CNN: Convolutional Neural Networks

Diane Lingrand and many contributors

UNIVERSITÉ
CÔTE D'AZUR
Master Data Science M1

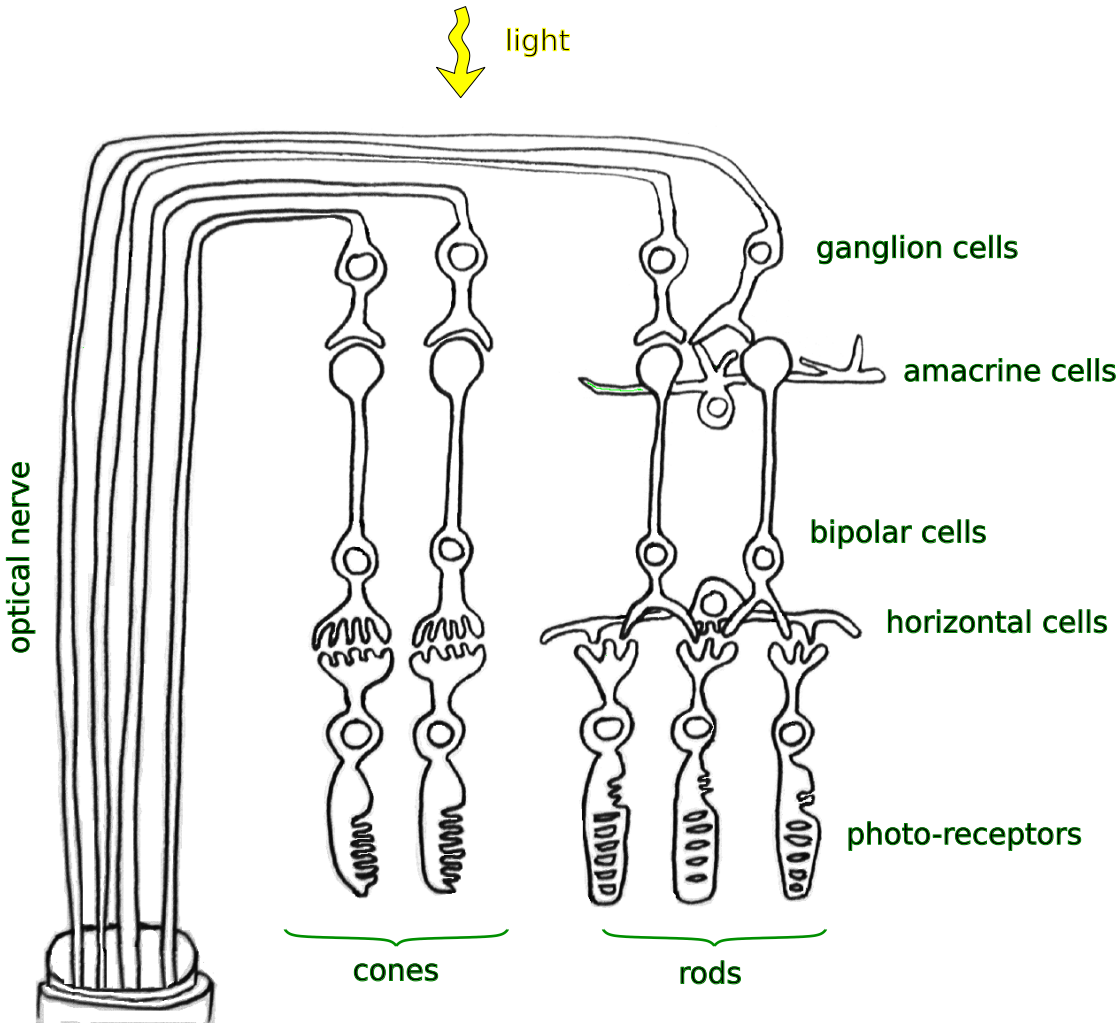
2021 - 2022

Outline

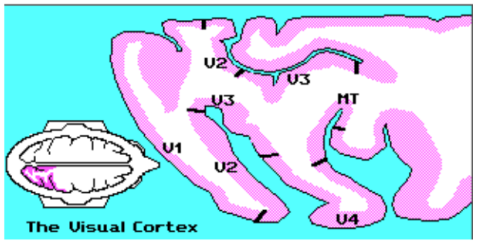
Outline

Deep Architecture in the Brain : the retina

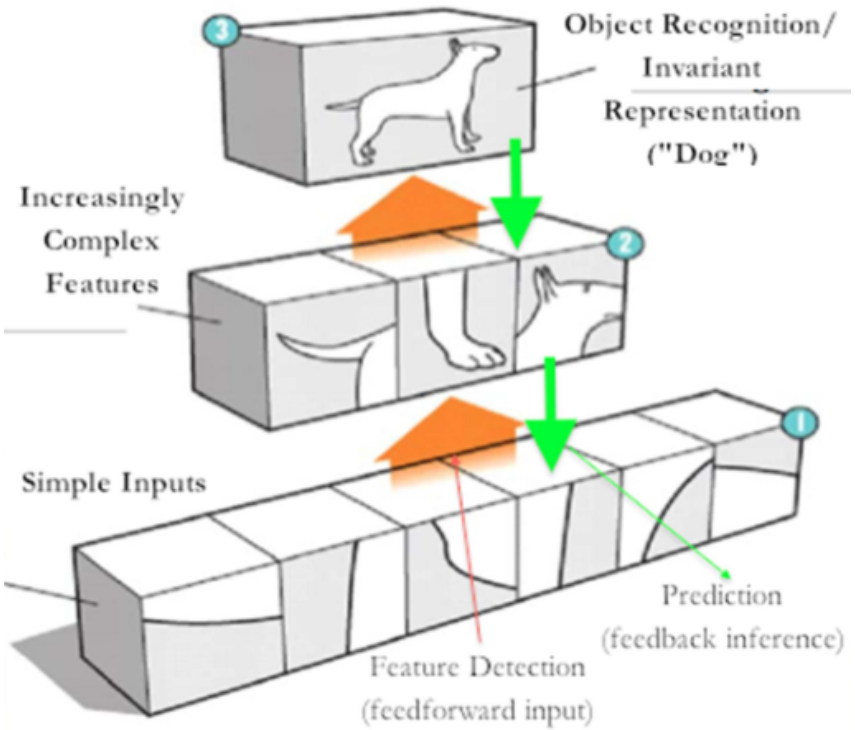
- photoreceptor cells : cones and rods
- bipolar cells (first neuron) : link to one or several photoreceptors
- ganglion cells (second neuron) link to one or several bipolar cells
- optical nerve (blind spot)



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



Motivation

- 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

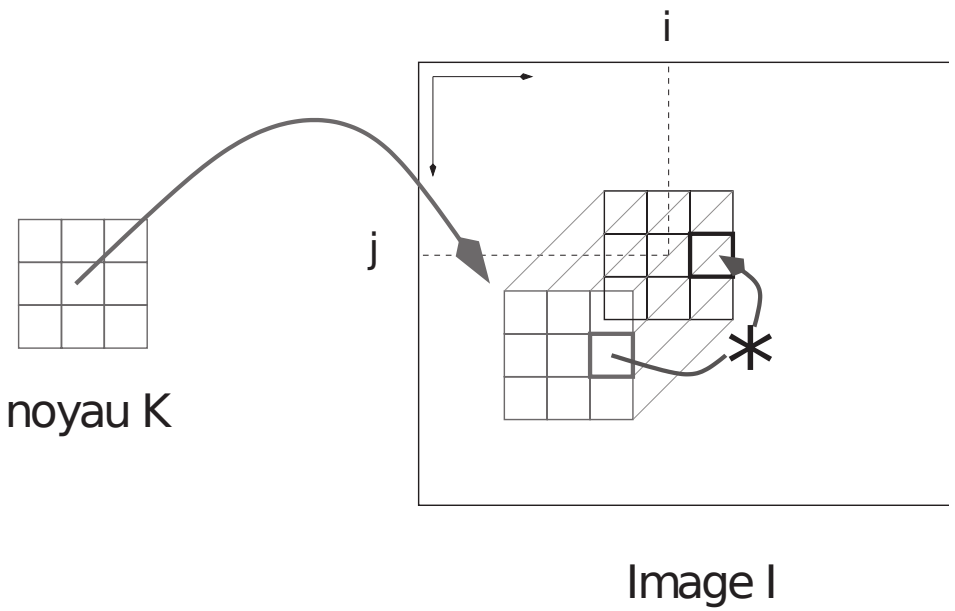
Back to convolution operator

- For f and g, discrete functions :

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

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

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

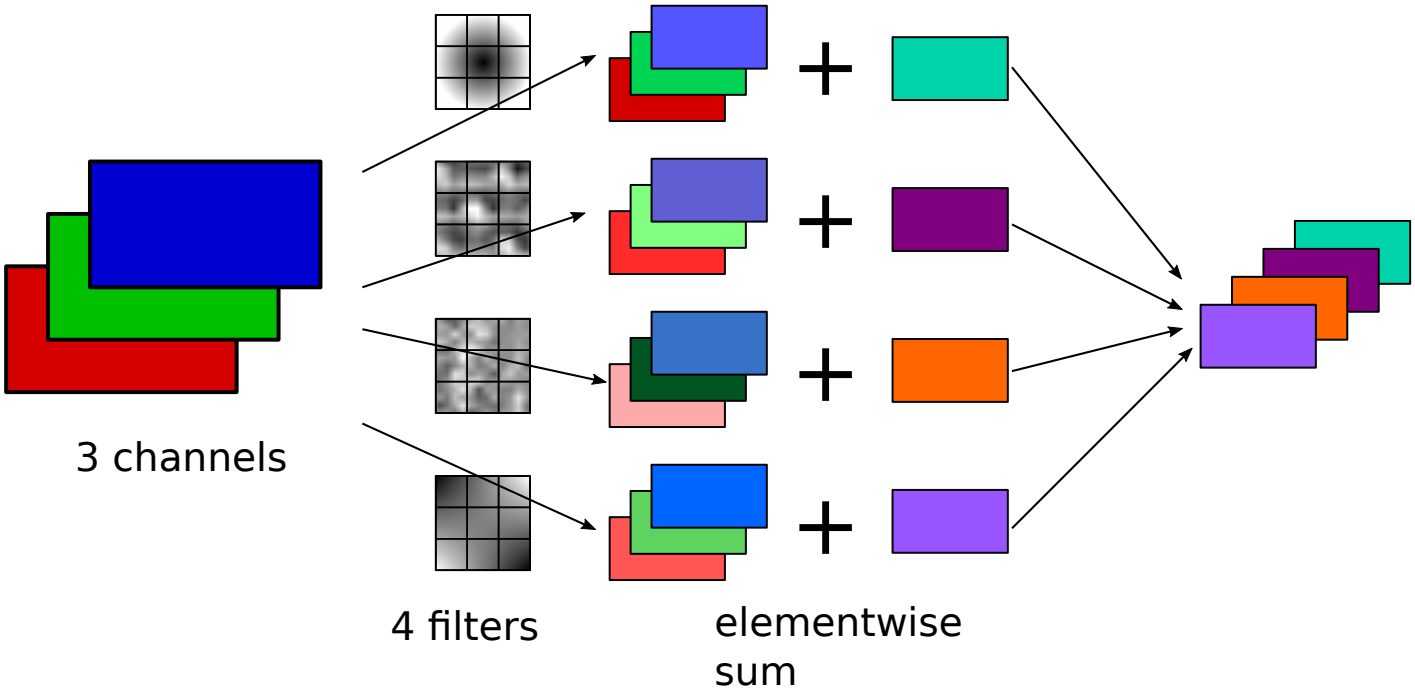


Idea of CNN : convolution layer

- 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 3x3 filter : 10 coefficients (9 for filter + 1 for bias)
- in keras : `Conv2D` instead of `Dense`
  - parameters : number of filters, size of filters (usually $k * k$)
  - 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



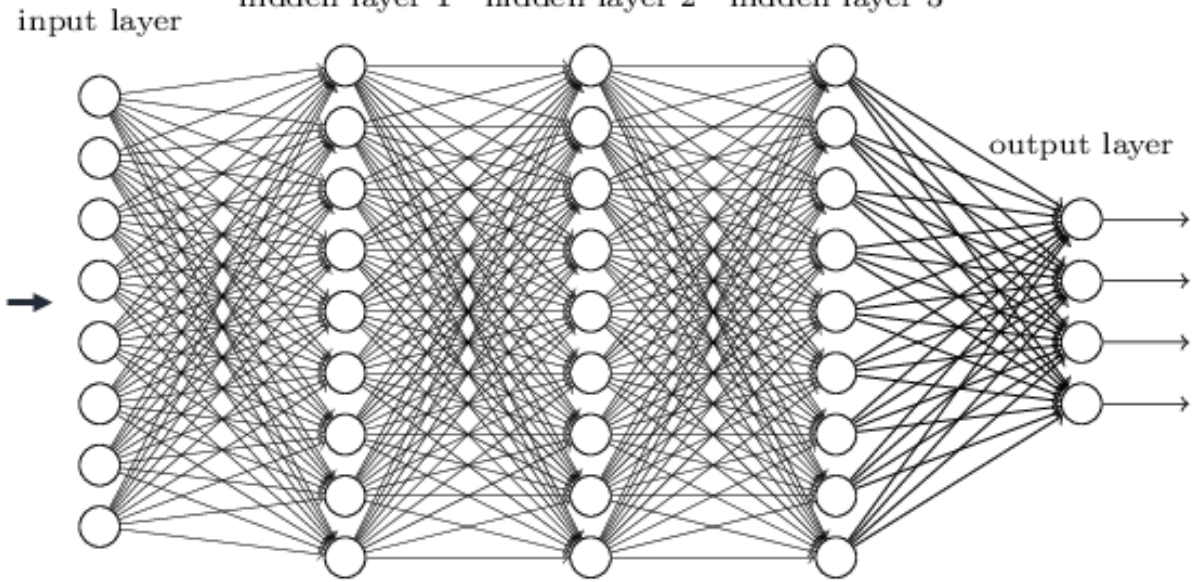
See `https://arxiv.org/pdf/1603.07285v1.pdf` for a complete discussion about dimensions

CNN : Convolutional Neural Network

- many convolutional layers
  - in order to extract the characteritics 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



Activation functions

- 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, ...

Softmax layer

- smooth approximation to the arg max function
  - assign probabilities to indices of having the highest response $z = [z_0 z_1 z_2 ... z_n]$ for each i,

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

  - every value in [0 1] and the sum is equal to 1

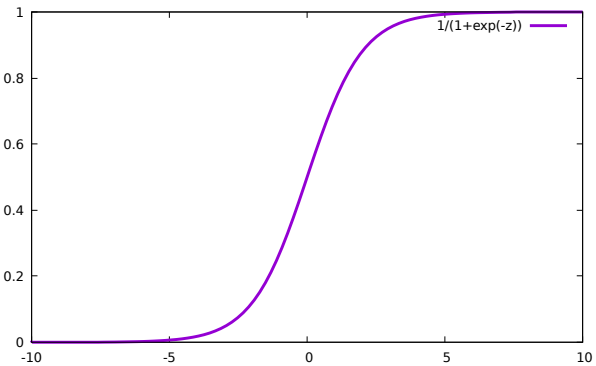
Why ReLU ?

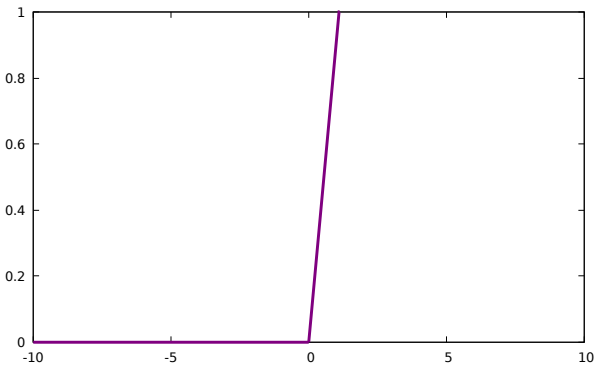
- problem of vanishing gradient
  - remember the delta rule and gradient descent :

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

and

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





- 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)

Learning a CNN

- 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, ...)

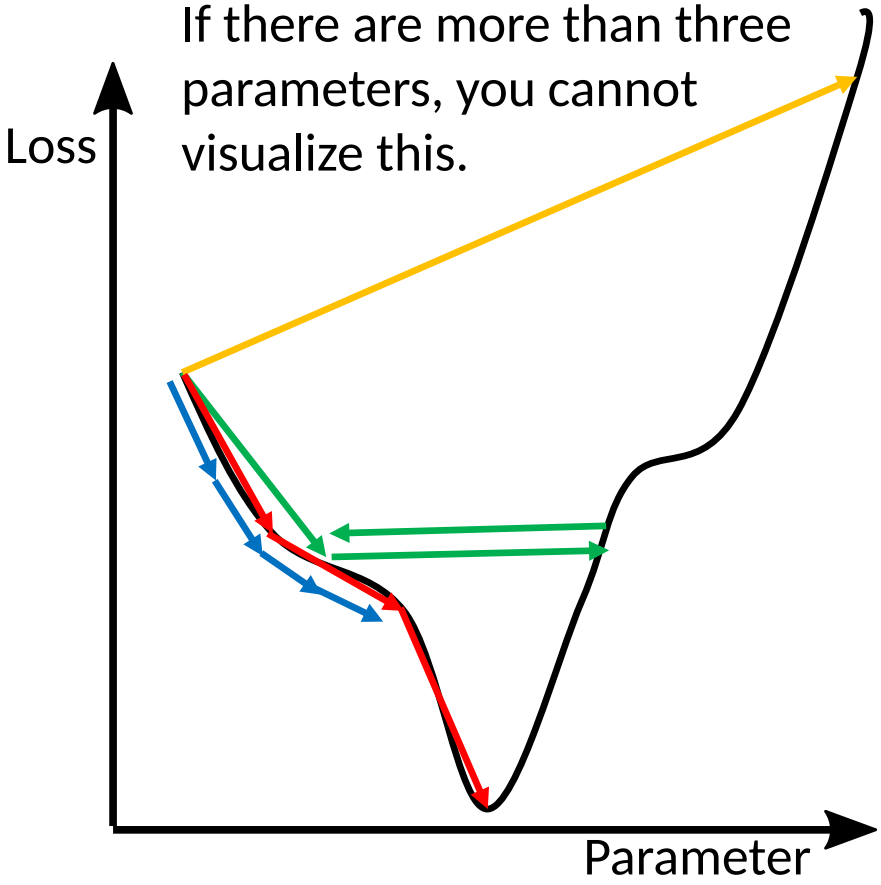
dropout

- some neurons are switched off
  - randomly chosen at each iteration
  - they do not contribute anymore to the output
  - there 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 ...

