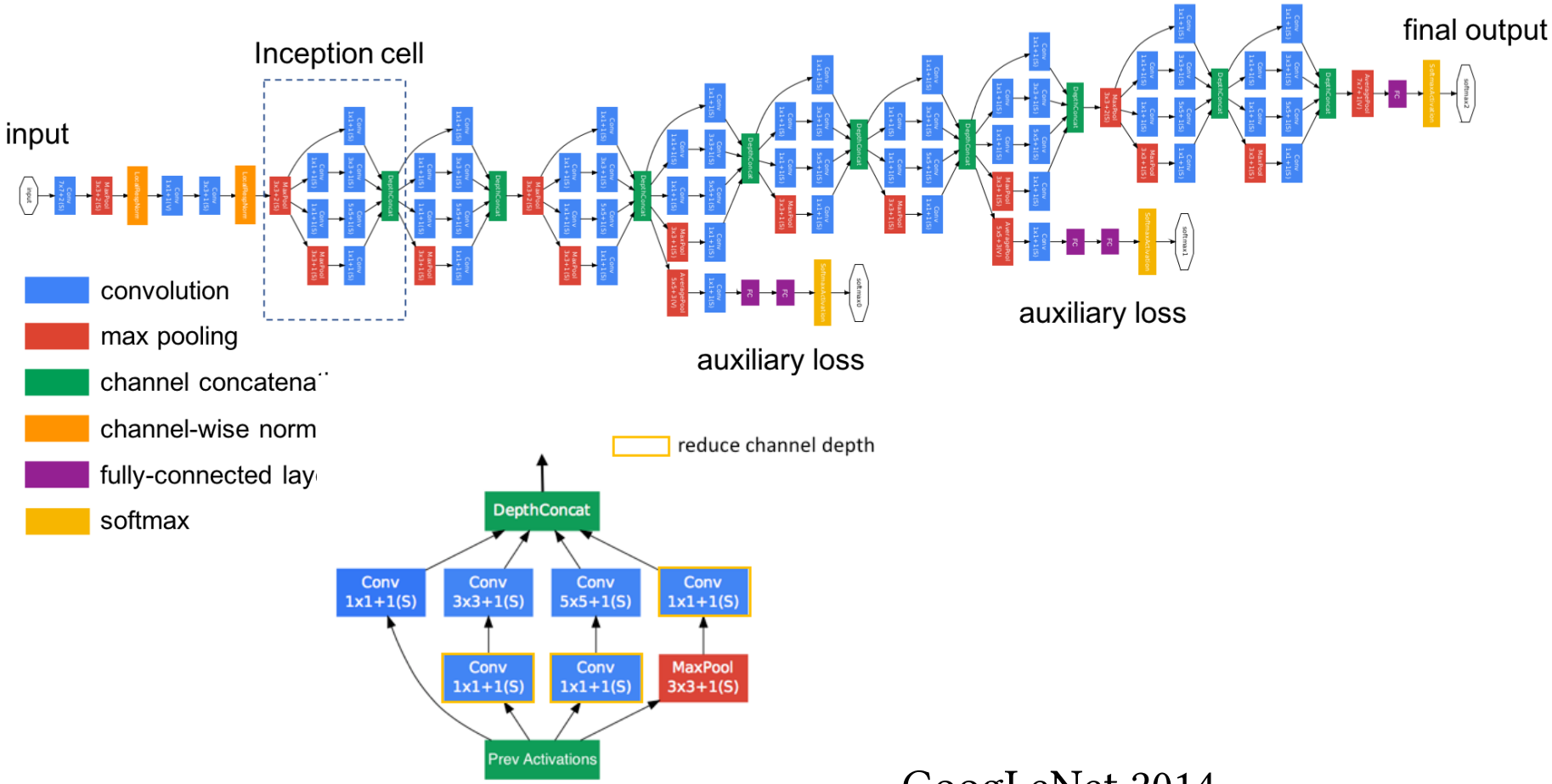
Réseaux neuronaux convolutifs

Olivier Ricou

2021

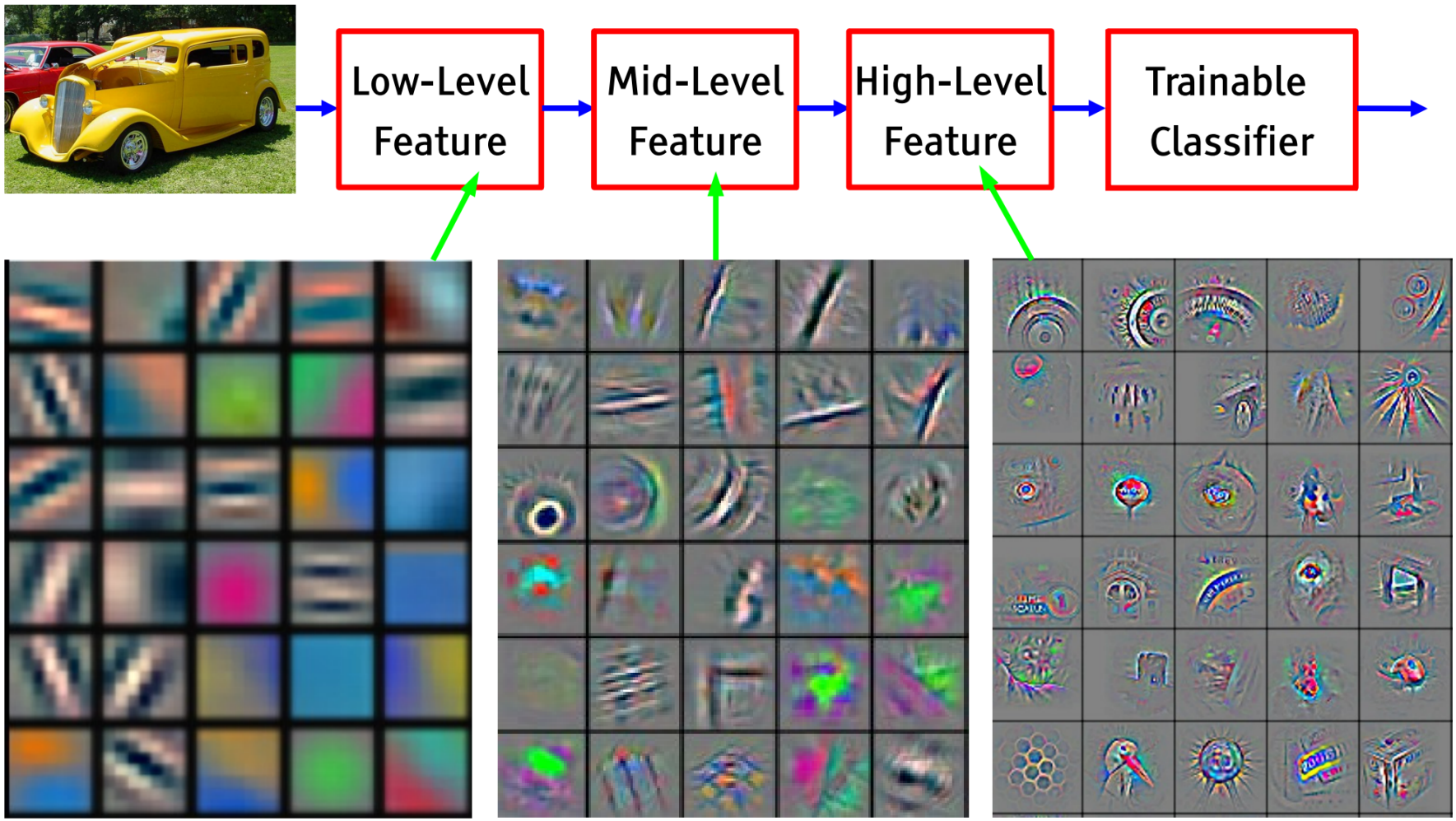
Un réseau neuronal convolutif

Les convolutions sont les boites bleues.



GoogLeNet 2014

Le but est d'extraire des caractéristiques



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Les formules de convolution

Continue 1D :

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t)\ g(t)\,\mathrm{d}t = \int_{-\infty}^{+\infty} f(t)\ g(x-t)\,\mathrm{d}t$$

