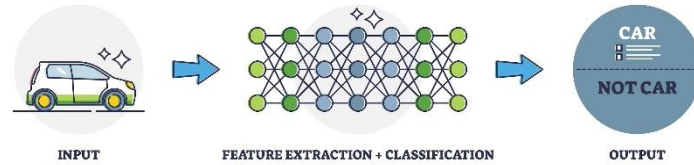


## DEEP LEARNING

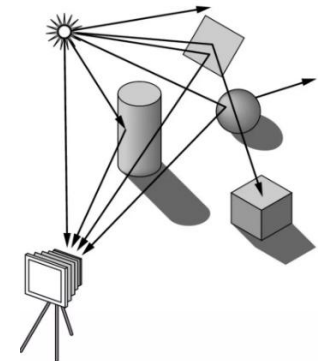


# MACHINE LEARNING COMPUTER VISION COMPUTER GRAPHICS

Alexander Meyer <sup>1</sup>

<sup>1</sup> SAARA team, LIRIS laboratory

Master ID3D, AI and DISS



Computer Graphics

# Timetable

- Master
  - IA: option « ML and Images »
  - ID3D: « ATIV3D »
  - DISS
- Lectures (CM): Thursday 14h-15h30
- Practice (TP):
  - Thursday 15h45-17h30++: IA and DISS
  - Tuesday 9h45-11h15: ID3D (most of the time)

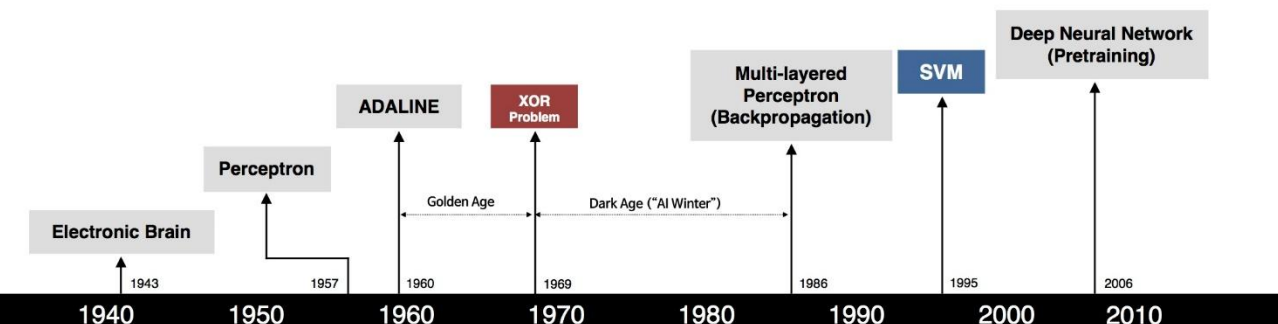
# Deep learning history

- Deep Learning needs DATA: images, texts
  - Computer vision (CV)
  - Natural Language Processing (NLP)



Golden age of deep learning : data + GPU/TPU

CNN, AE, GAN, Transformer, Diffusion...



<p>S. McCulloch - W. Pitts</p> <ul style="list-style-type: none"> <li>• Adjustable Weights</li> <li>• Weights are not Learned</li> </ul>	<p>F. Rosenblatt</p> <ul style="list-style-type: none"> <li>• Learnable Weights and Threshold</li> </ul>	<p>M. Minsky - S. Papert</p> <ul style="list-style-type: none"> <li>• XOR Problem</li> </ul>	<p>D. Rumelhart - G. Hinton - R. Williams</p> <ul style="list-style-type: none"> <li>• Solution to nonlinearly separable problems</li> <li>• Big computation, local optima and overfitting</li> </ul>	<p>V. Vapnik - C. Cortes</p> <ul style="list-style-type: none"> <li>• Limitations of learning prior knowledge</li> <li>• Kernel function: Human Intervention</li> </ul>	<p>G. Hinton - S. Ruslan</p> <ul style="list-style-type: none"> <li>• Hierarchical feature Learning</li> </ul>
--	--	--	---	---	--

# Computer vision in 2012 ...

- Computer vision need algorithms
- Hard problem since “inverse problem”
- Remains limited compared to human vision, even if enormous progress has been made ...

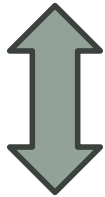


26x27 pixels

# Deep Learning ⇔ Computer Vision

## Machine Learning (ML)

- Classification
- Regression
- Clustering
- Dimension reduction
- ...



## Computer Vision (CV)

- Classification / recognition
- Segmentation
- Tracking
- ...
- Motion capture (Human body)
- 3D Reconstruction
- Image generation
- Novel views generation

## DataSet

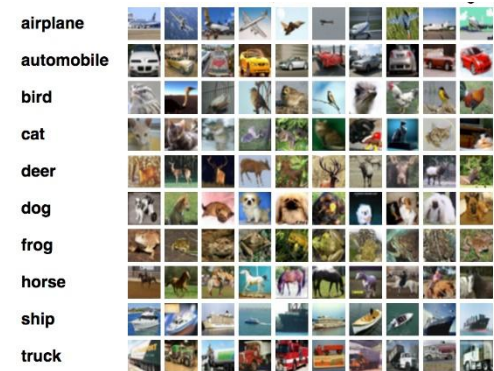
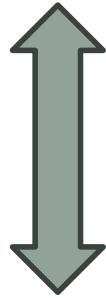


Figure 1: We introduce the **Common Objects in 3D (CO3D)** dataset comprising 1.5 million multi-view images of almost 19k objects from 50 MS-COCO categories annotated with accurate cameras and 3D point clouds (visualized above).

# Deep Learning ↔ Computer Vision

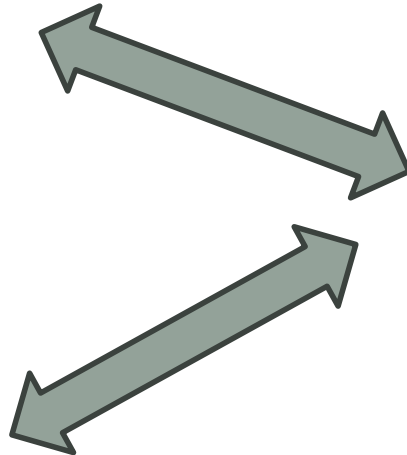
## Machine Learning

- Classification
- Regression
- Clustering
- ...



## Computer Vision

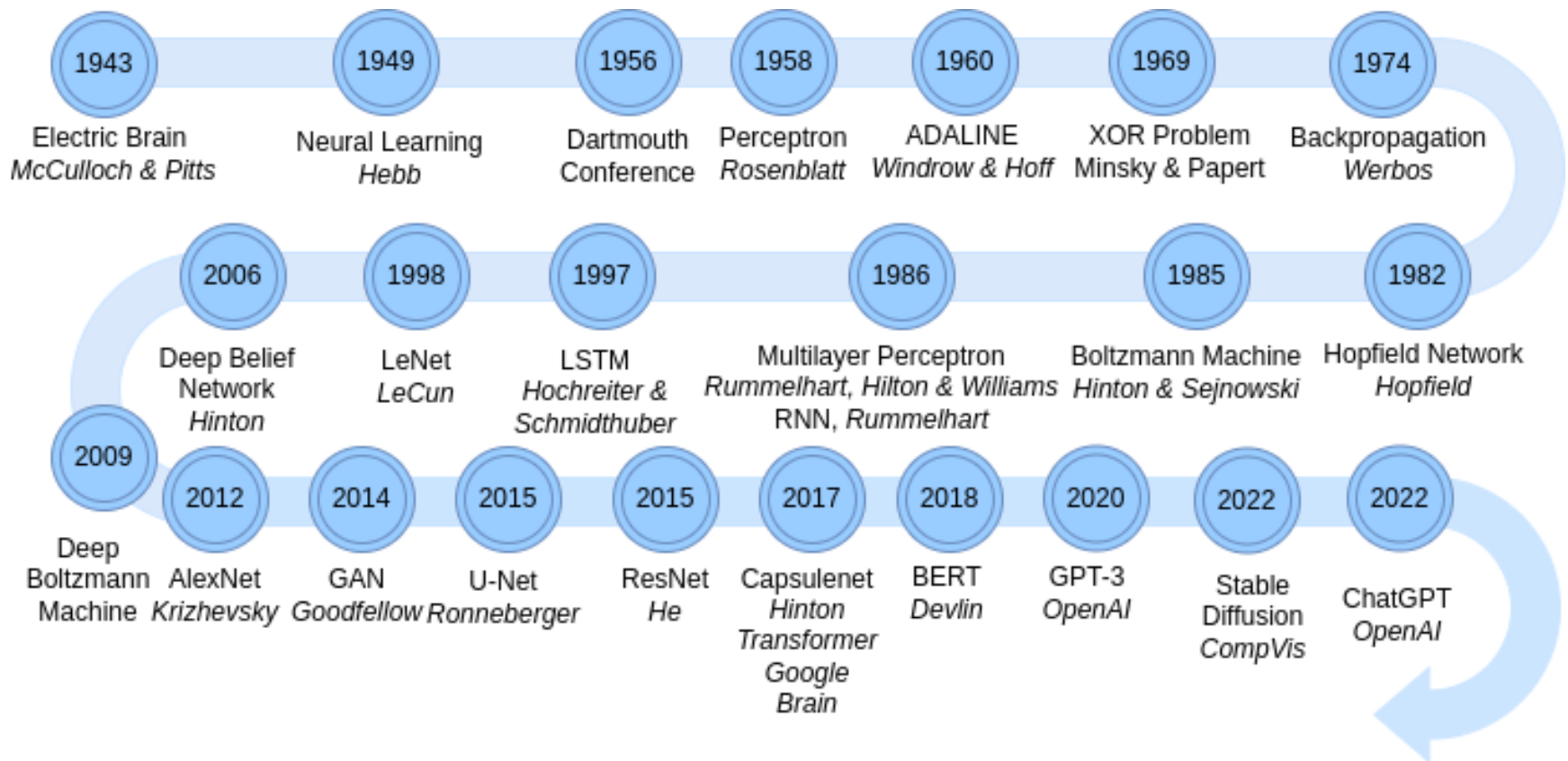
- Classification / recognition
- Segmentation
- Tracking
- ...
- 3D Reconstruction
- Image generation
- 3D Shape/Texture generation
- Novel views generation
- Motion capture (Human body)



## Computer Graphics (CG)

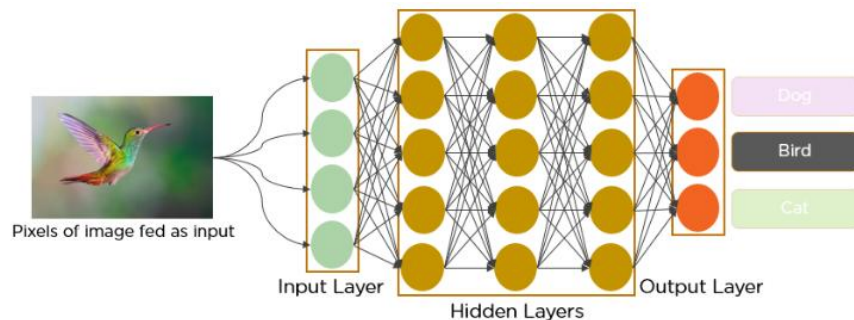
- Rendering=Image generation
  - Geometry (mesh)
  - Texture
  - Light
  - BRDF
  - Animation
- Data
  - From artist
  - From tools (3D models)
  - From computer vision
    - Machine learning

# Deep learning history





before



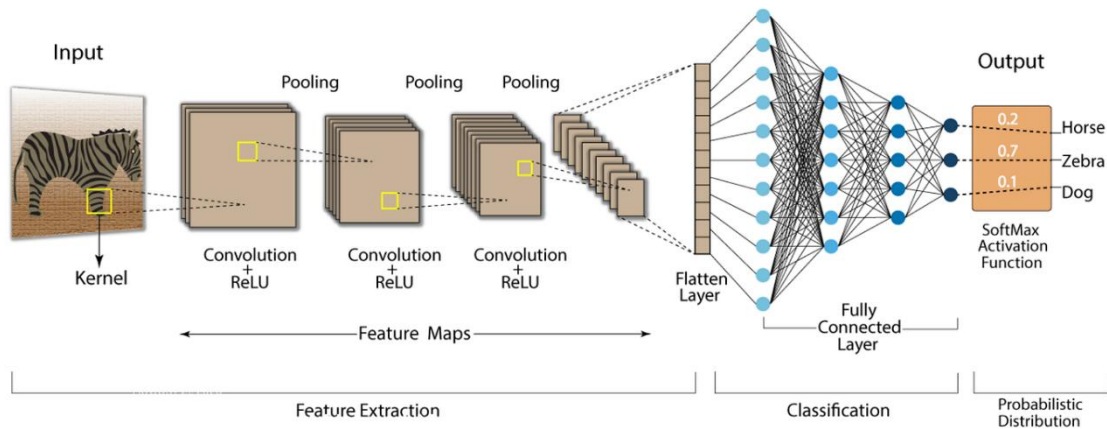
## DEEP LEARNING AND IMAGES

# CONVOLUTION NEURAL NETWORK

## CNN

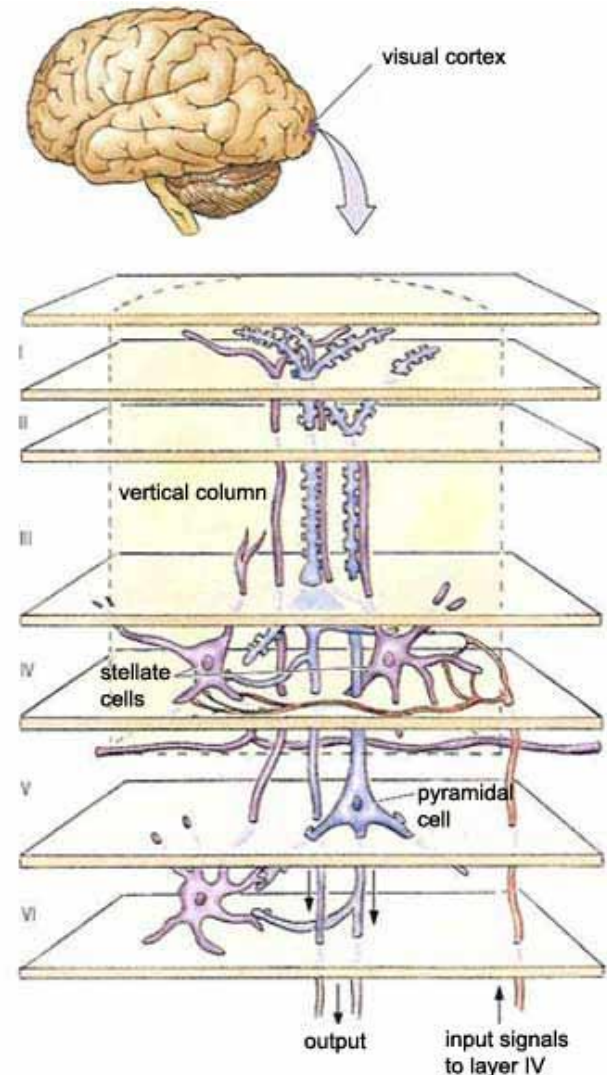
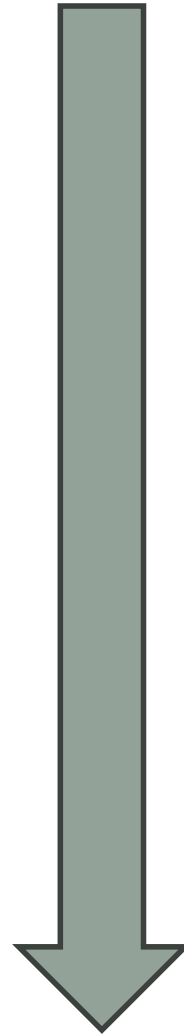
### Convolution Neural Network (CNN)

after



# Computer vision and human vision

- Image processing
  - Change brightness
  - Highlight certain aspects
  - ...
- Pattern recognition
  - Find lines, circles, etc.
  - ...
  - Find faces
- Computer vision
  - Identifying patterns
  - “We see a human face”
  - “This is Paul”
  - Identify action
  - “The person is screwing a bolt”
  - Analyze an action

