Projet Science des données - Défi IA INSA Toulouse

Bryan JACOB - Pierre GARDELLE - Irvin KERAUDREN - Omar AITICHOU

January 7, 2019

1 Introduction

Dans le cadre de notre projet, nous avons eu l'occasion de travailler sur des données fournies par Airbus. Ces données contenaient 85000 images. L'objectif de notre projet était de reconnaître la présence ou non d'une éolienne sur une image.

Ces 85000 images étaient divisées en deux:

- un échantillon d'apprentissage de 68000 images
- un échantillon de test composé de 17056 images

L'échantillon d'apprentissage était divisé en deux, une partie des images étant contenu dans un dossier "target" et ne contenant que des images d'eoliennes et un dossier "other" ne contenant que des images sans éoliennes dessus.

L'idée est donc d'entraîner un algorithme à partir de l'échantillon d'apprentissage pour pouvoir ensuite prédire ou non la présence d'éoliennes sur l'échantillon de test.

Pour cela, nous avons décidé d'utiliser des méthode de machine learning, toutes basées sur les réseaux de neurones.

Dans un premier temps, nous allons expliquer en détail les bases théoriques d'un réseau de neurones, puis nous évoquerons les étapes de la réflexion que nous avons mené en décrivant les différents réseaux utilisés.

2 Introduction aux réseaux de neurones

2.1 Description d'un neurone

D'un point de vu scientifique, un neurone est une cellule principale d'un système nerveux, dont le rôle est de traiter les informations dans l'organisme.

Dans le cadre du machine learning, les réseaux de neurones artificiels se sont basés sur cette structure afin d'effectuer des tâches, notamment de classification supervisée ou non-supervisée. Ils sont organisés en un ensemble de neurones, virtuels, qui sont disposés en un réseau lui aussi virtuel. Chaque neurone reçoit des informations, et émet, après traitement spécifique au neurone, de l'information.

Chaque neurone à une valeur calculée de la manière suivante:

$$v = (\sum_{i=1}^{n} w_i x_i) + b \tag{1}$$

où b est le biais, x est le vecteur des données d'entrées de taille n, x_i la valeur à la i-ème position de x, w_i le poids associé à la i-ème valeur de x.

Une fois qu'on a le résultat de cette combinaison linéaire, on va appliquer une fonction d'activation σ à ce résultat, la sortie du neurone est alors:

$$v_{sortie} = \sigma((\sum_{i=1}^{n} w_i x_i) + b) \tag{2}$$

Dans certains cas de figure, le neurone peut renvoyer un 0, on dit qu'il reste silencieux (par exemple, si une caractéristique n'est pas présente, il n'émettra pas l'information correspondante à la présence de cette caractéristique).

Un neurone peut-être représenté par:

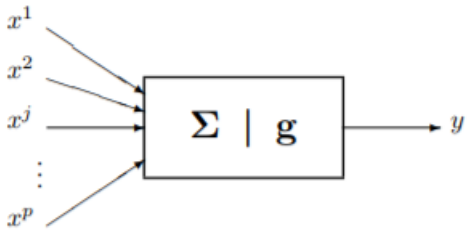


fig. 1: Représentation d'un neurone formel.
(https://www.math.univ-toulouse.fr/-besse/Wikistat/pdf/st-m-app-rn.pdf)

où g est la fonction d'activation du neurone.

Les fonctions d'activations possibles sont:

- ReLU

$$f(x) = max(0, x) \tag{3}$$

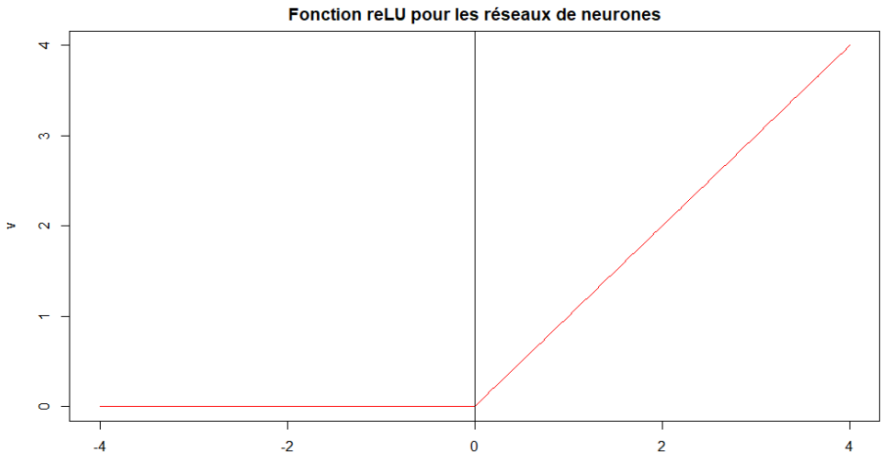


fig. 2: Modélisation de la fonction d'activation ReLU.

