INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Deep Learning

Introduction à la reconnaissance d'image

Brendan Guillouet

2018

Institut National des Sciences Appliquées

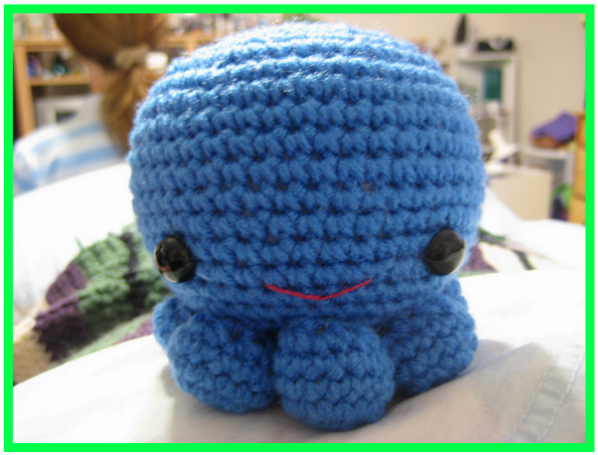
Plan de Présentation

Introduction

Application - Classification simple

Your specialized huggable recommendation

Go for it! Hug it out. With a score of **0.695779**, we're somewhat sure.



Your specialized huggable recommendation

Don't hug that. Please. With a score of **0.999875**, we're pretty sure.

Your specialized huggable recommendation

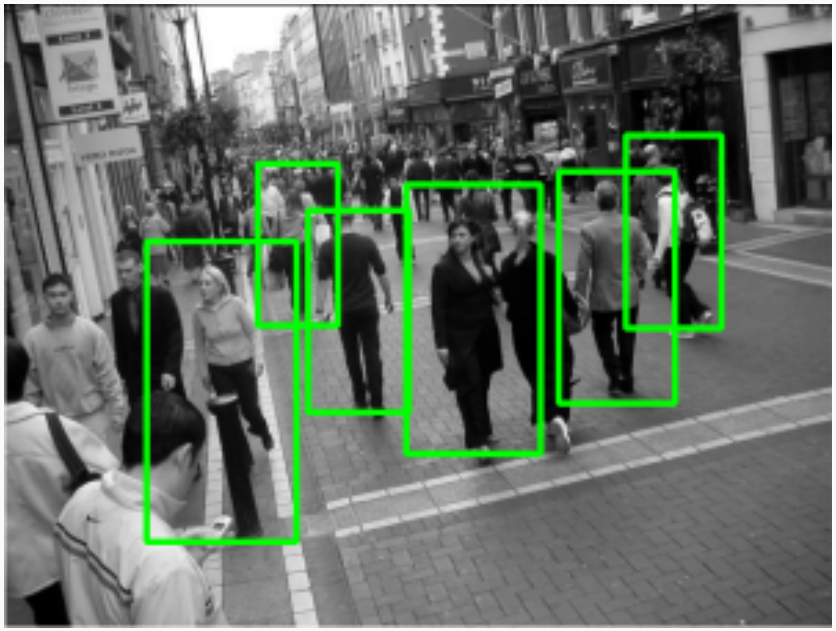
Don't hug that. Please. With a score of **0.778686**, we're pretty sure.

Your specialized huggable recommendation

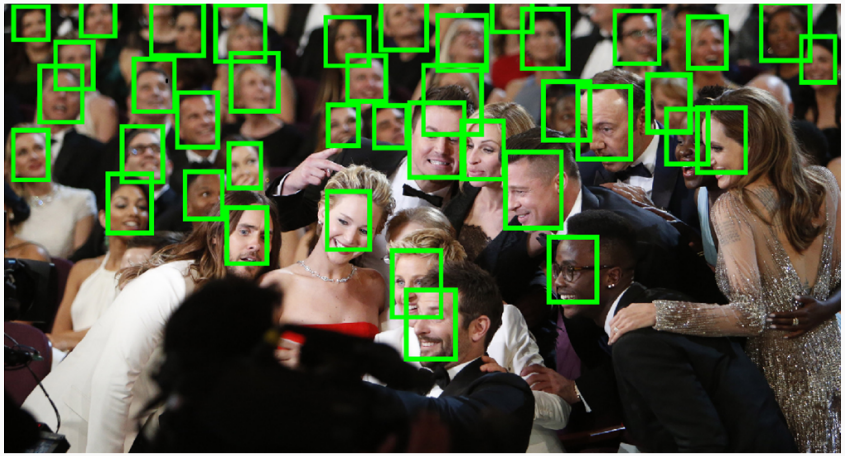
Go for it! Hug it out. With a score of **0.958104**, we're pretty sure.

Application - Reconnaissance de patterns

Caméra de sécurité



Facebook



Dense (MLP) network

- Number of parameters increases exponentially
  - ex :Image from défi IA-2019, dim :(128,128,3), features : 49.152

```
#Model
model = km.Sequential()
model.add(kl.Dense(512, activation='relu', input_shape=(49152,)))
model.summary()
#OUTPUT
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 512)               25166336
=================================================================
Total params: 25,166,336
Trainable params: 25,166,336
Non-trainable params: 0
_________________________________________________________________
```

- Spatial information is lost with *flatten*.
- Huge number of parameters : 25.166.336

Dense (MLP) network

- Number of parameters increases exponentially
  - ex :Image from défi IA-2019, dim :(128,128,3), features : 49.152

```
#Model
model = km.Sequential()
model.add(kl.Dense(512, activation='relu', input_shape=(49152,)))
model.summary()
#OUTPUT
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 512)               25166336
=================================================================
Total params: 25,166,336
Trainable params: 25,166,336
Non-trainable params: 0
_________________________________________________________________
```

- Spatial information is lost with *flatten*.
- Huge number of parameters : 25.166.336

$\Longrightarrow$ Convolutional network.

Convolutional Neural Network

Convolution - How it works?

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

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

4 X 4

Convolution - How it works?

3_{1}	0_{0}	1_{-1}	2	7	4
1_{1}	5_{0}	8_{-1}	9	3	1
2_{1}	7_{0}	2_{-1}	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

4 X 4

Convolution - How it works?

3_{1}	0_{0}	1_{-1}	2	7	4
1_{1}	5_{0}	8_{-1}	9	3	1
2_{1}	7_{0}	2_{-1}	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

4 X 4

Convolution - How it works?

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 1 + 5 \times 1 + 7 \times 1 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

3_{1}	0_{0}	1_{-1}	2	7	4
1_{1}	5_{0}	8_{-1}	9	3	1
2_{1}	7_{0}	2_{-1}	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

-5			

4 X 4

Convolution - How it works?

3	0_{1}	1_{0}	2_{-1}	7	4
1	5_{1}	8_{0}	9_{-1}	3	1
2	7_{1}	2_{0}	5_{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

-5			

4 X 4

Convolution - How it works?

3	0_{1}	1_{0}	2_{-1}	7	4
1	5_{1}	8_{0}	9_{-1}	3	1
2	7_{1}	2_{0}	5_{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

-5			

4 X 4

Convolution - How it works?

3	0_{1}	1_{0}	2_{-1}	7	4
1	5_{1}	8_{0}	9_{-1}	3	1
2	7_{1}	2_{0}	5_{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

-5	-4		

4 X 4

Convolution - How it works?

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

6 X 6

$\star$

Convolution Filter

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

3 X 3

$=$

-5	-4	0	8
-10	-2	2	3
0	-2	-4	7
-3	-2	-3	16

4 X 4

Edge detection

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

6 X 6

$\star$

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

Convolution
Filter
3 X 3

$=$

0	30	30	0
0	10	10	0
0	-10	-10	0
0	-30	-30	0

4 X 4

Edge detection

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

6 X 6

$\star$

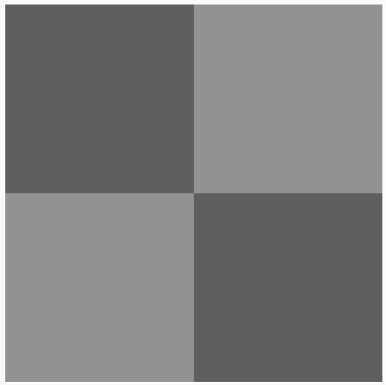
1	0	-1
1	0	-1
1	0	-1

Convolution
Filter
3 X 3

$=$

0	30	30	0
0	10	10	0
0	-10	-10	0
0	-30	-30	0

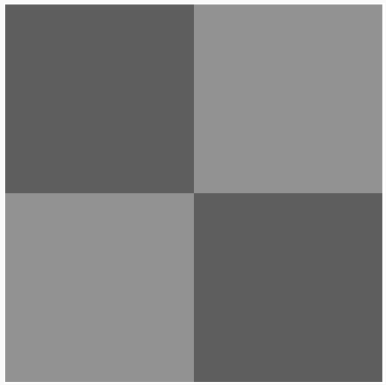
4 X 4



Edge detection

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

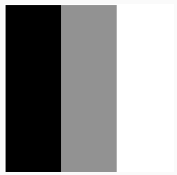
6 X 6



$\star$

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

Convolution
Filter
3 X 3



$=$

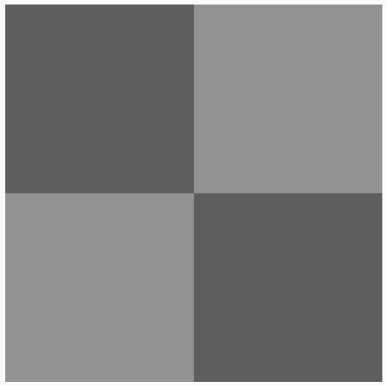
0	30	30	0
0	10	10	0
0	-10	-10	0
0	-30	-30	0

4 X 4

Edge detection

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

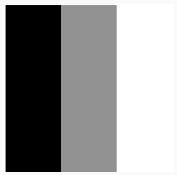
6 X 6



$\star$

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

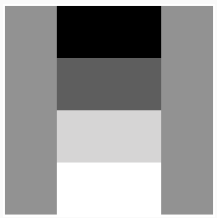
Convolution
Filter
3 X 3



$=$

0	30	30	0
0	10	10	0
0	-10	-10	0
0	-30	-30	0

4 X 4



Convolution

Discrete 2D-convolution :

$$(f \star g)(x, y) = \sum_m \sum_n f(n, m) \cdot g(x - m, y - n)$$

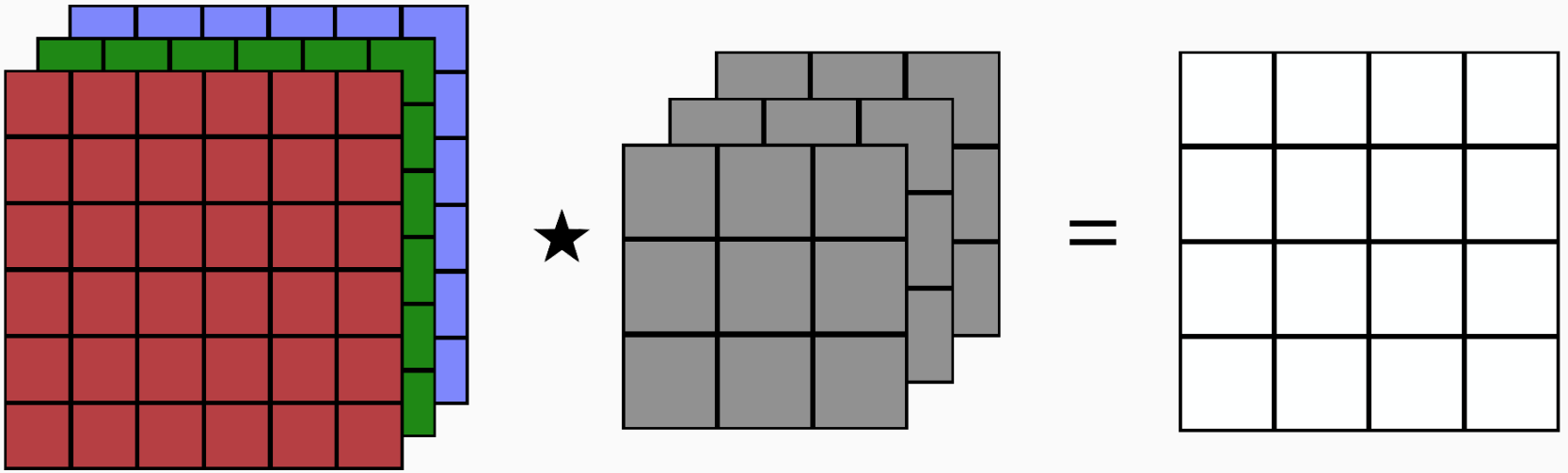
Discrete 2D-Cross-correlation (often used !) :

$$(f \star g)(x, y) = \sum_m \sum_n f(n, m) \cdot g(x + m, y + n)$$

- Take spatial organisation into account
- translation invariant,
- *parameter sharing*,
- *local connectivity*.

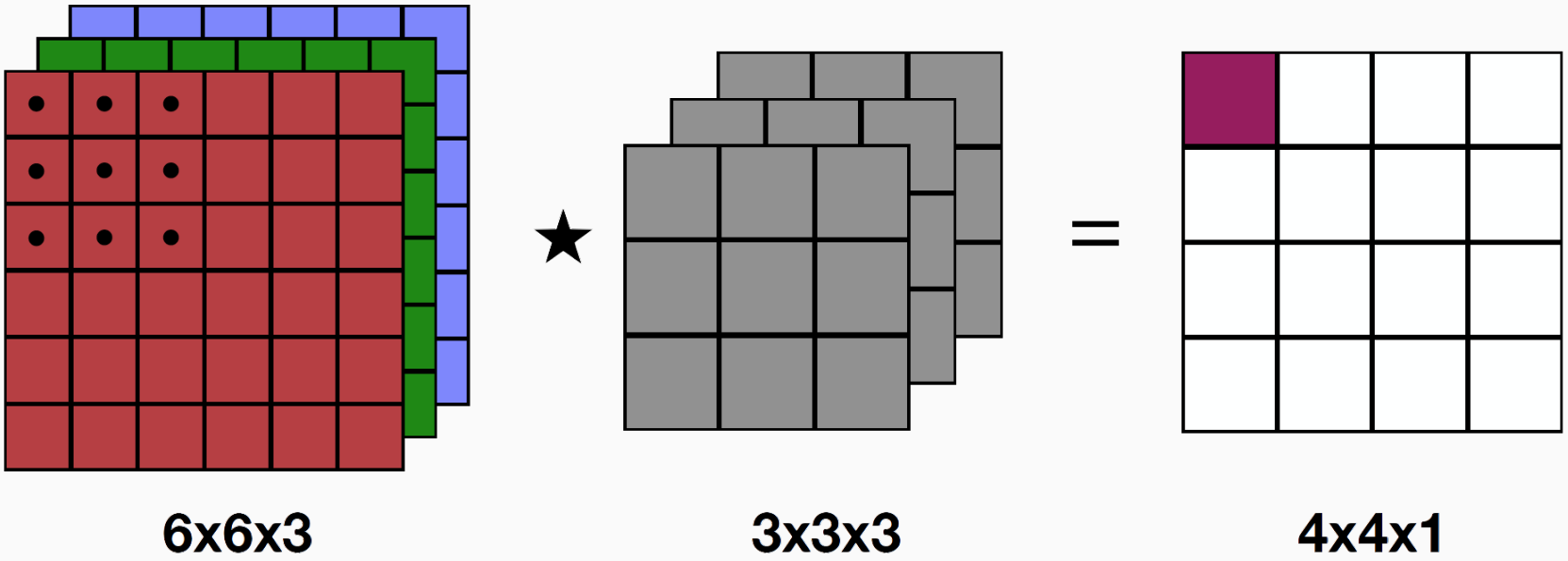
Convolution - channels

Colored image in 3 dimensions : (height, width, channels (RGB))



Convolution - channels

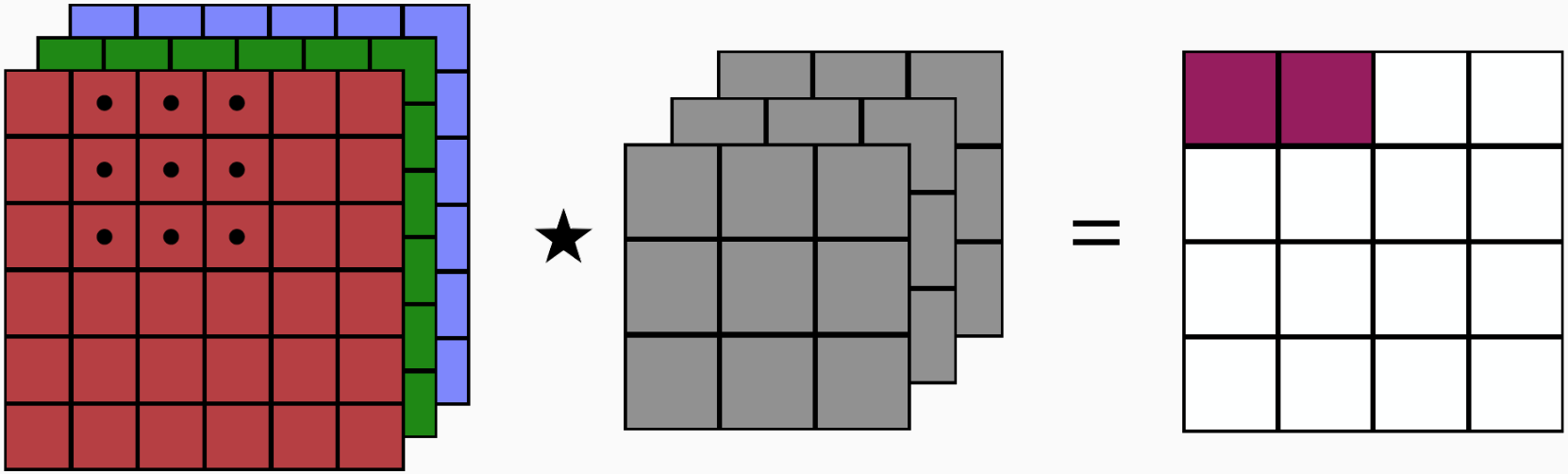
Colored image in 3 dimensions : (height, width, channels (RGB))



$$(F \star img)(x, y) = \sum_{c} \sum_{m} \sum_{n} F^{c}(n, m) \cdot Img^{c}(x + m, y + n)$$

Convolution - channels

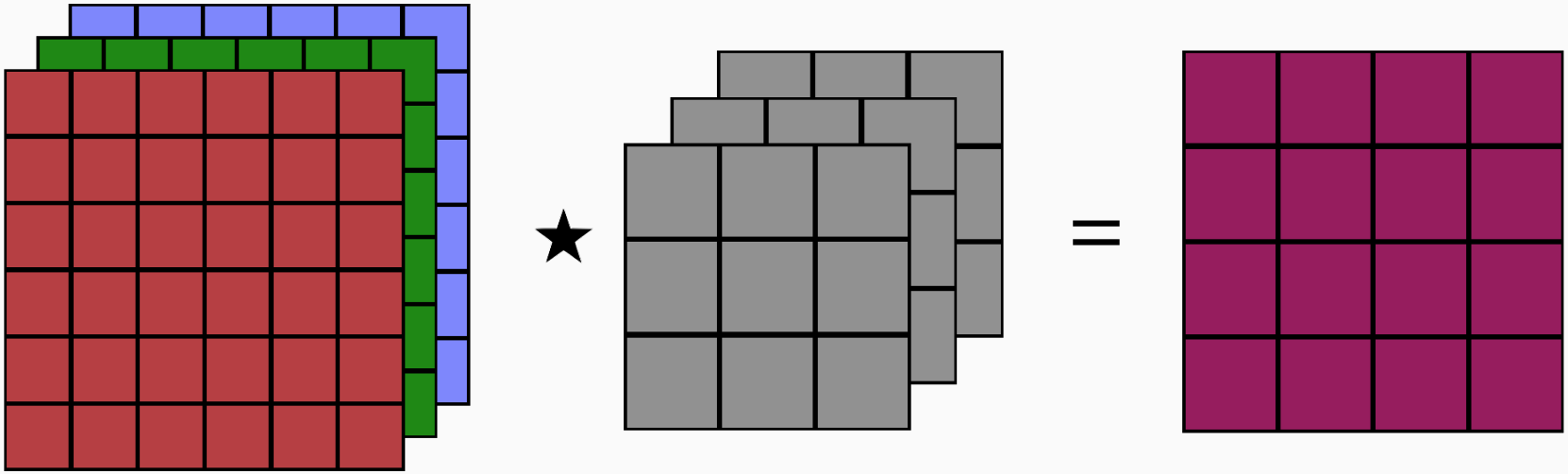
Colored image in 3 dimensions : (height, width, channels (RGB))



$$(F \star img)(x, y) = \sum_{c} \sum_{m} \sum_{n} F^{c}(n, m) \cdot Img^{c}(x + m, y + n)$$

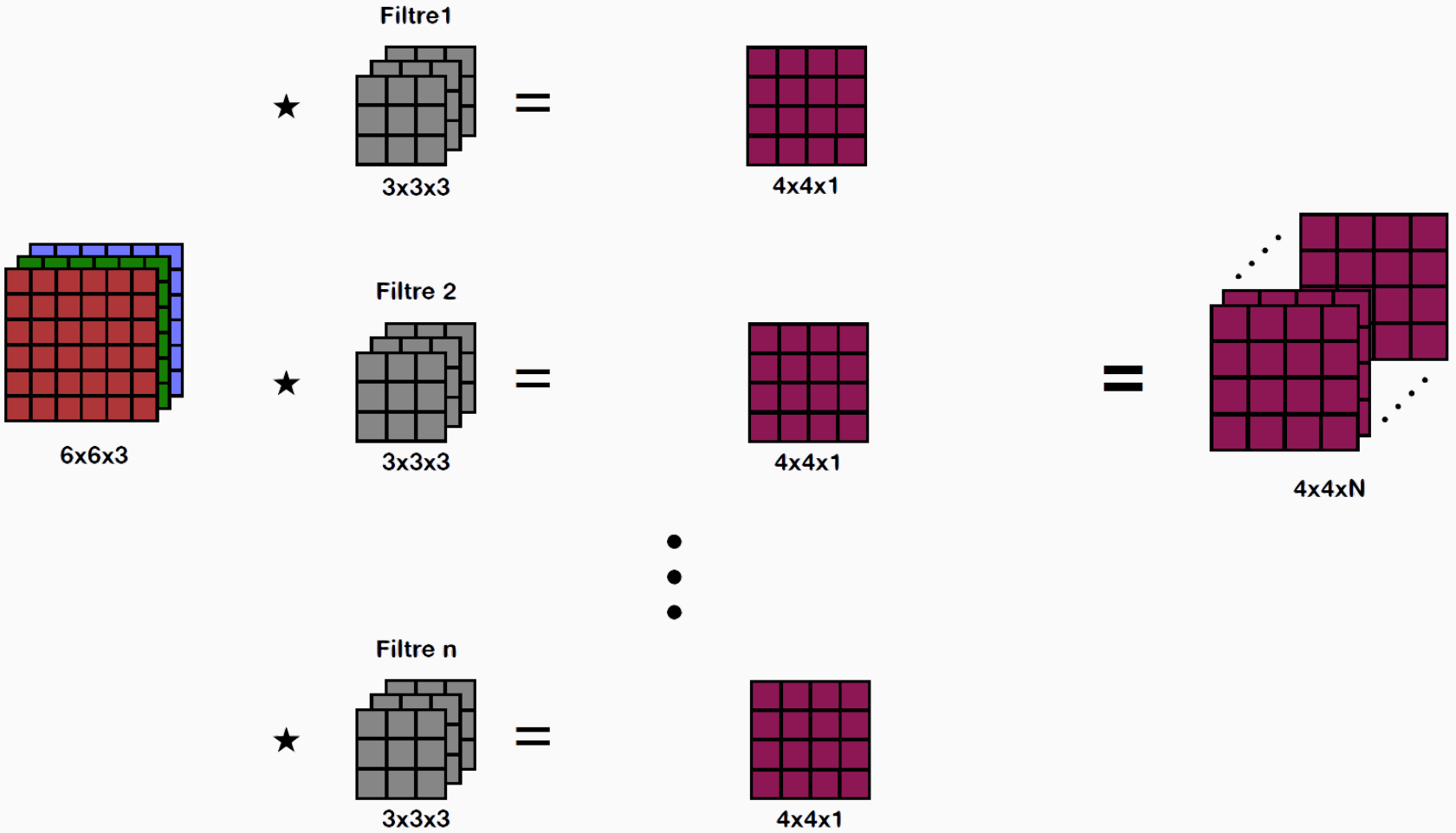
Convolution - channels

Colored image in 3 dimensions : (height, width, channels (RGB))