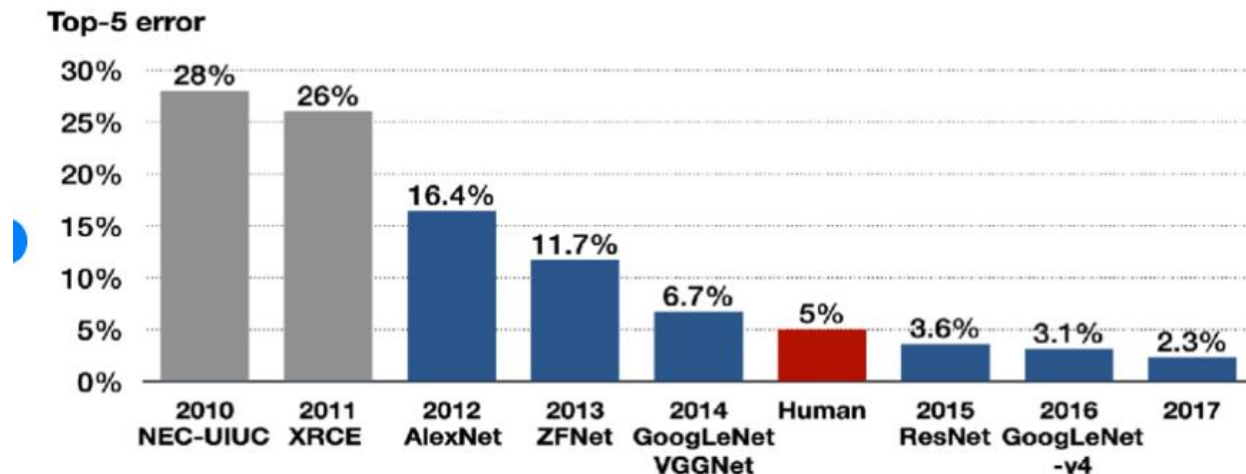


# IMAGENET=image corpus

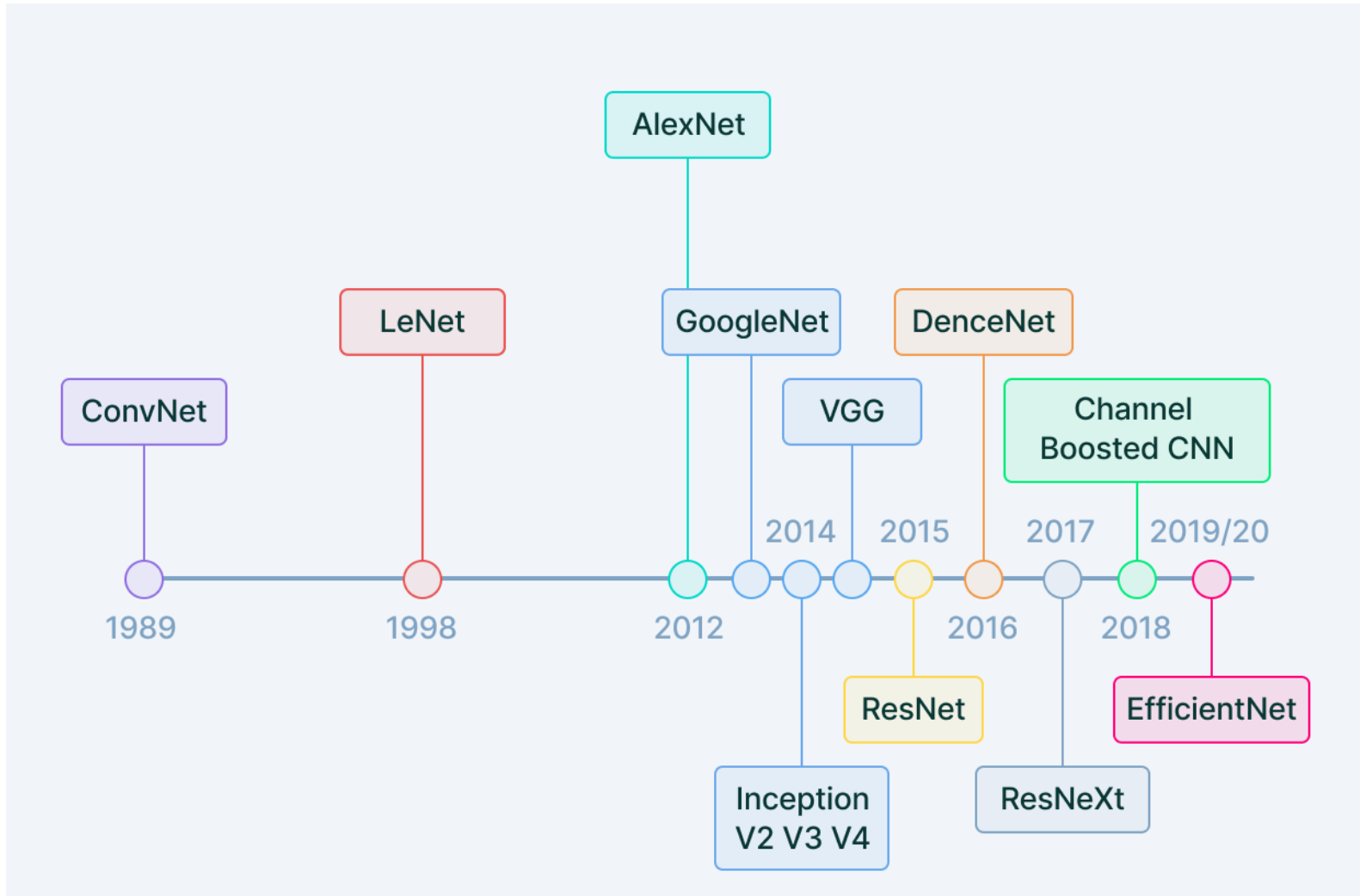


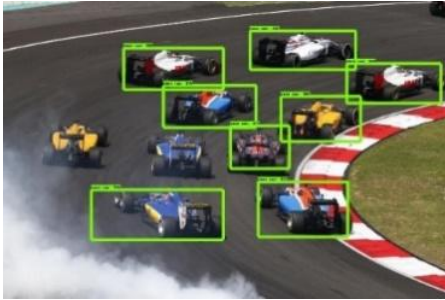
- Classification

- Historical human vision: spotting a predator or family member quickly
- IMAGENET Competition → put a label on an image
  - 14 million images with 20,000 labels
  - In blue, network-based methods

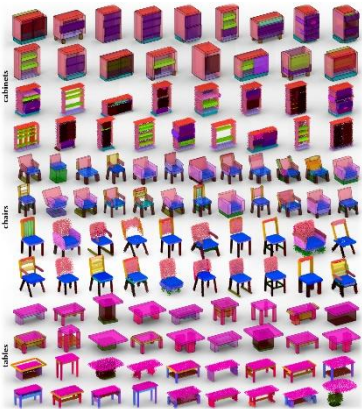


# CNN history

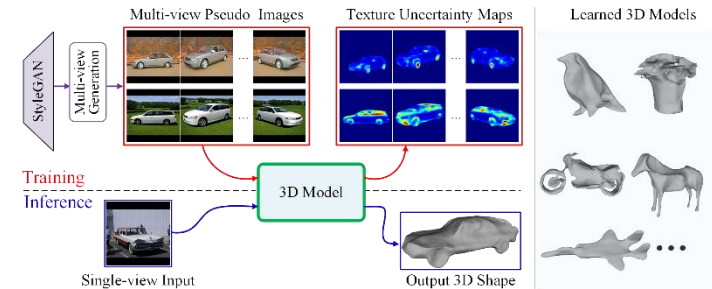
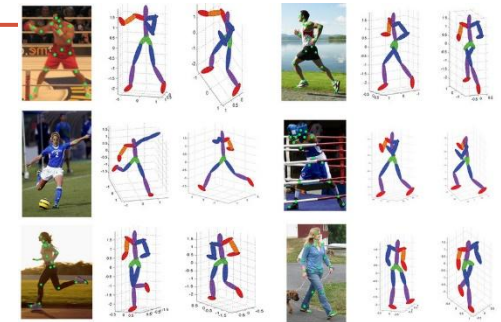




# DEEP LEARNING HAS ENABLED MAJOR ADVANCES IN COMPUTER VISION



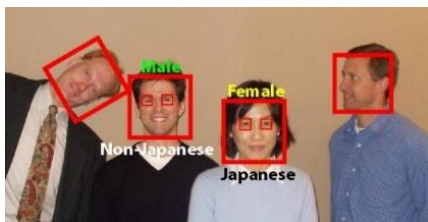
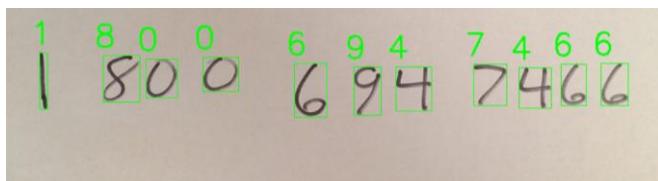
- Image classification
- Segmentation
- Object detection and tracking
- Action recognition
- Motion capture (Human body)
- 3D Reconstruction
- Image generation
- 3D Shape/Texture generation
- ...



# Deep Learning and Recognition

## Image classification

- from a family of objects: "it's a cat"
  - of a specific object: "it's Paul", "it's an 8"
  - of an expression/style: "the face smiles"
  - ...
- Simplest computer vision problem, but was complex for a while before NN



# High-level vision: recognition



What are you see In this image?

- Extremely difficult task
    - The tiger must be recognized from all angles, sometimes hidden, with different lighting in each photo
- ➔ Quite close to a small Turing test

# A picture

- An image is a 2D matrix or 2D tab



157	153	174	168	150	152	129	151	172	161	155	166
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	94	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	60	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	166
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	60	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218



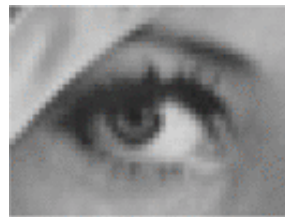
# Gaussian Filter: Blurring

$$\frac{1}{9} \begin{matrix} & & F \\ \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \end{matrix}$$



# Filter = convolution

- Identity



Original

•0	•0	•0
•0	•1	•0
•0	•0	•0

= ?

- Shift right



Original

•0	•0	•0
•0	•0	•1
•0	•0	•0

= ?

# Filter = convolution

- Original – Smoothed  $\rightarrow$  details

original - smoothed (5x5) = detail

•0	•0	•0
•0	•1	•0
•0	•0	•0

•1	•1	•1
•1	•1	•1
•1	•1	•1

$-\frac{1}{9}$

- Original + Details  $\rightarrow$  Sharpened

original + a detail = sharpened

•0	•0	•0
•0	•1	•0
•0	•0	•0

•0	•0	•0
•0	•1	•0
•0	•0	•0

•1	•1	•1
•1	•1	•1
•1	•1	•1

$-\frac{1}{9}$

•0	•0	•0
•0	•2	•0
•0	•0	•0

•1	•1	•1
•1	•1	•1
•1	•1	•1

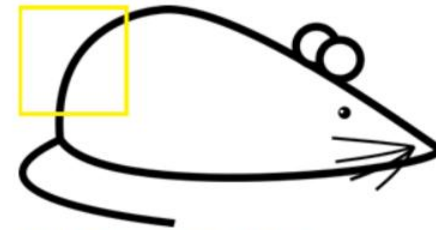
$-\frac{1}{9}$

# Filters

- Detector of curves



Original image



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

\*

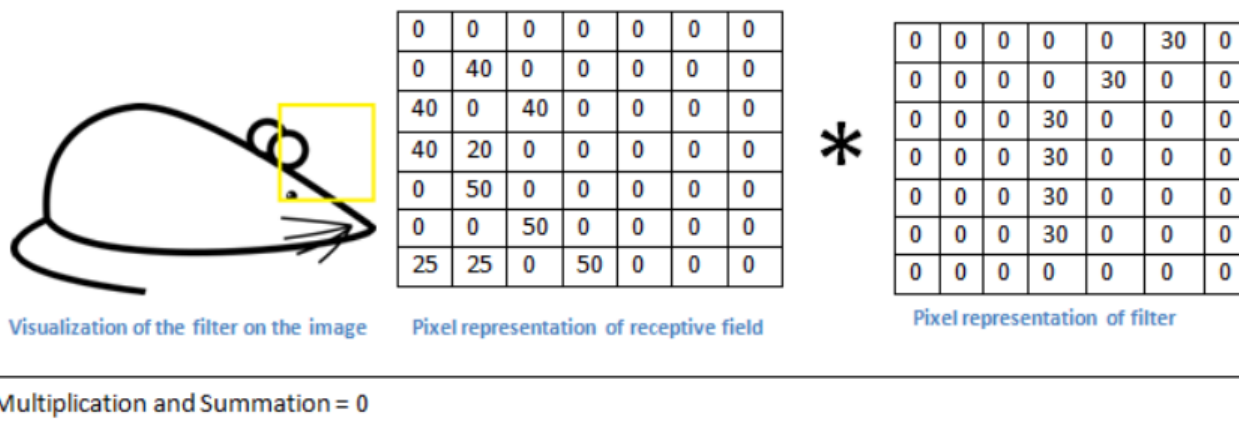
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0

Pixel representation of filter

Multiplication and Summation =  $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$  (A large number!)

# Filter: curve detector

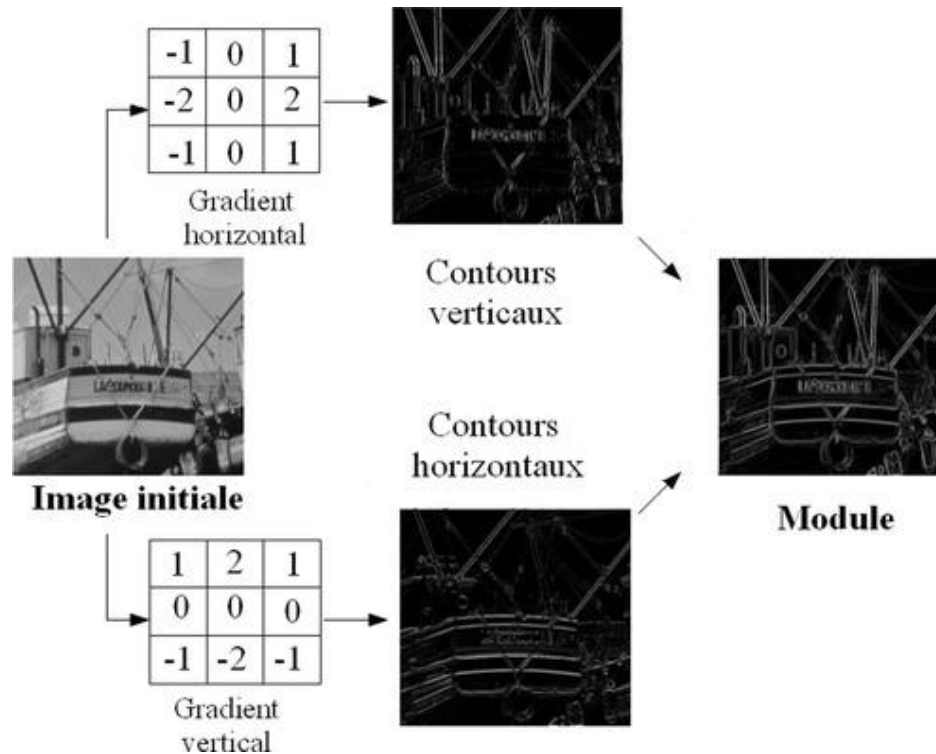
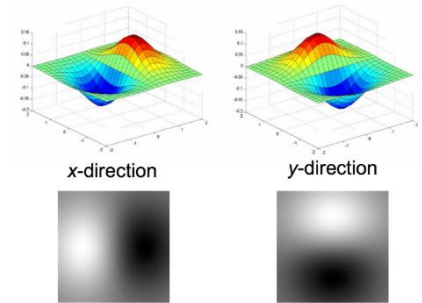
- The same filter elsewhere



- We therefore obtain an activation map for this filter
  - Feature map

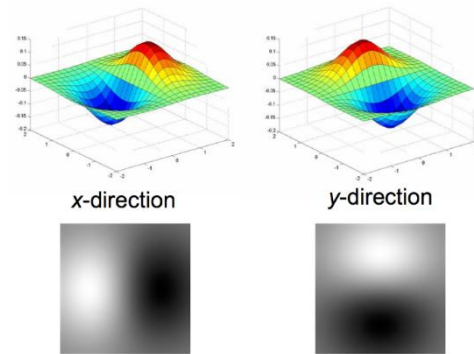
# Sobel

- Derivative of the Gaussian filter in x and y →
- 2 discrete direction filters

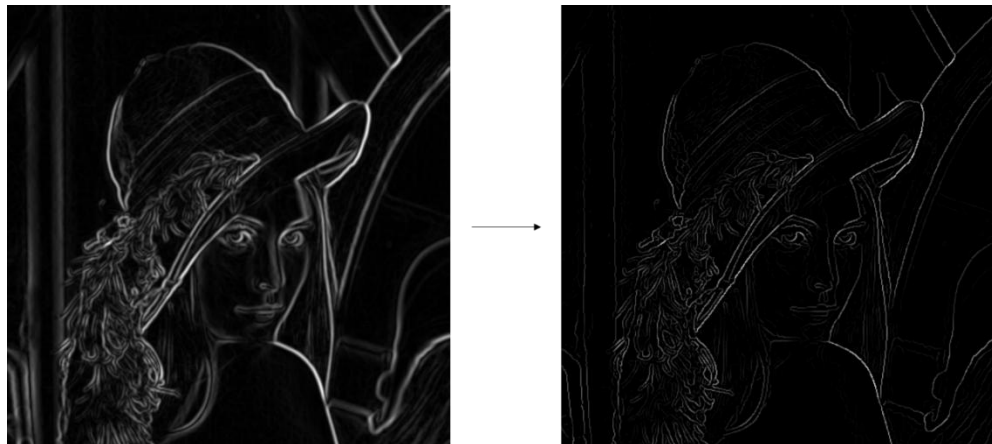


# Filter: Canny edge detector

- Derivative of the Gaussian filter in x and y



- Removing non-maximum pixels



# “Compass” filter

- 8 Directions Detection

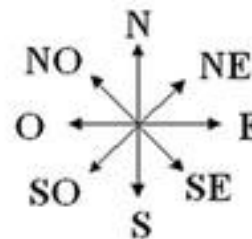
**Exemple :**  
**Robinson (3)**

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

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

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

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



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

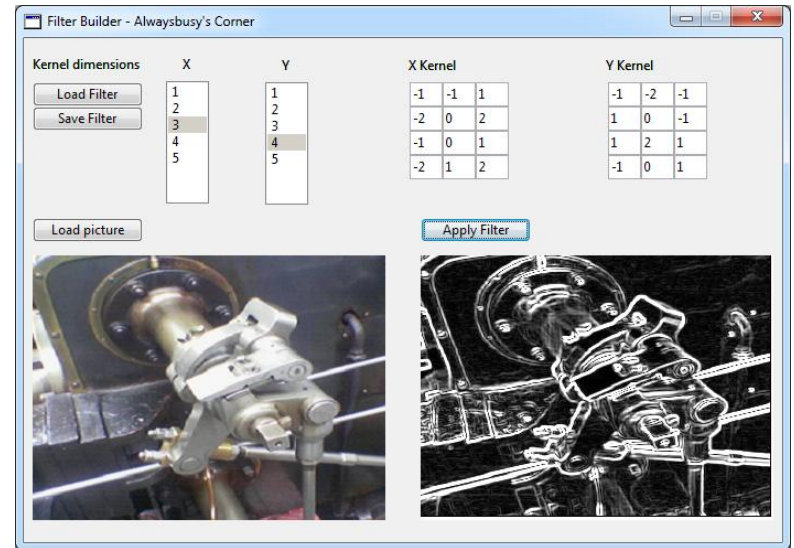
0	-1	-1
1	0	-1
1	1	0

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

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

# Filters / convolution

DEMO



<https://setosa.io/ev/image-kernels/>

→ Test and observe the results

<https://generic-github-user.github.io/Image-Convolution-Playground/src/>

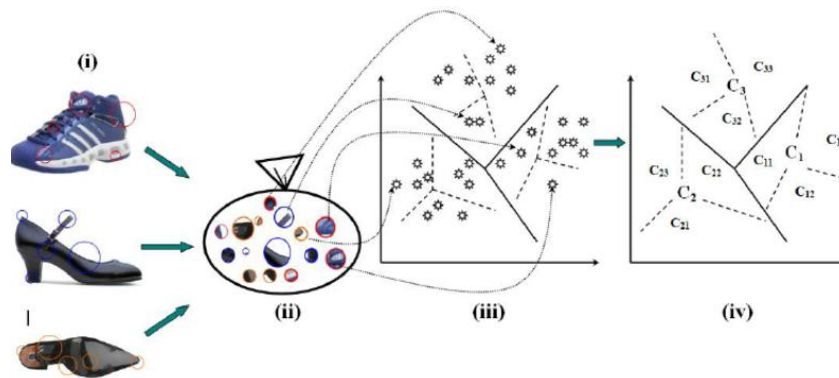
→ Test random !!!

# The Difficulties

What sequence/combination of filters will allow an image to be properly classified?

