ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Deep Learning

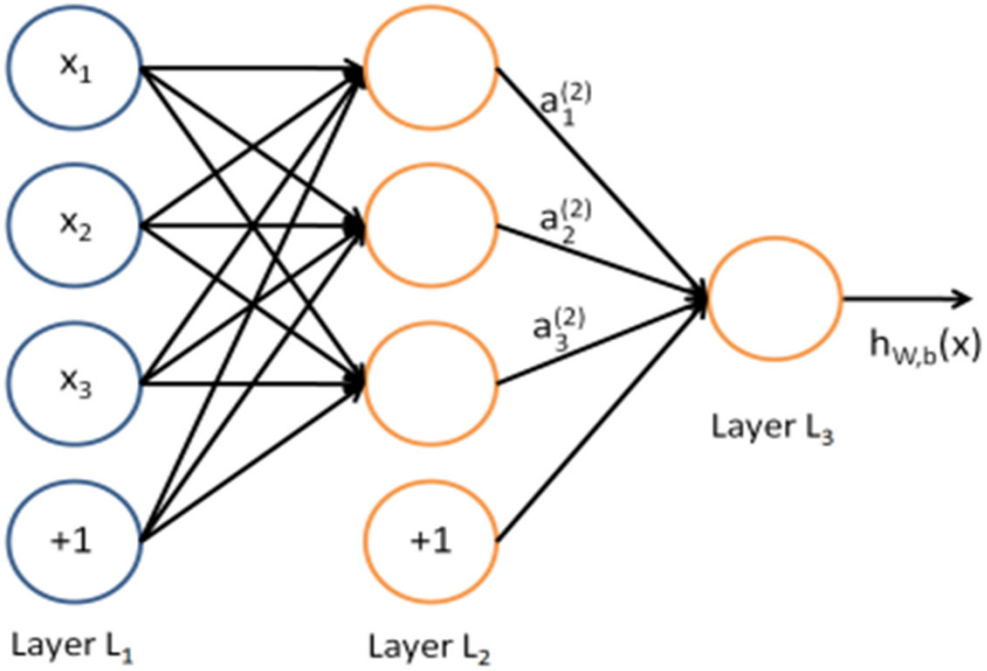
LECTURE : CONVOLUTIONAL NEURAL NETWORKS (CONVNETS)

Petia Georgieva
(petia@ua.pt)

Outline

- **Deep Neural Networks**
- **Convolutional Neural Networks (CNN) - basic building blocks**
- **Classical CNN – LeNet-5, AlexNet, VGG**
- **Visualization of what convolutional layers learn**

Shallow (3 layers) Neural Network



$$(W,b) = \left(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\right)$$

Standard **NN notation**:
W –matrix of network (model) weights/parameters
b- bias/intercept (the weight of the input +1)

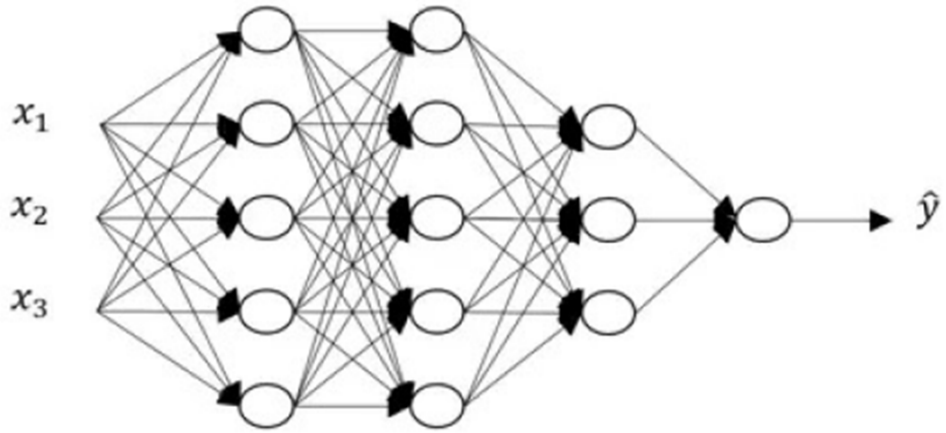
Why deep learning ?

Hardware get smaller.
Sensors get cheaper, widely available IoT devices with high sample-rate.
Data sources: sound, vibration, image, electrical signals, accelerometer, temperature, pressure, LIDAR, etc.

Big Data: Exponential growth of data, (IoT, medical records, biology, engineering, etc.)

How to deals with **unstructured data** (image, voice, text, EEG, ECG, etc.) => needs for feature extraction (data mining).

Deep Neural Networks: first extract (automatically) the features, then solve ML tasks (classification, regression)



Conventional ML vs. Deep Learning

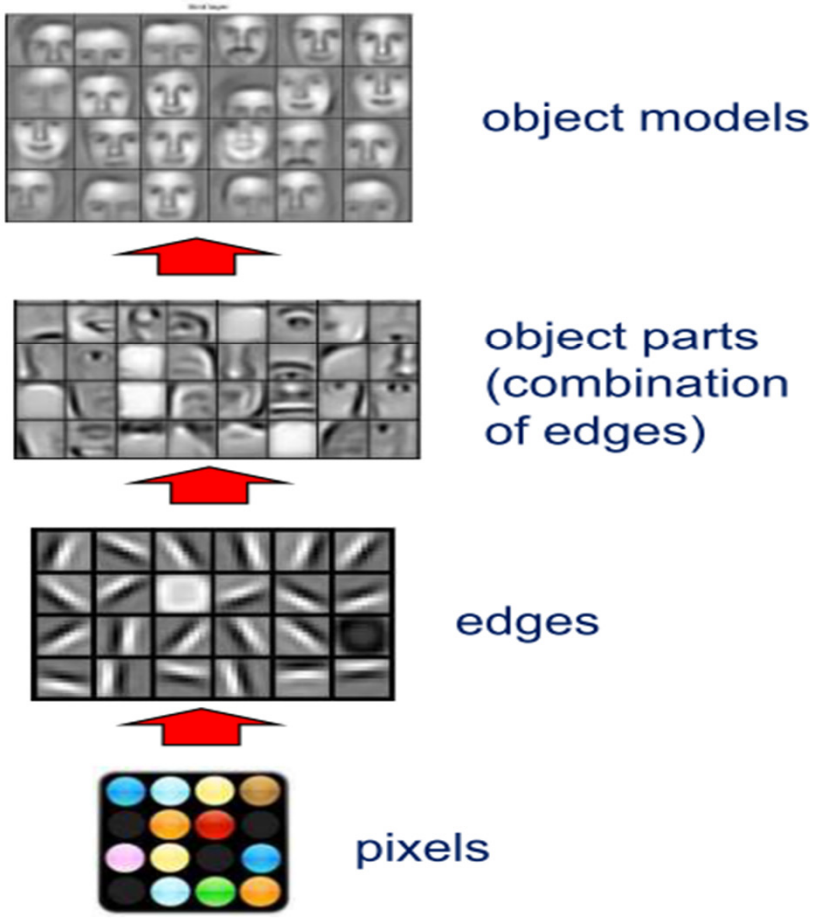
Conventional Machine Learning (ML)

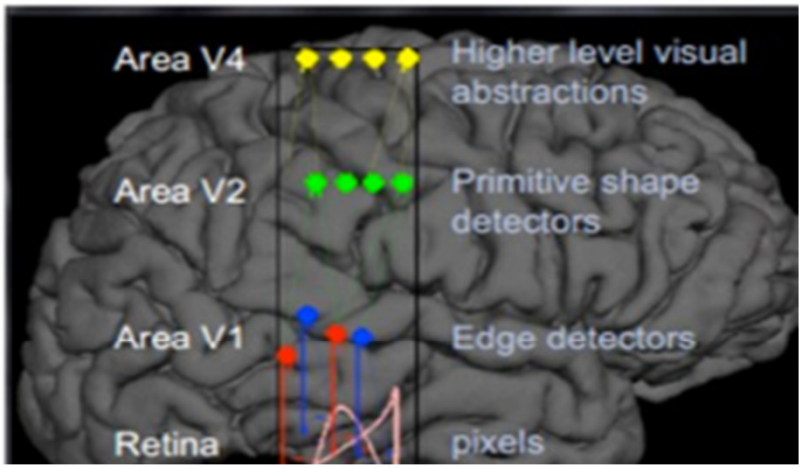
- Needs feature engineering – hard, time-consuming task, requires expert knowledge.
- Don't work well with raw data (e.g. pixels/voxels of images).
- Better than DL if good features are found.

