Apprentissage statistique

Apprentissage supervisé: apprentissage neuronal profond

Cours pour non-spécialistes

Aurélien Garivier

Réseaux de neurones

Réseau mono-couche

Src : http://insanedev.co.uk/open-cranium/

Réseau mono-couche

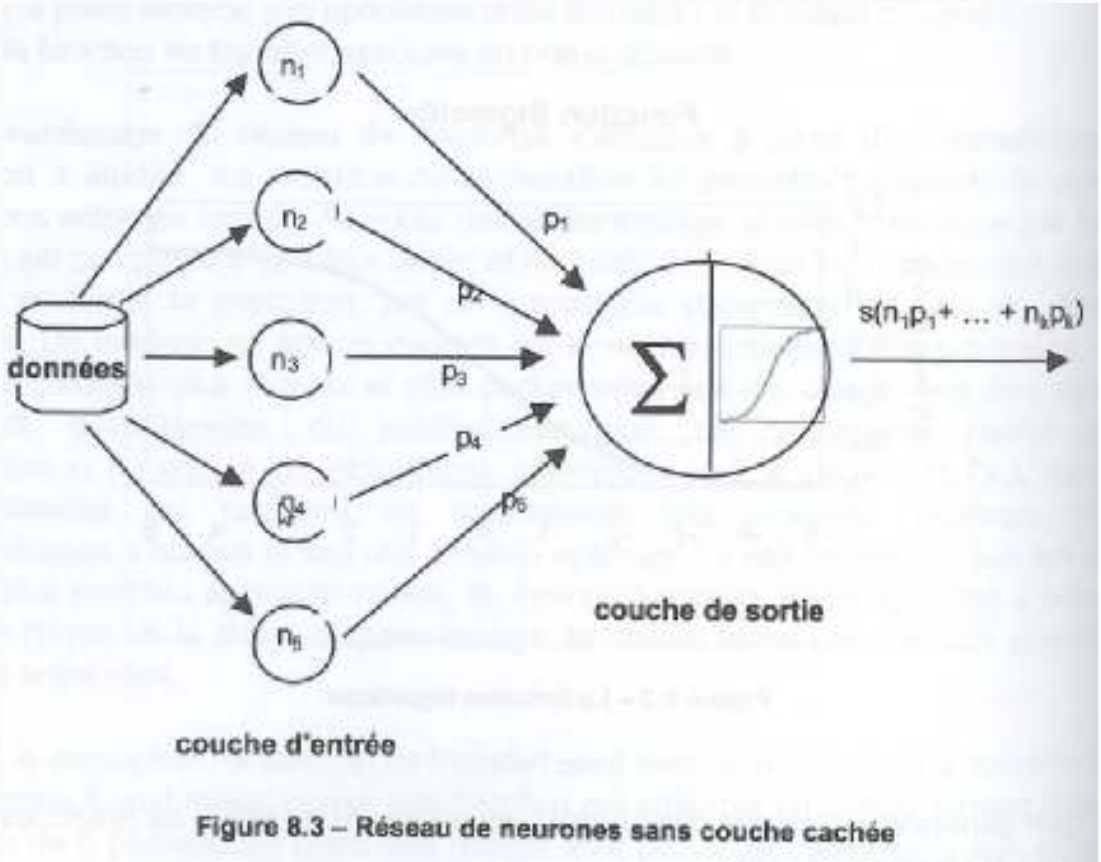
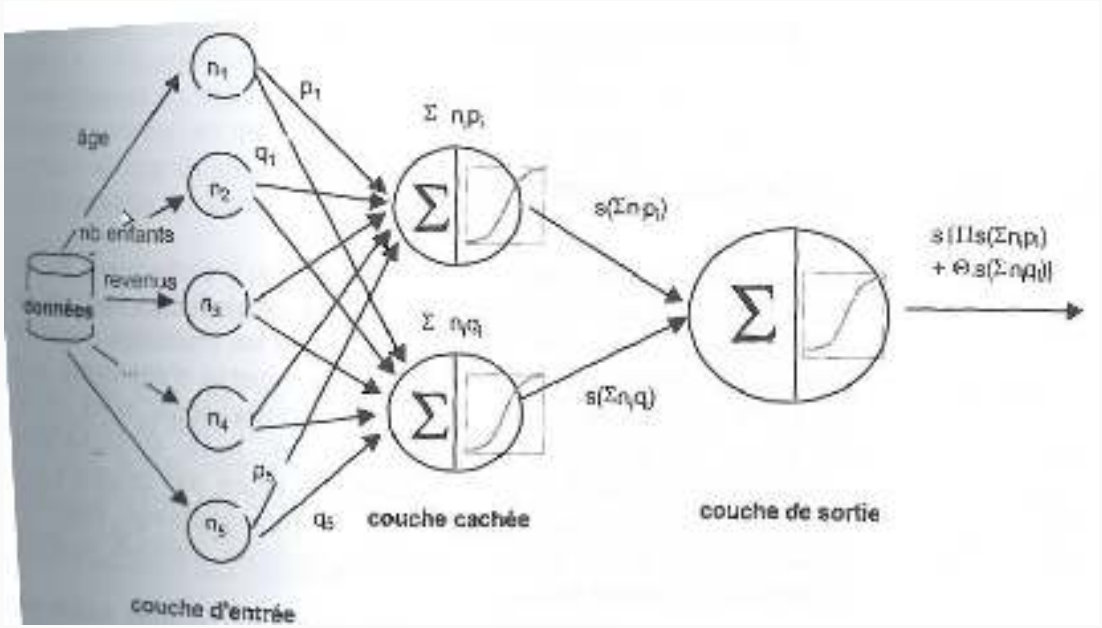


