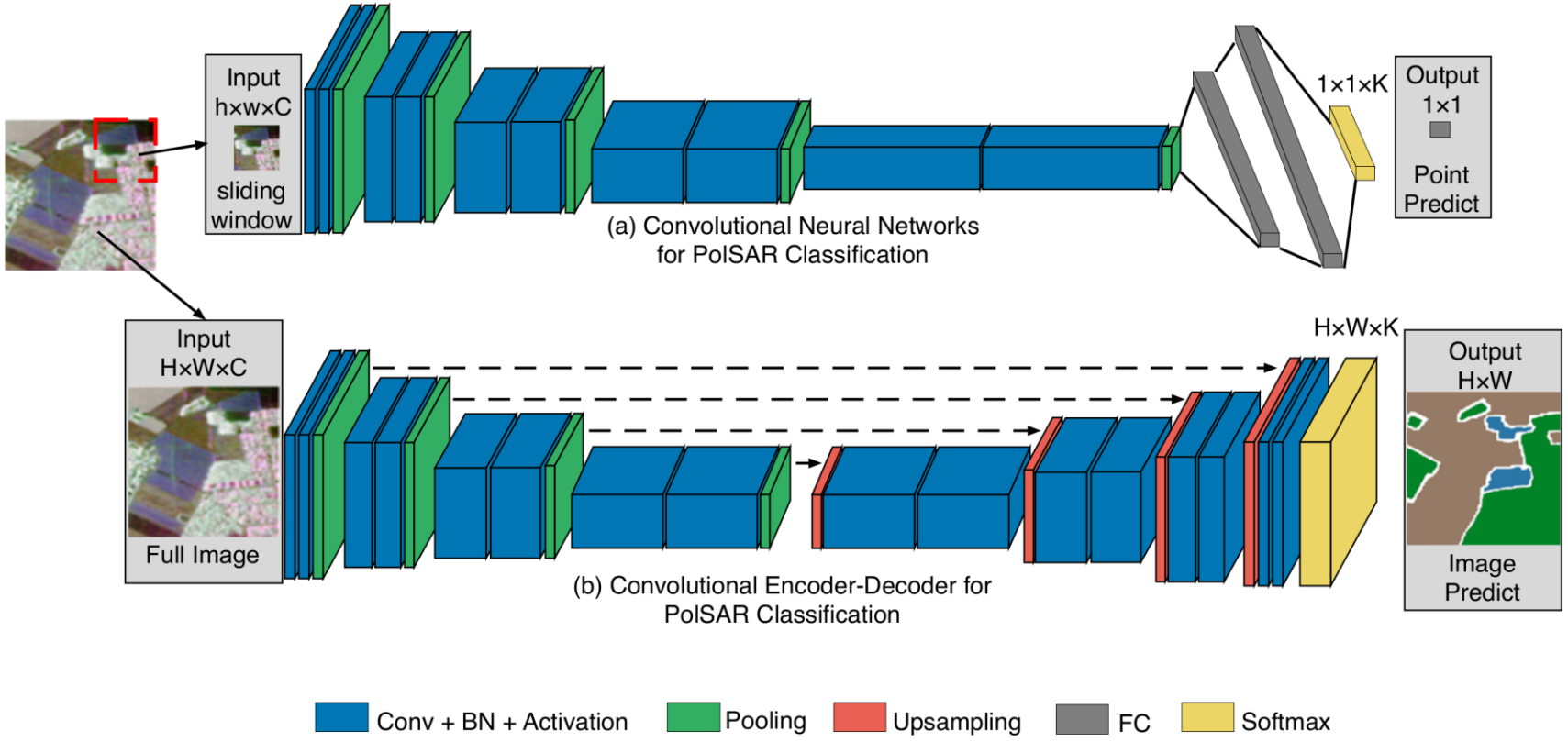
UNIVERSITE ABDELMALEK ESSAADI
FACULTE DES SCIENCES
TETOUAN

جامعة عبد المالك السعدي
كلية العلوم
تطوان

Apprentissage profond pour la vision par ordinateur

Presented by Pr. Abdellatif El Ouissari

Architectures



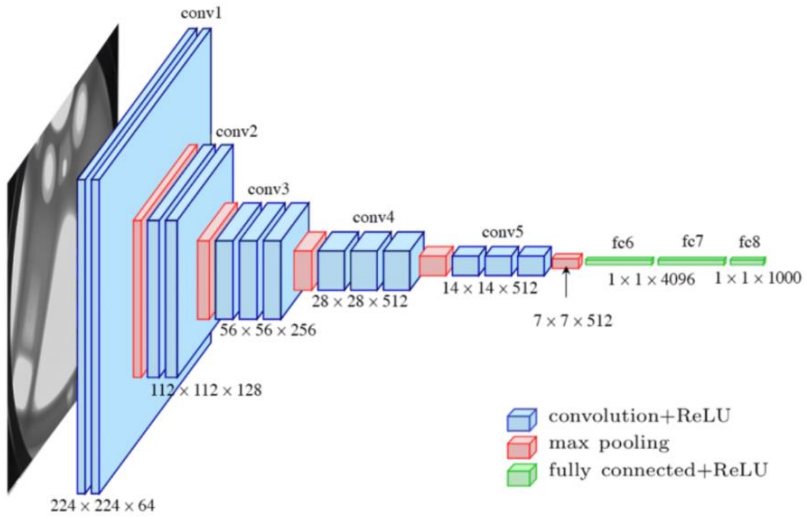
Introduction

Avec l'explosion des données visuelles dans le monde numérique — images, vidéos, caméras intelligentes — la capacité d'analyser automatiquement ces données est devenue essentielle. C'est dans ce contexte que les **réseaux de neurones convolutifs** ou **CNN (Convolutional Neural Networks)** ont révolutionné le domaine de la **vision par ordinateur**.

Les CNN sont des **modèles d'apprentissage profond** capables d'extraire automatiquement des caractéristiques pertinentes à partir d'images, sans nécessiter de traitement manuel préalable. Depuis les premiers modèles simples jusqu'aux architectures avancées comme **AlexNet**, **VGG**, **GoogLeNet** ou **ResNet**, les CNN ont connu une évolution rapide et exponentielle.

Introduction

Ce chapitre a pour objectif de présenter les **principales architectures CNN**, leur **structure interne** et leur **impact sur les performances** dans les tâches de classification d'images, de détection d'objets ou de segmentation.



Introduction

- Décrire et comparer les architectures célèbres : **LeNet**, **AlexNet**, **VGG**, **GoogLeNet (Inception)**, **ResNet**
- Identifier les **avantages et limites** de chaque architecture
- Analyser les performances selon les métriques comme **Top-1 / Top-5 accuracy**
- Comprendre l'évolution vers des réseaux **plus profonds**, **plus rapides** et **plus efficaces**

Large Scale Visual Recognition Challenge (ILSVRC)

Le projet ImageNet est une grande base de données destinée à la recherche sur les logiciels de reconnaissance visuelle d'objets.

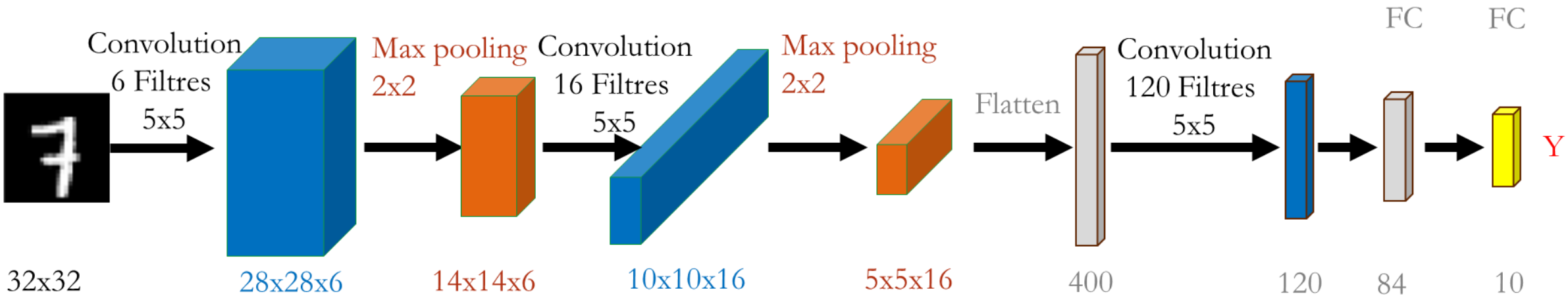
Le projet ImageNet organise un concours annuel, ImageNet Large Scale Visual Recognition Challenge (ILSVRC), dans le cadre duquel des logiciels s'affrontent pour classer correctement des objets et des scènes.

Large Scale Visual Recognition Challenge (ILSVRC)

ImageNet est une gigantesque base de données de plus de **14 millions d'images labellisées** réparties dans plus de 1000 classes, en 2014.
En 2007, une chercheuse du nom de Fei-Fei Li a commencé à travailler sur l'idée de créer un tel jeu de données. Certes la modélisation est un aspect très important pour obtenir des bonnes performances, mais disposer de données de grande qualité l'est tout autant pour avoir un apprentissage de qualité. Les données ont été collectées et étiquetées depuis le web par des humains. Elles sont donc **Open source** et n'appartiennent pas à une entreprise en particulier.

Depuis 2010 s'organise, chaque année, une compétition ImageNet Large Scale Visual Recognition Challenge dont le but est de challenger des modèles de traitement d'images. La compétition s'effectue sur un sous-ensemble d'ImageNet composé de : 1,2 million d'images d'entraînement, 50000 pour validation et 150000 pour tester le modèle.

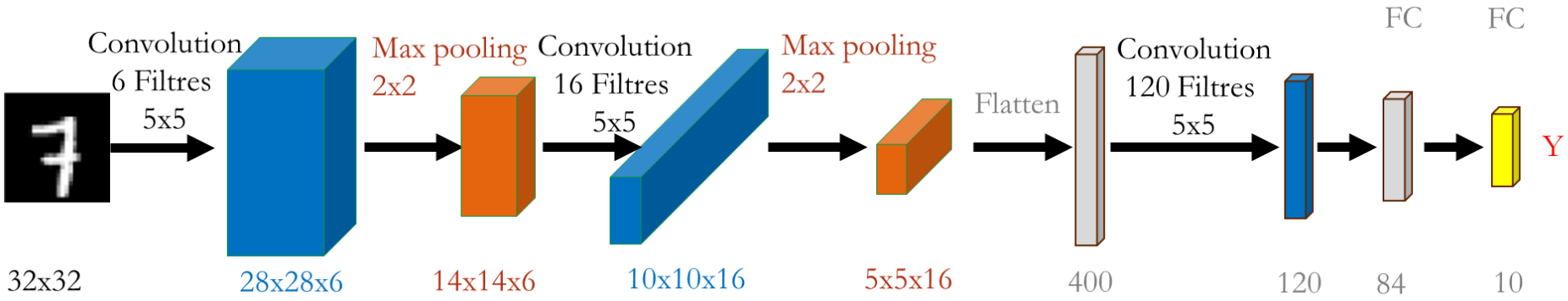
LeNet-5


Convolution
6 Filtres
5x5
Max pooling
2x2
Convolution
16 Filtres
5x5
Max pooling
2x2
Flatten
Convolution
120 Filtres
5x5
FC
FC
Y
32x32
28x28x6
14x14x6
10x10x16
5x5x16
400
120
84
10

LeNet-5

LeNet est un réseau neuronal convolutionnel introduit par Yann LeCun en 1989. LeNet est un terme commun pour LeNet-5, un réseau neuronal convolutionnel simple.

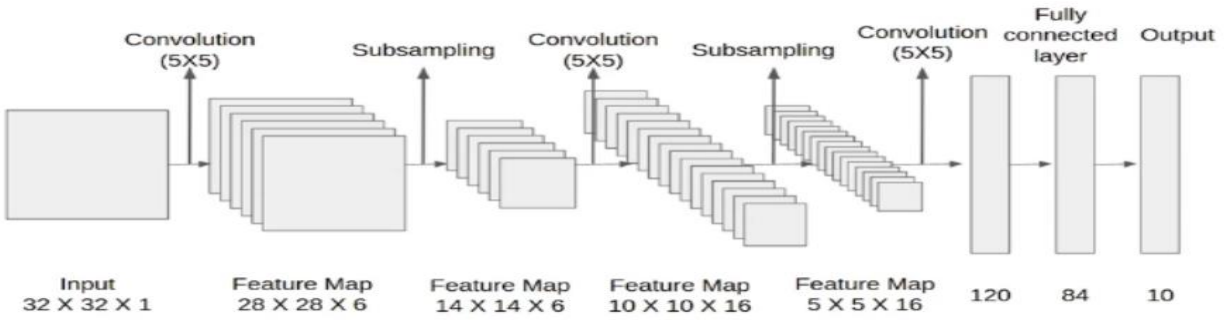
Le LeNet-5 marque l'émergence du CNN et décrit ses principaux composants. Cependant, il n'était pas populaire à l'époque en raison d'un manque de matériel, notamment de GPU (Graphics Process Unit, un circuit électronique spécialisé conçu pour modifier la mémoire afin d'accélérer la création d'images pendant un tampon destiné à la sortie vers un appareil de spectacle) et d'algorithmes alternatifs, comme le SVM, qui pouvaient réaliser des effets similaires, voire meilleurs, que ceux du LeNet.