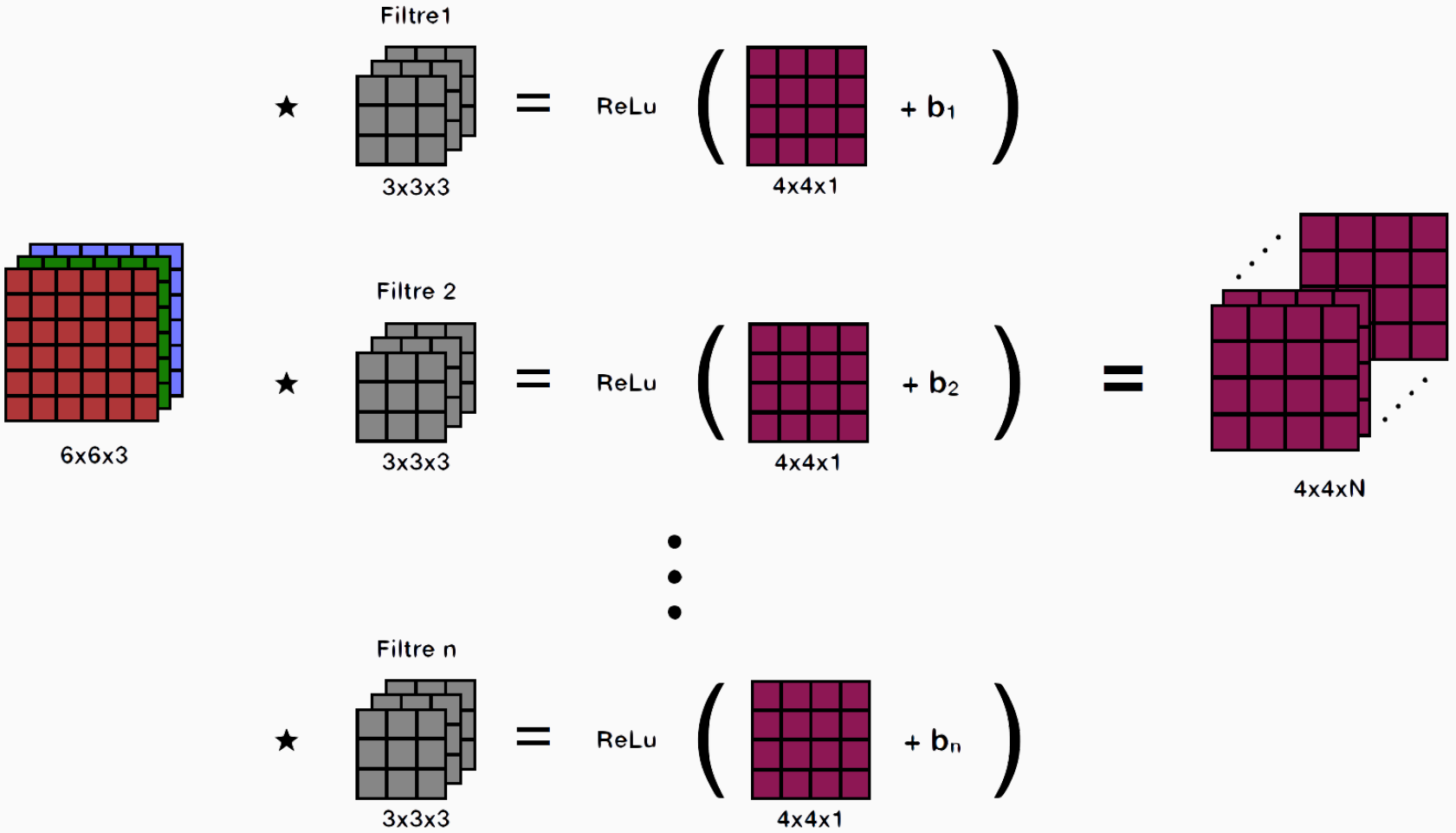
$$(F \star img)(x, y) = \sum_{c} \sum_{m} \sum_{n} F^{c}(n, m) \cdot Img^{c}(x + m, y + n)$$

Convolution - Layer



- Input Dimension : (h, w, c_i)
- Filter Dimension : (f, f), Nb filter : N
- Output Dimension : $(h - f + 1, w - f + 1, N)$

Convolution - Layer



- Input Dimension : (h, w, c_i)
- Filter Dimension : (f, f), Nb filter : N
- Output Dimension : $(h - f + 1, w - f + 1, N)$

Convolution - Dimensions and parameters

- ex :Image from défi IA-2019, dim :(128,128,3), parameters : 49.152

```
#Model
model = km.Sequential()
model.add(kl.Conv2D(512, kernel_size=(3,3), activation='relu', input_shape=(128,128,3)))
model.summary()
#OUTPUT
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 512)     14336
=================================================================
Total params: 14,336
Trainable params: 14,336
Non-trainable params: 0
_________________________________________________________________
```

- Number of parameters : 14.336$(= (27 + 1) * 512)$ ...
- ...VS $25,166,336$ for *Dense* layer.

Strides

Stride = 2

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

7 X 7 X 1

$\star$

3	4	4
1	0	2
-1	0	3

3 X 3 X 1

$=$

3 x 3 X 1

Strides

Stride = 2

2_3	3_4	7_4	4	4	2	9
6_1	6_0	9_2	8	7	4	3
3_{-1}	4_0	8_3	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7 X 7 X 1

$\star$

3	4	4
1	0	2
-1	0	3

3 X 3 X 1

$=$

91		

3 x 3 X 1

Strides

Stride = 2

2	3	7_3	4_4	4_4	2	9
6	6	9_1	8_0	7_2	4	3
3	4	8_{-1}	3_0	8_3	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7 X 7 X 1

$\star$

3	4	4
1	0	2
-1	0	3

3 X 3 X 1

$=$

91	100	

3 x 3 X 1

Strides

Stride = 2

2	3	7	4	4	2	9
6	6	9	8	7	4	3
3_3	4_4	8_4	3	8	9	7
7_1	8_0	3_2	6	6	3	4
4_{-1}	2_0	1_3	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7 X 7 X 1

$\star$

3	4	4
1	0	2
-1	0	3

3 X 3 X 1

$=$

91	100	83
69		

3 x 3 X 1

Strides

Stride = 2

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

7 X 7 X 1

$\star$

3	4	4
1	0	2
-1	0	3

3 X 3 X 1

$=$

91	100	83
69	91	127
44	71	74

3 x 3 X 1

Padding

Padding = 1

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

6 X 6

$\star$

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

3 X 3

$=$

-5	-4	0	8
-10	-2	2	3
0	-2	-4	7
-3	-2	-3	16

4 x 4

Padding

Padding = 1

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

8 X 8

$\star$

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

3 X 3

$=$

-5	-4	0	8
-10	-2	2	3
0	-2	-4	7
-3	-2	-3	16

4 x 4

Padding

Padding = 1

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

8 X 8

$\star$

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

3 X 3

$=$

-5	-5	-6	-1	6	10
-12	-5	-4	0	8	11
-13	-10	-1	2	3	11
-10	0	-2	-4	7	10
-7	-3	-2	-3	16	12
-6	0	-2	1	-9	5

6 x 6

Padding

Padding = 1

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

8 X 8

$\star$

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

3 X 3

$=$

-5	-5	-6	-1	6	10
-12	-5	-4	0	8	11
-13	-10	-1	2	3	11
-10	0	-2	-4	7	10
-7	-3	-2	-3	16	12
-6	0	-2	1	-9	5

6 x 6

- Input and output image dimensions are the same.
- Border usually fill with zero.
- In practice :
  - VALID = No padding
  - SAME = Pad so that input and output sizes are the same.

Dimensions formula - Convolutional Layer

Input Dimensions : $(h \times w \times c)$

- h : height of the image,
- w : width of the image,
- c : channel of the image.

Convolution parameter : (f, p, s)

- f : filter size
- p : padding size
- s : stride size
- n : number of layer

Output Dimensions :

$$\left(\left[\frac{h + 2p - f}{s} + 1 \right] \times \left[\frac{w + 2p - f}{s} + 1 \right] \times n \right)$$

Number of parameters : $N_p = \big((h \cdot w \cdot c) + 1\big) \cdot n$

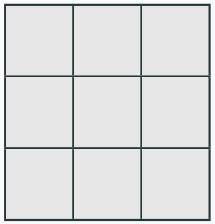
Pooling

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

6 X 6

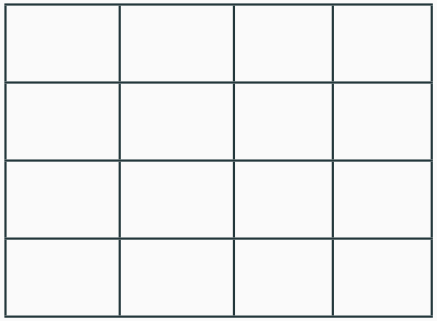
Convolution
Filter

Pool



3 X 3

=



4 X 4

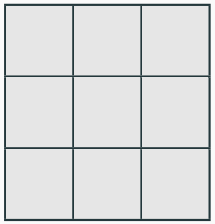
Pooling

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

6 X 6

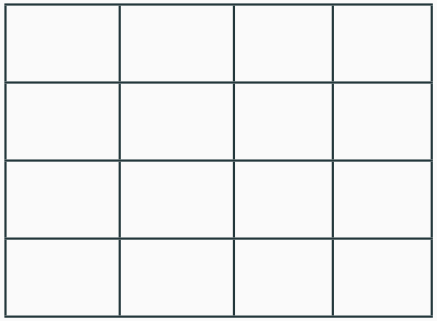
Convolution
Filter

Pool



3 X 3

=



4 X 4

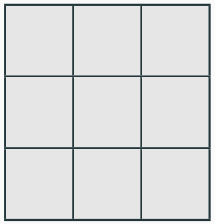
Pooling

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

6 X 6

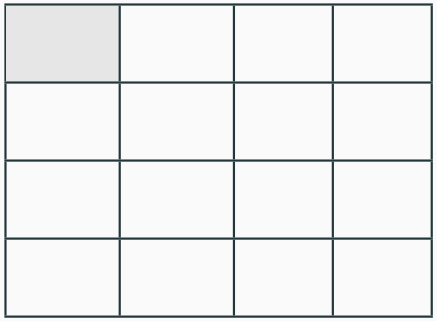
Convolution
Filter

Pool



3 X 3

=



4 X 4

Pooling

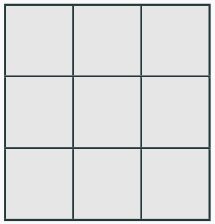
$$max\,[3, 1, 2, 0, 5, 7, 1, 8, 2] = 8$$

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

6 X 6

Convolution
Filter

Pool



3 X 3

=

8			

4 X 4

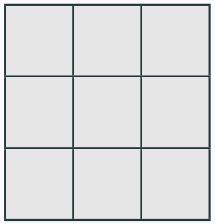
Pooling

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

6 X 6

Convolution
Filter

Pool



3 X 3

=

8			

4 X 4

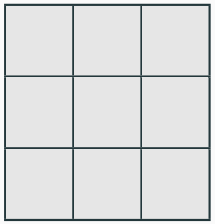
Pooling

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

6 X 6

Convolution
Filter

Pool



3 X 3

=

8			

4 X 4

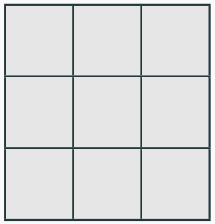
Pooling

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

6 X 6

Convolution
Filter

Pool



3 X 3

=

8	9		

4 X 4

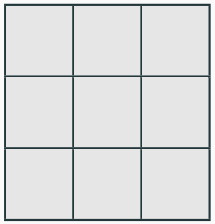
Pooling

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

6 X 6

Convolution
Filter

Pool



3 X 3

=

8	9	9	9
8	9	9	9
7	7	7	8
5	6	7	8

4 X 4

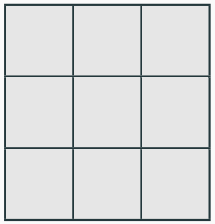
Pooling

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

6 X 6

Convolution Filter

Pool



3 X 3

=

8	9	9	9
8	9	9	9
7	7	7	8
5	6	7	8

4 X 4

Different pooling filter : Max Pooling, Average Pooling...

Dimensions formula - Pooling Layer

Input Dimensions : $(h \times w \times c)$

- h : height of the image,
- w : width of the image,
- c : channel of the image.

Pooling parameter : (f, s) (No padding!)

- f : filter size
- s : stride size

Output Dimensions :

$$\left(\left[\frac{h-f}{s} + 1 \right] \times \left[\frac{w-f}{s} + 1 \right] \times c \right)$$

Number of parameters : No parameters to estimate!

Dimensions formula - Pooling Layer

Input Dimensions : $(h \times w \times c)$

- h : height of the image,
- w : width of the image,
- c : channel of the image.

Pooling parameter : (f, s) (No padding!)

- f : filter size
- s : stride size

Output Dimensions :

$$\left(\left[\frac{h-f}{s} + 1 \right] \times \left[\frac{w-f}{s} + 1 \right] \times c \right)$$

Number of parameters : No parameters to estimate!

But, lot of information is lost.

"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.", G. Hinton

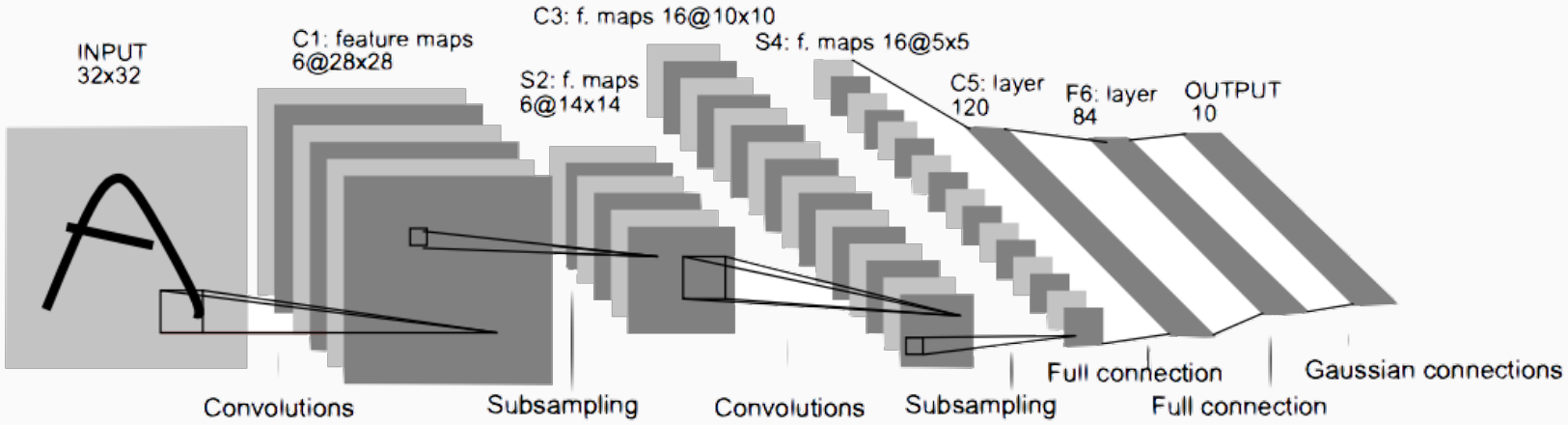
Architecture

LeNet-5 and Mnist Classification

Objectif :

- Manuscrit number from 0 to 9 hand-written
- Dimension (32,32,1)
- Image for training : $N_{train} = 60.000$
- Image for testing : $N_{test} = 10.000$

LeNet-5 Architecture : LeCun et al. [1998]



Size and height decrease while channel increase.

LeNet-5 - Keras

```
LeNet5model = km.Sequential()
#Conv Layer 1
LeNet5model.add(kl.Conv2D(filters = 6,  kernel_size = 5, strides = 1,  activation = 'tanh',
input_shape = (32,32,1)))
#Pooling layer 1
LeNet5model.add(kl.MaxPooling2D(pool_size = 2, strides = 2))
#Conv Layer 2
LeNet5model.add(kl.Conv2D(filters = 16, kernel_size = 5,strides = 1, activation = 'tanh'))
#Pooling Layer 2
LeNet5model.add(kl.MaxPooling2D(pool_size = 2, strides = 2))
#Flatten
LeNet5model.add(kl.Flatten())
#Fully connected layer 1
LeNet5model.add(kl.Dense(units = 120, activation = 'tanh'))
#Fully connected layer 2
LeNet5model.add(kl.Dense(units = 84, activation = 'tanh'))
#Output Layer
LeNet5model.add(kl.Dense(units = 10, activation = 'softmax'))

Trainable params: 61,706
```

- **Hyperbolic tangent** activation for all but last layer.
- **Softmax** activation for last layer

$$softmax(X) = \frac{e^{x_i}}{\sum_{j \in 10} e^{x_j}}$$

Loss function - Classification

```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy')
model.fit(X_train ,Y_train, steps_per_epoch = 10, epochs = 42)
```

Binary Cross Entropy :

$$\frac{\text{-1}}{N_{train}}\mathcal{L}(y_i, \tilde{y}_i) = \frac{\text{-1}}{N_{train}} \sum_{i=1} N_{train} \left[y_i \log \tilde{y}_i + (1 - y_i) \log(1 - \tilde{y}_i) \right]$$

where $y_i \in [0, 1]$, $\tilde{y}_i = P(X_i = 1)$

Categorical Cross Entropy :

$$\frac{\text{-1}}{N_{train}}\mathcal{L}(y_i, \tilde{y}_i) = \frac{\text{-1}}{N_{train}} \sum_{i=1}^{N_{train}} \sum_{c=1}^{10} 1_{[y_i == c]} \log \tilde{y}_i^c$$

where $y_i \in [0, \ldots, 10]$, $\tilde{y}_i = \left[\tilde{y}_i^0, \ldots, \tilde{y}_i^{10} \right]$ = output of softmax layer

Alex Net (Krizhevsky et al. [2012])

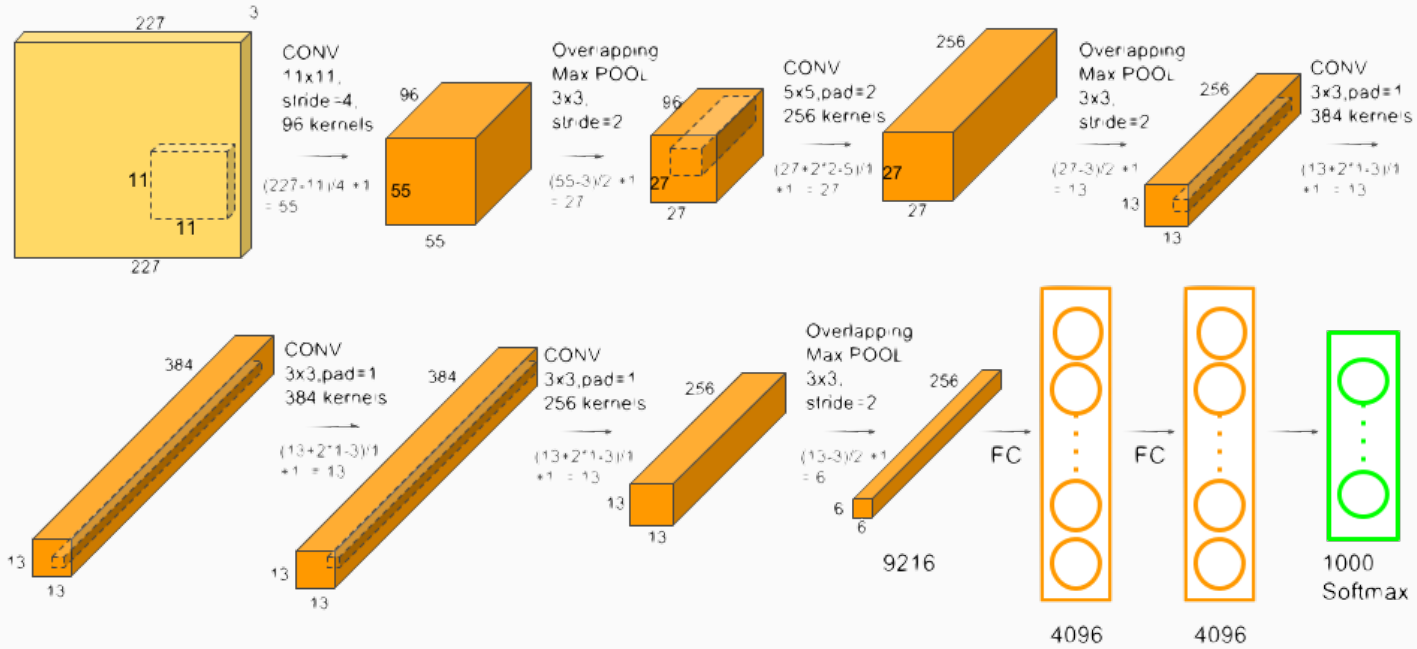


Image source : https ://www.learnopencv.com/understanding-alexnet/

- $\simeq 60M$ parameters
- Similar to LeNet-5 but Much bigger
- Only ReLu is different.
- Learned on huge amount of data : **ImageNet** .

Image net

http ://image-net.org/

- 1000 classes
  - 1,2 M training images,
  - 100k test images.

VGG16 - (simonyan2014very)

Use always same layer :

- Convolution : 3×3 filter, stride = 1, padding= SAME
- Max Pooling : 2×2 filter, stride = 2

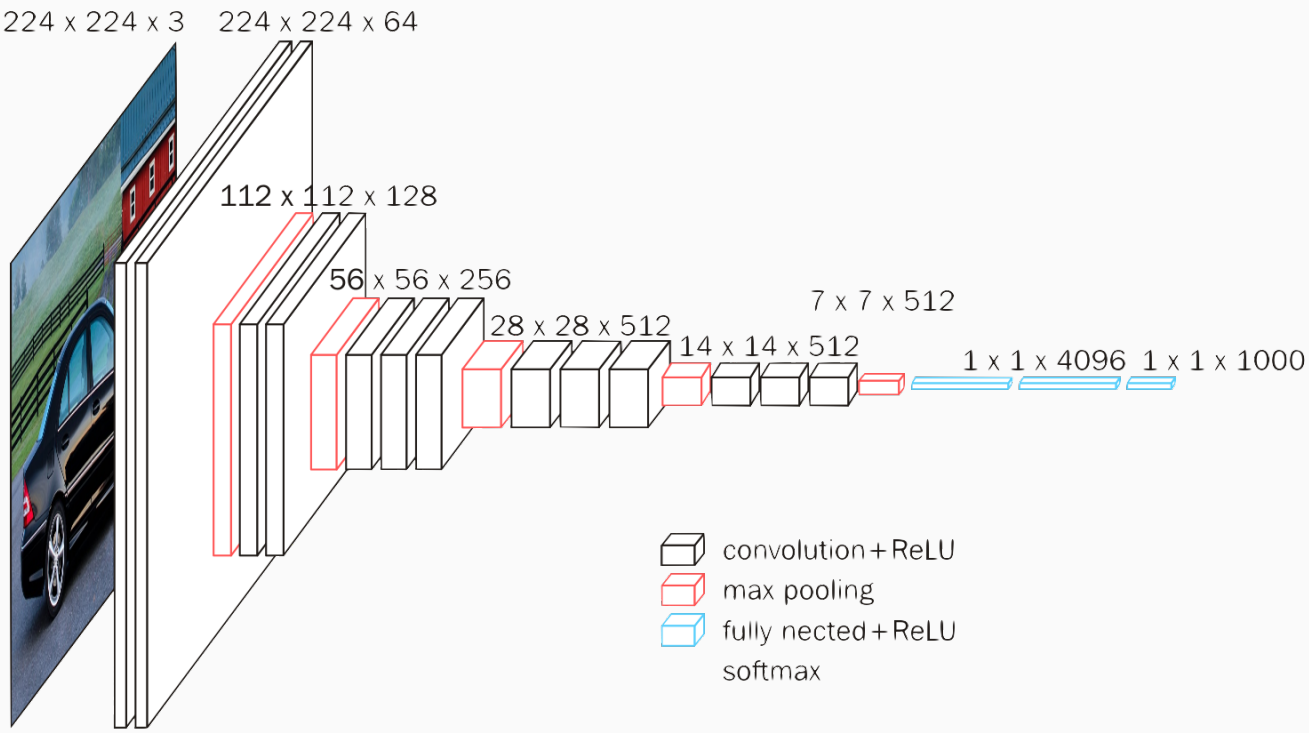


Image source : https ://blog.datawow.io/cnn-models-ef356bc11032, realized with Tensorflow.