Par conséquent, si la combinaison linéaire est négative, alors le neurone reste silencieux.

- Sigmoïd

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

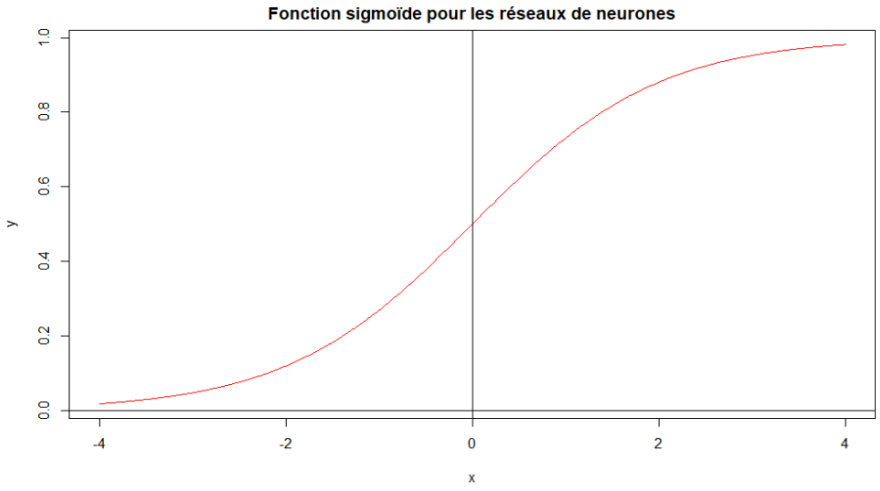


fig. 3: Modélisation de la fonction d'activation Sigmoïd.

Ainsi, on a selon les valeurs:

- La combinaison linéaire est fortement négative, son activation est quasiment nulle.
- La combinaison linéaire est proche de 0, alors l'activation sera aux alentours de 50
- La combinaison linéaire est fortement positive, alors elle sera proche de 100

- Linéaire

$$f(x) = x \tag{5}$$

- ...

Du fait de cette fonction, l'activation variera fortement pour des valeurs proches de 0, moins pour des valeurs dites extrêmes.

Une fois que la fonction d'activation a été appliquée, on a une information, qui sera à son tour renvoyée au prochain neurone qui s'en servira (en l'occurrence à la couche suivante).

On a vu jusqu'ici comment un neurone réceptionnait l'information et comment il la renvoyait.

2.2 Réseaux de neurones artificiels

Mais quelle est l'utilité d'un réseau de neurones? Les utilités sont multiples, mais dans notre cas de figure, la sortie du réseau sera un seul neurone qui sera activé ou non. On utilisera une fonction sigmoïde en sortie.

Si le neurone reste silencieux (c'est à dire si la valeur de sortie est inférieure à 0.5), on considérera qu'il n'y a pas d'éolienne sur l'image, le cas échéant (c'est à dire si le neurone est activé et donc la valeur de sortie est supérieure à 0.5) on partira du principe qu'une ou plusieurs éoliennes sont présentes.

On peut également étendre ces réseaux à des cas de classifications multiclasses ou à de la régression, les réseaux de neurones ne sont pas cantonnés à la classification binaire.

2.2.1 Perceptrons multicouches

Un perceptron multicouche est un type de réseau de neurones formels qui prend en argument un nombre d'entrées n, ce qui peut par exemple être le nombre de variables. Il est alors composé de plusieurs couches caches, chaque couche étant composée d'un certain nombre de neurones prenant en entrée les valeurs de sortie des neurones de la couche précédente. Chaque couche possède sa propre fonction d'activation. La dernière couche est la couche de sortie et doit être définie spécifiquement à notre problème. Par exemple, dans le cas d'une classification binaire, la couche de sortie doit comporter un neurone de fonction d'activation sigmoïde.

