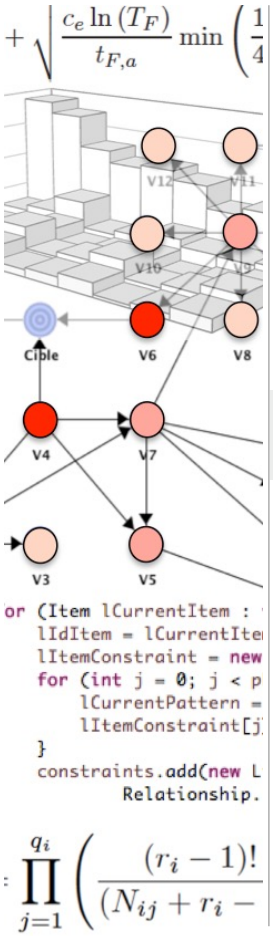
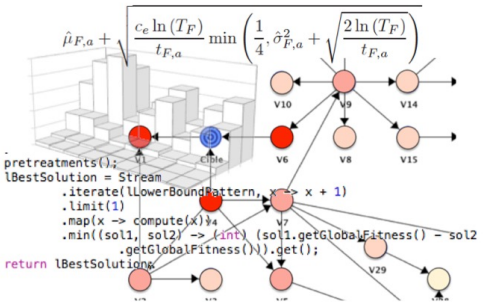


DEEP LEARNING

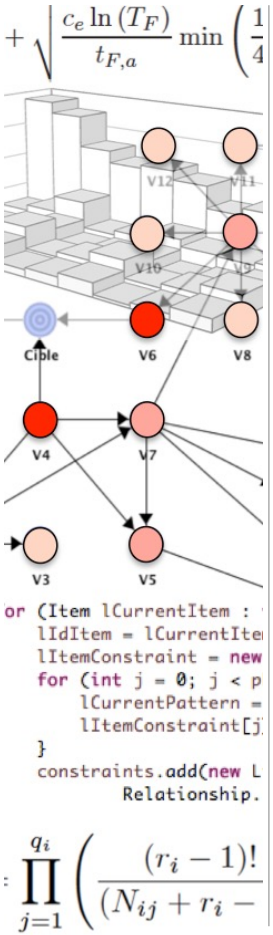
Transfer Learning

Stéphane BONNEVAY

Polytech Lyon
Laboratoire ERIC
stephane.bonnevay@univ-lyon1.fr



Plan

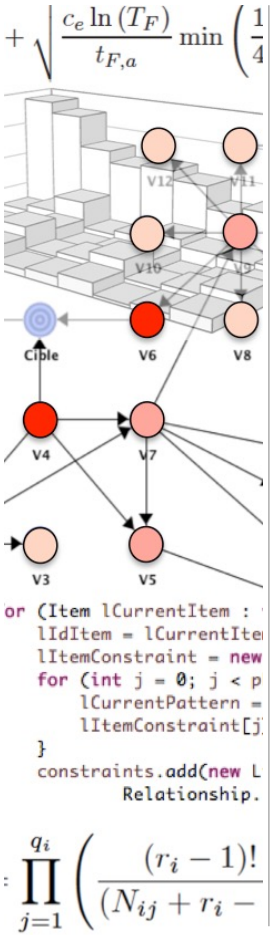


1. Introduction

2. Classification d'images avec
   - un réseau multi-couches
   - un réseau de neurones convolutifs
   - un réseau pré-entrainé (transfer learning)

3. Détection d'objets dans une image

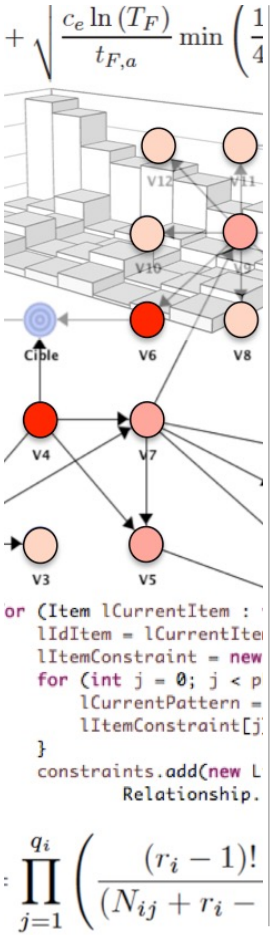
Plan



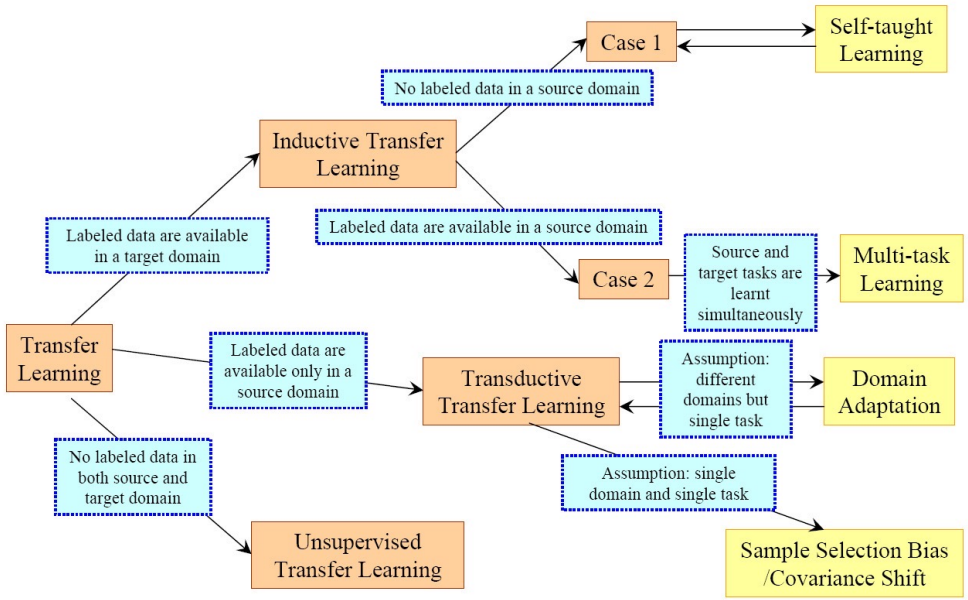
1. Introduction

2. Classification d'images avec
   - un réseau multi-couches
   - un réseau de neurones convolutifs
   - un réseau pré-entrainé (transfer learning)

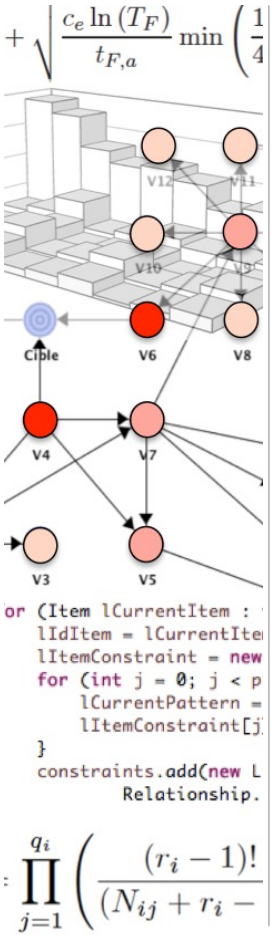
3. Détection d'objets dans une image



Le **Transfer Learning** consiste à réutiliser l'expérience acquise au terme d'un apprentissage dans le cadre d'un domaine ou d'une tâche.

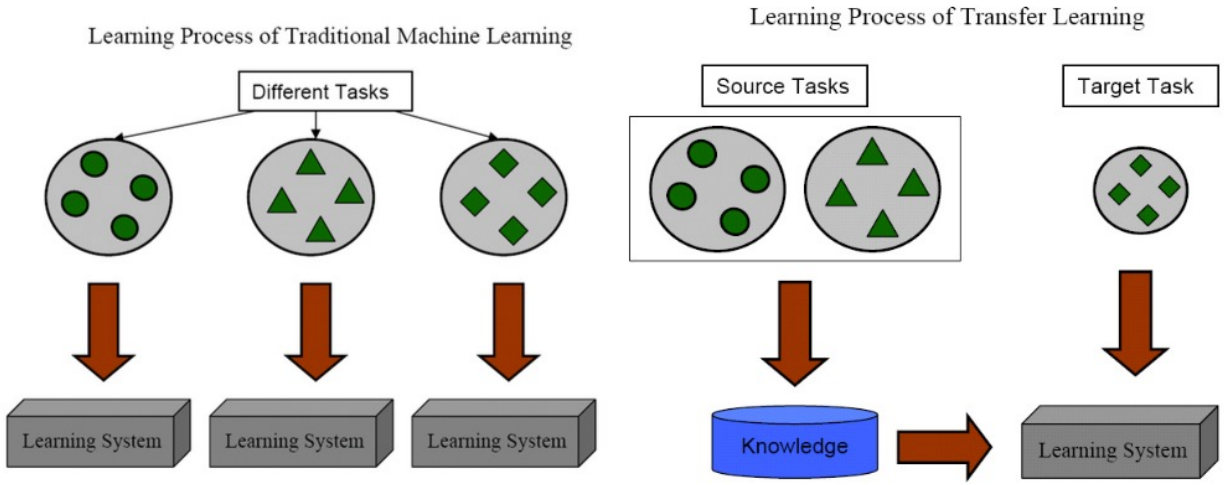


Pan and Q. Yang. « A survey on Transfer Learning », IEEE Transactions on Knowledge and Data Engineering, Vol.22(10), October, 2010.

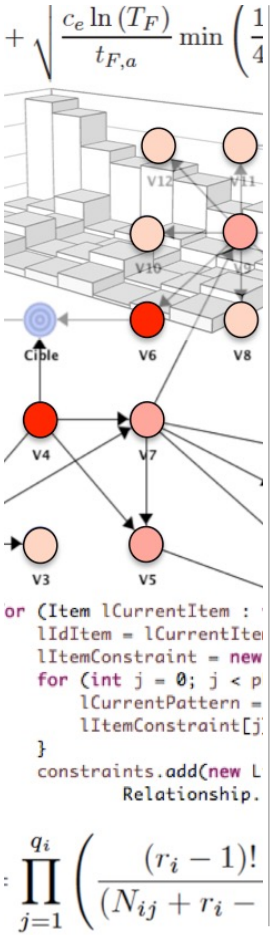


Idée :

adapter un modèle existant, destiné à une certaine tâche et à un certain domaine
à un autre domaine pour réaliser une autre tâche

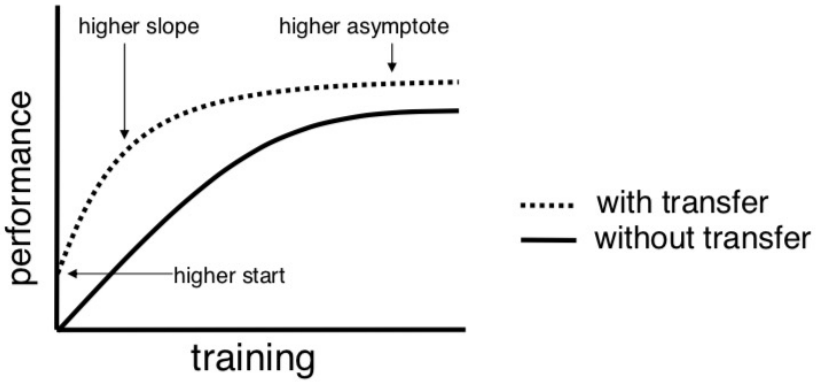


Pan and Q. Yang. « A survey on Transfer Learning », IEEE Transactions on Knowledge and Data Engineering, Vol.22(10), October, 2010.



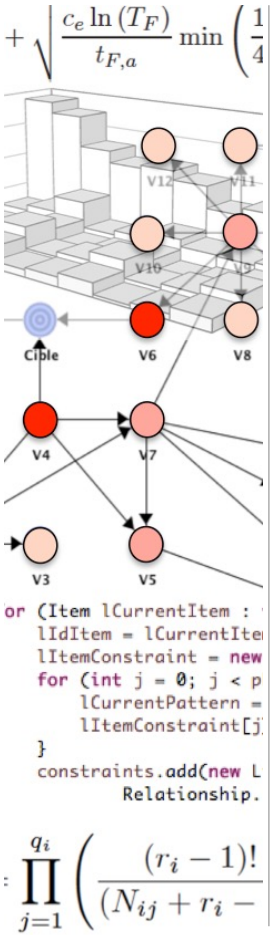
Avantages :

- amélioration de la qualité de l'apprentissage
- adaptation de « gros » modèles nécessitant des moyens techniques hors de portée pour son propre problème
- réduction du temps de développement d'un modèle
- performance finale accrue

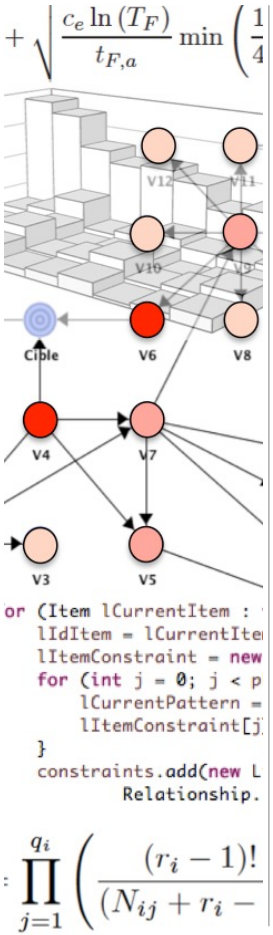


Torrey, L. and Shavlik, J. "Transfer Learning. In: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques", IGI Global, Hershey, 242-264, 2010.

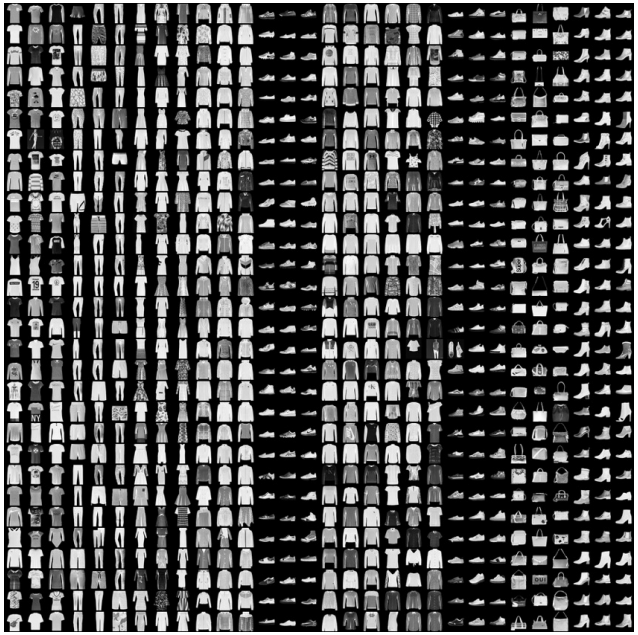
Plan



1. Introduction
2. Classification d'images avec
   - un réseau multi-couches
   - un réseau de neurones convolutifs
   - un réseau pré-entrainé (transfer learning)
3. Détection d'objets dans une image
4. Classification de textes



Exemple : jeu de données « Fashion MNIST »



Dataset : https://github.com/zalandoresearch/fashion-mnist

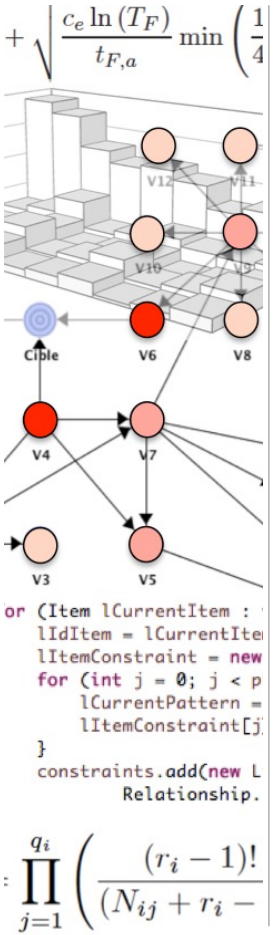
70 000 images en niveau de gris de taille 28x28