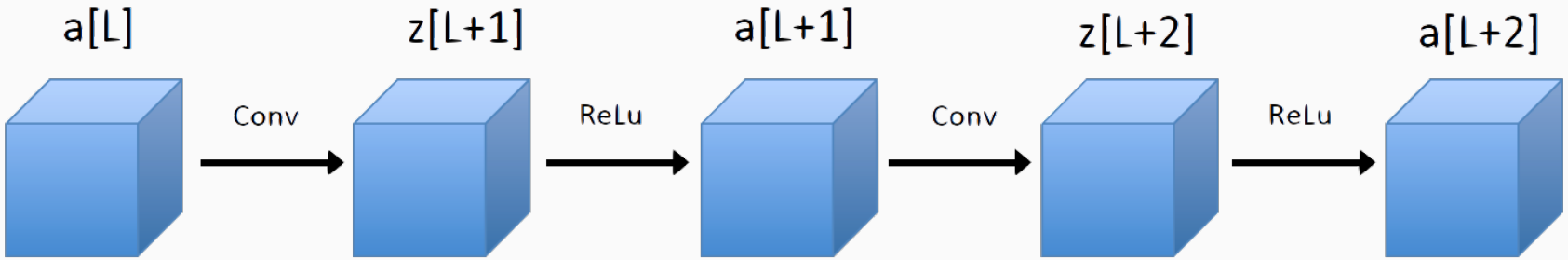
138M parameters.

HIERARCHICAL REPRESENTATION

https ://cs.nyu.edu/ fergus/papers/zeilerECCV2014.pdf

ResNet Block - (He et al. [2016])

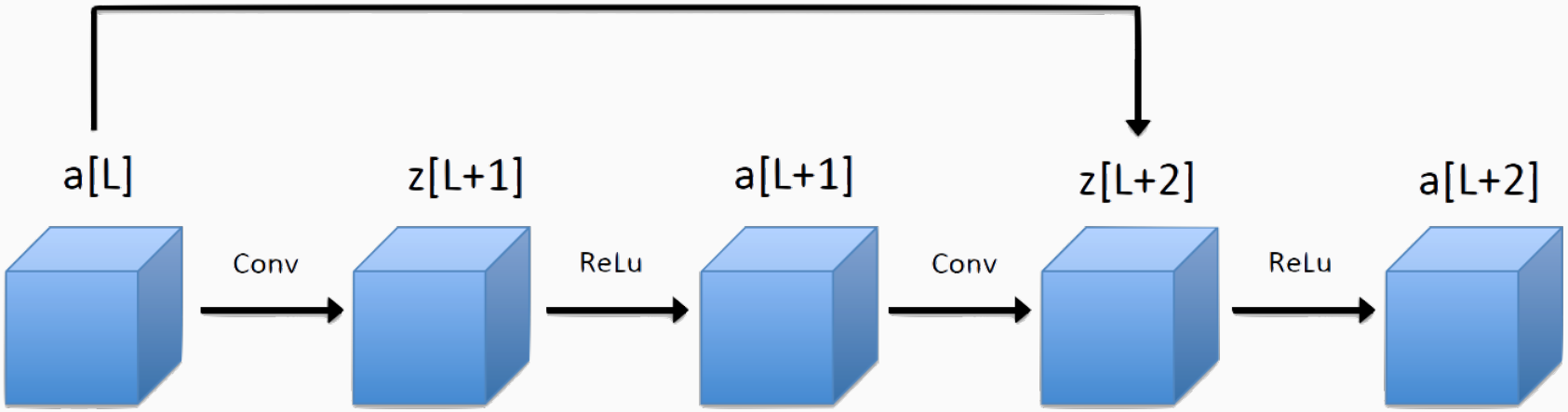
Convolution with padding = SAME



$$z[l+1] = Conv(a[l])$$
$$a[l+1] = ReLu(z[l+1])$$
$$z[l+2] = Conv(a[l+1])$$
$$a[l+2] = ReLu(z[l+2] \qquad)$$

ResNet Block - (He et al. [2016])

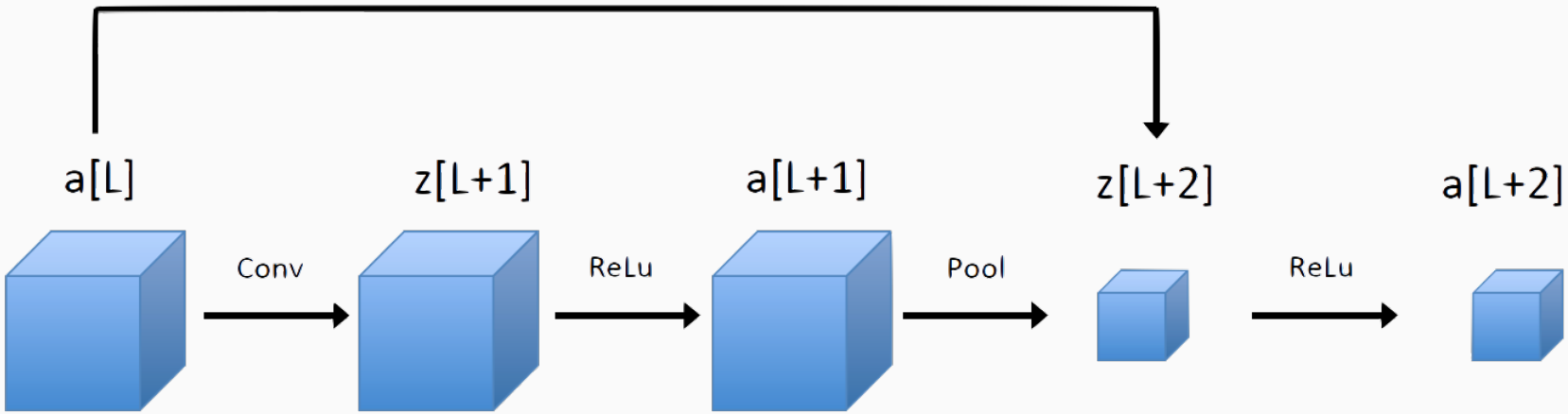
Convolution with padding = SAME



$$z[l+1] = Conv(a[l])$$
$$a[l+1] = ReLu(z[l+1])$$
$$z[l+2] = Conv(a[l+1])$$
$$a[l+2] = ReLu(z[l+2] + \quad a[l])$$

ResNet Block - (He et al. [2016])

MaxPooling



$$z[l+1] = Conv(a[l])$$
$$a[l+1] = ReLu(z[l+1])$$
$$z[l+2] = Conv(a[l+1])$$
$$a[l+2] = ReLu(z[l+2] + W \cdot a[l])$$

ResNet Block - (He et al. [2016]) - Properties

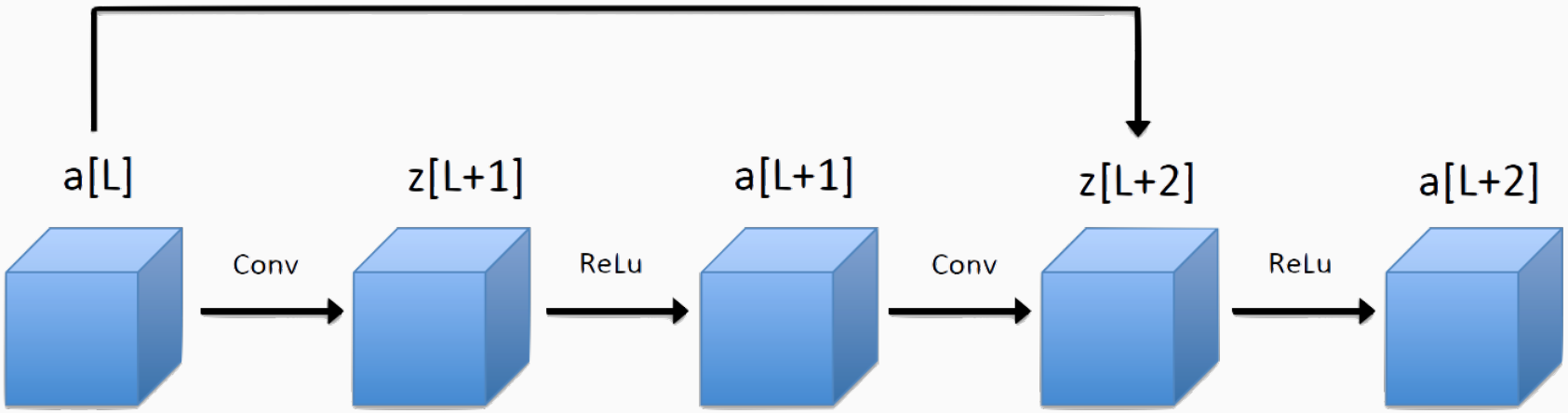
Making a convolutional network to deep can hurt its performance.

ResNet Block - (He et al. [2016]) - Properties

Making a convolutional network to deep can hurt its performance.

With Resnet, you can make your network deeper !!

Convolution with padding = SAME

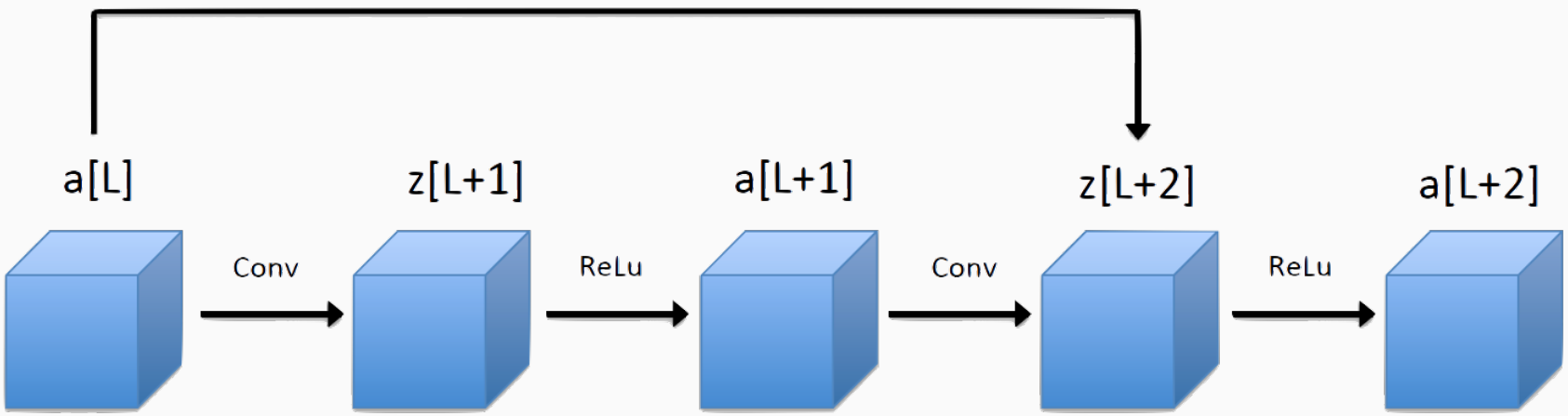


ResNet Block - (He et al. [2016]) - Properties

Making a convolutional network to deep can hurt its performance.

With Resnet, you can make your network deeper !!

Convolution with padding = SAME



$$z[l+1] = Conv(a[l])$$
$$a[l+1] = ReLu(z[l+1])$$
$$z[l+2] = Conv(a[l+1])$$
$$a[l+2] = ReLu(z[l+2] + a[l])$$

- Without hurting its performance..
- ... and with a bit of luck, improving it !

ResNet He et al. [2016]

Convolution with 3x3 filters and padding = SAME

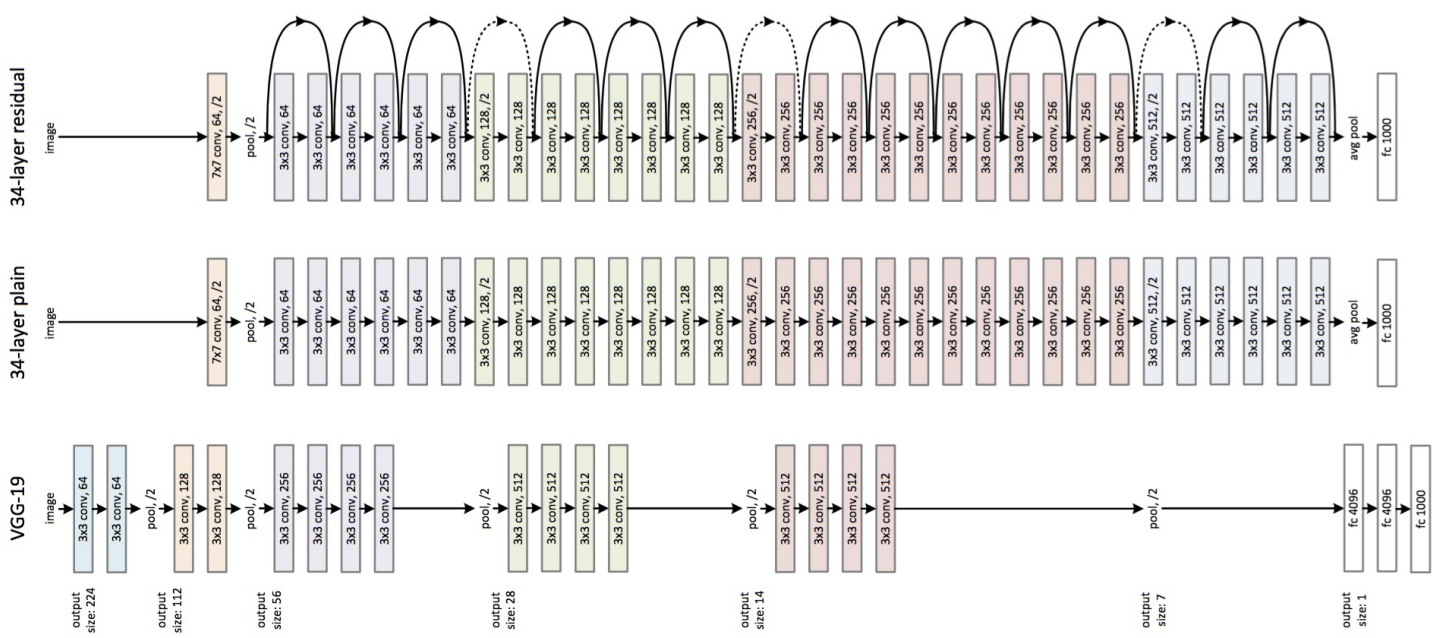
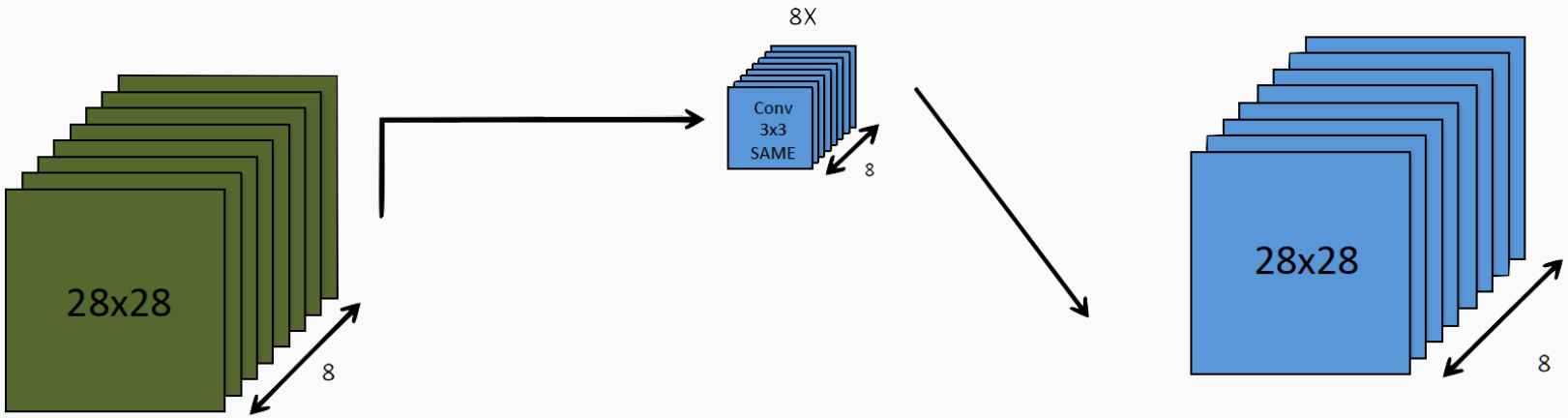
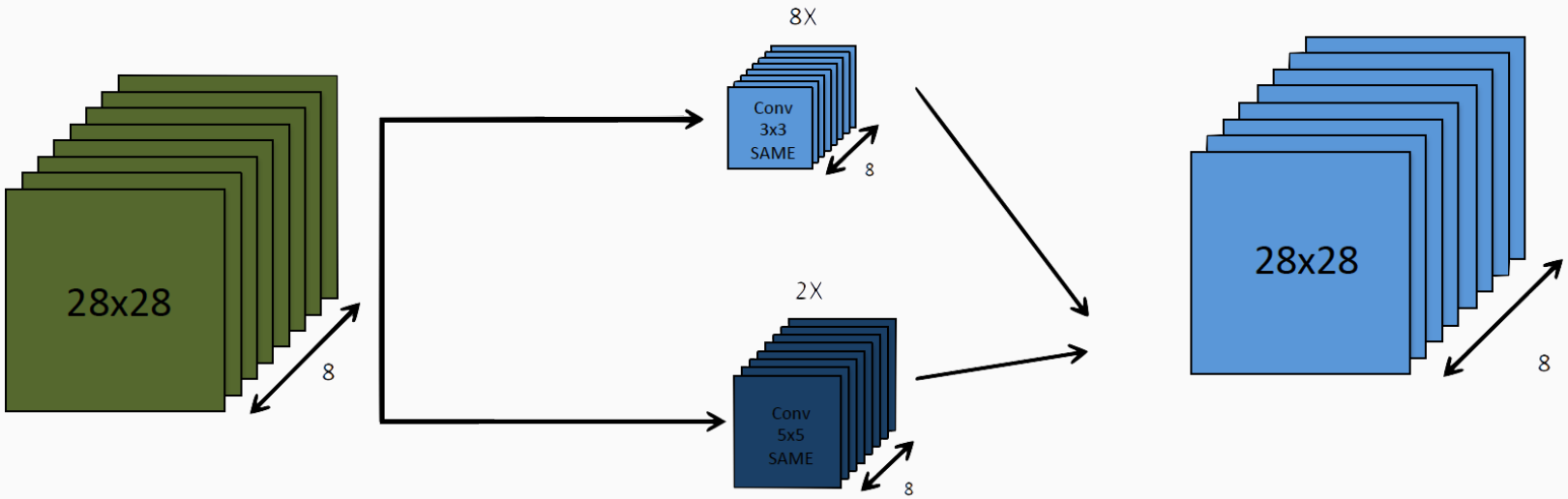


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

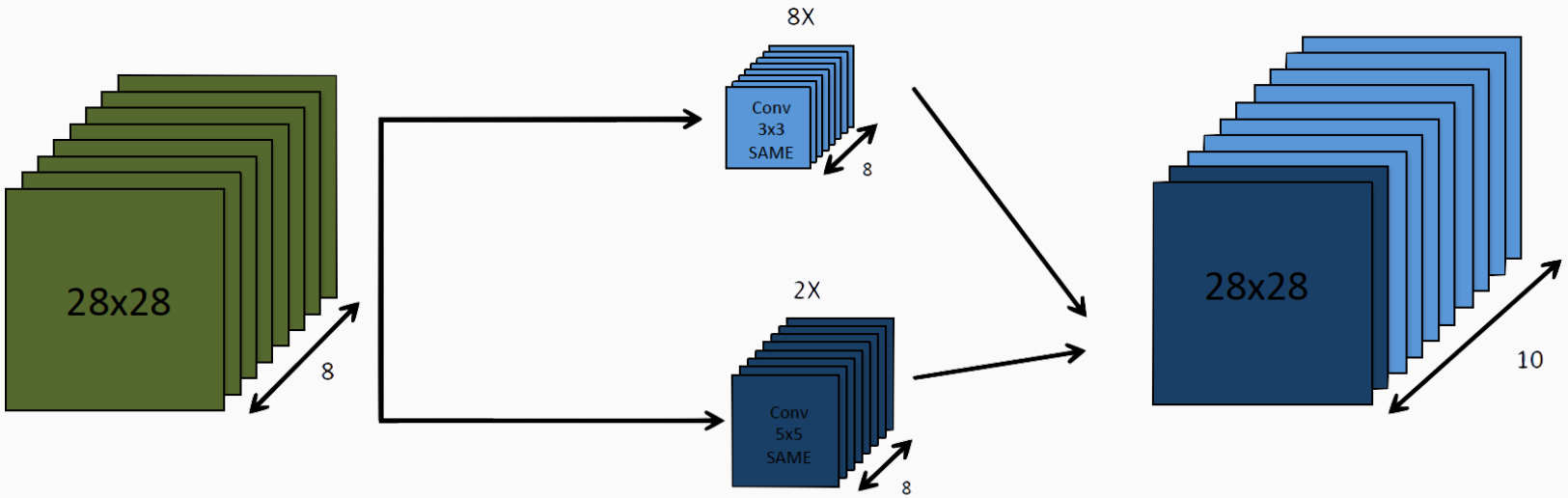
Inception Block - Szegedy et al. [2015]



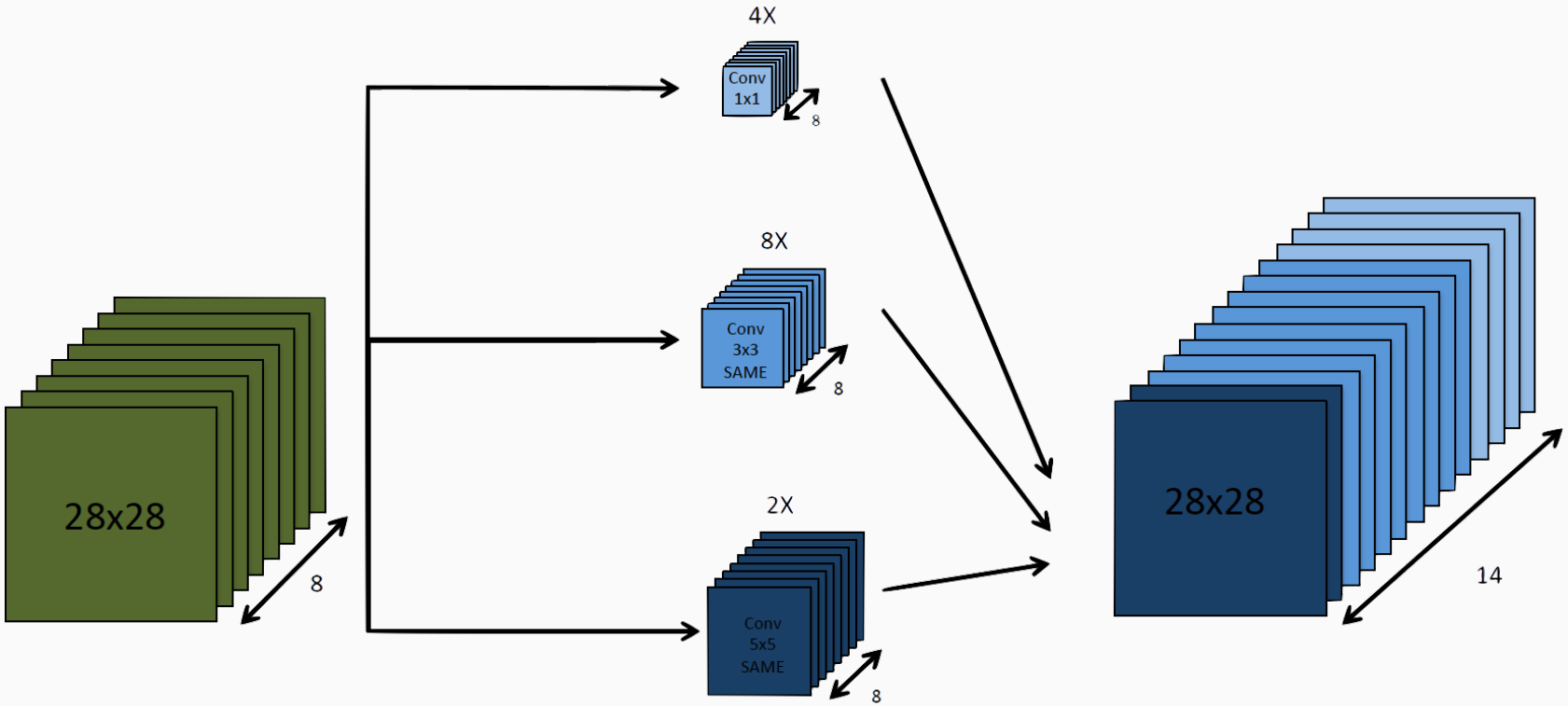
Inception Block - Szegedy et al. [2015]