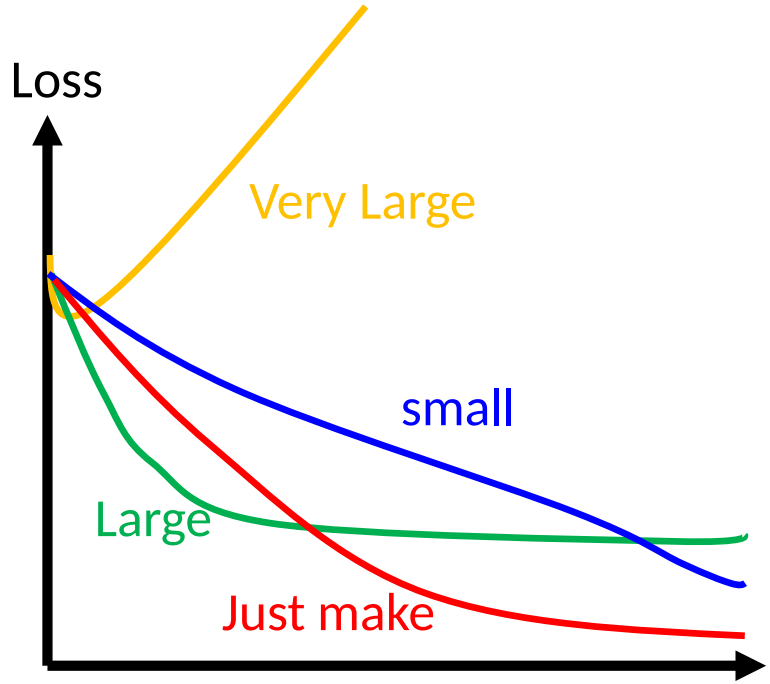
batch normalisation

- 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

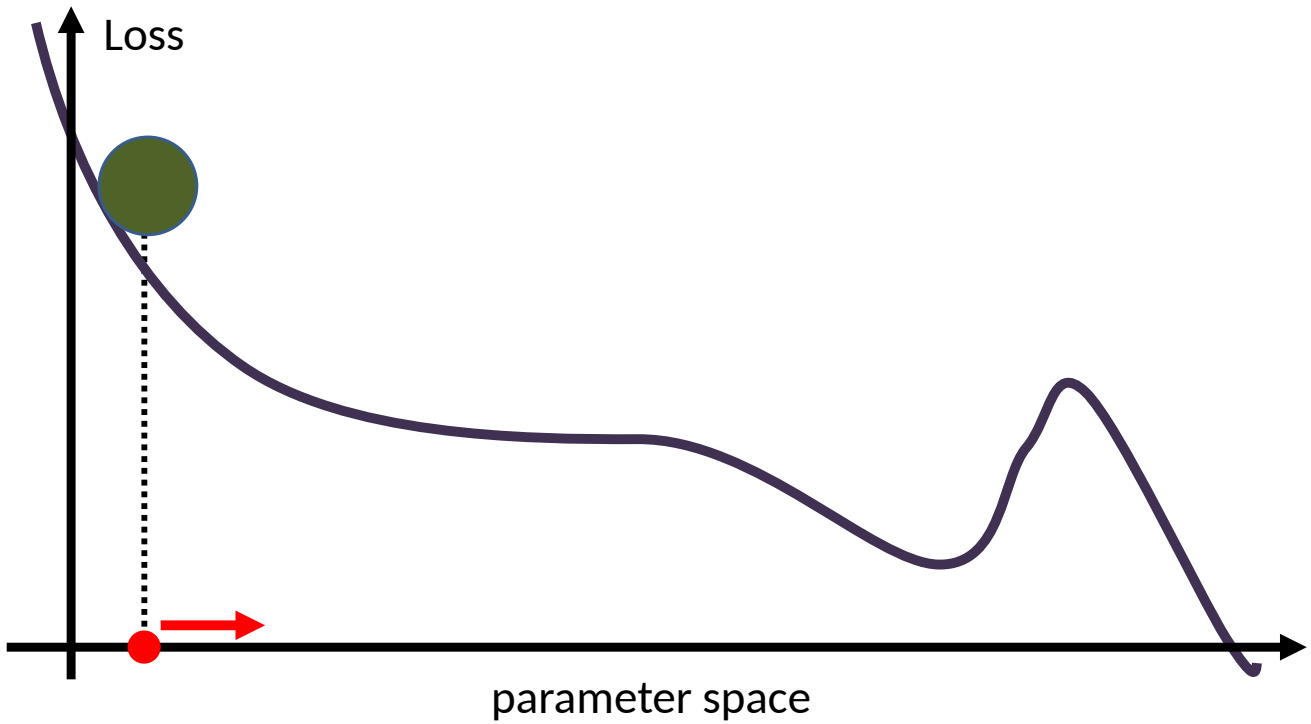


Set the learning rate η carefully

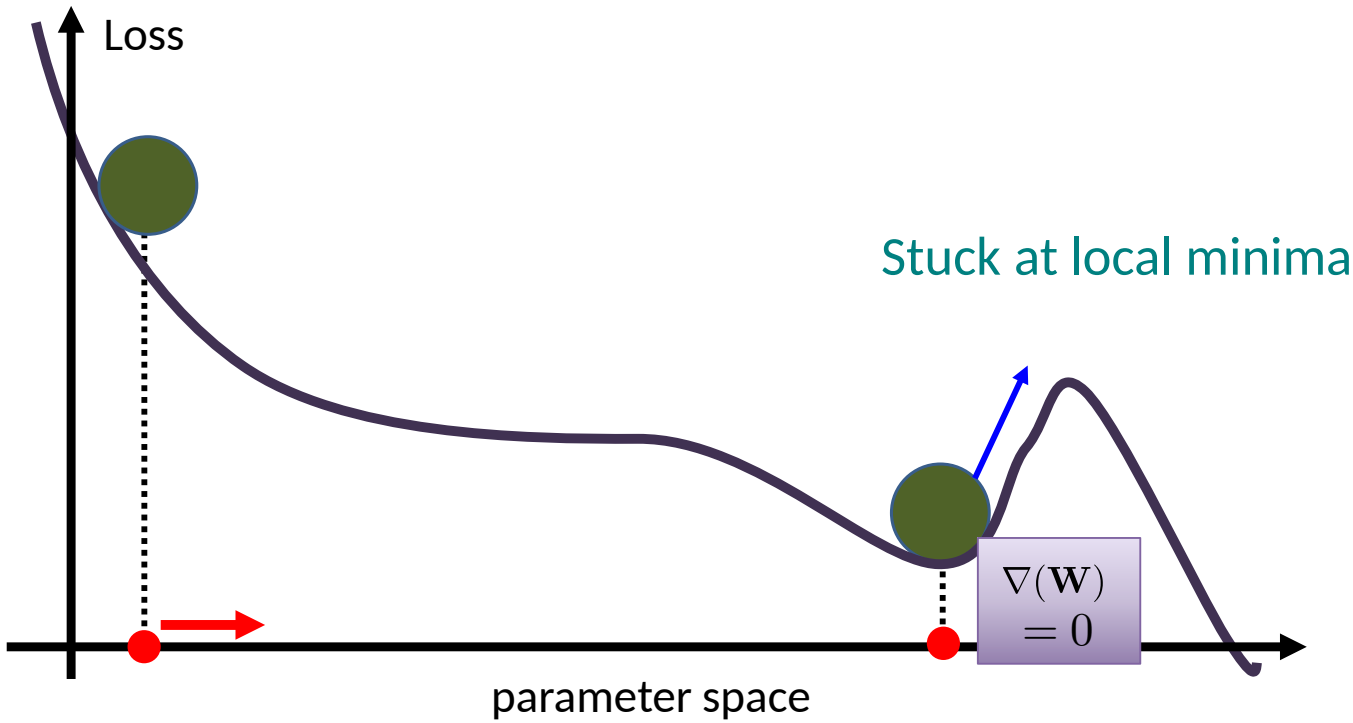


But you can always visualize this.

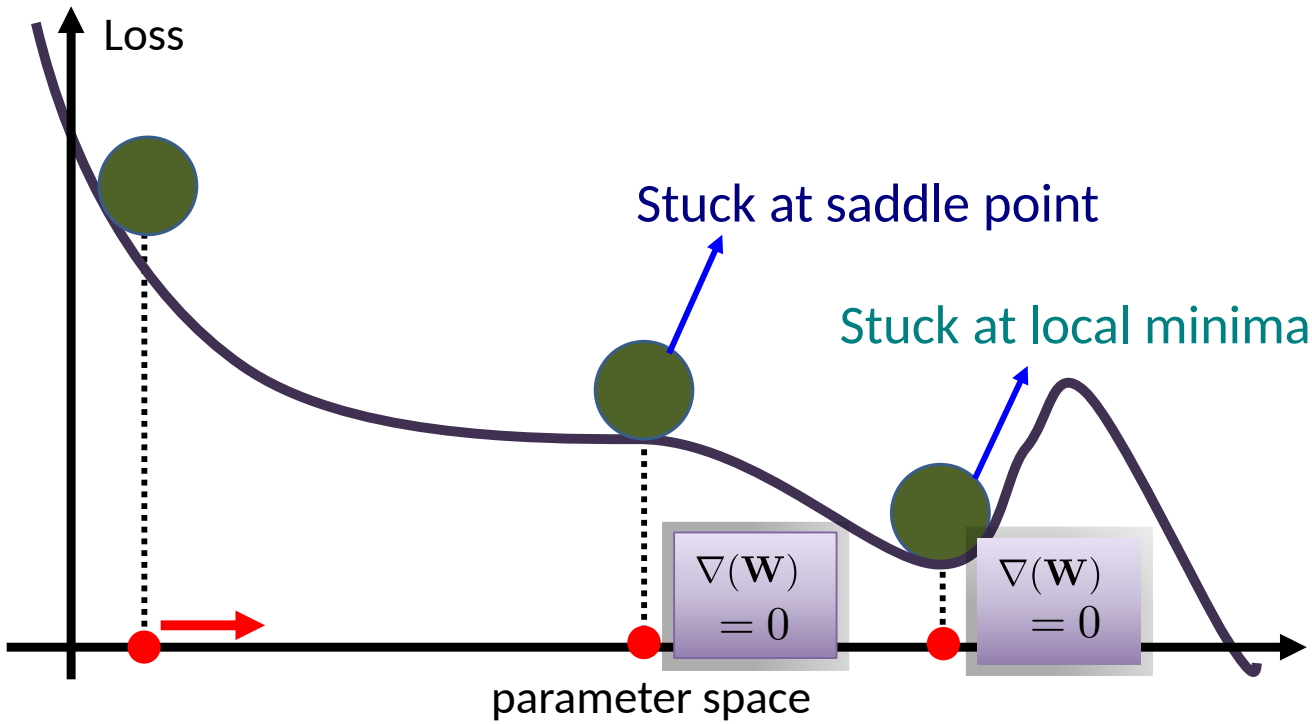
Momentum : avoid local minima



Momentum : avoid local minima

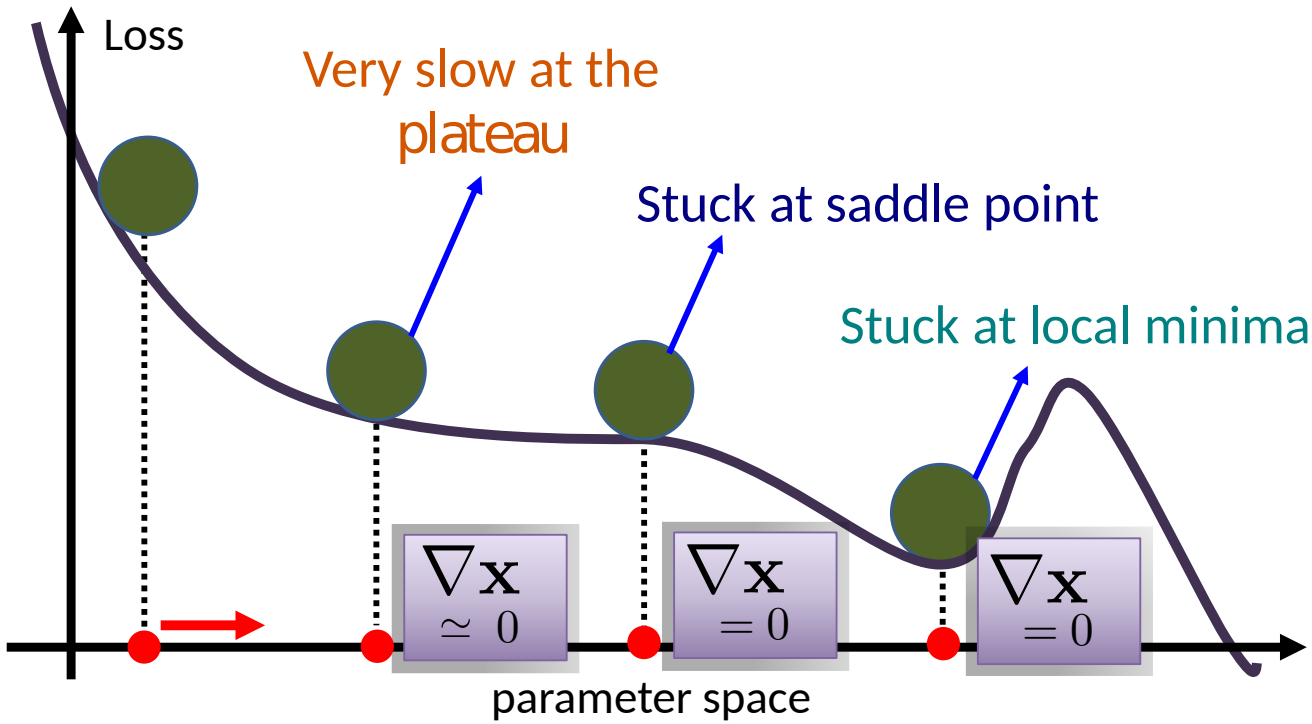


Momentum : avoid local minima

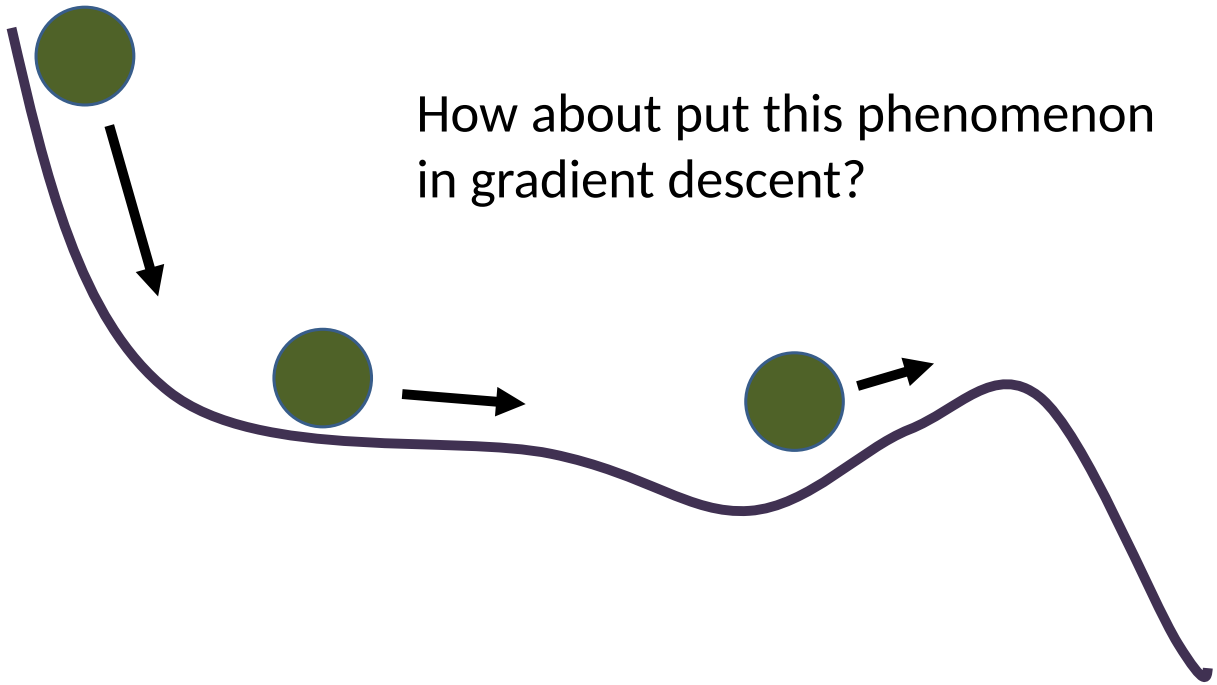


123

Momentum : avoid local minima

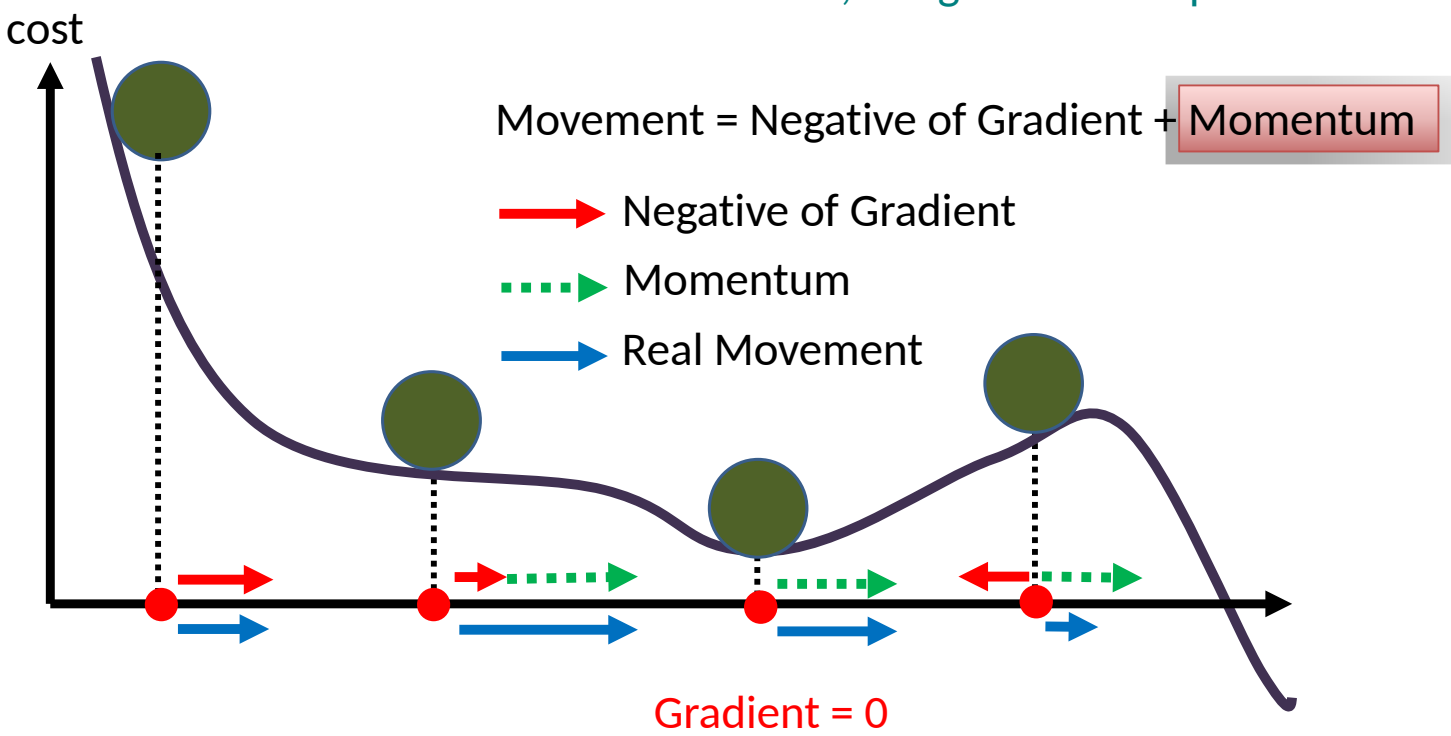


Momentum : avoid local minima



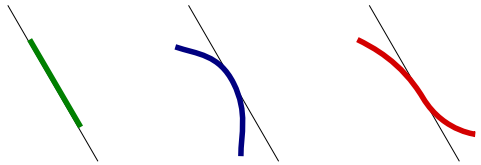
Momentum : avoid local minima

Still not guarantee reaching global minima, but give some hope .....



Local minima - Pathological curvatures

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

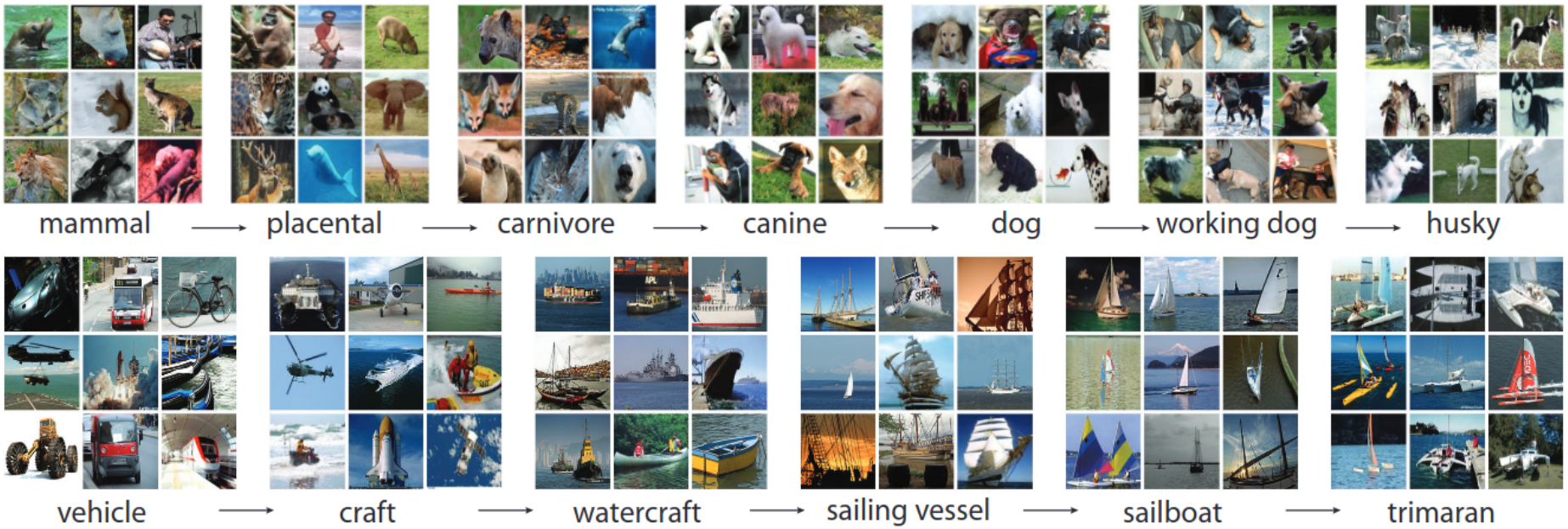


- Momentum
  - accumulates the gradient of the previous steps in order to compute the new weights
- RMSProp (Root Mean Square Propogation)
  - different learning rates for each weight
- AdaM (Adaptive Moment Optimization)
  - combining both Momentum and RMSProp

Outline

ImageNet

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



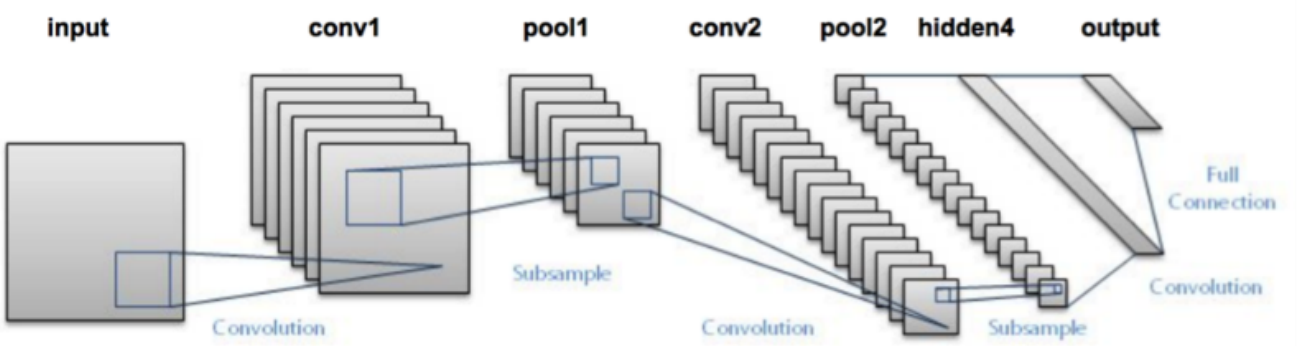
Architectures

from `https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-mor`

- LeNet 5 1998
  - by Le Cun *et al*
  - was applied to recognise hand-written numbers on checks (32x32 pixel grey images).
- AlexNet 2012 AlexNet2012.png
- 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

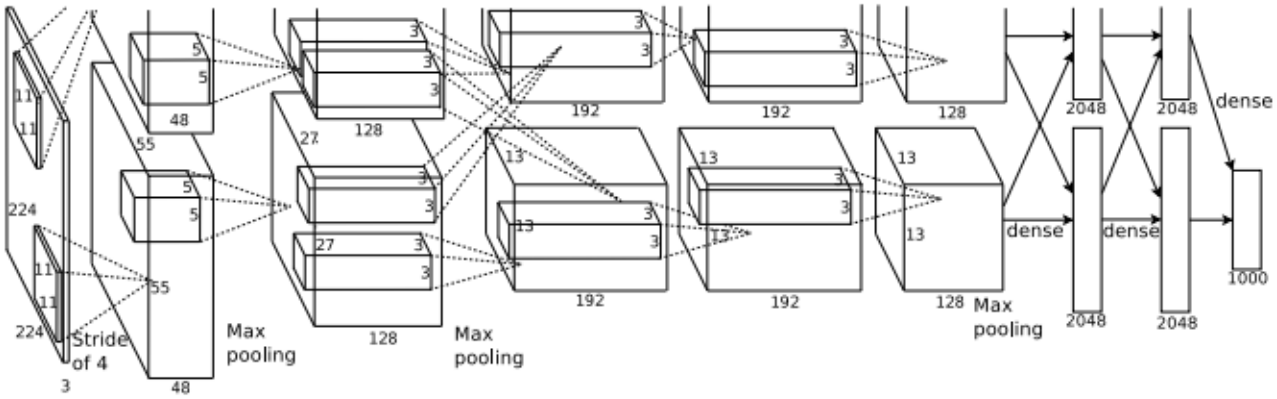
More details on `https://www.youtube.com/watch?v=Y2Tna77k1aI`.