Figure 8.3 – Réseau de neurones sans couche cachée

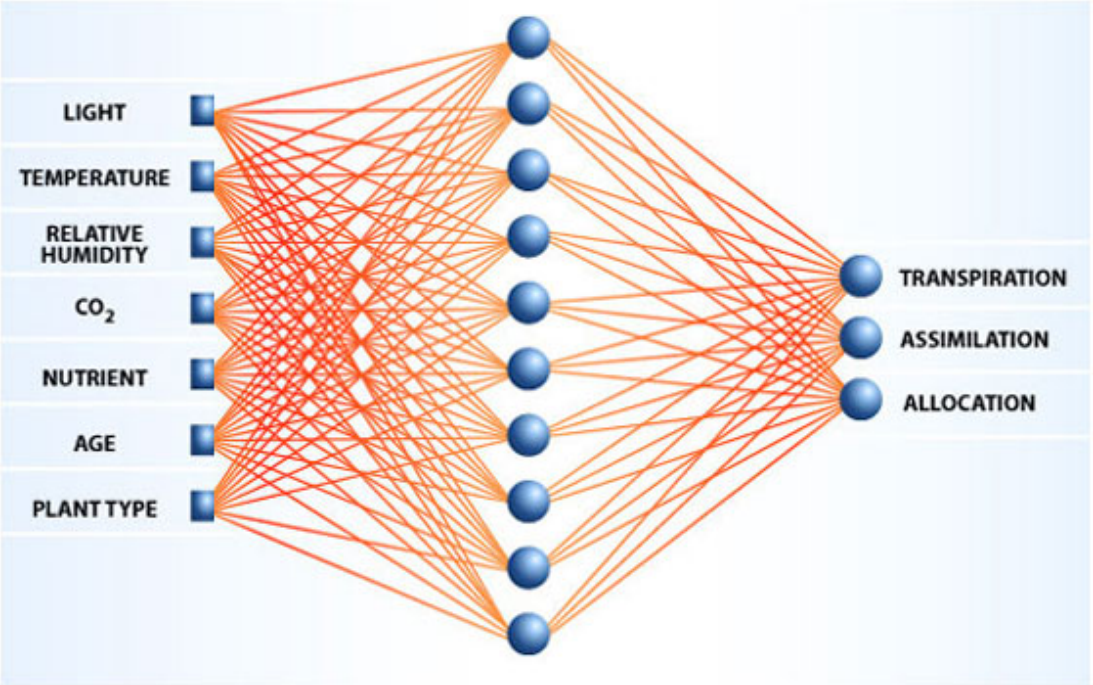
Source : [Tufféry, Data Mining et Informatique Décisionnelle]

Réseau avec couche intermédiaire



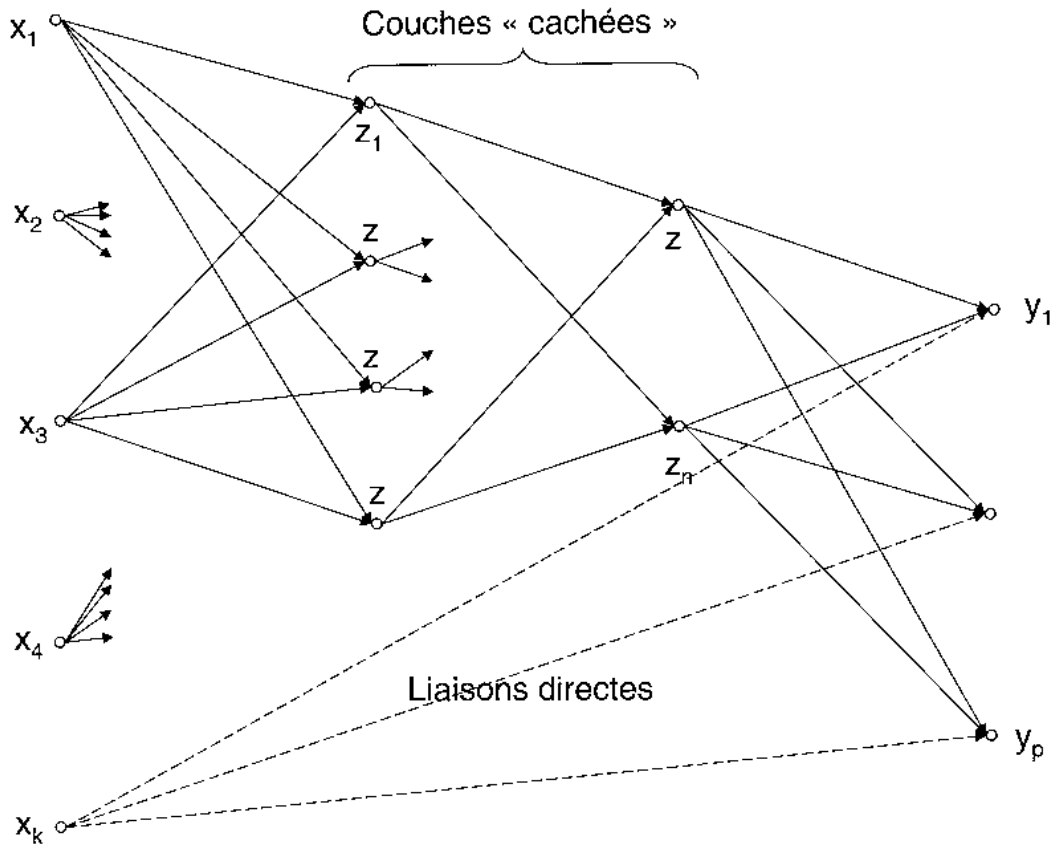
Source : [Tufféry, Data Mining et Informatique Décisionnelle]

Réseau mono-couche



Src : `http://www.makhfi.com`

- Un réseau de neurone est l'association d'objets élémentaires : les neurones formels. C'est en général l'organisation de ces neurones, le nombre de neurones, et leurs types, qui distinguent ces différents réseaux.



- Le neurone formel est un modèle qui se caractérise par un état interne $s \in \mathcal{S}$, des signaux d'entrée $x_1, \cdots, x_p$ et une fonction de transition d'état

$$s = f\left(\beta_0 + \sum_{j=1}^{p} \beta_j x^j\right).$$

- β_0 correspond au biais du neurone.

- Combinaison affine est déterminée par un vecteur de poids $[\beta_0, \cdots, \beta_p]$ associé à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage ("mémoire" ou "connaissance répartie" du réseau).

- Il y a plusieurs fonctions de transition possibles :
  - $f(x) = x$ neurone linéaire
  - $f(x) = 1/(1 + e^x)$ neuron sigmoïde
  - $f(x) = \mathbf{1}_{[0;+\infty[}$ fonction de seuillage
  - $f(x) = 1$ avec la probabilité $1/(1 + e^{-H(x)})$ neurone stochastique

- La plupart des problèmes d'apprentissage utilisent les deux premières fonctions de transition.

- La fonction représentant vraisemblablement le mieux la réalité est la fonction de seuillage. Mais elle n'est pas différentiable et donc peu adaptée pour les statisticiens.

- Réseau composé de couches successsives.
  - Pas de connexion entre des réseaux de la même couche.
  - Couche d'entrée $\mathcal{C}_1$: un neurone par variable x^j.
  - Couche de sortie fournit la prédiction y.
- Plusieurs couches cachées.
- Chaque neurone de la couche cachée $\mathcal{C}_k$ est connectée en entrée à chaque neurone de la couche précédente $\mathcal{C}_{k-1}$.
- y est donc calculé par le biais de

$$y = \phi(x^1, \cdots, x^p, \beta) \text{ où } \beta = (\beta_{j,k,l}),$$

$\beta_{j,k,l}$ représente le paramètre de la $j$ème entrée du $k$ème neurone de la $l$ème couche.

