CNN: Convolutional Neural Networks

Diane Lingrand and many contributors

UNIVERSITÉ
CÔTE D'AZUR

SI4

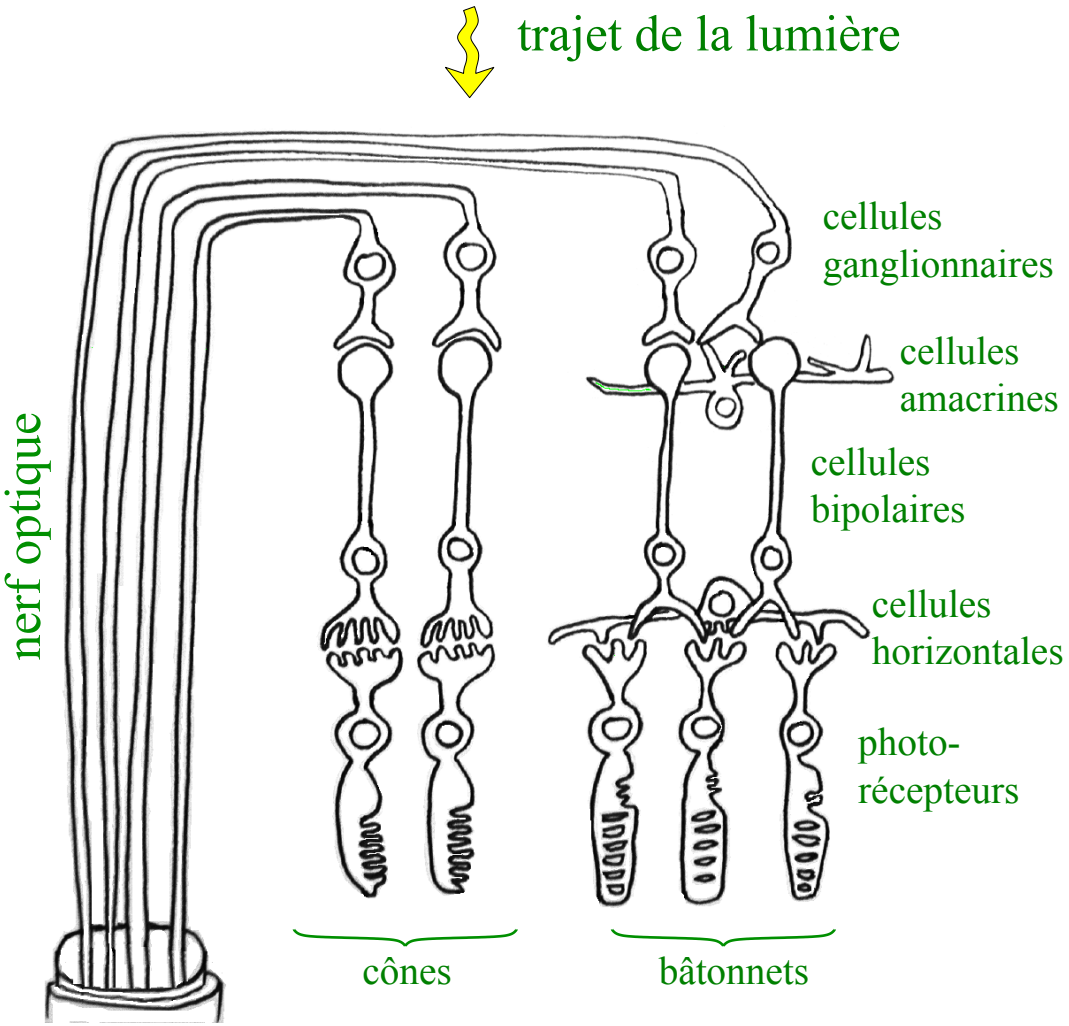
2024 - 2025

Outline

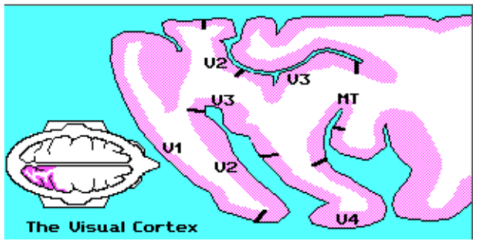
Outline

Deep Architecture in the Brain : the retina

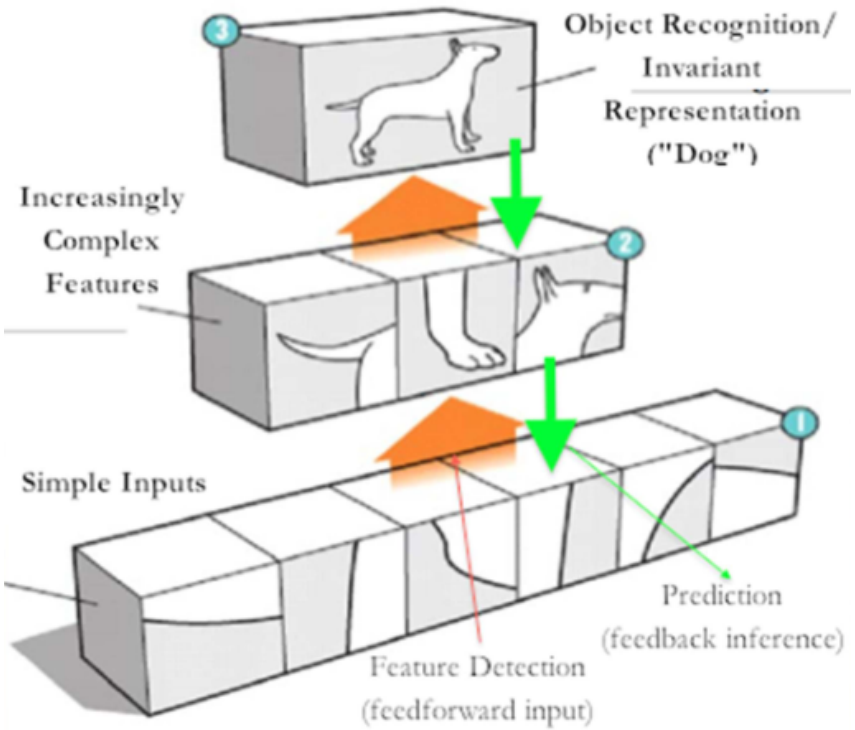
- photoreceptor cells : cones and rods
- bipolar cells (first neuron) : link to one or several photoreceptors
- ganglion cells (second neuron) link to one or several bipolar cells
- optical nerve (blind spot)



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

- pattern matching
  - use of correlation
  - if the kernel or filter is symmetric, convolution and correlation are equivalent

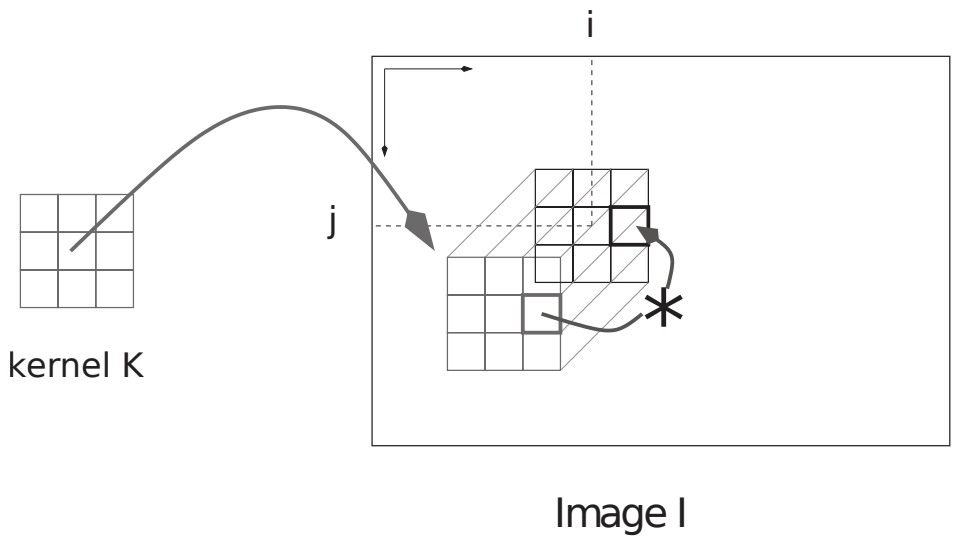
Back to convolution operator

- 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 $(2p + 1)x(2q + 1)$:

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



Back to cross-correlation

- For f and g discrete functions :

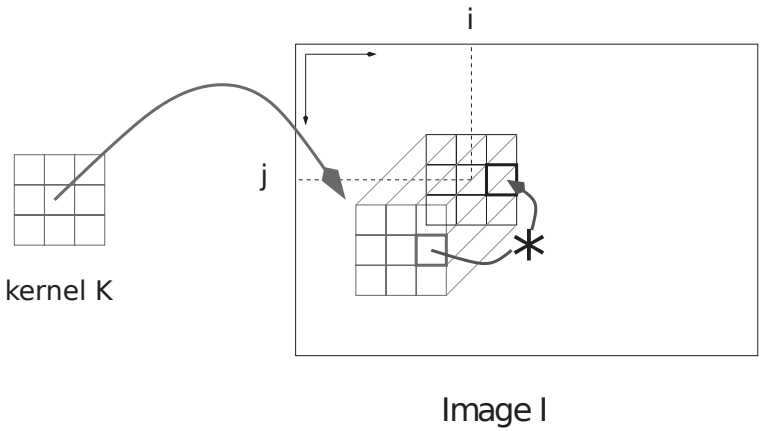
$$f \star g(x,y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(x,y)g(x-u,y-v)$$

- Applied to image I_1 by kernel k of shape $(2p+1)x(2q+1)$:

$$I_2[x,y] = I_1 \star k[x,y]$$

$$= \sum_{i=0}^{2p} \sum_{j=0}^{2q} I_1[x+i-p, y+j-q]k[i][j]$$

  - same kernel applied to different locations on the image

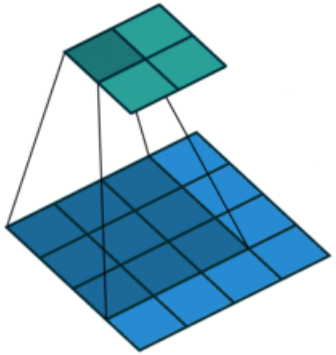


Idea of CNN : convolution layer

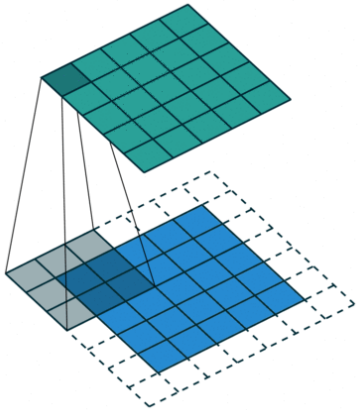
- learning the convolution filters
  - convolution : sum of weighted entries
  - 1 convolution filter = 1 neuron
  - the weights are the filter coefficients
- sharing weights with different neurons
  - the same convolution is applied to every pixels in the image
  - less weights to learn !
  - example : 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 $k * k$), stride
  - input : size of image and number of channels (3 if RGB)
  - output : tensor of dimension (new) dim of images and nb. filters,
  - `Conv1D` and `Conv3D` also exists

Padding and stride parameters in `keras.layers.Conv2D`

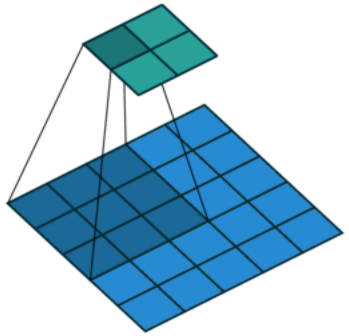
- padding
  - `padding = 'valid'` (no padding)



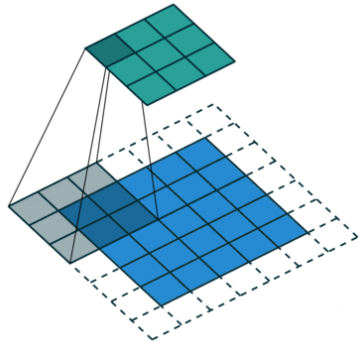
  - `padding = 'same'`



- stride
  - no stride is : `strides=(1,1)`
  - stride of 2 with no padding



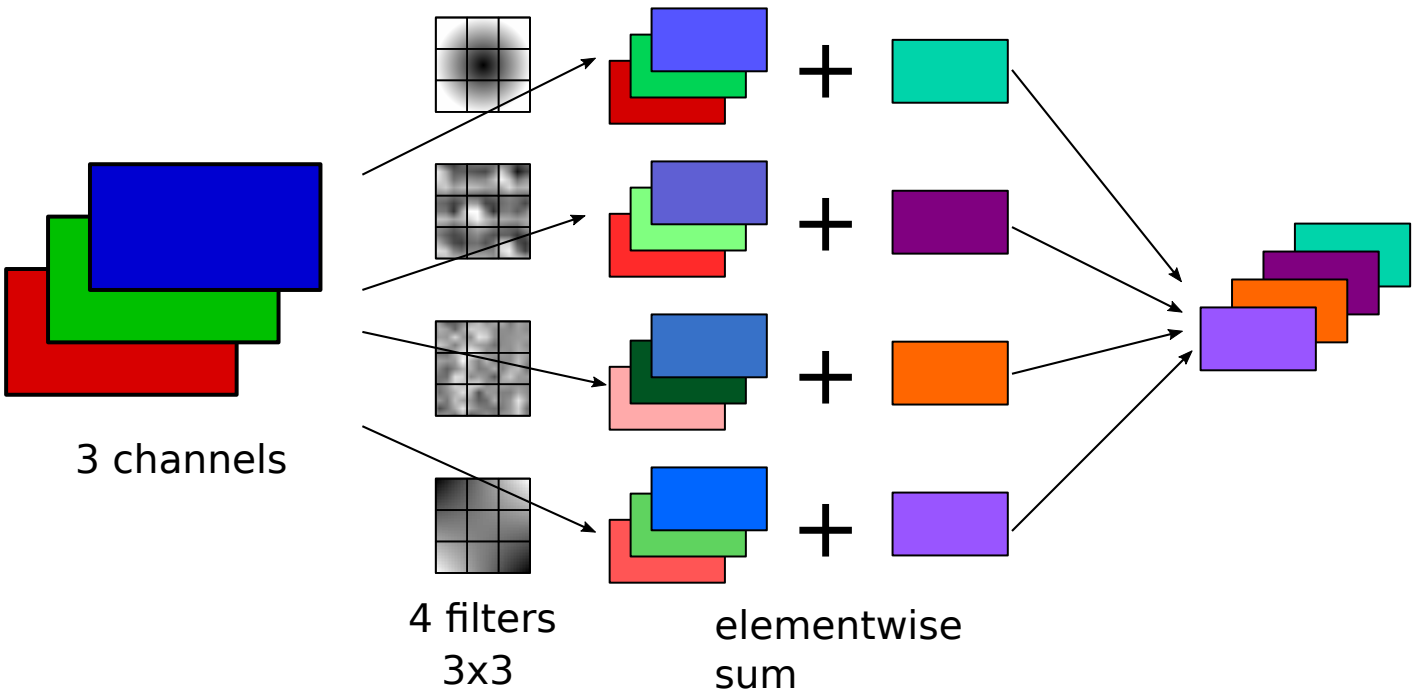
  - stride of 2 : `strides=(2,2)` with padding 'same'



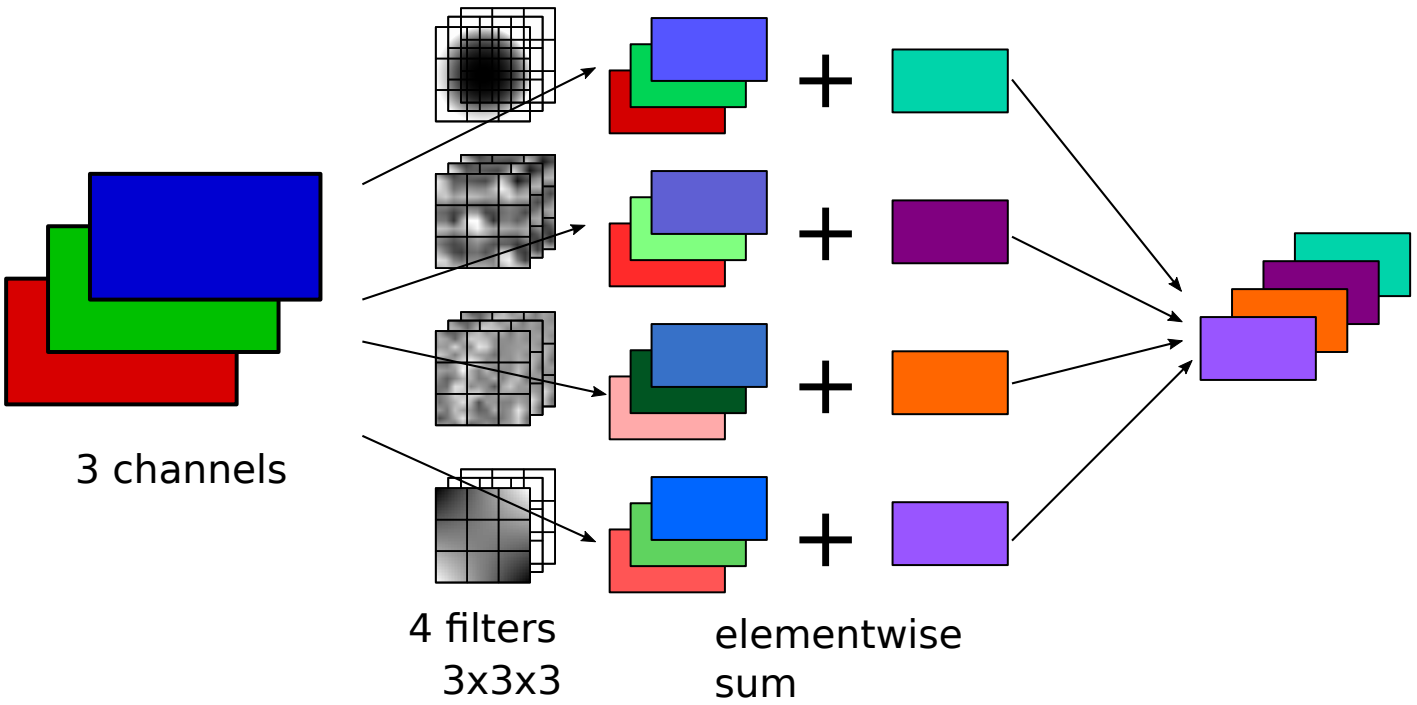
animations from `https://github.com/vdumoulin/conv_arithmetic/tree/master`