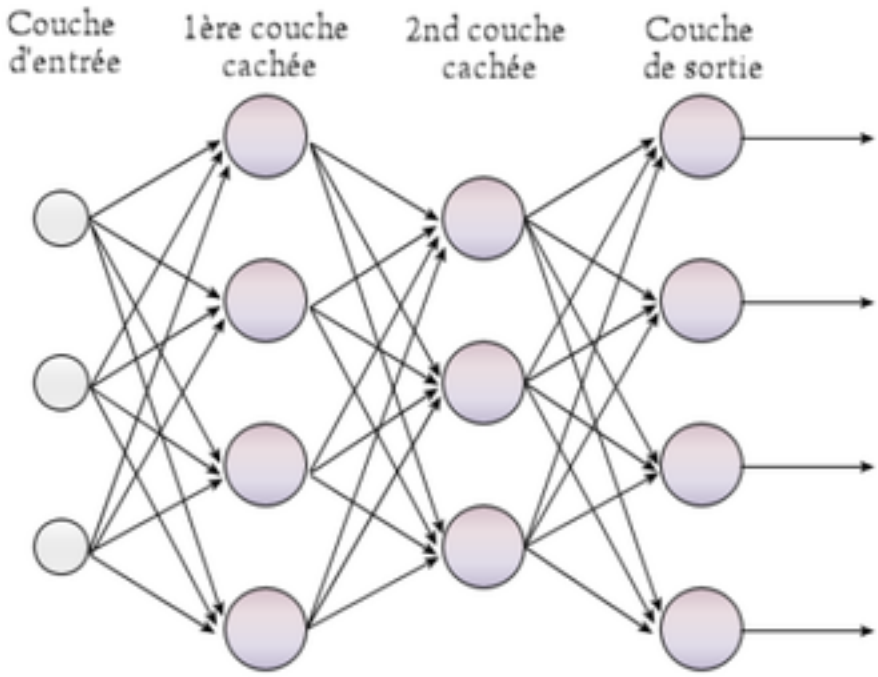


fig. 4: Perceptron multicouche à deux couches cachées.
(https://fr.wikipedia.org/wiki/Perceptron-multicouche)

Ainsi la représentation mathématique de cette application s'écrit alors:

$$f(x) = \sigma_n(w_n(...(\sigma_2(w_2(\sigma_1(w_1 x))))))) \tag{6}$$

où x est le vecteur entrée des données, w_i la matrice des poids associé à la couche i et σ_i est la fonction d'activation à la couche i.

Les paramètres à optimiser dans un perceptron multicouches sont les poids w_i ainsi que les biais b associé à chaque couche.

2.2.2 Problèmes liés au réseaux de neurones

Le principal problème du réseau de neurones est qu'il est long, très long à entraîner. On parle d'entraînement lorsqu'on cherche à trouver les poids optimaux.

En effet après initialisation des poids de manière aléatoire, on utilise un algorithme de descente de gradient pour minimiser la fonction de perte quadratique (ou la fonction d'entropie en classification).

Ce faisant on fera varier les poids jusqu'à arriver à une situation où la fonction de perte, si elle est passée en dessous d'un certain seuil est considérée comme optimisée.

Ces étapes prennent du temps et nécessitent une forte puissance de calcul, pour cela nous avons choisi de suivre les conseils de Mr Tavenard en choisissant des réseaux de neurones déjà pré-entraînés et en les adaptant à nos données.

2.2.3 Programmation d'un réseau de neurones

On peut programmer un réseau de neurones de plusieurs manières, nous avons choisi d'utiliser Python et le module Keras. Afin d'avoir plus de puissance, nous avons décidé d'utiliser Google colaboratory, un environnement de notebook jupyter qui permet d'acquérir plus de puissance pour calculer les poids et biais des modèles.

De manière générale, dans un réseau de neurones on doit choisir divers paramètres:

- Les variables d'entrée et de sortie (normalisées ou non)
- L'architecture du réseau (nombre de couches cachées, nombre de neurones par couche)
- Le nombre d'itérations, l'erreur maximale tolérée qui entreront en compte dans la descente de gradient et le terme de régularisation (pour éviter le sur-apprentissage) qui permet d'éviter que les poids des neurones prennent des valeurs trop importantes.
- Le taux d'apprentissage (ou learning rate en anglais)

2.3 Les réseaux de neurones convolutifs

Maintenant, que nous avons introduit ce qu'est un réseau de neurones, notamment les perceptrons multicouches, nous allons voir comment il est possible d'appliquer un réseau de neurones dans le cas où nos données sont des images.

