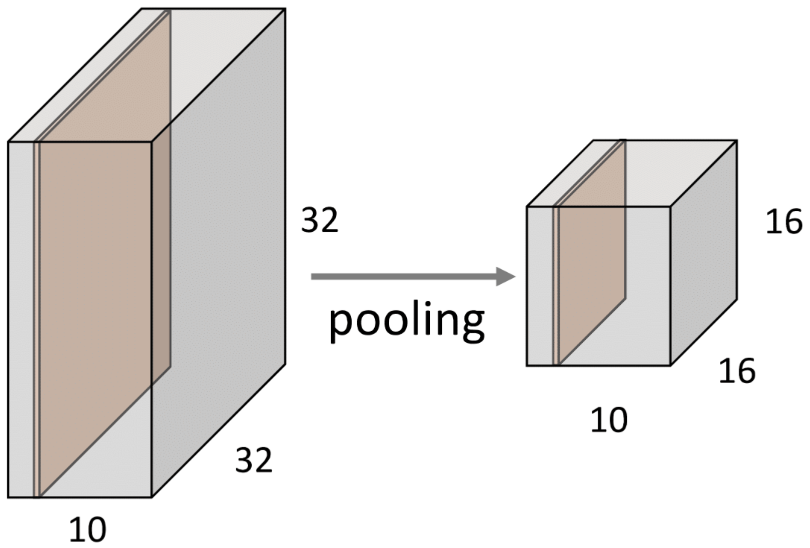
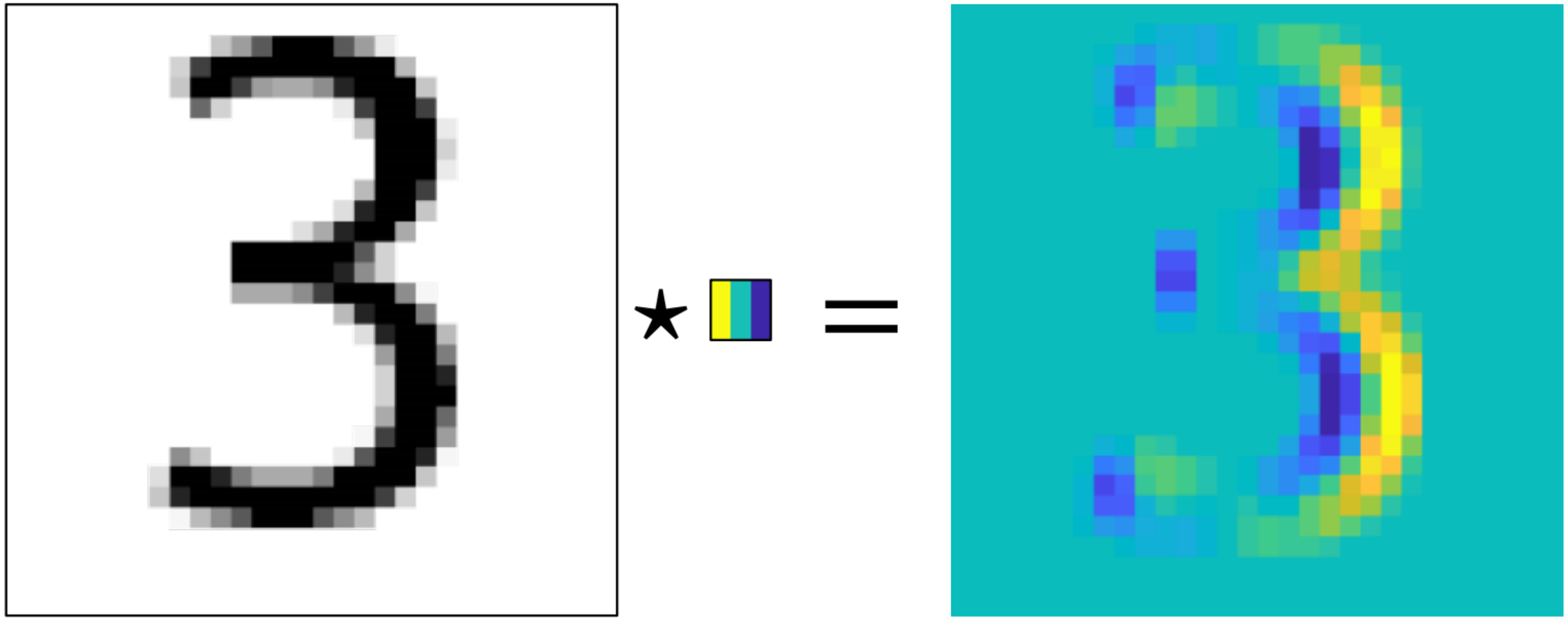


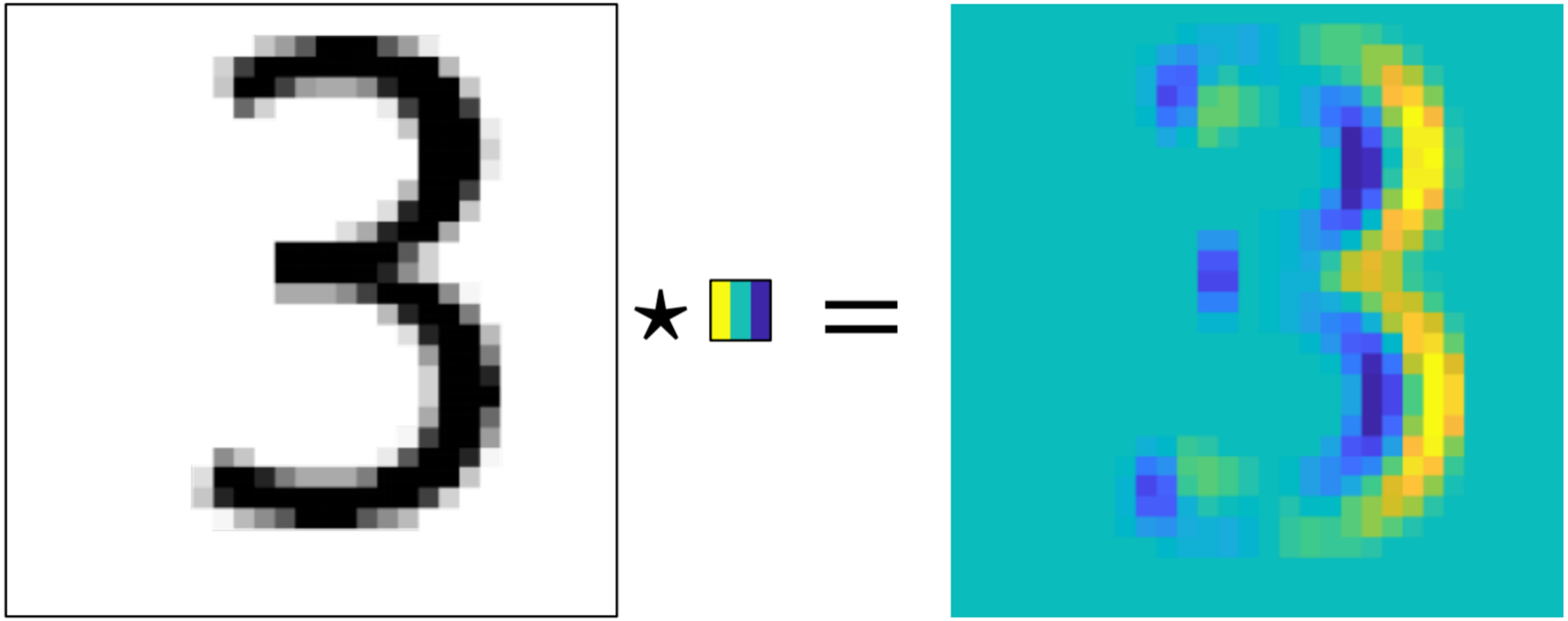
Figure: adapted from Fei Fei et al.

Equivariance in CNNs



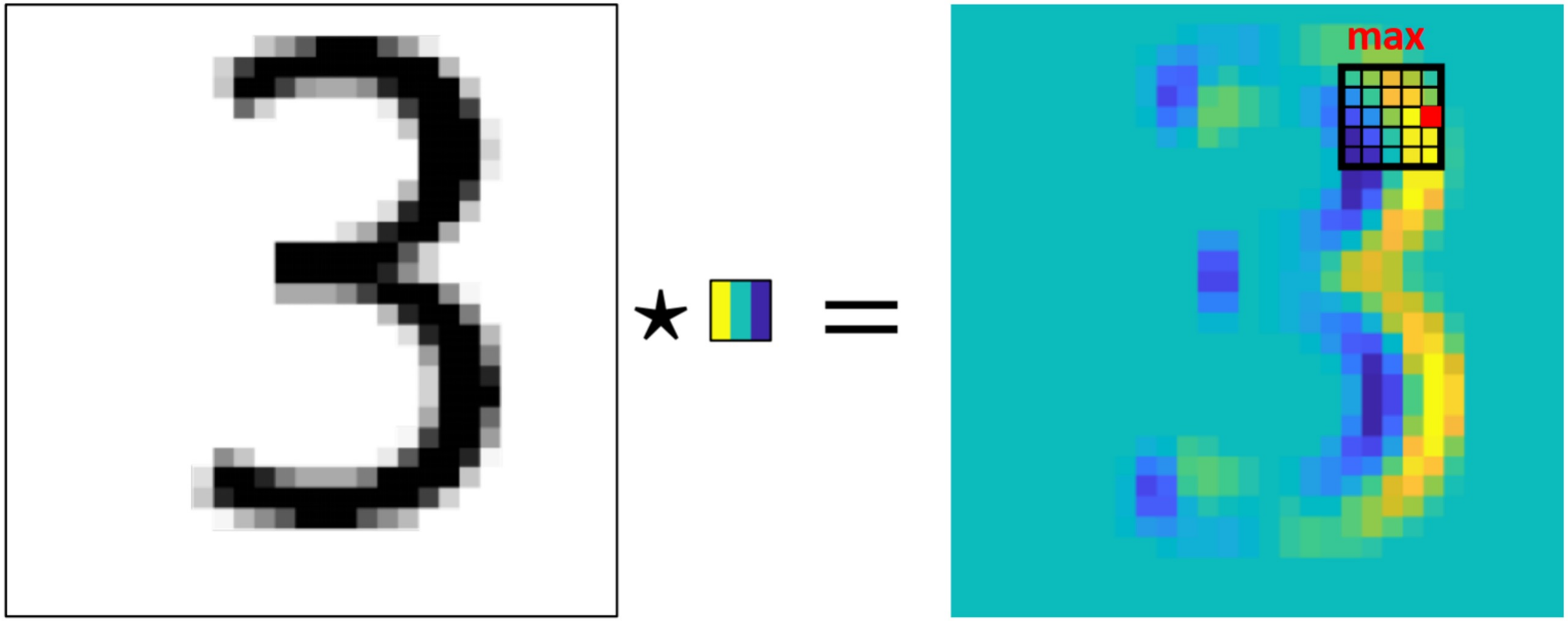
Output of convolutional layer (shift equivariant)

