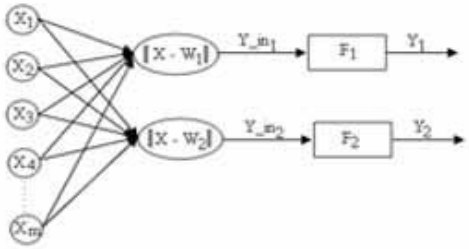
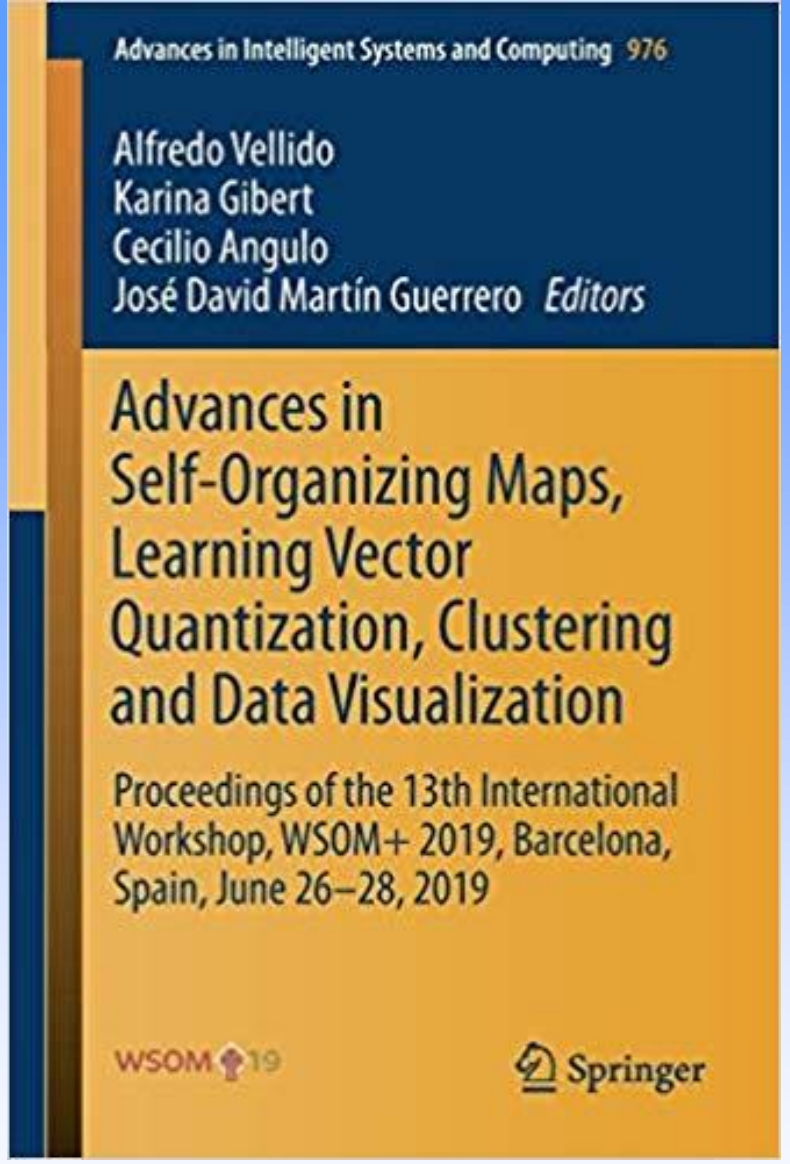


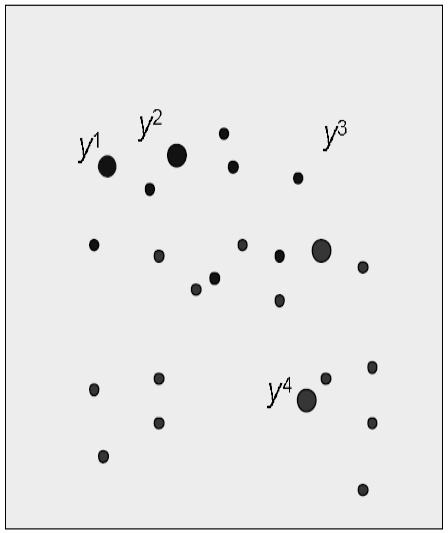
GARUDA
CYBER INDONESIA

Learning Vector Quantization



Learning vector quantization

- Supplementary concept of « classes »
- → mixing of « quantization » and « classification » concepts!
- Algorithms: « neural » principle



LVQ1 algorithm

- Choice of initial codebook
- Selection of the winner:

$$d(\boldsymbol{x}^i, \boldsymbol{y}^k) \le d(\boldsymbol{x}^i, \boldsymbol{y}^j), \quad 1 \le j,k \le Q$$

- Adaptation of the winner:

$$\boldsymbol{y}^k(t+1) = \boldsymbol{y}^k(t) + \alpha\left(\boldsymbol{x}^i - \boldsymbol{y}^k(t)\right)$$

if $\boldsymbol{y}^k(t)$ and $\boldsymbol{x}^i$ belong to the same class, or

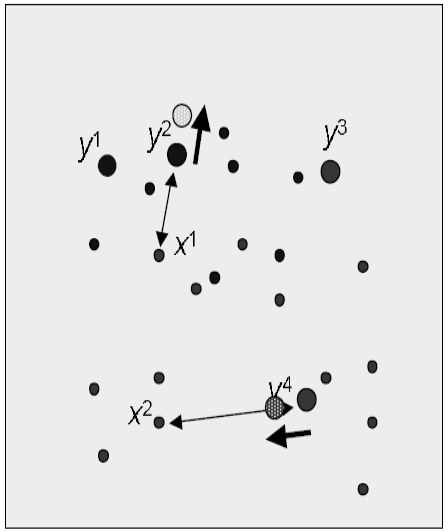
$$\boldsymbol{y}^k(t+1) = \boldsymbol{y}^k(t) - \alpha\left(\boldsymbol{x}^i - \boldsymbol{y}^k(t)\right)$$

if $\boldsymbol{y}^k(t)$ and $\boldsymbol{x}^i$ belong to different classes

LVQ1 algorithm

- $\boldsymbol{x}^1$ and $\boldsymbol{y}^2(t)$ belong to different classes
  → $\boldsymbol{y}^2(t)$ is moved away from $\boldsymbol{x}^1$

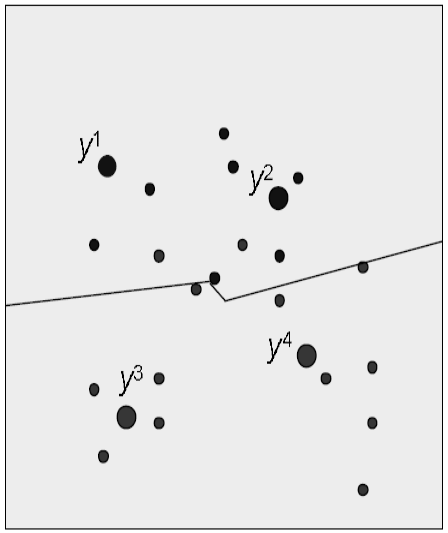
- $\boldsymbol{x}^2$ and $\boldsymbol{y}^2(t)$ belong to the same class
  → $\boldsymbol{y}^4(t)$ is moved towards $\boldsymbol{x}^2$



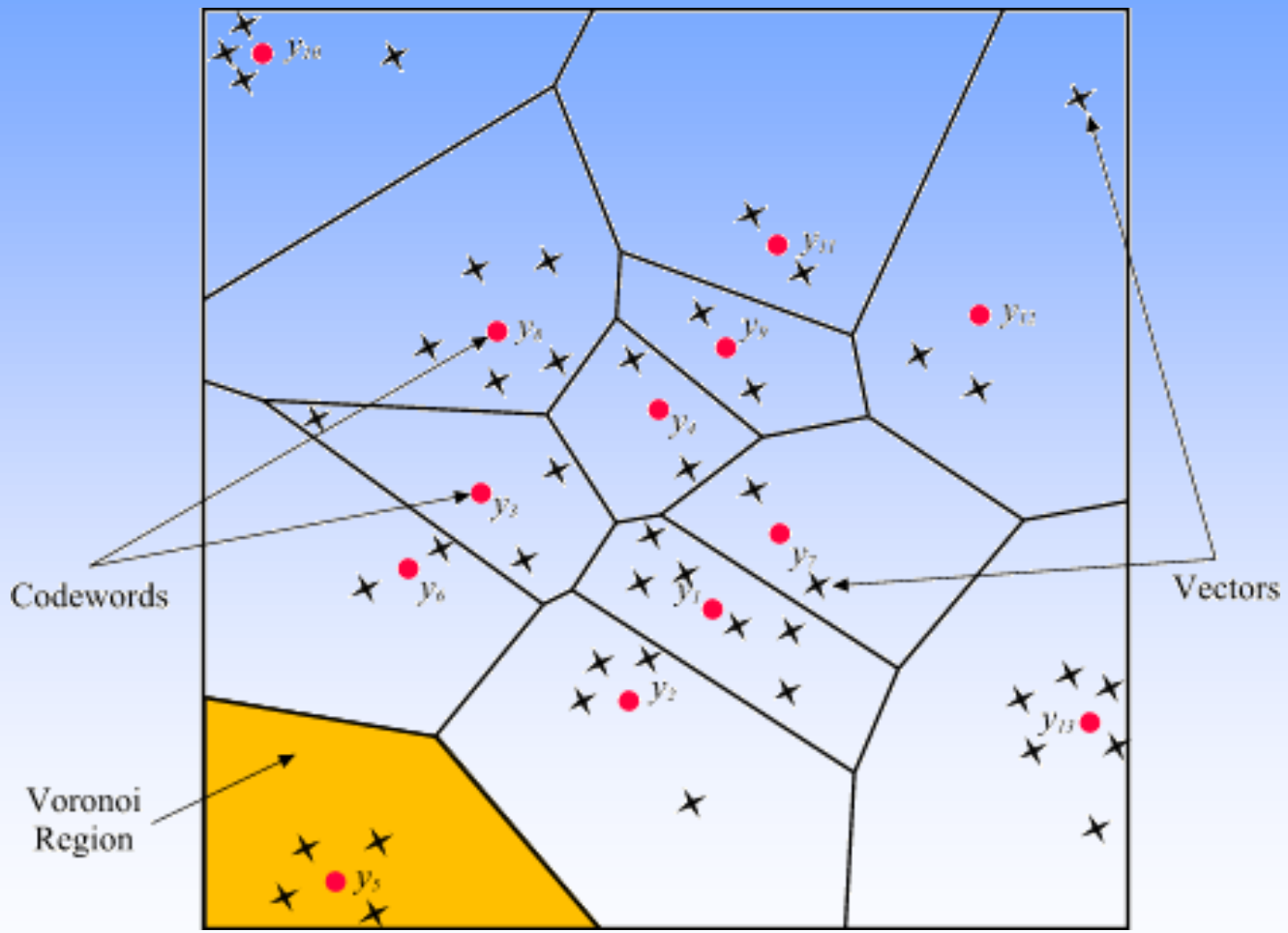
LVQ1 algorithm

- after convergence : centroids are located in regions where there are data points of the *same* class

- consequence: classification



Voronoi Region, Codeword



LVQ1 algorithm

- Same advantages and drawbacks as Competitive Learning:
  - adaptive algorithm
  - codebook convergences (under Robbins-Monro conditions)
  - convergence to local minima (choice of initial codebook, $\alpha(t)$, …)
  - lost centroids
  - neither a good VQ method, nor a good classification one!

LVQ2 algorithm

- Choice of initial codebook
- Selection of two winners:

$$d(\boldsymbol{x}^i, \boldsymbol{y}^{k1}) \leq d(\boldsymbol{x}^i, \boldsymbol{y}^j), \quad 1 \leq j, k1 \leq Q$$
$$d(\boldsymbol{x}^i, \boldsymbol{y}^{k2}) \leq d(\boldsymbol{x}^i, \boldsymbol{y}^j), \quad 1 \leq j, k2 \leq Q \setminus \{k1\}$$

- Adaptation of the winners:
  - 3 conditions to fulfill
  - under these 3 conditions, the two winners are moved

LVQ2 algorithm

- 3 conditions:
  - one of the two winners (say $\boldsymbol{y}^{ka}$) and $\boldsymbol{x}^i$ belong to the same class
  - the other winner (say $\boldsymbol{y}^{kb}$) and $\boldsymbol{x}^i$ belong different classes
  - $\boldsymbol{x}^i$ is located inside a window centered on the midpoint of $\{\boldsymbol{y}^{ka}, \boldsymbol{y}^{kb}\}$

  Note a=1 and b=2 or a=2 and b=1 !

- 2 moves:

$$\boldsymbol{y}^{ka}(t+1) = \boldsymbol{y}^{ka}(t) + \alpha\left(\boldsymbol{x}^i - \boldsymbol{y}^{ka}(t)\right)$$

$$\boldsymbol{y}^{kb}(t+1) = \boldsymbol{y}^{kb}(t) - \alpha\left(\boldsymbol{x}^i - \boldsymbol{y}^{kb}(t)\right)$$

Parameters Used

Following are the parameters used in LVQ training process as well as in the flowchart

- $\mathbf{x}$ = training vector $(x_1,...,x_i,...,x_n)$
- $\mathbf{T}$ = class for training vector $\mathbf{x}$
- $\mathbf{w_j}$ = weight vector for $\mathbf{j^{th}}$ output unit
- $\mathbf{C_j}$ = class associated with the $\mathbf{j^{th}}$ output unit

Training Algorithm

Step 1 – Initialize reference vectors, which can be done as follows –

- **Step 1(a)** – From the given set of training vectors, take the first "**m**" (number of clusters) training vectors and use them as weight vectors. The remaining vectors can be used for training.
- **Step 1(b)** – Assign the initial weight and classification randomly.
- **Step 1(c)** – Apply K-means clustering method.

Step 2 – Initialize reference vector α

Step 3 – Continue with steps 4-9, if the condition for stopping this algorithm is not met.

Step 4 – Follow steps 5-6 for every training input vector **x**.

Step 5 – Calculate Square of Euclidean Distance for **j = 1 to m** and **i = 1 to n**

$$D(j) = \sum_{i=1}^{n}\sum_{j=1}^{m}(x_i - w_{ij})^2$$

Step 6 – Obtain the winning unit **J** where **D(j)** is minimum.

Step 7 – Calculate the new weight of the winning unit by the following relation –

$$\text{if } \mathbf{T} = \mathbf{C_j} \text{ then } w_j(new) = w_j(old) + \alpha[x - w_j(old)]$$

$$\text{if } \mathbf{T} \neq \mathbf{C_j} \text{ then } w_j(new) = w_j(old) - \alpha[x - w_j(old)]$$

Step 8 – Reduce the learning rate α.

Step 9 – Test for the stopping condition. It may be as follows –

- Maximum number of epochs reached.
- Learning rate reduced to a negligible value.

LVQ2

As discussed, the concept of other variants of LVQ above, the condition of LVQ2 is formed by window. This window will be based on the following parameters –

- **x** – the current input vector
- $\mathbf{y_c}$ – the reference vector closest to **x**
- $\mathbf{y_r}$ – the other reference vector, which is next closest to **x**
- $\mathbf{d_c}$ – the distance from **x** to $\mathbf{y_c}$
- $\mathbf{d_r}$ – the distance from **x** to $\mathbf{y_r}$

The input vector **x** falls in the window, if

$$\frac{d_c}{d_r} > 1 - \theta \; and \; \frac{d_r}{d_c} > 1 + \theta$$

Here, θ is the number of training samples.

Updating can be done with the following formula –

$$y_c(t+1) = y_c(t) + \alpha(t)[x(t) - y_c(t)] \textbf{ (belongs to different class)}$$

$$y_r(t+1) = y_r(t) + \alpha(t)[x(t) - y_r(t)] \textbf{ (belongs to same class)}$$

Here α is the learning rate.

LVQ2.1

In LVQ2.1, we will take the two closest vectors namely $\mathbf{y_{c1}}$ and $\mathbf{y_{c2}}$ and the condition for window is as follows –

$$Min\left[\frac{d_{c1}}{d_{c2}}, \frac{d_{c2}}{d_{c1}}\right] > (1 - \theta)$$

$$Max\left[\frac{d_{c1}}{d_{c2}}, \frac{d_{c2}}{d_{c1}}\right] < (1 + \theta)$$

Updating can be done with the following formula –

$y_{c1}(t + 1) = y_{c1}(t) + \alpha(t)[x(t) - y_{c1}(t)]$ **(belongs to different class)**

$y_{c2}(t + 1) = y_{c2}(t) + \alpha(t)[x(t) - y_{c2}(t)]$ **(belongs to same class)**

Here, α is the learning rate.

