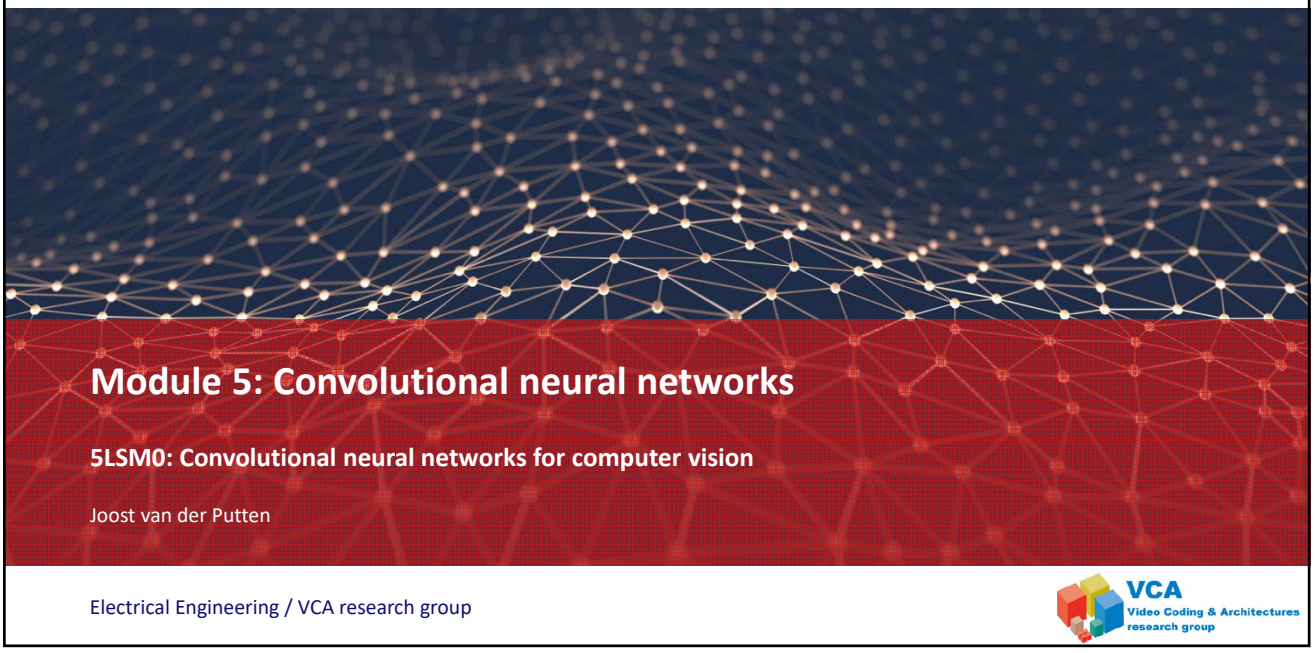


Module 5: Convolutional neural networks

5LSM0: Convolutional neural networks for computer vision

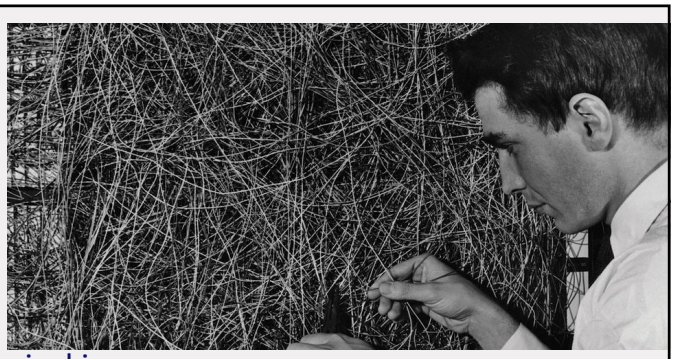
Joost van der Putten

Electrical Engineering / VCA research group

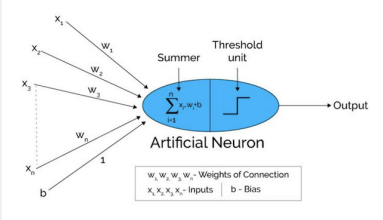
Module outline

- History of CNNs
- Another cat video
- CNNs today
- The convolution layer
  - Convolutions
  - Output dimensions of convolution process
  - Padding
- Activation functions
- Pooling

History of CNNs

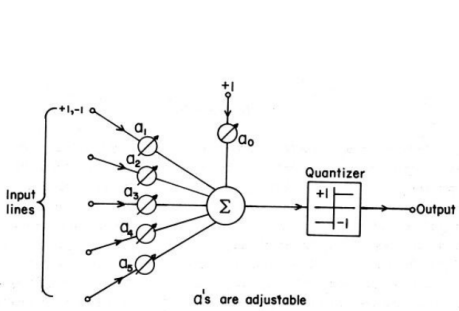


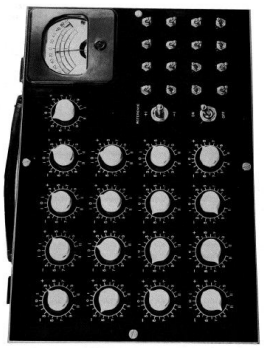
- The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
- The machine was connected to a camera that used 20x20 cadmium sulfide photocells to produce a 400-pixel image.
- $f(x) = \begin{cases} 1 & if\ w \cdot x + b > 0 \\ 0 & otherwise \end{cases}$

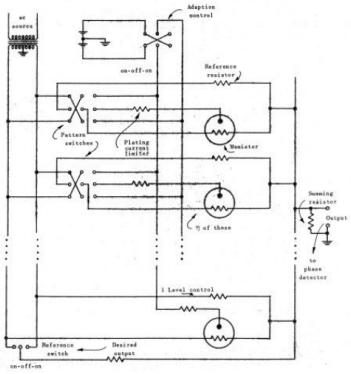


Frank Rosenblatt, 1957: Perceptron

History of CNNs



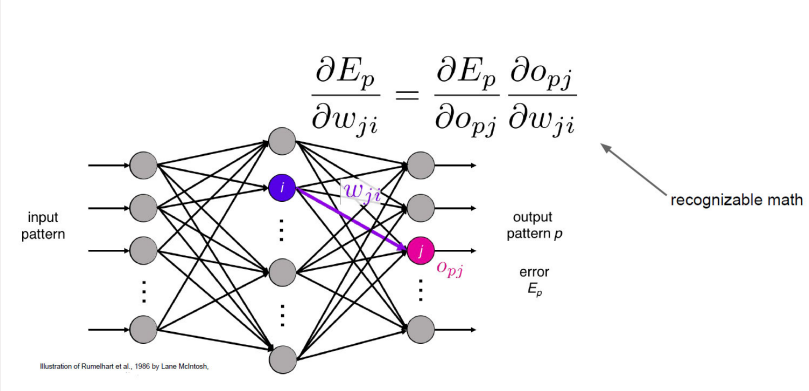




Widrow and Hoff, ~1960: Adaline/Madaline

- Still no backprop

History of CNNs



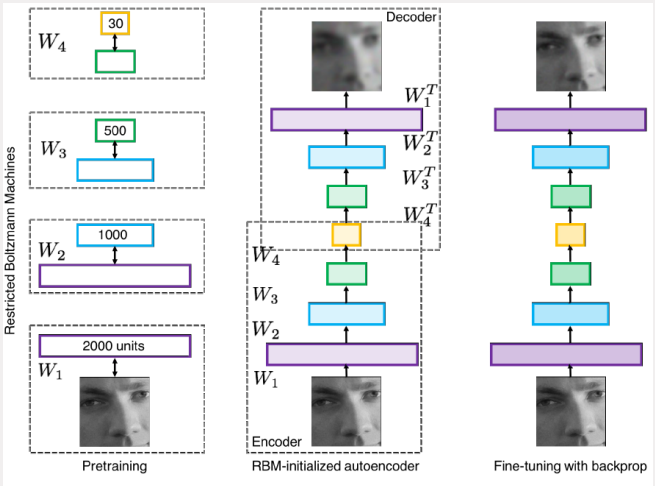
Rumelhart et al., 1986: First time back-propagation was used

History of CNNs

And then ….. Nothing.

Until …

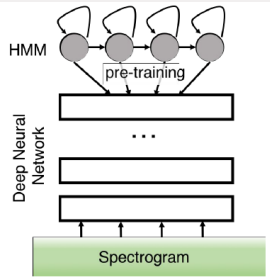
Hinton and Salakhutdinov 2006



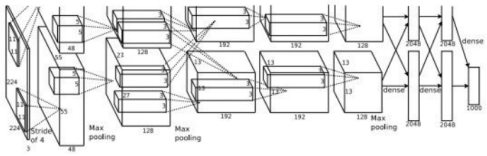
History of CNNs

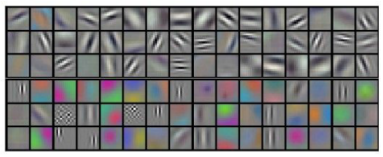
First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012



Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012.