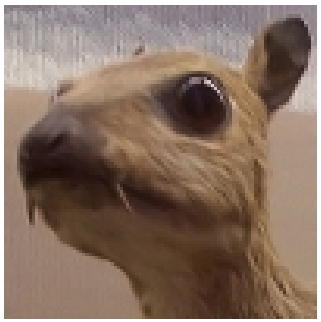
Discrète 2D :

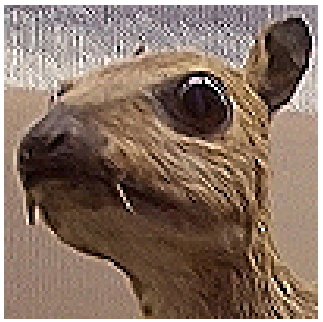
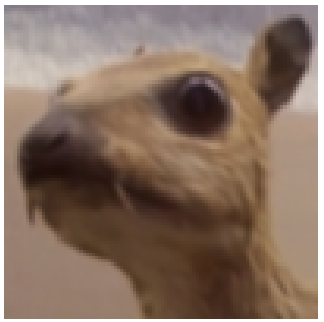
$$(f * \omega)(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(a+dx, b+dy)\ f(x+dx, y+dy)$$

f est l'image. ω est le noyau, son support est $[-a,a] \times [-b,b]$.

Exemple de noyaux ω (WP Noyau_(traitement_d'image)) :

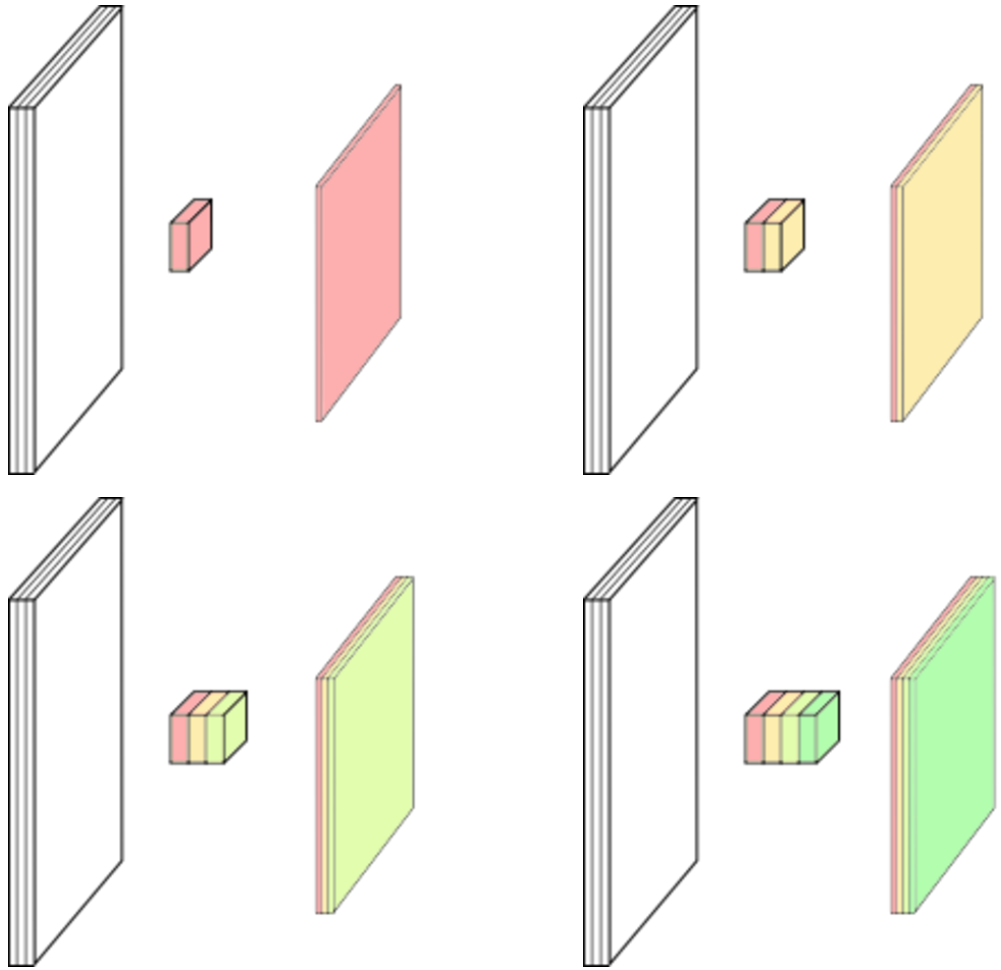
$$[1] \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