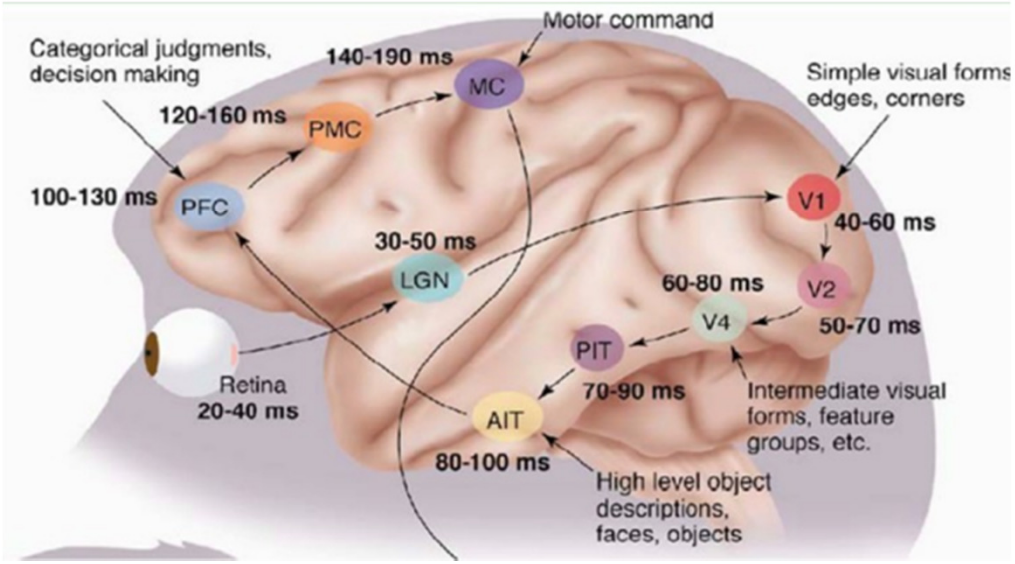
Deep Learning

- Representation-learning methods with multiple levels of representation.
- Layers that extract automatically features from raw data (latent structures not seen/difficult to be designed by humans)
- Needs large amounts of labelled training data
- Needs massive computational resources (e.g. GPUs, parallel solvers)

Representation Learning inspired by visual neuroscience







By Yann LeCun, Yoshua Bengio & Geoffrey Hinton
Nature 521, 436–444, 2015, doi:10.1038/nature14539

Deep Neural Networks (DNN)

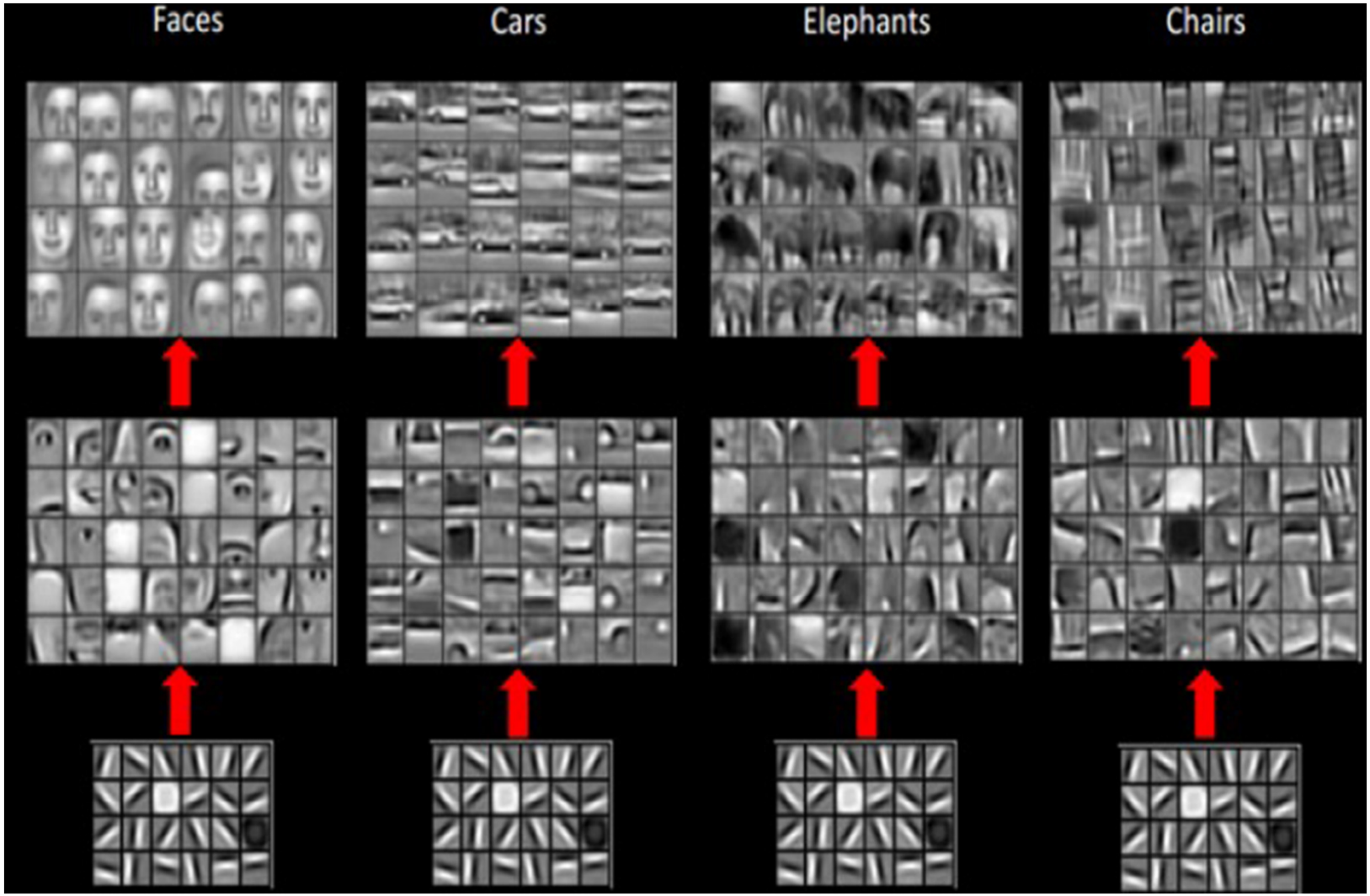
DNNs have a large number of hidden layers. The first hidden layer finds simple functions like identifying the edges in the image. As we go deeper into the network these simple functions combine together to form more complex functions like identifying the face.

Common Examples:

Face Recognition:
Image => Edges => Face parts => Faces => Desired face

Audio recognition:
Audio => Low level audio wave features => Phonemes => Words => Sentences

Representation Learning



By Yann LeCun, Yoshua Bengio & Geoffrey Hinton
Nature 521, 436–444, 2015, doi:10.1038/nature14539

Deep Learning – major applications

Image, Video, Speech, Text processing (large scale, big data) :

- **Computer Vision (object recognition)**
- **Speech Recognition**
- **Natural Language Processing (topic classification, sentiment analysis, language translation, etc.)**
- **Bioinformatics (genomics, drug discovery)**
- **Brain imaging (brain computer interfaces)**

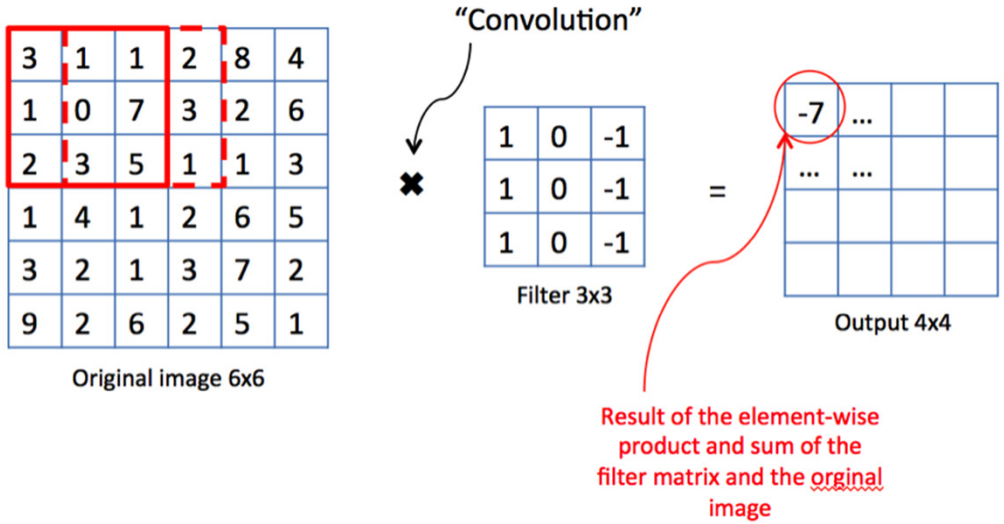
Why Convolution Learning ?