Equivariance in CNNs



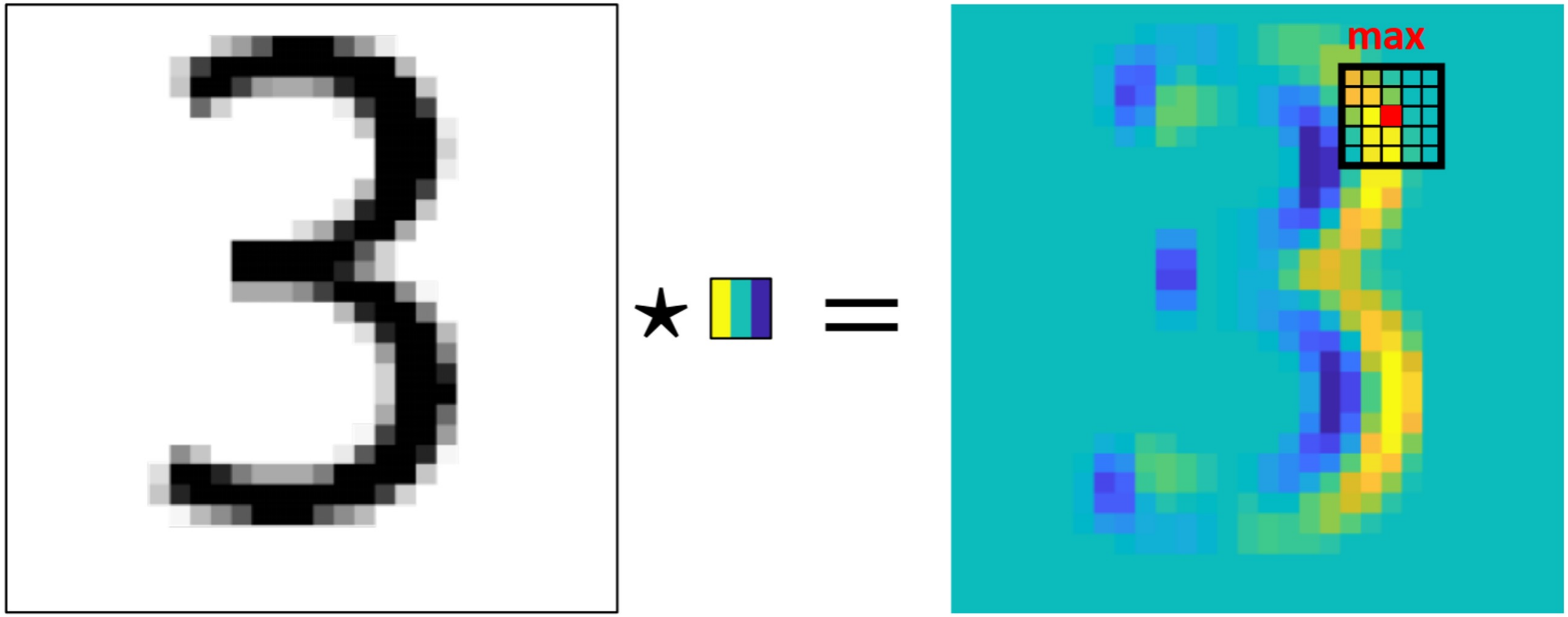
Output of convolutional layer (shift equivariant)

Approximate invariance in CNNs with pooling



Output of convolutional layer+max pooling (~shift invariant)

Approximate invariance in CNNs with pooling

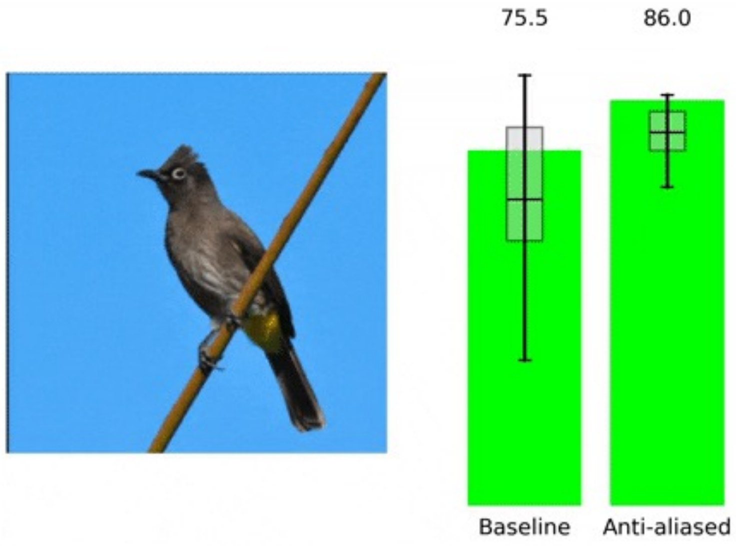


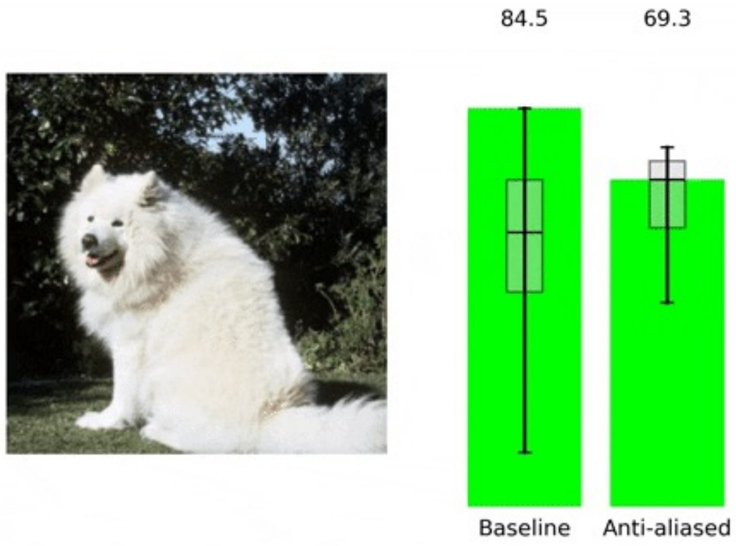
Output of convolutional layer+max pooling (~shift invariant)

Not the full story…

- But striding ignores the Nyquist sampling theorem and aliases

Nyquist sampling theorem = sample at least twice as fast to keep all information



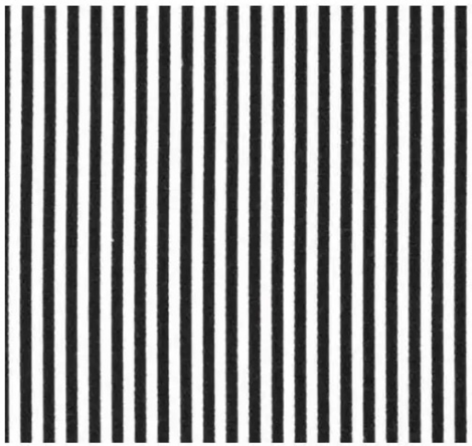


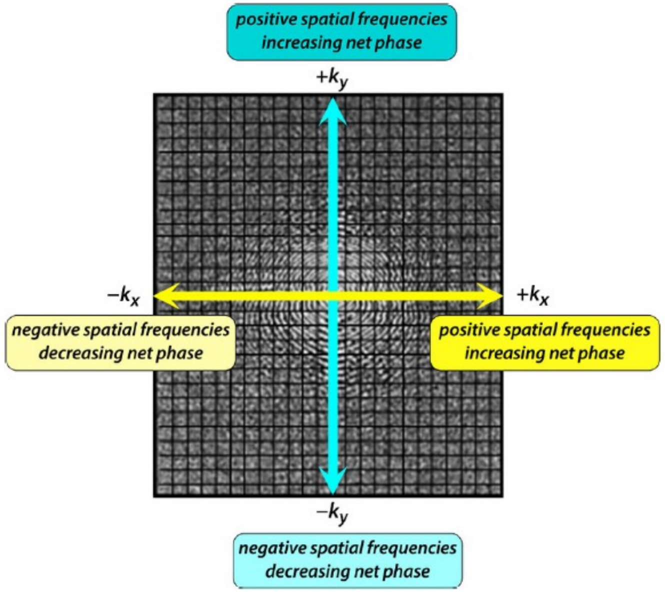
R. Zhang.
Making Convolutional Networks Shift-Invariant Again.
In ICML, 2019.