LeNet-5

Cette architecture est composée des couches suivantes:
Couches (Layers):

- Convolution: 6 filtres, chacun de taille: 5x5
- Max Pooling: 2x2
- Convolution: 16 filtres, chacun de taille: 5x5
- Max Pooling: 2x2
- Convolution: 120 filtres, chacun de taille: 5x5
- Fully connected: 84 neurones
- Sortie: Fully connected de 10 neurones



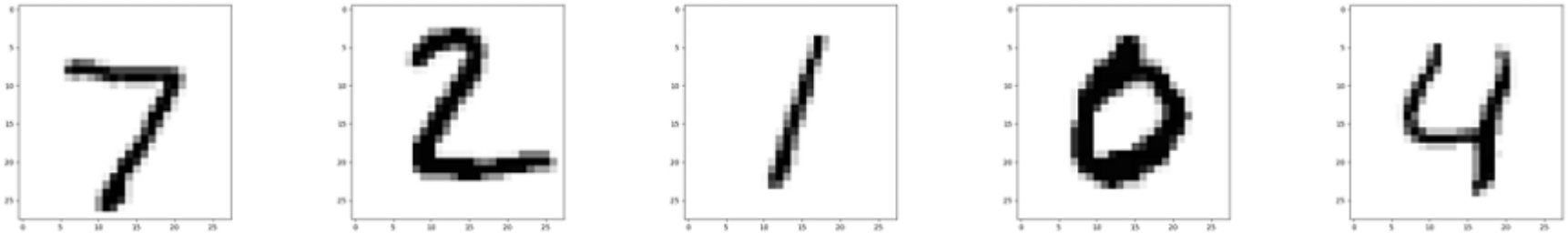
Architecture en détail

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

Application

Base MNIST

- La base est formée de 60000 données d'apprentissage et de 10000 données de test.
- Chaque donnée est de la forme : [une image, le chiffre attendu].
- Chaque image est de taille 28×28 pixels, chaque pixel contenant un des 256 niveaux de gris (numérotés de 0 à 255).



LeNet-5

Préparation de l'environnement

```
# charger la base d'entrainement et de validation
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
# Normaliser les données pour qu'elles soient entre 0 et 1
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255
```

LeNet-5

Redimensionner les images

```
import tensorflow as tf
train_images = tf.expand_dims(train_images, axis=-1)  # Add channel dimension
train_images = tf.image.resize(train_images, [32, 32])  # Resize train_images

test_images = tf.expand_dims(test_images, axis=-1)  # Add channel dimension
test_images = tf.image.resize(test_images, [32, 32])  # Resize test_images
```

Construction et Entraînement du Modèle LeNet-5

```python
from tensorflow.keras import layers, models

# Définition du modèle LeNet-5
model = models.Sequential([
    # Première couche de convolution, avec padding 'same' pour adapter la taille de l'entrée
    layers.Conv2D(input_shape=(32, 32, 1), filters=6, kernel_size=(5, 5), strides=(1, 1), activation='relu'),
    # Première couche de pooling
    layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
    # Deuxième couche de convolution
    layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='relu'),
    # Deuxième couche de pooling
    layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
    # Troisème couche de convolution
    layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='relu'),
    # Aplatir les cartes de caractéristiques pour les passer aux couches entièrement connectées
    layers.Flatten(),
    # Première couche entièrement connectée
    layers.Dense(84, activation='relu'),
    # Couche de sortie avec activation softmax pour la classification de 10 chiffres
    layers.Dense(10, activation='softmax')
])
```

Construction et Entraînement du Modèle LeNet-5

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

Construction et Entraînement du Modèle LeNet-5

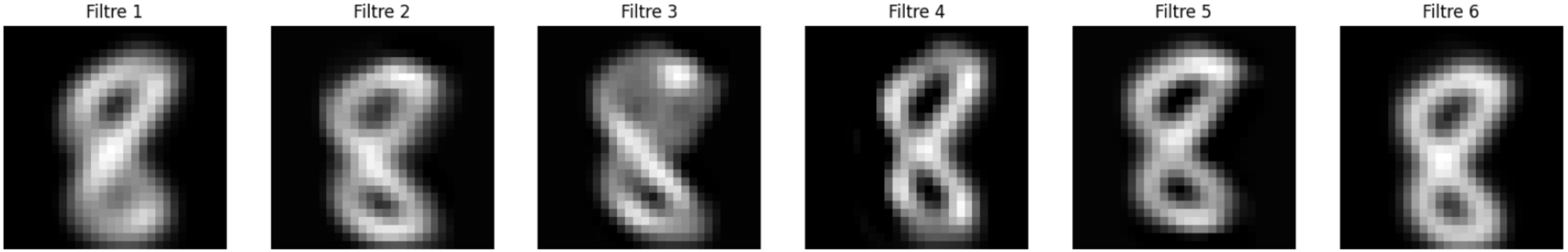
```
[ ]  # Afficher l'architecture du modèle
     model.summary()
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 28, 28, 6)         156

 max_pooling2d_6 (MaxPooling  (None, 14, 14, 6)        0
 2D)

 conv2d_11 (Conv2D)          (None, 10, 10, 16)        2416

 max_pooling2d_7 (MaxPooling  (None, 5, 5, 16)         0
 2D)

 conv2d_12 (Conv2D)          (None, 1, 1, 120)         48120

 flatten_2 (Flatten)         (None, 120)               0

 dense_6 (Dense)             (None, 84)                10164

 dense_7 (Dense)             (None, 10)                850

=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

Résultats de la première couche de convolution

```
# Obtenir la sortie de la première couche de convolution
first_conv_layer = tf.keras.Model(inputs=model.input, outputs=model.layers[0].output)
first_conv_output = first_conv_layer.predict(test_image_preprocessed)

# Afficher les résultats de la convolution
n_filters = first_conv_output.shape[-1]
fig, axes = plt.subplots(1, n_filters, figsize=(20, 20))
for i in range(n_filters):
    axes[i].imshow(first_conv_output[0, :, :, i], cmap='gray')
    axes[i].set_title(f'Filtre {i+1}')
    axes[i].axis('off')

plt.show()
```

Résultats de la première couche de convolution

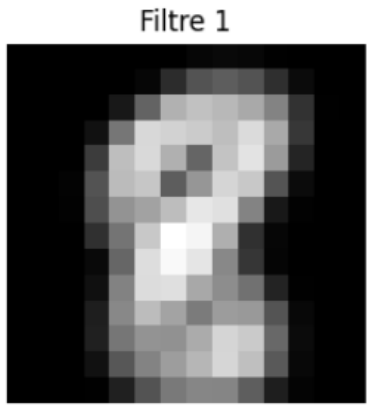


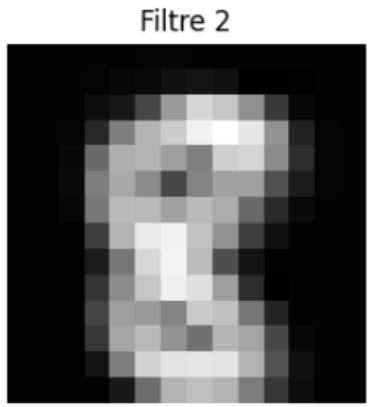
On remarque que chaque filtre apprend à extraire des caractéristiques distinctes de l'image d'entrée, ce qui permet au réseau de construire une représentation riche et variée des données visuelles.

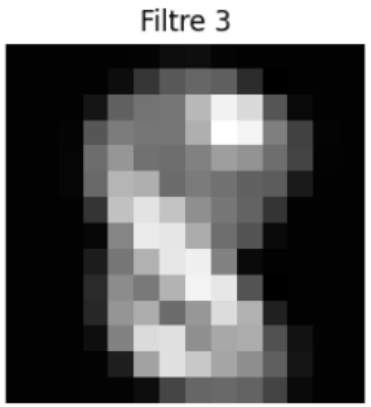
Résultats de la première couche de pooling

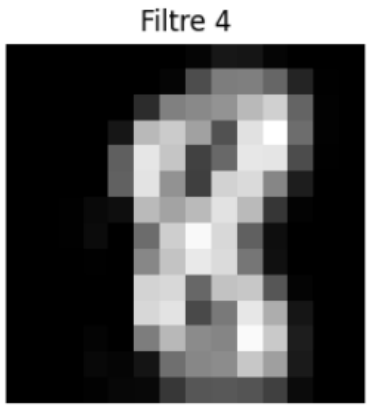
```
# Obtenir la sortie de la première couche de convolution
first_pooling_layer = tf.keras.Model(inputs=model.input, outputs=model.layers[1].output)
first_pooling_output = first_pooling_layer.predict(test_image_preprocessed)

# Afficher les résultats de la convolution
n_filters = first_pooling_output.shape[-1]
fig, axes = plt.subplots(1, n_filters, figsize=(20, 20))
for i in range(n_filters):
    axes[i].imshow(first_pooling_output[0, :, :, i], cmap='gray')
    axes[i].set_title(f'Filtre {i+1}')
    axes[i].axis('off')

plt.show()
```

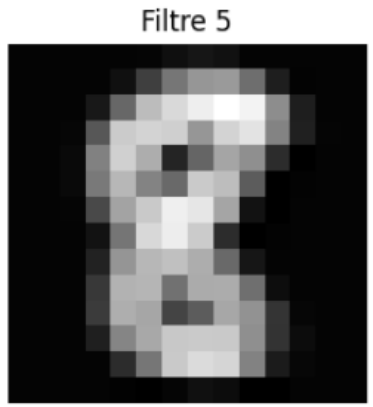
Résultats de la première couche de pooling

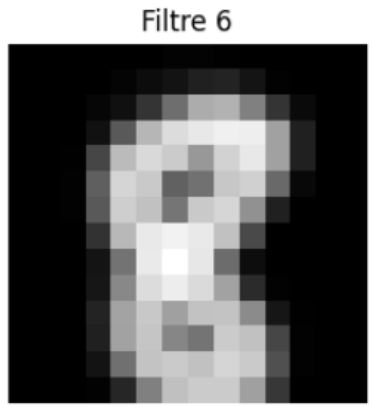










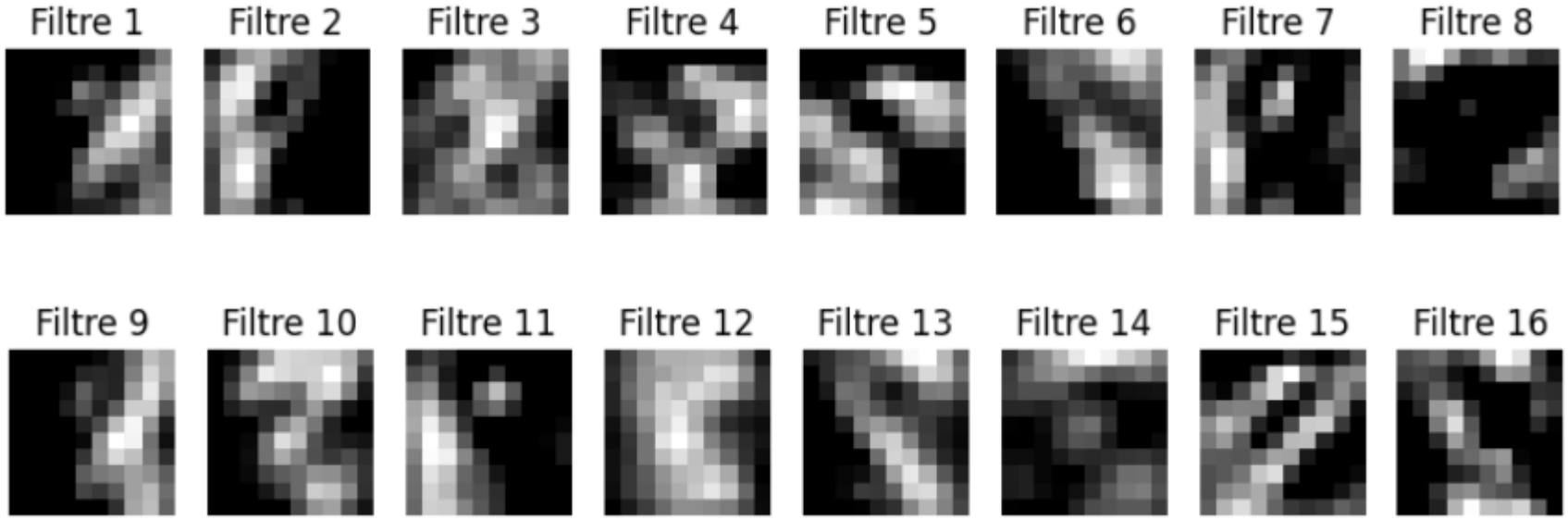


La couche de pooling réduit généralement la taille spatiale des cartes de caractéristiques tout en conservant les informations les plus importantes. On observe comme résultat, des images plus petites qui contiendront toujours les caractéristiques importantes détectées par les filtres de la première couche de convolution.

Résultats de la deuxième couche de convolution