Deep learning on large images

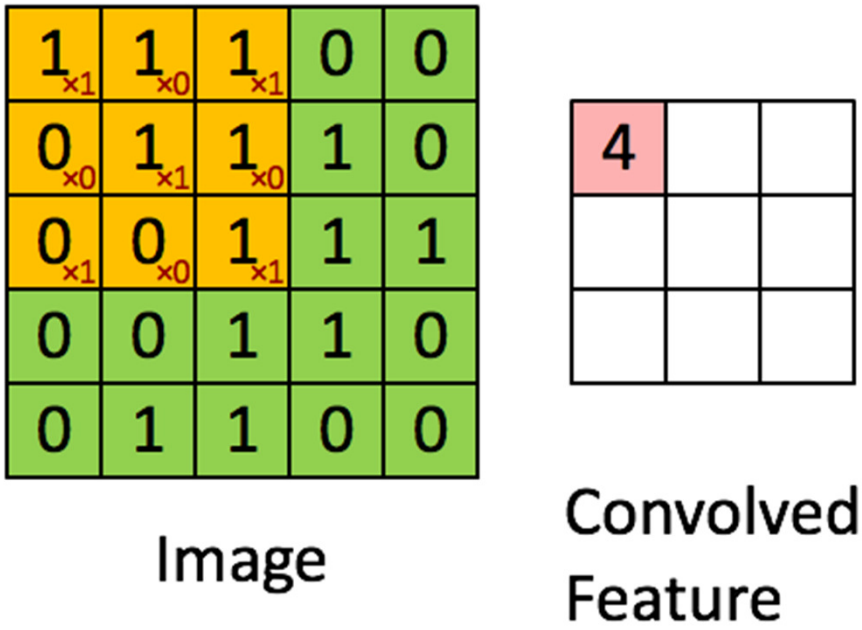
If the image has 1000x1000x3 (RGB) pixels=3 million features (inputs)
If the first hidden layer has 1000 nodes =>
The parameter matrix between the input and the hidden layer has (1000x3million) 3billion parameters.

1st problem: Difficult to get enough data to prevent model overfitting
2nd problem: Computational (memory) requirements to train such networks are not feasible.

Solution: implement convolution operation

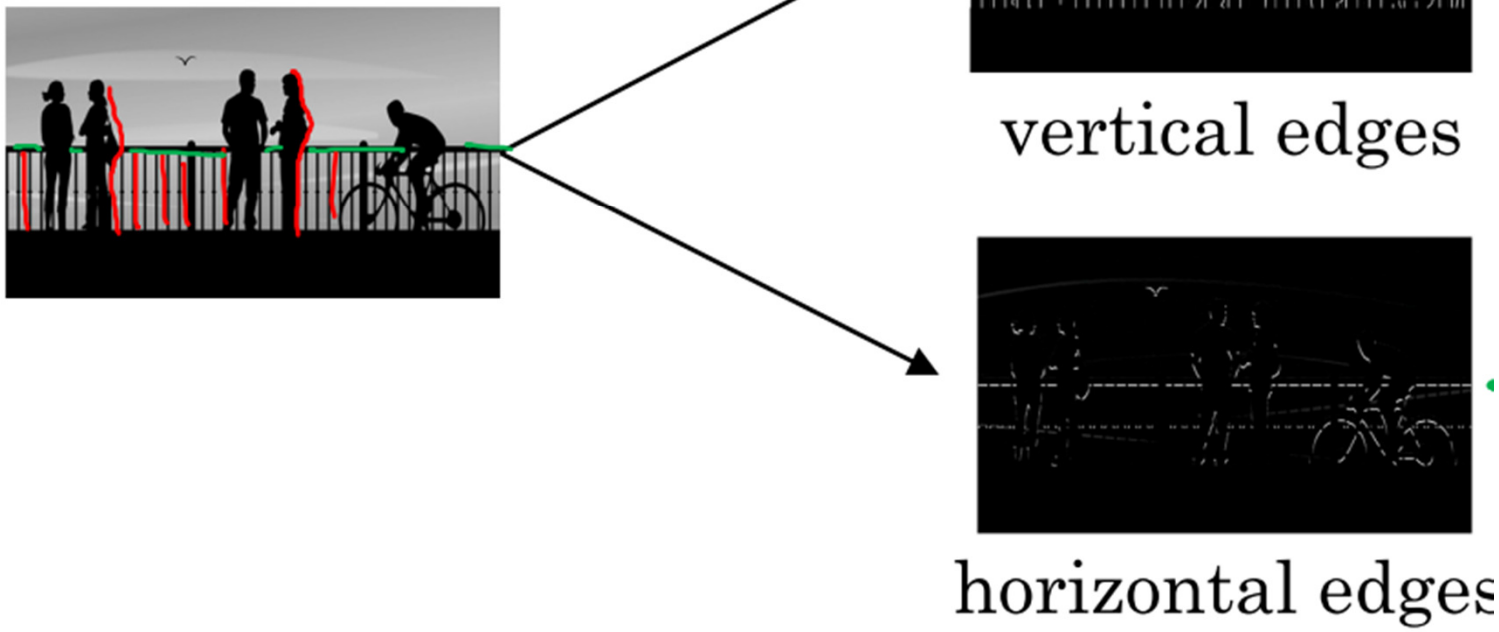


OPERATION CONVOLUTION



Remark: There is some inconsistency between the standard operation convolution from the signal processing view and the way it is defined in Deep learning literature.
However, by convention in CNN this operation is called convolution.

Detect horizontal/vertical edges



VERTICAL EDGES DETECTOR

Ilustrative example:

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

*

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

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*

Detection of bright to dark transition (+30)

Hand-picked convolutional filters(kernels)

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

=> Horizontal edge detector

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$\Rightarrow$ **Sobel filter**

$$\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$$

$\Rightarrow$ **Sharr filter**

CONVOLUTIONAL FILTERS (KERNELS)

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Hand-picking the filter values is difficult.
Why not let the computer to learn them?
Treat the filter numbers as parameters (w), and let the computer learn them automatically.
Other than vertical and horizontal edges, such computer-generated filter can learn information from different angles (e.g. 45°, 70°, 73°) and is more robust than hand-picking values.

By convention the conv filter is a square matrix with odd size (typically 3x3; 5x5; 7x7, also 1x1) .
It is nice to have a central pixel and it facilitates the padding.

PADDING

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

Ex. Take 6 × 6 image, apply 3 × 3 conv filter, get 4 × 4 output matrix, because we shift the filter one row down or one column right and therefore we have 4 × 4 possible positions for the 3 × 3 filter to appear in the 6 × 6 input matrix.

In general: given $n \times n$ input matrix and $f \times f$ filter matrix, the convolution operation will compute $(n-f+1) \times (n-f+1)$ output matrix by applying one pixel up/down left/right rule.

1st problem: Shrink the matrix size as we continue to further apply convolution. The image will get very small if we have many convolution layers.
2nd problem: Pixels on the corner of the image are used only once while the pixels in the centre of the image are used many times. This is uneven, loose of inf.

Solution: Padding.

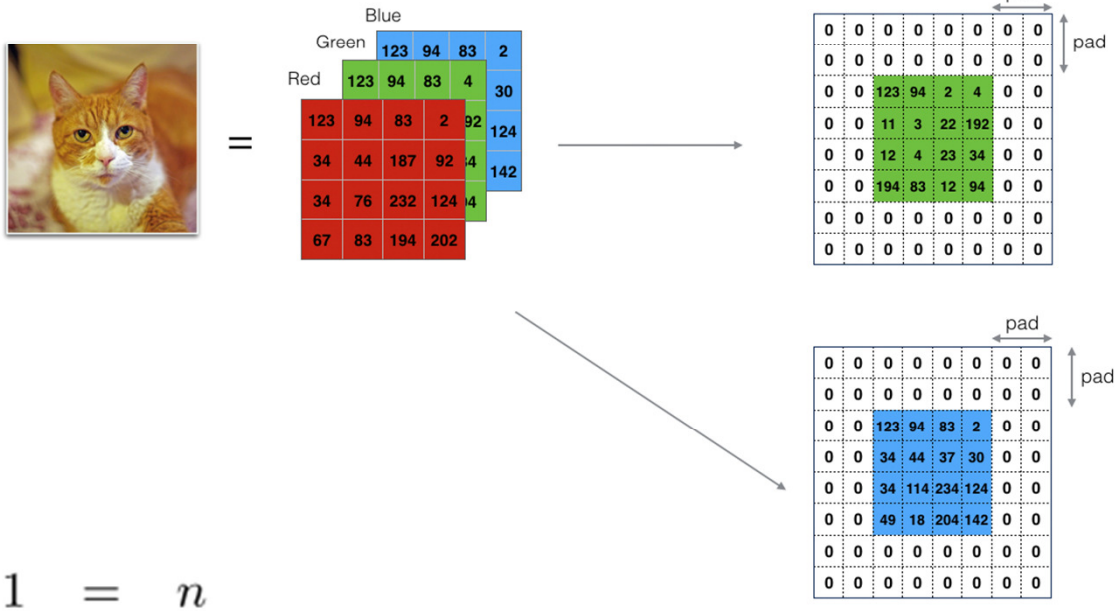
SYMMETRIC PADDING

Add p extra columns and rows at the image borders with 0 values (zero padding). Output matrix size:

$(n+2p-f+1) \times (n+2p-f+1)$

"valid" convolution =>
no padding

"same" covolution =>
Pad so that output size is the same as the input size. Formula for choosing p
(f is usually odd number!):



$$
\begin{aligned}
n + 2p - f + 1 &= n \\
2p - f + 1 &= 0 \\
2p &= f - 1 \\
p &= (f-1)/2
\end{aligned}
$$

STRIDED CONVOLUTION

$$\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{bmatrix}$$

Stride: how many pixels (steps) we shift to the right or down after each convolution.