frequencies in images

- https://www.youtube.com/watch?v=js4bLBYtJwY

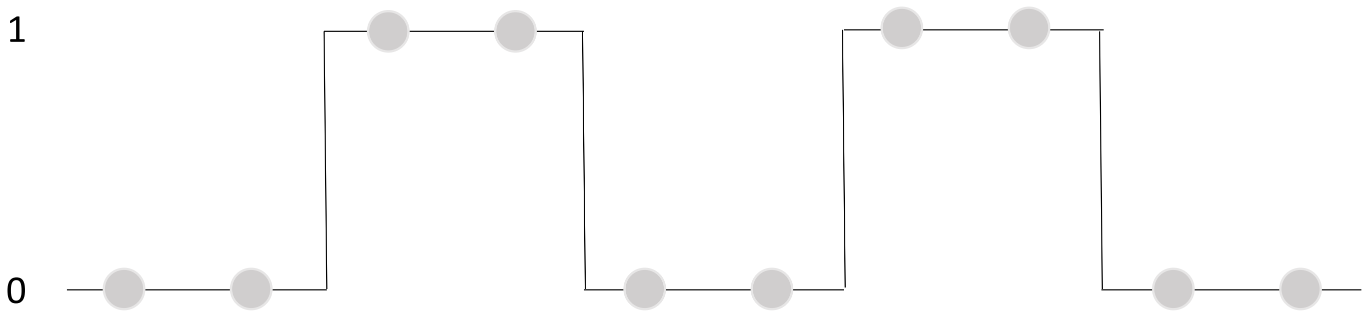
- https://medium.com/@shashimalsenarath.computer-vision-d179b8c0f723





Simple example

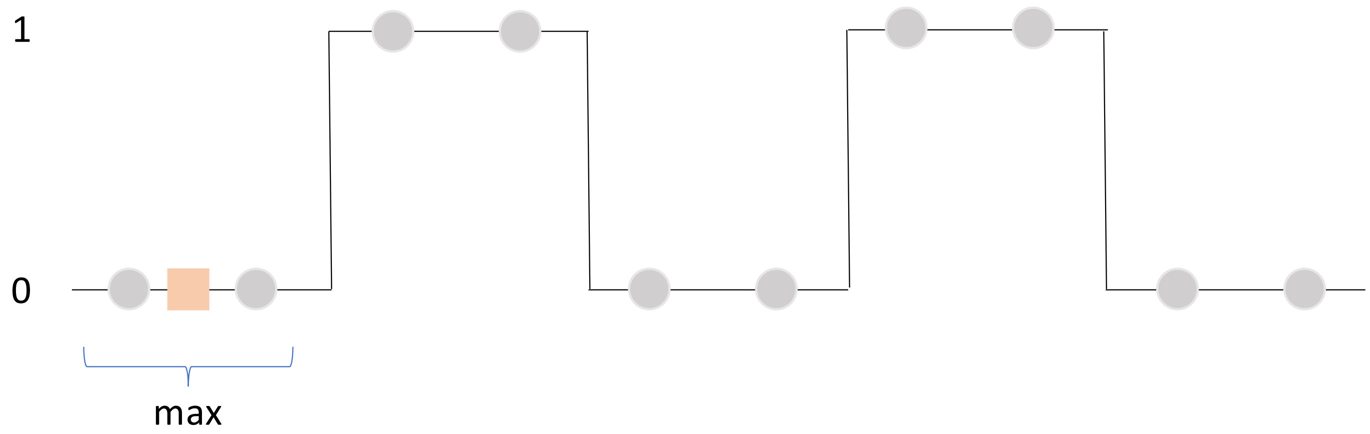
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

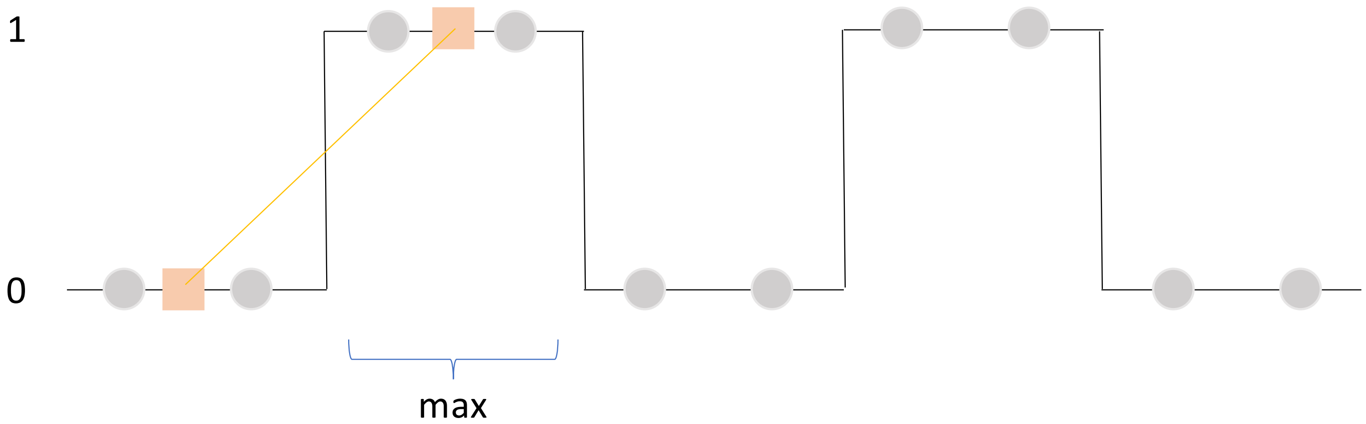
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

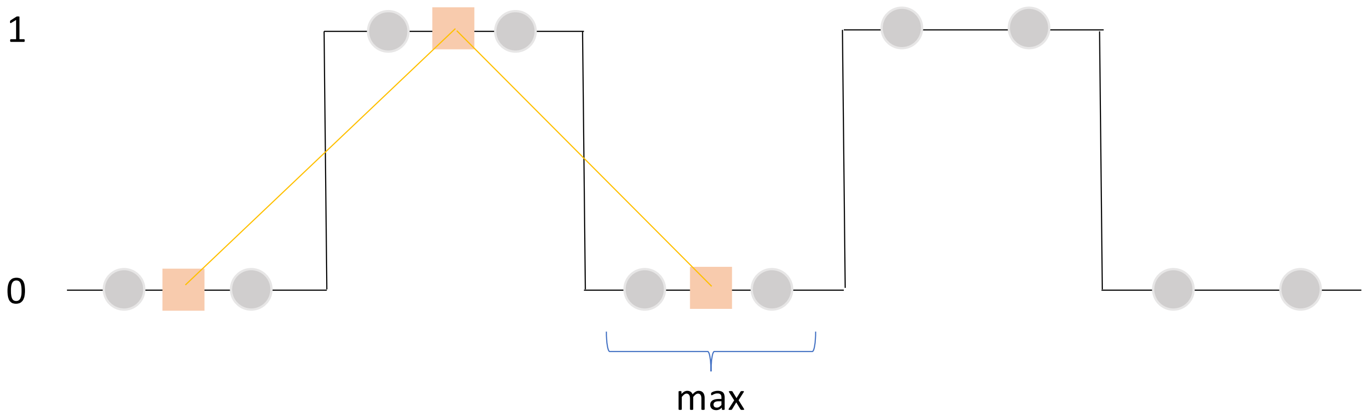
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

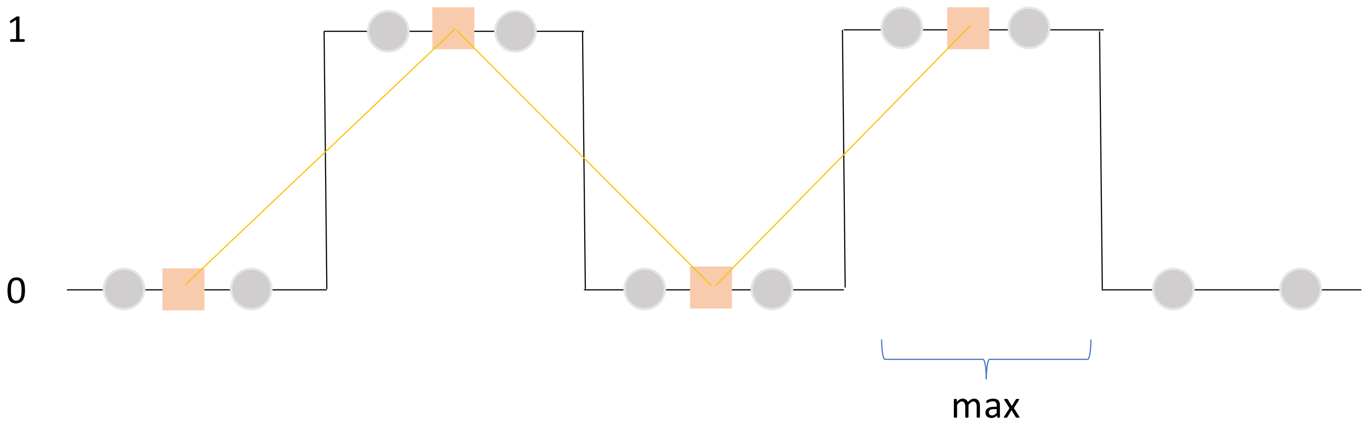
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

