Projet 3A

Classification d'images de déchets par machine learning

Paul Lebaigue

6 février 2019

Table des matières

Introduction

Nous vivons aujourd'hui entourés de technologies et d'objets visant à simplifier notre quotidien, à tel point que l'on peut entendre parfois que l'homme des pays développés n'a jamais atteint un niveau de confort équivalent par le passé. Cependant, si notre quotidien ne cesse de s'améliorer, notre consommation de ressources et notre production de déchets croissent également de manière exponentielle. La terre ne disposant pas d'une quantité de ressources infinies, il semble naturel de penser qu'un des enjeux majeurs des prochaines années sera la revalorisation et le recyclage de nos déchets.

En amont de la revalorisation de ces déchets se trouve une étape clef du processus : le tri. En effet, chaque matériau ayant des propriétés qui lui sont propres, il est indispensable de séparer les déchets en différentes catégories, qui pourront être recyclées par groupes. Ce projet s'inscrit dans cette démarche d'économie circulaire, et vise à contribuer à ce tri des déchets par de l'analyse d'images en utilisant du machine learning.

Dans une première approche, je me suis intéressé à un article scientifique traitant du même sujet. J'ai donc commencé par lire et décortiquer cet article. Les deux auteurs de ce document ont créé une base d'images eux-mêmes, j'ai utilisé cette même base d'images pour reproduire les résultats de cet article.

Deux approches indépendantes sont comparées, la première consistant en une classification par SVM à l'aide de descripteurs extraits des images, et la seconde en l'entraînement d'un réseau convolutif (ici AlexNet) prenant donc en entrées directement les images.

1 Présentation de la base d'images

1.1 Exemples

Cette base contient entre 400 et 500 images pour les catégories papier/carton/plastique/verre et 150 pour la catégorie “autres déchets”. Les images sont stockées sous la forme de dossiers portant le nom de la catégorie, ce qui en facilitera l'utilisation pour la suite. Cette petite base d'images artisanales comporte en tout 2526 images.

1.1.1 Les cartons

402 éléments dans le dossier d'images.

Figure 1 – 3 images extraites du fichier "cardboard"

1.1.2 Les verres

501 éléments dans le dossier d'images.

Figure 2 – 3 images extraites du fichier "glass"

1.1.3 Les métaux

410 éléments dans le dossier d'images.

Figure 3 – 3 images extraites du fichier "metal"

1.1.4 Les papiers

594 éléments dans le dossier d'images.

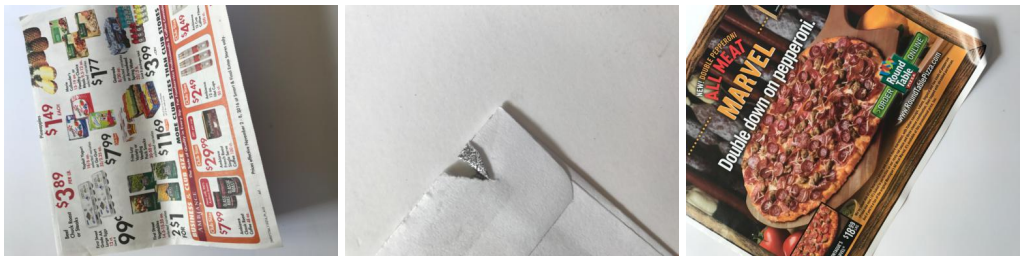


Figure 4 – 3 images extraites du fichier "paper"

1.1.5 Les plastiques

482 éléments dans le dossier d'images.

Figure 5 – 3 images extraites du fichier "plastic"

1.1.6 Les autres

137 éléments dans le dossier d'images.

Figure 6 – 3 images extraites du fichier "trash"

1.2 La Cross validation

Classiquement, on choisit de découper la base en 2 parties : une pour l'entraînement et une pour le test. Le modèle est tout d'abord entraîné sur la base d'entraînement. On évalue ensuite les performances du modèle sur la base de test, qui n'a pas été utilisée pour l'apprentissage. Cela permet de s'assurer que l'apprentissage n'est pas un apprentissage « par cœur » et que le modèle développé est capable de se généraliser à des données qu'il n'a encore jamais vu.

Cependant, séparer la base ainsi présente plusieurs inconvénients. Tout d'abord, la base de test peut être mal choisie et ne pas être assez représentative, par exemple si elle est trop petite ou si elle se révèle être un cas particulier. Par ailleurs, séparer la base en une partie entraînement et une partie test, se révèle être très peu robuste en termes de variance.

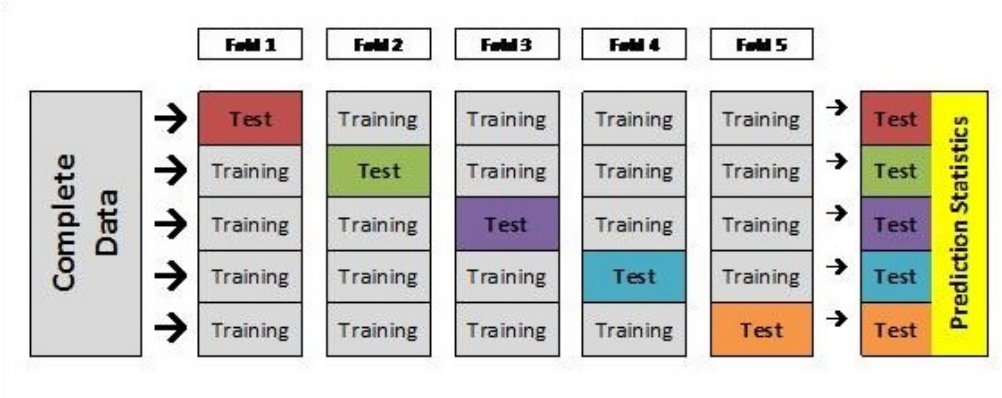


Figure 7 – Illustration du principe de cross-validation

Source : quantdev.ssri.psu.edu

Pour pallier ces difficultés on va plutôt choisir d'utiliser la k-folds cross validation (validation croisée à k morceaux). On sépare la base en k parties égales, avec classiquement k=5 ou k=10. On va alors entraîner plusieurs sous-modèles, on garde à chaque fois une des k parties pour le test et les autres parties sont utilisées pour l'entraînement. En répétant l'opération sur chacune des parties, on obtient k modèles entrainés et testés sur k bases de tests différentes.

On choisit ensuite la moyenne des performances des modèles comme une estimation de la performance du modèle final. On réduit de cette façon considérablement la variance du résultat et on exclut la possibilité d'avoir une base de test particulièrement mal choisie.

1.3 Découpage de la base d'images

Dans toute la suite nous allons évaluer des performances moyennes de modèles par validation croisée. Ainsi, dès qu'il sera question de performance d'un modèle, il s'agira en fait d'une moyenne sur les performances des différents modèles entrainés par les différents découpages de la base d'images.

1.3.1 Choix du k de la k-fold Cross Validation

J'ai choisi ici d'utiliser une 5-Fold Cross Validation pour plusieurs raisons.

Pour commencer, la base d'images étant assez petite et segmentée en 6 catégories d'images à classifier, il se pose le problème de la taille de la base de test. En effet, pour obtenir un score de validation pertinent, il faut qu'il y ait suffisamment d'éléments dans la partie test de la base. Découper la base en 5 plutôt qu'en 10 permet donc d'avoir un jeu de test 2 fois plus conséquent pour chaque modèle, ce qui est dans notre cas une nécessité (la catégorie "Trash" ne comporte en effet que 137 éléments, obtenir un score cohérent sur moins de 14 éléments semble donc déraisonnable).

Par ailleurs, le fait de ne segmenter les données qu'en 5 parties au lieu de 10 permet de n'entraîner que 5 modèles à la fois, contre 10 dans le cas contraire. La puissance et les temps de calculs ayant beaucoup joués dans l'obtention de résultats et l'avancement de ce projet, il était très intéressant de les réduire de moitié.

Segmenter en moitié moins de morceaux présente cependant des inconvénients. Tout d'abord, on ne possède que 5 simulations du modèle pour calculer le score ; ainsi la variance de l'estimateur du score du modèle s'en voit forcément impacté. Il est donc nécessaire de surveiller cette variance pour s'assurer que les résultats sont bien cohérents avec la réalité.

Ensuite, les modèles ne sont entrainés que sur 80 % des données, au lieu de 90%, ce qui peut en diminuer les performances. Il est donc bien évident qu'avec plus de données et plus de temps ou de puissance de calcul, il aurait été préférable d'utiliser une 10-Fold Cross Validation au lieu d'une 5-Fold.

1.3.2 Sélection des 5 sous-parties

L'idéal serait de répartir les images de façon équilibrée en fonction de leurs caractéristiques dans chacun des morceaux. Mais cette question est vraiment subjective et on risquerait d'influencer l'entraînement du modèle de façon indésirable (meilleur que ce qu'il est pour de vrai, ou moins performant).

Pour pallier ce problème, il a été choisi de répartir les images de façon aléatoire dans les parties. La répartition est effectuée de la façon suivante : on découpe aléatoirement chaque classe en 5 sous-ensembles de taille égale, chaque morceau de la base étant composé d'un sous-ensemble distinct de chaque classe.

2 Le modèle SVM

2.1 Principe d'une SVM

2.1.1 Idée générale

SVM signifie "Support Vector Machine". Cette méthode de classification repose sur une idée simple : essayer de séparer les données linéairement. Si les données sont décrites sur n variables, on cherche donc un hyperplan séparateur de dimension n-1. Malheureusement, les données ne sont pas toujours linéairement séparables en dimension n. Pour rendre possible ou améliorer cette séparation, les modèles SVM ont recourt à ce qui est appelé l'astuce du noyau. Le principe est de prendre les données en dimension n et les projeter dans un espace de dimension m, avec $m > n$, grâce à une fonction $\phi : R^n \to R^m$.

En fait, pour trouver l'hyper-plan séparateur optimum (au sens du maximum de marge), on est ramené à un problème d'optimisation classique faisant intervenir un produit scalaire de deux vecteurs de la base de données. On remplace ce produit scalaire $x_i.x_j$ par le produit scalaire $\phi(x_i).\phi(x_j)$; pour des raisons d'optimisation, on va réécrire ce produit sous la forme $K(x_i, x_j)$, K étant appelé noyau. On pourra ensuite optimiser en utilisant un noyau de K choisi, mais sans connaître la fonction ϕ, transparente dans le processus.