```
# Obtenir la sortie de la première couche de convolution
second_conv_layer = tf.keras.Model(inputs=model.input, outputs=model.layers[2].output)
second_conv_output = second_conv_layer.predict(test_image_preprocessed)

# Afficher les résultats de la convolution
n_filters = second_conv_output.shape[-1]
fig, axes = plt.subplots(1, n_filters, figsize=(20, 20))
for i in range(n_filters):
    axes[i].imshow(second_conv_output[0, :, :, i], cmap='gray')
    axes[i].set_title(f'Filtre {i+1}')
    axes[i].axis('off')

plt.show()
```

Résultats de la deuxième couche de convolution



La deuxième couche de convolution construit sur les caractéristiques détectées par la première couche, permettant de capturer des motifs plus complexes. On remarque des cartes de caractéristiques qui montrent des combinaisons de motifs détectés par les filtres de la première couche.

Evaluation du modèle

```
Epoch 1/4
1875/1875 [==============================] - 30s 15ms/step - loss: 0.4679 - accuracy: 0.8648 - val_loss: 0.1913 - val_accuracy: 0.9416
Epoch 2/4
1875/1875 [==============================] - 29s 15ms/step - loss: 0.1626 - accuracy: 0.9509 - val_loss: 0.1151 - val_accuracy: 0.9630
Epoch 3/4
1875/1875 [==============================] - 29s 15ms/step - loss: 0.1129 - accuracy: 0.9656 - val_loss: 0.0832 - val_accuracy: 0.9759
Epoch 4/4
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0890 - accuracy: 0.9737 - val_loss: 0.0723 - val_accuracy: 0.9784
```

Accuracy :

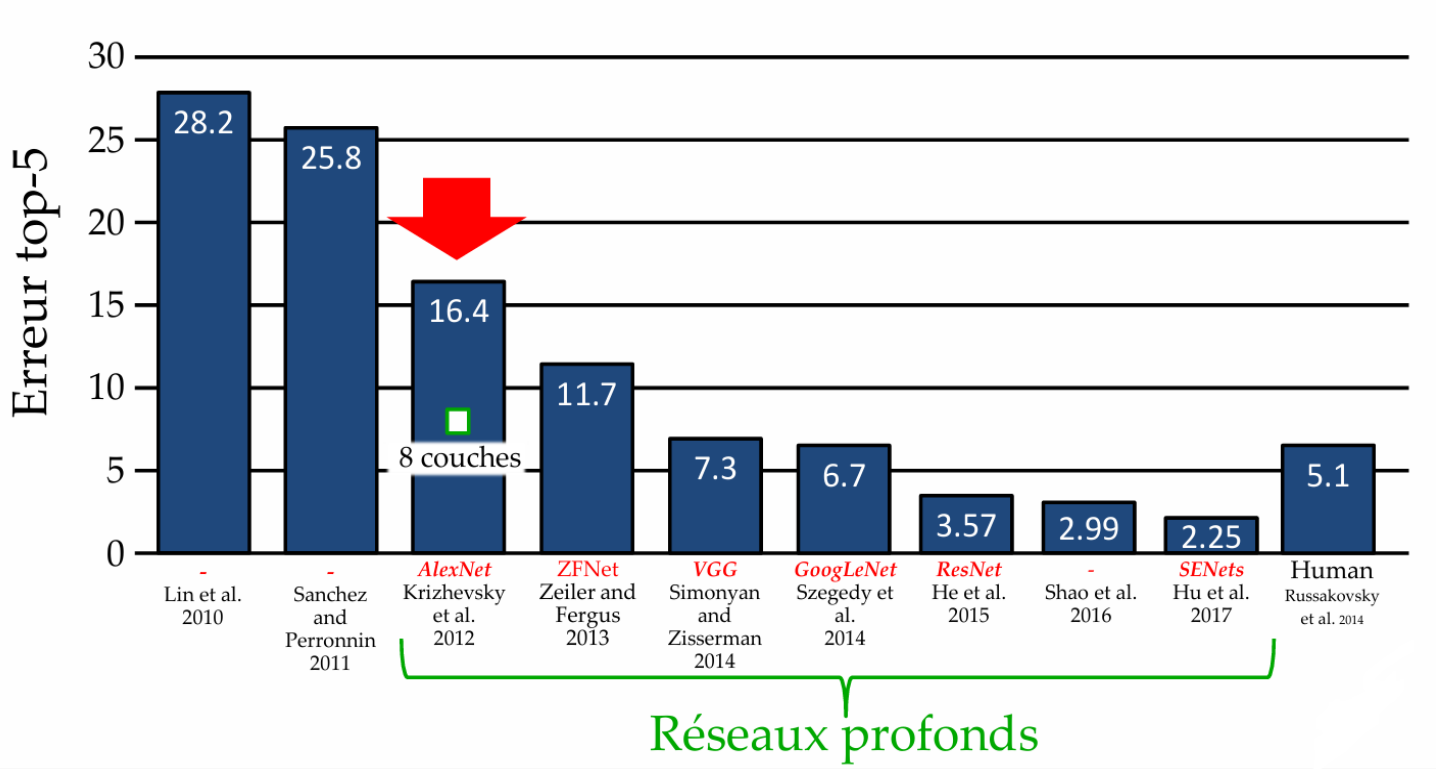
1. **Entraînement** : 98,93%
2. **Validation** : 98,83%
3. Les valeurs sont très proches, ce qui suggère une bonne généralisation du modèle sans surapprentissage (overfitting).

Loss :

1. **Entraînement** : 0,0346
2. **Validation** : 0,0344
3. Les valeurs sont similaires, confirmant la cohérence des performances entre les ensembles d'entraînement et de validation.

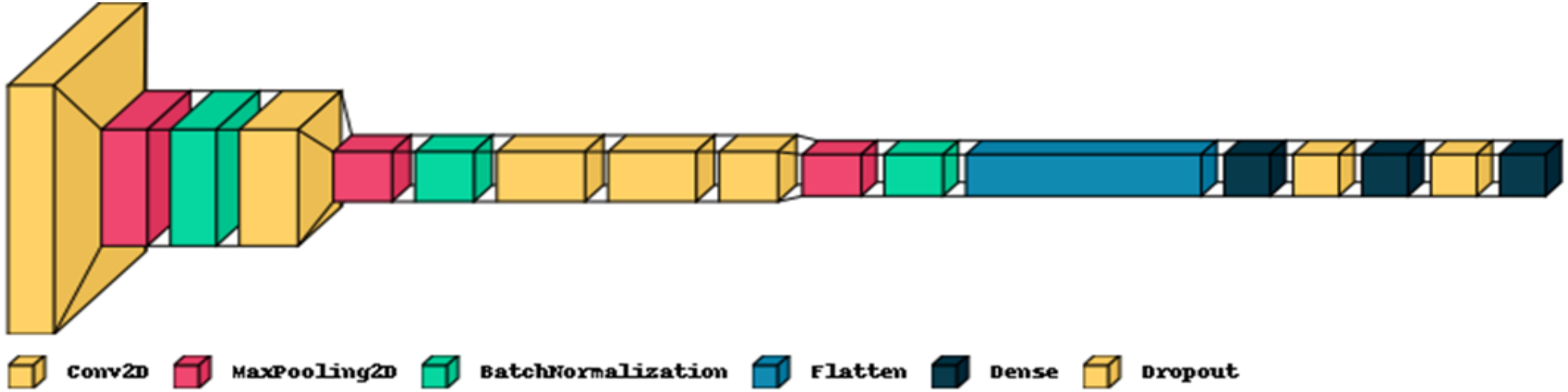
Large Scale Visual Recognition Challenge

Large Scale Visual Recognition Challenge
- Image Classification Challenge :
– 1,000 classes d'objets
– 1,431,167 image