Before (the deep learning )

- A human (engineer/researcher) proposed
  - filters (less than 10) according to his knowledge
  - Producing a set of descriptors (a few hundred values)
- Then automatic classification (SVM, etc.)



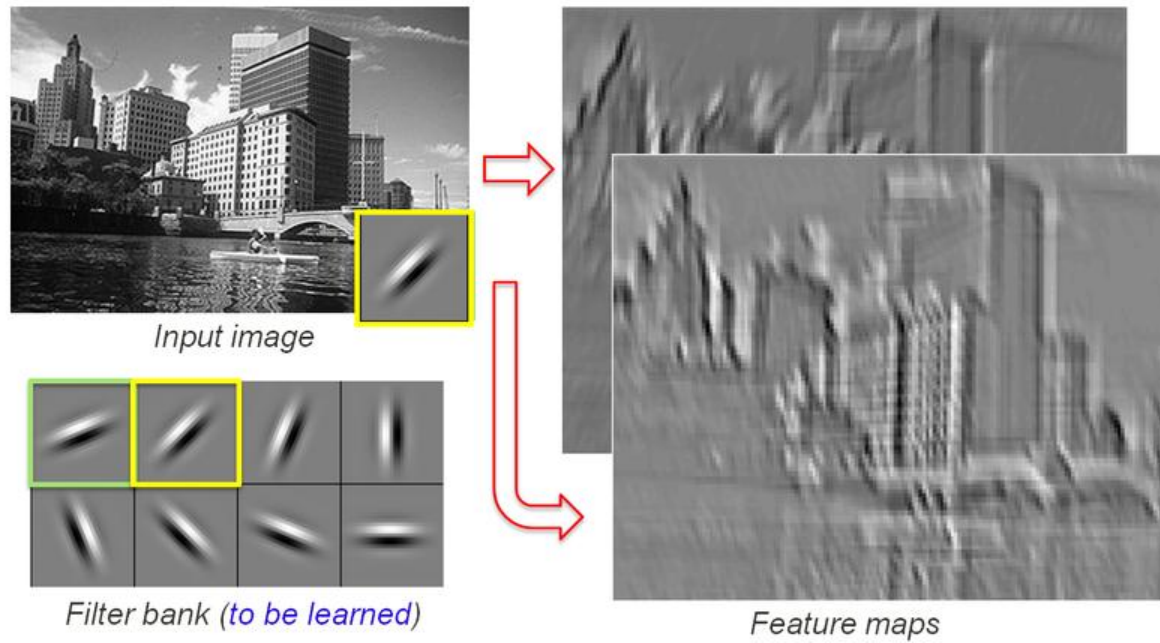
# CNN: change of paradigm

What sequence/combo of filters will allow an image to be properly classified?

- Before
  - a human (engineer/researcher) proposed filters (less than 10) according to his know-how to produce descriptors (a few hundred values), then classification (SVM, Random Forest, etc.)
- **ConvNeuralNET (CNN)**
  - Optimizes weights of multiple filters (hundreds) to produce the descriptors ( *feature maps* ). = One layer of ConvNN
  - Several layers ConvNN
  - Then the last fully connected layers do the classification

# Convolutional Network ConvNET

- Apply a bank of  $N$  filters to an image
  - Gives  $N$  images

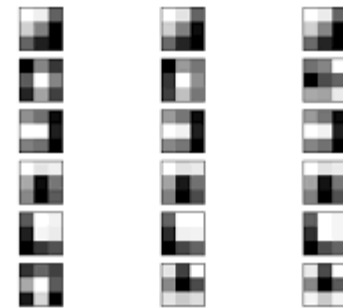
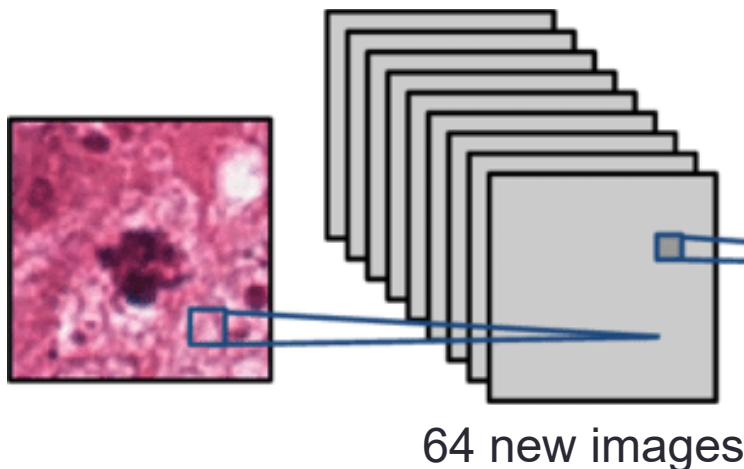


# Convolutional Network ConvNET

## First set of convolutions (filters)

The 64 filters are initialized randomly

C1: feature maps



The 64 filters are optimized in the same process than the network (forward + backpropagation)

# CNN example

- A 3x3 convolution (filter) on a 5x5x1 image
  - Produce a 3x3 images because of the two borders ( $3 = 5-2$ )
  - Resulting images:  $W-2 \times H-2$

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

\*

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

=

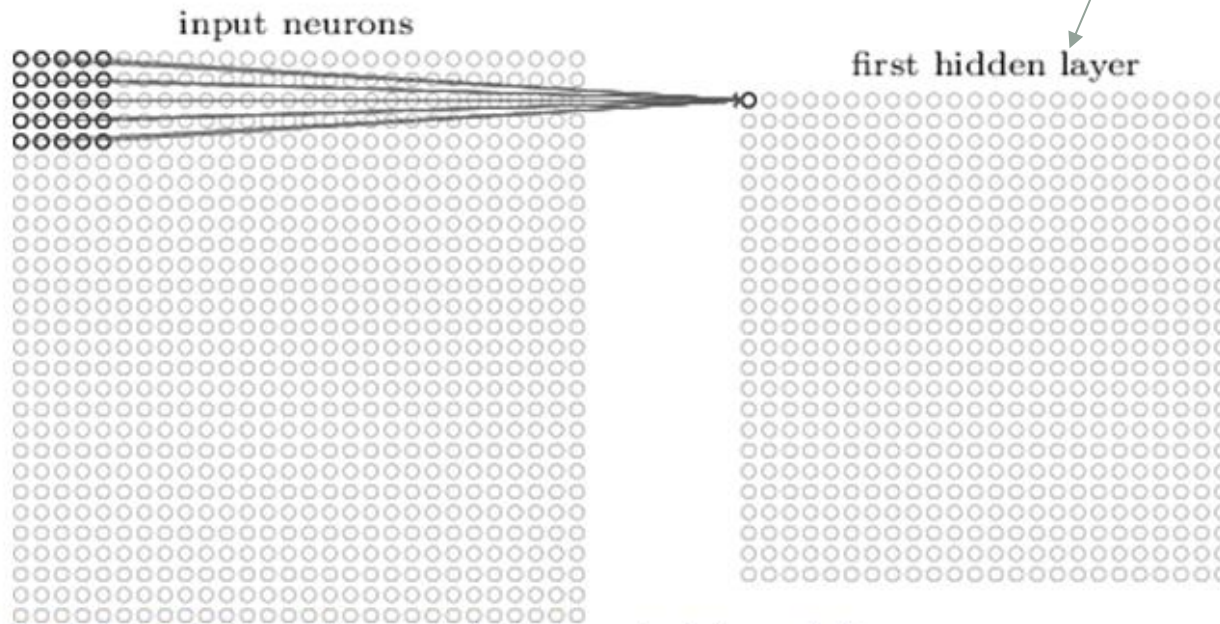
6		

$$\begin{aligned} &7 \times 1 + 4 \times 1 + 3 \times 1 + \\ &2 \times 0 + 5 \times 0 + 3 \times 0 + \\ &3 \times -1 + 3 \times -1 + 2 \times -1 \\ &= 6 \end{aligned}$$

# A convolution

- Input:  $32 \times 32 \times 1 \rightarrow \text{Conv} (5.5) \rightarrow 28 \times 28 \times 1$
- $32 - 2 \times 2 = 28$

Activation map  
Feature map  
Feature Image



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

# 0 padding

Sometime loosing resolution is painful !  
 Virtually increase the size of the image with 0

5x5 → conv 3x3 and 0 padding → 5x5

0	0	0	0	0	0
0					0
0					0
0					0
0					0
0					0
0	0	0	0	0	0

a) A zero-padding with size 1

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0					0	0
0	0					0	0
0	0					0	0
0	0					0	0
0	0					0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

b) A zero-padding with size 2

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

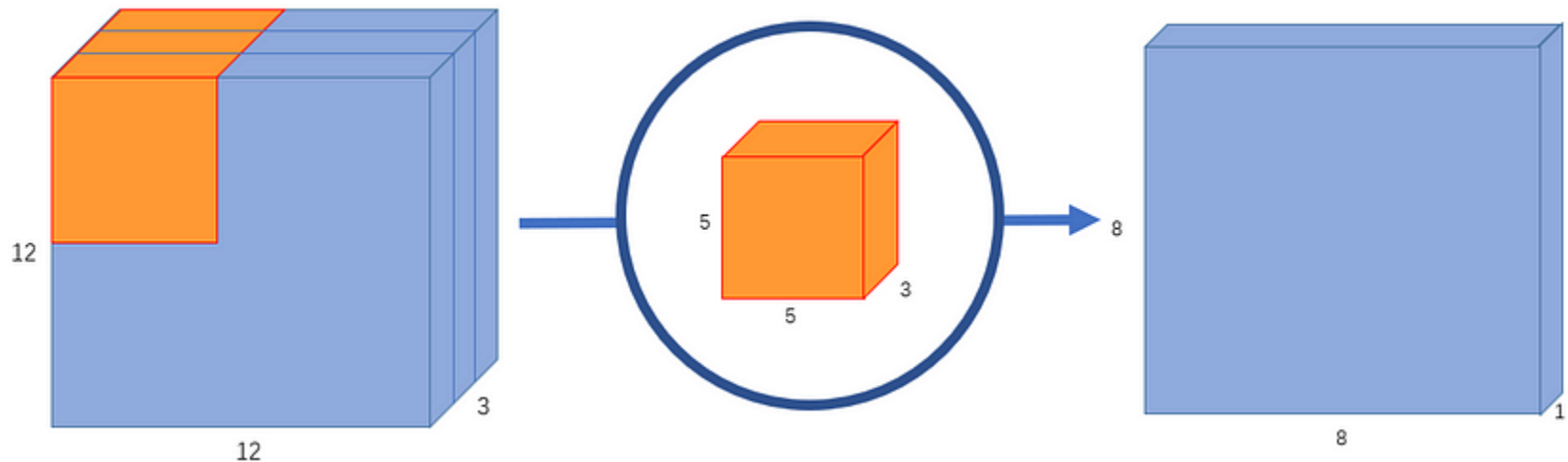
Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

# CNN with RGB images

- With RGB images: input are  $W \times H \times 3$
- Example
  - Input:  $12 \times 12 \times 3 \rightarrow \text{Conv}(5, 5, 3) \rightarrow 8 \times 8 \times 1$
  - The convolution is done on the 3 channels: **filter  $5 \times 5 \times 3$**

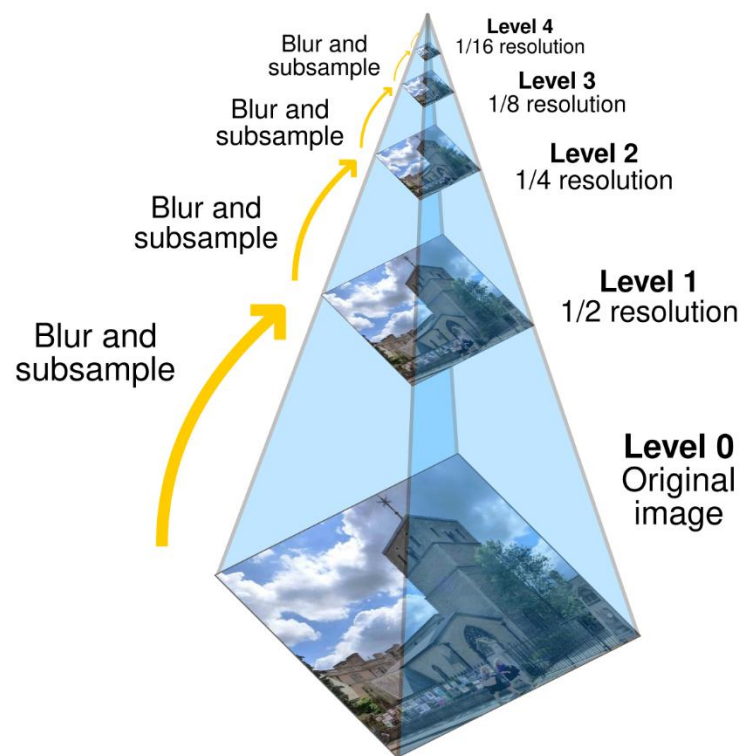


# Pyramid of images

- Before ConvNN, multi-resolutions were often used
  - Detecting different features



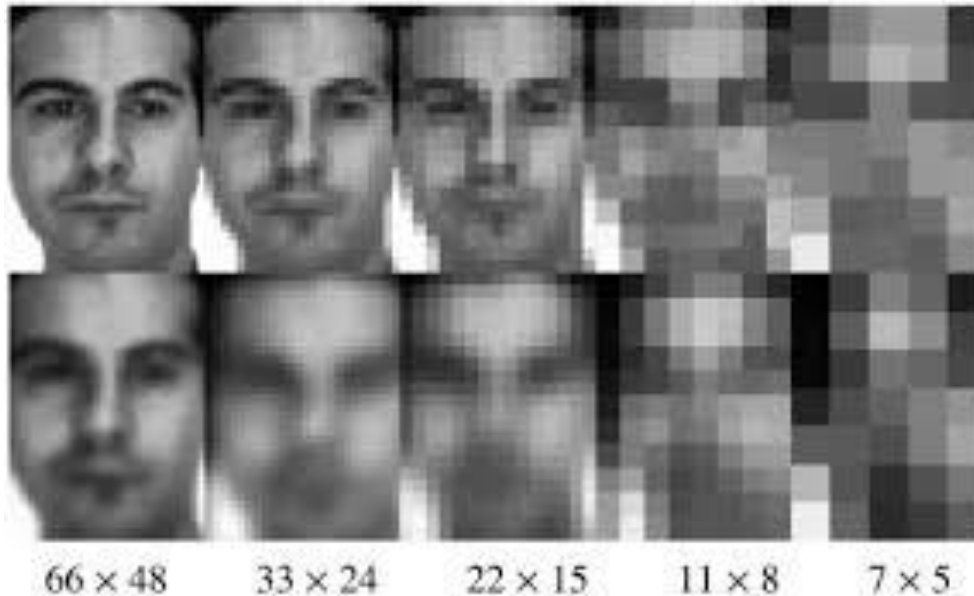
Intuitive idea: image = shape+detail



Pyramid of images

# Pyramid of images

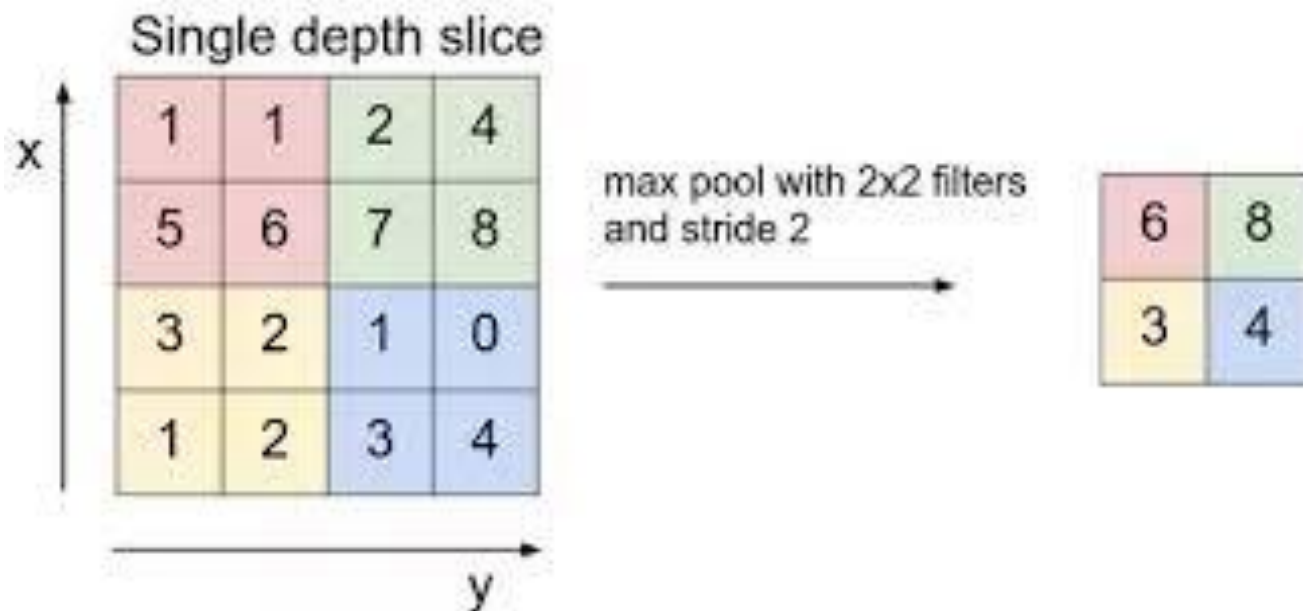
- Before ConvNN, multi-resolution were often useful
  - Detecting different features
  - Contents / shapes ..... small features / texture



# Pyramid of images → Max Pooling

- Subsampling: reduction of dimensions
  - Pooling is the easiest operation invented (re-use for CNN)
  - Remember: have to be applied on many images (dataset + conv)