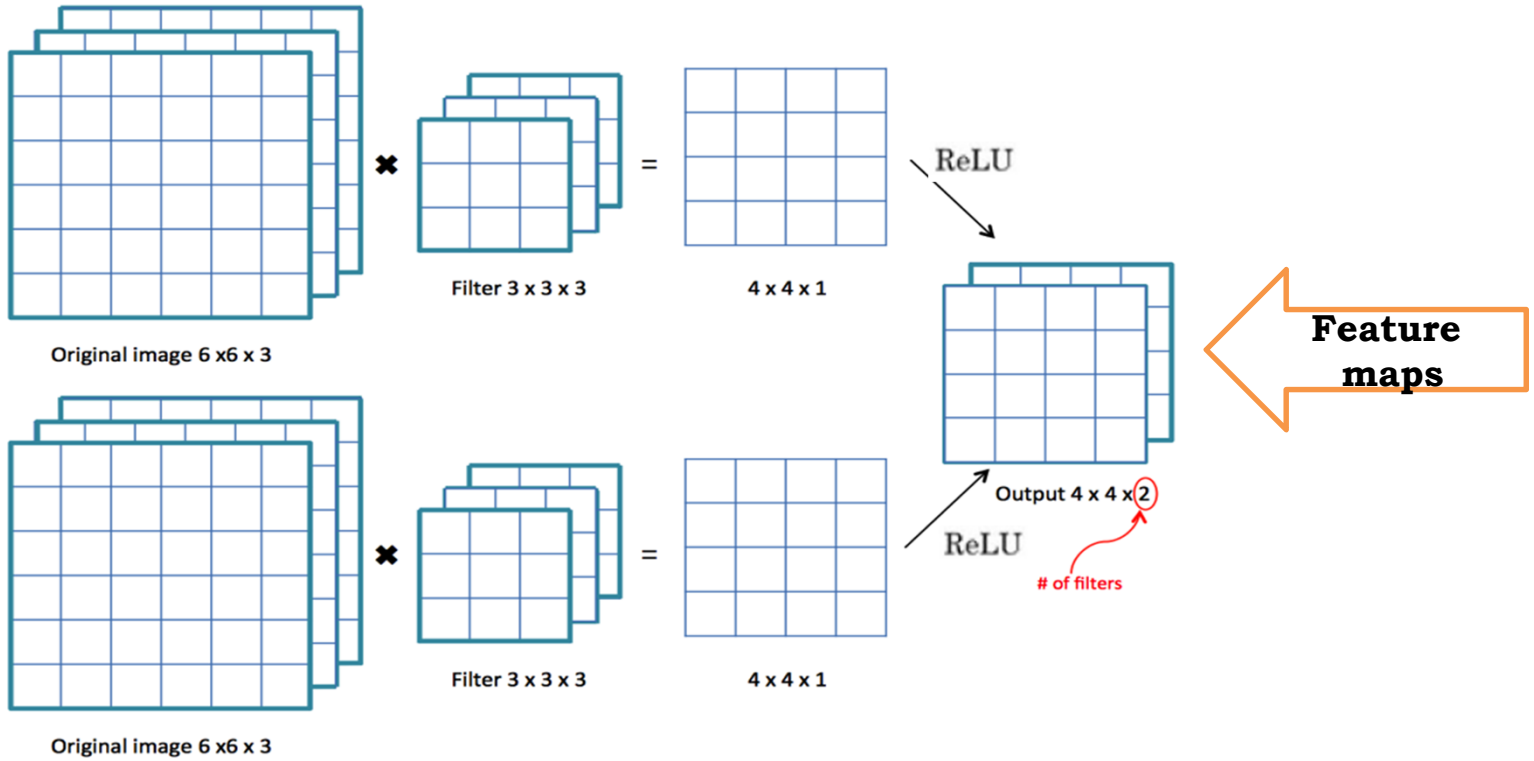
In general: given $n \times n$ input matrix and $f \times f$ filter with padding p and stride s the convolution operation will compute output matrix with size:

$$\lfloor \frac{n+2p-f}{s} + 1 \rfloor \text{ by } \lfloor \frac{n+2p-f}{s} + 1 \rfloor$$

If the formula computes a non-integer value => choose the closest lower integer.

Ex.: no padding (p=0) , stride s=2 => $(7 + 0 - 3)/2 + 1 = 4/2 + 1 = 3$ => 3 ×3 matrix

Multiple Conv Filters over Volumes (3D filters)



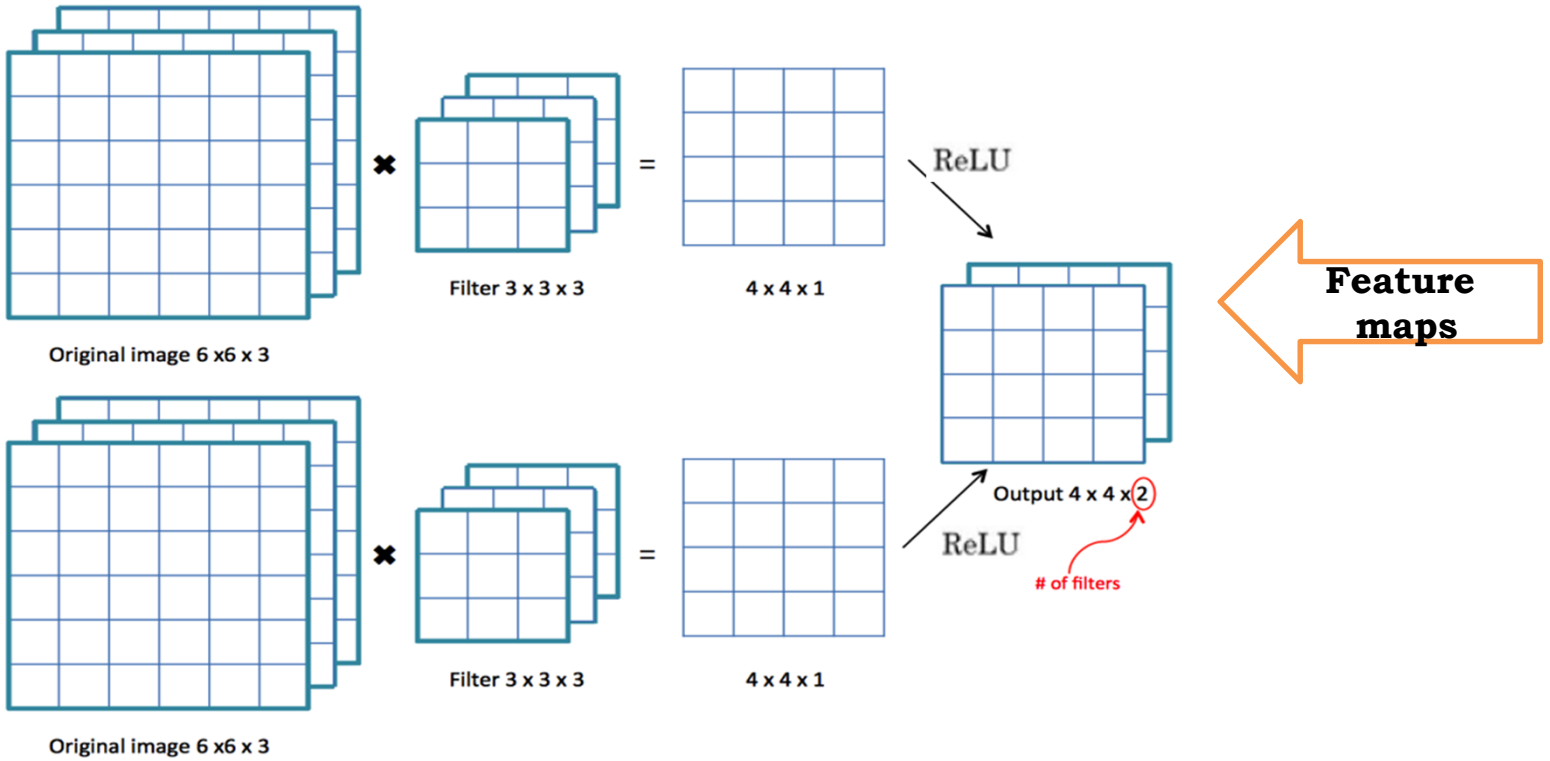
RGB images have 3 dimensions: height, width, and number of channels (3D volume).