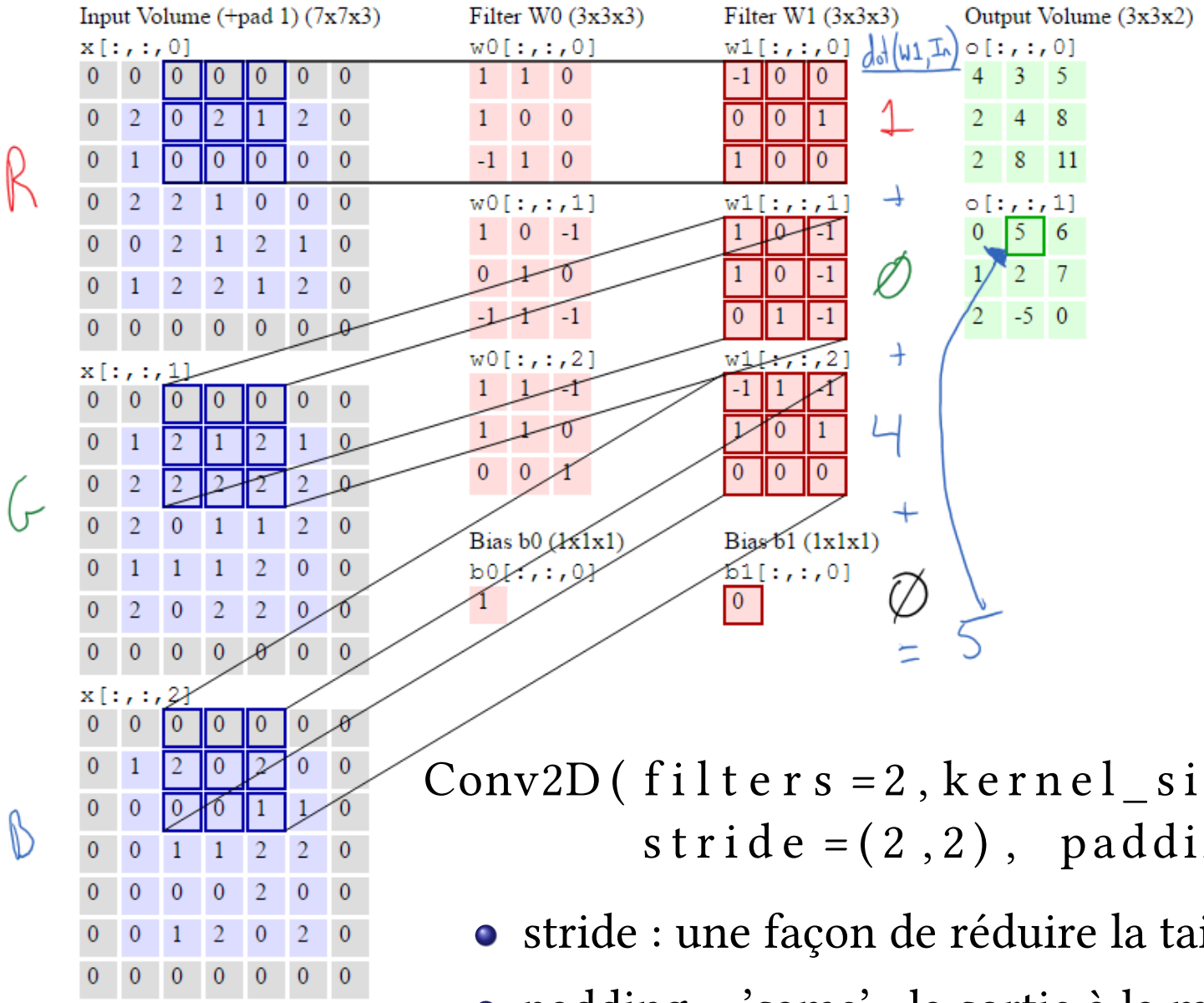
Conv2D

```
x = kl.Conv2D(filters=4, kernel_size=(5,5))(x)
```



L'image d'entrée a 3 canaux $\rightarrow$ chaque fitre a $5 \times 5 \times 3 + 1$ poids
L'image de sortie a 4 canaux, elle pert 4 pixels dans chaque direction.

Conv2D en détail + stride + padding = 'same'



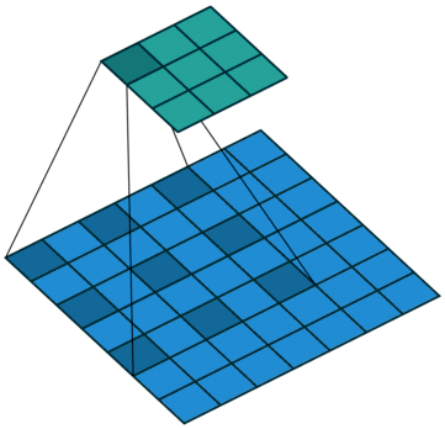

```
Conv2D(filters=2, kernel_size=(3,3),
       stride=(2,2), padding='same')
```

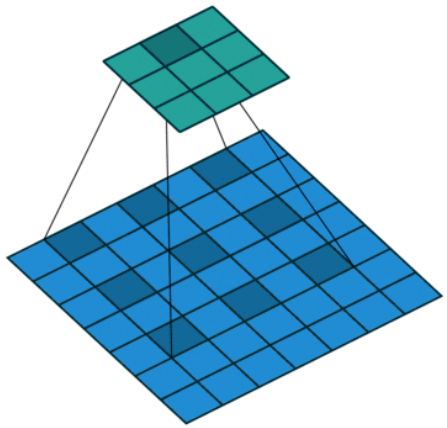
- stride : une façon de réduire la taille d'une image.
- padding = 'same' : la sortie à la même taille.

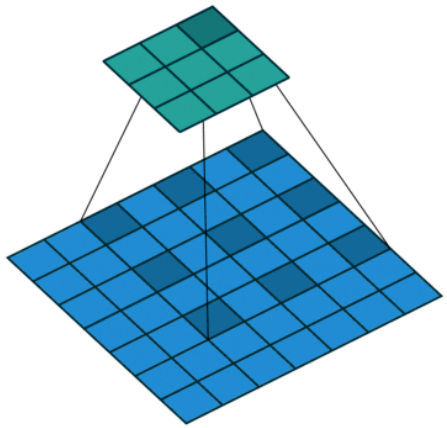
Convolution à trous

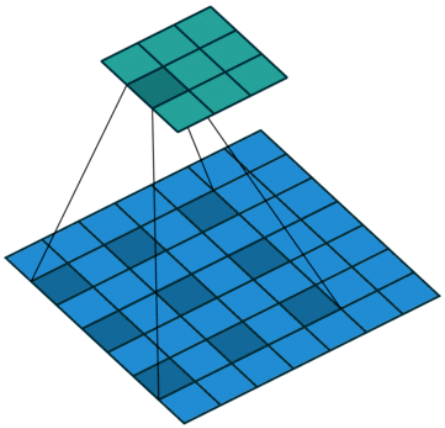
En anglais : *atrous convolution*

```
Conv2D(32, kernel_size=3, dilatation_rate=(2,2))
```









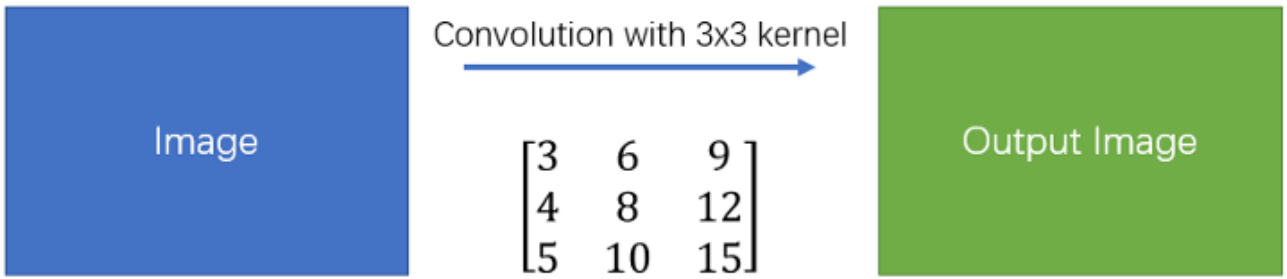
Couvre la même surface qu'un noyau 5×5,
Ou que 2 convolutions 3×3 à la suite,
Mais pour un coût moins cher (en poids).