LVQ3

In LVQ3, we will take the two closest vectors namely $\mathbf{y_{c1}}$ and $\mathbf{y_{c2}}$ and the condition for window is as follows –

$$Min\left[\frac{d_{c1}}{d_{c2}}, \frac{d_{c2}}{d_{c1}}\right] > (1 - \theta)(1 + \theta)$$

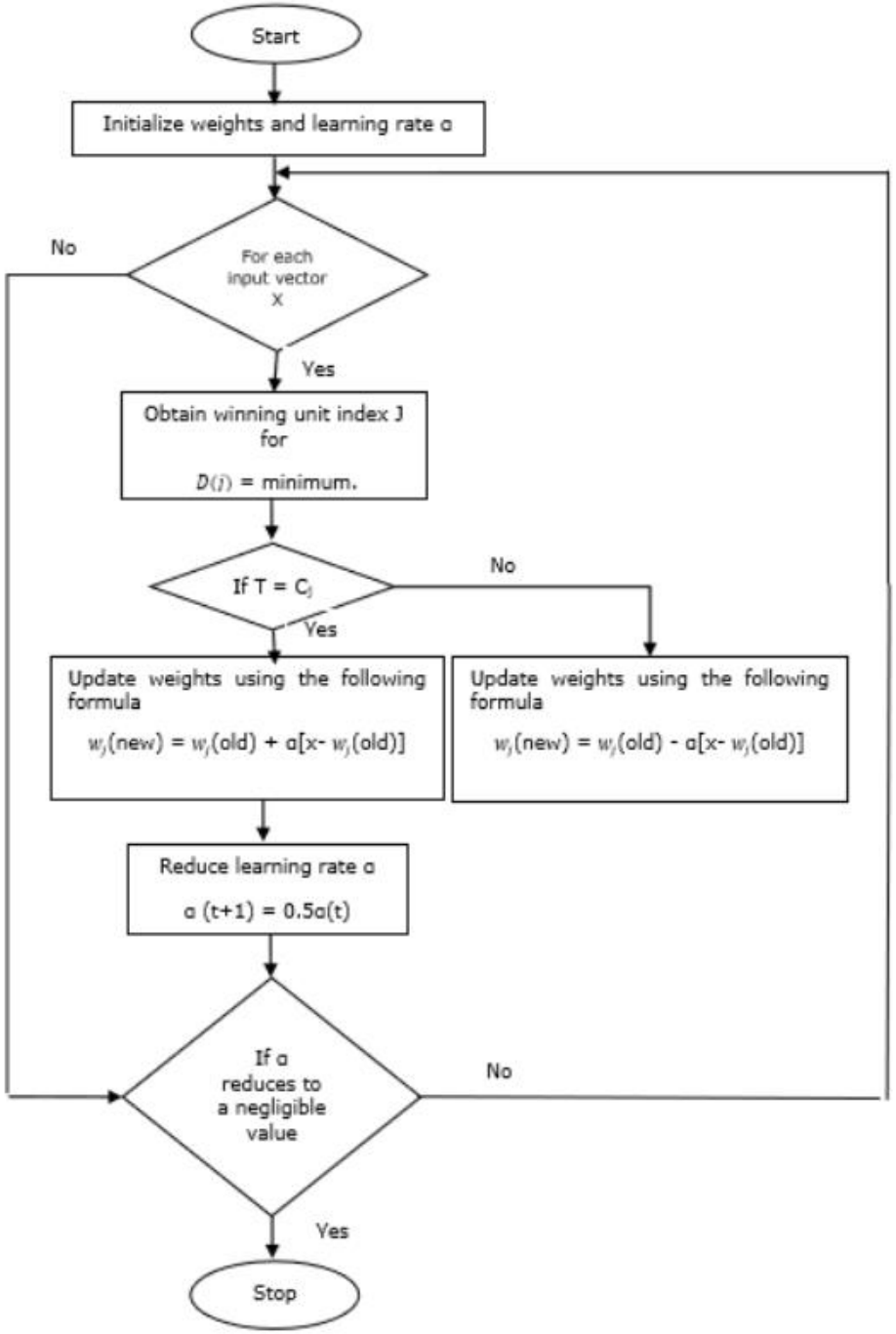
Here $\theta \approx 0.2$

Updating can be done with the following formula –

$y_{c1}(t + 1) = y_{c1}(t) + \beta(t)[x(t) - y_{c1}(t)]$ **(belongs to different class)**

$y_{c2}(t + 1) = y_{c2}(t) + \beta(t)[x(t) - y_{c2}(t)]$ **(belongs to same class)**

Here β is the multiple of the learning rate α and $\beta = m\alpha(t)$ for every **0.1 < m < 0.5**


Start
Initialize weights and learning rate α
For each input vector X
No
Yes
Obtain winning unit index J for
D(j) = minimum.
If T = Cj
No
Yes
Update weights using the following formula
wj(new) = wj(old) + α[x- wj(old)]
Update weights using the following formula
wj(new) = wj(old) - α[x- wj(old)]
Reduce learning rate α
α (t+1) = 0.5α(t)
If α reduces to a negligible value
No
Yes
Stop

Convolutional neural network (CNN)

History

Yann LeCun, Professor of Computer Science
The Courant Institute of Mathematical Sciences
New York University
Room 1220, 715 Broadway, New York, NY 10003, USA.
(212)998-3283 yann@cs.nyu.edu

- In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

About CNN's

- CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.
- They designed a network structure that implicitly extracts relevant features.
- Convolutional Neural Networks are a special kind of multi-layer neural networks.

About CNN's

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as handwritten characters).

A bit of history:

Hubel & Wiesel,

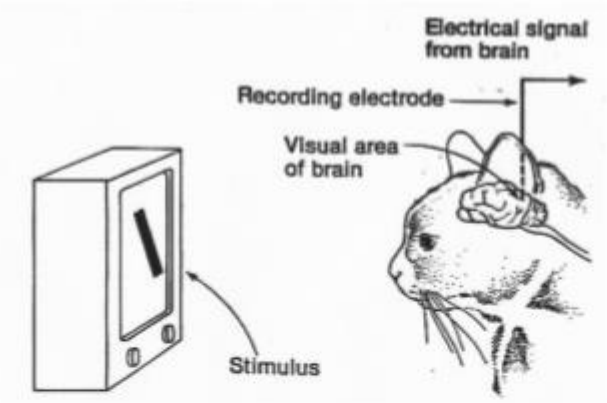
1959

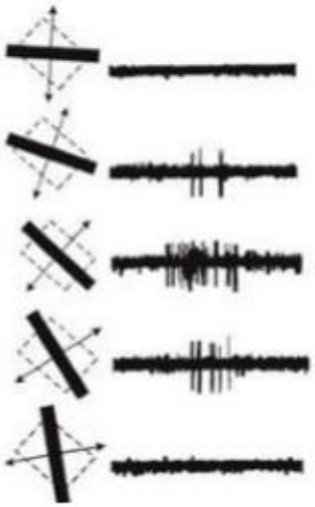
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

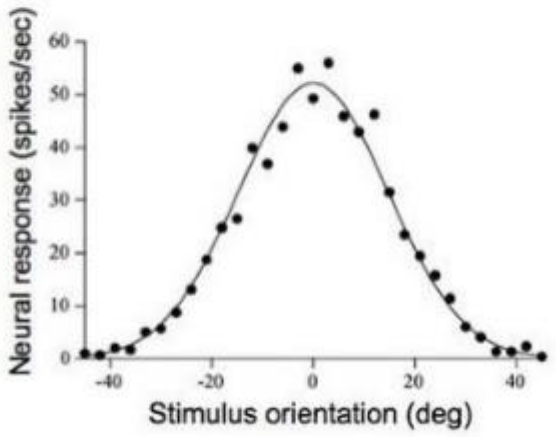
1962

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

1968…

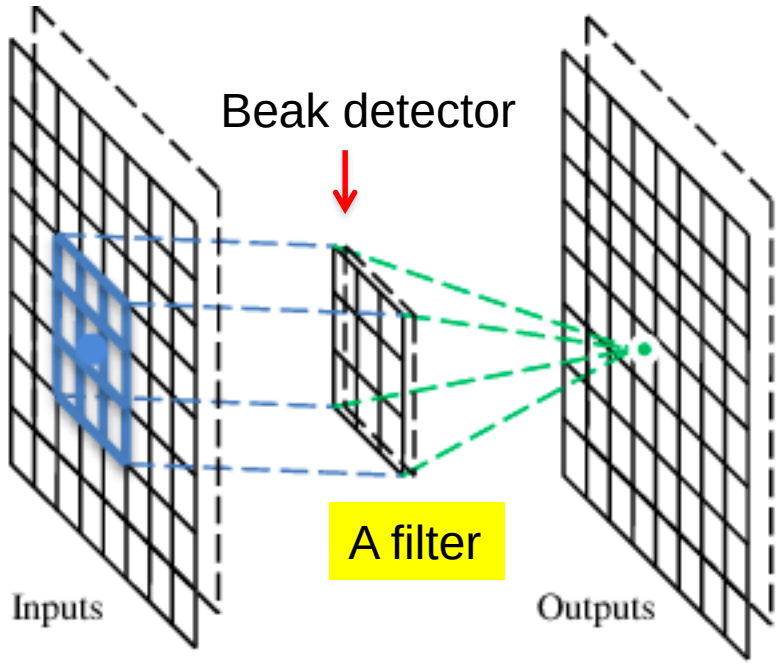






A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product →

3 -1

6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

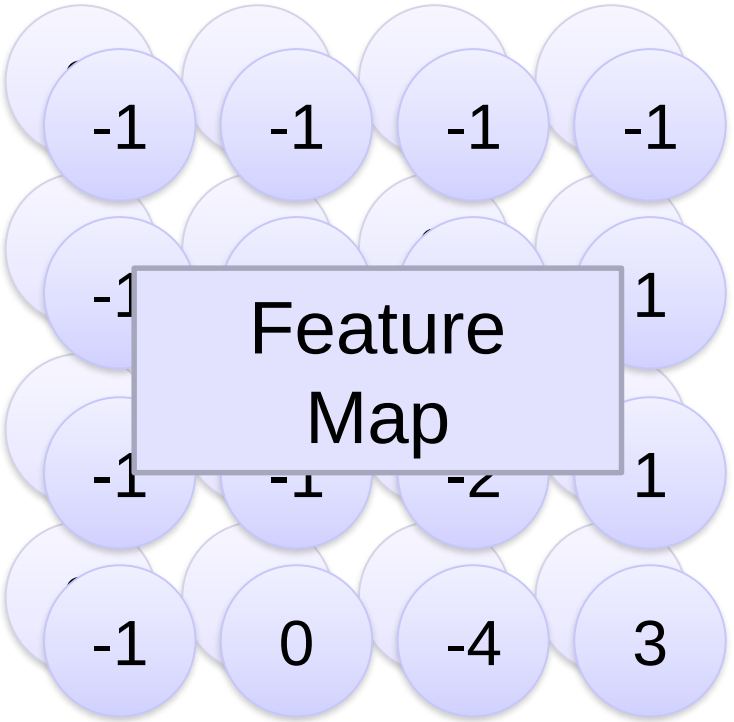
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

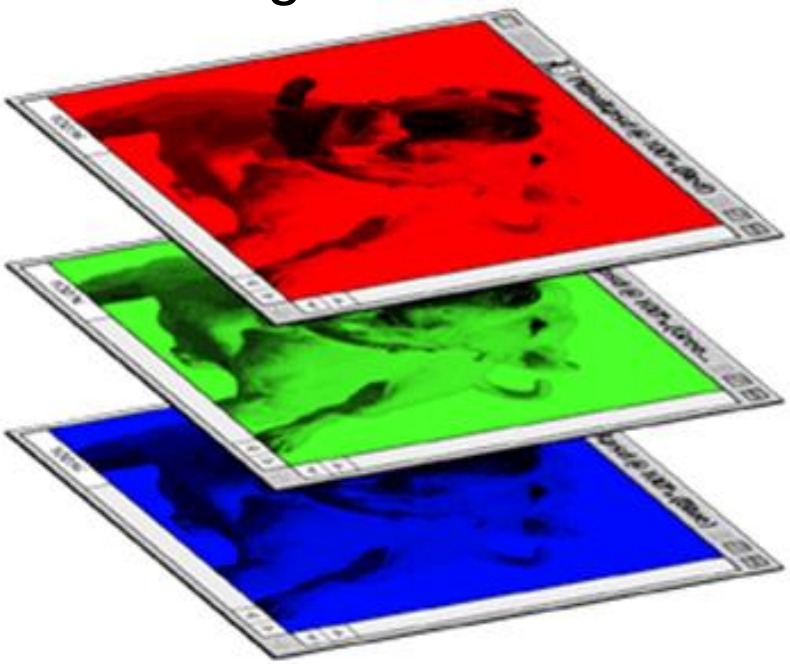
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

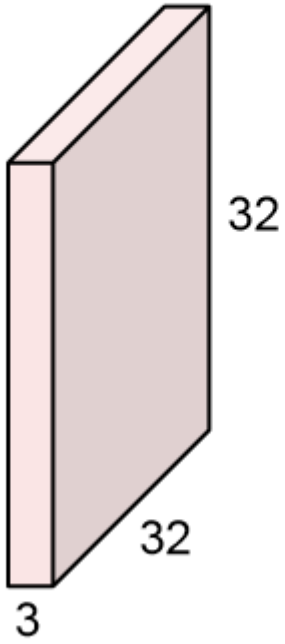
Color image



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Convolution Layer

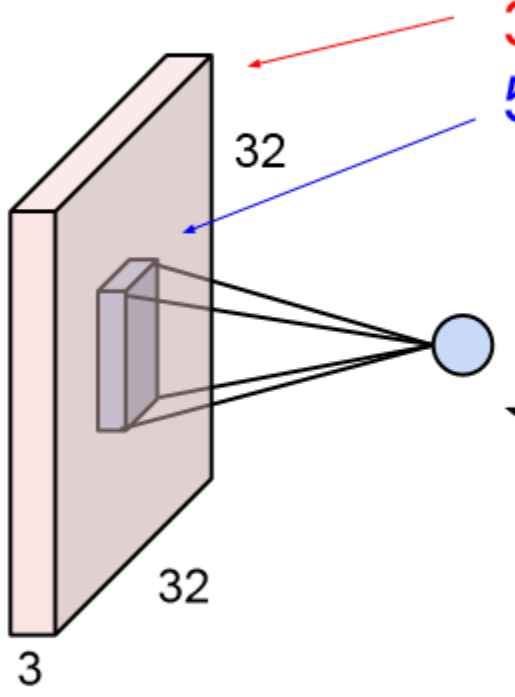
32x32x3 image



5x5x3 filter

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

Convolution Layer



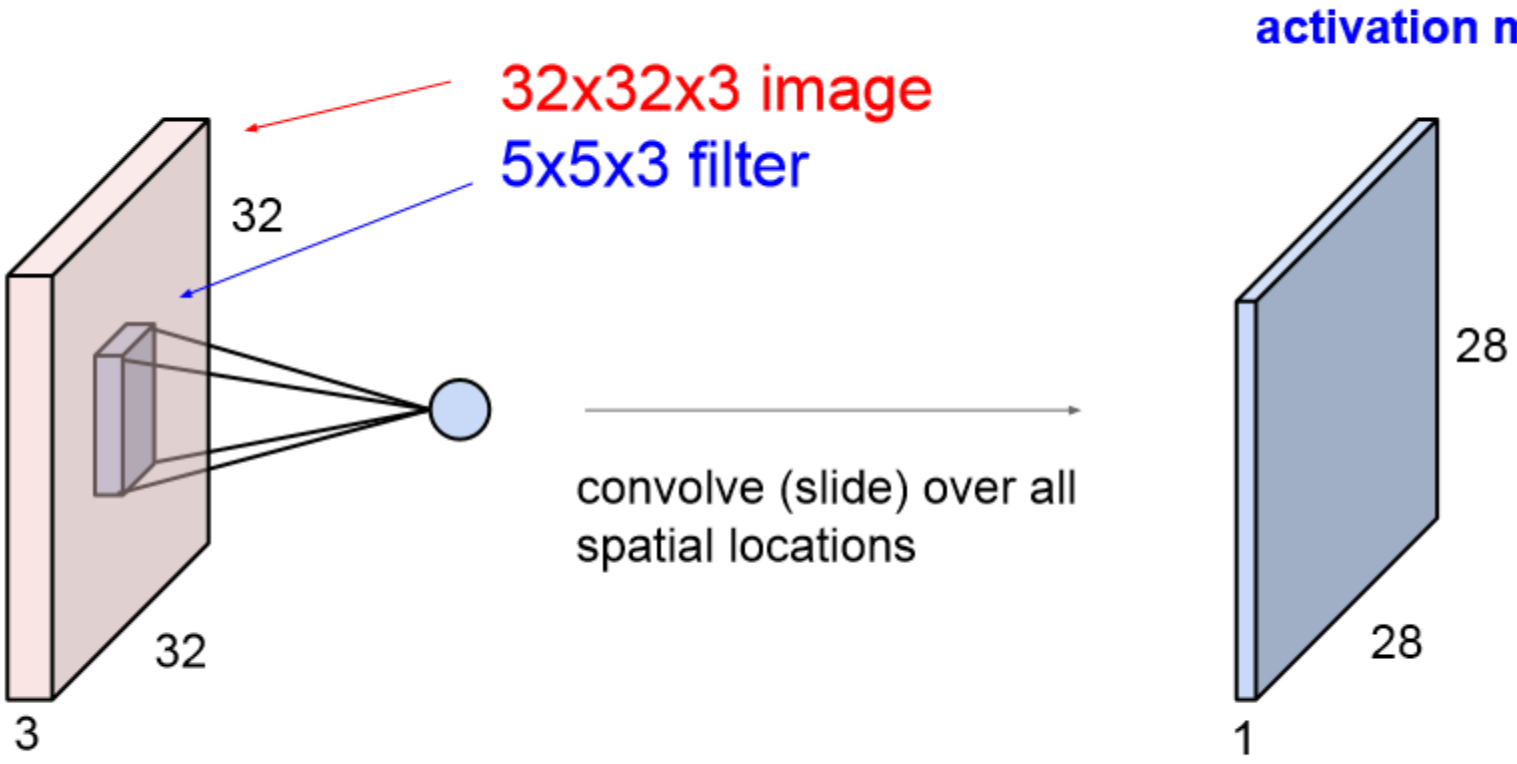
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

