

# From Fully Connected to Convolutional Neural Networks

---

**Ju Sun**

Computer Science & Engineering  
University of Minnesota, Twin Cities

March 29, 2025

# Our roadmap

## **Covered: Fundamentals**

Fundamental belief: universal approximation theorem

Basics of numerical optimization

Training DNNs: basic methods and tricks

## **Now: Unstructured data: images, sequences, graphs**

Work with images: convolutional neural networks & applications

Work with sequences: recurrent neural networks & applications

Working with graphs: graph neural networks & applications

Transformers, large-language models, and foundation models

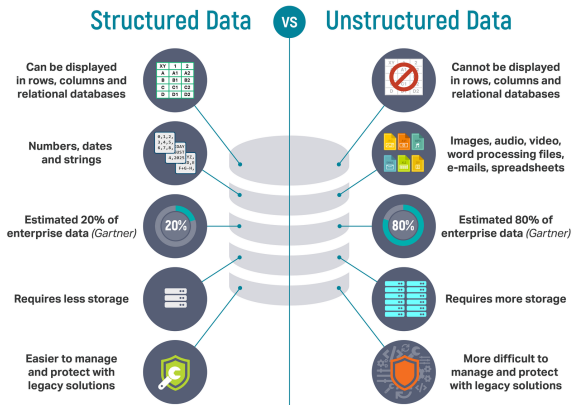
## **Generative/unsupervised/self-supervised/reinforcement learning**

Learning probability distributions: generative models

Learning representation without labels: dictionary learning, autoencoders, self-supervised learning

Gaming time: deep reinforcement learning

# Structured vs. unstructured data

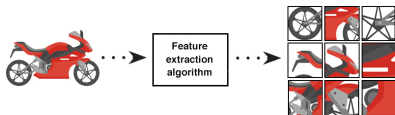


Credit: <https://lawtomated.com/>

[structured-data-vs-unstructured-data-what-are-they-and-why-care/](#)

- structured data also called **tabular data**
- structured data often directly fed into classical ML tools
- the success of DL mostly lies at **learning useful features/patterns from unstructured data**, i.e., **representation learning**

# Feature engineering for unstructured data: old and new



**Feature engineering:** derive features for **efficient** learning

Credit: [Elgendy, 2020]

## Traditional learning pipeline



- feature extraction is "independent" of the learning models and tasks
- features are handcrafted and/or learned

## Modern learning pipeline



- end-to-end DNN learning

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

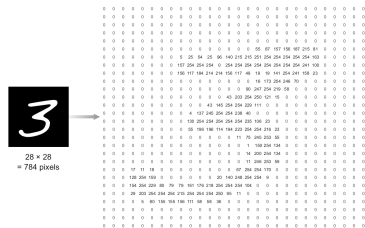
Thanks to the cats

Architectures for classification

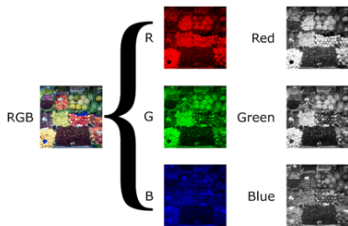
Practical tips

Suggested reading

# Digital images



(Credit: [Elgendy, 2020])



(Credit: Wikipedia)

- pixels: entries in the matrix or tensors
- bit/pixel-depth  $2^n$  (typical  $2^8$ , i.e., ranging from 0 to  $2^8 - 1 = 255$ )
- compression formats: PNG, JPEG (JPG), SVG, GIF, JPEG2000, etc
- normalization:  $/(2^n - 1)$ , zero-mean unit-variance (over a batch of images), min-max

## How to find a pattern in images?

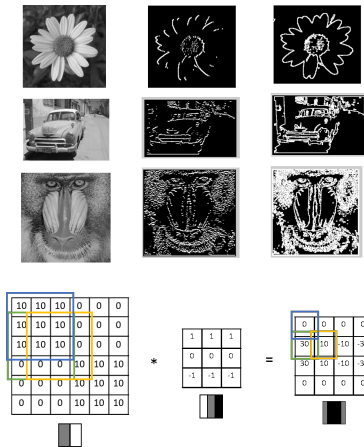


- Each time **inner product** of the original (red) and overlapped (green) patches (i.e., matrices) are taken
- The output matrix is the **correlation**
- Position(s) with the **largest magnitude** is candidate match—**detection**
- Care about the **largest magnitude only** if only interested in Yes/No—**max pooling**

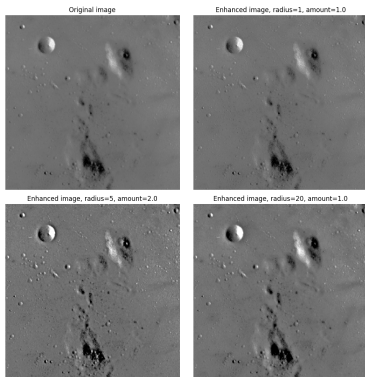
BTW, anything wrong with this?

# Template matching prevails in (classic) image processing

## edge detection



## image sharpening

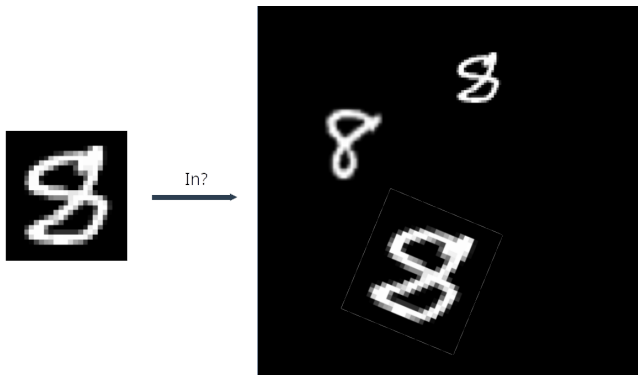


$$x' = x + \beta(x - k * x) \quad k: \text{blur kernel}$$

(Credit: scikit-image)



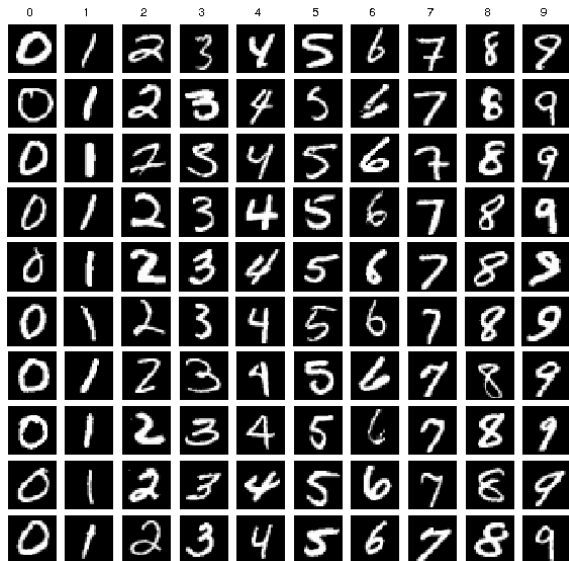
## Problem with template matching



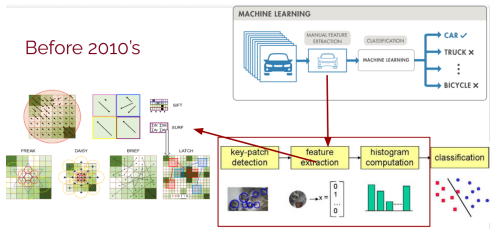
It handles the uncertainty about location (i.e., translation), but not