History of CNNs

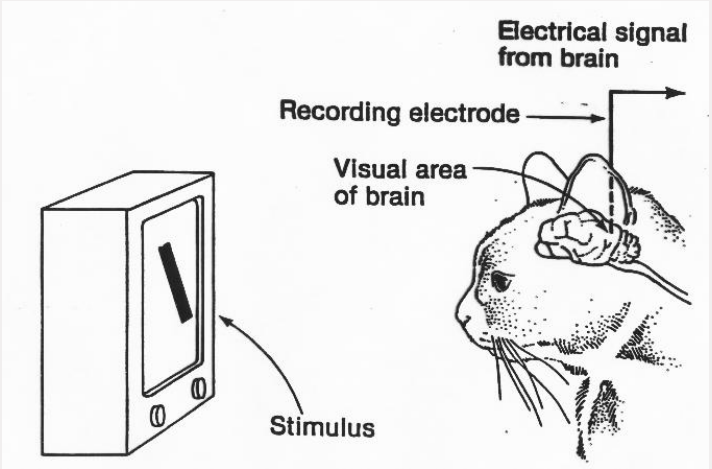
Hubel & Wiesel,

1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

1962

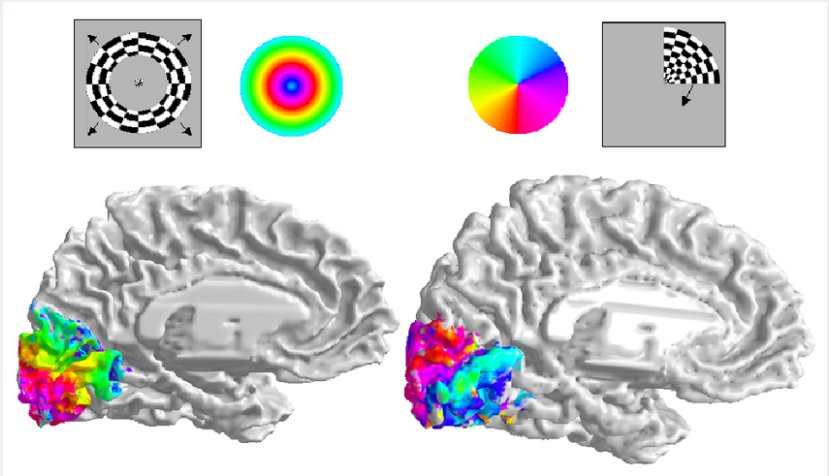
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX



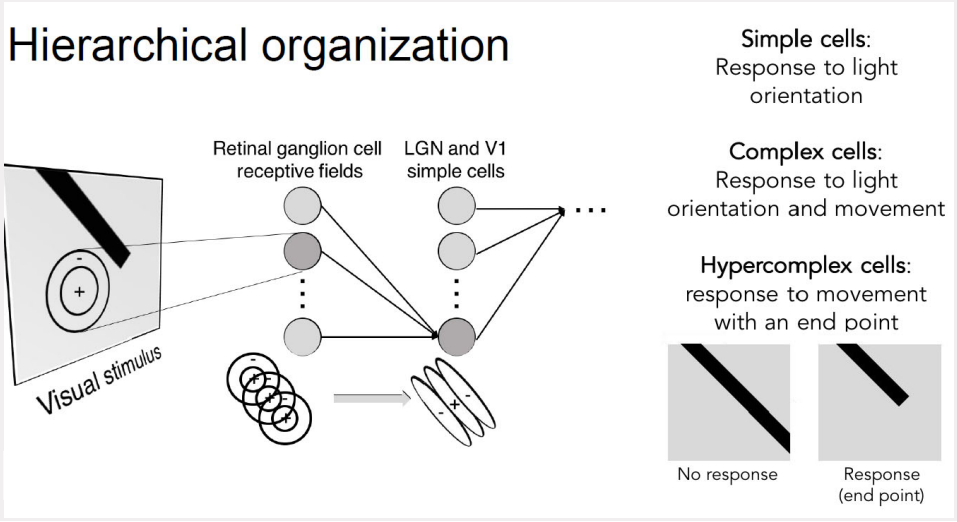
Cat video

History of CNNs

- Nearby cells in visual cortex represent nearby regions in The visual field

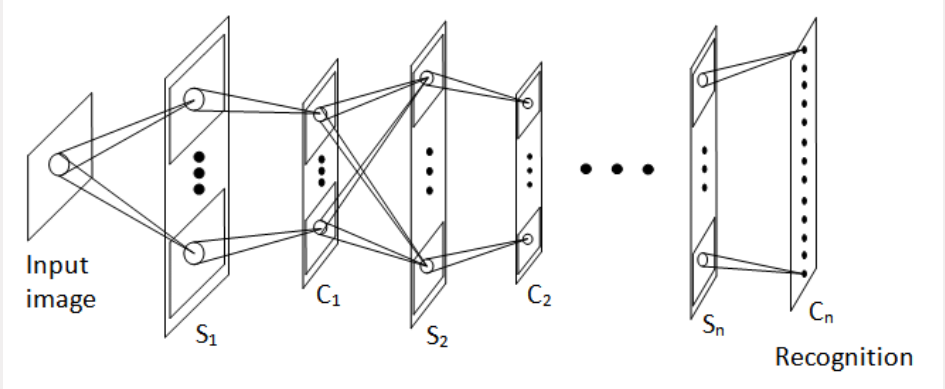


History of CNNs



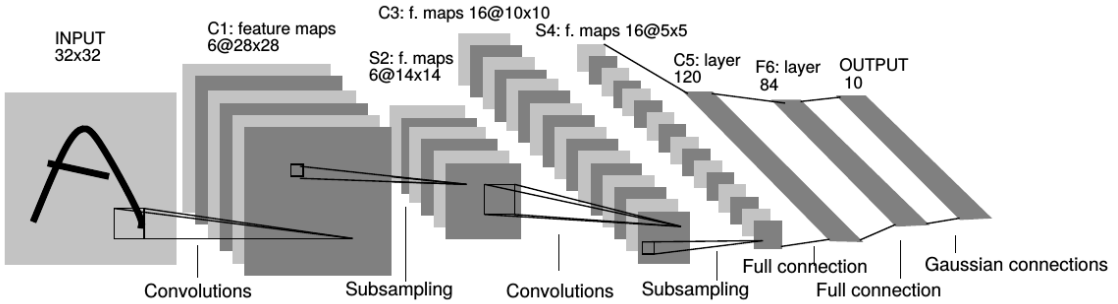
History of CNNs

- Neurocognitron (Fukushima 1980)
- Combination of Simple cells (modifiable parameters) and Complex cells (perform pooling)



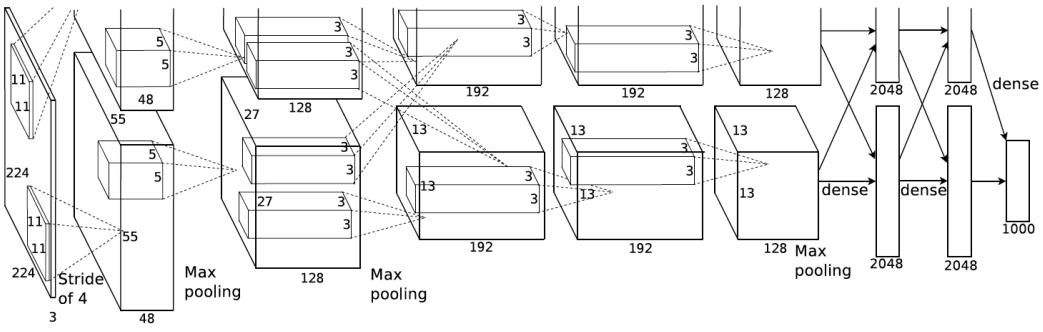
History of CNNs

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]



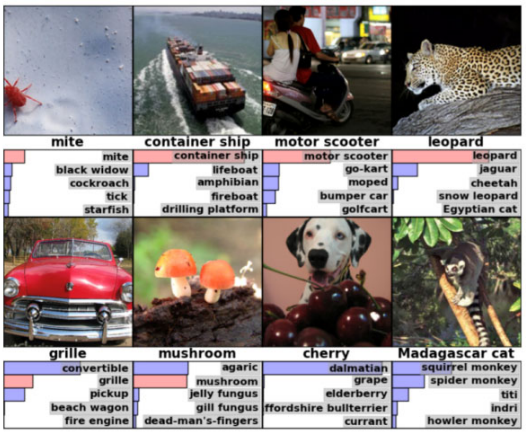
History of CNNs

ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]



