

DEEP LEARNING

INTRODUCTION À LA RECONNAISSANCE D'IMAGE

Brendan Guilloyet

2018

Institut National des Sciences Appliquées

CONVOLUTIONAL NEURAL NETWORK

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.



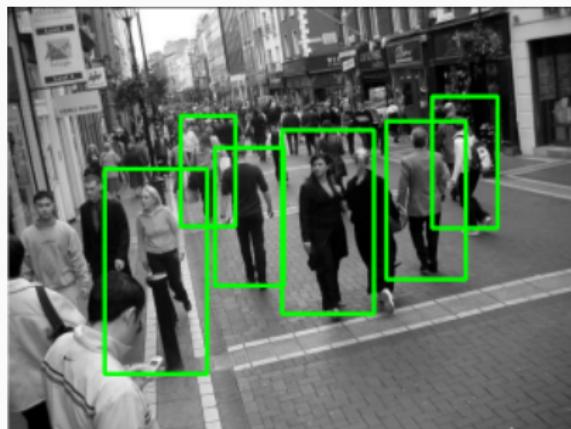
Your specialized huggable recommendation

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

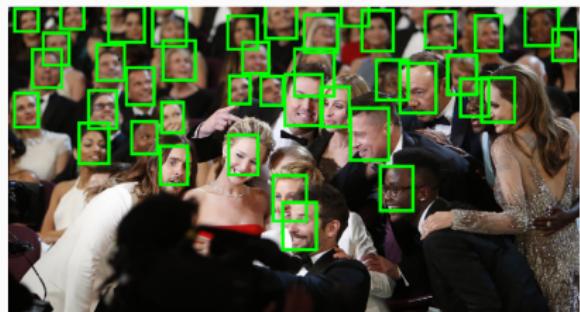


APPLICATION - RECONNAISSANCE DE PATTERNS

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

⇒ CONVOLUTIONAL 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

*

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

*

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

6×6

*

Convolution
Filter

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

3×3

=

4×4

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

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

*

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

*

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

*

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

*

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

*

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



*

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



*

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



*

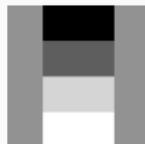
1	0	-1
1	0	-1
1	0	-1

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 :

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

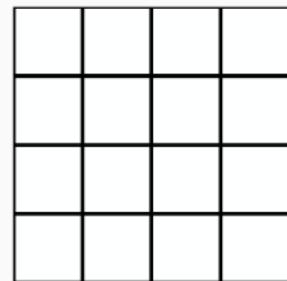
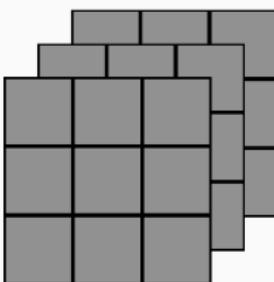
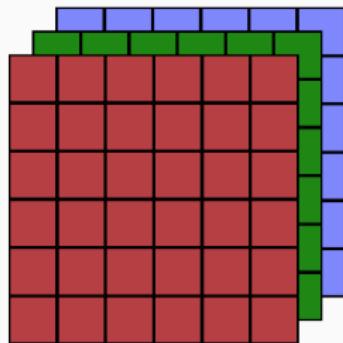
DISCRETE 2D-CROSS-CORRELATION (OFTEN USED!) :

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

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

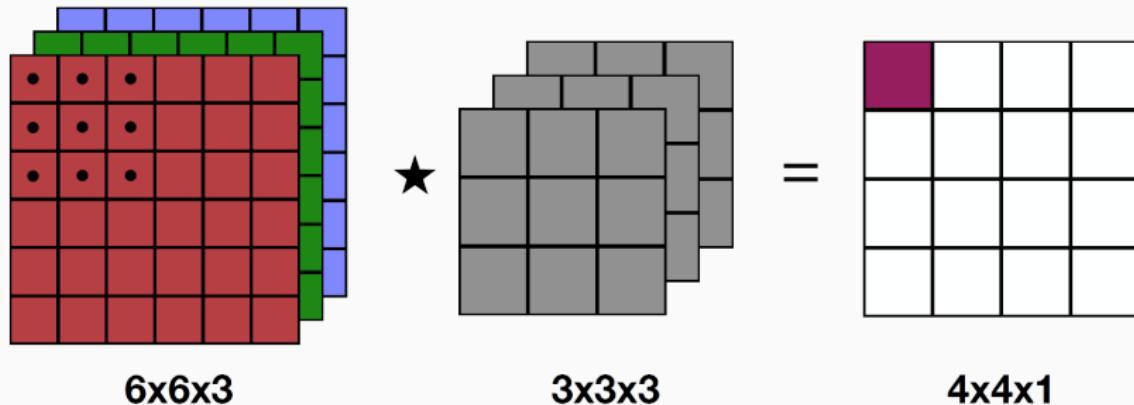
CONVOLUTION - CHANNELS

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



CONVOLUTION - CHANNELS

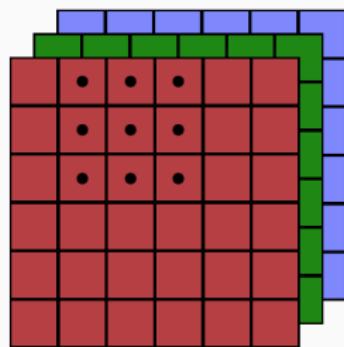
Colored image in 3 dimensions : (height, width, channels (RGB))



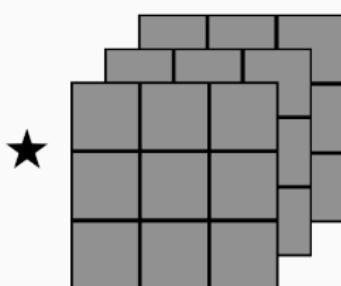
$$(F \star img)(x, y) = \sum_c \sum_m \sum_n F^c(n, m) \cdot img^c(x + m, y + n)$$

CONVOLUTION - CHANNELS

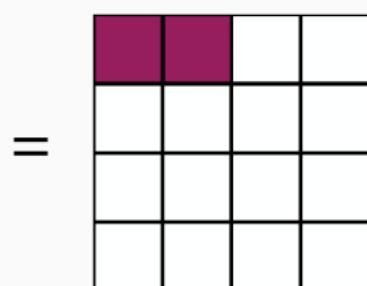
Colored image in 3 dimensions : (height, width, channels (RGB))



6x6x3



3x3x3

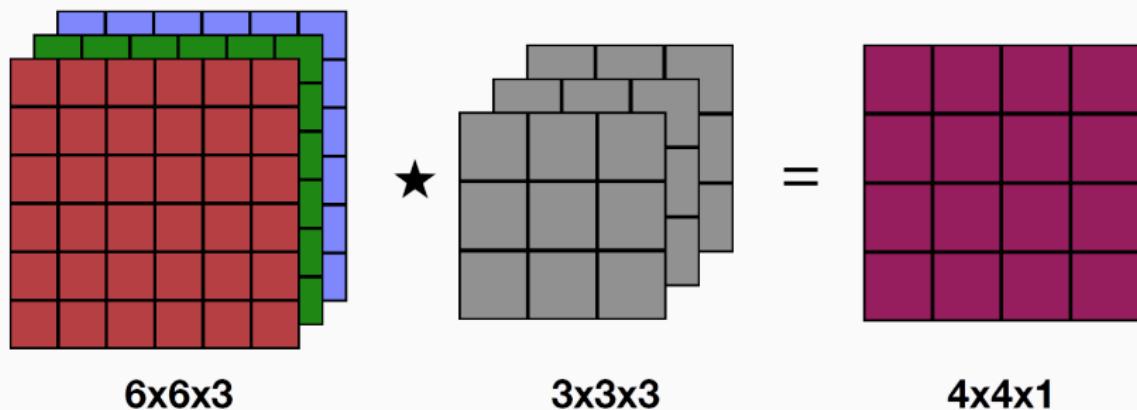


4x4x1

$$(F \star img)(x, y) = \sum_c \sum_m \sum_n F^c(n, m) \cdot img^c(x + m, y + n)$$

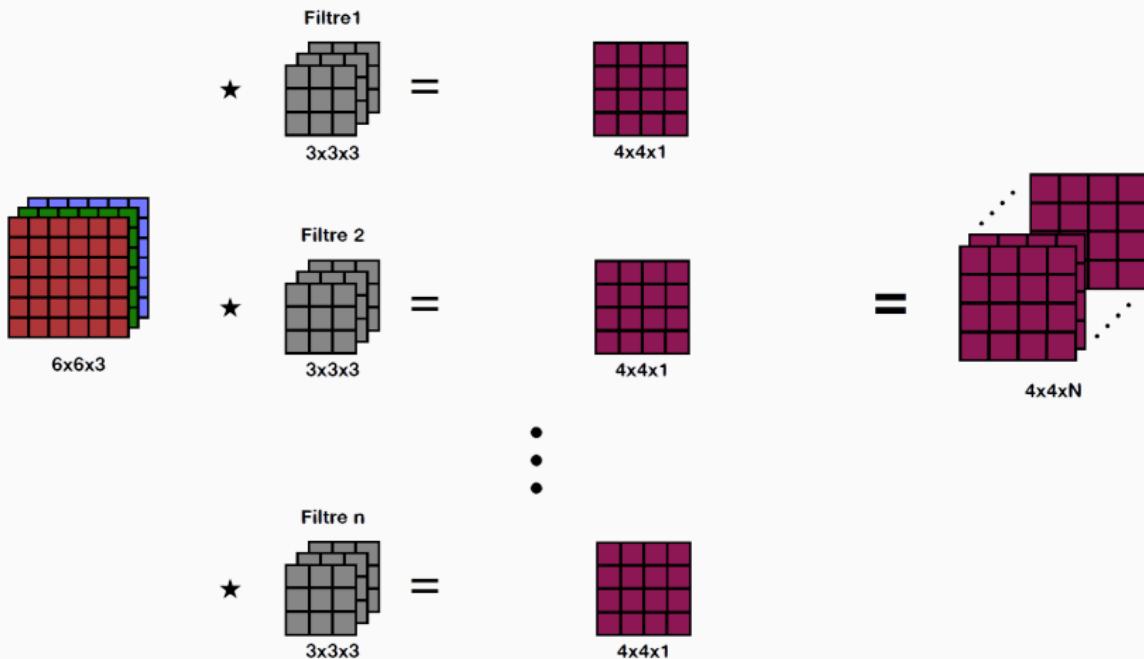
CONVOLUTION - CHANNELS

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



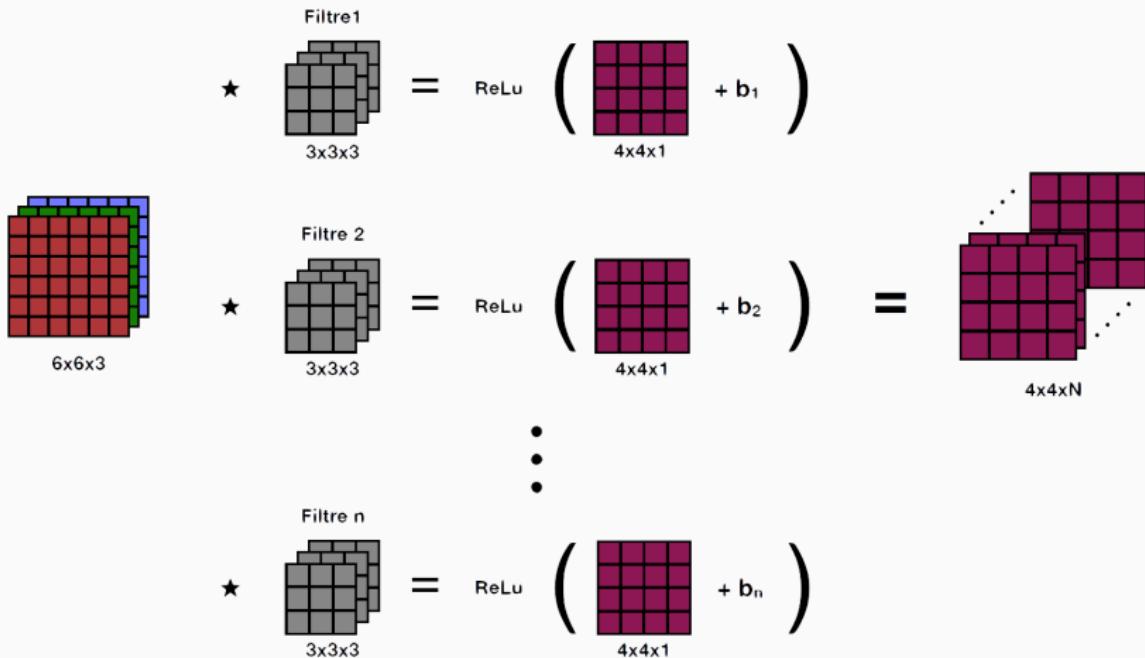
$$(F * img)(x, y) = \sum_c \sum_m \sum_n F^c(n, m) \cdot img^c(x + m, y + n)$$