- On dispose de $(x_i^1, \cdots, x_i^p, y_i)$ n observations de varibales explicatives $X^1, \cdots, X^p$ et Y variable à prédire.
- On cherche $\hat{\beta}$ solution du problème

$$\hat{\beta} = \text{argmin}_{\mathbf{b}} \mathcal{Q}(\mathbf{b}) \quad \text{où} \quad \mathcal{Q}(\mathbf{b}) = \frac{1}{n} \sum_{i=1}^{n} \left[y_i - \phi\left(x_i^1, \cdots, x_i^p, \mathbf{b} \right) \right]^2 .$$

- C'est un problème hautement non linéaire, le plus souvent. On adopte une méthode de descente de gradient.

- En tout point **b**, le gradient $\mathcal{Q}$ pointe dans la direction de l'erreur croissante.

- Il suffit donc de se déplacer en sens contraire, l'algorithme est itératif modifiant les poids de chaque neurone selon

$$b_{jkl}(i) = b_{jkl}(i-1) + \Delta b_{jkl}(i).$$

- La correction $\Delta b_{jkl}(i)$ est proportionnelle au gradient et à l'erreur attribuée à l'entrée concernée $\epsilon_{jkl}(i)$ et incorpore un terme d'inertie $\alpha b_{jkl}(i-1)$ permettant d'amortir les oscillations du système

$$\Delta b_{jkl}(i) = -\tau \epsilon_{jkl}(i) \frac{\partial \mathcal{Q}}{\partial b_{jkl}}(\mathbf{b}) + \alpha b_{jkl}(i-1).$$

- Le coefficient de proportionnalité τ est appelé taux d'apprentissage. Il peut être fixe, à déterminer par l'utilisateur ou varier en cours d'exécution selon certaines règles paramétrées par l'utilisateur.
- Intuitivement : τ doit être grand au début pour aller plus vite puis décroître pour aboutir à un réglage plus fin au fur et à mesure que le système s'approche d'une solution.
- Une amélioration importante consiste à introduire un terme de pénalisation ou régularisation dans le critère à optimiser :

$$\hat{\beta} = \text{argmin}_{\mathbf{b}} \mathcal{Q}(\mathbf{b}) + \delta ||\mathbf{b}||^2.$$

Le paramètre δ (decay) doit être fixé par l'utilisateur. Plus il est important et moins les paramètres ou poids peuvent prendre des valeurs "chaotiques"' contribuant ainsi à limiter le risque de sur-apprentissage.

- Initialisation
  - Les poids $\beta_{j,k,l}$ par tirage aléatoire selon une loi uniforme sur $[0; 1]$.
  - Normalisation dans $[0; 1]$ des données d'apprentissage.
- Tant que $\mathcal{Q} >$ errmax ou niter $<$ itermax Faire
  - Ranger la base d'apprentissage dans unn nouvel ordre aléatoire.
  - Pour chaque élément $i = 1, \cdots, n$ de la base Faire
    - ⋆ Calculer $\epsilon(i) = y_i - \phi(x_i^1, \cdots, x_i^p, \mathbf{b}(i-1))$ en propageant les entrées vers l'avant.
    - ⋆ L'erreur est "rétropropagée" dans les différentes couches afin d'affecter à chaque entrée une responsabilité dans l'erreur globale.
    - ⋆ Mise à jour de chaque poids $b_{jkl}(i) = b_{jkl}(i-1) + \Delta b_{jkl}(i)$.
  - Fin du Pour
- Fin du Tant que

L'utilisateur doit déterminer

- les variables d'entrée et celle de sortie.
- L'architecture du réseau :
  - le nombre de couches cachées, en général 1 ou 2, qui correspond à une aptitude à triater des problèmes de non-linéarité.
  - le nombre de neurones par couche cachée.
  - Ces 2 choix conditionnent directement le nombre de paramètres (poids) à estimer. Ils participent à la recherche d'un bon compromis biais/variance, i.e. équilibre entre qualité d'apprentissage et qualité de prévision.
  - En pratique, on considère qu'il faut un échantillon d'apprentissage au moins 10 fois plus grand que le nombre de paramètes à estimer.
- Le nombre de maximum d'itérations pour la recherche de $\hat{\beta}$, l'erreur maximale tolérée et le terme de régularisation (decay). En renforçant ces critères, on améliore la qualité d'apprentissage au détriment éventuel de celle de la prévision.
- Le taux d'apprentissage et son éventuelle stratégie d'évolution.

En pratique, tous ces paramètres ne sont pas réglés simultanément par l'utilisateur. Celui-ci est confronté à des choix concernant principalement le contrôle du sur-apprentisage.

- choix du paramètre : limiter le nombre de neurones ou la durée d'apprentissage ou encore augmenter le coefficient de pénalisation de la norme des paramètres.
- choix du mode d'estimation de l'erreur : échantillon test, validation croisée ou bootstrap.

Ces choix sont souvent pris par défaut dans les logiciels. Il est important d'en connaître les implications.

Le nombre de couches reste restreint. En effet toute fonction continue dun compact de $\mathbb{R}^p$ dans $\mathbb{R}^q$ peut être approchée avec une précision arbitraire par un réseau de neurones à une couche cachée en adaptant le nombre de neurones.

Le contrôle de la complexité du modèle peut se faire à l'aide de plusieurs paramètres :

- le nombre de neurones,
- une pénalisation de la norme du vecteur des poids
- ou par la durée de l'apprentissage.

Ces paramètres sont optimisés en considérant un échantillon de validation. Le plus simple consiste à arrêter l'apprentissage lorsque

- l'erreur sur l'échantillon de validation commence à se dégrader
- tandis que celle sur l'échantillon sur l'échantillon d'apprentissage ne peut que continuer à décroître.

Initialization

- The input data have to be normalized to have approximately the same range.
- The biases can be initialized to 0. The weights cannot be initialized to 0 since for the tanh activation function, the derivative at 0 is 0, this is a saddle point.
- They also cannot be initialized with the same values, otherwise, all the neurons of a hidden layer would have the same behaviour.
- We generally initialize the weights at random : the values $W_{i,j}^{(k)}$ are i.i.d. Uniform on $[-c, c]$ with possibly $c = \frac{\sqrt{6}}{N_k + N_{k-1}}$ where N_k is the size of the hidden layer k. We also sometimes initialize the weights with a normal distribution $\mathcal{N}(0, 0.01)$ (see Gloriot and Bengio, 2010).

Optimization algorithms

- Many algorithms can be used to minimize the loss function with hyperparameters to calibrate.
- The Stochastic Gradient Descent (SGD) algorithm :

$$\theta_i^{new} = \theta_i^{old} - \varepsilon \frac{\partial L}{\partial \theta_i}(\theta_i^{old}),$$

where ε is the *learning rate* , and its calibration is very important for the convergence of the algorithm.