CNNs today

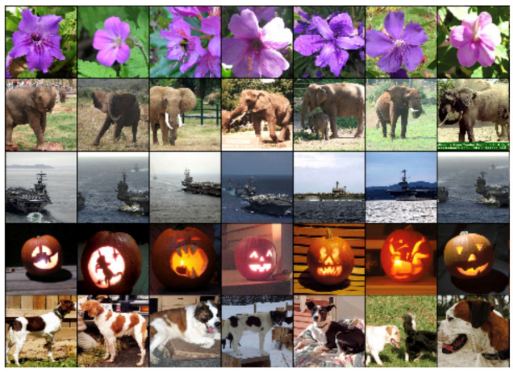
Classification



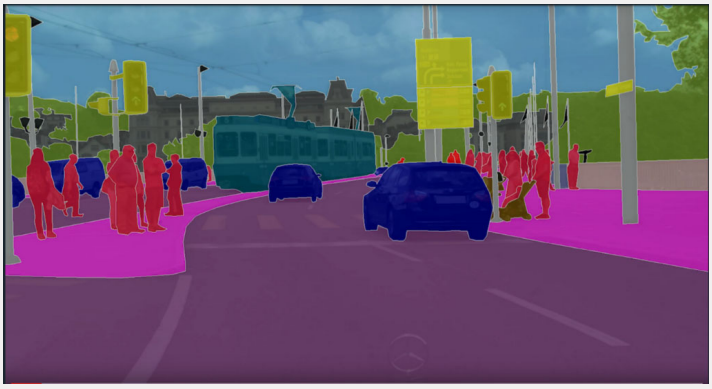
Detection

CNNs today

Image retrieval

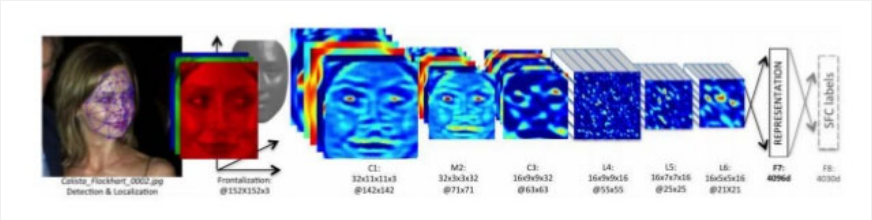


Semantic segmentation

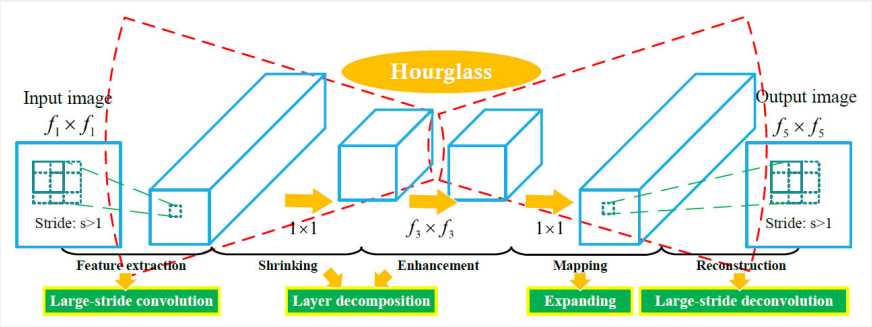


CNNs today

Face verification

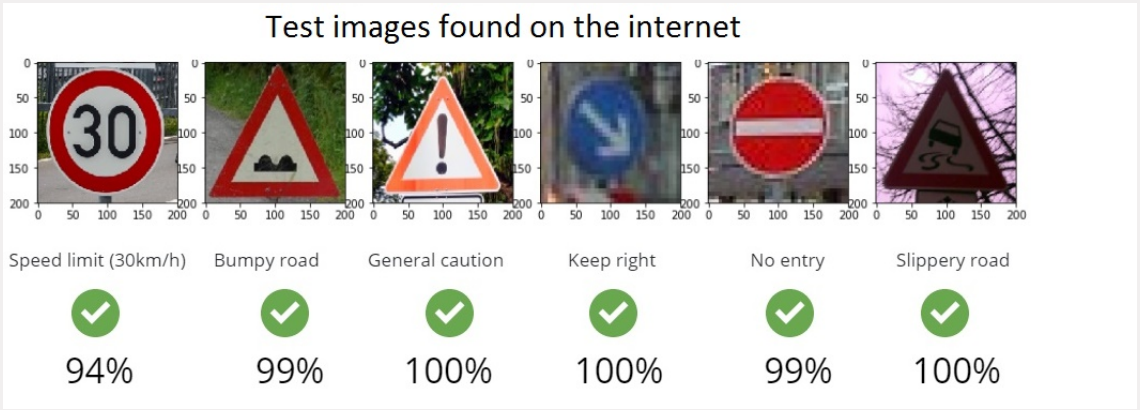


Compression



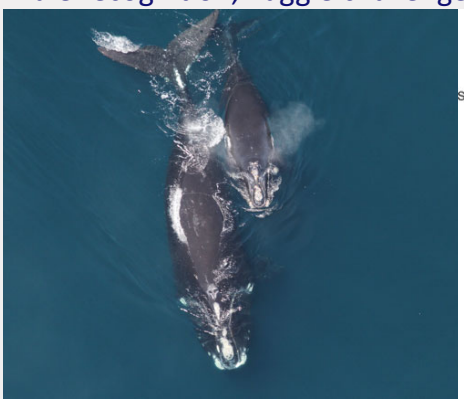
CNNs today

Trafic sign detection

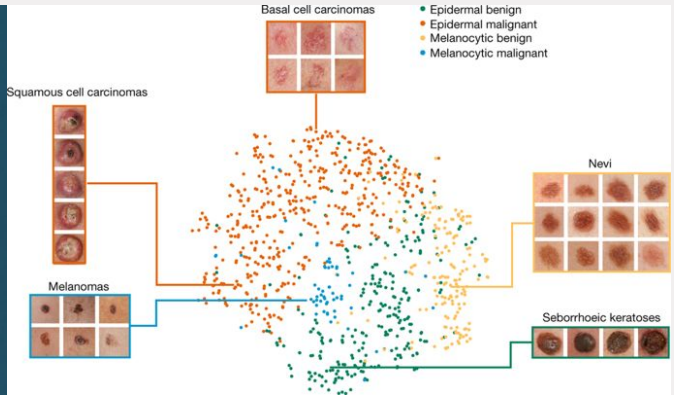


CNNs today

Whale recognition, Kaggle challenge



Skin cancer classification



Pose estimation

https://www.youtube.com/watch?v=pW6nZXeWIGM&t=7s

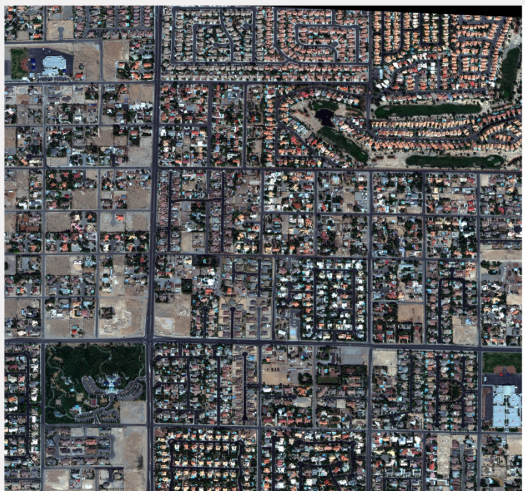
AlphaStar

CNNs today

Deep dream

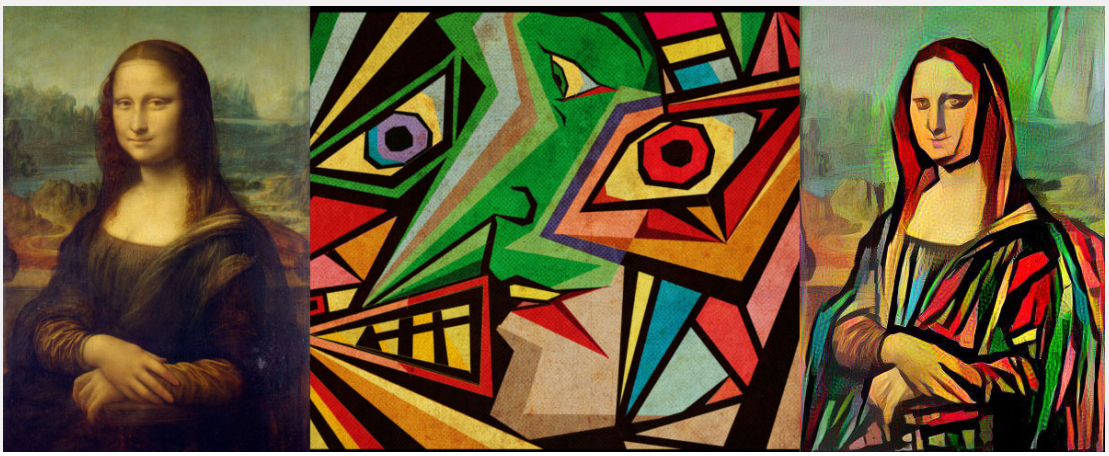
CNNs today

Road segmentation



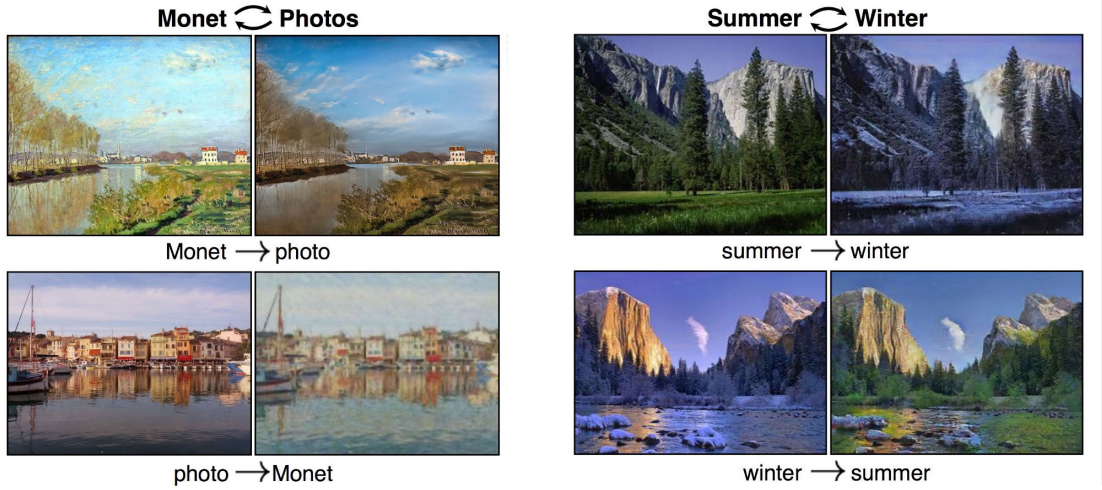
CNNs today

Style transfer



CNNs today

Style transfer



Cnns today

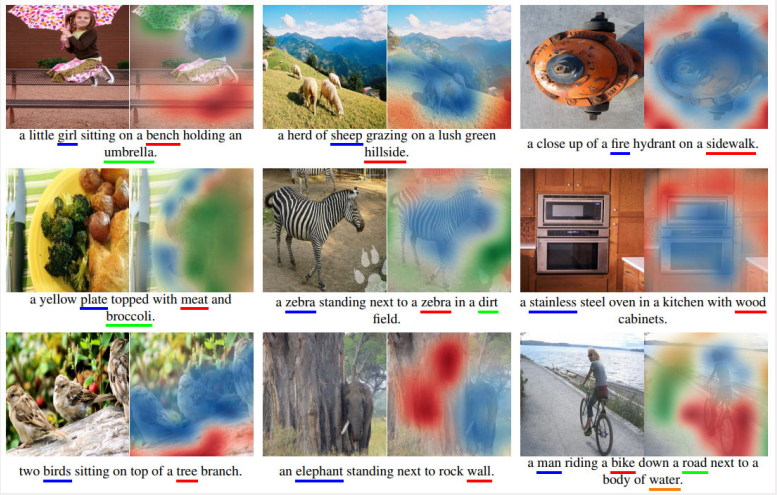
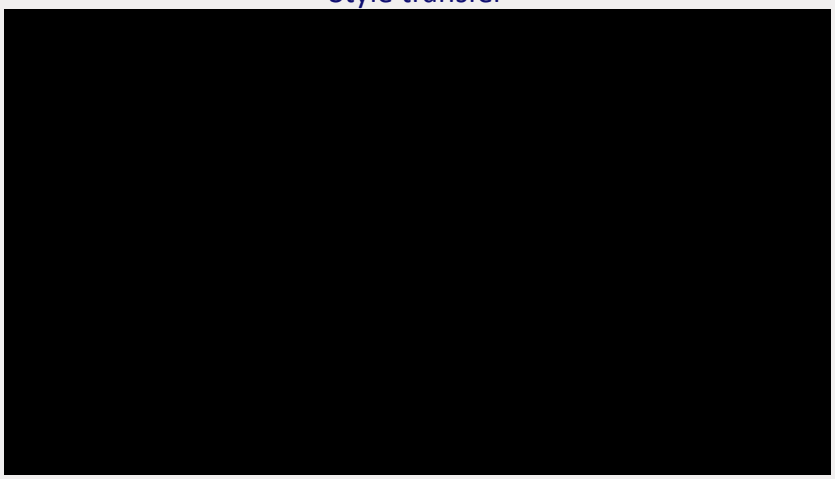


Image captioning

CNNs today

Style transfer