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

*

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

*

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

$7 \times 7 \times 1$

*

3	4	4
1	0	2
-1	0	3

$3 \times 3 \times 1$

91	100	

$3 \times 3 \times 1$

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

$7 \times 7 \times 1$

*

3	4	4
1	0	2
-1	0	3

$3 \times 3 \times 1$

91	100	83
69		

$3 \times 3 \times 1$

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

*

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

$$\star \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & 7 \\ \hline -3 & -2 & -3 & 16 \\ \hline \end{array}$$

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

$$\begin{array}{c} \star \\ \text{3 X 3} \end{array} = \begin{array}{c} \begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & 7 \\ -3 & -2 & -3 & 16 \end{matrix} \\ \text{4 X 4} \end{array}$$

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

*

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

$$\begin{array}{c} \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} & \star & \begin{array}{|c|c|c|}\hline -5 & -5 & -6 \\ \hline -12 & -5 & -4 \\ \hline -13 & -10 & -1 \\ \hline -10 & 0 & -2 \\ \hline -7 & -3 & -2 \\ \hline -6 & 0 & -2 \\ \hline \end{array} & = & \begin{array}{|c|c|c|}\hline 6 & 10 & \\ \hline 8 & 11 & \\ \hline 3 & 11 & \\ \hline 7 & 10 & \\ \hline 16 & 12 & \\ \hline -9 & 5 & \\ \hline \end{array} \end{array}$$

3 X 3

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 : $N_p = ((h \cdot w \cdot c) + 1) \cdot n$

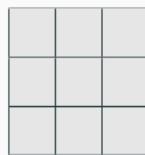
POOLING

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

6×6

Convolution
Filter

Pool



3×3

=

4×4

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

6×6

Convolution
Filter
Pool 3×3

=

4×4

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

6×6

Convolution
Filter
Pool 3×3

=

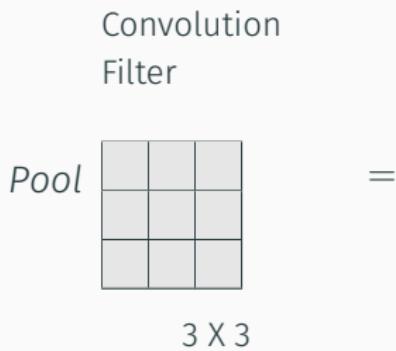
4×4

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



8			

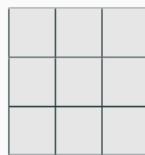
POOLING

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

6×6

Convolution
Filter

Pool



3×3

=

8			

4×4

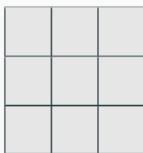
POOLING

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

6×6

Convolution
Filter

Pool 3×3



=

8			

4×4

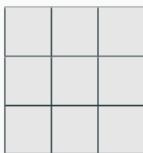
POOLING

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

6×6

Convolution
Filter

Pool



3×3

=

8	9		

4×4

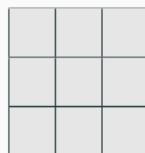
POOLING

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

6×6

Convolution
Filter

Pool



3×3

=

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

4×4

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

6×6

Convolution
Filter

Pool

3×3

4×4

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

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

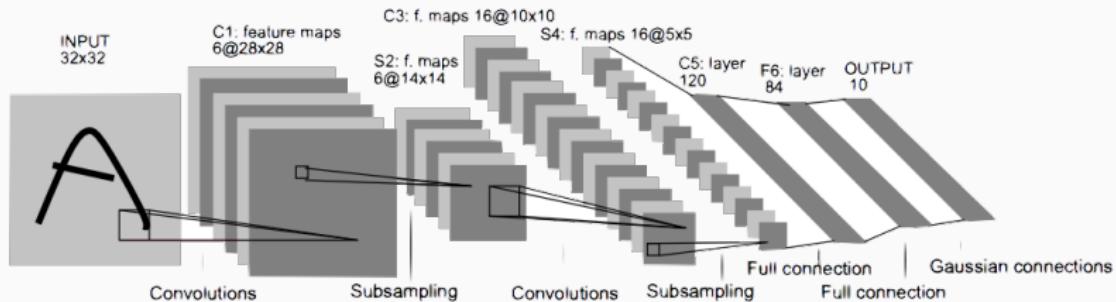
LENET-5 AND MNIST CLASSIFICATION

OBJECTIF :



- Manuscript 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

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

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 :

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

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

CATEGORICAL CROSS ENTROPY :

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

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

ALEX NET (Krizhevsky et al. [2012])

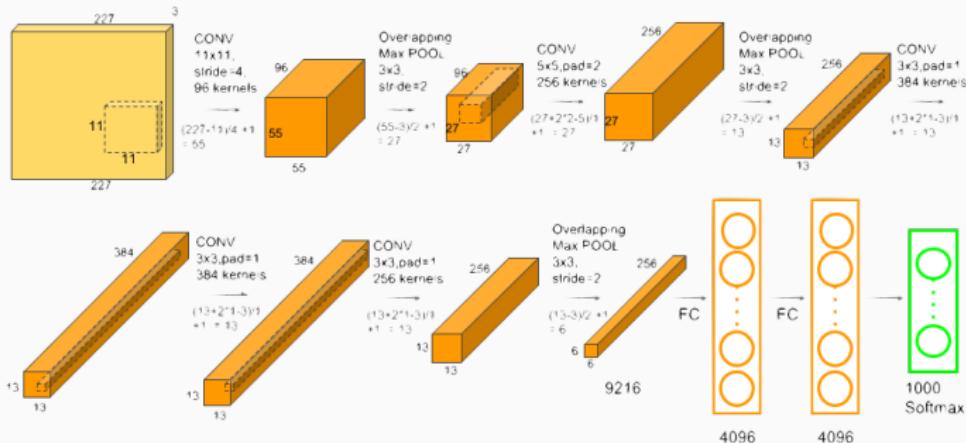


Image source : <https://www.learnopencv.com/understanding-alexnet/>

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

<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

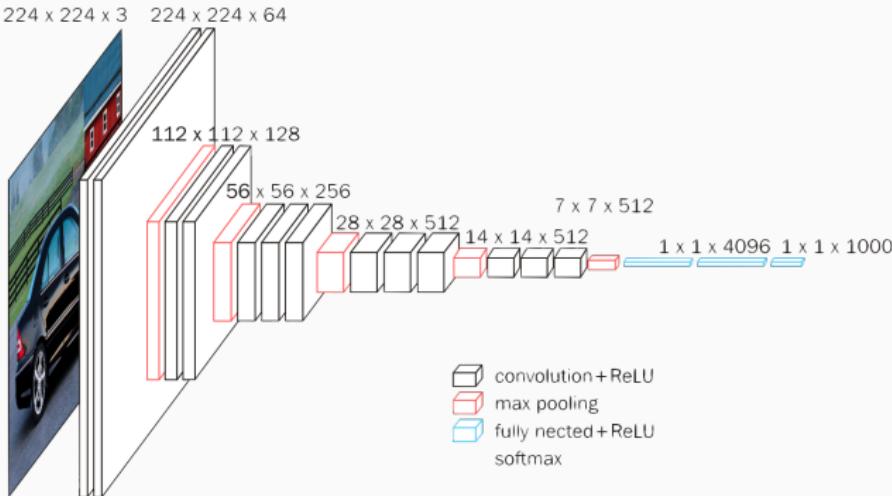


Image source : <https://blog.datawow.io/cnn-models-ef356bc11032>, realized with Tensorflow.

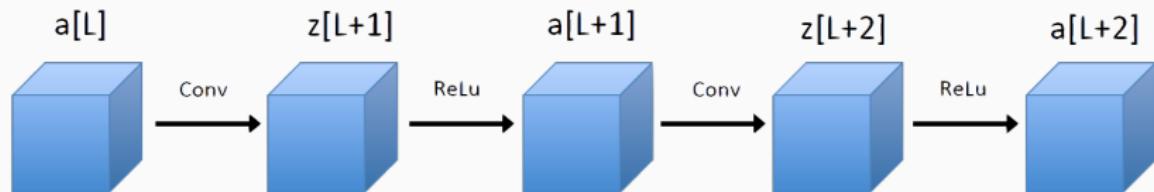
138M parameters.

HIERARCHICAL REPRESENTATION

<https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>

RESNET BLOCK - (HE ET AL. [2016])

Convolution with padding = SAME



$$z[l + 1] = \text{Conv}(a[l])$$

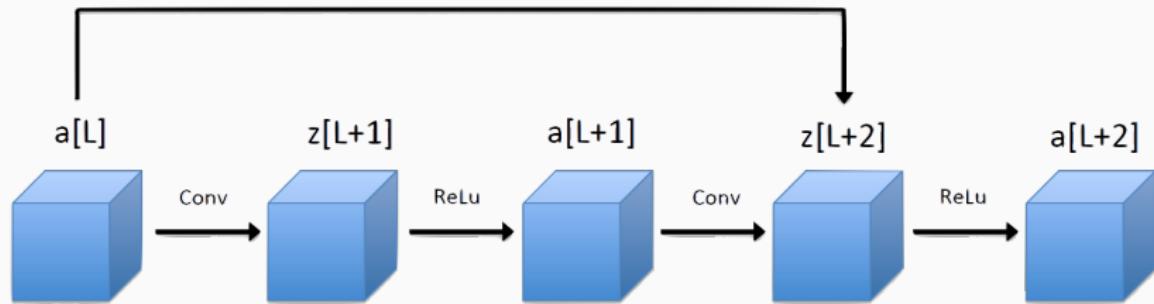
$$a[l + 1] = \text{ReLU}(z[l + 1])$$

$$z[l + 2] = \text{Conv}(a[l + 1])$$

$$a[l + 2] = \text{ReLU}(z[l + 2]) \quad)$$

RESNET BLOCK - (HE ET AL. [2016])