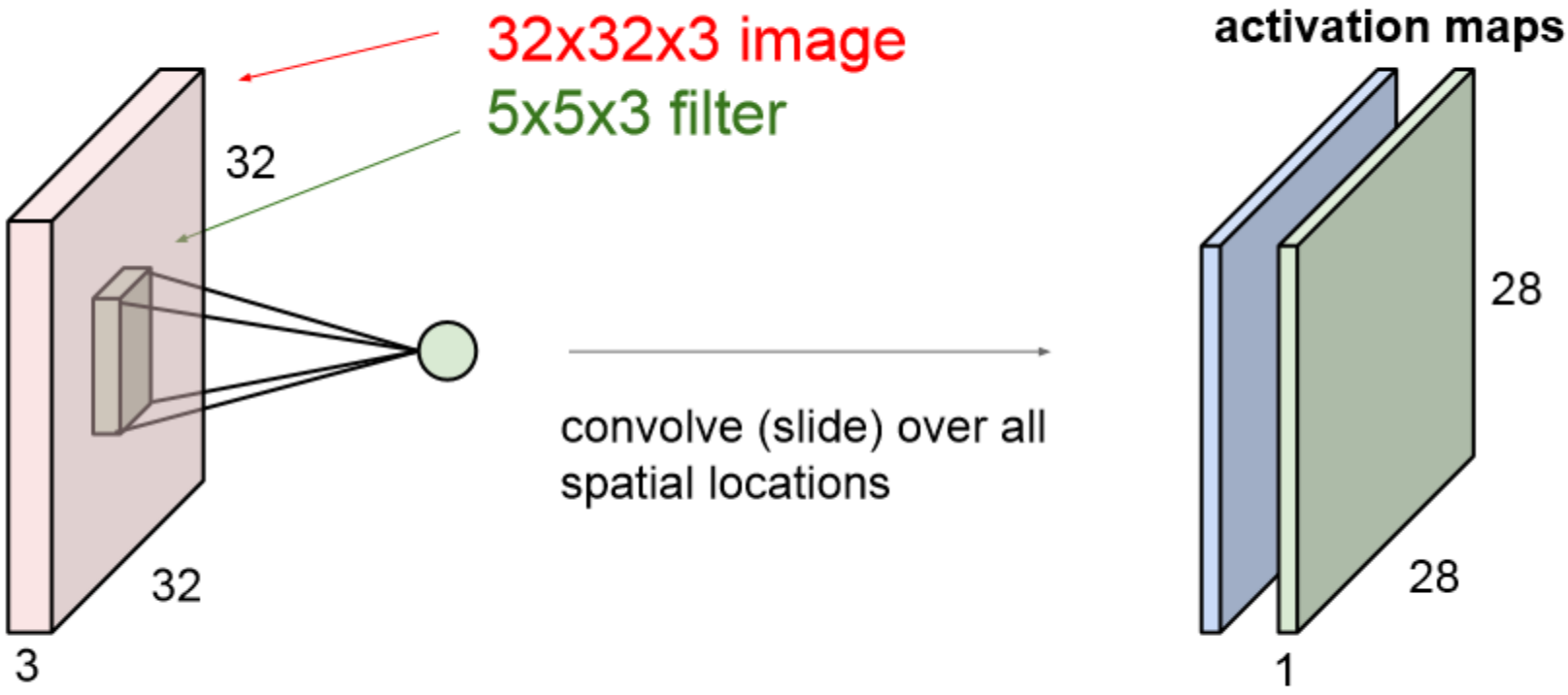
$$w^T x + b$$

Convolution Layer

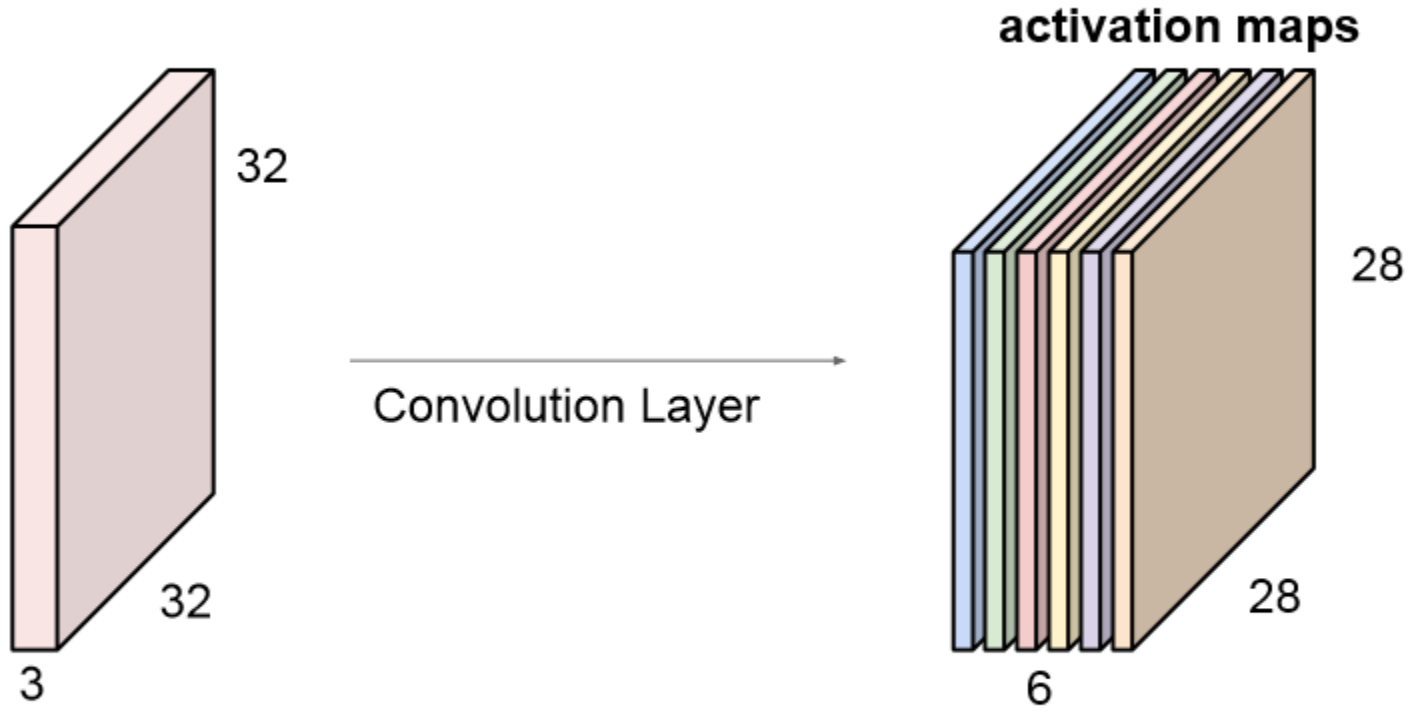


Convolution Layer

consider a second, green filter

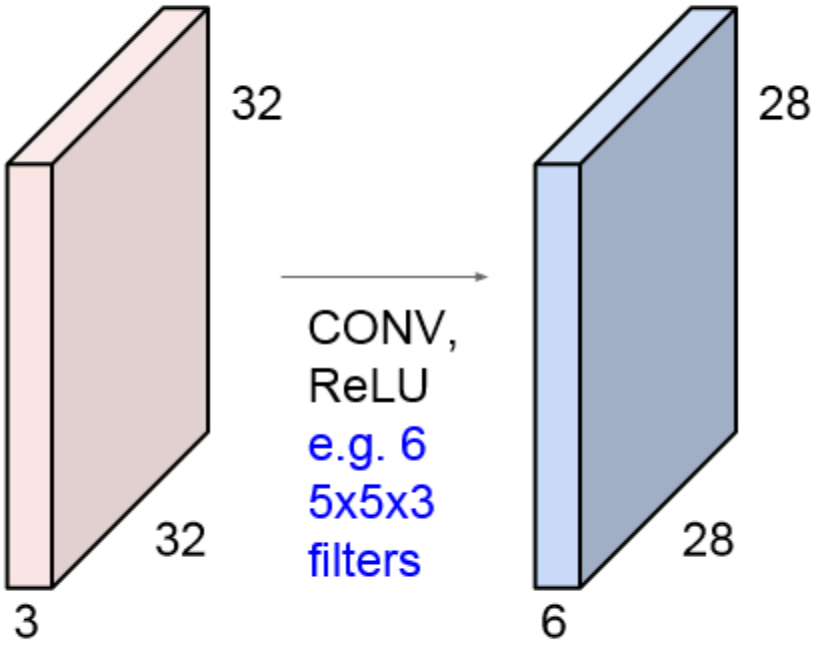


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

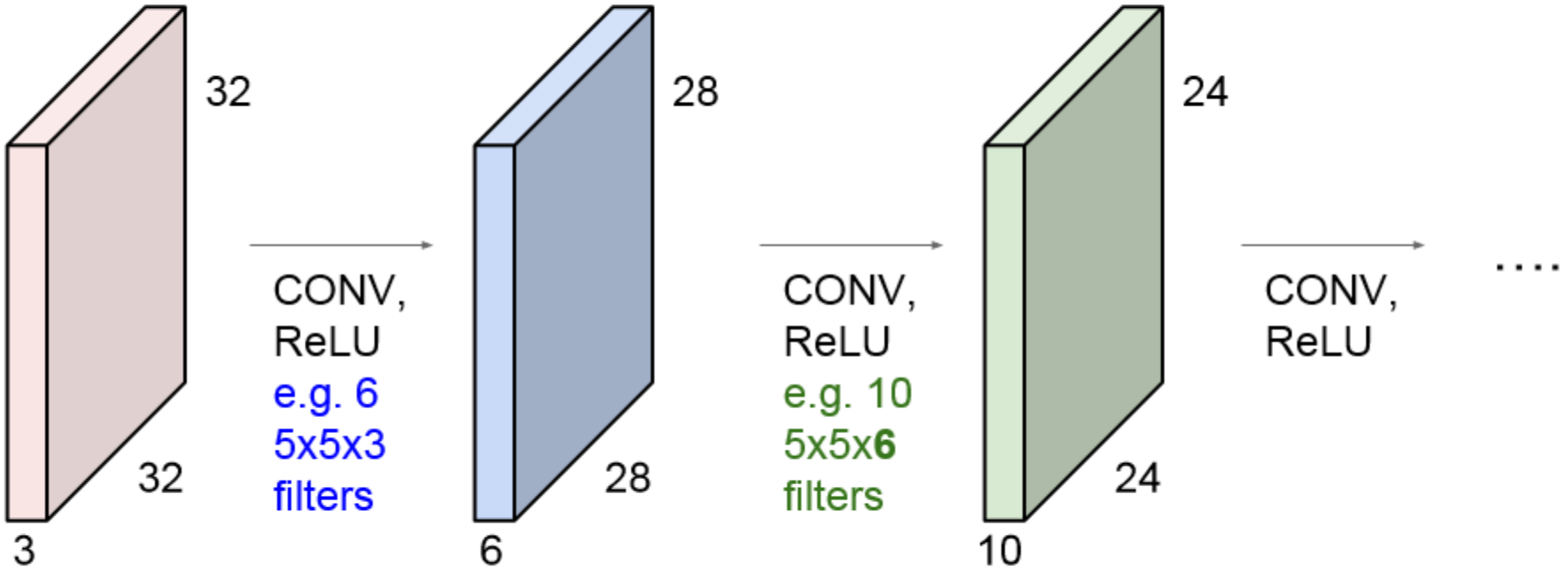


We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

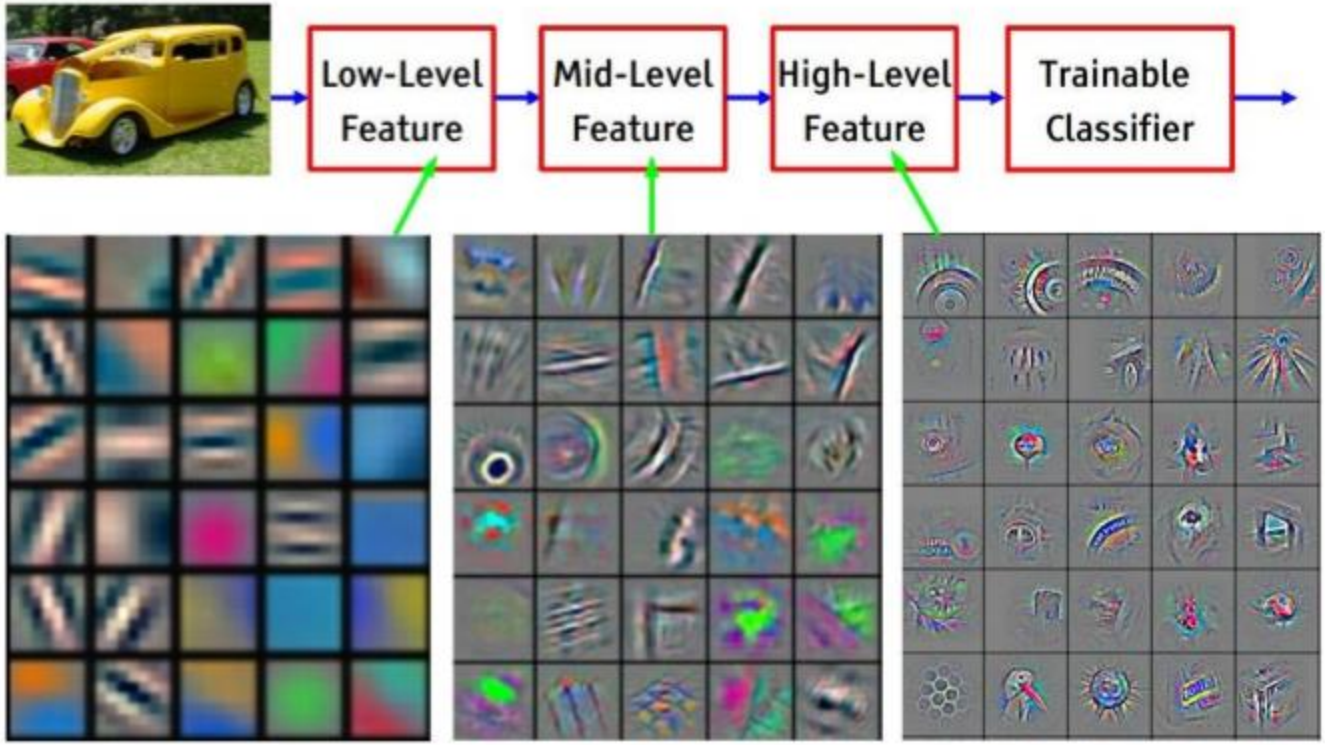


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



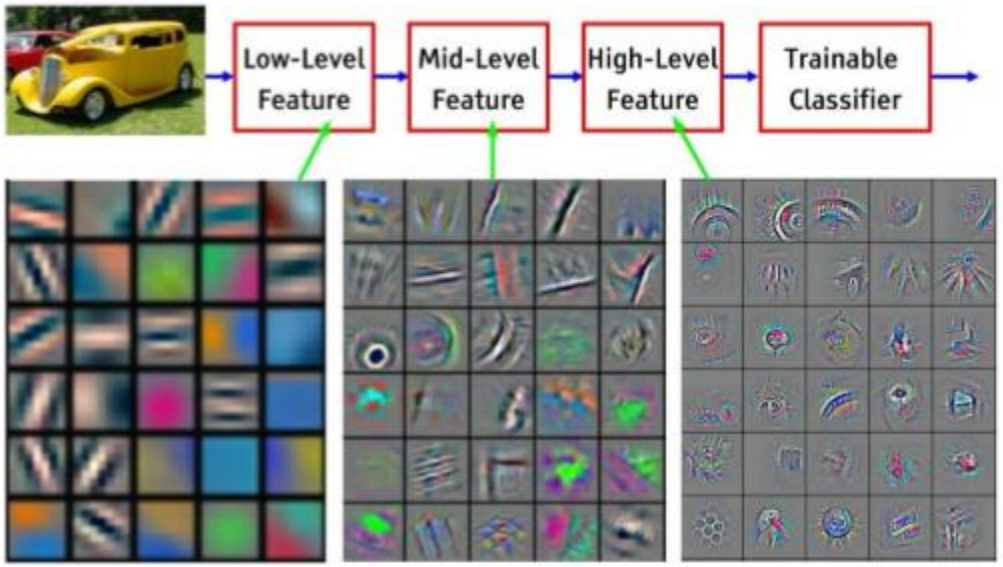
Preview

[From recent Yann LeCun slides]



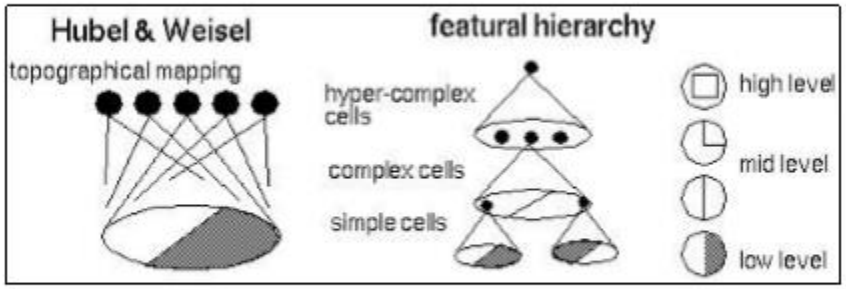
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

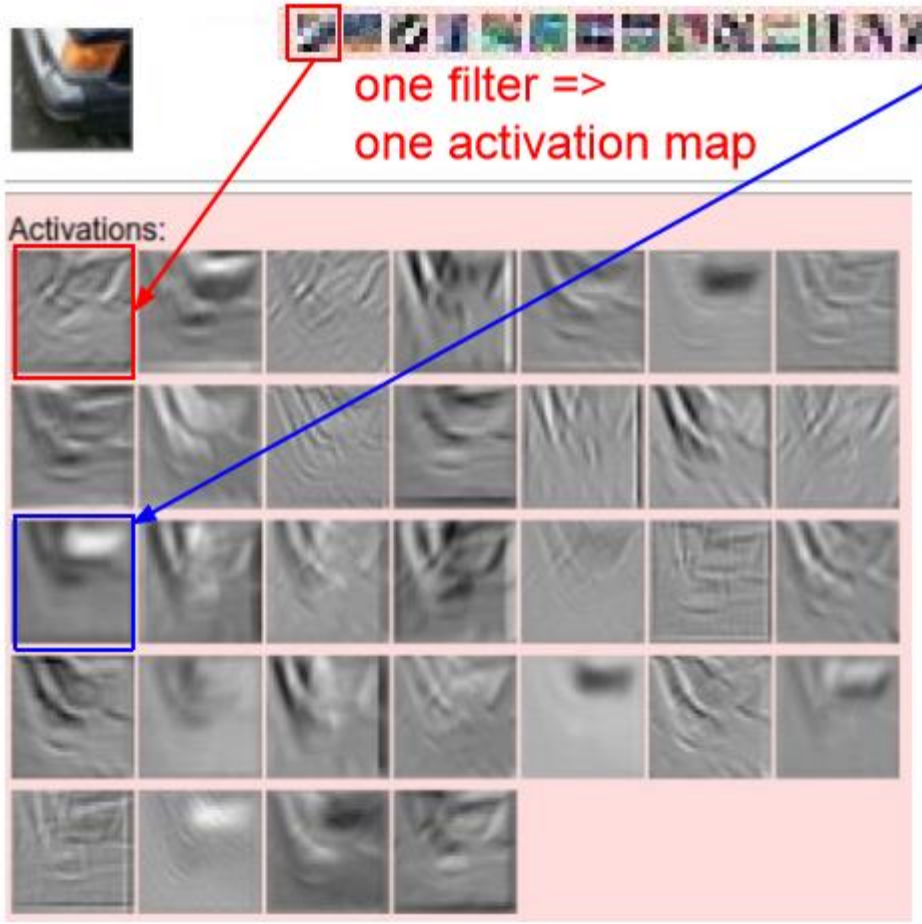
Preview



[From recent Yann LeCun slides]

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]