CNNs today

Image colorization

CNNs today

Super resolution

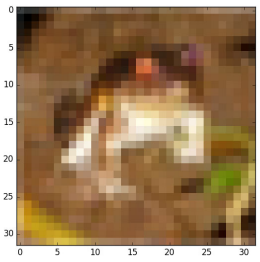
CNNs today

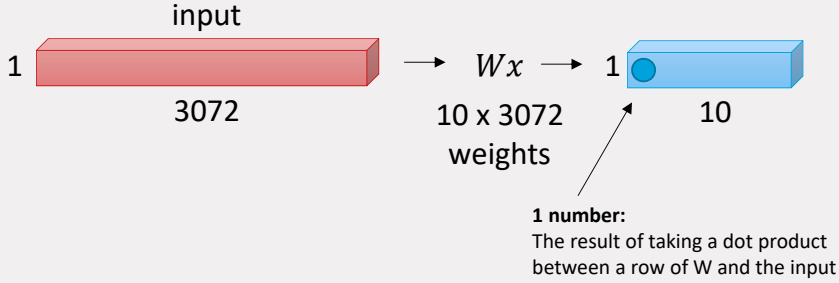
Image restoration

Convolutional Neural Networks

Fully connected layer

32x32x3 image -> stretch to 3072x1



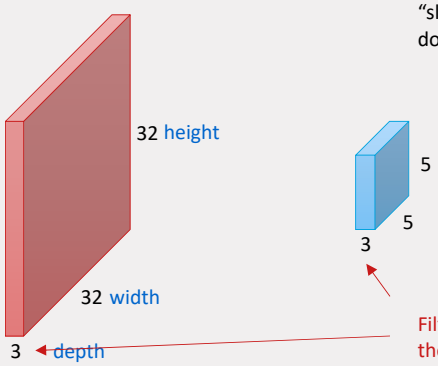


1 number:
The result of taking a dot product between a row of W and the input

Convolutional layer

32x32x3 image

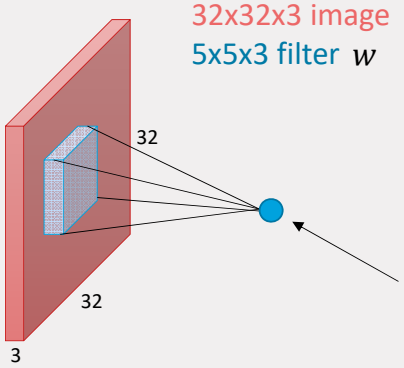
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"



Convolutional layer

32x32x3 image

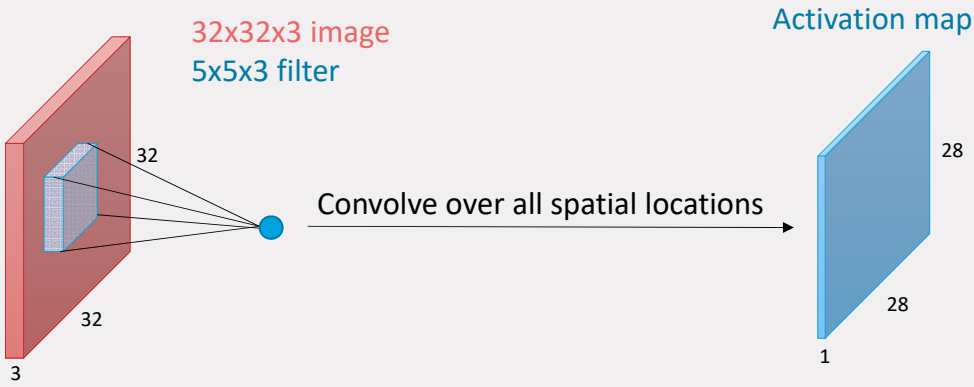
5x5x3 filter w



1 number:
The result of taking a dot product between the filter and a small 5x5x3 chunk of the image

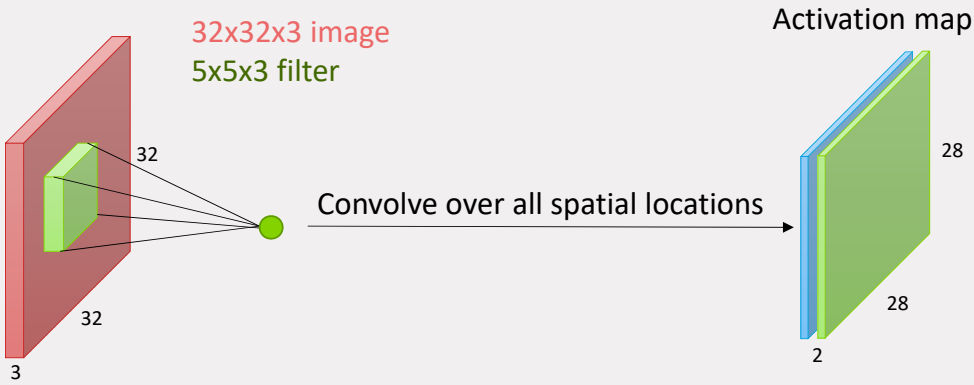
$w^T x + b$

Convolutional layer



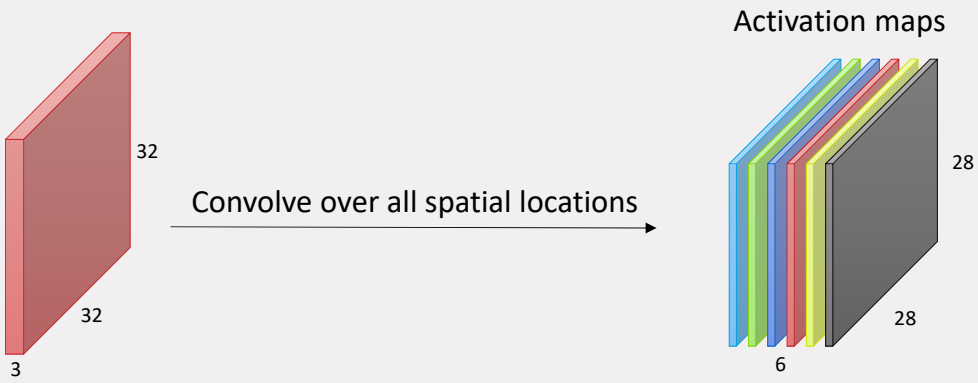
Convolutional layer

Consider a second, green filter



Convolutional layer

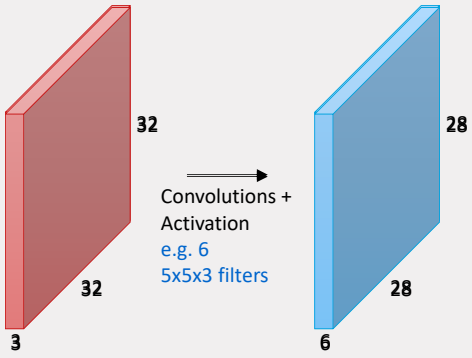
With 6 5x5 filters, we obtain 6 28x28 activation maps