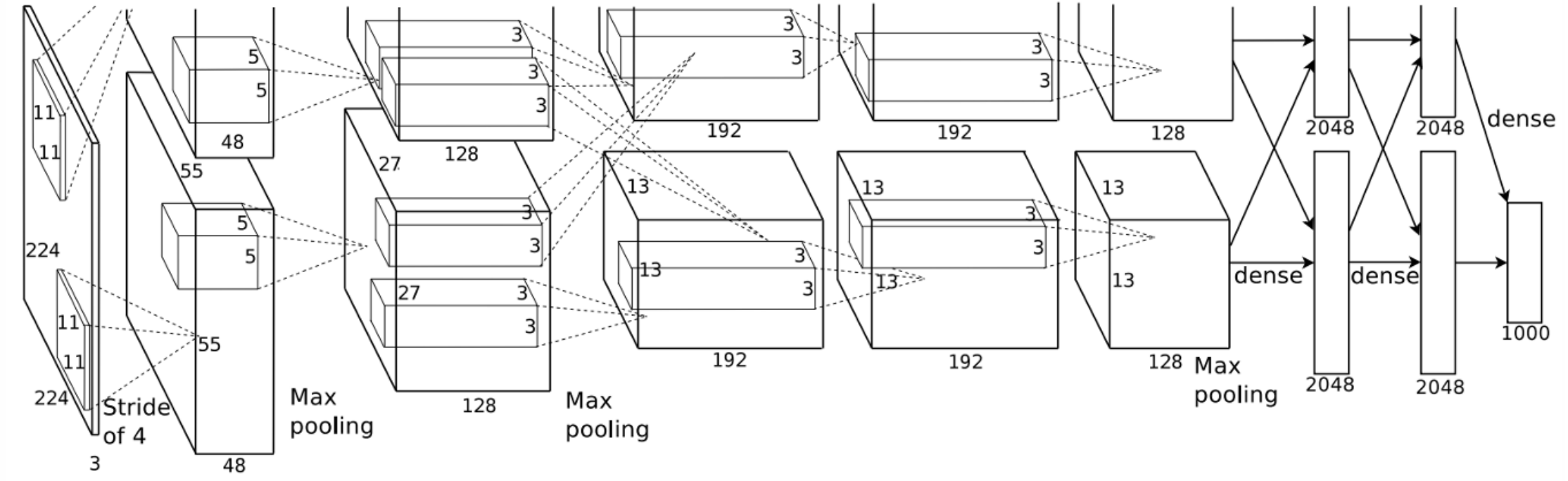


AlexNet

Nous allons par la suite implémenter une autre architecture CNN appelée AlexNet qui est une architecture de réseau de neurones convolutifs (CNN) qui a remporté la compétition ImageNet Large Scale Visual Recognition Challenge (ILSVRC) en 2012, marquant une avancée majeure dans le domaine de la vision par ordinateur et de l'apprentissage profond.



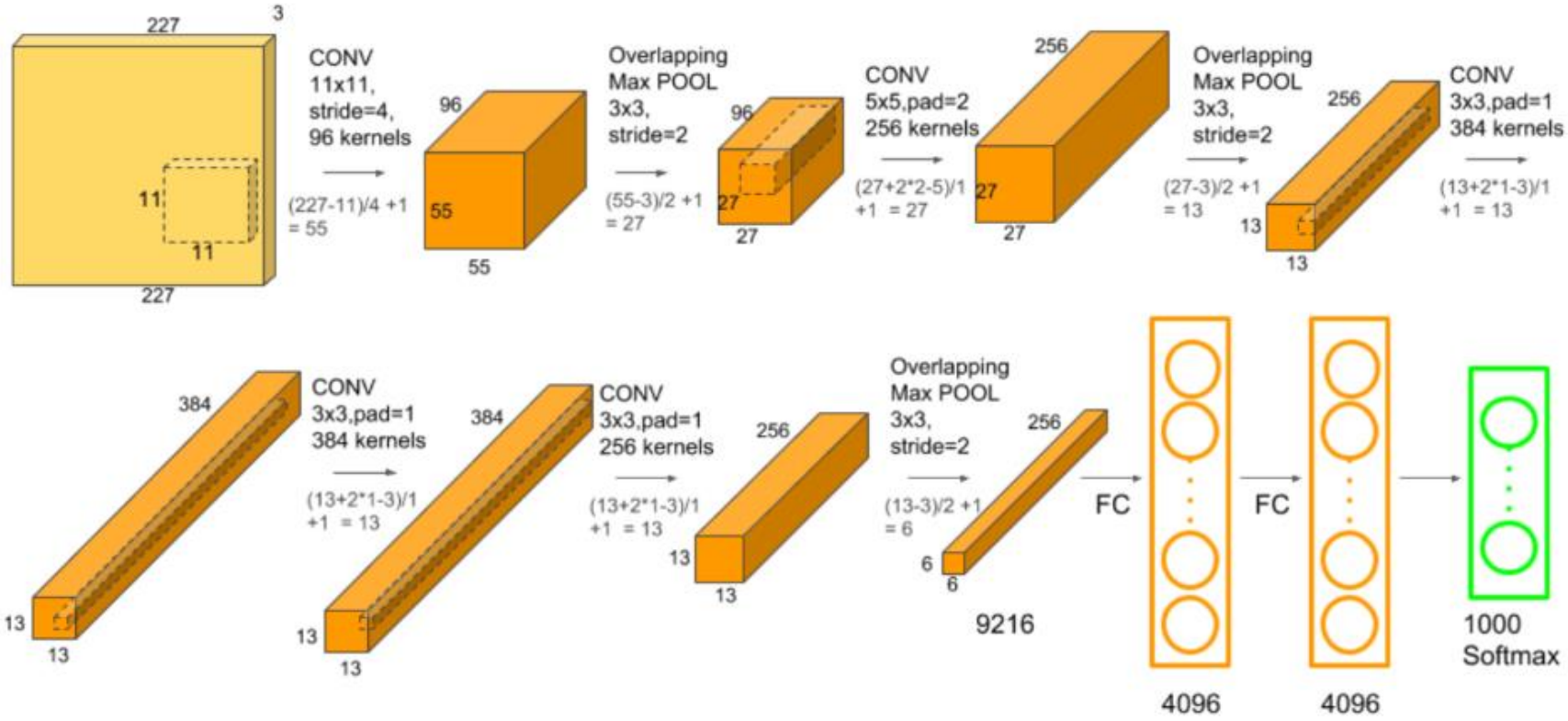
AlexNet



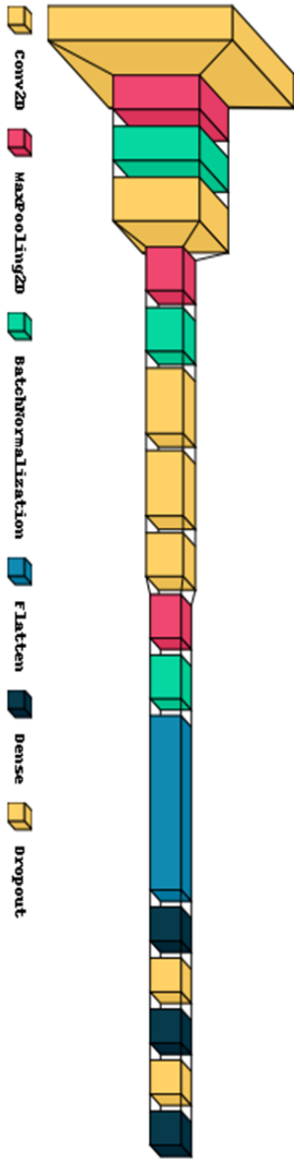
- 8 couches
- 60M paramètres
- Apparition des ReLU
- Dropout de 0.5
- Entraîné sur deux cartes GTX 580 (3 Go) en parallèle

AlexNet

AlexNet est composé de 5 couches convolutives avec une combinaison de couches de mise en commun maximale, de 3 couches entièrement connectées et de 2 couches d'exclusion. La fonction d'activation utilisée dans toutes les couches est Relu. La fonction d'activation utilisée dans la couche de sortie est Softmax. Le nombre total de paramètres dans cette architecture est d'environ 60 millions.



AlexNet



majorité des paramètres

[1000] FC8: 1000 neurons (class scores)
[4096] FC7: 4096 neurons
[4096] FC6: 4096 neurons

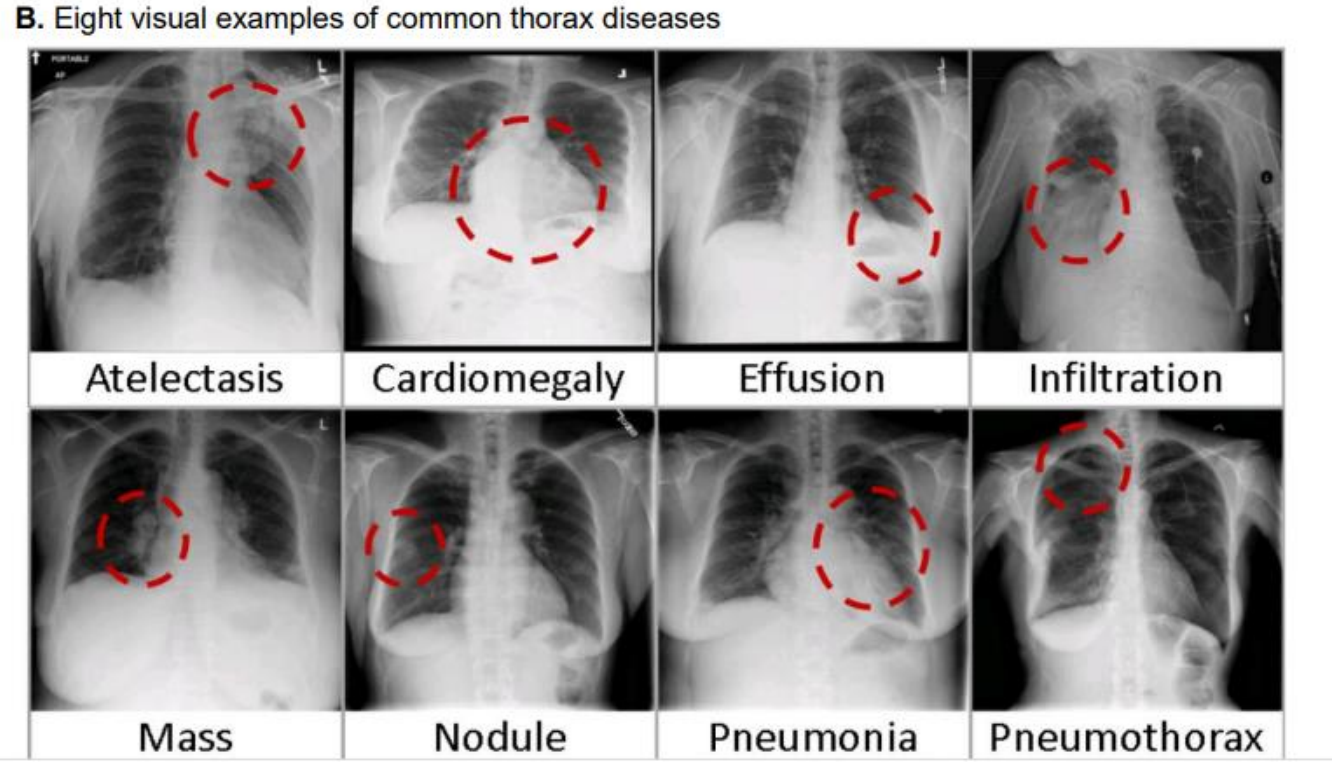
Classificateur puissant (besoin dropout)

[6x6x256] MAX POOL3: 3x3 filters at stride 2
[13x13x256] CONV5: 256 **3x3** filters at stride 1, pad 1
[13x13x384] CONV4: 384 **3x3** filters at stride 1, pad 1
[13x13x384] CONV3: 384 **3x3** filters at stride 1, pad 1
[13x13x256] NORM2: Normalization layer
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[27x27x256] CONV2: 256 **5x5** filters at stride 1, pad 2
[27x27x96] NORM1: Normalization layer
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[55x55x96] CONV1: 96 **11x11** filters at stride 4, pad 0
[227x227x3] INPUT

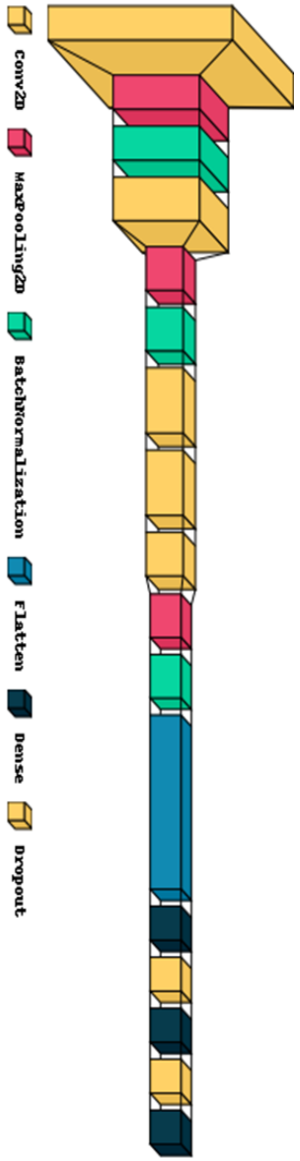
réduction rapide

Application

Nous allons utiliser le dataset des rayons X thoraciques, appelé Chest X-ray, qui est une collection d'images de rayons X du thorax utilisées principalement pour la détection et le diagnostic de maladies pulmonaires telles que la pneumonie.



Application