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 : 3x3x(number of input channels) +1
  - 4 filters
  - thus 4x(3x3x3 + 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 characteritics of images
  - but not only convolutional layers :
    - pooling
    - flatten
- two hidden fully connected layers at the output
  - the function of classification

Pooling layers

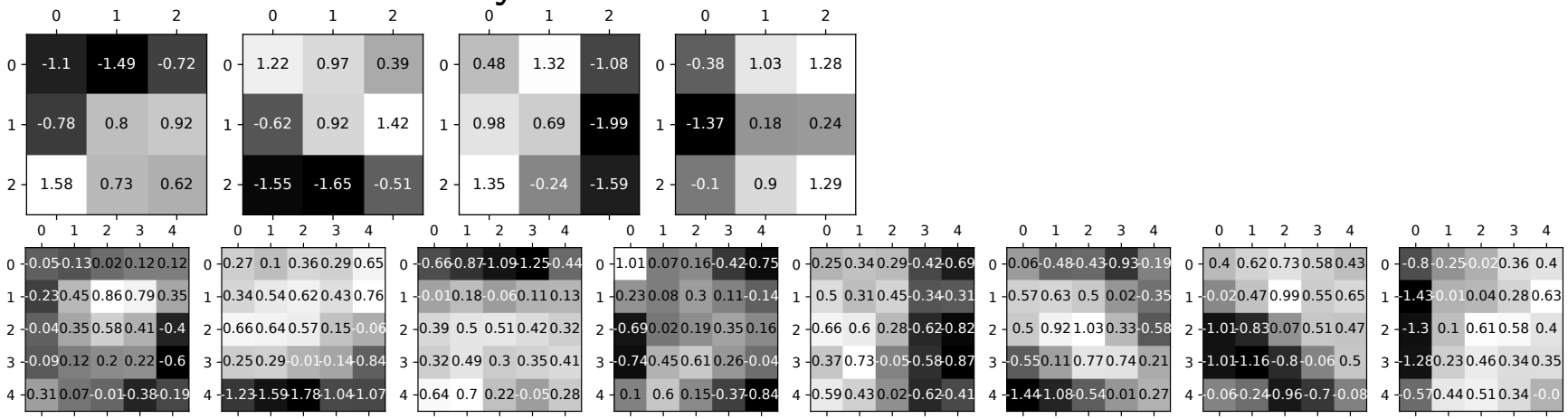
- usually after a convolutional layer
- reduces the size of data (downsampling)
- type of pooling
  - max pooling `from keras.layers import MaxPooling2D`
  - average pooling `from keras.layers import AveragePooling2D`
- dimension of pooling
  - default : 2X2 with stride 2
  - dimension
  - stride : factor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

Link with what you did in SI3 (1)

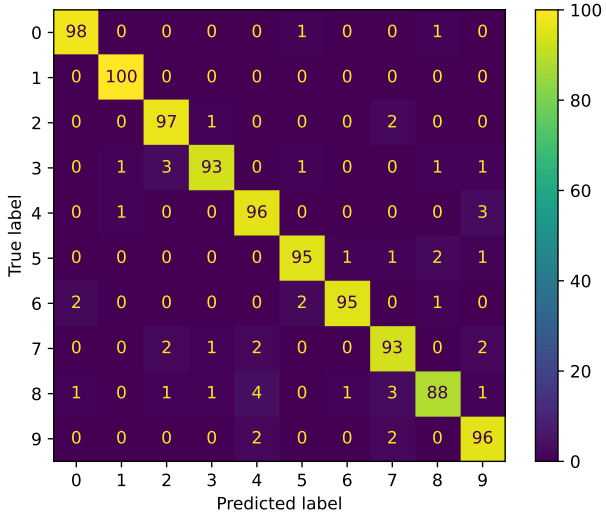
- main idea :
  - the results of correlation/convolution of some filters will help to represent an image in a more compact and mean full than the $28 * 28 = 784$ pixel values
- which filters ?
  - can be handcrafted : $\cap, \cup, \bigcirc, \subset, \supset, |, -, /$
  - I can give you some other filters
- It is not necessary to compute corr./conv. at every position :
  - skip some rows and cols
  - for example : 1 over 3 cols and 1 over 3 rows
    - it will speed the computations
- normalisation
  - divide every MNIST digit by 255
  - use almost normalised filters
  - and thus skip the normalisation of correlation in order to speed
  - eventually, remove all negative values
- when each image is represented using some corr./conv. results
  - perform a classification :
    - using kNN or logistic regression

Link with what you did in SI3 (2)

- Here are some filters you could test :



- Result that you may obtain :
  - Confusion matrix computed on the test set (accuracy = 95.1%) :



Learning a CNN

- GPUs !
- "Baby sitting" your 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 ...

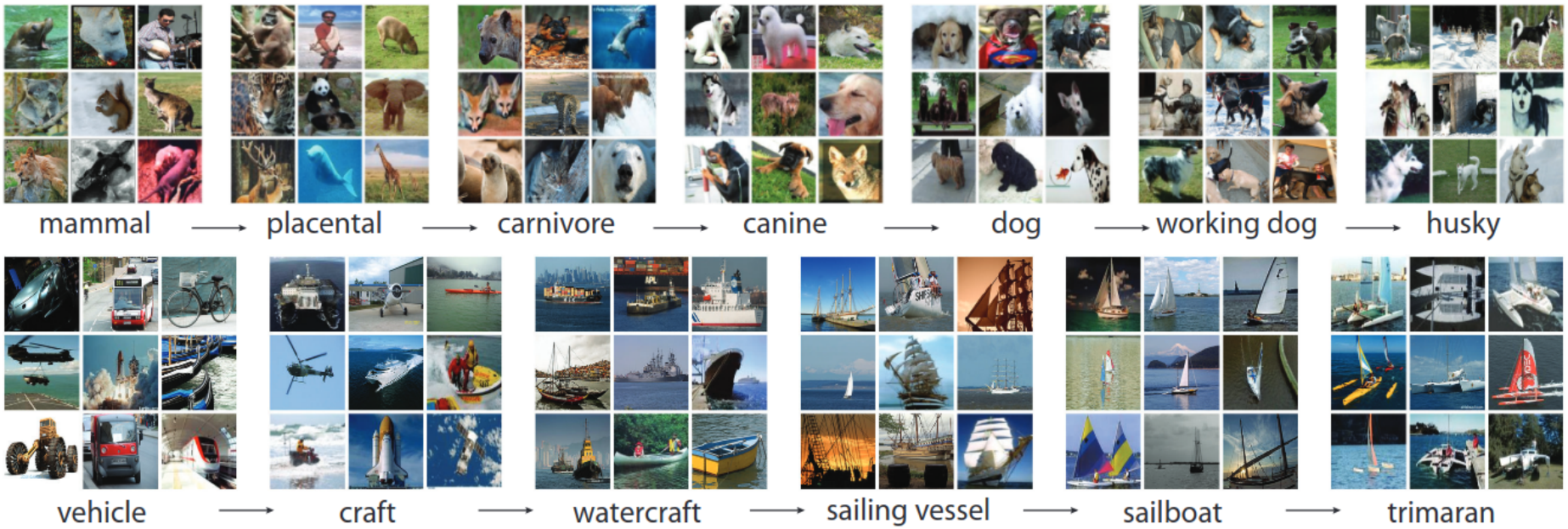
batch normalisation

- 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

Outline

ImageNet

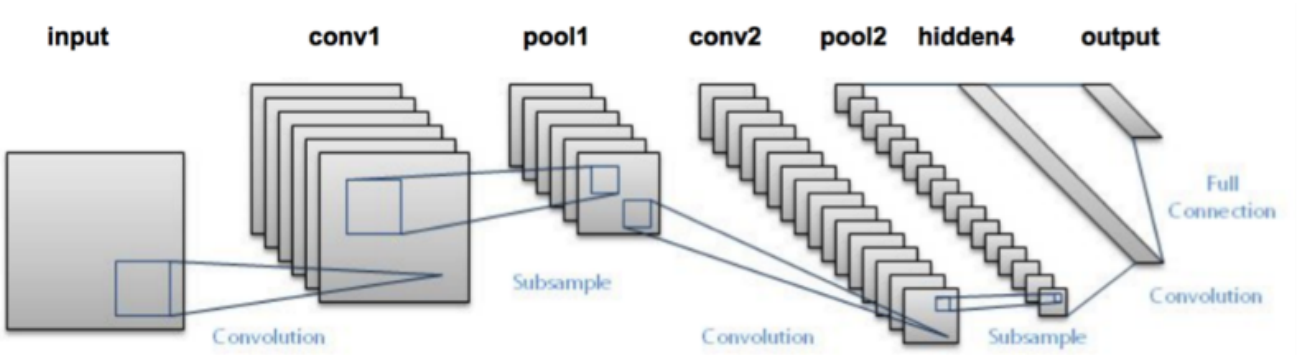
- more than 1000 classes
- 50 million labeled images
- in the ILSRVC : ImageNet Large Scale Visual Recognition Challenge



Architectures

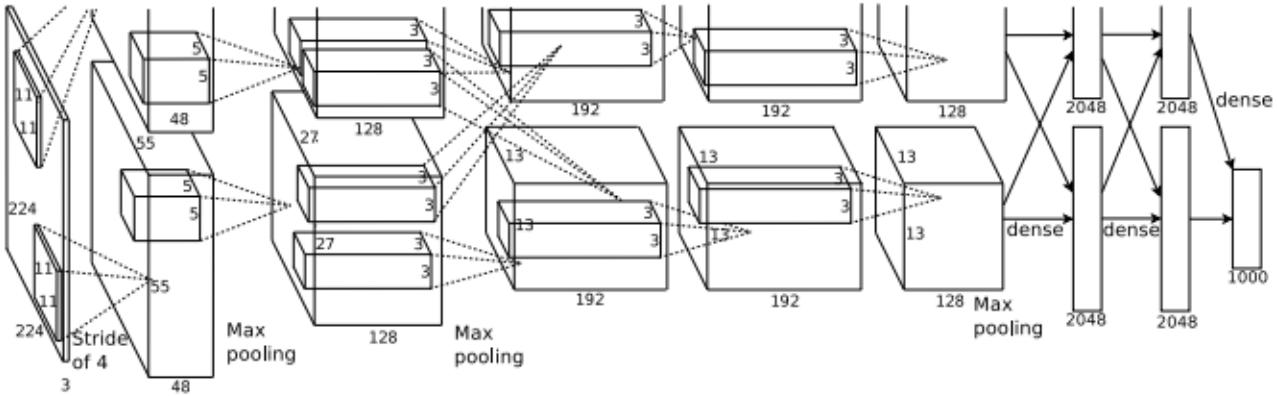
- LeNet 5 1998
  - historical, by Le Cun *et al*
- AlexNet 2012
  - first CNN winning ILSRCV competition in 2012
- VGGNet (2014)
  - 16 layers to be learned
  - VGG19 : 19 layers
- GoogLeNet Inception 2014
  - 22 layers
- ResNet 2015
  - introduce residual connections
- Xception (= eXtreme Inception)
  - 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).

AlexNet, 2012



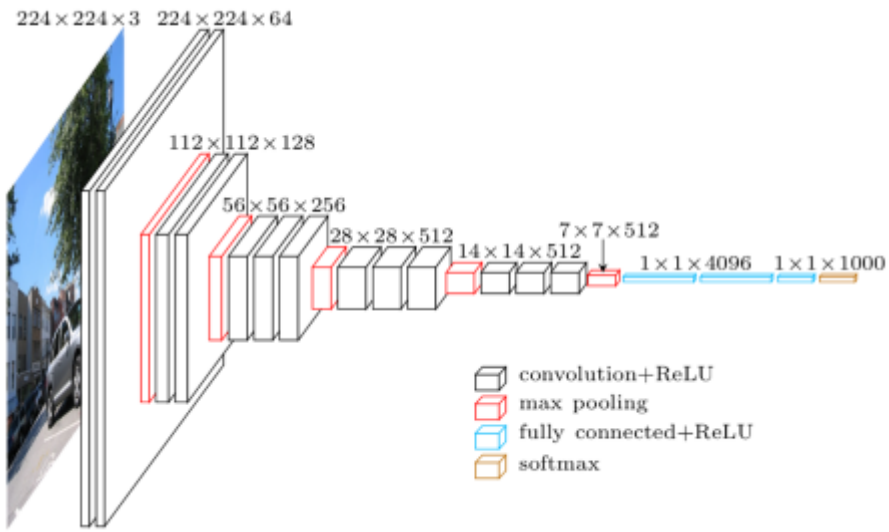
- 11x11, 5x5 and 3x3 convolutions, overlapping max pooling, local normalisation, dropout, data augmentation, ReLU activations, SGD with momentum. ReLU activations after every conv. and FC layer. 2 GPUs (memory) : grouped convolutions
- architecture :

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

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
```

VVG-16, VGG-19, 2014



- deeper
  - VGG-16 : 16 refers to the number of weighted layers (conv or dense)
  - VGG-19 : 19 layers to learn
- small convolution filters (less parameters)
  - same receptive field for 3 stacked 3x3 conv. layer and 1 7x7 conv. layer
  - deeper : more non-linearities and fewer params ($3x(3x3xch_{in}xch_{out})$ compared to $7x7xch_{in}xch_{out}$)
- 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

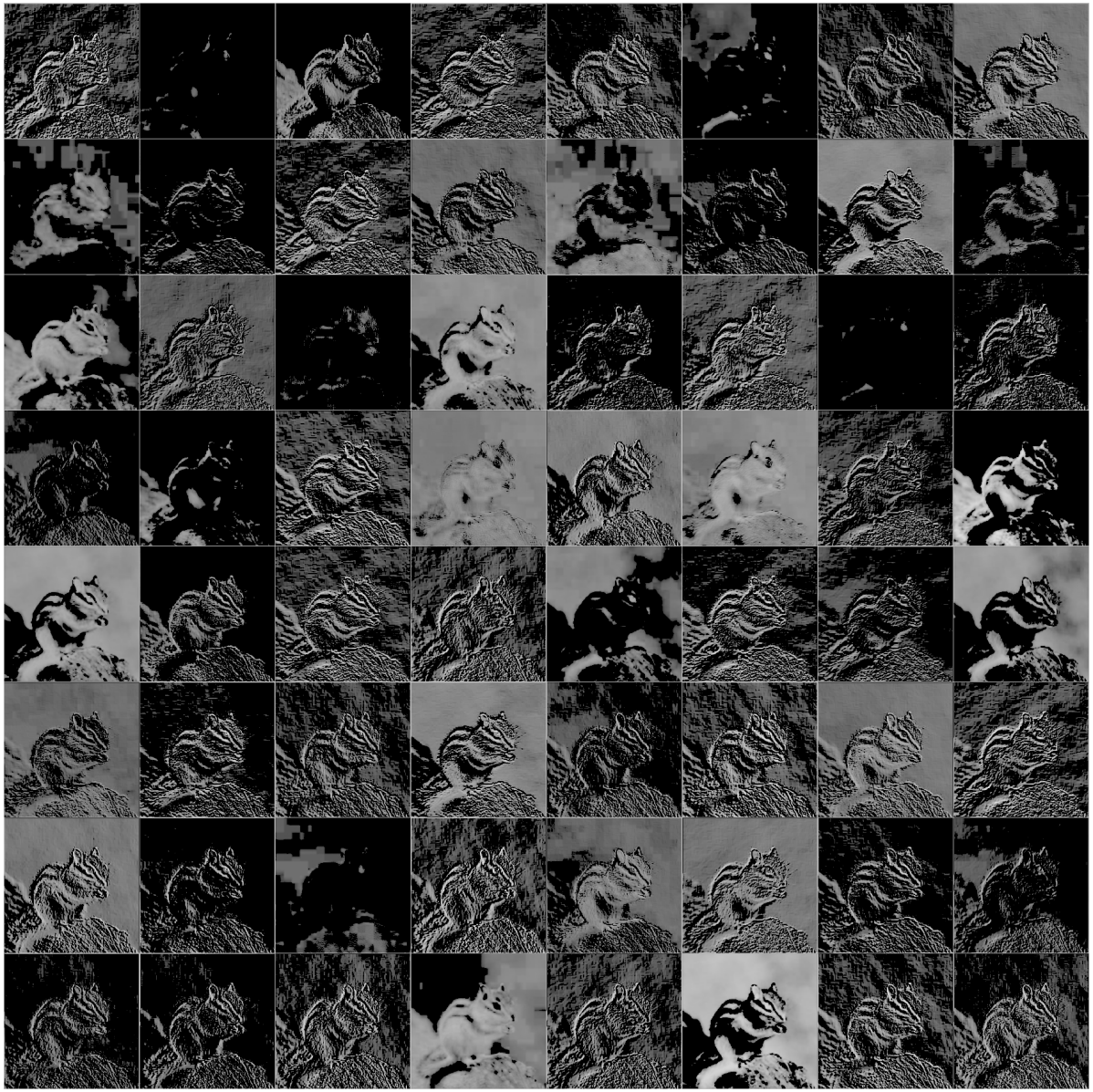
Topology of VGG16

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

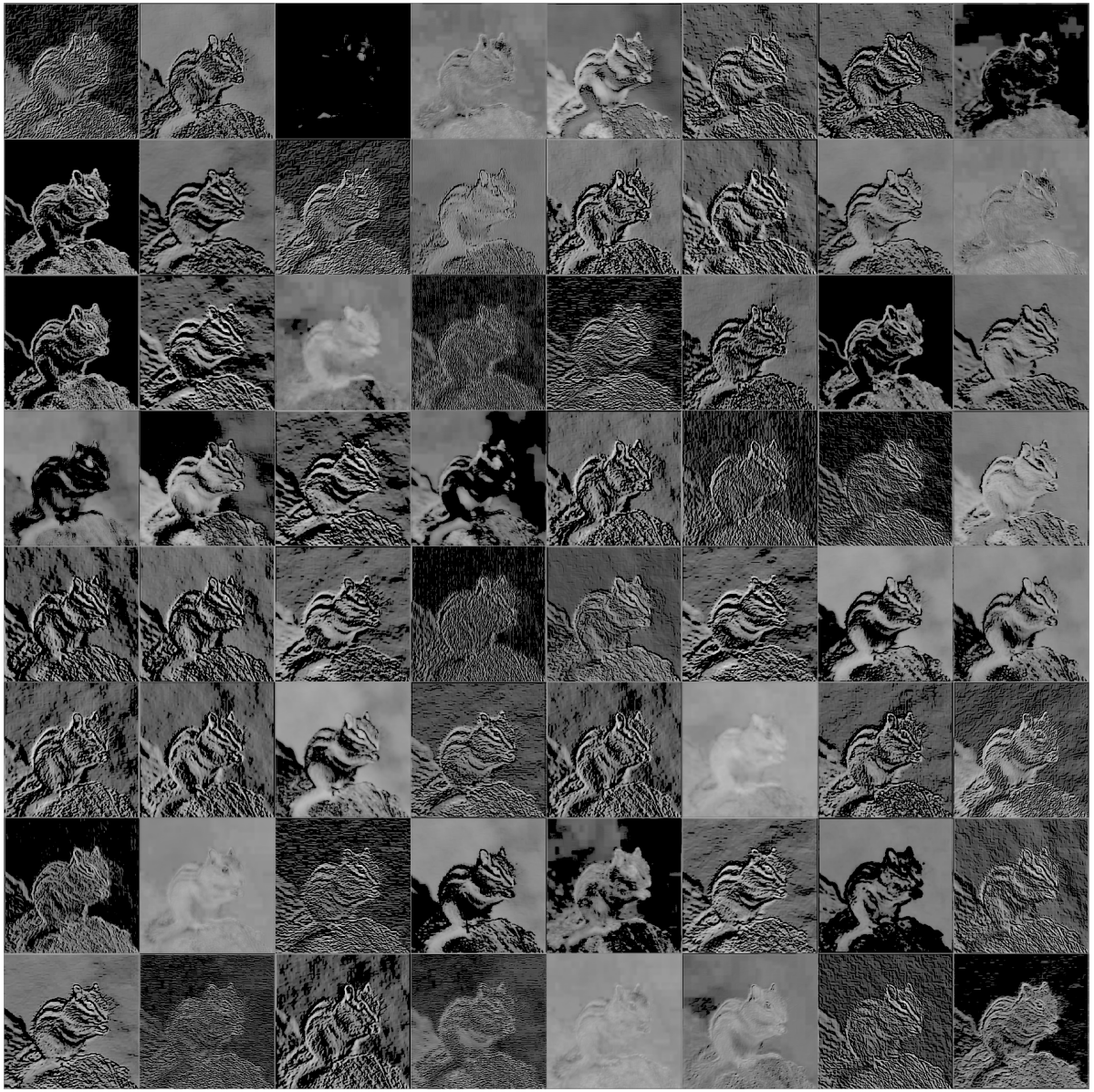
```
VGGmodel = VGG16(weights='imagenet', include_top=False)
VGGmodel.summary()
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)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