valeurs de 0 à 255

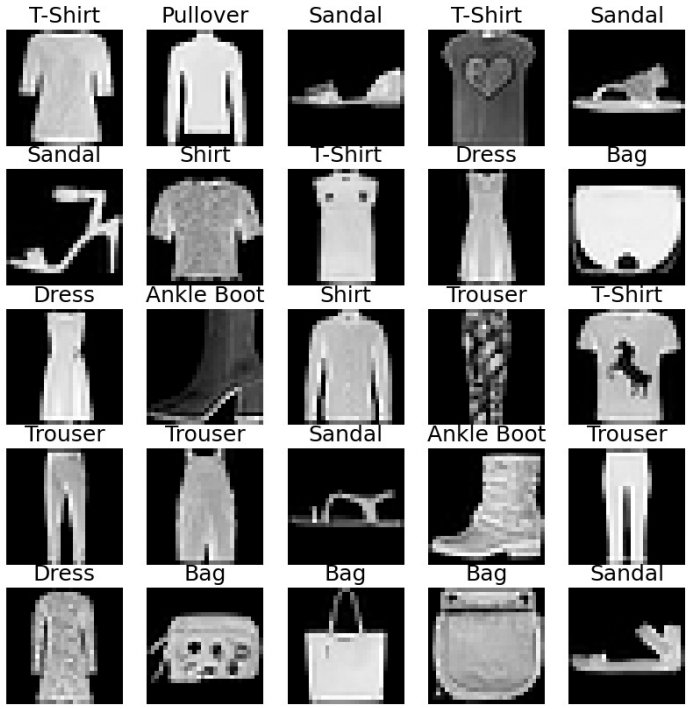
60 000 pour l'apprentissage
10 000 pour le test

```
from tensorflow.keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

10 catégories



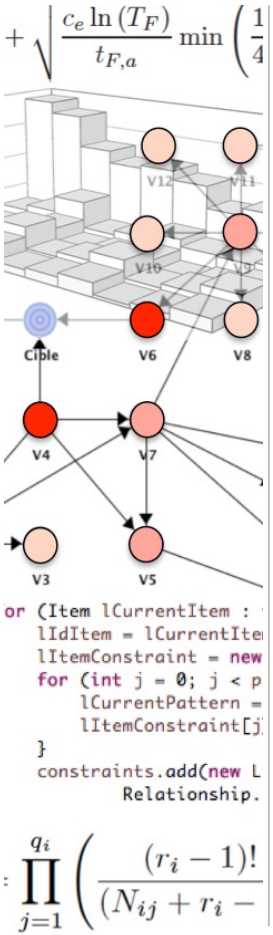
Exemple : jeu de données « Fashion MNIST »

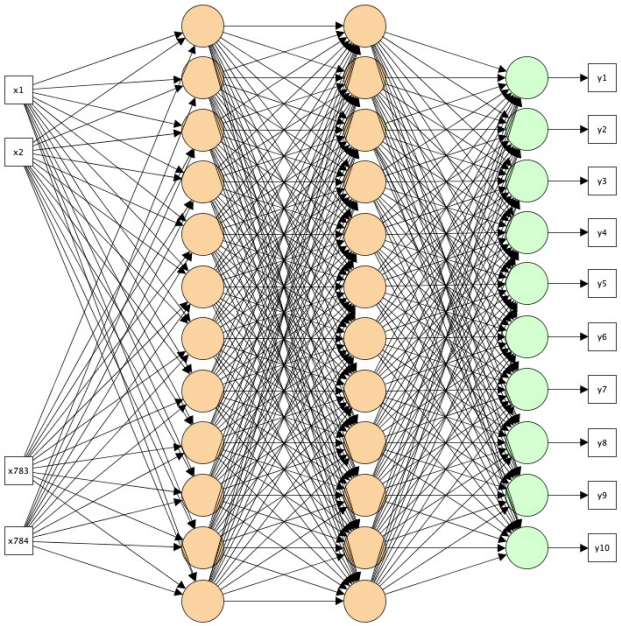


10 catégories :

- 0: T-shirt
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

Classification d'images - Réseaux multicouches

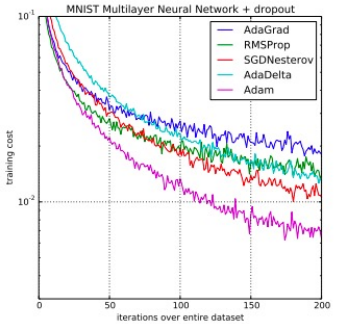




Il est possible de combiner différents types de fonctions d'activation (à ce jour, ReLu est la plus utilisée).

Il existe plusieurs algorithmes d'apprentissage :

- AdaGrad
- Adam ⟵ le plus utilisé
- RMSProp
- AdaDelta
- …

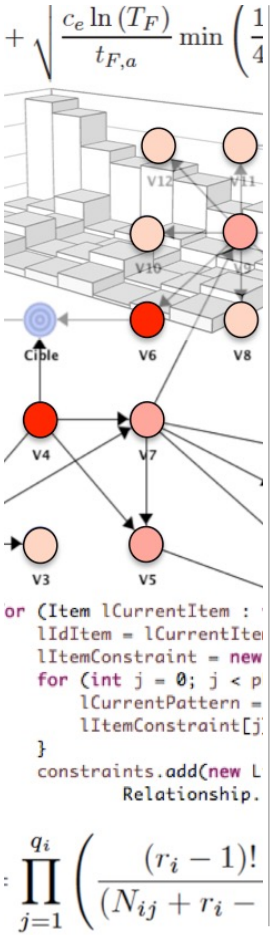


in « A Method for Stochastic Optimization », 2015

Paramètres conseillés :

Learning rate = 0.001
Beta = 0.9
Beta2 = 0.999
Epsilon = 1e-08

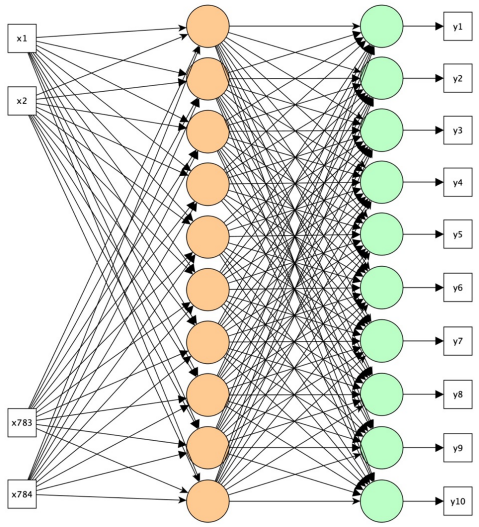
Classification d'images - Réseaux multicouches



Réseau de neurones : 1 couche cachée, 10 neurones

Matrice de taille
28x28 → Vecteur d'entrée de taille 784

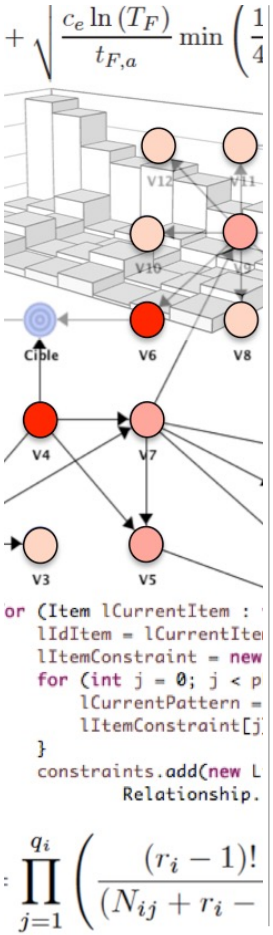
54 000 x 28 x 28 → 54 000 x 784
6 000 x 28 x 28 → 6 000 x 784
10 000 x 28 x 28 → 10 000 x 784



Choix arbitraire de 10 neurones
sur la couche cachée

Couche de sortie de taille 10
une par catégorie

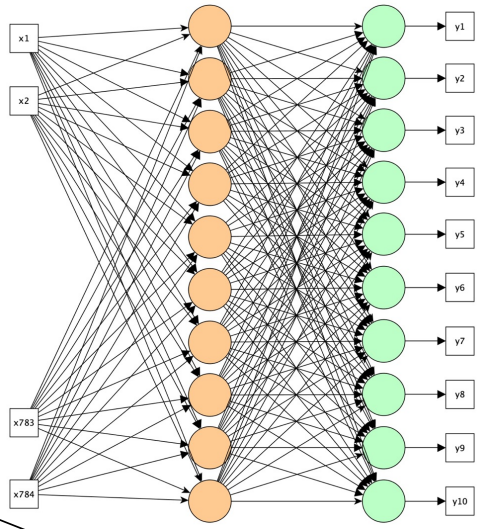
```
x_train = x_train.reshape(x_train.shape[0], -1) / 255.0
x_test = x_test.reshape(x_test.shape[0], -1) / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```



Réseau de neurones : 1 couche cachée, 10 neurones

```
model = Sequential()
model.add(Dense(10, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

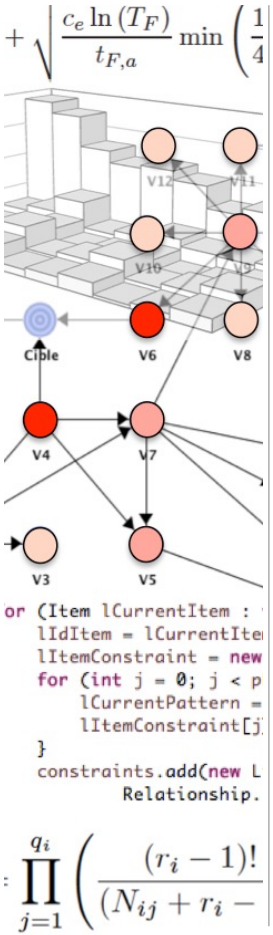
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                7850
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
_________________________________________________________________
```



(784x10+10)+(10x10+10)

Fonctions d'activation

- « Relu » sur les couches cachées
- « Softmax » si plus de 2 catégories sur la couche de sortie
- « Sigmoid » si 2 catégories sur la couche de sortie



Réseau de neurones : 1 couche cachée, 10 neurones

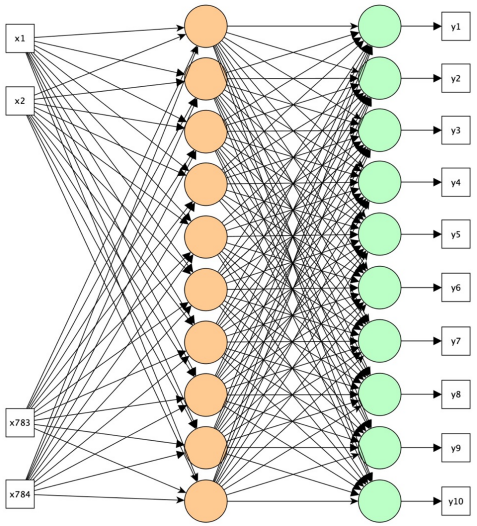
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

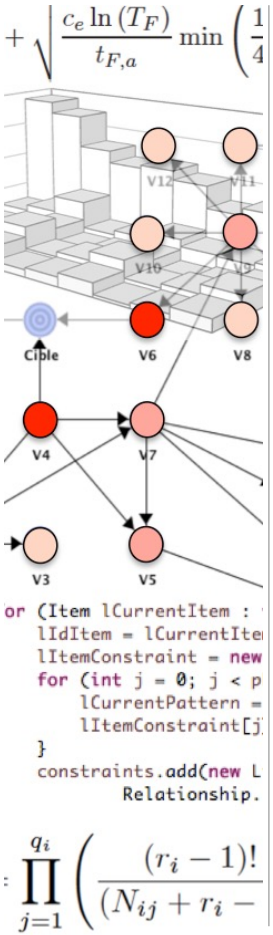
Paramètres

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

- « categorical_crossentropy » pour du multi-catégories
- « binary_crossentropy » pour 2 catégories
- « adam » ou « rmsprop » sont les 2 meilleures
- « accuracy » pour la métrique

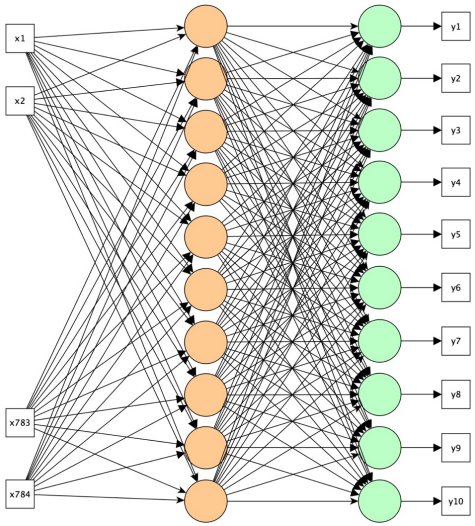
$$\frac{\text{nombre de bonnes prédictions}}{\text{nombre total de prédictions}}$$





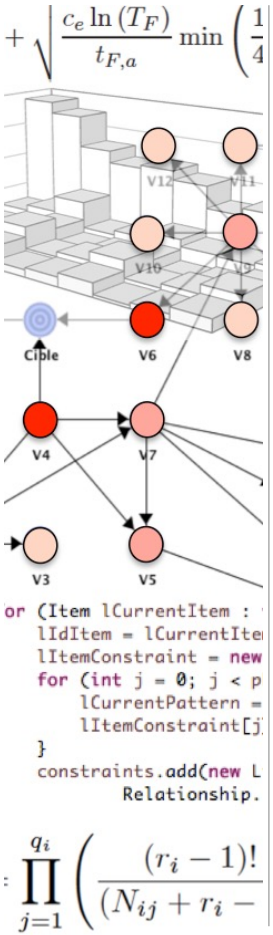
Réseau de neurones : 1 couche cachée, 10 neurones

```
model.fit(x_train, y_train,
          epochs=10,
          batch_size=32,
          validation_split=0.1)
```



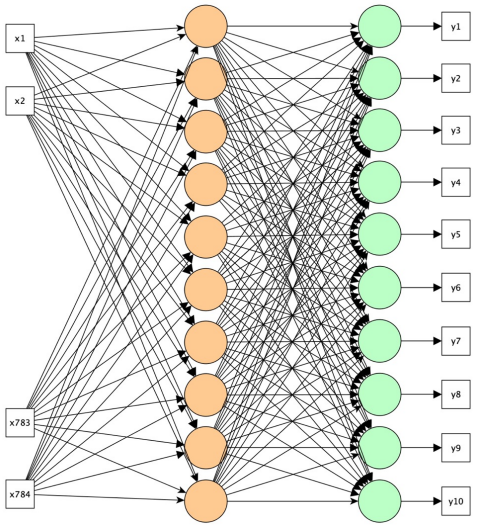
Paramètres

- 10 itérations (epochs)
  - 1 epoch = passage de l'ensemble des données dans le réseau
- Batch_size = nombre d'enregistrements qui passent dans le réseau avant la mise à jour des poids
- 10% des données d'apprentissage pour la validation après chaque epoch
  - 54 000 images pour l'apprentissage
  - 6 000 images pour la validation

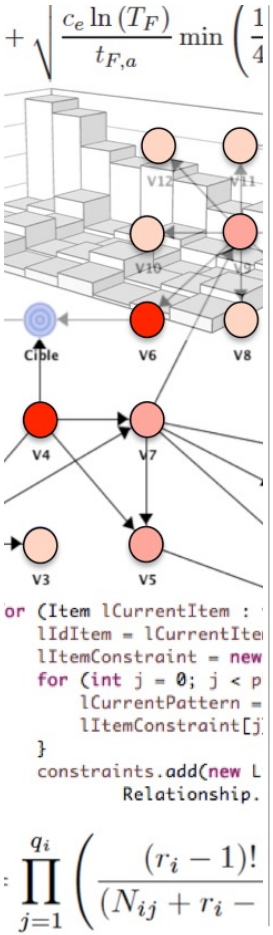


Réseau de neurones : 1 couche cachée, 10 neurones

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 3s 49us/sample - loss: 0.7834 - accuracy: 0.7354 - val_loss: 0.5066 -
val_accuracy: 0.8220
Epoch 2/10
54000/54000 [==============================] - 2s 32us/sample - loss: 0.4820 - accuracy: 0.8331 - val_loss: 0.4505 -
val_accuracy: 0.8402
Epoch 3/10
54000/54000 [==============================] - 2s 33us/sample - loss: 0.4457 - accuracy: 0.8447 - val_loss: 0.4391 -
val_accuracy: 0.8472
Epoch 4/10
54000/54000 [==============================] - 2s 35us/sample - loss: 0.4264 - accuracy: 0.8506 - val_loss: 0.4296 -
val_accuracy: 0.8453
Epoch 5/10
54000/54000 [==============================] - 2s 34us/sample - loss: 0.4153 - accuracy: 0.8546 - val_loss: 0.4217 -
val_accuracy: 0.8505
Epoch 6/10
54000/54000 [==============================] - 2s 34us/sample - loss: 0.4067 - accuracy: 0.8584 - val_loss: 0.4216 -
val_accuracy: 0.8543
Epoch 7/10
54000/54000 [==============================] - 2s 36us/sample - loss: 0.4011 - accuracy: 0.8601 - val_loss: 0.4137 -
val_accuracy: 0.8527
Epoch 8/10
54000/54000 [==============================] - 2s 33us/sample - loss: 0.3961 - accuracy: 0.8615 - val_loss: 0.4396 -
val_accuracy: 0.8430
Epoch 9/10
54000/54000 [==============================] - 2s 36us/sample - loss: 0.3902 - accuracy: 0.8622 - val_loss: 0.4120 -
val_accuracy: 0.8542
Epoch 10/10
54000/54000 [==============================] - 2s 33us/sample - loss: 0.3863 - accuracy: 0.8637 - val_loss: 0.4095 -
val_accuracy: 0.8555
```



85,55% de précision sur les données de validation



Réseau de neurones : 1 couche cachée, 10 neurones

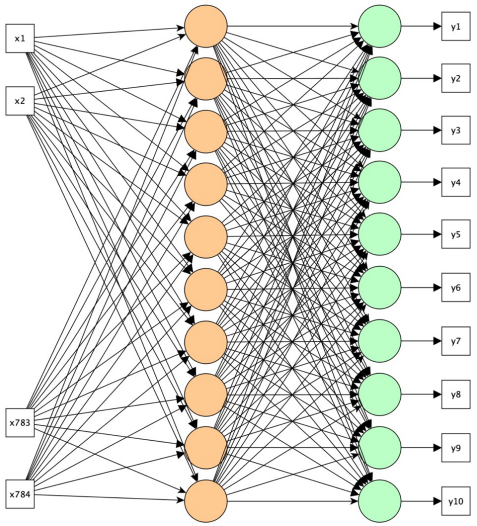
```
_, test_acc = model.evaluate(x_test, y_test)
print(test_acc)
```

```
- 0s 18us/sample - loss: 0.3917 - accuracy: 0.8460
```

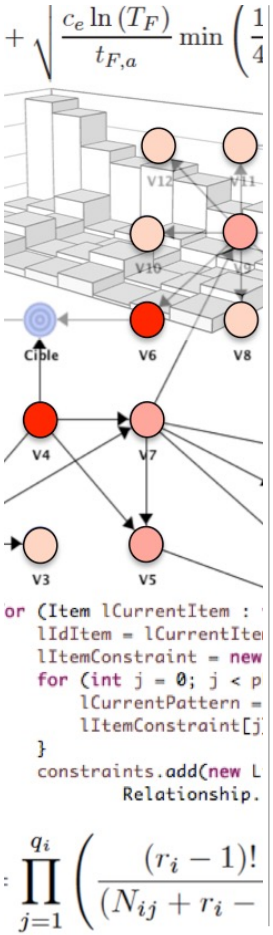
84,60% de précision sur les données de test

↓

Le réseau de neurones classe correctement 8460 images sur les 10000 images des données de test



Classification d'images - Réseaux multicouches



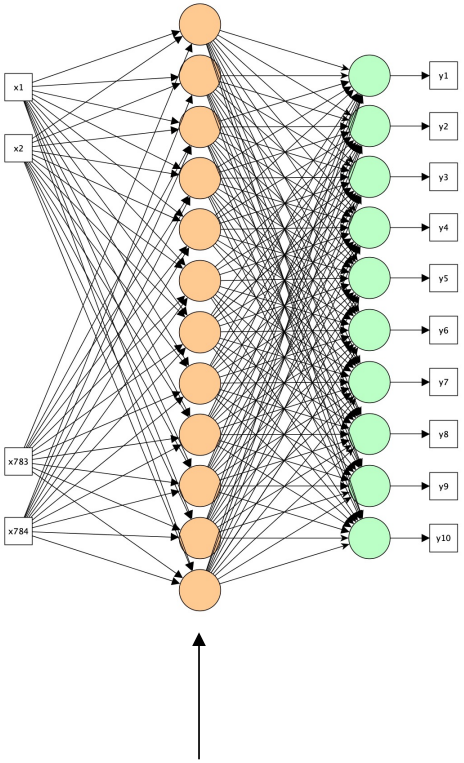
Réseau de neurones : 1 couche cachée, 50 neurones

