Machine Learning

Lauriane Moriceau, Lucas Aurouet

Decembre 2020

1 Introduction

Dans ce travail, notre but est de participer à la compétition Kaggle "Petals to the metal". L'objectif de cette compétition est la résolution d'un problème de reconnaissance d'image. Les données comprennent 12753 images de fleurs d'une résolution de 192x192 pixels. Les fleurs sont reparties en 104 classes représentant leur espèce (Roses, Lilas, Orchidées, ...). Le principe étant d'entraîner un algorithme de machine learning pour apprendre à correctement classer les image dans la catégorie qui lui correspond. Pour tester le modèle nous avons en notre possession 232 images sur lesquelles notre modèle ne sera pas entraîné. nous utiliserons des Machines à Supports vectoriels avec différentes fonctions kernel, et différents réseaux de neurones avec une profondeur et une structure particulière.

2 Gestion des données

Nous avons commencé à traiter le problème en regardant la forme des données pour avoir une idée de comment les traiter et les adapter à un SVM et un réseau de neurones. Les images étant des fichiers particuliers à traiter. Nous avons du chercher comment passer d'une image jpeg à des données utilisables pour des problèmes de classification. Dans un premier temps nous avons constaté que les données étaient stockées dans un format particulier, à savoir un "tf-record". Ce format permet de stocker des volumes de données importants en facilitant leur utilisation dans Tensorflow. Malheureusement nous comptions utiliser principalement la librairie sickit-learn qui n'est pas directement compatible avec les tf-records. Il a fallu utiliser la fonction pdtfr.tfrecords_to_pandas de pandas_tfrecords pour ouvrir les fichiers. une fois les fichiers ouverts, nous constatons que les données sont de forme (n, 3) avec n le nombre d'observations. Les 3 colonnes du dataframe sont constituées de l'identifiant de l'image, de la classe de la fleur représentée, et de l'image. cette image est exprimée en bytes et a donc besoin d'être transformée en image jpeg avec 3 channels (R, G, B). pour cela nous avons écrit la fonction stringToRGB qui permet de retourner une image ouvrable avec PIL à partir de la chaîne de bytes initialement présente.

Nous avons ainsi obtenu un dataframe qui contenait n images jpeg auxquelles est associée la classe correspondante parmi les 104.

Les SVM et ANN sont des méthodes qui reposent sur de l'optimisation numérique elles sont donc incapables de traiter des images directement. Il est nécessaire de trouver une méthode pour transformer une image jpeg en une collection de points représentant l'image par des nombres qui eux, pourront être compris par les modèles. Pour ce faire, nous avons écrit la fonction featureExtraction qui transforme les images en numpy array, array qui est ensuite "applati". Une image RGB est essentiellement un array à 3 dimensions, une pour chaque couleur (R, G, B), chaque array contient lui-même un array de taille (w, l) avec w la largeur en pixels de l'image et l la longueur. En "applatissant" ces numpy array, on assemble ces 3 arrays de (w, l) en un seul array de taille (1, 3 * (w, l)). Les 3 dimensions et le format de l'image sont passés dans une seule ligne qui contient toutes les informations. chaque colonne de l'array contient une valeur entre 0 et 255 qui représente l'intensité de rouge, bleu, ou vert, d'un pixel en particulier. La collection de toutes ces colonnes donne donc l'intensité de rouge, bleu, vert pour tous les pixels de l'image sous forme de nombres dans l'intervalle [0 : 255] Nos données sont désormais quasiment prêtes pour être employées. Une dernière étape consiste à normaliser les données, les réseaux de neurones requièrent des données normalisées et il est régulièrement recommandé, dans la littérature, de normaliser les données dans un problème de reconnaissance d'images. Nous divisons donc les arrays par 255 pour obtenir des valeurs comprises dans [0 : 1]. Nos données sont prêtes à être modélisées.