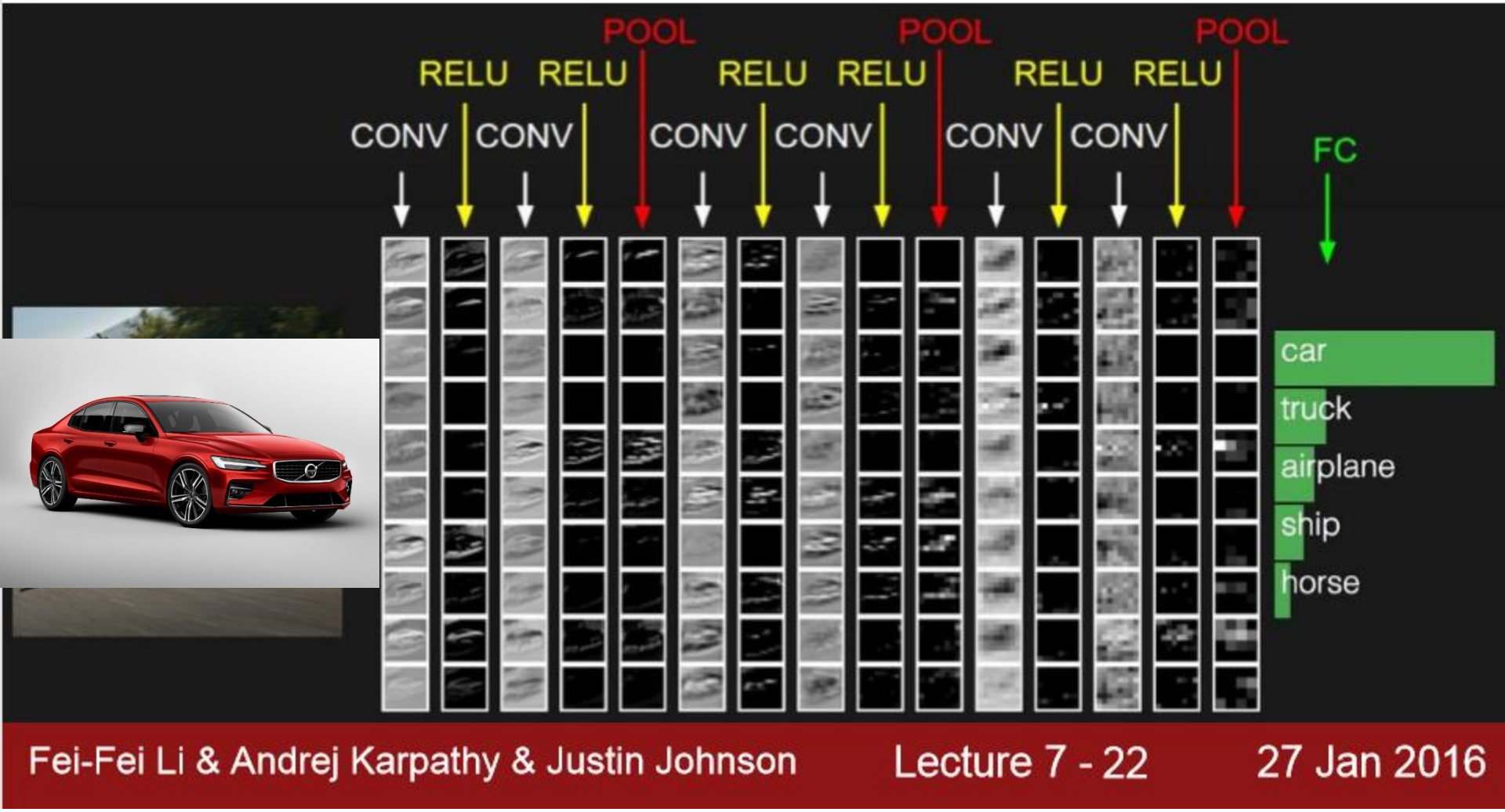
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

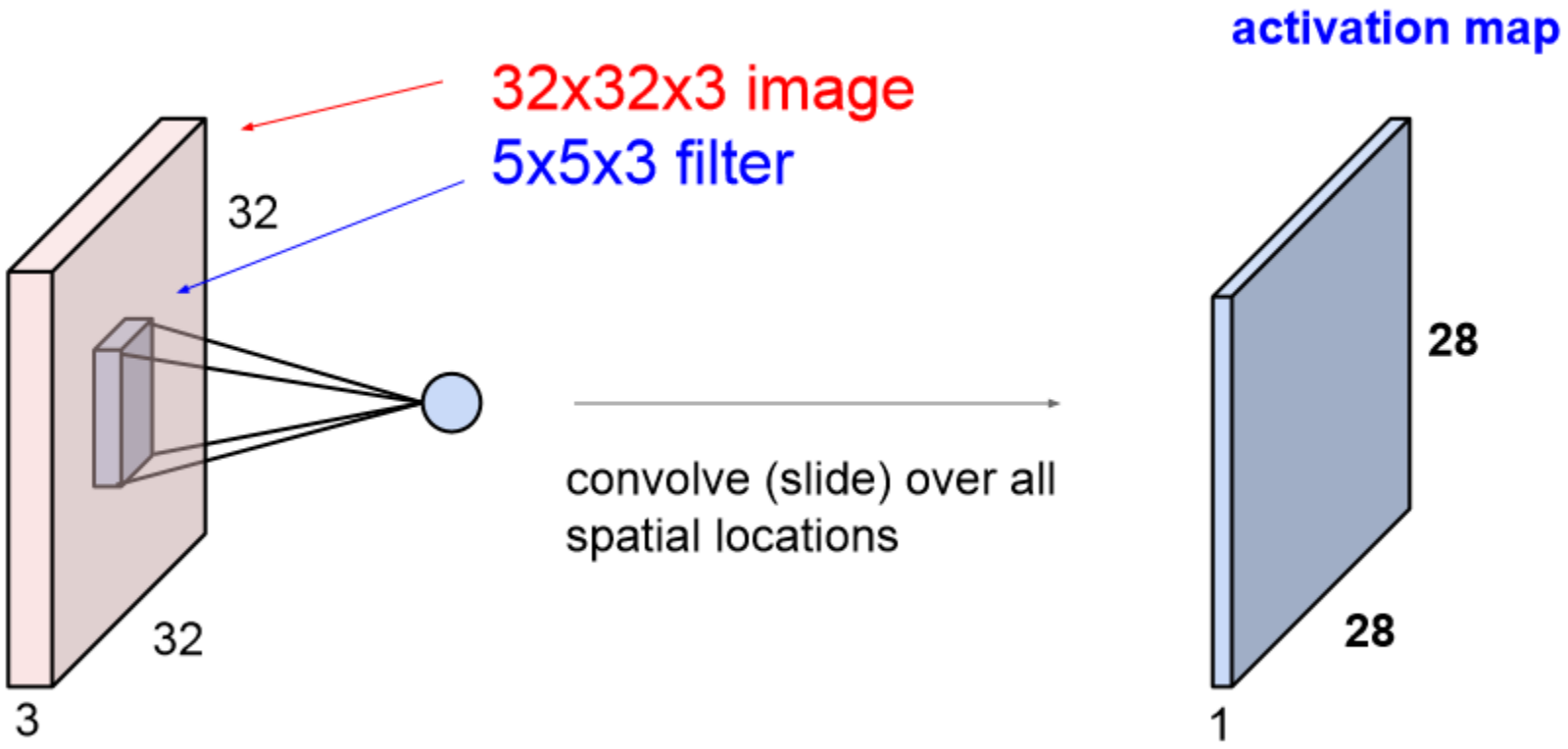
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

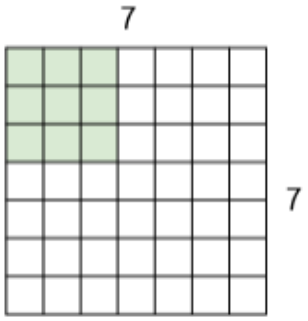
preview:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:

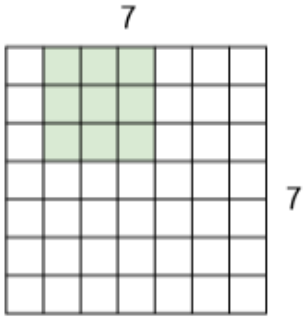


A closer look at spatial dimensions:



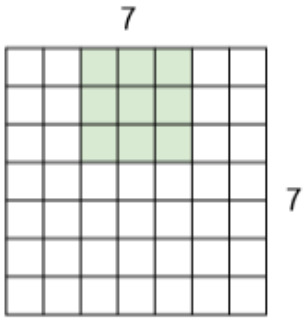
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



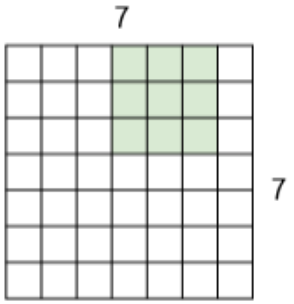
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



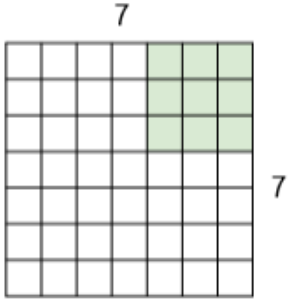
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

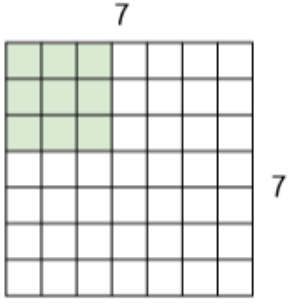
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

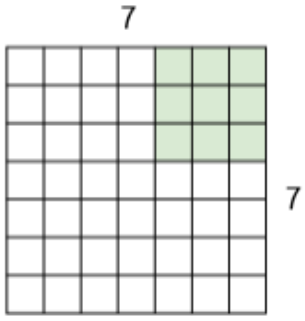
=> 5x5 output

A closer look at spatial dimensions:



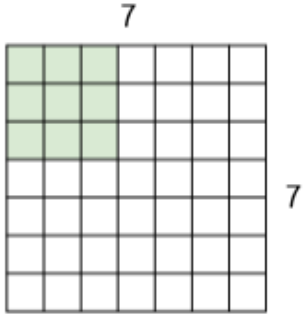
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



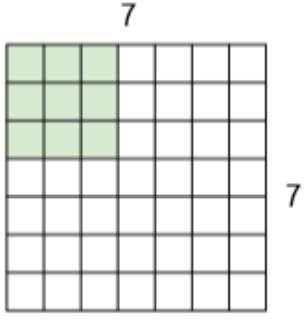
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



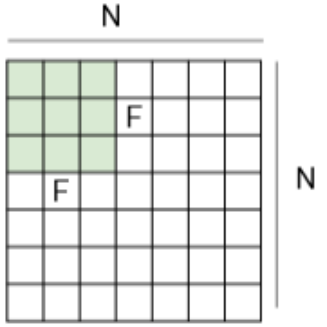
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

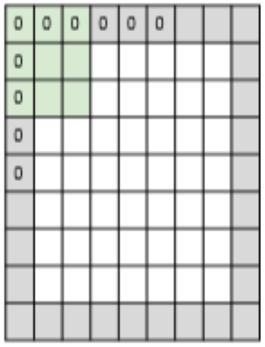
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

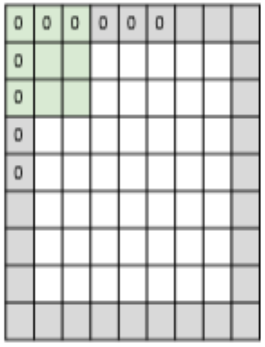
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

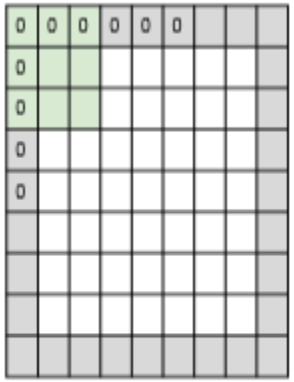
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

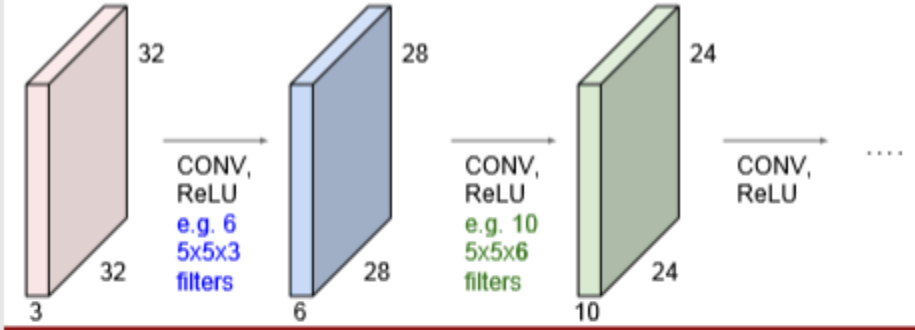
In practice: Common to zero pad the border



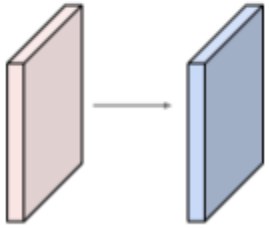
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to...
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



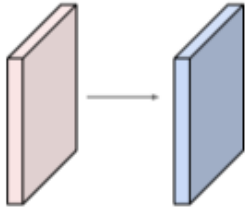
Examples time:



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

Output volume size: ?

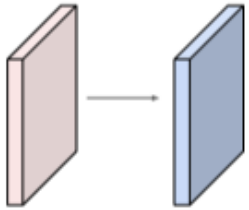
Examples time:



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

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

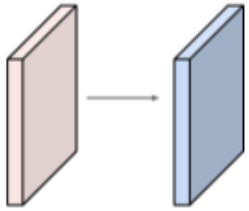
Examples time:



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

Number of parameters in this layer?

Examples time:



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

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- 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 zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- 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.

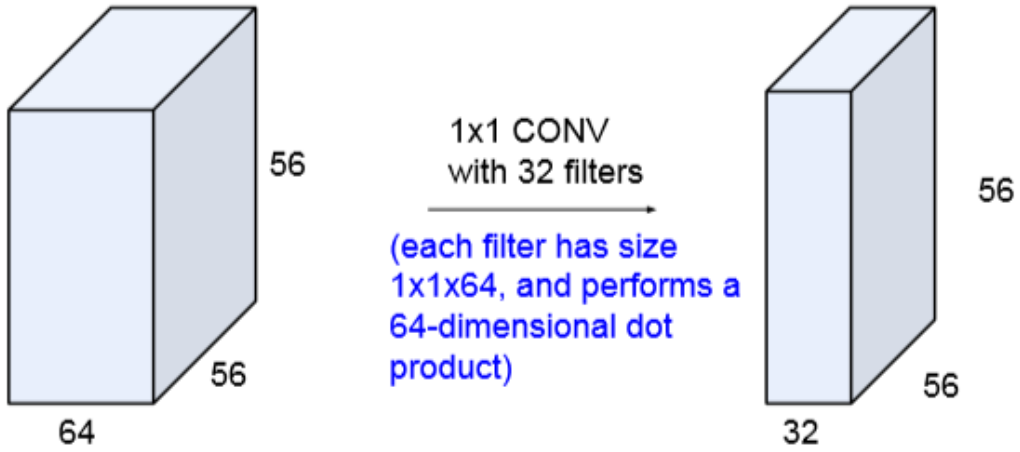
Summary. To summarize, the Conv Layer:

- 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 zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- 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 = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

(btw, 1x1 convolution layers make perfect sense)



Example: CONV layer in Torch

Summary. To summarize, the Conv Layer:

- 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 zero padding P.

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

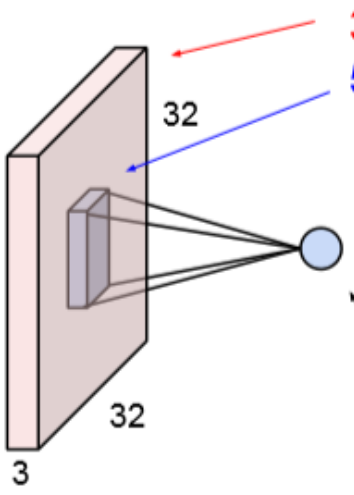
- `nInputPlane` : The number of expected input planes in the image given into `forward()`.
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1`.
- `dH` : The step of the convolution in the height dimension. Default is `1`.
- `padW` : The additional zeros added per width to the input planes. Default is `0`, a good number is `(kW-1)/2`.
- `padH` : The additional zeros added per height to the input planes. Default is `padW`, a good number is `(kH-1)/2`.

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width`, the output image size will be `nOutputPlane x oheight x owidth` where