```
model2 = Sequential()
model2.add(Dense(50, input_dim=784, activation='relu'))
model2.add(Dense(10, activation='softmax'))
model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 50)                39250
_________________________________________________________________
dense_5 (Dense)              (None, 10)                510
=================================================================
Total params: 39,760
Trainable params: 39,760
Non-trainable params: 0
_________________________________________________________________
```

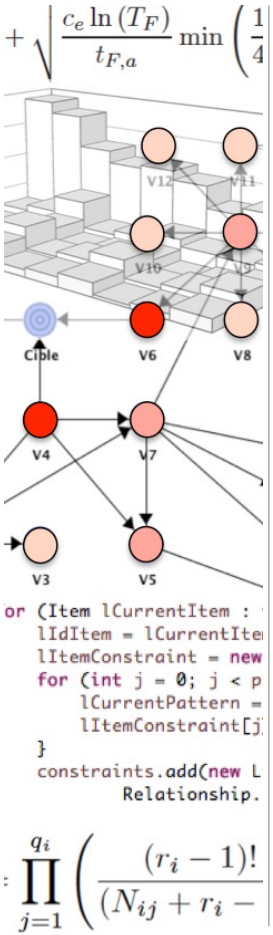
88,05% de précision sur les données de validation

87,43% de précision sur les données de test



50 neurones sur la couche cachée
au lieu de 10

Classification d'images - Réseaux multicouches



Réseau de neurones : 2 couches cachées, 50 neurones

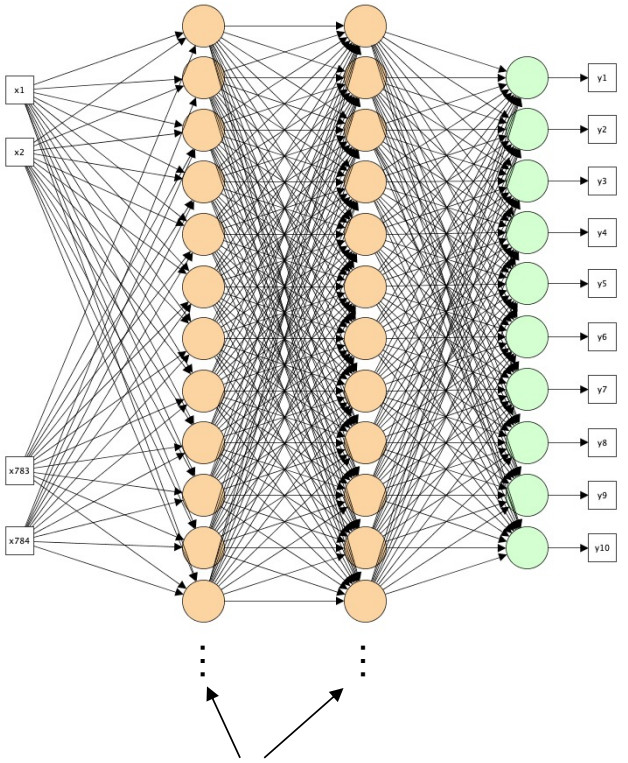
```
model3 = Sequential()
model3.add(Dense(50, input_dim=784, activation='relu'))
model3.add(Dense(50, activation='relu'))
model3.add(Dense(10, activation='softmax'))
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 50)                39250
_________________________________________________________________
dense_7 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_8 (Dense)              (None, 10)                510
=================================================================
Total params: 42,310
Trainable params: 42,310
Non-trainable params: 0
_________________________________________________________________
```

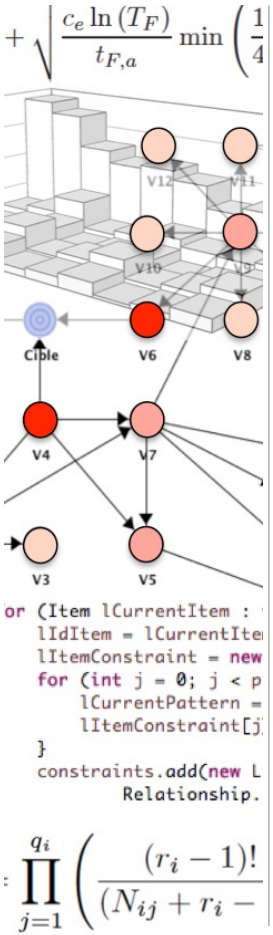
88,25% de précision sur les données de validation

87,64% de précision sur les données de test

Résultats quasi identiques qu'avec une seule couche



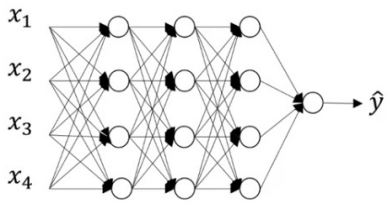
2 couches cachées de
50 neurones chacune

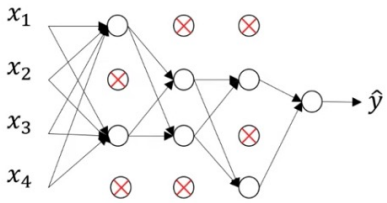


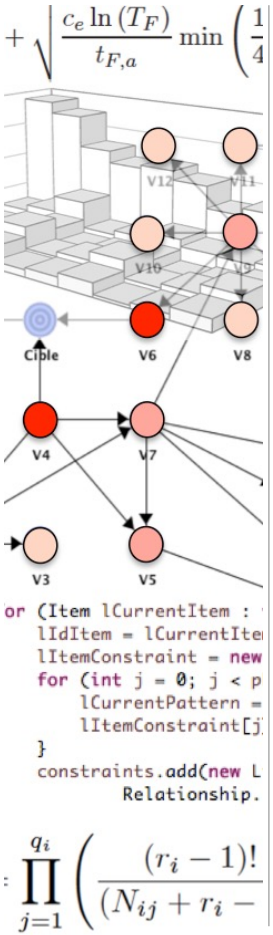
Si les résultats montrent que le modèle n'est pas bon en généralisation (mauvaise précision les données de test), il est possible d'appliquer quelques techniques de régularisation, comme :

- Augmenter artificiellement le jeu de données; dans le cas de l'images on applique des opérateurs de transformation (rotation, translation, zoom, …) :

- Dropout : désactiver une certaine proportion de neurones pendant la phase d'apprentissage



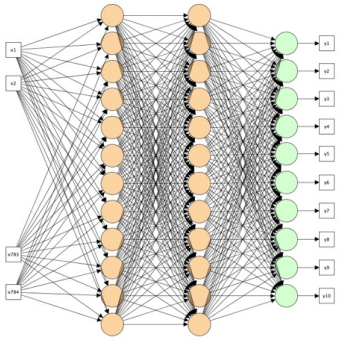




Dropout avec Keras

```
model3 = Sequential()
model3.add(Dense(50, input_dim=784, activation='relu'))
model3.add(Dense(50, activation='relu'))
model3.add(Dense(10, activation='softmax'))
model3.summary()
```

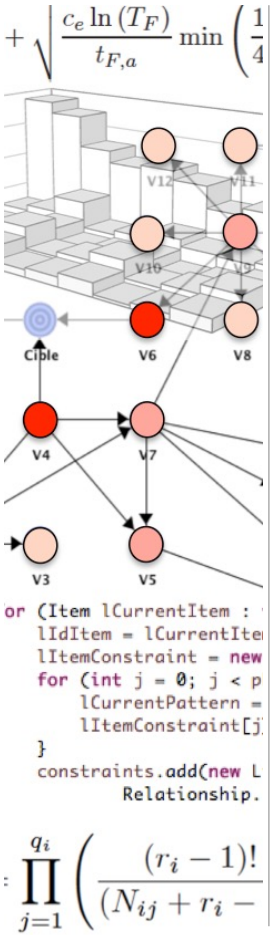
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 50)                39250
_________________________________________________________________
dense_7 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_8 (Dense)              (None, 10)                510
=================================================================
Total params: 42,310
Trainable params: 42,310
Non-trainable params: 0
_________________________________________________________________
```



```
model4 = Sequential()
model4.add(Dense(50, input_dim=784, activation='relu'))
model4.add(Dropout(0.5))
model4.add(Dense(50, activation='relu'))
model4.add(Dropout(0.5))
model4.add(Dense(10, activation='softmax'))
model4.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 50)                39250
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                2550
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                510
=================================================================
Total params: 42,310
Trainable params: 42,310
Non-trainable params: 0
_________________________________________________________________
```

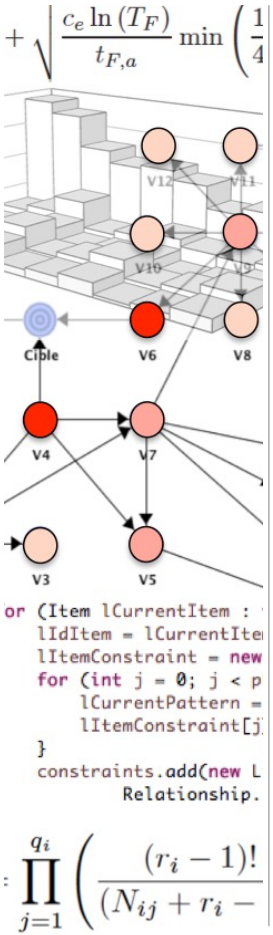
Plan



1. Introduction

2. Classification d'images avec
   - un réseau multi-couches
   - un réseau de neurones convolutifs
   - un réseau pré-entrainé (transfer learning)

3. Détection d'objets dans une image

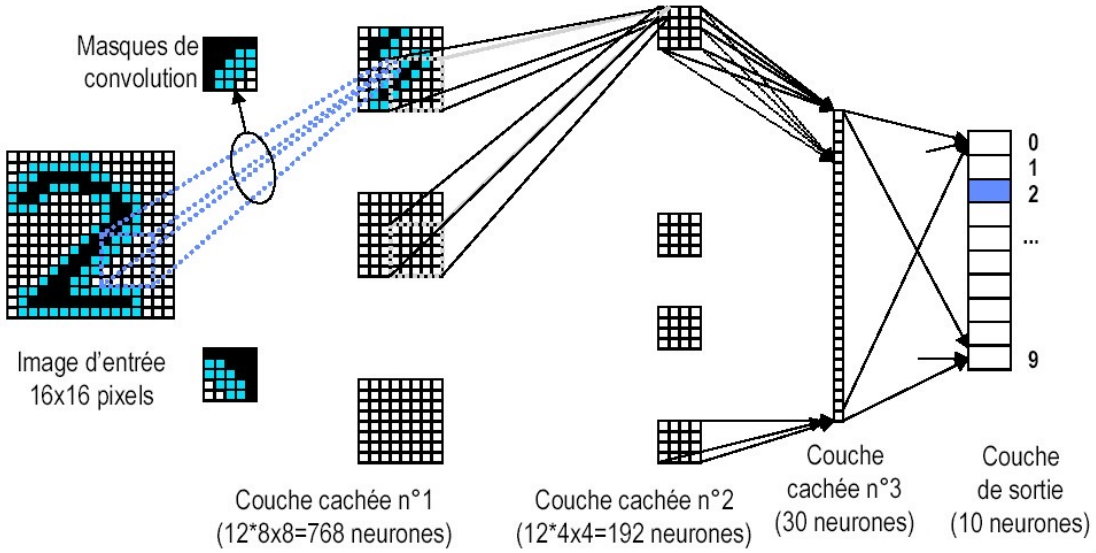


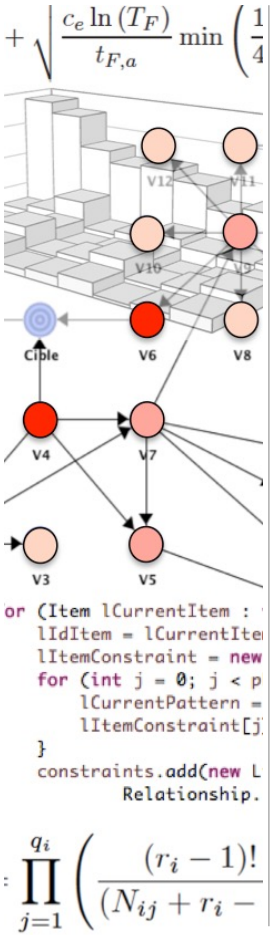
Matrice de taille 28x28 en entrée

Conserver la structure de l'image

Extraire des propriétés liées à la topologie ← Masques de convolution

« LeNet » LeCun 1989

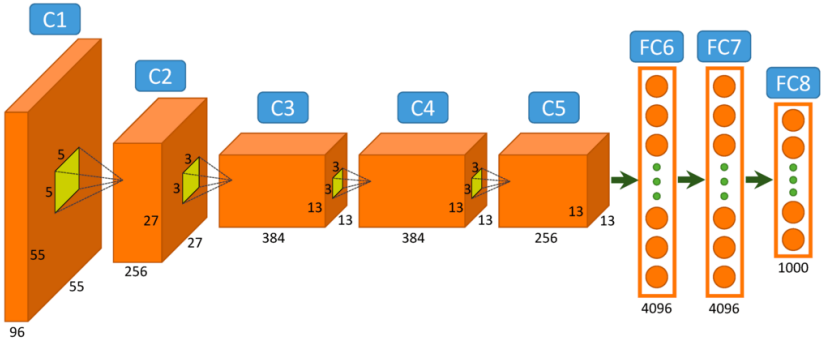


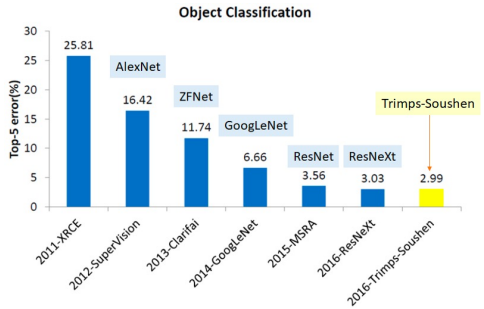


Même si les réseau de neurones convolutifs sont nés à la fin des années 80, en particulier avec « LeNet » de Yann Lecun, il fait attendre 2012 pour que la notion de Deep learning prenne tout son sens.

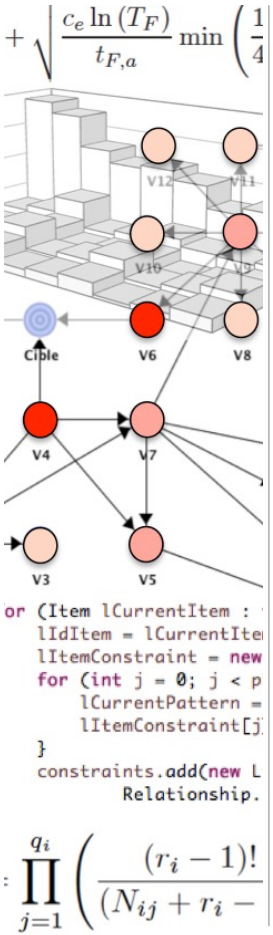
Lors de la compétition de reconnaissance d'images IMAGENET de 2012, des chercheurs ont explosé le record de reconnaissance grâce à une technique efficace de rétropropagation du gradient mais aussi l'usage de grosses puissances de calcul.

https://www.image-net.org/challenges/LSVRC/





Depuis, des équipes ont développés des réseaux avec plusieurs millions de neurones et donc plusieurs milliards de paramètres … et beaucoup de données !



Masques de convolution

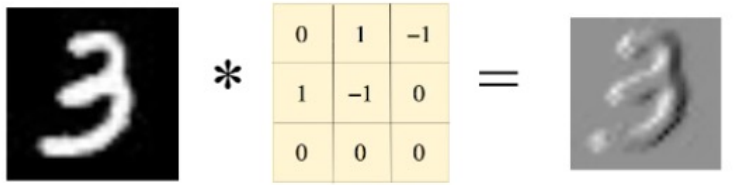
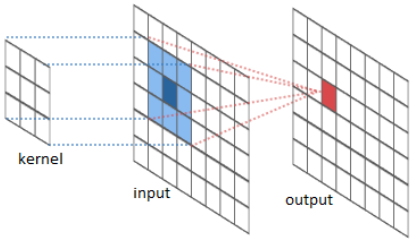
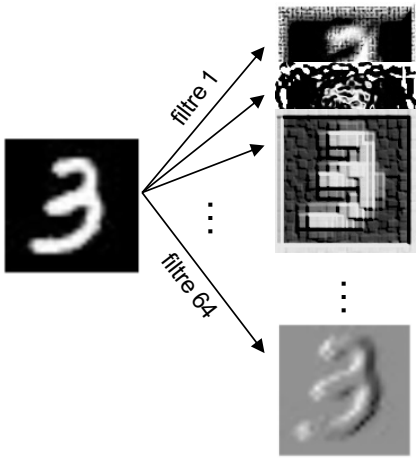


image 28x28

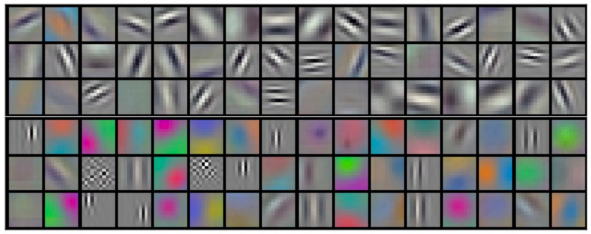
filtre 3x3

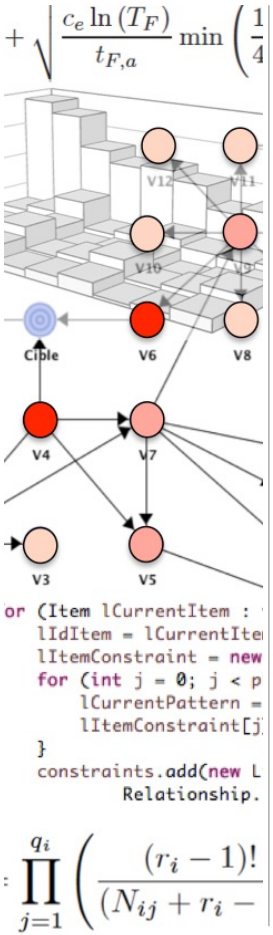
image résultat 26x26 ou 28x28 avec padding



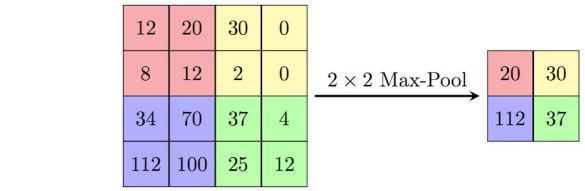


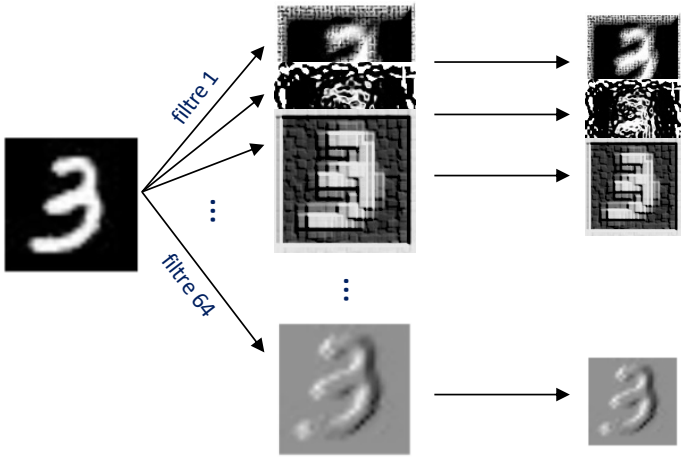
La première couche d'un réseau convolutif consiste à apprendre automatiquement, lors de la phase d'apprentissage, les valeurs des filtres.