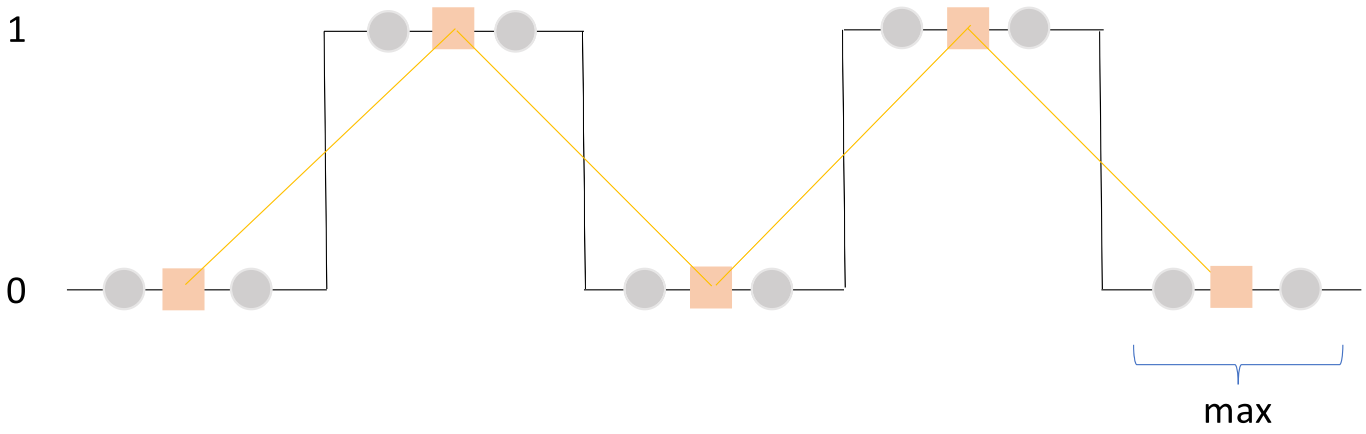
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

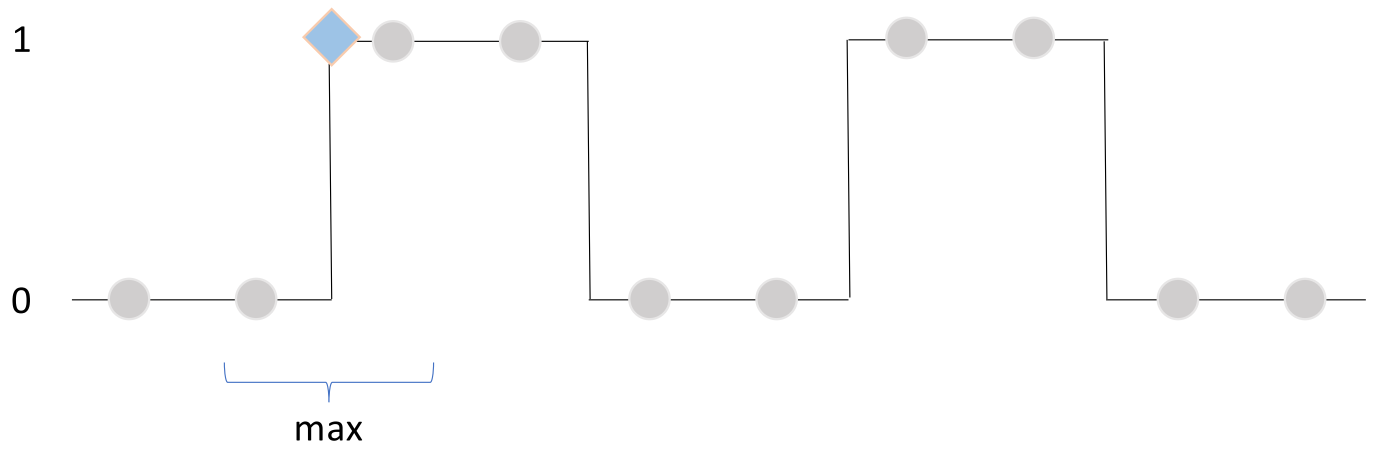
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

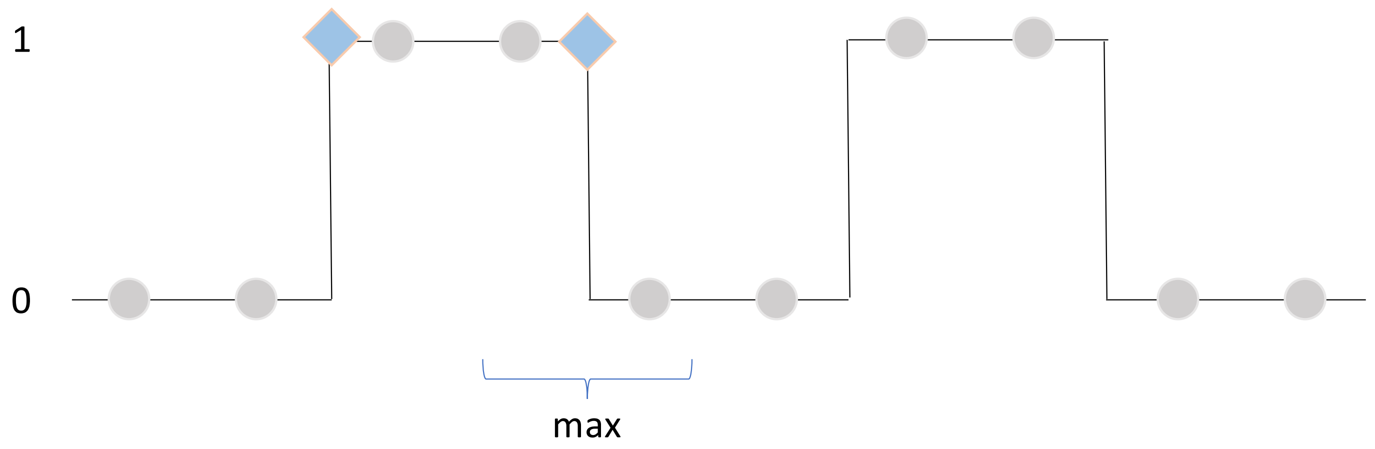
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

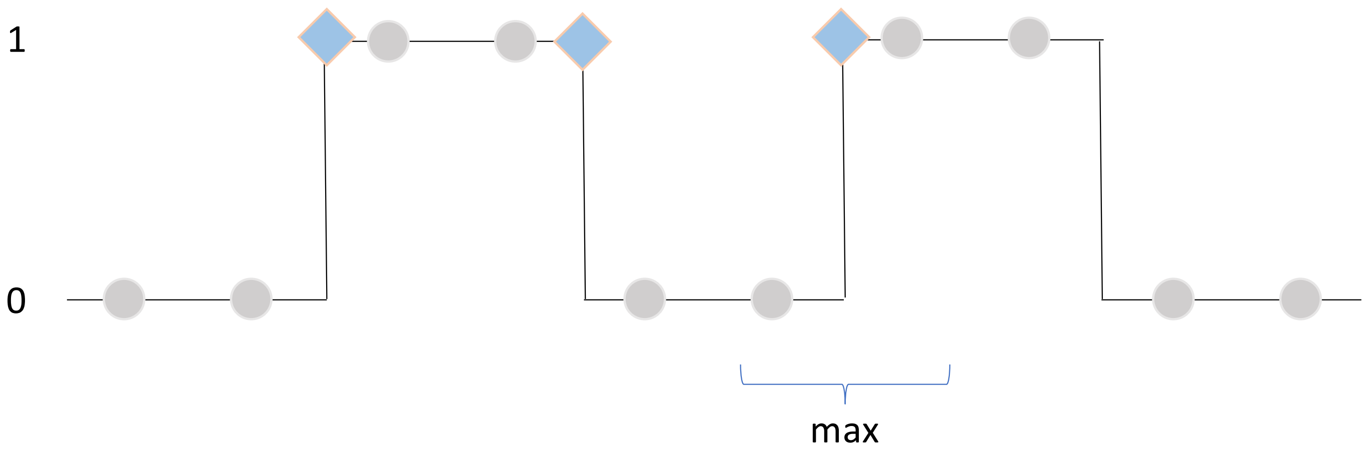
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

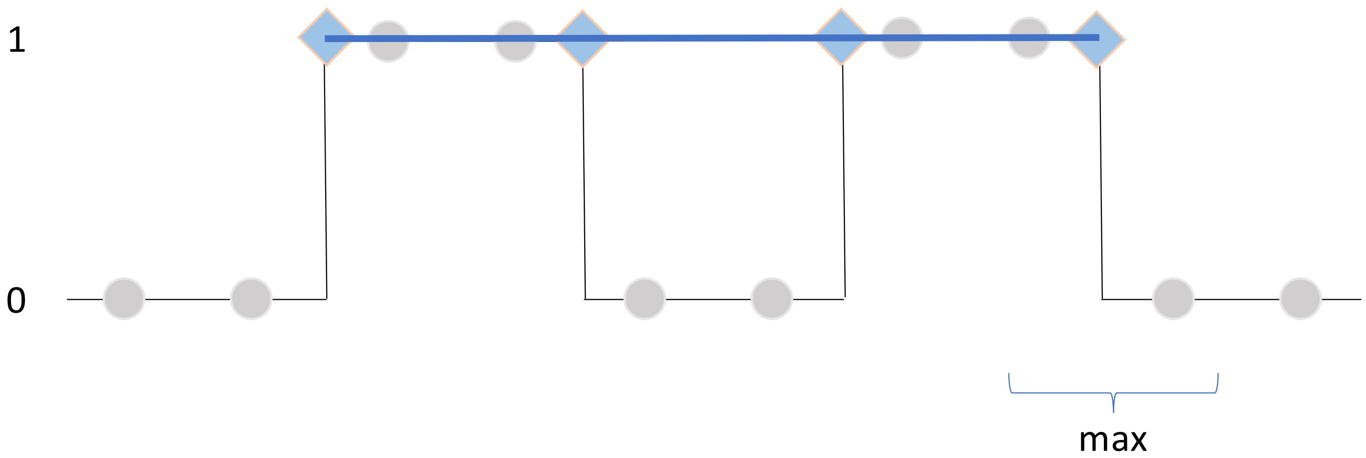
- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