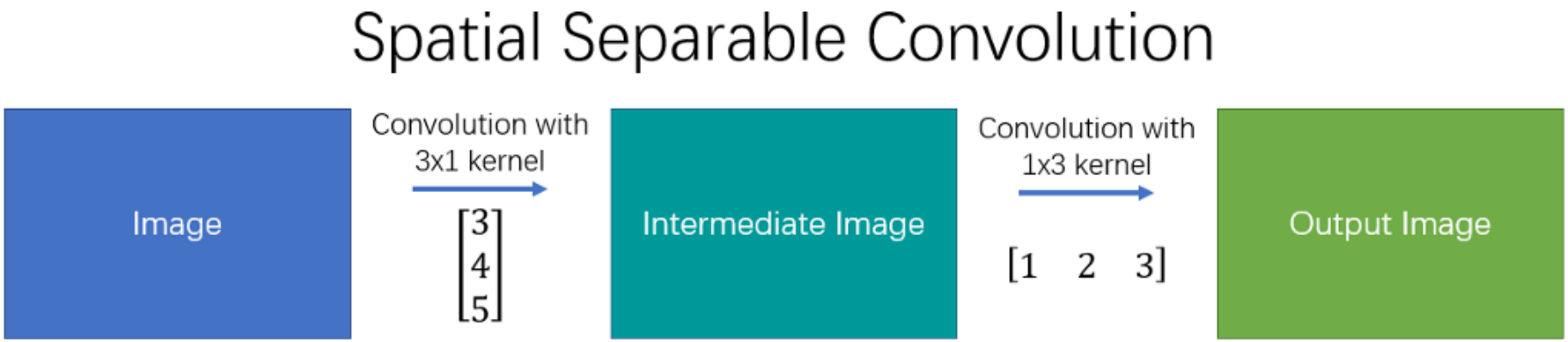
Sans réduction de la taille de l'image : padding = 'same'

Convolution séparée 1/2

- spaciale : une conv. 2D $\rightarrow$ 2 conv. 1D
- profondeur : N conv 2D sur M couches $\rightarrow$ M conv 2D puis N conv 1D

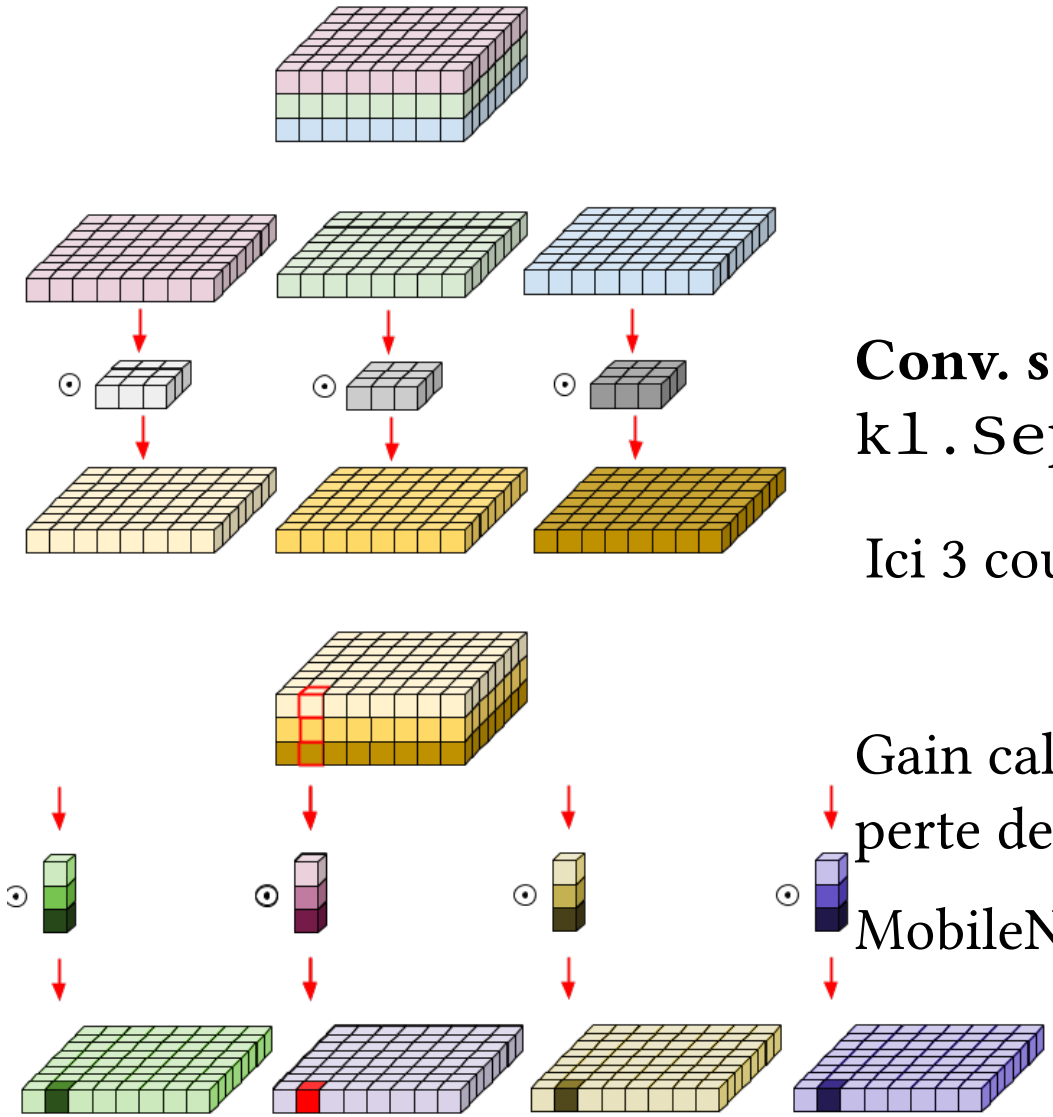
Convolution séparée spatiale





Gain calcul faible, grosse perte de représentation $\rightarrow$ pas utilisé.