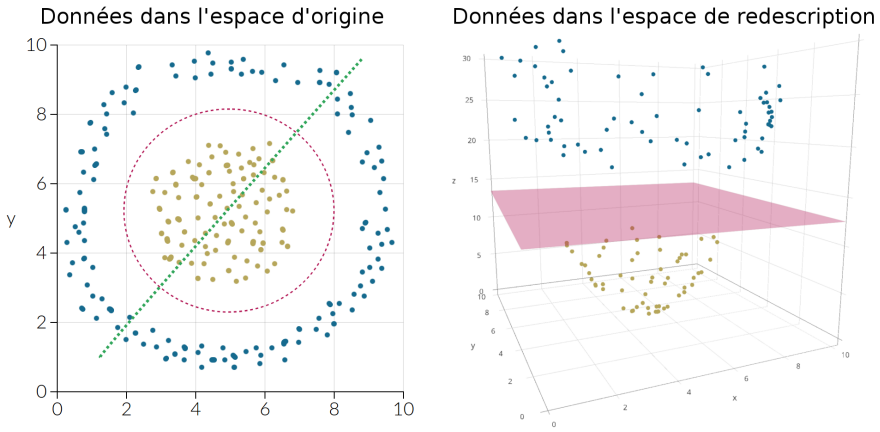


Figure 9 – Exemple d'utilisation de l'astuce du noyau

Source : Wikipedia

2.1.2 Problème de minimisation

Une fois les points décrits dans le nouvel espace, et afin de généraliser à un modèle qui tolère des erreurs, on va optimiser les paramètres de l'hyperplan séparateur de sorte à minimiser une fonction coût. Notons $\mathscr{L}$ cette fonction et β_i le paramètre associé à la variable i de l'hyperplan. On prendra également f la fonction de classification, qui à un

x donné associe sa classe (-1 ou 1 dans un problème de classification binaire). On se ramène donc à un problème de minimisation avec pénalisation de façon à accepter des erreurs mais à les minimiser. On note $\Omega(\beta)$ cette pénalisation en fonction des coefficients de l'hyperplan.

Le problème d'optimisation est alors le suivant, on cherche le vecteur β solution de :

$$\inf_{\beta} \quad C.\sum_{i} \mathscr{L}(f(x_i), y_i) + \Omega(\beta), \quad C > 0$$

Classiquement on utilise la pénalisation de Ridge, c'est à dire $\Omega(\beta) = \frac{1}{2}\|w\|^2$, et la fonction de pénalisation

$$\mathscr{L}(f(x), y) = \begin{cases} 0 & \text{si x est bien classé} \\ |f(x) - y| & \text{si x est mal classé} \end{cases}$$

qui permet de pénaliser le modèle proportionnellement aux erreurs commises lorsqu'un élément est mal classé.

2.1.3 Interprétation des hyperparamètres

Une fois le noyau K choisi, on peut le plus souvent jouer sur deux hyperparamètres.

- C est une constante propre au modèle SVM et pénalise les erreurs, choisir C trop petit donnera au modèle de faibles performances tandis que prendre un C trop grand générera du sur-apprentissage et rendra moins efficace la généralisation du modèle.
- γ est inclus dans le noyau (si le noyau possède cet hyperparamètre), cette constante va permettre de jouer sur la distance d'influence des points sur l'hyperplan séparateur. Si γ est petit, même les points les plus éloignés de la frontière ont une influence sur celle-ci, alors qu'avec un γ grand, seuls les points les plus proches ont réellement un impact sur la position de cet hyperplan. Ces points sont appelés vecteurs supports. En avoir trop aura tendance à augmenter le biais du modèle mais à en diminuer la variance, tandis que trop peu induira une grande variance du modèle, γ va donc permettre de trouver compromis acceptable entre biais et variance pour obtenir un modèle le plus efficace possible.

Le réglage de ces deux hyperparamètres est évidement crucial pour les performances du classifieur.

2.1.4 Exemples de noyaux les plus courants

- Linear kernel :

$$K(x, y) = {}^{\mathrm{t}}x.y$$

- Polynomial kernel :
$$K_d(x, y) = (\gamma\ {}^t x.y + C_0)^d$$
- Radial Basis Function (RBF) kernel :
$$K(x, y) = exp(-\gamma \left\| x - y \right\|^2)$$
- χ^2 kernel :
$$K(x, y) = exp(-\gamma \sum_i \frac{(x_i - y_i)^2}{x_i + y_i})$$
- Sigmoid kernel :
$$K(x, y) = tanh(\gamma\ {}^t x.y + C_0)$$
- Laplacian kernel :
$$K(x, y) = exp(-\gamma \left\| x - y \right\|_1)$$

2.2 Descripteurs SIFT/SURF

Afin de n'utiliser que les éléments les plus importants d'une image, il est possible d'utiliser des descripteurs d'images. Les SIFT (Scale-Invariant Feature Transform) et les SURF (Speeded Up Robust Features) seront les deux types que nous utiliserons. Ces descripteurs analysent les images localement et essaient d'en extraire des composantes indépendantes de l'échelle de la photographie.

Ils sont donc très intéressants et très robustes pour le traitement des images, notamment parce qu'ils permettent de mettre en valeur les points d'intérêt d'un objet sans trop de contraintes sur la prise de la photographie.

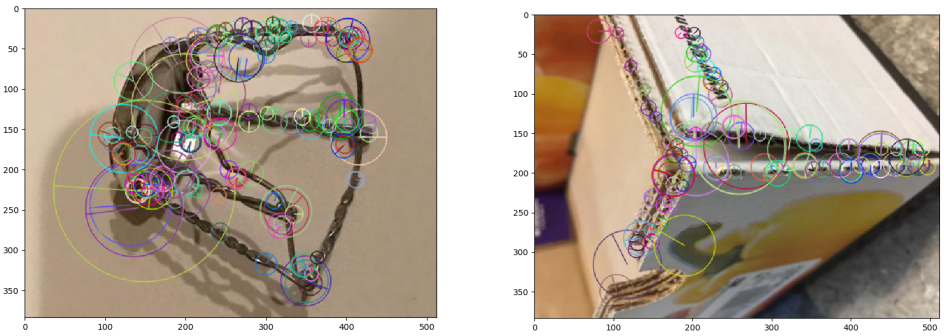


Figure 11 – Extraction des principaux descripteurs SURF sur 2 exemples

Cependant ces descripteurs fonctionnent par analyse de pics d'intensités, et ne peuvent ainsi être utilisés que sur un seul canal de couleur. Pour les utiliser il faut donc par exemple passer l'image en niveaux de gris et ensuite extraire les descripteurs de cette image. Malgré sa robustesse, cette méthode possède un gros inconvénient qui est qu'elle ne permet pas de conserver l'information contenue dans les 3 canaux de couleurs des images.

2.3 Principe de classification par « bags of words »

Pour pouvoir créer un modèle de classification sur des images à l'aide d'une SVM, il a fallu en réduire la dimension. Ainsi à partir des images on a commencé par extraire des descripteurs, chaque image possédant son « sac » de descripteurs. Une fois que toutes les images de la base d'apprentissage possèdent ce sac de descripteurs, on va regrouper ces descripteurs par groupes de proximité. Pour ce faire, on va utiliser de la classification par apprentissage non supervisé. On fixe un nombre de groupes au départ et on rassemble les descripteurs par groupes de façon à minimiser l'inertie dans les groupes. C'est l'algorithme K-Means, K représentant le nombre de groupes dans ce cas.

Une fois ce clustering sur les descripteurs effectué, on associe chaque descripteur à son groupe et on le considère ensuite en tant que tel. Chaque image se retrouve ainsi être décrite par la liste de fréquences d'apparitions de chaque classe dans ces descripteurs. Ces répartitions de fréquences d'apparition des groupes de descripteurs seront ensuite utilisées comme paramètres d'entrée du modèle SVM.

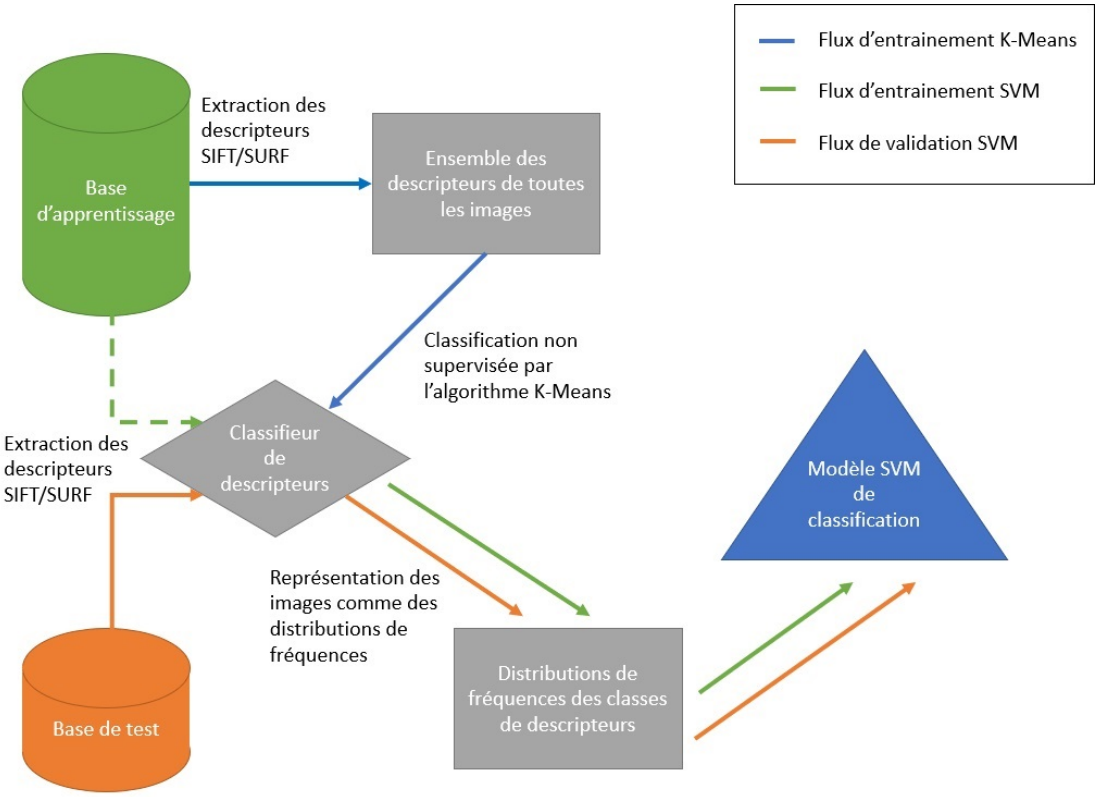


Figure 12 – Illustration de la classification d'images par "bags of words"

Le fait d'utiliser les fréquences d'apparition plutôt que le nombre d'occurrences d'une classe de descripteurs dans une image permet non seulement la normalisation du vecteur d'entrée, mais également de pouvoir considérer les entrées comme des répartitions de probabilités.