```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

# Définir l'architecture AlexNet
model = Sequential([
    # 1ère couche convolutionnelle
    Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=(127, 127, 3)),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    BatchNormalization(),

    # 2ème couche convolutionnelle
    Conv2D(256, (5, 5), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    BatchNormalization(),

    # 3ème, 4ème et 5ème couches convolutionnelles
    Conv2D(384, (3, 3), padding='same', activation='relu'),
    Conv2D(384, (3, 3), padding='same', activation='relu'),
    Conv2D(256, (3, 3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    BatchNormalization(),

      # Couches entièrement connectées
      Flatten(),
      Dense(4096, activation='relu'),
      Dropout(0.5),
      Dense(4096, activation='relu'),
      Dropout(0.5),
      Dense(1, activation='sigmoid')  # Couche de sortie pour la classification binaire
  ])
```

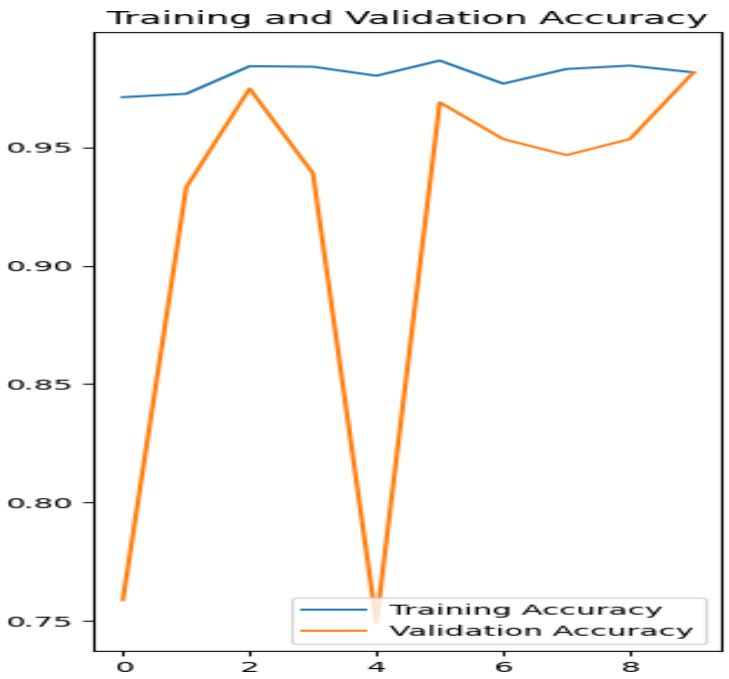
Application

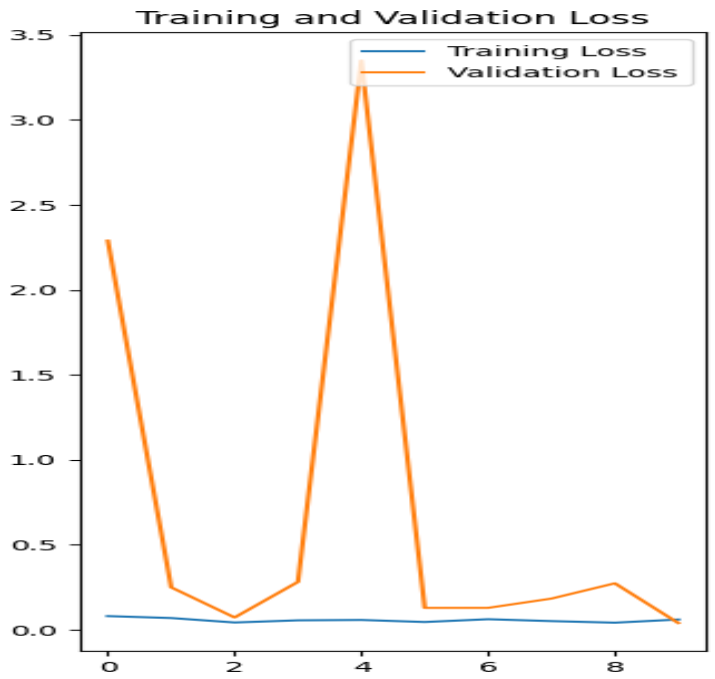
```
Epoch 8/10
132/132 [==============================] - 31s 223ms/step - loss: 0.0799 - accuracy: 0.9738 - val_loss: 0.1734 - val_accuracy: 0.9370
Epoch 9/10
132/132 [==============================] - 31s 222ms/step - loss: 0.0765 - accuracy: 0.9702 - val_loss: 1.2936 - val_accuracy: 0.7626
Epoch 10/10
132/132 [==============================] - 31s 223ms/step - loss: 0.0777 - accuracy: 0.9750 - val_loss: 0.4046 - val_accuracy: 0.8266
```

- **Entraînement :**
  - Accuracy : ~97.5% (excellente maîtrise des données d'entraînement).
  - Loss : ~0.077 (erreur très faible).
- **Validation :**
  - Accuracy : Inconstante (93.7% à l'epoch 8, puis chute à 76.26% à l'epoch 9, remontée à 82.66% à l'epoch 10).
  - Loss : Explosion à 1.2936 (epoch 9), signe d'un problème majeur.

Application

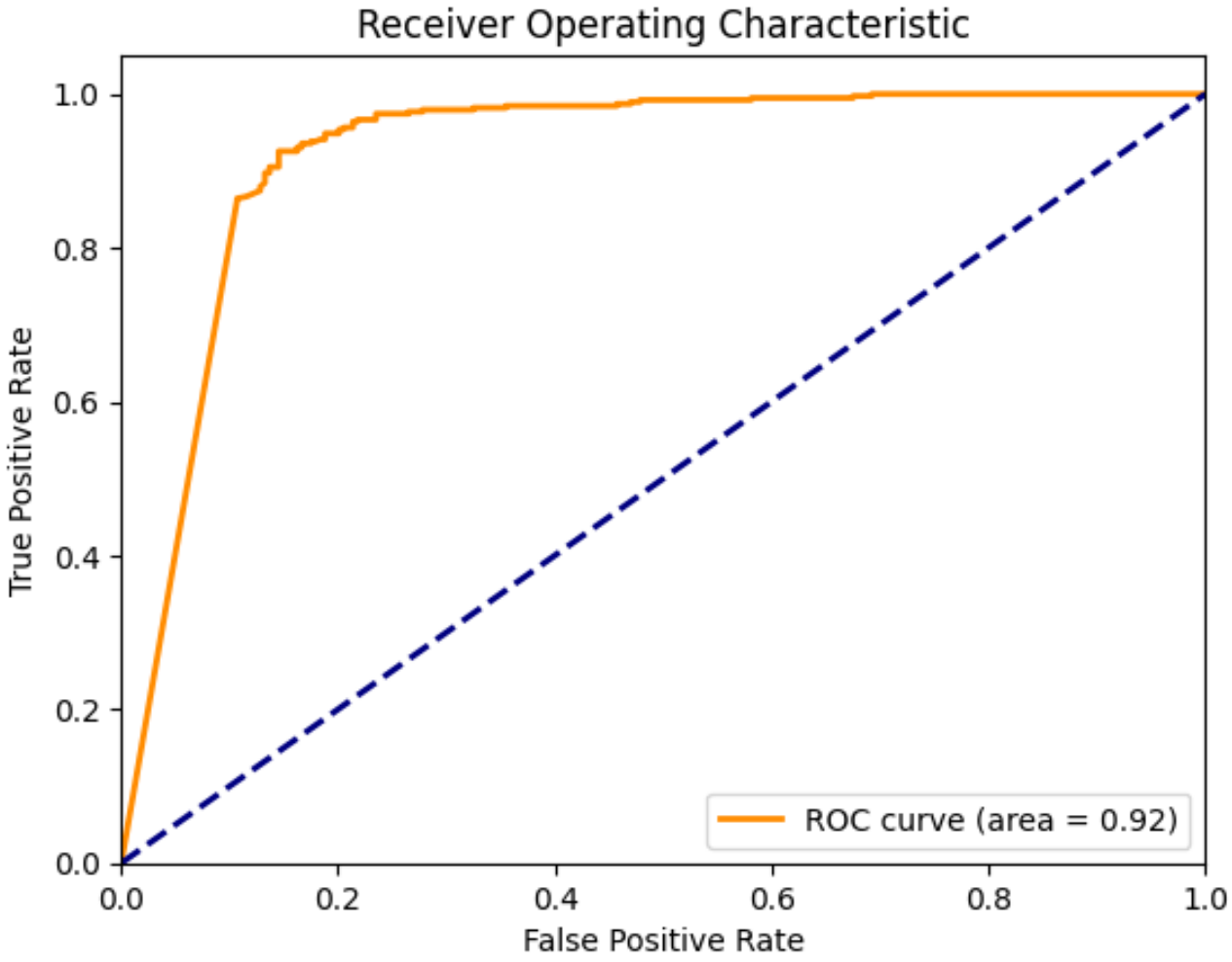
On peut remarquer que les courbes de précision et de perte de validation montrent des fluctuations importantes d'une époque à l'autre.
Cela peut indiquer que le modèle n'est pas stable pendant l'entraînement.



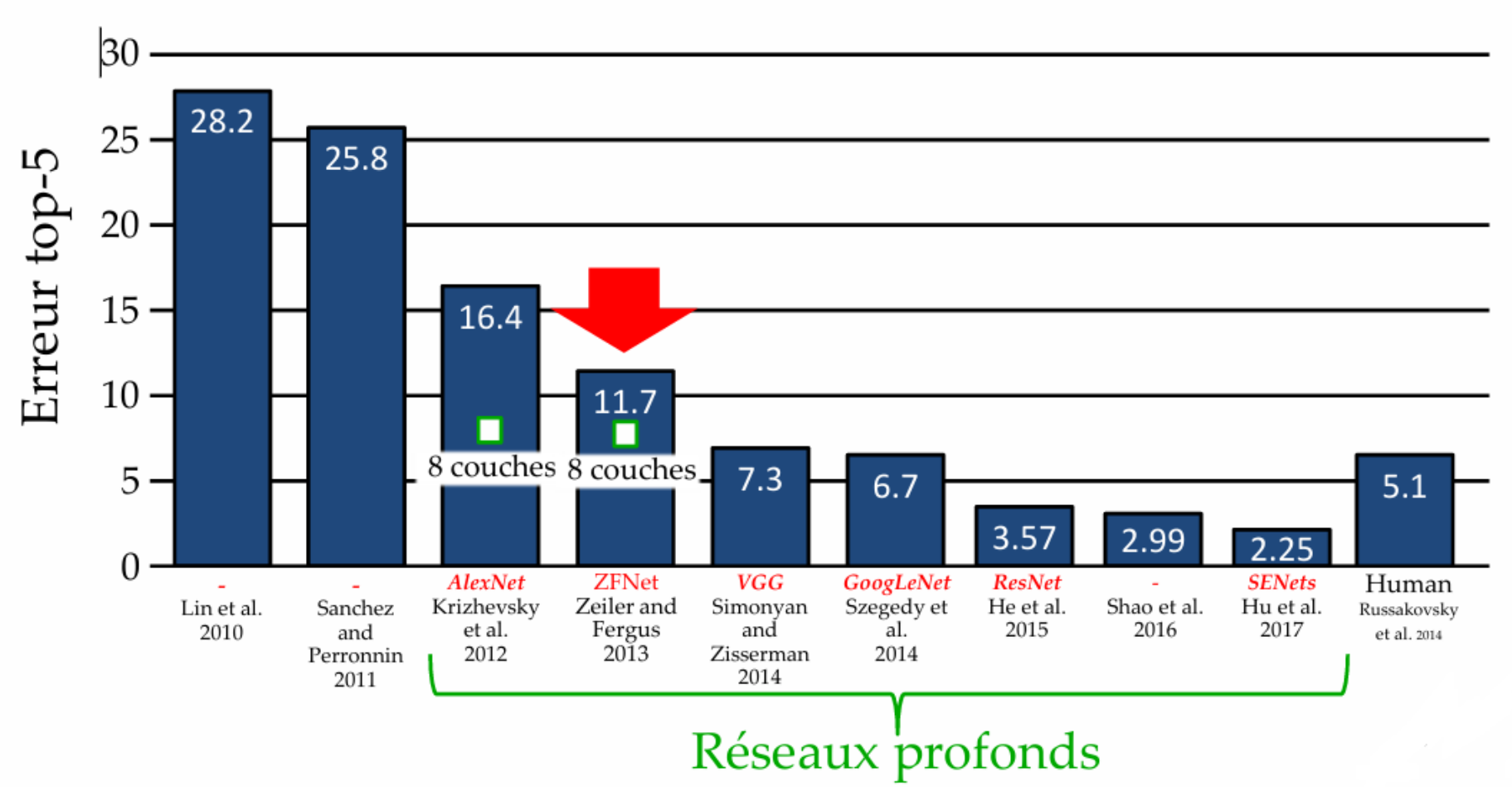


Ces fluctuations combinées à une précision d'entraînement élevée, peuvent indiquer un surapprentissage (overfitting), où le modèle apprend trop bien les détails de l'ensemble d'entraînement mais ne généralise pas bien aux nouvelles données

Application

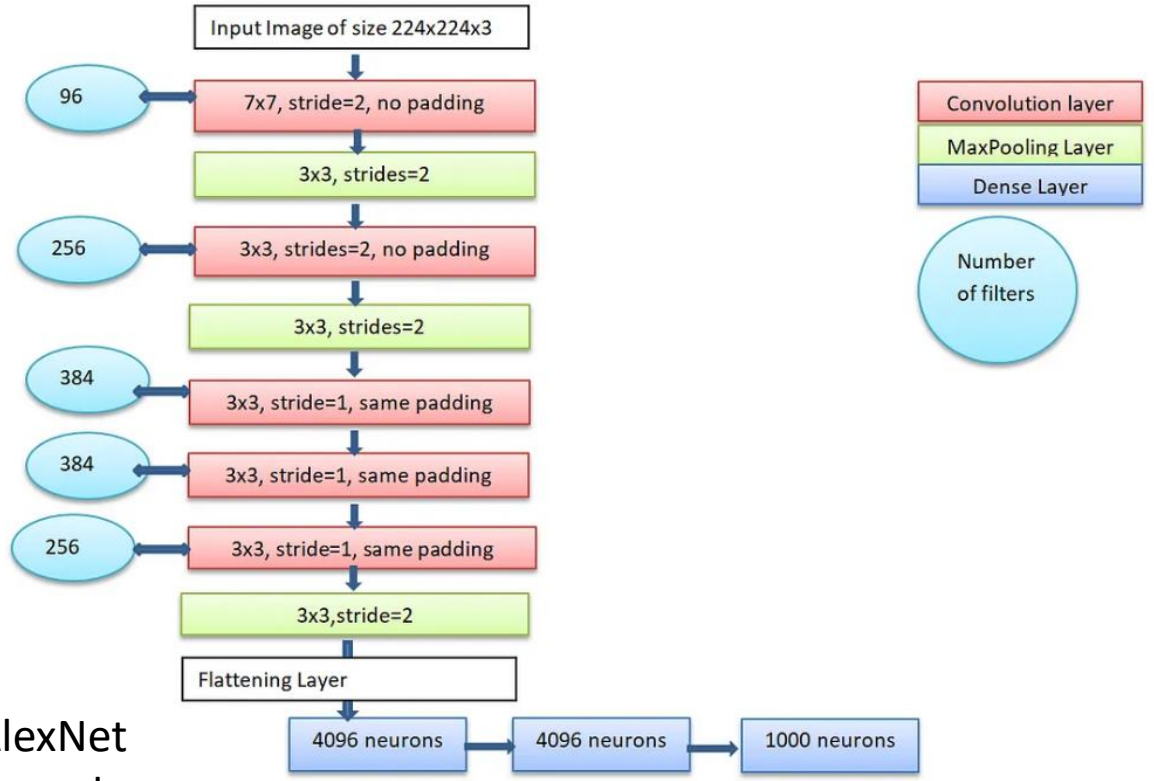


Large Scale Visual Recognition Challenge



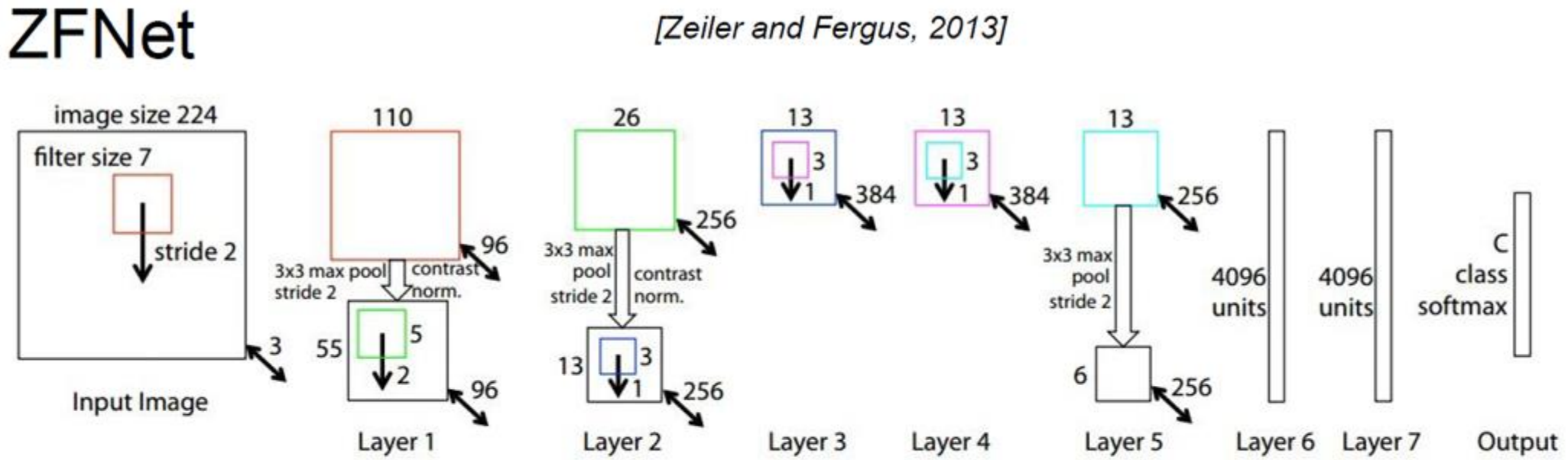
ZFNet

- Rob Fergus et Matthew D.Zeiler ont présenté ZFNet.
- ZFNet est nommé d'après leurs noms de famille Zeiler et Fergus.
- ZFNet est une légère amélioration par rapport à AlexNet .
- L'ILSVRC 2013 a été remporté par ZFNet.



- Il a permis de visualiser les performances de chaque couche d'AlexNet et les paramètres qui peuvent être réglés pour obtenir une plus grande précision.