  - If it is too small, the convergence is very slow and the optimization can be blocked on a local minimum.
  - If the learning rate is too large, the network will oscillate around an optimum without stabilizing and converging.
  - A classical way to proceed is to adapt the learning rate during the training : recommended to begin with a "large " value of ϵ, and reduce this value during the successive iterations : The observation of the evolution of the loss function can give indications on the way to proceed.

Optimization algorithms

- Another stopping rule, called *early stopping* is also used : we stop learning when the loss function for a validation sample stops to decrease.
- *Batch learning* is used for computational reasons, backpropagation algorithms need to store intermediate values : for big data sets, such as millions of images, this is not feasible, all the more that the deep networks have millions of parameters to calibrate.
- The batch size m is also a parameter to calibrate. Small batches generally lead to better generalization properties.
- The particular case of batches of size 1 is called *On-line Gradient Descent*. The disadvantage of this procedure is the very long computation time.

SGD algorithm

Summary of Stochastic Gradient Descent algorithm

- Fix the parameters ε : learning rate, m : batch size, nb : number of epochs.
- For $l = 1$ to nb epochs
- For $l = 1$ to n/m,
  - Take a random batch of size m without replacement in the learning sample : $(X_i, Y_i)_{i \in B_l}$
  - Compute the gradients with the backpropagation algorithm

$$\tilde{\nabla}_\theta = \frac{1}{m} \sum_{i \in B_l} \nabla_\theta \ell(f(X_i, \theta), Y_i).$$

  - Update the parameters

$$\theta^{new} = \theta^{old} - \varepsilon \tilde{\nabla}_\theta.$$

SGD algorithm

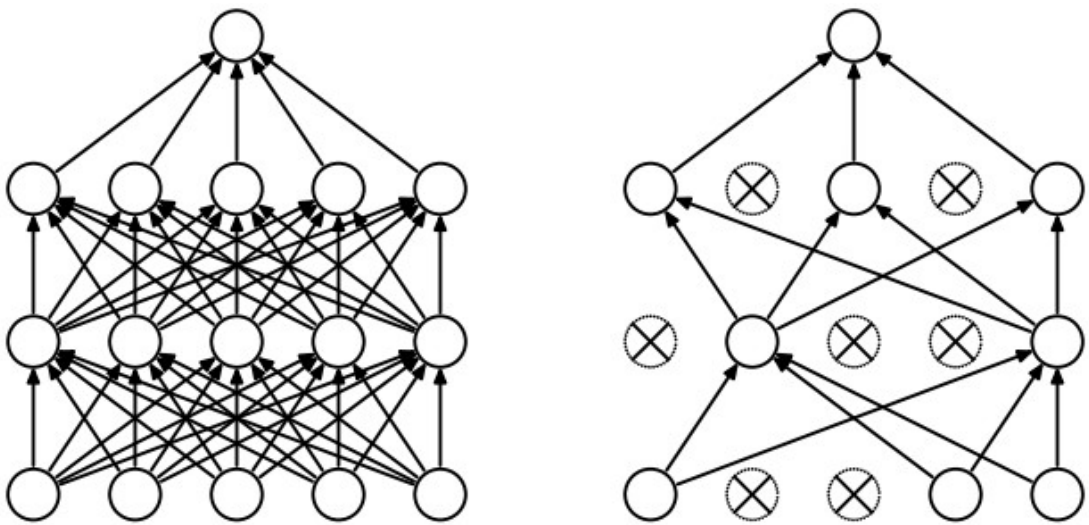
- Since the choice of the learning rate is delicate, variations of the algorithm that are less sensitive to this parameter have been proposed.
- **Nesterov accelerated gradient** : Nesterov (1983) and Sutskever et al. (2013)
- **RMSProp** algorithm, due to Hinton (2012) .

Regularization

- We have already mentioned $\mathbb{L}^2$ or $\mathbb{L}^1$ penalization ; we have also mentioned early stopping.
- For deep learning, the mostly used method is the **dropout** introduced by Hinton et al. (2012).
- With a certain probability p, and independently of the others, each unit of the network is set to 0.
- It is classical to set p to 0.5 for units in the hidden layers, and to 0.2 for the entry layer.
- The computational cost is weak since we just have to set to 0 some weights with probability p.

Dropout

- This method improves significantly the generalization properties of deep neural networks and is now the most popular regularization method in this context.
- The disadvantage is that training is much slower (it needs to increase the number of epochs).



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

FIGURE: Dropout

Convolutional neural networks

- For some types of data, especially for images, multilayer perceptrons are not well adapted.
- They are defined for vectors. By transforming the images into vectors, we loose spatial informations, such as forms.
- The **convolutional neural networks (CNN)** introduced by LeCun (1998) have revolutionized image processing.
- CNN act directly on matrices, or even on tensors for images with three RGB color chanels.

Convolutional neural networks

- CNN are now widely used for image classification, image segmentation, object recognition, face recognition ..

FIGURE: Image annotation

Convolutional neural networks

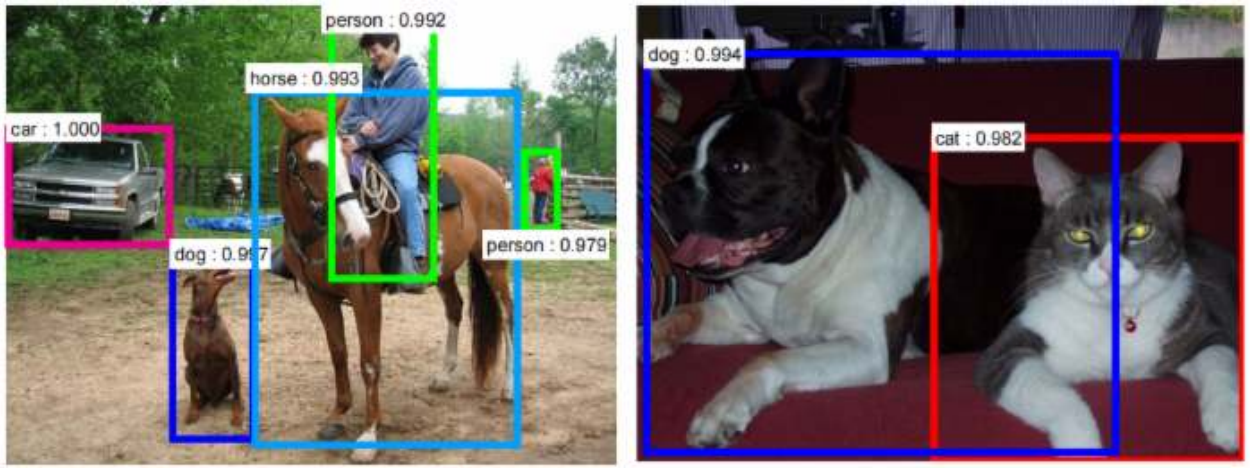


FIGURE: Image Segmentation.