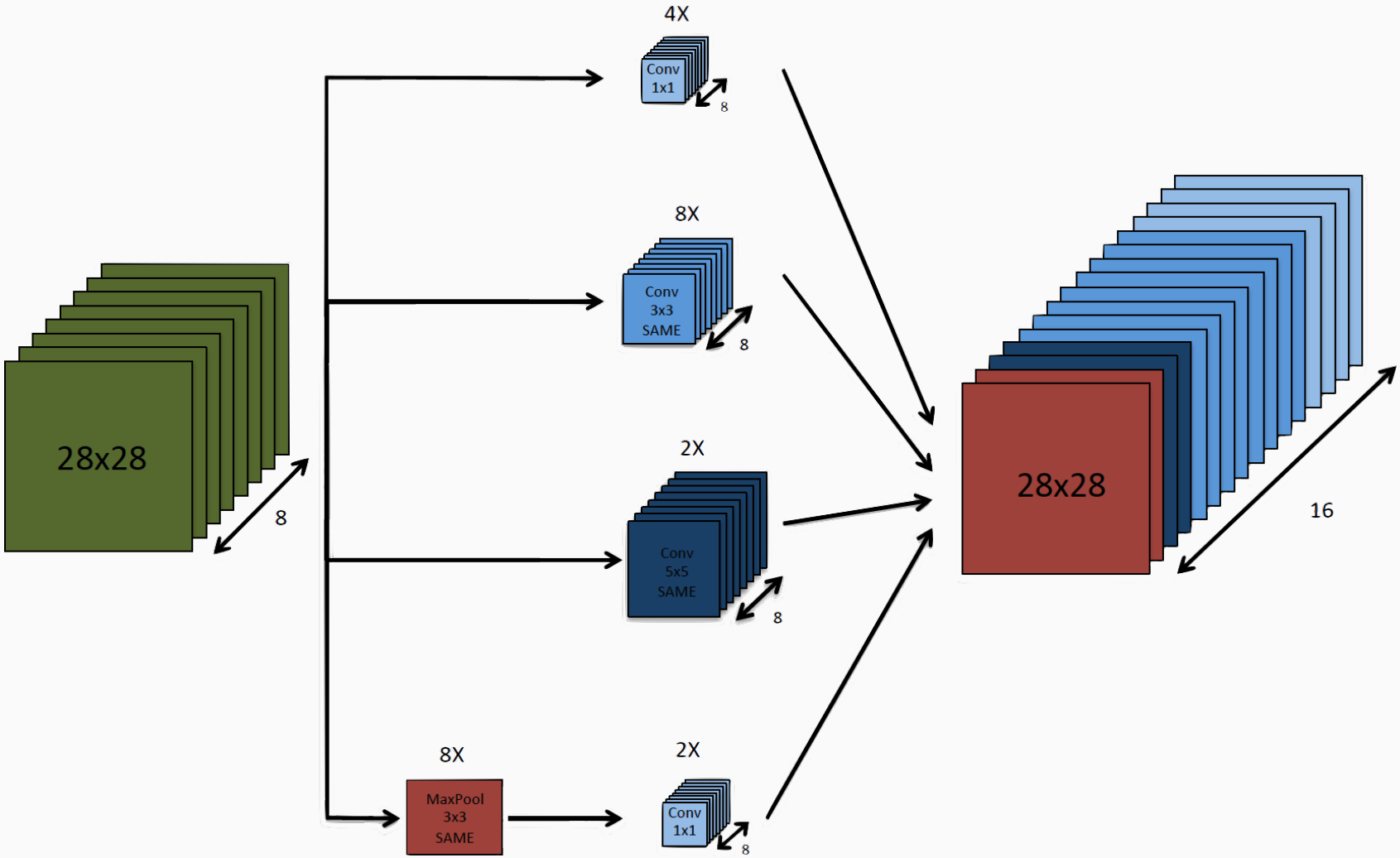
Inception Block - Szegedy et al. [2015]



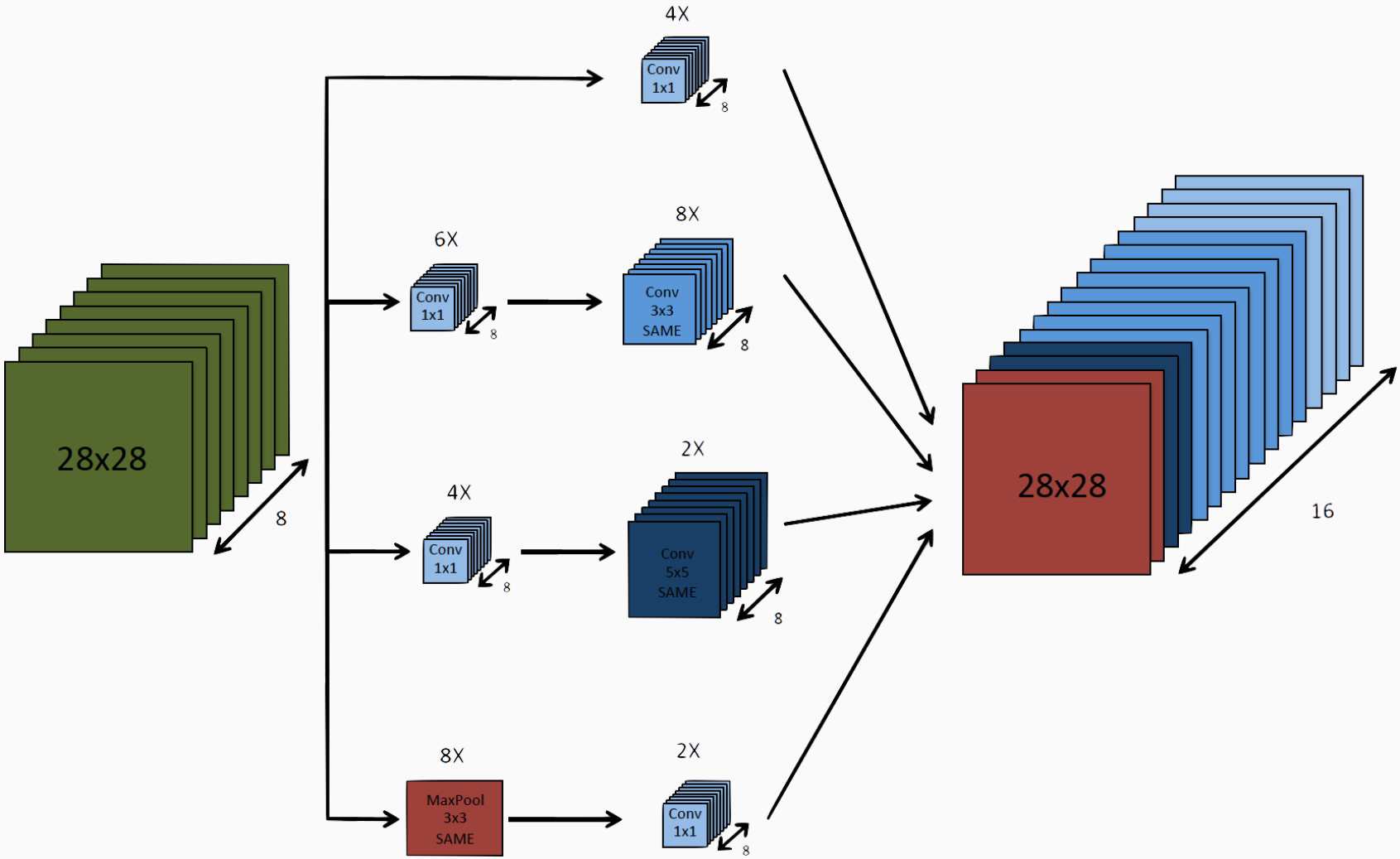
Inception Block - Szegedy et al. [2015]



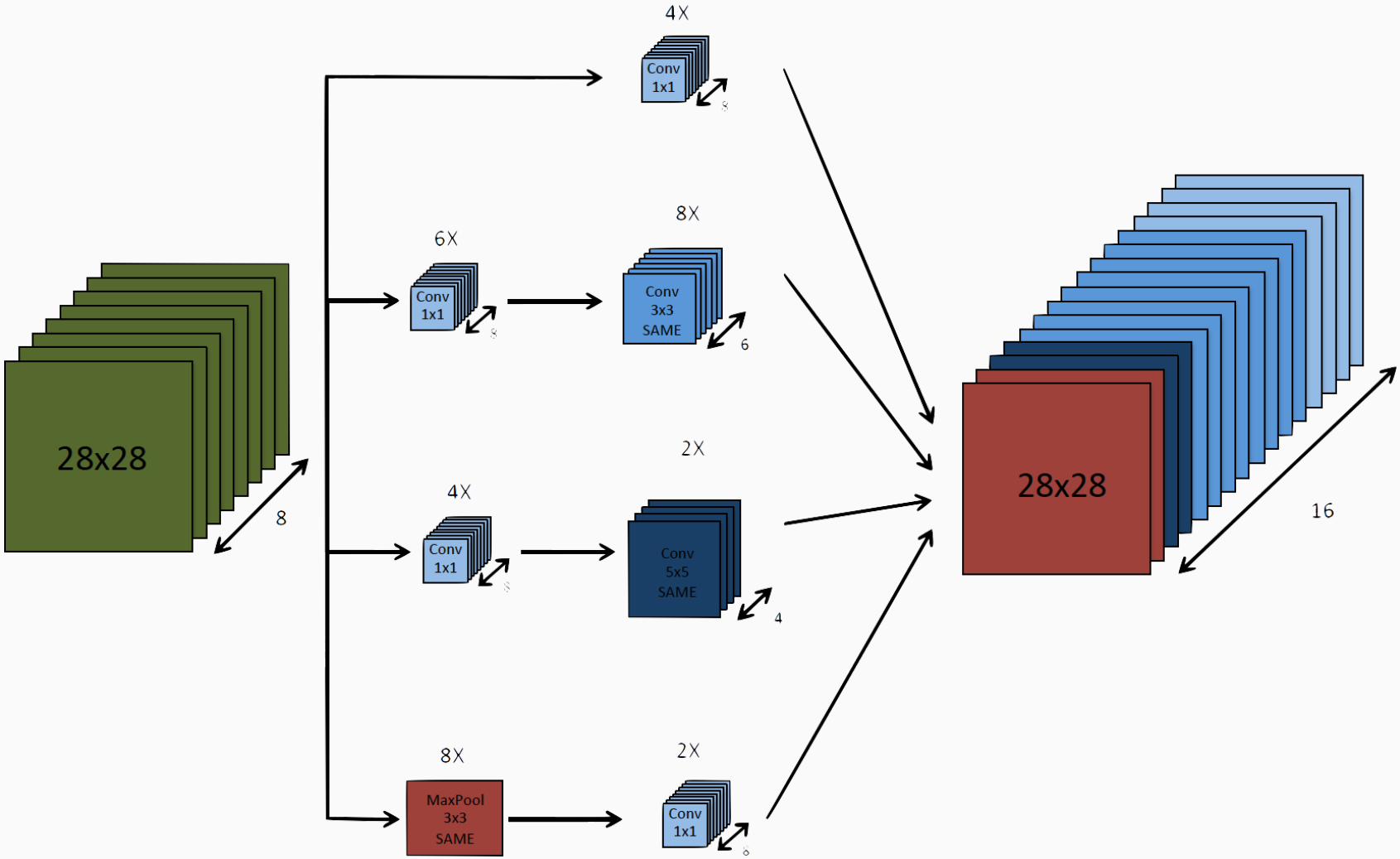
Inception Block - Szegedy et al. [2015]



Inception Block - Szegedy et al. [2015]

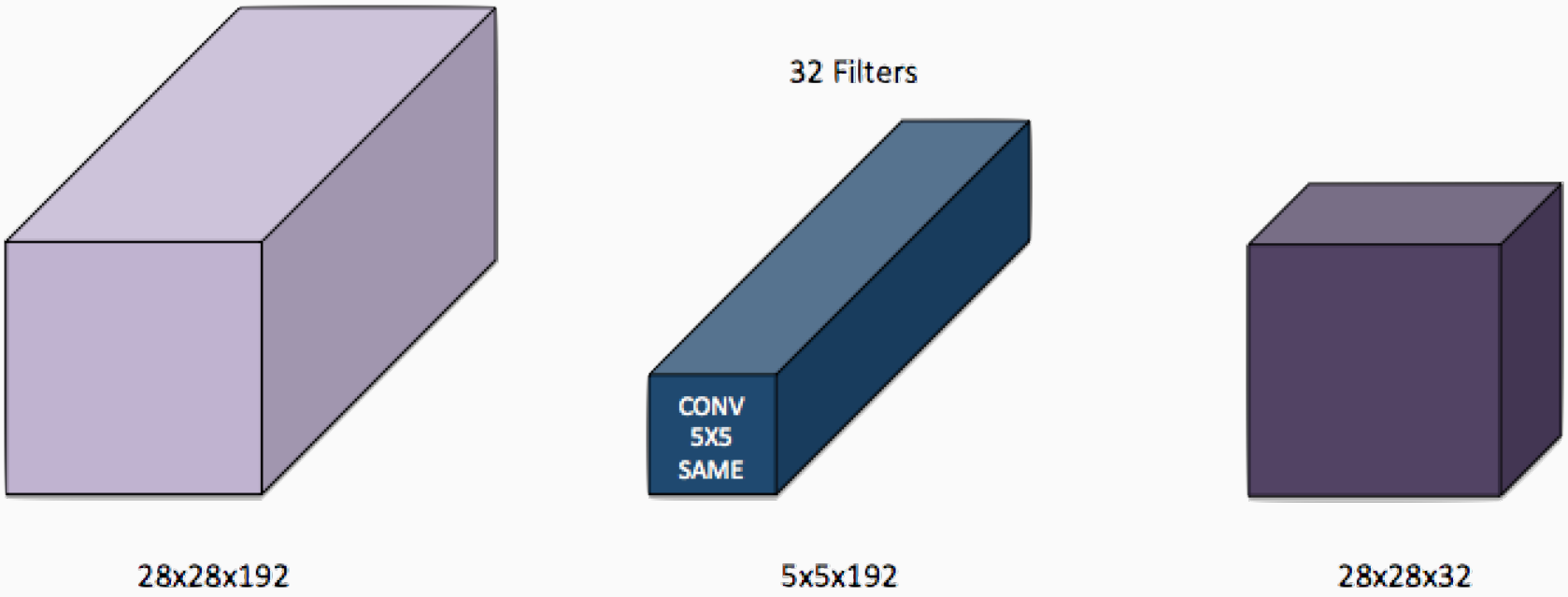


Inception Block - Szegedy et al. [2015]



1 x 1 convolution - Problem in computational cost

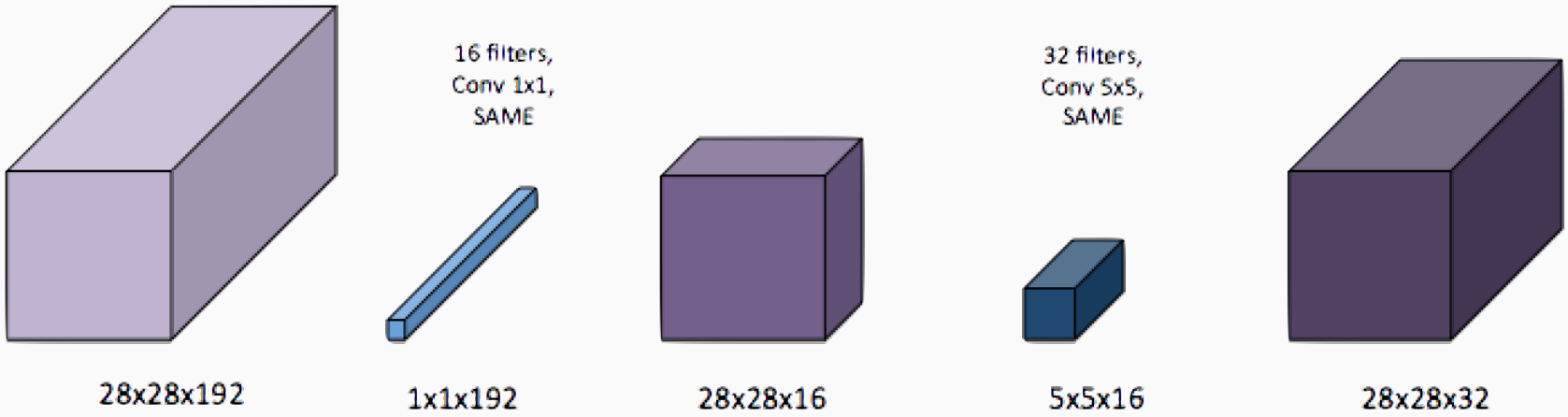
Network in Network Lin et al. [2013]



- Parameters : 5x5x192x32 = 153.600
- Operations : 153.600 x 28x28 = 120.422.400

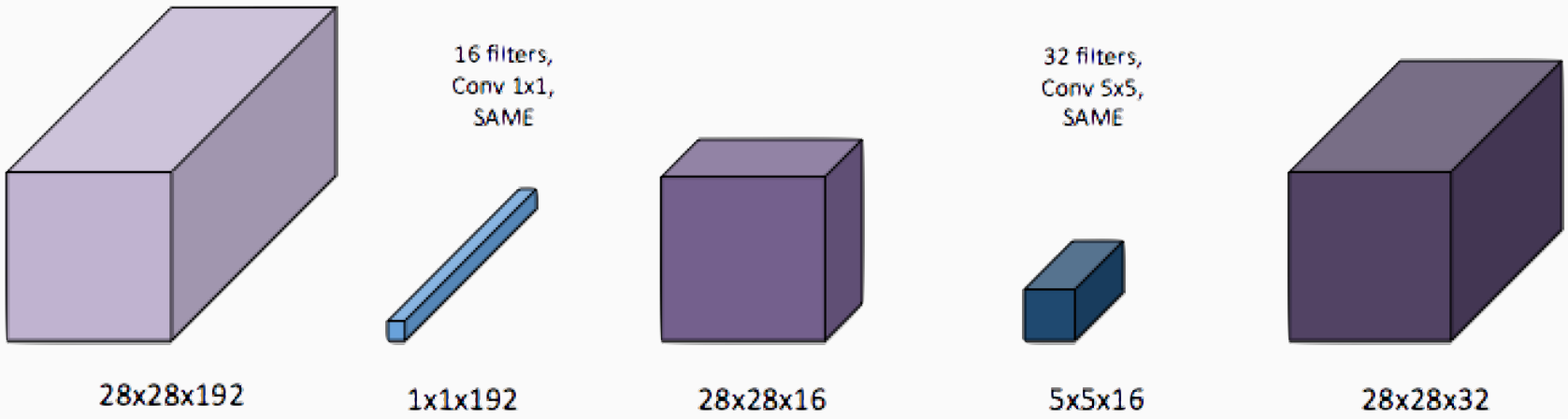
1 x 1 convolution - Problem in computational cost

Network in Network Lin et al. [2013]



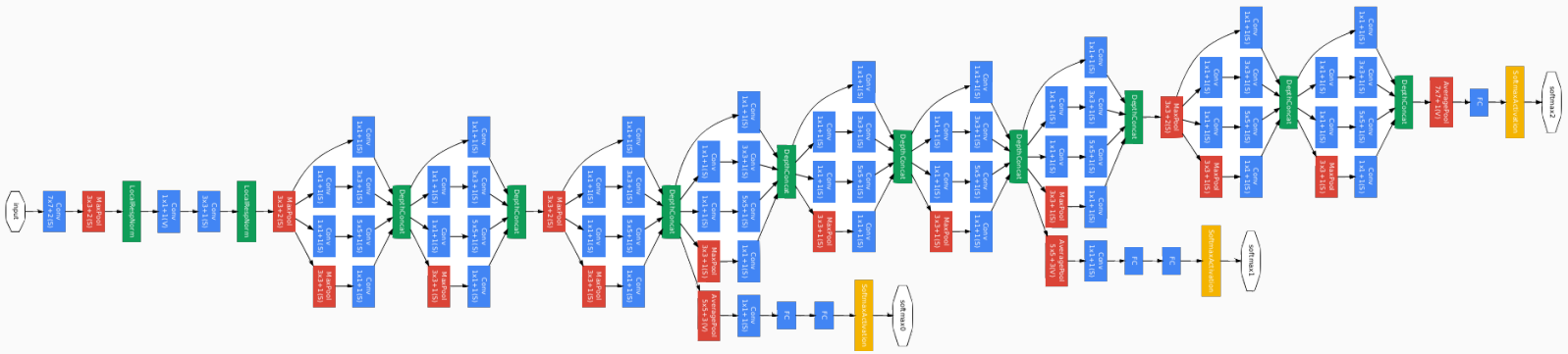
1 x 1 convolution - Problem in computational cost

Network in Network Lin et al. [2013]

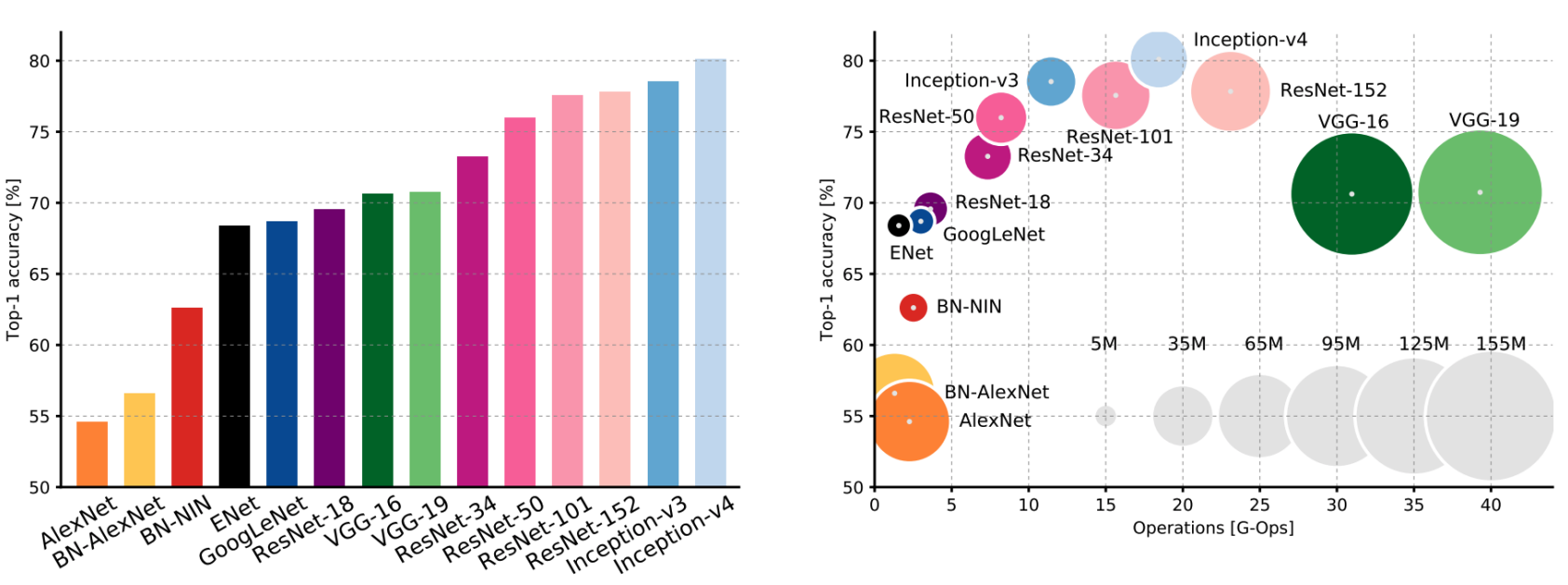


- Parameters : 1x1x192x16, 5x5x16x32 = (3072, 12800)
- Operations : 3.072x28x28 + 12.800 = 2.408.448+ 10.035.200 = 12.443.648

Inception Model - Szegedy et al. [2015]



State of the art



Image

source : Canziani et al. [2016]

- In constant evolution,
- new architecture regularly proposed.

Problématique pour des applications industrielles :

- réflexion sur l'architecture du réseau,
- nécessite beaucoup de données,
- réseau plus complexe,
- temps d'apprentissage très long.

Some solutions

Pre-trained model

- Some models using previous architectures have already be trained on massive amount of data (ImageNet).
- Theses models are available on easily usable on deep learning such that keras.
- Two ways to used them :
  - **Transfer Learning**
  - **Fine Tuning**

Data Augmentation“

More details later...