- not rotation or scaling
- local deformation

## Do we have a template at all?



# Feature-based approach!



Method	NL <sup>†</sup>	SR <sup>†</sup>	RC <sup>†</sup>	TL <sup>†</sup>	mAA(5°) <sup>†</sup>	mAA(10°) <sup>†</sup>	ATE <sup>‡</sup>	Rank
CV-SIFT	2577.6	96.7	94.1	3.95	.5309	.6261	.4721	14
VL-SIFT	3030.7	97.9	95.4	4.17	.5273	.6283	.4669	13
VL-Hessian-SIFT	3209.1	97.4	94.1	4.13	.4857	.5866	.5175	16
VL-DoGAff-SIFT	3061.5	98.0	96.2	4.11	.5263	.6296	.4751	12
VL-HesAffNet-SIFT	3327.7	97.7	95.2	4.08	.5049	.6069	.4897	15
CV- $\sqrt{\text{SIFT}}$	3312.1	98.5	96.6	4.13	.5778	.6765	.4485	9
CV-SURF	2766.2	94.8	92.6	3.47	.3897	.4846	.6251	18
CV-AKAZE	4475.9	99.0	95.4	3.88	.4516	.5553	.5715	17
CV-ORB	3260.3	97.2	91.1	3.45	.2697	.3509	.7377	22
CV-FREAK	2859.1	92.9	91.7	3.53	.3735	.4653	.6229	20
L2-Net	3424.9	98.6	96.2	4.21	.5661	.6644	.4482	10
DoG-HardNet	4001.4	99.5	<b>97.7</b>	<b>4.34</b>	<b>.6090</b>	<b>.7096</b>	<b>.4187</b>	1
DoG-HardNetAmos+	3550.6	98.8	96.9	4.28	.5879	.6888	.4428	6
Key-Net-HardNet	3366.0	98.9	96.7	4.32	.5391	.6483	.4622	11
Key-Net-SOSNet	<b>5505.5</b>	100.0	<b>98.7</b>	<b>4.46</b>	.5989	<b>.7038</b>	.4286	2
GeoDesc	3839.0	99.1	97.2	4.26	.5782	.6803	.4445	8
ContextDesc	3732.5	99.3	97.6	4.22	<b>.6036</b>	<b>.7035</b>	<b>.4228</b>	3
DoG-SOSNet	3796.0	99.3	97.4	4.32	<b>.6032</b>	.7021	<b>.4226</b>	4
LogPolarDesc	4054.6	99.0	96.4	4.32	.5928	.6928	.4340	5
D2-Net (SS)	<b>5893.8</b>	<b>99.8</b>	97.5	3.62	.3435	.4598	.6361	21
D2-Net (MS)	<b>6759.3</b>	<b>99.7</b>	<b>98.2</b>	3.39	.3524	.4751	.6283	19
R2D2 (wasf-n8-big)	4432.9	<b>99.7</b>	97.2	<b>4.59</b>	.5775	.6832	.4333	7
DoG-AffNet-HardNet	4671.3	<b>99.9</b>	<b>98.1</b>	<b>4.56</b>	<b>.6296</b>	<b>.7267</b>	<b>.4021</b>	1*
DoG-MKD-Concat	3507.4	98.5	96.1	4.17	.5461	.6476	.4668	11*
DoG-TFeat	2905.3	97.1	94.8	4.04	.5270	.6261	.4873	14*

see the survey

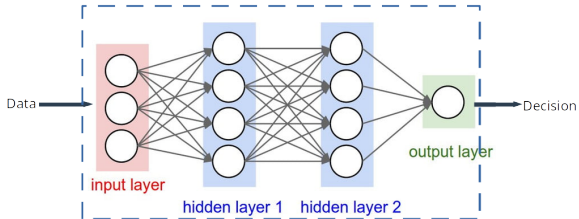
[Jin et al., 2020]

# Transition to representation learning

traditional learning pipeline



modern learning pipeline



# Outline

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

Architectures for classification

Practical tips

Suggested reading

# Complexity

Input sizes

image



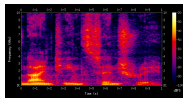
$\sim 10^6$

video



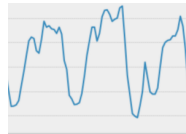
$\sim 10^8(10s)$

audio (spectrogram)



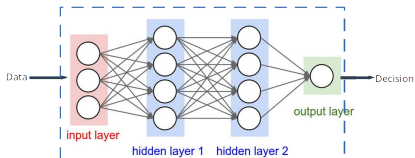
$\sim 10^8(10s)$

time series



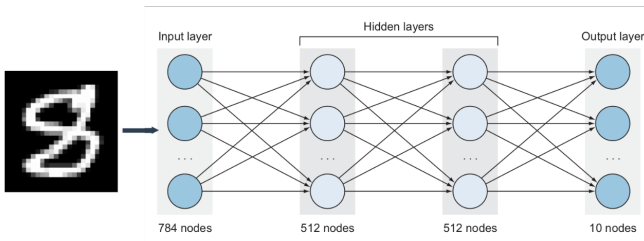
1/resol

100 hidden nodes at layer 1  $\implies$  10 billions variables in the first layer!



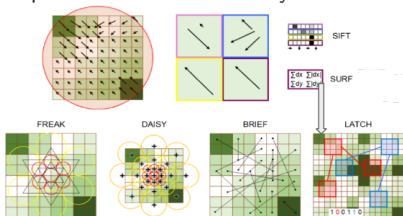
- storage: 80 billion bytes  $\sim$  80GB!
- computation
- data: need enough data to fit complex models

# Locality and ordering



Fully-connected neural network

spatial features are mostly localized!



Can we learn spatial features **easily**?