3 Support Vector Machine

Le premier modèle considéré est le Support Vector Machine. Le but est de trouver une frontière de décision dans un espace en n dimensions (une droite dans un espace en 2d, un plan en 3d et un hyperplan pour les dimensions supérieures). cette frontière est atteinte en trouvant l'espace maximal entre 2 points de groupes (classes) différents, une fois cet espace trouvé, on peut tracer deux vecteurs parallèles qui passent par ces points ("supports"). La droite parallèle qui passe entre ces deux vecteurs est la frontière de décision qui nous permet de classifier nos observations. L'avantage et ce qui fait tout l'attrait des SVM réside dans leur capacité à trouver des séparateurs linéaires dans un problème de classification qui n'est pas initialement linéairement séparable. En passant dans une dimension supérieure (en passant de R^2 à R^3 par exemple), les différentes classes peuvent être linéairement séparées. Passer dans une autre dimension requiert cependant une puissance de calcul phénoménale et les SVM utilisent une technique pour simuler ce passage en dimension supérieure sans avoir à effectivement s'y rendre. la résolution du problème se fait dans la dimension originale en émulant le passage en dimension supérieure en quelque sorte. Cette technique est souvent appelée "kernel trick", du fait qu'une fonction kernel (linéaire, polynomiale ou gaussienne) est utilisé pour "émuler" ce

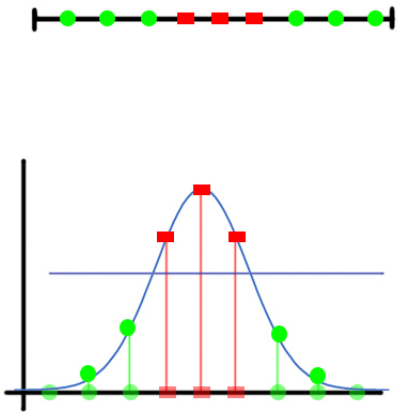

Figure 1: Effet d'un kernel gaussien sur les données

passage en dimension supérieure. Le schéma de la figure 1 montre ce passage en dimension supérieur pour un kernel gaussien. Les données initialement impossible à séparer avec une seule droite (non linéairement séparables), une fois passée par le kernel, peuvent être séparées simplement par la droite bleue.

4 Artificial Neural Networks

Les Neural Nets sont des modèles construit pour représenter le processus de pensée d'un cerveau humain. plusieurs neuronnes sont reliés entre eux et s'activent ou se désactivent selon les liens et la force des liens qui existent entre chacun d'entre eux. Chaque neurone reçoit l'information transmise par tous les neurones précédents[1], cette information est pondérée par des poids w et sommée. Si la somme de l'information transmise par les neurones précédents dépasse un certain seuil, le neurone s'active, transmettant à son tour de l'information aux neurones en aval. Le nombre de neurones, les poids w, le seuil d'activation sont tous des paramètres à définir lors de l'entraînement d'un réseau de neurones. Ici, nous testerons plusieurs nombre de couches et de neurones pour visualiser l'impact que ces hyperparamètres peuvent avoir sur la précision. Les poids

[1] Tous les neurones de la couche précédente

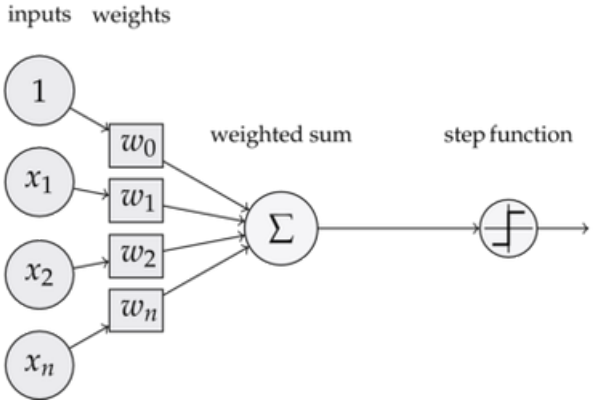

Figure 2: Perceptron

quant à eux sont mis à jour de manière itérative par descente de gradient[2]. A chaque itération, on calcule la direction du gradient de la fonction de perte et on voyage d'une unité dans la direction qui fait diminuer la fonction de perte le plus abruptement. On met alors les poids à jour, et on calcule une nouvelle erreur, on fait un pas de plus dans la direction où le gradient est maximal etc... La taille de cette unité est aussi un hyperparamètre à prendre en compte. Trop petite, la convergence peut être très lente, trop grande, l'algorithme peut "louper" le minimum global de la fonction de perte et ne jamais converger. Le schéma de la figure 3 représente un exemple simplifié avec un seul neurone appelé "perceptron".

On voit comment le neurone recoit une somme pondérée de tous les inputs, cette somme passe dans une fonction d'activation, si le seuil d'activation est franchi, la fonction σ renvoie une valeur. 0 ou 1 dans le cas binaire, ou bien un nombre entre 0 et 1 dans le cas où on a plus de deux classes. En général, si l'on a plus de deux classes, les neurones terminaux sont au nombre de classes et la valeur qu'ils renvoient correspond à la probabilité que l'image appartienne à telle ou telle classe[3]. Le schéma suivant représente le chemin emprunté par la descente de gradient dans un espace en 3 dimensions. En partant de la condition initiale, on emprunte le chemin le plus abrupt, on avance d'un pas (on met à jour les poids), on repète le processus un nombre suffisant de fois et on arrive au minimum de la fonction. Cette fonction étant une fonction de perte, la descente de gradient revient à minimiser la fonction de perte de manière itérative, quand aucune solution analytique n'existe[4].