2.4 Réglages et résultats

En suivant cette procédure pour trouver un modèle, on se rend compte que les moyens d'influencer les performances du modèle sont :

- Le nombre de mots du cluster des descripteurs
- La valeur de γ lorsque le noyau l'inclus
- La valeur de C
- Le choix de la fonction noyau

2.4.1 Elbowplot

Une technique classique pour choisir la valeur du nombre de mots dans le cluster de descripteurs est de tracé un « Elbowplot ». C'est une courbe représentant l'inertie du clus-

ter en fonction du nombre de mots dans ce cluster. L'inertie du cluster est la somme des distances intragroupe, c'est-à-dire la somme des distances entre les points d'un groupe et la moyenne de ce groupe. Si le cluster est optimisé correctement, cette courbe est évidement décroissante puisque plus il y a de groupes et plus les distances à l'intérieur des groupes sont faibles.

Dans certain cas, la courbe de cette inertie en fonction du nombre de mots change de pente brusquement à un certain nombre de mots, et forme un coude. On estime que ce coude est le nombre de mots « optimal », au sens où c'est un bon compromis entre nombre de mots et inertie.

L'idée principale est que ce coude marque une séparation entre deux décroissances provenant de deux phénomènes différents. La première pente est plus forte et provient de l'algorithme, lorsqu'un groupe est identifié correctement, l'inertie du groupe est alors minimale et l'inertie totale a décrue de manière importante. La deuxième partie correspond à une décroissance plus faible, provenant du fait que l'ajout d'un groupe fait nécessairement décroître l'inertie totale (à condition que l'optimisation ait été faite correctement). Ainsi, cette inertie va décroître jusqu'à atteindre 0 : le moment où il y a autant de groupes que d'éléments, c'est-à-dire un seul élément par groupe.

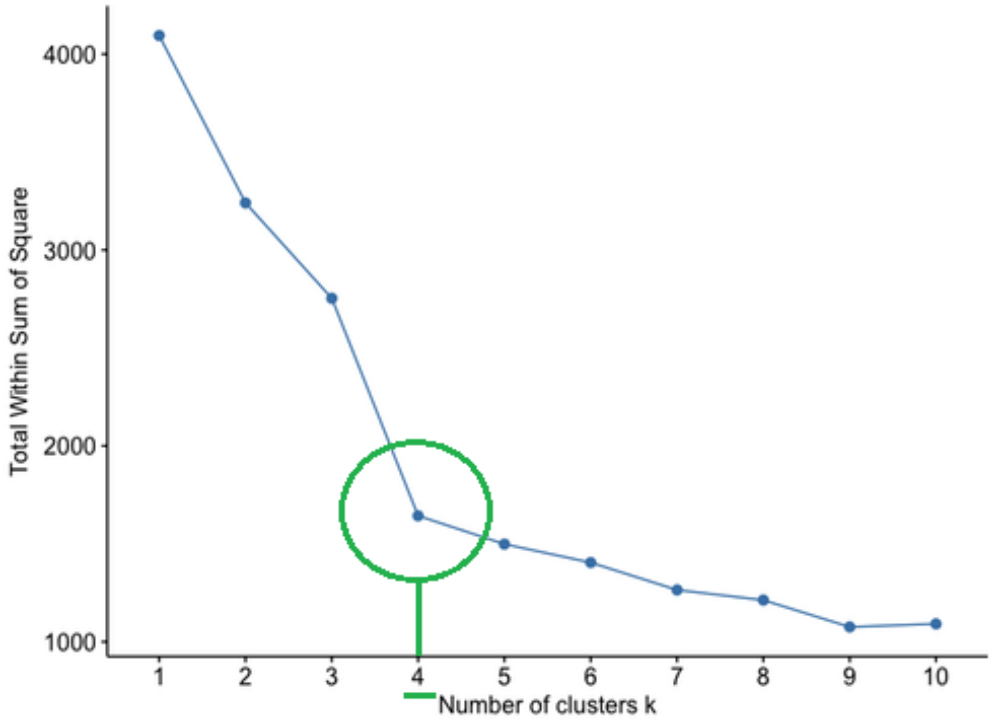
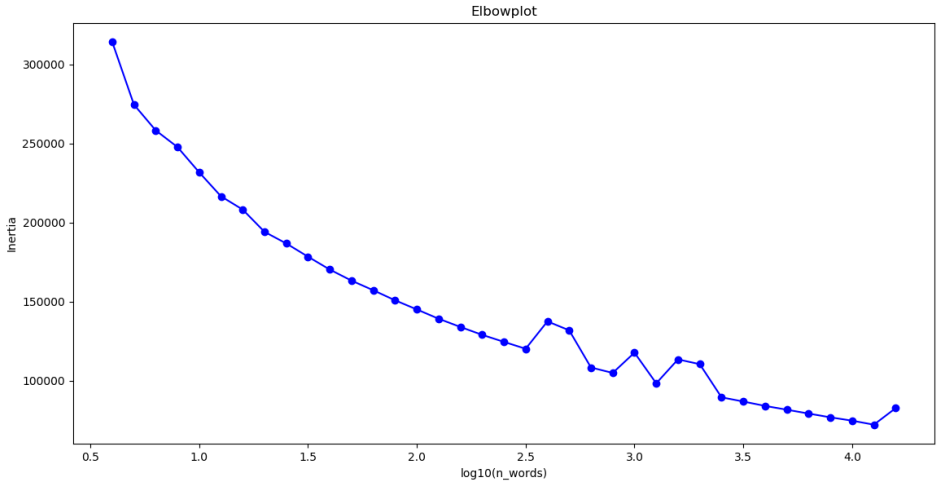


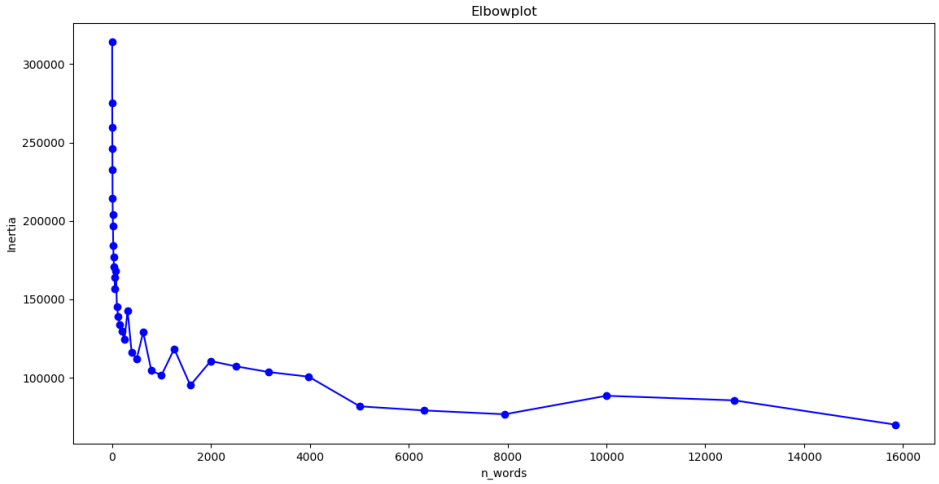
Figure 13 – Exemple d'un Elbowplot possédant un coude évident

En théorie, ce coude indique donc la fin de la décroissance liée au clustering efficace et le début de la pente douce inévitable liée à l'augmentation du nombre de groupes. On estime donc que le nombre de groupes associés à ce coude est le nombre de groupes réellement présents dans les données. Cependant, si on constate l'apparition nette de ce coude dans certains cas, il n'est pas toujours possible de trancher par cette technique. En fait, plus les données forment des groupes distincts et plus le coude sera marqué sur ce graphique. Ne pas arriver à distinguer clairement une rupture de pente signifie donc que les données ne sont pas spécialement disposées de façon à former des agrégations, et tendent plus vers une répartition uniforme.

Nous avons donc tracé cet Elbowplot pour notre ensemble de descripteurs extraits de la base d'images.



(a) Avec transformation logarithmique des abscisses



(b) Sans transformation logarithmique des abscisses

Figure 14 – Tracé de l'Elowplot avec la base d'images

L'elbowplot tracé avec l'axe des abscisses en échelle logarithmique ne permet pas de distinguer clairement une cassure dans la décroissance de l'inertie. Même s'il vise à présenter une plus grande plage de nombre de groupes de manière lisible, la transformation écrase le coude et ne permet pas d'en tirer des conclusions. À l'inverse, le tracé sans la transformation logarithmique permet d'identifier une cassure nette. On peut estimer avec ce graphique que le nombre de mots optimal se situe entre 1000 et 2000 ; il reste cependant très difficile ici de donner une valeur plus précise.

Les irrégularités sont dues à des optimisations non parfaites de l'algorithme K-Means ; cependant, on devine facilement l'allure que devrait avoir la courbe si tout avait été parfaitement optimisé.

La technique d'identification du nombre de groupes par le tracé de ce graphique n'étant ni très précise, ni complètement fiable, il a fallu trouver une autre méthode afin de préciser et valider la valeur choisie pour le nombre de groupes. Nous allons maintenant utiliser des cartes de niveaux, que nous appellerons aussi colormaps à cause de leur aspect, pour déterminer les hyperparamètres optimaux.

2.4.2 Colormaps

2.4.2.1 Choix des noyaux

Nous avons choisi ici de nous intéresser à 4 noyaux particuliers que sont le noyau linéaire, le noyau gaussien (RBF), le noyau laplacien et le noyau du χ^2.

C'est un choix arbitraire qui peut être remis en cause et amélioré, cependant la métrique du χ^2 semble particulièrement adaptée dans notre cas. En effet, cette métrique est conçue pour l'étude des distances entre distributions de probabilités. Ici, nous représentons les images comme un ensemble de fréquences d'apparition de chaque descripteur, c'est-à-dire, à très peu de choses près, comme une distribution de probabilité d'apparition de descripteurs. Il semble ainsi naturel *a priori* que ce noyau donne des résultats intéressants.