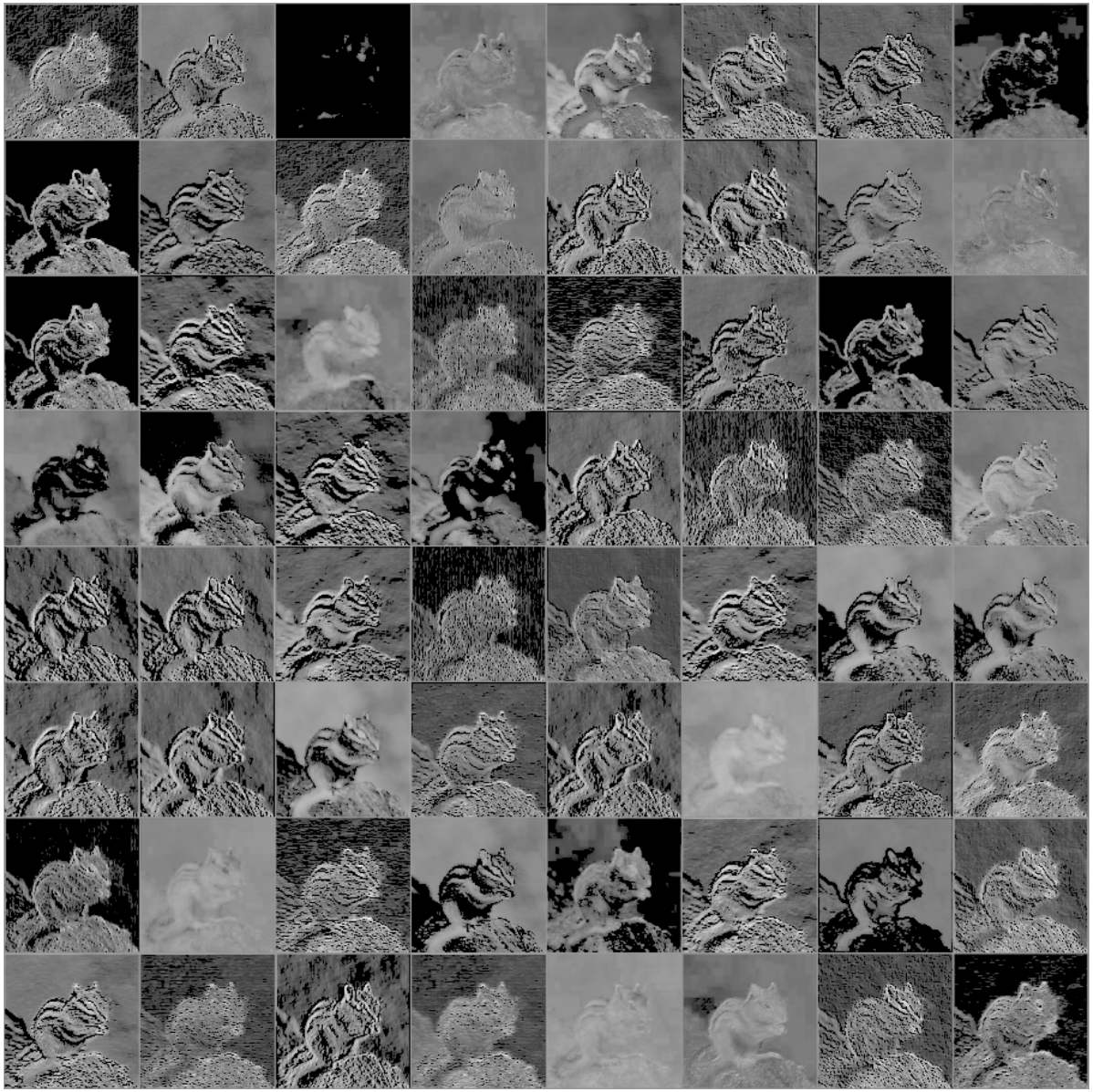
VGG16 : VGGscoiattolo_block1_conv1



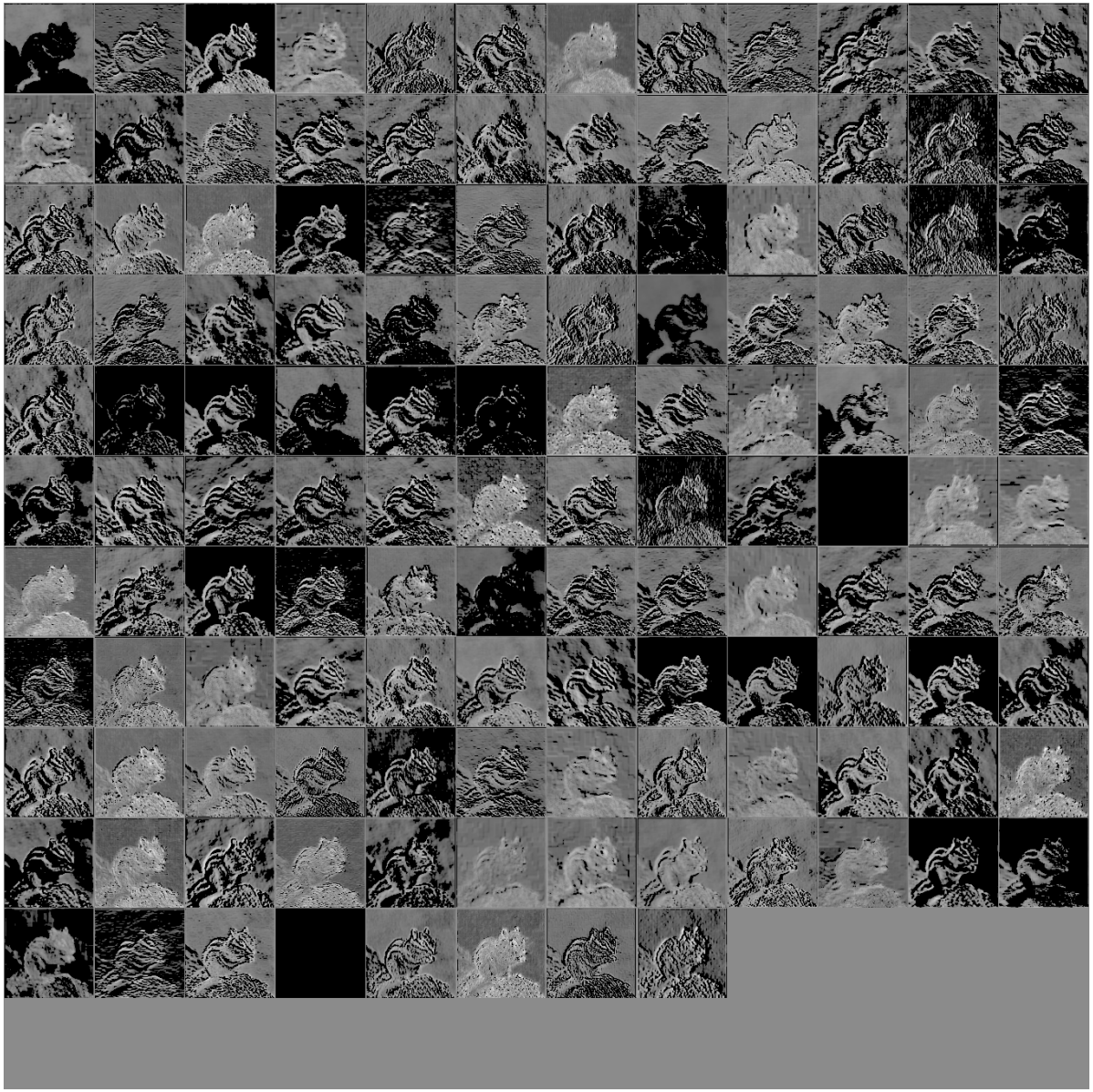
VGG16 : VGGscoiattolo_block1_conv2



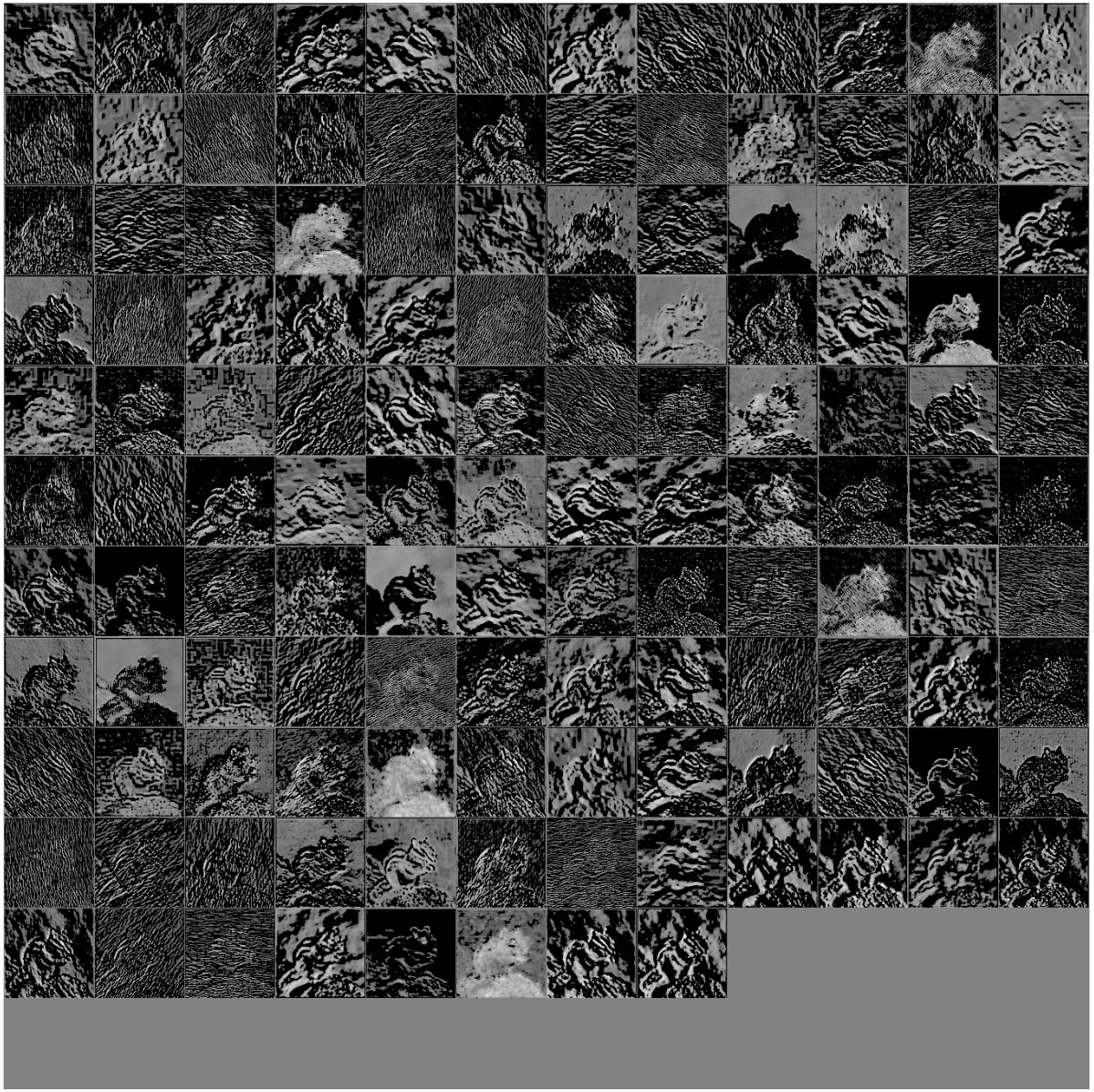
VGG16 : VGGscoiattolo_block1_pool



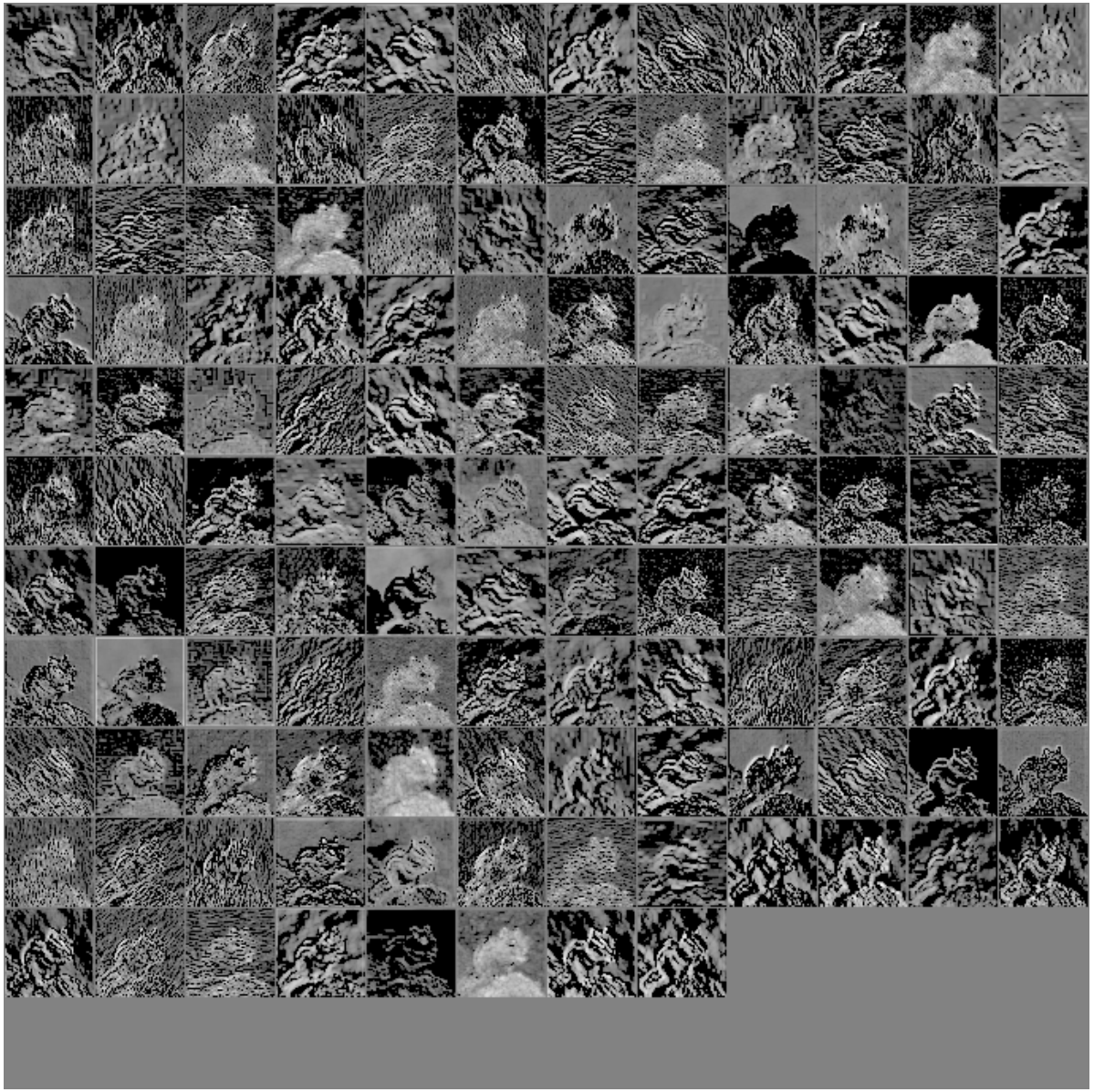
VGG16 : VGGscoiattolo_block2_conv1



VGG16 : VGGscoiattolo_block2_conv2

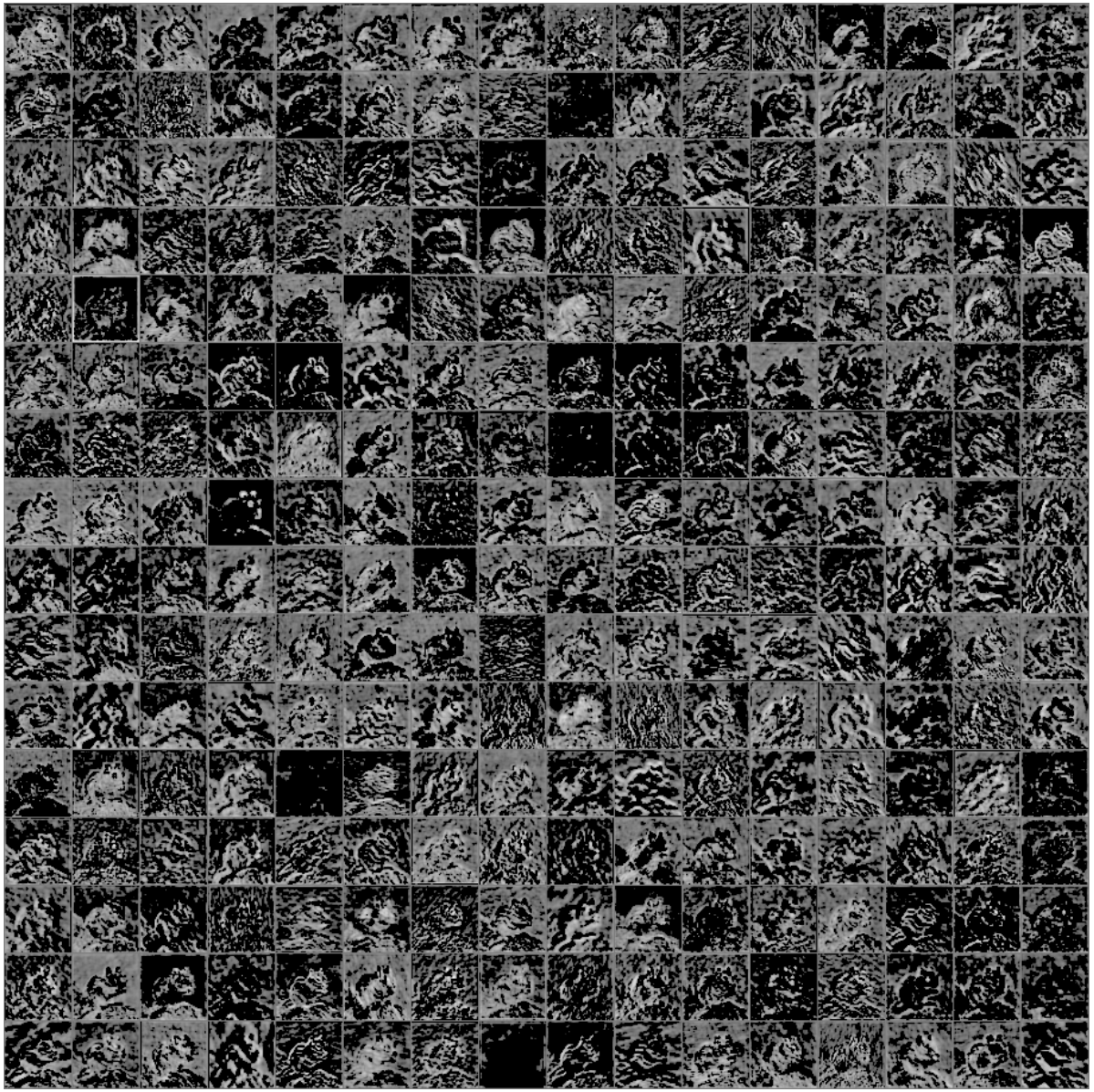


VGG16 : VGGscoiattolo_block2_pool

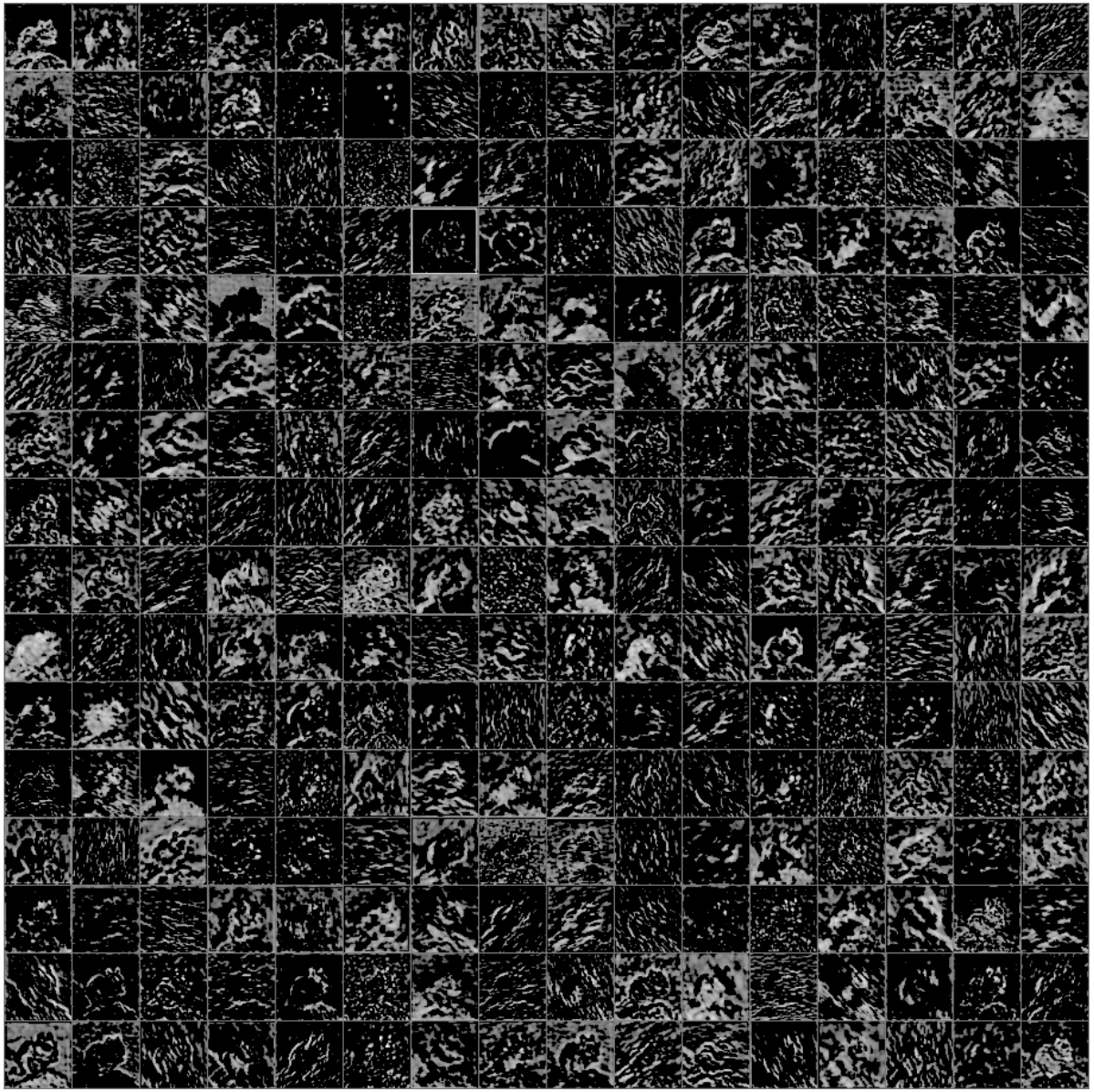


VGG16 : VGGscoiattolo_block3_conv1

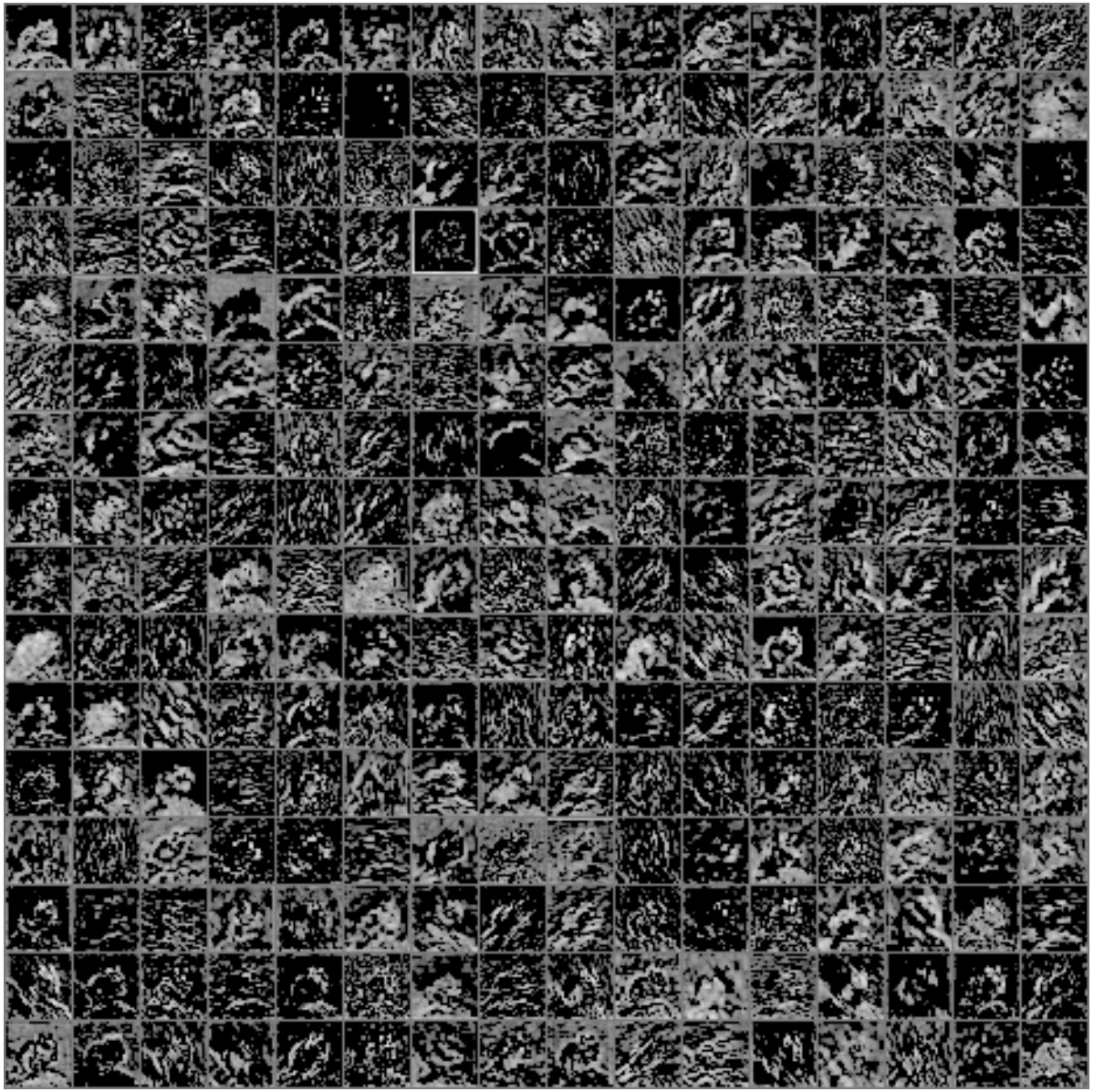
VGG16 : VGGscoiattolo_block3_conv2



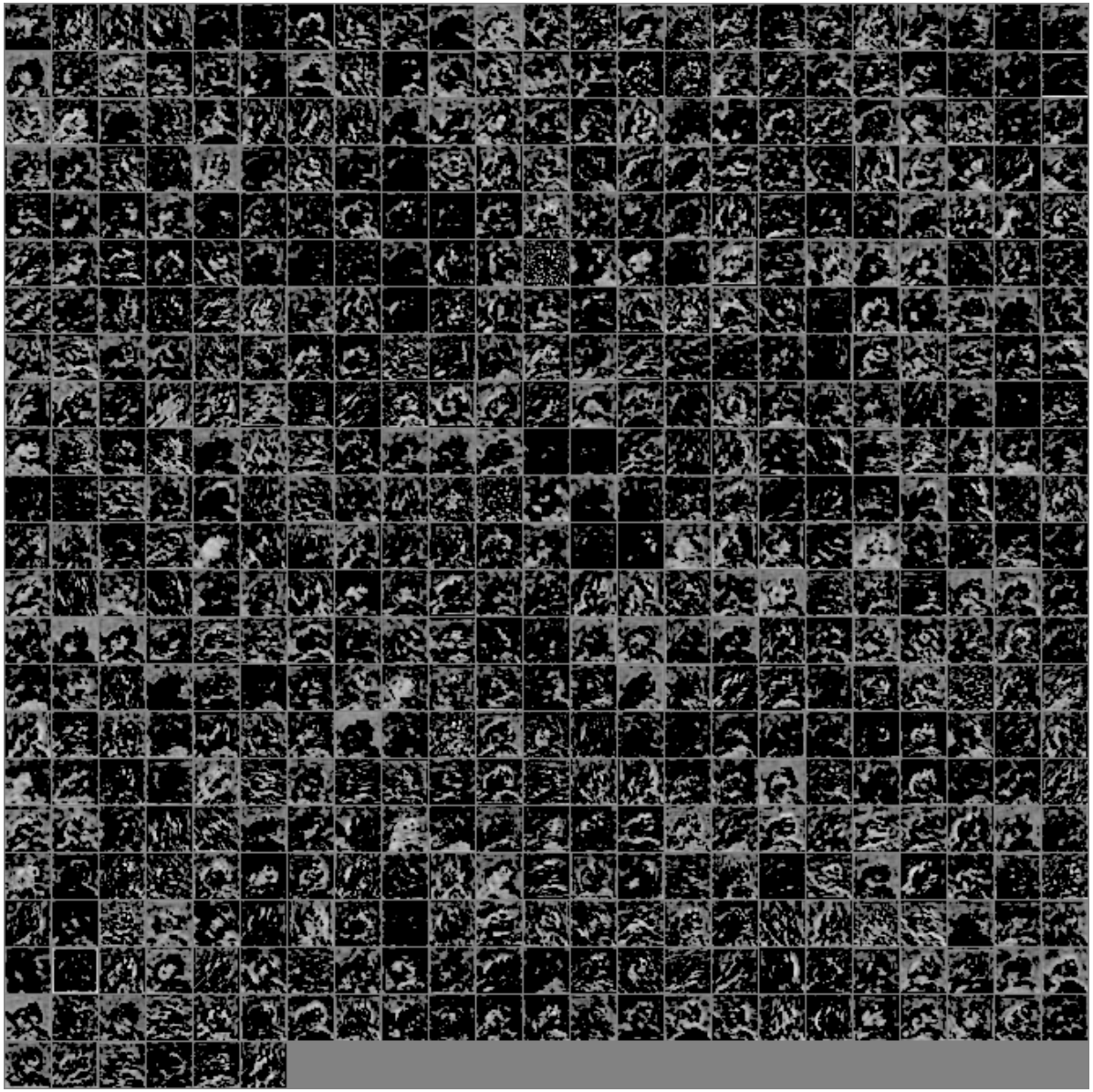
VGG16 : VGGscoiattolo_block3_conv3