Stack the activation maps to obtain a “new image” of size 28x28x6

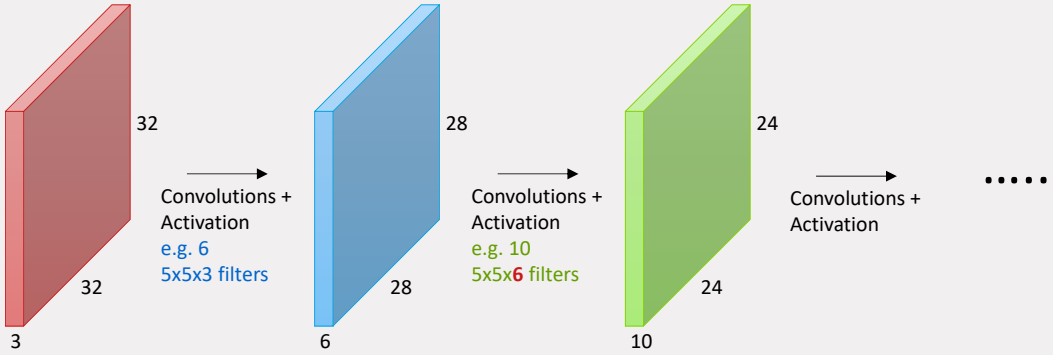
Preview to ConvNets

A **ConvNet** is a sequence of convolutional layers interspersed with activation functions.

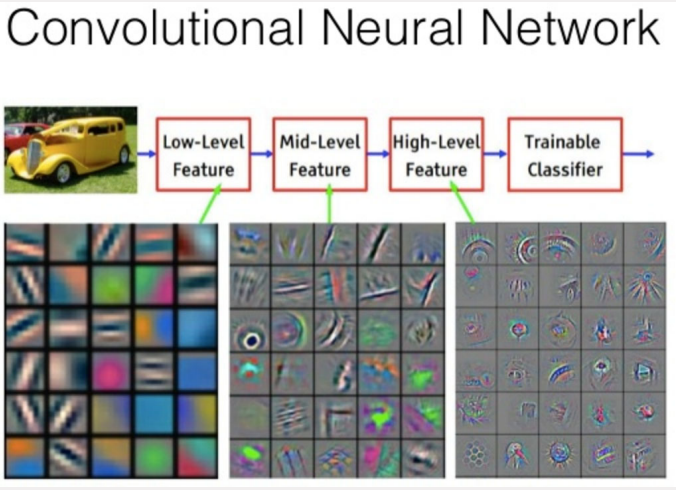


Preview to ConvNets

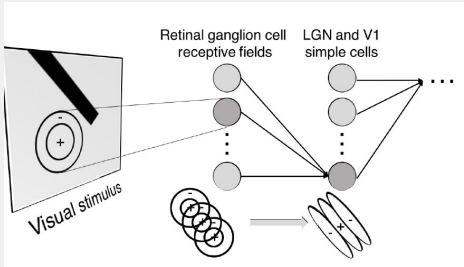
A **ConvNet** is a sequence of convolutional layers interspersed with activation functions.



Preview to ConvNets

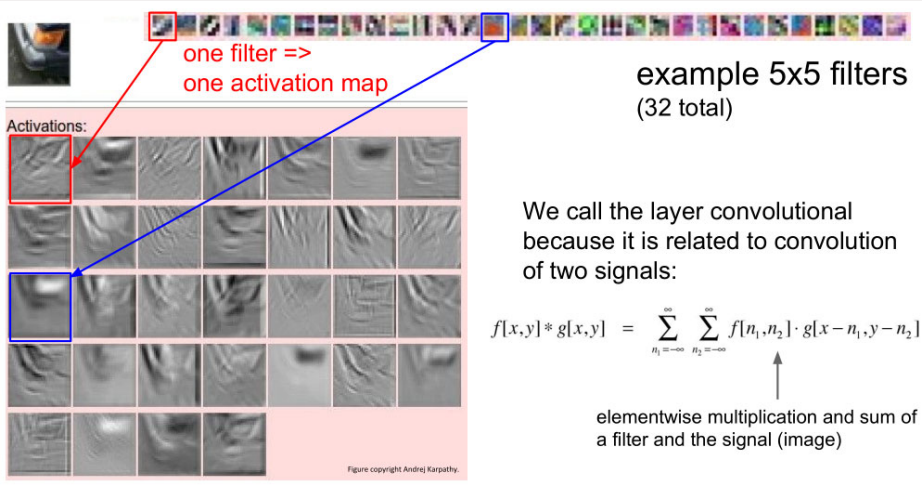


Hubel & Wiesel

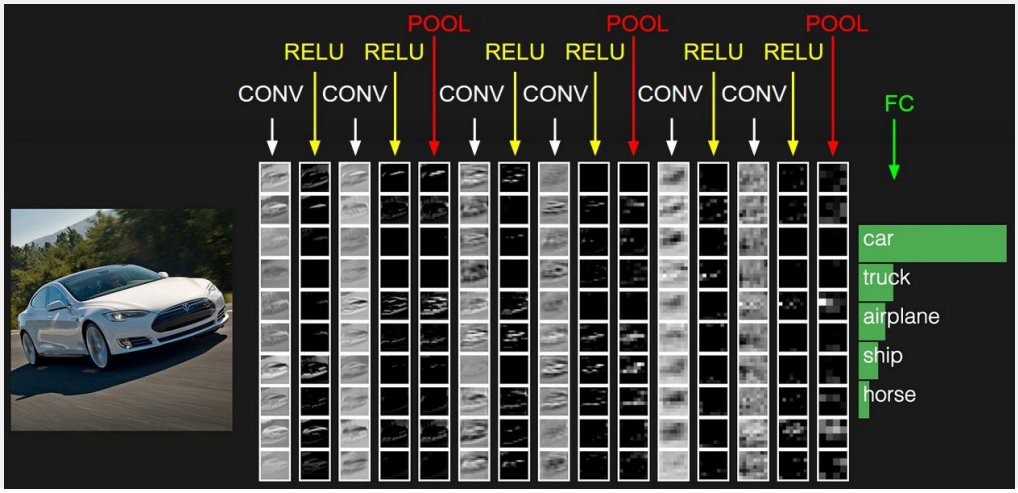


Convolutional neural network learns hierarchical structure on its own

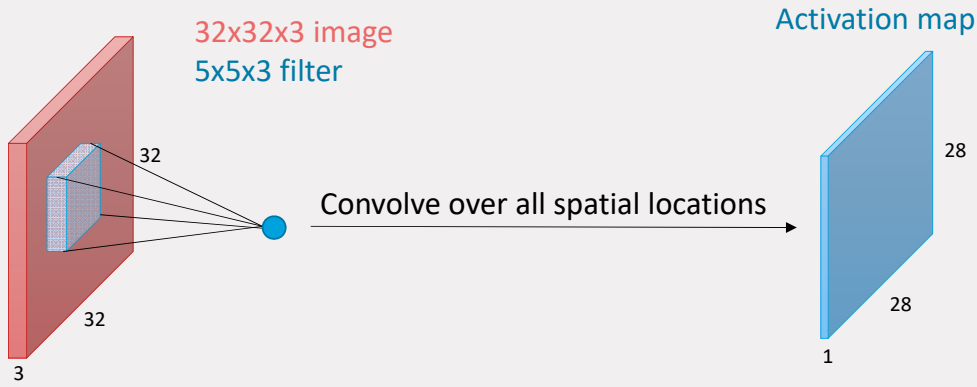
Preview to ConvNets



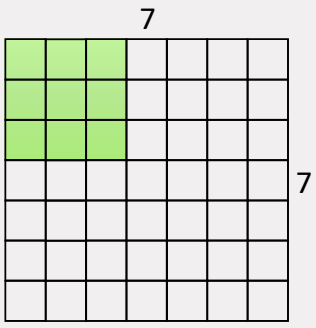
Preview to ConvNets



A closer look at spatial dimensions



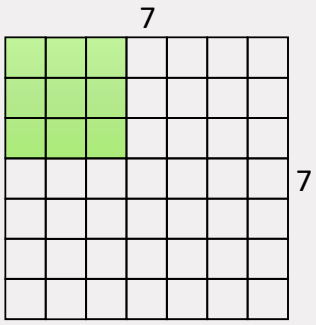
A closer look at spatial dimensions



7x7 input
3x3 filter

=> 5x5 output

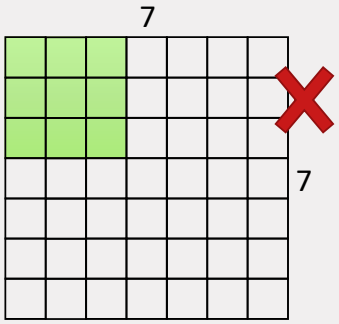
A closer look at spatial dimensions



7x7 input
3x3 filter applied with **Stride 2**

=> 3x3 output

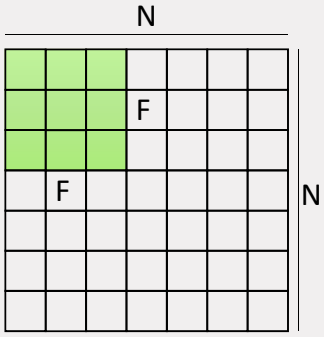
A closer look at spatial dimensions



7x7 input
3x3 filter applied with **Stride 3**

Doesn't fit!
Cannot apply 3x3 filter on
7x7 input with stride 3

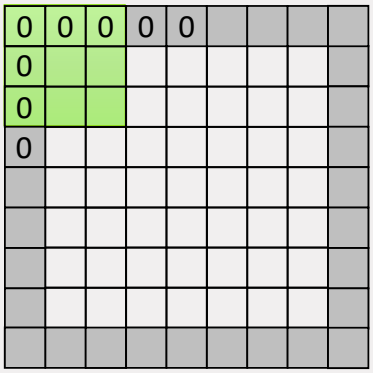
A closer look at spatial dimensions



Output size:
(N-F)/stride + 1

e.g. N=7, F=3:
Stride 1 -> (7-3)/1 + 1 = 5
Stride 2 -> (7-3)/2 + 1 = 3
Stride 3 -> (7-3)/3 + 1 = 2.33 ☹