Nous allons donc tracer ces colormaps en faisant varier 2 paramètres et pour ces 4 noyaux. Il nous reste cependant 3 paramètres à faire varier, il va falloir en fixer un arbitrairement pour observer l'influence des 2 autres. Le risque est de fixer ce paramètre restant trop loin de la zone d'intérêt et ainsi d'observer un résultat homogène et de très mauvaise précision. Même si la précision de l'elbowplot n'est pas très bonne, nous allons tout de même fixer la valeur du nombre de mots à 1000. C'est en effet le seul paramètre pour lequel nous avons une bonne intuition pour l'instant, nous nous occuperons de le régler plus tard.

2.4.2.2 Détermination conjointe des paramètres γ et C

Afin de couvrir un grande plage de valeurs, nous prenons $\gamma \in [10^{-5}, 10^{4}]$ et $C \in [10^{-4}, 10^{5}]$ et affinerons la recherche par la suite. Évidemment les axes de la carte de niveaux sont en échelle logarithmique pour une meilleure lisibilité.

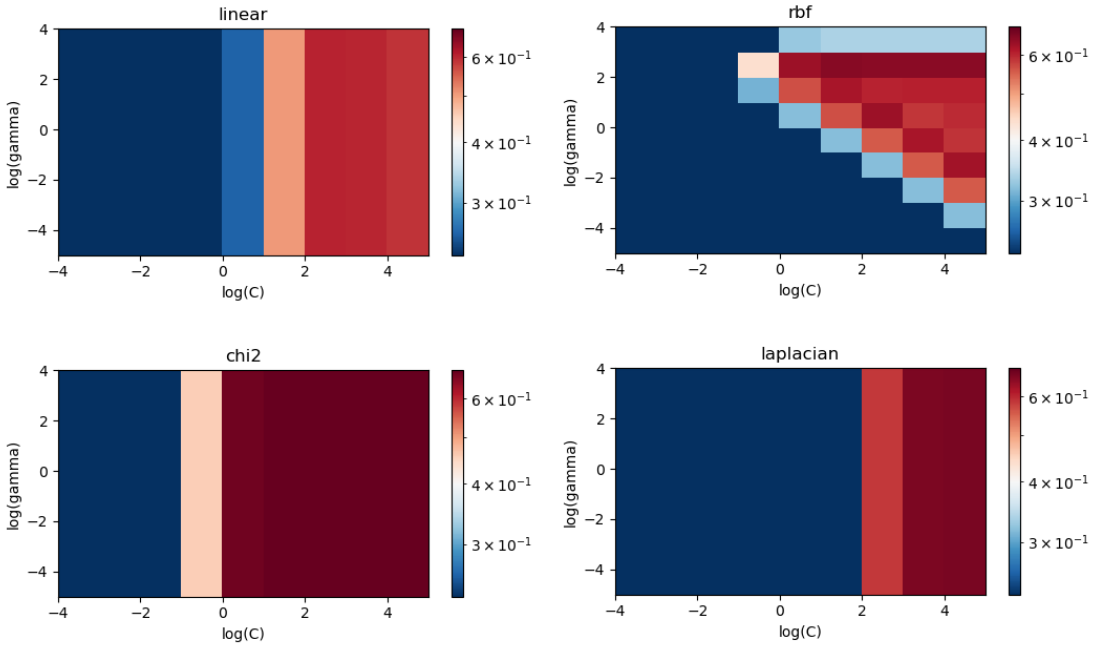


Figure 15 – Colormap pour 1000 groupes de descripteurs

On constate que le noyau du χ^2 semble donner légèrement de meilleurs résultats. Par ailleurs, et de façon surprenante, son efficacité et celle du noyau laplacien ne paraissent pas dépendre de la valeur de γ. Si cela n'a rien de surprenant pour le noyau linéaire, qui lui ne dépend réellement pas de γ, ce résultat est moins facilement explicable pour les deux autres.

Pour ces deux raisons nous allons considérablement faciliter l'obtention des paramètres optimaux en ne considérant plus que 2 hyperparamètres : le nombre de clusters et C pour le noyau du χ^2 qui ne semble pas dépendre de γ.

2.4.2.3 Détermination conjointe des paramètres n_{words} et C

Nous allons donc maintenant tracer la colormap du score de classification en fonction seulement de C et du nombre de groupes de descripteurs. Pour s'assurer que nous ne faisons pas fausse piste depuis le début, nous allons tracer cette carte pour une large plage de nombre de clusters : on prendra ce nombre $n_{words} \in [10, 2.10^4]$.

Une valeur supérieure n'aurait de toute façon pas beaucoup de sens au vu du peu d'images considérées, on approcherait en effet d'un groupe par mot, ce qui n'a aucun intérêt pour la classification.

Afin de contrôler le sur-apprentissage et le bon comportement du modèle, on affiche en même temps une carte représentant la variance des scores de Cross-Validation. Si la variance est trop importante, il est possible que le modèle soit trop instable et donc suboptimal.

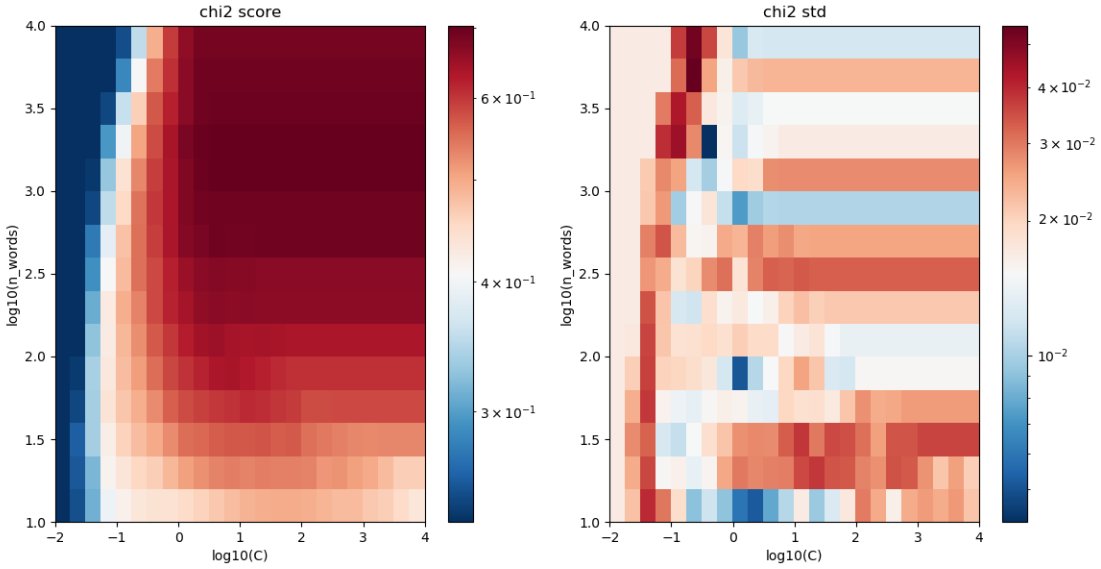


Figure 16 – Colormap des scores et variances des modèles pour le noyau χ^2

La zone de performances maximale semble effectivement correspondre à $n_{words} \in [1000, 2000]$. On note également que la valeur du paramètre C doit être supérieure à 1. Il ne semble pas y avoir de différences de performances pour de grandes valeurs de C, nous allons regarder sur une fenêtre plus retreinte pour avoir une meilleure idée.

Comme mentionné précédemment, une grande valeur de C pénalise grandement les erreurs. Ainsi, prendre un C trop important favorisera le sur-apprentissage, on aura donc tendance à prendre le plus petit C possible parmi ceux donnant les meilleures performances.

Nous avons observé avec une plus grande résolution la zone d'intérêt pour choisir au mieux nos hyperparamètres.

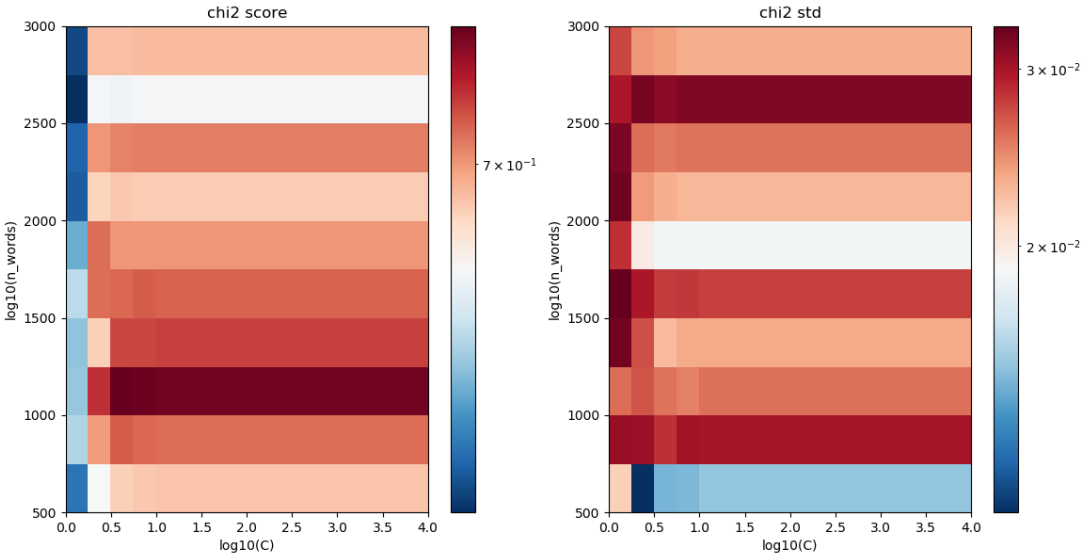


Figure 17 – Colormap des scores et variances des modèles pour le noyau χ^2

Malgré une optimisation probablement non parfaite des groupes de mots, il semble tout de même se dessiner une tendance. De ce graphique, on est tenté d'interpréter que le nombre de groupes optimaux se situe dans $n_{words} \in [1000, 1200]$. Par ailleurs, au vu des critères évoqués pour la pénalisation des erreurs, $C \in [10^{0.5}, 10^{0.8}]$ soit $C \in [3, 6]$ parait confortable.