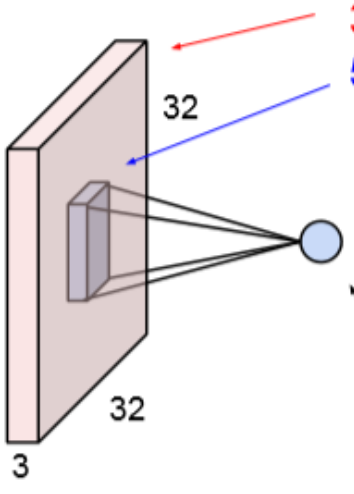
```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

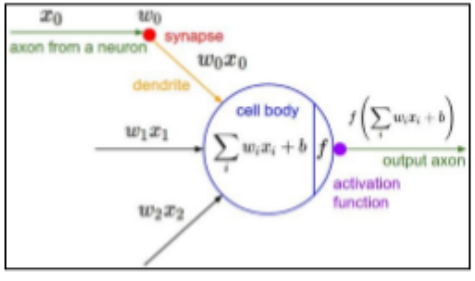
The brain/neuron view of CONV Layer



1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer

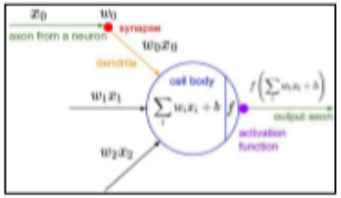


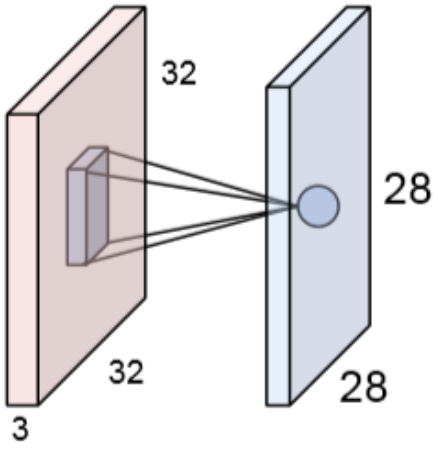


It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer



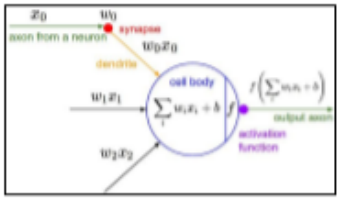


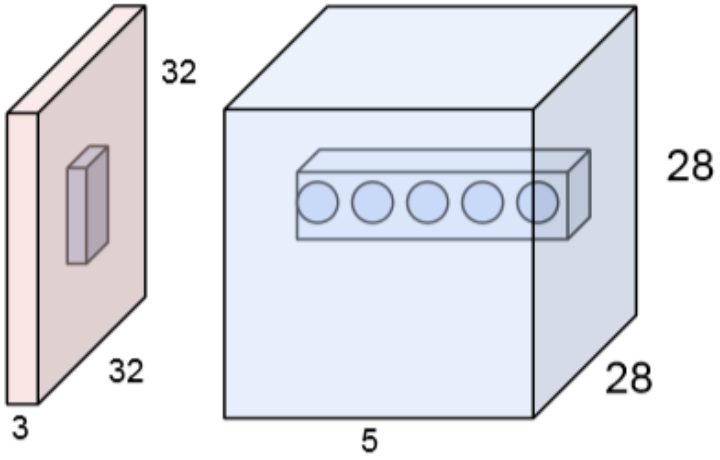
An activation map is a 28x28 sheet of neuron outputs:

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

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

The brain/neuron view of CONV Layer

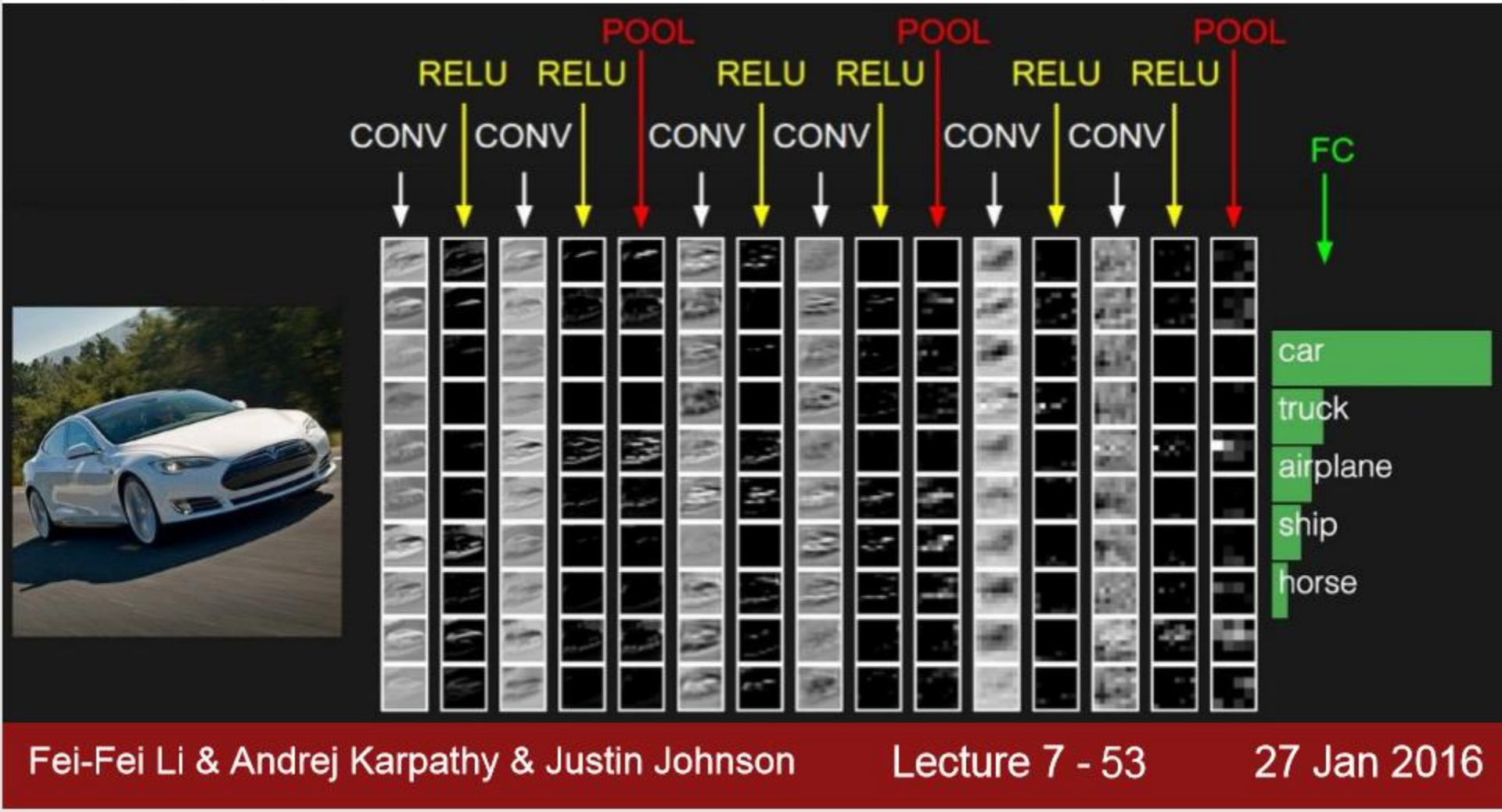




E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

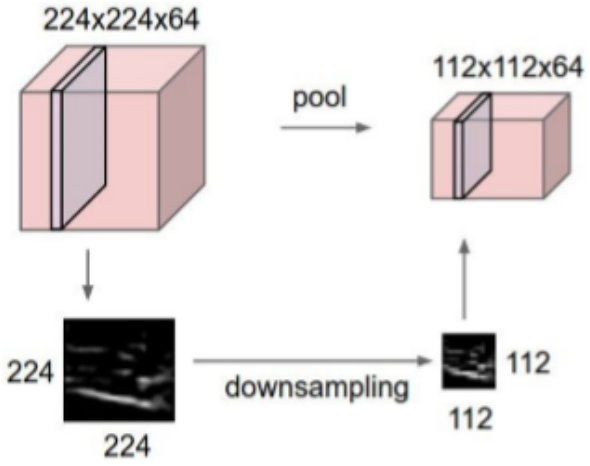
There will be 5 different neurons all looking at the same region in the input volume

two more layers to go: POOL/FC

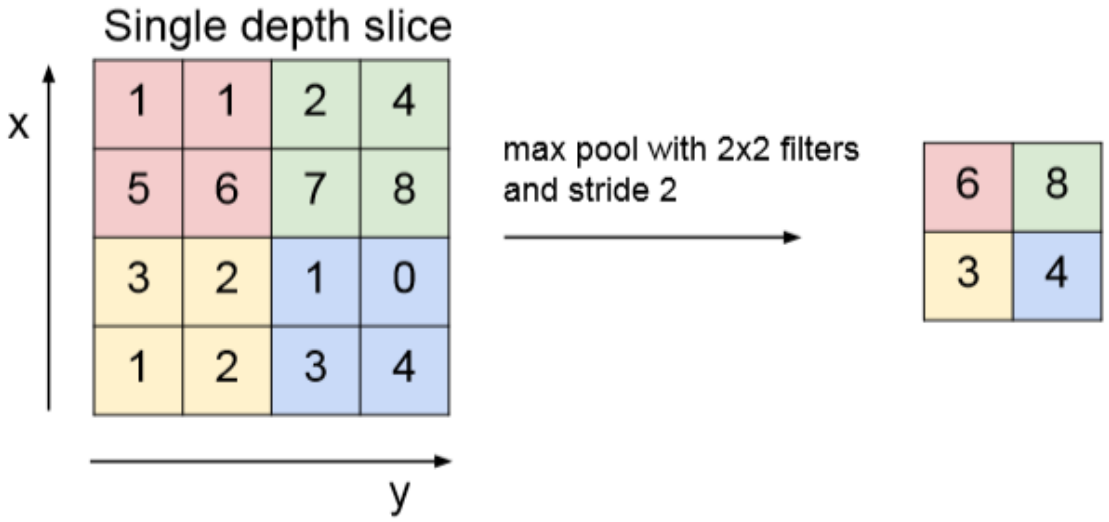


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

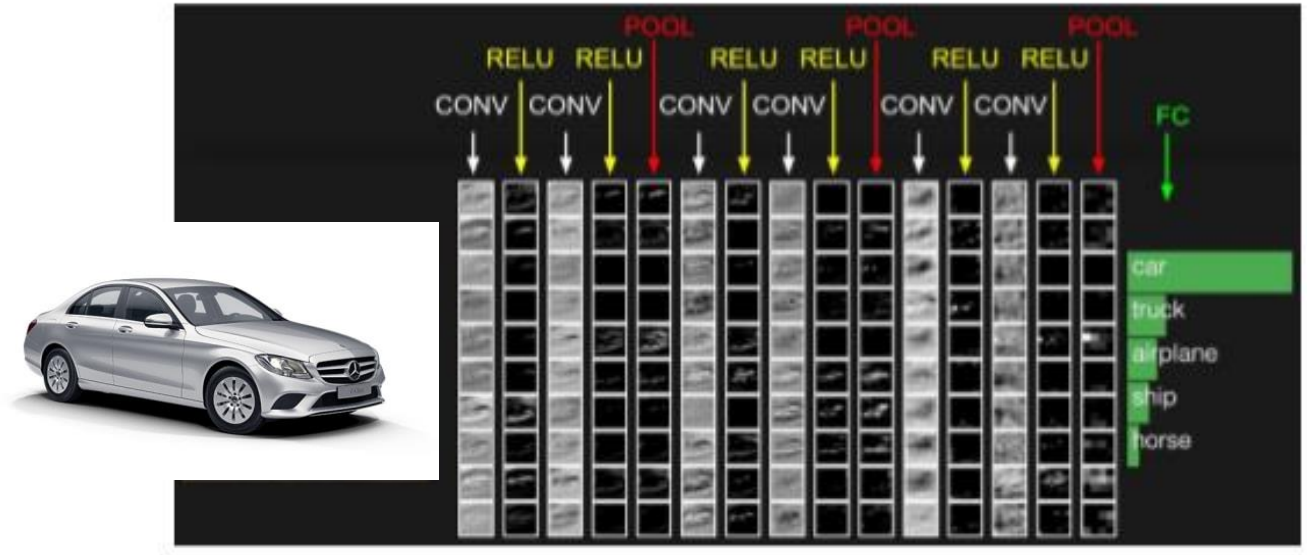
Common settings:

F = 2, S = 2
F = 3, S = 2

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

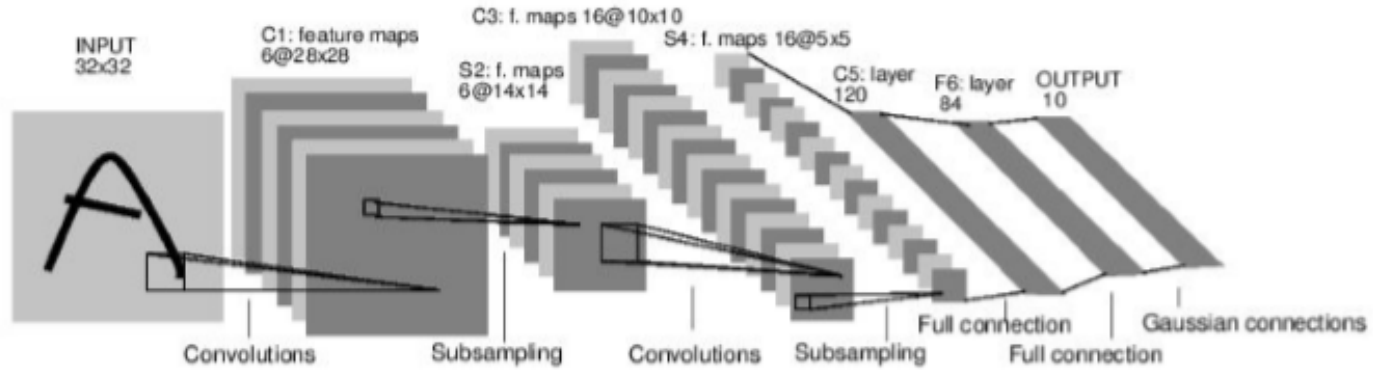


[ConvNetJS demo: training on CIFAR-10]

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Case Study: LeNet-5

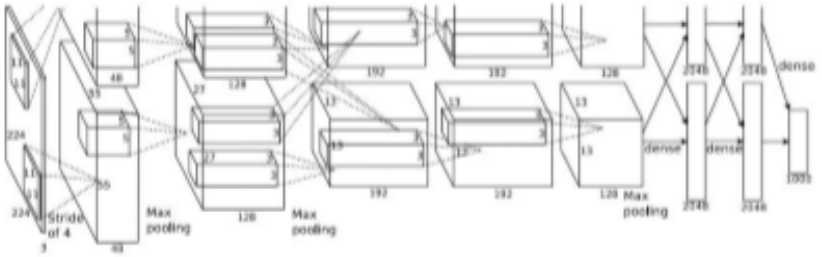
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

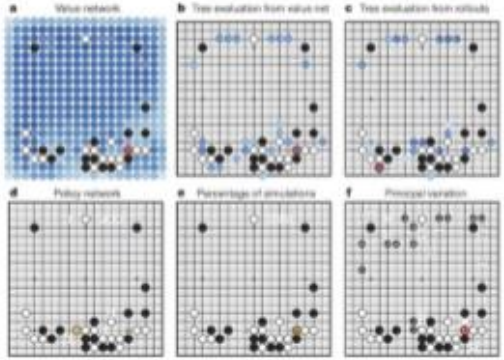
[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study Bonus: DeepMind's AlphaGo

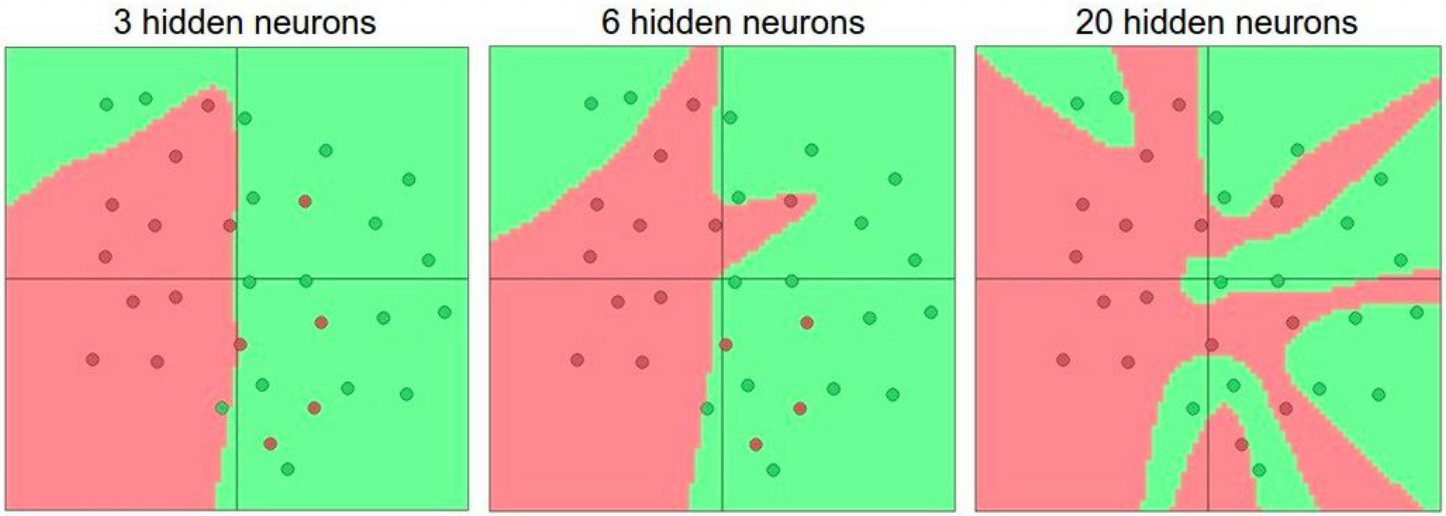


The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

Activation functions

Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function $W_1 W_2 x = Wx$



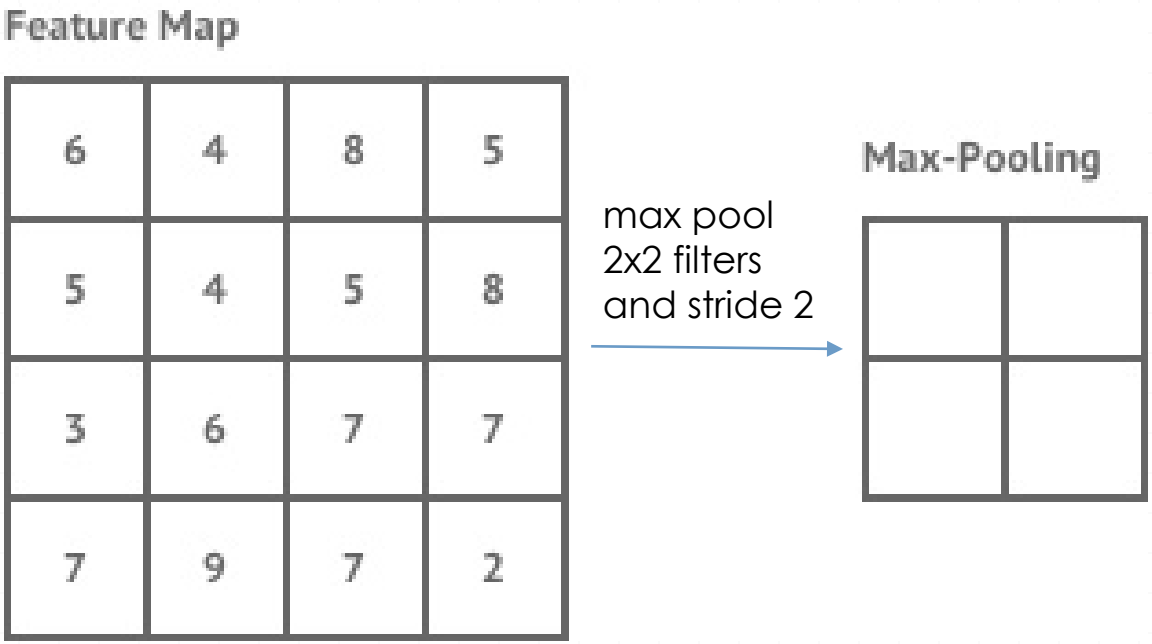
http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