LeNet 5, 1998



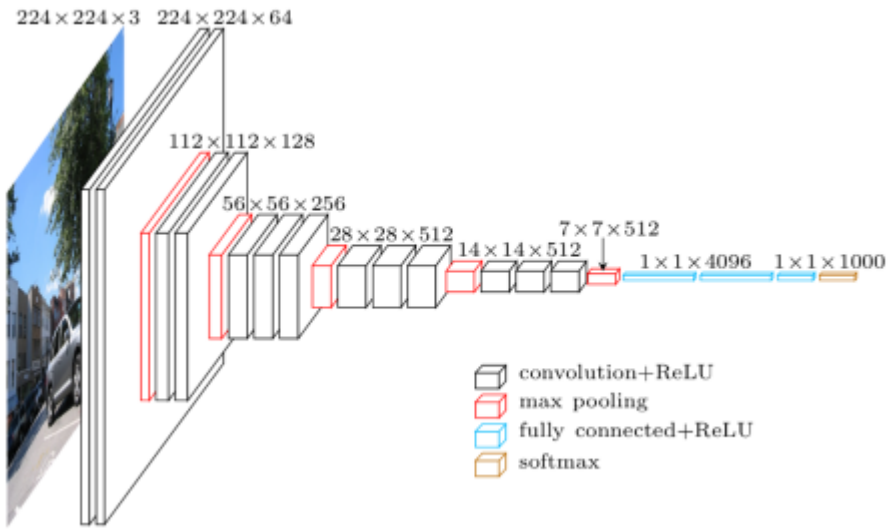
- historical
- by Le Cun et al
- was applied to recognise hand-written numbers on checks (32x32 pixel grey images).

AlexNet, 2012



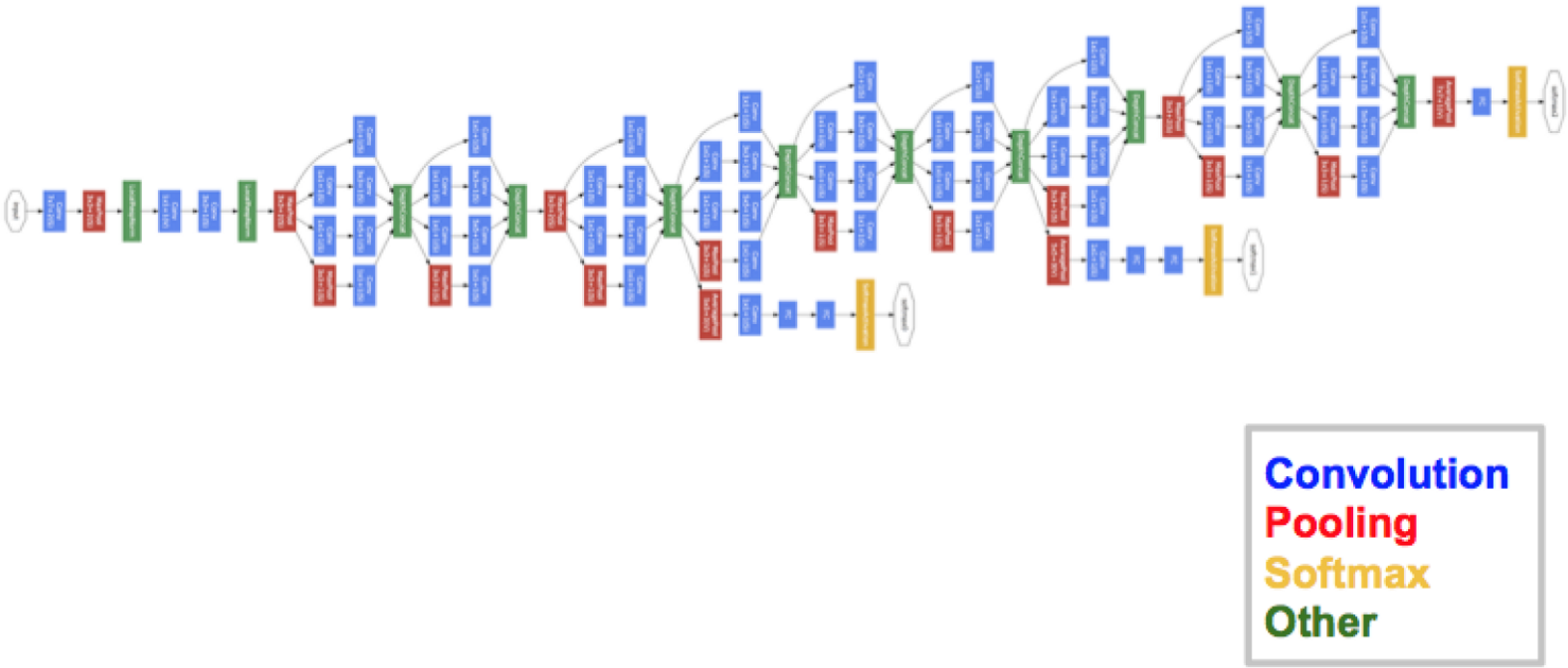
- 11x11, 5x5 and 3x3 convolutions, overlapping max pooling, local normalisation, dropout, data augmentation, ReLU activations, SGD with momentum. ReLU activations after every convolutional and fully-connected layer, 2 GPUs

VVG-16, VGG-19, 2014



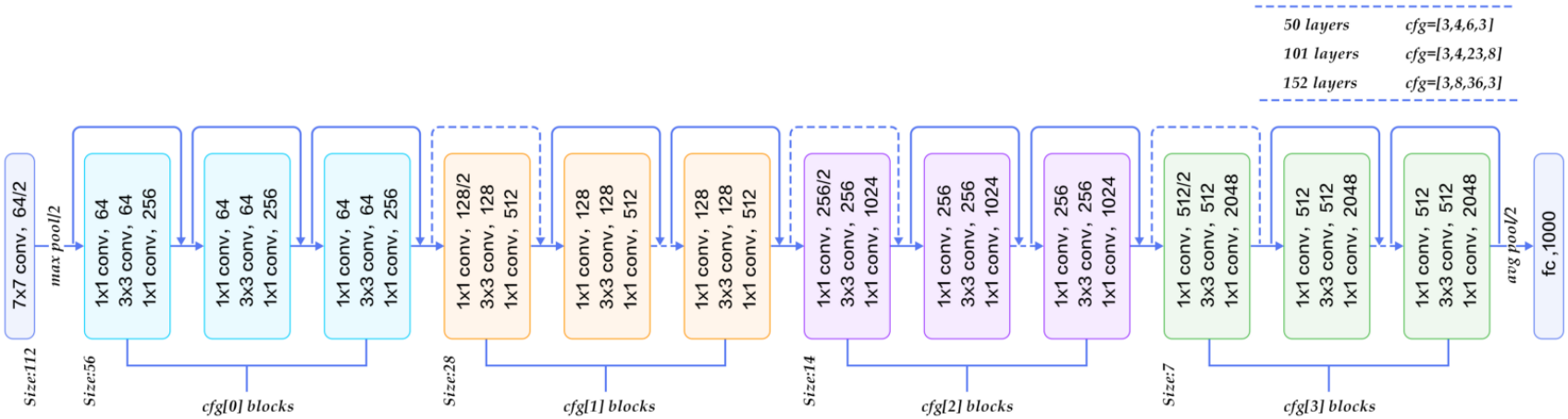
- VGG-16 : 16 refers to the number of weighted layers (conv or dense)
  - VGG-19 : 19 layers to learn
  - smaller convolution filters (less parameters)
- Simonyan and Zisserman (University of Oxford and Google Deep Mind) : `https://arxiv.org/pdf/1409.1556.pdf` - ImageNet Challenge (ILSVRC) 2014

GoogLeNet Inception v1, 2014



- 22 layers
  - Inception module (1x1, 3x3, 5x5 convolutions and 3x3 max pool.)
  - RCNN
  - Networks in Network
- v2 and v3 published in 2015
  - `https://arxiv.org/pdf/1512.00567v3.pdf`

ResNet, 2015



- introduce residual connections or shortcut connections

a residual block learns easily the identity function which is not the case for other layers (when very deep). If we add a res block, at least, it could not perturb the result but we hope it will learn something useful.

Architectures with keras

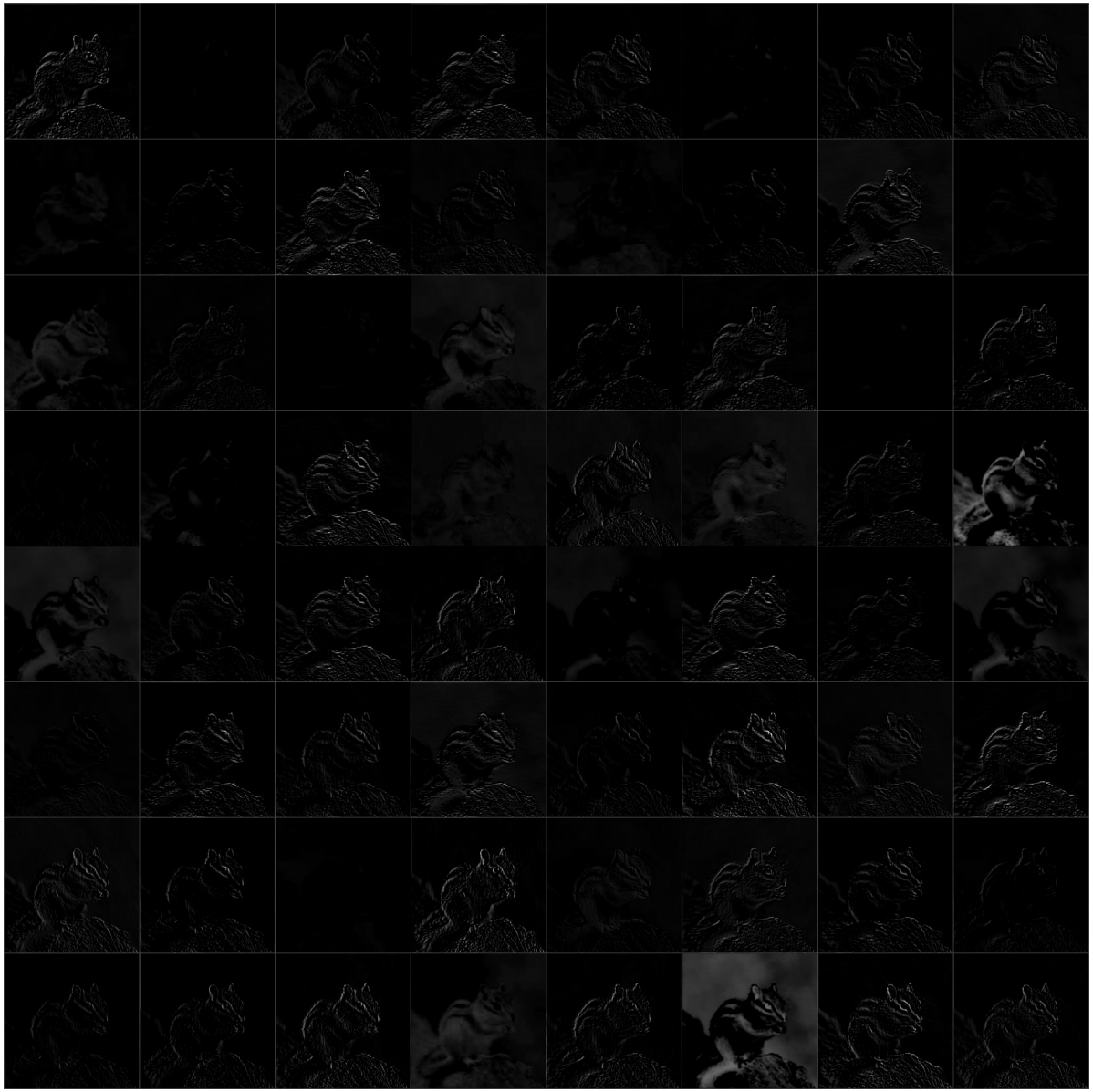
Everything is on : `https://keras.io/api/applications/`

Example of VGG16

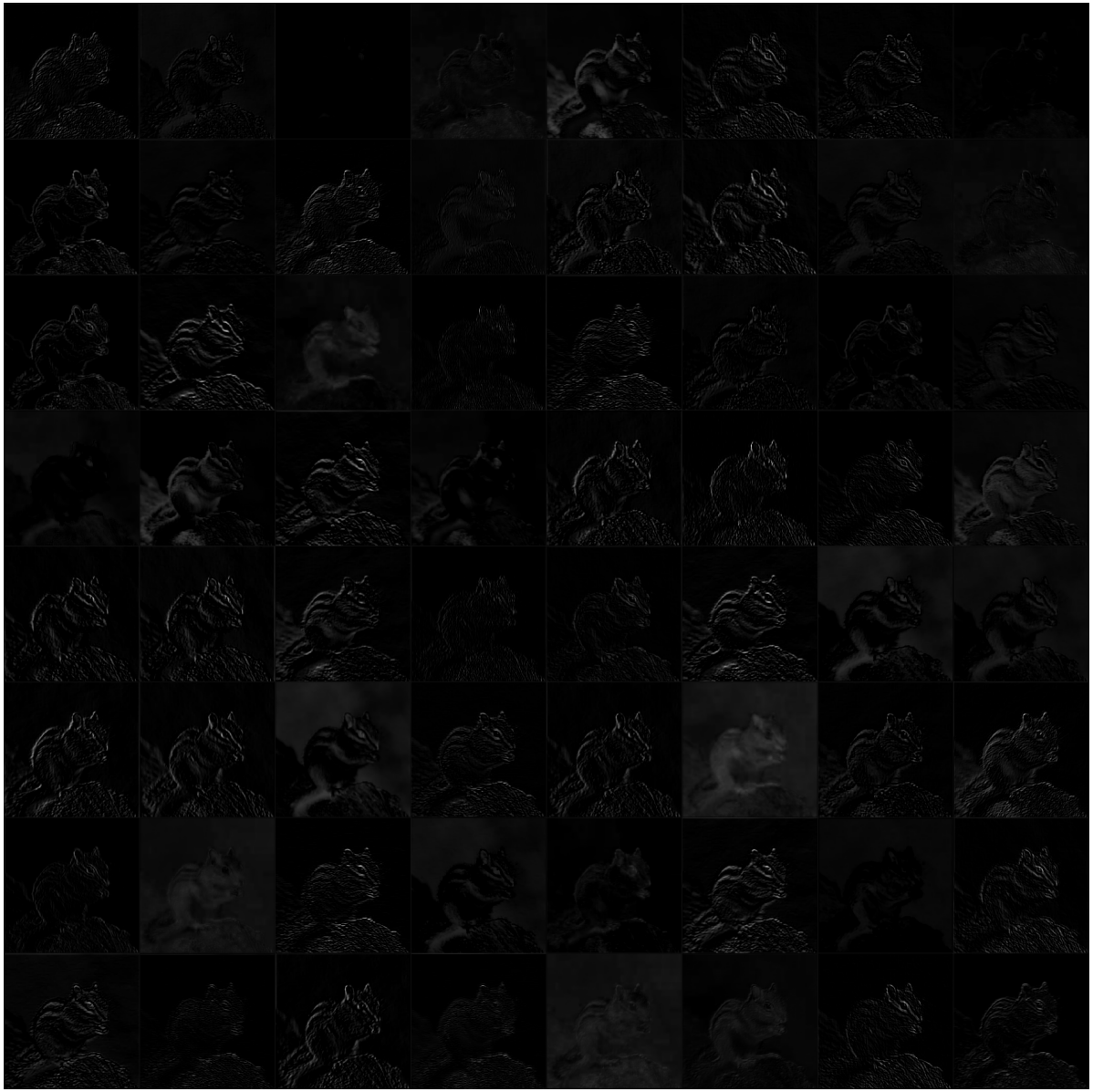
```
VGGmodel = VGG16(weights='imagenet', include_top=False)
VGGmodel.summary()
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(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
```