[2]Dans notre analyse nous utilisons l'optimiseur "Adam", extension de l'agorithme de descente de gradient

[3]le schéma provient de pythonmachinelearning

[4]Le schéma provient de mathworks

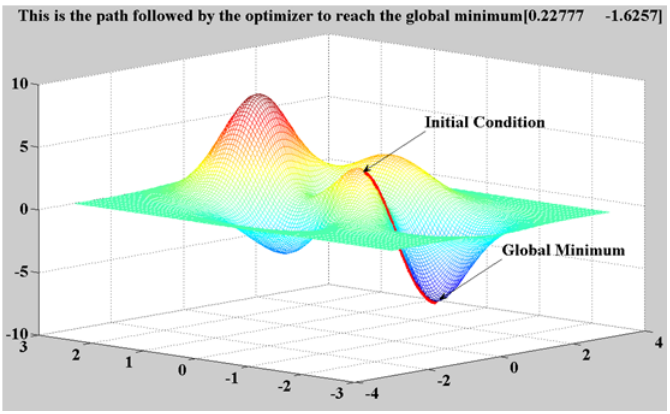

Figure 3: Descente de gradient

5 Convolutional Neural Nets

Les CNN sont des réseaux de neurones spécifiquement designés pour la reconnaissance d'images. Leur fonctionnement reste le même pour les couches standard, en revanche, on y ajoute 1 à 3 couches supplémentaires qui ont un objectif bien précis. Le première couche exclusive aux CNN est la couche "convolutive". Cette couche passe un filtre d'une taille (a, b) fixée (en nombre de pixels). Ce filtre se déplace progressivement sur l'image et passe par tous les pixels, à chaque passage le filtre transforme les données (en l'occurrence la valeur entre 0 et 1 de chaque pixel). Cette transformation permet à notre réseau de mieux capter les caractéristiques d'une image, comme les angles, les arrêtes, les formes géométriques etc... On peux ajouter une seconde couche de "pooling", cette couche transforme la valeur des pixels en la moyenne des x pixels environnants. Cela permet de capter les caractéristiques principales de l'image en évitant d'inclure un surplus d'information et donc, de limiter le risque de sur-apprentissage. La dernière couche dite de "dropout" peut être ajoutée elle aussi pour limiter le risque de sur-apprentissage. Cette couche retire aléatoirement un pourcentage des neurones dans les couches du réseau. Le nombre de couche et de neurones représentent en un sens le degré de non-linéarité du réseau; plus le réseau contient un nombre important de neurones, plus il sera capable de s'adapter à des problèmes de classification non-linéaires. Autrement dit, un réseau a d'autant plus de chance de sur-apprendre qu'il possède de neurones et/ou de couches. Pour tenter de réduire la possibilité de sur-apprendre, la couche dropout sacrifie une partie des neurones de manière aléatoire, permettant au modèle, tout comme la couche de pooling, de considérer des caractéristiques de l'image moins précises, mais plus générales.

6 Méthode

Pour classifier les types de fleurs, nous avons choisi un total de 5 modèles pour évaluer la précision de chacun et l'impact des hyperparamètres sur la qualité et le temps d'éxecution de chaque méthode. Nous utilisons un SVM linéaire, un SVM gaussien optimisé par grid search, un réseau de neurones "classique" (Multi Layer Perceptron) et deux réseaux convolutifs, un sans pooling ni dropout et un plus complexe avec ces deux couches ajoutées. La précision est évaluée avec la Sparse Categorical Cross Entropy. Nous passons en revue chaque modèle puis dans la dernière partie nous comparerons les résultats de chacun d'entre eux.

a priori, nous attendons de meilleurs résultats pour les réseaux de neurones que pour les SVM, en particulier les CNN, normalement mieux adaptés à la reconnaissance d'images. Le SVM gaussien devrait être meilleur que le SVM linéaire, la fonction kernel gaussienne étant généralement plus efficace que la fonction linéaire. Le CNN comprenant les couches de pooling et dropout devrait obtenir de meilleures prédictions grâce à l'ajout de 2 couches supposées réduire le risque de sur-apprentissage.