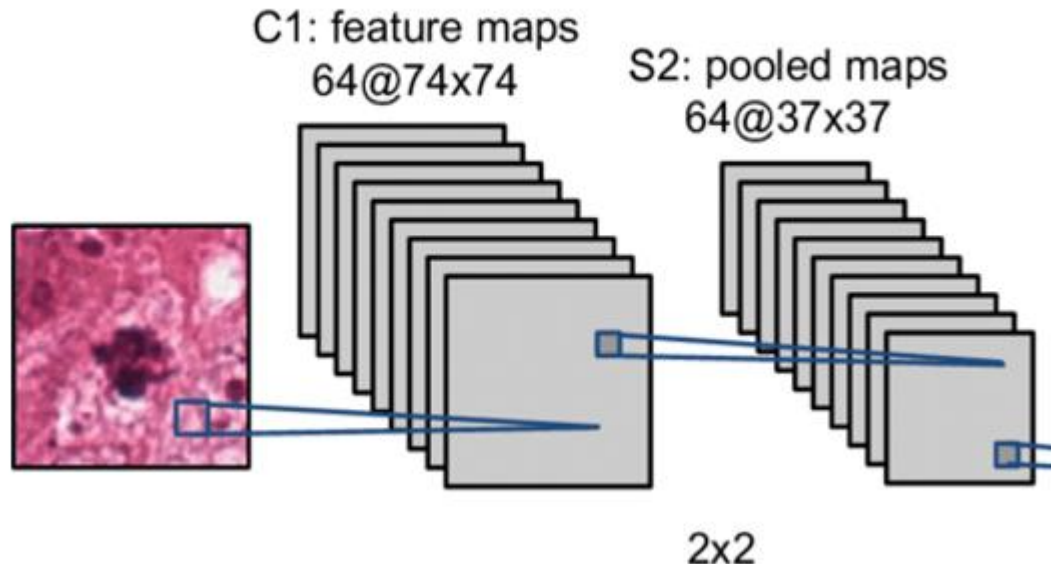
## Example of pooling 2x2



# CNN: one step = Conv + MaxPooling

Input: 1 image

- One layer of CNN  $\rightarrow$  N images (N=number of conv)
- One layer of MaxPooling (2x2)  $\rightarrow$  N images with half resolution



# CNN: idea of pyramid

Input: 1 image  $W \times H$

- One layer of CNN  $\rightarrow$   $N$  images  $W \times H$  ( $N$ =number of conv, 0 padding)
- One layer of MaxPooling (2x2)  $\rightarrow$   $N$  images with half resolution
  - regouped in one tensor ( $N, W/2, H/2$ )

**Again**, input:  $(N, W/2, H/2)$

- One layer of CNN  $\rightarrow$   $M$  images ( $M$ =number of conv)
  - For instance, filter size =  $(N, 3, 3)$  (3D filter, remember RGB)
- One layer of MaxPooling (2x2)  $\rightarrow$  images with half resolution
  - Regrouped in one tensor  $(M, W/4, H/4)$

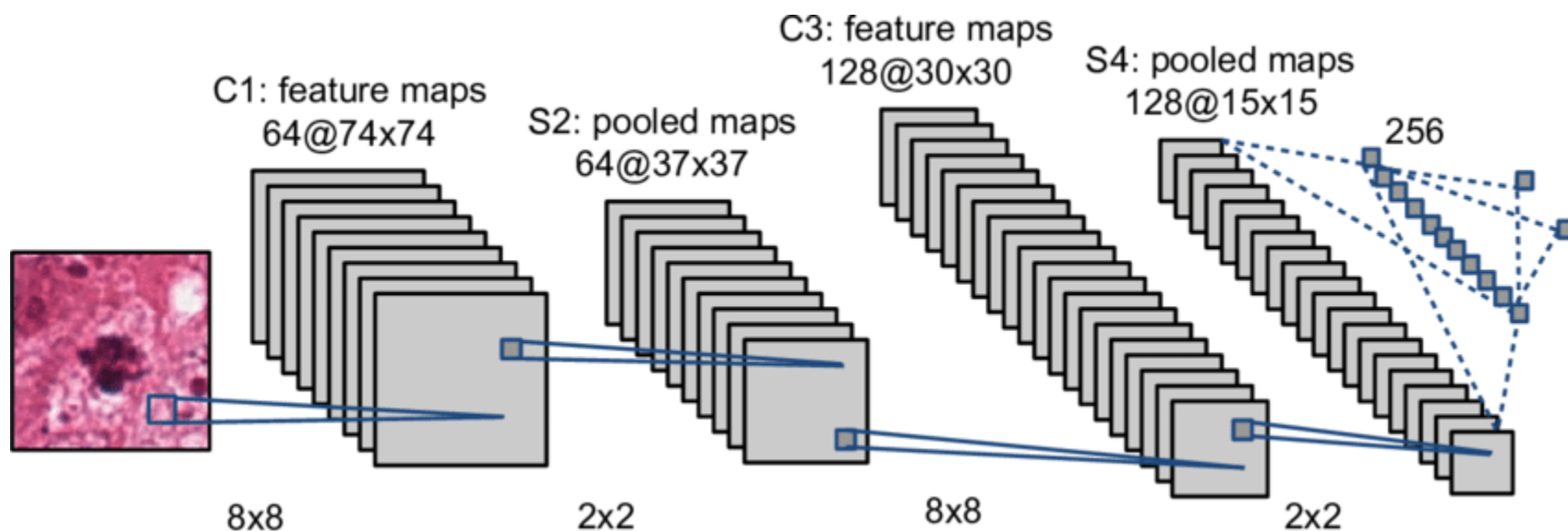
And **classification**

# Convolutional Network ConvNET

Recognizing an object with a convolutional network

Full network with 2 or 3 layers of Convolution

→ good results (in 2000)



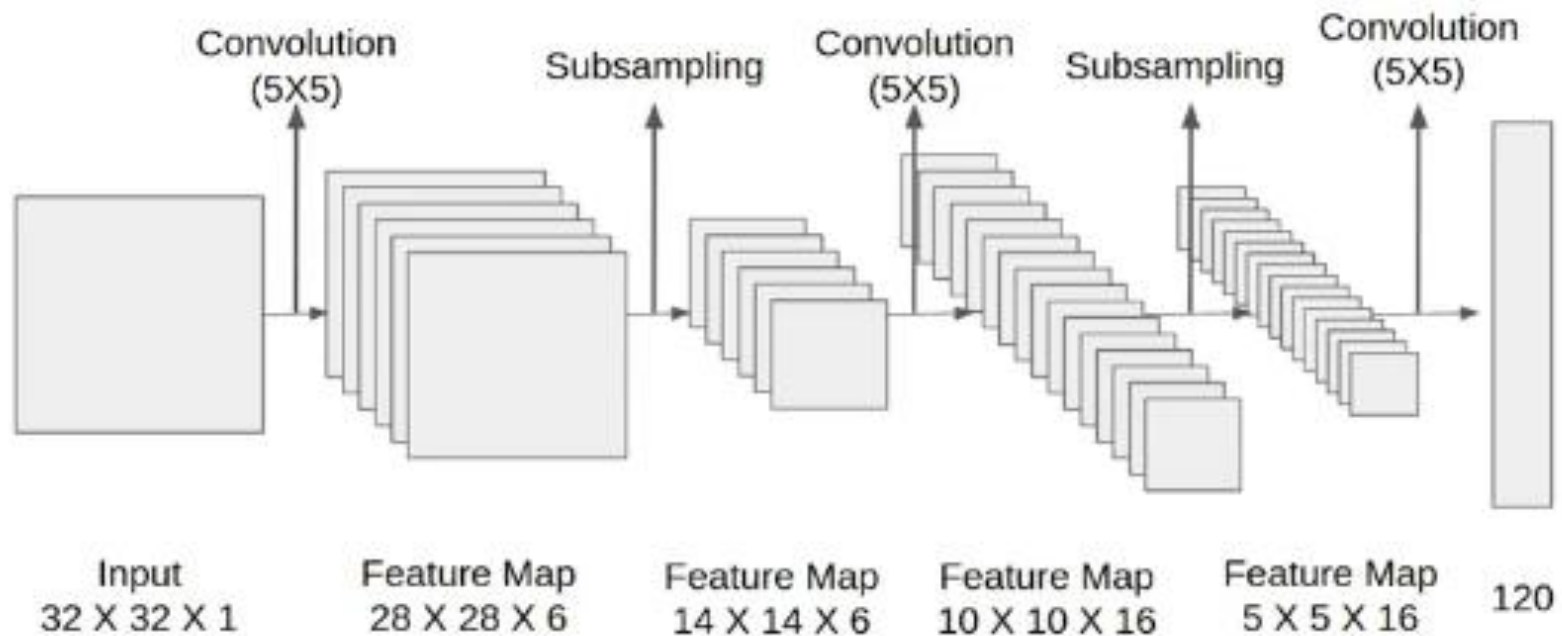
## First good point

Network with 2 or 3  
layers of Convolution

LeNet-5 (Yann LeCun et al., 1998)

# CNN example

- A good network to start the deep learning on images  
(without padding here)



# With code (python)

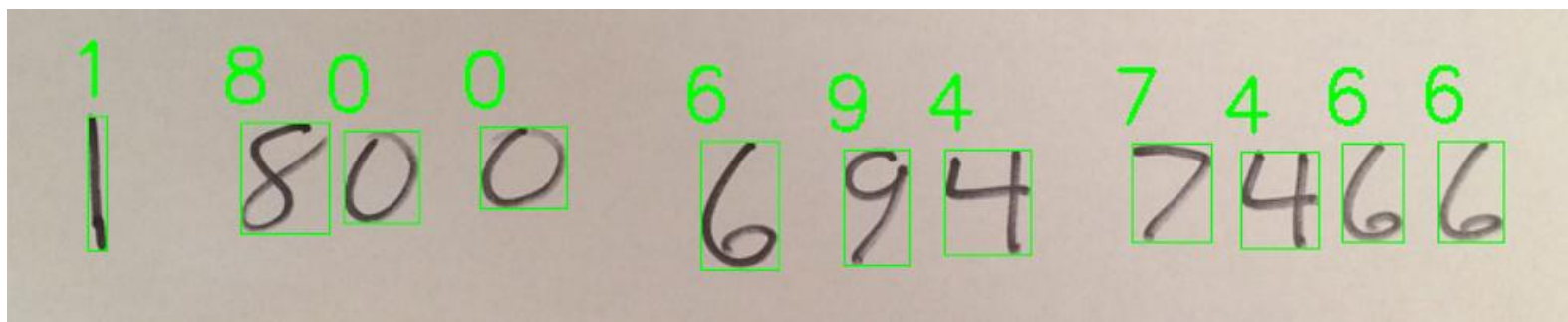
- Input : RGB images, 3 channels
- Conv: 32 conv to be learned of size 3x3, padding 1
- RELU
- Conv, input 32 channels, 64 conv to be learned of size 3x3, padding 1
- RELU

```
import torch.nn as nn

block = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.ReLU()
)
```

# Handwriting recognition

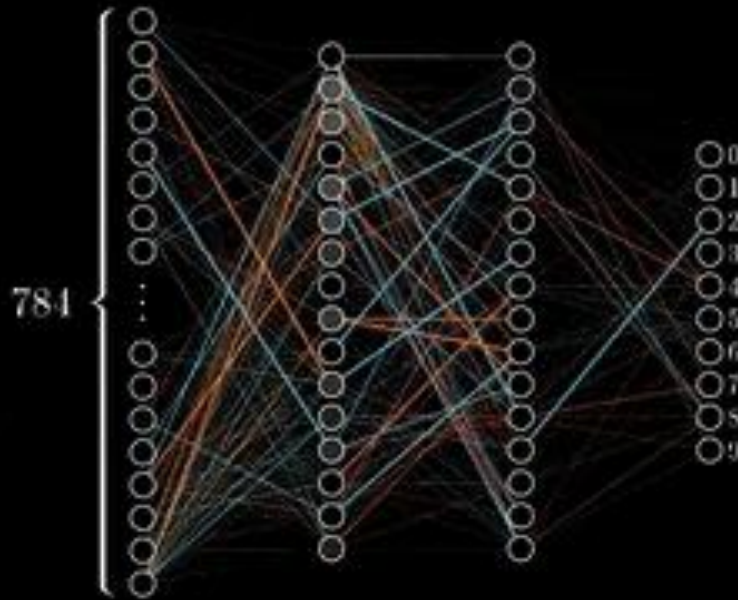
- Handwriting recognition
  - “La Poste”: postal codes on envelopes
  - Writing on tablet/mobile
  - The classic problem for learning CNNs
  - Handwritten digits database: MNIST
- On MNIST, good recognition rate
  - Fully Connected Network: 90%
  - With a ConvNET (CNN) > 99% and less weight to learn



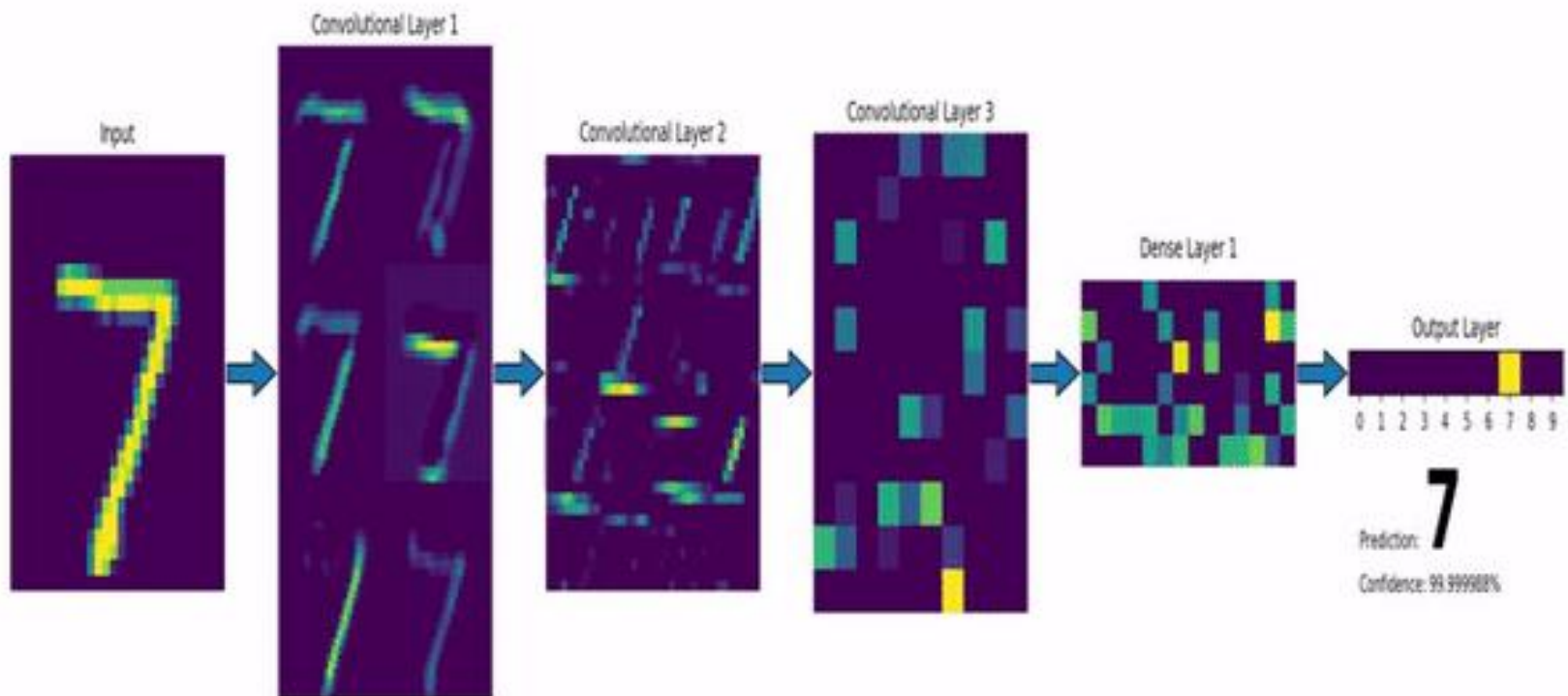
# Training (fully connected)

Training in  
progress...

$q \rightarrow 9$

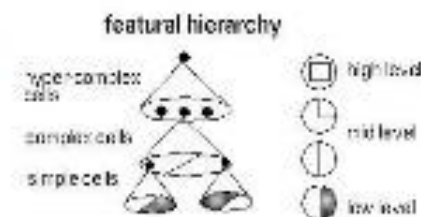
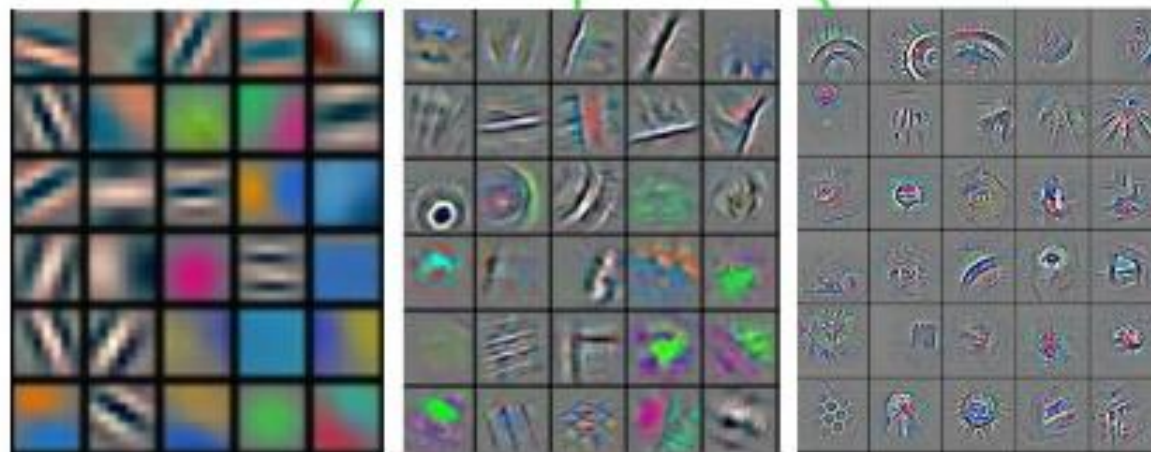