VGG16 : VGGscoiattolo_input_1

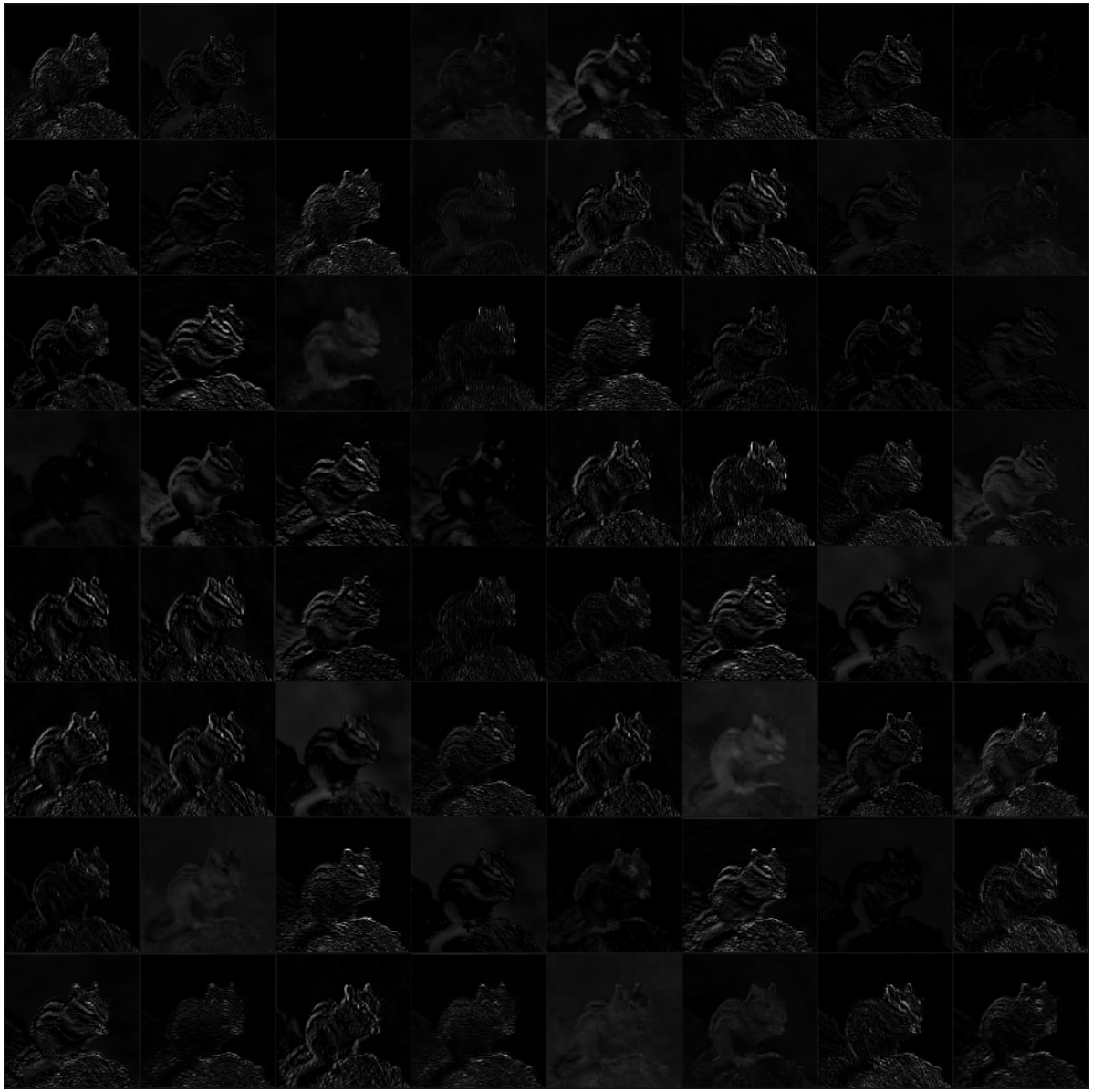
VGG16 : VGGscoiattolo_block1_conv1



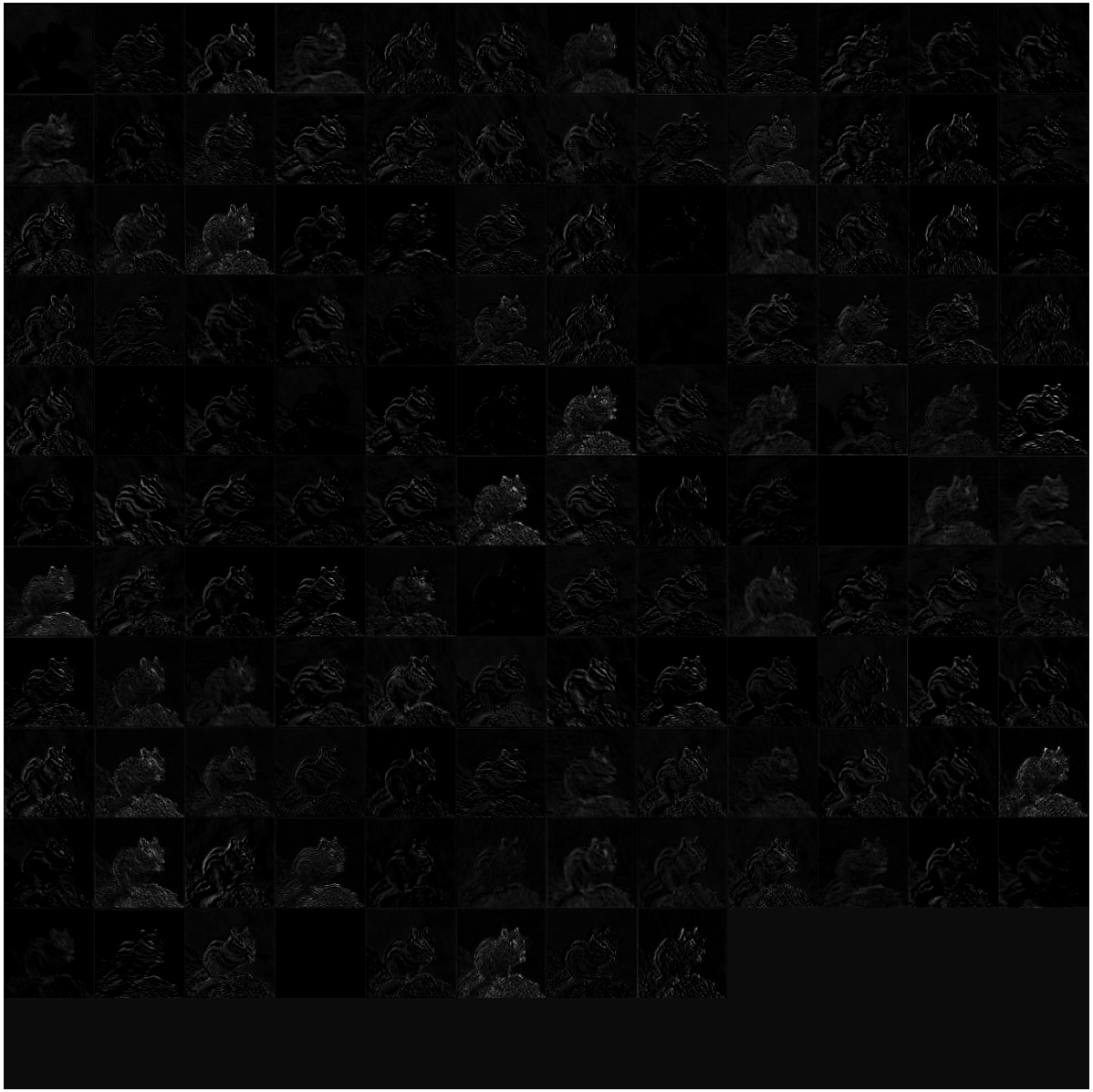
VGG16 : VGGscoiattolo_block1_conv2



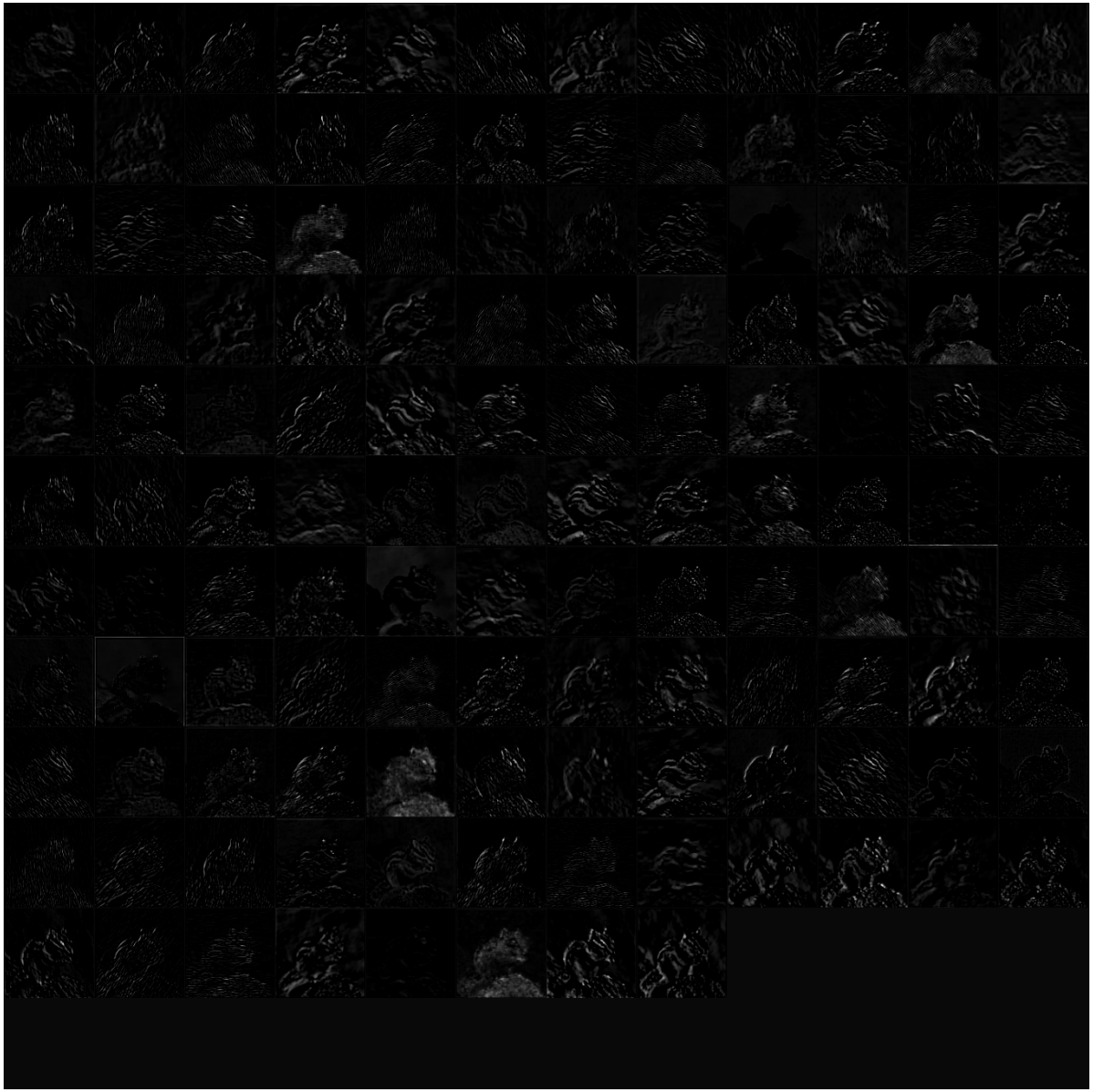
VGG16 : VGGscoiattolo_block1_pool



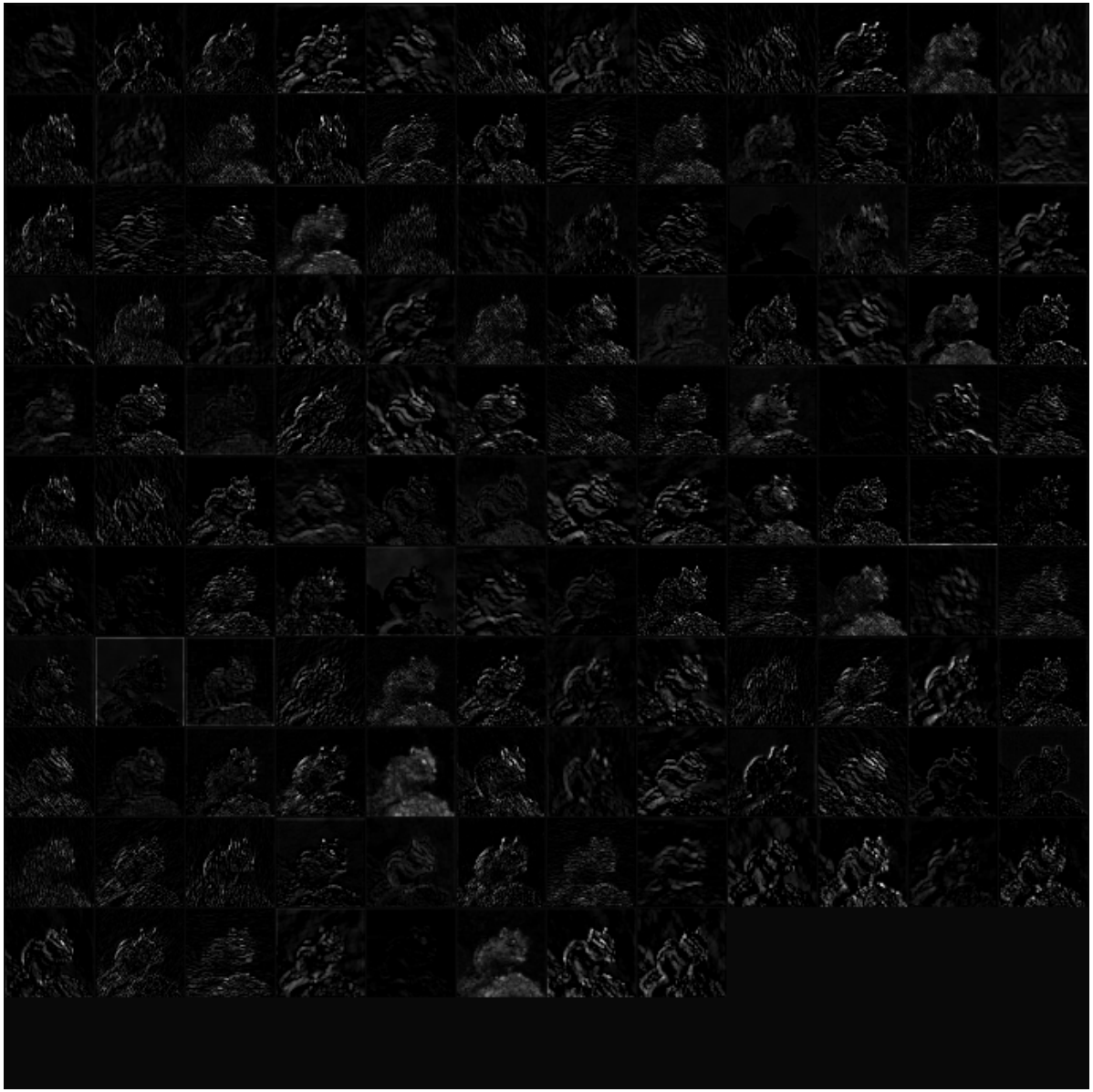
VGG16 : VGGscoiattolo_block2_conv1



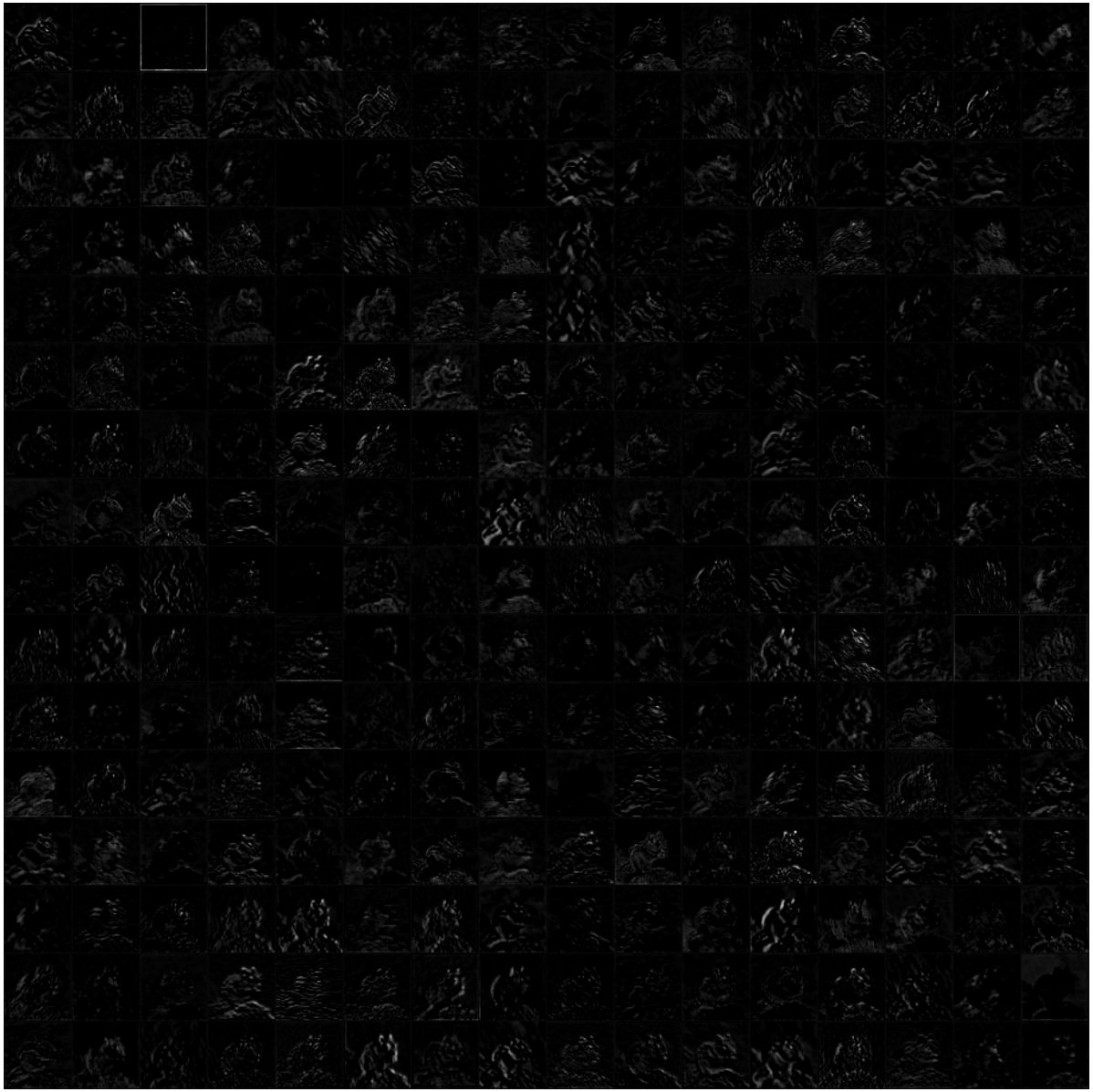
VGG16 : VGGscoiattolo_block2_conv2



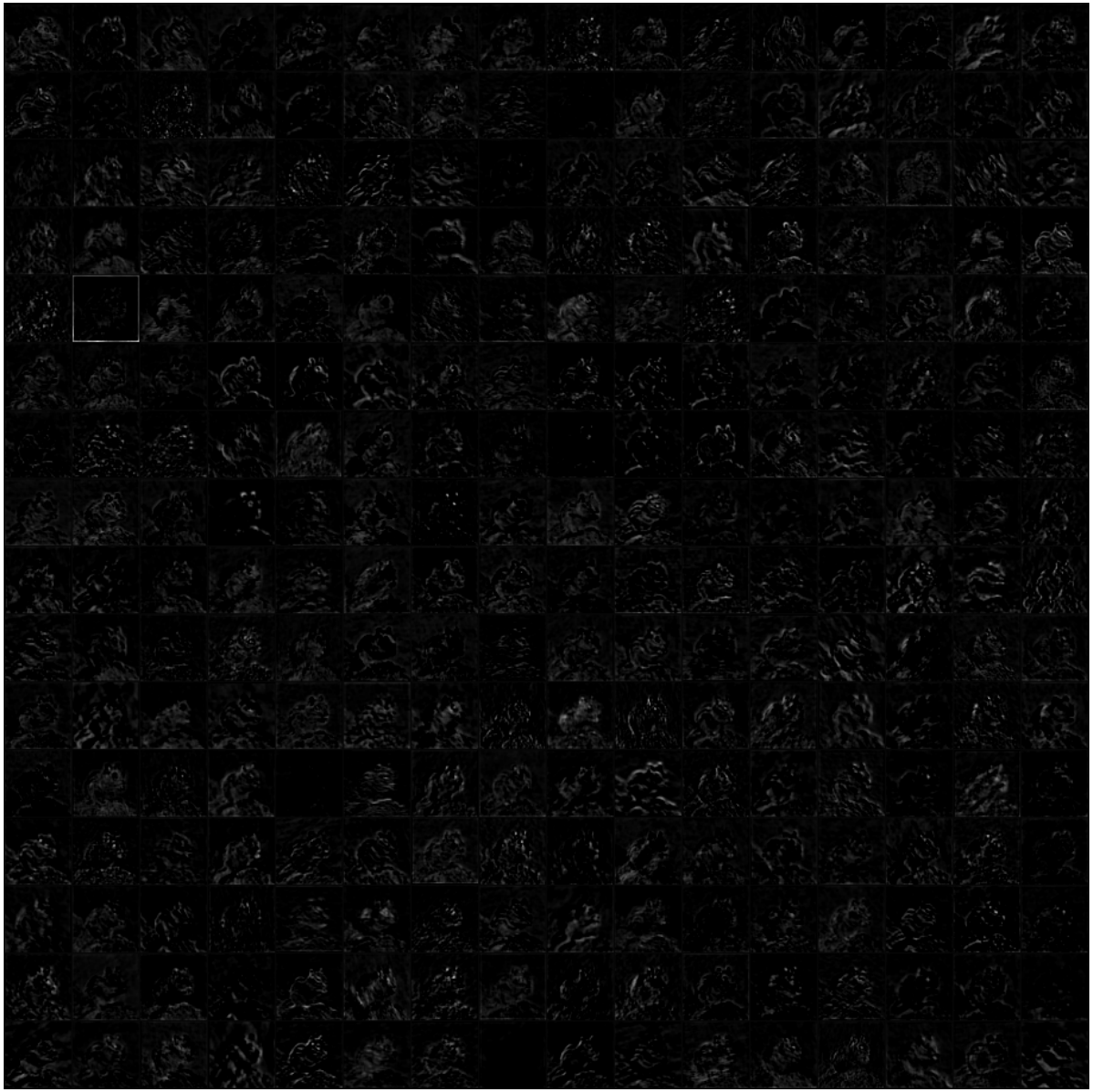
VGG16 : VGGscoiattolo_block2_pool



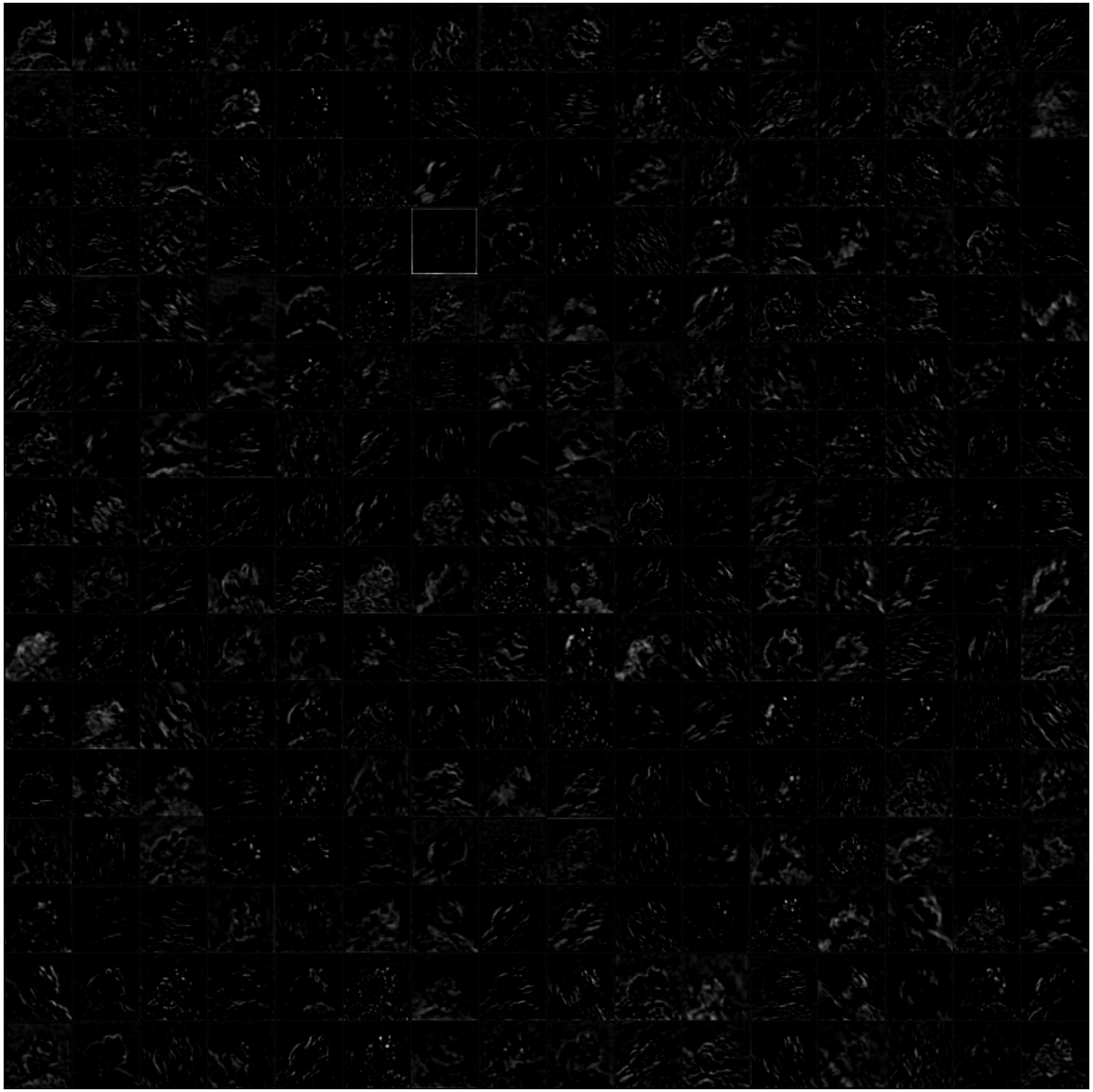
VGG16 : VGGscoiattolo_block3_conv1



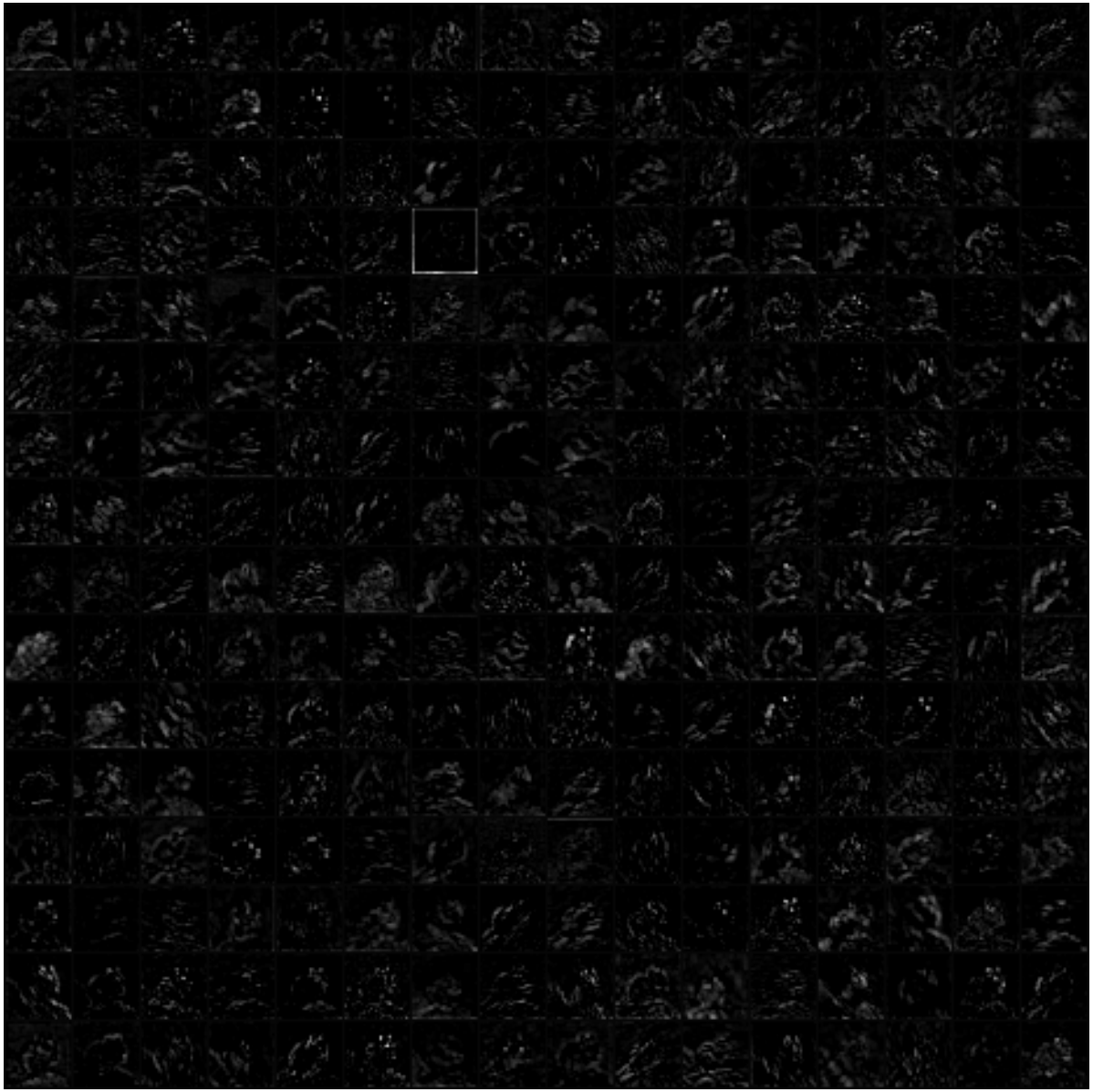
VGG16 : VGGscoiattolo_block3_conv2