Classification d'image avec Keras (TP)

Deux cas d'études :

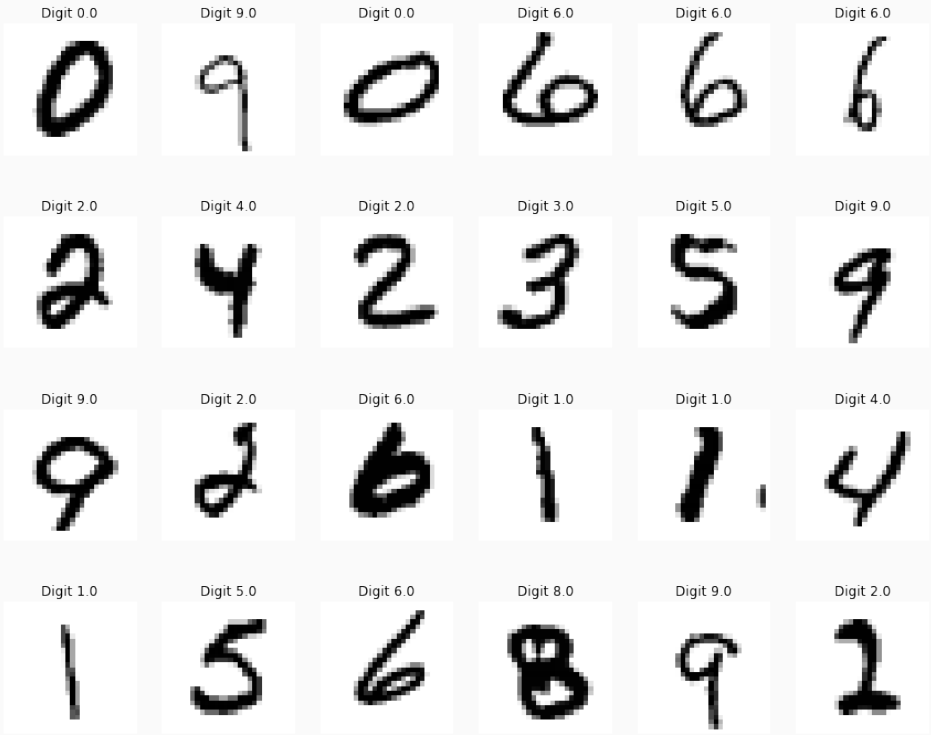
- Problème *simple* : MNIST.
  - Comparaison avec les autres algorithmes étudiés (*random forest*).
  - Performance du GPU.
- Problème plus *complexe* : Cats VS Dogs.
  - Utilisation des modèles pré-entrainés.
  - Performance du GPU.

Classification d'image avec Keras (TP)

mnist

MNIST

- 28 x 28 = 784 pixels.
- 1 niveau de couleur : gris.
- Apprentissage = 60.000, Test = 10.000.



MNIST - Perceptron multicouche

Input

```
model = km.Sequential()
model.add(kl.Dense(512, activation='relu', input_shape=(784,)))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(512, activation='relu'))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(N_classes, activation='softmax'))
```

input_shape = (nombre de paramètre,)

Output

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
```

669,706 paramètres

MNIST - Réseau de convolution

Input

```
# Convolution
model.add(kl.Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28,28, 1), data_format="channels_last"))
model.add(kl.Conv2D(64, (3, 3), activation='relu'))
model.add(kl.MaxPooling2D(pool_size=(2, 2)))
model.add(kl.Dropout(0.25))

# 2D to 1D
model.add(kl.Flatten())

# Classifier
model.add(kl.Dense(128, activation='relu'))
model.add(kl.Dropout(0.5))
model.add(kl.Dense(N_classes, activation='softmax'))
```

input_shape = (Nombre de pixel x, Nombre de pixel y, nombre de niveaux de couleurs)
data_format = ”Ou est le niveau de couleur”

MNIST - Réseau de convolution

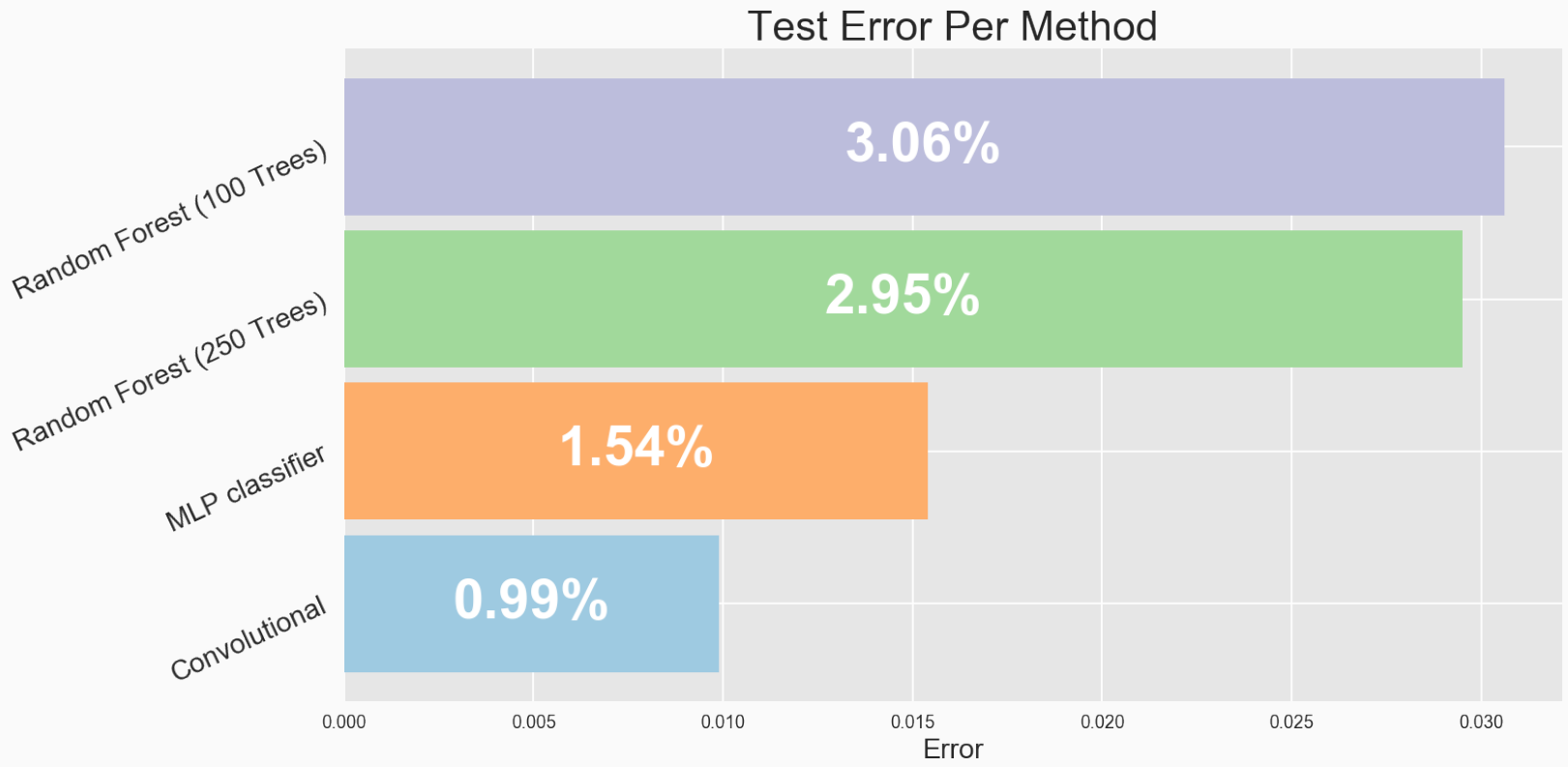
Output

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
```

1,199,882 paramètres

MNIST - Erreur de classification - Test

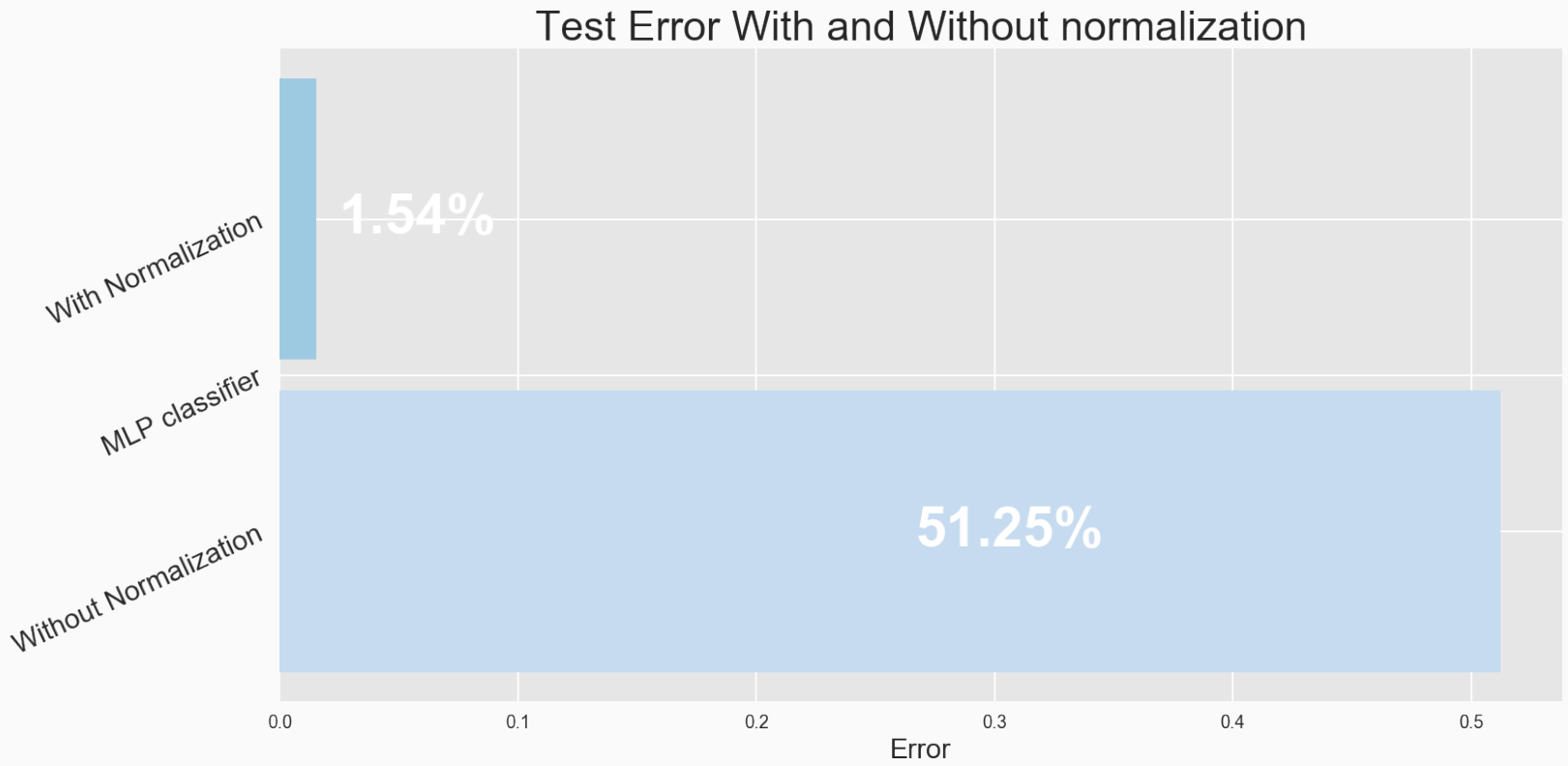
batch_size = 128, epochs = 20



Amélioration assez nette des résultats.

MNIST - Erreur de classification - Test - Normalisation

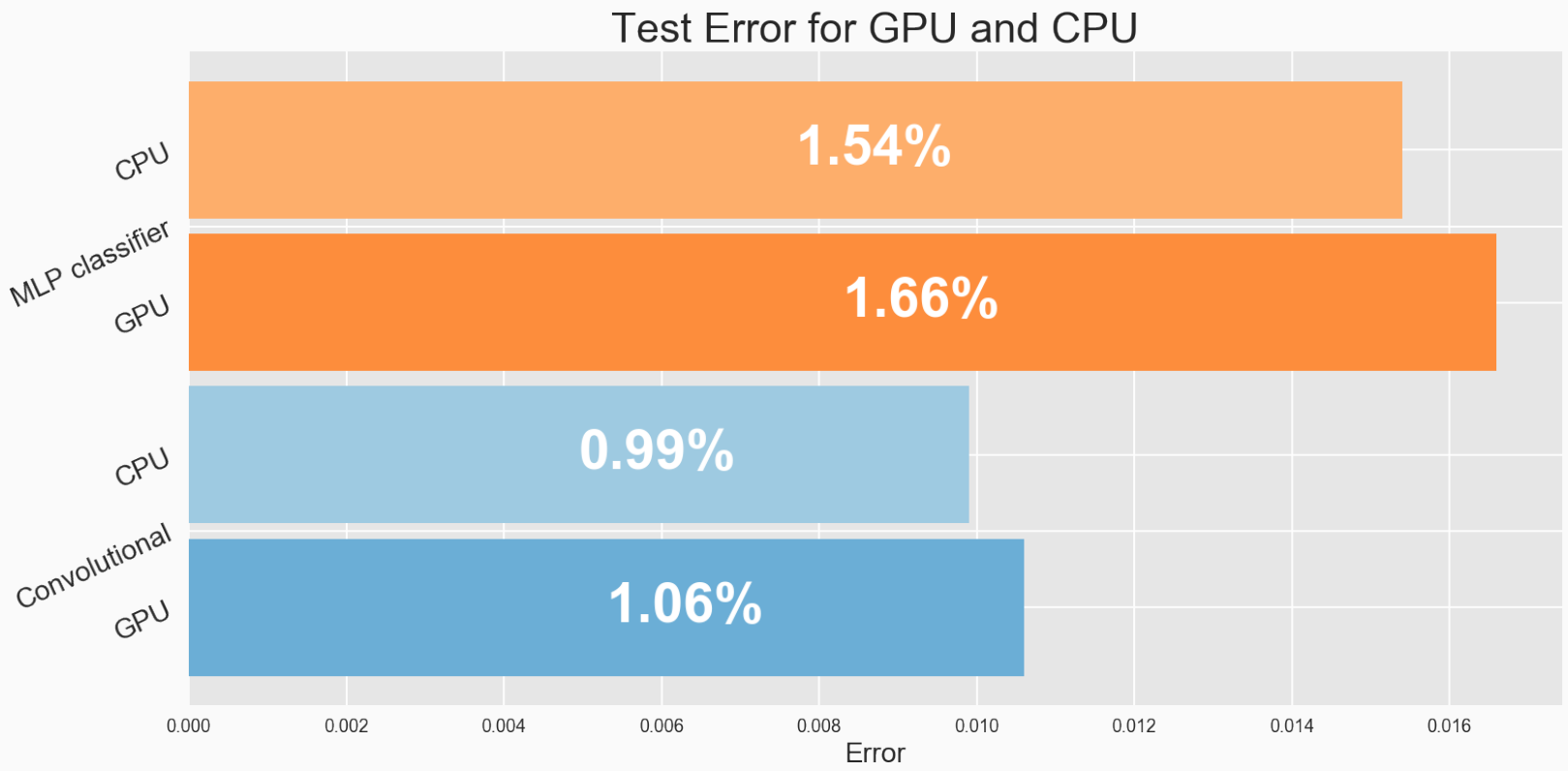
batch_size = 128, epochs = 20



Très grande influence de la normalisation des images sur le résultat.

MNIST - Erreur de classification - Test - CPU/GPU

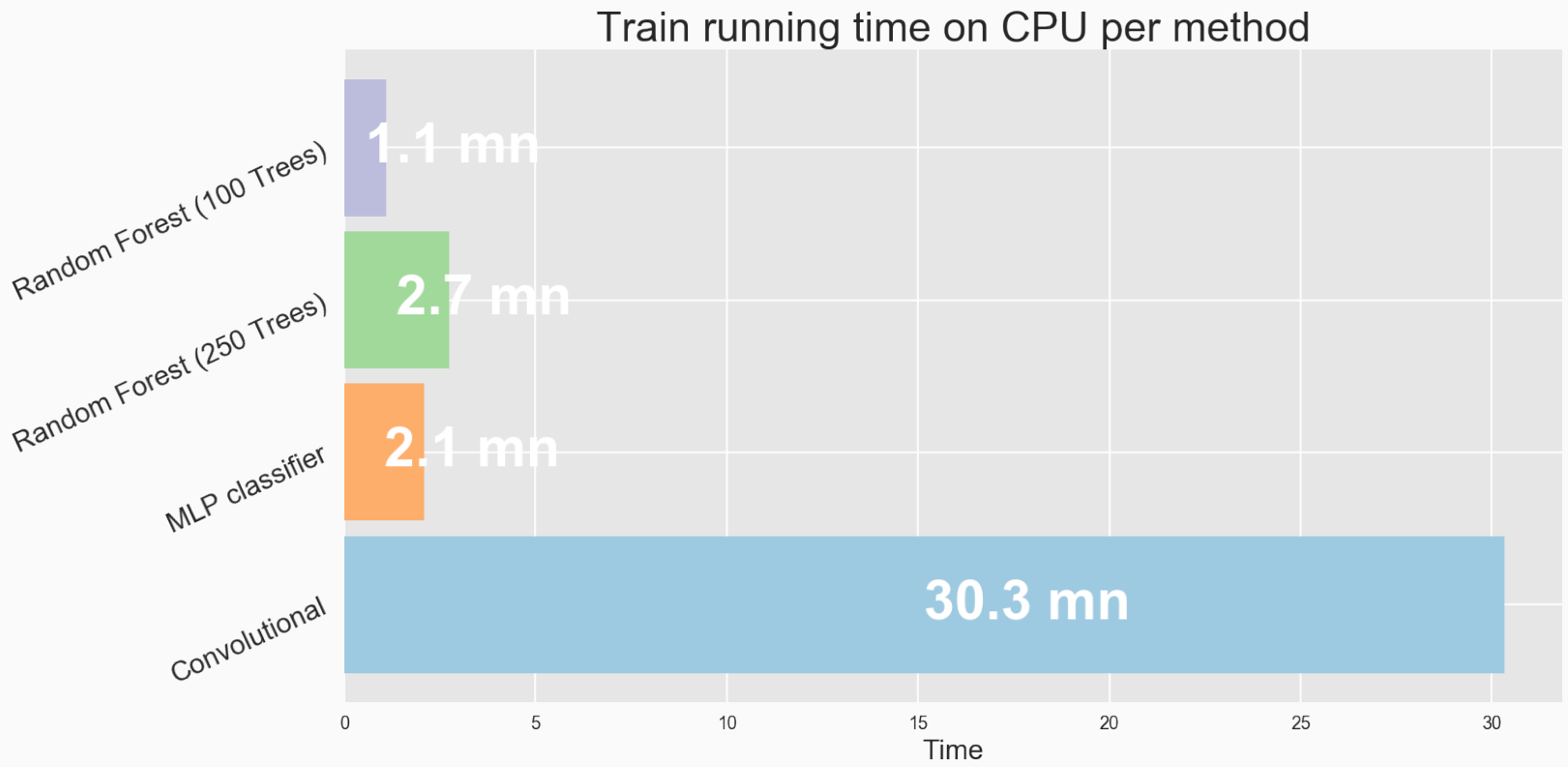
batch_size = 128, epochs = 20



Des résultats stables entre le CPU et le GPU.

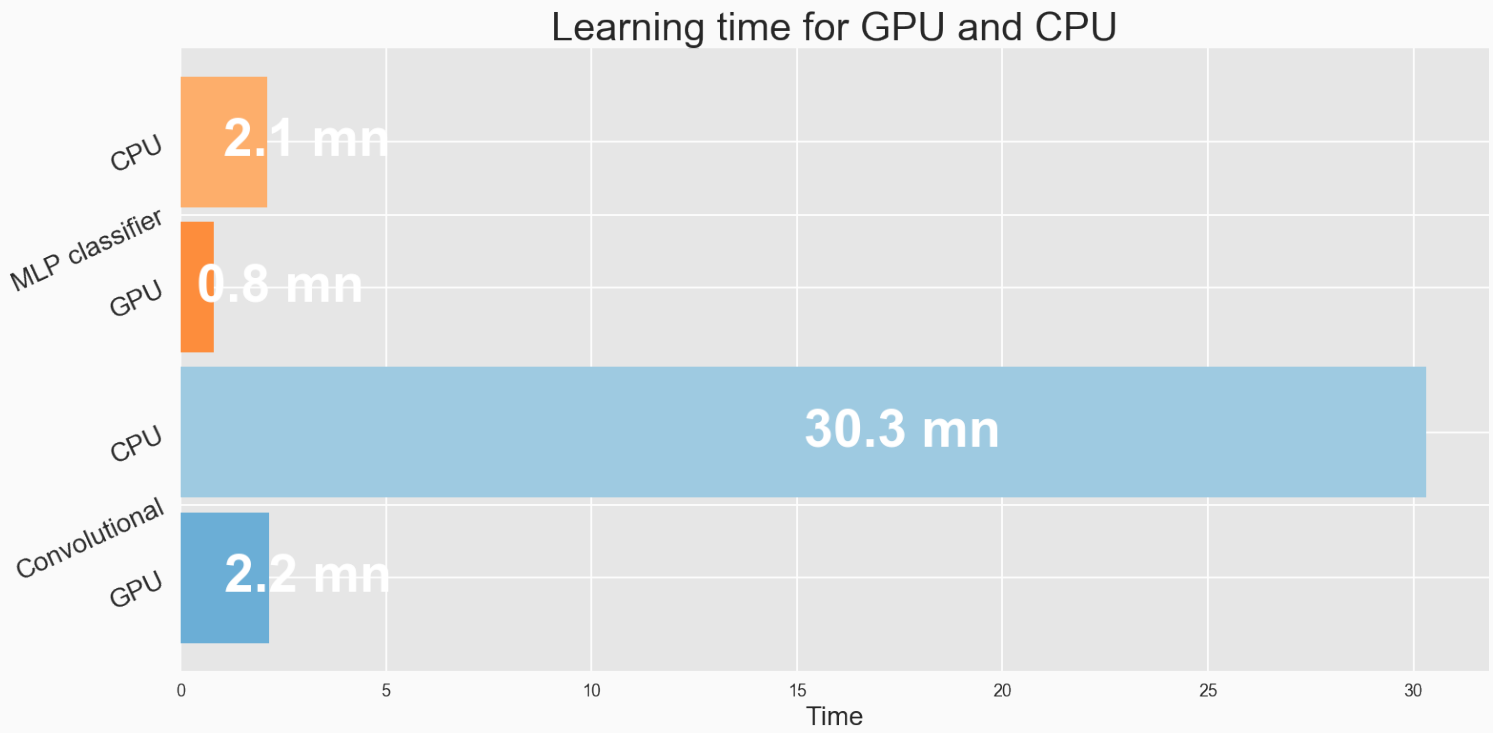
MNIST - Temps d'apprentissage - CPU

batch_size = 128, epochs = 20 - Machine GEI-103



MNIST - Temps d'apprentissage - CPU/GPU

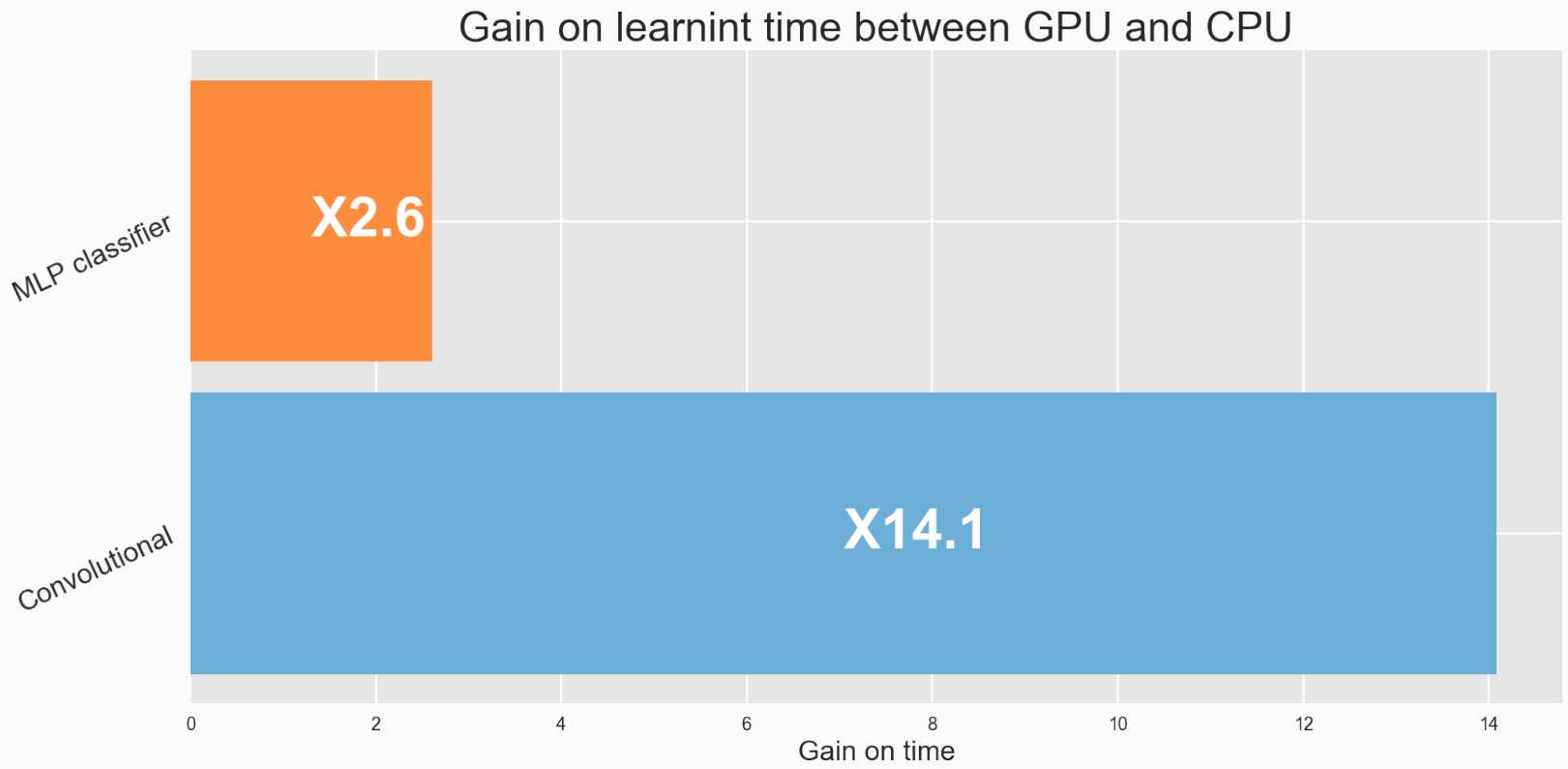
batch_size = 128, epochs = 20 - Machine GEI-103



Le GPU permet un très net gain sur le temps d'apprentissage.