Convolution séparée 2/2



Conv. séparée en profondeur
`kl.SeparableConv2D`

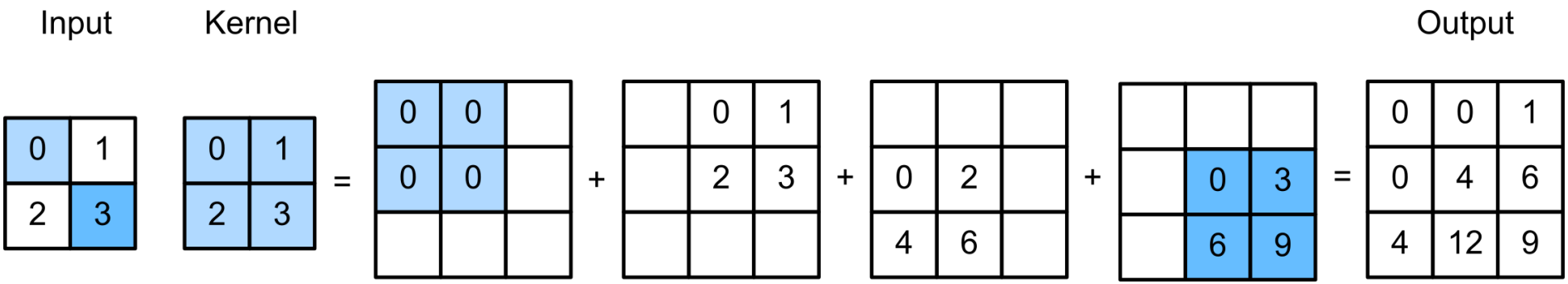
Ici 3 couches $\rightarrow$ 3 conv 2D + 4 conv 1D
$\rightarrow$ 4 couches en sortie.

Gain calcul important,
perte de représentation $\rightarrow$ utilisé.

MobileNet https ://arxiv.org/abs/1704.04861

La convolution transposée (ou déconvolution)

Convolution : **concentre** en un pixel un bloc de pixel (fois un noyau).
Conv. transposée : **distribue** un pixel (fois un noyau) à un bloc de pixel.



Mathématiquement les deux sont des convolutions mais
la conv. transposée a pour but de simuler l'opération inverse de la conv.

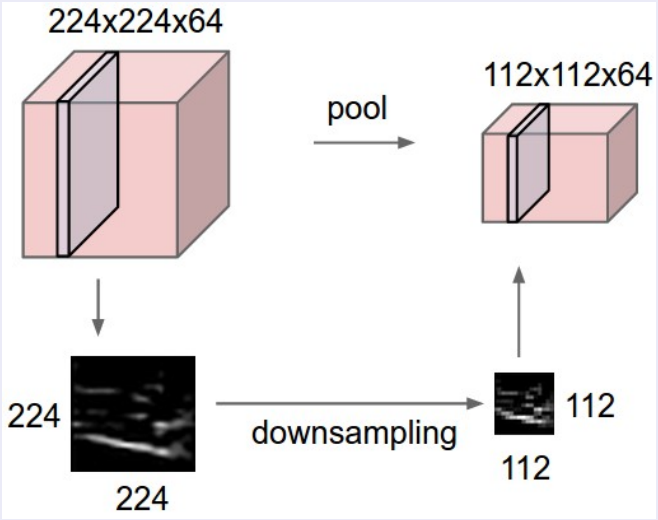
$$\text{propagation conv. transposée} \leftrightarrow \text{rétro-propagation conv.}$$
$$\text{retro-propagation conv. transposée} \leftrightarrow \text{propagation conv.}$$

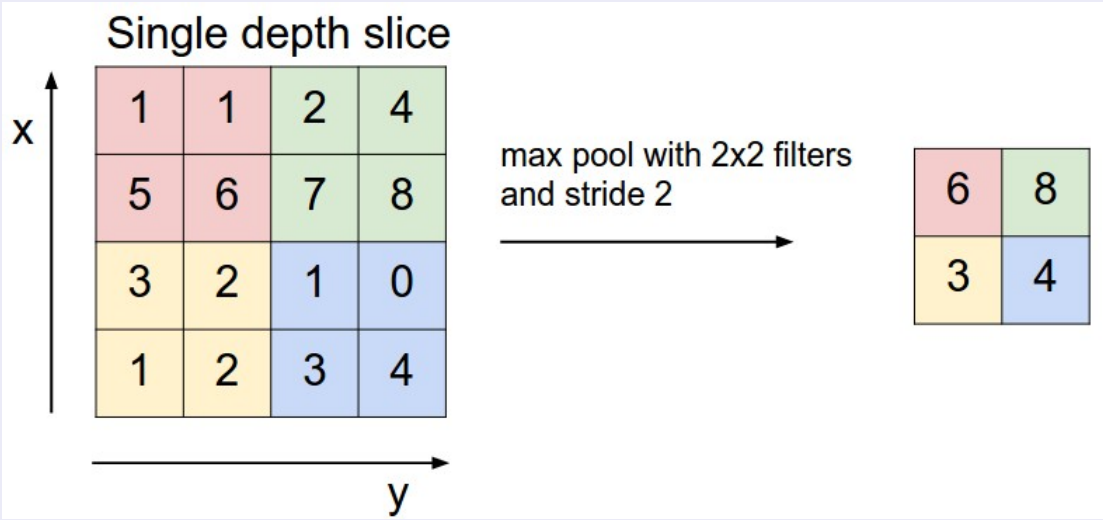
https ://arxiv.org/pdf/1603.07285v1.pdf chapitre 4
https ://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

Trucs d'architecture

Pooling

```
kl.MaxPooling2D(pool_size=(2, 2))
```





Si on veut augmenter le nombre de couche il faut diminuer la taille de l'image sinon BOUM.

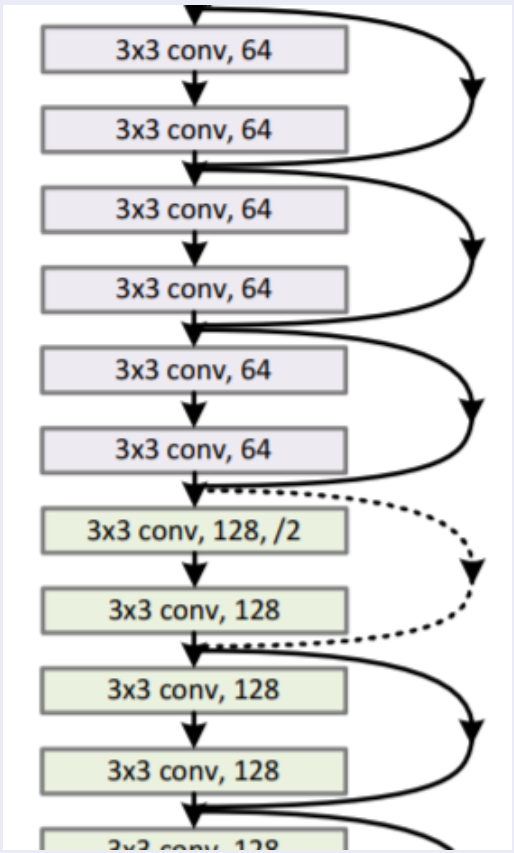
Si on veut une vision multi-échelle il faut diminuer la taille de l'image + ponts.

