

CNN: Convolutional Neural Networks

Diane Lingrand and many contributors



2021 - 2022

1 CNN

2 Architectures

3 Art style transfert

4 Adversarial examples

Outline

1 CNN

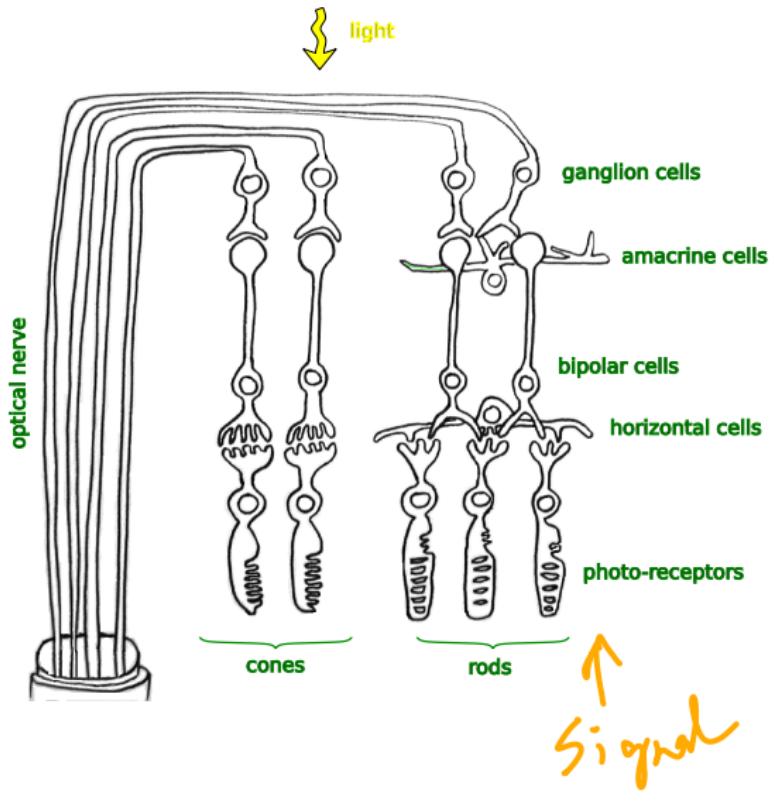
2 Architectures

3 Art style transfert

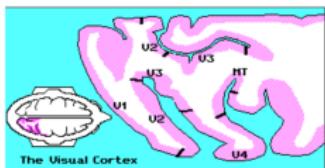
4 Adversarial examples

Deep Architecture in the Brain : the retina

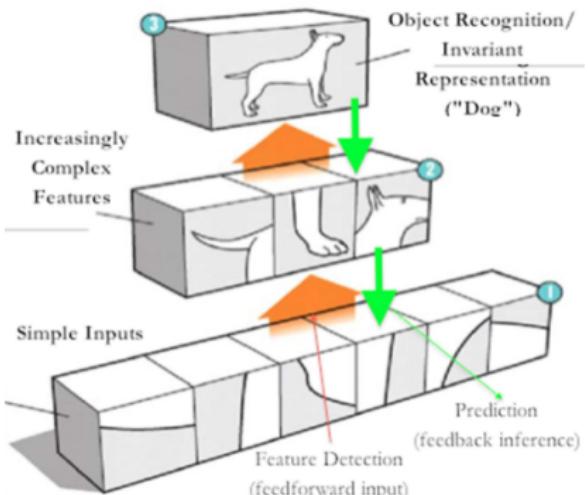
- photoreceptor cells : cones and rods
- bipolar cells (first neuron) : link to one or several photoreceptors
- ganglion cells (second neuron) link to one or several bipolar cells
- optical nerve (blind spot)



Deep Architecture in the Brain : the visual cortex



- area V1 : edge detectors (all directions)
- area V2 : primitive shape detectors
- area V3, V4 and more : higher level visual abstractions



- image classification

- neural network as a function for image classification
 - huge number of weights to learn (nb. layers x nb. neurons)
 - even with GPUs
- inspired by old fashion image processing methods
 - 70's : Sobel, Laplace, Kirsch, Prewitt, Mean or Gaussian smoothing
 - 80's : power of two \Rightarrow integer
 - 90's : SIFT, SURF, ...
 - all are based on convolution functions



Back to convolution operator

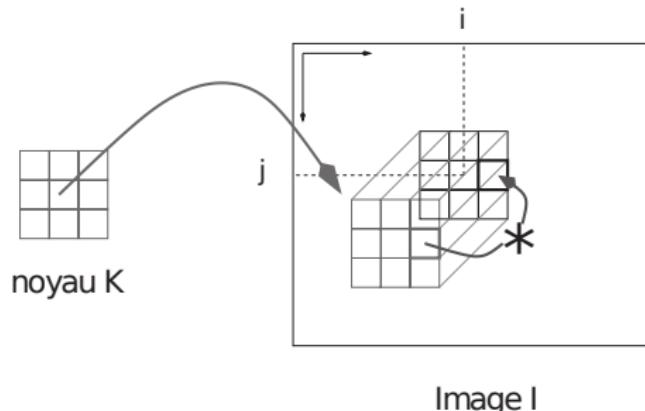
- For f and g , discrete functions :

$$f * g(x, y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(x, y)g(u - x, v - y)$$

F: filter

- Convolution of image I_1 by a kernel K of dimension $(2p + 1) \times (2q + 1)$:

$$I_2[i][j] = \sum_{k=0}^{2p} \sum_{l=0}^{2q} I_1[i - k + p][j - l + q]K[k][l]$$

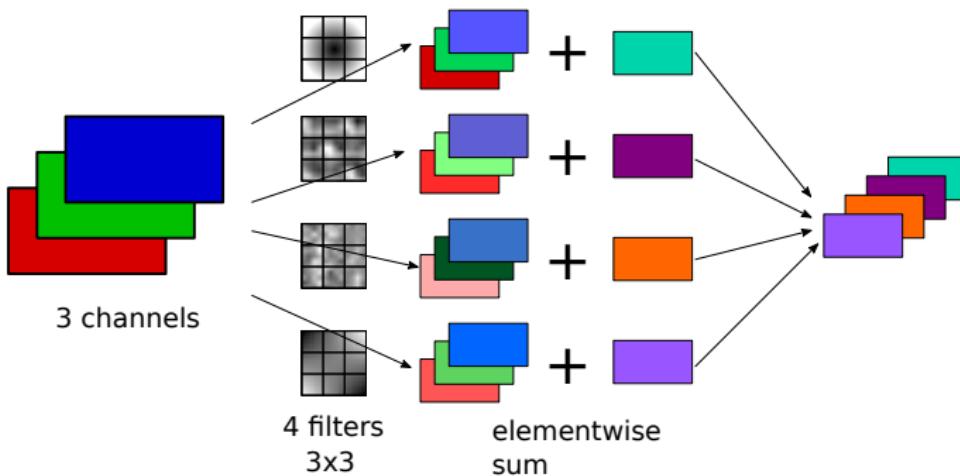


Idea of CNN : convolution layer

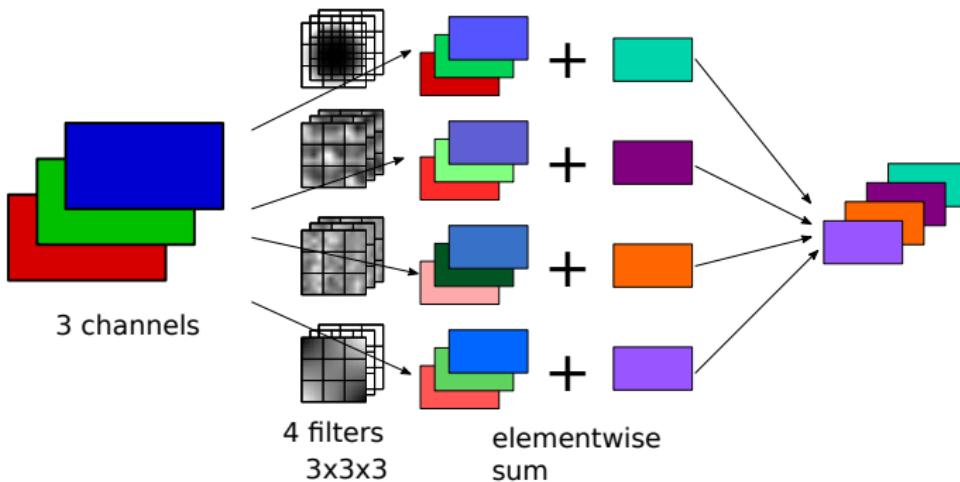
- learning the convolution filters
 - convolution : sum of weighted entries
 - 1 convolution filter = 1 neuron ↗
 - the weights are the filter coefficients
- sharing weights with different neurons
 - the same convolution is applied to every pixels in the image
 - less weights to learn !
 - example : one 3×3 filter : 10 coefficients (9 for filter + 1 for bias)
- in keras : Conv2D instead of Dense ↗
 - parameters : number of filters, size of filters (usually $k \times k$), stride
 - input : size of image and number of channels (3 if RGB)
 - output : tensor of dimension nb. filters, (new) dim of images
 - Conv1D and Conv3D also exists

More details on dimensions for a Conv2D layer

- parameters :
 - input : tensor of dimension $w \times h \times ch$
 - filters : nb filters, dimension $k \times k$
 - output : tensor of dimension $w' \times h' \times nb$
- question : how did the number of channels ch dimension vanished ?
 - for each filter, elementwise addition of the result of the convolution of each channel by this filter



More details on dimensions for a Conv2D layer



- number of parameters to learn :
 - each filter : $3 \times 3 \times (\text{number of input channels}) + 1$
 - 4 filters
 - thus $4 \times (3 \times 3 \times 3 + 1) = 112$

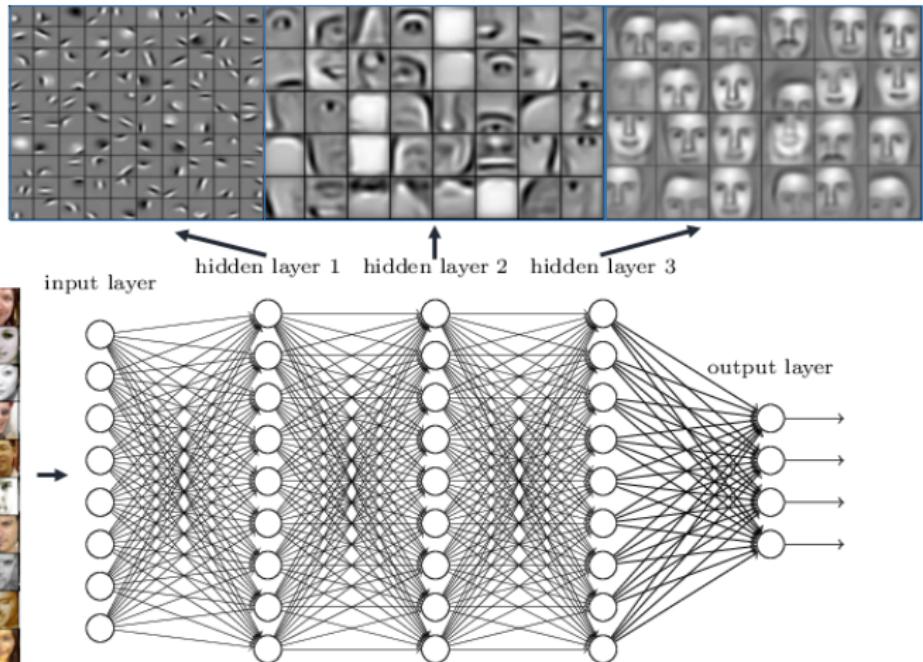
See <https://arxiv.org/pdf/1603.07285v1.pdf> for a complete discussion about dimensions

CNN : Convolutional Neural Network

- many convolutional layers
 - in order to extract the characteristics of images
 - but not only convolutional layers :
 - pooling
 - flatten
 - ReLU activation or Leaky ReLU ...
- two hidden fully connected layers at the output
 - the function of classification

CNN : Representative layers

Deep neural networks learn hierarchical feature representations



- Main activation functions :
 - Sigmoid
 - ReLU : Rectified Linear Unit
 - Leaky ReLU : in order to avoid exploding gradients
 - Softmax : transforms output values into values that can be interpreted as probabilities

- Other activation functions :

- tanh, LeakyReLU, PReLU, ELU, SELU, GELU ...

→ Best For Hidden Layer
or it oppose
vanishing
gradient.

→ it is mainly used at the Final layer (o/p)
for Classification

Softmax layer

- smooth approximation to the arg max function
 - assign probabilities to indices of having the highest response

$z = [z_0 z_1 z_2 \dots z_n]$ for each i ,

Probability 

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Softmax operation 

$z \in [0, 1]$

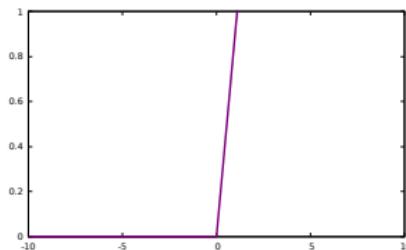
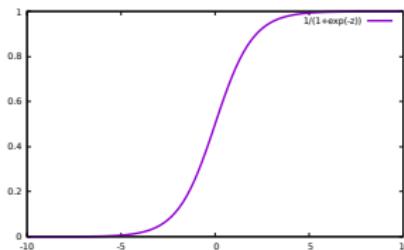
Why ReLU ?

- problem of vanishing gradient
 - remember the delta rule and gradient descent :

$$\delta_i = y_i(1 - y_i) \sum_{k=0}^n w_{ik} \delta_{ik}$$

and

$$\forall j \in [0 \ m] \quad w_j \leftarrow w_j - \eta \delta_i y_{ij}$$



- different solutions :
 - change to activation function to ReLU
 - residual networks
 - batch normalisation

Pooling layers

- usually after a convolutional layer
- reduces the size of data (downsampling)
- type of pooling
 - ✓ max pooling from keras.layers import MaxPooling2D
 - ✓ average pooling from keras.layers import AveragePooling2D
- dimension of pooling
 - ✓ default : 2X2 with stride 2
 - dimension
 - stride : factor of downsampling
 - padding ('VALID' means discard borders, 'SAME' zero padding)
- also exists : global pooling
 - max and average
 - reduces the dimensions to (batchSize, channels)

Cross Entropy Loss

$$H(p, q) = -$$

- GPUs!
- “Baby sitting” you deep network
 - variations of loss function or metrics with respect to epochs
 - regularisation (L1, L2, Elastic)
 - drop out
 - batch normalisation
 - optimisation function (Momentum, RMSProp, Adam, ...)

dropout

- some neurons are switched off
 - randomly chosen at each iteration
 - they do not contribute anymore to the output
 - their weights are not updated
 - they are randomly chosen according to some ratio
- benefits :
 - regularisation
 - reduce over-fitting (better generalisation)
- in keras, using a ratio of 20% :
 - ✓ ● input layer : `model.add(Dropout(0.2, input_shape=(784,)))`
 - ✓ ● hidden layer (between 2 layers) : `model.add(Dropout(0.2))`
- in practice :
 - impact on values (`kernel_constraint=maxnorm(3)`)
 - impact on learning rate, momentum ...