ZFNet



TODO: remake figure

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

Some Key Features of ZFNet architecture

Couches convolutives :
Dans ces couches, des filtres convolutifs sont appliqués pour extraire les caractéristiques importantes. ZFNet se compose de plusieurs couches convolutives pour extraire les caractéristiques importantes.

· **MaxPooling Layers:**
Les couches MaxPooling sont utilisées pour réduire l'échantillonnage des dimensions spatiales de la carte des caractéristiques et sont constituées d'une fonction d'agrégation connue sous le nom de maxima.

· **Rectified Linear Unit:**
Le relu est utilisé après chaque couche de convolution pour introduire la non-linéarité dans le modèle, ce qui est crucial pour l'apprentissage de modèles complexes. Il rectifie la carte des caractéristiques en veillant à ce que les cartes des caractéristiques soient toujours positives.

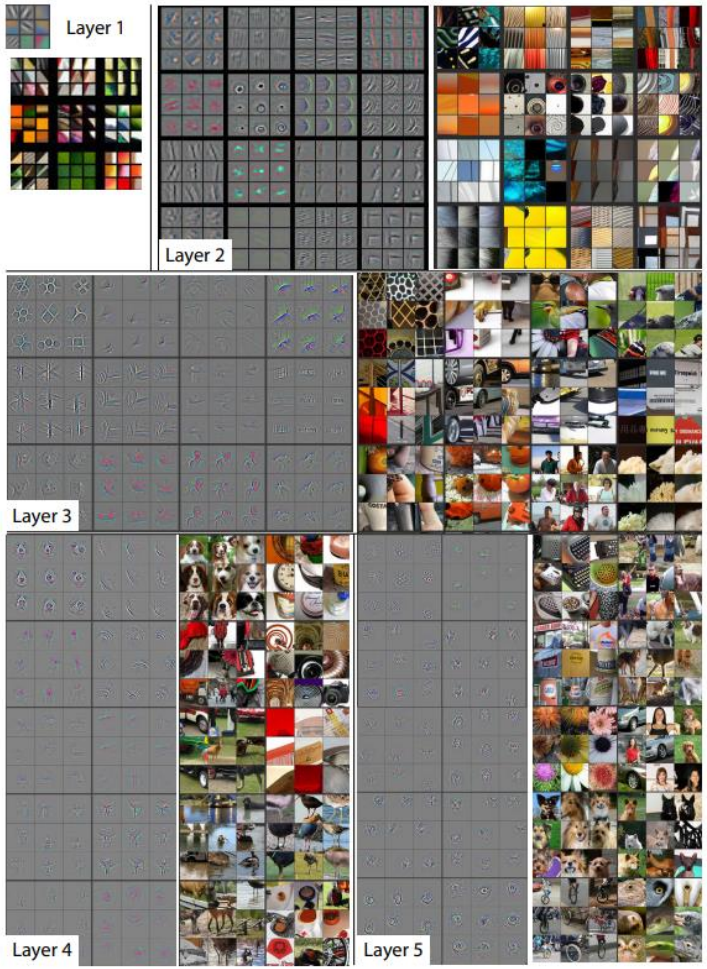
Some Key Features of ZFNet architecture

· **Fully Connected Layers:**
Dans la dernière partie de l'architecture de ZFNet, des couches denses entièrement connectées sont utilisées pour extraire des modèles à partir de caractéristiques.

· **SoftMax Activation:**
L'activation SoftMax est utilisée dans la dernière couche pour obtenir les probabilités d'appartenance de l'image aux 1000 classes.

· **Deconvolution Layers:** ZFNet a introduit une technique de visualisation impliquant des couches déconvolutionnelles (couches transposées). Ces couches donnent un aperçu de ce que le réseau a appris en projetant les activations des caractéristiques dans l'espace des pixels d'entrée.

Application



Visualizing and Understanding Convolutional Networks

Matthew D. Zeiler ZEILER@CS.NYU.EDU
Dept. of Computer Science, Courant Institute, New York University

Rob Fergus FERGUS@CS.NYU.EDU
Dept. of Computer Science, Courant Institute, New York University

Abstract

Large Convolutional Network models have recently demonstrated impressive classification performance on the ImageNet benchmark (Krizhevsky et al., 2012). However there is no clear understanding of why they perform so well, or how they might be improved. In this paper we address both issues. We introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier. Used in a diagnostic role, these visualizations allow us to find model architectures that outperform Krizhevsky *et al.* on the ImageNet classification benchmark. We also perform an ablation study to discover the performance contribution from different model layers. We show our ImageNet model generalizes well to other datasets: when the softmax classifier is retrained, it convincingly beats the current state-of-the-art results on Caltech-101 and Caltech-256 datasets.

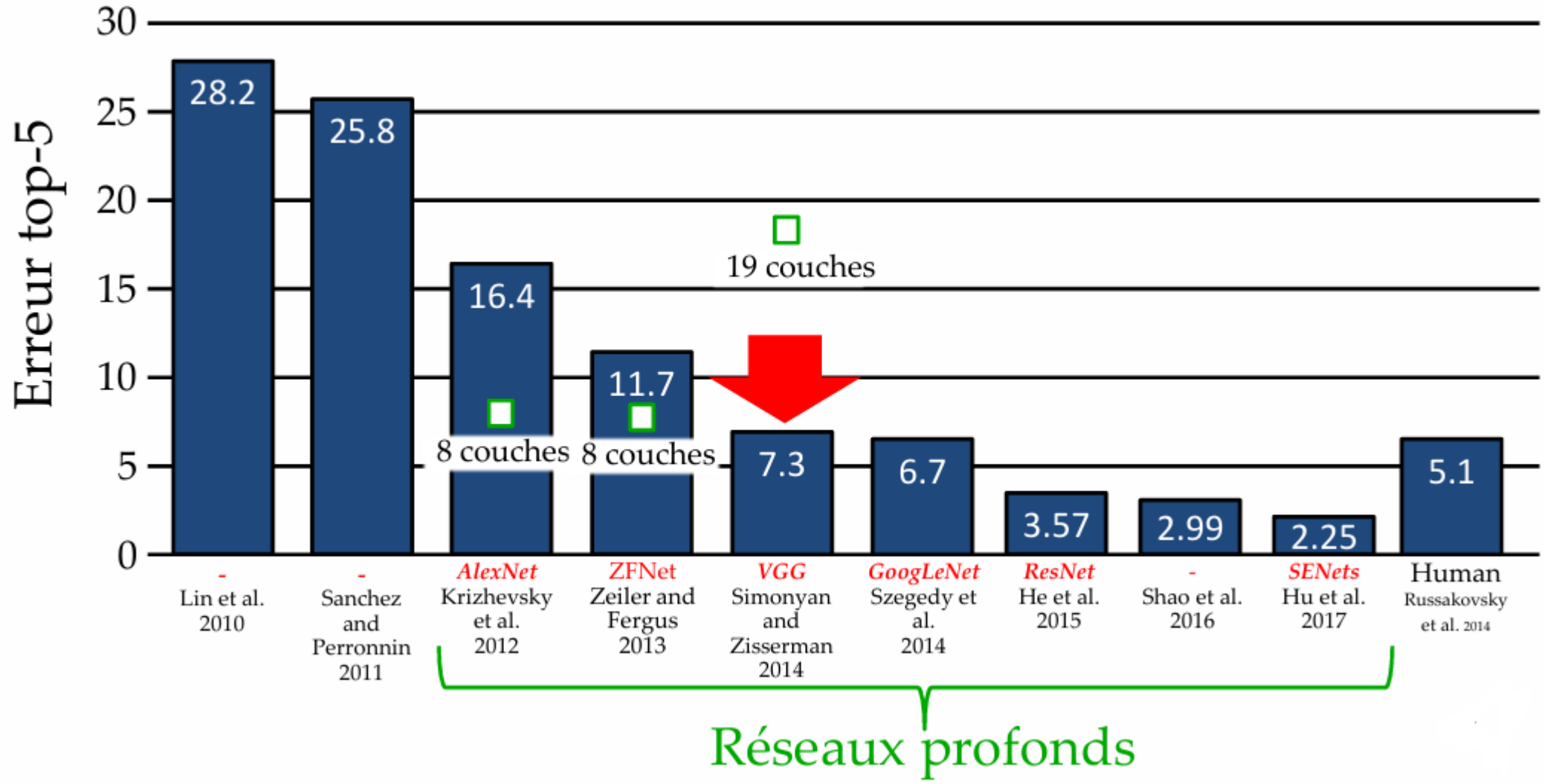
1. Introduction