Pour cela, il va être nécessaire de définir ce qu'est un réseau de neurones convolutif ou réseau de neurones CNN en anglais. Il s'agit d'un réseau de neurones qui va pouvoir traiter des données en dimensions supérieurs à 1, en opposition aux perceptrons multicouches, et donc traiter des images.

Un réseau de neurones convolutif est composé de deux parties :

- Une partie convolutive qui aura pour objectif de détecter les motifs e caractéristiques importantes d'un signal, d'une image, ...
- Une seconde partie qui aura pour objectif de traiter le problème de régression ou de classification, il s'agira en général d'un perceptron multicouche

Néanmoins, une question se pose : Comment fonctionne la partie convolutive de ce type de réseaux de neurones ? Pour répondre à cette question, nous allons nous limiter aux cadres de la classification d'images.

Il est possible de transformer une image en données. En effet, il suffit de la transformer en matrice de pixels, où chaque pixel sera transformé en une valeur. S'il s'agit d'une image en nuance de noir et de blanc, nous obtiendrons une matrice, soit une donnée en deux dimensions. S'il s'agit d'une image en couleur, nous obtiendrons une donnée en trois dimensions, un pour chaque valeur de couleur (Rouge, vert, bleu).

La partie convolutive d'un réseau de neurones convolutif va appliquer un filtre sur l'image, appelée filtre de convolution, dont la taille sera à fixer par l'utilisateur. Ce filtre de convolution va parcourir l'ensemble de l'image, ou plutôt la matrice de l'image, de la manière suivante :

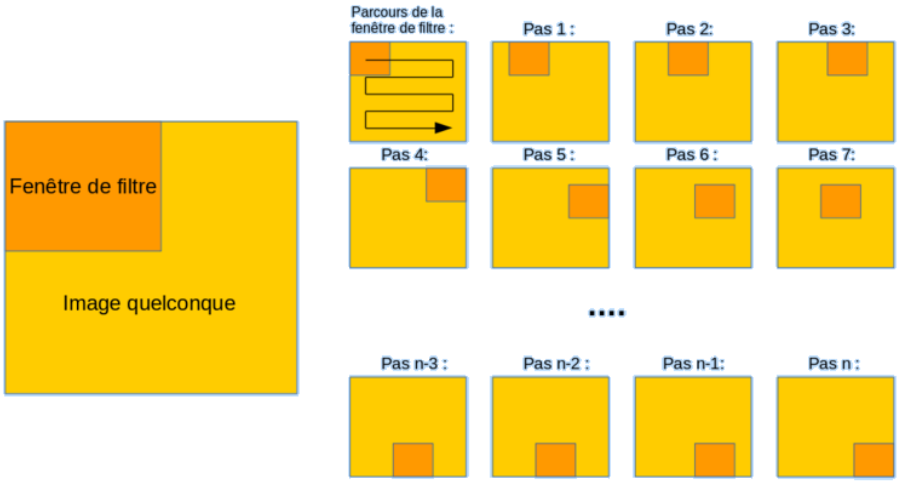


fig. 5: Étapes du balayage du filtre de convolution sur une image. (http://www.natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-neurones-convolutifs)

La taille de la fenêtre est définie par le filtre de convolution. Si nous choisissons une taille 3×3 pour une image de dimension 64×64, nous aurons donc 62×62 filtres de convolution. Ce qui est équivaut donc à un nombre de pas de 3844 sur l'image ci-dessus. Chaque filtre de convolution définira alors une matrice de la taille du filtre contenant chacune des valeurs numériques. Il faut maintenant appliquer une opération sur chacune de ces matrices afin d'en retirer de la manière suivante:

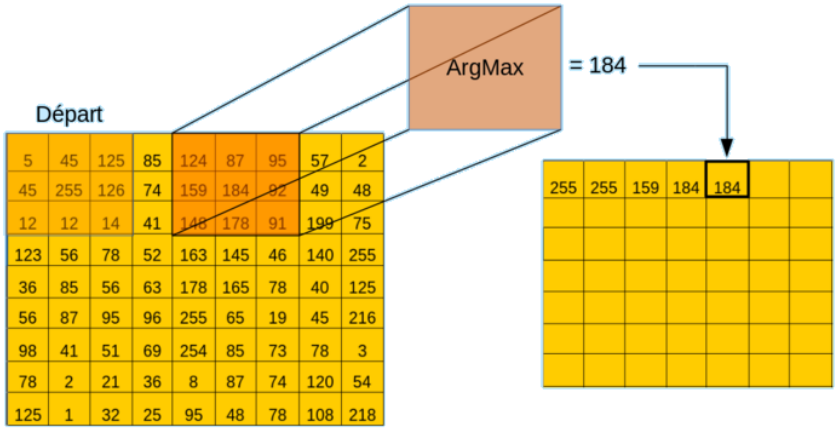


fig. 6: Opérations sur les filtres d'une couche convolutive. (http://www.natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-neurones-convolutifs)