- FCNN treats the input as a **vector**
- FCNN is insensitive to any universal permutation of the coordinates to all inputs
- implication: ordering and locality are lost together

# Invariance



where the pattern is found shouldn't matter much

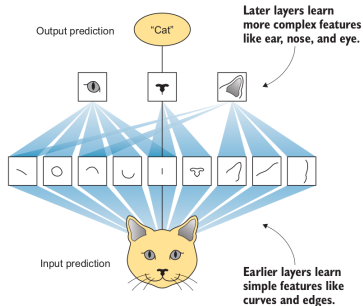
- For FCNN, all possible translated copies should be available for training
- Similarly for rotation, scaling, local deformation



# Ideal neural networks for spatial data

Problems with FCNNs: high **complexity** and lack of **locality** and **invariance**

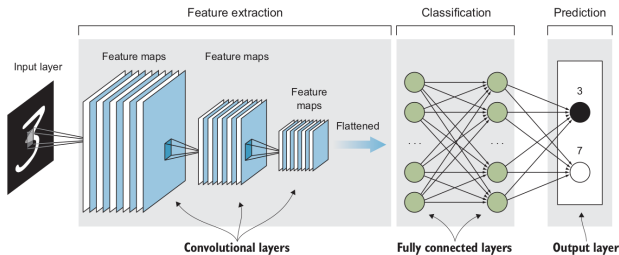
**Goal:** build these into the architecture directly



(Credit: [Elgandy, 2020])

- Extracted features invariant to translation, rotation, local deformation
- Low complexity

# A quick preview of convolutional neural networks (CNN)



(Credit: [Elgendy, 2020])

- Input layer
- **Convolutional layers** for feature extraction
- FC layers for classification
- Output layer for prediction

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

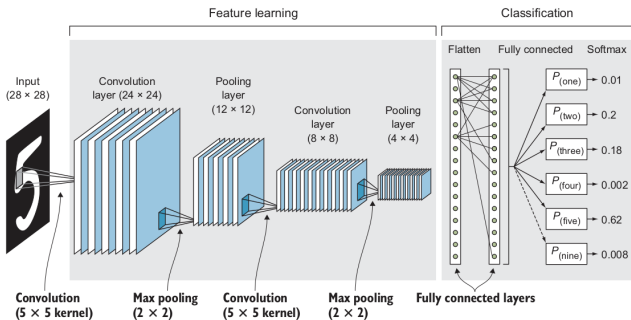
Thanks to the cats

Architectures for classification

Practical tips

Suggested reading

# A closer look at CNNs



(Credit: [Elgandy, 2020])

- convolutional layers
- pooling layers
- fully-connected layers

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

Architectures for classification

Practical tips

Suggested reading

# Convolution is a misnomer!

## 2D Correlation

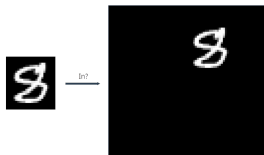
Initial position for $w$	Correlation result	Full correlation result
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9 & 8 & 7 \\ 0 & 6 & 5 & 4 \\ 0 & 3 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 8 & 7 & 0 \\ 0 & 0 & 6 & 5 & 4 & 0 \\ 0 & 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

## 2D Convolution

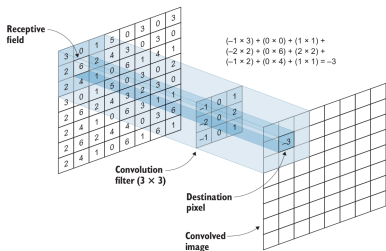
2D flipped $w$	Convolution result	Full convolution result
$\begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

- The only difference is the flipped template
- People actually implement correlation (not convolution; they're equivalent from learning perspective—the template is to be learned!)
- Math notations:  $*$  for convolution, and  $\star$  for (cross)-correlation

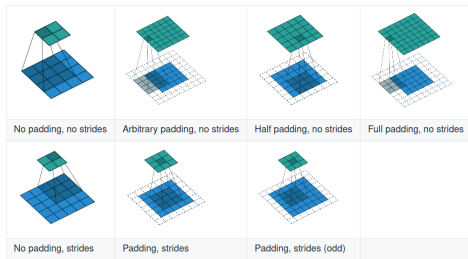
Is correlation/convolution a surprise? locality and translation invariance (when coupled with max pooling)



# More on convolution/correlation



(Credit: [Elgandy, 2020])

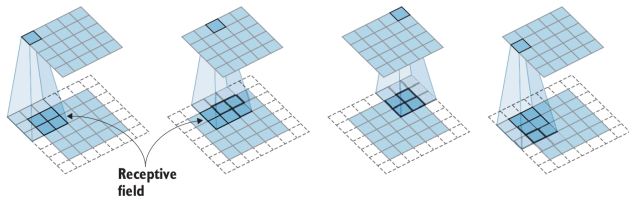


[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

## Key concepts:

- filter/kernel
- inner product  $\langle \cdot, \cdot \rangle$  at each location
- (zero)-padding—dealing with boundaries
- strides/steps

# Connection to fully-connected NN



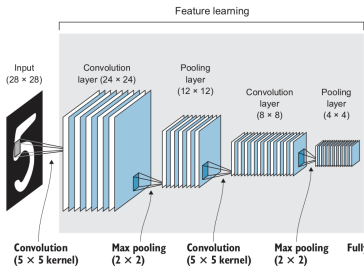
(Credit: [Elgandy, 2020])

input: a whole matrix    output: neuron outputs organized into a matrix

- **local/sparse connectivity**: each neuron connects only to its receptive field
- **weight sharing**: all neurons share the same weight pattern



# Multiple filters each layer



(Credit: [Elgandy, 2020])

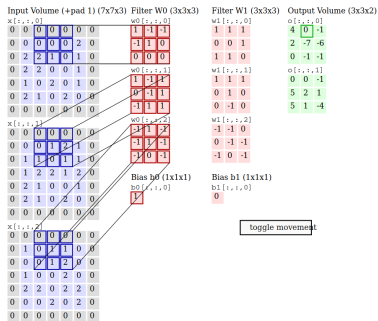
for the first conv layer:

- each filter generates an output, called **feature map**
- $k$  filters will generate  $k$  feature maps/**channels**

what happens in later conv layers?

- input: tensor with  $C_1$  channels
- output: tensor with  $C_2$  channels

what are the operations?



(Credit: <https://cs231n.github.io/convolutional-networks/>)

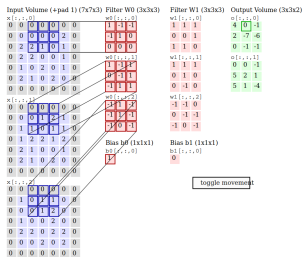
<https://cs231n.github.io/convolutional-networks/>

## Demo of multi-channel (spatial) convolutions

<https://animatedai.github.io/>

(Thanks to Sasha Hydrie!)