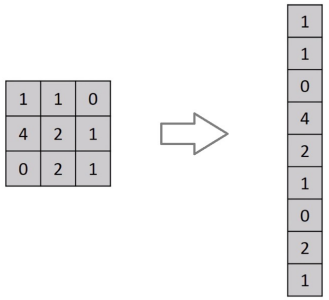


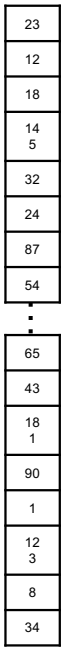
Max pooling : diminuer la taille du réseau

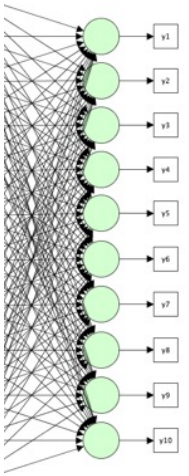


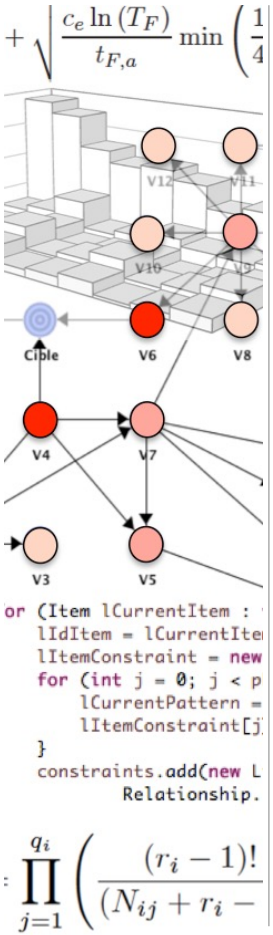


Flatten :

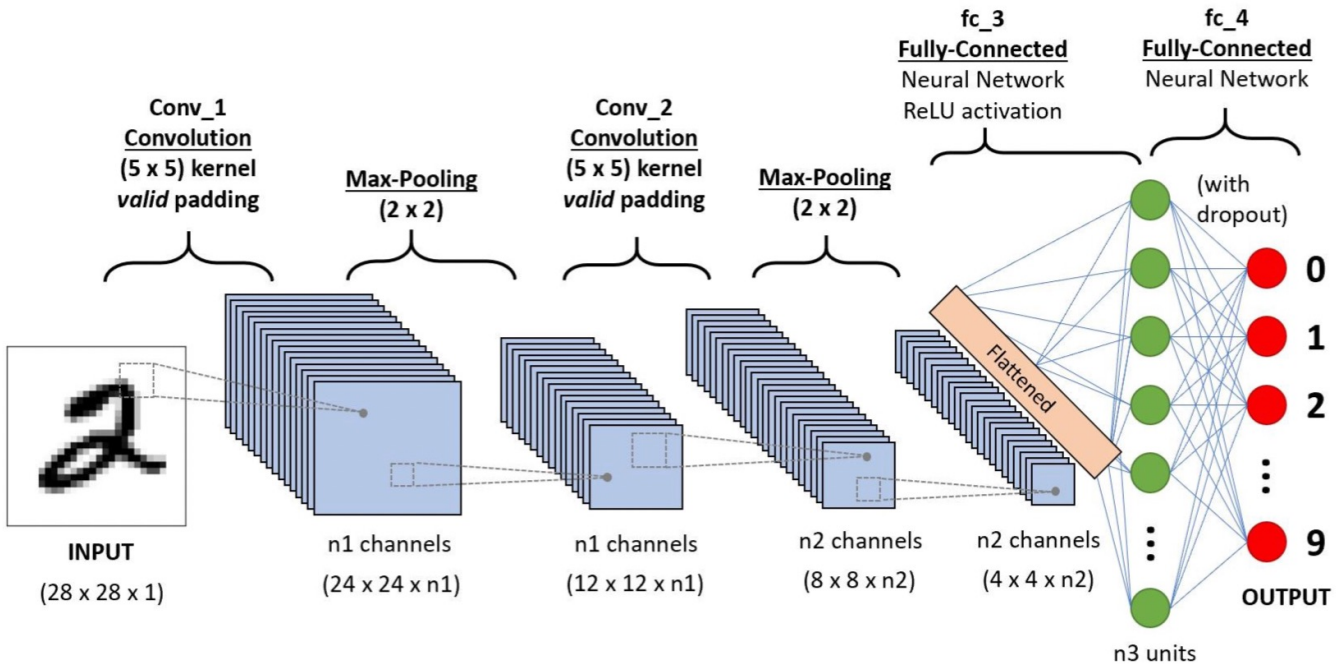




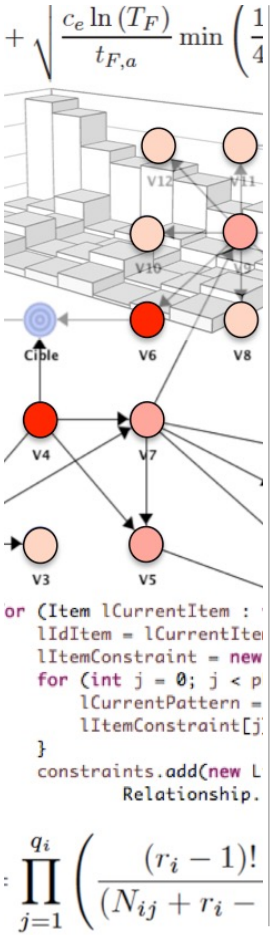




Exemple :



Source : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53



Matrice de taille 28x28 en entrée

1 channel car niveau de gris

54 000 x 28 x 28 x 1

6 000 x 28 x 28 x 1

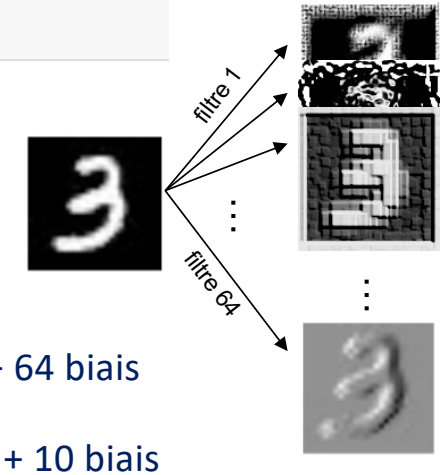
10 000 x 28 x 28 x 1

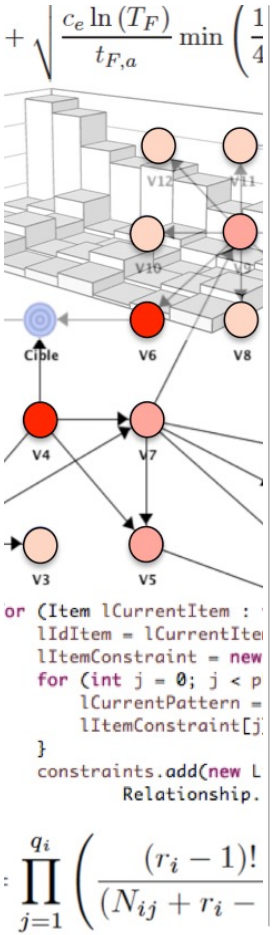
```
model4 = Sequential()
model4.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28,28, 1)))
model4.add(MaxPooling2D(pool_size=2))
model4.add(Flatten())
model4.add(Dense(10, activation='softmax'))
model4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                125450
=================================================================
Total params: 125,770
Trainable params: 125,770
Non-trainable params: 0
_________________________________________________________________
```