The conv filter will be also a 3D volume : $f \times f \times 3$ (number of channels has to be equal in the image and the filter)

$(n \times n \times n_c)$ *image* * $(f \times f \times n_c)$ *filter => (n-f+1) x (n-f+1) x n_filters (no padding)*

Play conv_kiank

ONE CONV LAYER OF CNN



Different 3D filters (kernels) are applied to the 3D input image and the result matrices are stacked to form a 3D output volume.

After the convolution operation the result is passed through an activation function (e.g. ReLU, or sigmoid, linear, etc.).

The outputs of the conv layers are known as feature maps.

Number of parameters in one layer

Ex. If you have 10 filters that are $3 \times 3 \times 3$, in one layer of a CNN, how many parameters does that layer have ?

Answer: $3 \times 3 \times 3$ =27 + 1 (bias) =28 x 10 (filters) => in total 280 parameters

Major property of CNN:

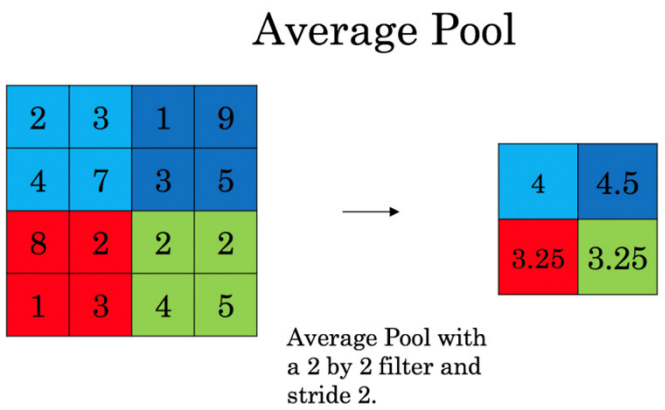
The number of parameters does not depend on the image size (or on the input from the previous layer).

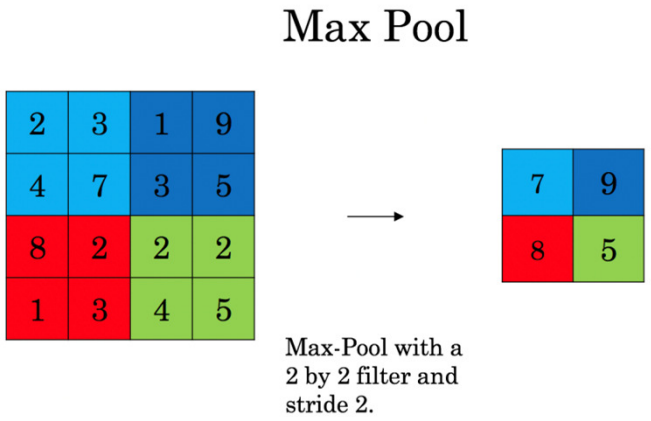
Even in a very large image, we end up with a small number of parameters.

This makes them less prone to overfitting.

The filters detect different features (horizontal, vertical edges, etc.)

POOLING (POOL)





Pooling operation reduces the size of the representation to speed up the computation and make the features more robust.

Ex. Divide the input in regions (e.g. 2 × 2 filter), choose stride (e.g. s=2), each output will be the max (**max pooling**) or the average (**average pooling**) from the corresponding regions.

Some intuition:
Large number means there is some strong feature (edge, eye) detected in this part of the image, which is not present in another part.
Max Pool: whenever this feature is detected it remains preserved in the output.

SUMMARY OF POOLING

The size of the regions (f) and the stride (s) are hyper-parameters.
Common choice f =2, s=2 has the effect of shrinking the height and width of the representation by a factor of 2.

There are no parameters to learn (by the optimization method).

The pooling is done independently for each of the input channels.

Max pooling is much often used than average pooling.

There is no theoretical proofs why pooling works well.
It is just a fact in practice that this approach works well with some data sets.

Softmax Layer

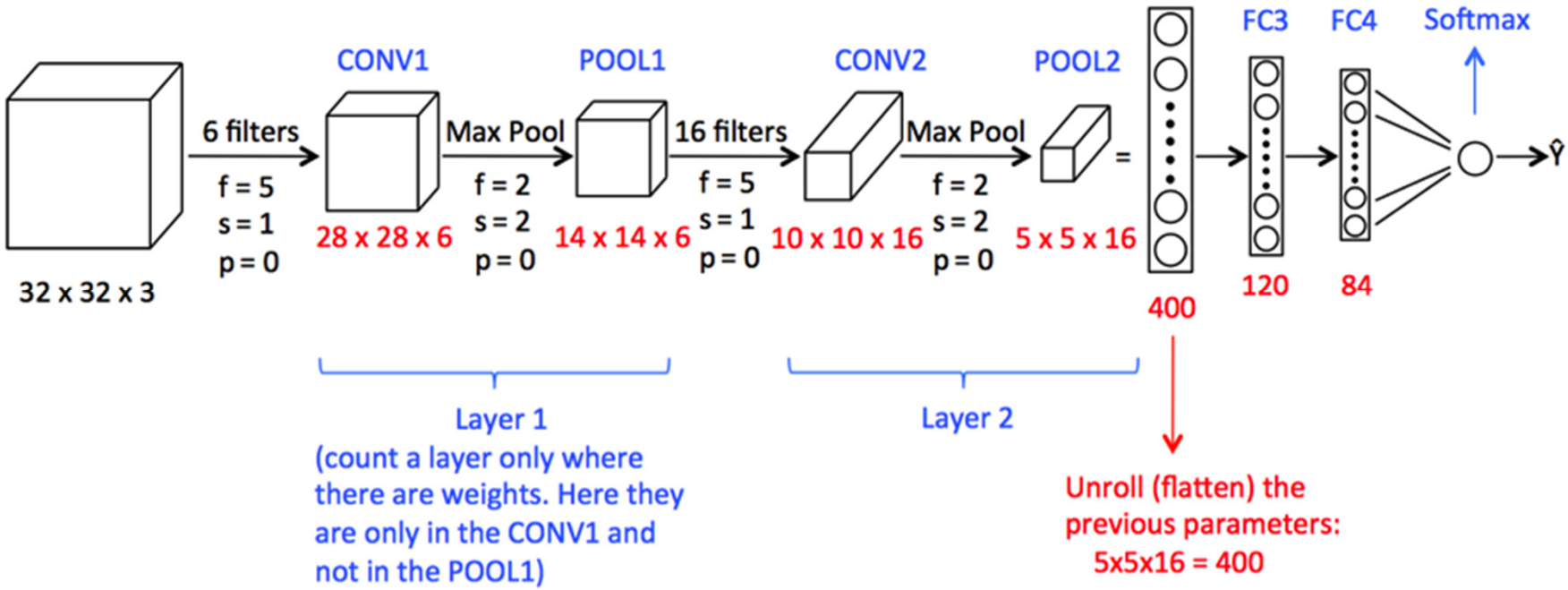
Softmax Layer (SL) estimates the probability that an example $x^{(i)}$ belongs to each of the k classes ((j=1,2,..k).

$$p(y^{(i)} = j|x^{(i)};\theta) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}$$

SL outputs *k* dimensional vector with estimated probability for each class k:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Classical CNN Example: LeNet5*



***LeCun et al., 1998, "Gradient-based learning applied to document recognition".** Original LeNet5 applied to handwritten digit recognition (grey scale images). Avg pooling, no padding, softmax classifier; ReLU and sigmoid/tanh neurons in the Fully Connected (FC) layers.

The activation function is always present after the convolution, even if it is not drawn on the CNN diagram.

General trend: CNNs start with large image, then height and width gradually decrease as it goes deeper in the network, where as the number of channels increase.

Convolution Benefits

Major advantages of conv layers over fully connected (FC) layers:

(1) parameter sharing
(2) sparsity of connections

Ex. Take 32 × 32 × 3 RGB image (3,072 inputs), using 6 filters (5 × 5 × 3) , we get 28 × 28 × 6 dimensional output (4,704 neurons). If we connect every neuron to the inputs (as in a fully connected Neural Networks), the number of parameters would be about 14 million. This is a lot of parameters to train and we have just a small (32 × 32 × 3) image.