# Multiple-channel convolution



(Credit: <https://cs231n.github.io/>)

[convolutional-networks/](https://cs231n.github.io/convolutional-networks/))

$C_1$  input channels( $\mathcal{X}$ ),  $C_2$  output channels

- each filter  $F_i$  is a size  $w \times h \times C_1$  tensor, i.e., with  $C_1$  channels
- all channels of the filters get convolved with the corresponding channels of  $\mathcal{X}$ , and then summed up (plus an optional bias)
- so each filter generates a 2D map, and there are  $C_2$  filters to generate  $C_2$  output channels

$$\sum_{i=0}^{C_1-1} F_i[:, :, i] \star \mathcal{X}[:, :, i] + b$$

```
CLASS torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]],  
stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T,  
Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')
```

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  $(N, C_{out}, H_{out}, W_{out})$  can be precisely described as:

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

where  $\star$  is the valid 2D cross-correlation operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

## Do we reduce the complexity?

Suppose  $C_1$  input channels and  $C_2$  output channels of size  $H \times W$

- # parameters if implementing fully connected layer?  $O(C_1 C_2 H^2 W^2)$
- # parameters if implementing convolution of  $h \times w$ ?  $O(C_1 C_2 h w)$

$h, w$  often small constants, e.g., 3 in practice

Find patterns in an image

Problems with fully connected networks (FCNNs)

## Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

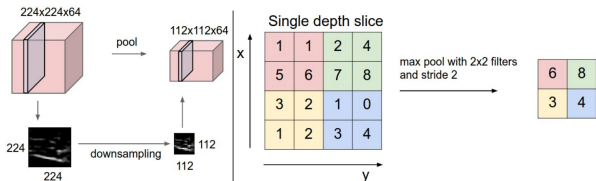
Architectures for classification

Practical tips

Suggested reading

# Pooling

Convolution helps to achieve locality, and (much) reduced complexity, what about invariance?



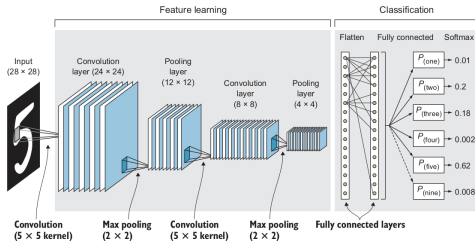
Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size  $[224 \times 224 \times 64]$  is pooled with filter size 2, stride 2 into output volume of size  $[112 \times 112 \times 64]$ . Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little  $2 \times 2$  square).

(Credit: Stanford CS231N)

- max pooling (i.e., max within the receptive field)
- average pooling (i.e., weighted average within the receptive field)
- strides and receptive field size (often  $2/2$  or  $2/3$ )

# Why pooling?

reduce complexity (with stride  $\geq 2$ )



(Credit: [Elgendy, 2020])

- deeper layer: more filters  $\implies$  subsampling avoids explosion in computation
- subsampling keep important features



Figure 3.25 Pooling layers reduce image resolution and keep the image's important features.

(Credit: [Elgendy, 2020])

# Why pooling?

## (approximate) local translation/distortion invariance

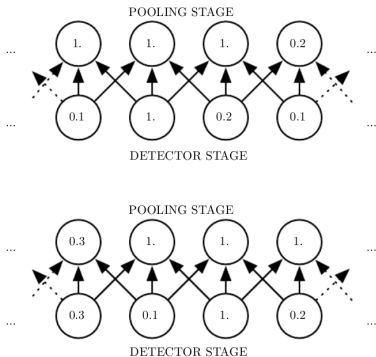


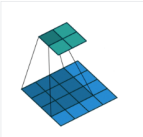
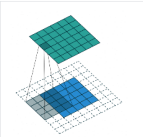
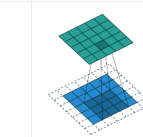
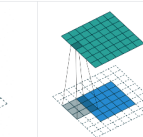
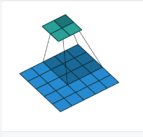
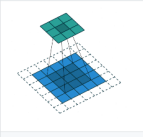
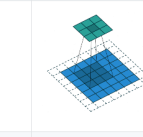
Figure 9.8: Max pooling introduces invariance. *(Top)* A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of max pooling, with a stride of one pixel between pooling regions and a pooling region width of three pixels. *(Bottom)* A view of the same network, after the input has been shifted to the right by one pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed, because the max pooling units are only sensitive to the maximum value in the neighborhood, not its exact location.

(Credit: [Goodfellow et al., 2017])



# Combine convolution and pooling—convolution with strides

idea: convolution with stride  $\geq 2 \approx$  convolution + subsampling

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

So use all convolution (with large strides) layers only, no pooling

[Springenberg et al., 2014]

Find patterns in an image

Problems with fully connected networks (FCNNs)

## Components of CNNs

- Convolutional layers

- Pooling layers

### Why multilayers?

- Computation

Thanks to the cats

Architectures for classification

Practical tips

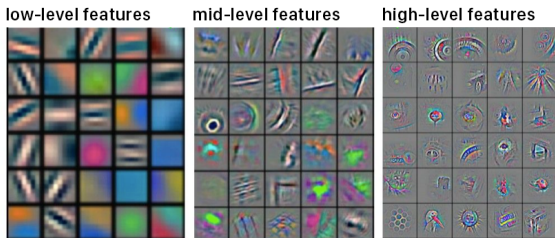
Suggested reading

# Why not single layer?

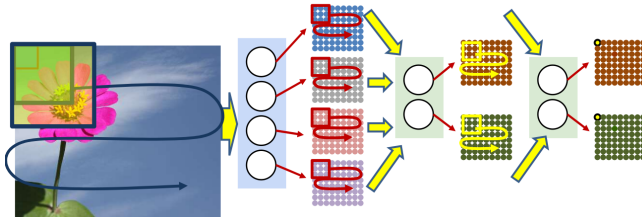


using a one-layer CNN ...

- **efficiency**: one kernel for each variation of 8? for each variation of every object?
- **better**: share kernels across digits or all object categories, but low-level features (edges, corners, etc) likely shareable  $\implies$  **form hierarchy**



# Hierarchical scan



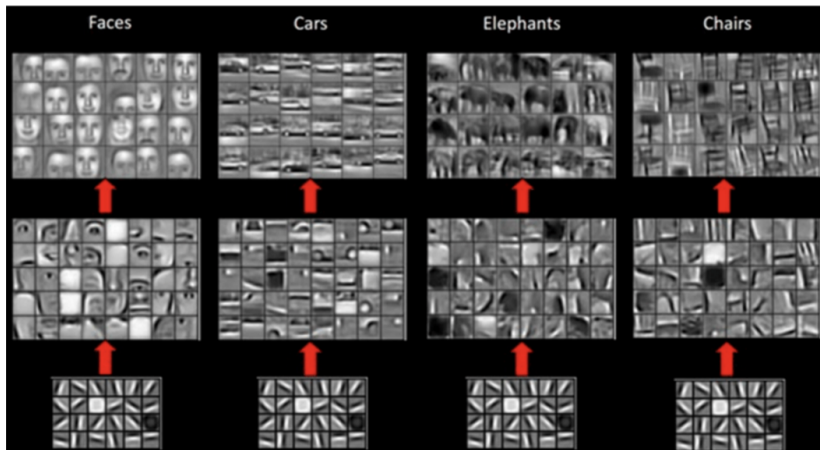
- Later neurons have **increasingly large** effective receptive fields on the input image, i.e., scanning using **composition** of the filters

$$k_L * \dots * k_1 * x = k * x$$

where the effective  $k$  is much larger in spatial extent

- composition (with pooling layers or strides) allows local translation- and distortion-invariance