Convolution with padding = SAME



$$z[l + 1] = \text{Conv}(a[l])$$

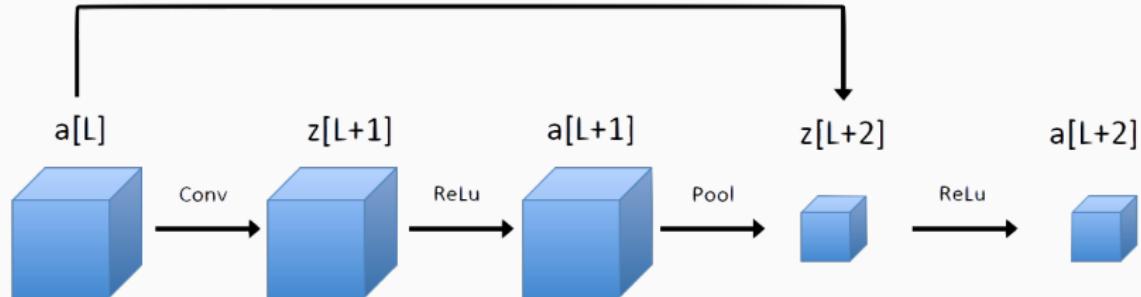
$$a[l + 1] = \text{ReLU}(z[l + 1])$$

$$z[l + 2] = \text{Conv}(a[l + 1])$$

$$a[l + 2] = \text{ReLU}(z[l + 2] + a[l])$$

RESNET BLOCK - (HE ET AL. [2016])

MaxPooling



$$z[l + 1] = \text{Conv}(a[l])$$

$$a[l + 1] = \text{ReLU}(z[l + 1])$$

$$z[l + 2] = \text{Conv}(a[l + 1])$$

$$a[l + 2] = \text{ReLU}(z[l + 2] + W \cdot a[l])$$

RESNET BLOCK - (HE ET AL. [2016]) - PROPERTIES

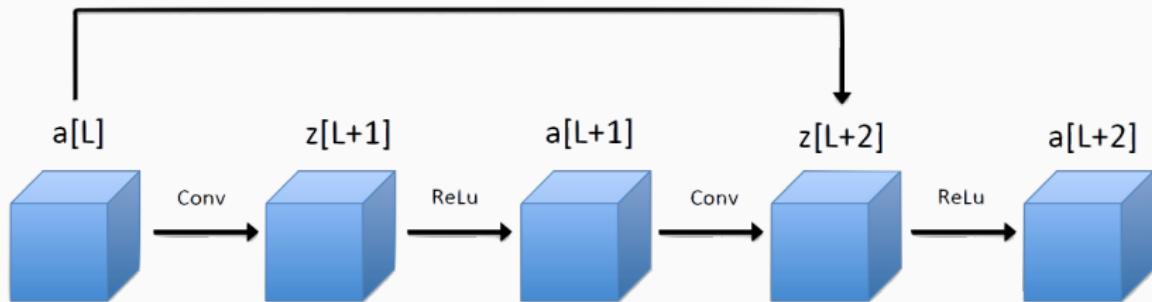
Making a convolutional network too deep can hurt its performance.

RESNET BLOCK - (HE ET AL. [2016]) - PROPERTIES

Making a convolutional network too deep can hurt its performance.

With Resnet, you can make your network deeper!!

Convolution with padding = SAME

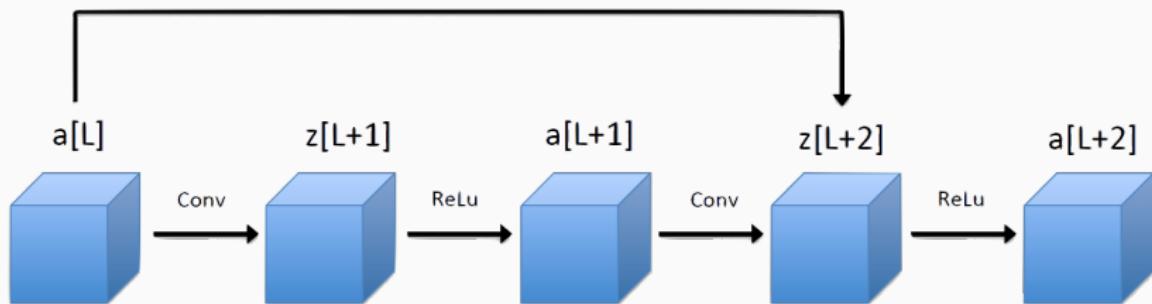


RESNET BLOCK - (HE ET AL. [2016]) - PROPERTIES

Making a convolutional network too deep can hurt its performance.

With Resnet, you can make your network deeper!!

Convolution with padding = SAME



$$z[l + 1] = \text{Conv}(a[l])$$

$$a[l + 1] = \text{ReLU}(z[l + 1])$$

$$z[l + 2] = \text{Conv}(a[l + 1])$$

$$a[l + 2] = \text{ReLU}(z[l + 2] + a[l])$$

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

Convolution with 3x3 filters and padding = SAME

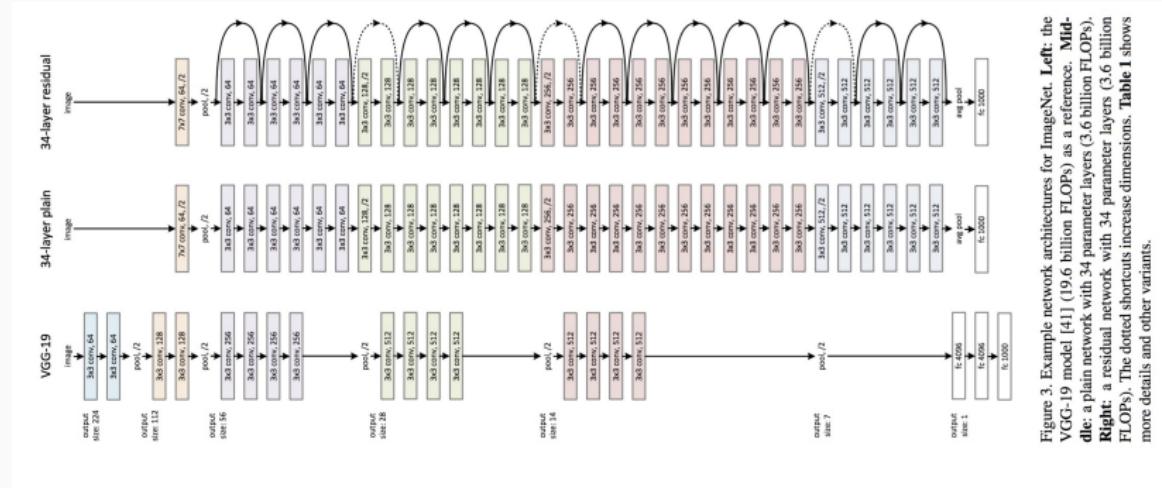
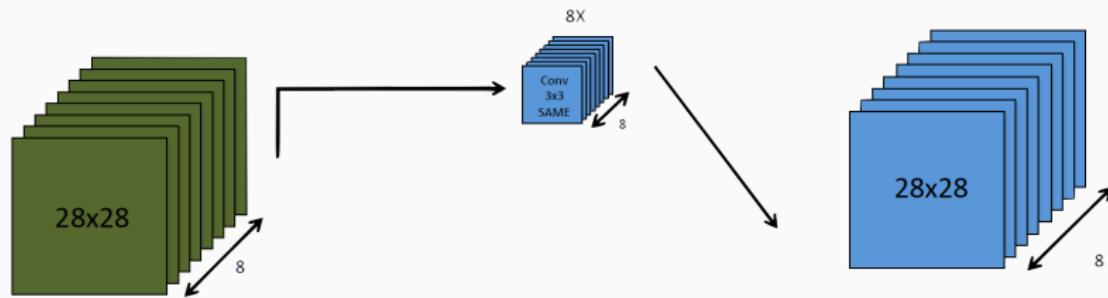
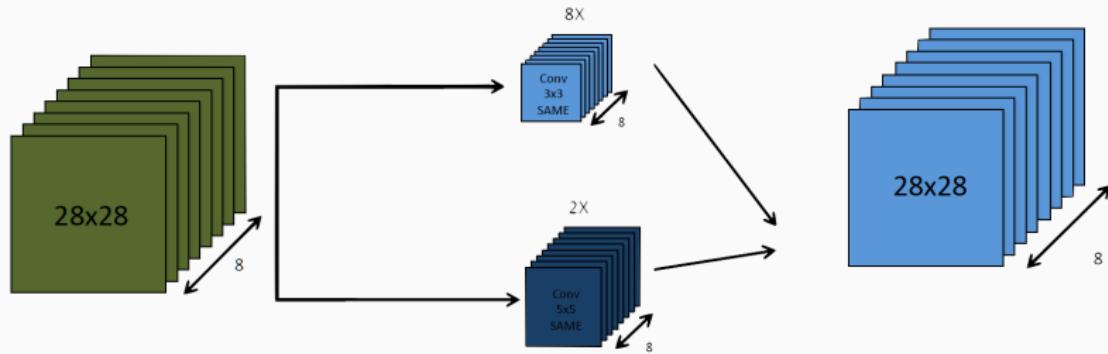


Figure 3. Example network architectures for ImageNet. **Left:** the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle:** a plain network with 34 parameter layers (3.6 billion FLOPs). **Right:** a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase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

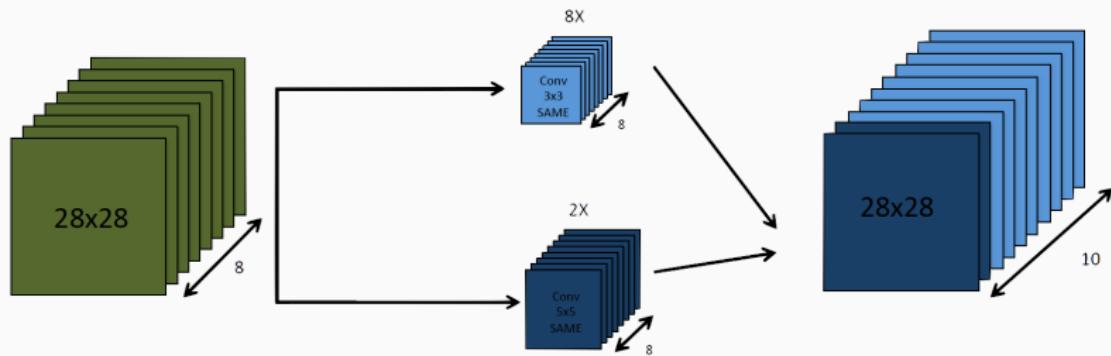
INCEPTION BLOCK - SZEGEDY ET AL. [2015]



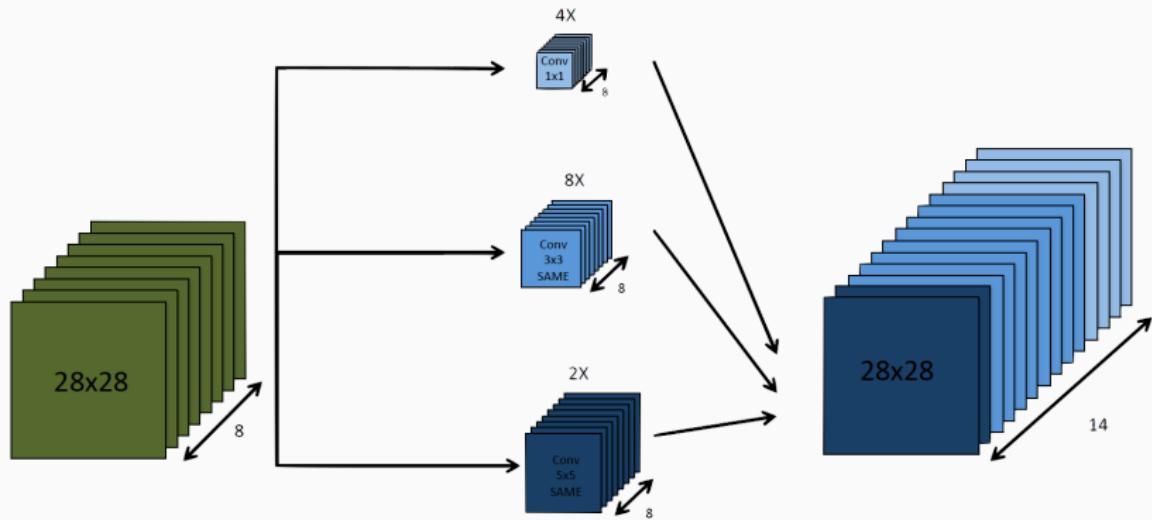
INCEPTION BLOCK - SZEGEDY ET AL. [2015]