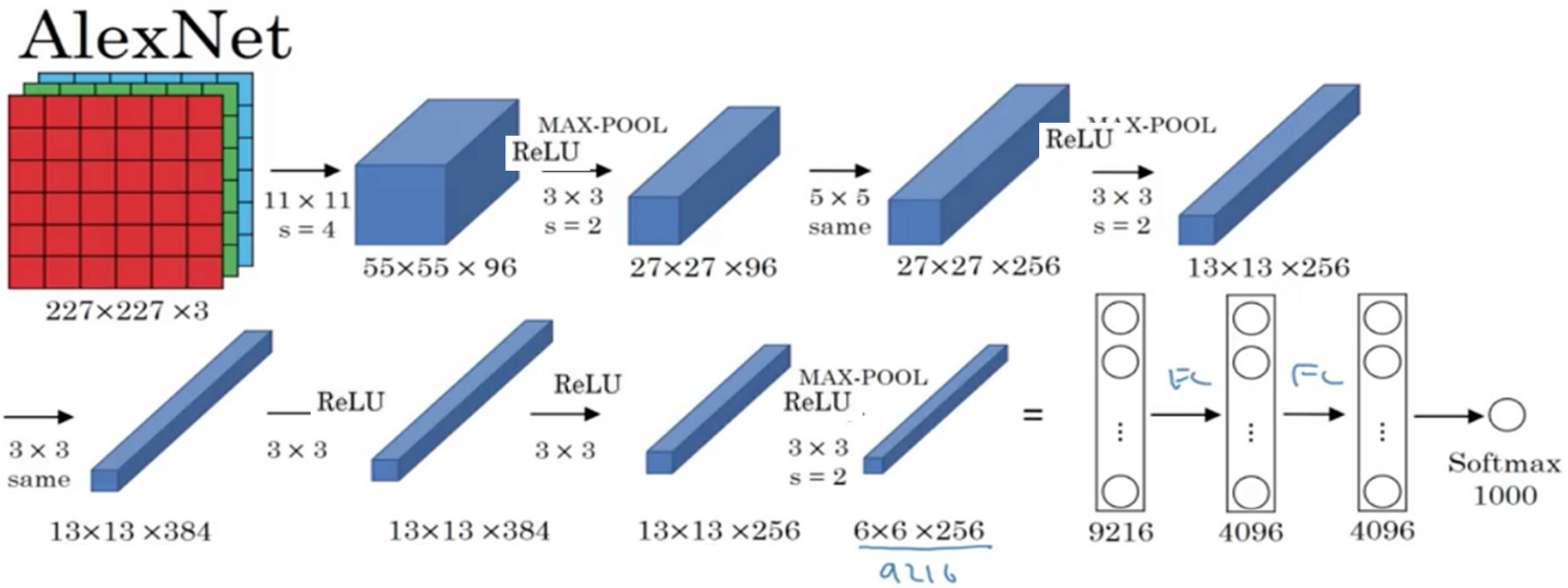
Parameter sharing is a feature detector (such as vertical edge detector) that is useful in one part of the image and is probably useful in another part of the image.

Sparsity of connections means that, in each layer, each output value depends only on a small number of inputs.

LeNet5 has only 60,000 parameters.

The conv layers have much less parameters than FC layers.

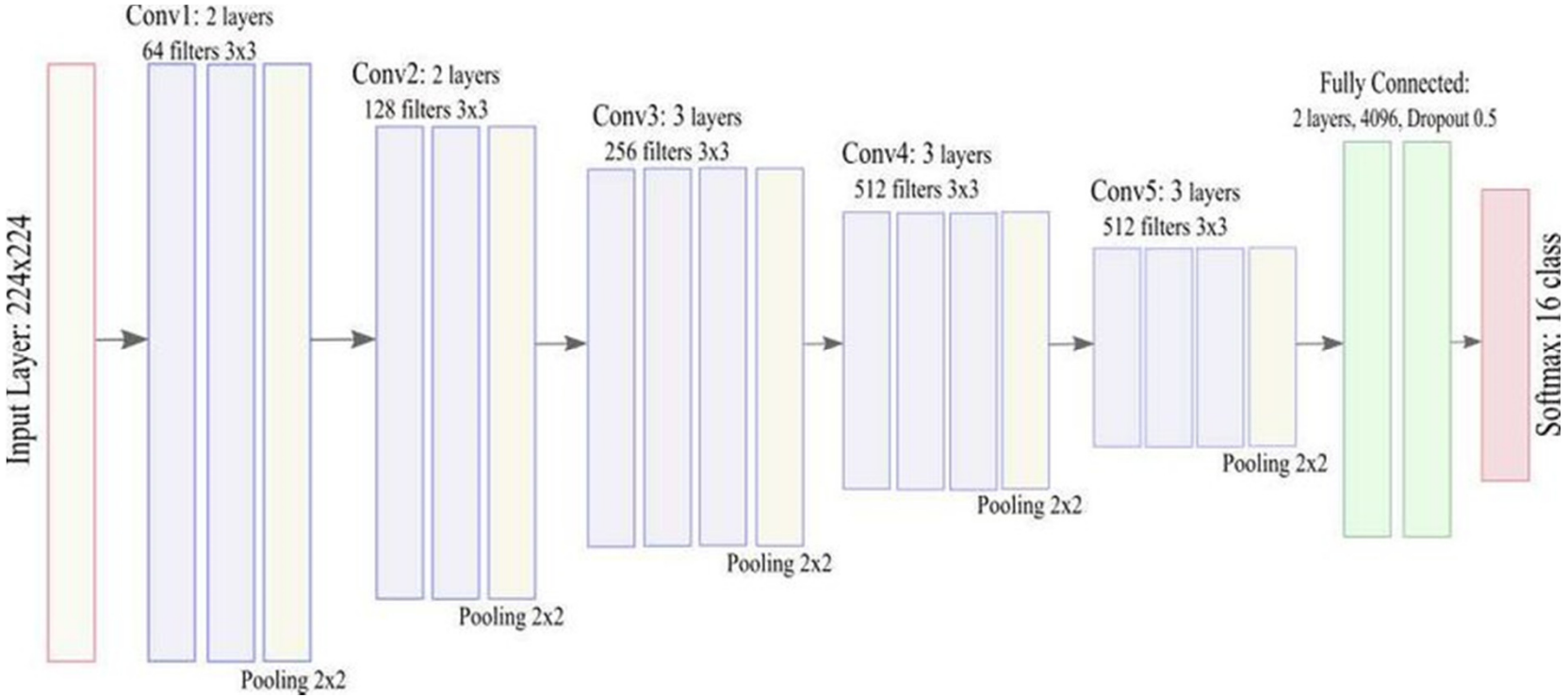
AlexNet*



*** Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012, ImageNet classification with deep convolutional neural networks.**
5 conv layers, 3 FC layers with softmax output, 60 million parameters in total.
AlexNet applied to ImageNet LSVRC-2010 dataset to recognize 1000 different classes.

This paper convinced the Computer Vision (CV) community that DL really works and will have a huge impact not only in CV but also in speech/language processing.

VGG-16 *



*** Karen Simonyan, Andrew Zisserman, 2015, Very Deep Convolutional Networks for Large-Scale Image Recognition**

VGG-16 has 16 layers (with weights !), 138 million parameters.

Unified architecture: All conv layers: (3x3) filters, s=1, same; All MaxPool =2x2, s=2.

At each convolution the height and width go down by a factor of 2, the channels go up by a factor of 2.

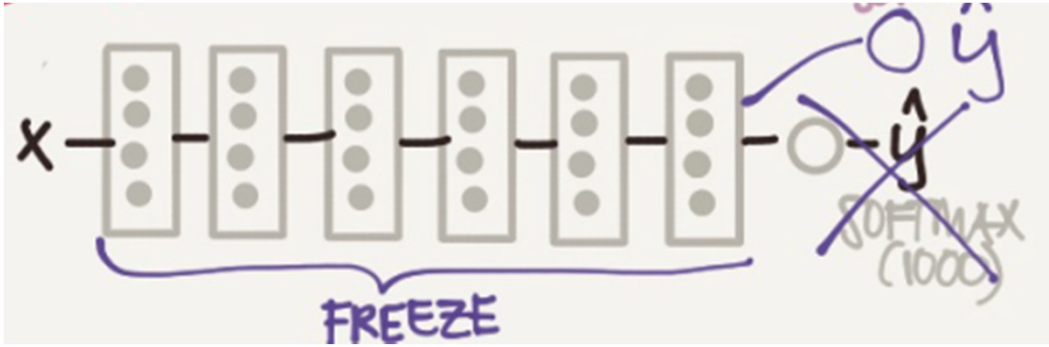
VGG-19 is a larger version of VGG-16, both have similar performance.

Transfer Learning

- Starting from an open-source architecture from web (e.g. cloned from github) is faster than implementing code from scratch.

- Training may takes weeks/months, many GPUs, better use a pre-trained model (pre-trained parameters) => that is Transfer Learning (TL)

Ex.: your problem has 3 classes (car, pedestrian, neither).
You have a small training set.
- Take a DNN trained for 1000 classes.
- Substitute the last classification layer (softmax with 1000 outputs)) with a softmax with 3 outputs.
- Freeze the parameters in all other layers, train only the softmax layer.
- Comp. trick: pre-compute and save on disc the features before softmax layer.