## Examples of learned features



Find patterns in an image

Problems with fully connected networks (FCNNs)

## Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

## Computation

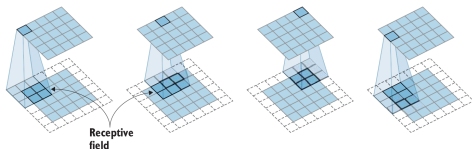
Thanks to the cats

Architectures for classification

Practical tips

Suggested reading

# How to compute convolution?



(Credit: [Elgandy, 2020])

- convolution layer is **locally connected, weight-sharing** fully connected layer
- if we vectorize both input and output, the operation can be represented as a **matrix multiplication**

$$\begin{pmatrix} x1 & x2 & x3 \\ x4 & x5 & x6 \\ x7 & x8 & x9 \end{pmatrix} * \begin{pmatrix} k1 & k2 \\ k3 & k4 \end{pmatrix} \iff \begin{pmatrix} k1 & k2 & 0 & k3 & k4 & 0 & 0 & 0 & 0 \\ 0 & k1 & k2 & 0 & k3 & k4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k1 & k2 & 0 & k3 & k4 & 0 \\ 0 & 0 & 0 & 0 & k1 & k2 & 0 & k3 & k4 \end{pmatrix} \cdot \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \\ x9 \end{pmatrix}$$

so we don't worry about forward and backward pass

To compute the convolution

- use (sparse) matrix-vector multiplication (early versions of cuDNN)
- use fast Fourier transform (introduced in later versions of cuDNN)

$$\mathcal{F}(\mathbf{w} * \mathbf{x}) = \mathcal{F}(\mathbf{w}) \odot \mathcal{F}(\mathbf{x})$$

[known as the **convolution theorem**; linear conv converted into circular conv by zero-padding]

To compute the max-pooling

- forward: simple
- backward? what's  $\nabla_{\mathbf{x}} \max(x_1, \dots, x_n)$ ?



# Outline

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

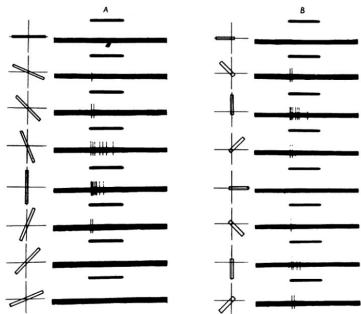
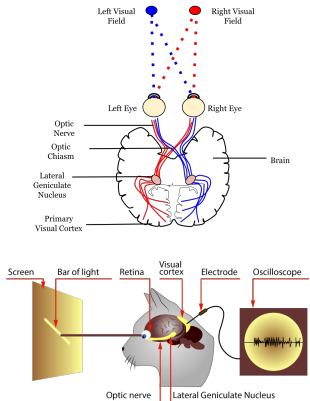
Architectures for classification

Practical tips

Suggested reading

# A brief history of CNN

Hubel and Wiesel 1959 [Hubel and Wiesel, 1959]



focused on the primary visual cortex (V1)

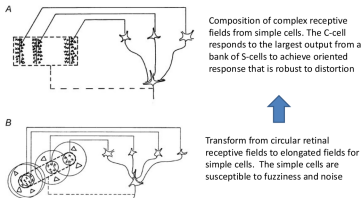
main discovery: directional selectivity of the neurons inside V1, and **local responsiveness** per cell

# A brief history of CNN

Hubel and Wiesel 1962 [[Hubel and Wiesel, 1962](#)]

Two types of cells: *simple S-cells* and *complex C-cells*

- correspond to two levels of processing
- C-cells robust to distortion, but S-cells not



- Complex C-cells build from similarly oriented simple cells
  - They "fine-tune" the response of the simple cell
- Show complex buildup – building *more complex patterns* by composing early neural responses
  - Successive transformation through Simple-Complex combination layers

S-cells: conv kernels    C-cells: max pooling

# A brief history of CNN

Fukushima 1980: Neocognitron [Fukushima, 1980]—unsupervised

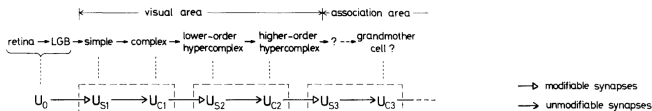
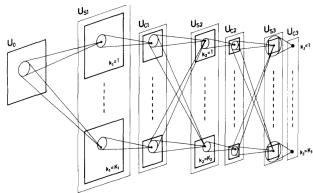
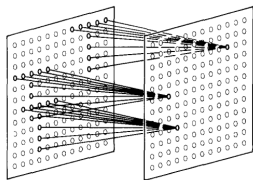


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

- multi-layers of S-C cells compositions
- only S-cells are learnable



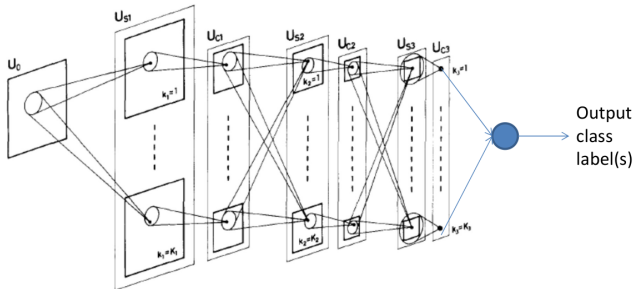
cell planes get smaller but number of planes increase going deeper