Layers in a CNN

- A Convolutional Neural Network is composed by several kinds of layers, that are described in this section :
  - convolutional layers
  - pooling layers
  - fully connected layers

Convolution layer

- For 2-dimensional signals such as images, we consider the 2D-convolutions : $(K * I)(i,j) = \sum_{m,n} K(m,n) I(i+n, j+m)$.
- K is a convolution kernel applied to a 2D signal (or image) I.

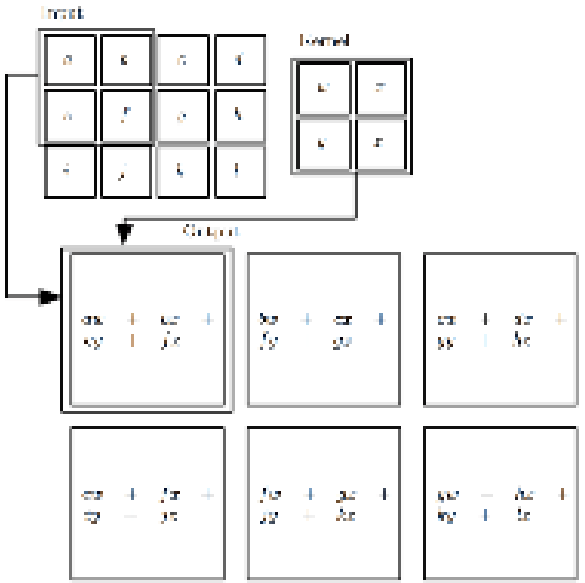


FIGURE: 2D convolution

Convolution layer

- The principle of 2D convolution is to drag a convolution kernel on the image.
- At each position, we get the convolution between the kernel and the part of the image that is currently treated.
- Then, the kernel moves by a number s of pixels, s is called the *stride*.
- When the stride is small, we get redondant information.
- Sometimes, we also add a *zero padding*, which is a margin of size p containing zero values around the image in order to control the size of the output.

Convolution layer

- Assume that we apply C_0 kernels, each of size $k \times k$ on an image.
- If the size of the input image is $W_i \times H_i \times C_i$ (W_i denotes the width, H_i the height, and C_i the number of channels, typically $C_i = 3$), the volume of the output is $W_0 \times H_0 \times C_0$, where C_0 corresponds to the number of kernels that we consider, and

$$W_0 = \frac{W_i - k + 2p}{s} + 1$$

$$H_0 = \frac{H_i - k + 2p}{s} + 1.$$

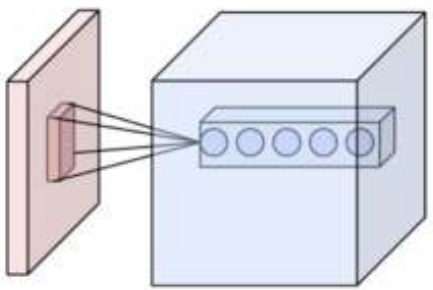
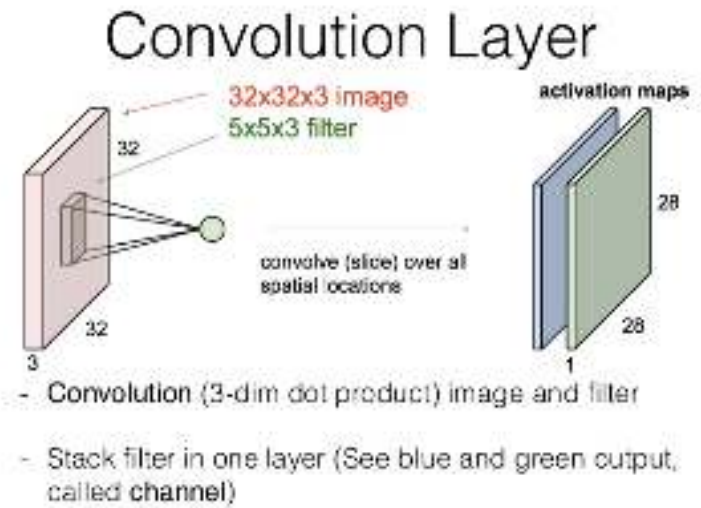


FIGURE: 2D convolution - Units corresponding to the same position but at various depths ; each unit applies a different kernel on the same patch of the

Convolution layer



- If the image has 3 channels and if K_l $(l = 1, \ldots, C_0)$ denote $5 \times 5 \times 3$ kernels, the convolution with the image I with the kernel K_l corresponds to the formula :

$$K_l * I(i,j) = \sum_{c=0}^{2} \sum_{n=0}^{4} \sum_{m=0}^{4} K_l(n,m,c) I(i+n-2, i+m-2, c).$$

Convolution layer

- For images with C^i channels, the shape of the kernel is (k, k, C^i, C^0) where C^0 is the number of output channels.
- The convolution operations are combined with an activation function ϕ (ReLu in general) : if we consider a kernel K of size $k \times k$, if $\boldsymbol{x}$ is a $k \times k$ patch of the image, the activation is obtained by sliding the $k \times k$ window and computing $z(\boldsymbol{x}) = \phi(K * x + b)$, where b is a bias.
- CNN learn the filters (or kernels) that are the most useful for the task that we have to do (such as classification). Several convolution layers are considered : the output of a convolution becomes the input of the next one.

Pooling layer

- CNN also have *pooling* layers, which allow to reduce the dimension, also referred as *subsampling*, by taking the mean or the maximum on patches of the image (mean-pooling or max-pooling).
- Like the convolutional layers, pooling layers acts on small patches of the image.
- If we consider 2×2 patches, over which we take the maximum value to define the output layer, with a stride 2, we divide by 4 the size of the image.

Pooling layer

- Another advantage of the pooling is that it makes the network less sensitive to small translations of the input images.