In practice: Zero padding

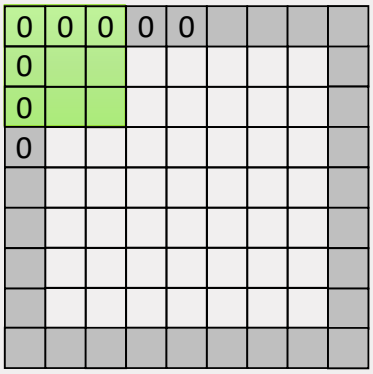


e.g. input **7x7**
3x3 filter, applied with **stride 1**
Pad with 1 pixel border
What are the output dimensions?

7x7 output!

Recall:
(N-F)/stride + 1

In practice: Zero padding



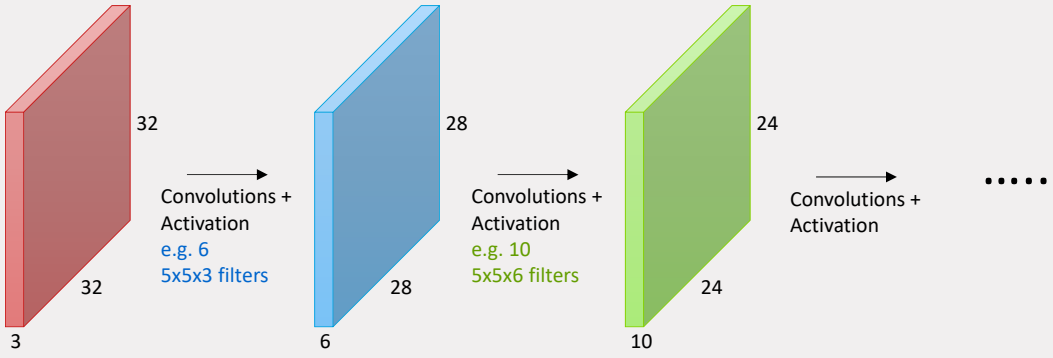
e.g. input **7x7**
3x3 filter, applied with **stride 1**
Pad with 1 pixel border
What are the output dimensions?

7x7 output!
In general, it is common to see convolutional layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (Will preserve size spatially)

F = 3 -> zero pad with 1 pixel
F = 5 -> zero pad with 2 pixels

In practice: Zero padding

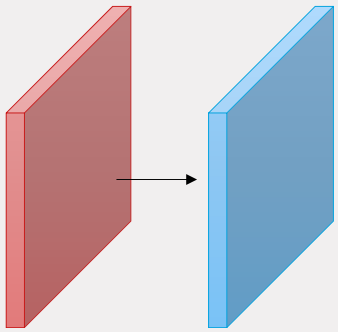
Repeated convolutions shrink volumes spatially! (32 -> 28 -> 24 …). Shrinking too fast does not work well in practice.



Spatial dimension and number of parameters example

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

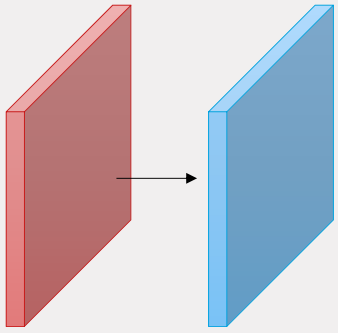
Output volume size: ?



Spatial dimension and number of parameters example

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

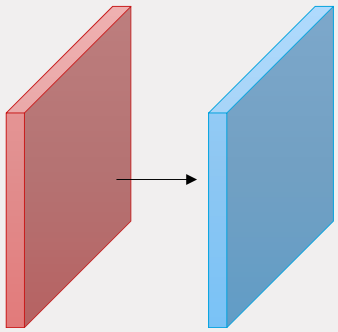
Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



Spatial dimension and number of parameters example

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

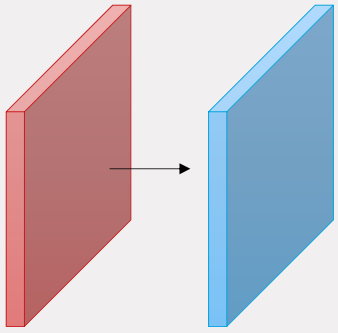
Number of parameters in this layer: ?



Spatial dimension and number of parameters example

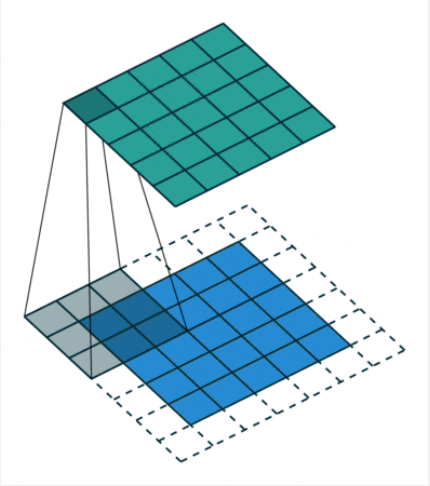
Input volume: **32x32x3**
10 5x5 filters with stride 1 , pad 2

Number of parameters in this layer:
Each filter has 5x5x3 + 1 = 76 params
-> 76 x 10 = 760



Each filter has a bias term

Summary



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters **K**
  - Their spatial extent **F**
  - The stride **S**
  - The amount of padding **P**
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume the **d**-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the **d**-th filter over the input volume with a stride of S, and then offset by **d**-th bias.

Summary

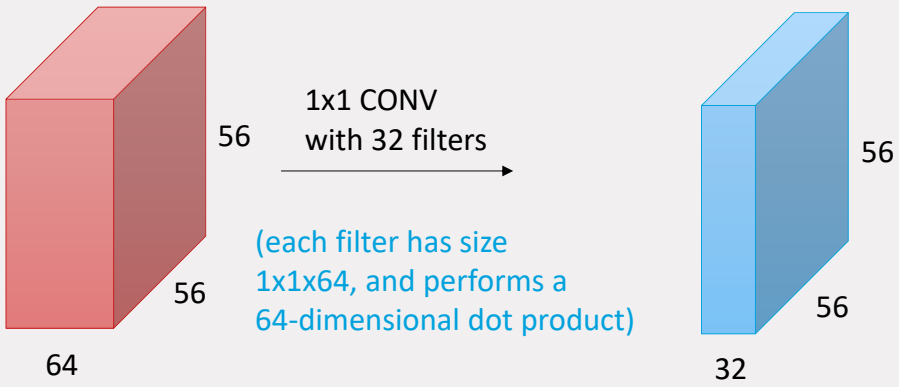
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters **K**
  - Their spatial extent **F**
  - The stride **S**
  - The amount of padding **P**
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume the **d**-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the **d**-th filter over the input volume with a stride of S, and then offset by **d**-th bias.

Common settings:

K = 32, 64, 128, 512 (power of 2)

- F = 3, S=1, P=1
- F = 5, S=1, P=2
- F = 5, S=2, P=?
- F = 1, S=1, P=0

1x1 convolutions



(each filter has size 1x1x64, and performs a 64-dimensional dot product)

Pytorch example

```
nn.Conv2d(in_channels=3,
          out_channels=16,
          kernel_size=3,
          stride=1,
          padding=1)
```

Conv2d

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

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

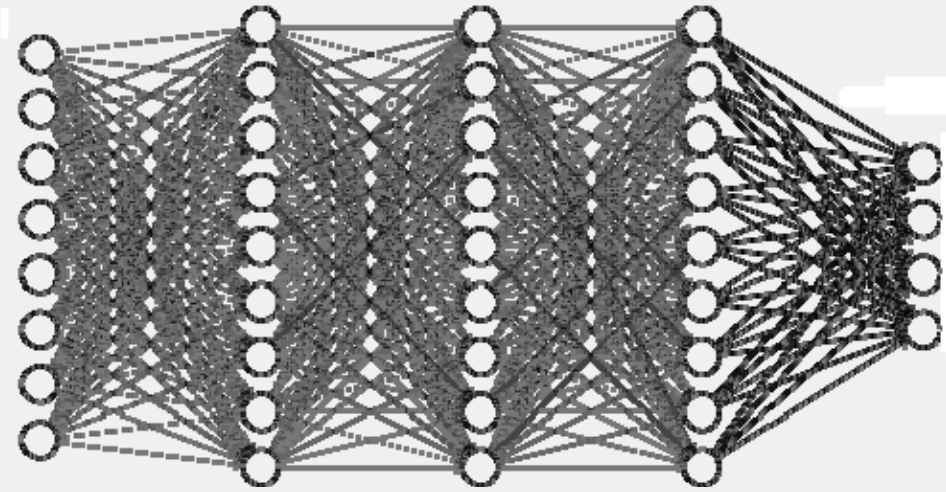
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