# Training: CNN example



# Convolutional Network ConvNET

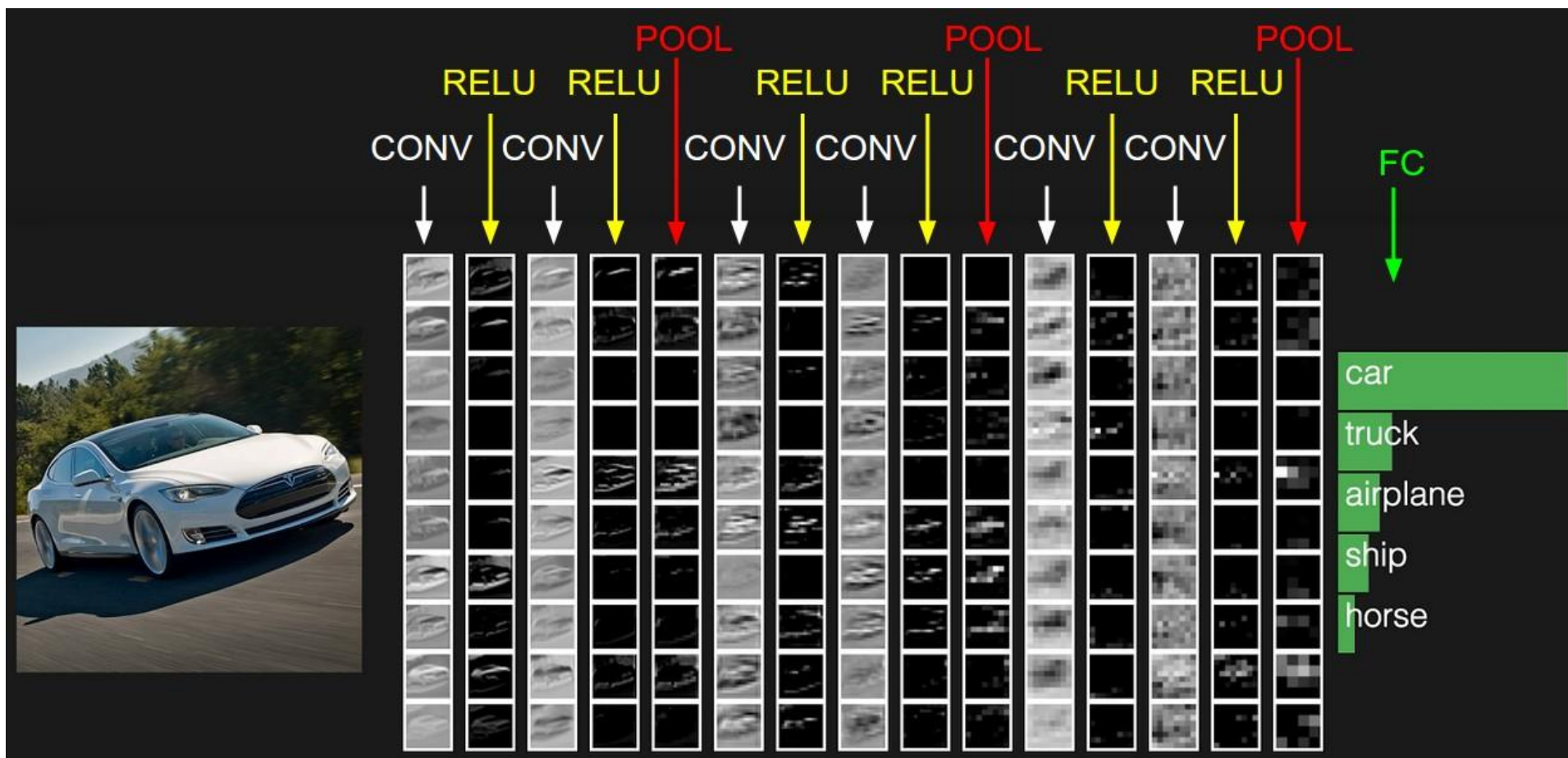
## ConvNet : Interpretation



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

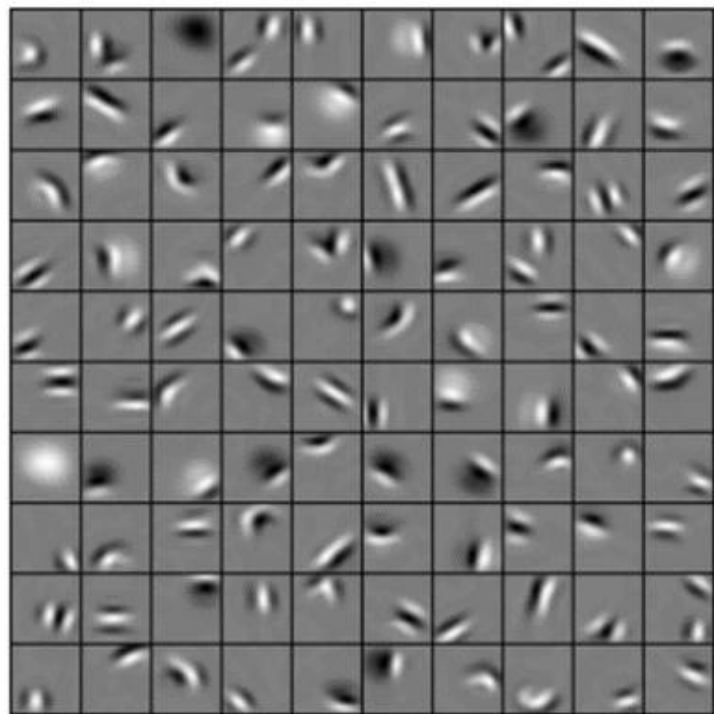
# Convolutional Network ConvNET

- Add a RELU after each convolution (IMPORTANT!!!)



# ConvNET: inspect layers

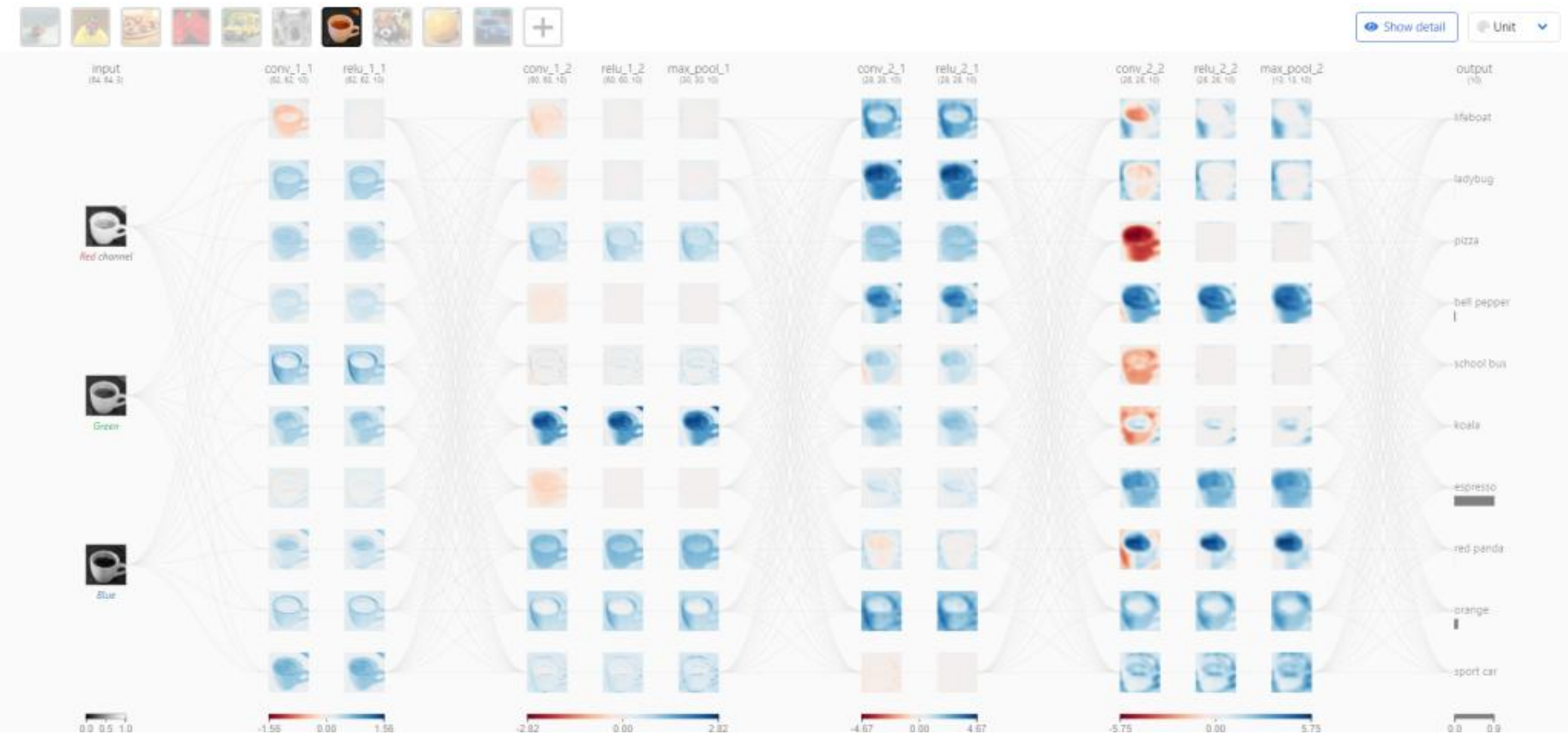
- Example of features (filters) produced by the network in the hidden layers



Early layers Deep layers

# CNN demo

- <https://poloclub.github.io/cnn-explainer/>



# CNN: code example

```
class Classifier(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.features = nn.Sequential(
            # 3 input image channel, 6 output channels (meaning 6 different convolutions), 5x5 square convolution
            nn.Conv2d(3, 6, 5)
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5)
            # ... to do
            # ...
        )

        self.classifier = nn.Sequential(
            nn.Linear(16 * 5 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10)
        )
        print(self.features)
        print(self.classifier)

    def forward(self, input):
        x = self.features(x)
        x = x.view(x.size(0), -1) # change the view in order to flatten the tensor
        x = self.classifier(x)
```

3 since RGB

6 images as output (6 conv)  
5x5 convolution

6 since previous output is 6  
16 images as output (=16 conv)  
5x5 convolution

16 since previous layer  
If input 32x32

- 1st conv → 28x28
- MaxPool → 14x14
- 2<sup>nd</sup> conv → 10x10
- MaxPool → 5x5

# CNN example : optimization

```
net = Classifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

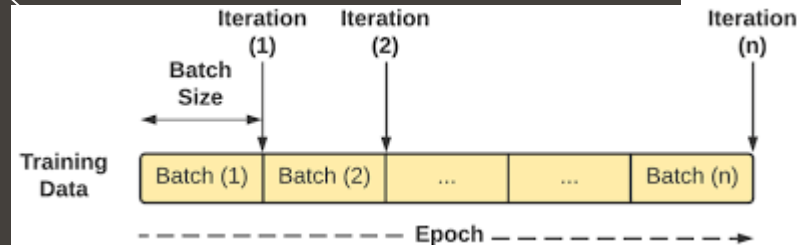
for epoch in range(2): # loop over the dataset multiple times ← Loop on epoch
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0): ←
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

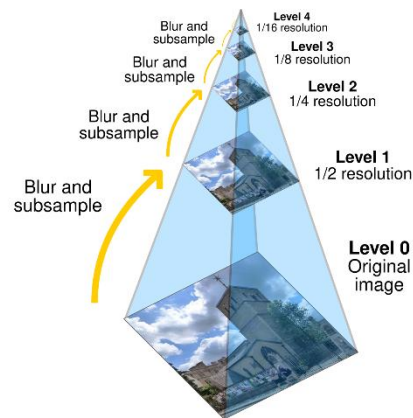
        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999: # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```



# Improving CNN

- Why only 2 or 3 ConvNN ?
  - Remember pyramid



- AlexNet (Krizhevsky et al., 2012)
    - 5 layers of conv + 3 layers fully connected
    - ReLU
    - Training on GPU
    - *dropout*
- ➔ DEEP LEARNING become the main tool

# VGG

K. Simonyan , A. Zisserman

[Very Deep Convolutional Networks for Large-Scale Image Recognition](#)

arXiv technical report, 2014

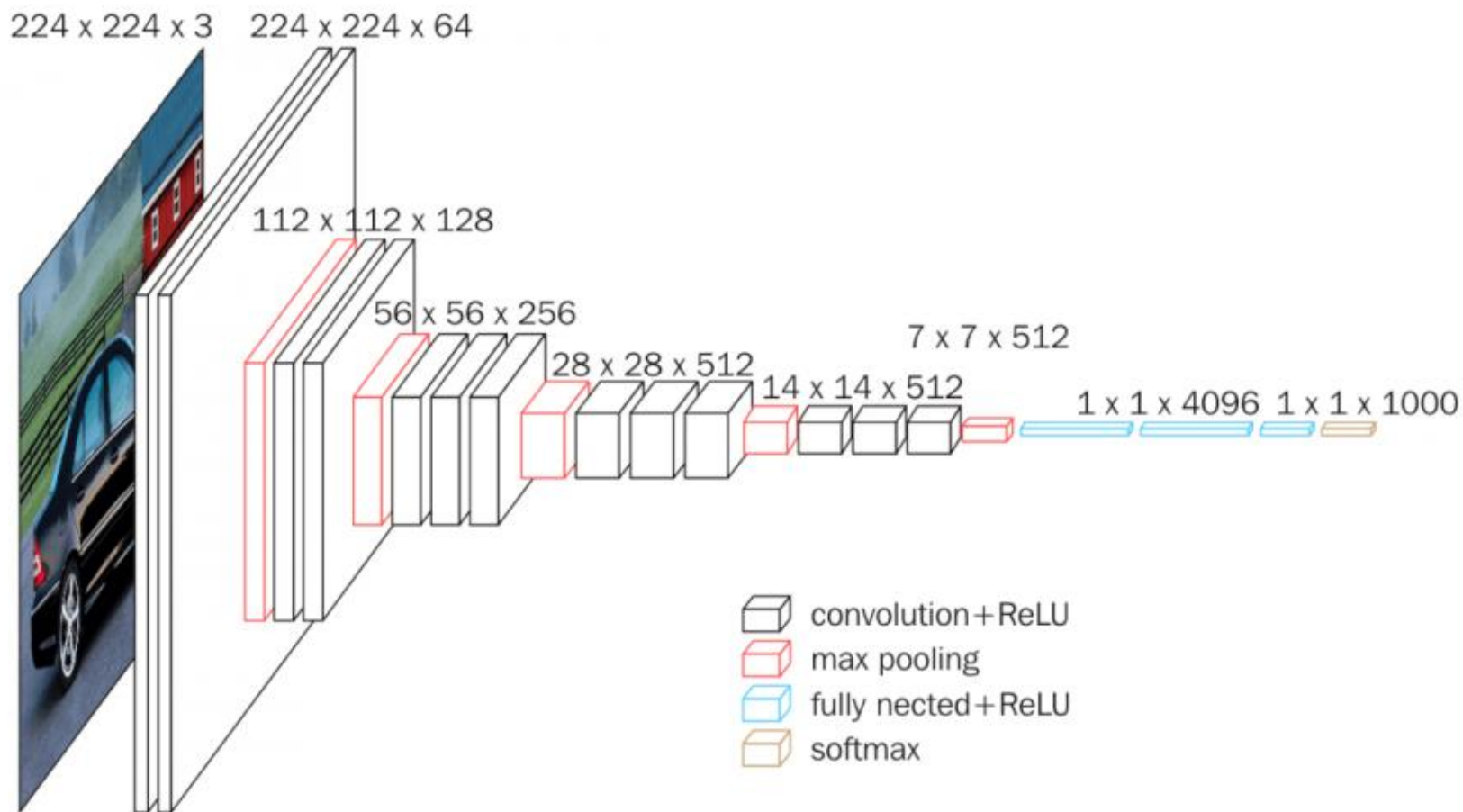
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGG

K. Simonyan , A. Zisserman

[Very Deep Convolutional Networks for Large-Scale Image Recognition](#)

arXiv technical report, 2014



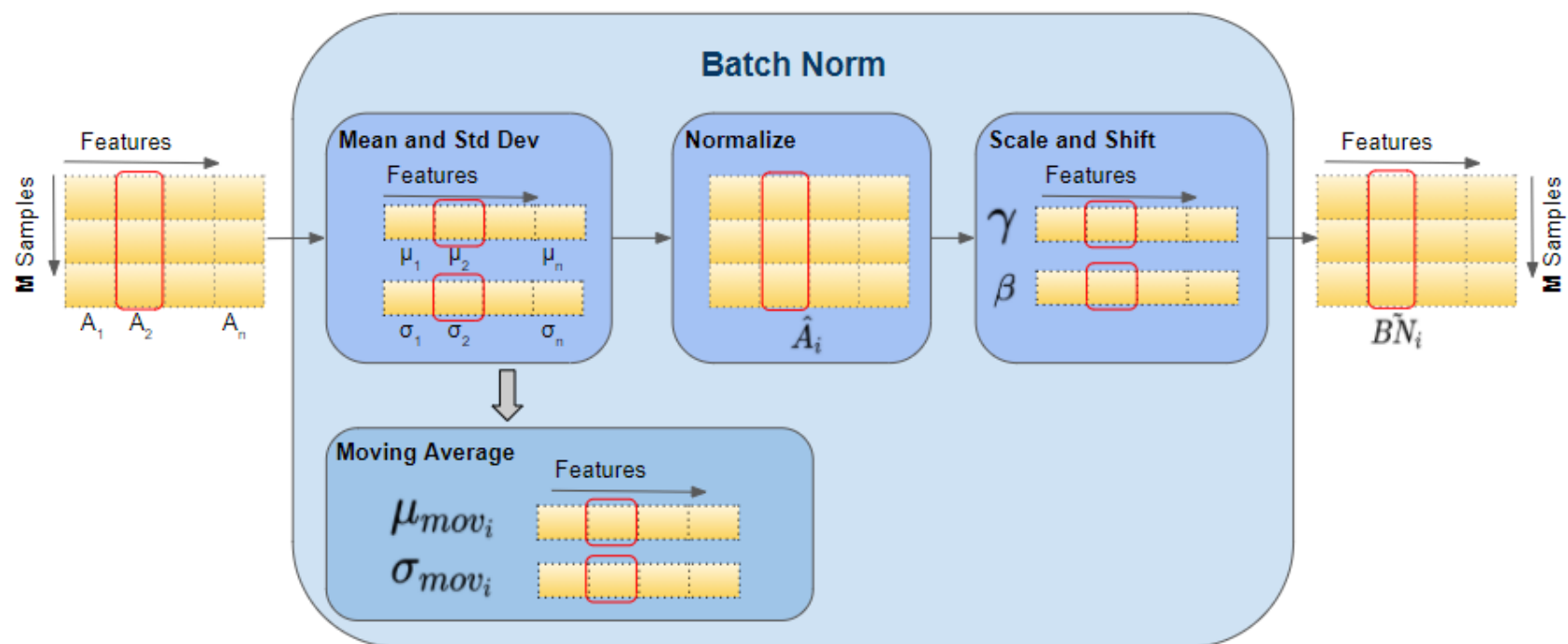
# VGG19: see TP to display layers