2.4.3 Modèle SVM final et résultats

Pour le modèle final, on a finalement choisi $C = 4$ et $n_{words} = 1000$ avec un noyau du χ^2, la valeur de γ n'ayant pas d'importance.

On obtient alors un taux de bonnes classifications de **71.0 %** en 5-Fold Cross Validation, et la matrice des confusions suivante :

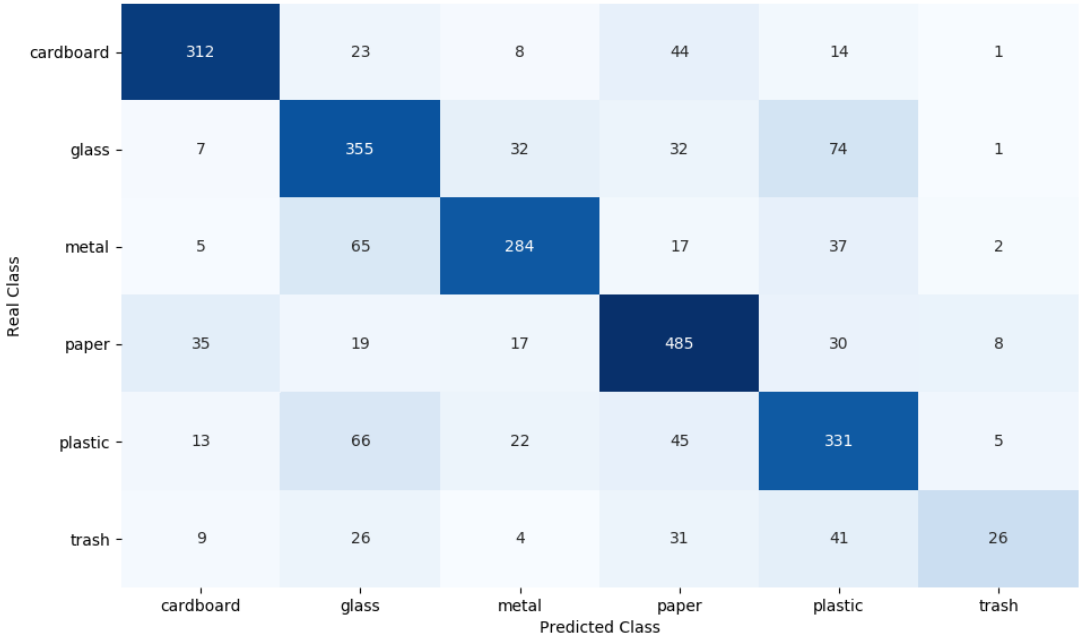


Figure 18 – Matrice de confusion cumulée du modèle SVM à noyau χ^2 en 5-Fold Cross Validation

Figure 19 – Exemples de mauvaise classifications par cette méthode (classe prédite/classe réelle)

3 Les modèles à réseaux convolutifs

3.1 Principe d'un CNN

3.1.1 Idée générale

Un réseau de neurones peut être vu comme une fonction de transfert, prenant en entrée des données et donnant en sortie une valeur. Dans notre cas, notre problème est un problème de classification, la valeur de sortie sera donc une des classe parmi lesquelles il est possible de classifier les images, les entrées étant les images brutes elles-mêmes.

Un CNN ou Convolutional Neural Network, est un réseau de neurones particulier, décomposé en 2 parties. La première partie, que nous appellerons compréhension, est une succession de transformations visant à extraire les composantes principales des données d'entrées, alors que la deuxième partie est un perceptron multicouche (MLP) entièrement connecté et visant directement à la classification des images. Cette deuxième partie, appelée classifieur, prend donc en entrée les données synthétisées par la première phase pour donner en sortie une catégorie parmi lesquelles on souhaite classifier nos données.

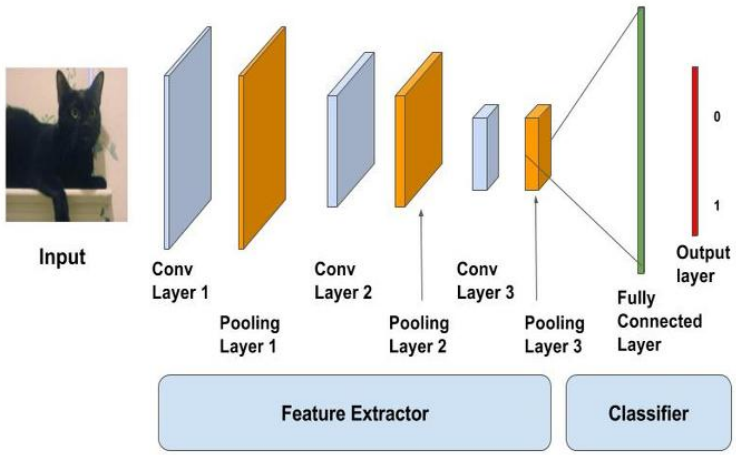


Figure 20 – Illustration d'un CNN

Source : learnopencv.com

Les CNN sont particulièrement efficaces pour la classification d'images, ce qui en fait un outil prioritaire de ce projet. En effet, ce genre de réseaux prend directement en entrée des images sous la forme de 3 canaux (RGB). Il n'y a donc pas de perte d'information liée à une étape intermédiaire de redescription de l'image. En particulier, ce modèle permet de conserver toutes les couleurs, ce qui n'était pas le cas du modèle SVM précédent,

qui travaillait exclusivement sur des images converties en niveau de gris, et perdait ainsi l'information contenue dans les couleurs.

L'architecture particulière du CNN lui permet de restreindre son nombre de paramètres, et donc de faciliter son optimisation. Ainsi il lui est possible de prendre les données plus volumineuses en entrée (comme des images) et de les faire traverser un nombre de couches plus important que le MLP. C'est cette particularité qui lui permet un apprentissage plus en profondeur, pour pouvoir capter de façon plus efficace et à différents niveaux de perception les caractéristiques d'une image.

3.1.2 Fonctionnement de la partie compréhension

La première phase du CNN peut être vue comme une succession de blocs. Classiquement, chaque bloc est composé, dans cet ordre, d'une convolution, d'une activation et d'un pooling. Nous allons définir ces trois opérations.

C'est cet enchaînement qui permet l'extraction des caractéristiques des données d'entrée, tout en en réduisant la taille au fur et à mesure du processus. L'étude des topologies des CNN est un domaine actif de la recherche en machine learning, nous ne verrons ici que des modèles simples et à topologies fixes de CNN.

3.1.2.1 Convolution

L'objectif principal de la convolution est d'extraire de l'image des caractéristiques locale. On parcourt en effet l'image avec un masque rectangulaire (souvent carré) de taille fixée, dans les deux directions. Ce masque ne considère donc simultanément que des zones restreintes de l'image. Il est formé d'un ensemble de poids qui seront associés à des pixels pour pondérer leurs valeurs.

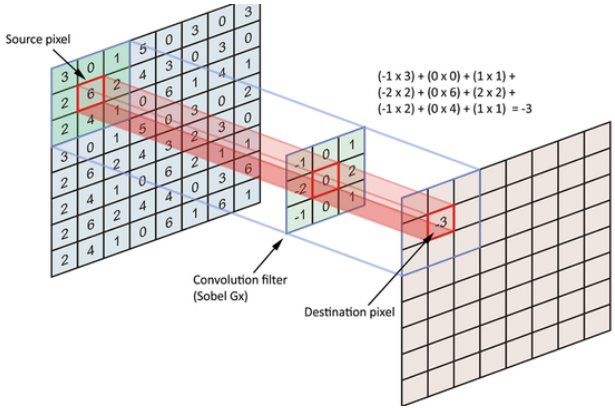


Figure 22 – Illustration d'une convolution

Source : medium.freecodecamp.org

Lorsque le masque passe sur une zone de l'image, on multiplie la valeur du pixel considérée par le poids du masque et on somme ces valeurs. On obtient ainsi une nouvelle matrice de valeurs, de dimension légèrement inférieure. Si on souhaite obtenir une taille de sortie égale à la taille d'entrée on peut rajouter des pixels (classiquement nul) tout autour de l'image. Le nombre de tours d'épaisseur 1 pixel rajoutés est appelée padding et noté pad.

On obtient donc la formule suivante pour calculer la taille de la matrice à la sortie de la convolution ; la formule est écrite ici pour la hauteur mais est généralisable pour chaque dimension.

$$h_{sortie} = h_{entree} - h_{masque} + 1 + 2pad$$

Si on veut par exemple $h_{sortie} = h_{entre}$, il faut donc prendre : $pad = \frac{h_{masque}-1}{2}$, Ce qui implique notamment d'avoir une hauteur de masque de taille impaire.

En réalité les données ne sont pas des matrices 2D, mais comportent plusieurs canaux (par exemple Rouge, Vert et Bleu pour une image). Les masques ne sont donc pas des simples rectangles de paramètres, mais plusieurs rectangles, autant que les données d'entrées possèdent de canaux.

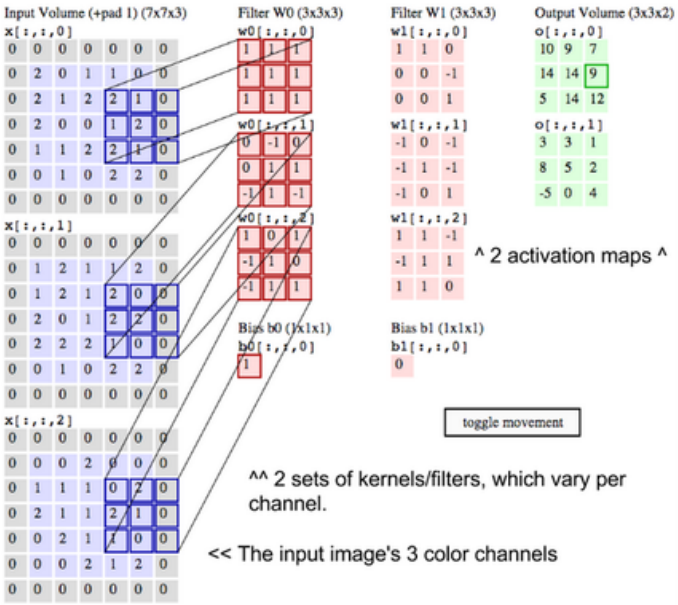


Figure 24 – Illustration d'une convolution à plusieurs canaux

Source : ensiwiki.ensimag.fr

On peut également choisir de définir plus d'un masque par couche, et renvoyer en sortie les résultats de plusieurs convolutions avec plusieurs masques. Le nombre de masques définit le nombre de canaux de la couche suivante (2 dans l'exemple ci-dessus).
On a donc :

Nombre de canaux de la couche n+1 = Nombre de masques de la couche n

3.1.2.2 Fonction d'activation

La sortie d'une convolution est donc un ensemble de matrices, elles-mêmes composées de combinaisons linéaires des entrées. Lorsque on considère un réseau de neurones à plusieurs couches, les neurones prennent en entrée les sorties des couches précédentes. Ainsi, sans modifications supplémentaires, mettre plusieurs couches n'apporte rien de plus qu'une seule couche, puisque cela reste des combinaisons linéaires des entrées du réseau.