Since their introduction by (LeCun et al., 1989) in the early 1990's, Convolutional Networks (convnets) have demonstrated excellent performance at tasks such as convnet models: (i) the availability of much larger training sets, with millions of labeled examples; (ii) powerful GPU implementations, making the training of very large models practical and (iii) better model regularization strategies, such as Dropout (Hinton et al., 2012).

Despite this encouraging progress, there is still little insight into the internal operation and behavior of these complex models, or how they achieve such good performance. From a scientific standpoint, this is deeply unsatisfactory. Without clear understanding of how and why they work, the development of better models is reduced to trial-and-error. In this paper we introduce a visualization technique that reveals the input stimuli that excite individual feature maps at any layer in the model. It also allows us to observe the evolution of features during training and to diagnose potential problems with the model. The visualization technique we propose uses a multi-layered Deconvolutional Network (deconvnet), as proposed by (Zeiler et al., 2011), to project the feature activations back to the input pixel space. We also perform a sensitivity analysis of the classifier output by occluding portions of the input image, revealing which parts of the scene are important for classification.

Using these tools, we start with the architecture of (Krizhevsky et al., 2012) and explore different archi-

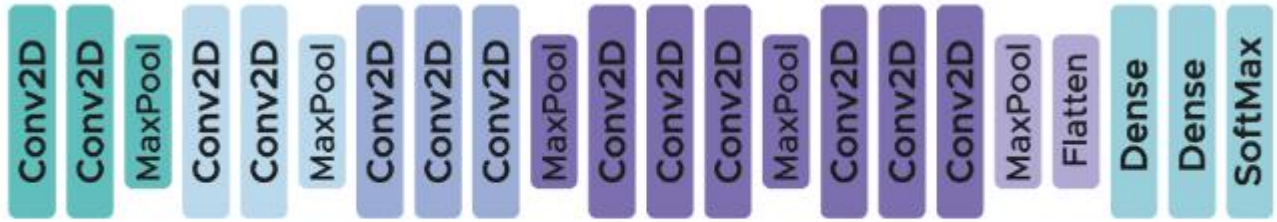
Large Scale Visual Recognition Challenge



VGG

Dans ImageNet, le modèle VGG16 atteint une précision d'environ 92,7 % dans le top 5 des tests. Un ensemble de données appelé ImageNet comprend plus de 14 millions de photos qui se répartissent en près de 1 000 types. Ce modèle a également été l'un des plus appréciés lors de l'ILSVRC-2014.

Il est nettement plus performant que le modèle AlexNet en substituant plusieurs filtres de la taille d'un noyau de 3x3 aux filtres de la taille d'un énorme noyau. Les GPU Nvidia Titan Black ont été utilisés pour entraîner le modèle VGG16 pendant plusieurs semaines.

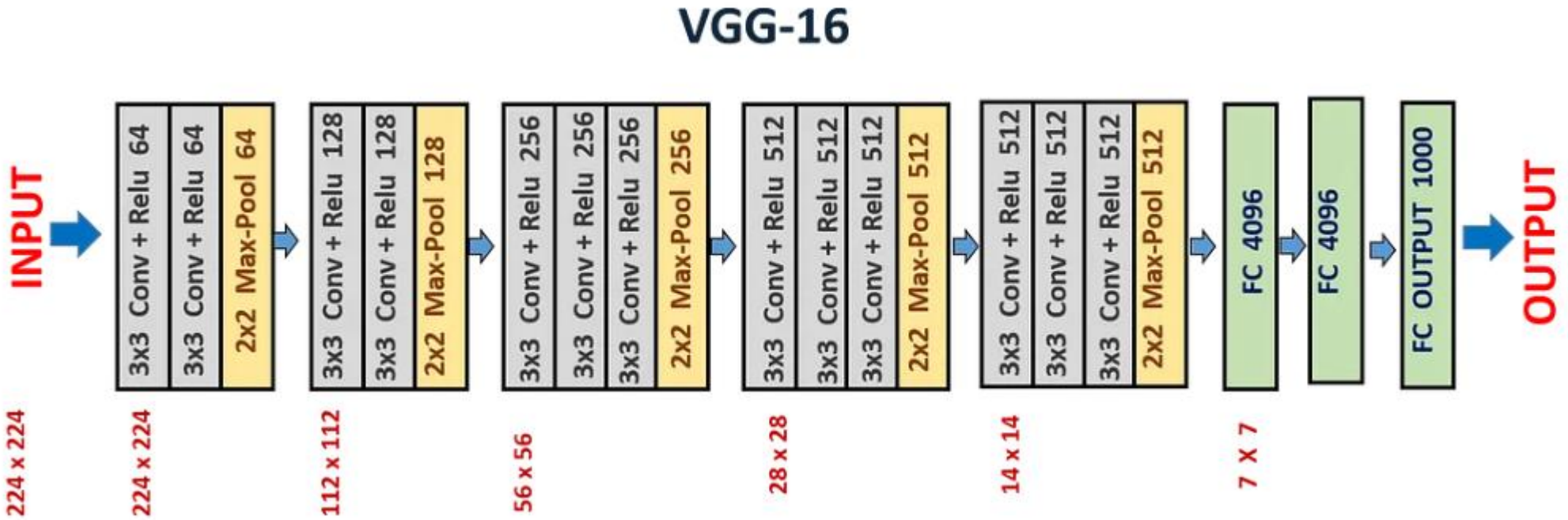


VGG16 Architecture

VGGNet

Le VGGNet-16 comporte 16 couches et peut classer les photos dans 1000 catégories d'objets différentes, dont le clavier, les animaux, le crayon, la souris, etc.

Le modèle accepte également les images d'une résolution de 224 par 224.



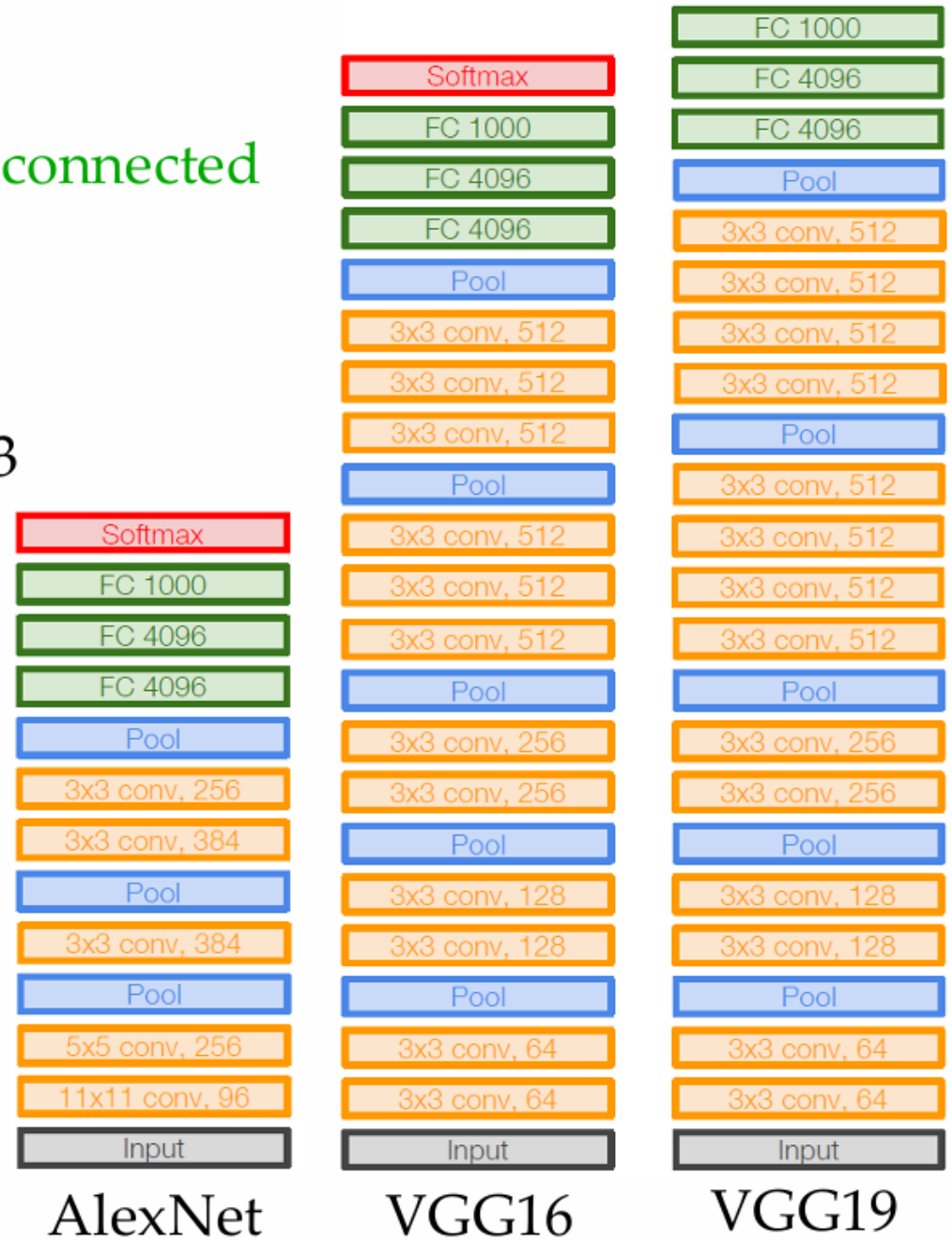
VGGNet

- Toujours 3 couches fully-connected comme classificateur
- 16-19 couches
- 138M paramètres
- Que des convolutions 3x3
- Empilement de 3 convolution 3x3 a le même champ récepteur qu'un filtre 7x7
  - Mais plus de non-linéarité (si ReLU)
  - Moins de paramètres :

    $3(3^2C^2)$ vs. 7^2C^2 , avec

    C channels en entrée-sortie

    (économie de 45%)



adapté de : cs231n

Analyse de la Procédure d'Entraînement de VGGNet