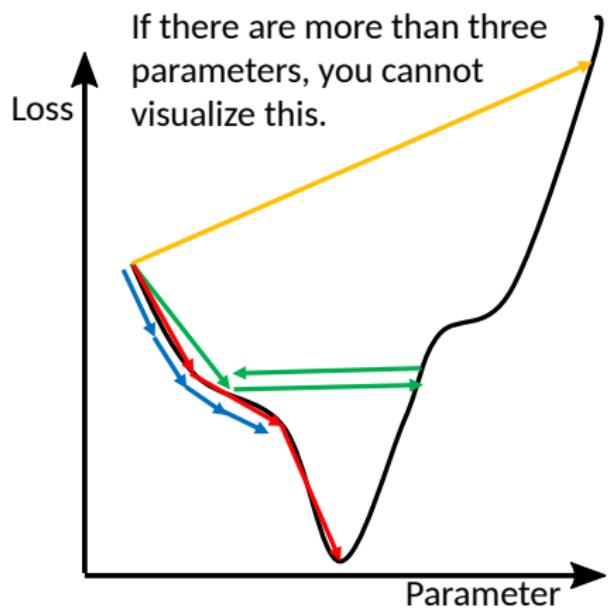


batch normalisation

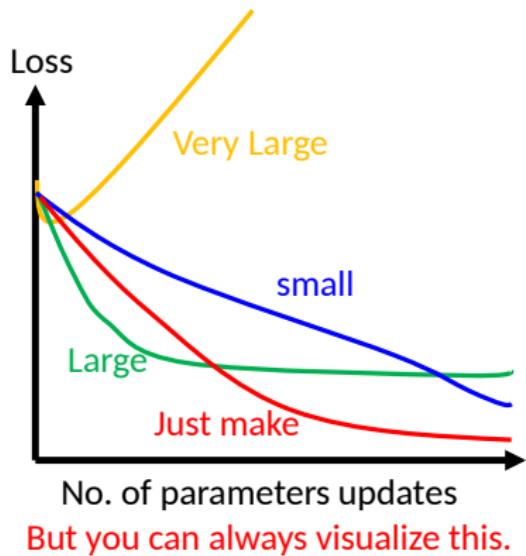
- normalization is needed at the entry of the NN
- normalization for the entry of a hidden layer
 - batch of learning data
 - typically 64, 128, 256
 - compute mean and standard deviation and normalize data
- advantages :
 - regularisation
 - avoid vanishing gradient for sigmoid activation
 - speed



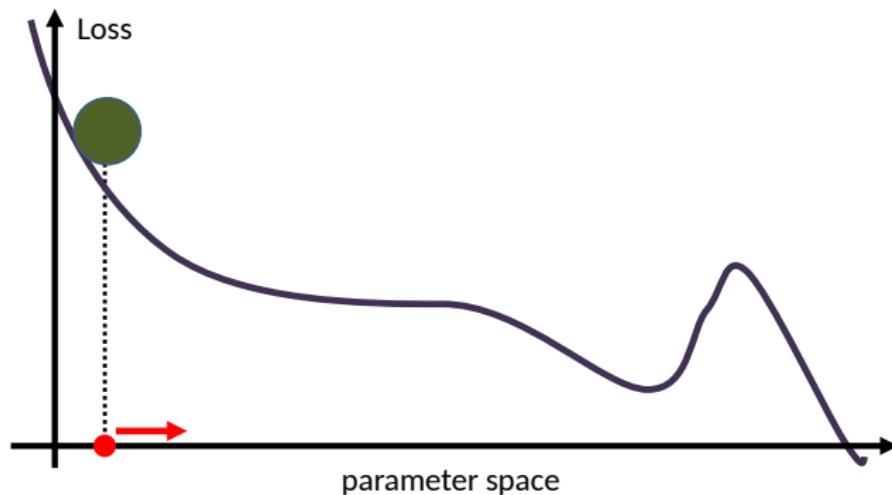
Learning rate



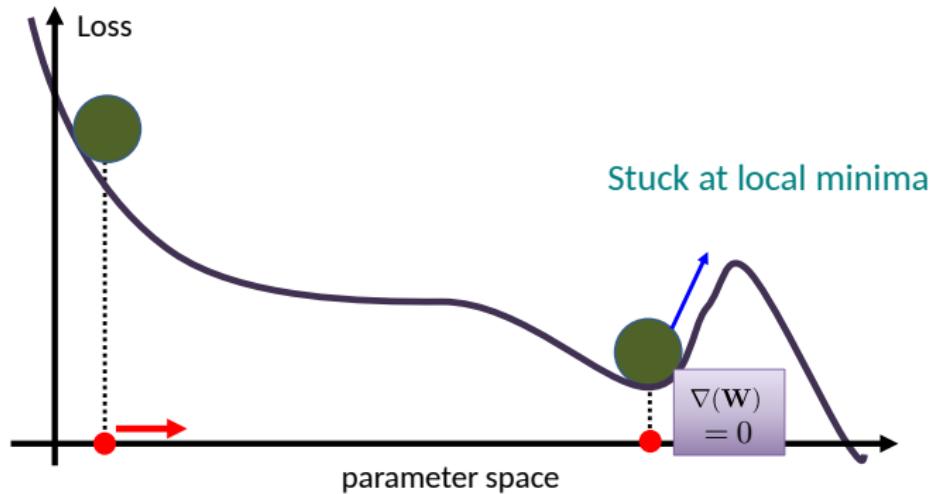
Set the learning rate η carefully



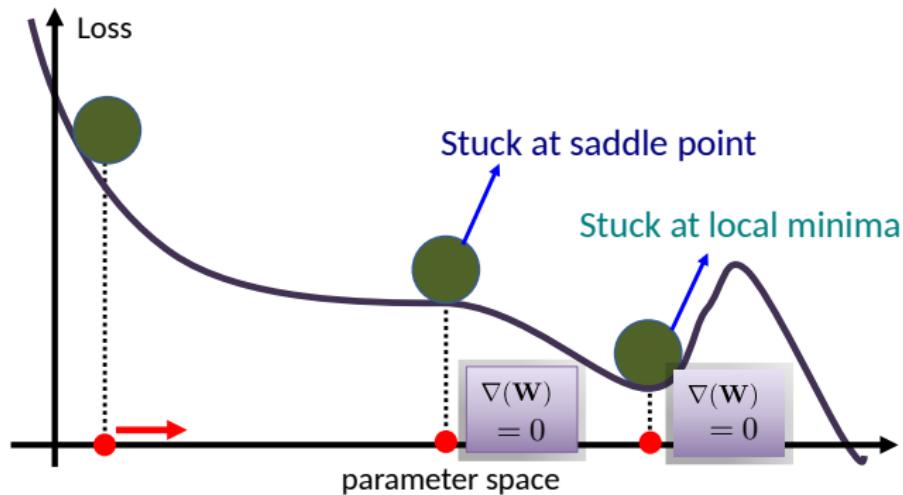
Momentum : avoid local minima



Momentum : avoid local minima

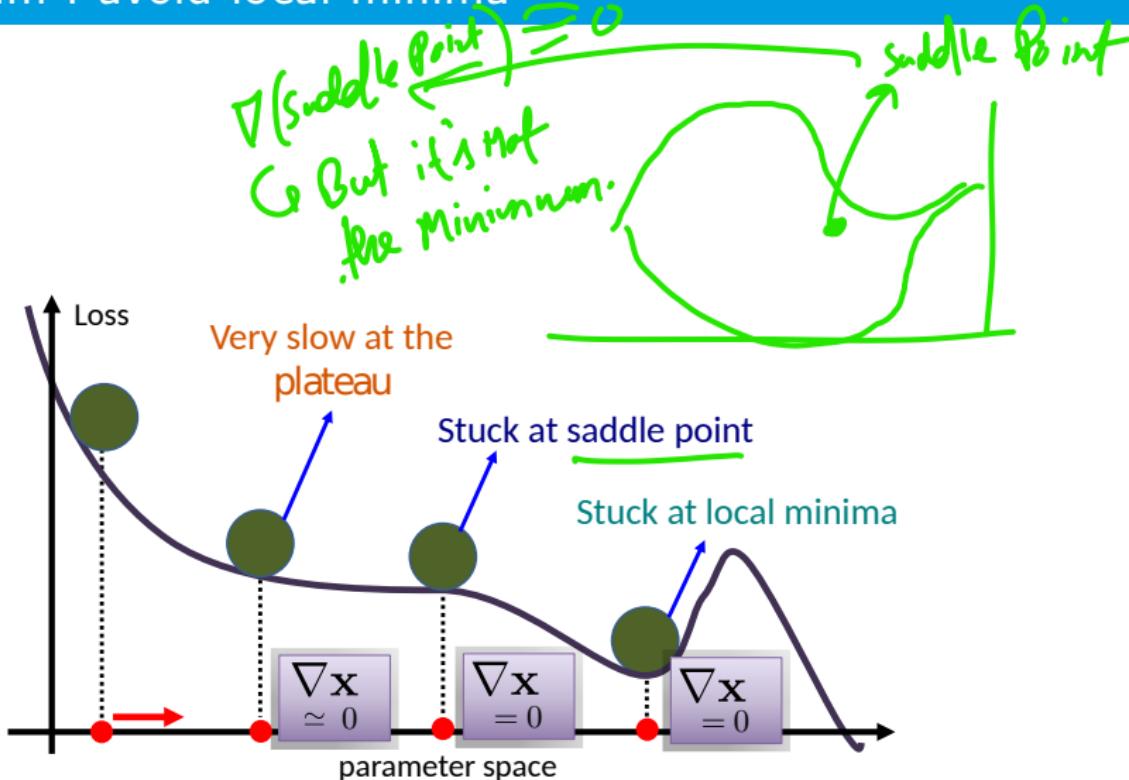


Momentum : avoid local minima

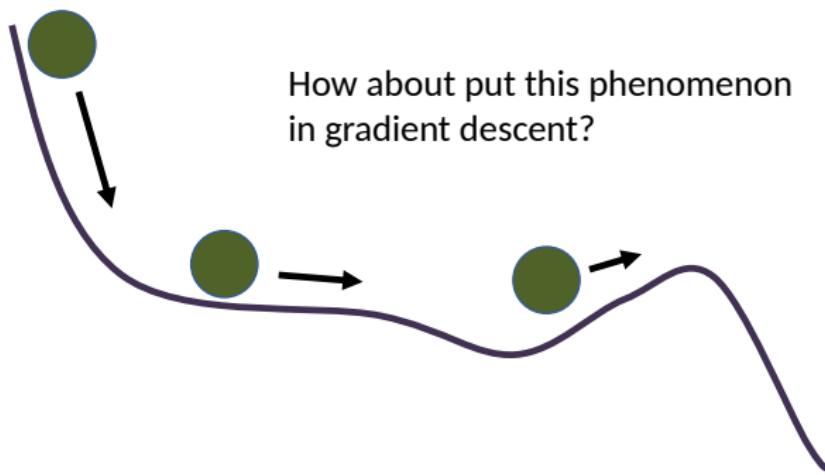


123

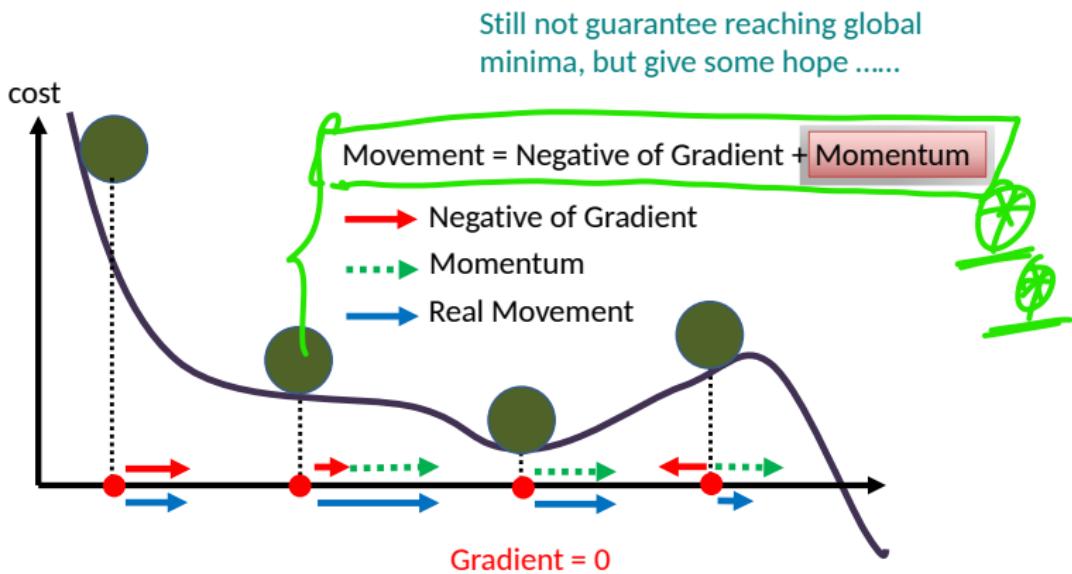
Momentum : avoid local minima



Momentum : avoid local minima



Momentum : avoid local minima



Local minima - Pathological curvatures

Idea : gradient doesn't tell everything. What about variations of gradient (second derivatives) ?