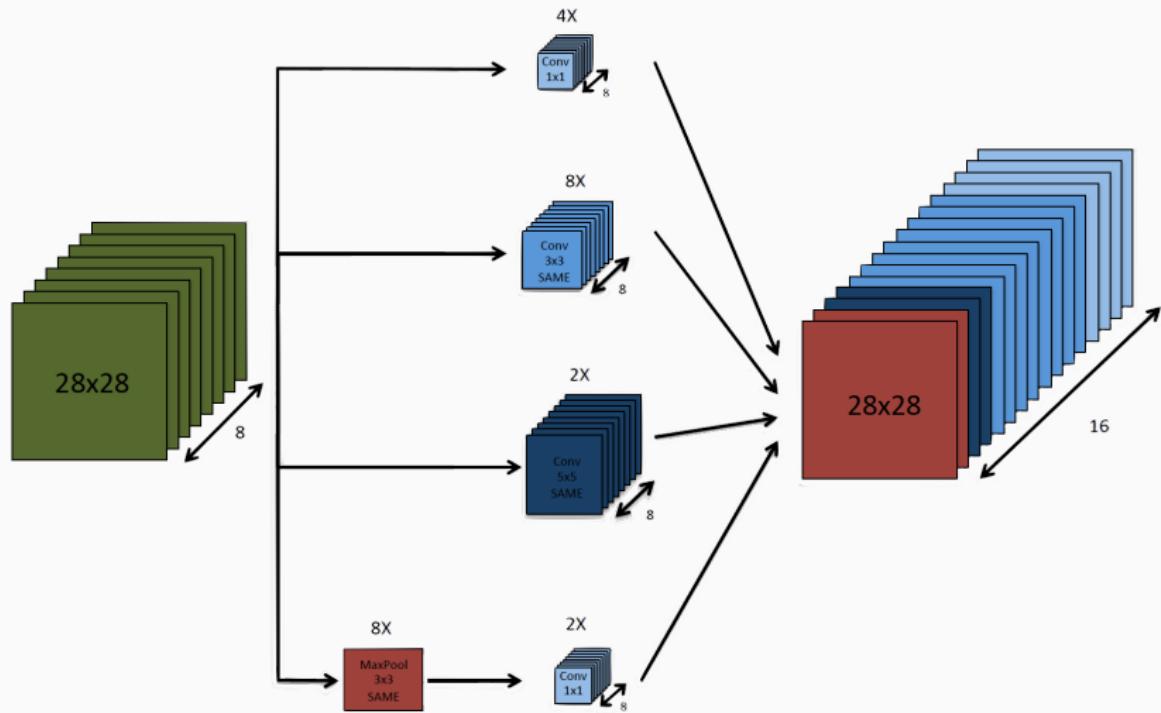
INCEPTION BLOCK - SZEGEDY ET AL. [2015]



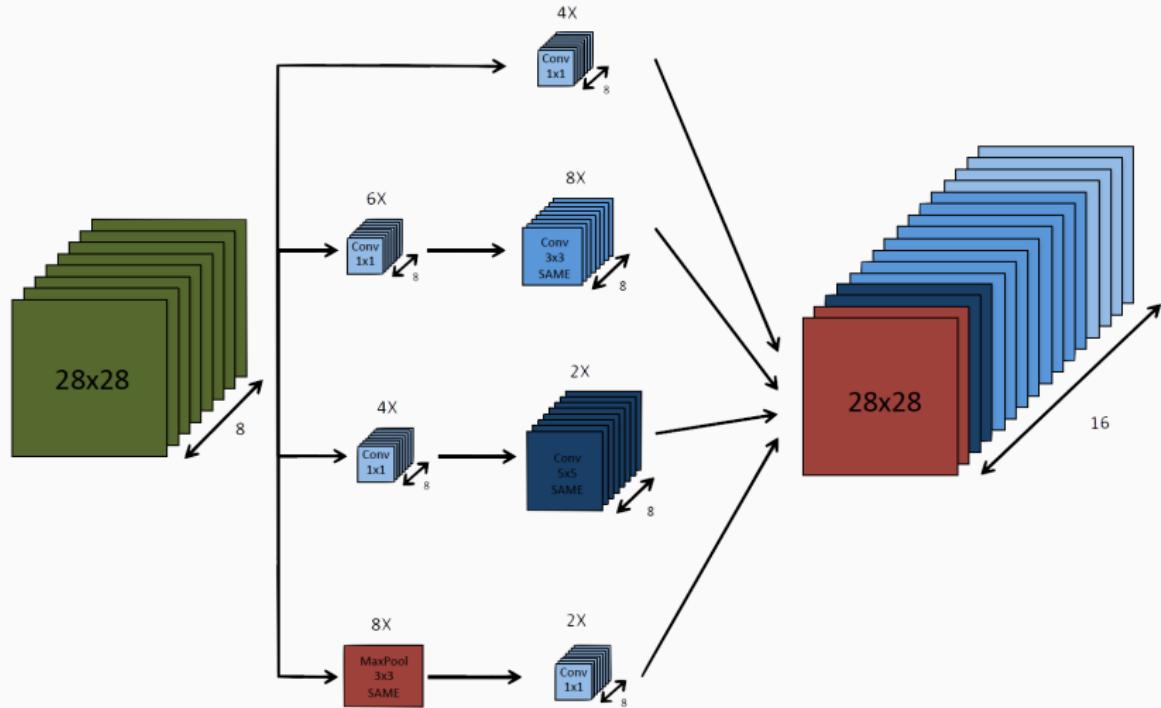
INCEPTION BLOCK - SZEGEDY ET AL. [2015]



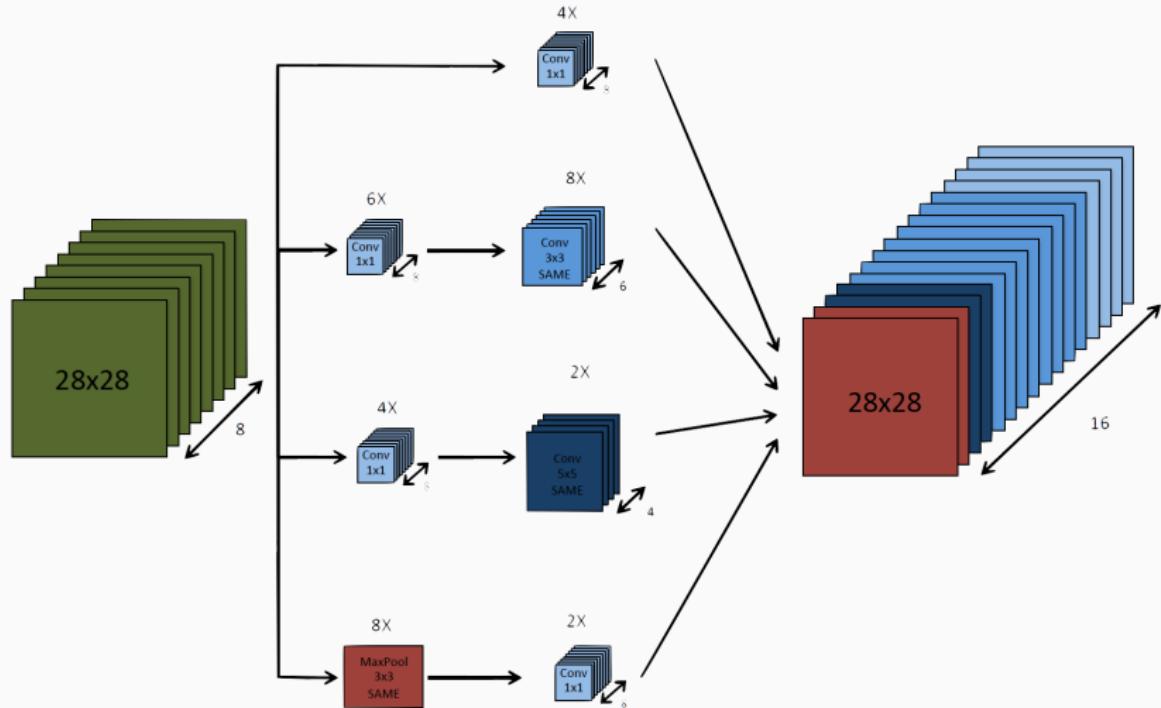
INCEPTION BLOCK - SZEGEDY ET AL. [2015]



INCEPTION BLOCK - SZEGEDY ET AL. [2015]

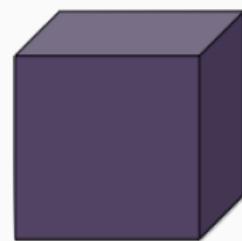
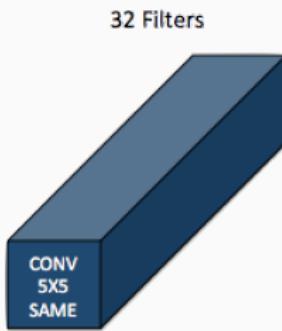
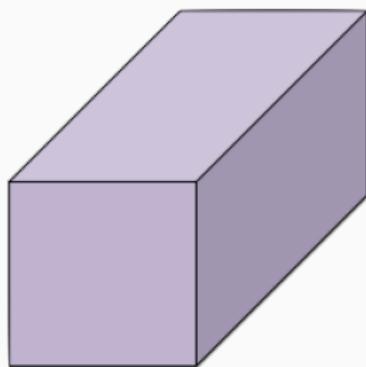


INCEPTION BLOCK - SZEGEDY ET AL. [2015]



1 X 1 CONVOLUTION - PROBLEM IN COMPUTATIONAL COST

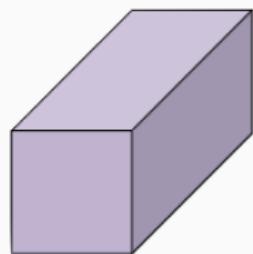
Network in Network Lin et al. [2013]



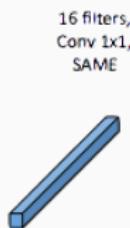
- Parameters : $5 \times 5 \times 192 \times 32 = 153.600$
- Operations : $153.600 \times 28 \times 28 = 120.422.400$

1 X 1 CONVOLUTION - PROBLEM IN COMPUTATIONAL COST

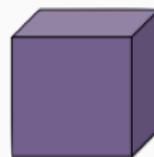
Network in Network Lin et al. [2013]