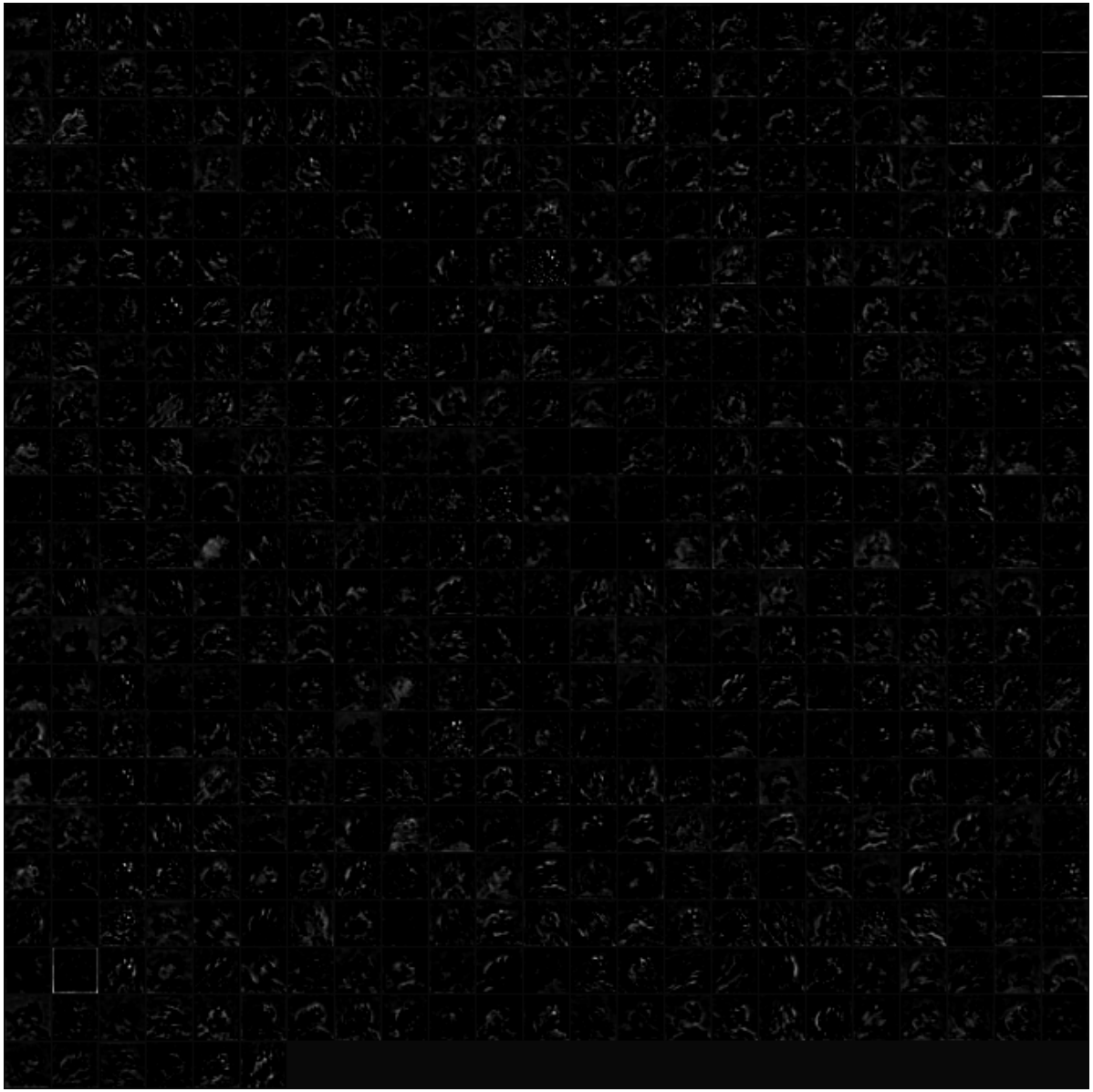
VGG16 : VGGscoiattolo_block3_conv3



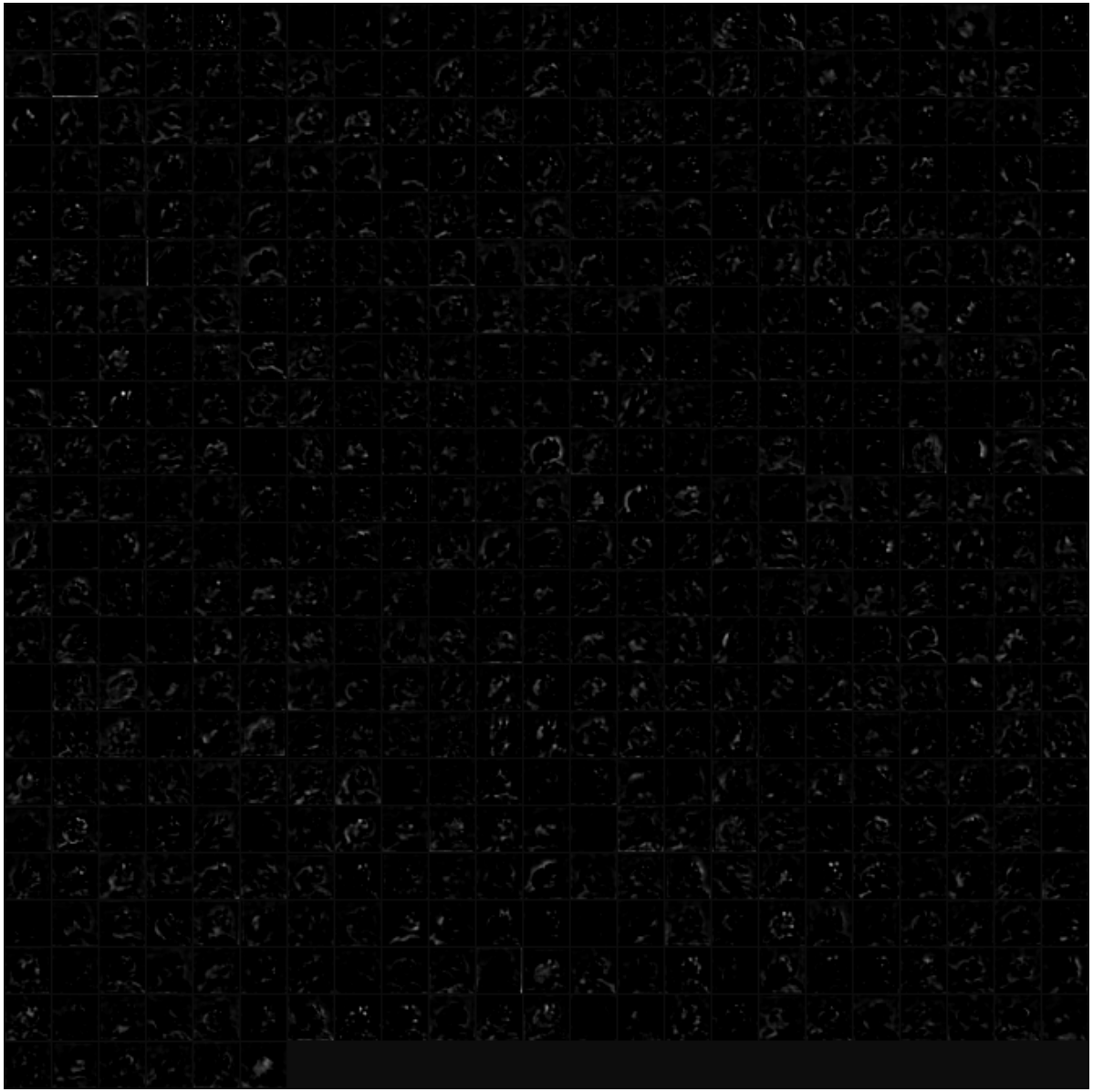
VGG16 : VGGscoiattolo_block3_pool



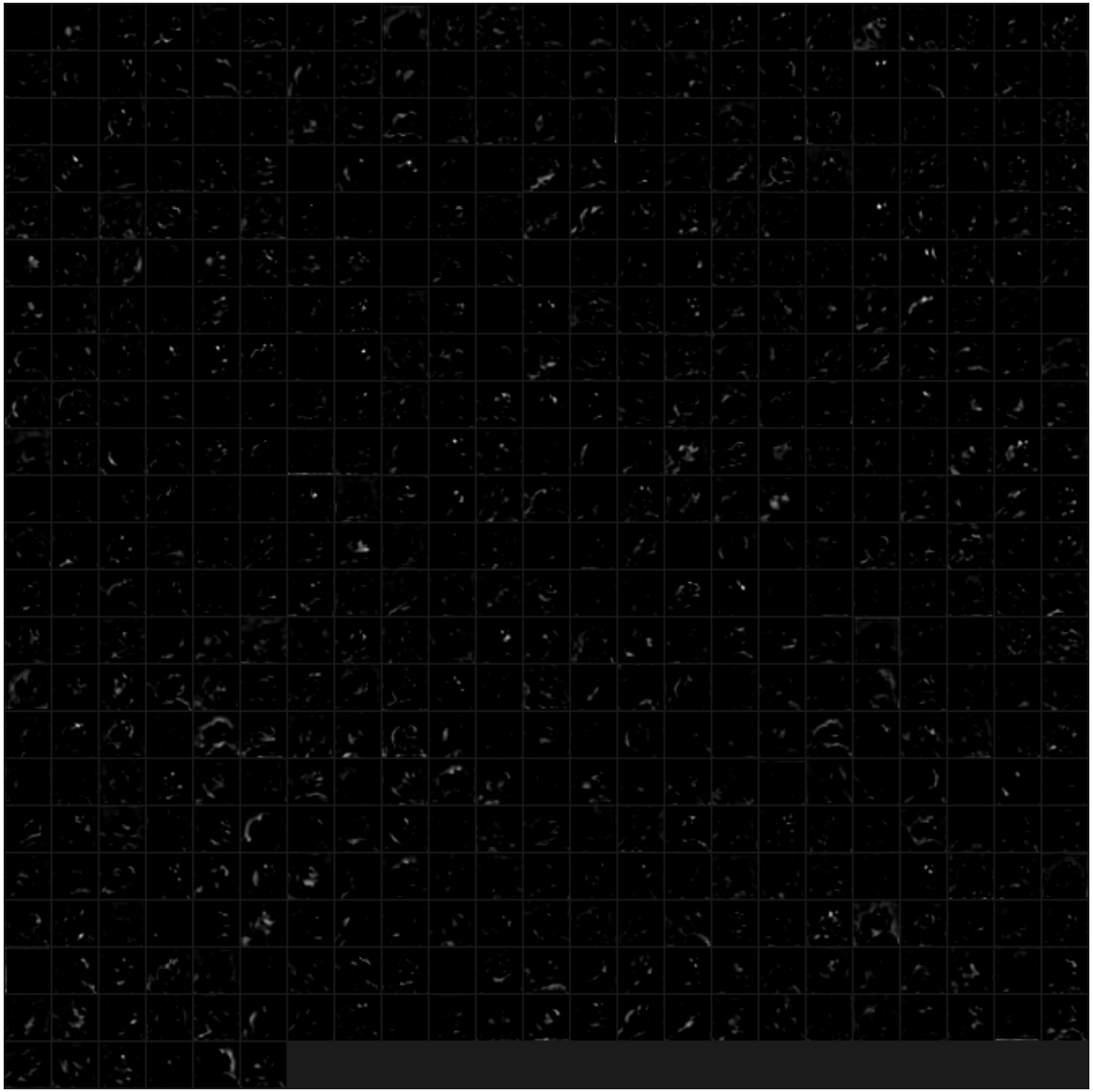
VGG16 : VGGscoiattolo_block4_conv1



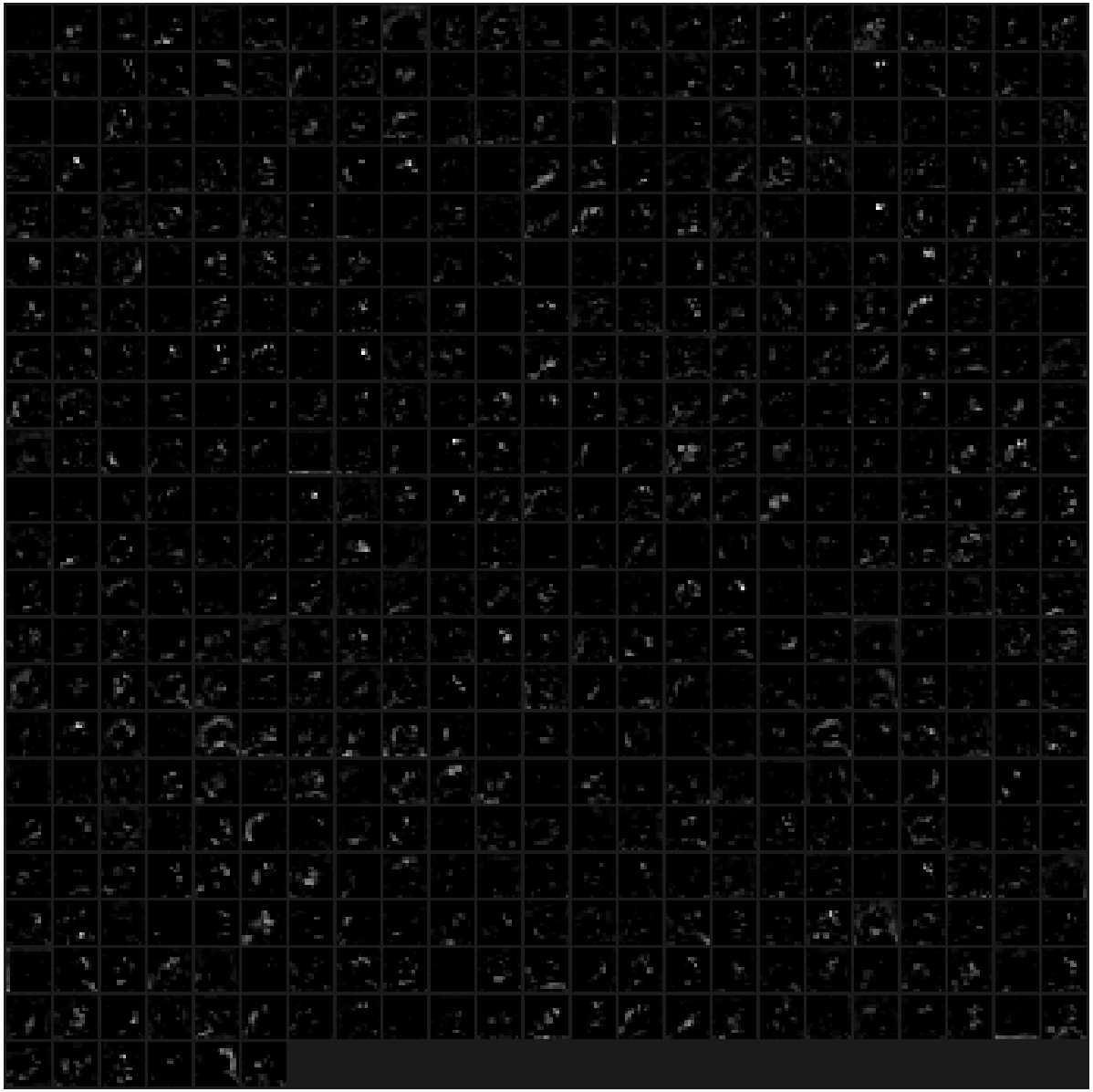
VGG16 : VGGscoiattolo_block4_conv2



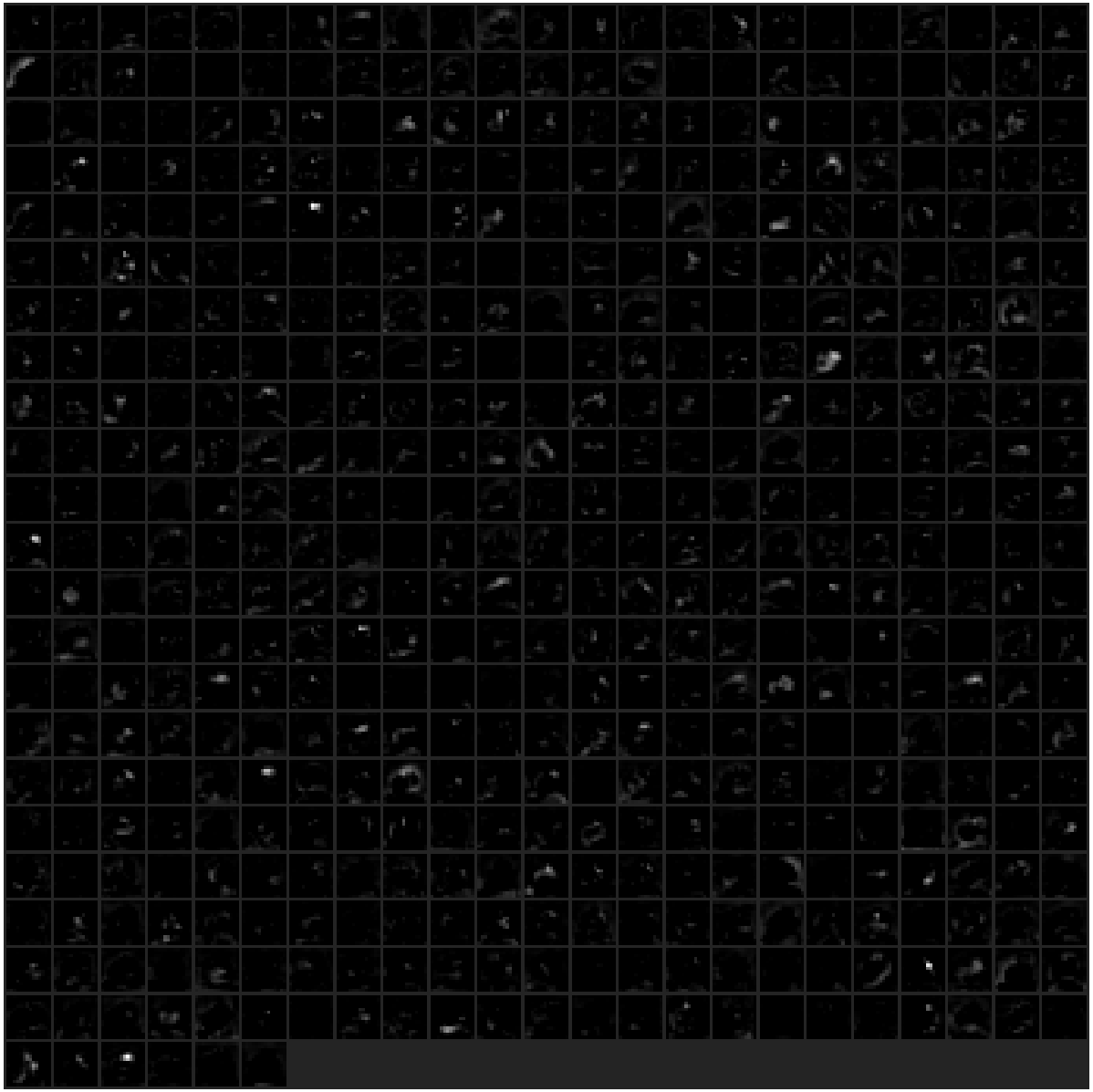
VGG16 : VGGscoiattolo_block4_conv3



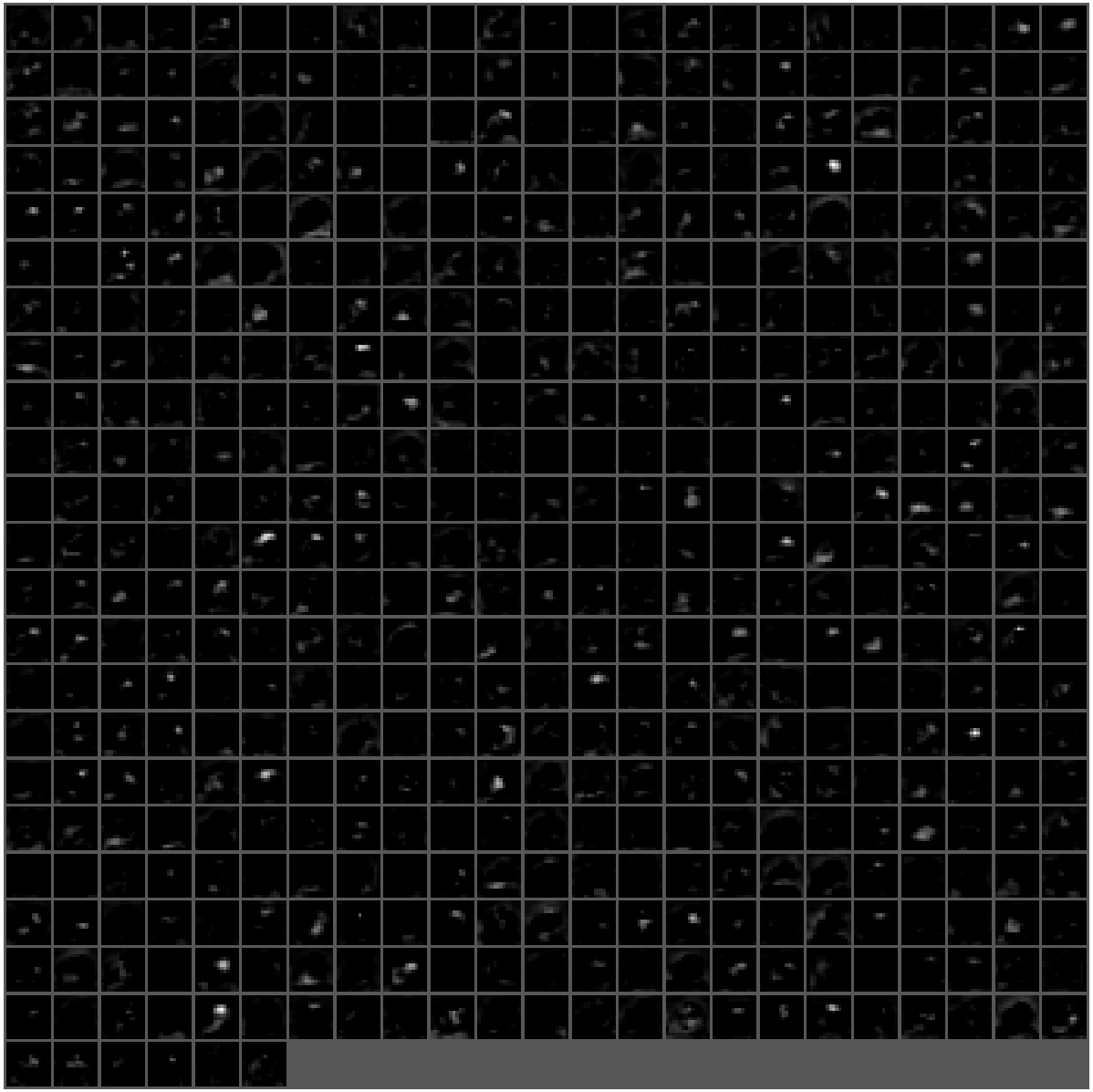
VGG16 : VGGscoiattolo_block4_pool



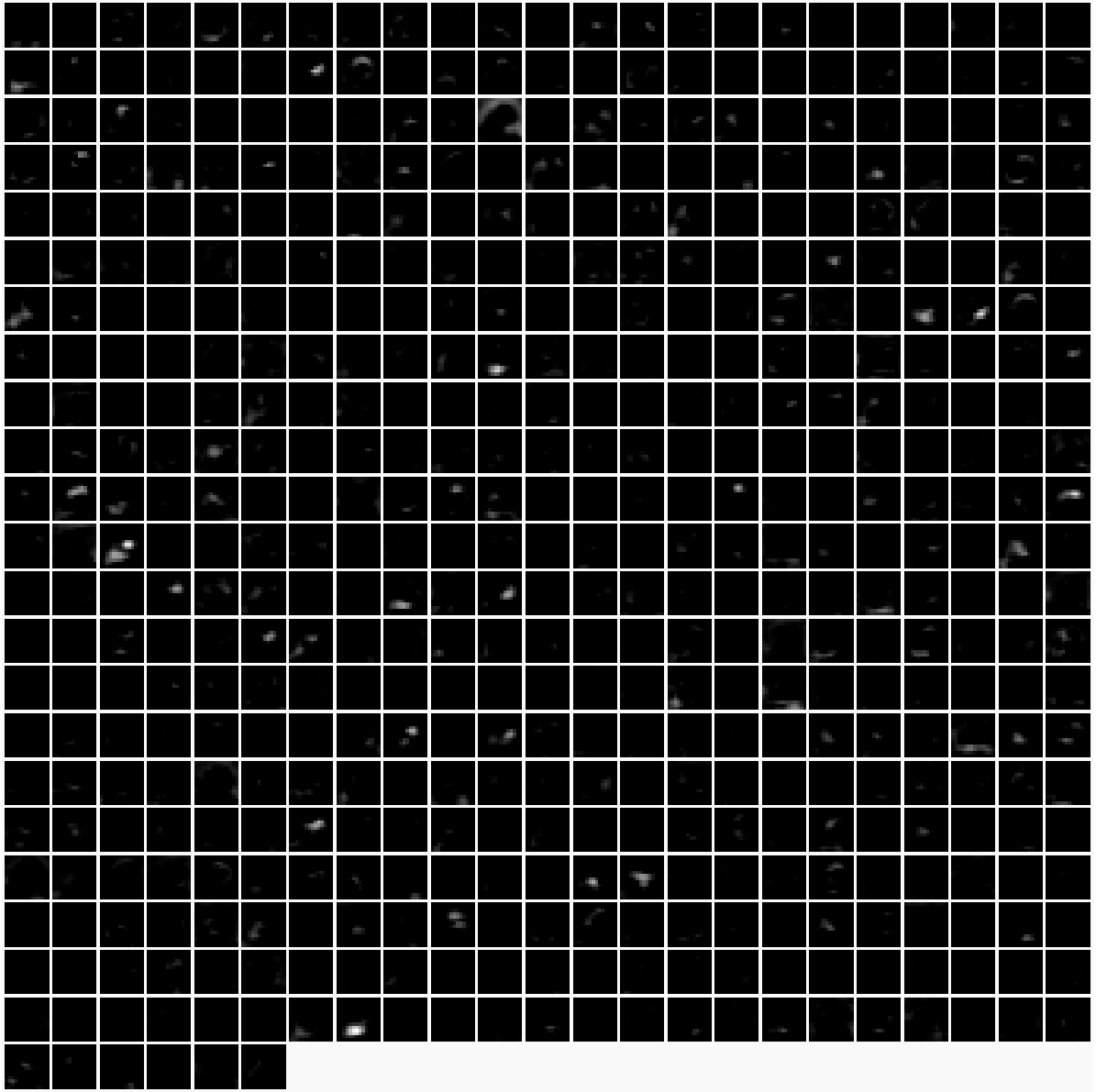
VGG16 : VGGscoiattolo_block5_conv1



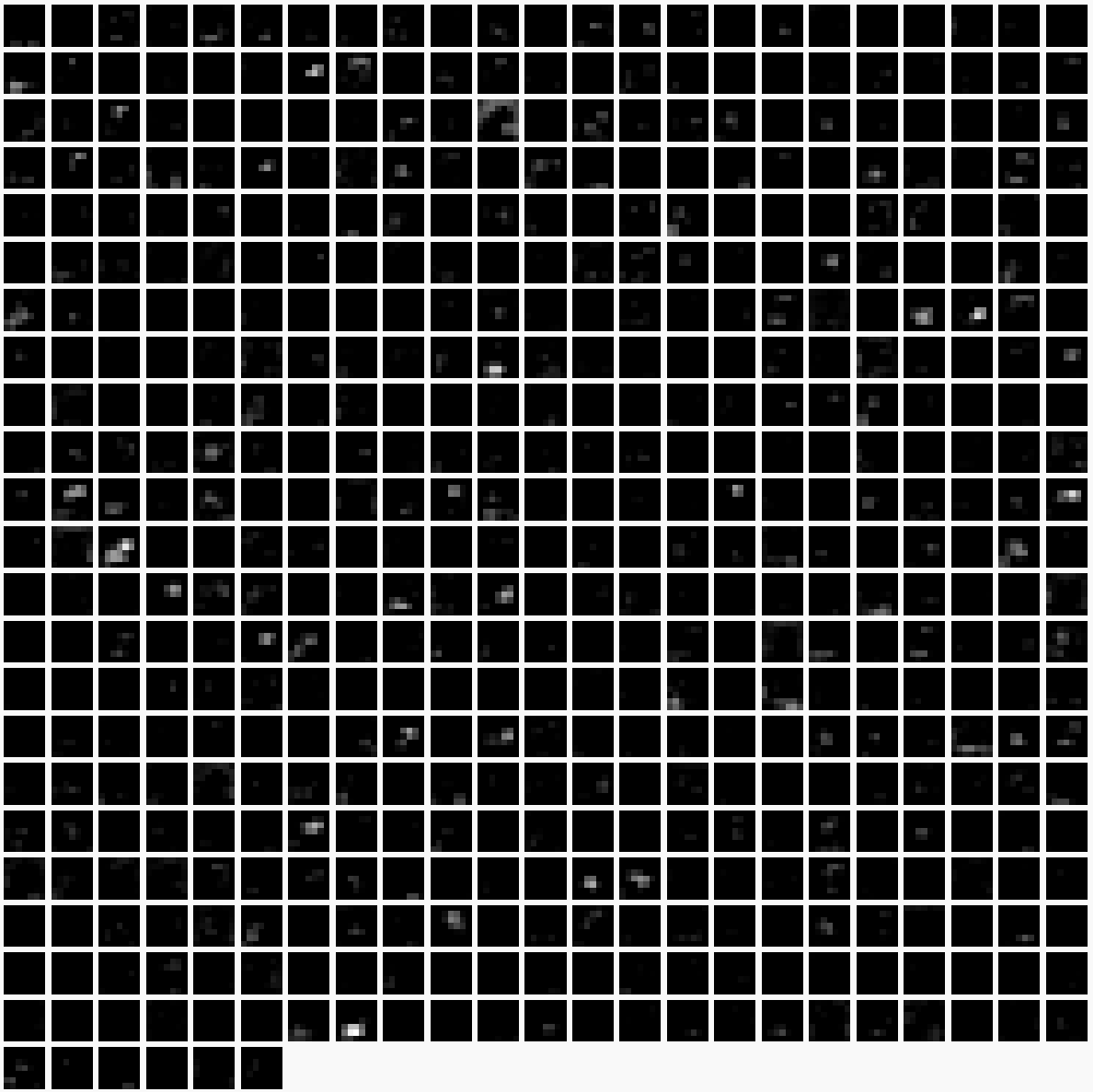
VGG16 : VGGscoiattolo_block5_conv2