64x(2x2) + 64 biais

12544x10 + 10 biais



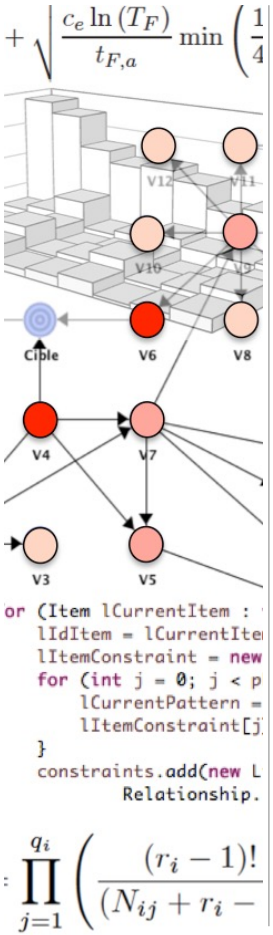



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 19s 357us/sample - loss: 0.4259 - accuracy: 0.8526 - val_loss: 0.3415
- val_accuracy: 0.8795
Epoch 2/10
54000/54000 [==============================] - 20s 370us/sample - loss: 0.3069 - accuracy: 0.8911 - val_loss: 0.3116
- val_accuracy: 0.8907
Epoch 3/10
54000/54000 [==============================] - 19s 345us/sample - loss: 0.2766 - accuracy: 0.9022 - val_loss: 0.2978
- val_accuracy: 0.8932
Epoch 4/10
54000/54000 [==============================] - 20s 376us/sample - loss: 0.2557 - accuracy: 0.9082 - val_loss: 0.2981
- val_accuracy: 0.8922
Epoch 5/10
54000/54000 [==============================] - 20s 374us/sample - loss: 0.2408 - accuracy: 0.9139 - val_loss: 0.2831
- val_accuracy: 0.8985
Epoch 6/10
54000/54000 [==============================] - 20s 378us/sample - loss: 0.2251 - accuracy: 0.9194 - val_loss: 0.2674
- val_accuracy: 0.9040
Epoch 7/10
54000/54000 [==============================] - 21s 392us/sample - loss: 0.2119 - accuracy: 0.9240 - val_loss: 0.2743
- val_accuracy: 0.9013
Epoch 8/10
54000/54000 [==============================] - 21s 387us/sample - loss: 0.2020 - accuracy: 0.9276 - val_loss: 0.2663
- val_accuracy: 0.9075
Epoch 9/10
54000/54000 [==============================] - 21s 394us/sample - loss: 0.1920 - accuracy: 0.9322 - val_loss: 0.2947
- val_accuracy: 0.8958
Epoch 10/10
54000/54000 [==============================] - 21s 393us/sample - loss: 0.1838 - accuracy: 0.9341 - val_loss: 0.2746
- val_accuracy: 0.9035
```

90,4% de précision sur les données de validation

90,2% de précision sur les données de test

Plan

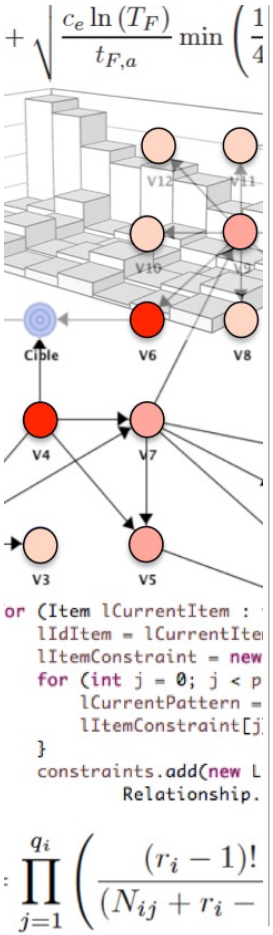


1. Introduction

2. Classification d'images avec
   - un réseau multi-couches
   - un réseau de neurones convolutifs
   - un réseau pré-entrainé (transfer learning)

3. Détection d'objets dans une image

Classification d'images - Transfer Learning



Problème

Reconnaissance et catégorisation d'images en deux categories : chat ou chien

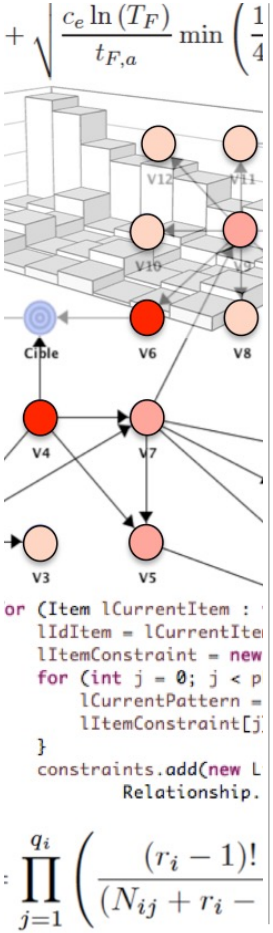
Images en couleur
de taille 150x150

https://www.kaggle.com/c/dogs-vs-cats/data

Contraintes

- On ne dispose que de 3000 images (1500 de chaque catégorie) pour l'apprentissage et 1000 images (500 de chaque catégorie) pour la validation et le test
- On ne dispose pas d'une grosse puissance de calculs

Classification d'images - Transfer Learning

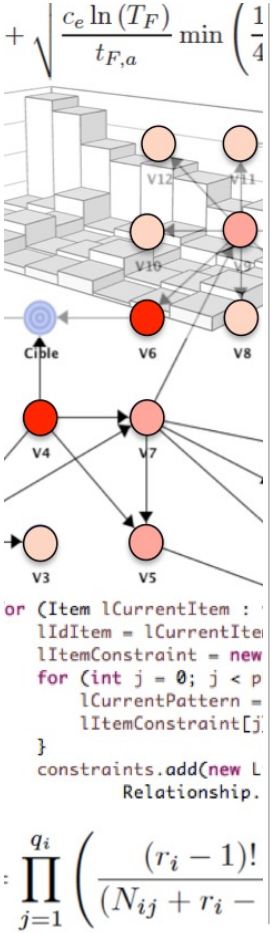


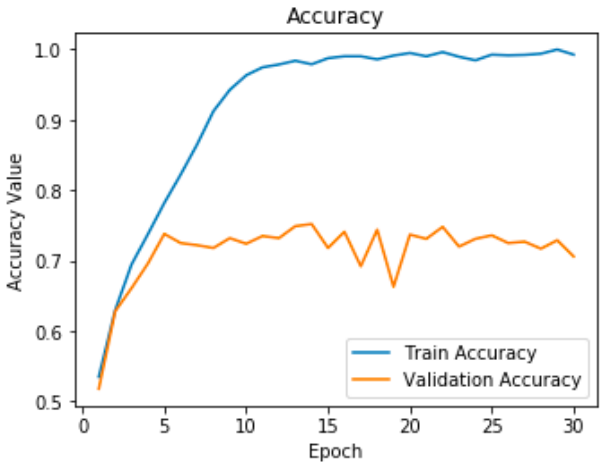
Apprentissage du CNN à partir des 3000 images dédiées à l'apprentissage et des 1000 images pour la validation (les images sont normalisées entre 0 et 1).

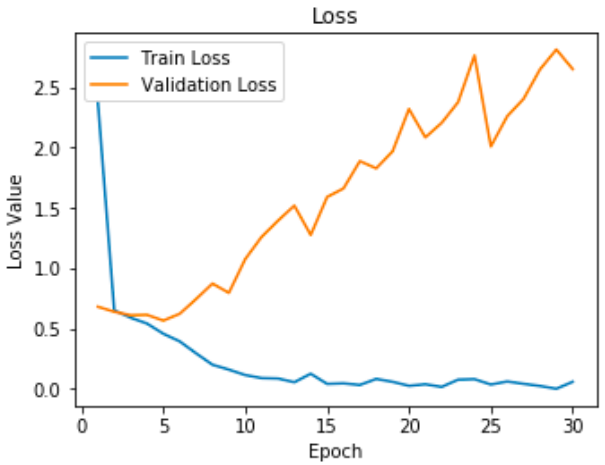
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d_1 (MaxPooling2)	(None, 74, 74, 16)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	9280
max_pooling2d_2 (MaxPooling2)	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2)	(None, 17, 17, 128)	0
flatten_1 (Flatten)	(None, 36992)	0
dense_1 (Dense)	(None, 512)	18940416
dense_2 (Dense)	(None, 1)	513

Total params: 19,024,513
Trainable params: 19,024,513
Non-trainable params: 0

Classification d'images - Transfer Learning

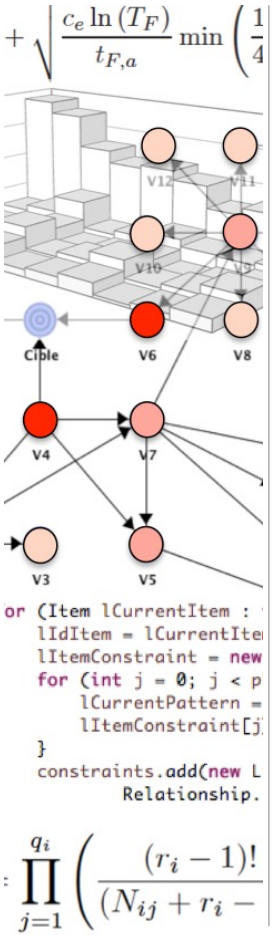






On observe un surapprentissage après 3 epochs
Néanmoins, la precision moyenne sur les données de validation est de l'ordre de 71%

Classification d'images - Transfer Learning



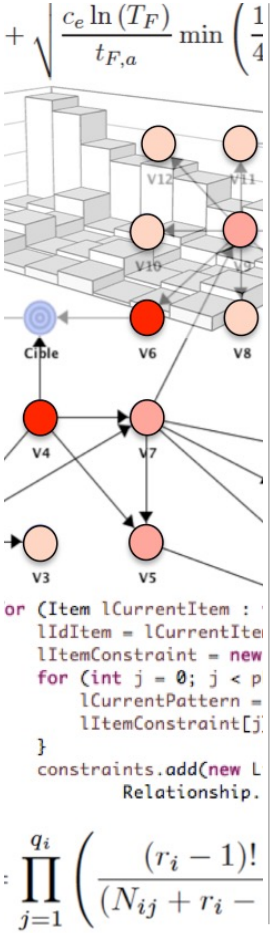
Ajout d'une couche supplémentaire mais surtout ajout d'un processus de dropout à 0.3 après chaque couche cachée dense

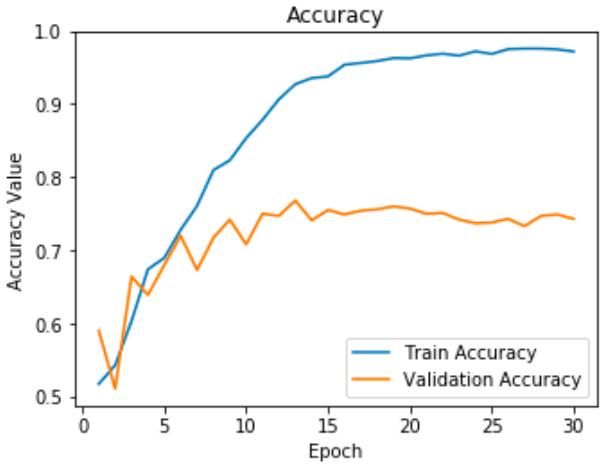
Apprentissage du CNN à partir des 3000 images dédiées à l'apprentissage et des 1000 images pour la validation (les images sont normalisées entre 0 et 1).

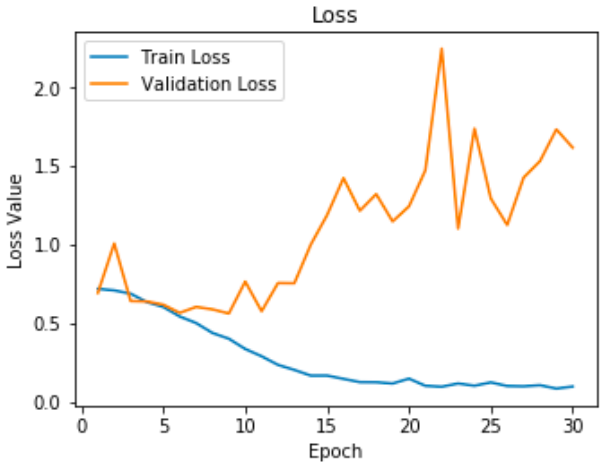
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 16)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	9280
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_2 (Dense)	(None, 512)	3211776
dropout_1 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 1)	513

Total params: 3,706,113
Trainable params: 3,706,113
Non-trainable params: 0

Classification d'images - Transfer Learning

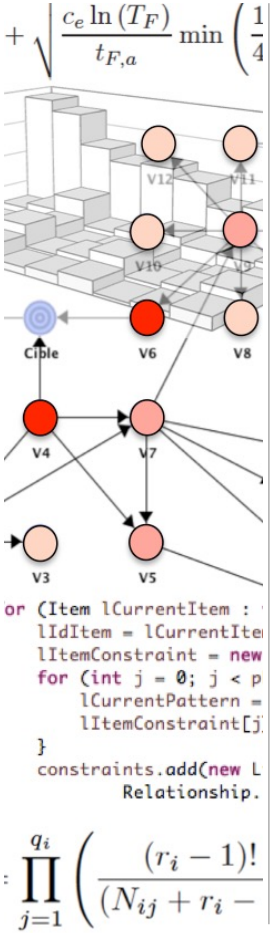




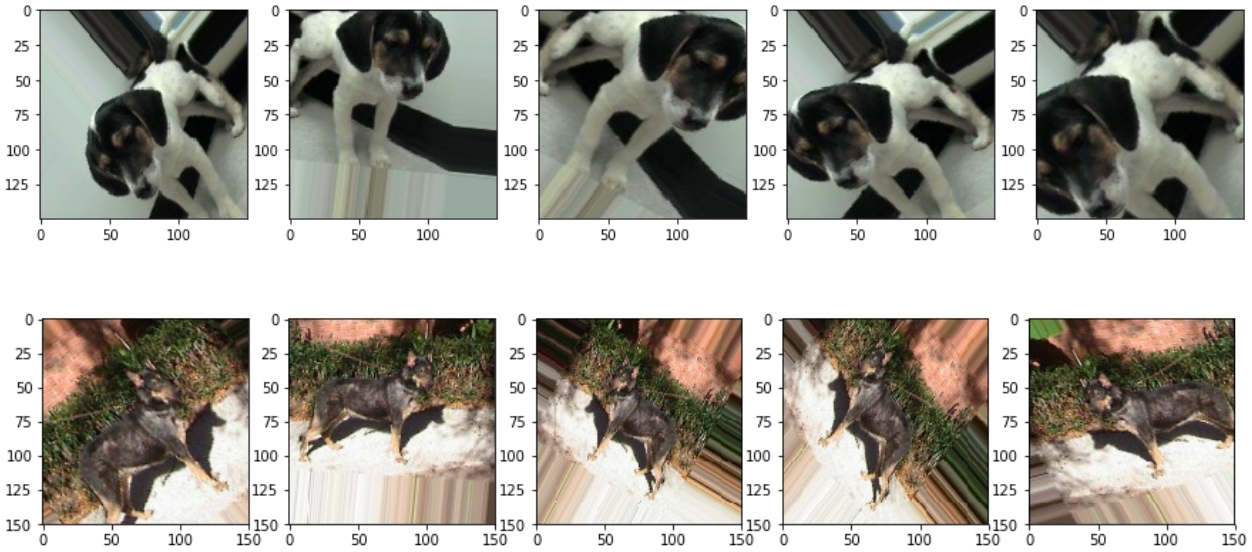


Il y a toujours un surapprentissage mais après plus d'epochs
Néanmoins, la precision moyenne sur les données de validation est un peu plus élevée de l'ordre de 74%

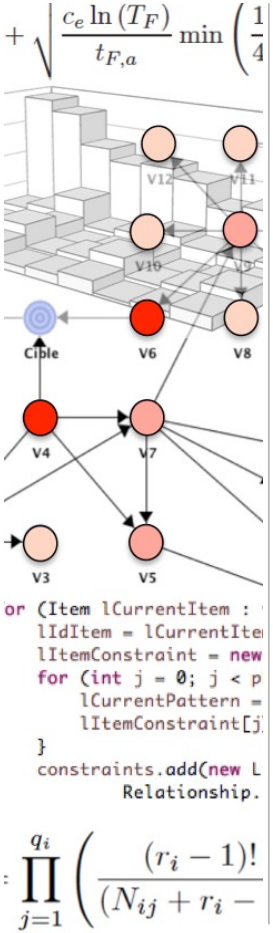
Classification d'images - Transfer Learning

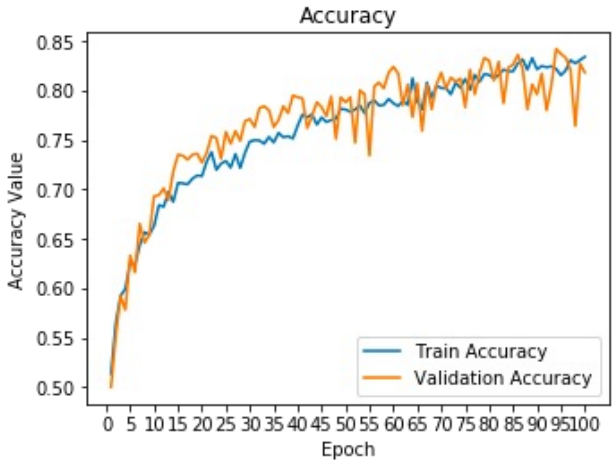


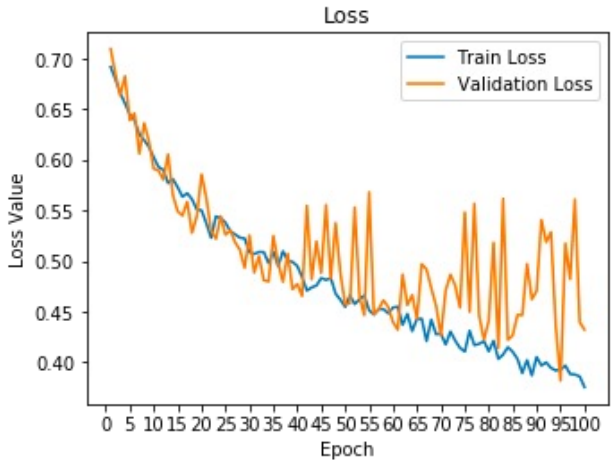
L'idée est de créer de nouvelles images à partir des 3000 images d'apprentissage en appliquant des opérateurs de rotation, de translation, de zoom, … afin d'augmenter le nombre d'images tout en produisant des Nouvelles.



Classification d'images - Transfer Learning

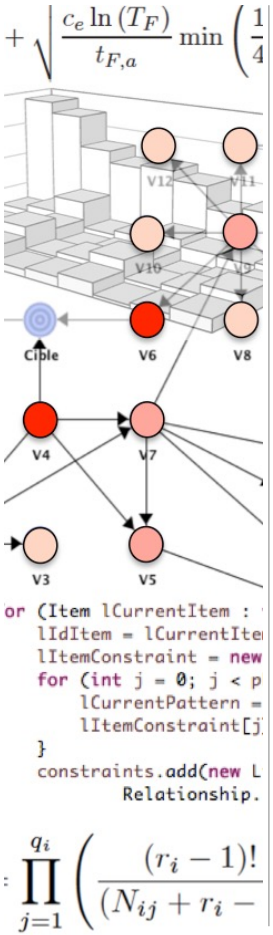


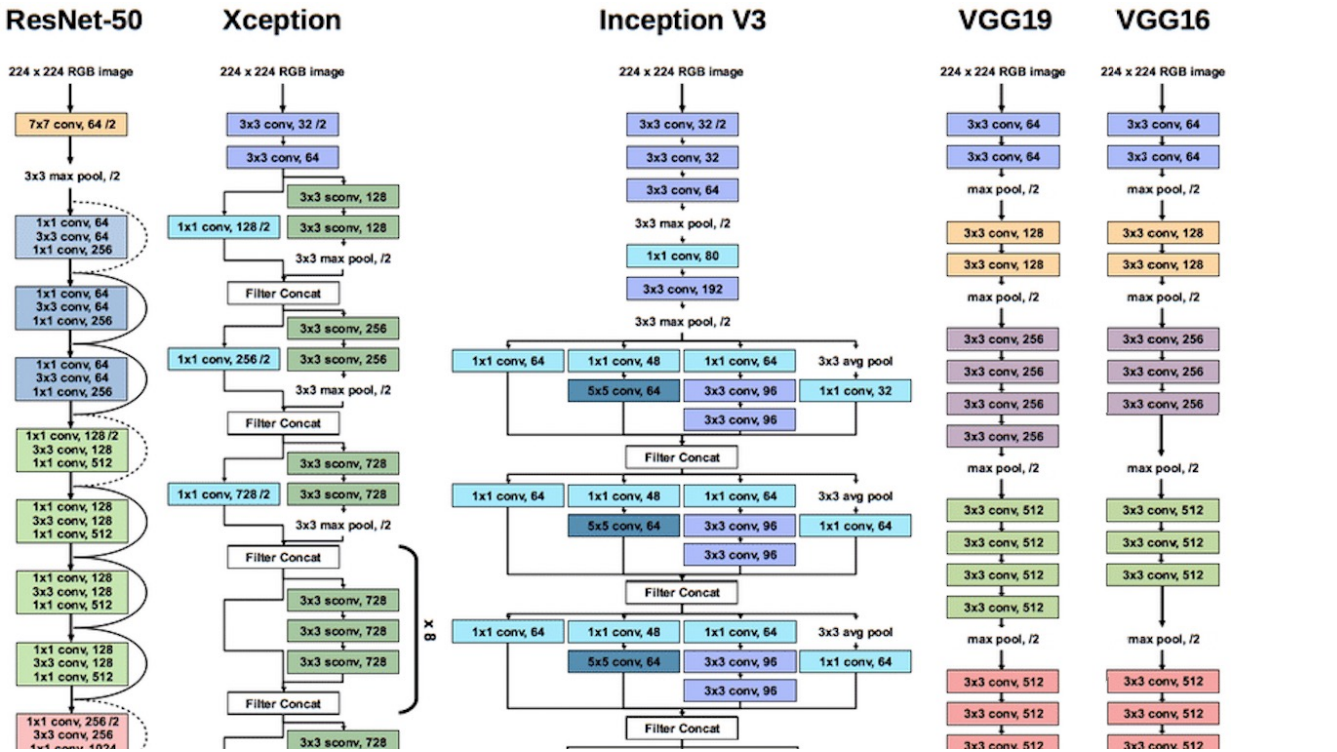




Il n'y a plus de surapprentissage !
La precision moyenne sur les données de validation est encore un plus élevée de l'ordre de 80%

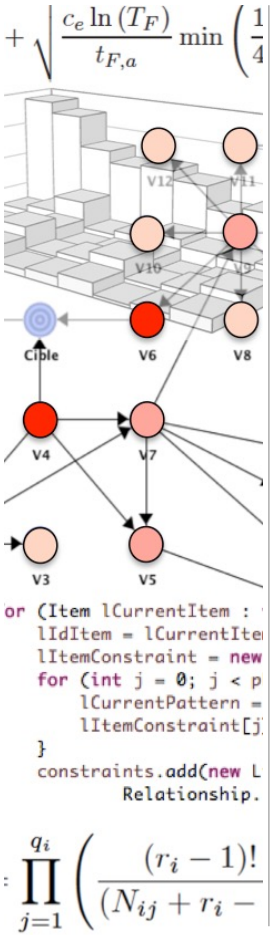
Classification d'images - Transfer Learning

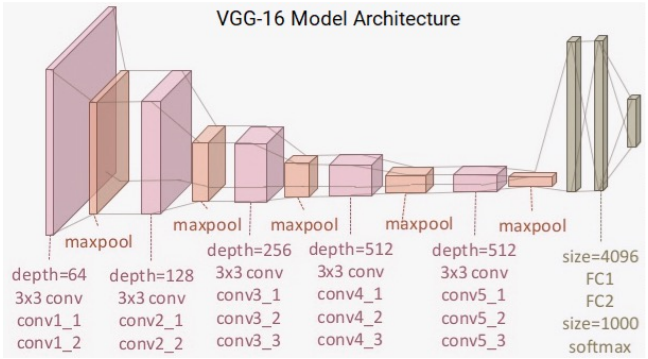




https://www.researchgate.net/publication/330478807_Deep_Feature-Based_Classifiers_for_Fruit_Fly_Identification_Diptera_Tephritidae/download

Classification d'images - Transfer Learning





Published as a conference paper at ICLR 2015

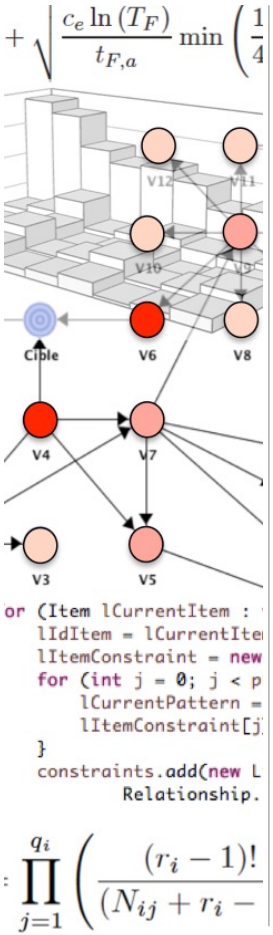
https://arxiv.org/abs/1409.1556

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

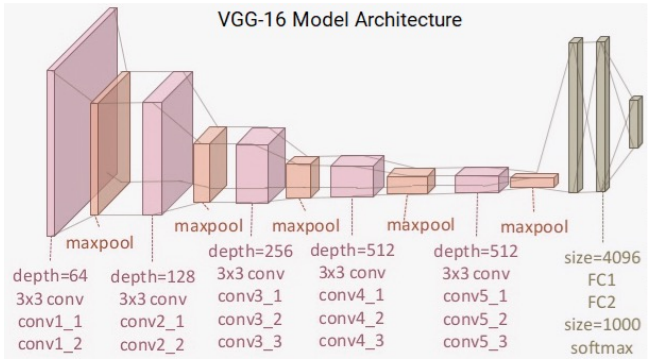
Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

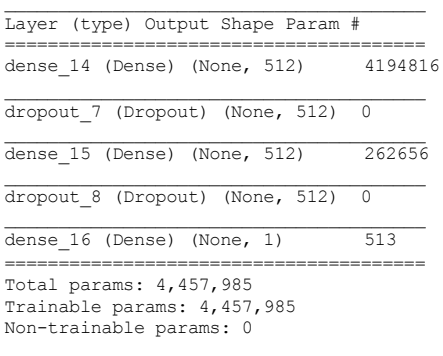
Apprentissage à partir de 1.3M d'images
50K images pour la validation
100K images pour les tets

```
Layer (type) Output Shape Param #
=================================================================
input_1 (InputLayer) (None, 150, 150, 3) 0
_________________________________________________________________
block1_conv1 (Conv2D) (None, 150, 150, 64) 1792
_________________________________________________________________
block1_conv2 (Conv2D) (None, 150, 150, 64) 36928
_________________________________________________________________
block1_pool (MaxPooling2D) (None, 75, 75, 64) 0
_________________________________________________________________
block2_conv1 (Conv2D) (None, 75, 75, 128) 73856
_________________________________________________________________
block2_conv2 (Conv2D) (None, 75, 75, 128) 147584
_________________________________________________________________
block2_pool (MaxPooling2D) (None, 37, 37, 128) 0
_________________________________________________________________
block3_conv1 (Conv2D) (None, 37, 37, 256) 295168
_________________________________________________________________
block3_conv2 (Conv2D) (None, 37, 37, 256) 590080
_________________________________________________________________
block3_conv3 (Conv2D) (None, 37, 37, 256) 590080
_________________________________________________________________
block3_pool (MaxPooling2D) (None, 18, 18, 256) 0
_________________________________________________________________
block4_conv1 (Conv2D) (None, 18, 18, 512) 1180160
_________________________________________________________________
block4_conv2 (Conv2D) (None, 18, 18, 512) 2359808
_________________________________________________________________
block4_conv3 (Conv2D) (None, 18, 18, 512) 2359808
_________________________________________________________________
block4_pool (MaxPooling2D) (None, 9, 9, 512) 0
_________________________________________________________________
block5_conv1 (Conv2D) (None, 9, 9, 512) 2359808
_________________________________________________________________
block5_conv2 (Conv2D) (None, 9, 9, 512) 2359808
_________________________________________________________________
block5_conv3 (Conv2D) (None, 9, 9, 512) 2359808
_________________________________________________________________
block5_pool (MaxPooling2D) (None, 4, 4, 512) 0
_________________________________________________________________
flatten_6 (Flatten) (None, 8192) 0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```



Modèle VGG-16 utilisé comme extracteur de features





Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 512)	4194816
dropout_7 (Dropout)	(None, 512)	0
dense_15 (Dense)	(None, 512)	262656
dropout_8 (Dropout)	(None, 512)	0
dense_16 (Dense)	(None, 1)	513

Total params: 4,457,985
Trainable params: 4,457,985
Non-trainable params: 0

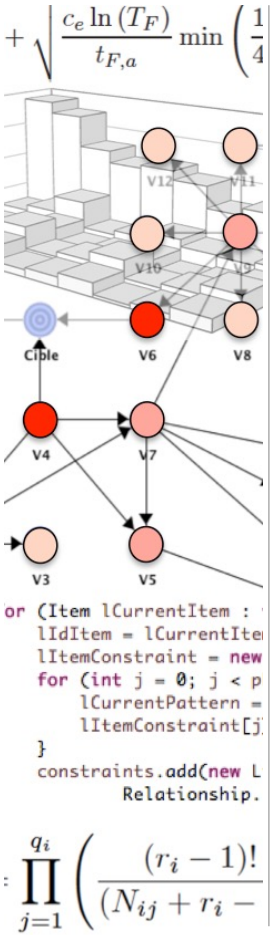
Extraction de caractéristiques
modèle pré-entrainé inchangé