$28 \times 28 \times 192$



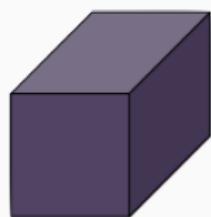
$1 \times 1 \times 192$



$28 \times 28 \times 16$



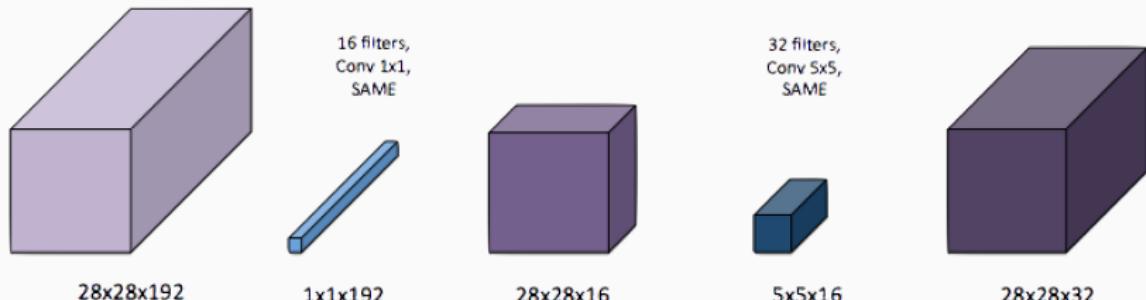
$5 \times 5 \times 16$



$28 \times 28 \times 32$

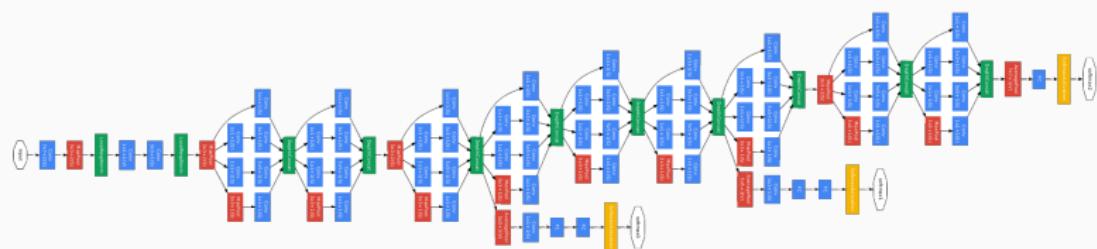
1 X 1 CONVOLUTION - PROBLEM IN COMPUTATIONAL COST

Network in Network Lin et al. [2013]

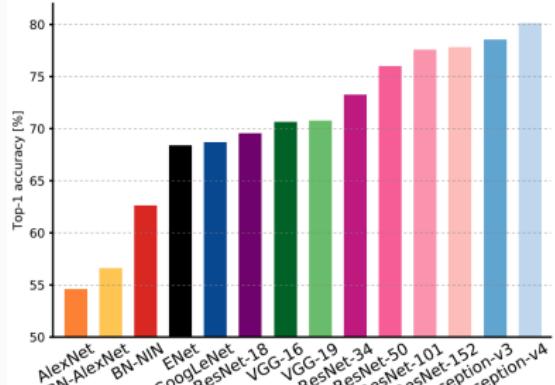


- Parameters : $1 \times 1 \times 192 \times 16, 5 \times 5 \times 16 \times 32 = (3072, 12800)$
- Operations : $3.072 \times 28 \times 28 + 12.800 = 2.408.448 + 10.035.200 = 12.443.648$

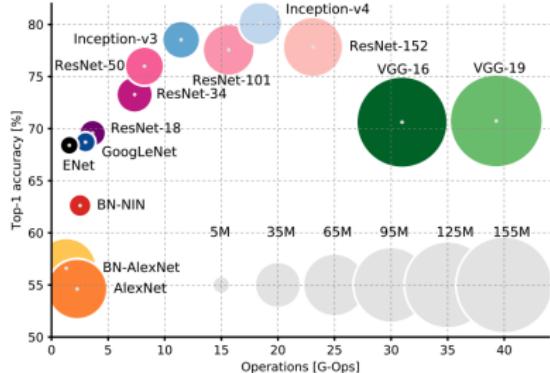
INCEPTION MODEL - SZEGEDY ET AL. [2015]



STATE OF THE ART



source : Canziani et al. [2016]



Image

- In constant evolution,
- new architecture regularly proposed.

RÉSEAU DE NEURONES CONVOLUTIFS - REMARQUE

PROBLÉMATIQUE pour des applications industrielles :

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

SOLUTION : Le *transfert learning*.

Utiliser des réseaux de convolutions déjà entraînés.

PRE-TRAINED MODEL

(labs)

TRANSFER LEARNING

(labs)

FINE TUNING

(labs)

DATA AUGMENTATION

(labs)

CLASSIFICATION D'IMAGE AVEC KERAS

Deux cas d'études :

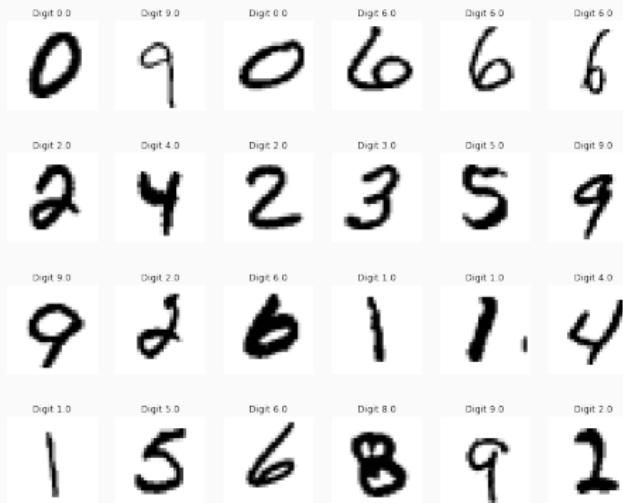
- Problème *simple* : **MNIST**.
 - Comparaison avec les autres algorithmes étudiés (*random forest*).
 - Performance du GPU.
- Problème plus *complexe* : **Cats VS Dogs**.
 - Utilisation des modèles pré-entraînés.
 - Performance du GPU.

CLASSIFICATION D'IMAGE AVEC KERAS

MNIST

MNIST

- $28 \times 28 = 784$ 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 = "Où est le niveau de couleur"

MNIST - RÉSEAU DE CONVOLUTION

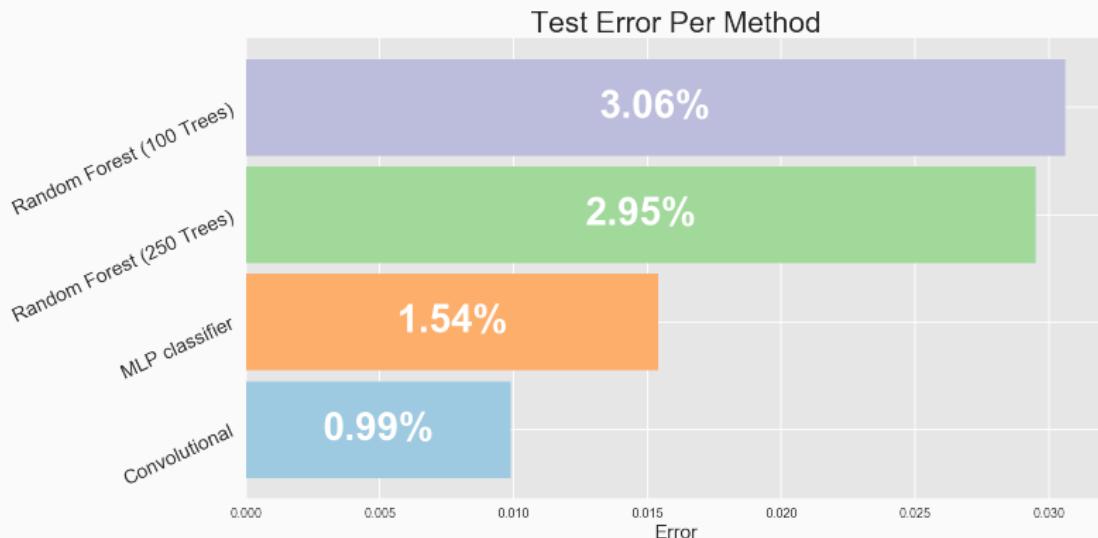
OUTPUT

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(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

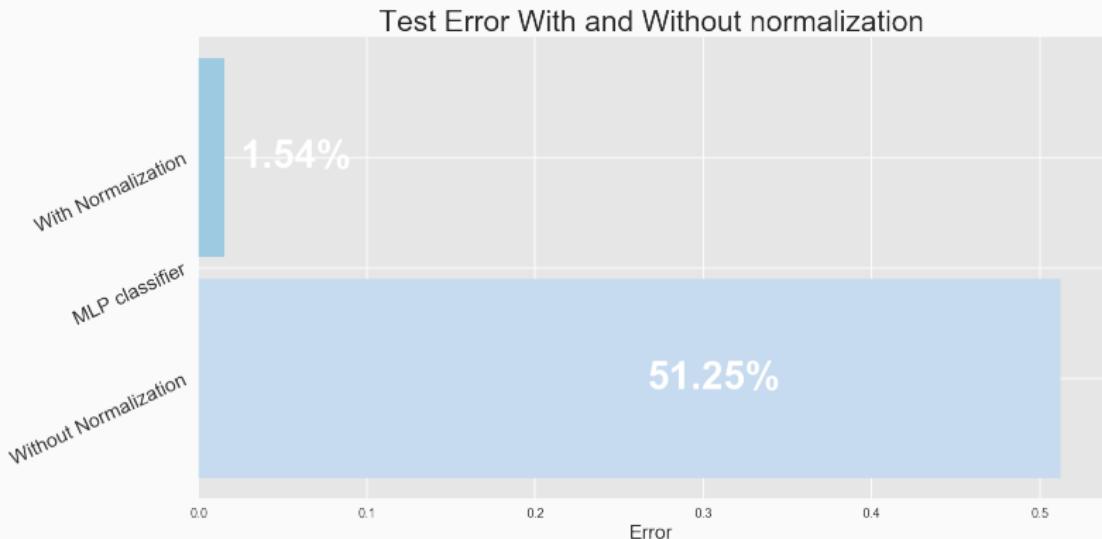
batch_size = 128, epochs = 20



Amélioration assez nette des résultats.