Afin de modéliser des comportements plus génériques, on applique une transformation non linéaire à la sortie d'une convolution. Cette transformation est appelée fonction de transfert et casse donc la linéarité pour permettre au modèle de généraliser un plus grand nombre de comportements.

Il existe un grand nombre de fonctions de transfert utilisées dans les réseaux de neurones, dans notre cas nous utiliserons une des plus connues : ReLU pour Rectified Linear Units. Elle est définie comme étant le maximum entre x et 0.

$$ReLU(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$$

En plus de casser la linéarité, ce qui est le principal but recherché, elle présente la particularité d'être extrêmement simple à calculer. Elle favorise ainsi l'entraînement d'un grand nombre de paramètres en ne rajoutant pas de complexité ni de temps de calcul supplémentaire lié à son évaluation.

Pour cette raison, c'est une des fonctions d'activation les plus utilisées de nos jours dans le deeplearning. Nous n'utiliserons qu'elle dans nos réseaux par la suite.

3.1.2.3 Pooling

L'opération de pooling vise elle à synthétiser l'information. Son utilité principale est d'extraire les éléments importants ou caractéristiques d'une matrice pour en réduire considérablement la taille. De manière analogue à la convolution, le pooling sur une matrice se base sur un rectangle qui va parcourir la matrice dans les deux directions.

Cette opération prend donc les valeurs à l'intérieur du rectangle de la matrice considérée, et renvoie un unique nombre synthétisant les données contenues dans ce rectangle. Souvent on choisit le MeanPooling qui renvoie la moyenne du rectangle, ou le MaxPooling pour le maximum.

On choisit également un pas, appelé $stride$, qui correspondra au décalage entre deux rectangles lors du parcours de l'image. En effet, comme l'objectif de cette opération est de diminuer la taille de l'objet manipulé, on va décaler le rectangle de pooling de plus d'une case pour effectuer l'opération suivante, et ainsi diviser considérablement la taille de la matrice de sortie.

Toujours pour les hauteurs de la matrice et du rectangle, mais de façon généralisable :

$$h_{sortie} = \frac{h_{entree} - h_{rectangle} + 1 + 2pad}{stride}$$

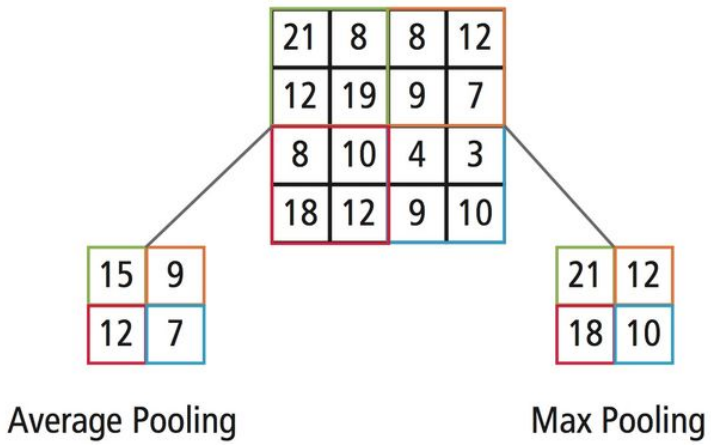


Figure 26 – Illustration du pooling

Source : davidsbatista.net

De façon générale, on privilégie le MaxPooling, qui aura tendance à capturer mieux les caractéristiques pour une image. En effet, la fonction max va mettre en avant les phénomènes saillants tels que des arêtes ou des ruptures dans les contours, qui traduisent bien souvent des zones d'importance d'une image.

Nous n'utiliserons que le Maxpooling dans les CNN que nous entraînerons.

3.2 Optimisation du réseau

Comme nous l'avons dit précédemment, un CNN peut être assimilé à une fonction de transfert dépendant d'un très grand nombre de paramètres. Dans notre cas, cette fonction de transfert prend en entrée des images et renvoie une classe parmi les 6 types de déchets possibles.

Afin que cette fonction de transfert donne les meilleurs résultats possibles, il est indispensable d'en optimiser ses paramètres.

3.2.1 Fonction de Loss

Pour optimiser ce CNN, il faut évidemment définir un critère de performance. Intuitivement on pourrait penser qu'il suffit de compter $+1$ lorsqu'une image est mal classée et 0 dans le cas contraire, puis chercher à minimiser le résultat de la somme sur les images. Ce genre de construction est appelé une fonction de Loss, le critère de performance étant donc de minimiser la valeur de la fonction de Loss.

Cependant cette fonction de Loss n'est pas différentiable, ce qui rend l'optimisation du réseau très complexe. Nous allons lui préférer une version moins naïve pour de la classification : la fonction Cross Entropy Loss. Cette fonction repose sur la fonction Softmax que nous allons introduire maintenant, et présente l'avantage d'être dérivable en tout points.

3.2.1.1 Softmax

La partie classifieur du CNN est essentiellement un perceptron multicouche. Comme on a 6 classes, on choisit d'avoir la dernière couche de ce perceptron composée de 6 neurones. Pour chaque image, ces 6 neurones peuvent *a priori* renvoyer des valeurs dans $\mathbf{R}$. Avec la méthode naïve précédente, on pourrait choisir de classifier une image dans la catégorie qui prend la valeur maximale en sortie, cependant, la notion de Cross Entropy repose sur une notion de probabilité.

Nous voulons en fait que les 6 sorties du classifieur ne soient plus des valeurs de $\mathbf{R}$, mais des nombres dans $[0, 1]$ correspondants à des probabilités. Pour cela on définit la fonction SoftMax, qui prendra en entrée la couche brute de sortie du MLP, que nous noterons Y, et la transformera en vecteur associé à la probabilité de classification pour chaque classe.

$$\forall c \in Classes, \quad SoftMax(Y_c) = \frac{exp(Y_c)}{\sum_{k \in Classes} exp(Y_k)}$$

On notera $S_{max}(Y)$ le vecteur composé de tous les $SoftMax(Y_c), \ \ c \in Classes$.

L'idée est globalement la même que la méthode naïve qui prend l'argument maximal des sorties. Une distribution de probabilité étant positive, on prend l'exponentielle des sorties pour s'en assurer. Finalement, comme la somme des valeurs possibles doit valoir 1, on normalise par la somme des exponentielles des valeurs possibles. On est ainsi ramené à une probabilité en sortie du réseau.

3.2.1.2 Cross Entropy

On note T le vecteur composé uniquement de 0 et un 1 dans une de ces coordonnées, ce vecteur T représente la probabilité 1 de trouver la classe associée à la coordonnée en question. Sur le principe de l'entropie, on définit la distance (appelée divergence de Kullback et Libeller) entre $S_{max}(Y)$ et T comme :

$$\mathscr{D}(S_{max}(Y), T) = -\sum_{k \in Classes} T_k.ln(S_{max}(Y_k))$$

En notant Y^X le vecteur Y brut associé à l'image X de la base et T^X la classe de cette image, on peut écrire la fonction de Cross Entropy Loss pour un échantillon E de la base d'images :

$$\mathscr{L}(E) = \frac{1}{|E|} \sum_{X \in E} \mathscr{D}(S_{max}(Y^X), T^X)$$

Représentant la moyenne des distances entropiques de l'échantillon. En fait, cette distance compare la probabilité d'une classe avec la classe réelle de l'image. De cette façon, si une image est mal classée, c'est que la probabilité de bien classer cette image était faible, et la distance associée sera grande.

Cette fonction de Loss introduit en fait la notion de confiance dans l'estimation qu'elle fait. Si le CNN effectue une classification avec une grande probabilité, il y a plus de chance qu'elle soit bien classée. Grâce à cette astuce, l'optimisation du CNN est plus efficace et les performances s'en voient améliorées.

3.2.2 Méthode d'optimisation

Optimiser les paramètres d'un CNN suivant cette fonction de Loss n'est pas une tâche facile. La méthode choisie est une méthode de descente stochastique de gradient. Cependant comme le nombre de paramètres du modèle est très important, et qu'il faut calculer la dérivée partielle de la fonction Loss par rapport à chacun d'eux pour les ajuster correctement, cette opération peut s'avérer extrêmement gourmande en terme de calculs.

Une astuce pour calculer de façon moins coûteuse ces dérivées partielles est la rétropropagation. En utilisant la règle de la chaîne, on peut remonter les couches en partant de la fonction de Loss. Il suffit de calculer la dérivée partielle du paramètre par rapport à la

couche suivante et de le multiplier aux dérivées déjà calculées de la couche précédente. Cette rétro-propagation est donc un calcul de dérivées partielles successives partant de la fin du réseau et utilisant les dérivées des couches suivantes, préalablement calculées.

Dans les faits, les dérivées sont en fait des différentielles et le pas de descente est fixé par un coefficient d'apprentissage choisi arbitrairement.

3.2.3 Augmentation des données

Comme les images vont circuler un certain nombre de fois dans le réseau avant que celui-ci ne donne des performances intéressantes, il y a des risques que celui-ci apprenne par cœur les images de la base d'apprentissage. Le problème de ce phénomène est que le CNN sera alors très bon pour prédire les données sur lesquels il s'est entraîné, mais n'aura pas de bonnes performances sur de nouvelles données, qu'il n'a jamais vu.

Ce problème de généralisation, le sur-apprentissage, est très gênant puisqu'il impacte directement les performances du modèle. Il est cependant détectable, par exemple par notre méthode de validation croisée.

Pour se prémunir contre ce sur-apprentissage, nous avons ici choisi de faire varier artificiellement nos données. Lors de leurs passages dans le CNN pour l'entrainement, les images peuvent aléatoirement subir des transformations telles que des légères rotations, des basculements verticaux ou encore un redimensionnement différent. De cette façon les données d'entrée sont plus variées et l'entrainement du CNN est moins sujet à l'apprentissage par cœur de la base d'entraînement.