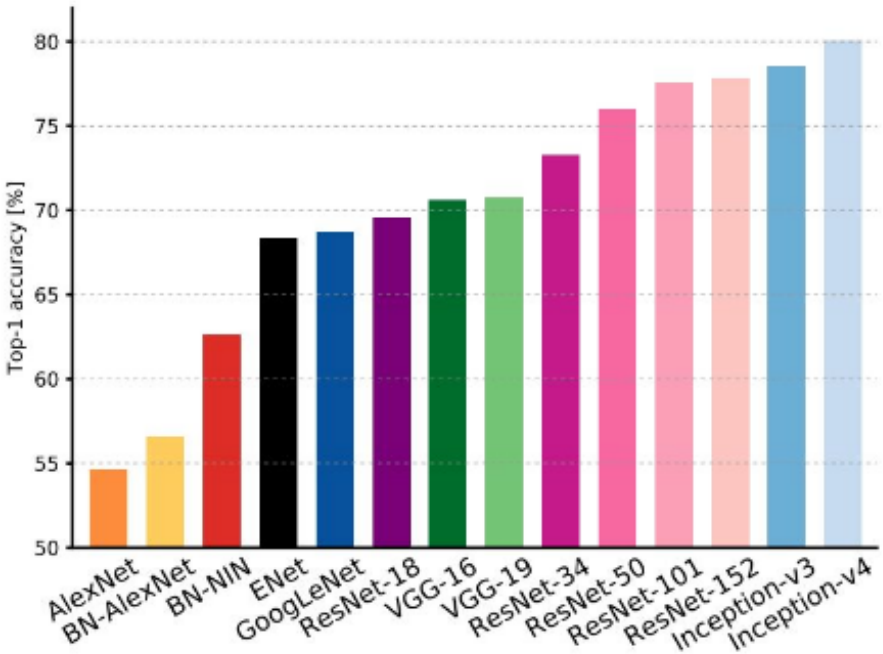
VGG16 : VGGscoiattolo_block5_conv3

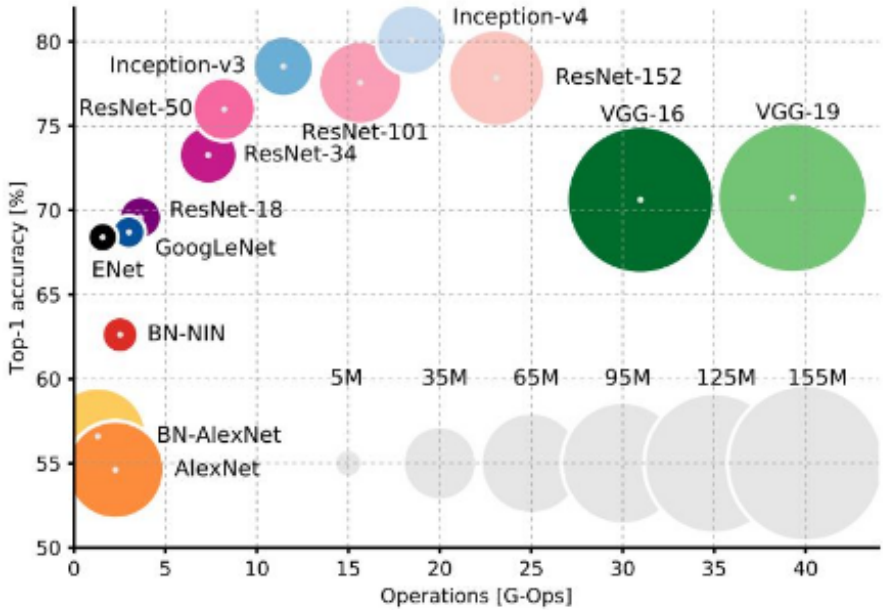


VGG16 : VGGscoiattolo_block5_pool



Different architectures





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

Applications

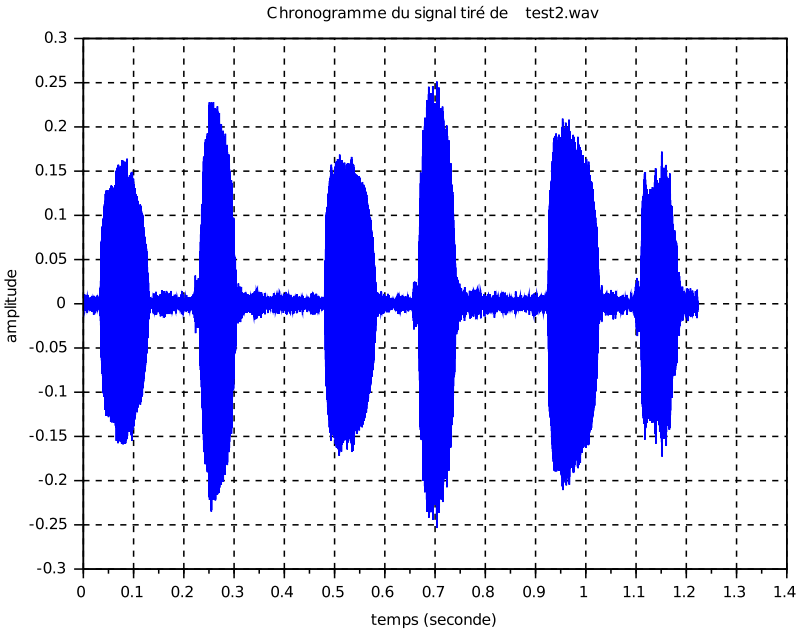
- image classification
- image descriptor
  - output of convolution layer as a vector describing an image
- transfert learning
  - learning only the last fully connected layers for a new image classification task
- sound recognition
  - spectrogram of sound as an image
  - transfert learning for sound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