MNIST - ERREUR DE CLASSIFICATION - TEST - NORMALISATION

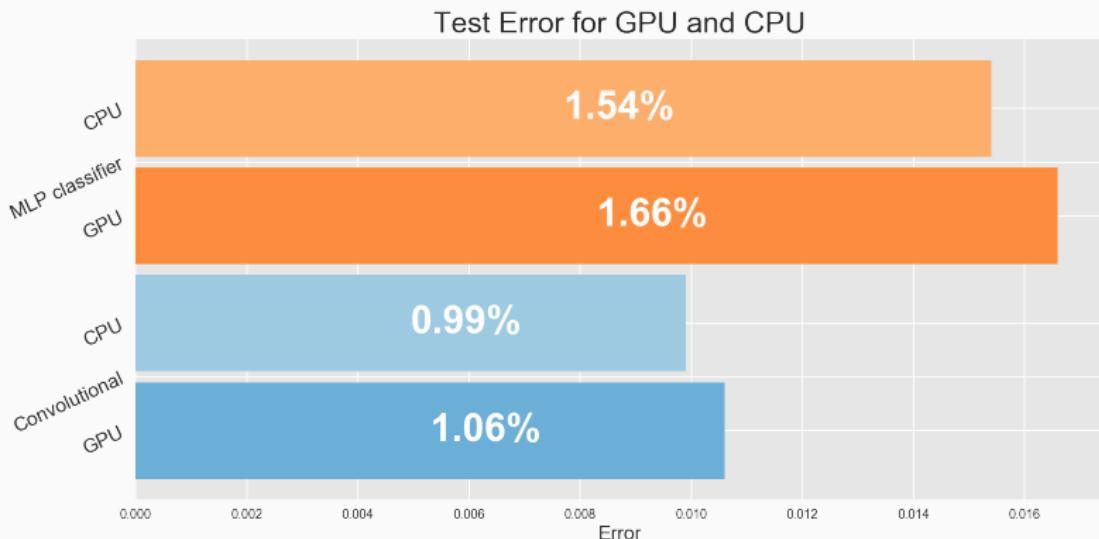
batch_size = 128, epochs = 20



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

MNIST - ERREUR DE CLASSIFICATION - TEST - CPU/GPU

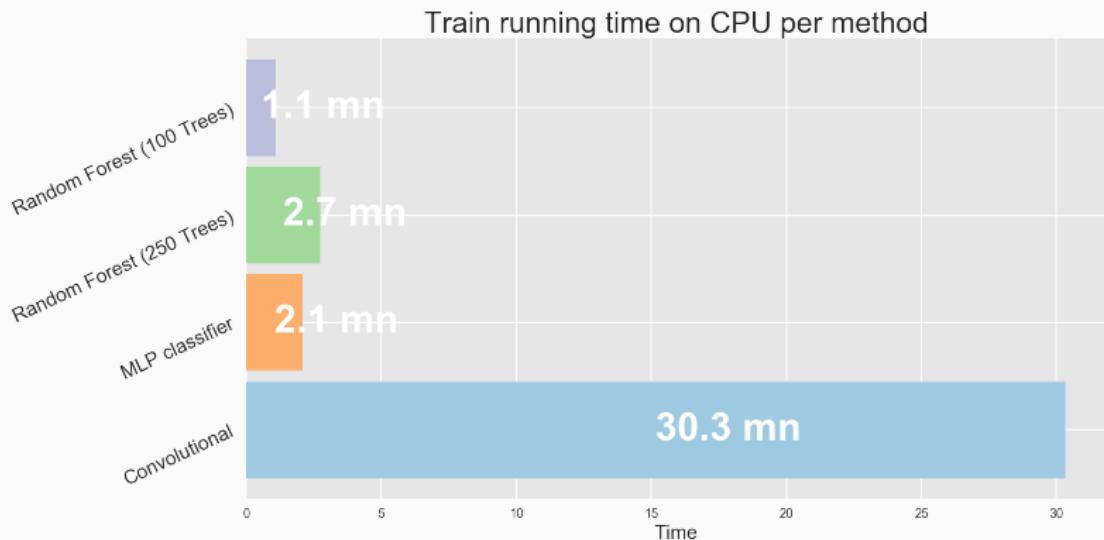
batch_size = 128, epochs = 20



Des résultats stables entre le CPU et le GPU.

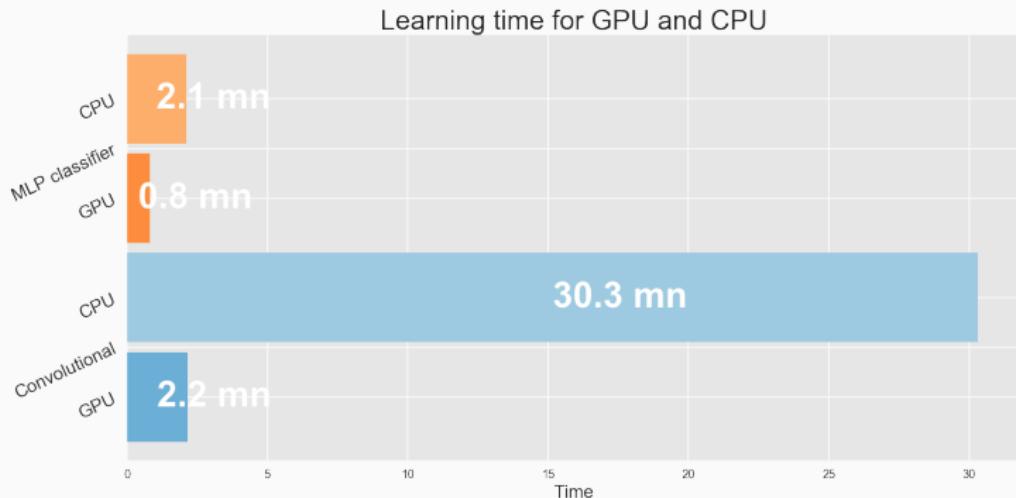
MNIST - TEMPS D'APPRENTISSAGE - CPU

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



MNIST - TEMPS D'APPRENTISSAGE - CPU/GPU

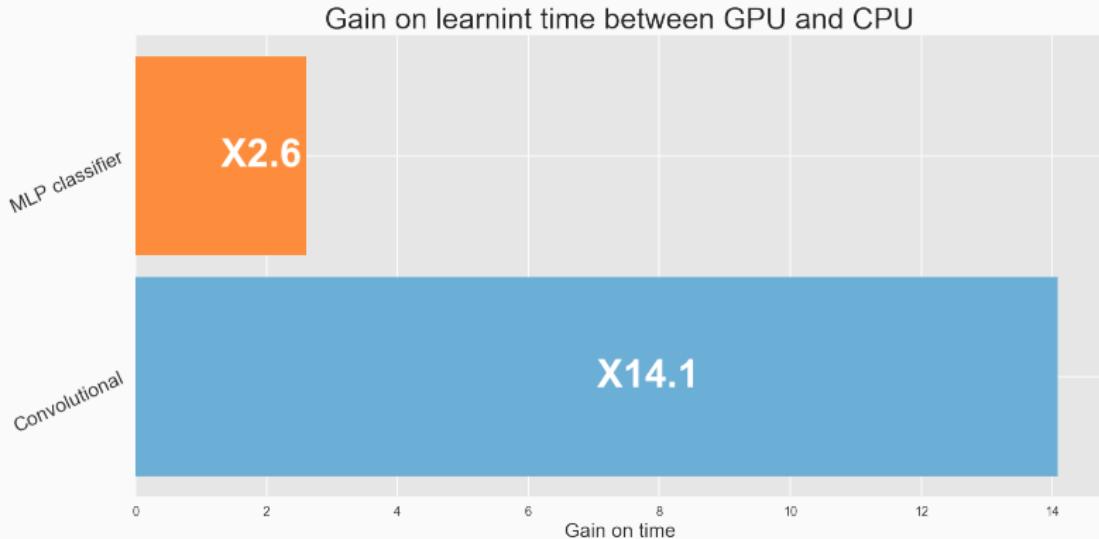
batch_size = 128, epochs = 20 - Machine GEI-103



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

MNIST - TEMPS D'APPRENTISSAGE - CPU/GPU

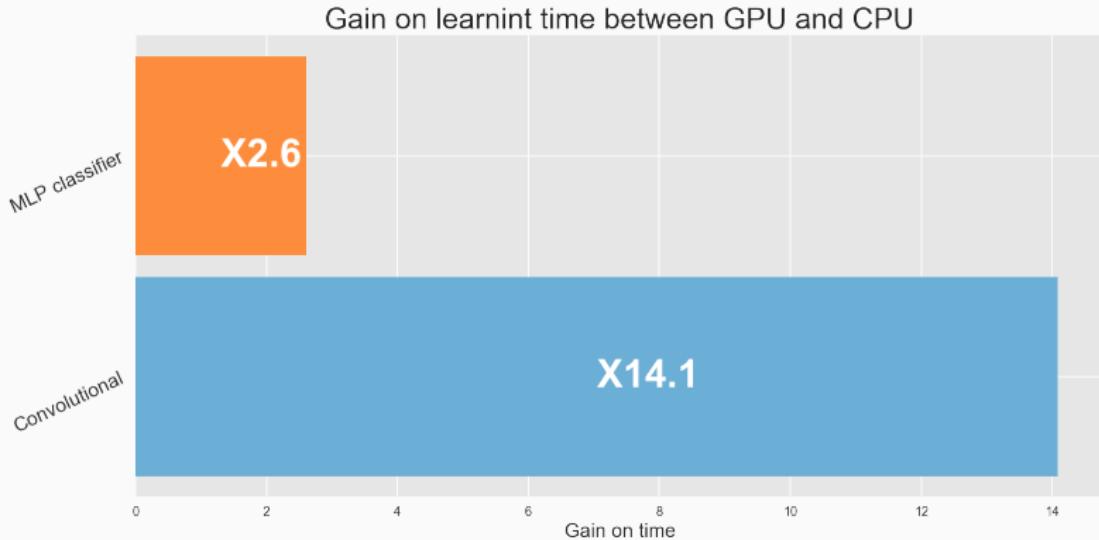
batch_size = 128, epochs = 20 - Machine GEI-103



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.

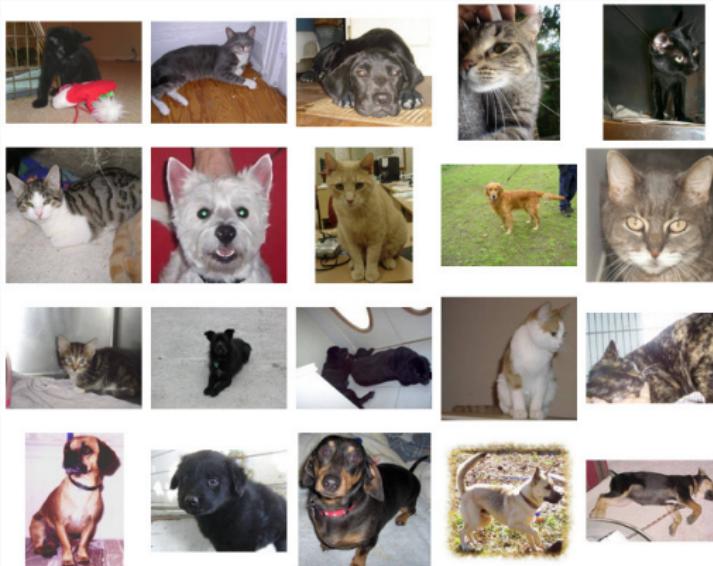
CLASSIFICATION D'IMAGE AVEC KERAS

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.

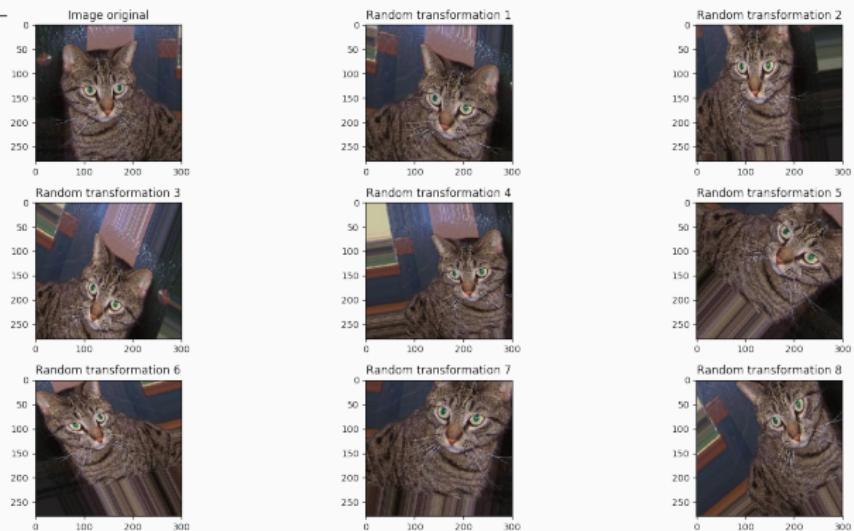
CATS VS DOGS - AUGMENTATION DES DONNÉES

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 = "Où est le niveau de couleur"