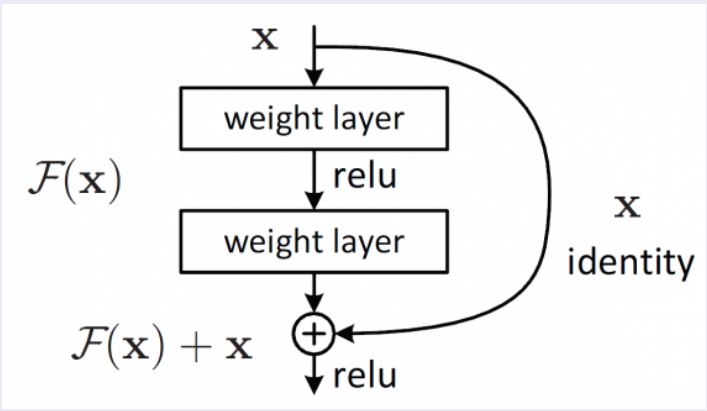
L'inverse du *pooling* est `kl.UpSampling(size=(2,2))`.

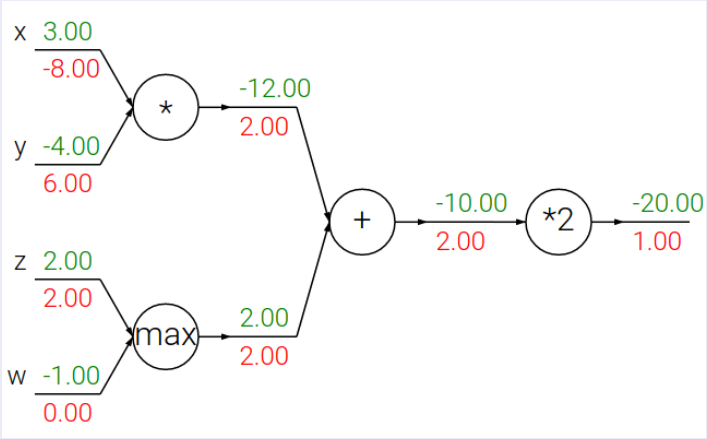
Trucs d'architecture

Ponts

La grande astuce de ResNet qui leur a permis de tout gagner.







Prog. et rétro-prog.

vert $\rightarrow$

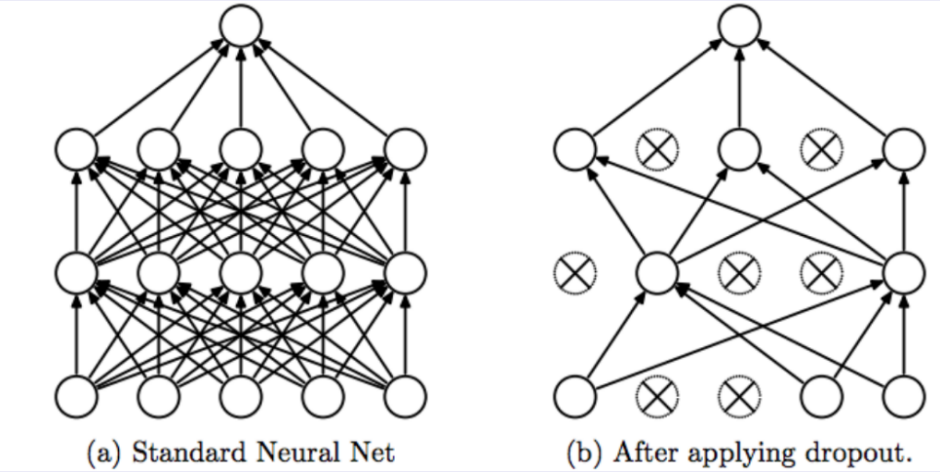
rouge $\leftarrow$

Lors de la rétro-propagation l'erreur prend le pont et les convolutions.
$\rightarrow$ les premières couches sont corrigées.

Trucs d'architecture

Dropout ou BatchNormalization

Pas besoin de Dropout
si BatchNormalization



(a) Standard Neural Net (b) After applying dropout.

Évite que des poids importants en bloquent d'autres.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

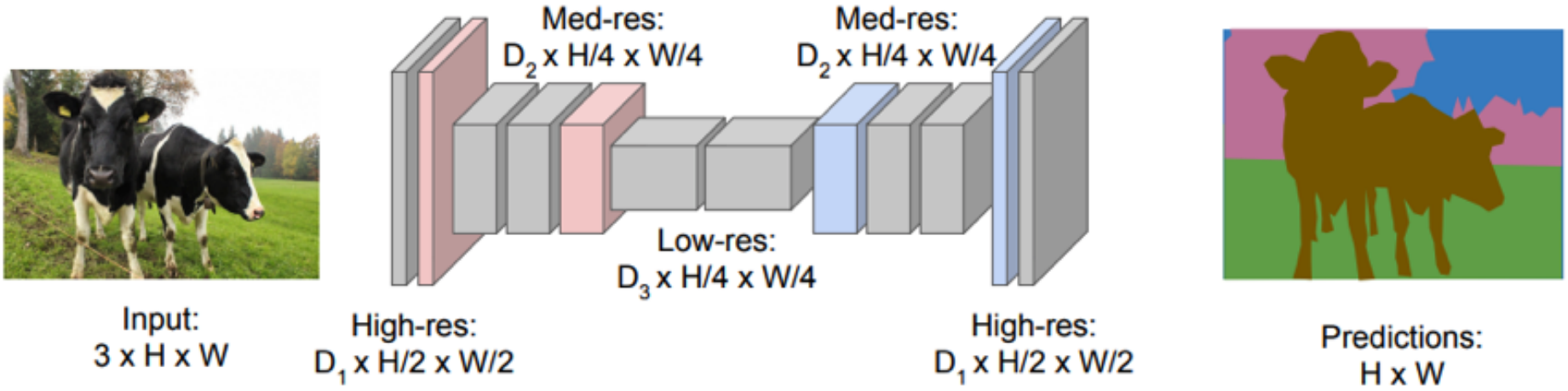
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