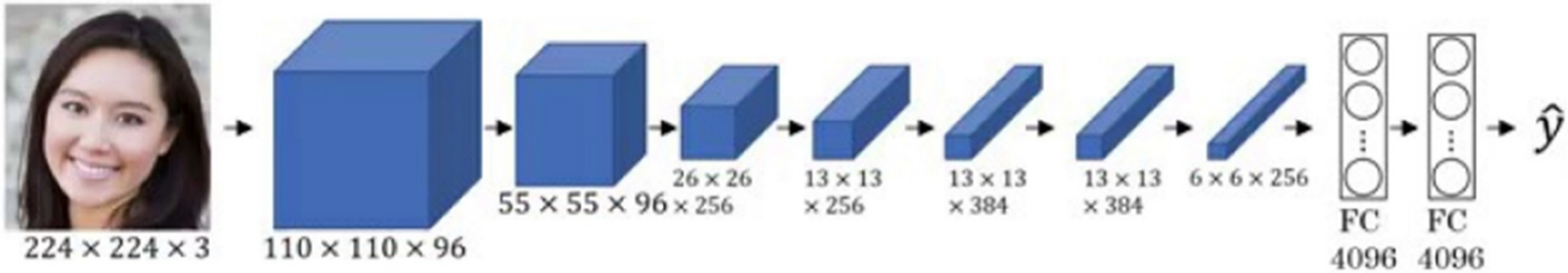


Transfer Learning (cont)

Ex. (cont): If you have larger training set: freeze fewer layers, train latter layers.
The more data you have, the more layers you may train.

If you have big training set, you may use the trained DNN only at the initialization stage, starting not from random parameters but from the parameters of the trained DNN. Then update all weights during the optimization.

Intuition: Hidden layers earlier in the network extract much more general features not specific to the particular task.



Data Augmentation

- Rotation,
- Random Noise
- Mirroring
- Random Croping
- Color shifting: add distorstions to the R (+20) G (-20) B (+20) channels
- Etc.

Off-line: generate all distortions and save the augmented data set.

On-line: common way of implementing data augmentation during training: CPU is constantly loading a stream of images coming from the hard disc and generate distortions to form mini-batches that are constantly passed to a Training algorithm (implemented on a different CPU or GPUs).
The two processes (data augmentation and training run in parallel).

Data augmentation also has hyper-parameters (what kind of distortion, how much distortion).

Data vs. Hand-Engineering

Most ML applications lay somewhere in this spectrum:

Little data <----------> Lots of data

We have lots of data for speech recognition;
Reasonable large data for image recognition (cats or dogs) ;
and much less data for object detection (bounding boxes) .

If Lots of data: the best way to get good performance is to build deep models (several layers), playing with network architectures, but less hand-engineering.

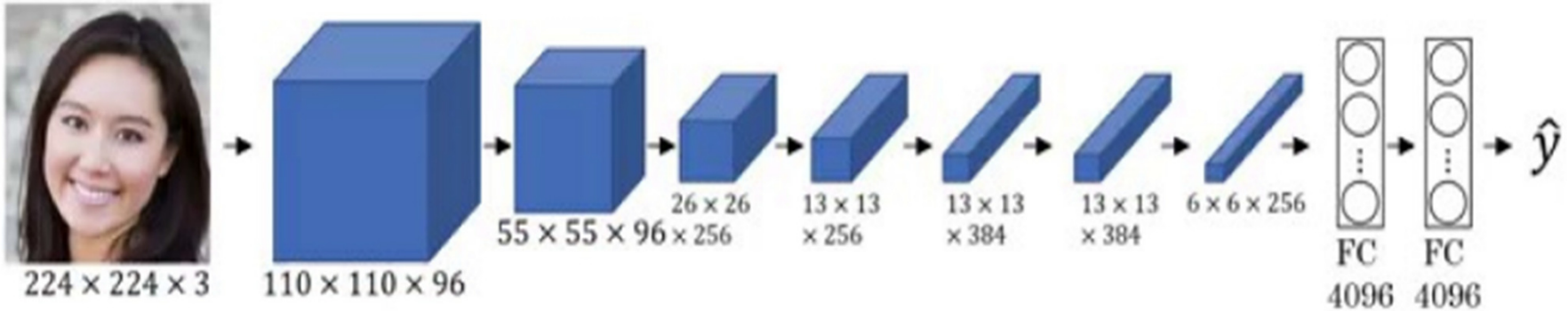
If Litte data: the best way to get good performance is hand-engineering – very difficult and skilful task that requires a lot of inside (expert) knowledge.

ML applications have two sources of data:

- Labelled data

- Hand-engineering features
Historically the hand-engineered data was the main source of data.

What are learning Conv Layers *



Lets say we have trained a ConvNet (e.g. AlexNet like network) and we want to visualize what the hidden units in different layers are computing.
Let's start with one hidden unit in Layer 1.
Suppose you scan through your entire training sets and find out what are the 9 image patches that maximize that unit's activation.
This particular hidden unit seems to be activated (to see) by edges or lines.

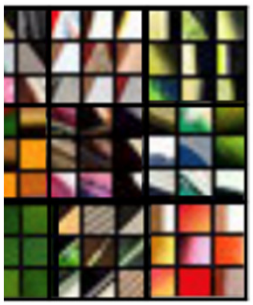
=

Repeat for other units in Layer 1 =>
This picture shows 9 different representative neurons and for each of them the 9 image patches that maximally activate them.

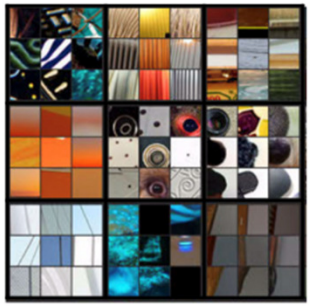
The trained hidden units in Layer 1 respond to relatively simple features such as edges or a particular shade of colour.

* ref. Zeiler&Fergus, 2013 Visualizing and understanding convolutional networks.

Visualizing deep layers: Layer 2

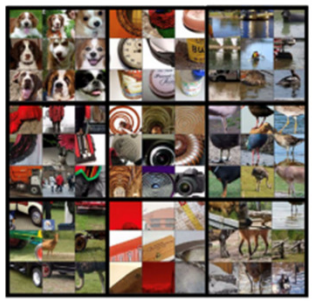


Layer 1



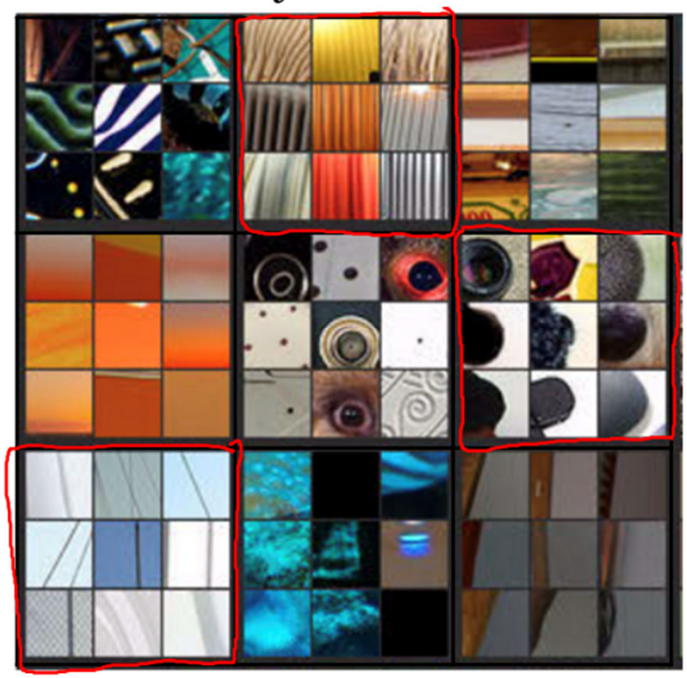
Layer 2

Layer 3



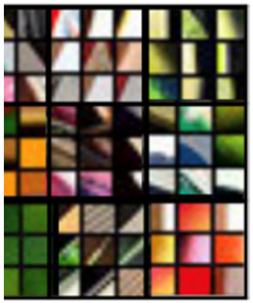
Layer 4

Layer 5

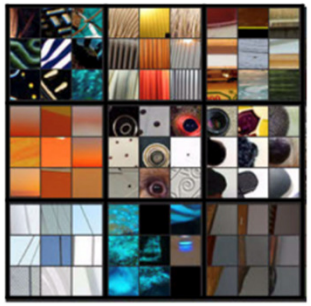


In deeper layers, the hidden units see larger regions of the image.
The units in Layer 2 are activated from more complex shapes and patterns, such as vertical lines with texture, rounder shapes at the left part of the image and so on.
The features that Layer 2 is detecting are getting more complicated.