- Momentum
 - accumulates the gradient of the previous steps in order to compute the new weights
- RMSProp (Root Mean Square Propogation)
 - different learning rates for each weight
- AdaM (Adaptive Moment Optimization)
 - combining both Momentum and RMSProp

Outline

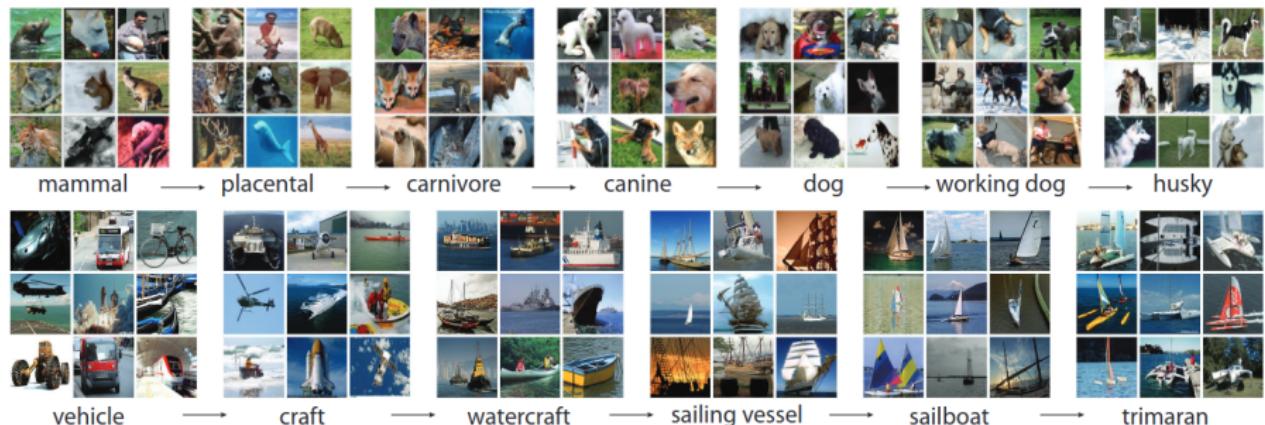
1 CNN

2 Architectures

3 Art style transfert

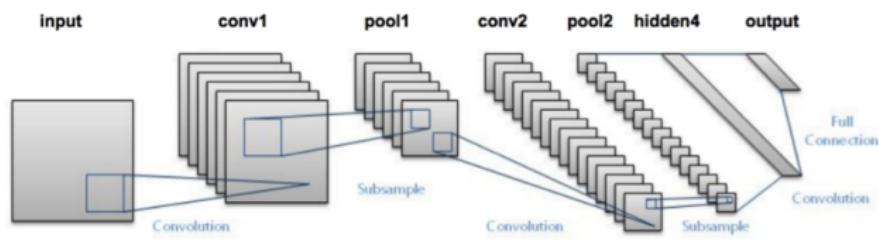
4 Adversarial examples

- more than 1000 classes
- 50 million labeled images
- in the ILSRVC : ImageNet Large Scale Visual Recognition Challenge

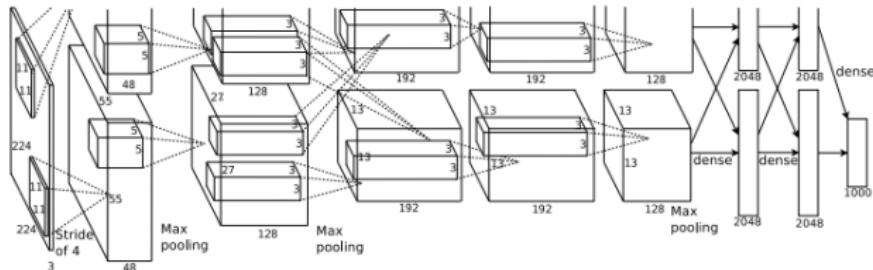


- LeNet 5 1998
 - historical, by Le Cun *et al*
- AlexNet 2012
 - first CNN winning ILSRCV competition in 2012
- VGGNet (2014)
 - 16 layers to be learned
 - VGG19 : 19 layers
- GoogLeNet Inception 2014
 - 22 layers
- ResNet 2015
 - introduce residual connections
- Xception (= eXtreme Inception)
 - separation depthwise and pixelwise convolutions
- EfficientNet 2019
- ConvNeXt 2022 (<https://arxiv.org/pdf/2201.03545.pdf>)

LeNet 5, 1998



- historical
- by Le Cun et al
- was applied to recognise hand-written numbers on checks (32x32 pixel grey images).



- 11x11, 5x5 and 3x3 convolutions, overlapping max pooling, local normalisation, dropout, data augmentation, ReLU activations, SGD with momentum. ReLU activations after every convolutional and fully-connected layer, 2 GPUs

input images	227x227x3
conv layer 1	96 kernels of size 11x11 with stride of 4 pixels size=(3x3), stride of 2 pixels
max pool layer	
conv layer 2	256 kernels of size 5x5, stride of 1 pixel size=(3x3), stride of 2 pixels
max pool layer	
conv layer 3	384 kernels of size 3x3
conv layer 4	384 kernels of size 3x3
conv layer 5	256 kernels of size 3x3 size=(3x3), stride of 2 pixels
max pool layer	
flatten layer	
dense layer 1	4096
dense layer 2	4096
dense layer 3	10 → op label

- architecture :

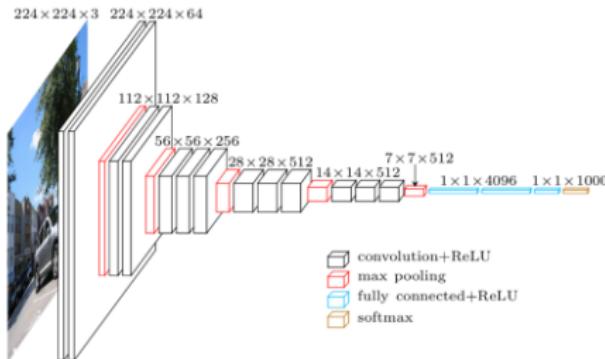
AlexNet : dimensions and params

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34944
batch_normalization (BatchNorm)	(None, 55, 55, 96)	384
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614656
batch_normalization_1 (BatchNorm)	(None, 27, 27, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 384)	885120
batch_normalization_2 (BatchNorm)	(None, 13, 13, 384)	1536
conv2d_3 (Conv2D)	(None, 13, 13, 384)	1327488
batch_normalization_3 (BatchNorm)	(None, 13, 13, 384)	1536
conv2d_4 (Conv2D)	(None, 13, 13, 256)	884992
batch_normalization_4 (BatchNorm)	(None, 13, 13, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37752832
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 10)	40970

Total params: 58,327,818

Trainable params: 58,325,066

Non-trainable params: 2,752



- deeper
 - VGG-16 : 16 refers to the number of weighted layers (conv or dense)
 - VGG-19 : 19 layers to learn
- smaller convolution filters (less parameters)
 - same receptive field for 3 stacked 3×3 conv. layer and 1 7×7 conv. layer
 - deeper : more non-linearities and fewer params ($3 \times (3 \times 3 \times ch_{in} \times ch_{out})$ compared to $7 \times 7 \times ch_{in} \times ch_{out}$)
- output of fc7 : good image features
- Simonyan and Zisserman (University of Oxford and Google Deep Mind) : <https://arxiv.org/pdf/1409.1556.pdf> - ImageNet Challenge