Cette technique vise également à améliorer la robustesse du classifieur face aux changements. Un réseau convolutif étant par construction invariant par translation mais pas par rotation, on peut par exemple utiliser cette transformation. Ainsi en ajoutant plusieurs fois la même image tourner d'un certain angle (par exemple pris aléatoirement), on va entraîner le CNN à reconnaître une image dans n'importe quel sens. Cette technique favorise la généralisation de l'apprentissage.

3.2.4 Transfert Learning

Un CNN comme AlexNet comporte plus de 60 millions de paramètres, et est donc très long et difficile à optimiser. En effet, au cours de l'entraînement et du fait de la descente de gradient, les paramètres proches de la sortie du réseaux varieront de façon plus conséquentes. Les paramètres les plus proches de l'entrée ont donc besoins d'un très grand nombre d'images afin d'être optimisés.

Notre base étant beaucoup trop petite pour pouvoir entraîner de manière approfondie le réseau AlexNet, le risque est que les premiers paramètres ne soient pas du tout optimisés. Le réseau ne serait alors véritablement optimisé que sur les couches proches de la classification et les performances finales seraient grandement diminuées.

Afin d'utiliser toute la puissance d'un tel réseau profond et optimisé, nous avons choisit d'utiliser du transfert learning. Le principe est de réutiliser les paramètres d'un réseau déjà optimisé sur une base d'image très importante et de le ré-entraîner avec nos images. De cette façon, le CNN est déjà habitué à classifier des images et à en extraire les caractéristiques principales puisque les paramètres des premières couches ont déjà été entraînés à cela. Cependant, comme cet énorme entraînement avait été effectué dans un but différent du nôtre, il est indispensable de l'entraîner de nouveau pour optimiser la fin. Les caractéristiques extraites peuvent alors être utilisées au mieux pour notre problème.

3.3 AlexNet

Nous avons donc implémenté un CNN très connu, **AlexNet**. C'est le deuxième à avoir été vraiment utilisé notamment grâce à ses performances remarquables pour l'époque (2012), le premier CNN étant **LeNet**, du français Yann Le Cun en 1998.

3.3.1 Architecture

AlexNet est un CNN composé de 5 convolutions et prenant en entrées des images de tailles 224 x 224. Sa particularité est qu'il comporte trois convolutions successives sans pooling pour les séparer.

Les pooling sont pris sans intersection, c'est-à-dire que la valeur de $stride$ est égale à la largeur du carré de pooling.

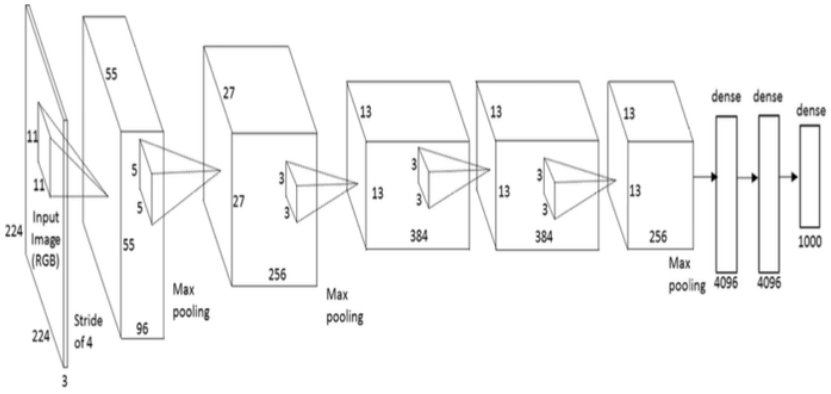


Figure 28 – Architecture d'AlexNet

Source : quora.com

Il est prévu initialement pour pouvoir classifier des images dans 1000 catégories différentes. Dans ce projet, nous cherchons à classifier en 6 catégories seulement, j'ai donc redimensionné les couches cachées du perceptron multicouche pour cela. La structure complètement connectée du classifieur initial était $(4096, 4096, 1000)$ et est maintenant $(4096, 512, 6)$, chaque valeur représentant ici le nombre de neurones de la couche.

3.3.2 Réglage de l'entraînement

Lors de l'entraînement du CNN, il est possible de jouer sur plusieurs curseurs qui influenceront le comportement de l'optimisation. Ces curseurs jouent directement sur la vitesse d'entraînement ainsi que sur les performances d'optimisation du réseau, et donc sur l'efficacité du CNN. Il est donc important de bien les choisir.

3.3.2.1 Le nombre d'epochs

Au cours de l'entraînement, les images de la base de test passent au travers du CNN. En fonction de ses performances sur les images, la descente stochastique de gradient va ajuster les paramètres pour donner de meilleurs résultats. Ce processus est réitéré un certain nombre de fois, théoriquement jusqu'à convergence de l'algorithme d'optimisation.

On appelle une epoch l'événement "la totalité de la base d'entraînement a été propagée une fois au travers du CNN". Ainsi, le nombre d'epochs correspond au nombre de fois où il a fallu faire recirculer la base d'entraînement dans le CNN. Il faut bien entendu plusieurs epochs au réseau pour être complètement entraîné. Cependant, un nombre trop grand d'epochs peut avoir des effets négatifs sur les performances.

En fait, un trop grand nombre de passages d'une image va forcer le CNN à avoir de bonnes performances sur cette image. Il va ainsi finir par s'entraîner à reconnaître spécifiquement cette image, plus que ses caractéristiques. Pour éviter ce sur-apprentissage, on arrête parfois l'entraînement avant sa fin, lorsque les performances du test commencent à diminuer et que le modèle devient moins général et commence à apprendre par cœur la base d'entraînement.

Il est donc crucial de surveiller les performances de la base de validation pendant l'entrainement du modèle. On peut ainsi choisir le nombre d'epoch le plus adapté pour notre jeu de données.

3.3.2.2 Le nombre d'images par batch

Un batch correspond à un ensemble d'images qui sera propagé pendant une même phase. Cet ensemble d'images va donner des résultats sur lesquels sera calculée la valeur

de la fonction de Loss. À la fin du passage d'un batch dans le CNN, la rétropropagation est effectuée et les paramètres du réseau sont ajustés.

Une grande valeur de taille de batch diminuera la variance dans la descente de gradient et favorisera donc l'apprentissage. Par ailleurs, plus la taille de batch est importante et plus le nombre d'images traversant la base simultanément l'est aussi, un grand batch diminue donc la durée de l'entrainement. Il faut *a priori* intérêt fixer la taille de batch aussi grande que possible.

Cependant pour des raisons de calcul, il est impossible de propager simultanément toute la base de test, ainsi la limite de la taille de batch est imposée par l'ordinateur sur lequel on travaille et par la puissance de calcul que l'on veut laisser disponible à l'utilisateur de cette ordinateur pendant l'entraînement. Sur mon ordinateur, j'ai pris une taille de batch égale à 30.

3.3.2.3 Le coefficient d'apprentissage

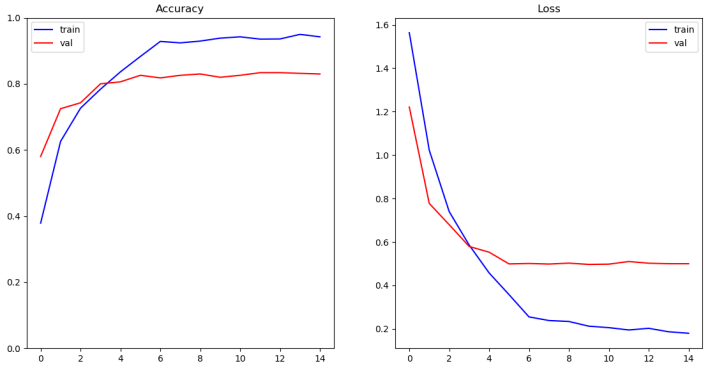
Le coefficient d'apprentissage, ou learning rate, permet de fixer la taille des pas de la méthode de descente du gradient. Sa valeur va donc influencer significativement la vitesse d'apprentissage de l'algorithme.

Avoir un learning rate trop petit va considérablement ralentir l'entraînement, à tel point que s'il est vraiment trop faible il est possible de ne pas voir les effets de l'apprentissage sur un grand nombre d'epochs. À l'inverse, un coefficient trop élevé va faire converger très vite les performances du CNN. Cependant ces performances oscilleront ensuite autour de la valeur optimale sans jamais l'atteindre du fait du pas trop important. Cela créé donc de l'instabilité dans le modèle et en affectera les résultats.

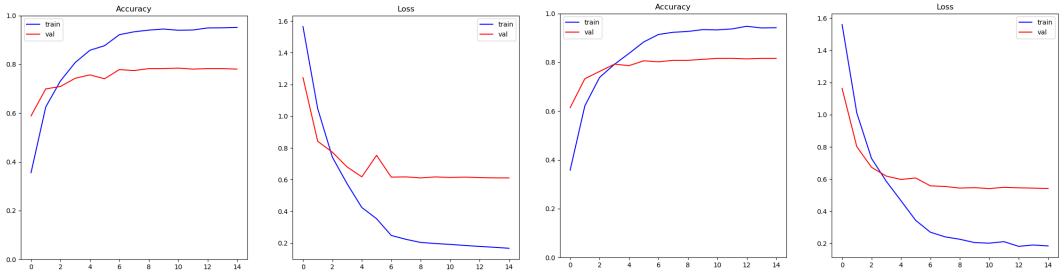
Ce coefficient est donc à choisir avec le plus grand soin afin d'avoir une vitesse de convergence raisonnable tout en conservant la stabilité du modèle.

3.3.3 Résultats

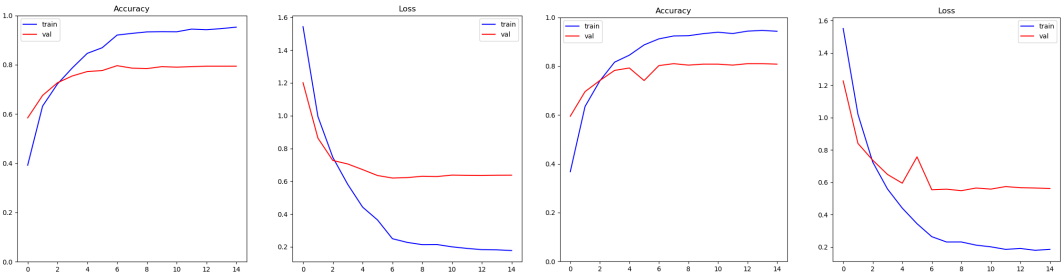
Après plusieurs expérimentations, il s'est avéré qu'un entraînement sur 15 epochs était amplement suffisant. Pour le contrôle, on a tracé les précisions sur la base d'apprentissage et sur la base de validation.