Dans l'exemple précédent, une image de dimension 64×64 avec un filtre de taille 3×3 nous donne 62×62 matrices. Si nous appliquons une opération sur ces matrices, nous obtenons alors 3844 résultats, qui seront alors stockés dans une matrice de taille 62×62.

Les opérations les plus communes sont le maximum, qui prend la valeur maximale contenue dans la matrice, et la moyenne, qui prend la moyenne des valeurs contenues dans la matrice. Il est également possible de définir ses propres opérations si nécessaires.

Nous avons donc vu qu'une couche de convolution est composée d'un filtre et d'une opération sur ces filtres. Mais il est également possible d'enchaîner plusieurs couches de convolutions.

La partie convolutive agit comme un détecteur de formes et va donc extraire des caractéristiques importantes de l'image. Les premières couches convolutives détectent des formes simples, comme des traits ou des figures géométriques mais les couches plus lointaines vont réussir à combiner ces informations afin de détecter des formes plus complexes comme des visages humains. Cette partie convolutive permet également de ne pas prendre en compte tous les pixels comme variables et donc de limiter le nombre de poids à optimiser.

Néanmoins, la partie convolutive ne va pas nous donner une sortie sous la forme de vecteurs mais plusieurs matrices. Or un perceptron multicouche prend en argument un nombre n d'entrées, qui peut être sous la forme d'un vecteur. Il va donc falloir aplatir les matrices en vecteurs de la manière suivante:

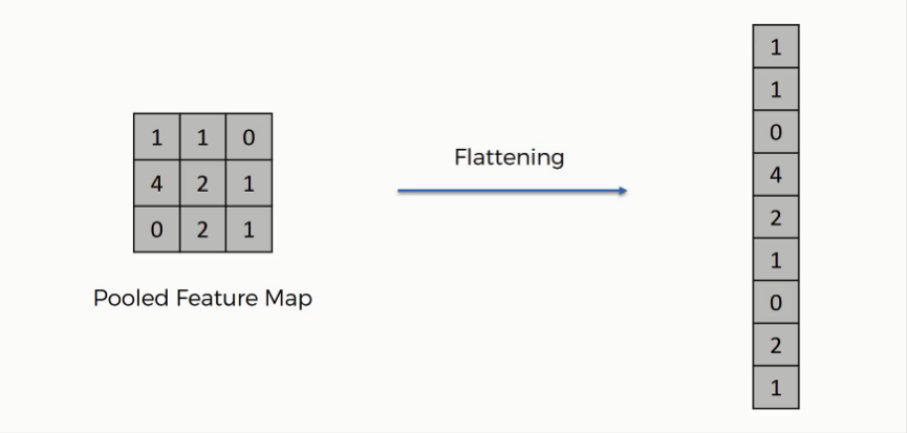


fig. 7: "Aplatissement" d'une matrice en un vecteur.
(https://www.superdatascience.com/convolutional-neural-networks-cnn-step-3-flattening/)

Nous le généralisons à toutes les matrices que nous donne notre couche convolutive:

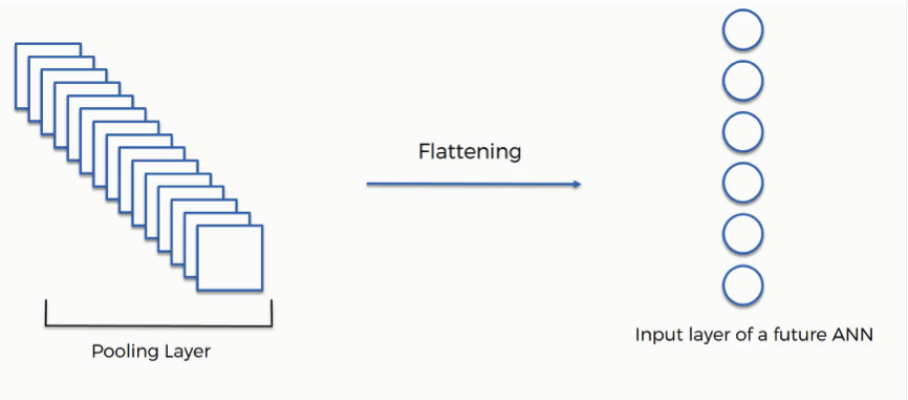


fig. 8: Passage d'une couche convolutive à un vecteur
(https://www.superdatascience.com/convolutional-neural-networks-cnn-step-3-flattening/)