- Procédure complexe d’entraînement
  – entraine petit réseau
  – puis insère des nouvelles couches au milieu, initialisées au hasard

Cela permettait une optimisation progressive, mais rendait le processus long et complexe.

- Procédure inutile : Il est intéressant de noter qu'après la soumission de l'article, nous avons constaté qu'il est possible d'initialiser les poids sans pré-entraînement en utilisant la procédure d'initialisation de Gloriot & Bengio (2010).

•Cela simplifie grandement l’entraînement des réseaux profonds comme VGGNet.

Glorot & Bengio (2010), "Understanding the difficulty of training deep feedforward neural networks".

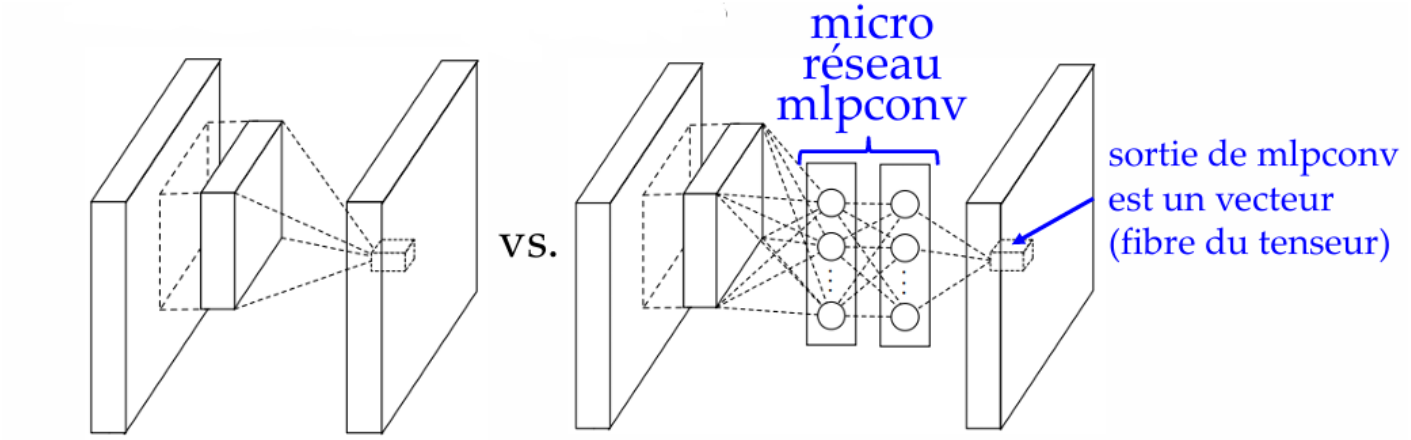
Limitation des CNN Classiques

- Les filtres traditionnels (convolution linéaire + activation) supposent que les **features sont linéairement séparables**.

- Cela limite leur capacité à modéliser des représentations complexes.

Network In Network (NIN)

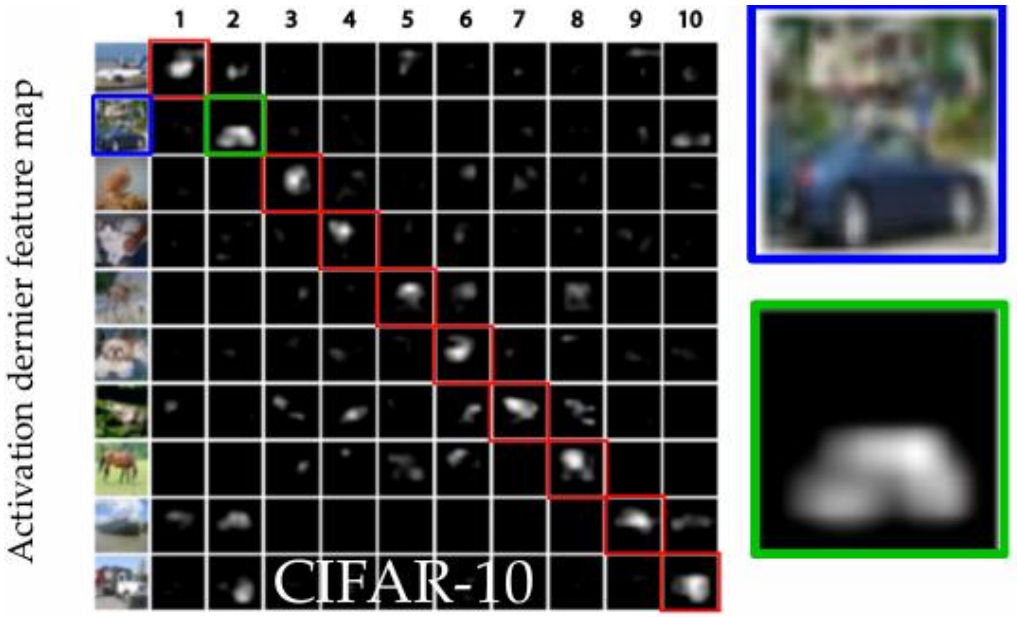
- Les filtres CNN supposent que les features sont linéairement séparables

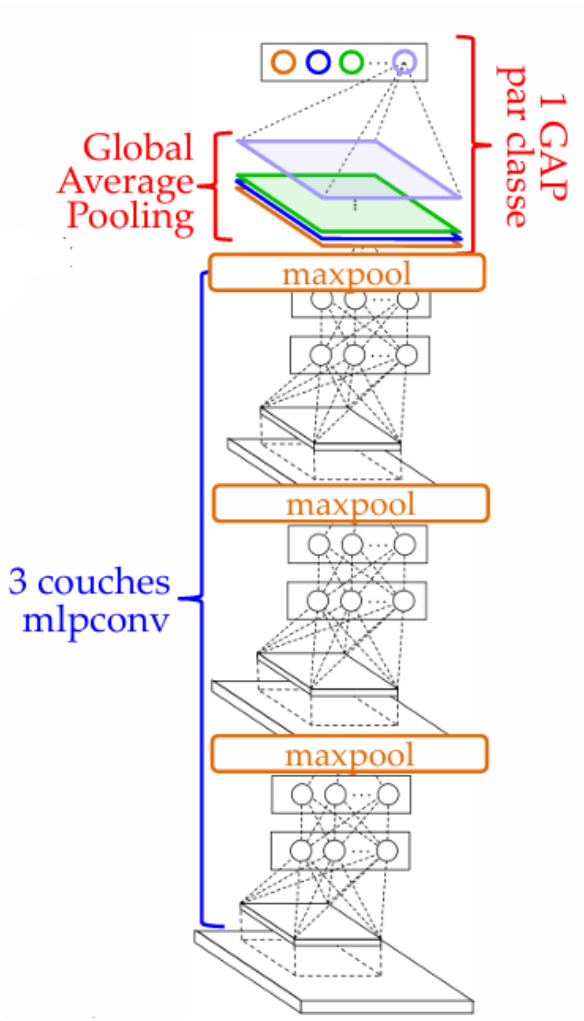


- Remplacé par un micro-réseau de neurones (mlpconv), qui peut exprimer des fonctions non-linéaires
- Partagés, comme dans les filtres CNN
- Utilisation des convolutions 1x1 (approfondi plus loin)

NiN

- Introduit le Global Average Pooling (GAP) sur les features map finaux
- Moyenne d'un feature map au complet
- 1 par classe, connecté au softmax
- Force la corrélation entre un feature map et une classe :





- Facilite l'interprétation des erreurs du réseau

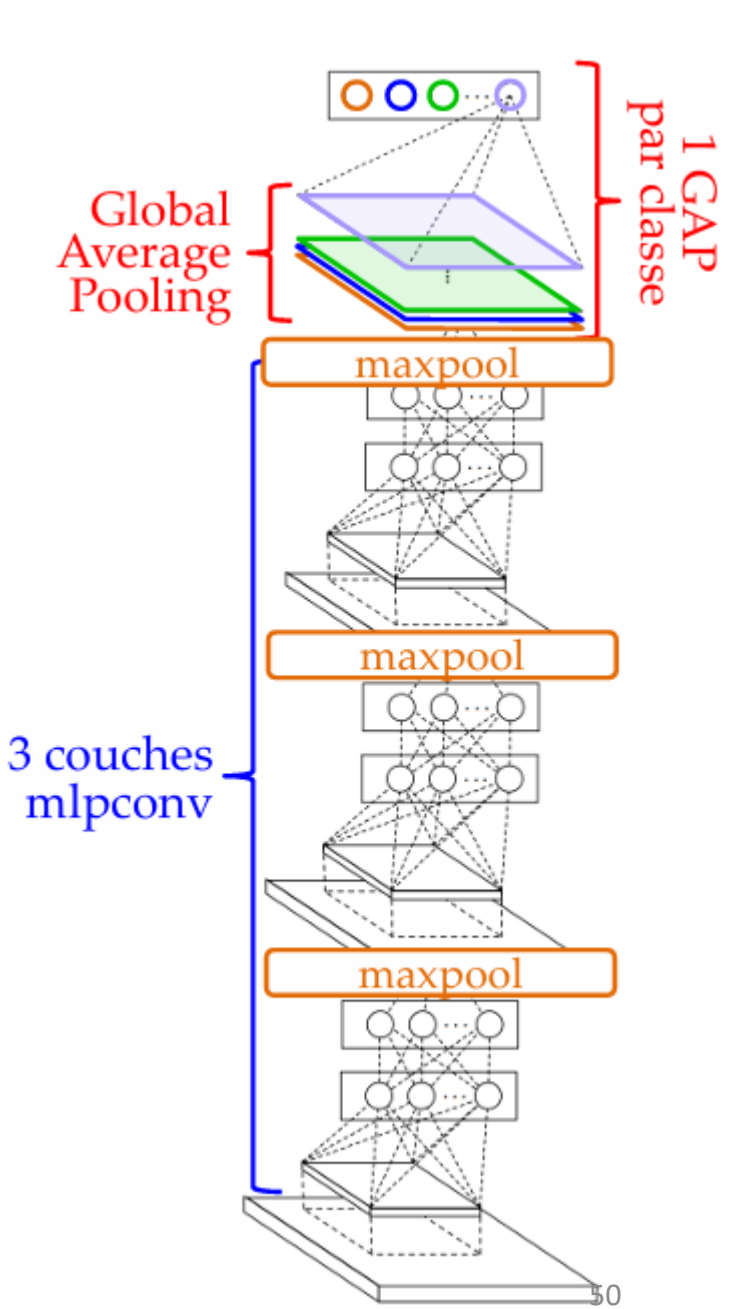
Comparaison avec les CNN Classiques

Aspect	CNN Traditionnel	NiN + GAP
Couche finale	Fully Connected (FC) lourde	Global Average Pooling légère
Paramètres	Nombreux (risque d'overfitting)	Minimalistes (efficace)
Interprétabilité	Faible (features abstraits)	Forte (1 feature map = 1 classe)

NiN

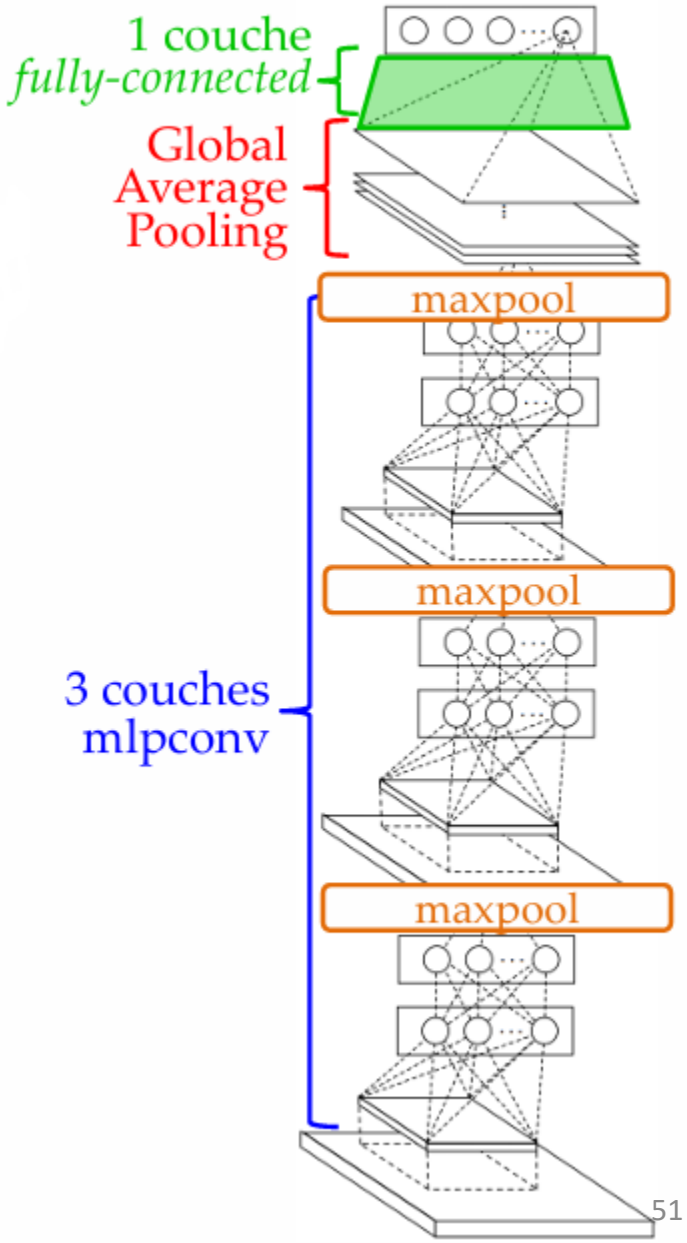
- GAP agit comme régularisateur structurel

Method	Testing Error
mlpconv + Fully Connected	11.59%
mlpconv + Fully Connected + Dropout	10.88%
mlpconv + Global Average Pooling	10.41%



- Puissance d'extraction des filtres micro-réseaux améliore tellement la qualité des features que le classificateur n'est plus nécessaire
- Dropout sur les sorties mlpconv 1 et 2

NiN



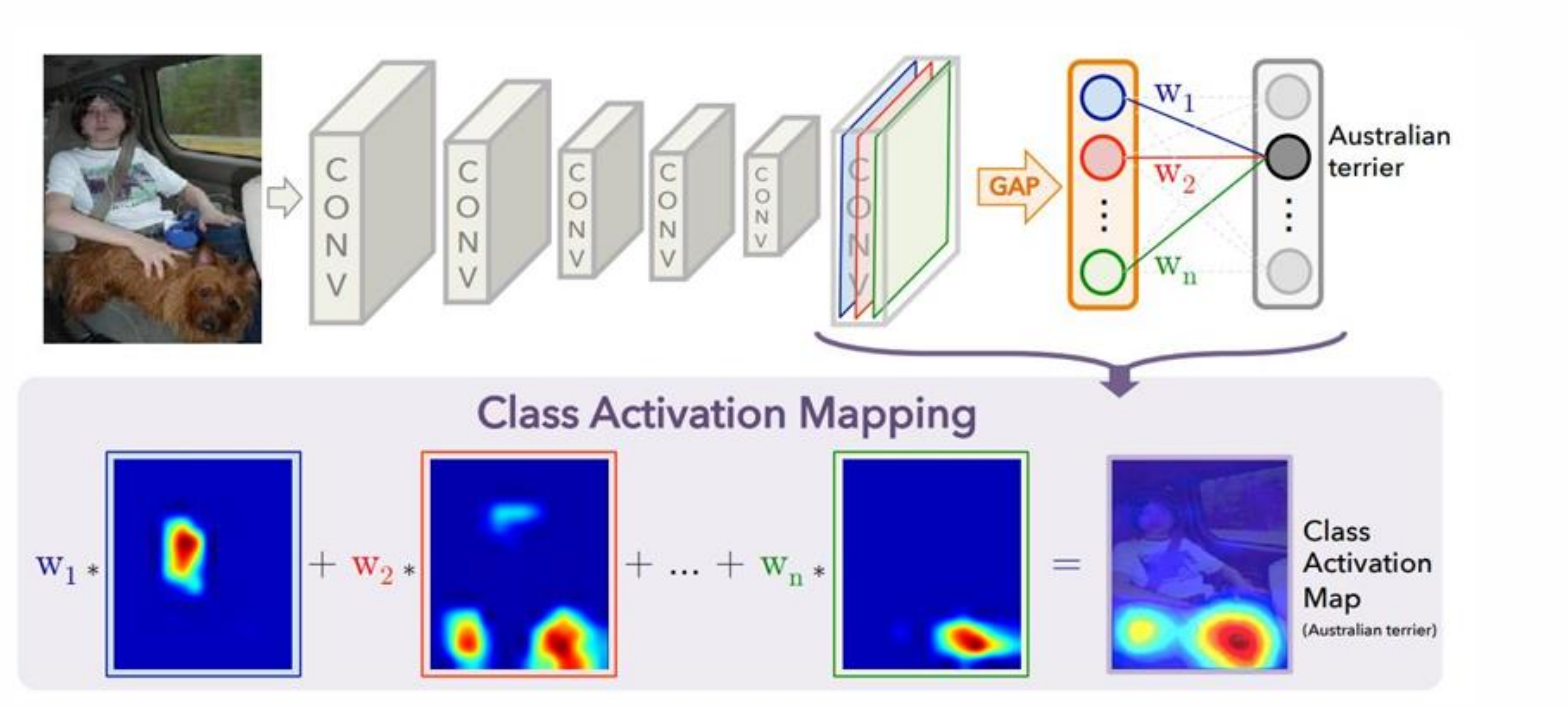
• Certaines implémentation de NiN semblent utiliser une couche de fully connected comme classificateur
– Beaucoup moins de paramètres
– Beaucoup moins d'overfit

• Take-home message reste le même : plus besoin d'un classificateur puissant en sortie

• Tendance forte des prochaines architectures

GAP : localisation d'objet gratuite!

http://cnnlocalization.csail.mit.edu/



- Donne une certaine interprétabilité aux résultats

Zhou et al., Learning Deep Features for Discriminative Localization, CVPR 2016.

Application