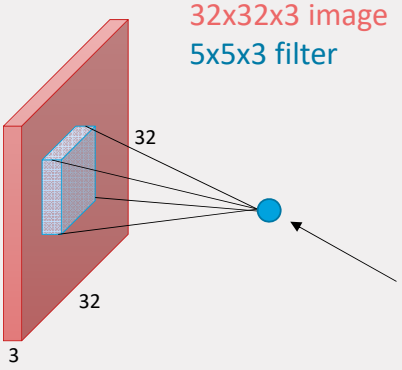
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

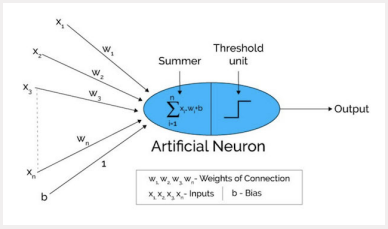
Recap: fully connected network



Parallel with the brain



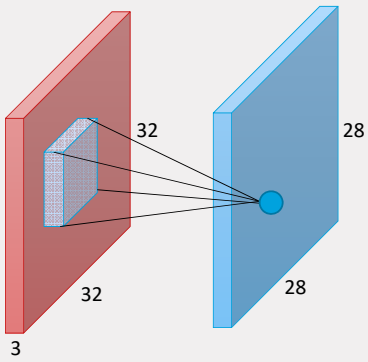
32x32x3 image
5x5x3 filter

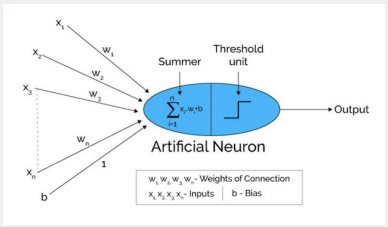


Its just a neuron with local connectivity

1 number:
The result of taking a dot product between the filter and a small chunk of the image

Parallel with the brain



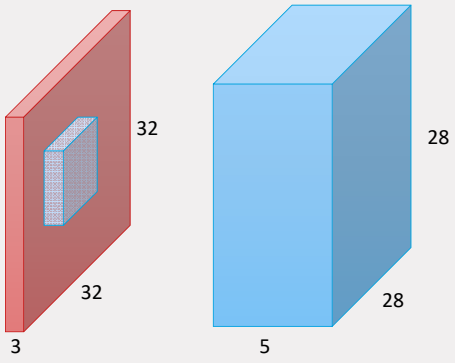


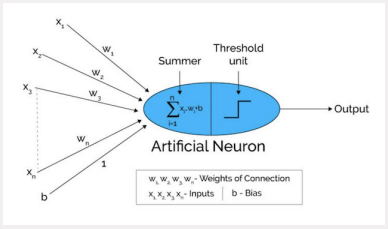
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share the same parameters

5x5 filter -> 5x5 'receptive field' for each neuron

Parallel with the brain

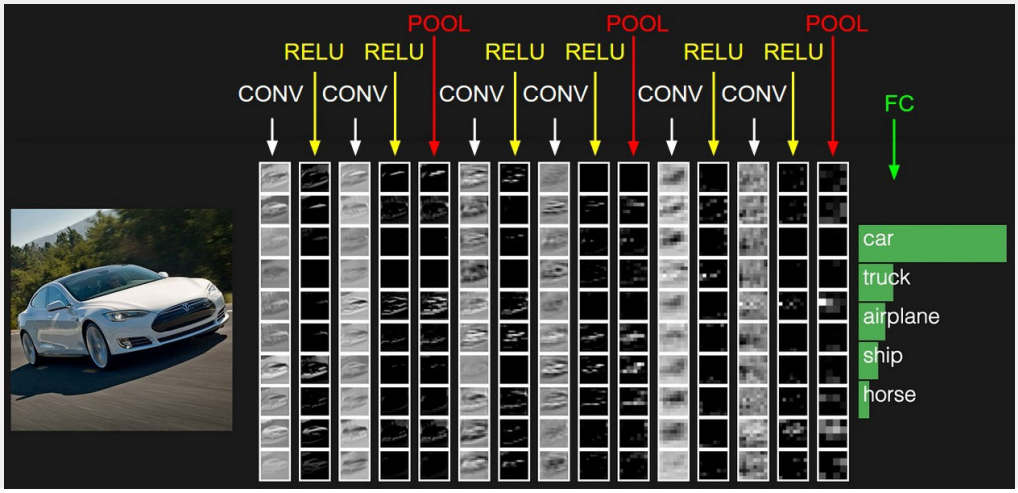




e.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

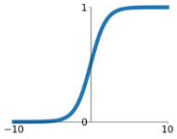
Activation functions



Activation Functions

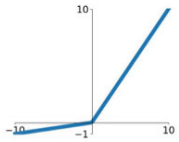
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



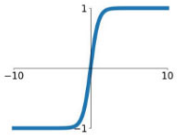
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

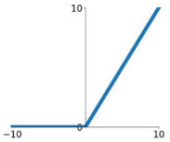


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

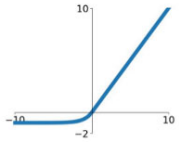
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation functions - Sigmoid

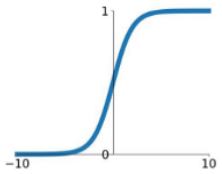
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating 'firing rate' of a neuron

3 problems

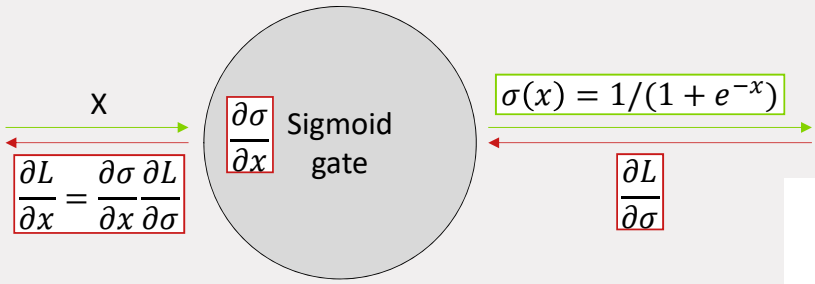
- Saturated neurons 'kill off' the gradient

Sigmoid

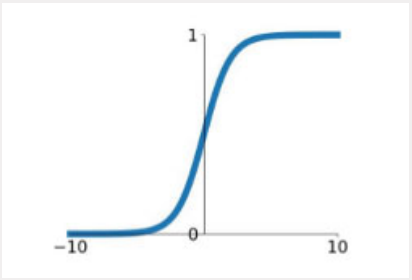
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Activation functions - Sigmoid



What happens when x = -10?
What happens when x = 0?
What happens when x = 10?



Activation functions - Sigmoid

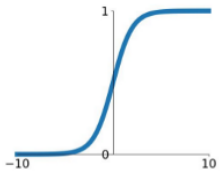
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating 'firing rate' of a neuron

3 problems

- Saturated neurons 'kill off' the gradient
- Sigmoid outputs are not zero-centered

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Activation functions - Sigmoid

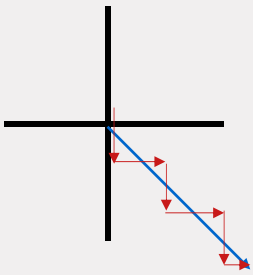
What happens when the input to a neuron is always positive …

$$f\left(\sum_i w_i x_i + b\right)$$

What can we say about the gradients on **w**?

Always all positive or all negative ☹

(This is also why we want zero-centered data)



Optimal w vector

Activation functions - Sigmoid

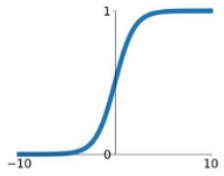
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating 'firing rate' of a neuron

3 problems

- Saturated neurons 'kill off' the gradient
- Sigmoid outputs are not zero-centered
- Exp() is relatively expensive

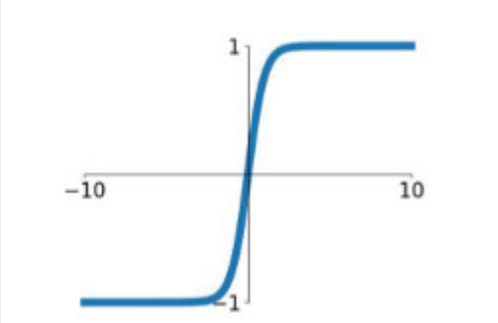
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Activation functions - Tanh

- Squashes numbers to range [-1,1]
- Zero-centered ☺
- Still kills gradients when saturated ☹