Lors du traitement d'un problème de classification d'images, le format des données entraîne de facto une explosion du volume d'informations à traiter. Dans la section précédente nous avons vu comment l'aplatissement d'images RGB entraîne la création d'arrays de taille (n, 3*(w, l)). Pour un ordinateur classique, traiter autant de points peut devenir très fastidieux. Le projet requérait initialement l'usage des TPU de Google, processeurs optimisés pour le machine learning. Nous avons décidé d'utiliser nos processeurs respectifs qui ne sont pas eux, optimisés. Dans ce contexte, les algorithmes de machine learning peuvent être soumis à des temps d'exécution très longs. Nous avons donc du réduire la dimensionalité du problème pour nous permettre d'exécuter notre programme dans un temps raisonable. Pour ce faire, nous avons réduit notre échantillon aux classes possédant 500 observations. Nous avons également réduit la taille des images, initialement avec une résolution 192x192, nous avons réduit les images à une résolution de 24x24. Une fois le programme complété, nous pourrons estimer les modèles sur l'échantillon complet[5].

- **SVM linéaire**: ce modèle est le plus simple d'entre tous, il n'y a aucun hyperparamètres à estimer. Le modèle se contente de trouver la frontière de décision entre chaque classes.

- **SVM gaussien**: l'utilisation d'un kernel gaussien requiert l'optimisation du paramètre C qui mesure la flexibilité du modèle vis à vis des observations mal classifiées. Un C élevé pénalise fortement les observations mal classées tandis qu'un C faible applique une pénalité négligeable. Ce paramètre qualifie la tolérance du modèle. Un second paramètre est à optimiser, gamma, qui représente la dispersion des fonctions gaussiennes. Un gamma faible signifie que les fonctions gaussiennes sont très larges

[5] les images resteront réduites à une résolution de 24 par 24 pixels pour nous permettre de lancer le programme

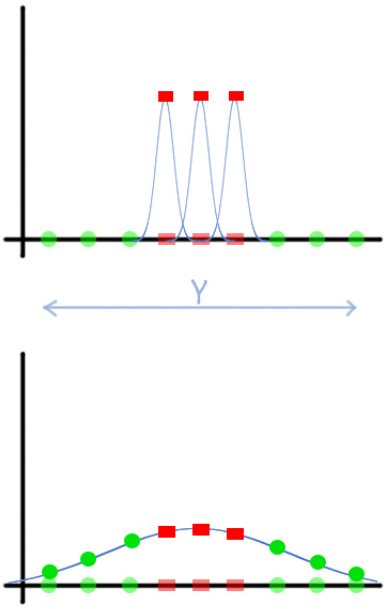

Figure 4: Effet de gamma sur la forme du kernel gaussien

et aplaties, tous les points alentours seront donc aussi transformés[6]. A l'inverse, un gamma élevé donne des fonctions très fines et chaque point est individuellement transformé. il peut paraître idéal d'avoir une dispersion très faible pour pouvoir transformé chaque point individuellement. Cependant, le risque de sur-apprentissage est très élevé si les fonctions kernel sont trop faibles. Pour optimiser les paramètres gamma et C nous utilisons une gridsearch par cross validation. Chaque couple de paramètres est évalué, l'erreur en cross validation est calculée et on retient les valeurs des paramètres qui donnent l'erreur la plus faible.

- **ANN standard (MLP)**: Ce réseau contient une couche cachée de 20 neurones et une couche de sortie de la même taille que le nombre de classes (4).

- **CNN simple**: notre premier réseau convolutif est relativement simple, il contient 2 couches cachées, une convolutive qui applique 32 filtres de 3 par 3 pixels, une couche dense (standard) de 50 neurones, et finalement une couche terminale avec 4 neurones. Notre modèle va donc appliquer une transformation de nos données avec une grille de 3x3 pixels qui se déplace sur tous les pixels de l'image pour analyser les caractéristiques principales (angles, arrêtes, formes, ...). L'image filtrée passe ensuite dans une couche

[6] Voir sur les graphiques

Modèle	Accuracy
$SVM_{lineaire}$	52%
$SVM_{gaussien}$	62%
MLP	59.5%
$CNN_{3couches}$	62.5%
$CNN_{7couches}$	67.9%

Table 1: Précisons sur un échantillon réduit

de 50 neurones classiques où les pixels de l'image sont pondérés et sommés, le neurone s'active ou non, transmettant une information aux 4 neurones finaux qui donnent chacun la probabilité que l'image appartienne à la classe qu'il représente.