Topology of VGG16

```
VGGmodel = VGG16(weights='imagenet', include_top=False)
VGGmodel.summary()
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[None, None, None, 3]	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

- $(9*3+1)*64 = 1792$
- $(9*64+1)*64 = 36928$
- ...

$\sim 3 \times 3$

224×224

3 Dense layers

Network in Network (*Lin et al 2013*) : 1x1 convolution

- one channel

3x3 convolution

$$out_{ij} = \sum_u \sum_v in_{(i-d+u)(j-d+v)} k_{uv}$$

1x1 convolution

$$out_{ij} = in_{ij} * k$$

- several channels

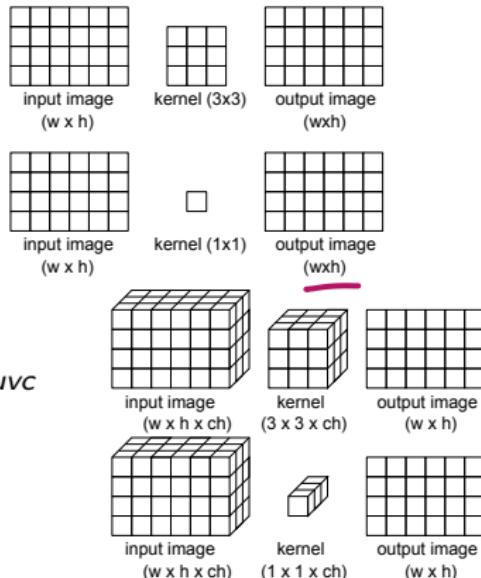
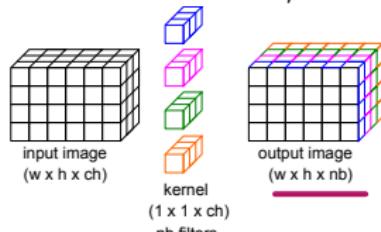
3x3 convolution

$$out_{ij} = \sum_c \sum_u \sum_v in_{(i-d+u)(j-d+v)c} k_{uvc}$$

1x1 convolution

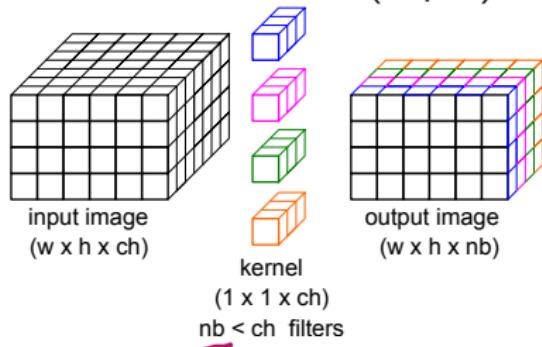
$$out_{ij} = in_{ijc} * k$$

- several channels, several filters and 1x1 convolution

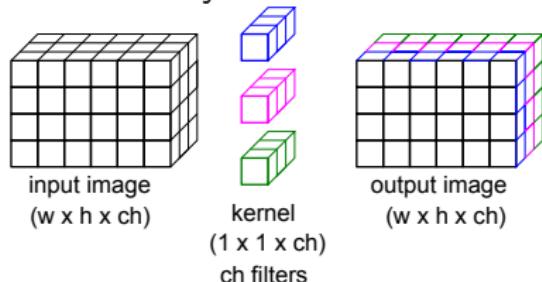


Using 1x1 convolution

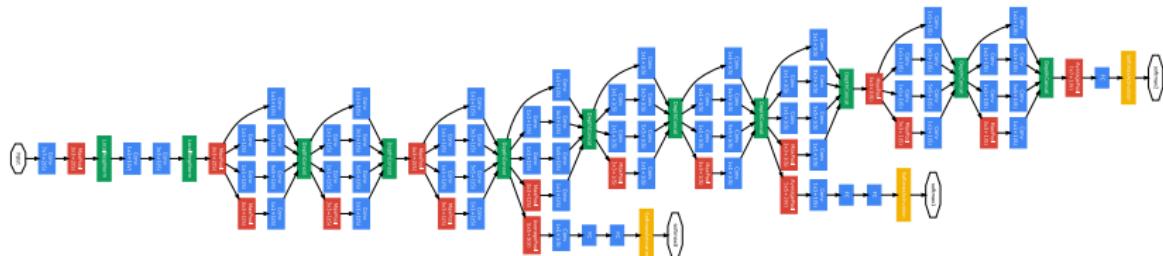
- dimension reduction (depth)



- non-linearity

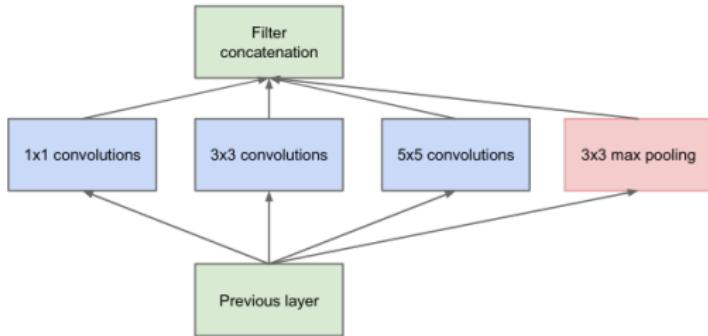


GoogLeNet Inception v1, 2014



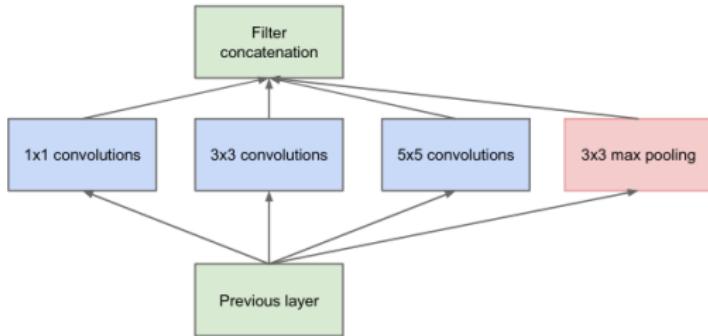
- 22 layers but 12x less params than AlexNet
 - Networks in Network
 - Inception module (1x1, 3x3, 5x5 convolutions and 3x3 max pool.)
 - do not choose the conv. size but try them all
 - paper : <https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf>
 - auxiliary classification outputs : inject add. gradient at lower layers
- v2 and v3 published in 2015
 - <https://arxiv.org/pdf/1512.00567v3.pdf>

Naive Inception module



(a) Inception module, naïve version

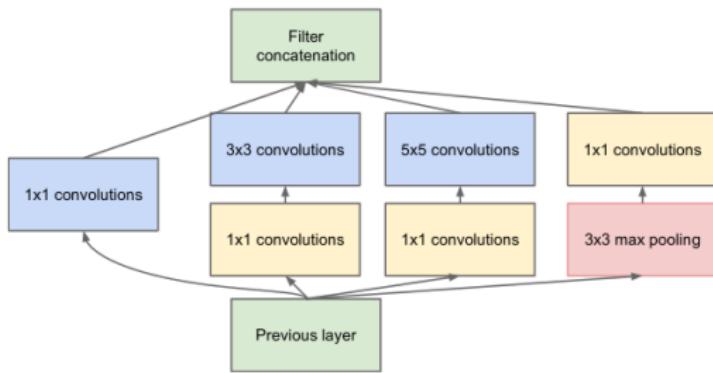
Naive Inception module



(a) Inception module, naïve version

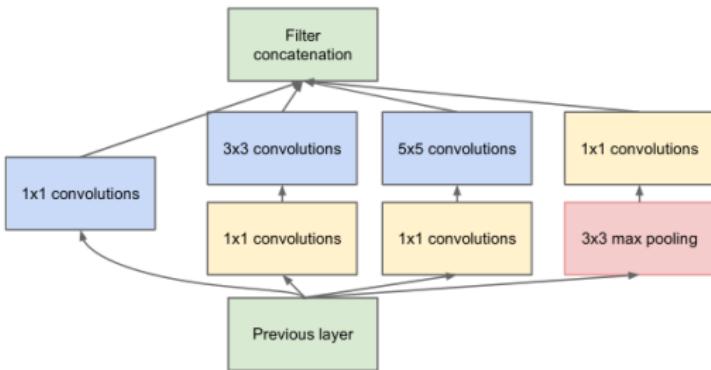
- Example of the 5x5 convolutions (96 filters) with a previous layer of dim $w*h*ch$
 - nb of coeff : $(25*ch+1)*96$
 - nb of op : $w*h*96*25*ch$

Inception module



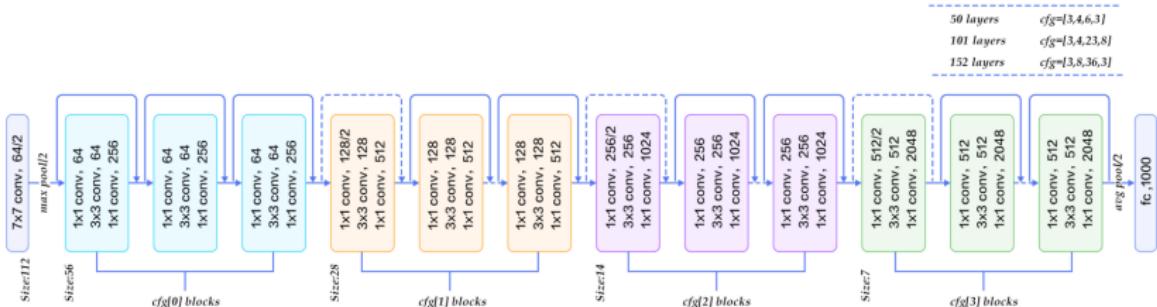
(b) Inception module with dimensionality reduction

Inception module



(b) Inception module with dimensionality reduction

- goal of 1x1 convolution is to reduce the depth
 - also for the pooling layer
- Back to the previous example of the 5x5 conv :
 - 1x1xnb followed by 5x5*96 with nb<ch and nb<96
 - nb of op : $w \cdot h \cdot 96 \cdot 25 \cdot nb + w \cdot h \cdot nb \cdot 1 \cdot ch = w \cdot h \cdot (96 \cdot 25 \cdot nb + nb \cdot ch)$



- introduce residual connections or shortcut connections
- 152 layers!
- paper : <https://arxiv.org/pdf/1512.03385.pdf>

Residual blocks

- Going deeper is not so easy

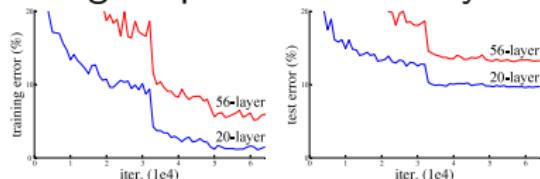