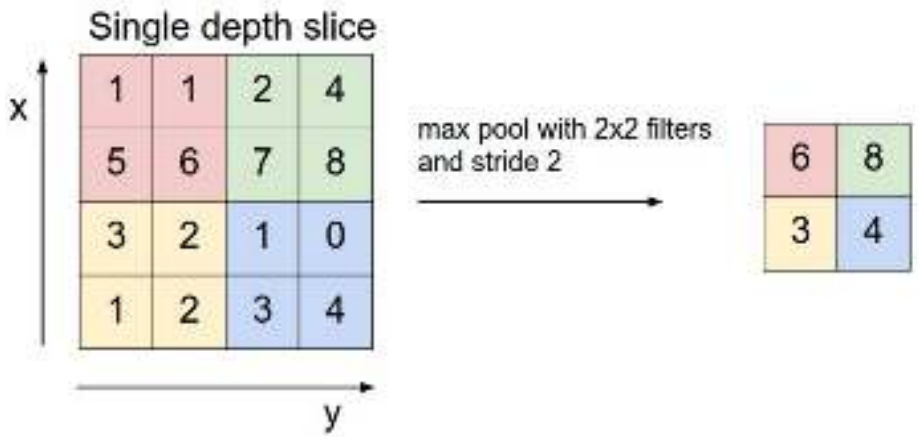


FIGURE: Maxpooling and effect on the dimension

Fully connected layers

- After several convolution and pooling layers, the CNN generally ends with several *fully connected* layers.
- The tensor that we have at the output of these layers is transformed into a vector and then we add several perceptron layers.

Architectures

- We have described the different types of layers composing a CNN.
- We now present how this layers are combined to form the architecture of the network.
- Choosing an architecture is very complex and this is more engineering that an exact science.
- It is therefore important to study the architectures that have proved to be effective and to draw inspiration from these famous examples.
- In the most classical CNN, we chain several times a convolution layer followed by a pooling layer and we add at the end fully connected layers.

Architectures

The **LeNet** network, proposed by the inventor of the CNN, Yann LeCun (1998) is of this type. This network was devoted to digit recognition. It is composed only on few layers and few filters, due to the computer limitations at that time.

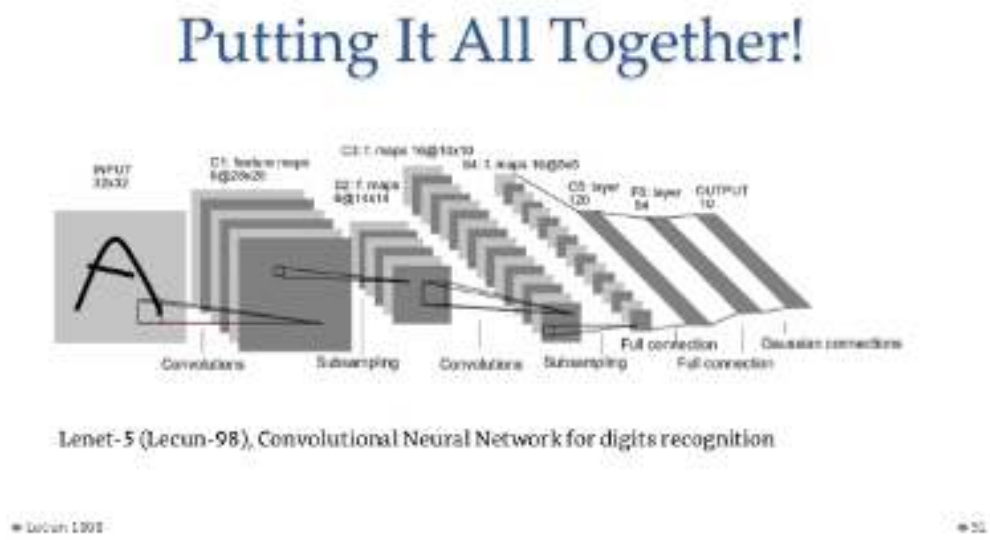


FIGURE: Architecture of the network Le Net. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998)

Architectures

- With the appearance of GPU (Graphical Processor Unit) cards, much more complex architectures for CNN have been proposed, like the network **AlexNet** (Krizhevsky (2012)).
- **AlexNet** won the ImageNet competition devoted to the classification of one million of color images (224×224) onto 1000 classes.
- AlexNet is composed of 5 convolution layers, 3 max-pooling 2×2 layers and fully connected layers.

Architectures

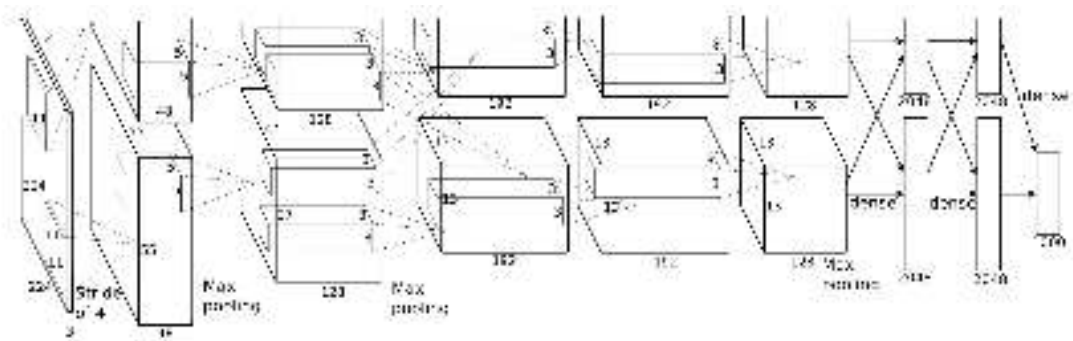


FIGURE: Architecture of the network AlexNet. Krizhevsky, A. et al (2012)

Architectures

Alexnet architecture

Input	227 * 227 * 3			
Conv 1	55*55*96	96	11 *11	filters at stride 4, pad 0
Max Pool 1	27*27*96		3 *3	filters at stride 2
Conv 2	27*27*256	256	5*5	filters at stride 1, pad 2
Max Pool 2	13*13*256		3 *3	filters at stride 2
Conv 3	13*13*384	384	3*3	filters at stride 1, pad 1
Conv 4	13*13*384	384	3*3	filters at stride 1, pad 1
Conv 5	13*13*256	256	3*3	filters at stride 1, pad 1
Max Pool 3	6*6*256		3 *3	filters at stride 2
FC1	4096	4096	neurons	
FC2	4096	4096	neurons	
FC3	1000	1000	neurons	(softmax logits)

Architectures

We present another example.