Classification
modèle à entrainer

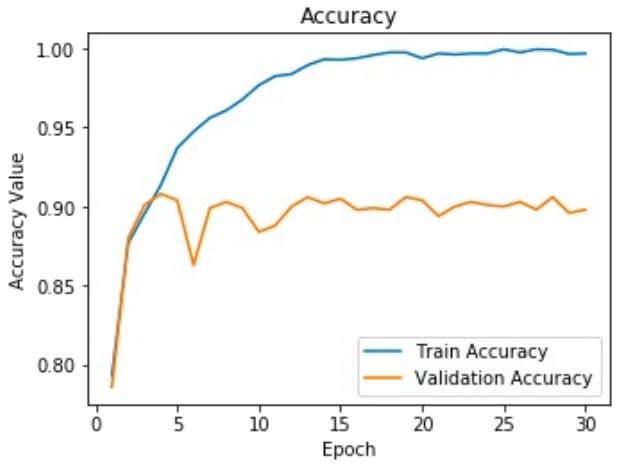
<u>Intérêt</u> :

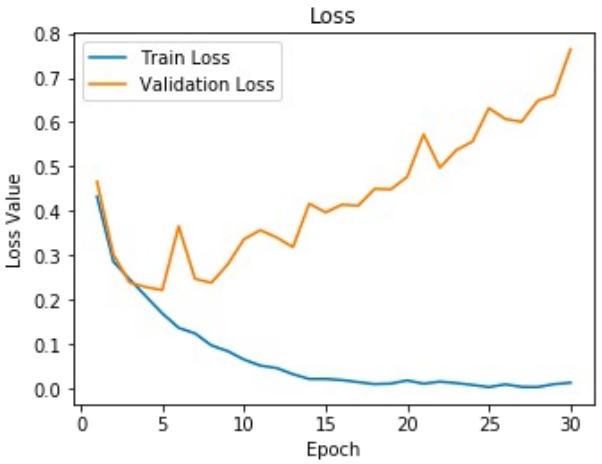
- Profiter d'un réseau plus efficace, en terme d'extraction de caractéristiques, qu'un réseau qu'on aurait construit nous même avec nos propres données
- Ne réaliser l'apprentissage que sur quelques couches (compatible avec notre puissance de calculs et notre quantité de données)

Classification d'images - Transfer Learning

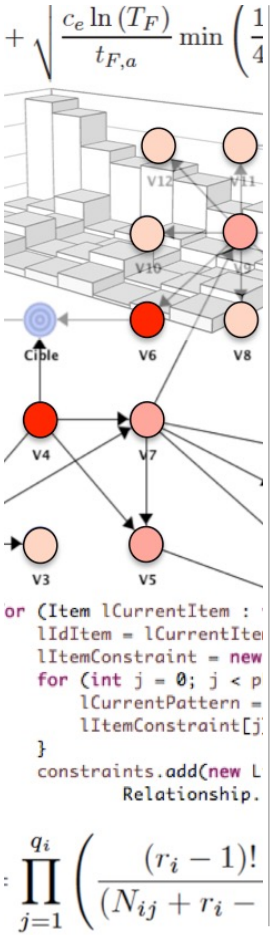


Modèle VGG-16 utilisé comme extracteur de features

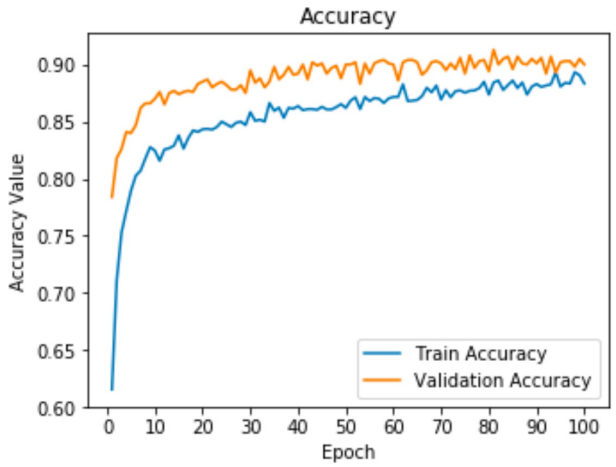


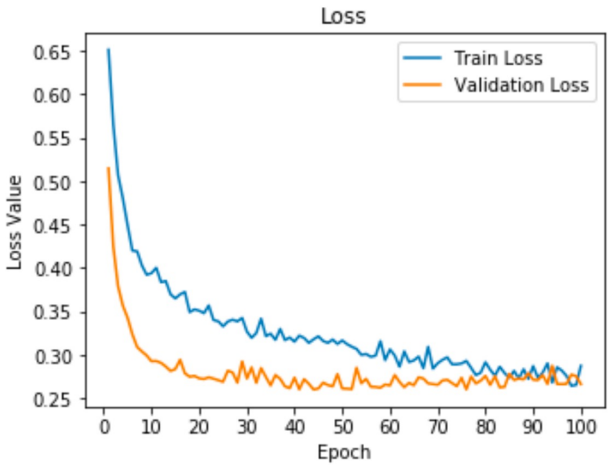


Il y a de nouveau du surapprentissage
Par contre, la precision moyenne sur les données de validation est de l'ordre de 88%



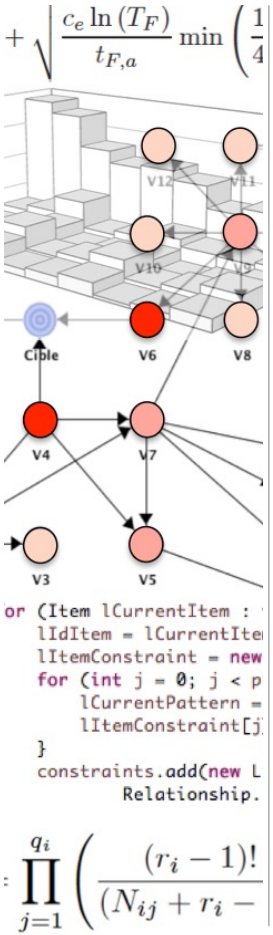
Modèle VGG-16 utilisé comme extracteur de features et + d'images





Il n'y a plus de surapprentissage

La precision moyenne sur les données de validation est de l'ordre de 90%



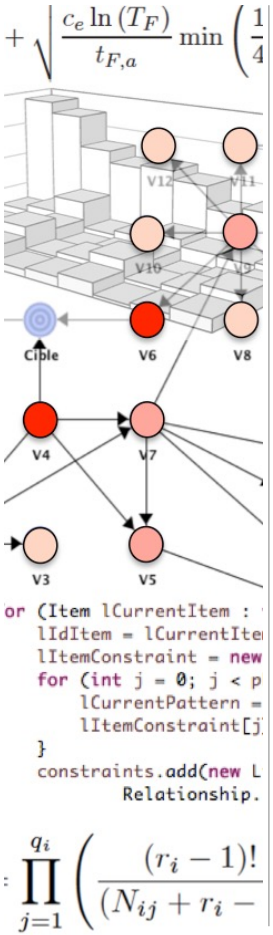
Modèle VGG-16 utilisé comme extracteur de features et + d'images et Fine-Tuning

	Layer Type	Layer Name	Layer Trainable
0	<keras.engine.input_layer.InputLayer object at 0xb499b8f98>	input_1	False
1	<keras.layers.convolutional.Conv2D object at 0xb4b7f8668>	block1_conv1	False
2	<keras.layers.convolutional.Conv2D object at 0xb4b6e4b38>	block1_conv2	False
3	<keras.layers.pooling.MaxPooling2D object at 0xb4b6f8588>	block1_pool	False
4	<keras.layers.convolutional.Conv2D object at 0xb4b6f8470>	block2_conv1	False
5	<keras.layers.convolutional.Conv2D object at 0xb4b9e0ac8>	block2_conv2	False
6	<keras.layers.pooling.MaxPooling2D object at 0xb4b9f7eb8>	block2_pool	False
7	<keras.layers.convolutional.Conv2D object at 0xb4b9f7c88>	block3_conv1	False
8	<keras.layers.convolutional.Conv2D object at 0xb56c23d68>	block3_conv2	False
9	<keras.layers.convolutional.Conv2D object at 0xb56c40f28>	block3_conv3	False
10	<keras.layers.pooling.MaxPooling2D object at 0xb56c72710>	block3_pool	False
11	<keras.layers.convolutional.Conv2D object at 0xb56c72748>	block4_conv1	True
12	<keras.layers.convolutional.Conv2D object at 0xb56ca2748>	block4_conv2	True
13	<keras.layers.convolutional.Conv2D object at 0xb56cc2b70>	block4_conv3	True
14	<keras.layers.pooling.MaxPooling2D object at 0xb56cf50b8>	block4_pool	True
15	<keras.layers.convolutional.Conv2D object at 0xb56cf57b8>	block5_conv1	True
16	<keras.layers.convolutional.Conv2D object at 0xb56d2c390>	block5_conv2	True
17	<keras.layers.convolutional.Conv2D object at 0xb56d48630>	block5_conv3	True
18	<keras.layers.pooling.MaxPooling2D object at 0xb56d65a58>	block5_pool	True
19	<keras.layers.core.Flatten object at 0xb56dd5f28>	flatten_6	True