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error.

- Residual block : ease the learning

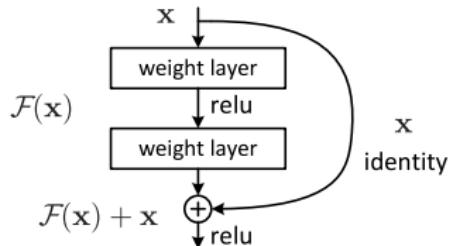


Figure 2. Residual learning: a building block.

- At least, learn the identity and do not perform less than a shallower network

ResNet efficiency

- no other FC than the classification one (softmax)
- with more than 34 layers
 - similarly to GoogLeNet

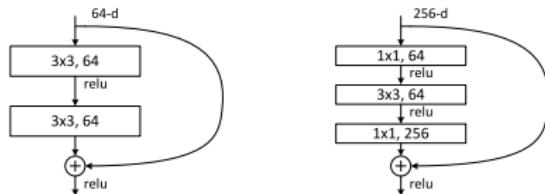
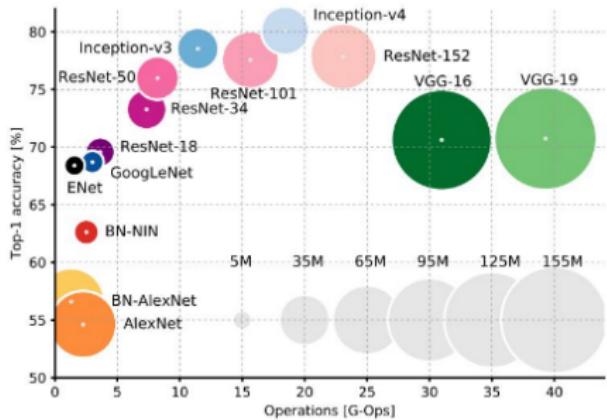
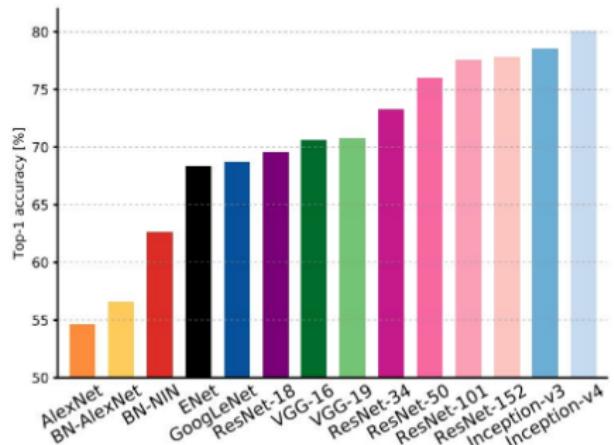


Figure 5. A deeper residual function \mathcal{F} for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

- Inception-v4 : ResNet + Inception

Different architectures



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

- Separation of convolution : spatial / channels
 - Depthwise convolution (spatial)
 - Pointwise convolution (cross channel)
- Simplified Inception Module vs Extreme Inception Module
 - Xception = Extreme Inception

- Improving by scaling up
 - the network depth (e.g. ResNet50 to ResNet200)
 - image dimension
- EfficientNet = family of networks
 - scaling up both width, depth and resolution

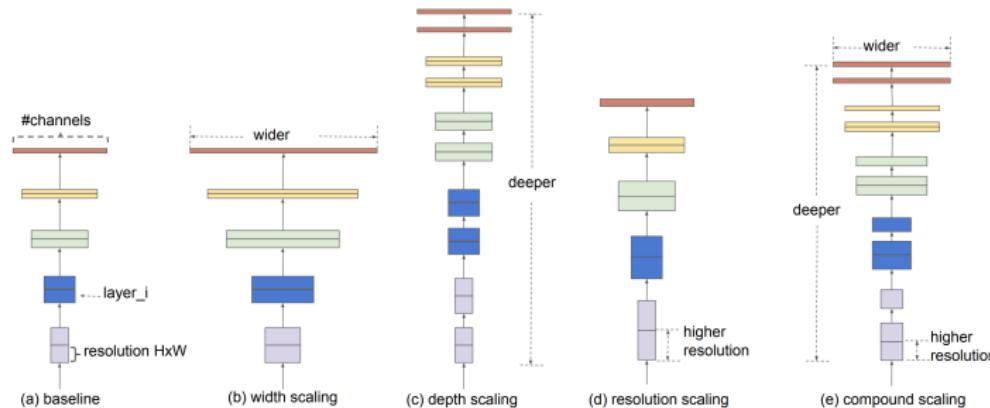


Figure 2. Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

from the paper <https://arxiv.org/pdf/1905.11946v5.pdf>

EfficientNet : compound scaling

- Compound scaling using ϕ (user control of resources availability) :

- depth : $d = \alpha^\phi$
- width : $w = \beta^\phi$
- resolution : $r = \gamma^\phi$
 - with : $\alpha\beta^2\gamma^2 \simeq 2$
 - and $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$

- which params ?

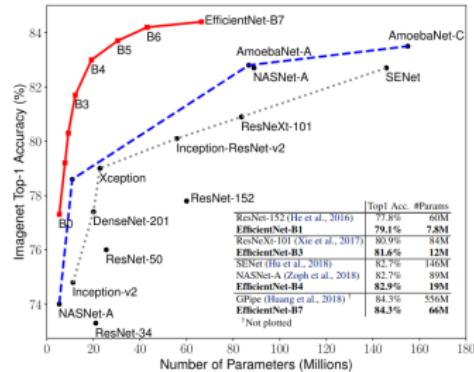
- set $\phi = 1$ and
 - compute α, β and γ by grid search
 - try different ϕ values

- FLOPS :

- proportional to d, w^2 and r^2
- increase in $(\alpha\beta^2\gamma^2)^\phi \simeq 2^\phi$

- State of the art accuracy with improving :

- inference speed
- size



Model Size vs. ImageNet Accuracy.

Learning the architecture

- Recent trends : learning the CNN architecture
 - topology
 - sub-networks or piecewise
 - image resolution (increasing)
 - data augmentation
- Methods :
 - progressing learning
 - grid search
 - reinforcement learning
- Examples :
 - NasNet (2018)
 - EfficientNetV2 (2021)

Architectures with keras

Model	Size (MB)	Top-1	Top-5	Parameters	Depth	Time (ms) per inference step	Time (ms) per inference step
		Accuracy	Accuracy			(CPU)	(GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	0.787	0.943	7200312	-	-	-
EfficientNetV2B1	34	0.798	0.95	8212124	-	-	-
EfficientNetV2B2	42	0.805	0.951	10178374	-	-	-