- Max-pooling breaks shift equivariance



https://www.youtube.com/watch?v=eZa56DqXTHg

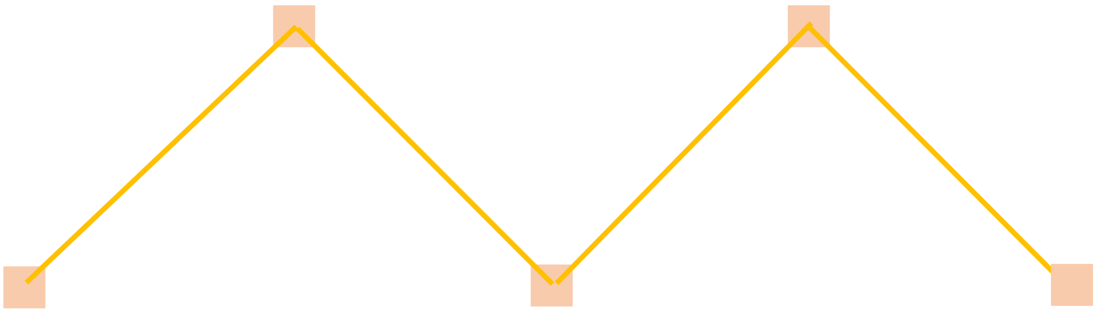
Simple example

- Max-pooling breaks shift equivariance

https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

- Max-pooling breaks shift equivariance

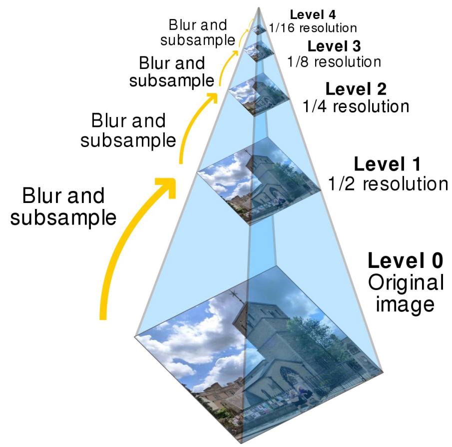


https://www.youtube.com/watch?v=eZa56DqXTHg

Simple example

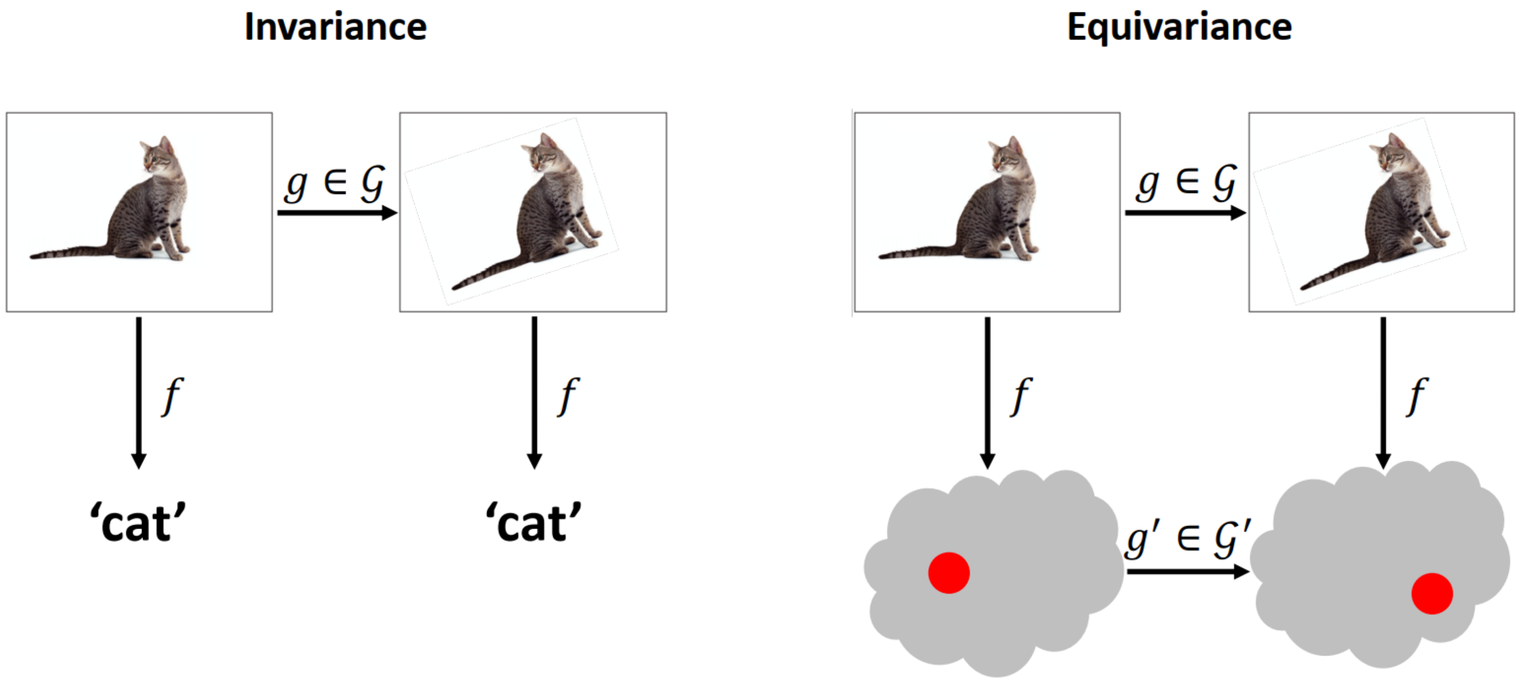
- Max-pooling breaks shift equivariance
- Partial solution: use what you learned about anti-aliasing in Computer Vision: blur and then down sample

R. Zhang.
Making Convolutional Networks Shift-Invariant Again.
In ICML, 2019.



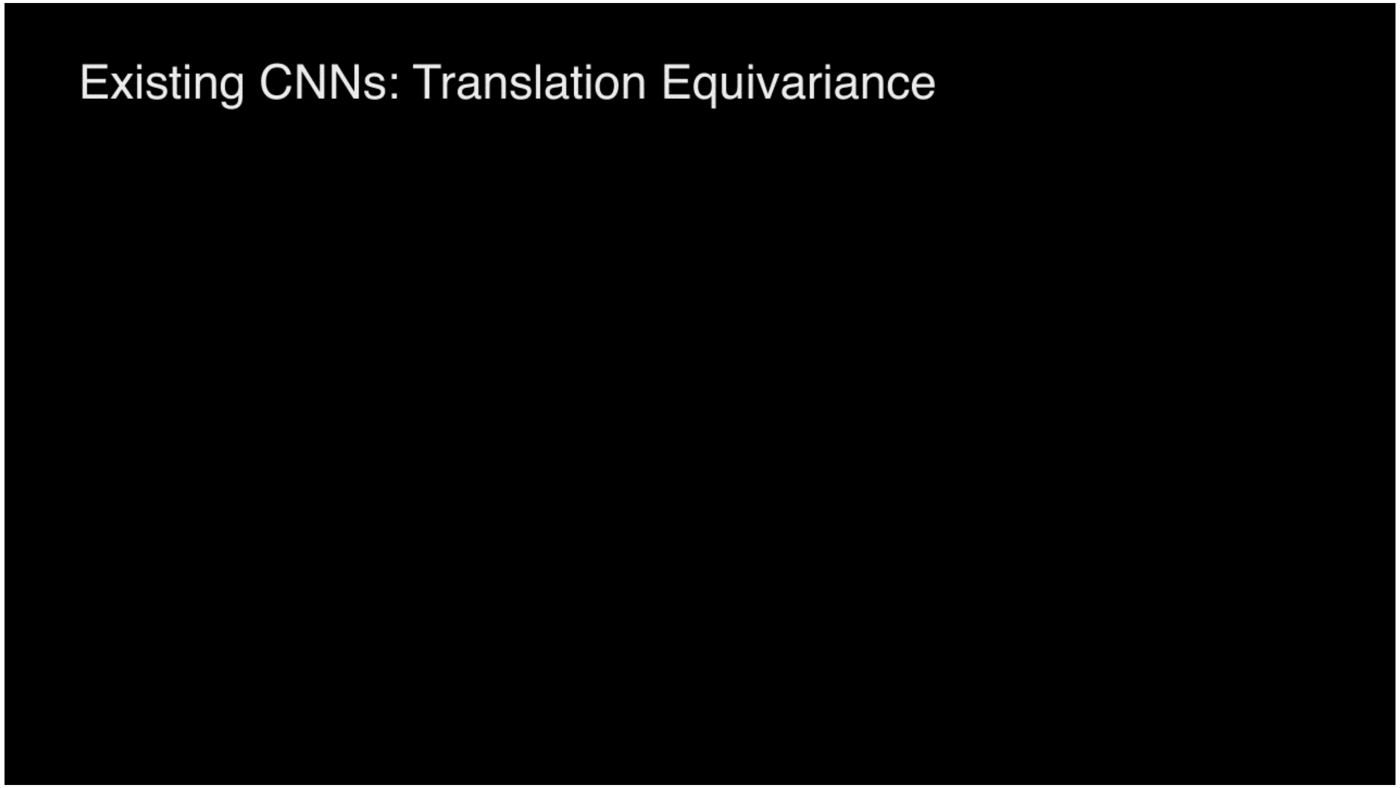
https://www.youtube.com/watch?v=eZa56DqXTHg