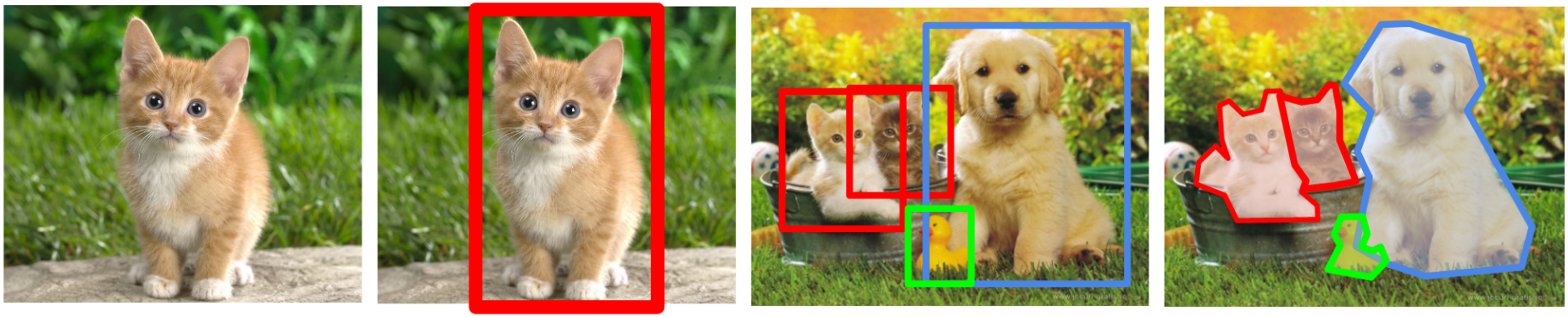
Après convolution,
avant fonction d'activation
Réduit le besoin de normaliser les données.

Types de problèmes en vision

Semantic segmentation



Classification | **Classification + Localization** | **Object Detection** | **Instance Segmentation**

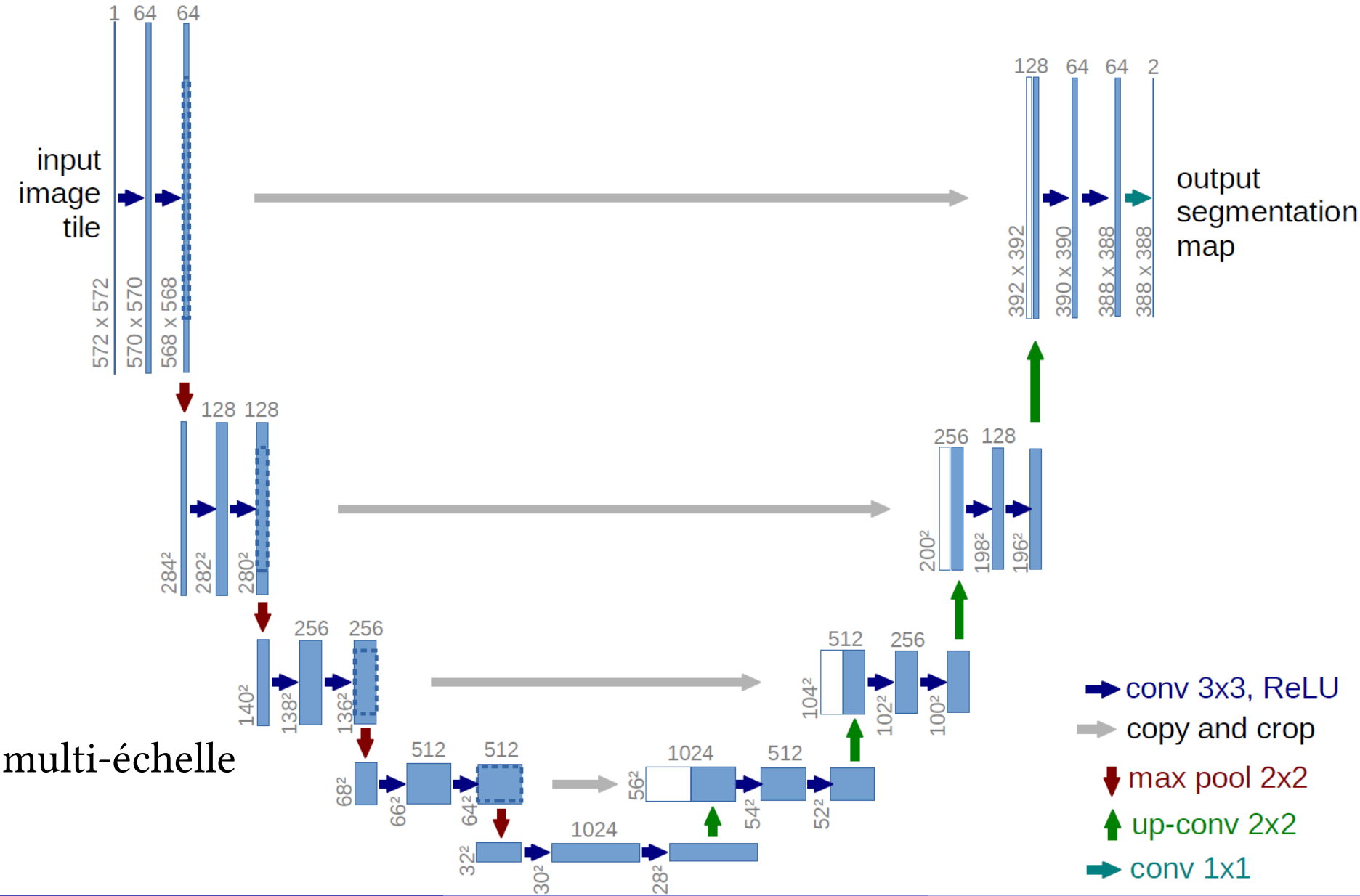


CAT | CAT | CAT, DOG, DUCK | CAT, DOG, DUCK

Single object | Multiple objects

U-net (2015)

Segmentation sémantique d'images médicales



Fonctions d'erreur pour la segmentation 1/2

Si chaque image de sortie représente les pixels appartenant à la classe k, alors on peut finir avec un `'softmax'` : $y_k = e^{z_k} / \sum_i e^{z_i}$

- Erreur quadratrique `'mse'` : pente douce, pas d'information d'exclusion
- Entropie croissée : $E = -\sum_k t_k \log y_k + (1 - t_k) \log(1 - y_k)$
  - `'binary_crossentropy'` avec t_k = 0 ou 1,
  - `'categorical_crossentropy'` avec resultats sous la forme `[0,0,..,1,..,0]` pour indiquer la classe k.
  - `'sparse_categorical_crossentropy'` avec les classes indiquées par des entiers.

```
1 y_true = [[1, 2],[0,2]]  # image 2x2 with 3 categories
2 y_pred = [[0.05, 0.95, 0], [0.1, 0.1, 0.8], # proba for each category
3           [0.7, 0.2, 0.1], [0.2, 0.2, 0.6]] # for each pixel
4 loss = keras.losses.SparseCategoricalCrossentropy()
5 loss(y_true, y_pred).numpy()
```

```
0.2854844
```

Fonctions d'erreur pour la segmentation 2/2

La fonction d'erreur est très importante et une petite modification peut apporter de grands résultats.