```
0 Conv2d(3, 64, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
1 ReLU ( inplace )
2 Conv2d(64, 64, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
3 ReLU ( inplace )
4 MaxPool2d( kernel_size =2, stride=2, padding =0, dilation=1, ceil_mode =False)
5 Conv2d(64, 128, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
6 ReLU ( inplace )
7 Conv2d(128, 128, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
8 ReLU ( inplace )
9 MaxPool2d( kernel_size =2, stride=2, padding =0, dilation=1, ceil_mode =False)
10 Conv2d(128, 256, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
11 ReLU ( inplace )
12 Conv2d(256, 256, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
13 ReLU ( inplace )
14 Conv2d(256, 256, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
15 ReLU ( inplace )
16 Conv2d(256, 256, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
17 ReLU ( inplace )
18 MaxPool2d( kernel_size =2, stride=2, padding =0, dilation=1, ceil_mode =False)
19 Conv2d(256, 512, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
20 ReLU ( inplace )
21 Conv2d(512, 512, kernel_size =(3, 3), stride=(1, 1), padding =(1, 1))
22 ReLU ( inplace )
```

# Batch Normalization

- For epoch
  - For batch
    - Normalize the batch

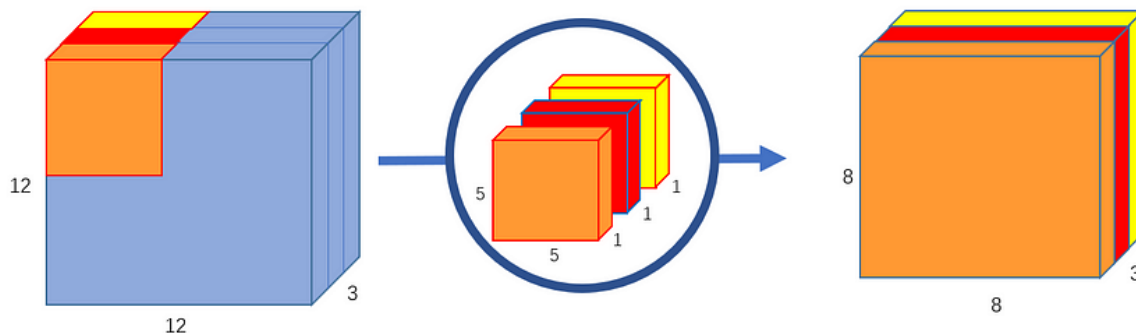
→ help a lot



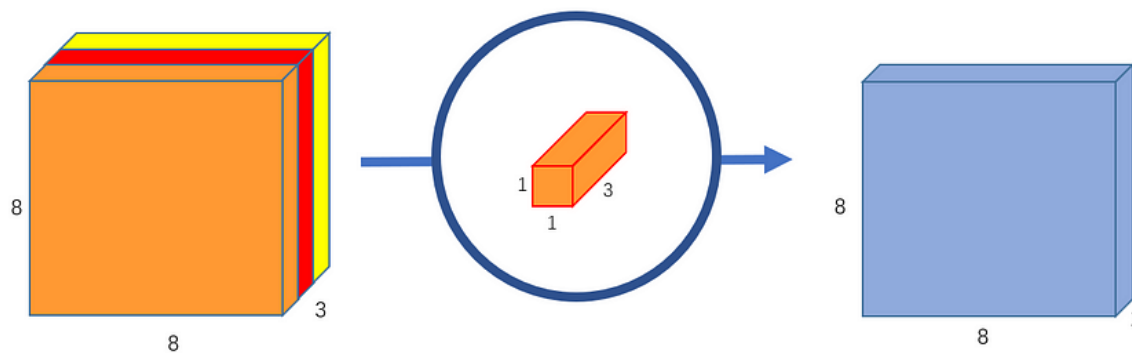
# Depthwise separable convolutions

- Recall:  $12 \times 12 \times 3 \rightarrow \text{CNN}(5,5) \rightarrow 8 \times 8 \times 1$ 
  - The RGB are “merged” with convolution into 1 value
- Depthwise separable convolutions into two steps

- 3 convolutions  
R, G, B



- Pointwise Convolution
  - Merge R, G, B



# Depthwise separable convolutions

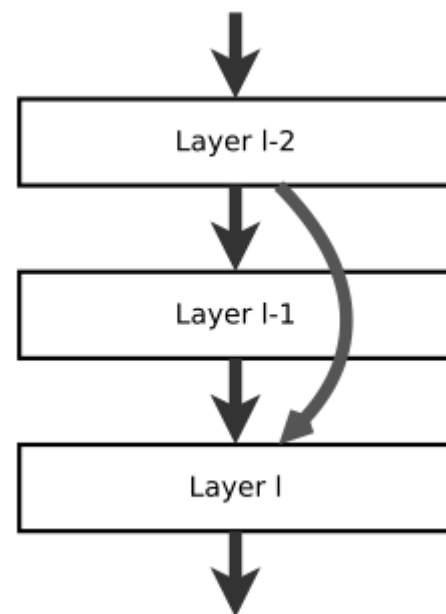
- Advantages ?
  - Increase performance for the same number of parameters
  - See Xception compared to Inception V3

	<b>Top-1 accuracy</b>	<b>Top-5 accuracy</b>
<b>VGG-16</b>	0.715	0.901
<b>ResNet-152</b>	0.770	0.933
<b>Inception V3</b>	0.782	0.941
<b>Xception</b>	<b>0.790</b>	<b>0.945</b>

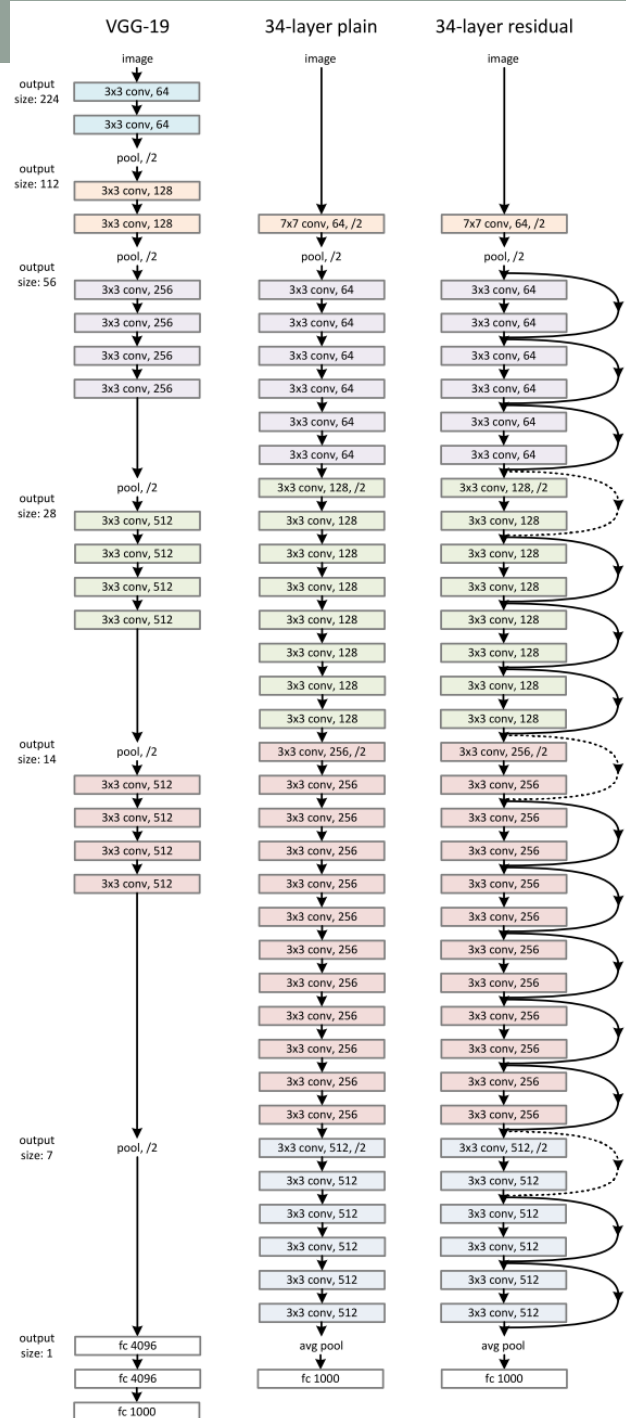
# Residual neural network: ResNet

For very deep network ( $> 50$  layers)

- Connection jumps or “shortcuts” are used to jump over certain layers
- **ResNet**
  - double- or triple-layer jump (ReLU+batch norm)
  - avoid the problem of gradient vanishing
  - stabilizes the training
  - avoid precision saturation
- Models with several parallel jumps are called **DenseNets** (better information transfer)



# ResNET-50



# Recently

- EfficientNet
  - resizing models by automatically balancing network width, depth and resolution → design smaller but high-performance models, optimizing the use of computing resources
- Vision Transformers (ViT, 2020)
  - ViT divides an image into parcels and treats them as an input sequence to a transformer
- Swin Transformer (2021)
  - introducing local windows that slide across the image+transformer enabling the model to capture dependencies at different scales

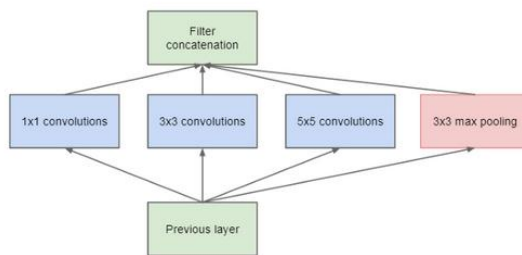
# Recently

- Hybrid Architectures
  - Combine CNNs and transformers: CONFORMER
  
- Meta-learning et Few-Shot Learning
  - MetaFormer Baselines for Vision (TPAMI 2024)  
<https://github.com/sail-sg/metaformer>

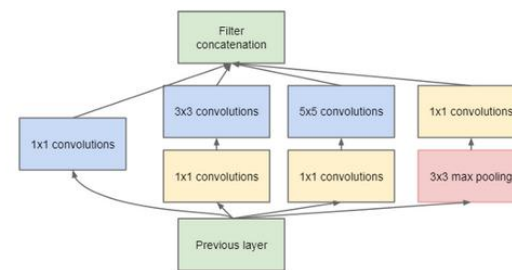
# Classification networks

Often used in pre-processing as “image processing”

- VGG
  - VGG16, VGG19
  - Series of Conv , Max- pooling , and activation, then fully-connected (FC)
- ResNet50
  - Great depth
- Inception v3
  - Wide
- Exception
  - extreme inception
- MobileNet
  - Optimized for mobile






(a) Inception module, naïve version



(b) Inception module with dimension reductions

- In KERAS / PyTorch these models are pre-trained

# The data

	VGGNet	DeepVideo	GNMT
Used For	Identifying Image Category	Identifying Video Category	Translation
Input	Image 	Video 	English Text 
Output	1000 Categories	47 Categories	French Text
Parameters	140M	~100M	380M
Data Size	1.2M Images with assigned Category	1.1M Videos with assigned Category	6M Sentence Pairs, 340M Words
Dataset	ILSVRC-2012	Sports-1M	WMT'14

Number of parameters (in millions), for popular neural networks.

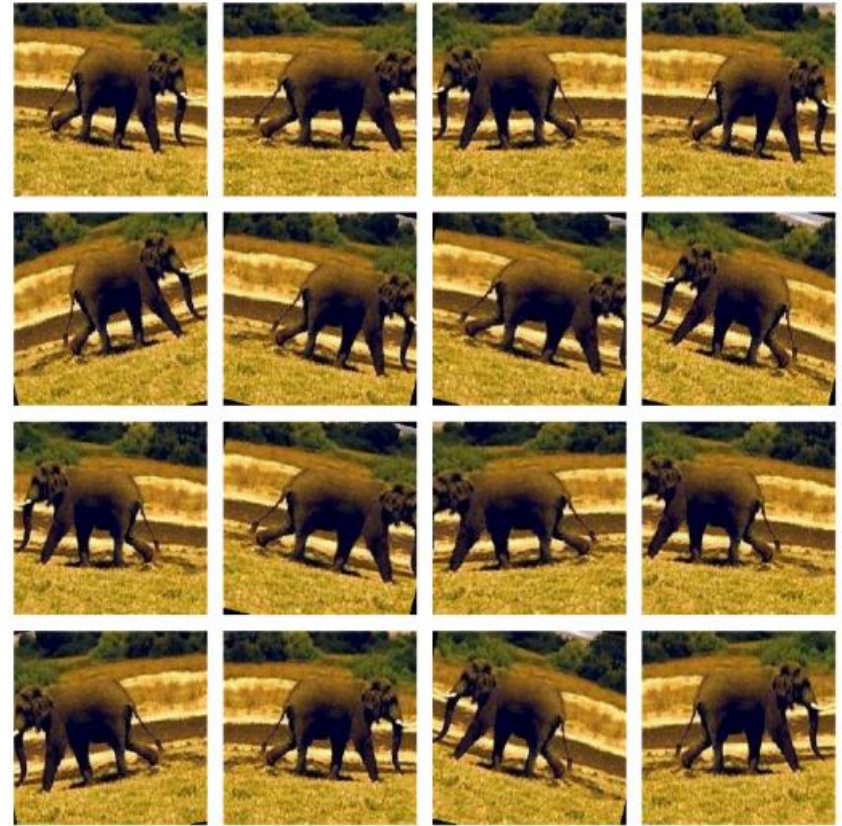
# Problem: lack of data for DL

- Size of some databases
  - MNIST: handwritten numbers, 70,000 images
  - EMNIST: handwritten letters
    - Balanced 134000 images
    - Otherwise, 814000 images
  - ImageNET : 14 million images with 20,000 labels
  - CelebFaces ( CelebA ): 202,599 face images of 10,177 celebrities
- For many other problems
  - Cohn Kanade : 486 videos of 97 people expressing an expression
  - ...
- Branches of machine learning that seek to work with less data
  - More efficient DL: GAN, etc.
  - One-shot learning / Few-shot learning

# Data augmentation

## The base

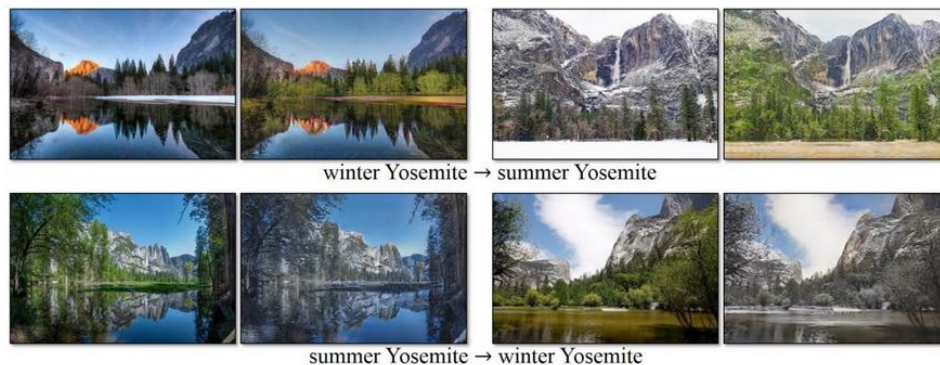
- Flip
- Rotation
- Scale
- Crop
- Translation
- Gaussian Noise



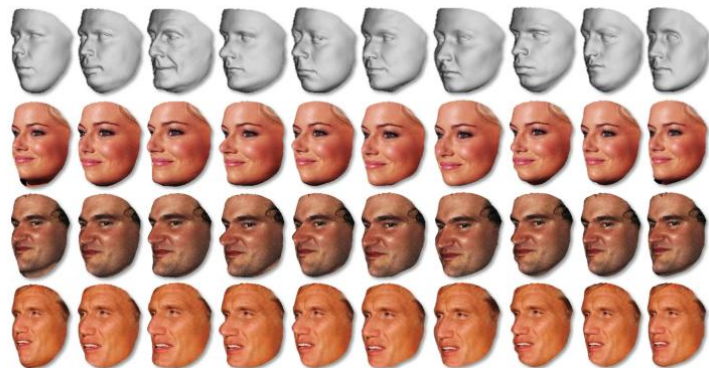
# Data augmentation

A little further

- GAN to transform images (color palette, style, etc.)



- Data from computer-generated images



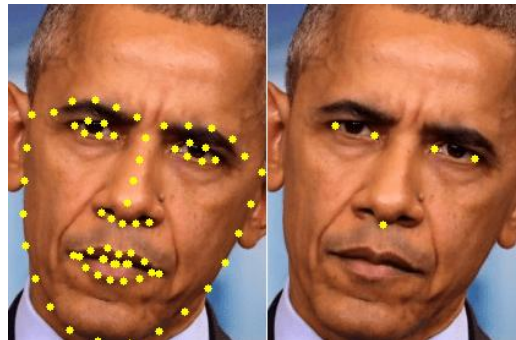
Do We Really Need to Collect Millions of Faces for Effective Face Recognition?

<https://arxiv.org/pdf/1603.07057.pdf>

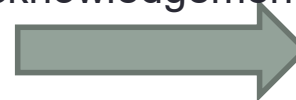
# Data augmentation: helping the network with additional data

For example: corpus of images/video of faces

- DL on images only, possible but...
  - Possibility of extracting points of the face by detection:  
corners of the mouth, nose, eyes
    - DL on characteristic points + images
- Much more efficient!!!



acknowledgement



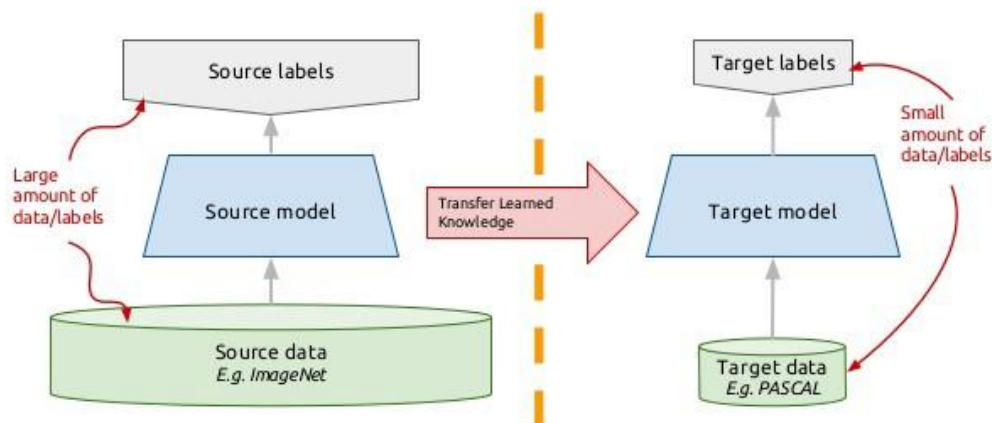
Expressions  
Name of the person  
Etc.

# Transfer learning with CNN

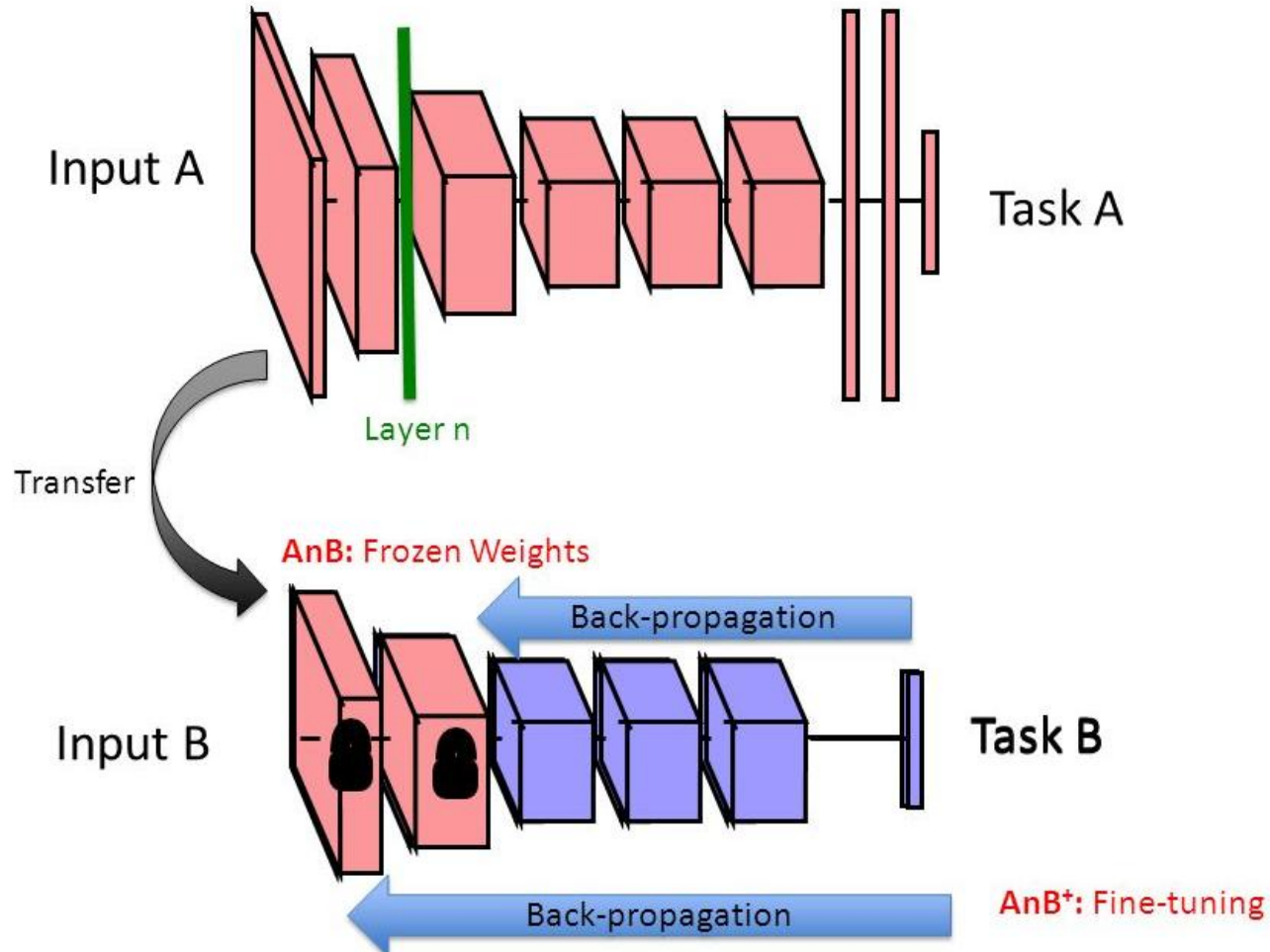
Simple approach to specializing

- Model pre-trained on a large, fairly general database
- Freeze certain parameters (weights): low convolution layers
- Adds “classifier” layers with its parameters to train on the specific data
- Train this network on specific data
- Optionally unfreeze all settings at the end

## Transfer learning: idea



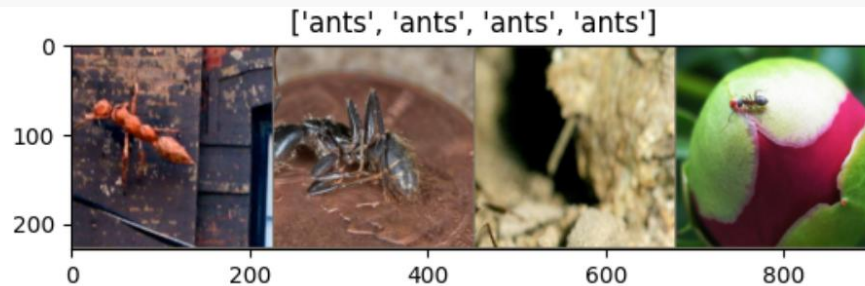
# Transfer learning with CNN



# Transfer learning with CNN

- An example with PyTorch