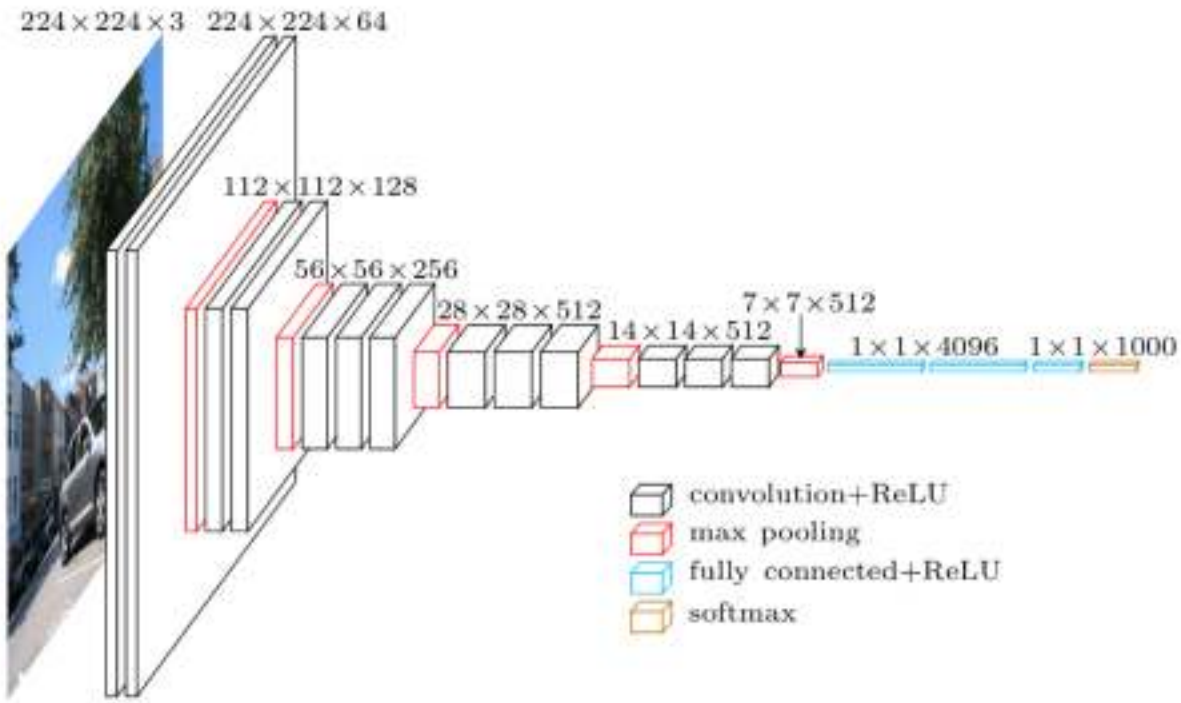


FIGURE: Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition (2014).

Architectures

The network that won the competition in 2014 is the network **GoogLeNet**(Szegedy et al. (2016)), which is a new kind of CNN, not only composed on successive convolution and pooling layers, but also on new modules called *Inception*, which are some kind of network in the network.

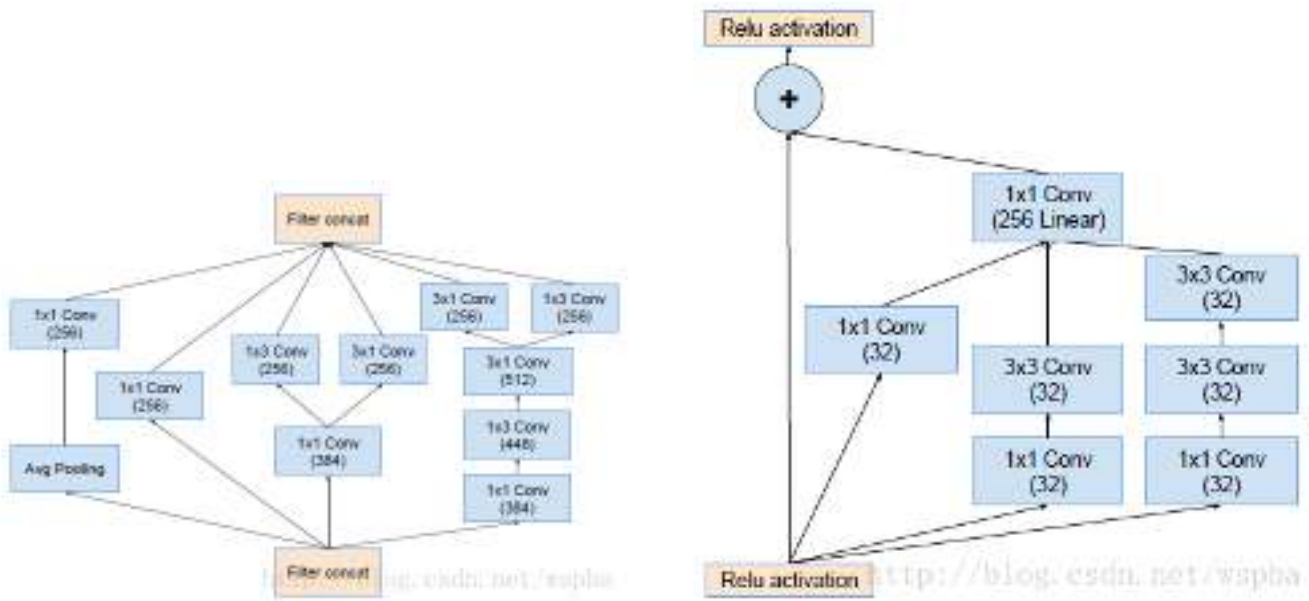


FIGURE: Inception modules, Szegedy et al. (2016)

Architectures

- The most recent innovations concern the **ResNet** networks (see Szegedy 2016).
- The originality of the ResNets is to add a connection linking the input of a layer (or a set of layers) with its output.
- In order to reduce the number of parameters, the ResNets do not have fully connected layers.
- GoogleNet and ResNet are much deeper than the previous CNN, but contain much less parameters.
- They are nevertheless much costly in memory than more classical CNN such as VGG or AlexNet.