Beyond shifts: group equivariance



$$f\big(g(\mathbf{x})\big) = g'\big(f(\mathbf{x})\big)$$

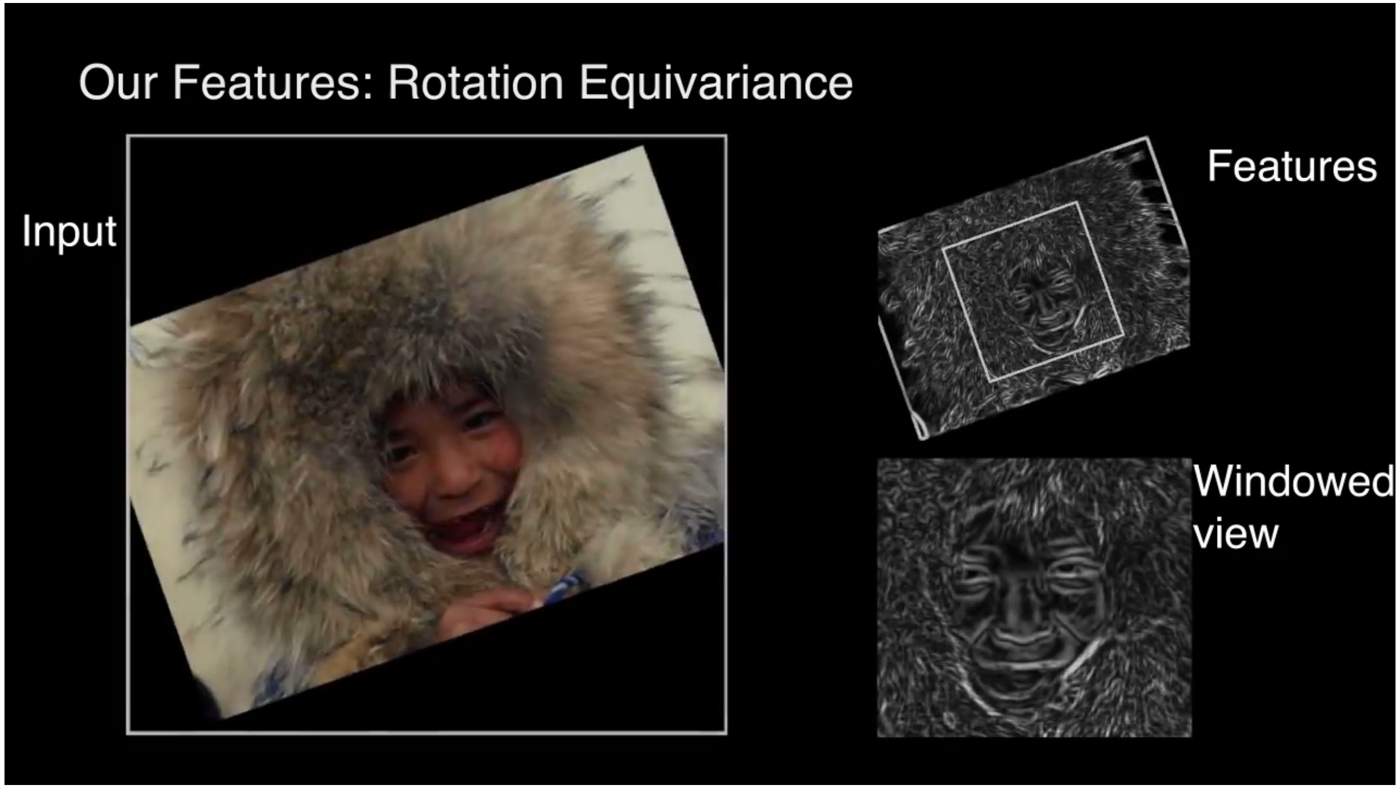
Rotation invariant CNNs



Daniel Worrall et al.: Harmonic Networks: Deep Translation and Rotation Equivariance

https://www.youtube.com/watch?v=qoWAFBYOtoU

Rotation invariant CNNs

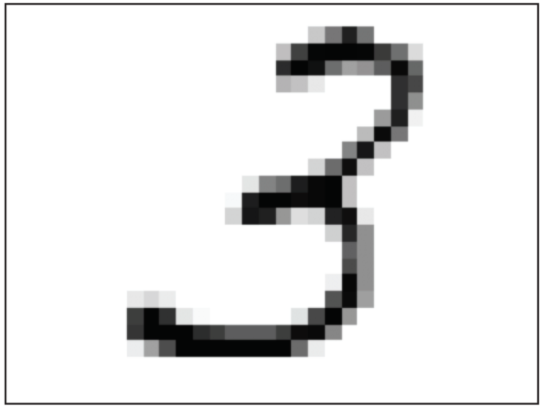


Daniel Worrall et al.: Harmonic Networks: Deep Translation and Rotation Equivariance

https://www.youtube.com/watch?v=qoWAFBYOtoU

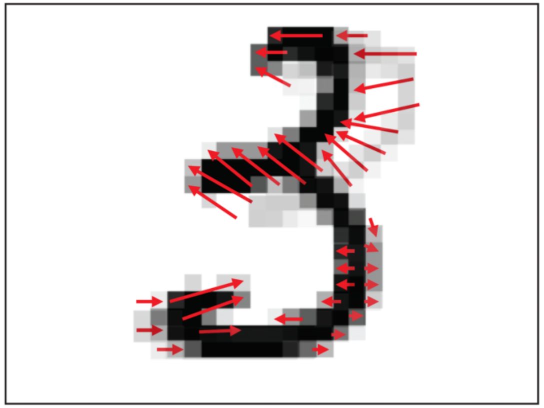
Approximate deformation invariance

Input $\mathbf{x}$



Output $f(\mathbf{x}) = 1$

Shifted input $S_v\mathbf{x}$



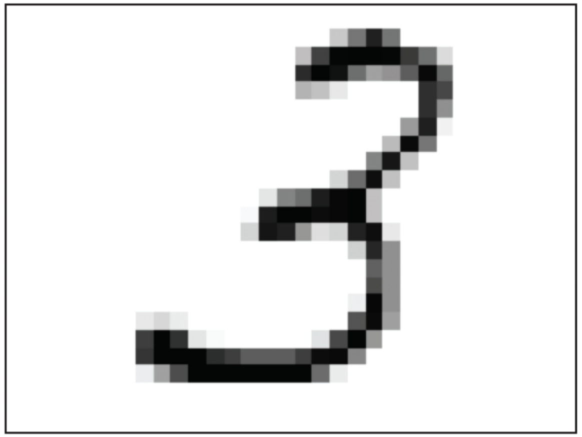
Output $f(D_{\boldsymbol{\tau}}\mathbf{x}) = 1$

- **'Digit 3 detector'** $f: \mathbb{R}^d \to \mathbb{R}$
- **Warp operator** $D_{\boldsymbol{\tau}}: \mathbb{R}^d \to \mathbb{R}^d$ warping the image by field $\boldsymbol{\tau}$

Deformation invariance: $f(\mathbf{x}) \approx f(D_{\boldsymbol{\tau}}\mathbf{x})$

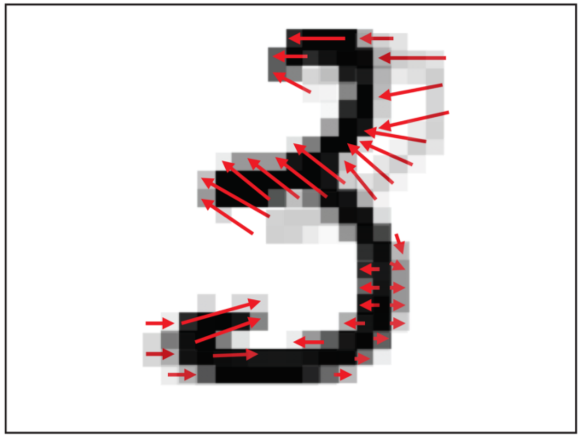
Approximate deformation invariance

Input $\mathbf{x}$



Output $f(\mathbf{x}) = 1$

Shifted input $S_v\mathbf{x}$



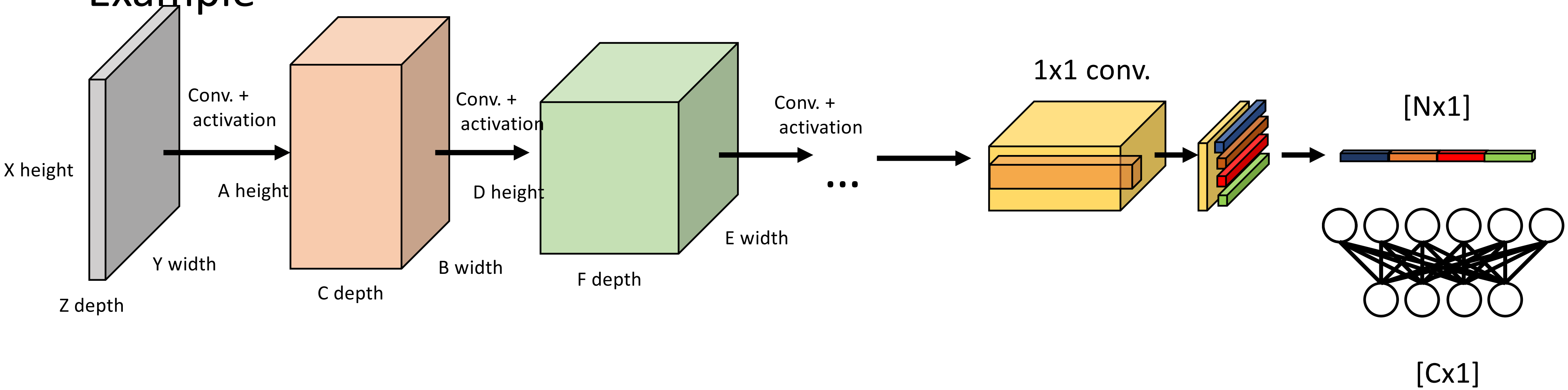
Output $f(D_{\boldsymbol{\tau}}\mathbf{x}) = 1$