EfficientNetV2B3	59	0.82	0.958	14467622	-	-	-
EfficientNetV2S	88	0.839	0.967	21612360	-	-	-
EfficientNetV2M	220	0.853	0.974	54431388	-	-	-
EfficientNetV2L	479	0.857	0.975	119027848	-	-	-
EfficientNetV2B0	29	0.787	0.943	7200312	-	-	-
EfficientNetV2B1	34	0.798	0.95	8212124	-	-	-
EfficientNetV2B2	42	0.805	0.951	10178374	-	-	-
EfficientNetV2B3	59	0.82	0.958	14467622	-	-	-
EfficientNetV2S	88	0.839	0.967	21612360	-	-	-
EfficientNetV2M	220	0.853	0.974	54431388	-	-	-
EfficientNetV2L	479	0.857	0.975	119027848	-	-	-

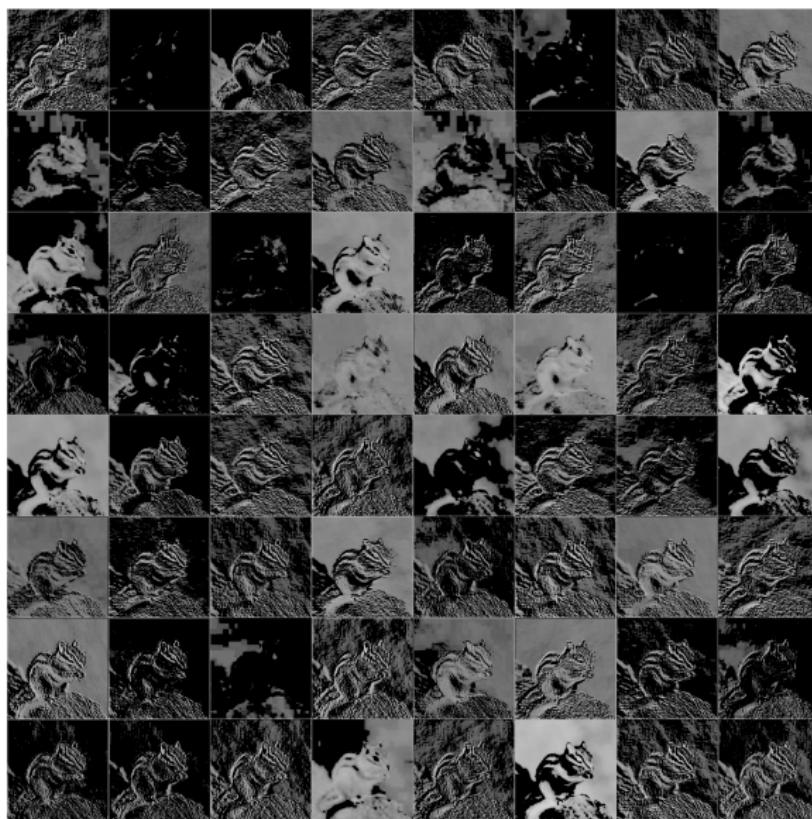
<https://keras.io/api/applications/>

VGG16 : VGGscoiattolo_input_1



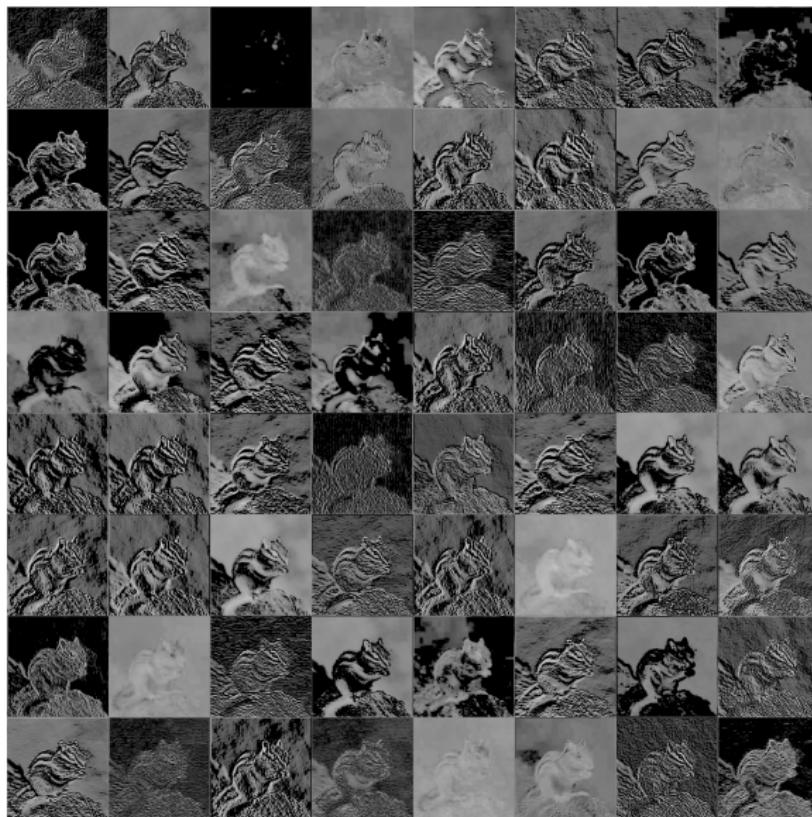
→ →

IN Conv Layer



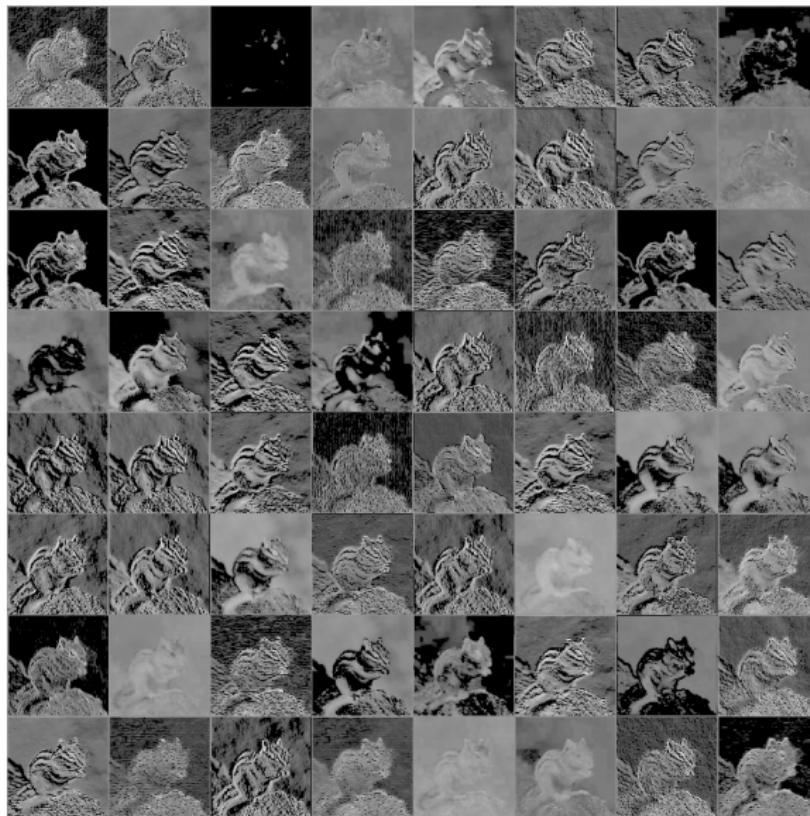
b4 Filter

VGG16 : VGGscoiattolo_block1_conv2

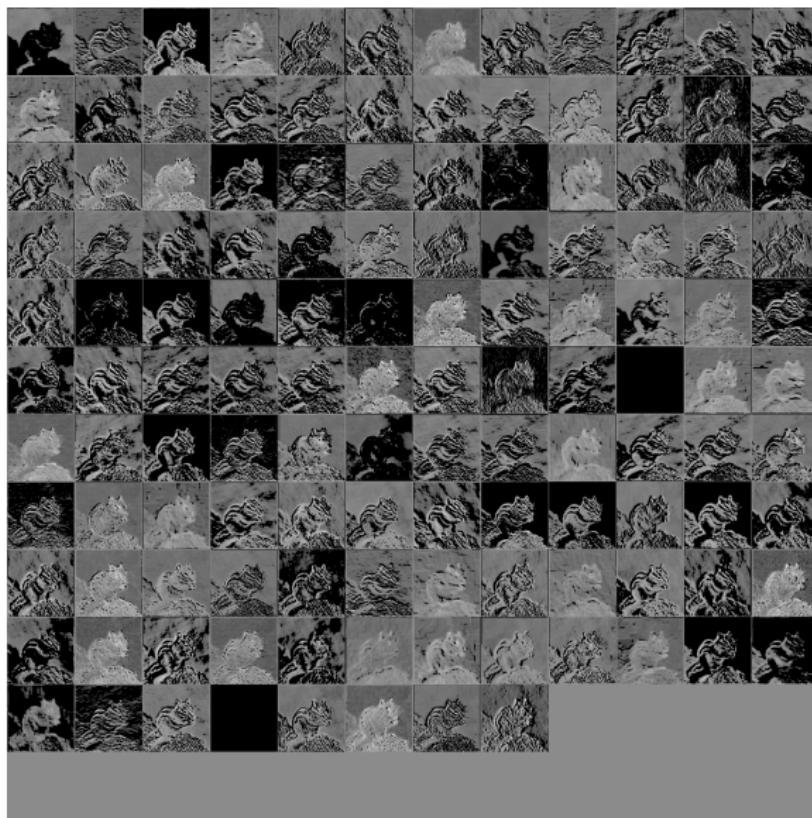


2nd

VGG16 : VGGscoiattolo_block1_pool



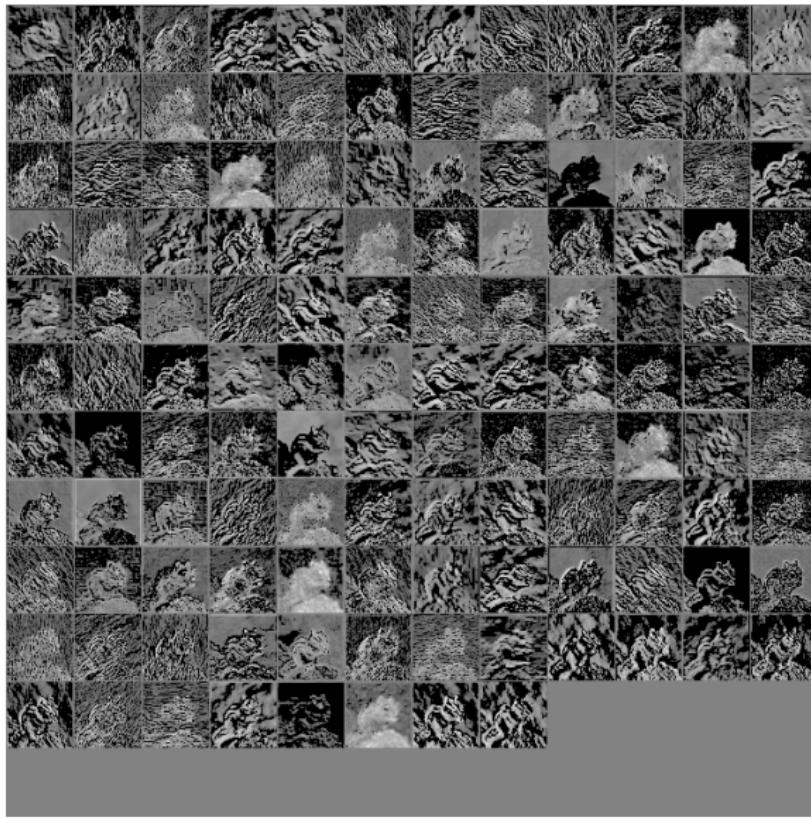
VGG16 : VGGscoiattolo_block2_conv1



VGG16 : VGGscoiattolo_block2_conv2



VGG16 : VGGscoiattolo_block2_pool



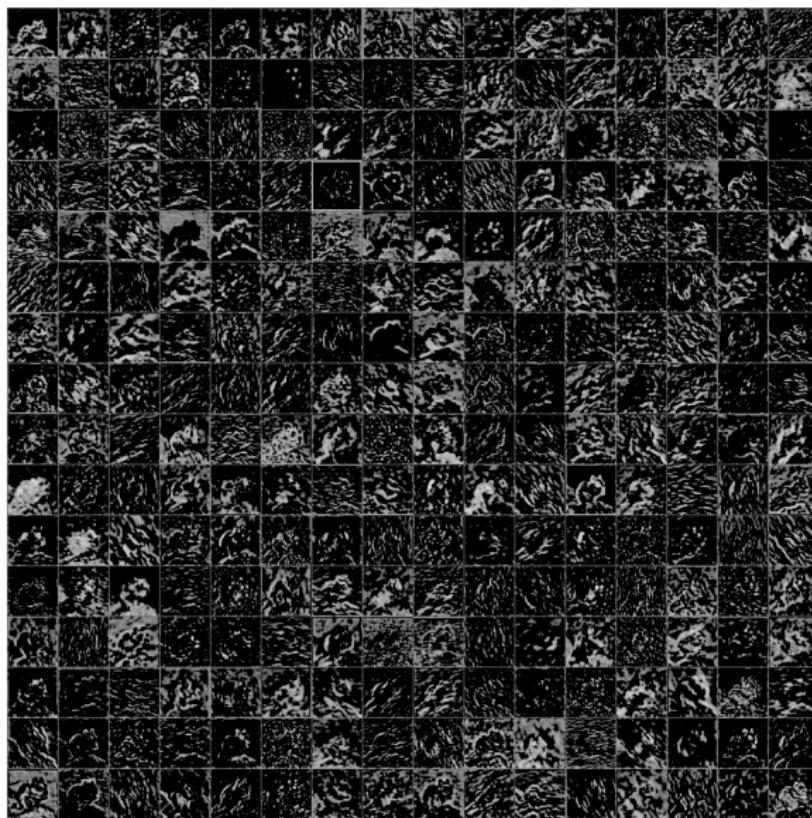
VGG16 : VGGscoiattolo_block3_conv1



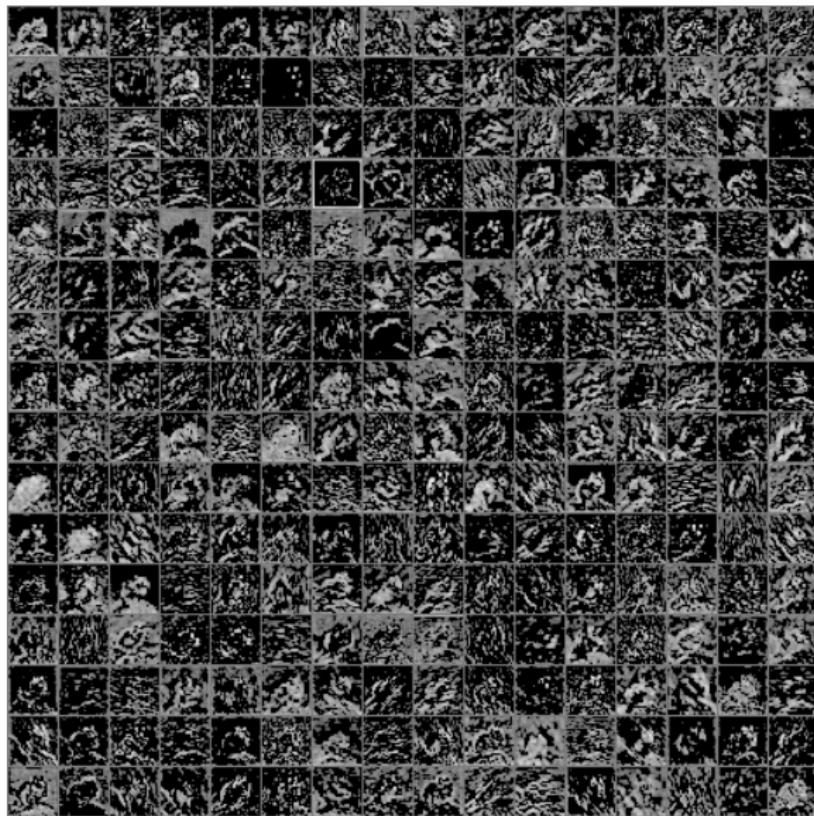
VGG16 : VGGscoiattolo_block3_conv2



VGG16 : VGGscoiattolo_block3_conv3



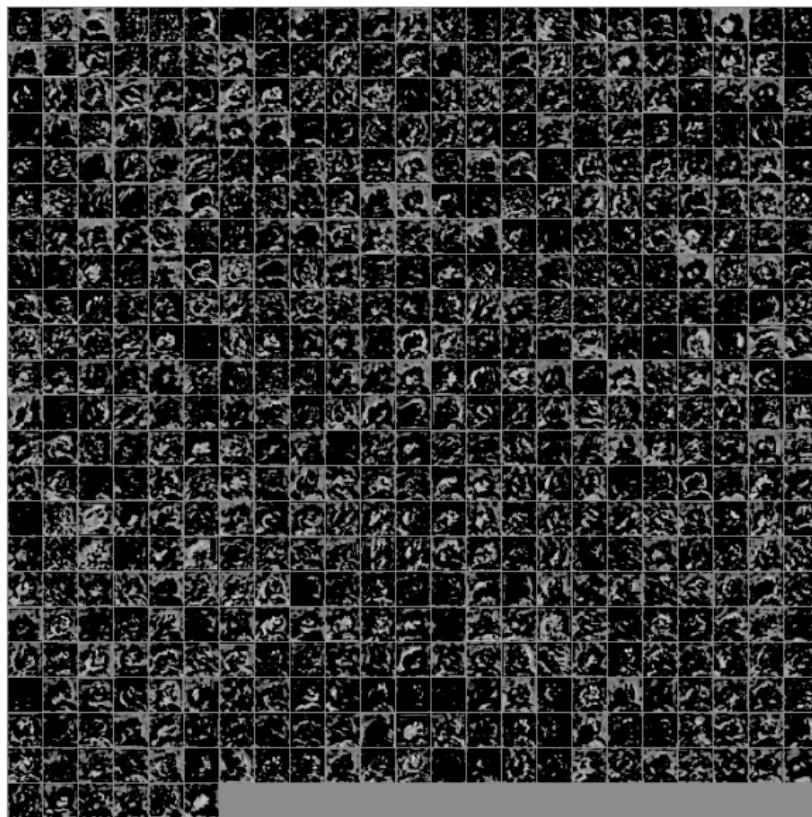
VGG16 : VGGscoiattolo_block3_pool



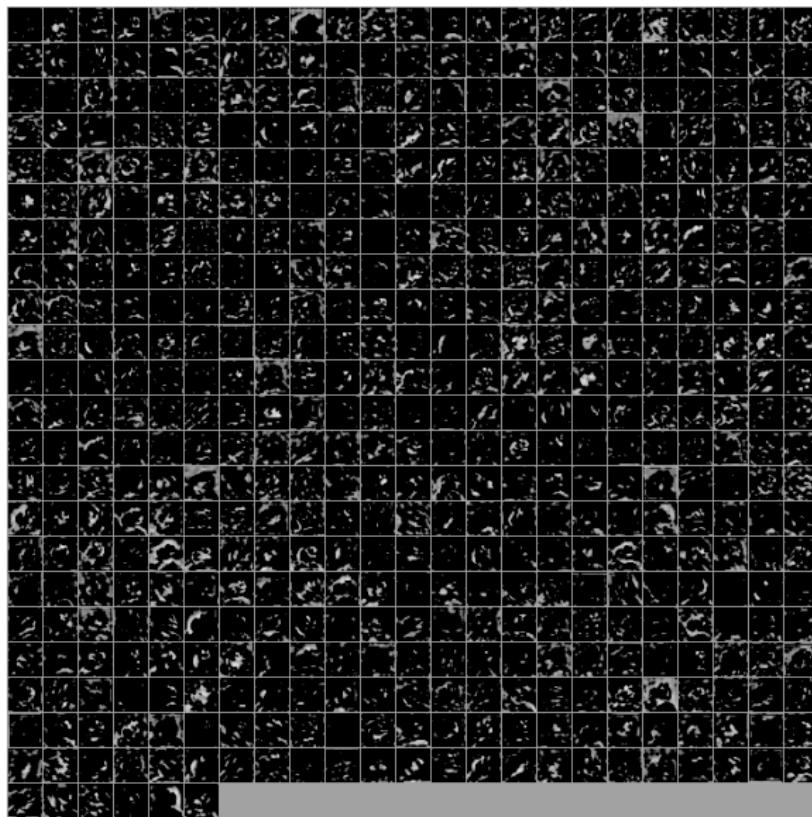
VGG16 : VGGscoiattolo_block4_conv1



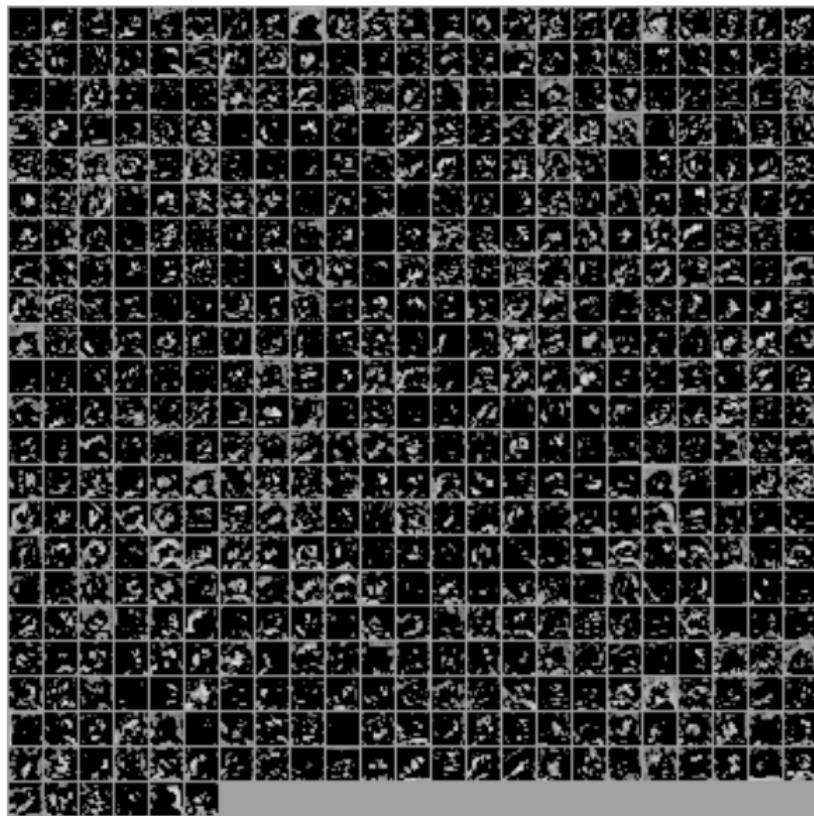
VGG16 : VGGscoiattolo_block4_conv2



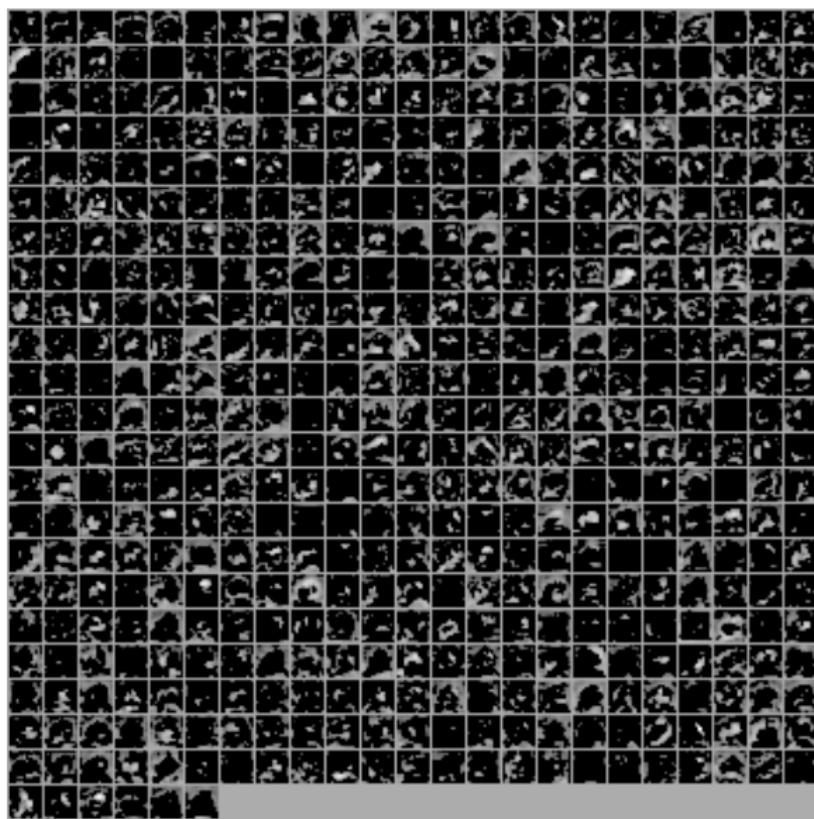
VGG16 : VGGscoiattolo_block4_conv3



VGG16 : VGGscoiattolo_block4_pool

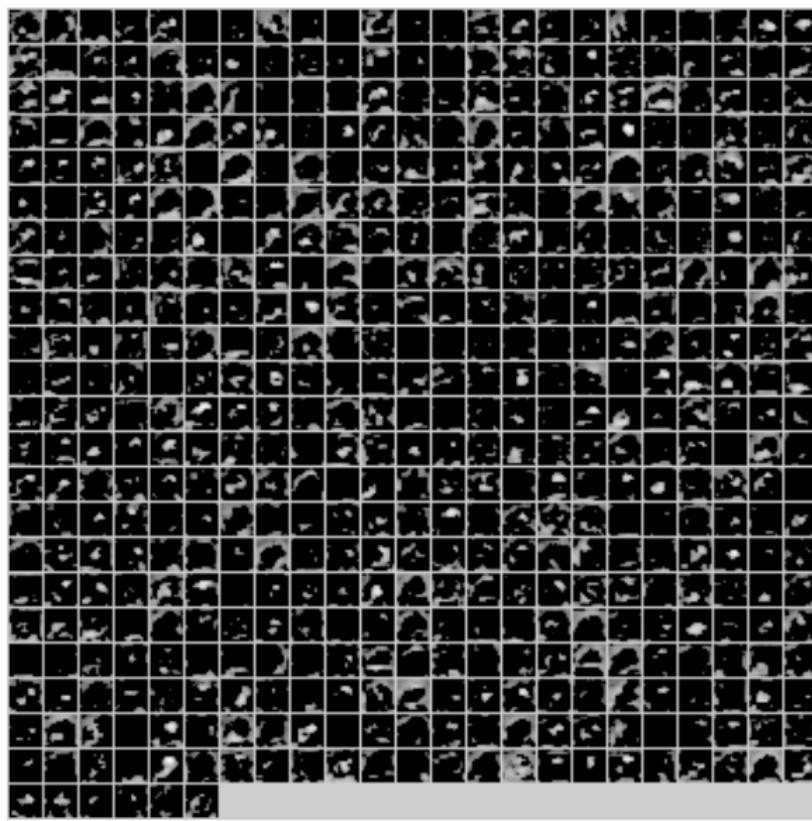


VGG16 : VGGscoiattolo_block5_conv1



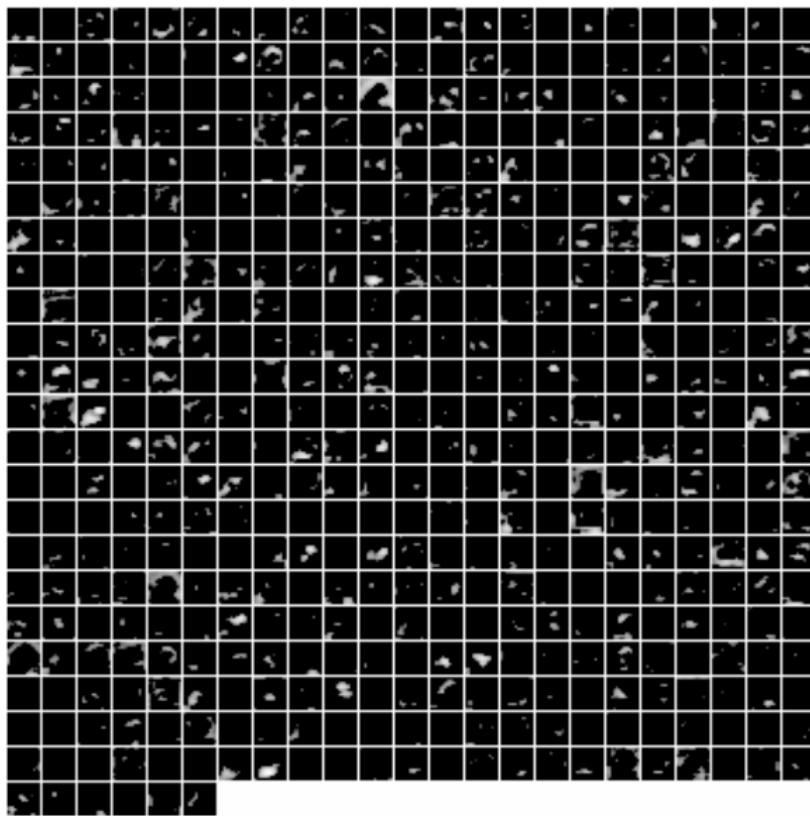
Red bracket highlighting the top row of the grid.

VGG16 : VGGscoiattolo_block5_conv2

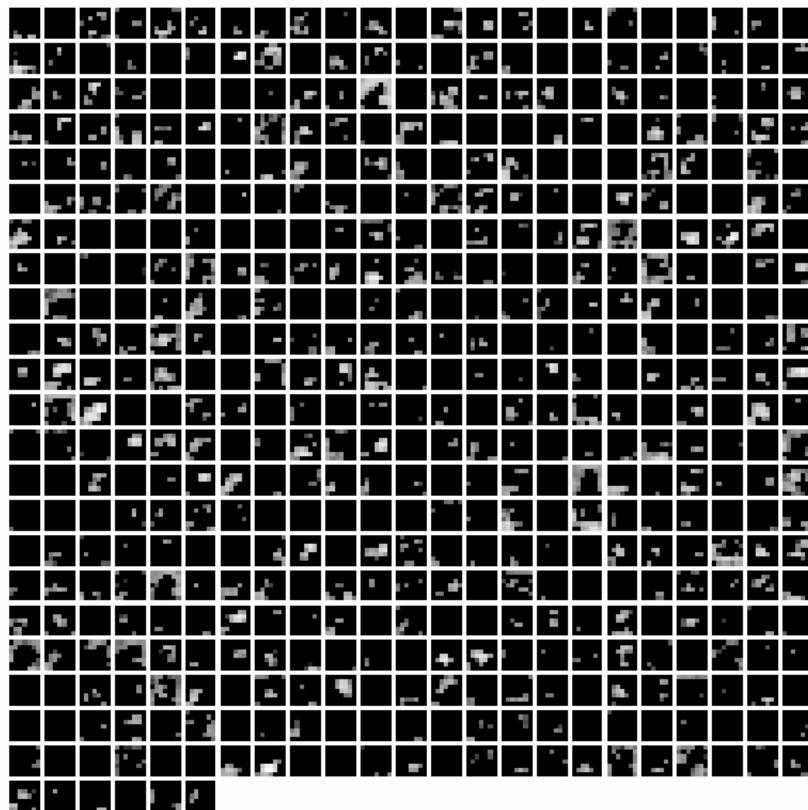


Red line pointing from the title to the image.

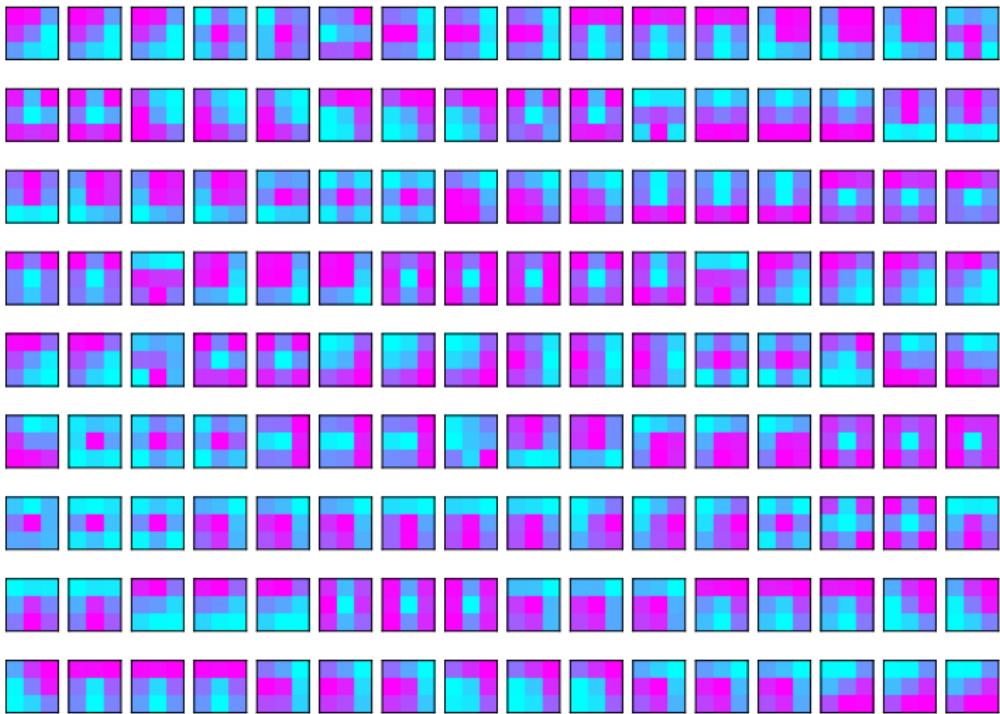
VGG16 : VGGscoiattolo_block5_conv3



VGG16 : VGGscoiattolo_block5_pool



VGG16 : filters visualisation



- image classification
- image descriptor
 - output of convolution layer as a vector describing an image
- transfert learning
 - learning only the last fully connected layers for a new image classification task
- sound recognition
 - spectrogram of sound as an image
 - transfert learning for sound classification
- art style transfert
- the problem of adversarial examples

Image descriptor using keras

Exemple from ExtractfeatureswithVGG16 :

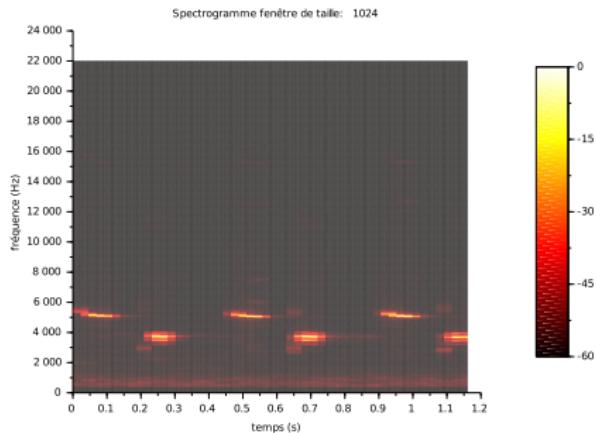
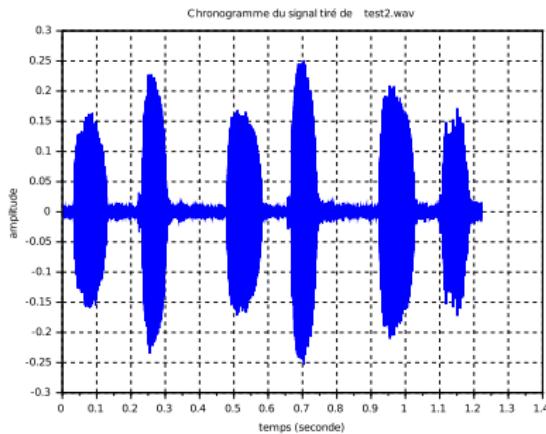
```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Spectrogram