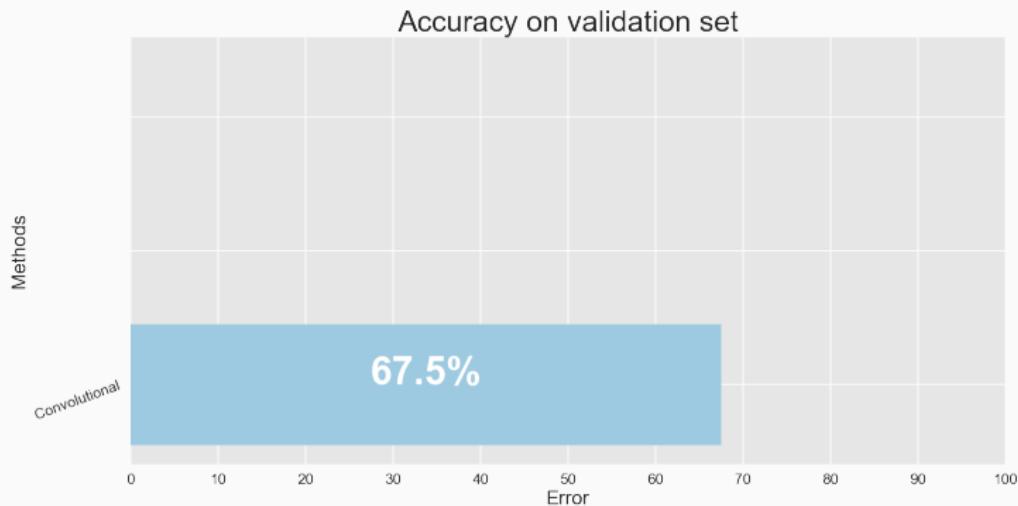
CATSVSDGOGS - RÉSEAU DE CONVOLUTION

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

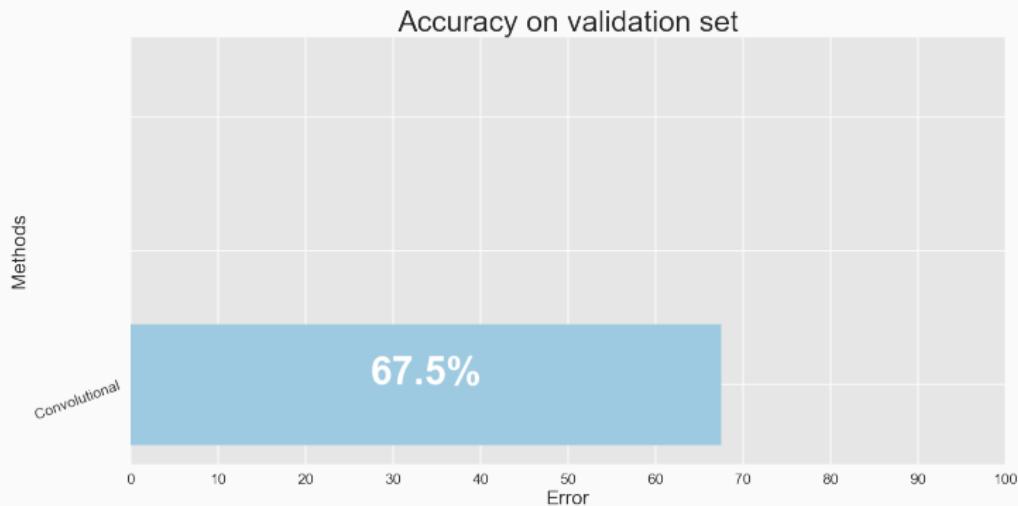
batch_size = 200, epochs = 15



Résultats assez faible.

CATSVSDGOGS - VALIDATION SCORE

batch_size = 200, epochs = 15



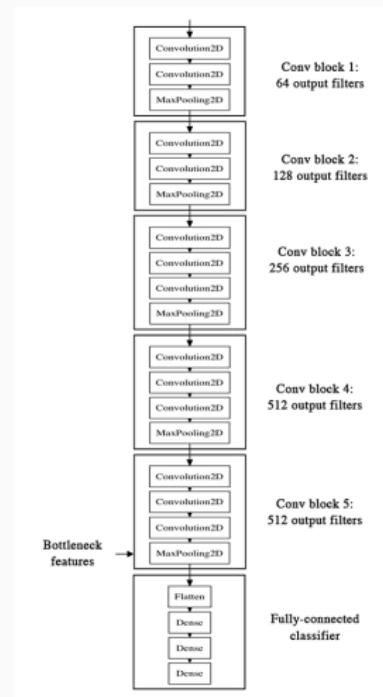
Résultats assez faible.

⇒ Utilisation d'un modèle pré-entraîné.

CATSVSDGOGS - RÉSEAU PRÉ-ENTRAÎNÉ

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

Utilisation en deux étapes :

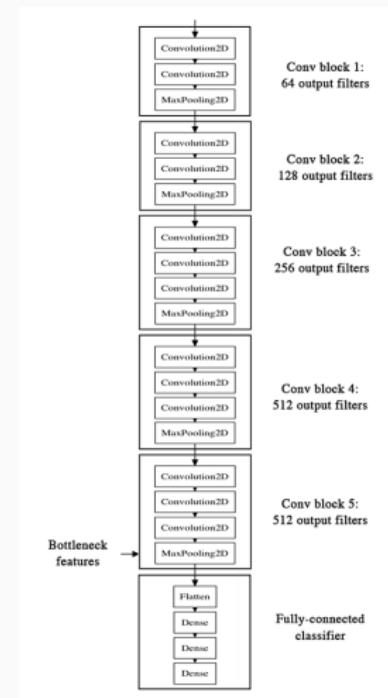


CATSVSDGOGS - RÉSEAU PRÉ-ENTRAINÉ

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

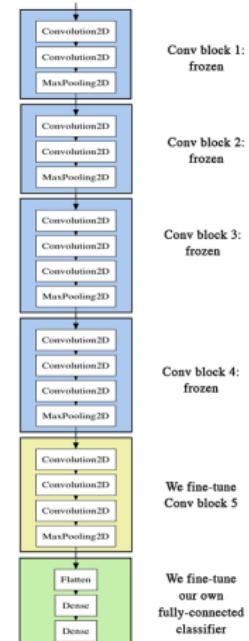


CATSVSDGOGS - RÉSEAU PRÉ-ENTRAINÉ

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 "aplatis" 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'))
```

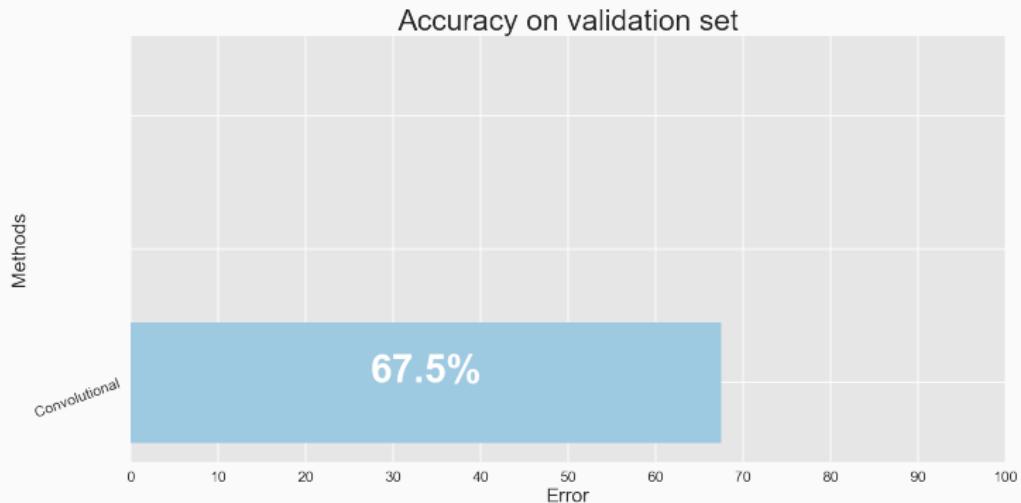
- input_shape de dimension (4, 4, 512)

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.

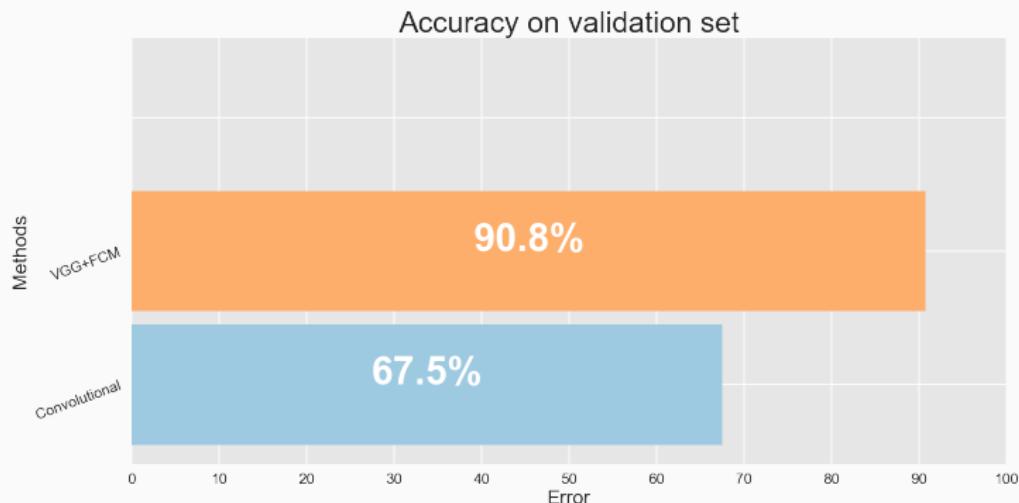
CATSVSDGOGS - VALIDATION SCORE

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 précédemment
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'))

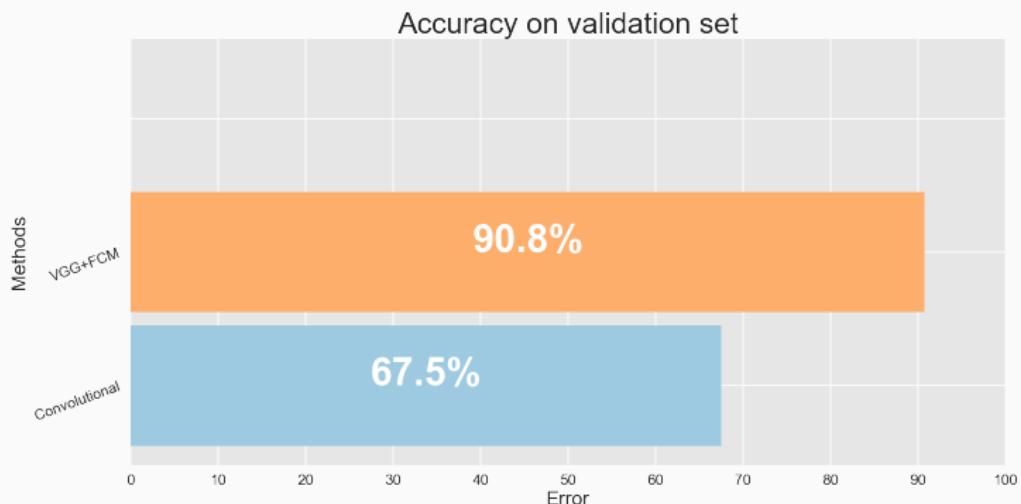
# On charge les poids du modèle évalué précédemment
top_model.load_weights(data_dir_sub+'/' + MODE + '_weights_model_VGG_fully_connected_model_'

# 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
```