Visualizing deep layers: Layer 3

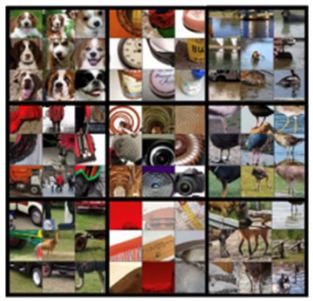


Layer 1



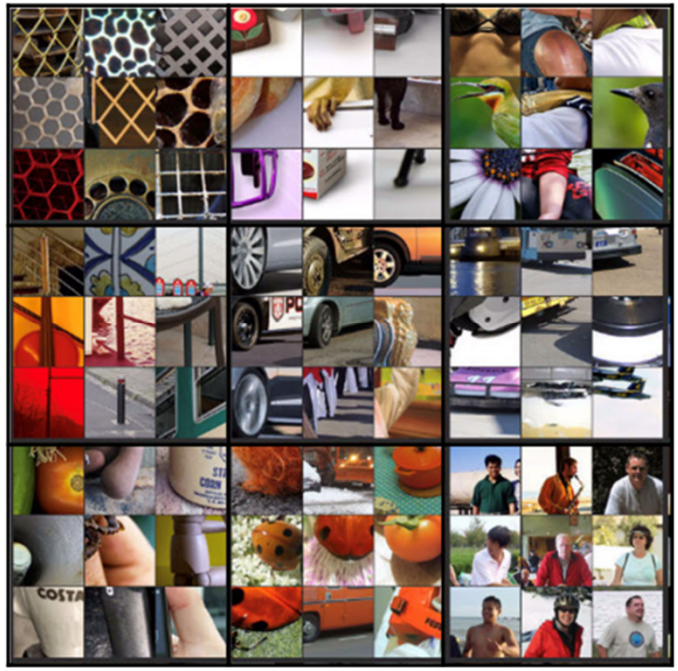
Layer 2

Layer 3



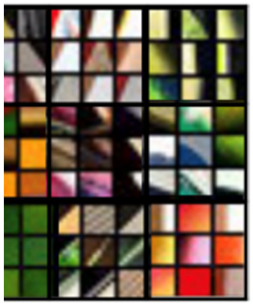
Layer 4

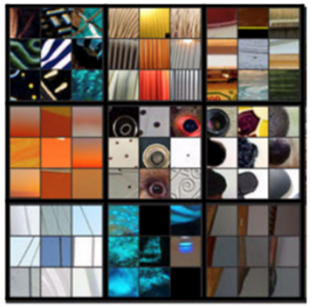
Layer 5



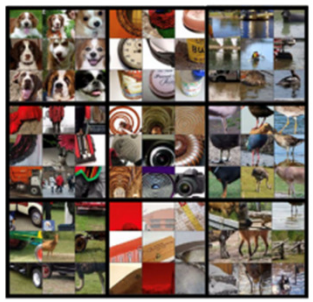
In Layer 3, the hidden units start detecting part of cars, irregular texture, even people, other shapes difficult to figure out what they are, but it is clearly starting to detect more complex patterns.

Visualizing deep layers: Layer 4

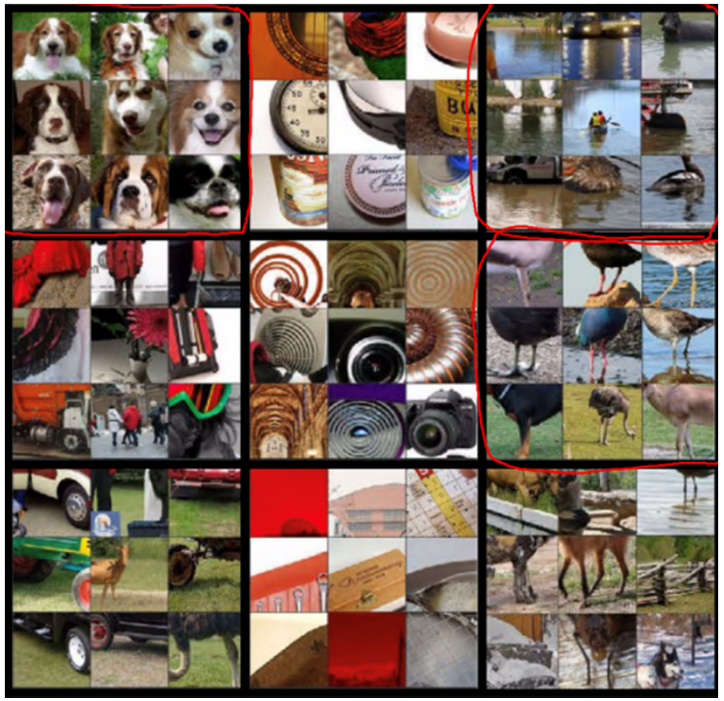

Layer 1


Layer 2


Layer 3

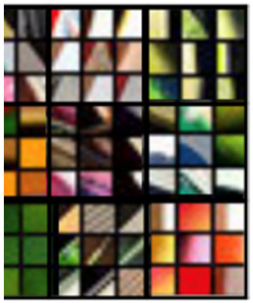

Layer 4


Layer 5

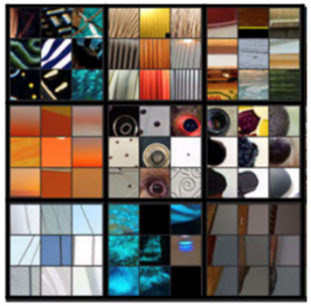


In Layer4, one unit seems to be a dog detector, but all dogs are somehow similar, other unit detects water, other unit detects legs of birds and so on.
Detected patterns are even more complex then in Layer3.

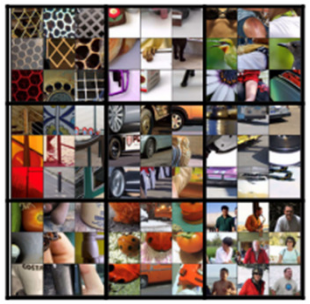
Visualizing deep layers: Layer 5



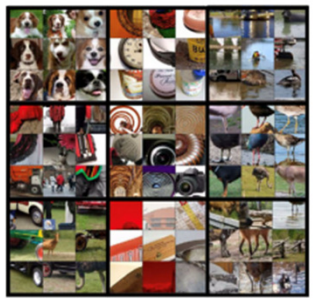
Layer 1



Layer 2

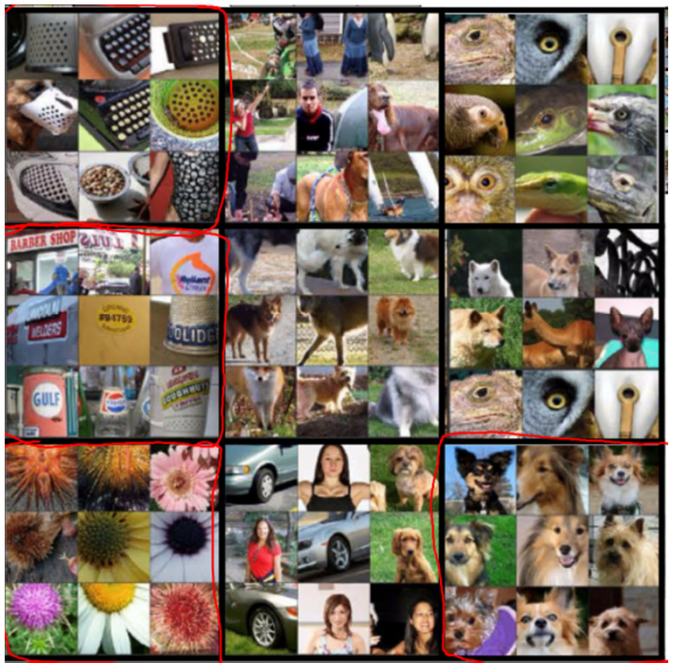


Layer 3



Layer 4

Layer 5



In Layer 5, one neuron seems to be a dog detector, but the detected dogs seem to be more varied; other detects flowers; other neuron seems to detect things with a keyboard like texture, or maybe lots of dots against background; one neuron may detects text, it's always hard to be sure.

Image classification/localization/detection

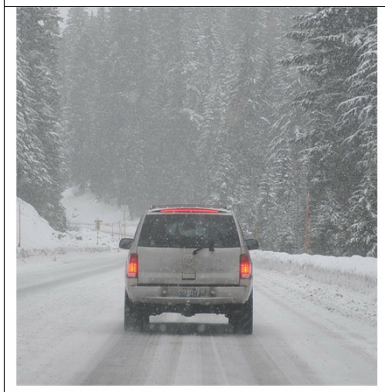
Image classification	Classification & Localization	Detection
	 b_x, b_y, b_h, b_w	
"car"	"car"	multiple objects

Image classification: input a picture to CNN and the output is a class label (e.g. person, bike, car, etc.)

Classification with localization: the algorithm gives not only the class label of the object but also draws a bounding box (the coordinates) of its position in the image.

(b_x , b_y , b_h , b_w) describes the bounding box.

Further Reading

- Ian Goodfellow, and Yoshua Bengio, Deep Learning, MIT Press, 2016

- Andrew Ng, Machine Learning Yearning, 2018 (https://www.deeplearning.ai/machine-learning-yearning/)