- **'Digit 3 detector'** $f: \mathbb{R}^d \to \mathbb{R}$
- **Warp operator** $D_{\boldsymbol{\tau}}: \mathbb{R}^d \to \mathbb{R}^d$ warping the image by field $\boldsymbol{\tau}$

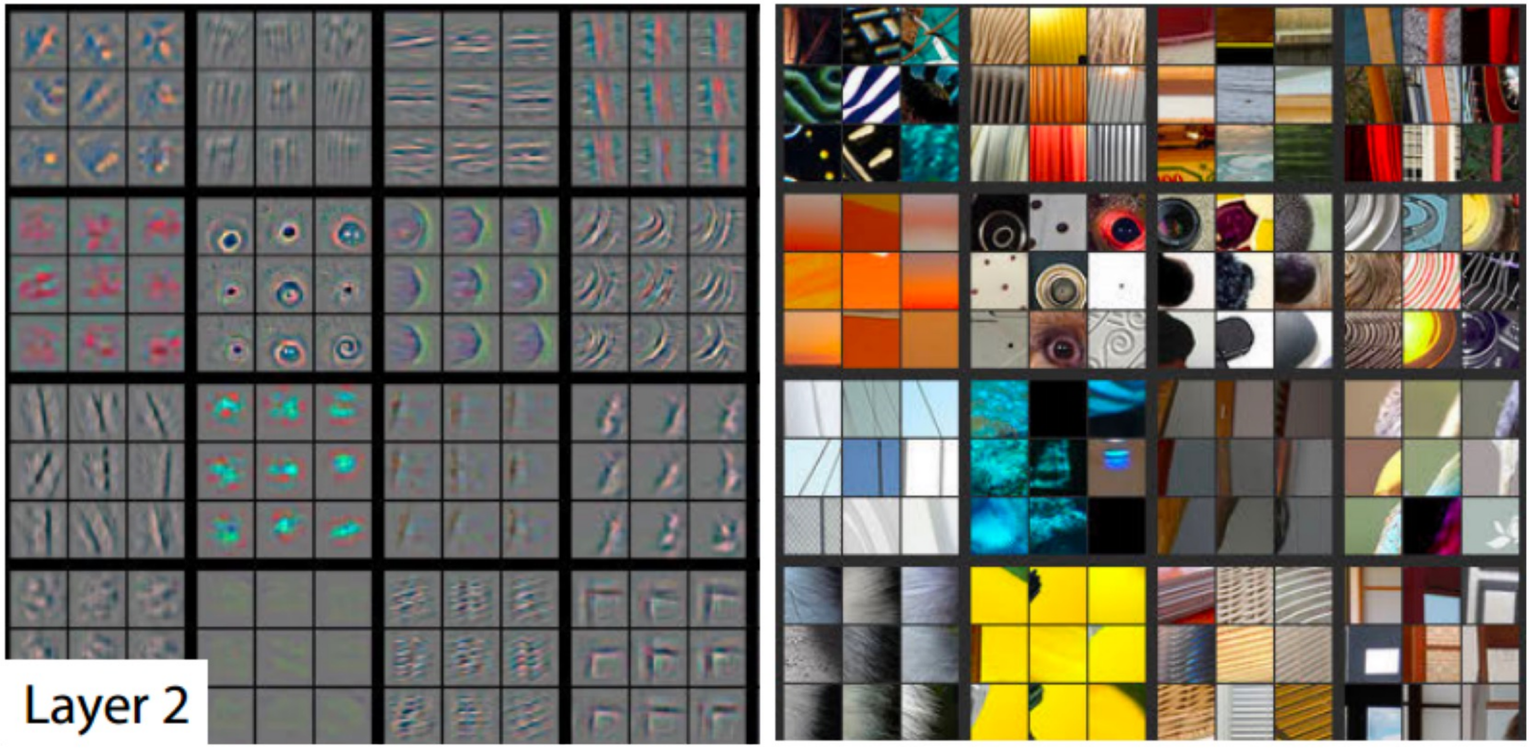
$$||f(\mathbf{x}) - f(D_{\boldsymbol{\tau}}\mathbf{x})|| \approx ||\nabla \boldsymbol{\tau}||$$