- **CNN optimisé**: le second modèle convolutif contient d'avantage de couches, notamment une couche de pooling et une 2 couches de dropout. La couche de pooling réduit encore la "dimension" de l'image en prenant la valeur maximale de 4 pixels adjacents. Cette valeur est conservée et les autres sont abandonnées, la couche de pooling construit progressivement une grille contenant la valeur maximale sur une grille de 2x2 pixels qui coulisse sur l'image à l'instar de la couche convolutive. Les eux couches de dropout quant à elle déterminent le nombre de neurones qui seront pris en compte, ignorant les autres. La première conserve 75% des neurones et la seconde 50%

A noter que Conv2D de keras est d'avantage adaptée aux images en nuances de gris, 2-dimensionnelles. Cependant après avoir testé d'entraîner des modèles sur des images grayscale, il s'est avéré que la classification était trop mauvaise, et conserver les valeurs RGB semblait plus pertinent.

Le tableau ci-dessous montre les erreurs de prévisions obtenues avec chaque modèle estimés sur les 4 classes possédant plus de 500 observations.

Nous constatons que les précisions se situent entre 50% et 70% ce qui reste relativement faible pour des modèles aussi complexes. Comme attendu, les réseaux de neurones convolutifs sont plus performants de part leur structure particulièrement adaptée au problème. Le SVM gaussien est meilleur que le Svm linéaire ce qui démontre la supériorité d'un kernel gaussien vis à vis d'un kernel linéaire. Le SVM gaussien est d'ailleurs meilleur que le MLP avec 65% de précision contre 59.5% pour le MLP. A première vue ce résultat peut paraitre surprenant, les réseaux de neurones étant généralement plus complexes on pouvait s'attendre à voir les SVM systématiquement en dessous en terme de précision. Néanmoins il faut se rappeler que le SVM gaussien est optimisé par gridsearch, lui permettant de choisir les hyperparamètres optimaux tandis que le MLP est très basique avec seulement une couche cachée de 20 neurones. Avant de passer à l'estimation sur l'échantillon complet nous pouvons visualiser graphiquement le comportement des modèles vis à vis des valeurs des hyper-paramètres. On peut par exemple représenter l'évolution de la précision en

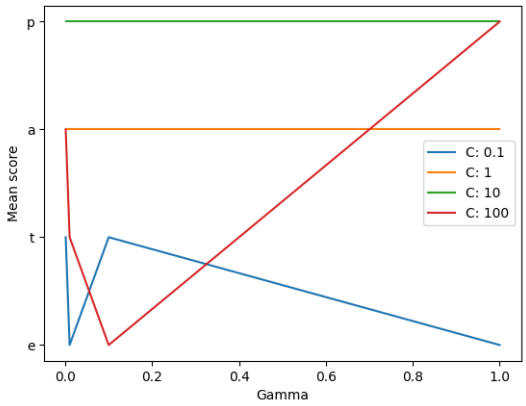

Figure 5: Effet de gamma et C sur la précision du SVM

fonction de gamma et C pour un SVM gaussien. On constate que pour C = 1 ou C = 10, le score n'évolue pas avec les valeurs de γ. Pour C = 100 la précision diminue pour des valeurs de γ en 0 et 0.1, pour un γ supérieur à 0.1, la précision s'améliore linéairement avec γ. Pour C = 0.1, la précision diminue avec γ.

Nous pouvons également regarder comment la précision du $CNN_7 couches$ évolue avec le nombre de filtres convolutifs appliqués. Nous faisons varier ce nombre de 2 à 64 avec un pas de 2 (2, 4, 16, 32, 64) et ce sur les deux couches convolutives existantes. La taille des filtres reste 3x3 comme le suggère la littérature.

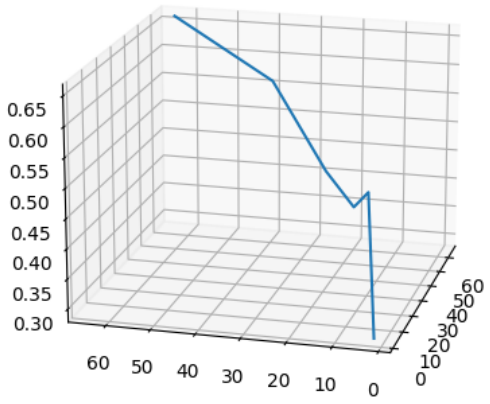

Figure 6: Effet de la taille des filtres sur la précision du CNN