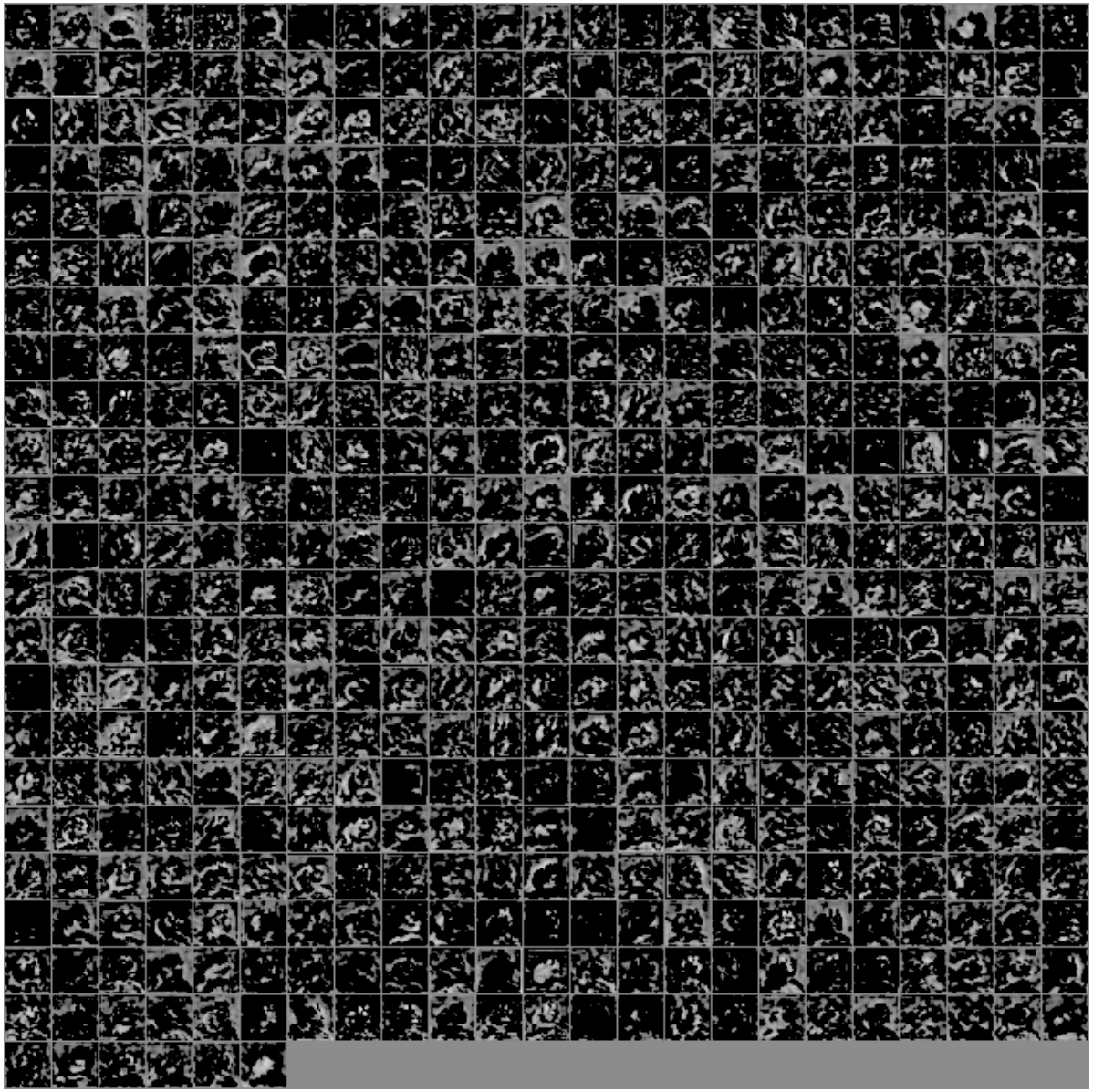
VGG16 : VGGscoiattolo_block3_pool



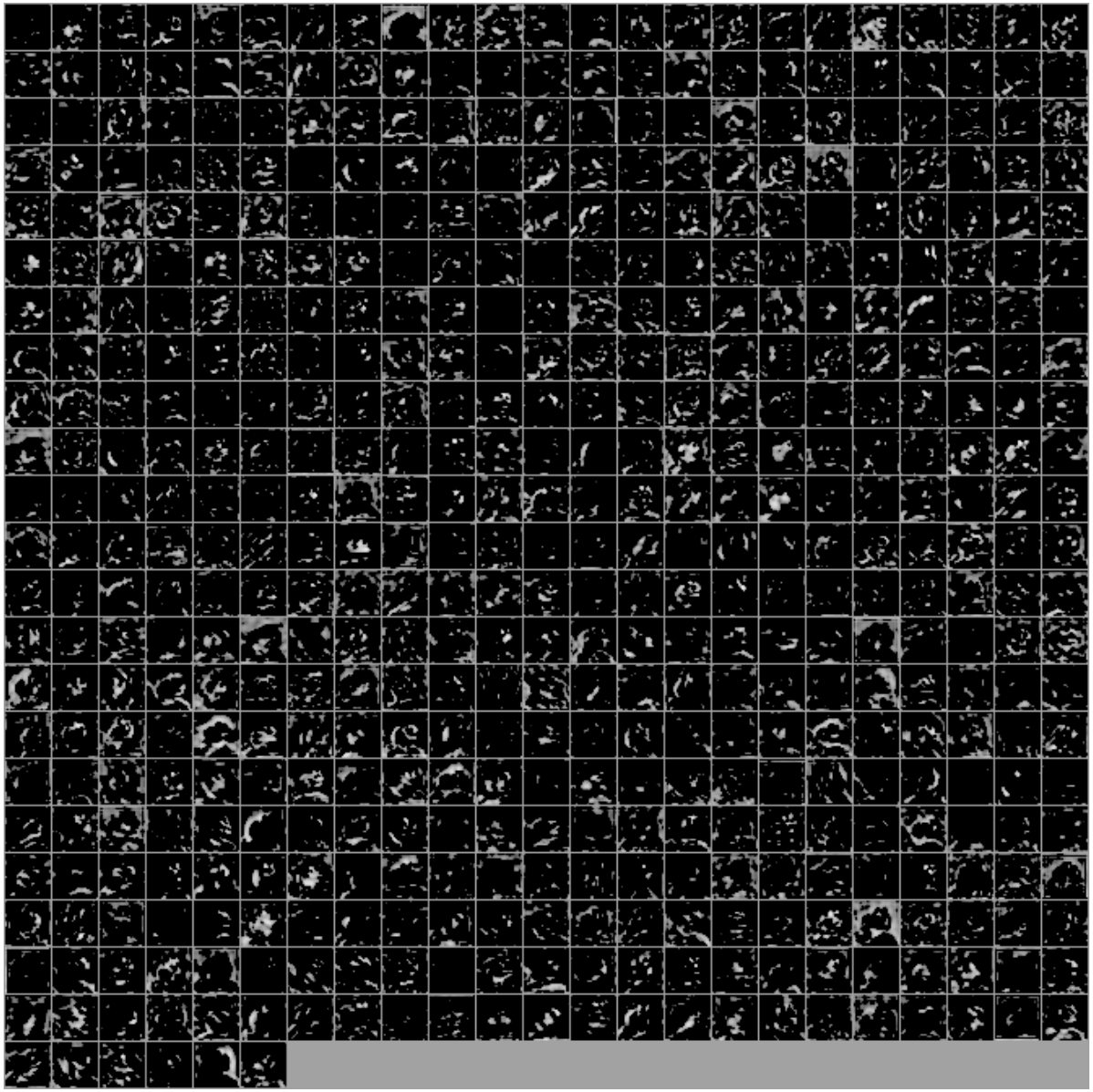
VGG16 : VGGscoiattolo_block4_conv1



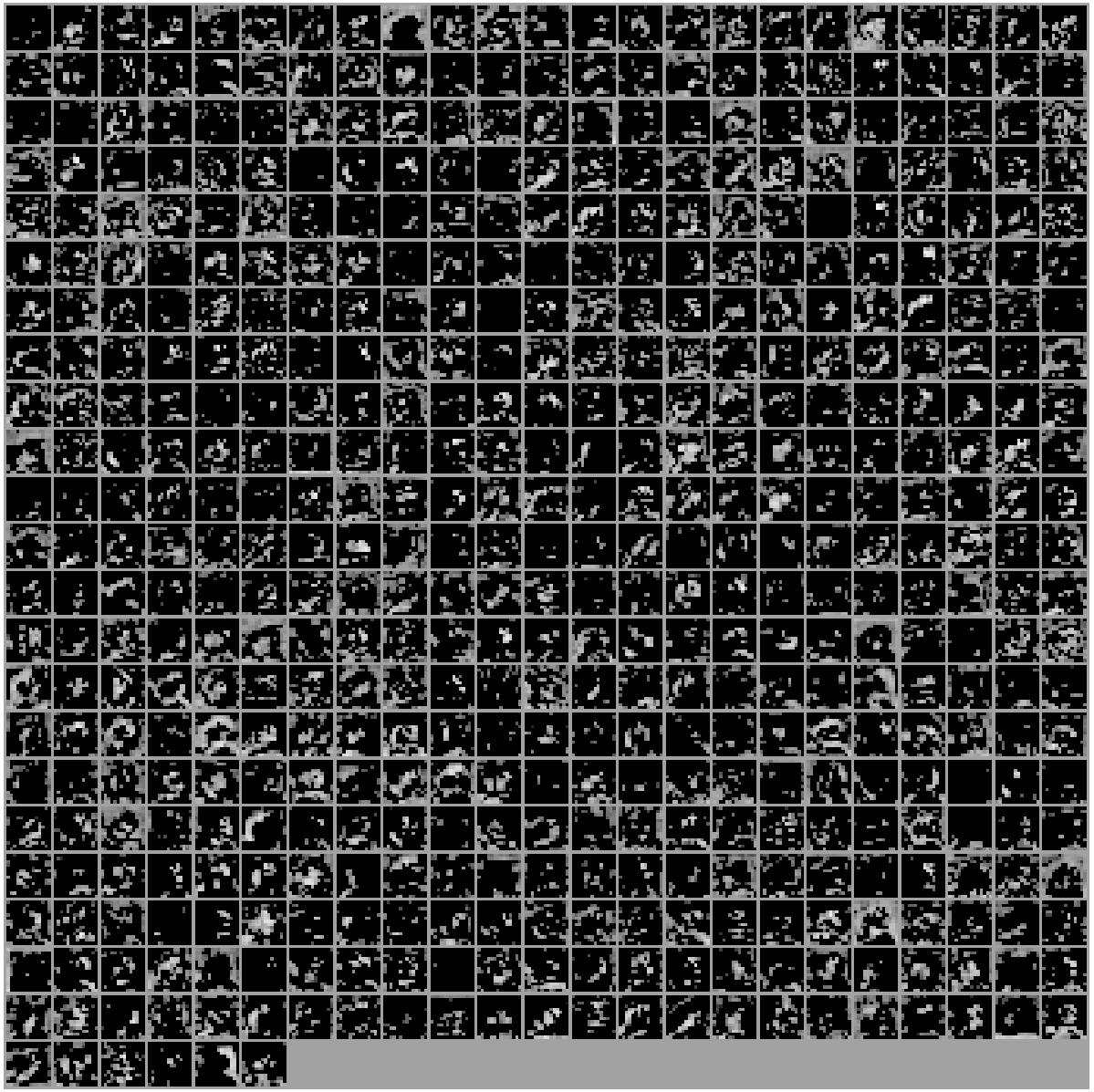
VGG16 : VGGscoiattolo_block4_conv2



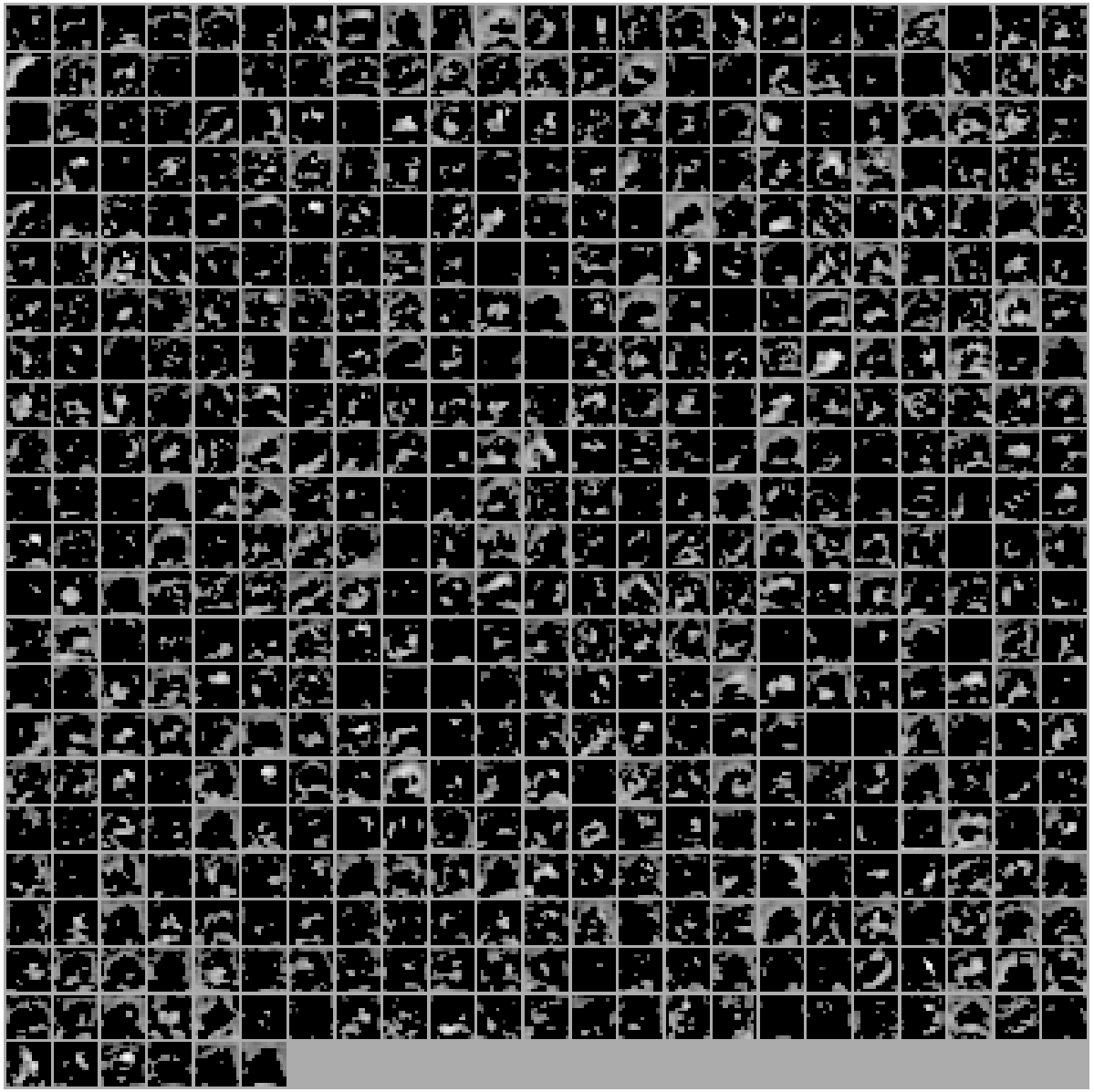
VGG16 : VGGscoiattolo_block4_conv3



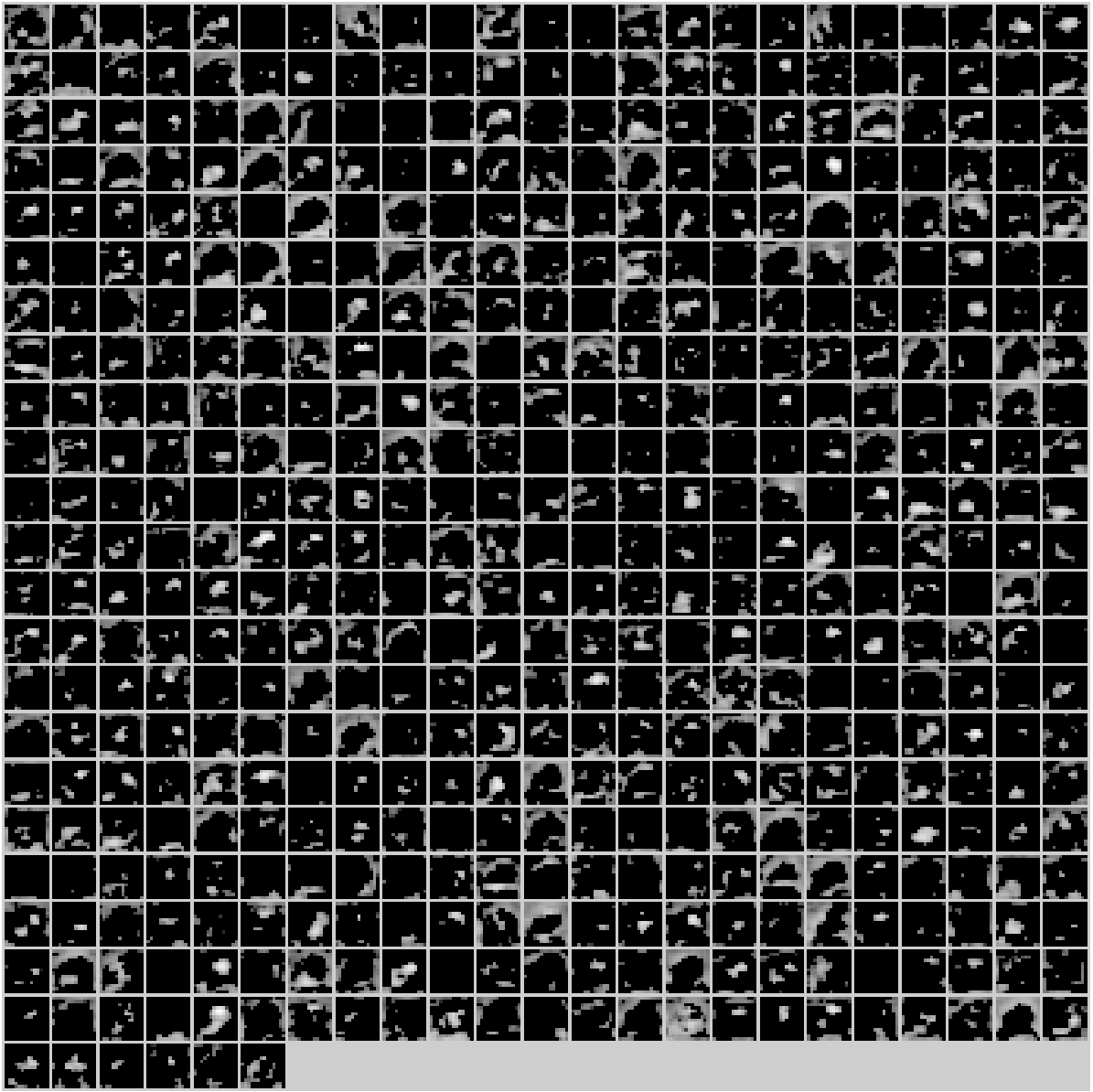
VGG16 : VGGscoiattolo_block4_pool



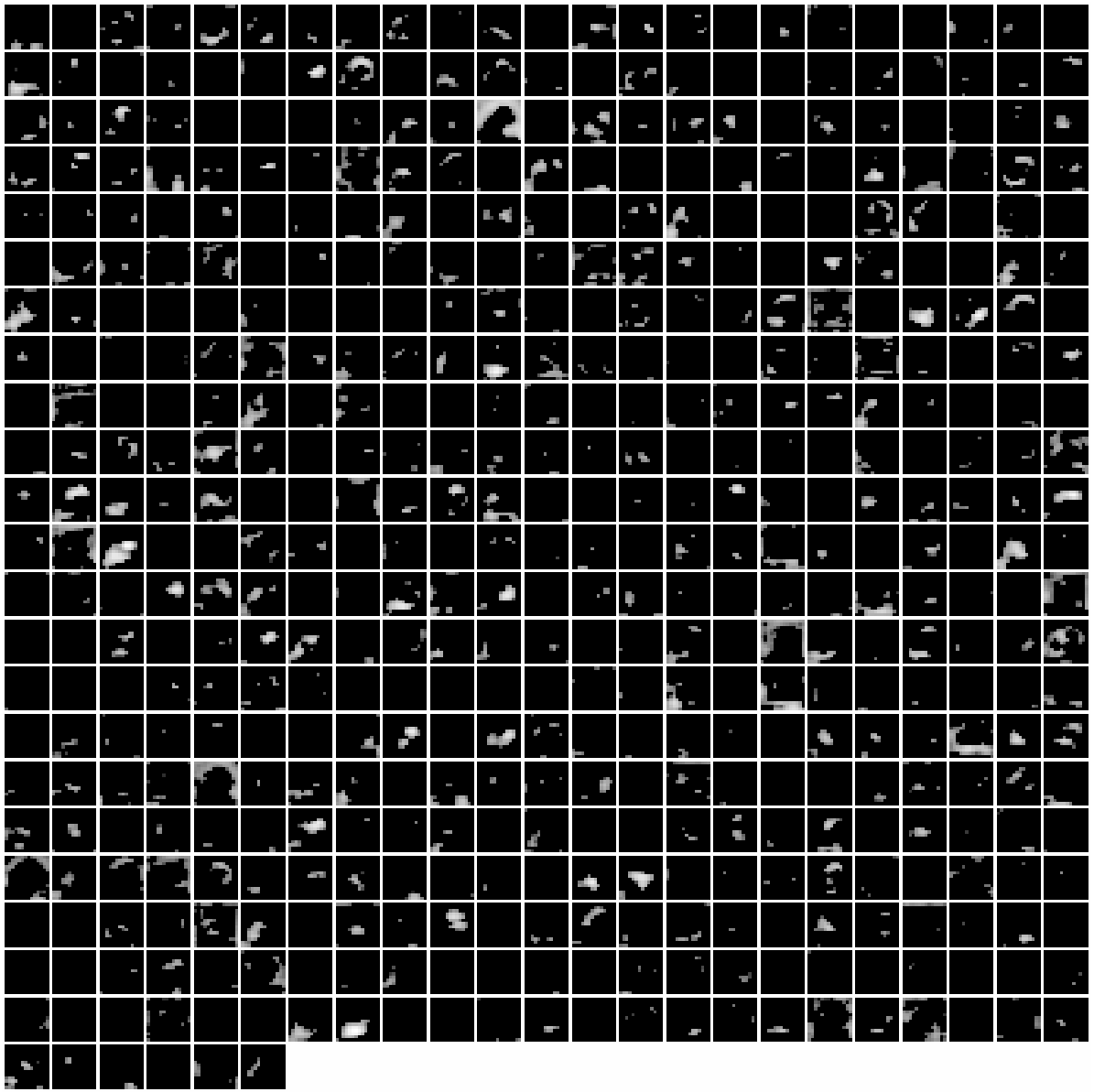
VGG16 : VGGscoiattolo_block5_conv1



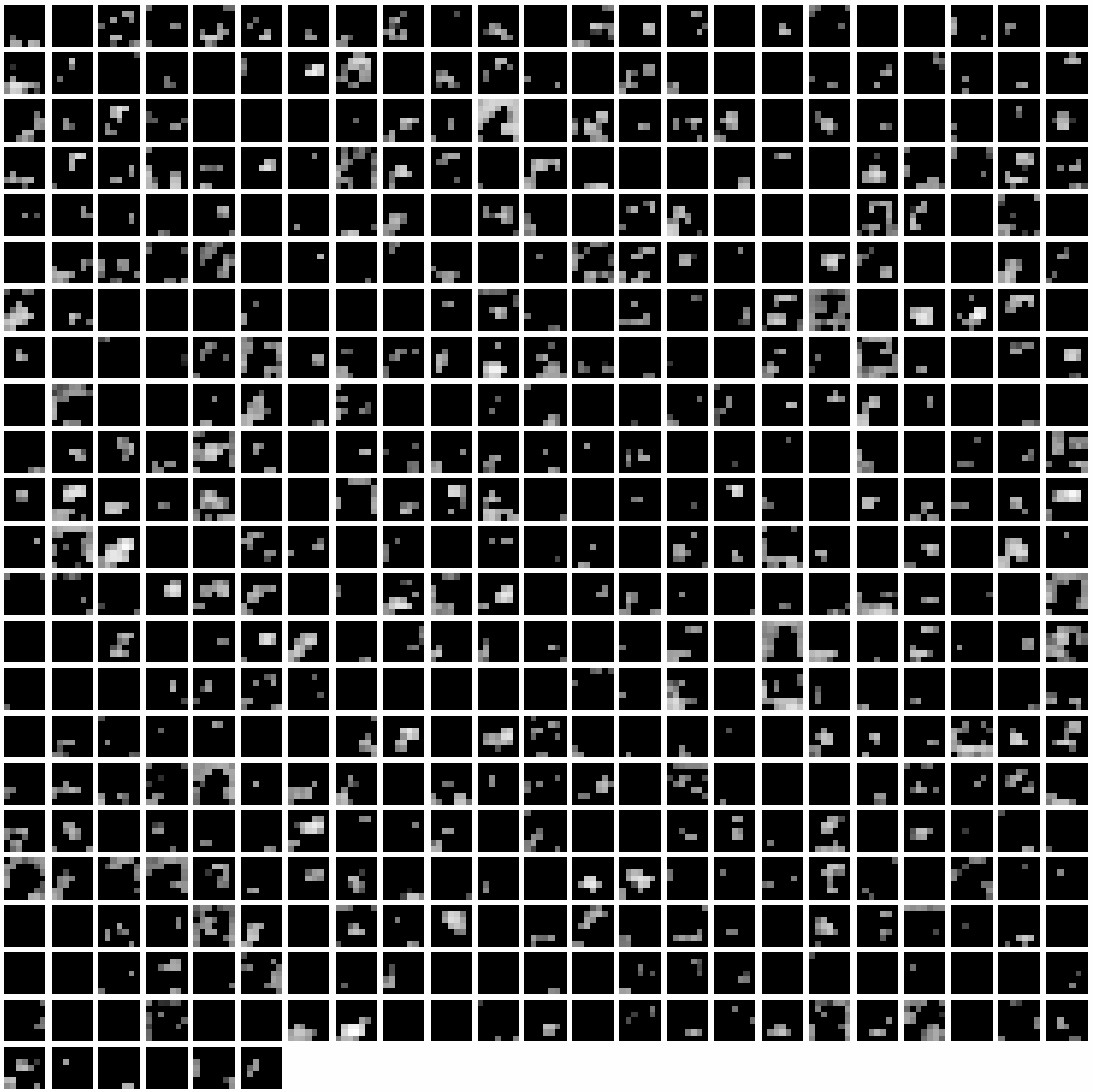
VGG16 : VGGscoiattolo_block5_conv2



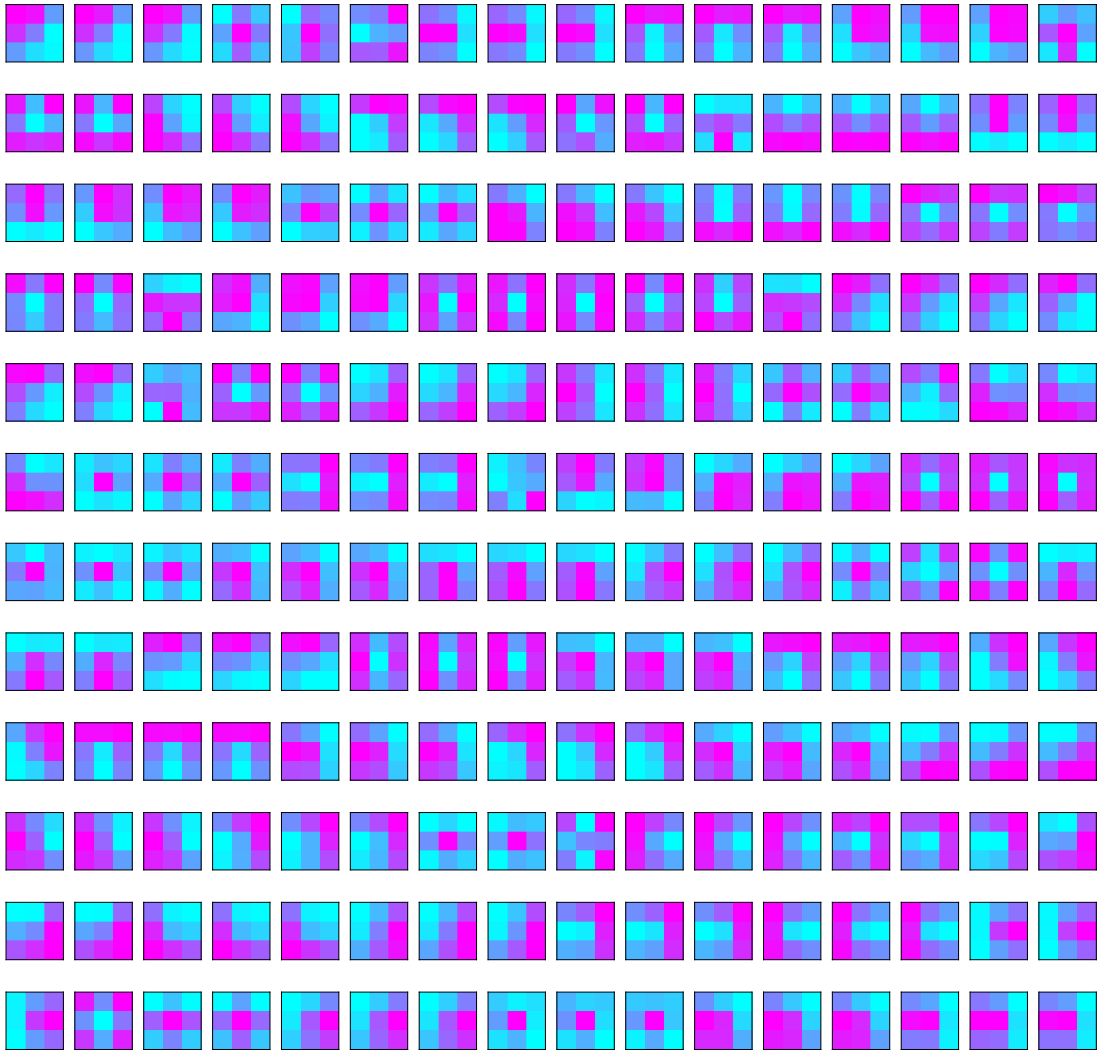
VGG16 : VGGscoiattolo_block5_conv3