```
model_conv = torchvision.models.resnet18(weights='IMAGENET1K_V1')
for param in model_conv.parameters():
    param.requires_grad = False
```

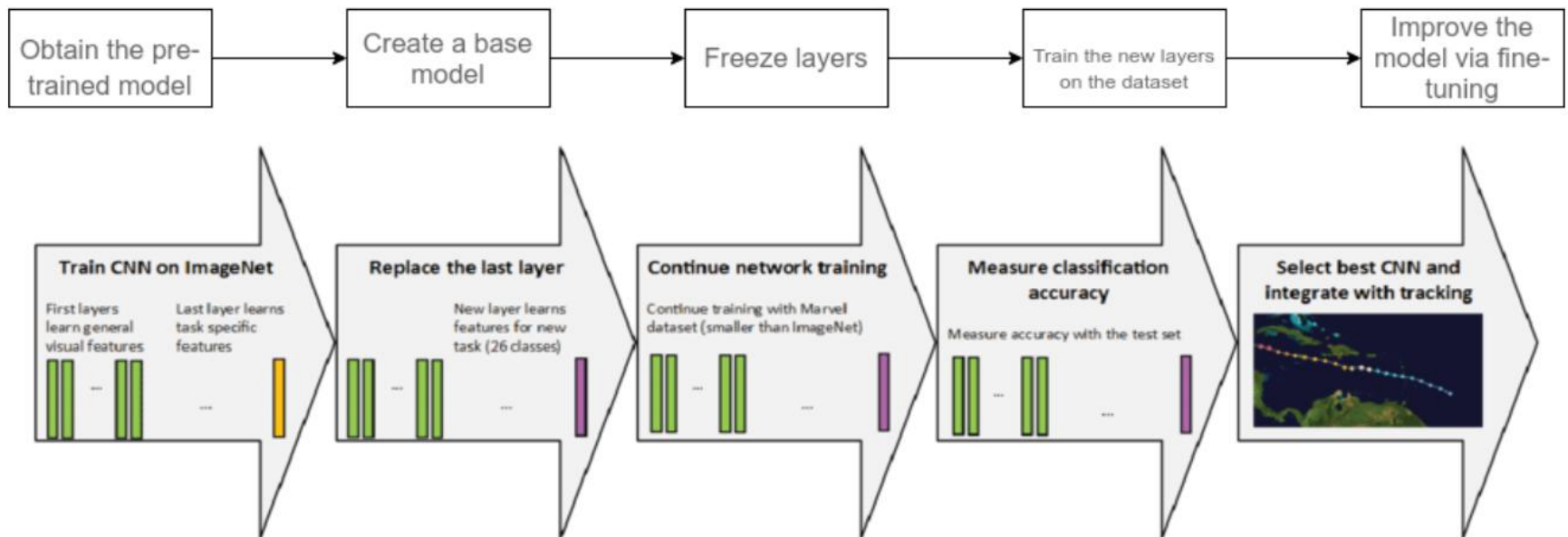


- Want to classify **ants** and **bees** with about 120 training images each for ants and bees
  - Usually, this is a very small dataset if trained from scratch.  
→ transfer learning, generalize reasonably well.

# Transfer learning with CNN

## Transfer learning (freezing, fine-tuning, ...)

- A vast domain
- Adapting learning is one of the strong areas of current research ...



How transferable are features in deep neural networks?  
[Jason Yosinski](#), [Jeff Clune](#), [Yoshua Bengio](#), [Hod Lipson](#)  
NIPS2014 (initial paper... → >10k citations)

# Advantages of this “ deep ” evolution

We take advantage

- Framework
  - GPU, Optimizer, etc.
  - Learning in python but after using in C++, C#, Java, etc.
  - Standard file format: Open Neural Network Exchange
- Large community
  - Many tutorials and explanations
    - [medium.com](https://medium.com)
    - [letslearnai.com](https://letslearnai.com)
    - Etc.
- Reproducible research ( Github )

# PRACTICAL WITH CNN

---

- Classification using CNN
- Apply CNNs to transfer the style of one image to another

# CNN example

```
class Classifier(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.features = nn.Sequential(
            # 3 input image channel, 6 output channels (meaning 6 different convolutions), 5x5 square convolution
            nn.Conv2d(3, 6, 5)
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5)

            # ... to do
            # ...
        )

        self.classifier = nn.Sequential(
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10)
        )

        print(self.features)
        print(self.classifier)

    def forward(self, input):
        x = self.features(x)
        x = x.view(x.size(0), -1) # change the view in order to flatten the tensor
        x = self.classifier(x)
```

3 since RGB  
6 images as output  
5x5 convolution

6 since previous output is 6  
16 images as output  
5x5 convolution

16 since previous layer  
If input 32x32

- 1st conv → 28x28
- MaxPool → 14x14
- 2<sup>nd</sup> conv → 10x10
- MaxPool → 5x5

# CNN example : optimization

```
net = Classifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

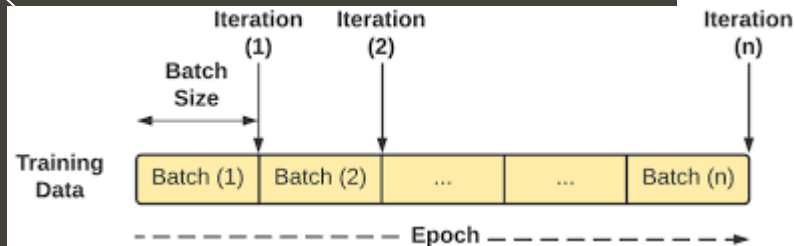
for epoch in range(2): # loop over the dataset multiple times ← Loop on epoch
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0): ←
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999: # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```



# Padding

- Padding = add borders around the input image before applying convolution.
  - Preserve image size after convolution.
  - Avoid loss of information at image edges
- Padding types
  - **Valid (or no padding)**: no borders added, reducing image size.
  - **Same (or with padding)**: borders are added so that the image size remains the same after convolution.
  - **Custom padding**: the user can specify the exact number of pixels to be added

# Padding

- Image 5×5 filter 3×3
  - stride = 1 padding = 1

- Before padding

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

- After padding

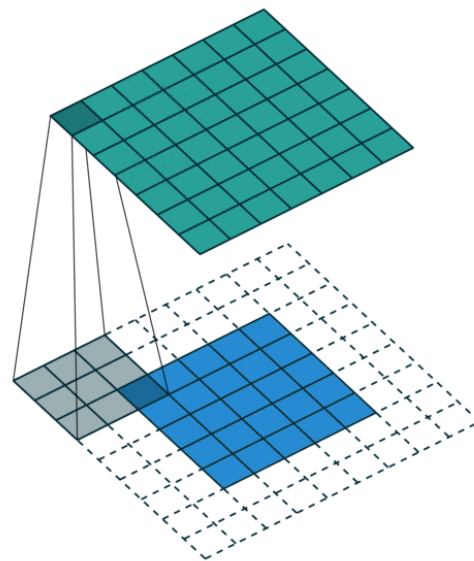
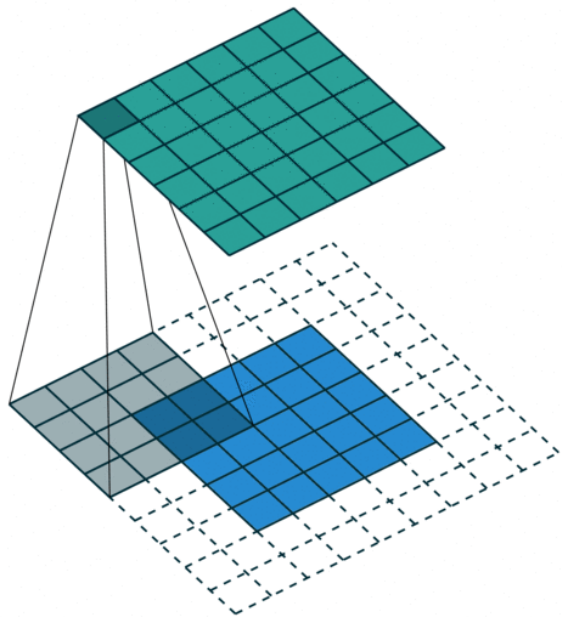
```
0 0 0 0 0 0 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 0 0 0 0 0 0
```

- Filter 3×3 does not reduce the image size

# Padding for enlarging an image

Padding: used to enlarge an image (in a decoder, for example)

- Left a 5x5 image, padding of 2 and Conv 4x4 becomes 6x6
- Right an image 5x5, padding 2 and Conv 3x3 becomes 7x7



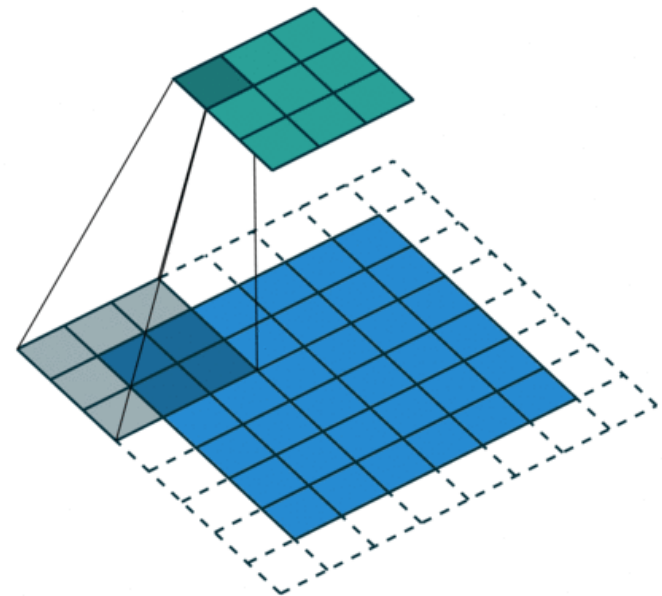
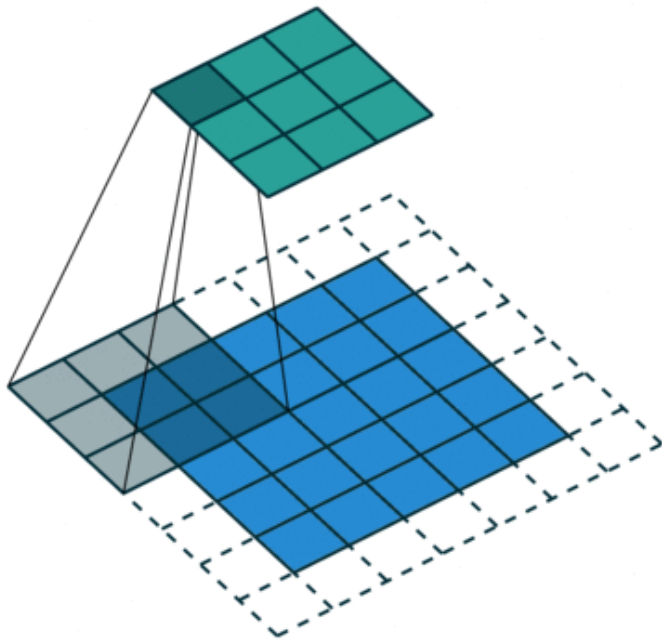
Input : blue    output :green

# Stride (pas)

- The stride is the number of pixels between each convolution filter (displacement at each step)
  - Stride of 1: the filter moves one pixel at a time (classic case)
  - Stride of 2: the filter moves two pixels at a time, reducing the size of the output.
  - Etc.
- **In short, padding helps control the size of the image after convolution, while stride controls the distance the filter moves.**

# Stride

- Left: image 5x5, stride of 1, padding of 1  $\rightarrow$  image 3x3
- Right: image 6x6, stride of 2, padding of 1  $\rightarrow$  image 3x3

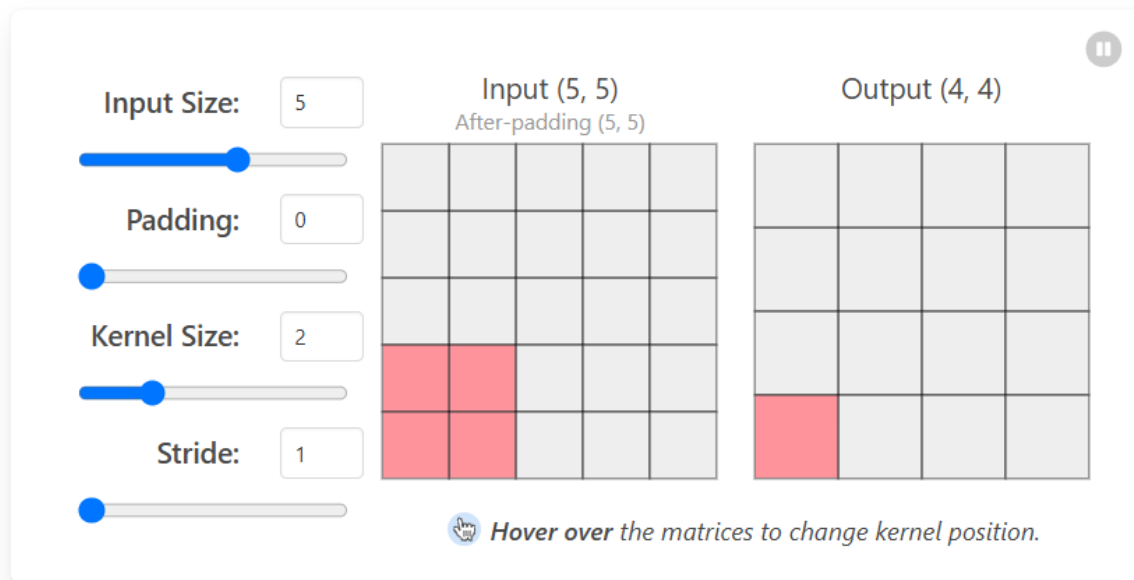


Input : blue    output : green

# Stride, padding: test online

- <https://poloclub.github.io/cnn-explainer/>

## Understanding Hyperparameters



The screenshot displays the CNN Explainer interface with the following settings and visualizations:

- Input Size:** 5
- Padding:** 0
- Kernel Size:** 2
- Stride:** 1

The interface shows two matrices:

- Input (5, 5) After-padding (5, 5):** A 5x5 grid where the bottom-left 2x2 cells are highlighted in red, representing the current kernel position.
- Output (4, 4):** A 4x4 grid where the bottom-left cell is highlighted in red, representing the output of the current kernel operation.

A tooltip at the bottom indicates: *Hover over the matrices to change kernel position.*

# STYLE TRANSFER BETWEEN IMAGES

## 1 Upload photo

The first picture defines the scene you would like to have painted.



## 2 Choose style

Choose among predefined styles or upload your own style image.