Nous obtenons alors un très grand vecteur, qui peut alors être passé en argument d'un perceptron multicouche. Il suffit alors de définir correctement ce perceptron afin de répondre à notre problème.

2.4 Deep Learning

Le Deep Learning, ou l'apprentissage par des réseaux de neurones dits "profonds", c'est-à-dire avec un grand nombre de couches de neurones intermédiaires prend forme dans les années 2010, avec la convergence de trois facteurs :

- des réseaux de neurones artificiels multicouches
- des algorithmes d'analyse discriminante et apprenants (dont l'émergence remonte aux années 1980)
- des machines dont la puissance de traitement permet de traiter des données massives

Pour comprendre comment fonctionne le Deep Learning, prenons un exemple concret de reconnaissance d'images, pour reconnaître les photos qui comportent au moins une éolienne. Pour pouvoir identifier les éoliennes sur les photos, l'algorithme doit être en mesure de distinguer et de reconnaître une éolienne de manière précise quel que soit l'angle sous lequel il est photographié.

Afin d'y parvenir, le réseau de neurones doit être entraîné. Pour ce faire, il est nécessaire de compiler un ensemble d'images d'entraînement (l'échantillon d'apprentissage, ou train) pour pratiquer le Deep Learning. Cet ensemble va regrouper des milliers de photos avec la présence d'éolienne, mélangés avec des images d'objets qui n'en sont pas des pas. Ces images sont ensuite converties en données et transférées sur le réseau. Les neurones artificiels assignent ensuite un poids aux différents éléments. La couche finale de neurones va alors rassembler les différentes informations pour déduire s'il s'agit ou non d'une éolienne.

Le réseau de neurones va ensuite comparer cette réponse aux bonnes réponses indiquées ultérieurement à la main. Si les réponses correspondent, le réseau garde cette réussite en mémoire et s'en servira plus tard pour reconnaître les éoliennes. Dans le cas contraire, le réseau prend note de son erreur et ajuste le poids placé sur les différents neurones pour corriger son erreur. Le processus est répété jusqu'à ce que le réseau soit capable de reconnaître une éolienne sur une photo dans toutes les circonstances. Cette technique d'apprentissage est appelée apprentissage supervisé.

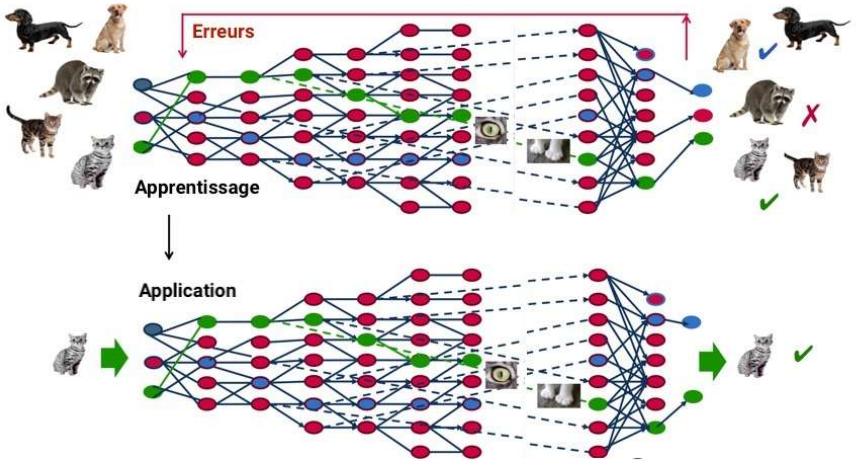


fig. 9: Modélisation d'un réseau de neurones de classification. (https://www.futura-sciences.com/tech/definitions/intelligence-artificielle-deep-learning-17262)

Plus un réseau de neurones est profonds plus ils sont performants, cependant il semble il y avoir l'existence d'une limite à la profondeur à cause de l'évanescence du gradient ; le gradient qui remonte le réseau de neurones, multiplié par les poids synaptiques et les dérivées des fonctions d'activation décroît exponentiellement avec la profondeur car lors de l'application de l'algorithme de rétropropagation pour ajuster les paramètres, l'intensité du gradient est multiplié par les poids synaptiques des synapses entre les couches.