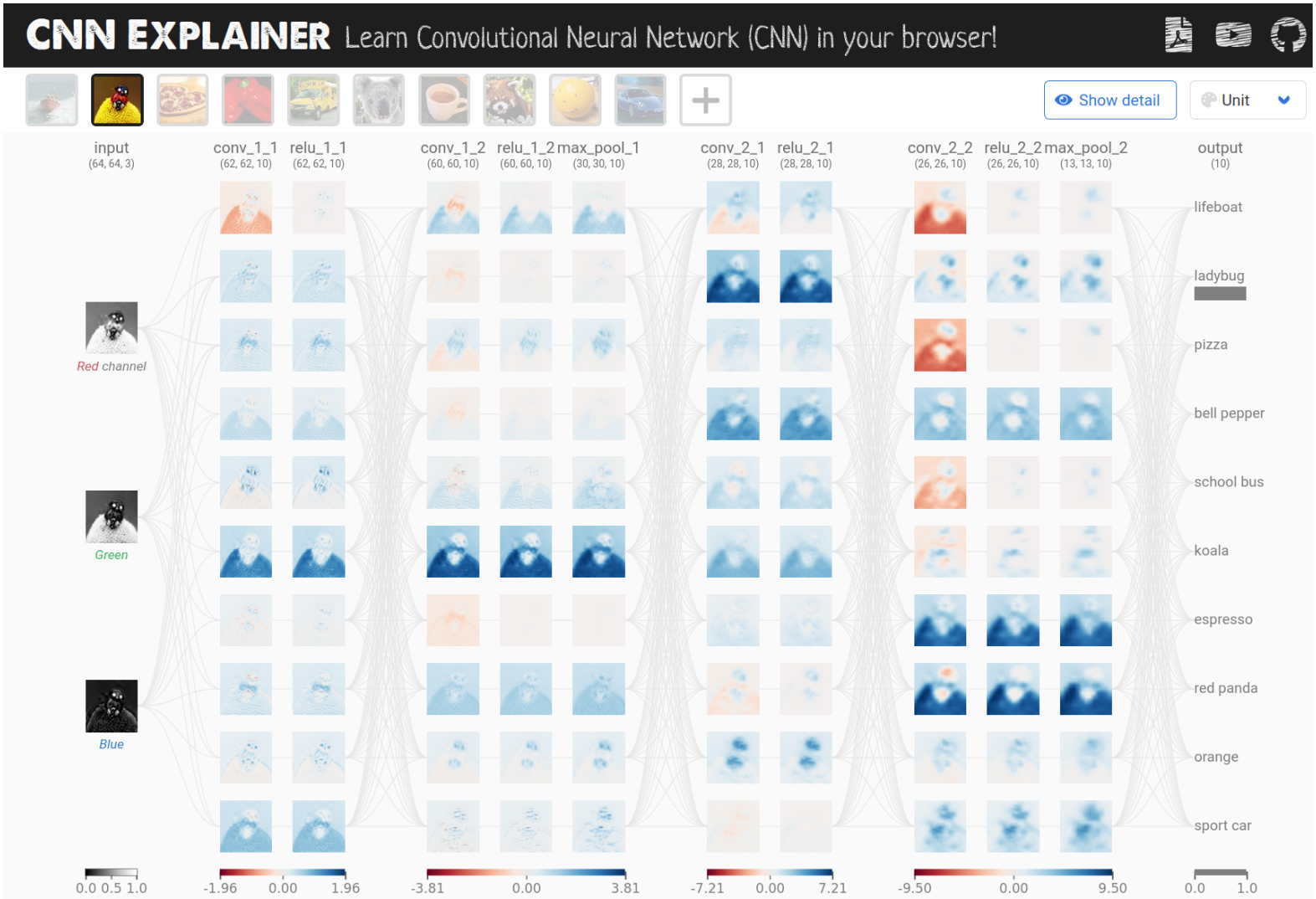
VGG16 : VGGscoiattolo_block5_pool



VGG16 : filters visualisation



CNN explainer



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

Network in Network (*Lin etal 2013*) : 1x1 convolution

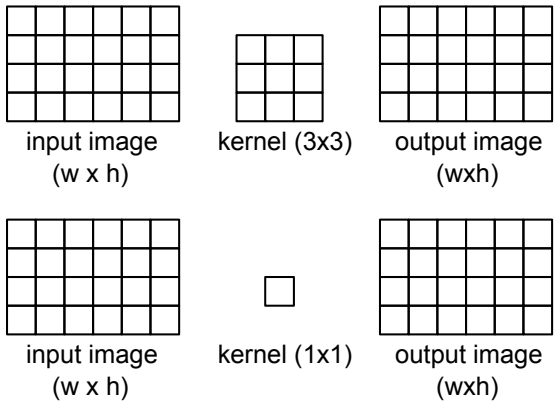
- one channel

  3x3 convolution

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

  1x1 convolution

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



Network in Network (*Lin etal 2013*) : 1x1 convolution

- one channel

  3x3 convolution

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

  1x1 convolution

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