MNIST - Temps d'apprentissage - CPU/GPU

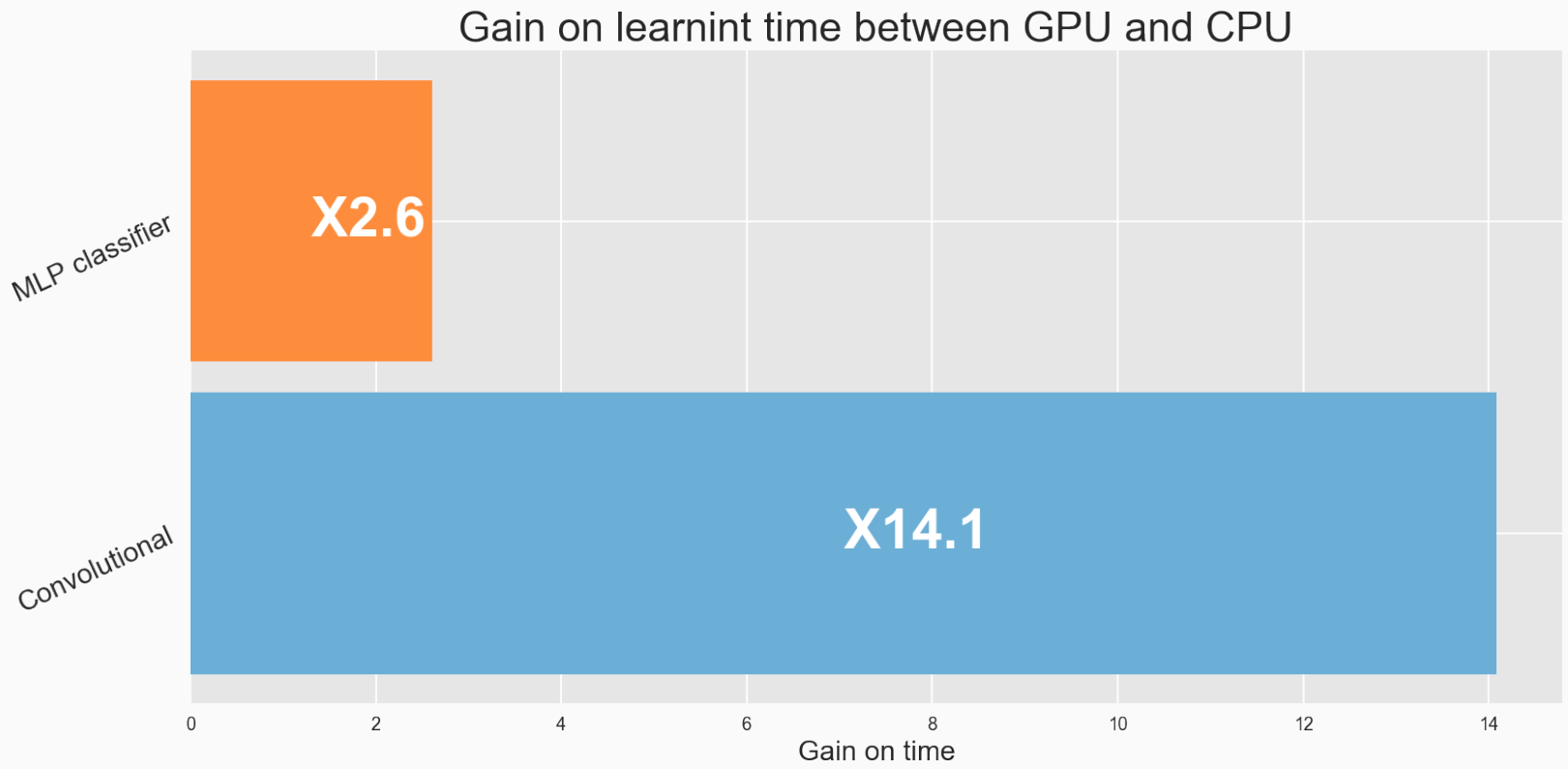
batch_size = 128, epochs = 20 - Machine GEI-103



Comment expliquer cette différence ?

MNIST - Temps d'apprentissage - CPU/GPU

batch_size = 128, epochs = 20 - Machine GEI-103



Comment expliquer cette différence ?

- Réseau de convolution plus complexe..
- ... Coût du transfert des données plus vite atténué.

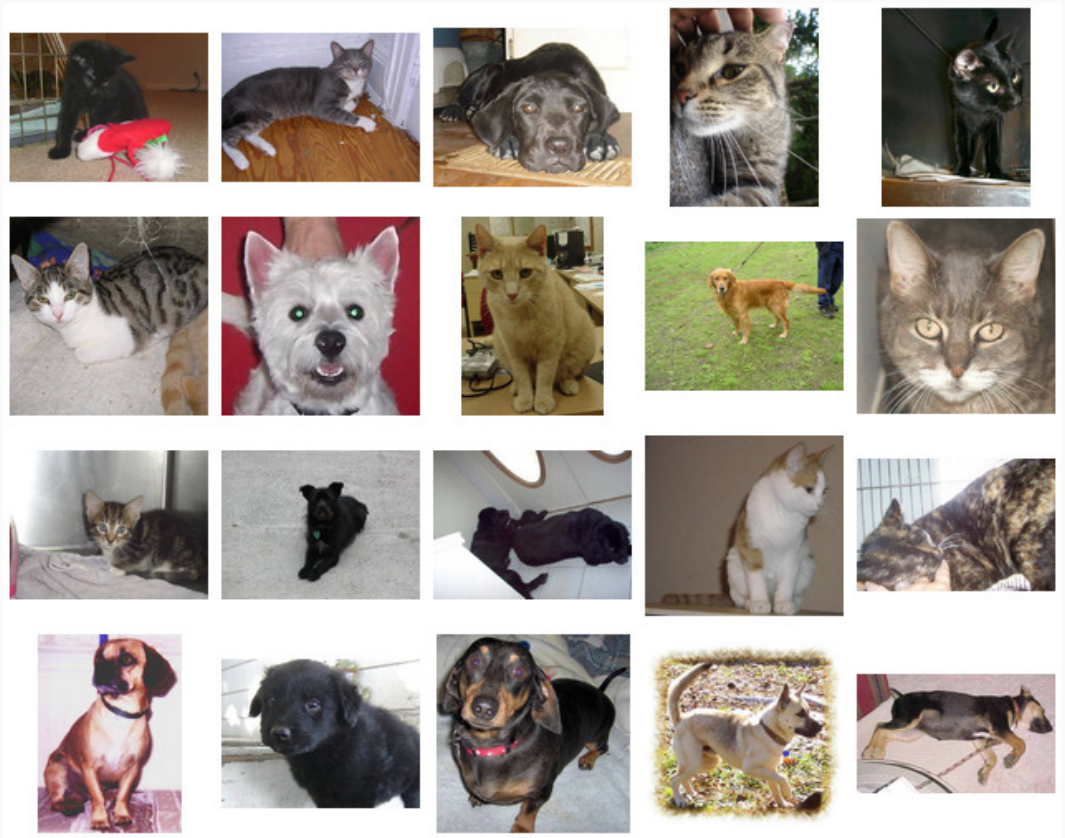
Classification d'image avec Keras (TP)

Cats Vs Dogs

Cats VS Dogs

Un jeu de donnée plus complexe :

- Images de tailles différentes.
- 3 niveaux de couleurs : bleu, rouge, vert.
- Jeu de données 25.000 images.



Cats VS Dogs - Augmentation des données

1/ Déterminer les transformations à appliquer :

```
datagen = kpi.ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

- Rotation.
- Translation.
- Standardisation.
- ZCA Whitening.

2/ Générer des transformations aléatoires depuis un dossier d'images

```
train_generator = train_datagen.flow_from_directory(
        data_dir_sub+"/train/",
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')
```

- Fonction Lazy.
- Image de même dimension.
- Evite le sur-apprentissage.

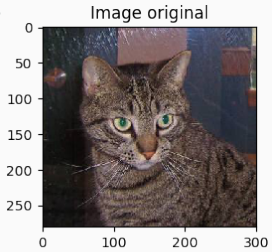
Cats VS Dogs - Augmentation des données

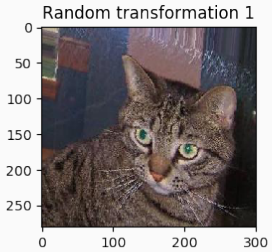
3/ Apprentissage sur les données augmentées

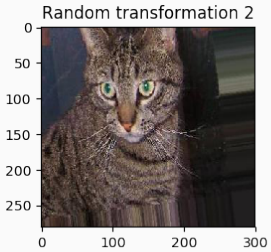
```
model_conv.fit_generator(train_generator, steps_per_epoch=N_train // batch_size, epochs=epochs,
      validation_data=validation_generator,validation_steps=N_val // batch_size)
```

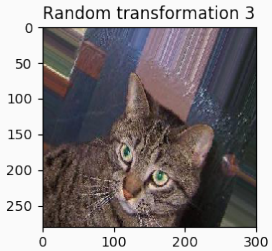
steps_per_epoch = Nb individu / batch_size

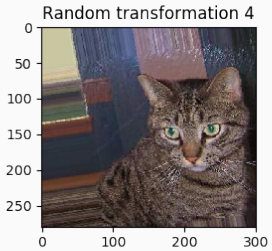
Exemple de transformation :

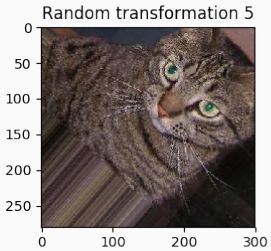


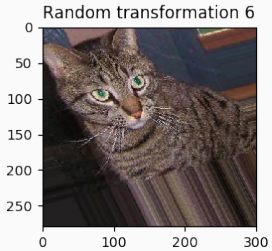


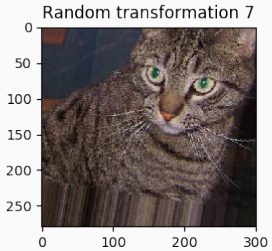


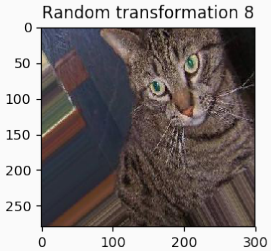












CatsVSDgogs - Réseau de convolution

Definition

```
model_conv = km.Sequential()
model_conv.add(kl.Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3),
                 data_format="channels_last"))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

model_conv.add(kl.Conv2D(32, (3, 3)))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

model_conv.add(kl.Conv2D(64, (3, 3)))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.MaxPooling2D(pool_size=(2, 2)))

model_conv.add(kl.Flatten())  # this converts our 3D feature maps to 1D feature vectors
model_conv.add(kl.Dense(64))
model_conv.add(kl.Activation('relu'))
model_conv.add(kl.Dropout(0.5))
model_conv.add(kl.Dense(1))
model_conv.add(kl.Activation('sigmoid'))
```

input_shape = (Nombre de pixel x, Nombre de pixel y, nombre de niveau de couleur)
data_format = ”Ou est le niveau de couleur”

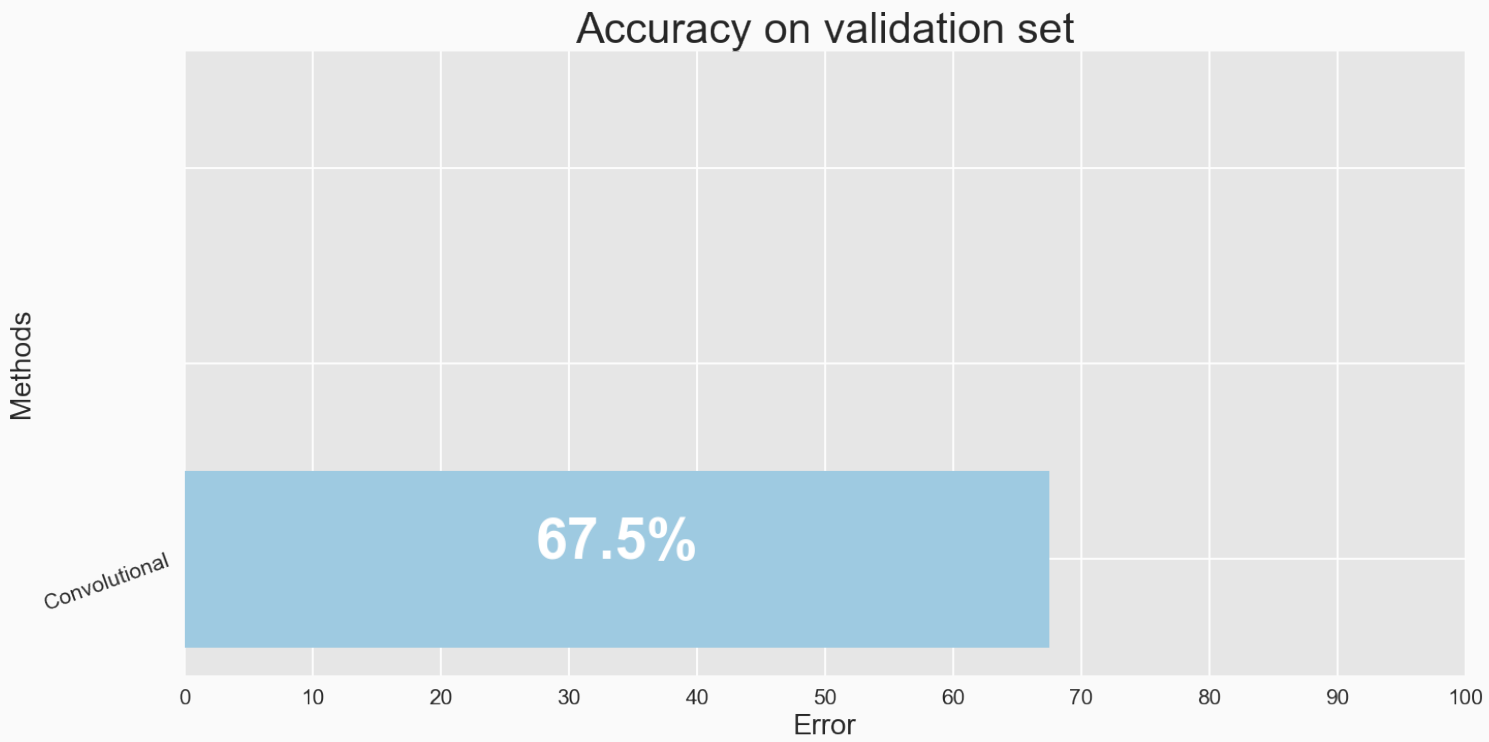
CatsVSDgogs - Réseau de convolution

Summary (1,212,513 paramètres)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                1183808
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
```

CatsVSDgogs - Validation score

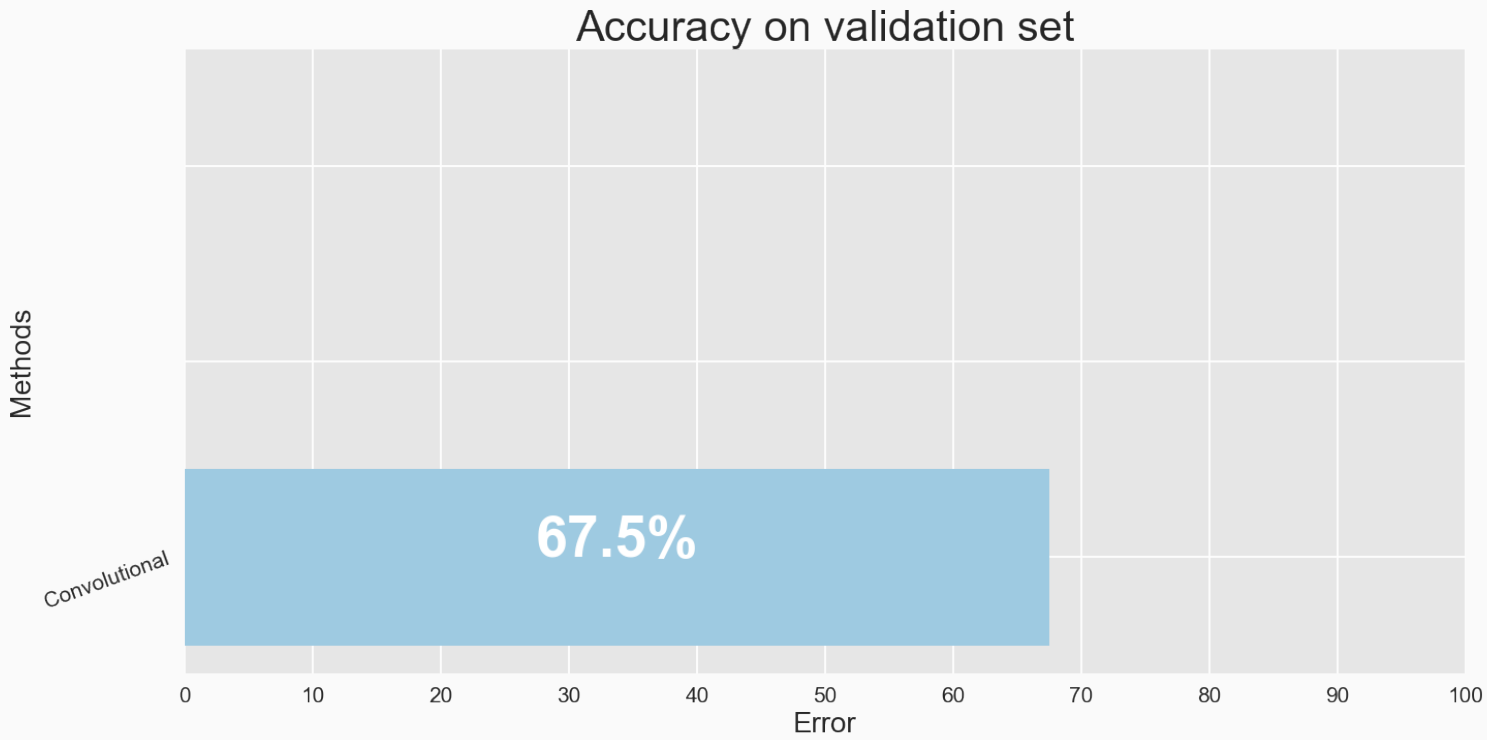
batch_size = 200, epochs = 15



Résultats assez faible.

CatsVSDgogs - Validation score

batch_size = 200, epochs = 15



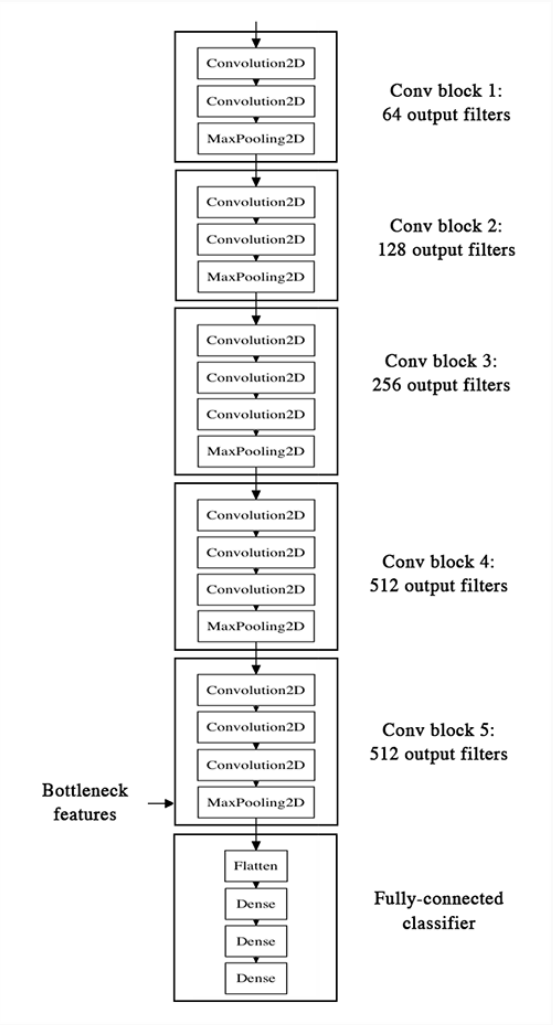
Résultats assez faible.

$\Longrightarrow$ Utilisation d'un modèle pré-entrainé.

CatsVSDgogs - Réseau pré-entrainé

Exemple : Réseau VGG-16 : 138,357,544 paramètres.

Utilisation en deux étapes :

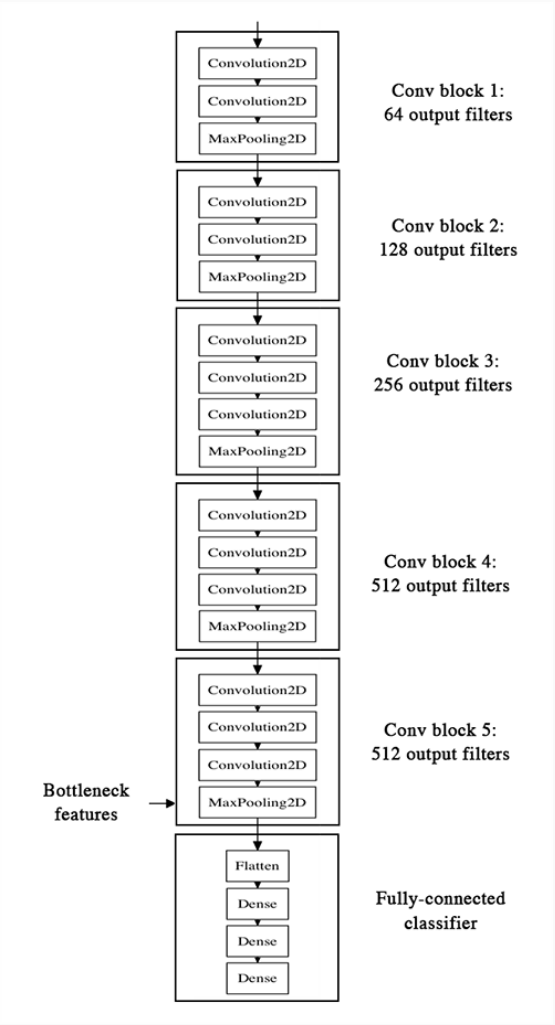


CatsVSDgogs - Réseau pré-entrainé

Exemple : Réseau VGG-16 : 138,357,544 paramètres.