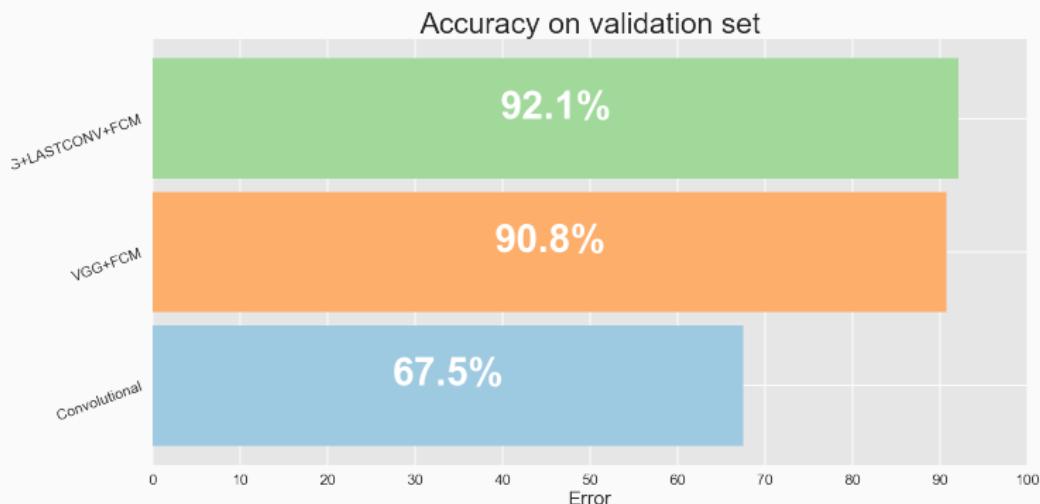
CATSVSDGOGS - VALIDATION SCORE

batch_size = 200, epochs = 15



CATSVSDGOGS - VALIDATION SCORE

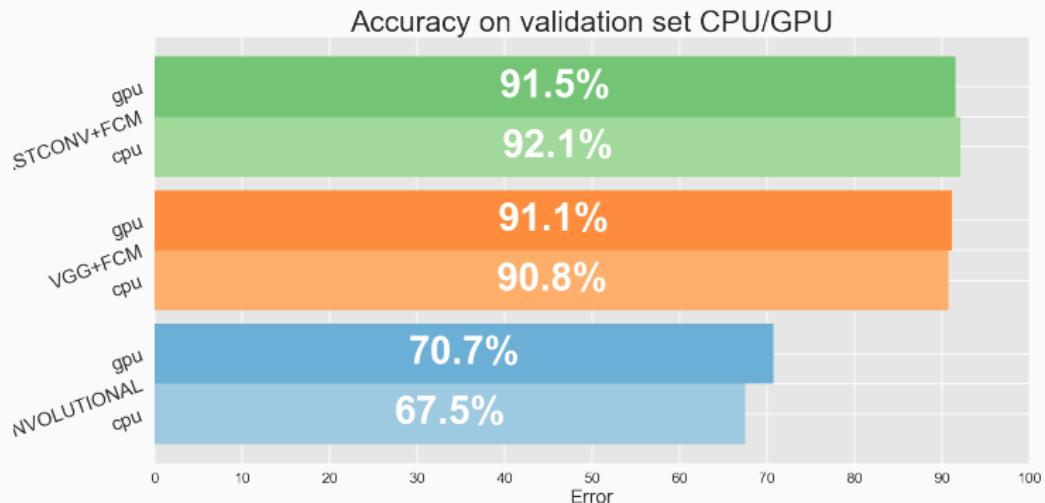
batch_size = 200, epochs = 15



Résultat encore amélioré

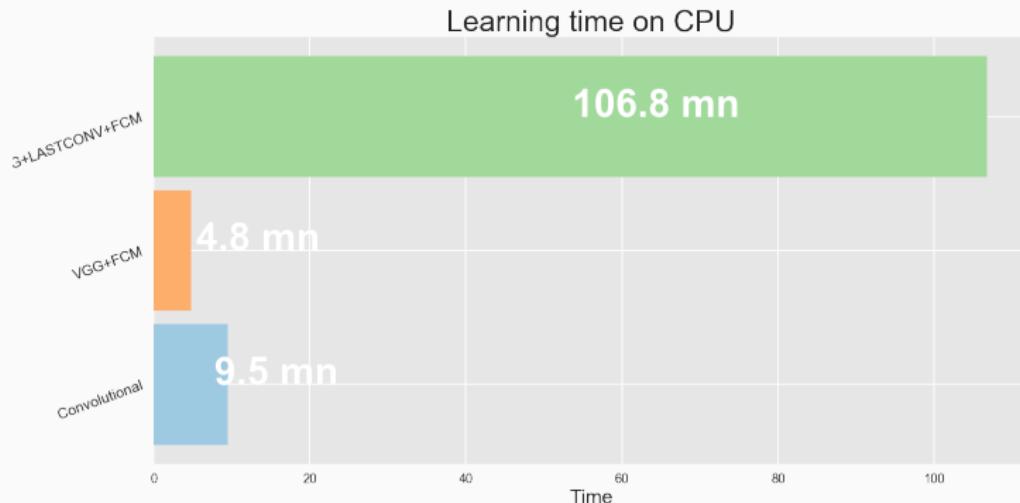
CATSVSDGOGS - VALIDATION SCORE - CPU/GPU

batch_size = 200, epochs = 15



CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU

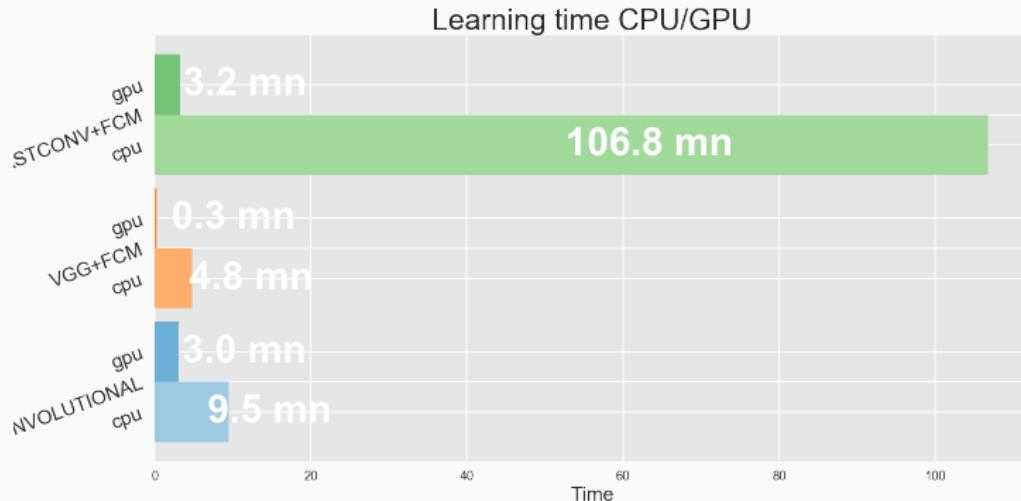
batch_size = 200, epochs = 15



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

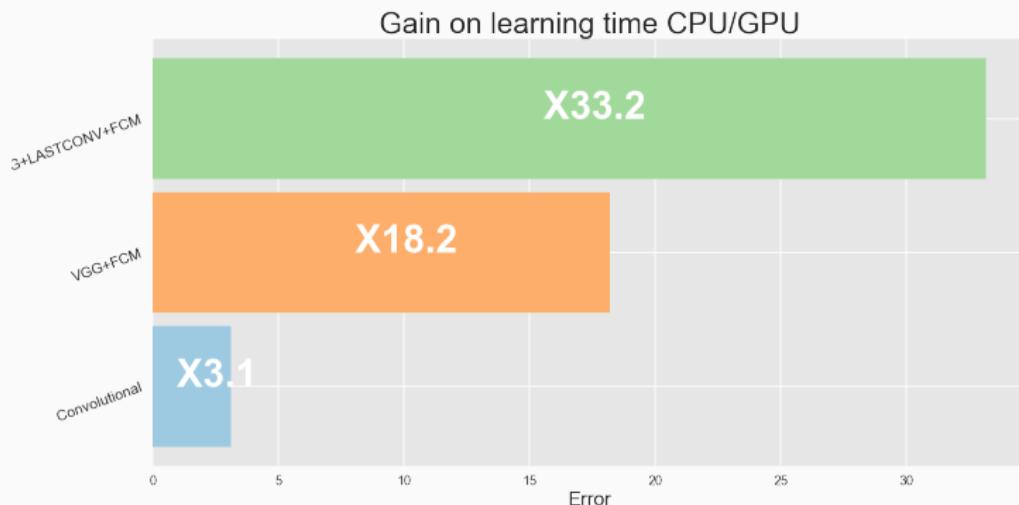
CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

batch_size = 200, epochs = 15



CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

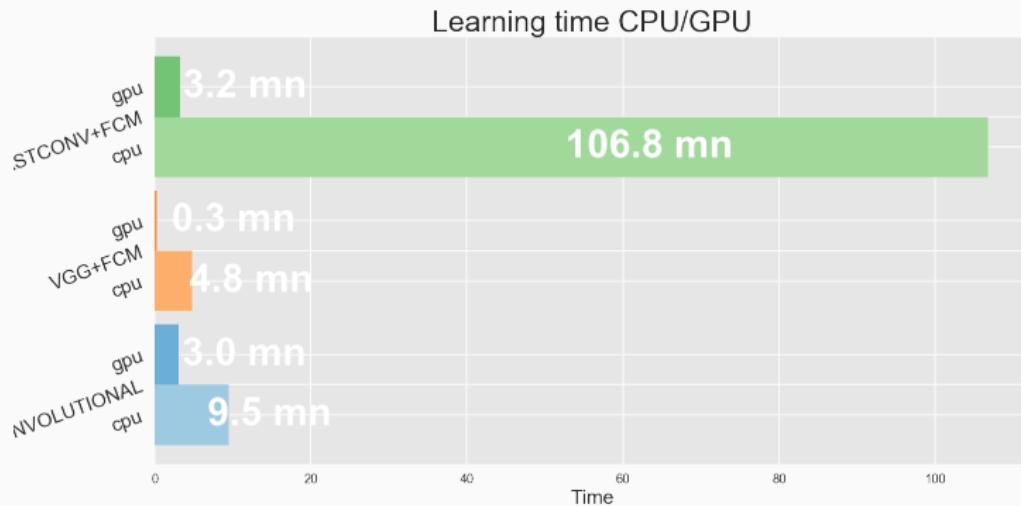
batch_size = 200, epochs = 15



Comment expliquer cette différence de gain ?

CATSVSDGOGS - TEMPS D'APPRENTISSAGE - CPU/GPU

batch_size = 200, epochs = 15

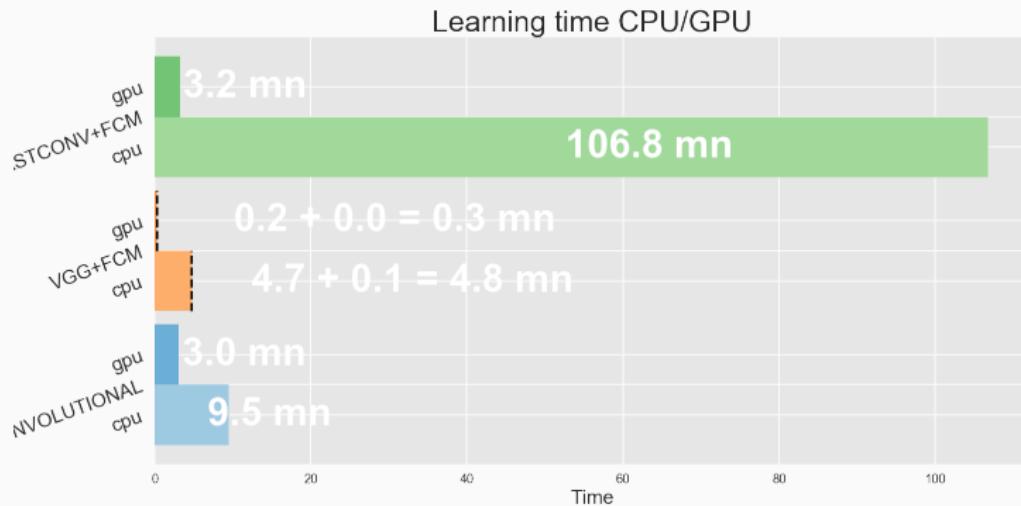


Dans la partie "Extraction de Features", c'est le calcul des features qui prend du temps!

Le modèle, très simple, ne prend pas beaucoup de temps sur le CPU. 61/61

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. 61/61

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/DLFrame-works.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>

REFERENCES I

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.