2.5 Data augmentation

L'augmentation des données est la création de copies modifiées de chaque instance dans un jeu de données d'apprentissage.

Quand nous utilisons des données d'image dans un réseau de neurones, nous souhaitons que le réseau de neurones se résume en un ensemble de nombres ou de pondérations. Dans le cas de la classification d'image, ces caractéristiques sont les pixels qui constituent l'objet (éoliennes dans notre cas) dans l'image. D'autre part, il y a aussi des caractéristiques que nous ne voudrions pas que le réseau de neurones intègre dans son étude des images Dans le cas de la classification d'images, ces caractéristiques ou bruit sont les pixels qui forment l'arrière-plan de l'image.

Pour assurer que le réseau de neurones peut différencier le signal du bruit, on peut adopter une solution très simple qui consiste à créer de multiples modifications de chaque image ou l'objet de l'image restant invariant, tandis que le bruit ou l'arrière-plan est déformé. Ces déformations incluent notamment le rognage, la mise à l'échelle et la rotation de l'image.

Par conséquent, le réseau de neurones observe l'invariance dans les images et code cette information ou ce signal dans l'ensemble de poids qui résument les données d'apprentissage.

Quelques types d'augmentation des données :

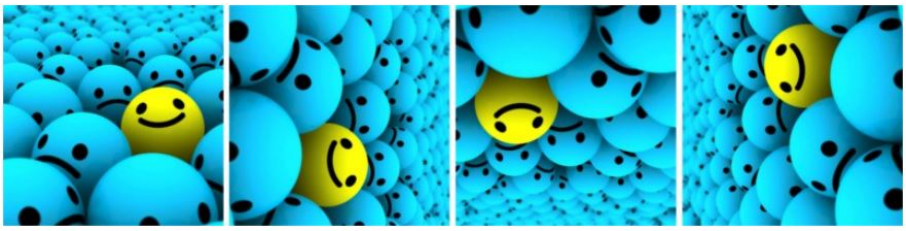
- 1. Retourner les images

  On retourner les images horizontalement et verticalement et selon différents angles.

Img 1: Illustration du pivotement des images.
(https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced)

- 2. Rotation

Si l'image est un carré, une rotation perpendiculaire préservera la taille de l'image. Si c'est un rectangle, une rotation de 180 degrés préservera la taille. La rotation de l'image par des angles plus fins modifiera également la taille finale de l'image.



Img 2: Illustration de la rotation des images. (https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced)

- 3. Échelonnage

L'image peut être redimensionnée vers l'extérieur ou vers l'intérieur. Lors de la mise à l'échelle vers l'extérieur, la taille de l'image finale sera plus grande que la taille de l'image d'origine. La plupart des cadres d'image découpent une section de la nouvelle image, avec une taille égale à l'image d'origine.

Img 3: Illustration de l'échelonnage des images. (https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced)

- 4. Surgir

Contrairement à la mise à l'échelle, nous échantillonnons de manière aléatoire une section de l'image d'origine. Nous redimensionnons ensuite cette section à la taille de l'image d'origine. Cette méthode est communément appelée recadrage aléatoire.

Img 4: Illustration du recadrage des images.
(https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced)

3 Modèles utilisés pour le défi

Cette partie aura pour objectifs de présenter les différents modèles utilisés pour classifier notre jeu d'images. Chaque modèle sera présenté de la même manière. Nous présenterons d'abord les raisons du choix du modèle, puis la structure du modèle, ces résultats et les raisons de la réussite ou de l'échec du modèle.

3.1 Modèle initial

3.2 Modèle avec VGG16 et ResNet50

3.3 Ajout d'une forêt aléatoire ou d'un Support Vector Machine sur la dernière couche

Afin d'améliorer la précision de notre modèle précédent qui utilisait un modèle pré-entraîné Resnet50, nous avons décidé d'allier un réseau de neurones avec une autre méthodes de classification comme les forêts aléatoires ou des Support Vector Machine. Pour cela, nous avons décidé d'entraîner le modèle précédent jusqu'au point de sur-apprentissage. C'est-à-dire jusqu'au point où le risque estimé sur l'échantillon, ou le risque empirique, est le plus faible.