Architectures

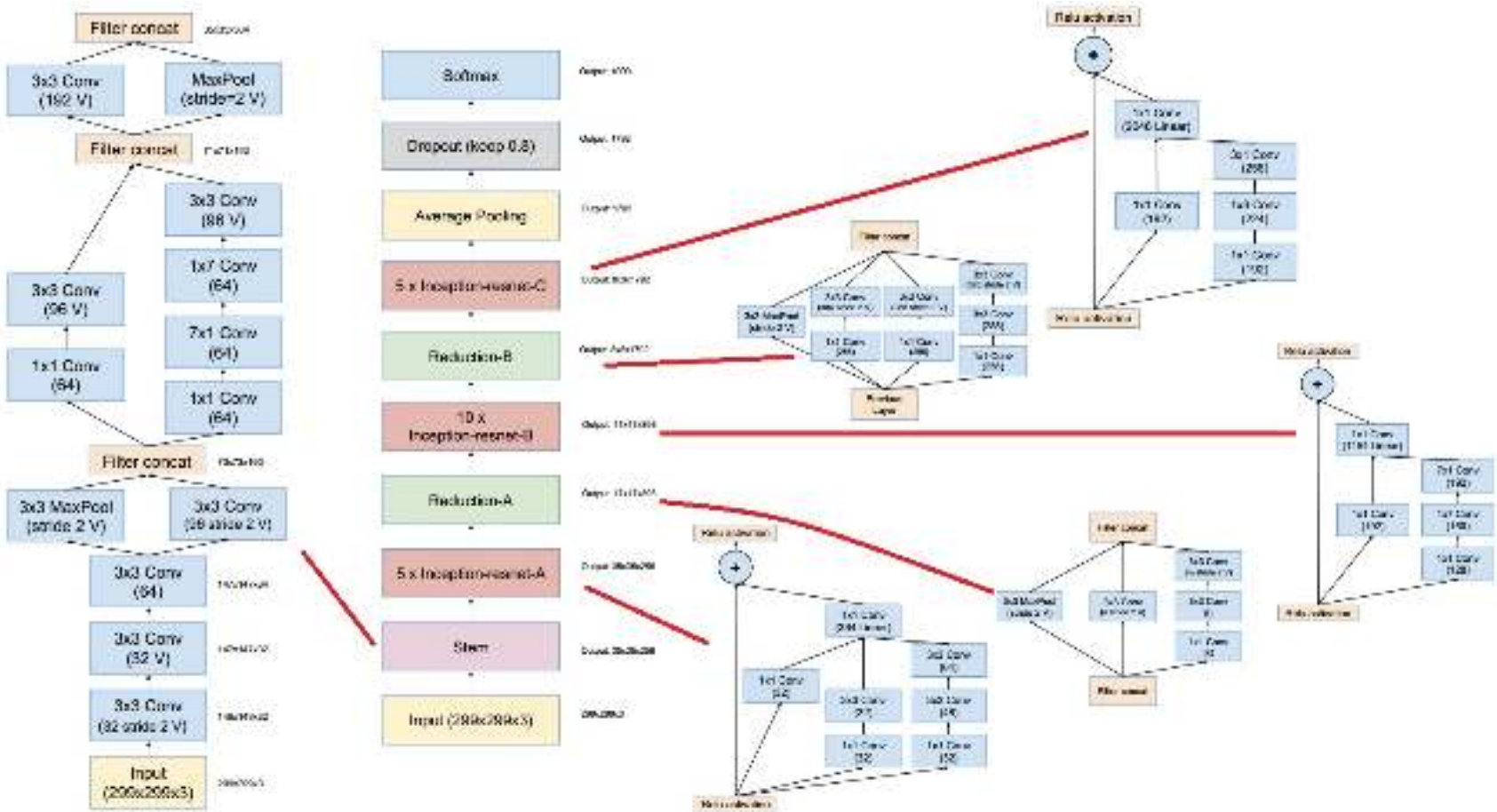


FIGURE: Inception-v4, Inception-resnet (Szegedy, C. et al. , 2016)

Architectures

The Figure shows a comparison of the depth and of the performances of the different networks, on the ImageNet challenge.

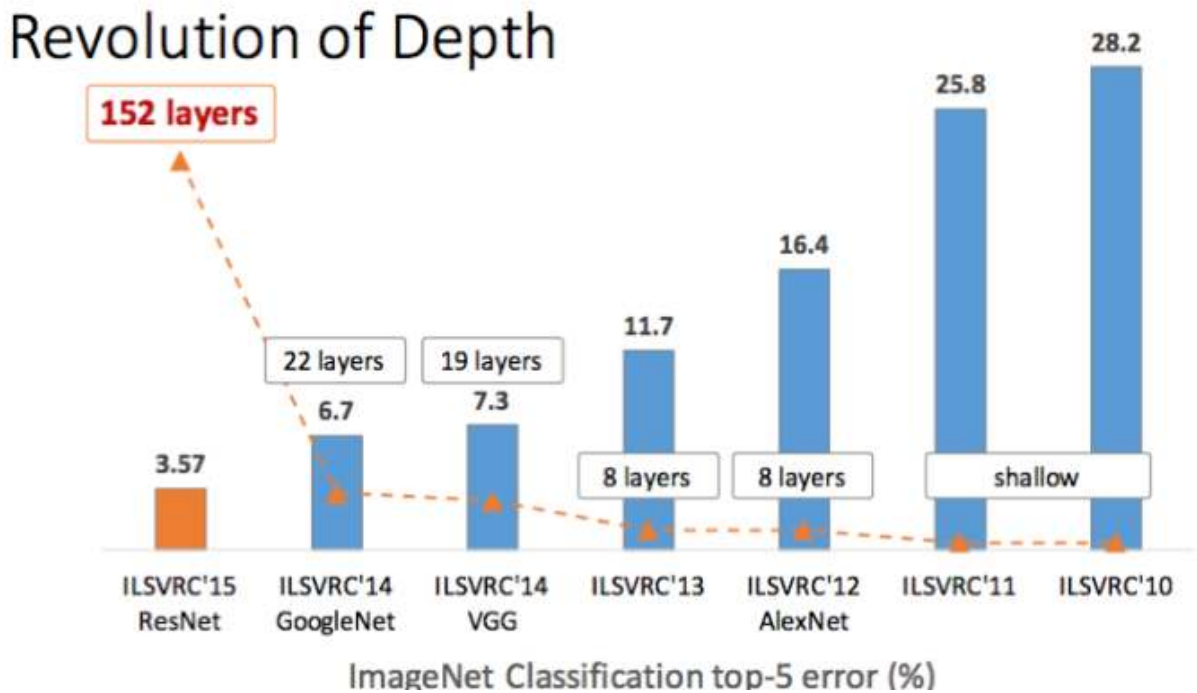


FIGURE: Evolution of the deepth of the CNN and the test error

References

- L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen (1984). *Classification and regression trees*. Chapman et Hall. CRC Press, Boca Raton.
- Giraud C. (2015) *Introduction to High-Dimensional Statistics* Vol. 139 of Monographs on Statistics and Applied Probability. CRC Press, Boca Raton, FL.
- Hastie, T. and Tibshirani, R. and Friedman, J, (2009), *The elements of statistical learning : data mining, inference, and prediction*, Springer.
- R.E. Shapire and Y. Freund *Boosting : Fundation and Algorithms* MIT Press, 2012

References

The main references for the course on Neural networks are :

- Ian Goodfellow, Yoshua Bengio and Aaron Courville *Deep learning*
- Bishop (1995) *Neural networks for pattern recognition*, Oxford University Press.
- Hugo Larochelle (Sherbrooke) :
  http ://www.dmi.usherb.ca/ larocheh/
- Charles Ollion et Olivier Grisel *Deep learning course*
  https ://github.com/m2dsupsdlclass/lectures-labs

Qualités du classifieur par réseau de neurones

Interprétabilité : NON !

Critique : NON

Consistance : NON (mais bonne approximation)

Minimax : NON

Parameter-free : NON

Vitesse : NON

Online : OUI