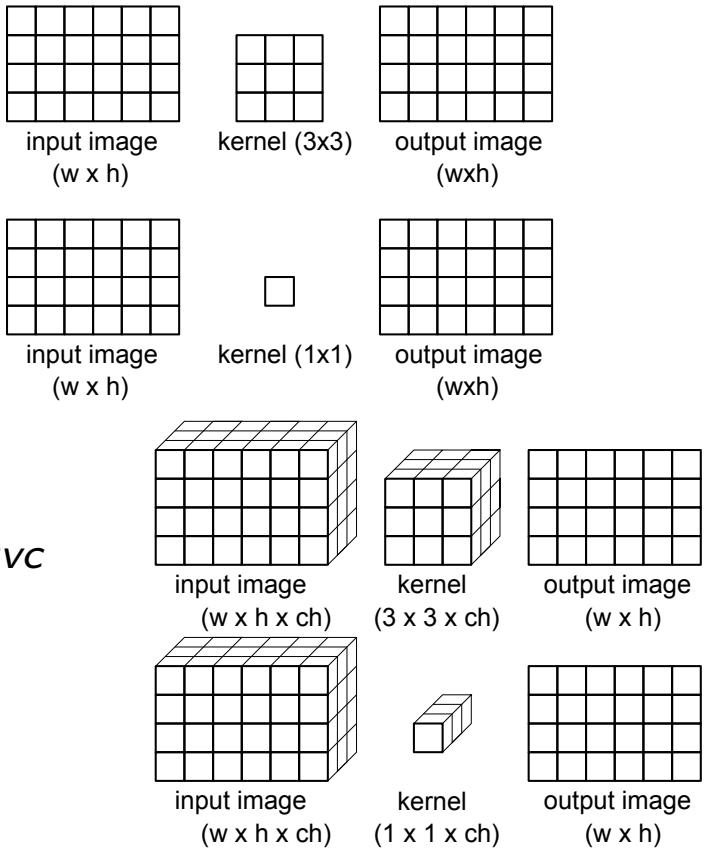
- several channels

  3x3 convolution

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

  1x1 convolution

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



Network in Network (*Lin etal 2013*) : 1x1 convolution

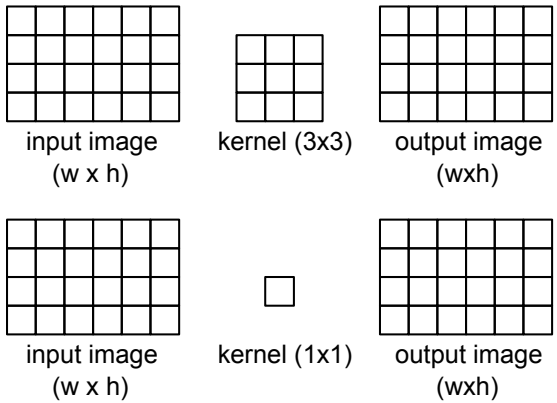
- one channel

  3x3 convolution

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

  1x1 convolution

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



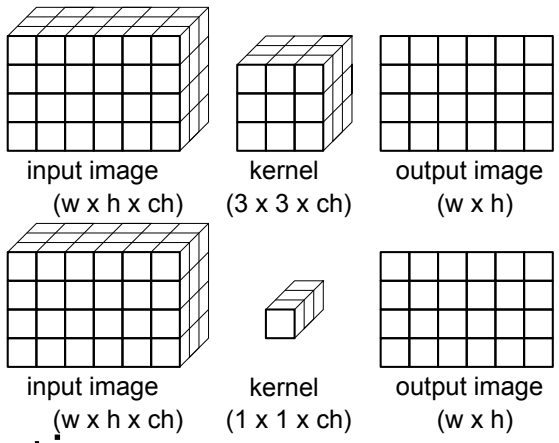
- several channels

  3x3 convolution

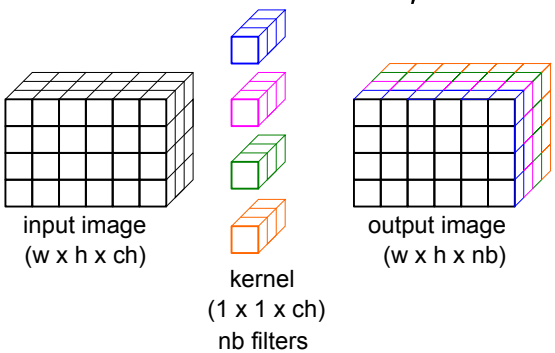
  $$out_{ij} = \sum_c \sum_u \sum_v in_{(i-d+u)\,(j-d+v)\,c}\, k_{uvc}$$