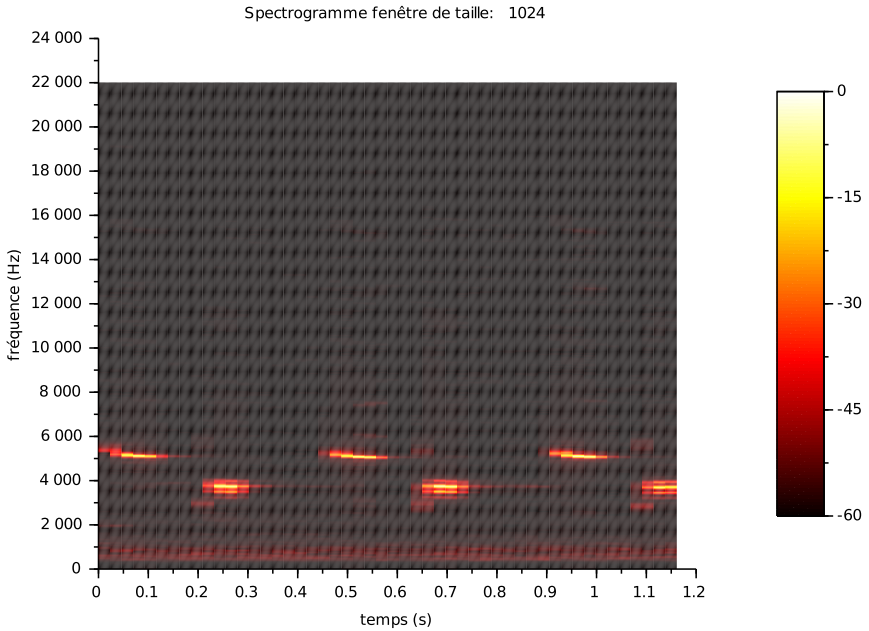
Image descriptor using keras

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

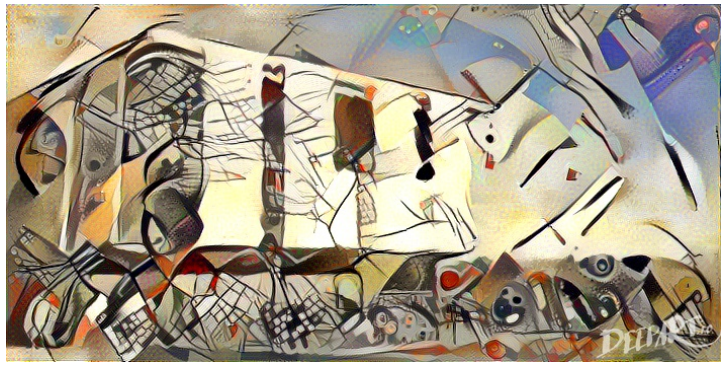




Outline

Art style transfert

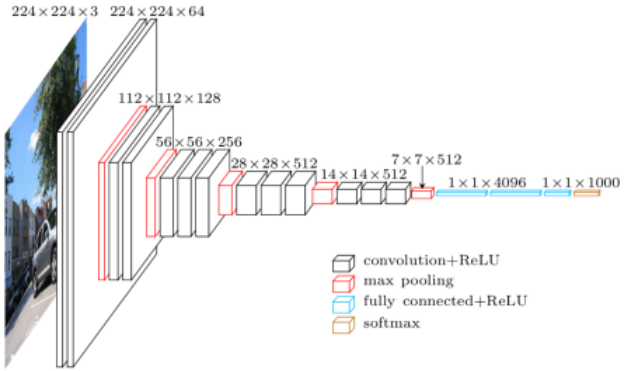





- 2015 *A Neural Algorithm of Artistic Style* by Gatys *etal*
  - `https://arxiv.org/abs/1508.06576`
- `http://www.subsubroutine.com/sub-subroutine/2016/11/12/painting-like-van-gogh-with-convolutional-neural-networks`
- try it at `deepart.io`

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 there 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 (highlighthing 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**.

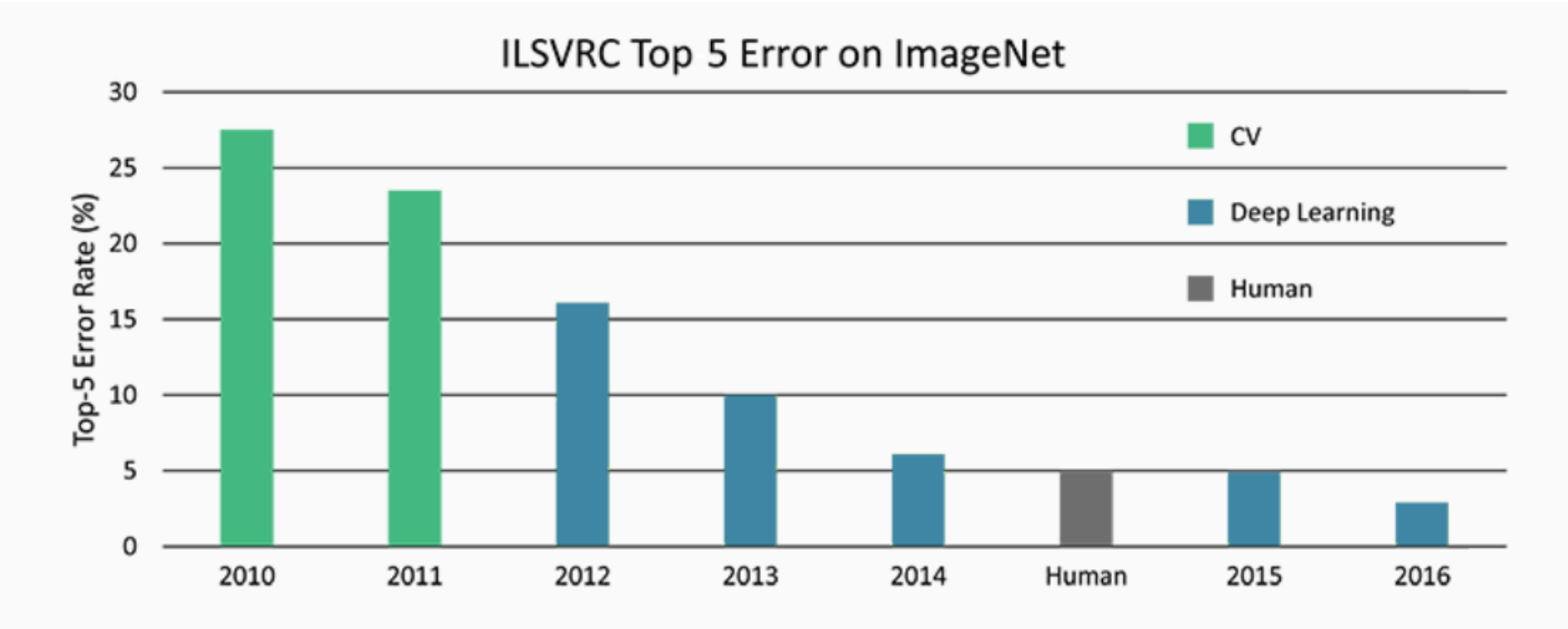
Deep dream

- amplify the neuron activations at some layer in the network
  - choose an image and pass it though 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`



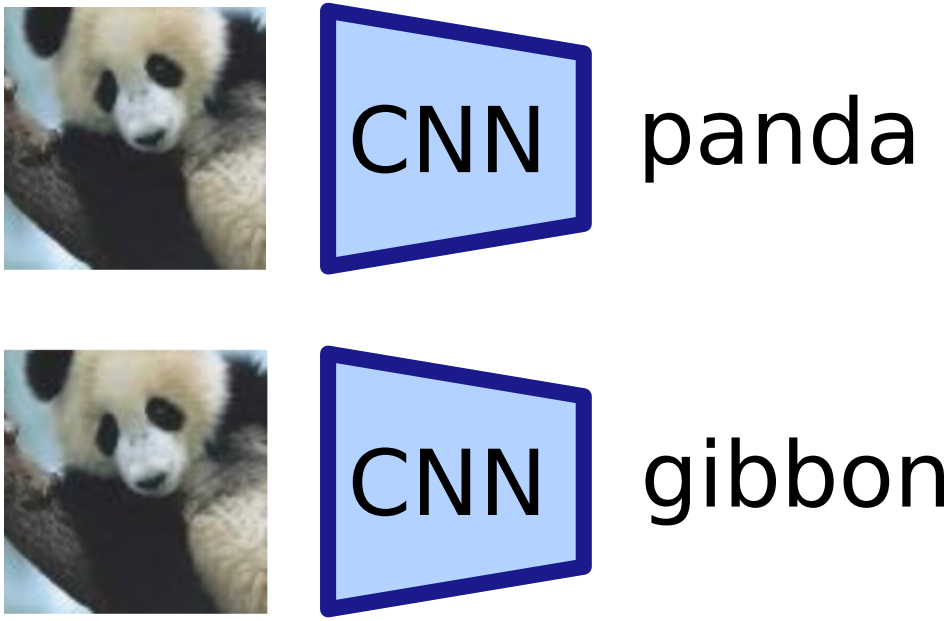
Outline

Deep networks are good at image classification



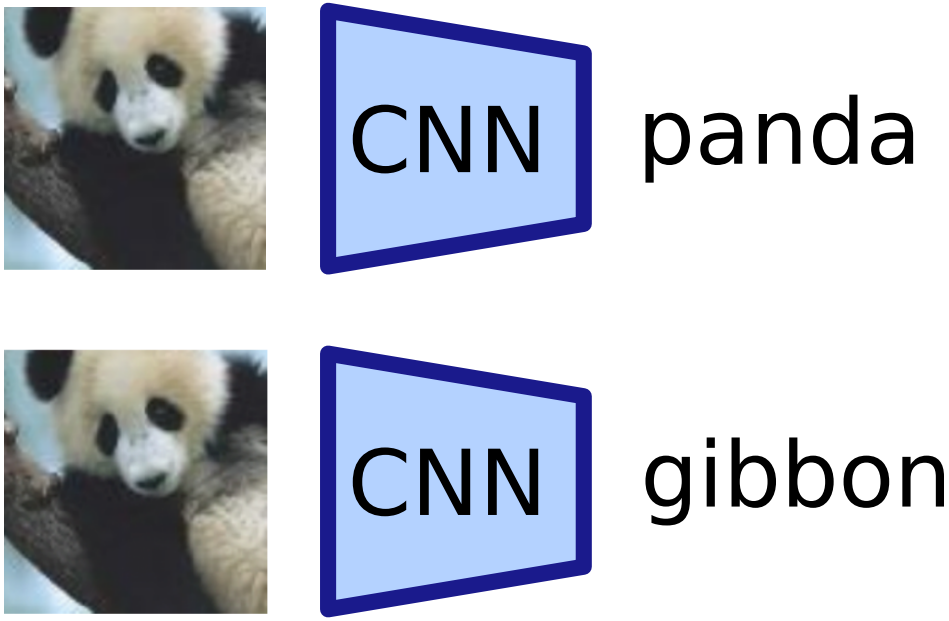
But ... how much does a deep network understands those tasks ?

What is an adverserial image ?

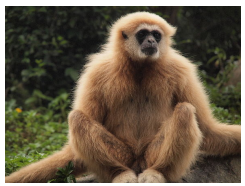


With high scores : gibbon with a confidence of 99.3%

What is an adverserial image ?



With high scores : gibbon with a confidence of 99.3%

For those how 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 distorsion $\|I - \hat{I}\| < \epsilon$, ($\epsilon > 0$, few pixels)
- given network's probabilities $f_\theta(I)$
- Different predictions :
  - $\arg\max f_\theta(I) \neq f_\theta(\hat{I})$

Adversarial examples

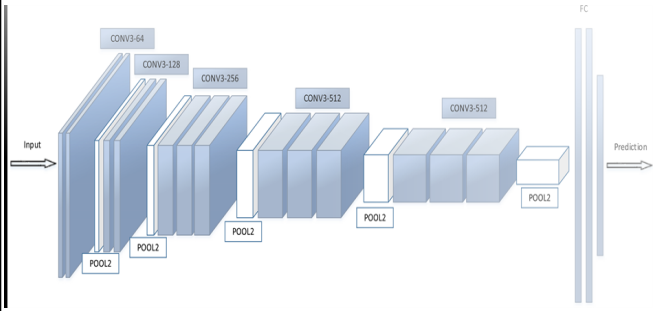
- images
- sounds
- synthesized 3D objects

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

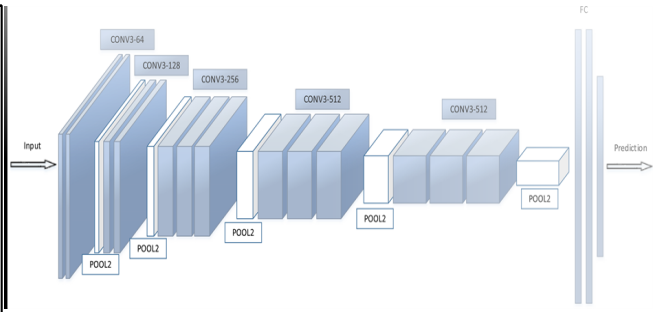
Input



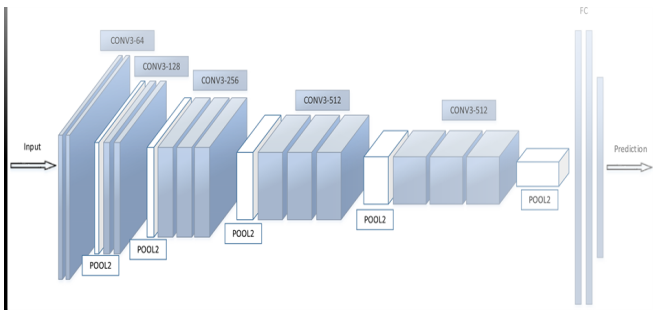
Network's prediction

" This is a car !"

backpropagation to modify the pixels



changing the prediction

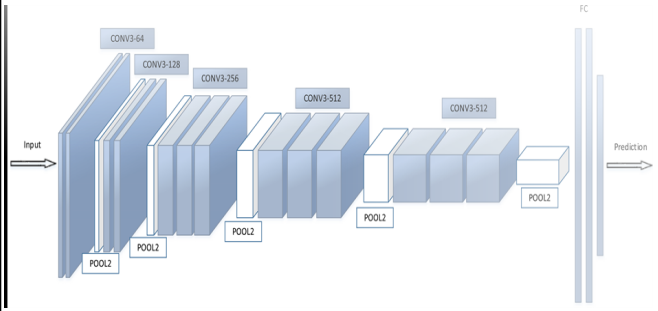


"This is a plane !"

from Ducoffe and Precioso

A simple experiment : what really happened

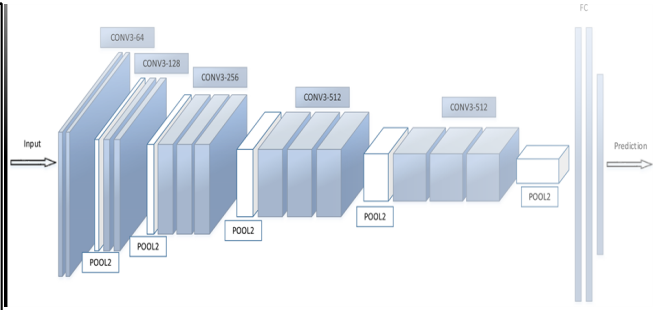
Input



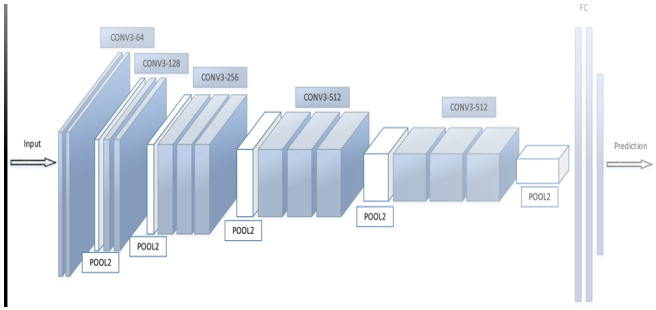
Network's prediction

" This is a car !"

backpropagation to modify the pixels



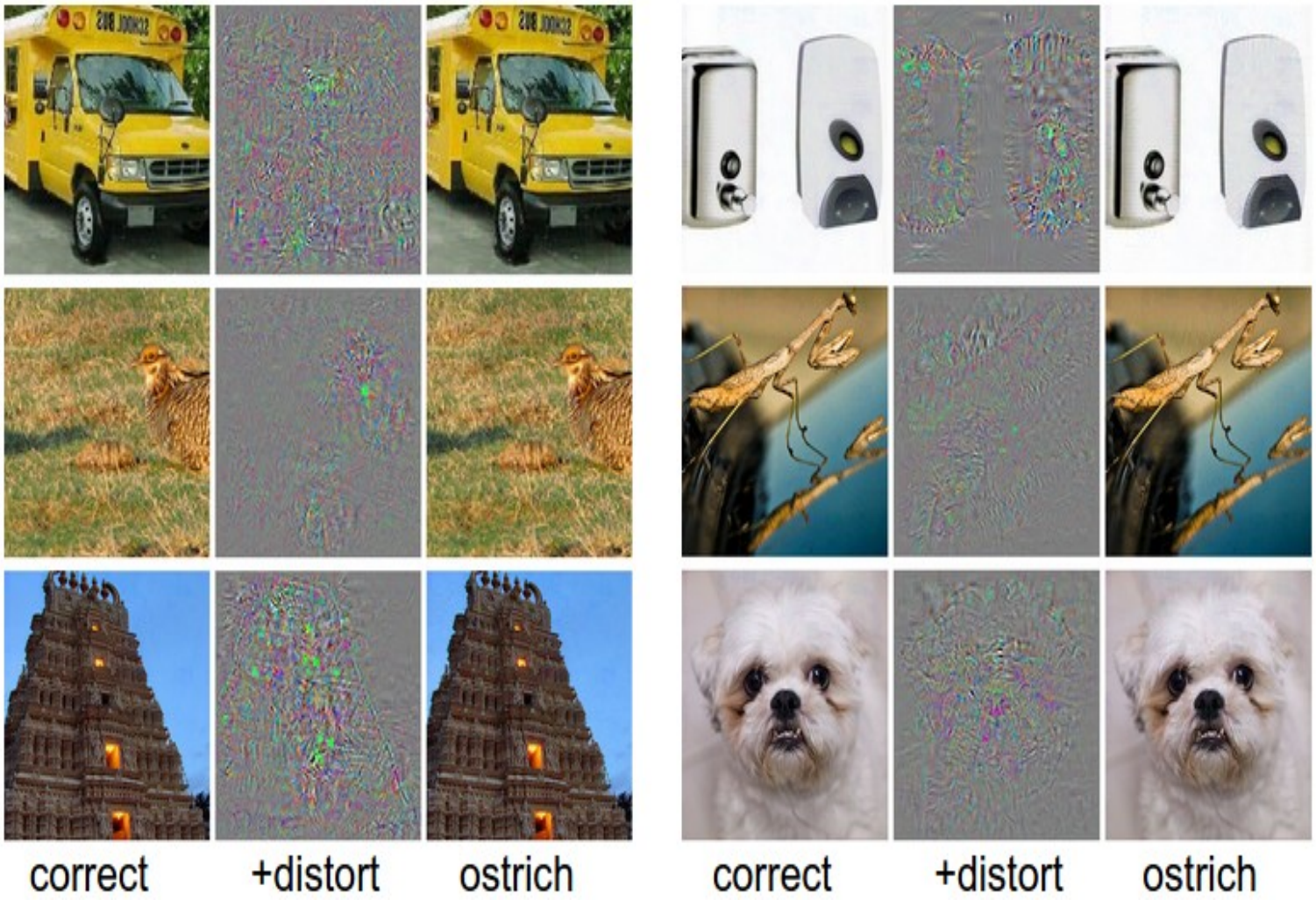
changing the prediction



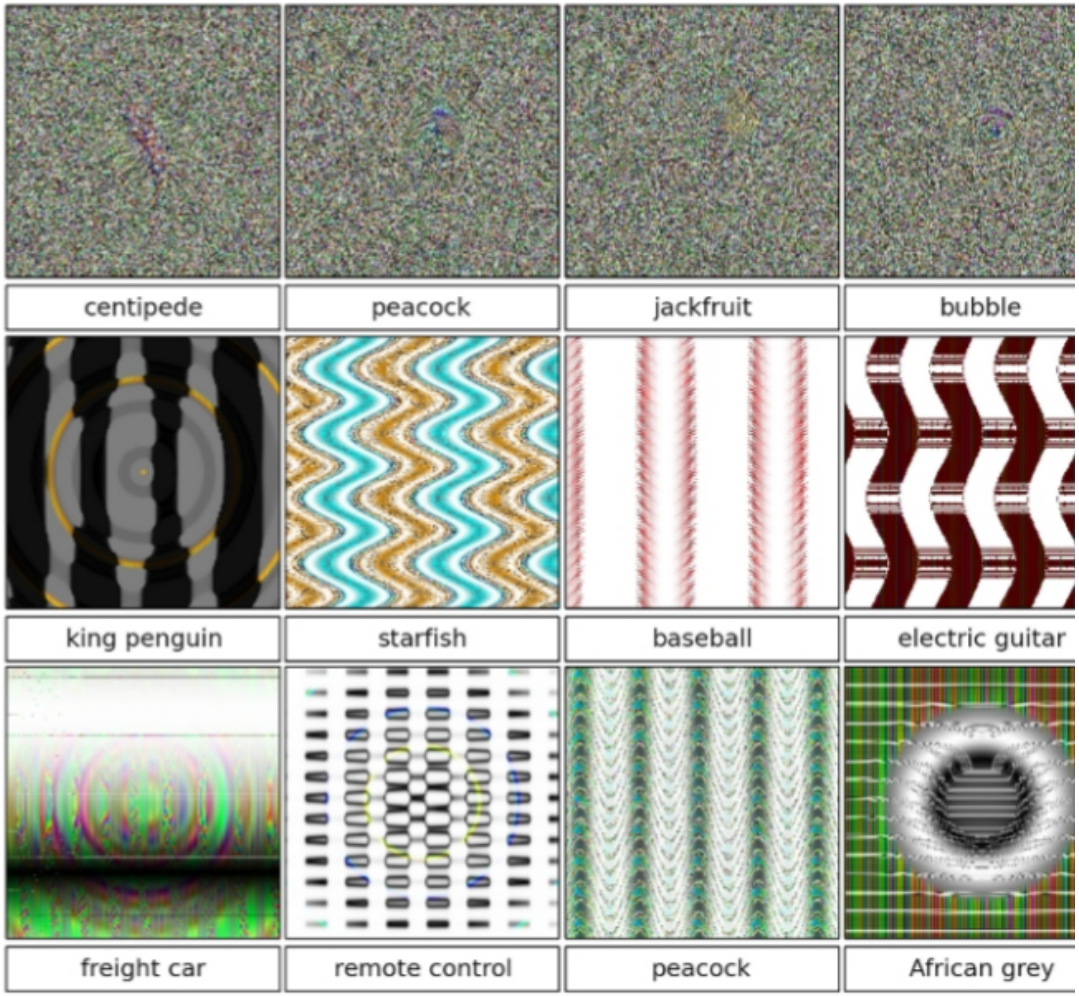
"This is a plane !"