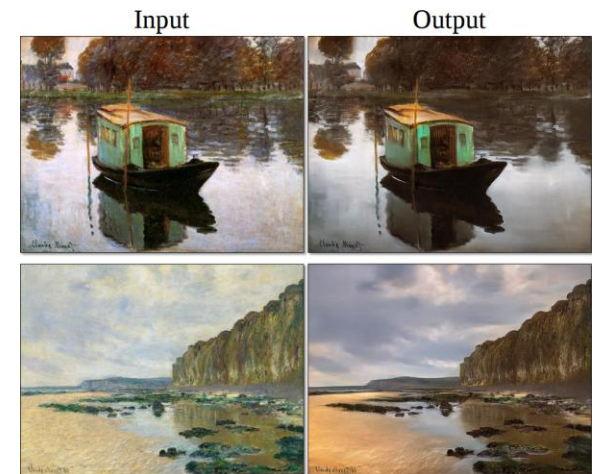
## 3 Submit

Our servers paint the image for you. You get an email when it's done.



# Style transfer between images

- All the qualities of a good TP
  - Using DL Frameworks
  - Optimization
  - Use of networks, but without needing to train them (reasonable calculation time for a practical exercise)



# Style transfer between images

- Differentiate
  - Image content: objects and their places/positions/orientations
  - Style : color and textures
- VGG19 for extracting features
  - Each convolution layer will produce a feature map
  - Optimization with two terms:  $\text{Cost\_Content} + \text{Cost\_Style}$



Features at different scales

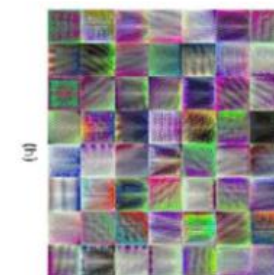
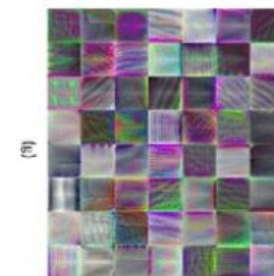
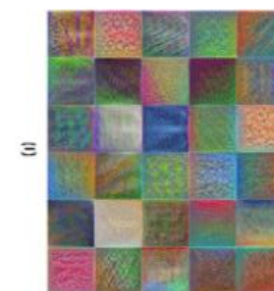
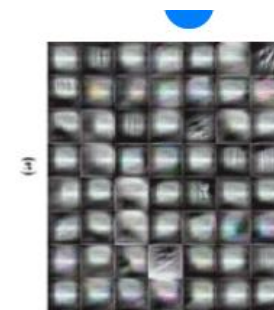
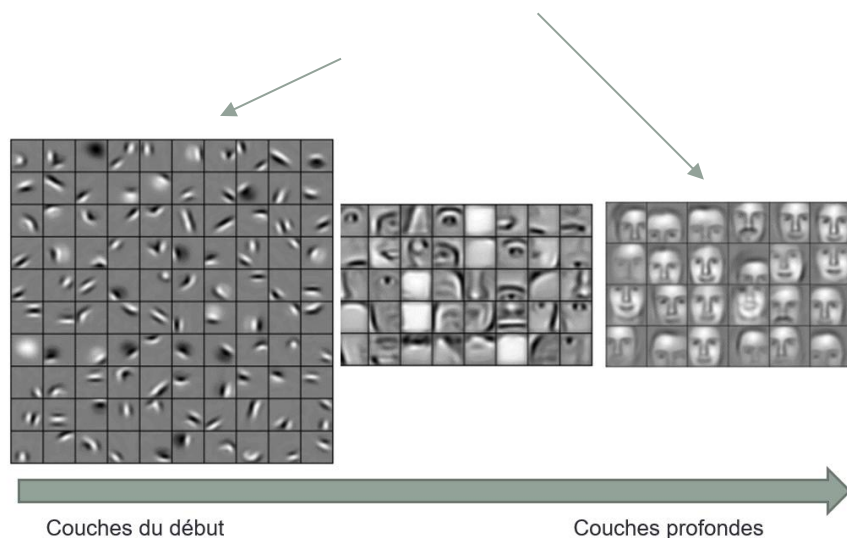
# Style transfer between images

VGG19 for extracting features

- Each convolution layer will produce a feature vector of dimension

Batch\_size x N\_features x Height x Weight

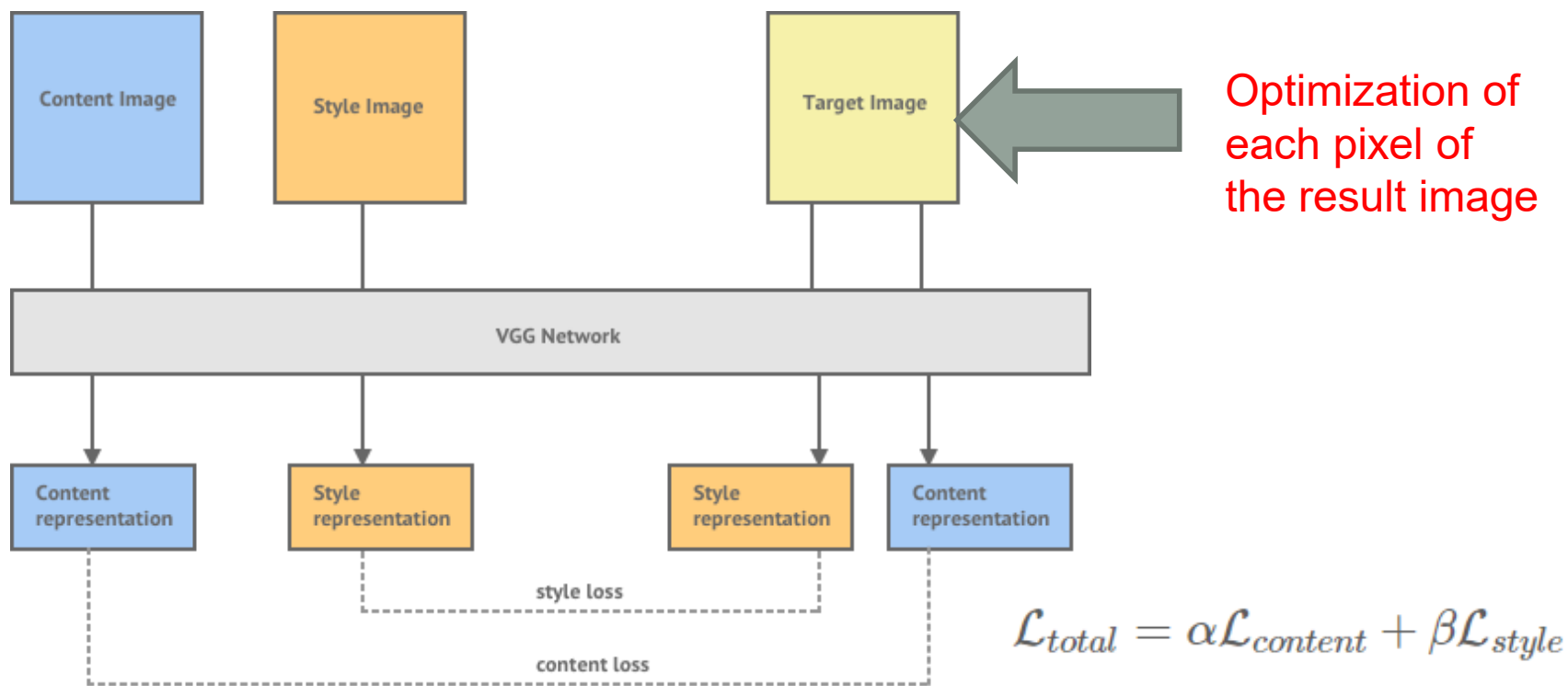
- Some layers encode content (towards the bottom of the network), others encode style (towards the beginning)



Features visualization of VGG network

# Style transfer: optimization

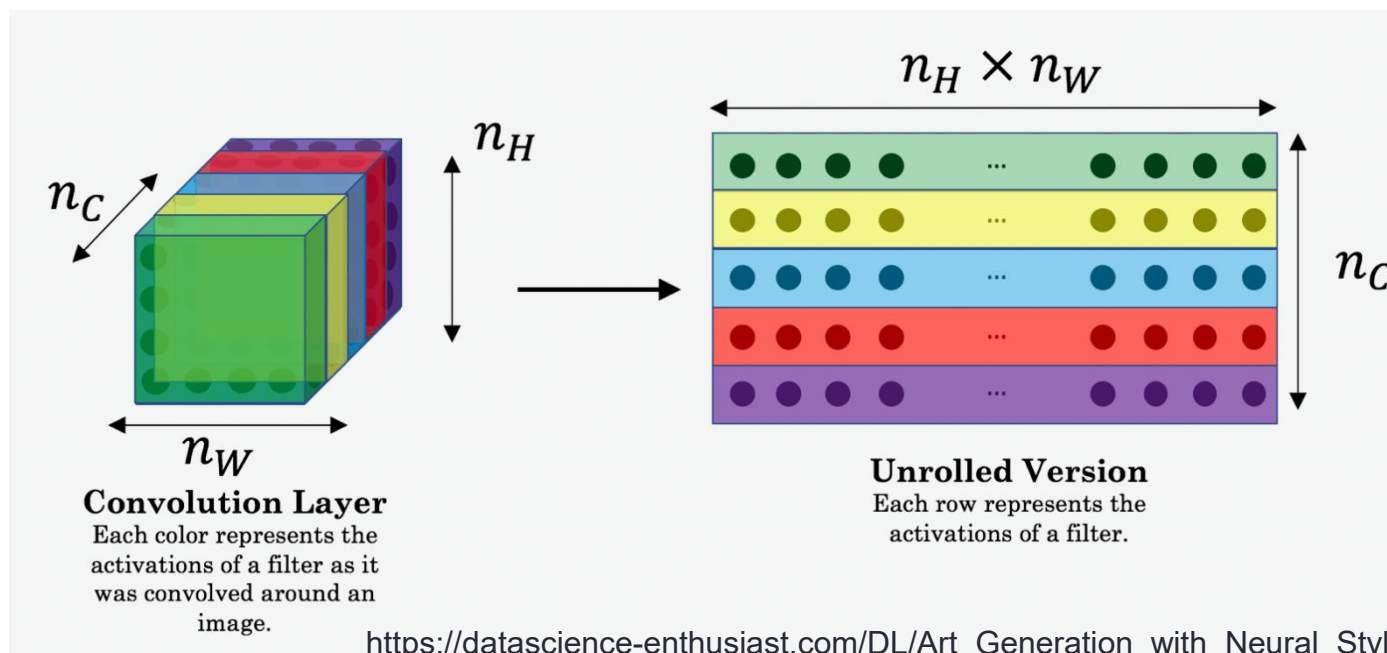
- There is no optimization of the neurons of a network
- The network (VGG) is used to produce the descriptors (features)
- Deep learning framework (Pytorch) is used to optimize the pixels



# Style transfer between images

## VGG19 for extracting features

- Each convolution layer will produce a feature vector of dimension  $N_{\text{features}}$  ( $N_c$  in the figure) x Height x Weight  
→ to flatten into  $N_{\text{features}} \times N_{\text{pixels}}$  with  $N_{\text{pixels}} = n_{\text{height}} \times n_{\text{weight}}$

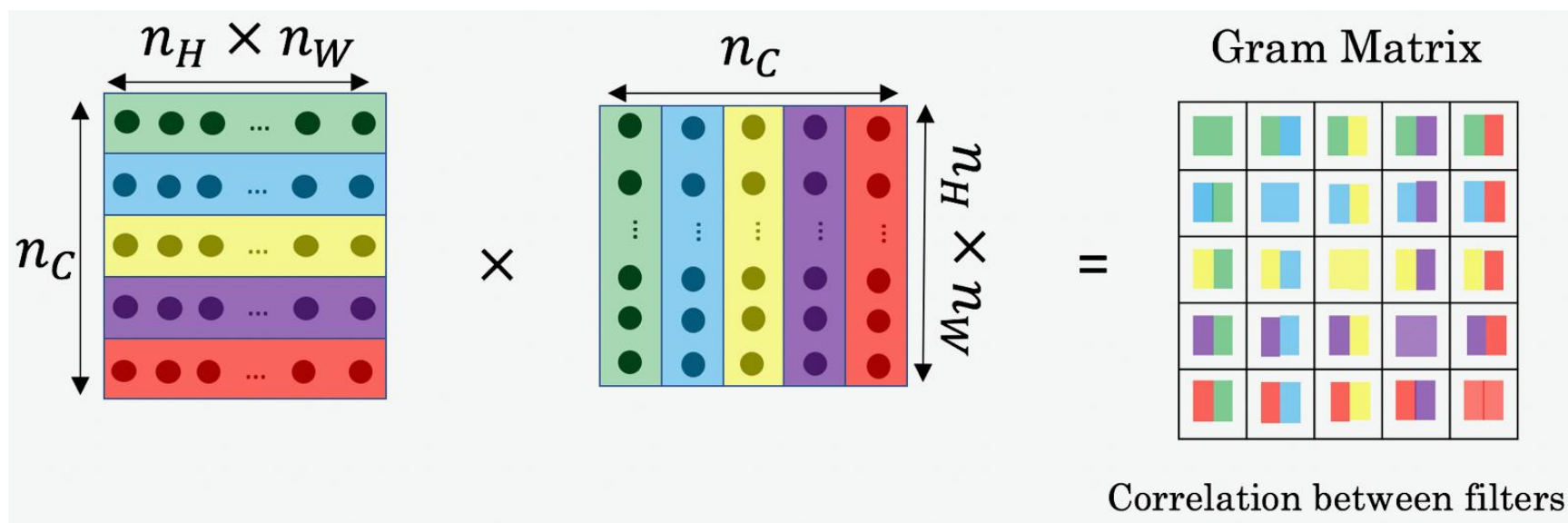


# Style transfer: Gram matrix

- **Gram matrix:  $M \times M^t$** 
  - Dot product between all features
  - Correlation between features
- With a feature matrix  $F$ , an entry of the Gram matrix  $G$  is the scalar product between 2 features

$$G_{ij} = \sum_k F_{ik} F_{jk}$$

# Style transfer: Gram matrix



If an entry in the Gram matrix has a value close to 0, this means that the 2 *features* do not activate simultaneously (no-correlation). And vice versa, if an input has a large value, it means that the 2 *features* activate simultaneously (correlation).

We try to create an image that replicates the same activation patterns of style *features*.

# Style Transfer: Content Cost

- If we can construct an image that has an equivalent feature map for a given convolution level to another image. These two images will have the same content (especially for the deep layers) — but not necessarily the same texture or style.
- Given a convolution layer **l** in VGG, the content cost function is defined as the mean squared error between the feature *map* **F** of the content image **C** and the *feature map* of the generated image **Y**.

$$\mathcal{L}_{content} = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

# Style Transfer: Style Cost

- The style cost calculation is similar to the content cost calculation, but it is calculated from the Gram matrix instead of directly using features .

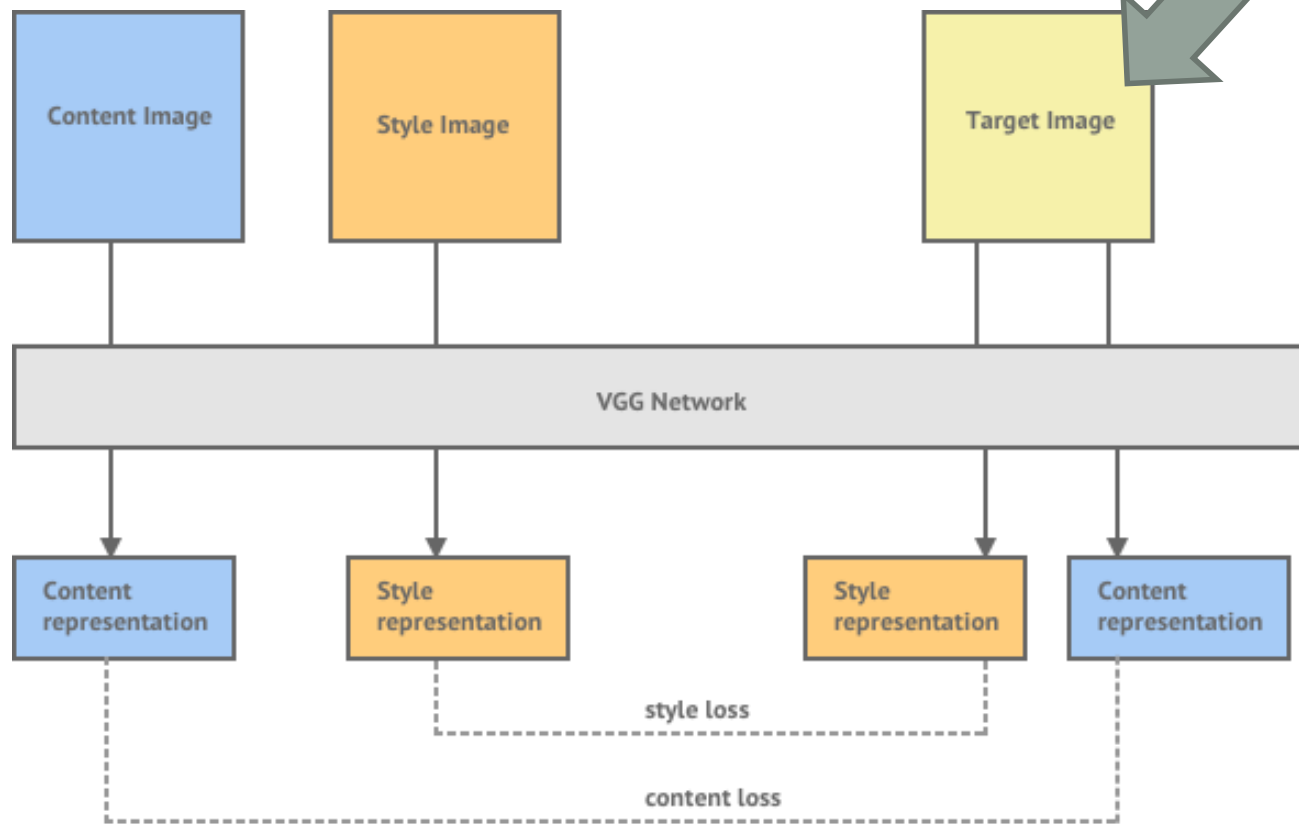
$$\mathcal{L}_{style} = \frac{1}{2} \sum_{l=0}^L (G_{ij}^l - A_{ij}^l)^2$$

# Style transfer: optimization

- The cost function to be optimized

Optimization of each pixel of the output image

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$



# Style transfer

content image



style image



generated image



VGG Network



# Conclusion

- CNN was the first Deep Learning Revolution  
→ TP
- Many approaches still to be seen...