Outline

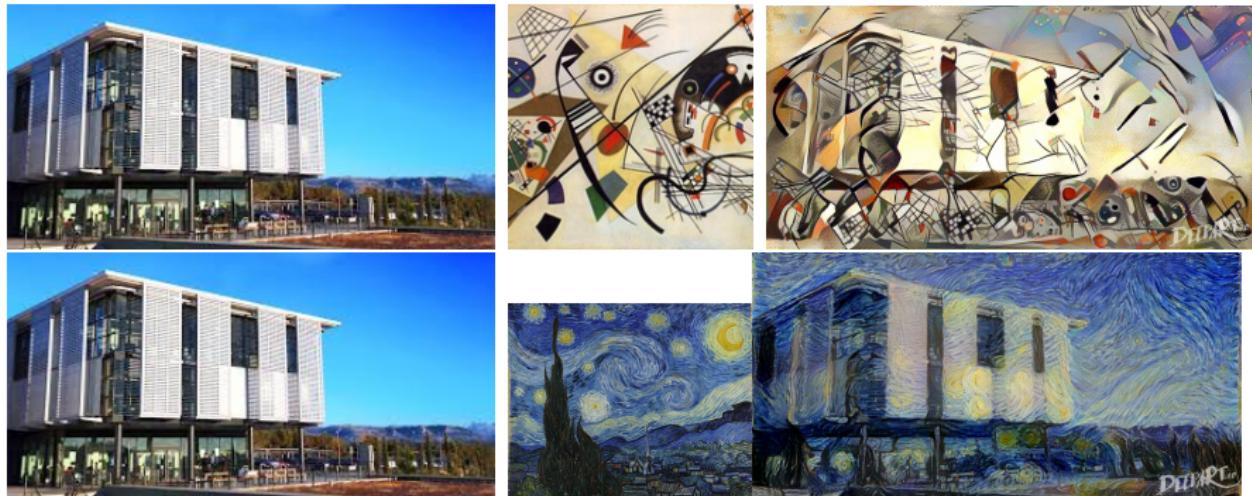
1 CNN

2 Architectures

3 Art style transfert

4 Adversarial examples

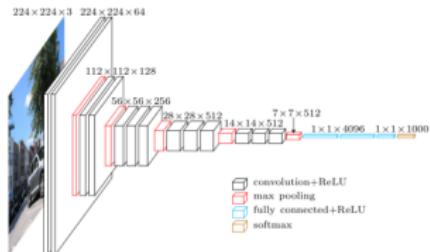
Art style transfert



- 2015 *A Neural Algorithm of Artistic Style* by Gatys et al
 - <https://arxiv.org/abs/1508.06576>
- <http://www.subsubroutine.com/sub-subroutine/2016/11/12/painting-like-van-gogh-with-convolutional-neural-networks>
- try it at deeprart.io

Art style transfert : How does it work ?

- Based on a VGG16 CNN (Diagram reproduced from the Heuritech blog) :



- Extract the content :
 - from layer 5_2 (2nd conv. layer from 5th conv. block) or 4_2
 - 2 images have same content if their output from layer 5_2 are similar (Euclidean distance)
- Capture the style :
 - from layers 1_1, 2_1, 3_1, 4_1 and 5_1.
 - Gram matrix (highlighting correlations, not specific details) : $F^T F$ where F is the output of the layers
- Algorithm :
 - start with a mix of content and random image
 - compute the output of all layers up to 5_2
 - compute the cost (similar content + similar style)
 - backpropagate the gradient in order to **modify pixels**.

Deep dream

- amplify the neuron activations at some layer in the network
 - choose an image and pass it through a CNN until a chosen layer
 - at this layer, set gradient equal to activation
 - backward : modify the input image
- try it at <https://deepdreamgenerator.com/>
- Example on layer inception_5b_pool_proj



Outline

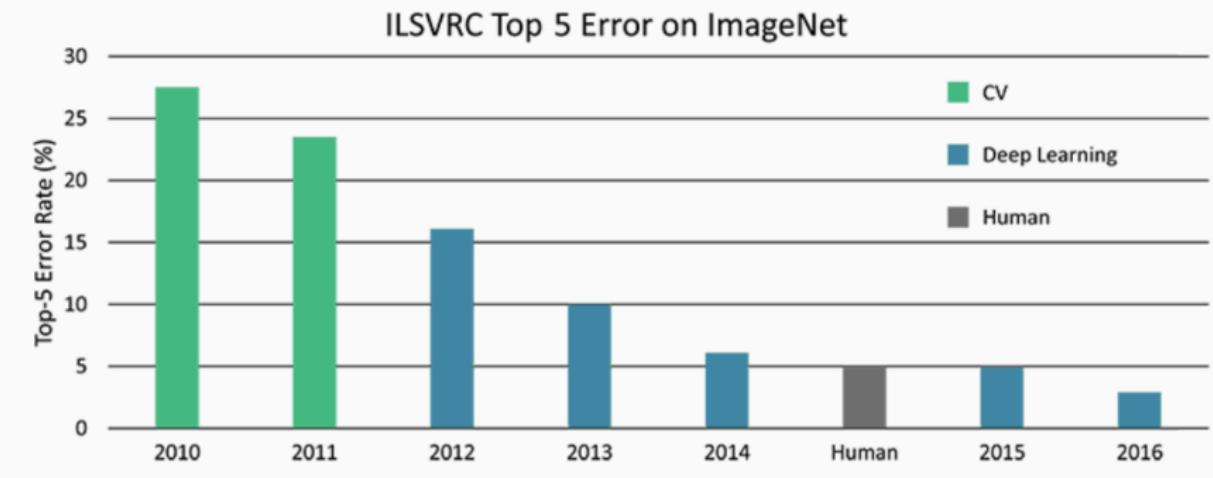
1 CNN

2 Architectures

3 Art style transfert

4 Adversarial examples

Deep networks are good at image classification



But ... how much does a deep network understand those tasks ?

What is an adversarial image?



panda



gibbon

With high scores : gibbon with a confidence of 99.3%

{ white box effect } attached
Black Box

A handwritten note in blue ink. It shows a brace grouping two terms: "white box effect" and "Black Box". An arrow points from the word "attached" to the brace, indicating that the "white box effect" is what is being "attached" to the original image.

What is an adversarial image ?



panda



gibbon

With high scores : gibbon with a confidence of 99.3%

For those who don't know what a gibbon is : 
from <https://fr.wikipedia.org/wiki/Hylobatidae>

Definition : Adversarial Example

Definition : \hat{I} is called adversarial if and only if :

- given image I
- low distortion $\|I - \hat{I}\| < \epsilon$, ($\epsilon > 0$, few pixels)
- given network's probabilities $f_\theta(I)$
- **Different predictions** :
 - $\arg \max f_\theta(I) \neq f_\theta(\hat{I})$

Adversarial examples

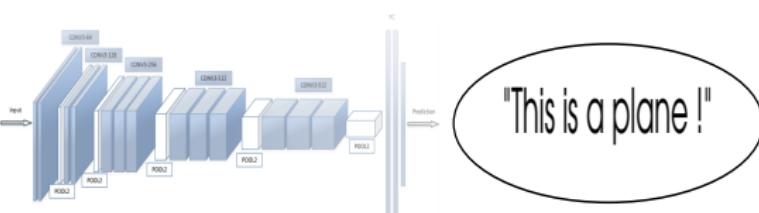
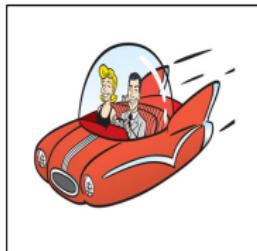
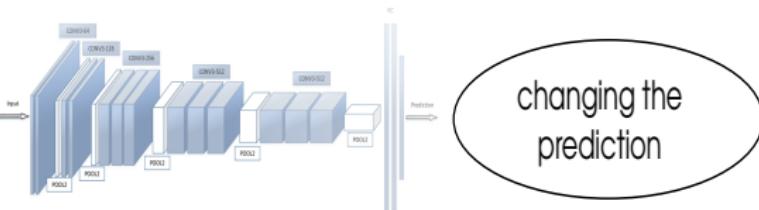
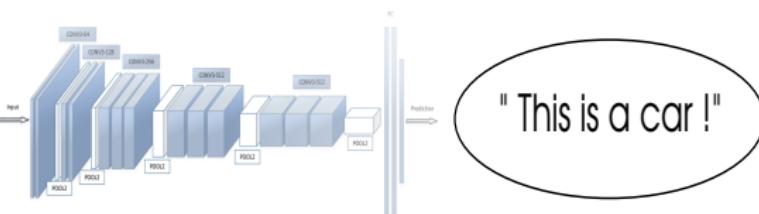
- images
- sounds
- synthesized 3D objects

Previously in this course

- We have modified pixels for
 - DeepDream
 - style transfert
- We know how to modify pixels in order to increase the answers of a particular neuron
- Let's modify any image well classified into another class

A simple experiment : what we expected

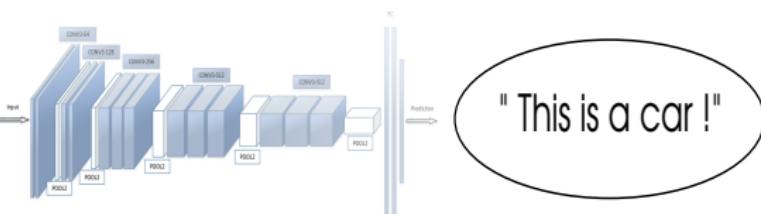
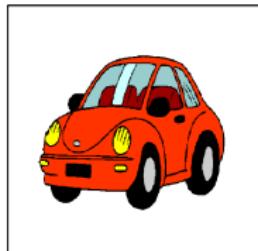
Input



from Ducoffe and Precioso

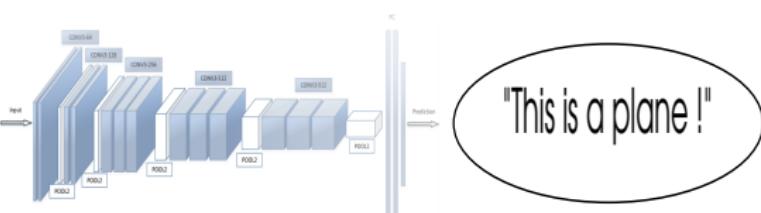
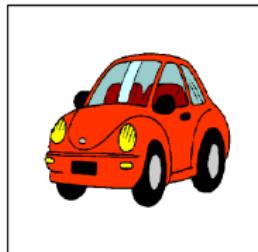
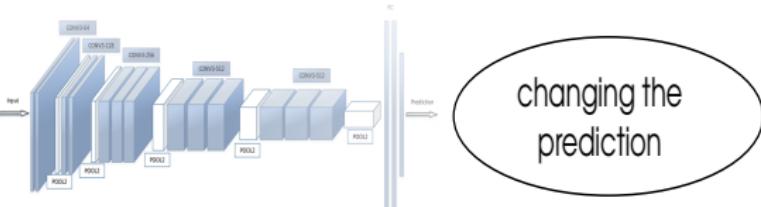
A simple experiment : what really happened

Input



Network's prediction

" This is a car !"