Utilisation en deux étapes :

- **Extracteur de features** : On utilise les blocs de convolutions pour extraire des features

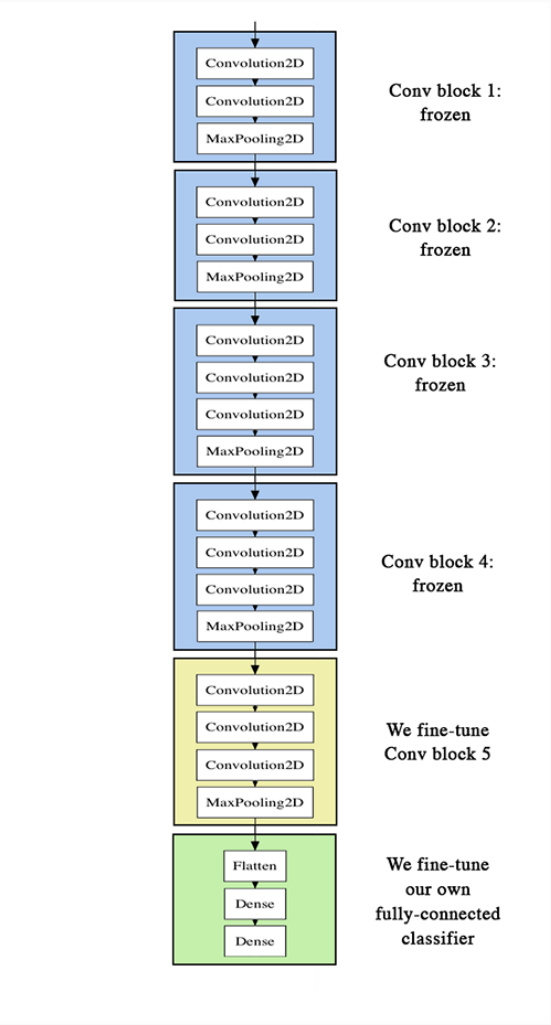


CatsVSDgogs - Réseau pré-entrainé

Exemple : Réseau VGG-16 : 138,357,544 paramètres.

Utilisation en deux étapes :

- **Extracteur de features** : On utilise les blocs de convolutions pour extraire des features
- **Fine Tuning** : On ré-apprend les paramètres des deux derniers blocs



CatsVSDgogs - VGG-16 - 1/ Extracteur de features

```
 # On charge le modèle sans le dernier bloc de classification :
model_VGG16_without_top = ka.VGG16(include_top=False, weights='imagenet')

# Model summary
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
....
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================

# Features extraction

datagen = kpi.ImageDataGenerator(rescale=1. / 255)
generator = datagen.flow_from_directory(
            data_dir_sub+"/train",
            target_size=(img_width, img_height),
            batch_size=batch_size,
            class_mode=None,
            shuffle=False)
features_train = model_VGG16_without_top.predict_generator(generator, N_train//batch_size,  verbose=1)
```

- Pas d'augmentation !
- features_train de dimension (4, 4, 512)

CatsVSDgogs - VGG-16 - 1/ Extracteur de features

On définit un classifieur simple prenant en entrée les données extraites du modèle VGG-16 :

```
model_VGG_fcm = km.Sequential()
# La première couche "aplatit" les données
model_VGG_fcm.add(kl.Flatten(input_shape=features_train.shape[1:]))
model_VGG_fcm.add(kl.Dense(64, activation='relu'))
model_VGG_fcm.add(kl.Dropout(0.5))
model_VGG_fcm.add(kl.Dense(1, activation='sigmoid'))
```

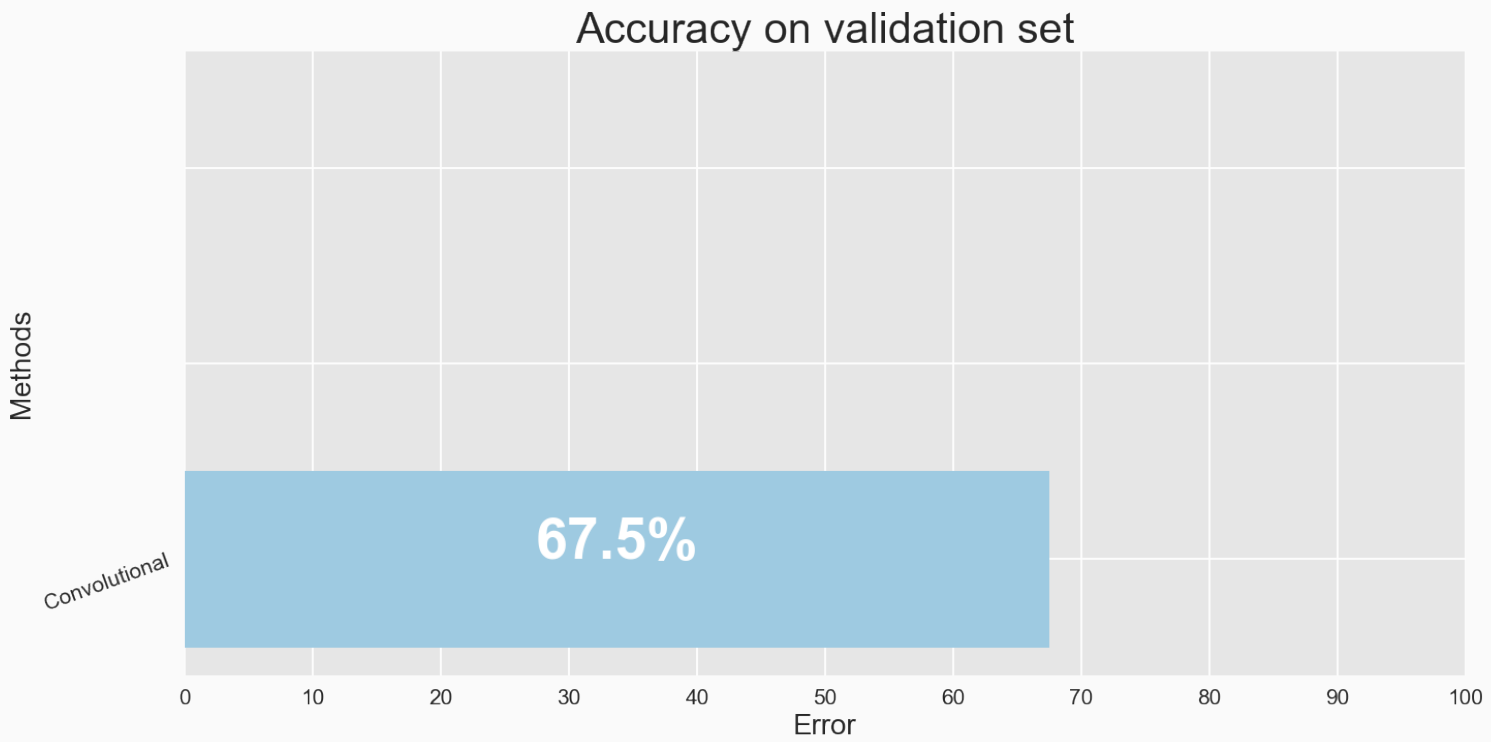
- input_shape de dimension (4, 4, 512)

```
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                524352
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
```

Modèle très simple utilisé sur des features issues d'un modèle très élaboré.

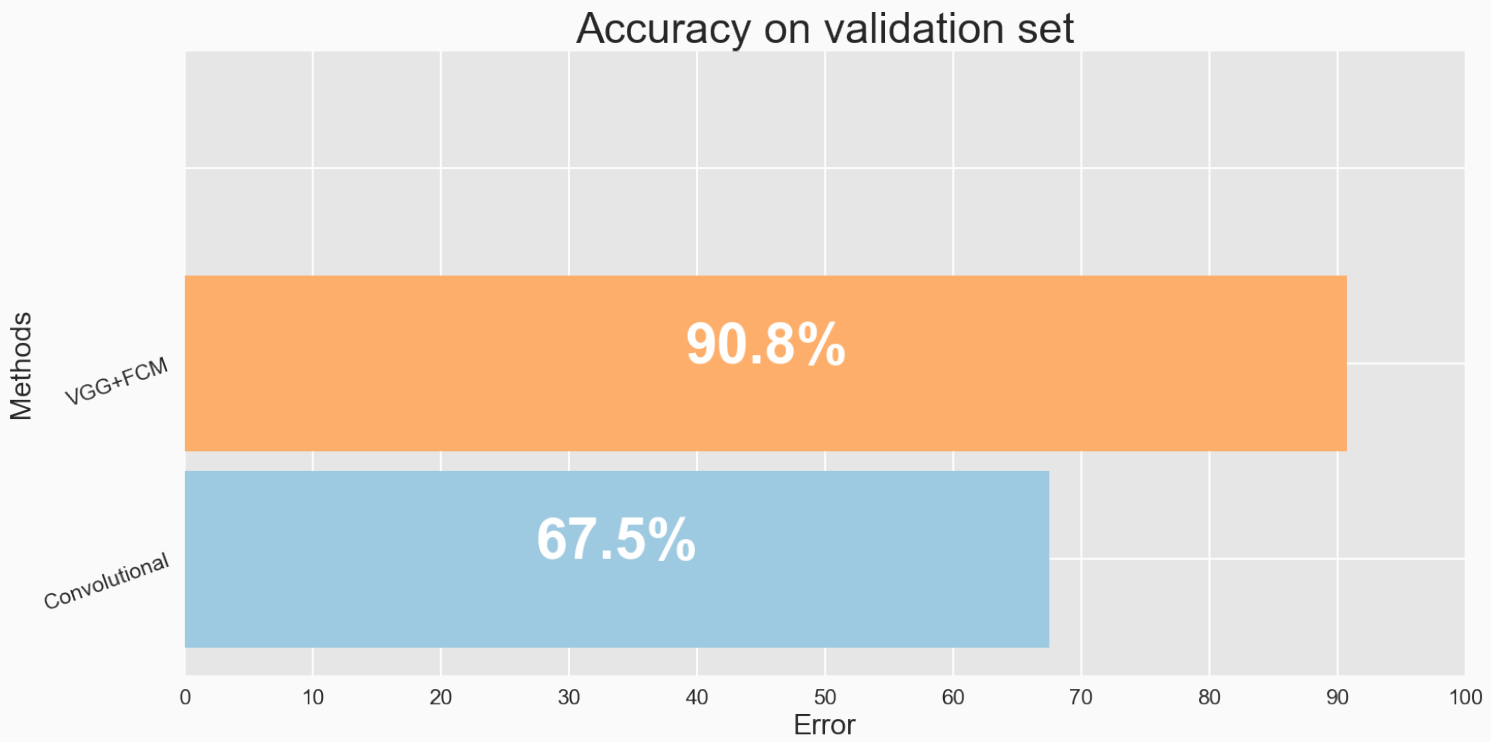
CatsVSDgogs - Validation score

batch_size = 200, epochs = 15



CatsVSDgogs - Validation score

batch_size = 200, epochs = 15



Nette amélioration !

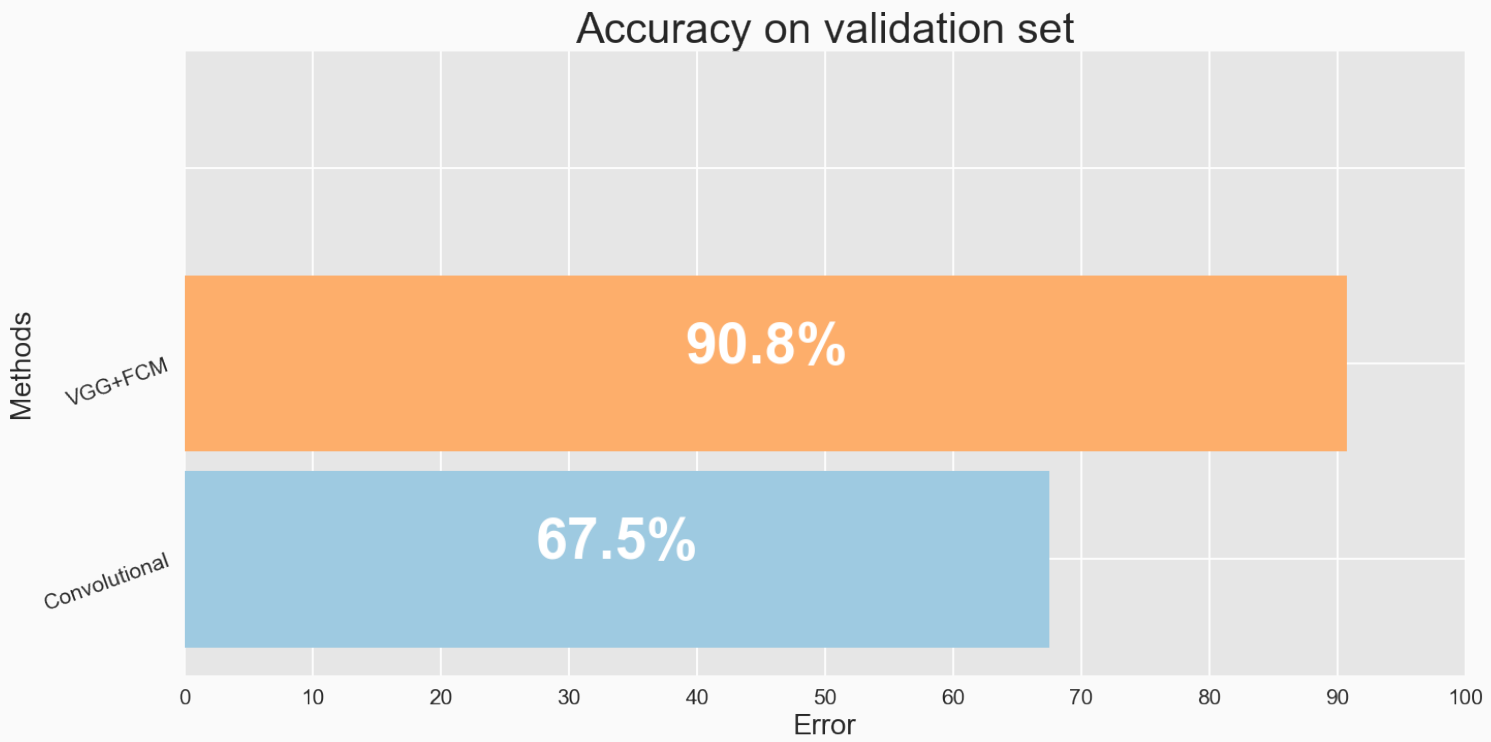
CatsVSDgogs - VGG-16 - 2/ Fine tuning

On règle les poids des deux derniers blocs du modèle :

```
 # Même modèle que précédement
top_model = km.Sequential()
top_model.add(kl.Flatten(input_shape=model_VGG16_without_top.output_shape[1:]))
top_model.add(kl.Dense(64, activation='relu'))
top_model.add(kl.Dropout(0.5))
top_model.add(kl.Dense(1, activation='sigmoid'))

 # On charge les poids du modèle évalué précédemment
top_model.load_weights(data_dir_sub+'/'+MODE+'_weights_model_VGG_fully_connected_model_%

# Add the classifier model on top of the convolutional base
model_VGG_LastConv_fcm = km.Model(inputs=model_VGG16_without_top.input,
outputs=top_model(model_VGG16_without_top.output))

#Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
...
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
sequential_5 (Sequential)    (None, 1)                 524417
=================================================================
Total params: 15,239,105
```

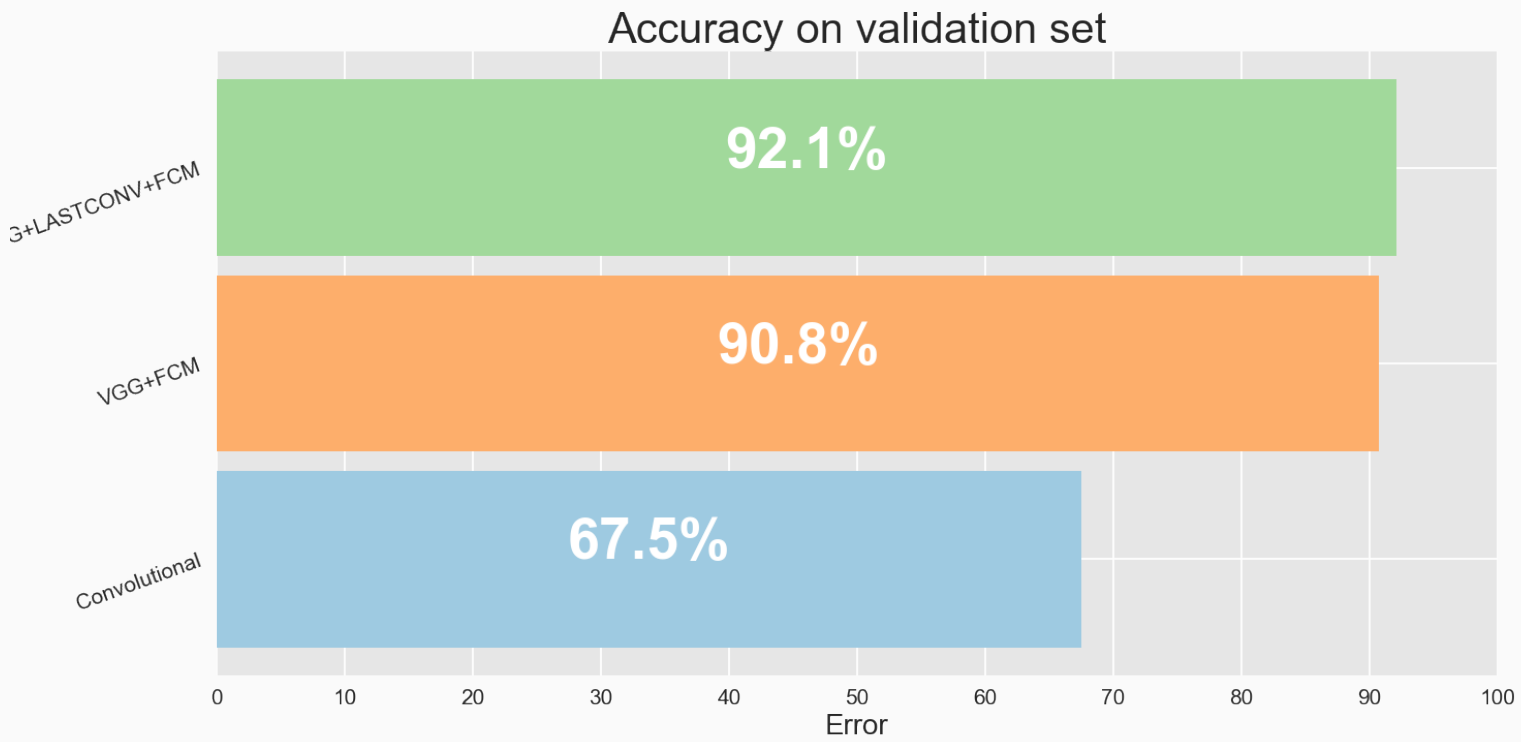
CatsVSDgogs - Validation score

batch_size = 200, epochs = 15



CatsVSDgogs - Validation score

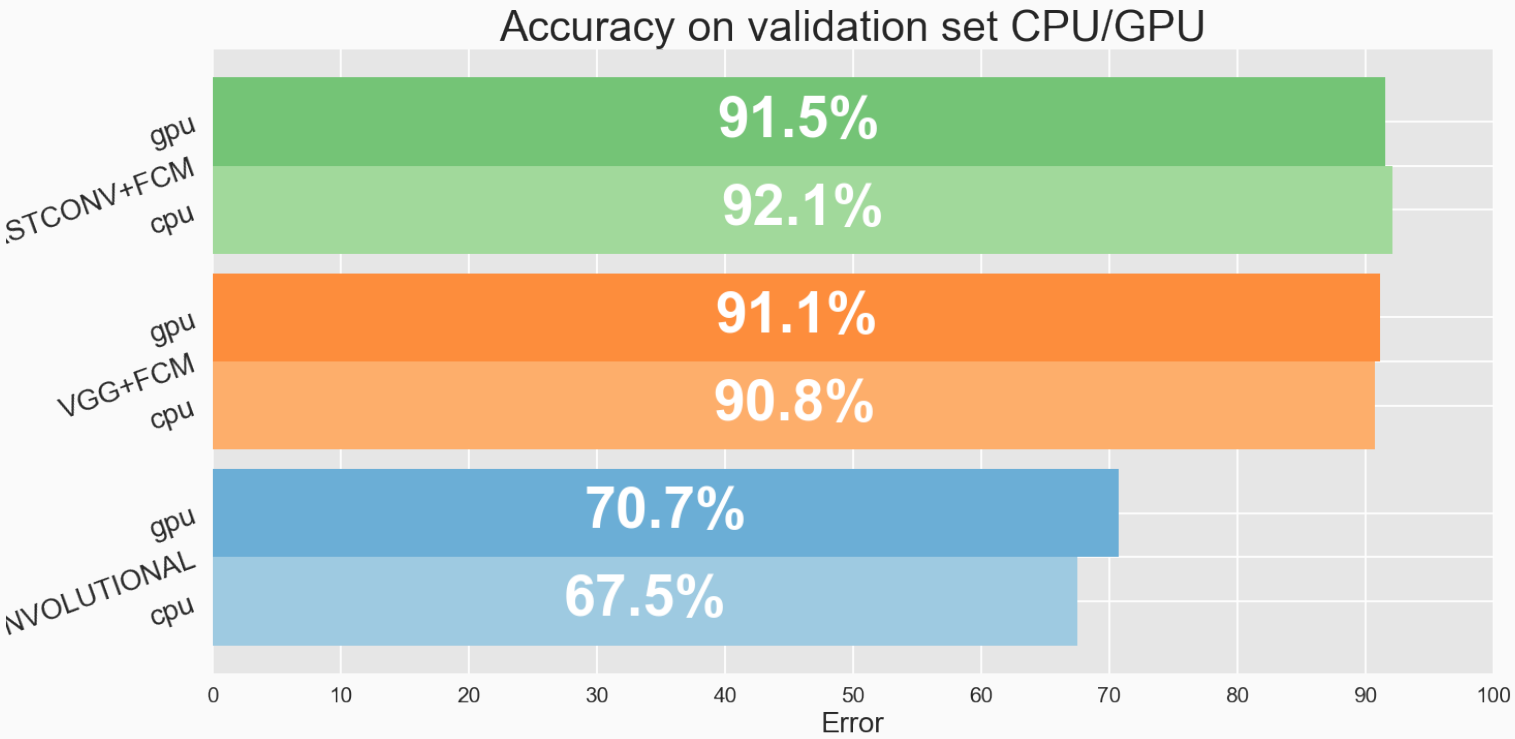
batch_size = 200, epochs = 15



Résultat encore amélioré

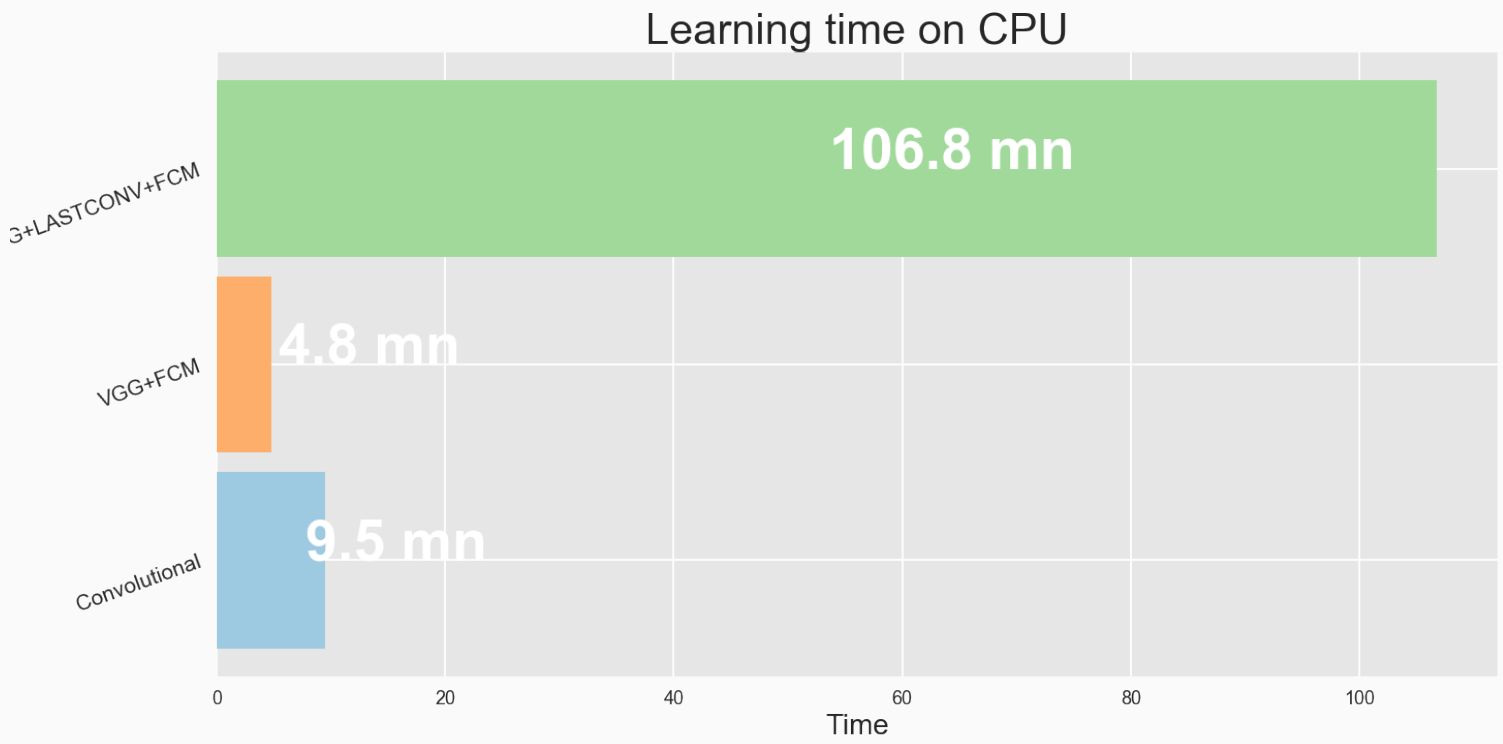
CatsVSDgogs - Validation score - CPU/GPU