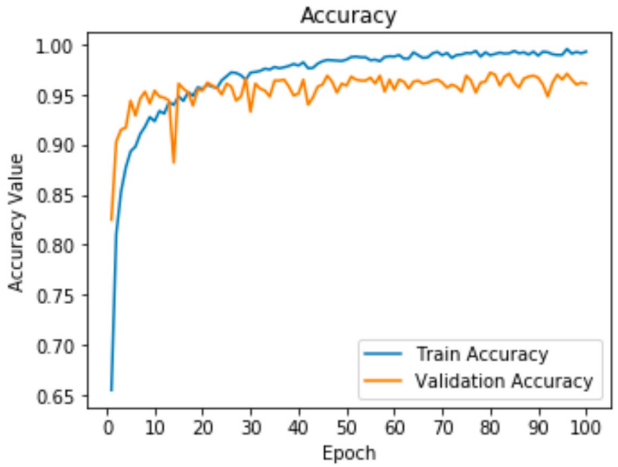
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 8192)              14714688
_________________________________________________________________
dense_23 (Dense)             (None, 512)               4194816
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_24 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_14 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_25 (Dense)             (None, 1)                 513
=================================================================
Total params: 19,172,673
Trainable params: 17,437,185
Non-trainable params: 1,735,488
_________________________________________________________________
```

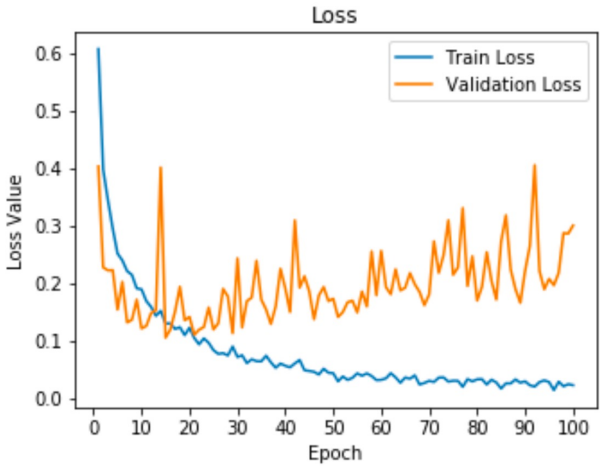
Classification

Classification d'images - Transfer Learning



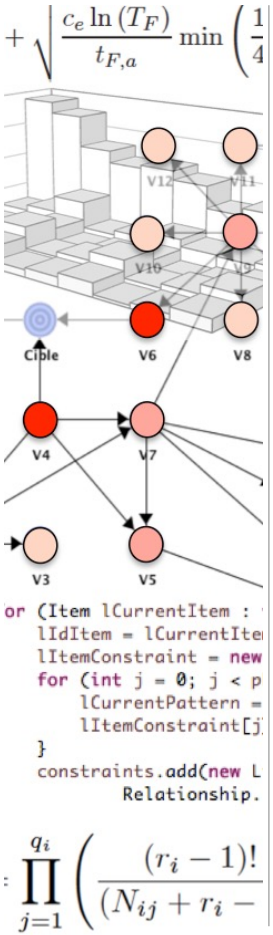
Modèle VGG-16 utilisé comme extracteur de features et + d'images et Fine-Tuning





La precision moyenne sur les données de validation est maintenant de l'ordre de 96%

Classification d'images - Transfer Learning



Plusieurs modèles, dédiés à la classification d'images, ont été construits sur ImageNet et mis à disposition (Google Inception model, Microsoft ResNet model, Oxford VGG model, … .).

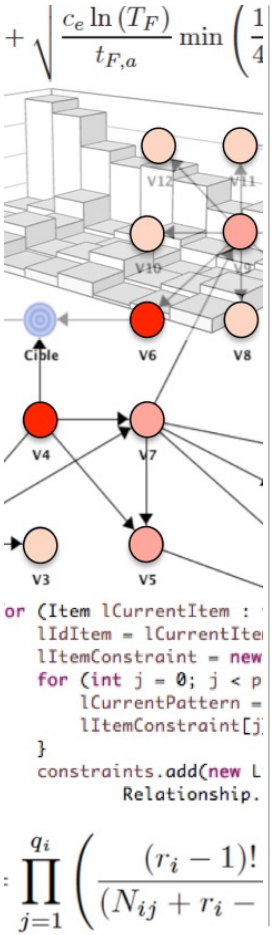
Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42	
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130	
MobileNet	16	70.4%	89.5%	4.3M	55	22	
MobileNetV2	14	71.3%	90.1%	3.5M	105	25	

Keras

keras.applications.inception_v3.InceptionV3(include_top=True, weights='imagenet', input_tensor=None, input_shape=None, pooling=None, classes=1000)

https://keras.io/api/applications/

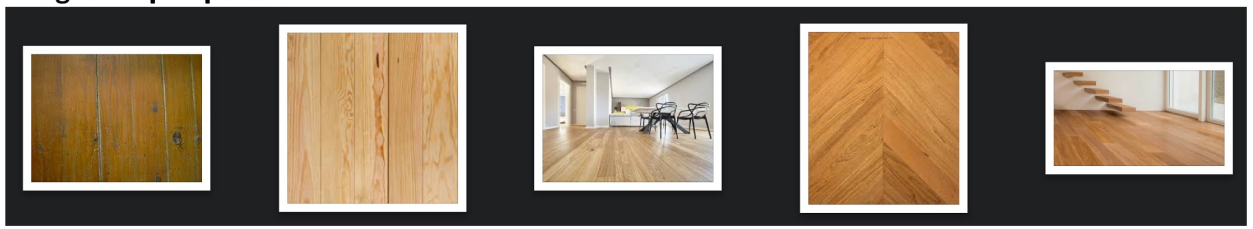
Classification d'images - Transfer Learning



Applications réelles :

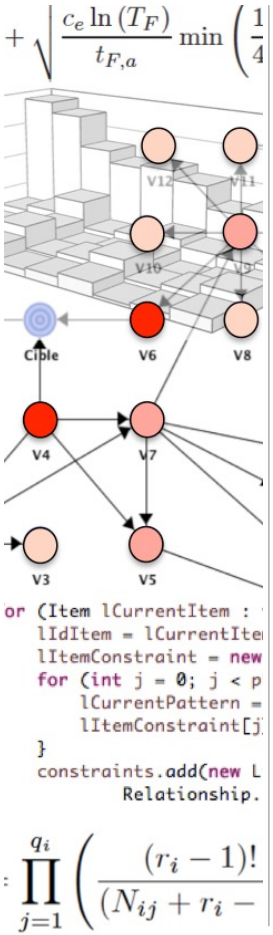
Images de **carrelages** :

Images de **parquets** :



Même avec seulement 500 images de chaque catégorie, on atteint 91% de precision moyenne.

Classification d'images - Transfer Learning



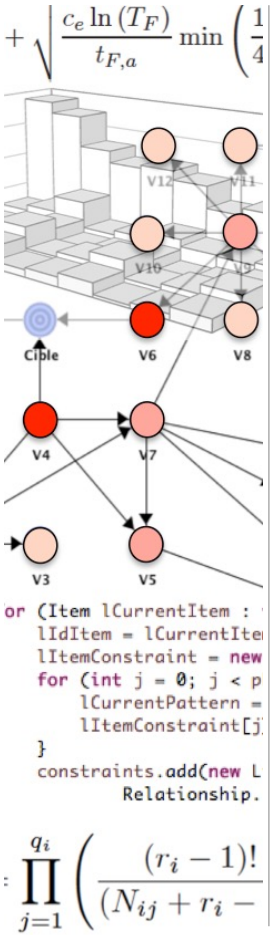
Applications réelles :

Images de "berline" :

Images de "pickup / 4x4" :

Avec 1000 images de chaque catégorie, on atteint seulement 74% de precision moyenne.

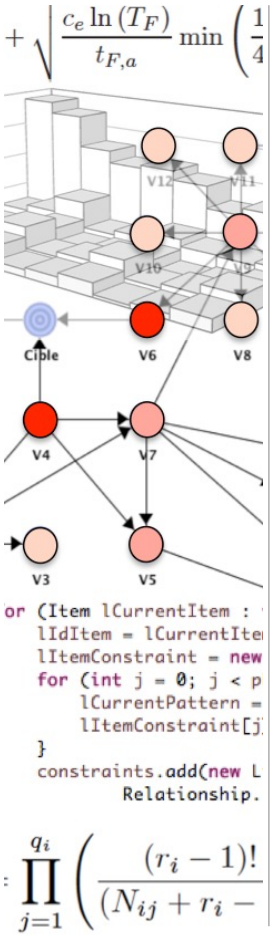
Plan



1. Introduction

2. Classification d'images avec
   - un réseau multi-couches
   - un réseau de neurones convolutifs
   - un réseau pré-entrainé (transfer learning)

3. Détection d'objets dans une image

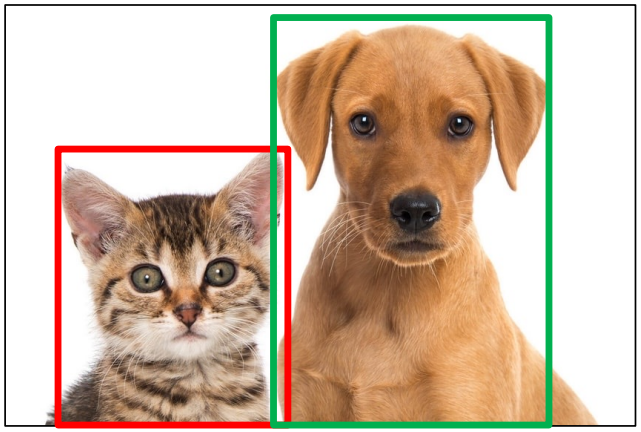


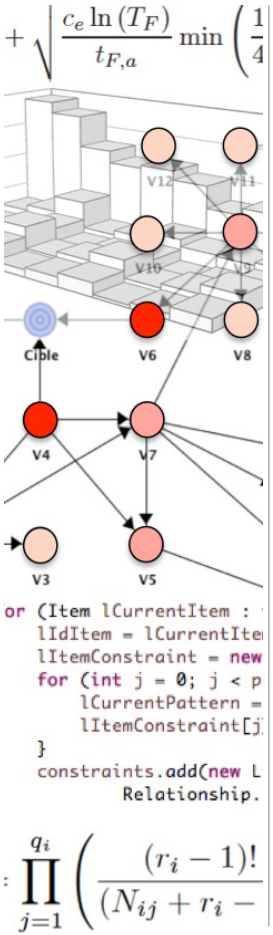
Classification

Chat

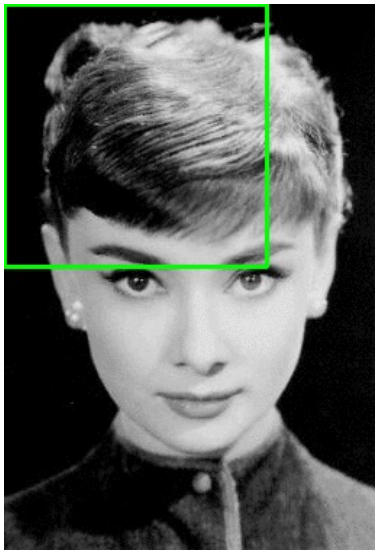
Chien

Détection
=
Localisation + Classification



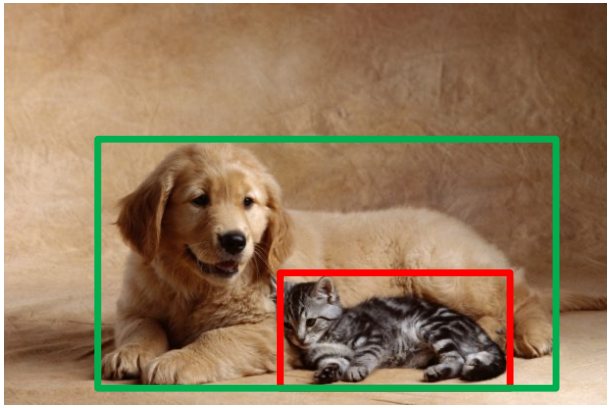


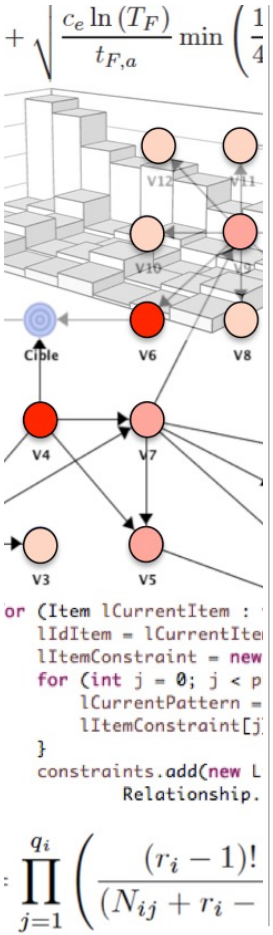
Pour une fenêtre glissante de taille donnée, la détection pourrait être une suite de classifications d'images obtenues en déplaçant la fenêtre dans l'image :



Problème

les objets sont de tailles variées

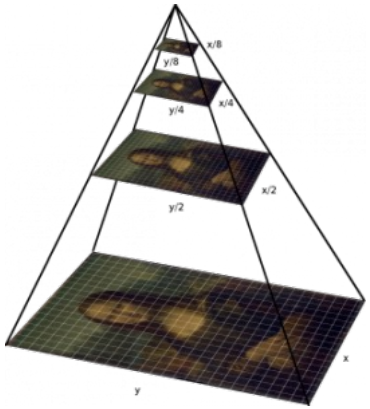




Idée :

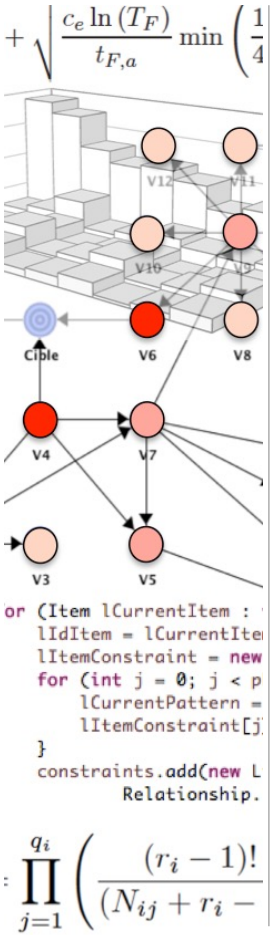
A partir de l'image originelle,
construire une « pyramide » d'images à différentes échelles de plus en plus petites (généralement de 64 niveaux) et,
avec une fenêtre glissante de taille fixe,
parcourir ces différentes images pour détecter les objets.

Inconvénient : les objets n'ont pas tous la même forme



Il existe de nombreux algorithmes dont :

- Histogram of Oriented Gradients (HOG)
- Region-based Convolutional Neural Networks (R-CNN)
- Spatial Pyramid Pooling (SPP-net)
- Fast R-CNN
- Faster R-CNN
- You only Look Once (YOLO)
- Single Shot MultiBox Detector (SSD)
- RetinaNet

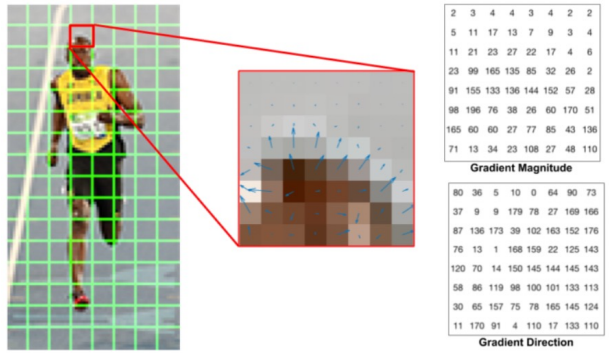


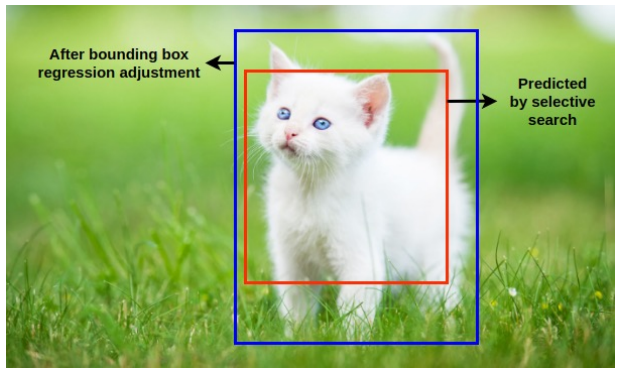
Histogram of Oriented Gradients (HOG) Navneet Dalal and Bill Triggs (2005)

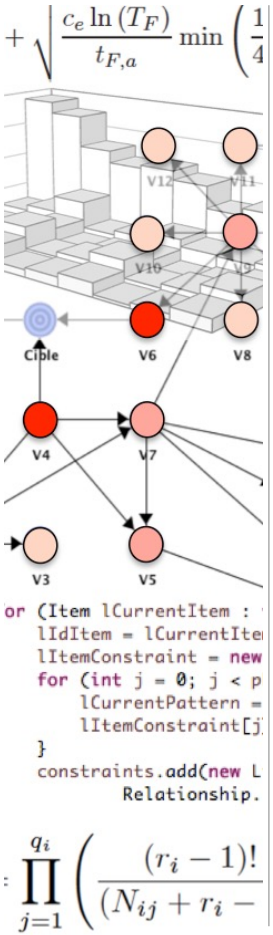
Sur chaque fenêtre glissante :

- calcul de paramètres (Hog features)
- mis ensuite en entrée d'un SVM qui détermine la classe
- puis ajustement de la zone par :

Bounding box regression

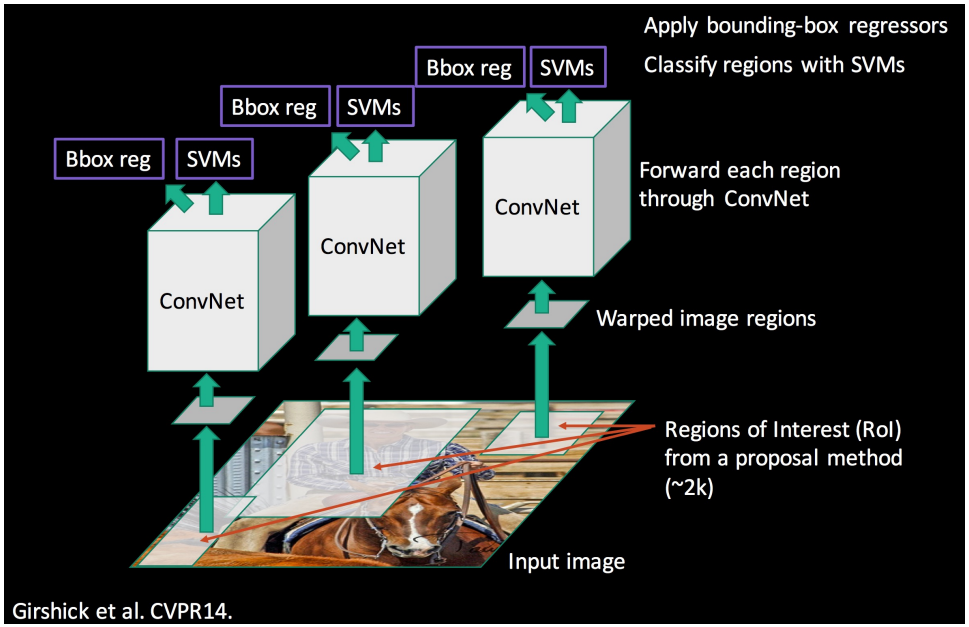






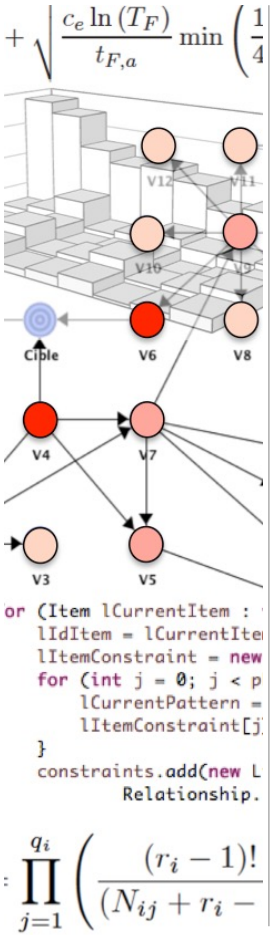
Region-based Convolutional Neural Networks (R-CNN)

Sélection de 2000 zones dans la pyramide ("Selective Search") sur lesquelles les calculs de paramètres sont remplacés par un CNN, puis un SVM pour le classement :

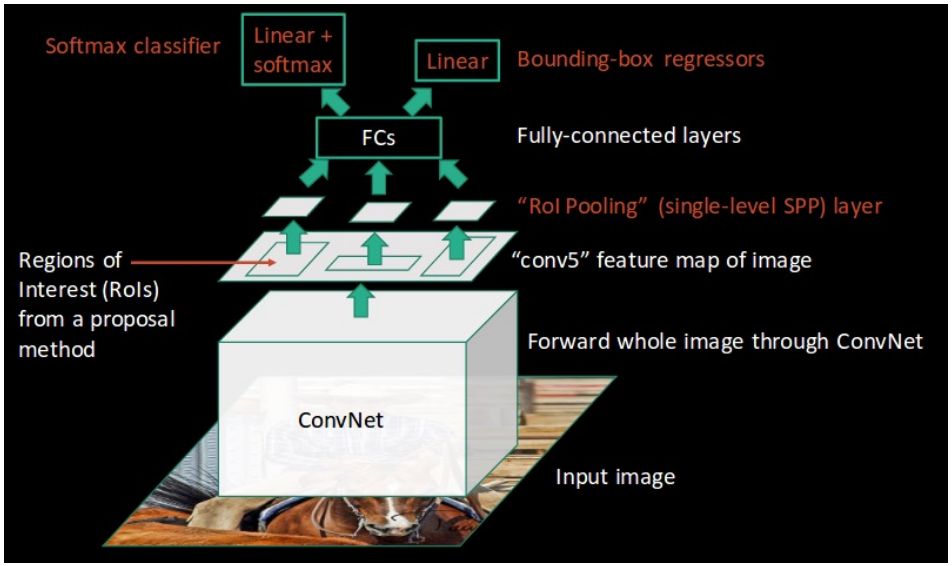


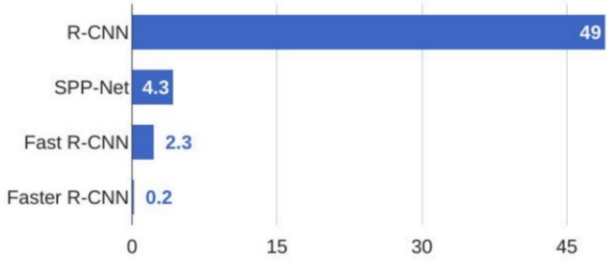
Très lent !

Source : https://arxiv.org/abs/1311.2524

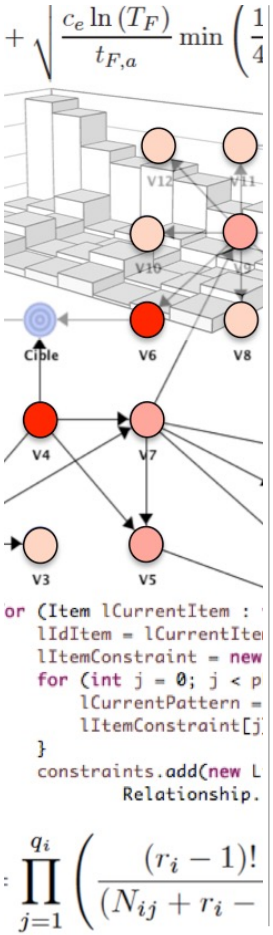


Faster R-CNN

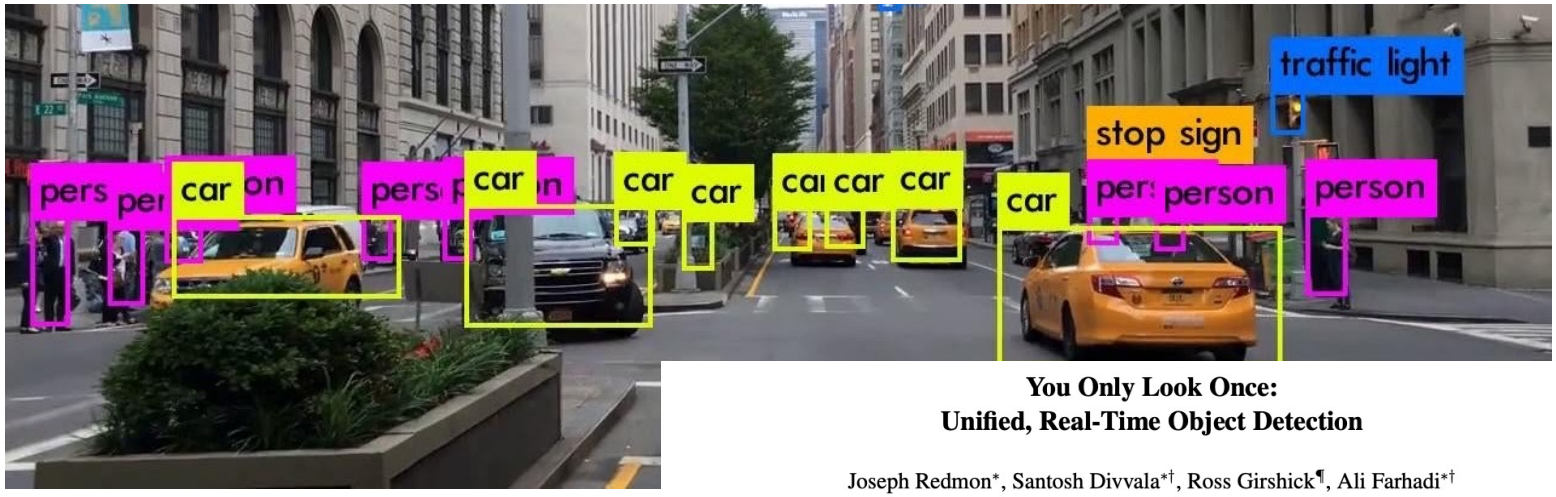




Source : https://arxiv.org/abs/1506.01497



You Only Look Once (YOLO)

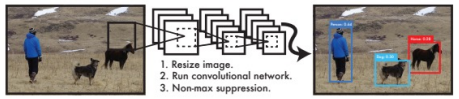


You Only Look Once:
Unified, Real-Time Object Detection

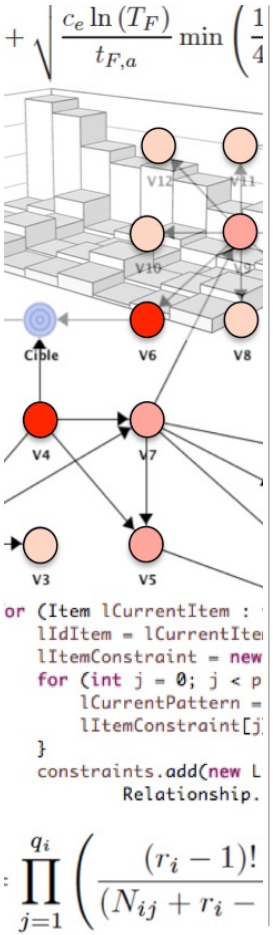
Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a re-



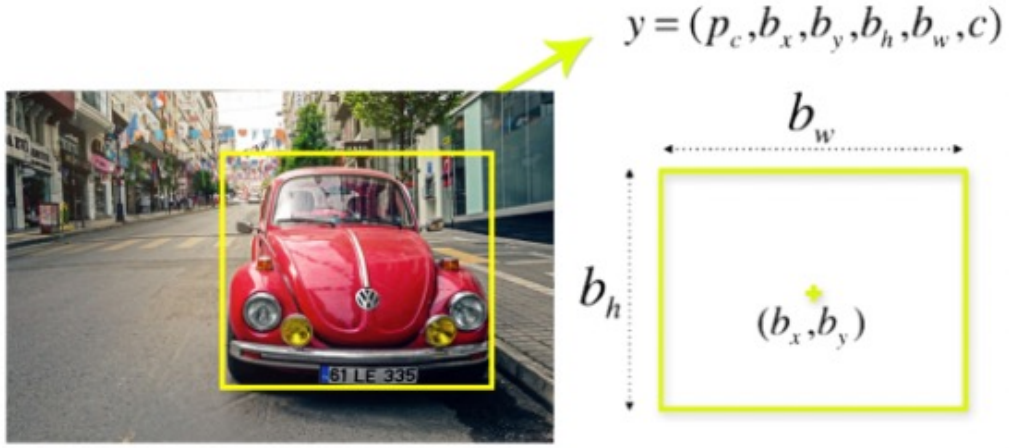
https://arxiv.org/pdf/1506.02640.pdf

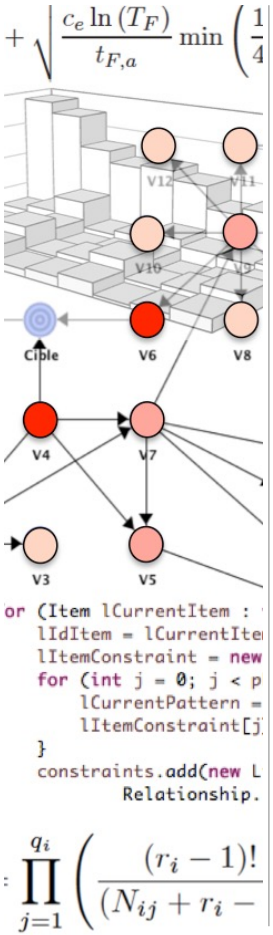


You Only Look Once (YOLO)

Pas de pyramide

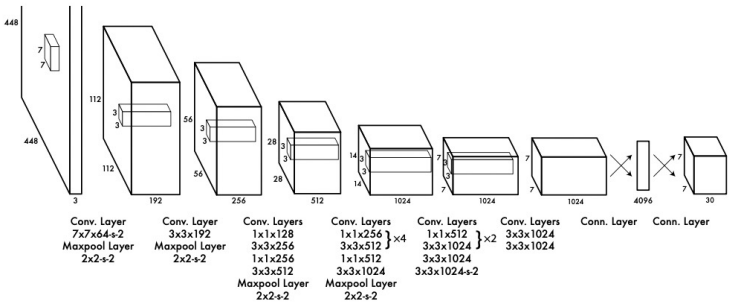
A partir de l'image entière, YOLO renvoie des "bounding box" avec une probabilité de présence d'un objet dans cette box, ainsi que la probabilité de chaque classe d'objets possibles (80)





You Only Look Once (YOLO)

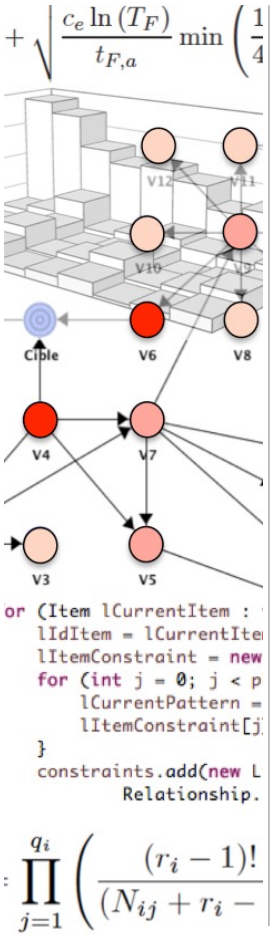
We pretrain our convolutional layers on the ImageNet 1000-class competition dataset [30]. For pretraining we use the first 20 convolutional layers from Figure 3 followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo [24]. We use the Darknet framework for all training and inference [26].



We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance [29]. Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224×224 to 448×448.

Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.

We train the network for about 135 epochs on the training and validation data sets from PASCAL VOC 2007 and 2012. When testing on 2012 we also include the VOC 2007 test data for training. Throughout training we use a batch

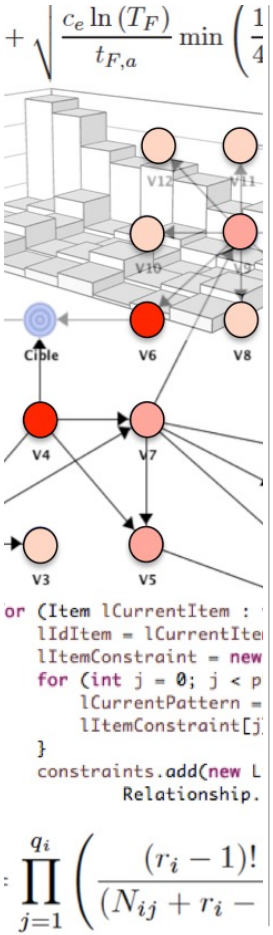


You Only Look Once (YOLO)

Données d'apprentissage

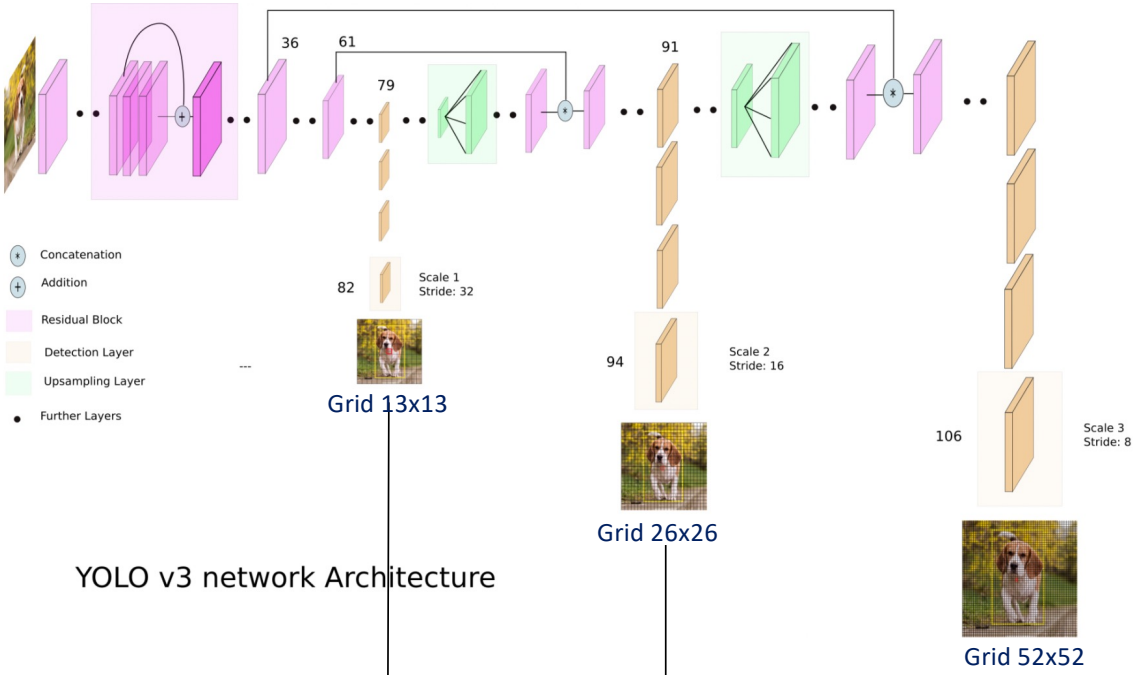
PASCAL VOC 2007