More layers and neurons can approximate more complex functions

Full list: https://en.wikipedia.org/wiki/Activation_function

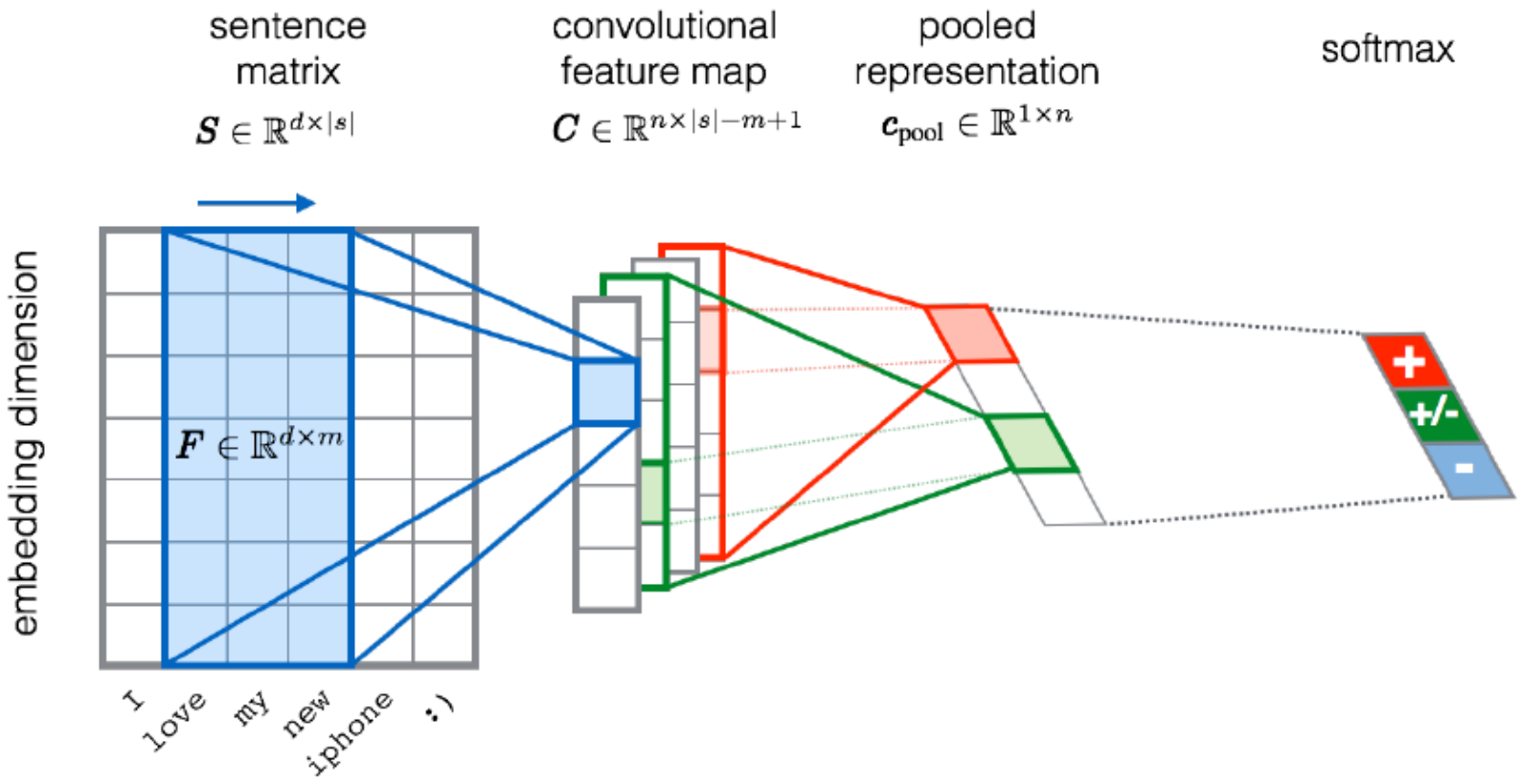
Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and **group them afterwards**



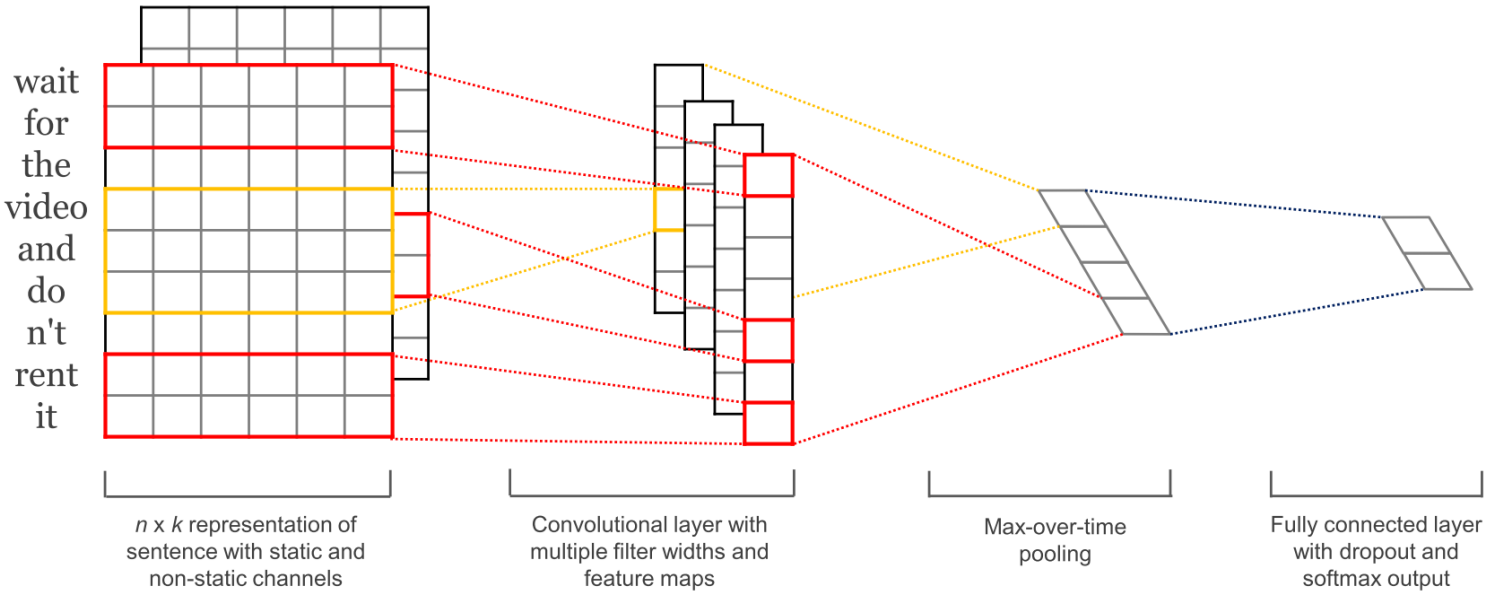
https://shafeentejani.github.io/assets/images/pooling.gif

CNN for text classification



Severyn, Aliaksei, and Alessandro Moschitti. "UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification." *SemEval@ NAACL-HLT*. 2015.

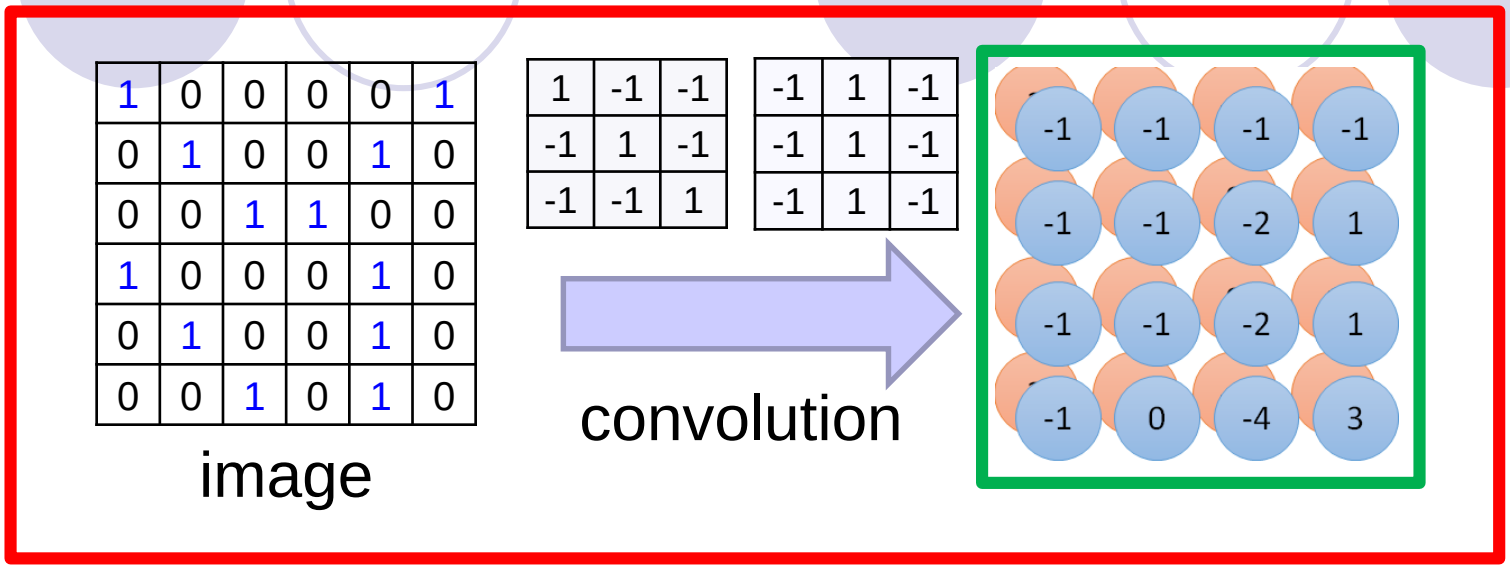
CNN with multiple filters



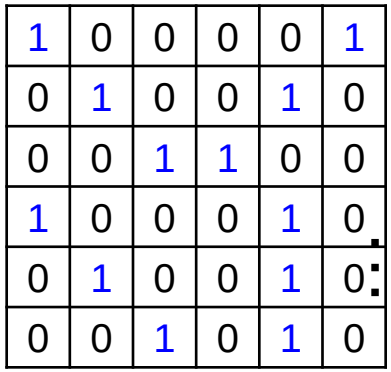
Kim, Y. "Convolutional Neural Networks for Sentence Classification", EMNLP (2014)

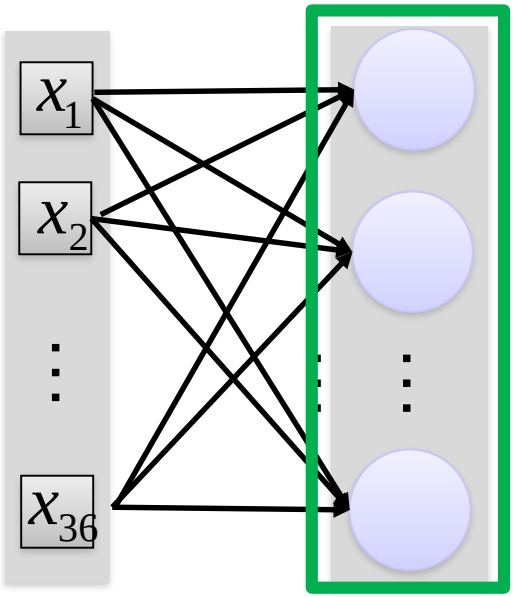
sliding over 3, 4 or 5 words at a time

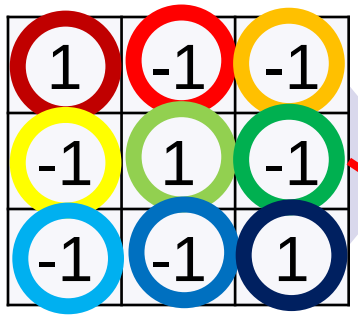
Convolution v.s. Fully Connected



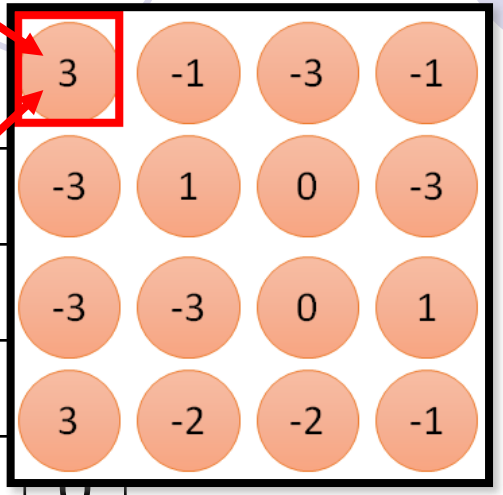
Fully-connected







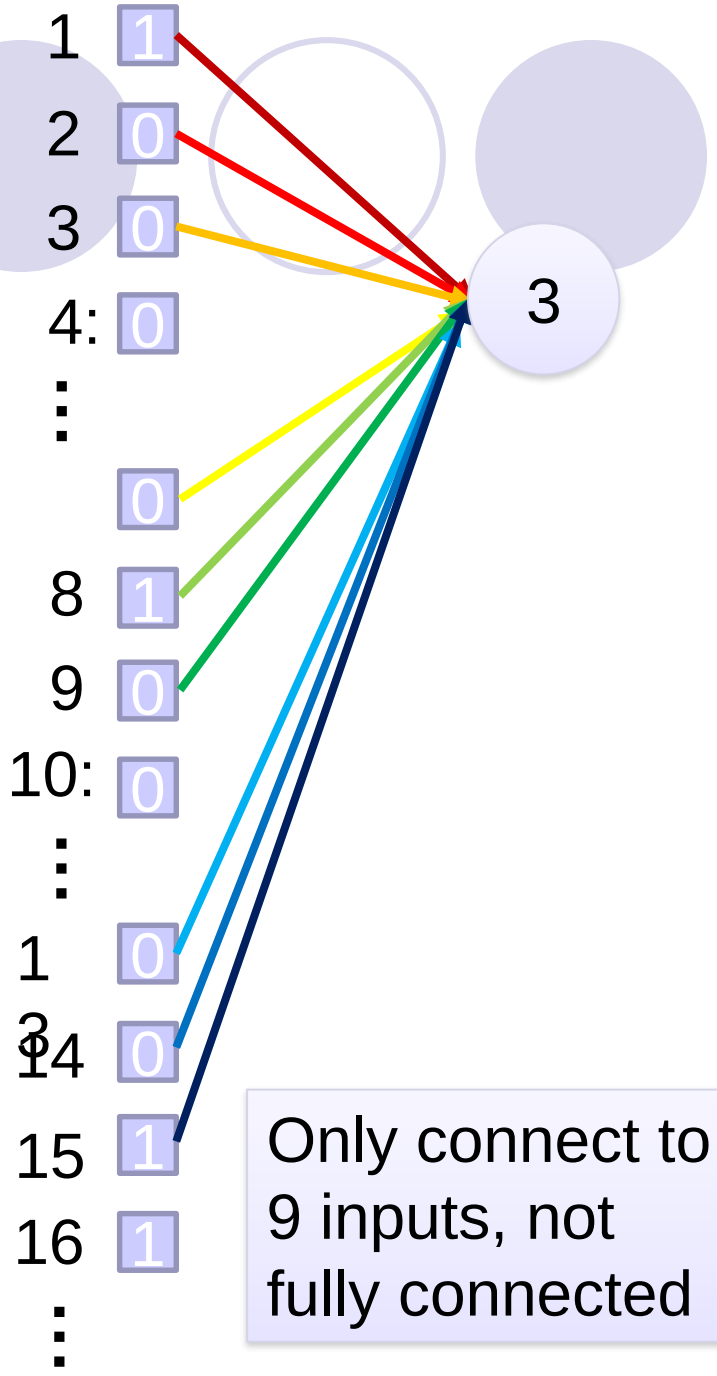
Filter 1



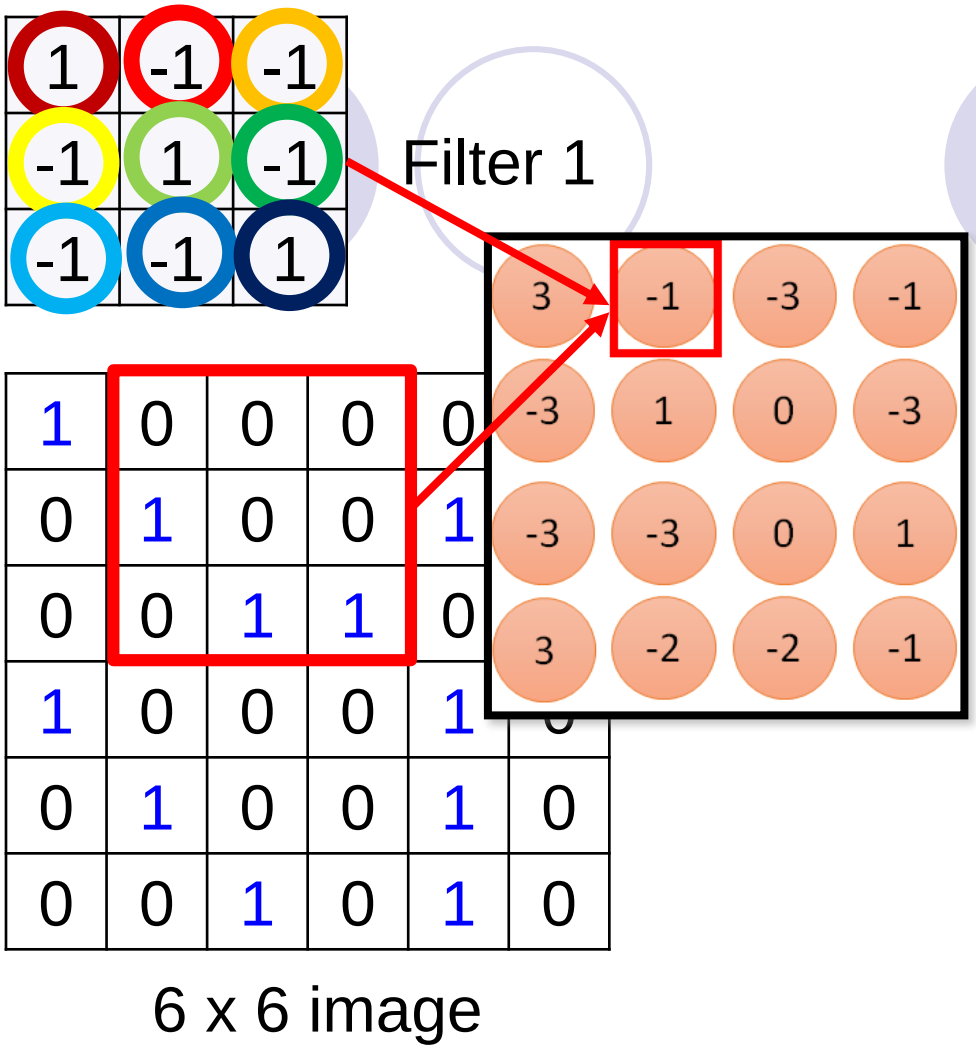
1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

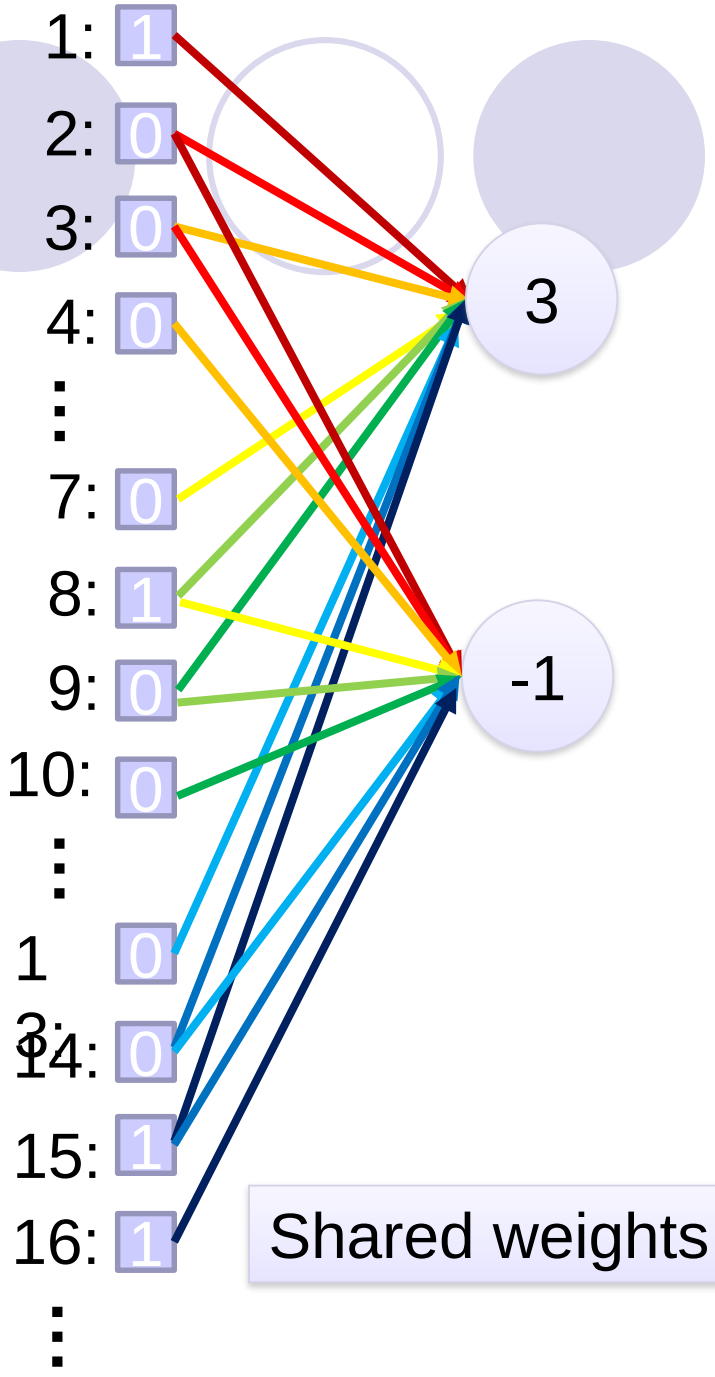
6 x 6 image

fewer parameters!



Only connect to 9 inputs, not fully connected



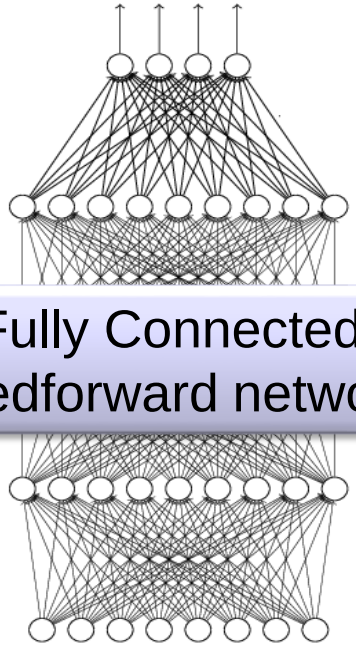


Fewer parameters

Even fewer parameters

The whole CNN

cat dog ……



Fully Connected Feedforward network

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flattened

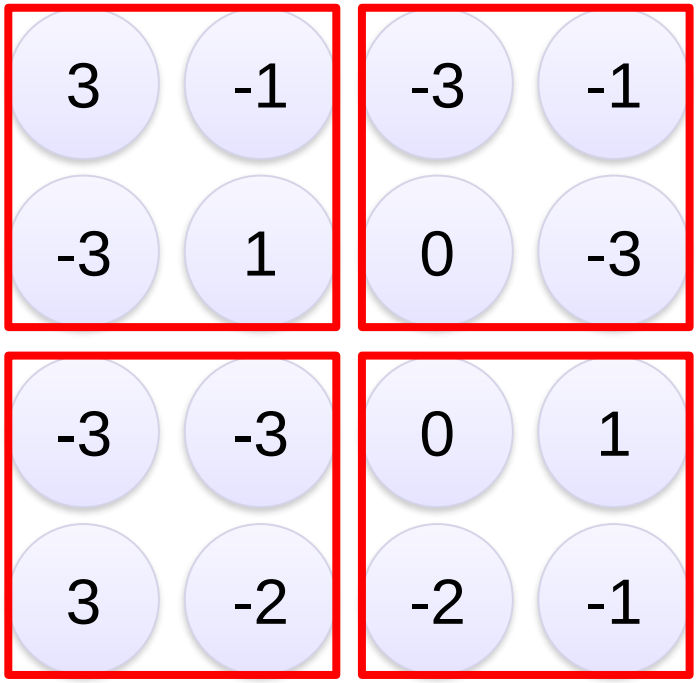
Max Pooling

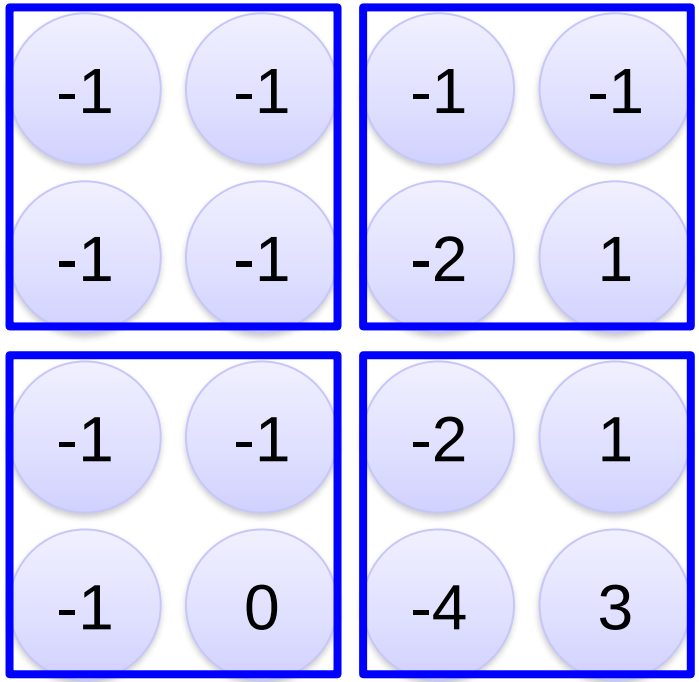
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

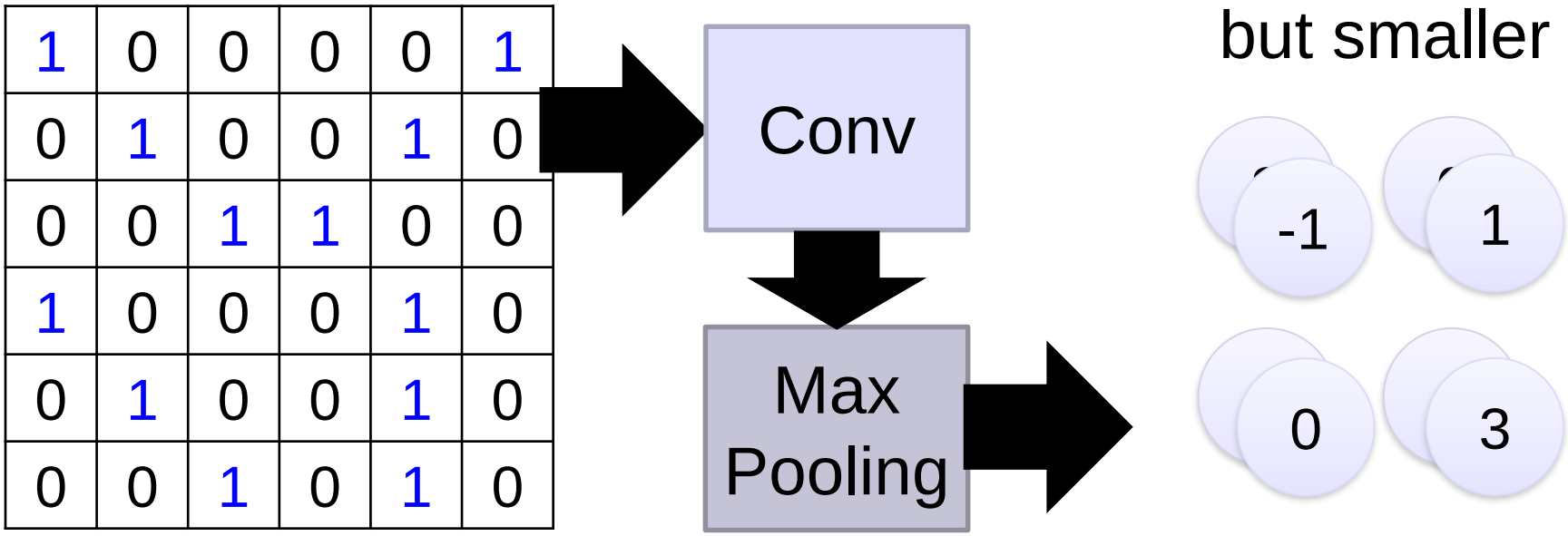
We can subsample the pixels to make image smaller

fewer parameters to characterize the image

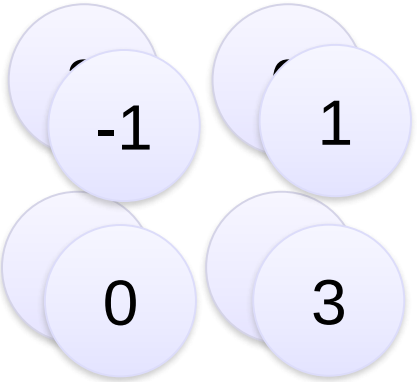
A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

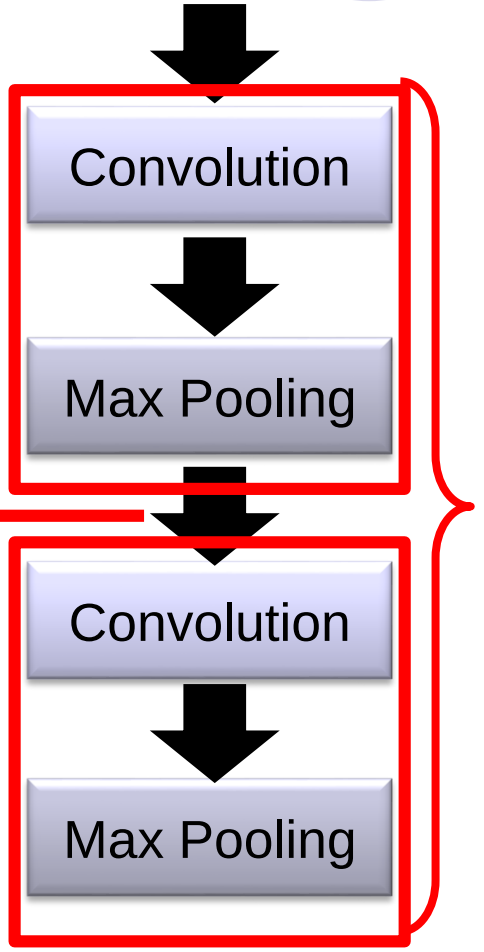
Max Pooling