  1x1 convolution

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

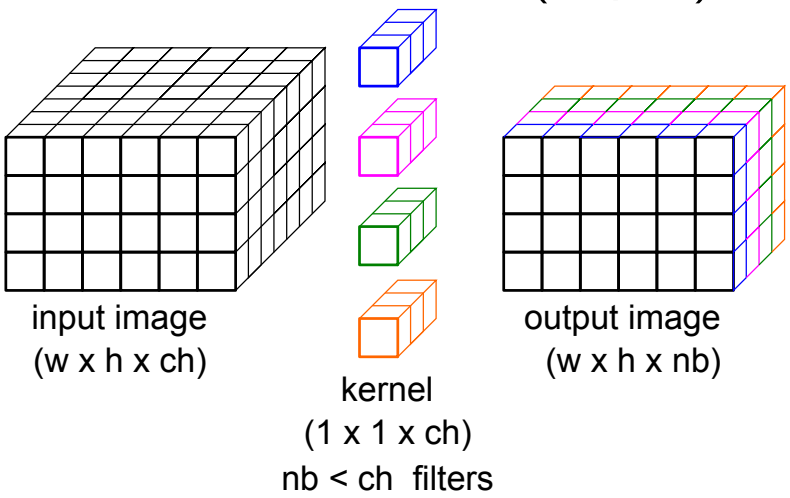


- several channels, several filters and 1x1 convolution

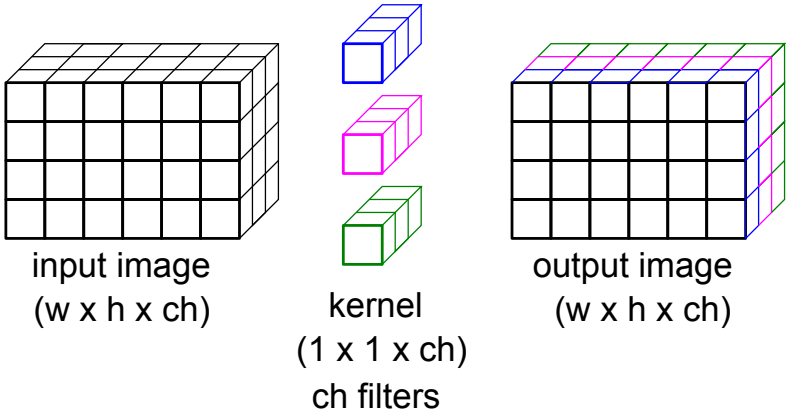


Using 1x1 convolution

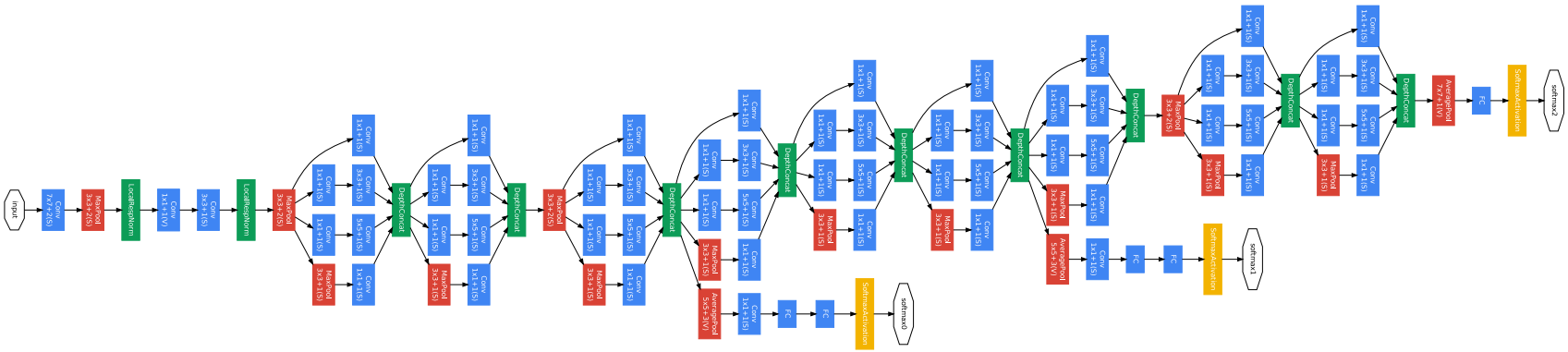
- dimension reduction (depth)



- non-linearity

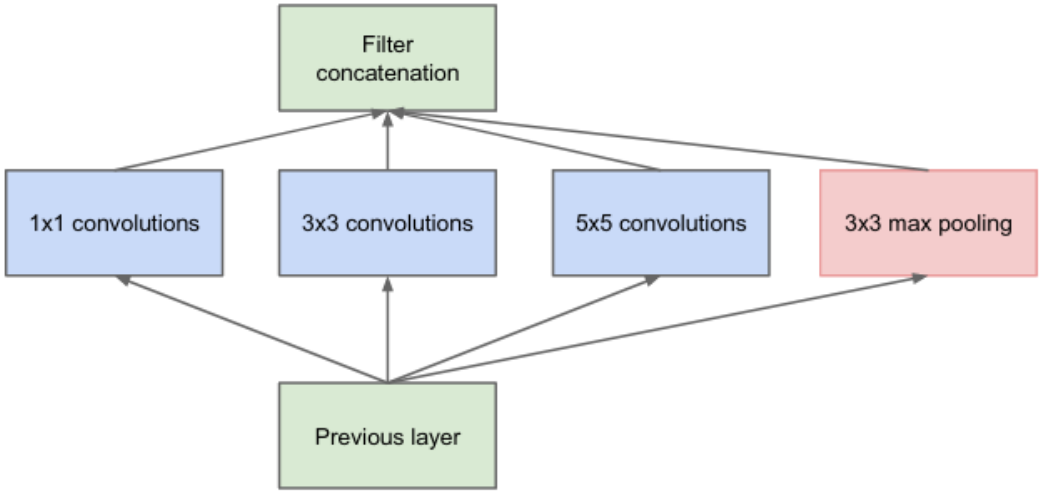


GoogLeNet Inception v1, 2014



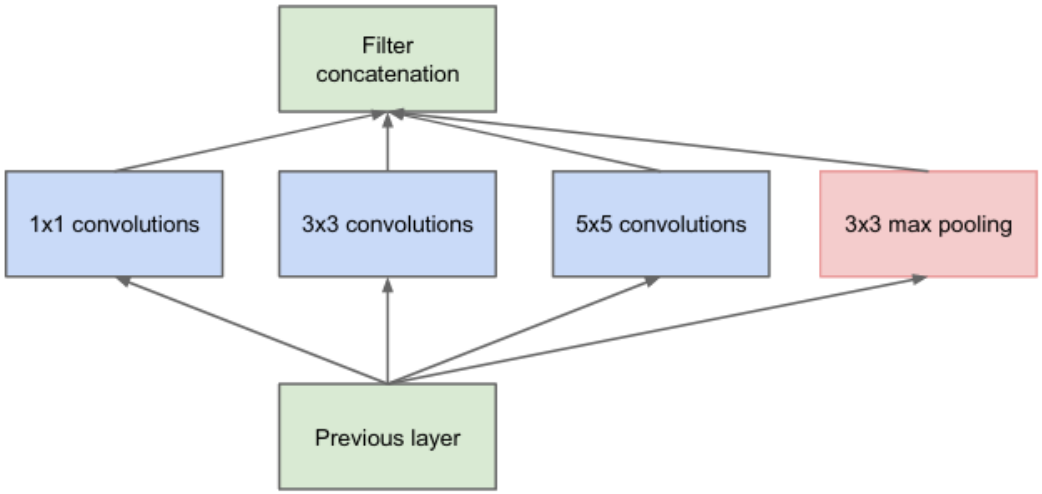
- 22 layers but 12x less params than AlexNet
  - Networks in Network
  - Inception module (1x1, 3x3, 5x5 convolutions and 3x3 max pool.)
    - do not choose the conv. size but try them all
  - paper : `https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf`
  - auxiliary classification outputs : inject add.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

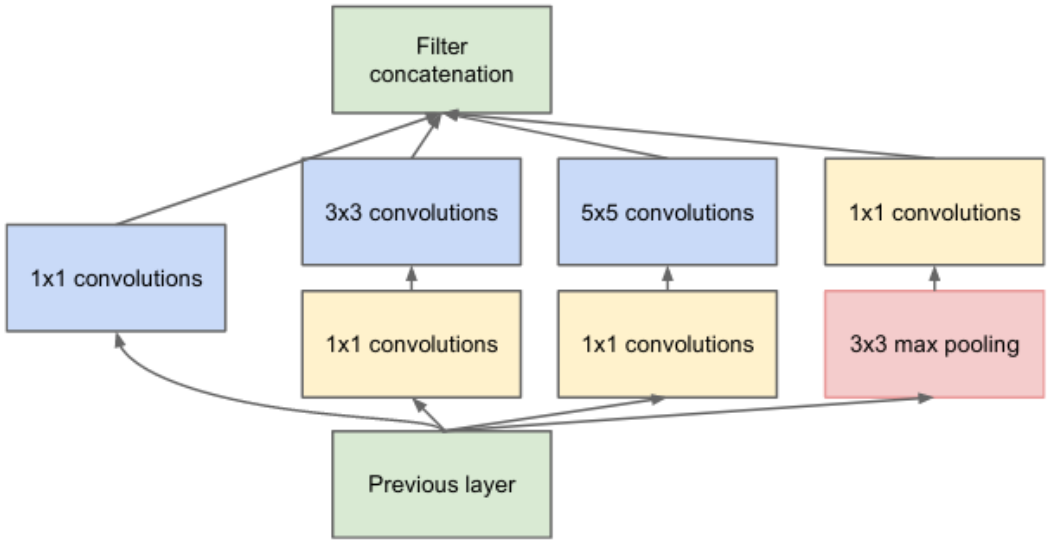
Naive Inception module



(a) Inception module, naïve version

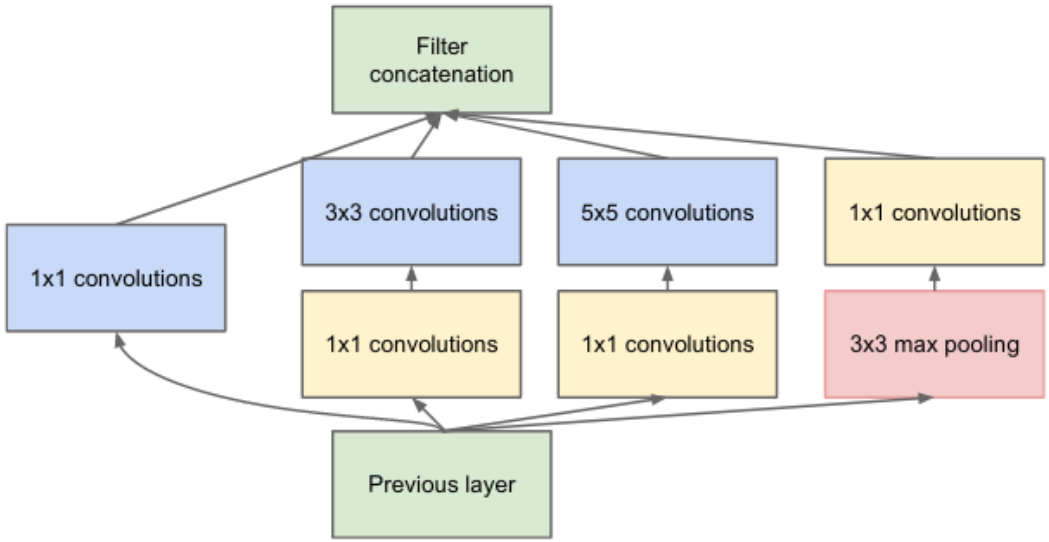
- Example of the 5x5 convolutions (96 filters) with a previous layer of dim w*h*ch
  - nb of coeff : (25*ch+1)*96
  - nb of op : w*h*96*25*ch

Inception module



(b) Inception module with dimensionality reduction

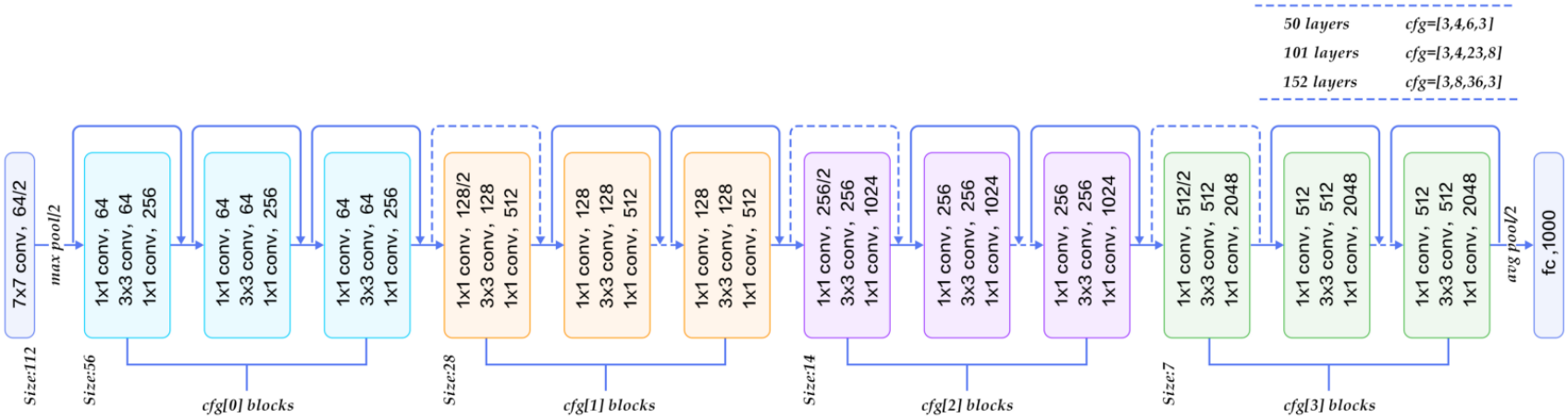
Inception module



(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*h*96*25*nb + w*h*nb*1*ch = w*h*(96*25*nb+nb*ch)

ResNet, 2015



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

Residual blocks

- Going deeper is not so easy

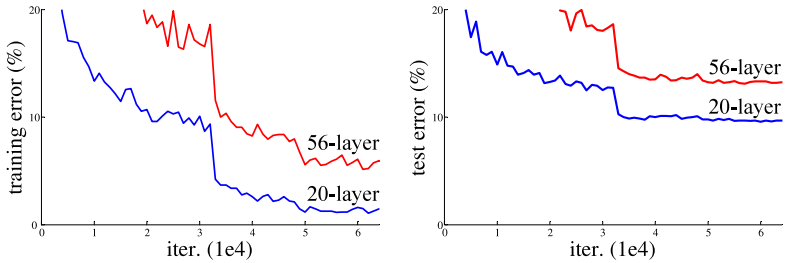


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error.

- Residual block : ease the learning

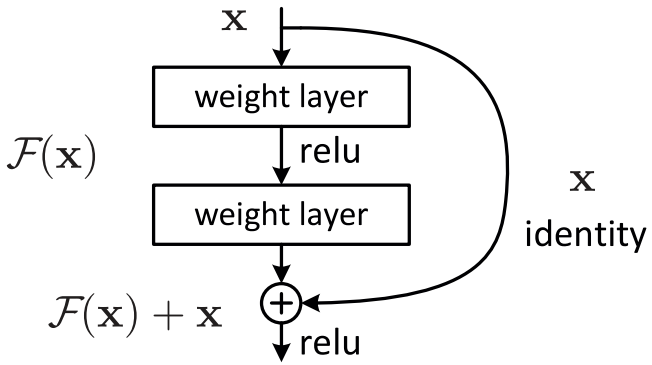
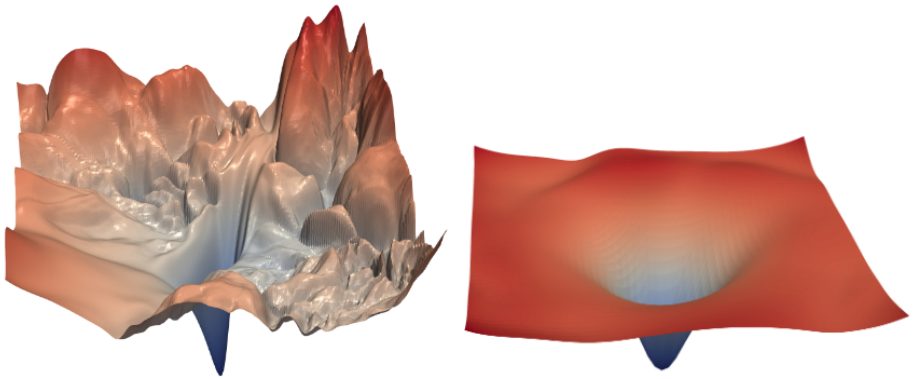


Figure 2. Residual learning: a building block.

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

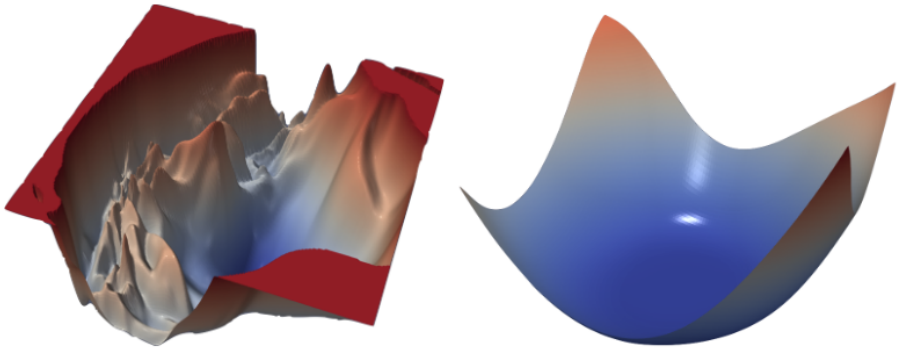
Residual connection as regularisation

From Visualizing the Loss Landscape of Neural Nets :



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

Figure 4: The loss surfaces of ResNet-110-noshort and DenseNet for CIFAR-10.