S cells have ReLU-like activation, C cells have ReLU+Max like activation

# A brief history of CNN

Lecun 1989: supervision added [LeCun et al., 1989, Lecun et al., 1998]



back-propagation used for supervised training for digit recognition

# Outline

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

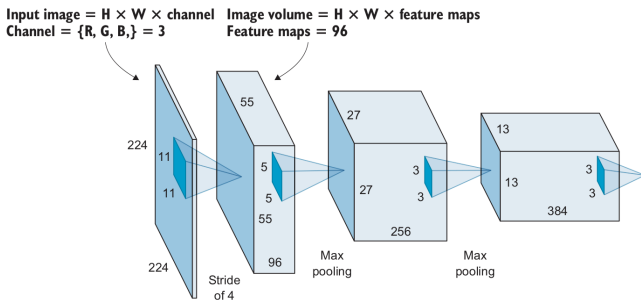
**Architectures for classification**

Practical tips

Suggested reading

# Typical design patterns

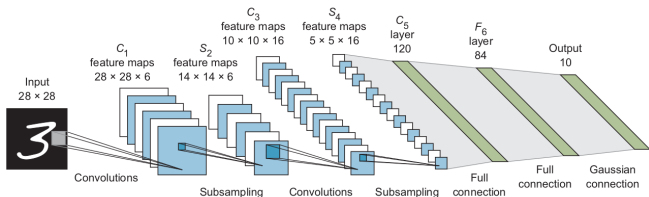
- feature extraction (CONV) + classification (fully connected)
- depth increases (more filters), dimension decreases (subsampling) when moving deeper



(Credit: [Elgandy, 2020])

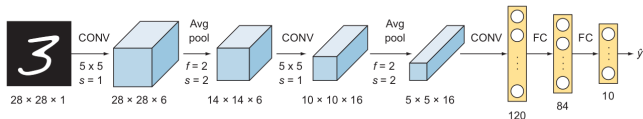
- one or two fully-connected layers for classification

# LeNet-5 (1998)



(Credit: [Elgandy, 2020])

- tanh used for activation
- $5 \times 5$  filters

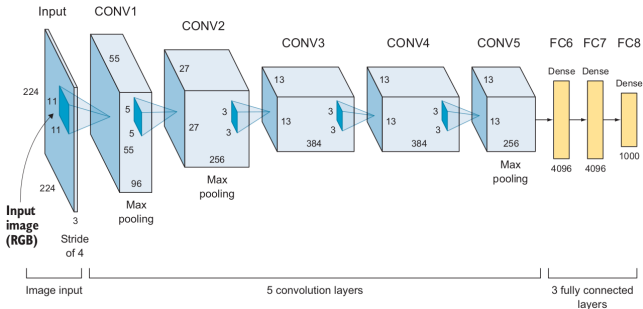


(Credit: [Elgandy, 2020])



# AlexNet (2012)

breakthrough on ImageNet competition in 2012 and impressed the computer vision community



(Credit: [Elgandy, 2020])

- ReLU used for activation
- large filters:  $11 \times 11$ ,  $5 \times 5$ ,  $3 \times 3$  filters
- dropout used for regularization
- weight decay/regularization

## VGG — Visual Geometry Group (Oxford U.)



Figure 5.8 VGGNet-16 architecture

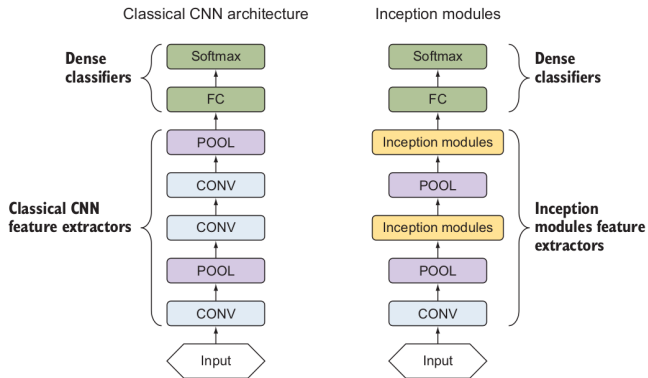
(Credit: [Elgandy, 2020])

- smaller filters ( $3 \times 3$ ) to make up for large ones in AlexNet. A nice property of convolution:

$$\mathbf{a} * (\mathbf{b} * \mathbf{c}) = (\mathbf{a} * \mathbf{b}) * \mathbf{c}$$

composition of filters covers larger receptive fields

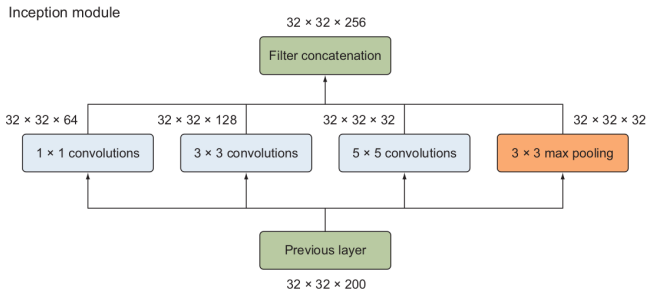
# Inception and GoogLeNet (2014)



(Credit: [Elgandy, 2020])

pack things into **inception modules**

# Inception module—basic version



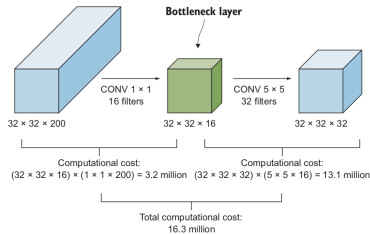
(Credit: [Elgandy, 2020])

idea: apply all filters together and (hopefully) the training process performs the suitable selection/combination itself

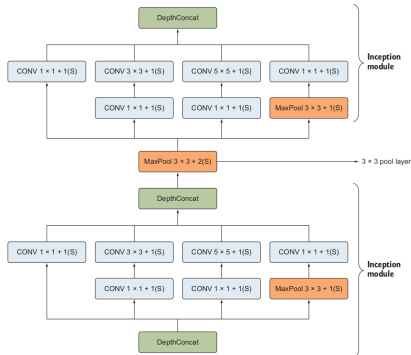
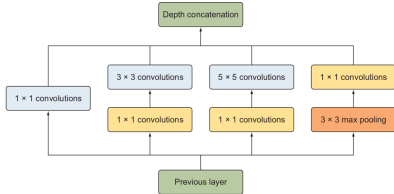
- filters can be short-circuited when the values are set to **0**

# Inception module with dimension reduction

$1 \times 1$  convolution helps to reduce the #channels  $\implies$  saves computation



Inception module with dimensionality reduction



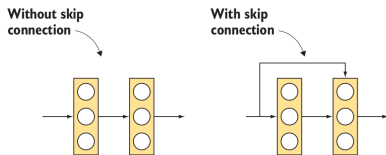
(Credit: [Elgandy, 2020])

(Credit: [Elgandy, 2020])

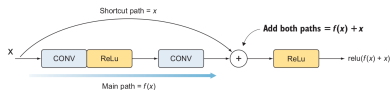
# ResNet (2015)

going really deep...sees performance **degradation**

a solution:



(Credit: [Elgandy, 2020])



a residual block (Credit: [Elgandy, 2020])

– skip connection

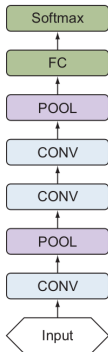
- \* allows short-circuit unnecessary layers—e.g., setting the kernels to zero—and thus avoids performance degradation when adding more layers