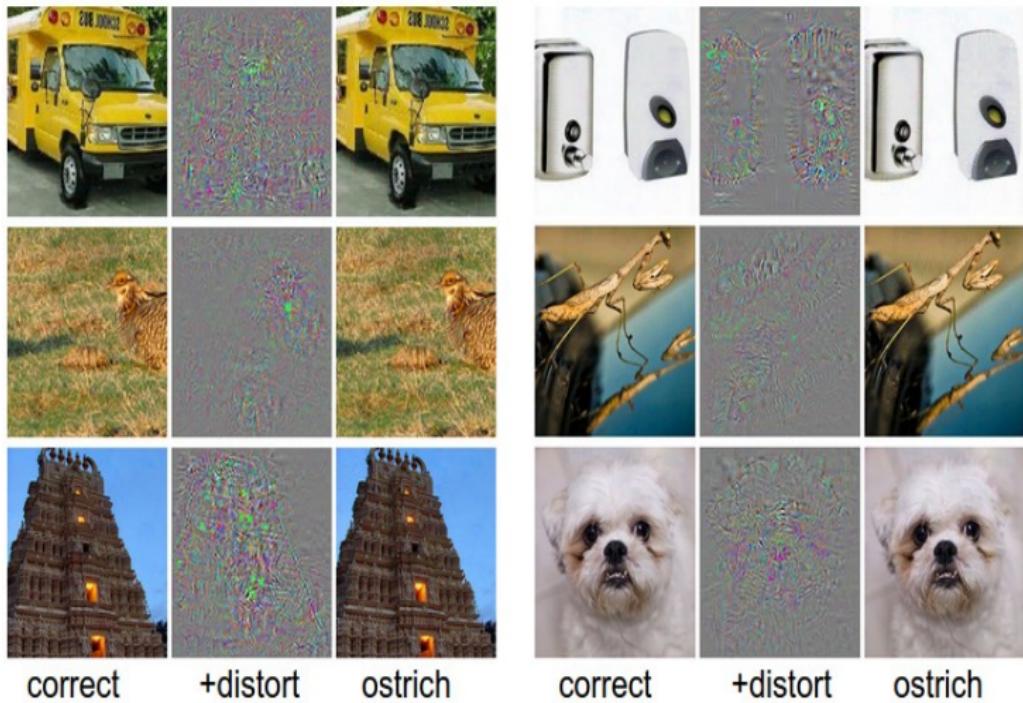


changing the prediction

"This is a plane !"

from Ducoffe and Precioso

Seeing ostrich everywhere



correct

+distort

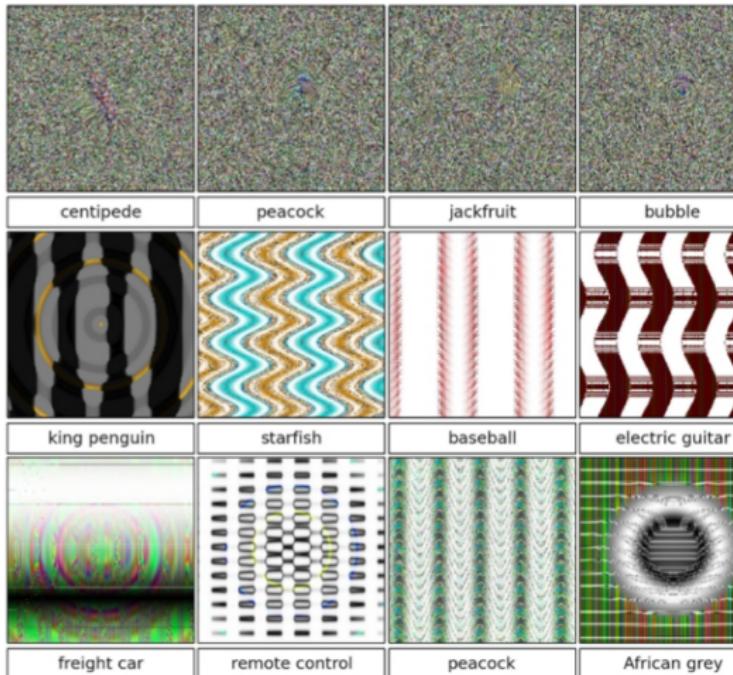
ostrich

correct

+distort

ostrich

Starting from noise



confidence $\geq 96\%$

Changing only one pixel



Cup(16.48%)
Soup Bowl(16.74%)



Bassinet(16.59%)
Paper Towel(16.21%)



Teapot(24.99%)
Joystick(37.39%)



Hamster(35.79%)
Nipple(42.36%)

How to generate an adversarial image?

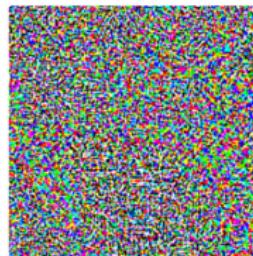
- The simplest method : Fast Gradient Sign Method (FDSM) :

$$I_{adv} = I - \epsilon \text{sign}(\nabla_I J(I, y_{target}))$$

- from Taylor development at order 1 of the loss function J
 - $J(I_{adv}) = J(I) + (I_{adv} - I)^T \nabla_I J(I)$
 - maximize $J(I_{adv})$ subject to $\|I_{adv} - I\| < \epsilon$
- Try to modify as less as possible the initial image I
- ϵ is the smallest value that leads to change the label
- Needs to know the loss function J or all classes probabilities



+ .007 ×



=



x

“panda”

57.7% confidence

$\text{sign}(\nabla_x J(\theta, x, y))$

“nematode”

8.2% confidence

$x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$

“gibbon”

99.3 % confidence

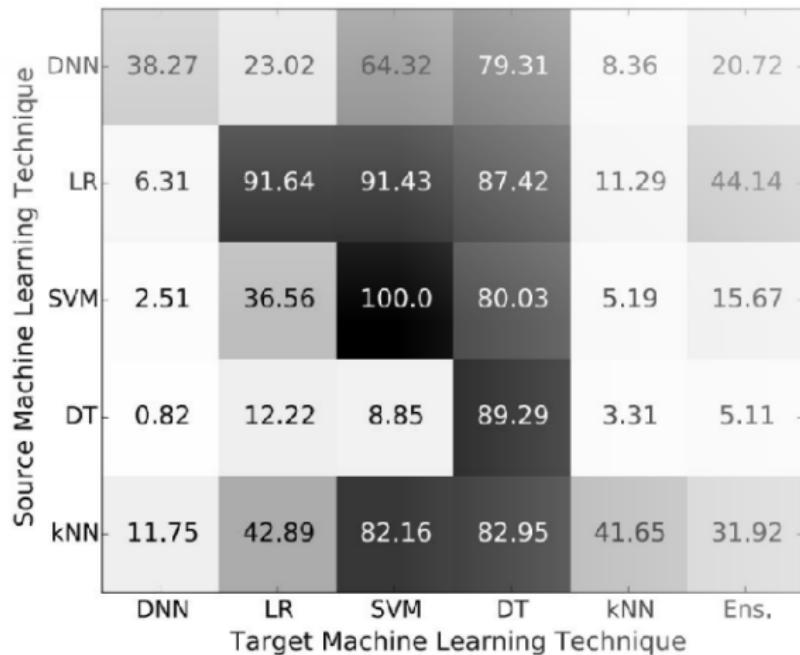
Is this particular to CNN ?

- many networks : CNN, RNN, MLP, ..
- but also other algorithms :
 - linear regression
 - linear SVM
 - Random Forest
 - reinforcement learning
 - ...

- gradient is proportional to weights
 - \Rightarrow use weights corresponding to the target class
- the same filter for all images (universal perturbations)



Cross-technique transferability



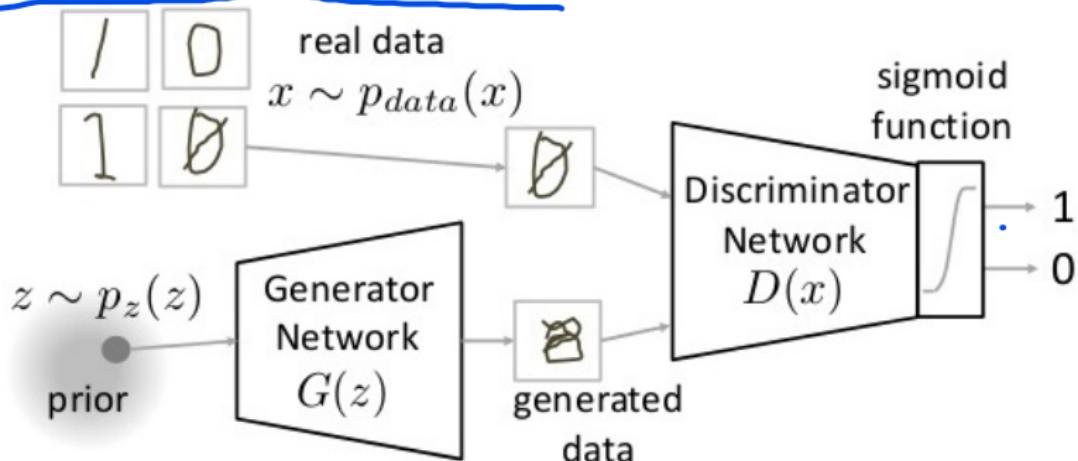
(Papernot 2016)

Adversarial images for black box algorithms

- For one image, you only get the highest score class
- Try to pass as lot as possible images through this black box and try to understand the way it works
- Use your own network/algorithm to compute adversarial images that will also probably work with this black box

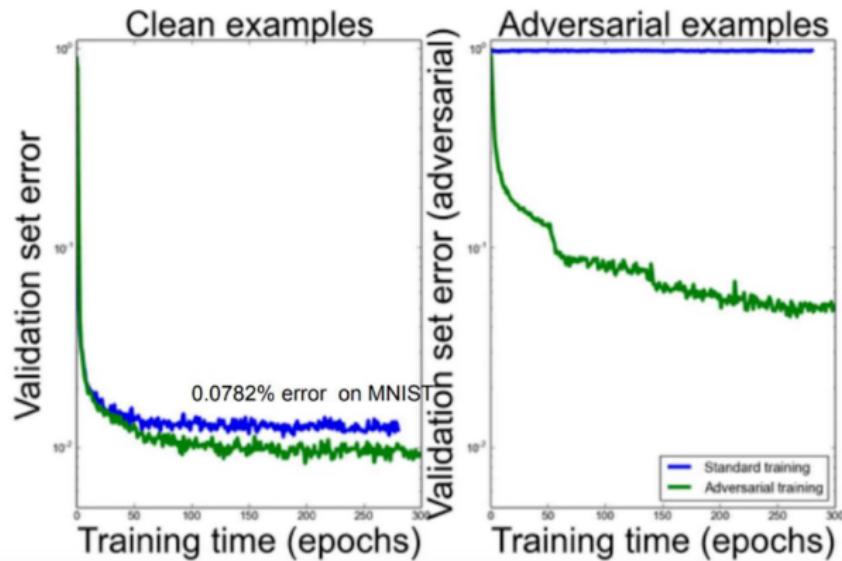
How to resist?

- Train another network to recognize adversarial images
 - it is what GAN were intended to do



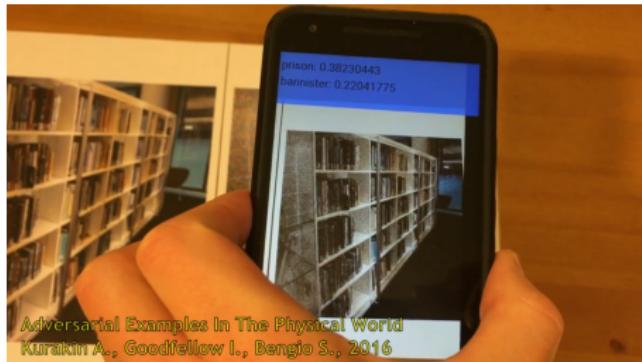
How to resist?

- Incorporate adversarial images into the training database



OK, but that's all about
synthesized images

Taking picture from adversarial images



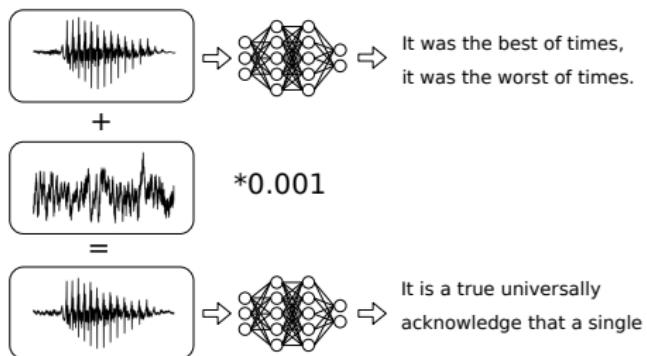
Adversarial examples in the physical world from Alexey Kurakin, Ian Goodfellow, Samy Bengio, 2016
https://youtu.be/zQ_uMenoBCk

Adversarial objects



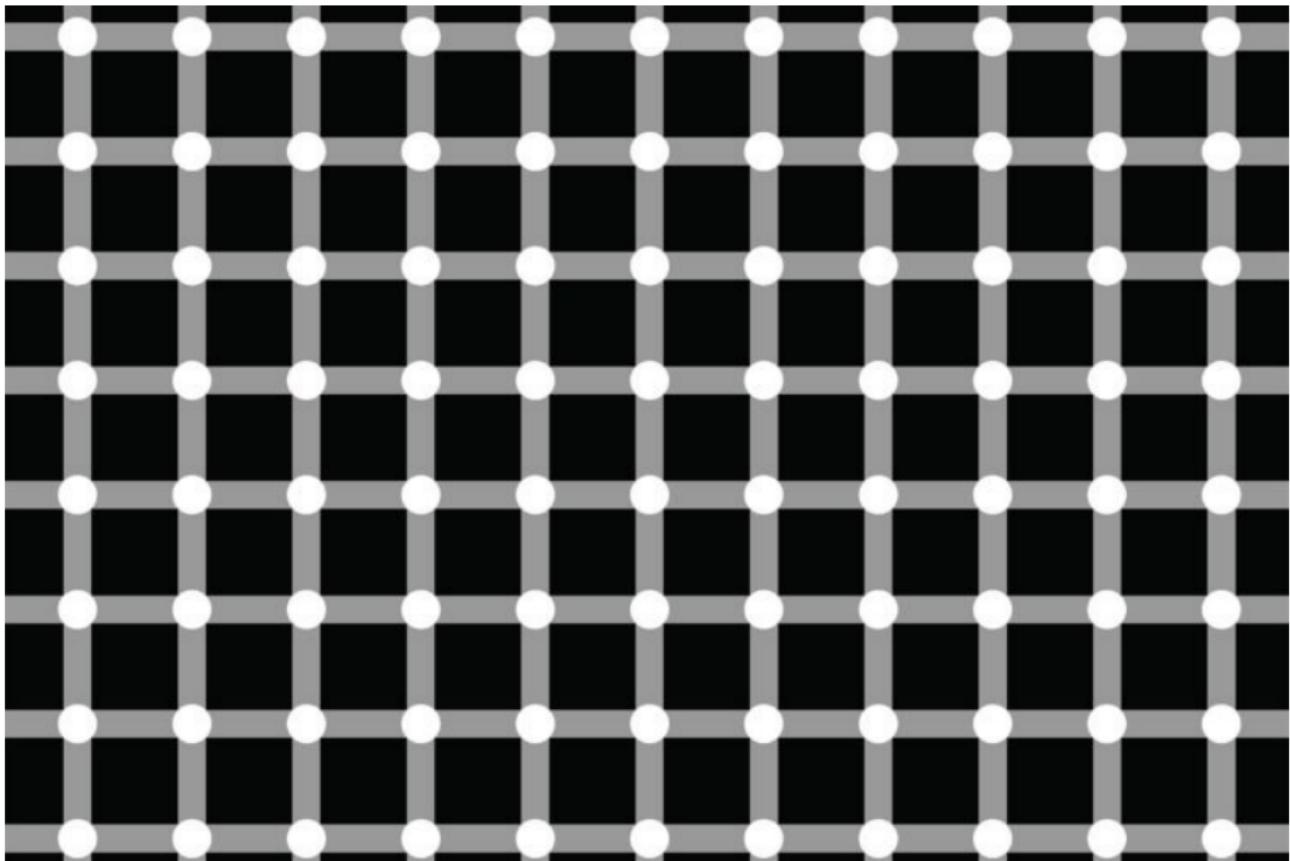
from <https://www.csail.mit.edu/news/fooling-neural-networks-w3d-printed-objects>

Adversarial sounds



from https://nicholas.carlini.com/code/audio_adversarial_examples/

We also, as human ...



Both human and computer