batch_size = 200, epochs = 15



CatsVSDgogs - Temps d'apprentissage - CPU

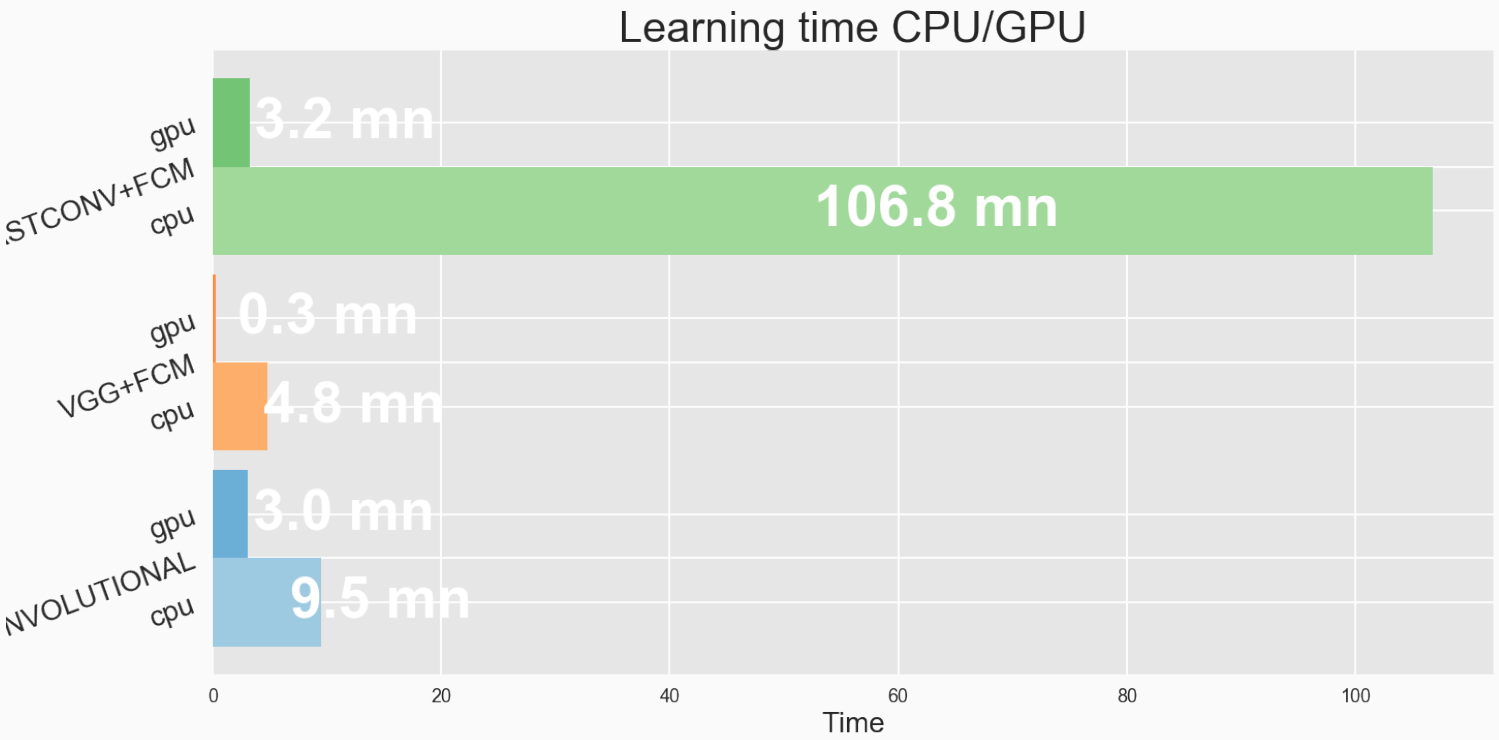
batch_size = 200, epochs = 15



L'apprentissage sur le modèle entier est très coûteux : pourquoi ?

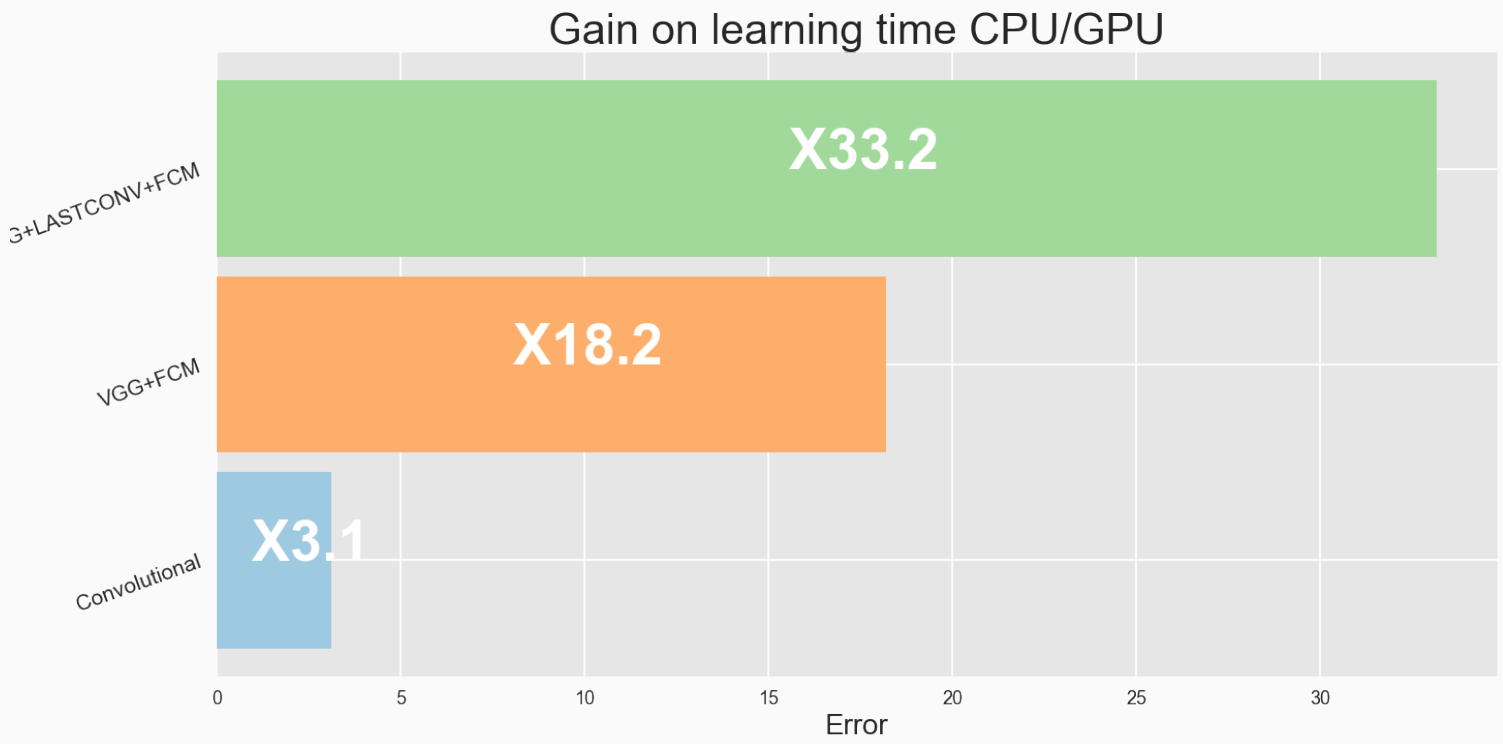
CatsVSDgogs - Temps d'apprentissage - CPU/GPU

batch_size = 200, epochs = 15



CatsVSDgogs - Temps d'apprentissage - CPU/GPU

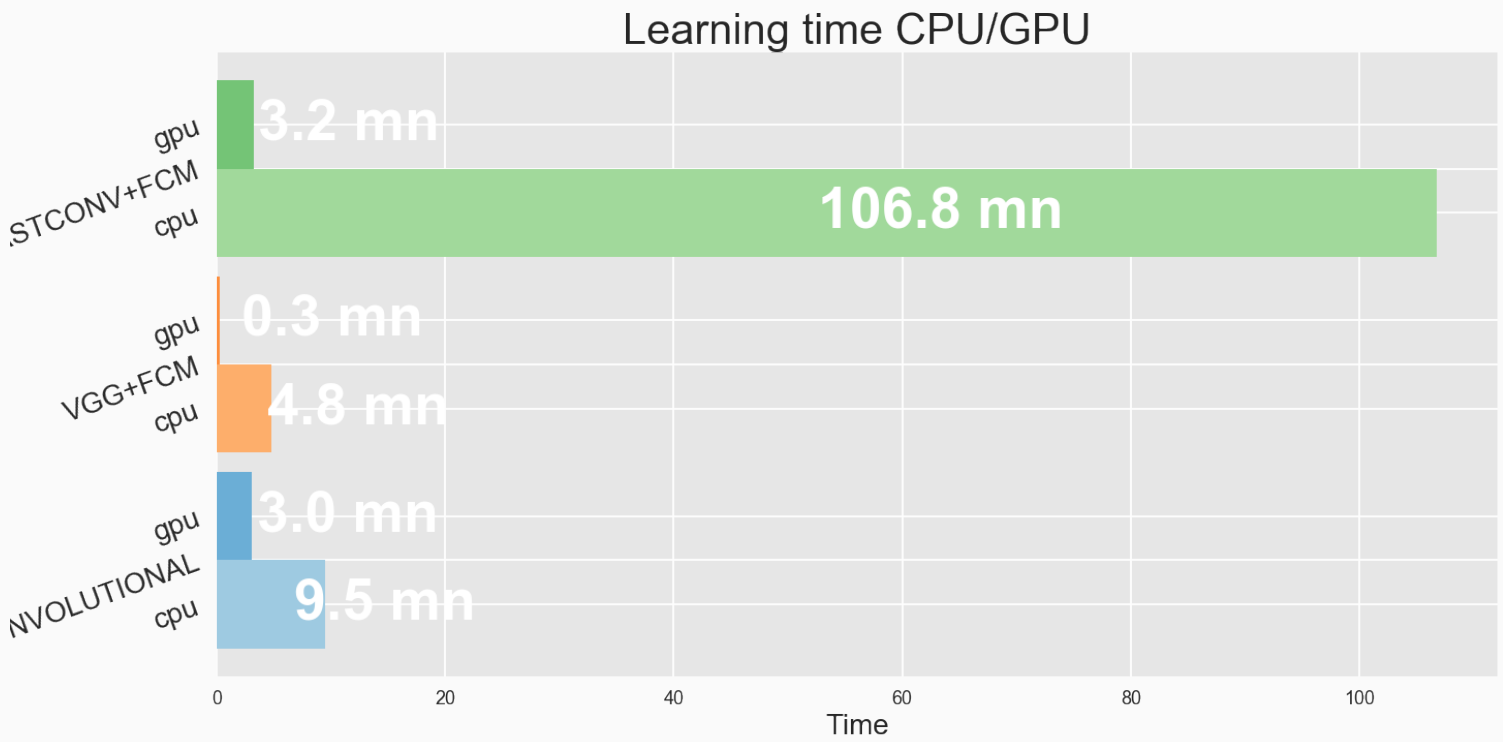
batch_size = 200, epochs = 15



Comment expliquer cette différence de gain ?

CatsVSDgogs - Temps d'apprentissage - CPU/GPU

batch_size = 200, epochs = 15

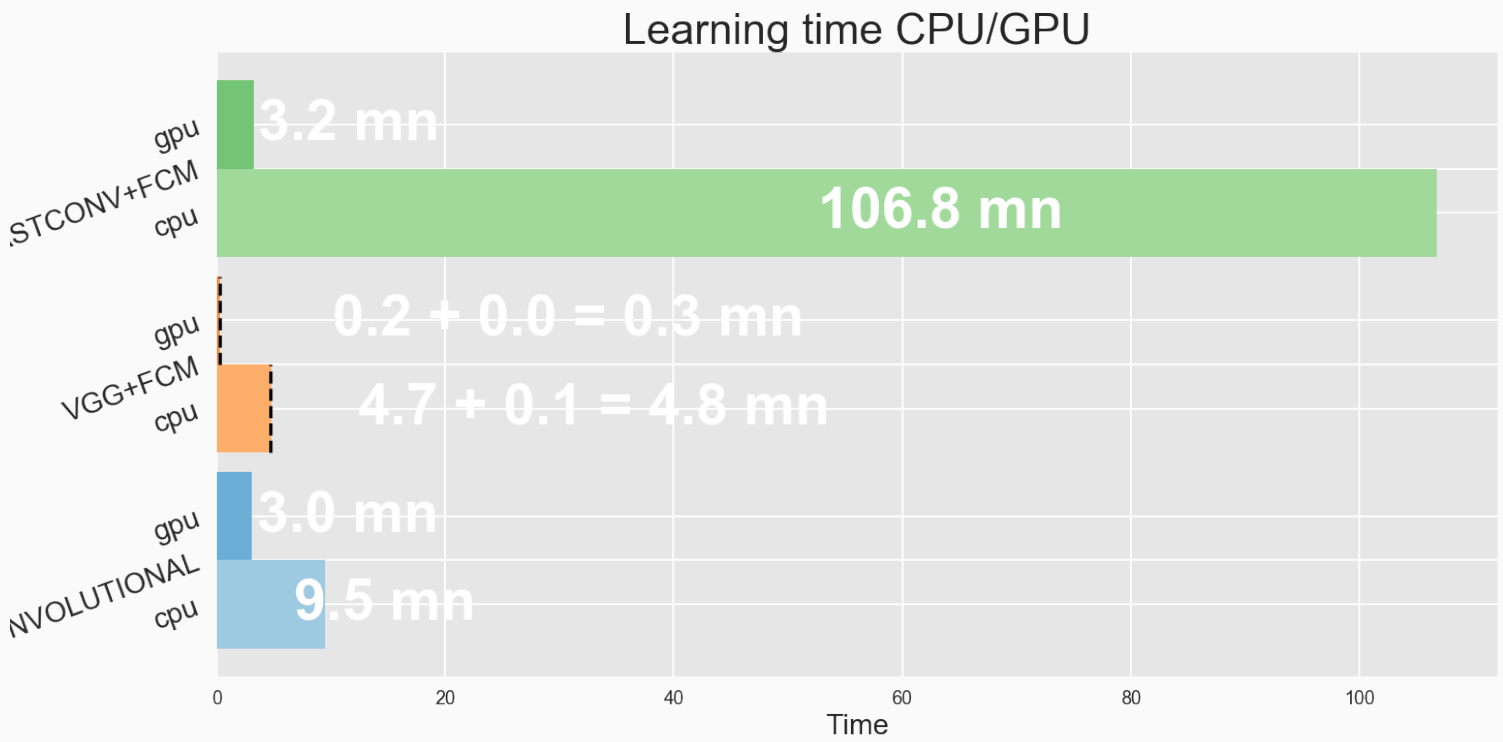


Dans la partie "Extraction de Features", c'est le calcul des features qui prend du temps !

Le modèle, très simple, ne prend pas beaucoup de temps sur le CPU.

CatsVSDgogs - Temps d'apprentissage - CPU/GPU

batch_size = 200, epochs = 15



Dans la partie "Extraction de Features", c'est le calcul des features qui prend du temps !

Le modèle, très simple, ne prend pas beaucoup de temps sur le CPU.

Références

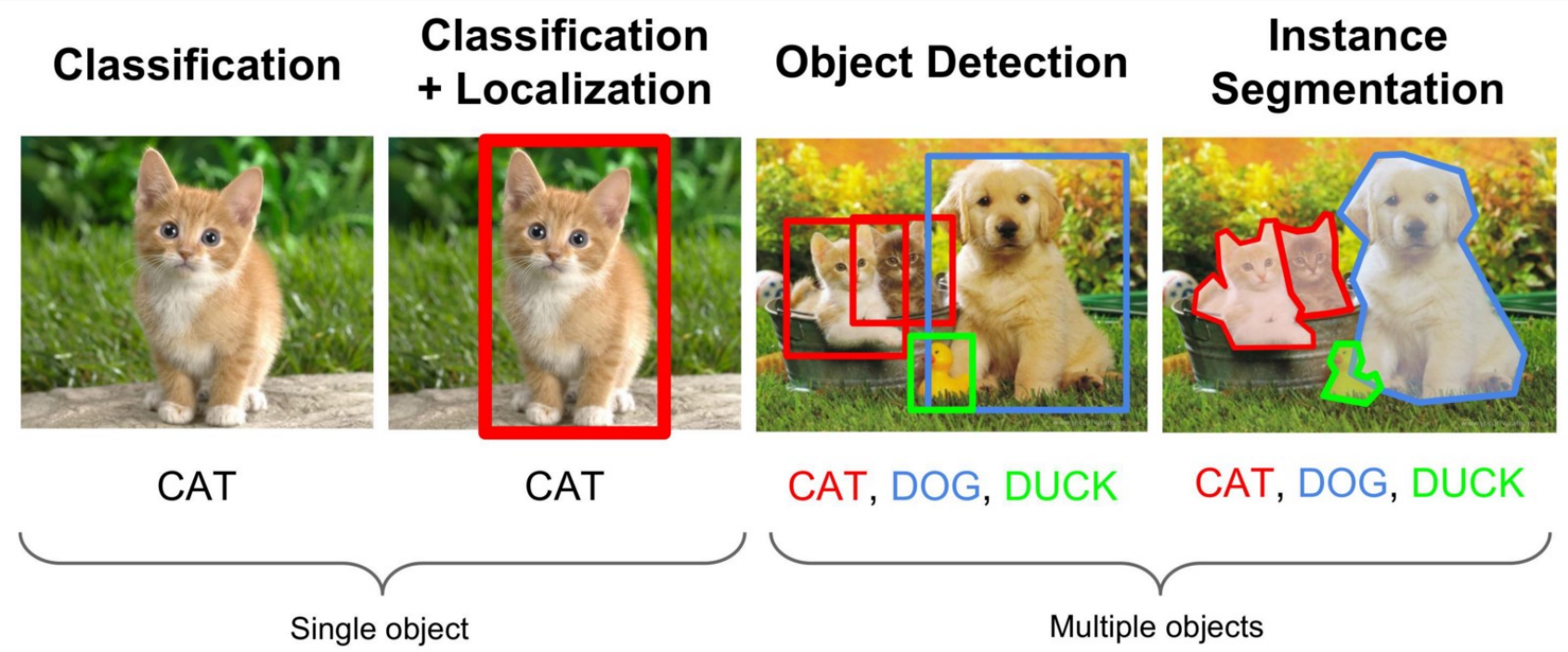
- https ://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/
- http ://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture8.pdf
- https ://project.inria.fr/deeplearning/files/2016/05/DLFrameworks.pdf
- https ://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicite/
- https ://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Autres application

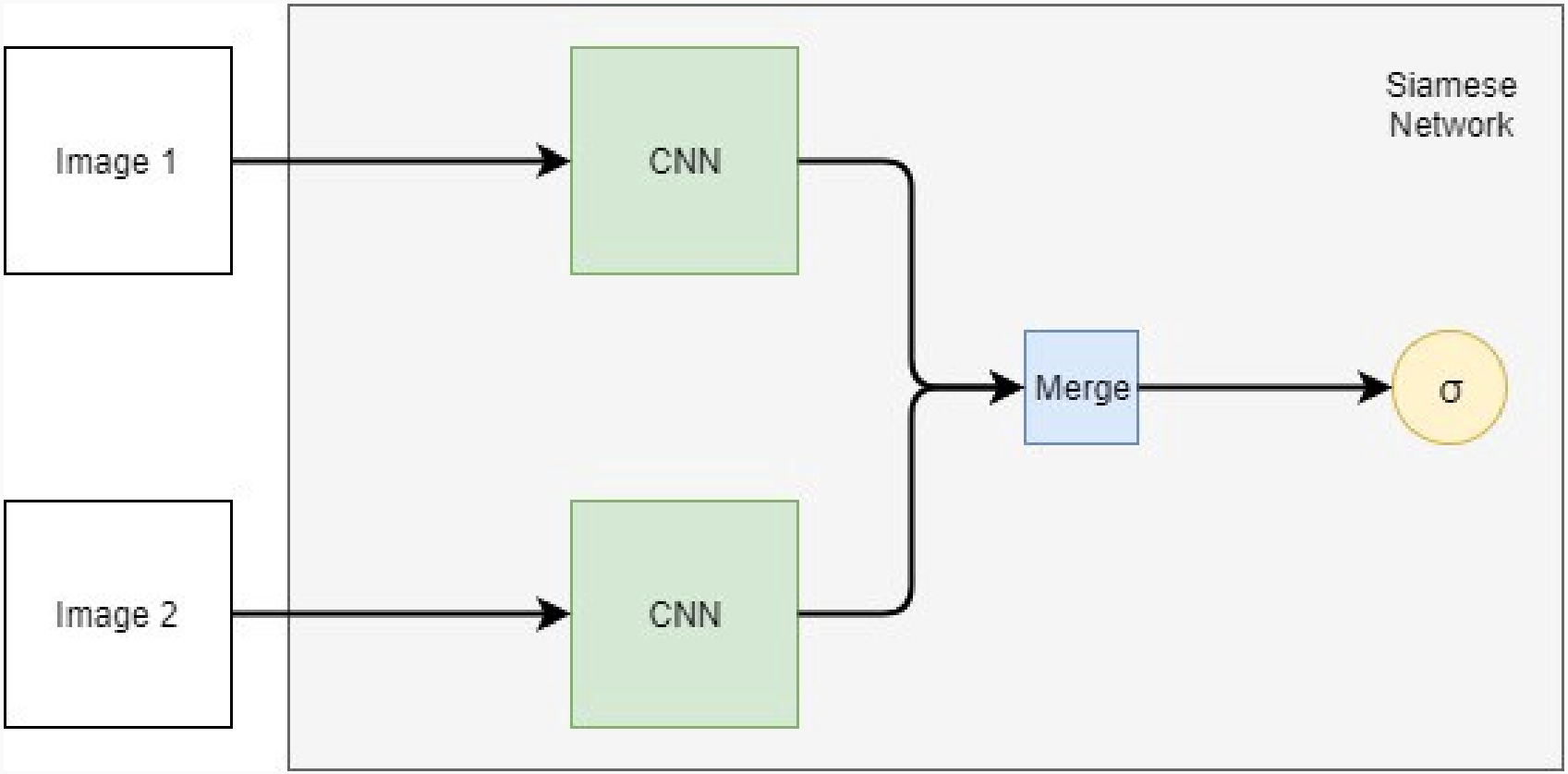
Localisation and Segmentation

Localisation : YOLO, SSD, Fast-RCNN

Segmentation.



Siamese Network - OneShotLearning



Références

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv :1605.07678*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

References II

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv :1312.4400*, 2013.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.