```
<annotation>
    <folder>VOC2007</folder>
    <filename>000037.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
        <flickrid>338124795</flickrid>
    </source>
    <owner>
        <flickrid>seudolog</flickrid>
        <name>?</name>
    </owner>
    <size>
        <width>500</width>
        <height>375</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>dog</name>
        <pose>Left</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>61</xmin>
            <ymin>96</ymin>
            <xmax>464</xmax>
            <ymax>339</ymax>
        </bndbox>
    </object>
</annotation>
```

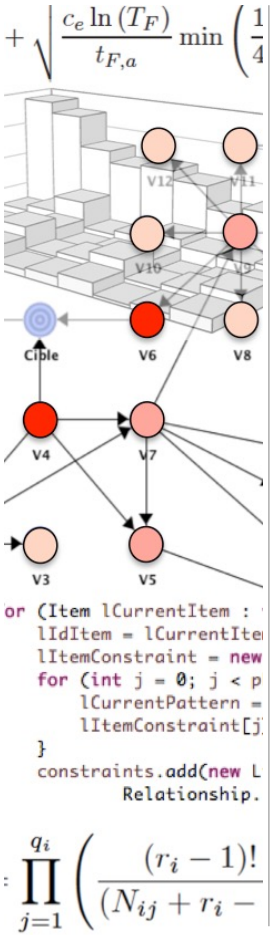


You Only Look Once (YOLO)

YOLO v3

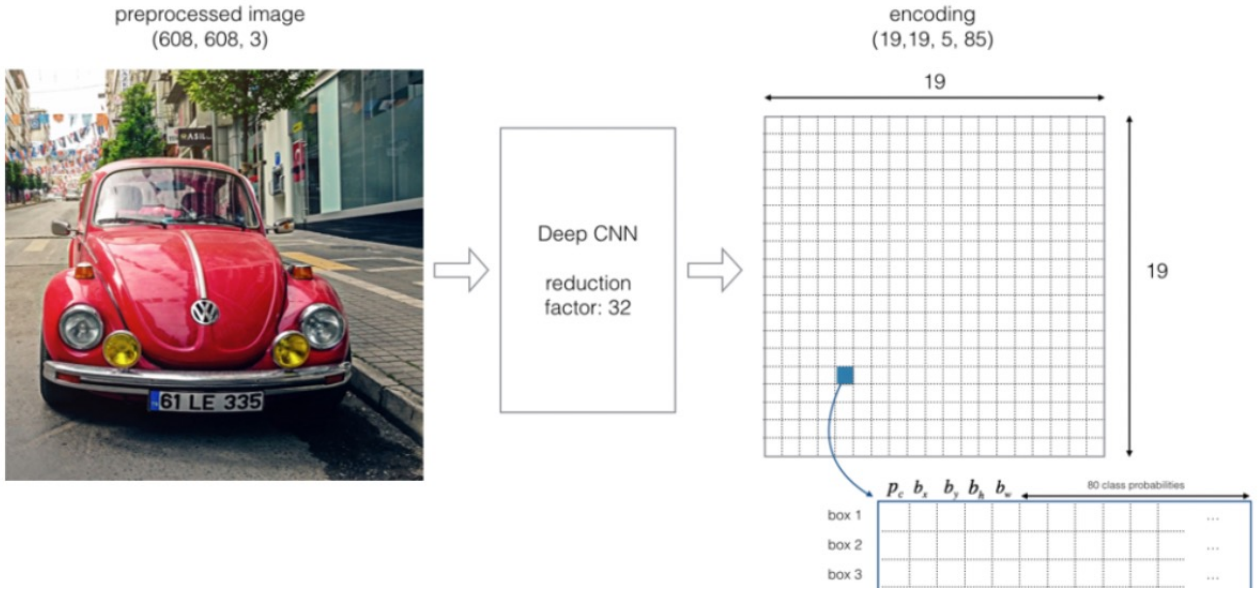


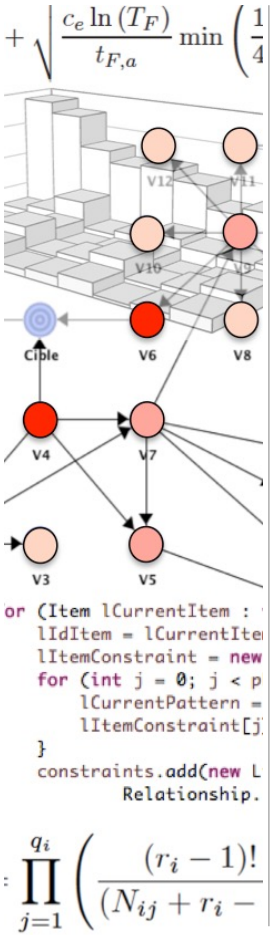
3 predictions de "bounding box" à 3 échelles différentes



You Only Look Once (YOLO)

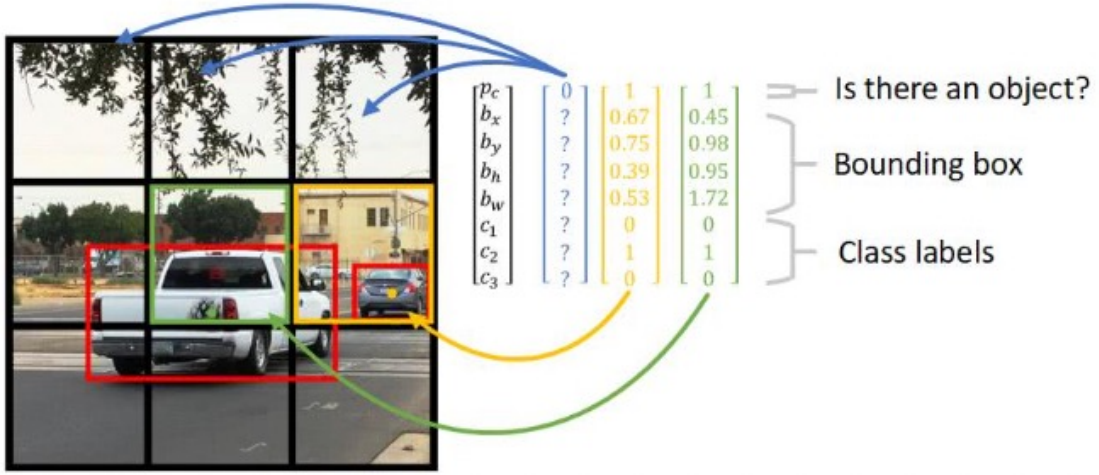
L'image est divisée en cellules qui sont chacune responsable de la prédiction de plusieurs "bounding box" (réalisé à plusieurs échelles : niveaux dans le CNN)

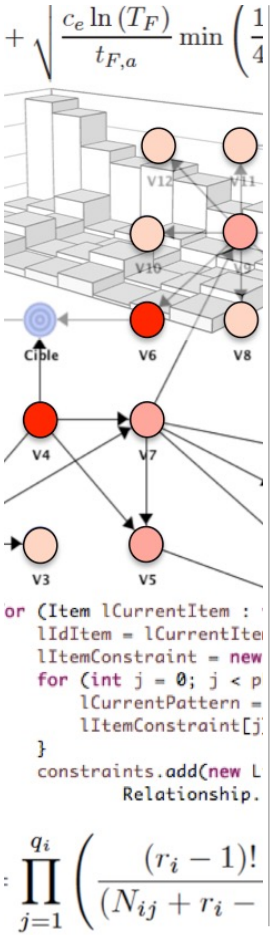




You Only Look Once (YOLO)

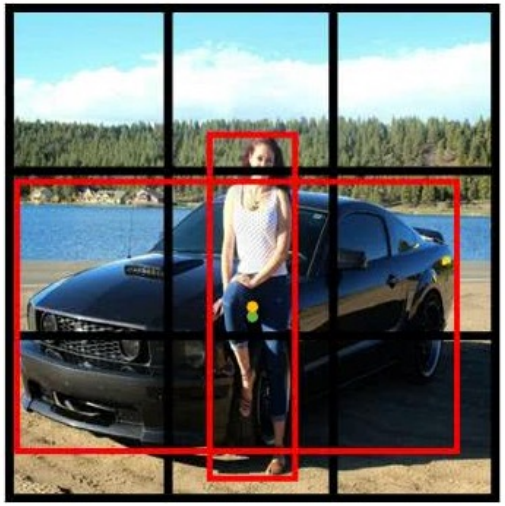
La voiture au centre est positionnée sur 6 cellules mais est affectée à la cellule verte, car le centre de la voiture (centre de la "bounding-box" rouge) est dans cette cellule.





You Only Look Once (YOLO)

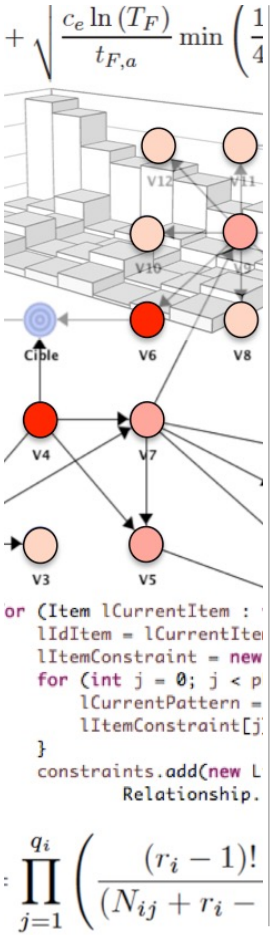
Une cellule peut contenir plusieurs centres de "bounding-box" et donc représenter plusieurs objets.



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

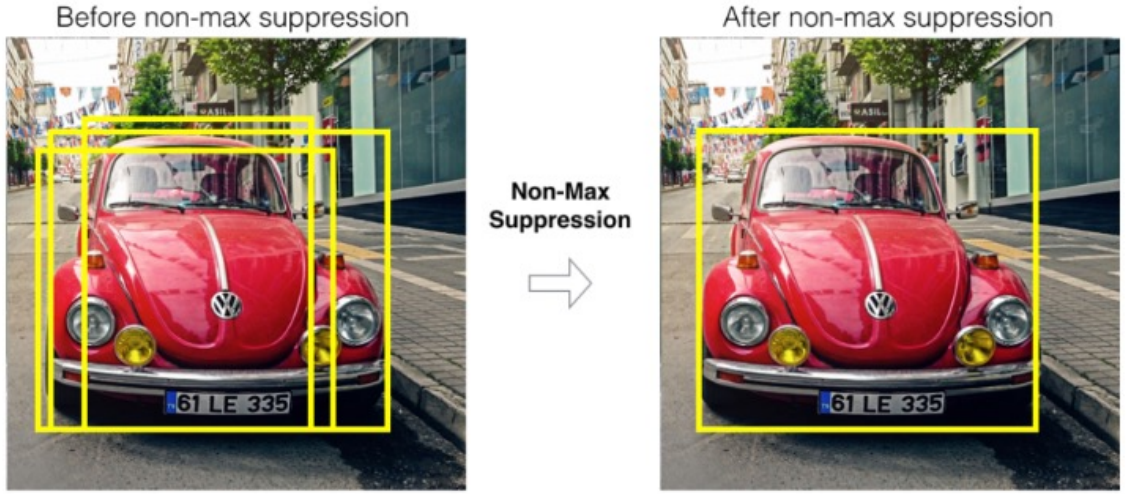
Anchor box 1
Pedestrian

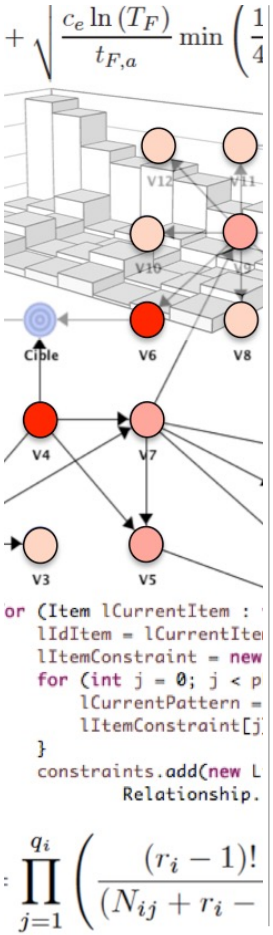
Anchor box 2
Car



You Only Look Once (YOLO)

Lorsque plusieurs "bouding-box" identifient le même objet, on ne conserve que la "bouding-box" la plus probable par "Non-Max Suppression" :





You Only Look Once (YOLO)

Notebook python …