1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
Conv
Max Pooling
New image but smaller
-1
1
0
3
6 x 6 image
2 x 2 image
Each filter is a channel

The whole CNN



A new image



Can repeat many times

Smaller than the original image

The number of channels is the number of filters

The whole CNN

cat dog ……

Convolution

Max Pooling

A new image

Convolution

Max Pooling

A new image

Flattened

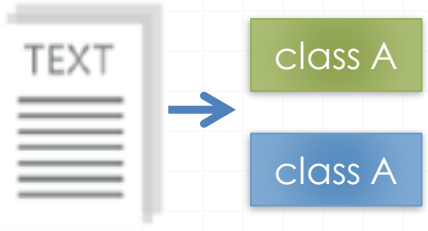
Fully Connected Feedforward network

Types of Learning

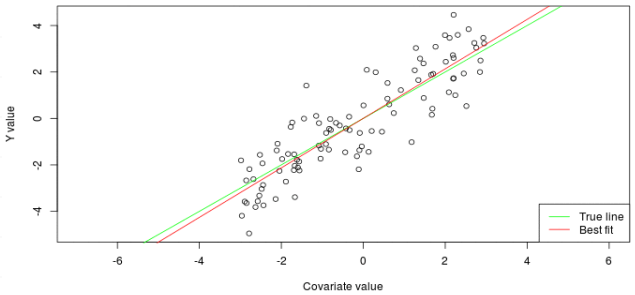
Supervised: Learning with a **labeled training** set
Example: email *classification* with already labeled emails

Unsupervised: Discover **patterns** in **unlabeled** data
Example: *cluster* similar documents based on text

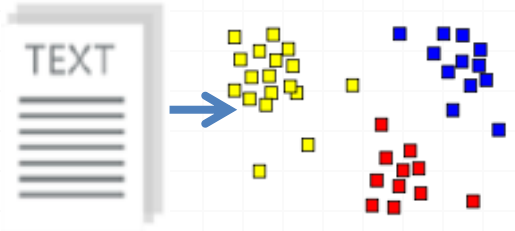
Reinforcement learning: learn to **act** based on **feedback/reward**
Example: learn to play Go, reward: *win or lose*



Classification



Regression



Clustering

Anomaly Detection
Sequence labeling
…

http://mbjoseph.github.io/2013/11/27/measure.html

Deep Learning Taxonomy

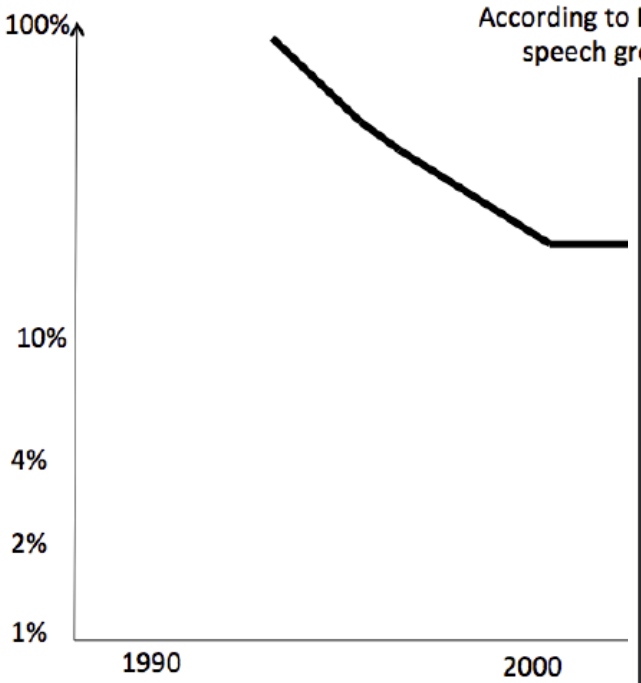
Supervised:

- Convolutional NN (LeCun)
- Recurrent Neural nets (Schmidhuber)

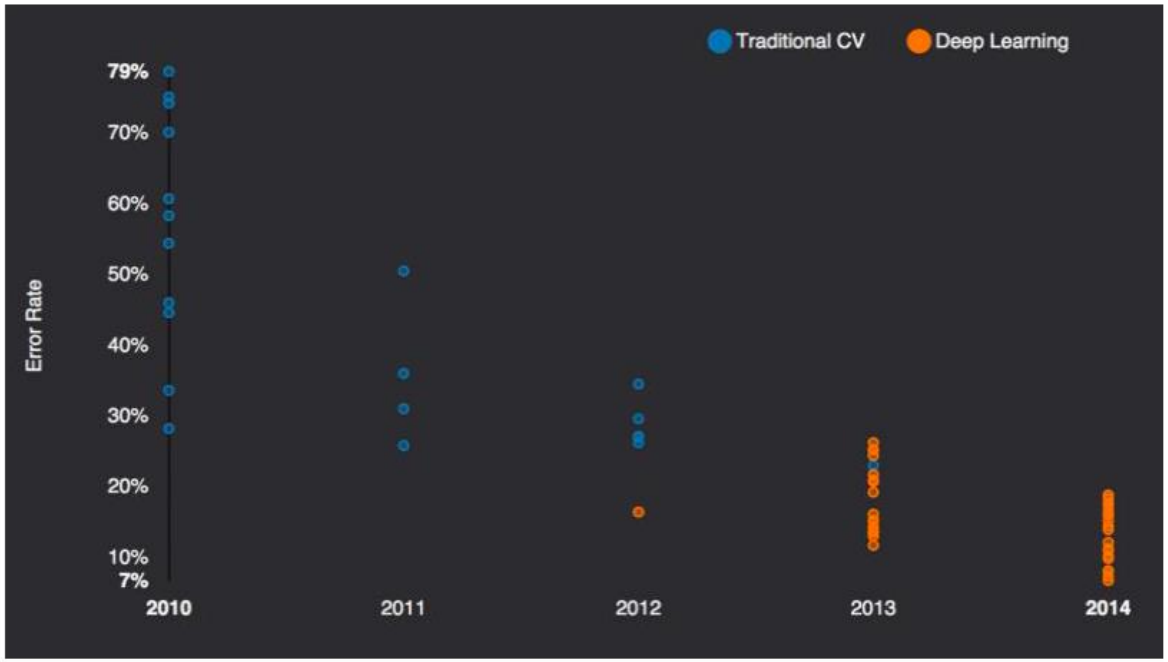
Unsupervised

- Deep Belief Nets / Stacked RBMs (Hinton)
- Stacked denoising autoencoders (Bengio)
- Sparse AutoEncoders (LeCun, A. Ng,)

State of the art in …



Deep Learning in Speech Recognition



ImageNet: The "computer vision World Cup"

Several big improvements in recent years in NLP

- ✓ Machine Translation
- ✓ Sentiment Analysis
- ✓ Dialogue Agents
- ✓ Question Answering
- ✓ Text Classification …

Leverage different levels of representation

- words & characters
- syntax & semantics

Disadvantages

Disadvantages

- From a memory and capacity standpoint the CNN is not much bigger than a regular two layer network.