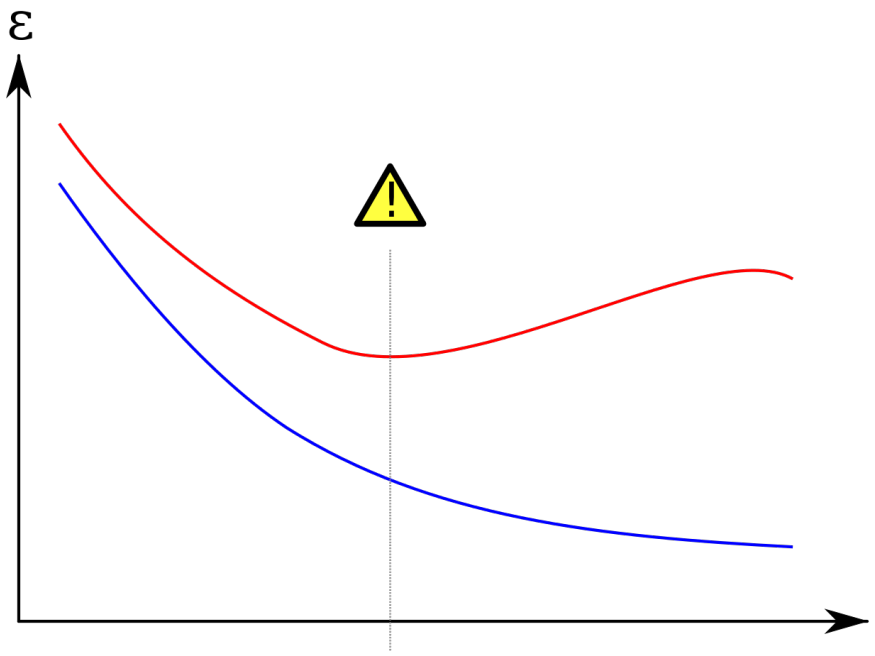


fig ?: Graphique représentant l'erreur d'entraînement (en bleu) et d'apprentissage (en rouge). (https://fr.wikipedia.org/wiki/Surapprentissage)

Une fois le réseau de neurones entraînés jusqu'à ce point, nous avons donc théoriquement le modèle optimal. Pour appliquer un autre modèle, nous allons appliquer ce réseau de neurones optimal à nos données mais récupérer les valeurs des neurones à l'avant-dernière

couche, et non à la dernière. Si l'avant-dernière couche à p neurones, nous obtiendrons pour chaque image un vecteur de taille p. Nous pouvons alors l'interpréter comme une observation de p variables. Vu que nous avons le libellé des images, il est alors possible d'appliquer un modèle d'apprentissage supervisé sur les valeurs de l'avant-dernière couche.

Dans notre modèle, la couche possède 256 neurones, donc cela nous fat 26 variables. On est donc dans un cas de grande dimension. Les Support Vector Machine étant insensible à la dimension et les forêts aléatoires sont des méthodes de bagging tirant aléatoirement des variables, la dimension ne posera pas de problèmes dans ces cas.

3.3.1 Forêt aléatoire

Par validation croisée, nous pouvons obtenir les valeurs des hyperparamètres du modèles. Nous obtenons un nombre d'arbres estimé à 1000, pour une profondeur des arbres maximale de 1 et un tirage aléatoire de 4 variables.

Le taux de précision sur l'échantillon test est de 97,51%, ce qui nous donne pas un taux sensiblement meilleur que le modèle uniquement composé du réseau de neurones.

3.3.2 SVM

Par validation croisée, nous pouvons obtenir les valeurs des hyperparamètres du modèles. Nous obtenons un Support Vector Machine à noyau gaussien, de coefficient de pénalite C = 1 et de coefficient de noyau gamma = 0.037.

Le taux de précision sur l'échantillon test est de 97,53%, ce qui nous donne pas un taux meilleur que le modèle uniquement composé du réseau de neurones.

3.3.3 Pourquoi cela ne donne pas de meilleurs résultats?

Nous n'avons pas vraiment d'explication sur le fait que nos résultats ne soit pas meilleurs. Nous supposons que puisque nous récupérons les valeurs de l'avant-dernière couche, définis à l'aide des poids et des biais, sont optimisés spécifiquement pour résoudre le problème via le réseau de neurones et non un autre modèle de classification.

3.4 Modèle final (meilleur taux de bonnes prédictions)

4 Conclusion