ResNet efficienc

y

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

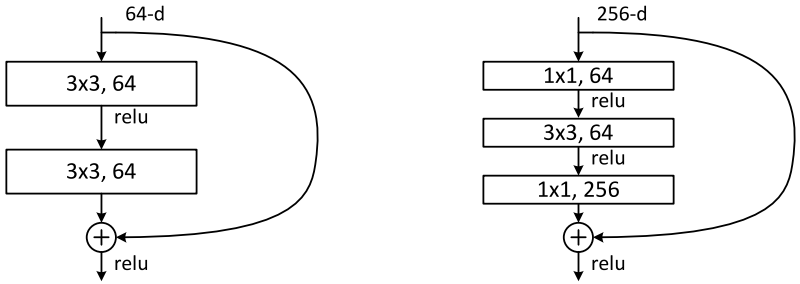
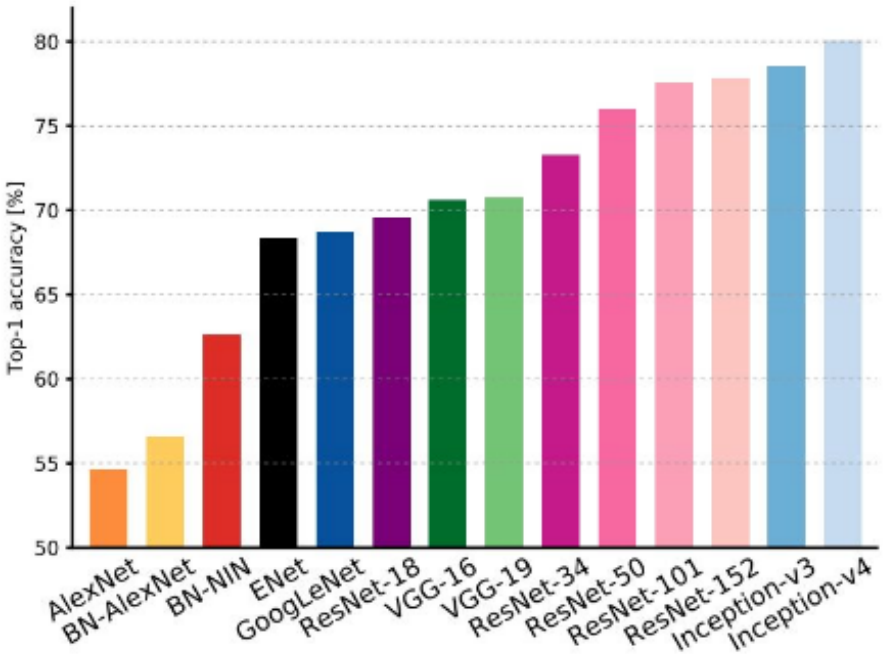
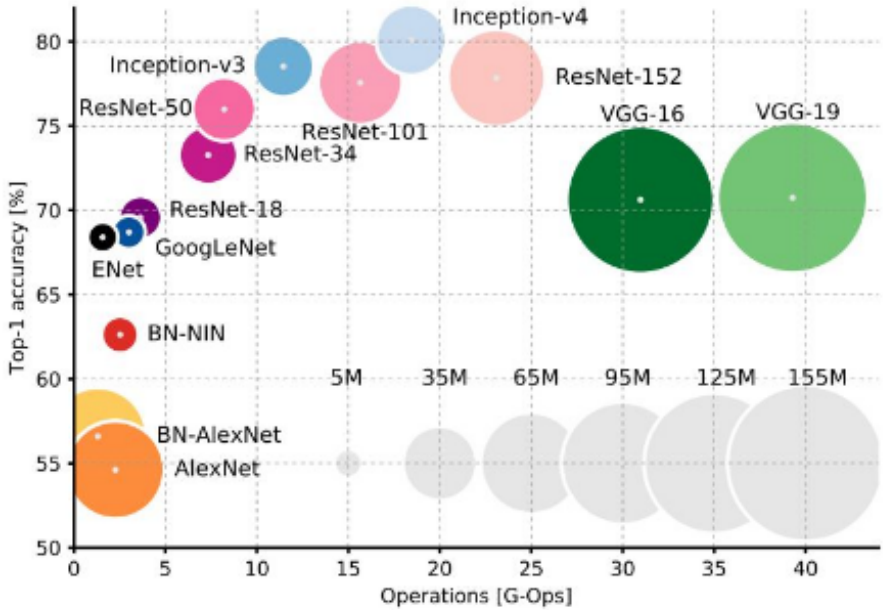


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

- Inception-v4 : ResNet + Inception

Different architectures





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

Xception, 2017

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

Spatially separable convolutions

- Example of one Sobel filter :

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

  - 6 parameters instead of 9
- kernel of size k*k, stride 1 and padding 0 on image of dimensions w*h
  - trad. conv. : $(w - 2 * k//2) * (h - 2 * k//2) * k * k$ multiplications
  - sep. conv. : $w * (h - 2 * k//2) * k + (w - 2 * k//2) * (h - 2 * k//2) * k$
  - ratio sep/trad : $\frac{2}{k} + \frac{2*k//2}{k(w-2*k//2)}$
    - if k small compared to w and h : ratio $\simeq \frac{2}{m}$
    - for a 3x3 kernel : $\frac{2}{3}$
- However, not all kernel are spatially separable

Depthwise separable convolutions

- n filters of size $k * k$ applied to a wxhxch image
- replaces by :
  - ch filters of dim $k * k$
  - n filters of dim $1 * 1 * ch$
- how many operations ?
  - standard conv. : w*h*ch*k*k*n
  - sep. conv. : w*h*ch*k*k + ch*w*h*n
  - ratio : $\frac{1}{n} + \frac{1}{k*k}$
    - if n=128 and k= 3 : ratio = 12%

Grouped convolution

- initially in AlexNet
- separation of filters in different groups
  - each group correspond to a group of channels
- example of 2 groups
  - each group is composed by n/2 filters and is applied to ch/2
  - at the output, the results are contatened
- if number of groups = ch : equivalent to depthwise sep.
- used in ResNeXt
  - easy to parallelize
  - less parameters
  - better model (read `https://blog.yani.ai/filter-group-tutorial/`)

EfficientNet, 2019

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

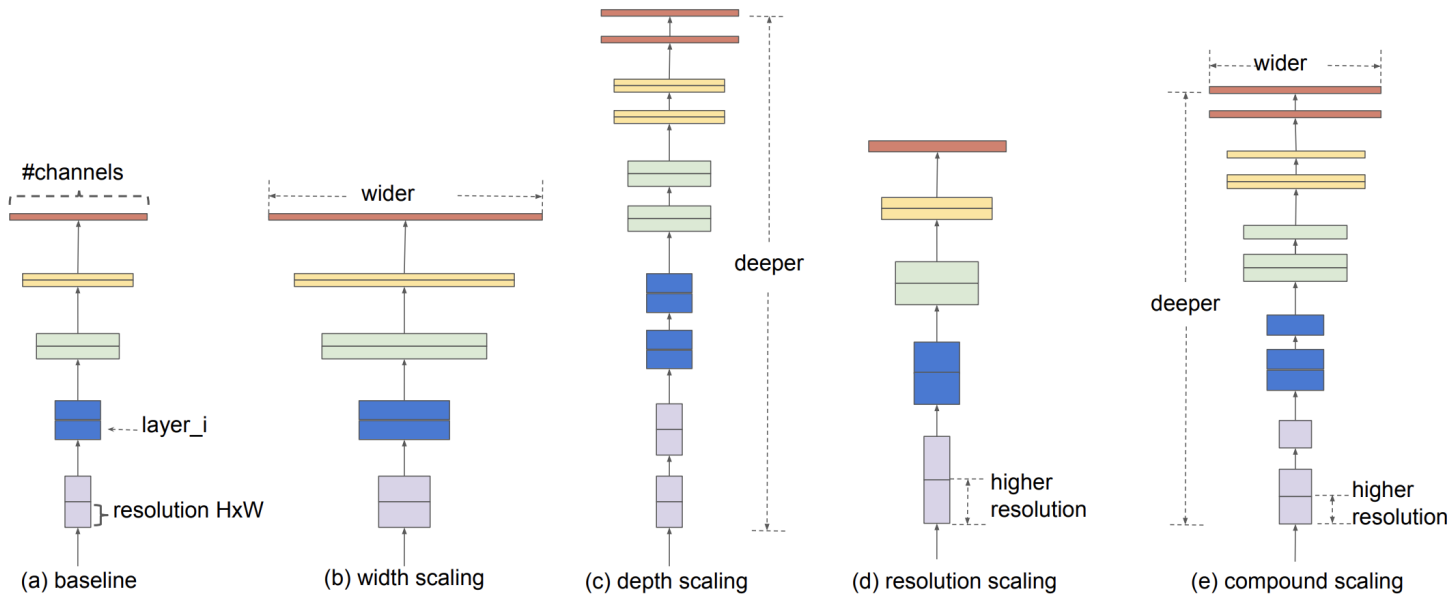
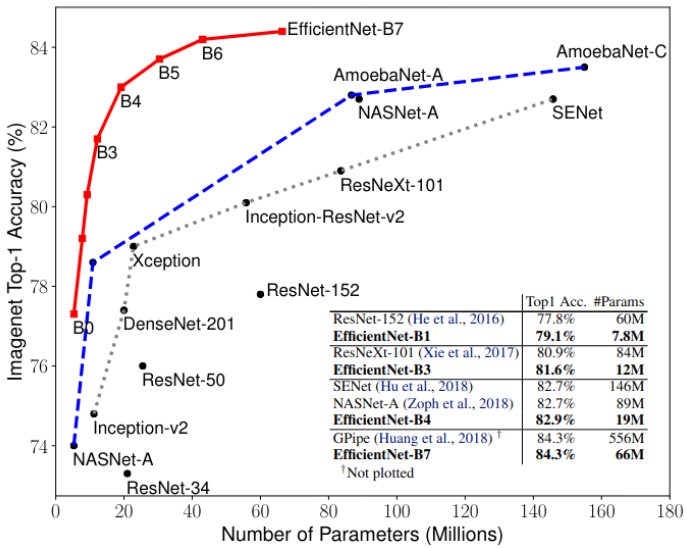


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

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



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

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 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (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/`

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
  - or for a regression task
- not only images but also any 2D data
  - ex. of sound recognition using :
    - spectrogram of sound as an image
    - transfert learning for sound classification
- art style transfert

Classification of a single image

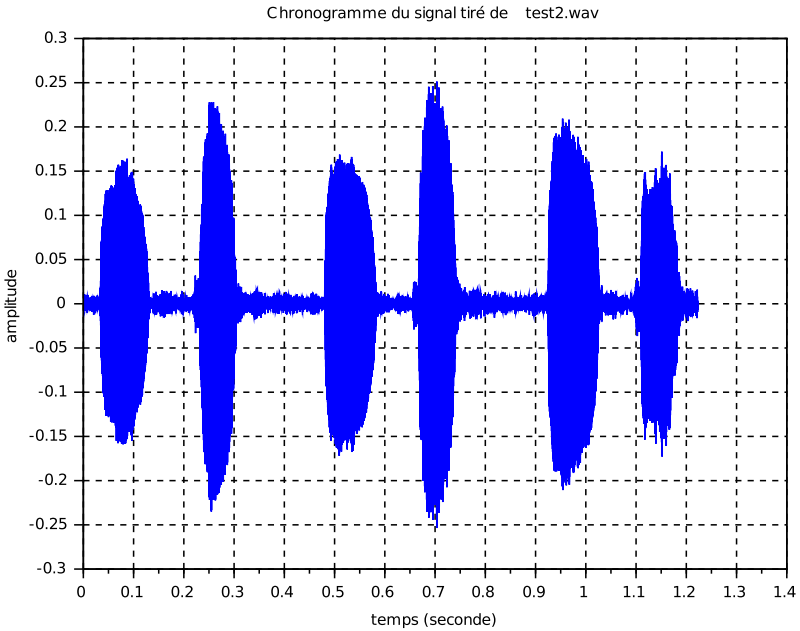
```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

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

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)

classNumber = model.predict(x)
```

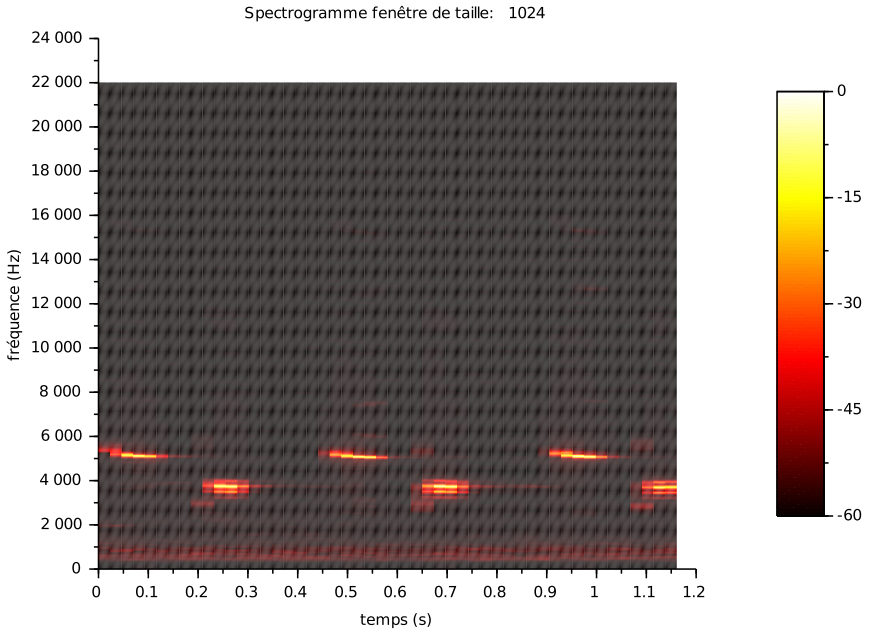
Image descriptor using keras

Example from https://keras.io/api/applications/#usage-examples-for-image-classification-models

```
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





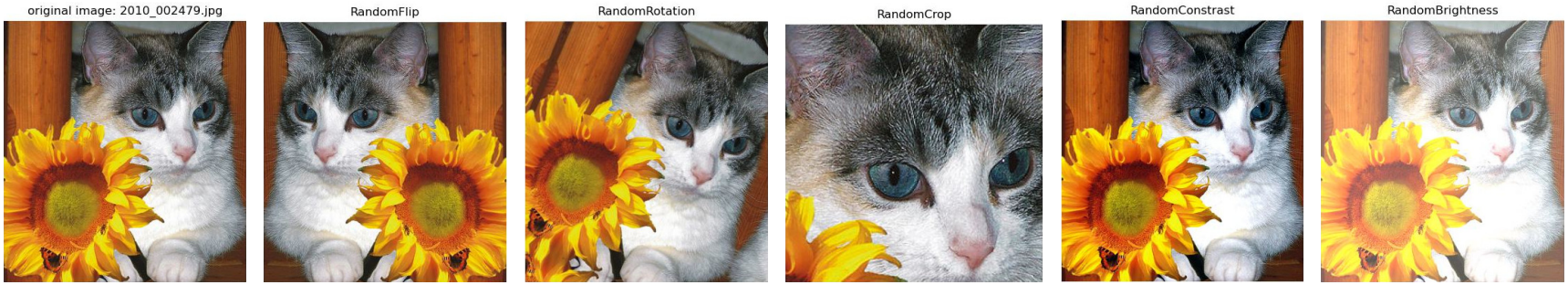
Outline

Why data augmentation ?

- Add new data
  - enable training of complexe functions when more data is needed
- Improve the generalisation
- Robustness

Simple transformations

- geometric transformations
  - crop, rotation, translation, flip, scaling
  - `keras.layer.RandomCrop`, `RandomRotation`, `RandomTranslation`, `RandomFlip`, `RandomZoom`,
- intensity tranformation
  - brightness, contrast ...
  - `RandomBrightness`, `RandomContrast`, `Solarization`



MixUp, 2018

Dog 0.5
Cat 0.5

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j \quad (1)$$

$$\tilde{y} = \lambda y_i + (1-\lambda)y_j \quad (2)$$

- where :
  - x_i and x_j are raw input data
  - y_i and y_j are their corresponding one-hot encoded labels
  - $\lambda \in [0, 1]$
  - i and j correspond to 2 random samples

`https://arxiv.org/pdf/1710.09412`

CutMix, 2019

Dog 0.6
Cat 0.4

Algorithm A1 Pseudo-code of CutMix

```
for each iteration do
    input, target = get_minibatch(dataset)        ▷ input is N×C×W×H size tensor, target is N×K size tensor.
    if mode == training then
        input_s, target_s = shuffle_minibatch(input, target)        ▷ CutMix starts here.
        lambda = Unif(0,1)
        r_x = Unif(0,W)
        r_y = Unif(0,H)
        r_w = Sqrt(1 - lambda)
        r_h = Sqrt(1 - lambda)
        x1 = Round(Clip(r_x - r_w / 2, min=0))
        x2 = Round(Clip(r_x + r_w / 2, max=W))
        y1 = Round(Clip(r_y - r_h / 2, min=0))
        y2 = Round(Clip(r_y + r_h / 2, min=H))
        input[:, :, x1:x2, y1:y2] = input_s[:, :, x1:x2, y1:y2]
        lambda = 1 - (x2-x1)*(y2-y1)/(W*H)        ▷ Adjust lambda to the exact area ratio.
        target = lambda * target + (1 - lambda) * target_s        ▷ CutMix ends.
    end if
    output = model_forward(input)
    loss = compute_loss(output, target)
    model_update()
end for
```

https://arxiv.org/pdf/1905.04899

Data augmentation in practice

- as a keras layer (pre-processing)
  - e.g. `model.add(RandomFlip(mode="horizontal"))`
    - randomly applied at training
    - disabled at prediction
- inside the data loader
  - as a pre-processing
- inside the training loop (`pytorch`)

Implementations of MixUp and CutMix

- in keras 3
  - available as a layer in `keras_cv`
    - but only with tensorflow back-end (soon with pytorch back-end)
    - `from keras_cv.layers import CutMix, MixUp`
- in pytorch

```
from torch.utils.data import TensorDataset, DataLoader
from torchvision.transforms import v2

dataset = TensorDataset(torch.from_numpy(xTrain), torch.from_numpy(yTrain))
trainLoader = DataLoader(dataset, shuffle=True, batch_size=32)
cutmix = v2.CutMix(num_classes=3)

for images, labels in trainLoader:
    images, labels = cutmix(images, labels)
```

  - More in this example.

The training loop in pytorch

Adapted from pytorch website.

```
dataset = TensorDataset(torch.from_numpy(xTrain), torch.from_numpy(yTrain))
trainLoader = DataLoader(dataset, shuffle=True, batch_size=32)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
loss_fn = torch.nn.CrossEntropyLoss()

def train_one_epoch(epoch_index):
    for batch in trainLoader:
        # Every data instance is an input + label pair
        images, labels = batch

        # Zero your gradients for every batch!
        optimizer.zero_grad()
        # Make predictions for this batch
        outputs = model(images)
        # Compute the loss and its gradients
        loss = loss_fn(outputs, labels)
        loss.backward()
        # Adjust learning weights
        optimizer.step()
        #[...]
```

Many other methods

- Implementation of a new Augmentation method in keras 3 :
  - using `keras_cv.layers.BaseImageAugmentationLayer`
- Generative models
  - VAEs, GAN, transformers
- Adversarial data
  - help the robustness of neural networks

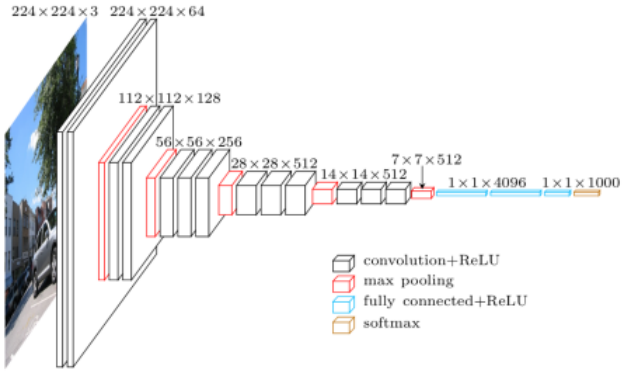
Outline

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**.