Focal loss

$$E_{FL} = -\sum_k t_k \, (1 - y_k)^\gamma \log y_k + (1 - t_k) \, (1 - y_k)^\gamma \log(1 - y_k)$$

Comme la pente du log est forte, elle favorise les cas simples à détecter.
On peut écraser la courbe de $(1 - y_k)^\gamma$ pour aider à trouver les cas difficiles.

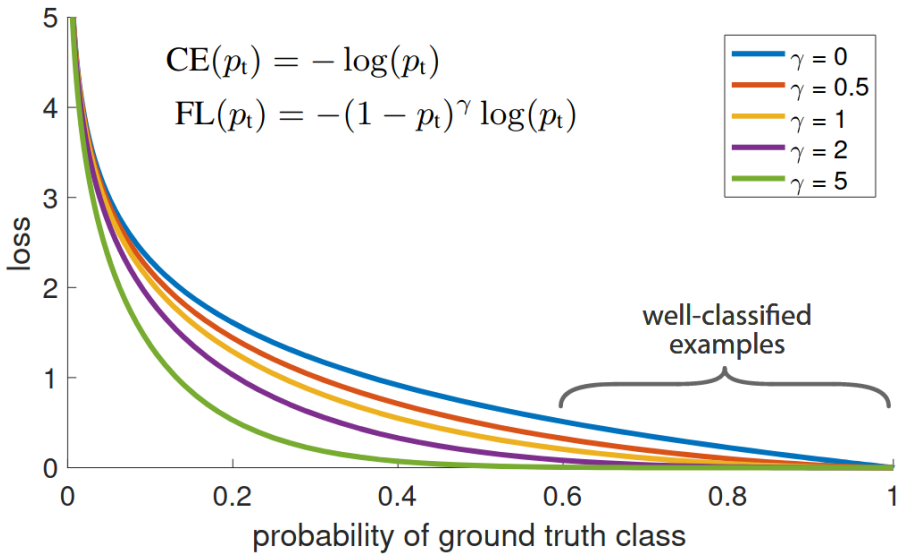
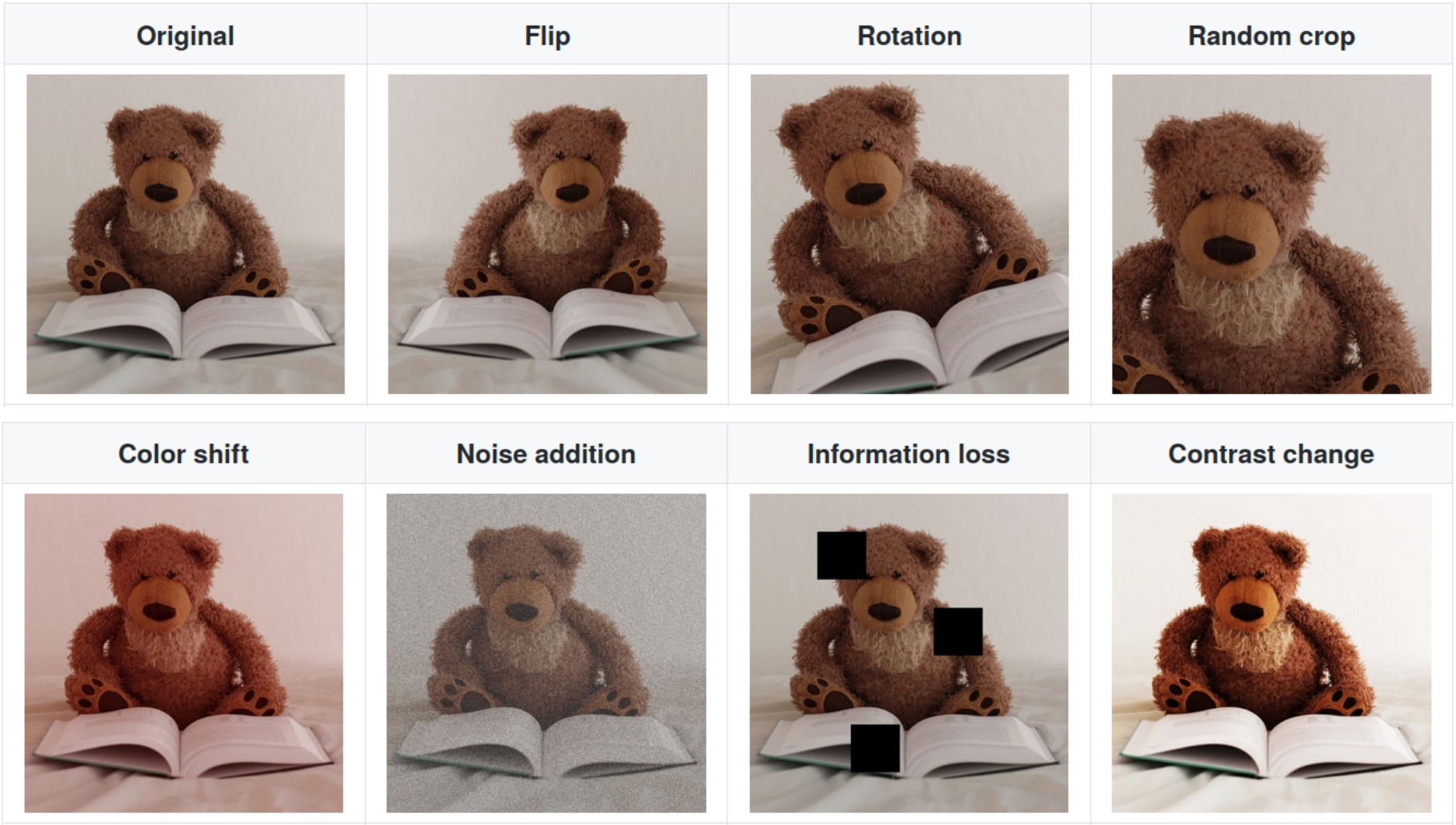


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

Augmenter le nombre de données 1/2



Souvent c'est bien utile, en particulier lorsqu'on manque de données.
Parfois ça rend la tache plus difficile et ça ne marche pas.

Augmenter le nombre de données 2/2

```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(x_train)
# fits the model on batches with real-time data augmentation:
model.fit(datagen.flow(x_train, y_train, batch_size=32),
          steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode="nearest",
    cval=0.0,
    horizontal_flip=False,
```