from Ducoffe and Precioso

Seeing ostrich everywhere



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) :

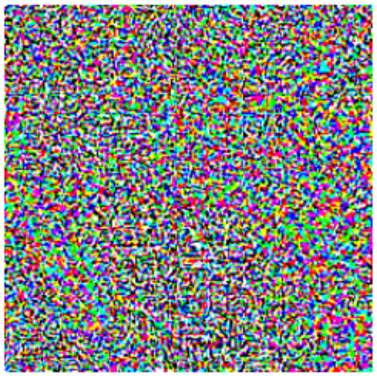
$$I_{adv} = I - \epsilon \text{sign}(\nabla_I J(I, y_{target}))$$

  - 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

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



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

"nematode"
8.2% confidence

$=$

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

"gibbon"
99.3 % confidence

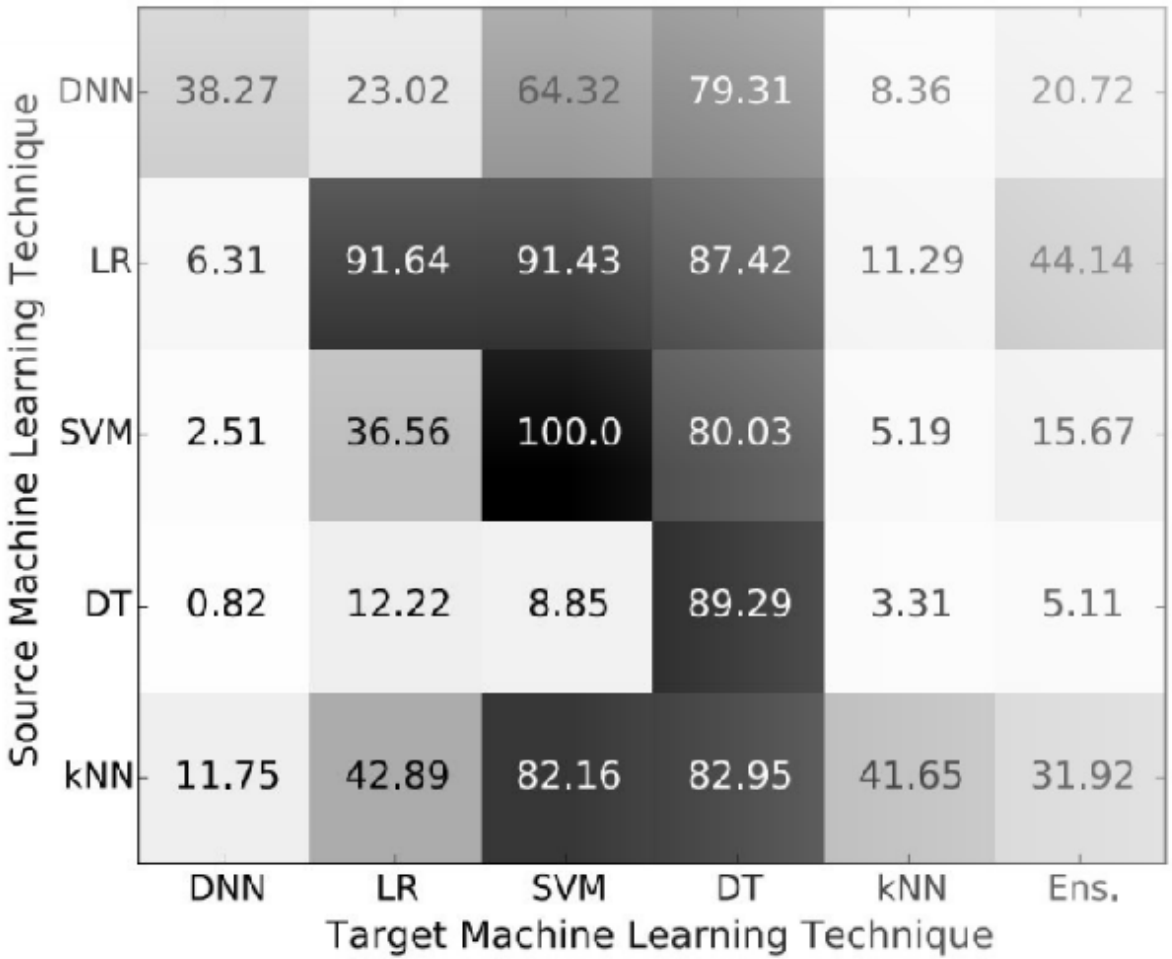
Is this particular to CNN ?

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

Linear classifiers

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

Cross-technique transferability



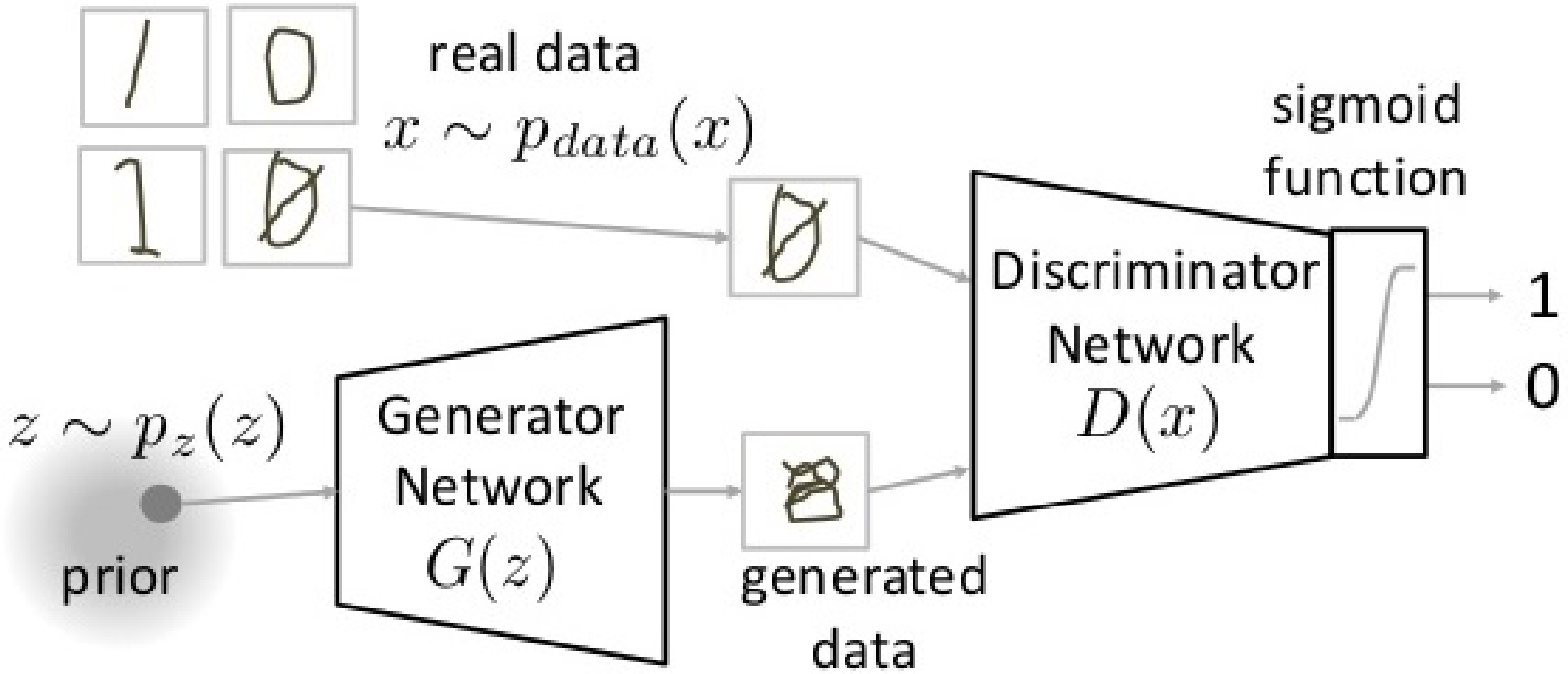
(Papernot 2016)

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

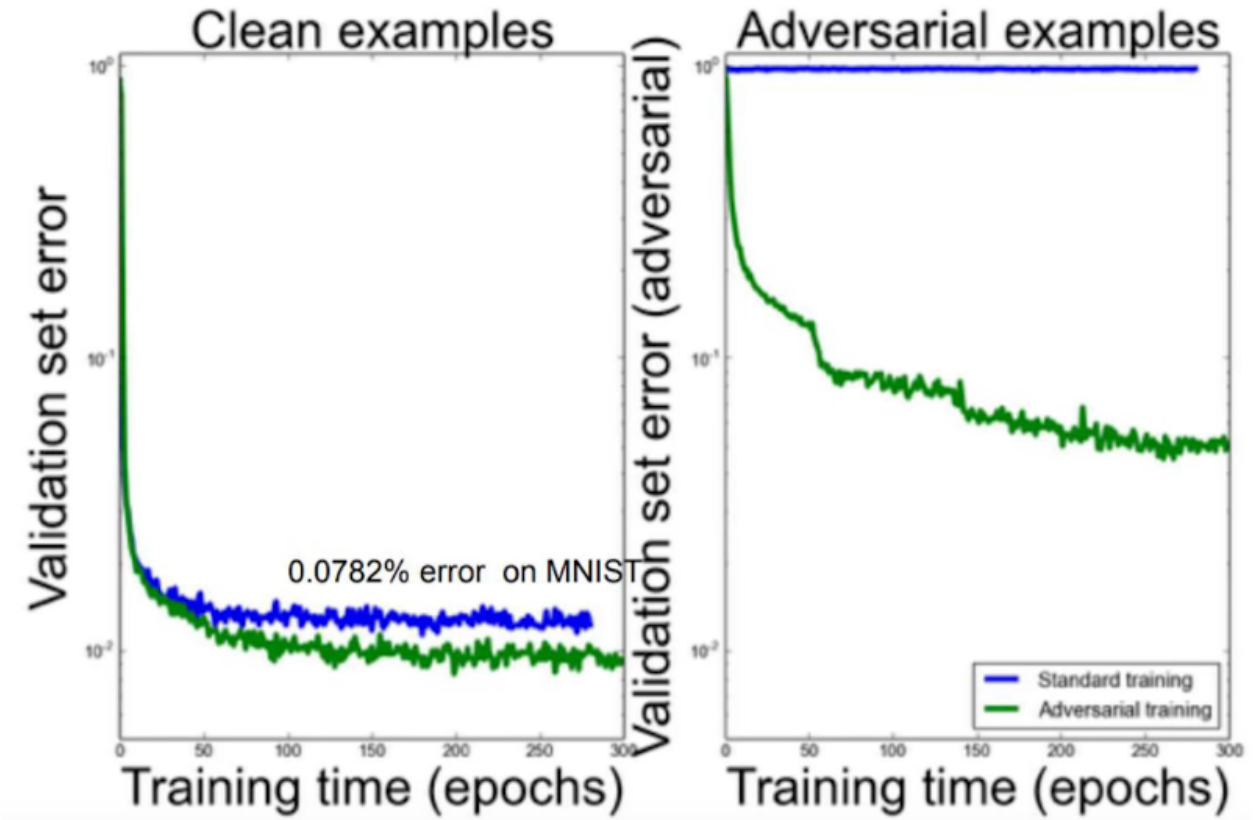
How to resist ?

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



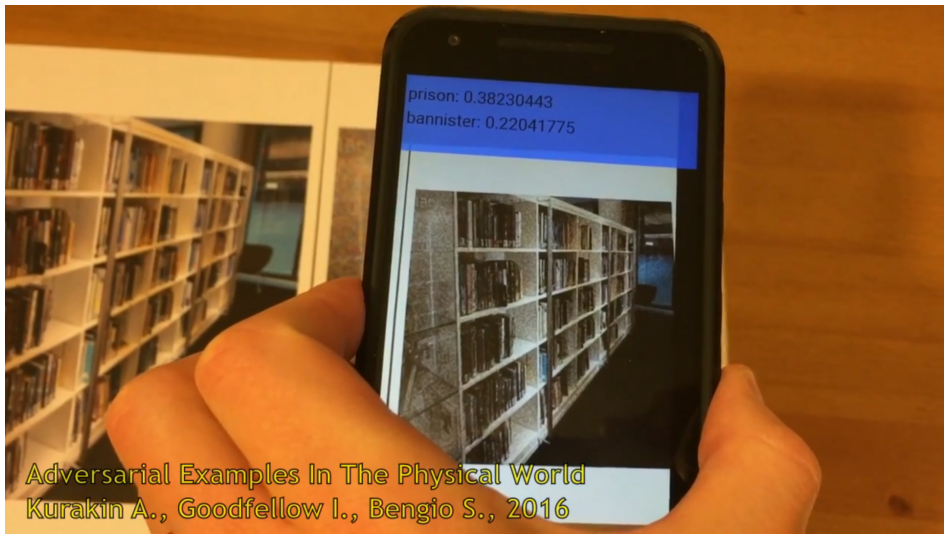
How to resist ?

- Incorporate adversarial images into the training database



OK, but that's all about synthesized images

Taking picture from adversarial images



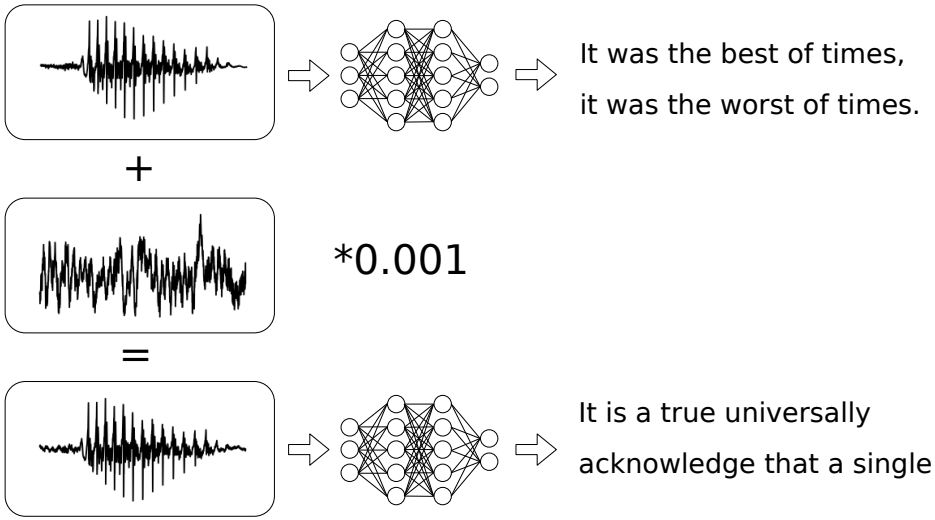
Adversarial examples in the physical world from Alexey Kurakin, Ian Goodfellow, Samy Bengio, 2016

`https://youtu.be/zQ_uMenoBCk`

Adversarial objects

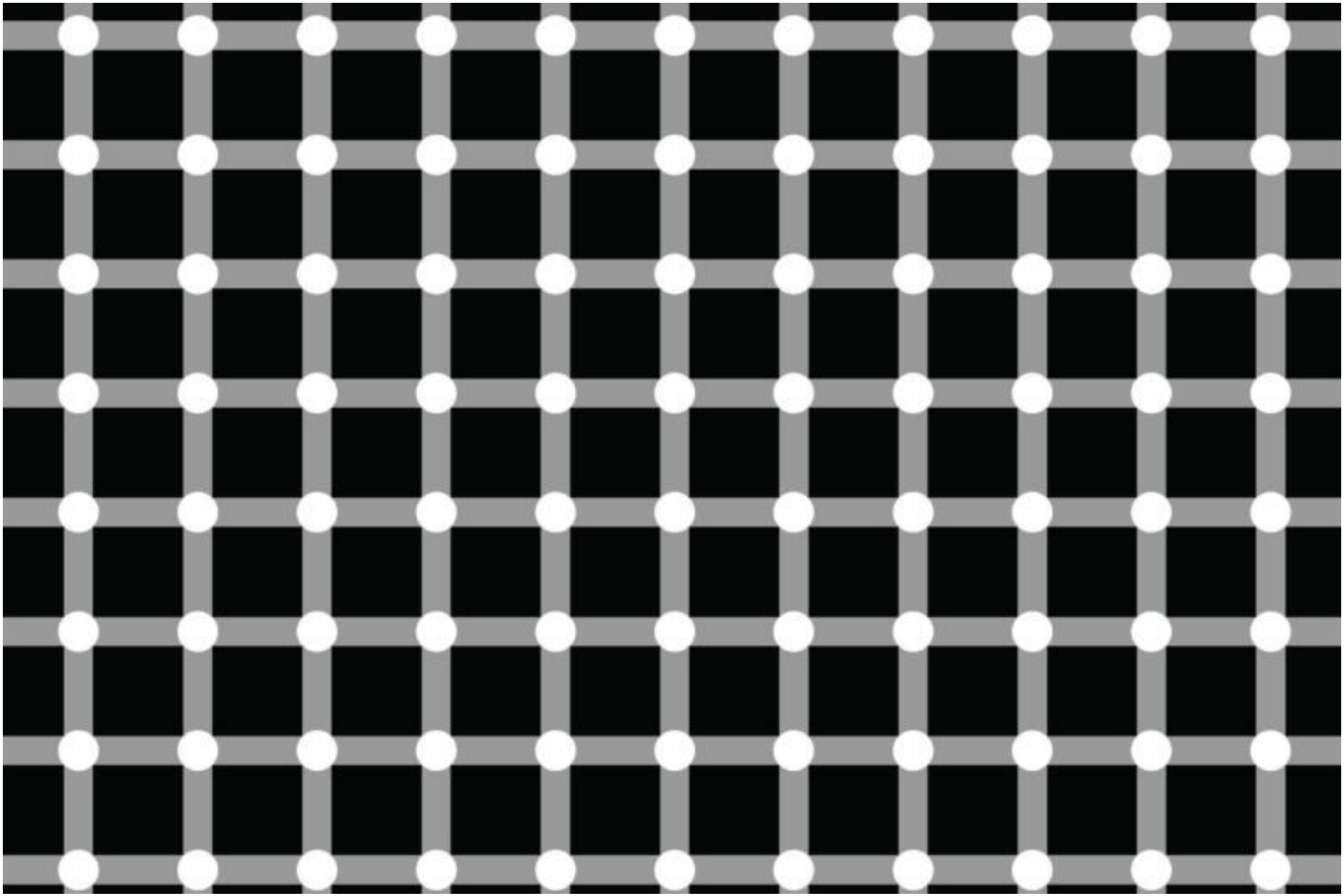
from `https://www.csail.mit.edu/news/fooling-neural-networks-w3d-printed-objects`

Adversarial sounds



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

We also, as human ...



Both human and computer