- At runtime the convolution operations are computationally expensive and take up about 67% of the time.
- CNN's are about 3X slower than their fully connected equivalents (size-wise).

Disadvantages

- **Convolution operation**
  - 4 nested loops (2 loops on input image & 2 loops on kernel)
- **Small kernel size**
  - make the inner loops very inefficient as they frequently JMP.
- **Cash unfriendly memory access**
  - Back-propagation require both row-wise and column-wise access to the input and kernel image.
  - 2-D Images represented in a row-wise-serialized order.
  - Column-wise access to data can result in a high rate of cash misses in memory subsystem.

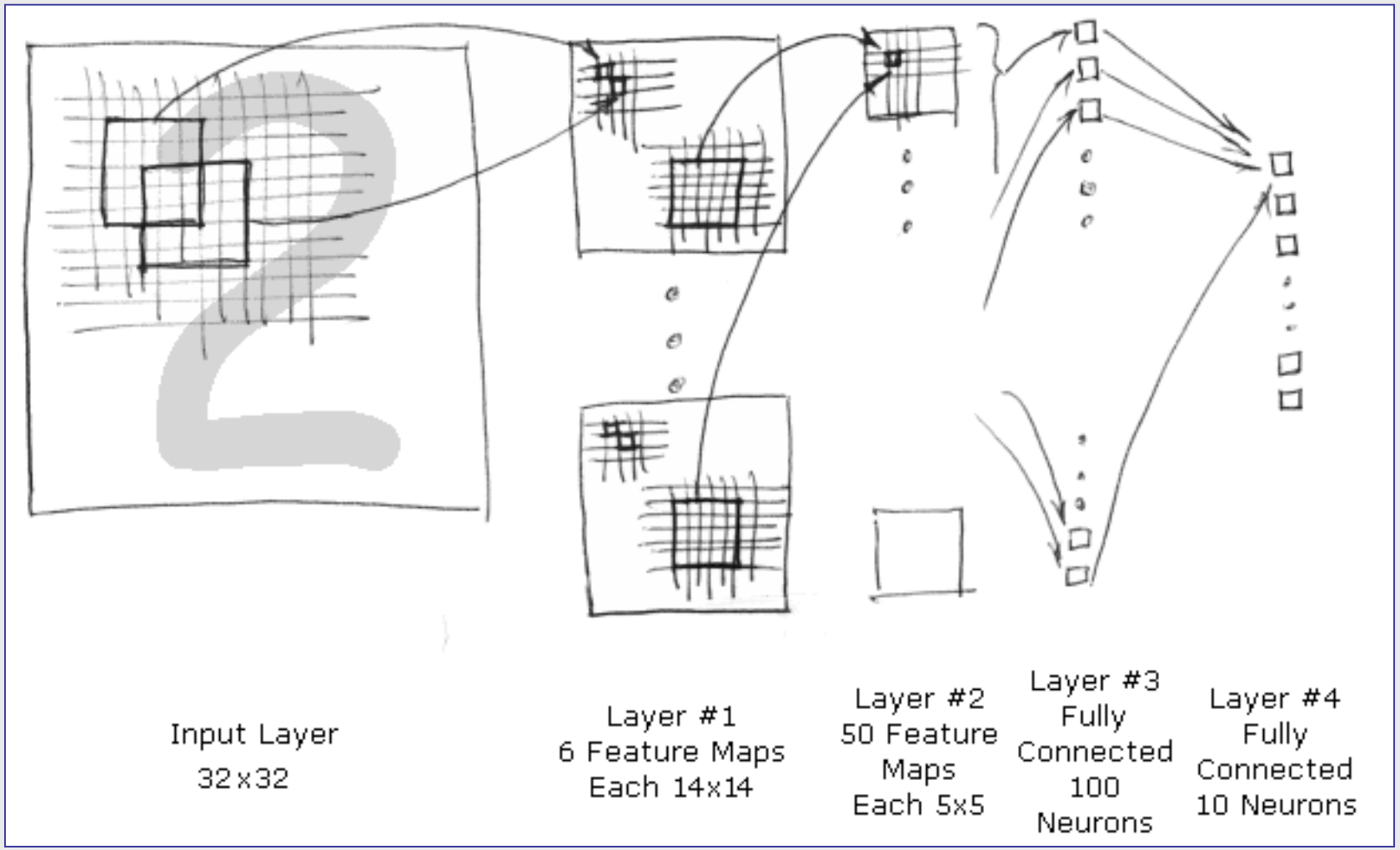
Application

Application

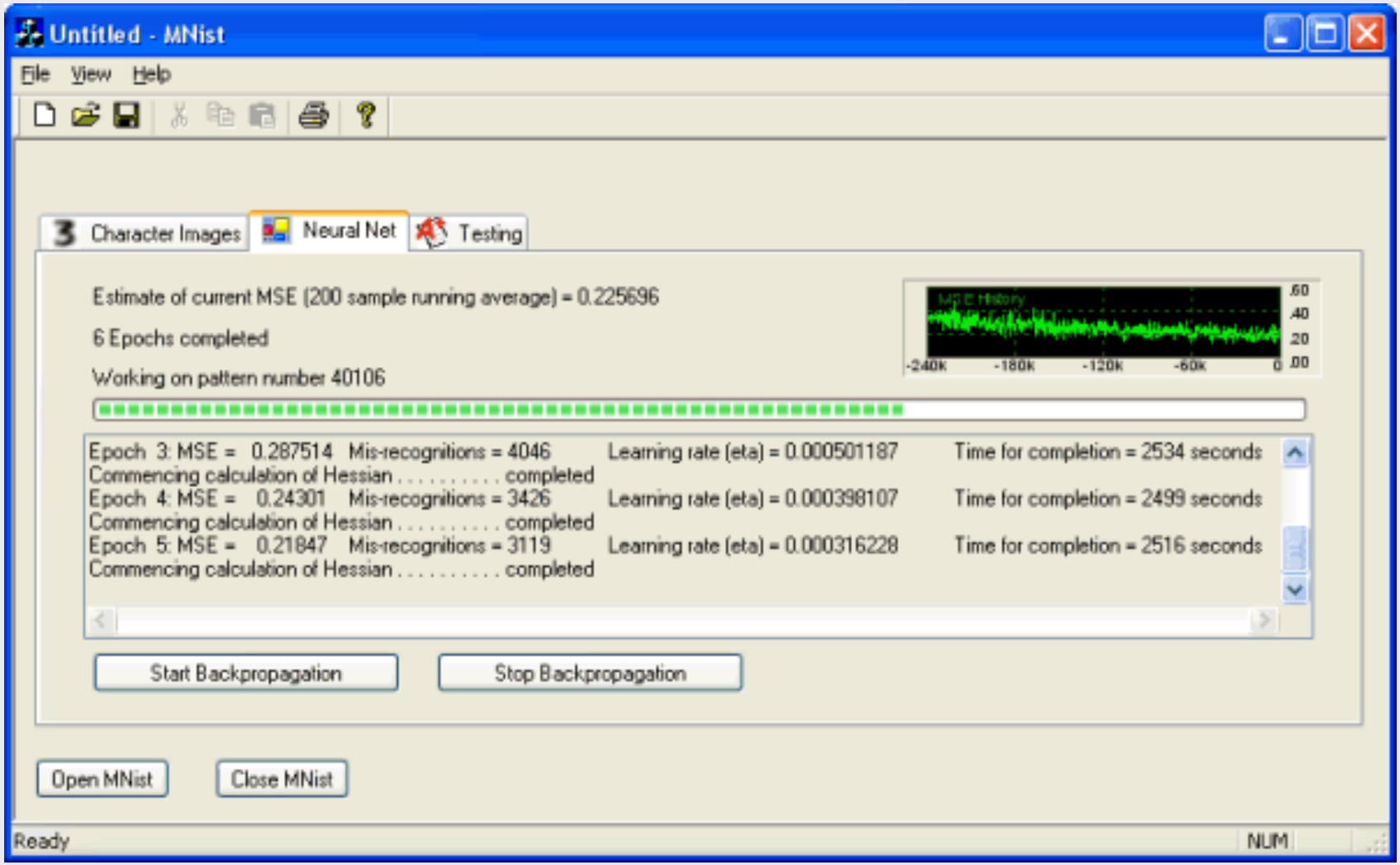
Mike O'Neill

- A Convolutional neural network achieves 99.26% accuracy on a modified NIST database of hand-written digits.
- MNIST database : Consist of 60,000 hand written digits uniformly distributed over 0-9.

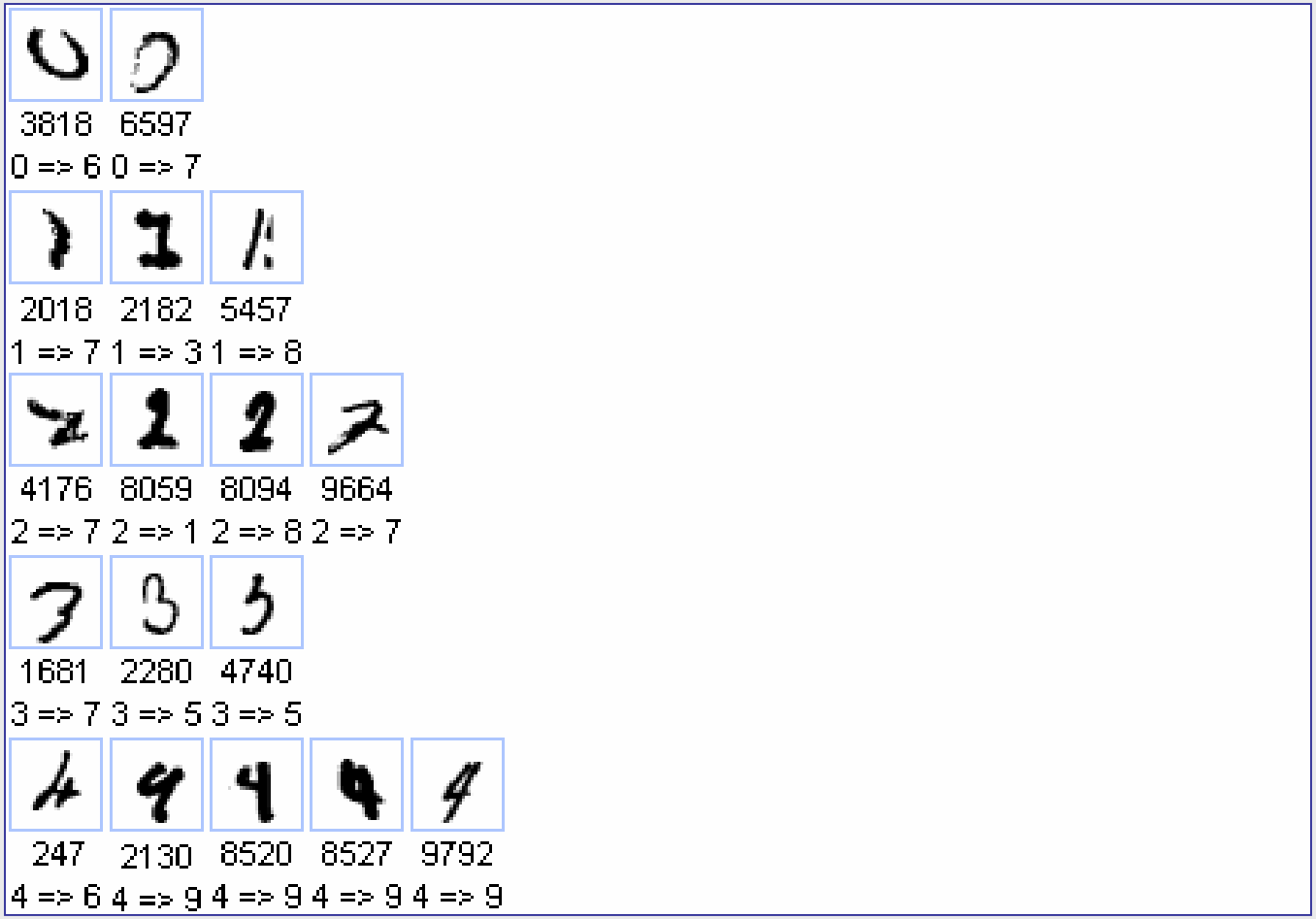
Application



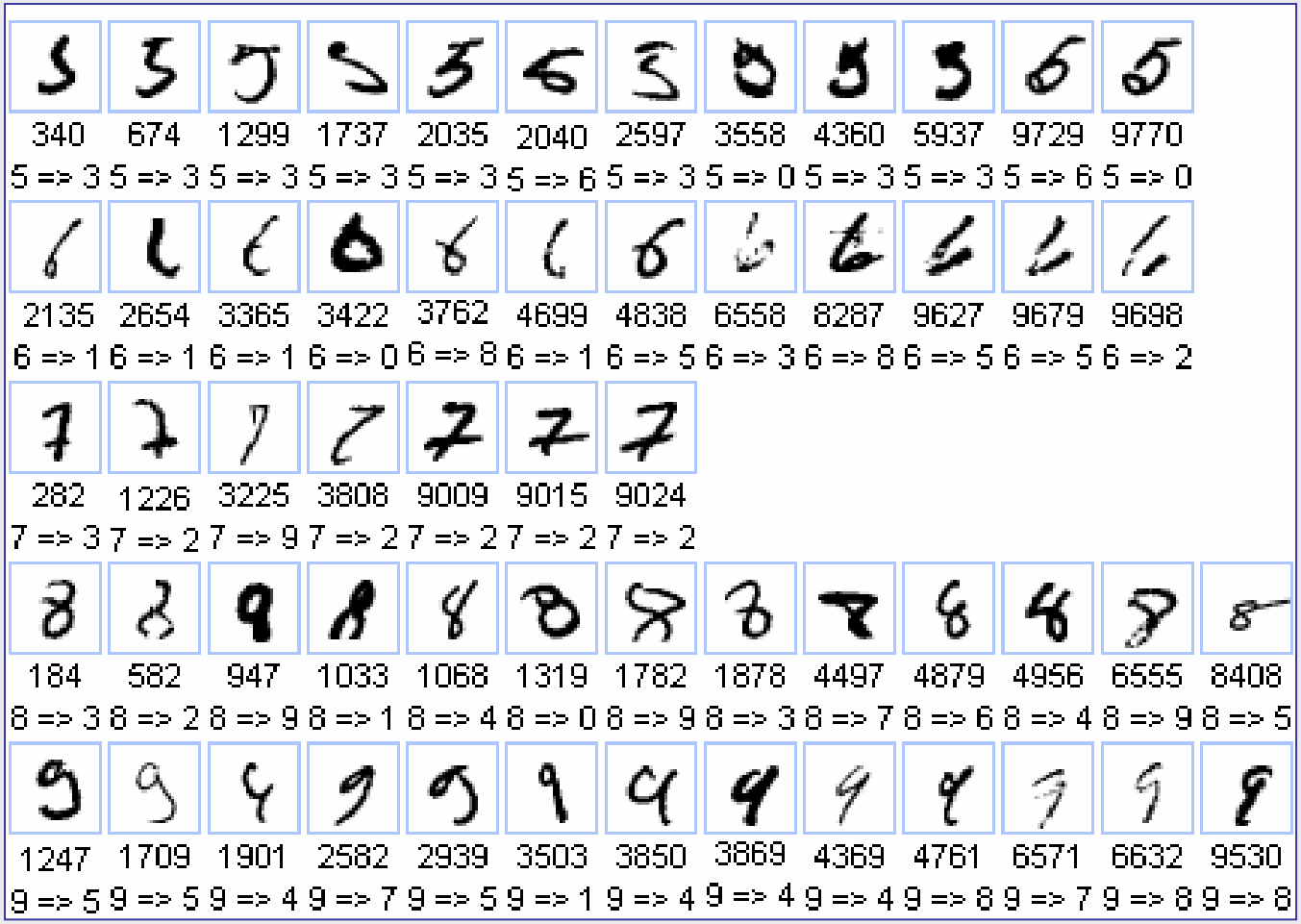
Application



Application



Application



References

[1].Y. LeCun and Y. Bengio.**"Convolutional networks for images, speech, and time-series."** In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[2].Fabien Lauer, ChingY. Suen, Gérard Bloch,**"A trainable feature extractor for handwritten digit recognition"**,Elsevier, october 2006.

[3].Patrice Y. Simard, Dave Steinkraus, John Platt, **"Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis,"** International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society, Los Alamitos, pp. 958-962, 2003.

Questions

Pairs of blurred license plates and their respective reconstructions computed by CNN.

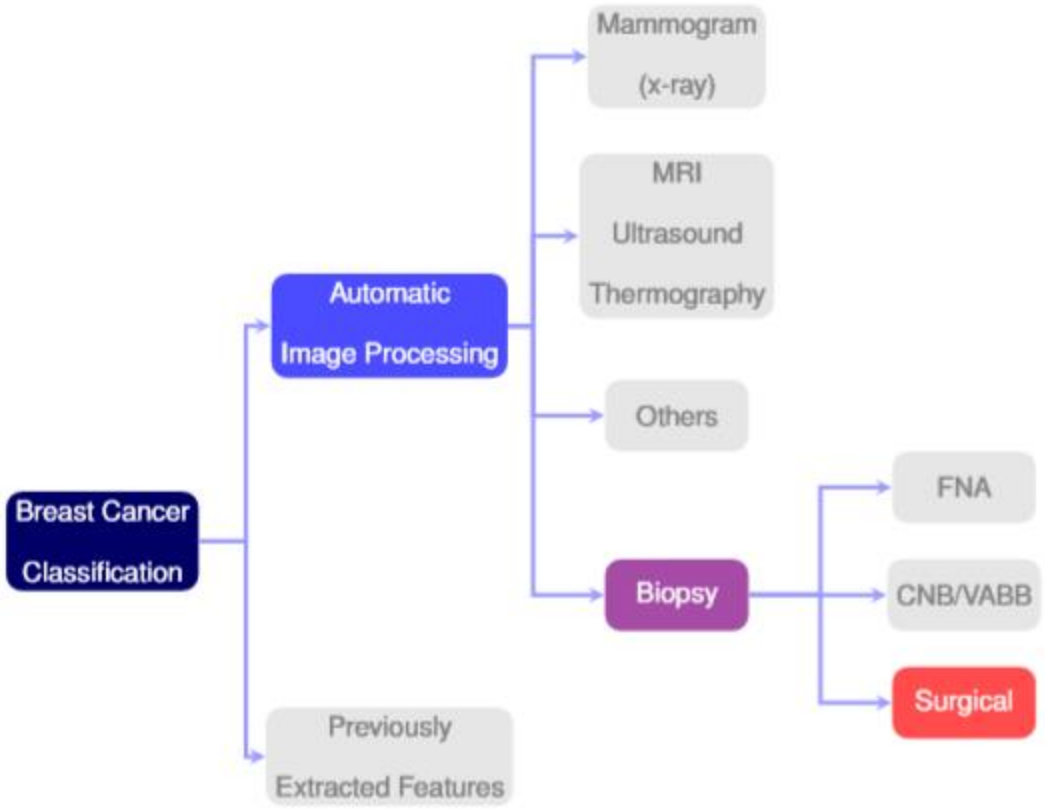


Figure 3.1: Categories of related works on breast cancer classification.
Source: The author (2015).