Flattening

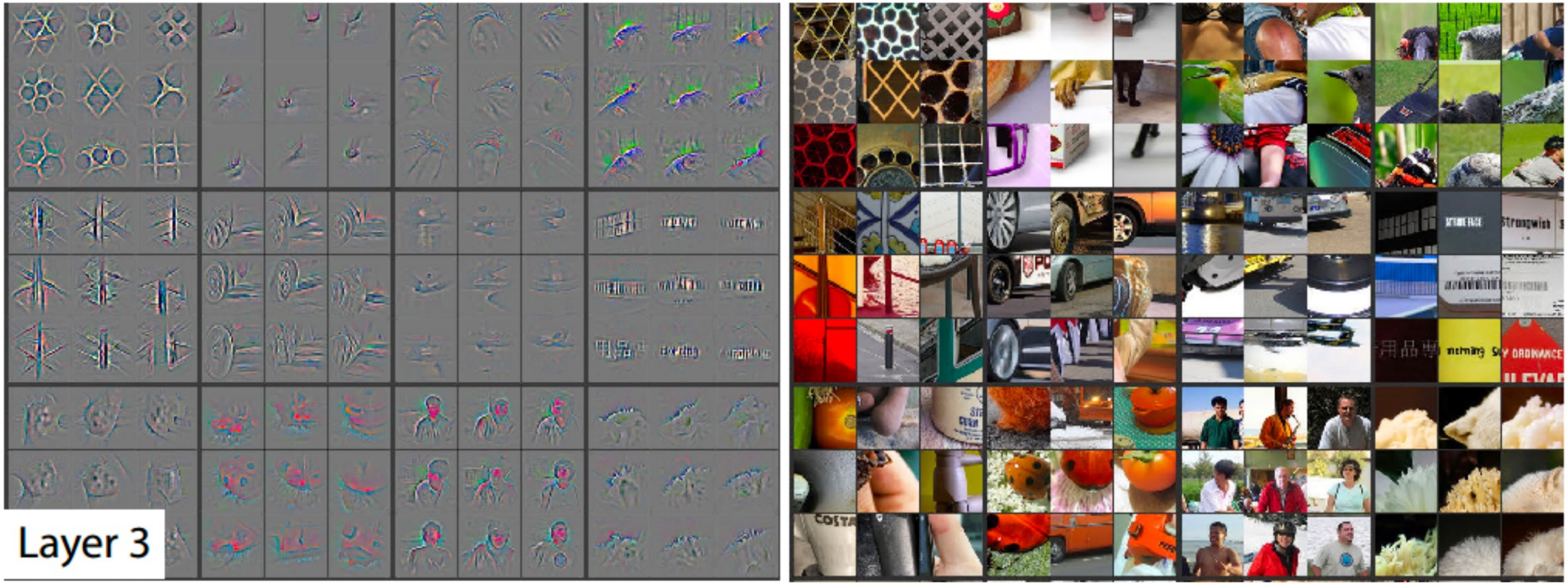
- Example



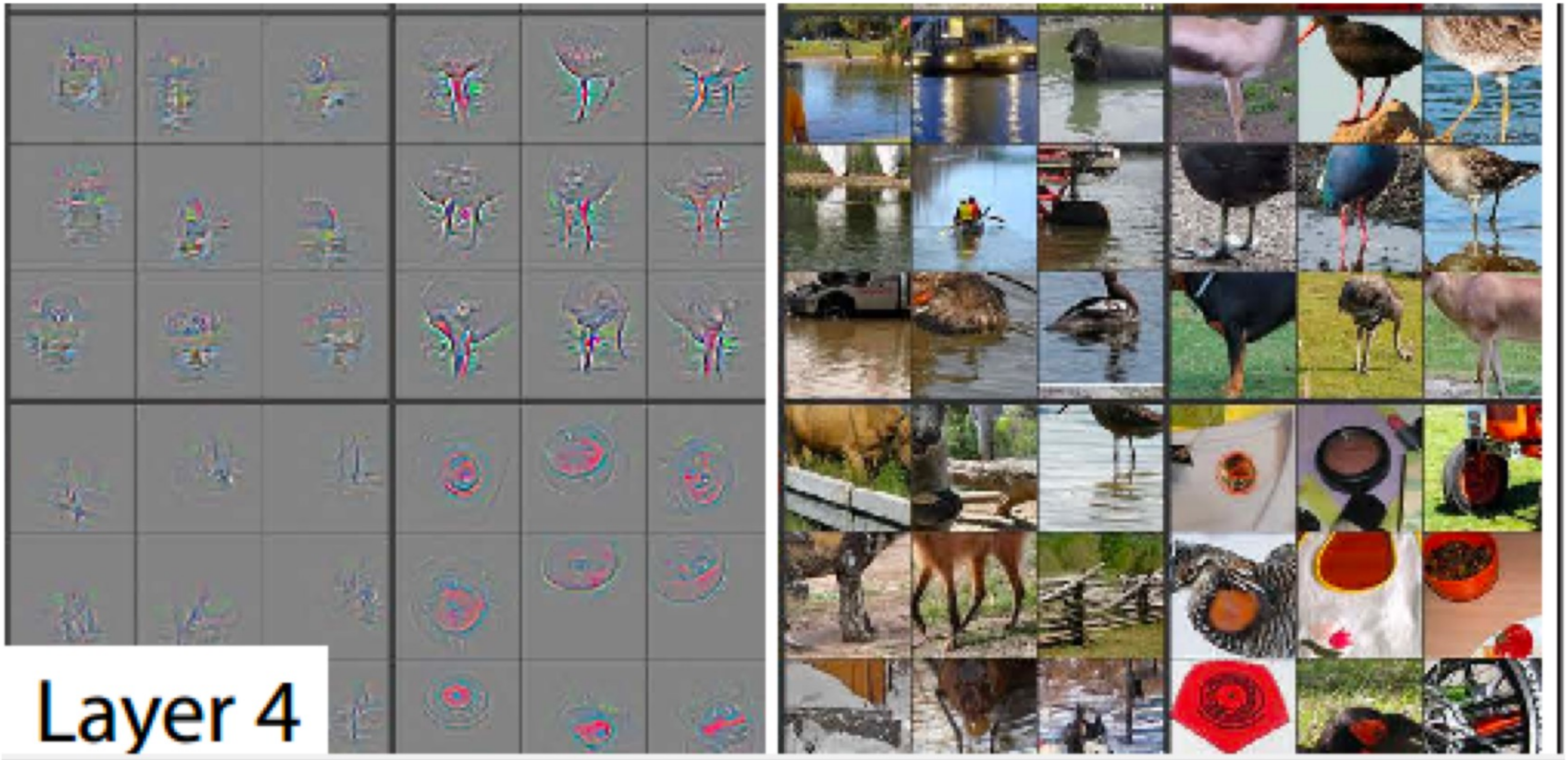
What CNNs learn?



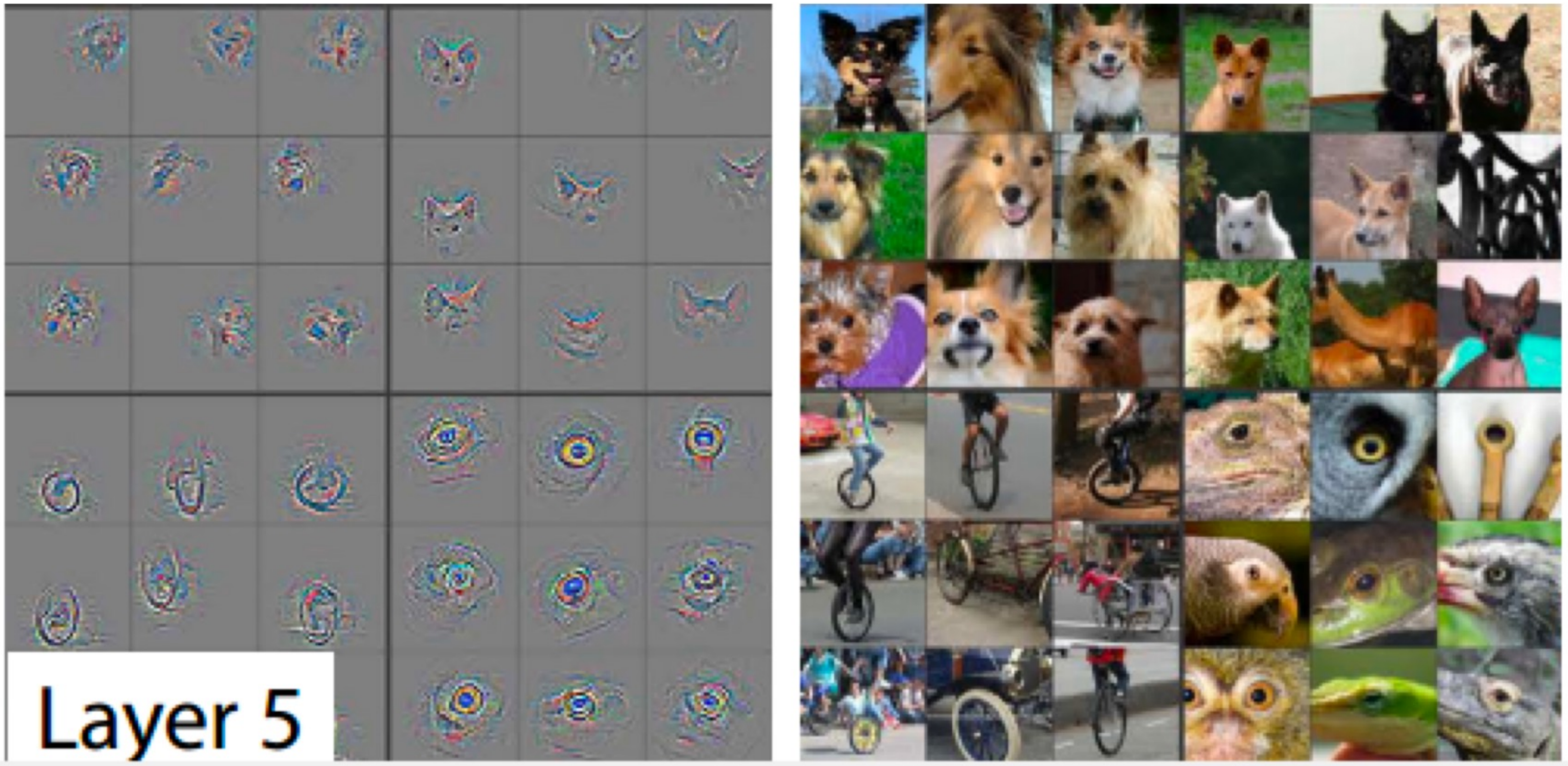
What CNNs learn?



What CNNs learn?

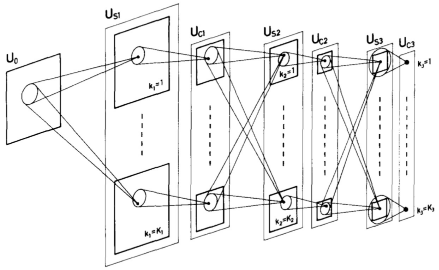


What CNNs learn?



Neocognitron

Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition <u>Unaffected by Shift in Position</u>



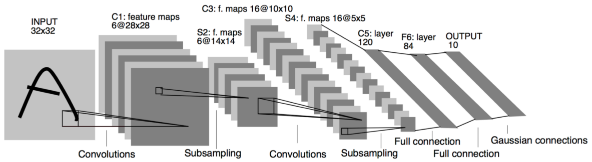
K. Fukushima

Fukushima 1980

Lacks backprop

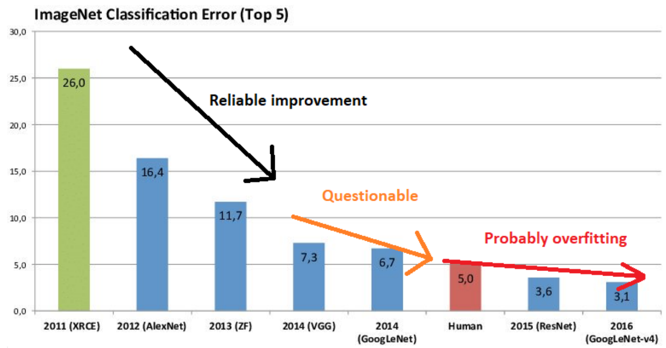
Gradient-Based Learning Applied to Document Recognition

YANN LECUN, MEMBER, IEEE, LÉON BOTTOU, YOSHUA BENGIO, AND PATRICK HAFFNER



LeCun et al. 1998

Adds backprop



Success in 2012!

No GPUs, no success for larger problems…

what do we learn from that?

- a) feature selection is important to build good representations. As we will see, the key of deep learning is to learn this feature selection instead of doing it manually.
- b) finding the right amount of features is key. Too few or too many will have a severe impact on the generalization abilities of your predictor model. Too few is easy too understand but too many requires an intuition about sample sparsity in high-dimensional spaces.
- c) the more features we choose as input the sparser our training samples will be distributed in the feature space. This means that decision boundaries become really tight around the used training samples because they all live close to each other at the boundaries of the space and our model will overfit the training data.

what do we learn from that?

- a) weight sharing reduces the number of parameters from n^2 in a multi-layer perception to a small number, for example 3 as in our experiment or 3 by 3 image filter kernels or similar
- b) these filter kernels can be learned through back propagation exactly in the same way as you would train a multi-layer perception. Each layer may have many filter-kernels, so it will produce many filtered versions of the input with different filter functions.
- c) for real-valued functions, of a continuous or discrete variable, convolution differs from cross-correlation only in that either f(x) or g(x) is reflected about the y-axis; so it is a cross-correlation of f(x) and g(−x), or f(−x) and g(x).

what do we learn from that?

- a) convolutions can massively reduce the computational complexity of neural networks but the real power of CNNs is revealed when priors are implemented and for example spatial structure is preserved. This is also one of the reasons why CNNs have been so successful in Computer Vision
- b) CNNs are pipeline of learnable filters interleaved with nonlinear activation functions producing d-dimensional feature maps at every stage. Training works like a common neural network: initialise randomly, present exampled from the training database, update the filter weights through backpropagation by propagating the error back through the network.
- c) convolution and pooling can be used to reduce the dimensionality of the input data until it forms a small enough representation space for either traditional machine learning methods for classification or regression or to steer other networks to for example generate a semantic interpretation like a mask of a particular object in the input.