- \* mitigates gradient explosion or vanishing— $\mathbf{J}_{f+I}(\mathbf{x}) = \mathbf{J}_f(\mathbf{x}) + \mathbf{I}$

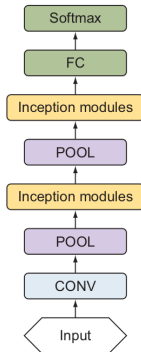
– batch normalization

# Comparison with previous models

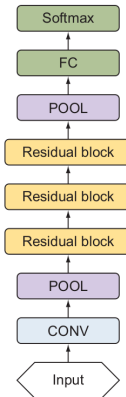
Classical CNN architecture



Inception modules



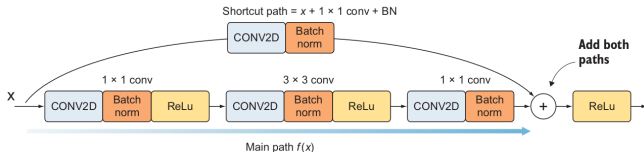
Residual blocks



(Credit: [Elgendy, 2020])

# Inside a residual block

Bottleneck residual block with reduce shortcut



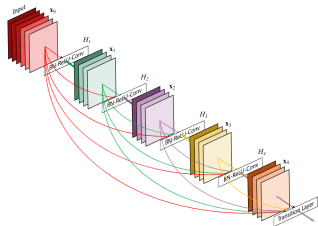
(Credit: [Elgandy, 2020])

- no pooling layers
- $1 \times 1$  conv before and after  $3 \times 3$  conv to control #channels and hence computation
- batch normalization (BN) after each conv layer
- $1 \times 1$  conv and BN added to the skip connection also to match dim for summation

full details see: [https://pytorch.org/hub/pytorch\\_vision\\_resnet/](https://pytorch.org/hub/pytorch_vision_resnet/)

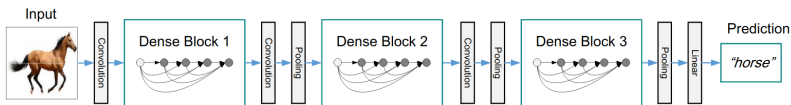


# DenseNet (2016)



**Figure 1:** A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

(Credit: [Huang et al., 2016])



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

(Credit: [Huang et al., 2016])

transition layers adjust the sizes of the feature maps

- inside the same dense block, any feature map “connected” to all subsequent feature maps—**dense**
- “connected” here means **concatenation** vs. the **summation** in ResNet
- concatenation enables feature reusing and hence higher efficiency

## Other models to watch

on accuracy:

- EfficientNet (2019) [Tan and Le, 2019]  
<https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>
- ResNeXt <https://arxiv.org/abs/1611.05431>

on compact models:

- SqueezeNet <https://arxiv.org/abs/1602.07360>
- ShuffleNet <https://arxiv.org/abs/1807.11164>
- MobileNet <https://arxiv.org/abs/1801.04381>

Pytorch official classification models

<https://pytorch.org/vision/stable/models.html#classification>

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

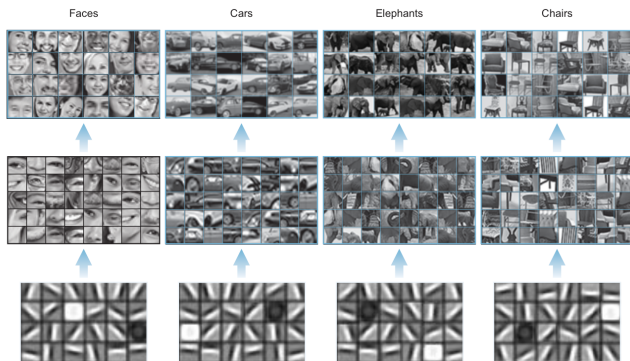
Architectures for classification

**Practical tips**

Suggested reading

# Transfer learning

Recall: (we hope) CNNs learn increasingly complex and semantically meaningful features



(Credit: [Elgandy, 2020])

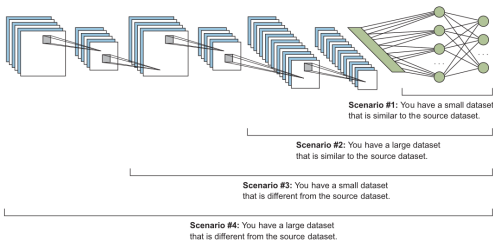
So: early layers trained on a large and diverse dataset, e.g., ImageNet, can be reused. This part is called a **pretrained** model

# Deep transfer learning as feature reuse

**source domain:** task domain of the pre-trained model

**target domain:** current task domain

Scenario	Size of the target data	Similarity of the original and new datasets	Approach
1	Small	Similar	Pretrained network as a feature extractor
2	Large	Similar	Fine-tune through the full network
3	Small	Very different	Fine-tune from activations earlier in the network
4	Large	Very different	Fine-tune through the entire network



┌ indicates  
trainable part  
(Credit:

[Elgandy, 2020])

Pytorch tutorial: [https://pytorch.org/tutorials/beginner/transfer\\_learning\\_tutorial.html](https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html)

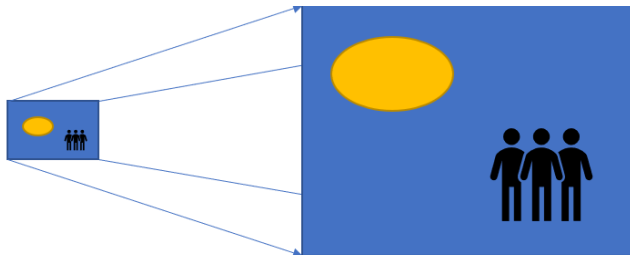
Stanford notes: <https://cs231n.github.io/transfer-learning/>

For domains that only need low-level features: [Peng et al., 2021]

# Transposed convolution

**convolution with strides:** downsampling

**transposed convolution:** upsampling



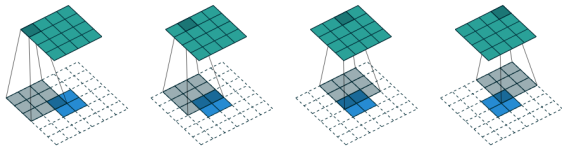
(Credit: <https://naokishibuya.medium.com/>)

often used for segmentation, generation, or other regression—outputs are structured objects such as images, videos, time series, speech, etc

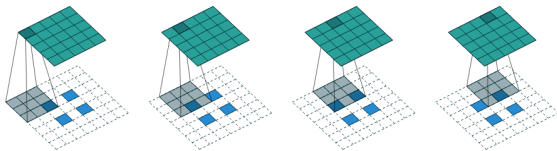
- traditional methods: e.g., nearest neighbor/bilinear/bicubic **interpolation**
- here: interpolation with a **learnable filter**