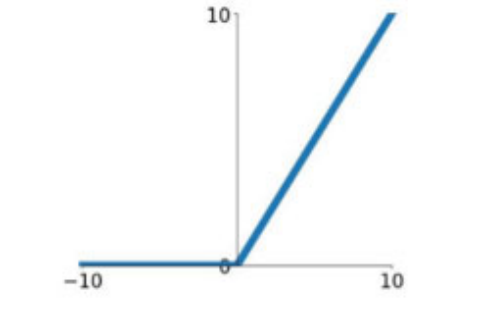


$\tanh(x)$

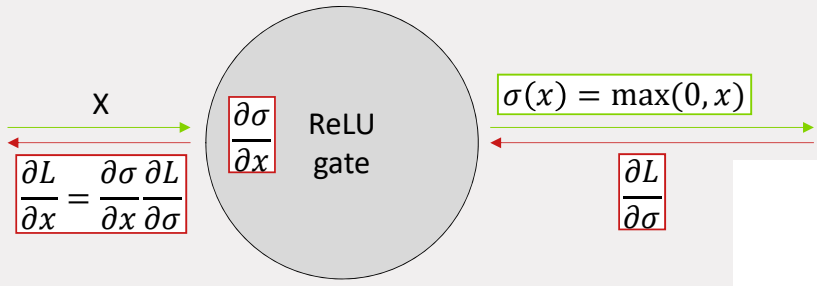
Activation functions - ReLU

- Computes $f(x) = \max(0, x)$

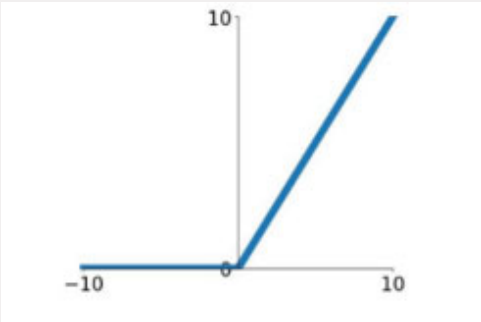
- **Does not saturate (in +region)**
- **Very computationally efficient**
- **Converges much faster than sigmoid/tanh in practice (~6x)**
- **Actually more biologically plausible than sigmoid**
- **Not zero-centered output**
- **One small annoyance?**



Activation functions - ReLU



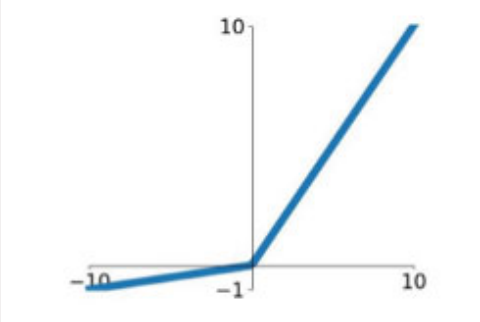
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?



Activation functions – Leaky ReLU

- Computes $f(x) = \max(0.01x, x)$

- **Does not saturate**
- **Very computationally efficient**
- **Converges much faster than sigmoid/tanh in practice (~6x)**
- **Will not 'die'**



- **Parametric ReLU** Computes $f(x) = \max(\alpha x, x)$

Learnable parameter α

Activation functions - Maxout

- Does not have the basic form of dot product -> nonlinearity
- Generalises ReLU and Leaky ReLU
- Linear regime! Does not saturate! Does not die!

Maxout

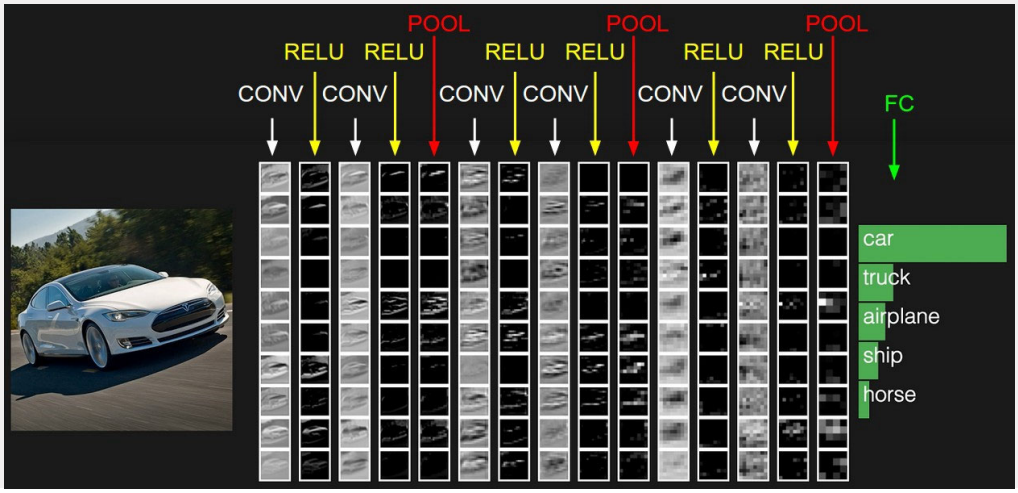
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- Doubles the number of parameters/neuron ☹

Activation functions: TLDR

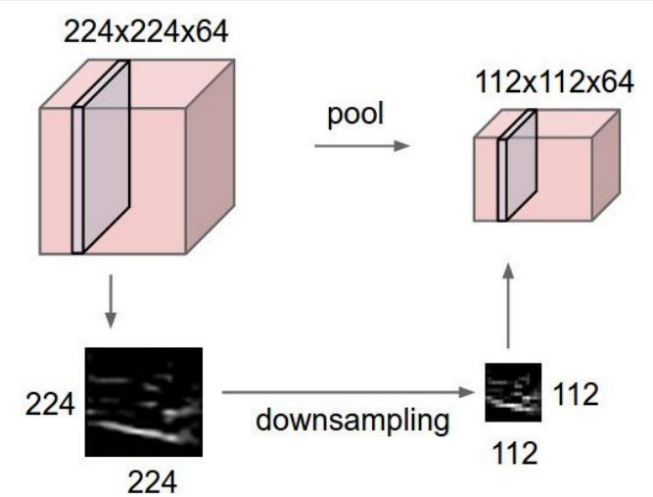
- Use **ReLU**.
- Try out **Leaky ReLU / Maxout**
- **Don't use sigmoid/tanh**

Pooling



Pooling layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



Pooling

Max pool with
2x2 filter and stride 2

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

6	8
3	4

Average pool with
2x2 filter and stride 2

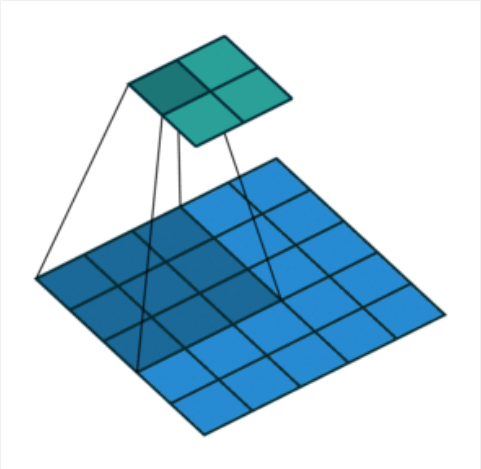
1	1	2	4
5	6	8	7
3	2	1	0
1	2	3	4

3.25	5.25
2	2

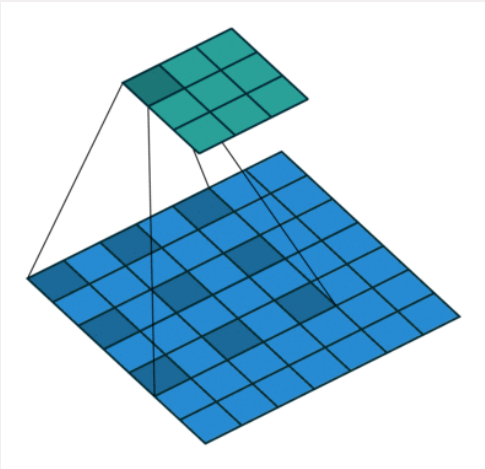
Typically no zero-padding in pooling operations

Pooling

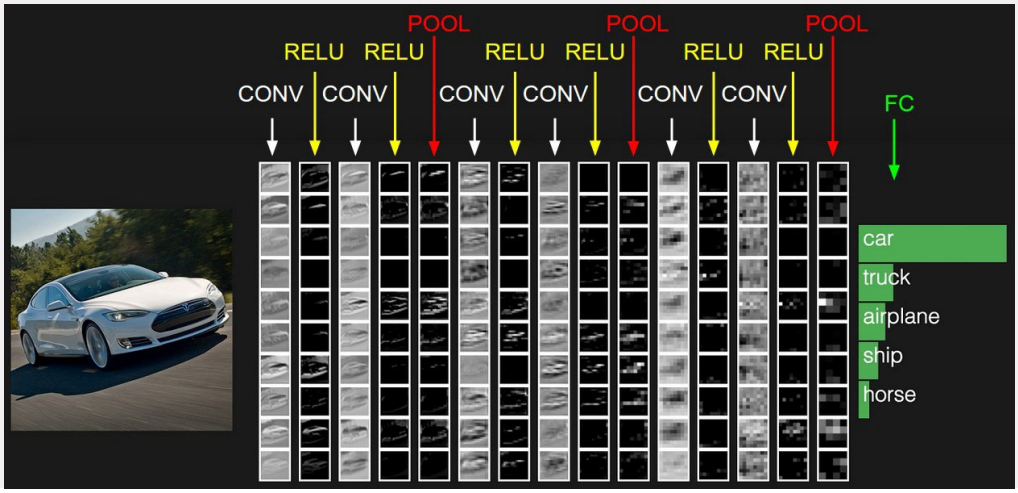
Strided convolutions



dilated convolutions



Fully connected layer (FC layer)



Summary

- ConvNets stack CONV, POOL and FC layers
- Trend towards smaller filters and deeper architectures
- Trend of getting rid of POOL/FC layers, use just convolutional layers instead
- Typical architectures look as follows:
  - [(CONV → Relu)*N → POOL] *M → (FC -> ReLU)*K → softmax
    where N is usually up to ≈3, M is ≈5, and 0 ≤ K ≤ 2.