# Transposed convolution

also called **fractionally strided convolutions** or deconvolution (misnomer): zero padding, zero interleaving (when forward stride  $> 1$ ), and then convolution



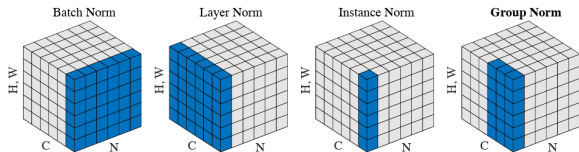
forward stride = 1



forward stride = 2

more details see [https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

# Normalization



**Normalization methods.** Each subplot shows a feature map tensor, with  $N$  as the batch axis,  $C$  as the channel axis, and  $(H, W)$  as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Credit: [Wu and He, 2018]

normalization in different directions/groups of the data tensors

- $N$  is the batch axis
- $C$  is the channel axis
- $WH$  is the per output dimension (1 for fully connected, but 2D for CNNs)

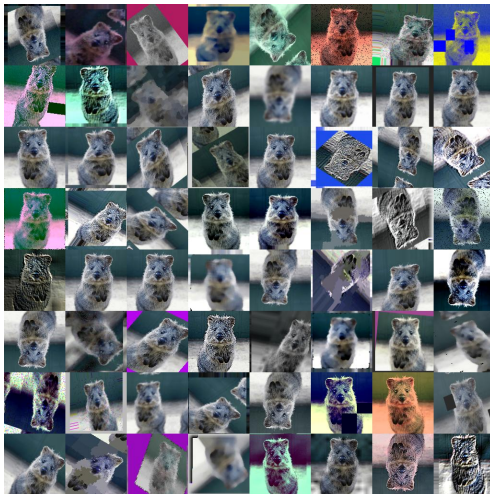
batch normalization is popular, but with **layer/group normalization**:

- small  $N$  (batch size) is possible
- simplicity: training/test normalizations are consistent



# Data augmentation

- More relevant data always help!
- Fetch more external data
- Generate more internal data: generate based on whatever **you want to be robust to**
  - \* vision: translation, rotation, background, noise, deformation, flipping, blurring, occlusion, etc



Credit: <https://github.com/aleju/imgaug>

See one example here [https:](https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html)

[//pytorch.org/tutorials/beginner/transfer\\_learning\\_tutorial.html](https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html) 65 / 72

# Are CNNs only for images?

Recall why CNN? **complexity, locality/ordering, translation-invariance**

These are desired also when processing video, text sequence, times series data, speech data, etc Examples:

- WaveNet for text-to-speech system  
<https://en.wikipedia.org/wiki/WaveNet>
- text classification <https://arxiv.org/abs/1408.5882>
- video analysis [Ji et al., 2013, Karpathy et al., 2014, Huang et al., 2018]
- time series analysis [Yu and Koltun, 2015, Borovykh et al., 2017]

see also *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling* [Bai et al., 2018]

# Outline

Find patterns in an image

Problems with fully connected networks (FCNNs)

Components of CNNs

- Convolutional layers

- Pooling layers

- Why multilayers?

- Computation

Thanks to the cats

Architectures for classification

Practical tips

Suggested reading

## Suggested reading

- Deep Learning for Vision Systems [[Elgendy, 2020](#)]
- Convolutional Networks for Images, Speech, and Time-Series [[LeCun et al., 1995](#)]
- A guide to convolution arithmetic for deep learning  
<https://arxiv.org/abs/1603.07285>
- Gradient-based learning applied to document recognition [[Lecun et al., 1998](#)]
- <https://cs231n.github.io/transfer-learning/>

- [Bai et al., 2018] Bai, S., Kolter, J. Z., and Koltun, V. (2018). **An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.** *arXiv:1803.01271*.
- [Borovykh et al., 2017] Borovykh, A., Bohte, S., and Oosterlee, C. W. (2017). **Conditional time series forecasting with convolutional neural networks.** *arXiv:1703.04691*.
- [Elgendy, 2020] Elgendy, M. (2020). **Deep Learning for Vision Systems.** MANNING PUBN.
- [Fukushima, 1980] Fukushima, K. (1980). **Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.** *Biological Cybernetics*, 36(4):193–202.
- [Goodfellow et al., 2017] Goodfellow, I., Bengio, Y., and Courville, A. (2017). **Deep Learning.** The MIT Press.
- [Huang et al., 2016] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2016). **Densely connected convolutional networks.** *arXiv:1608.06993*.

- [Huang et al., 2018] Huang, J., Zhou, W., Zhang, Q., Li, H., and Li, W. (2018). **Video-based sign language recognition without temporal segmentation.** *arXiv:1801.10111*.
- [Hubel and Wiesel, 1959] Hubel, D. H. and Wiesel, T. N. (1959). **Receptive fields of single neurones in the cat's striate cortex.** *The Journal of Physiology*, 148(3):574–591.
- [Hubel and Wiesel, 1962] Hubel, D. H. and Wiesel, T. N. (1962). **Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.** *The Journal of Physiology*, 160(1):106–154.
- [Ji et al., 2013] Ji, S., Xu, W., Yang, M., and Yu, K. (2013). **3d convolutional neural networks for human action recognition.** *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231.
- [Jin et al., 2020] Jin, Y., Mishkin, D., Mishchuk, A., Matas, J., Fua, P., Yi, K. M., and Trulls, E. (2020). **Image matching across wide baselines: From paper to practice.** *International Journal of Computer Vision*, 129(2):517–547.

- [Karpathy et al., 2014] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). **Large-scale video classification with convolutional neural networks**. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE.
- [LeCun et al., 1995] LeCun, Y., Bengio, Y., et al. (1995). **Convolutional networks for images, speech, and time series**. *The handbook of brain theory and neural networks*, 3361(10):1995.
- [LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). **Backpropagation applied to handwritten zip code recognition**. *Neural Computation*, 1(4):541–551.
- [Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). **Gradient-based learning applied to document recognition**. *Proceedings of the IEEE*, 86(11):2278–2324.
- [Peng et al., 2021] Peng, L., Liang, H., Luo, G., Li, T., and Sun, J. (2021). **Rethinking transfer learning for medical image classification**. *arXiv:2106.05152*.

- [Springenberg et al., 2014] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). **Striving for simplicity: The all convolutional net.** *arXiv:1412.6806*.
- [Tan and Le, 2019] Tan, M. and Le, Q. V. (2019). **Efficientnet: Rethinking model scaling for convolutional neural networks.** *arXiv:1905.11946*.
- [Wu and He, 2018] Wu, Y. and He, K. (2018). **Group normalization.** In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19.
- [Yu and Koltun, 2015] Yu, F. and Koltun, V. (2015). **Multi-scale context aggregation by dilated convolutions.** *arXiv:1511.07122*.