(a) Fold 1



(b) Fold 2 (c) Fold 3



(d) Fold 4 (e) Fold 5

Figure 30 – Résultats des entraînements d'AlexNet en 5-Folds Cross Validation

On peut remarquer que le score de validation est toujours meilleur que le score d'entraînement sur les premières epochs. Cela peut sembler très étonnant à première vue, cependant ce phénomène s'explique par le fait que le score de validation est effectué après l'entraînement alors que le score d'entraînement se calcule pendant celui-ci. En effet, au cours des premières epochs le CNN progresse très vite, il est donc logique que ses performances soient bien meilleures après son entraînement qu'au début.

De la même façon, il est probable qu'au cours d'une même epoch, les résultats de l'entraînement soient bien meilleurs vers la fin de l'epoch qu'au début de cette même epoch.

Cette tendance s'inverse tout naturellement par la suite, notamment à cause du phénomène de sur-apprentissage.

En moyenne des 5 parties de la validation croisée, ce modèle obtient un score de bonne classification de **80.8 %**, ce qui semble tout à fait honorable au vu de la taille de la base d'images et de la complexité du problème.

Il semble intéressant de regarder la matrice confusion cumulée. Cette matrice représente les nombres d'images classifiées dans les catégories en fonction de leur appartenance réelle. Comme c'est une matrice cumulée sur la Cross Validation, chaque image de la base de départ apparaît exactement une fois dans cette matrice.

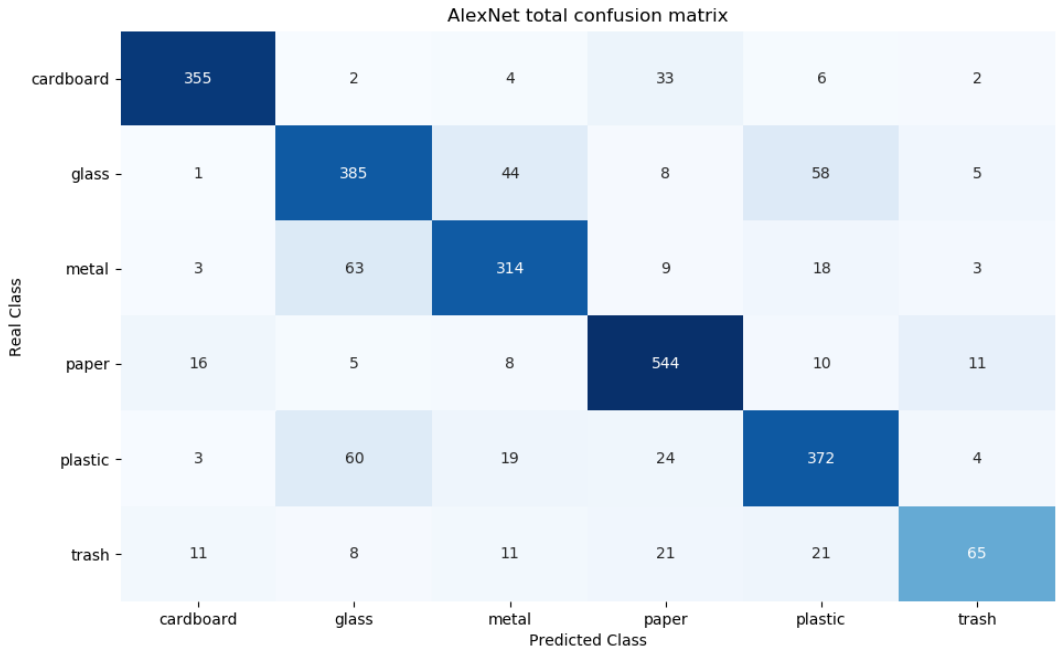


Figure 31 – Matrice de confusion cumulée d'AlexNet en 5-Fold Cross Validation

En analysant cette matrice, on se rend compte que les principales confusions surviennent entres les couples plastiques/verres, métaux/verres et papiers/cartons. S'il semble facile de comprendre la confusion entre verres et plastiques à cause de la transparence de ces matériaux et la présence de bouteilles dans chacun d'eux, ainsi que la confusion entre cartons et papiers puisque la frontière entre ces deux matériaux est assez mince, il est plus difficile d'expliquer le nombre important de confusions en métaux et verres.

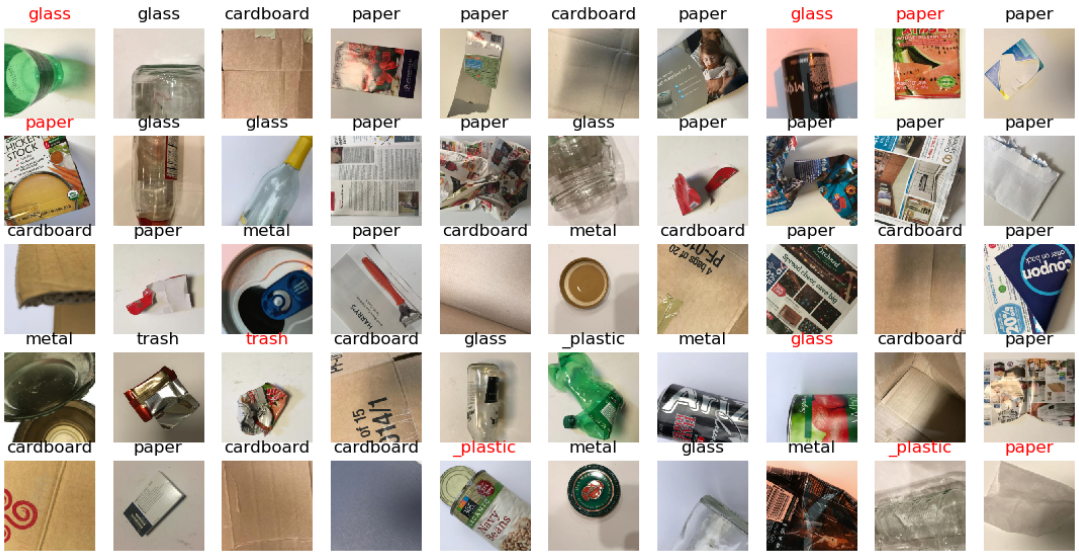


Figure 32 – Exemple de classification avec AlexNet (les mauvaises sont en rouge)

3.4 Resnet50

Afin d'augmenter encore les performances de la classification, j'ai voulu essayé un modèle plus complexe et plus profond. J'ai choisi ResNet50, un des plus performant aujourd'hui et pouvant être entrainé sur ma machine. Resnet exite sous plusieurs versions (ResNet18, ResNet34, Resnet50, ResNet101 et ResNet152), la valeur du nombre terminal correspondant au nombre de couches du réseau.

Du fait de sa grande profondeur, ce modèle capte mieux les informations intéressantes. En contrepartie, il est beaucoup plus coûteux de l'entrainer. J'ai notamment dû passer la taille de batch de 30 pour AlexNet à 3 pour ResNet50.

Après entrainement, on obtient un taux de bonne classification de 89.6% et la matrice des confusions suivante :

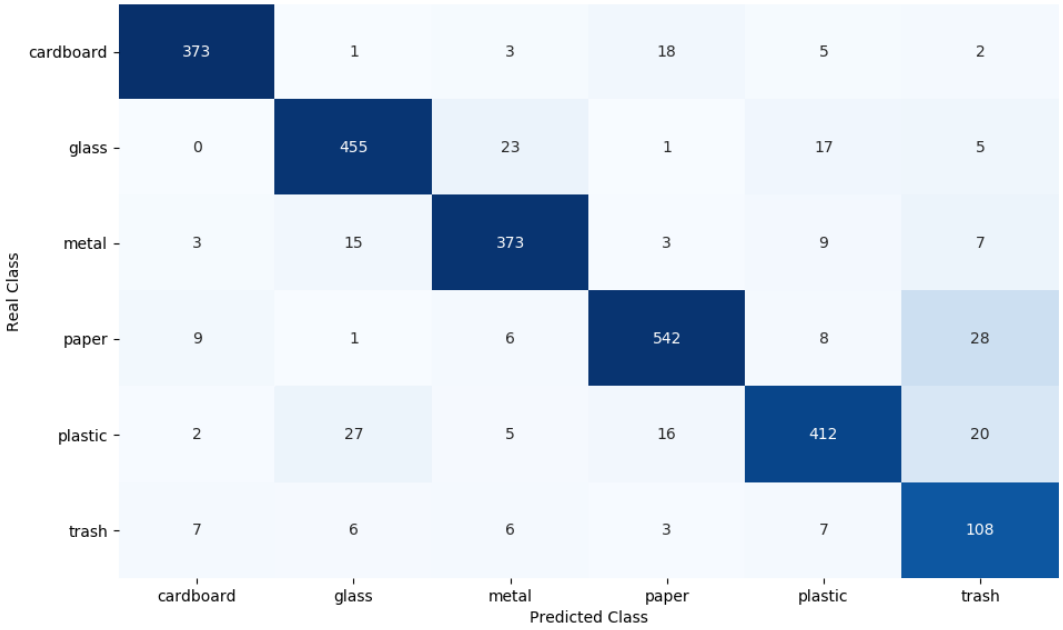


Figure 33 – Matrice de confusion cumulée de ResNet50 en 5-Fold Cross Validation

Les performances semblent bien meilleures, notamment pour l'identification de la catégorie $Trash$, qui était une des plus problématiques jusque-là.

4 Limites de l'apprentissage et critiques des modèles

4.1 Spécificités de la base d'images

La base d'images sur laquelle nous avons essayé de distinguer nos différentes catégories de déchets est en fait très spécifique. Sa taille très limitée ne permet pas un apprentissage raffiné des caractéristiques des matériaux. Par ailleurs, toutes les photographies sont prises sur un fond blanc uni, ce qui en fait un jeu de données très particulier.

De plus, cet ensemble d'images ayant été créé de façon artisanale, il comporte plusieurs fois des photos des mêmes objets pris sous des angles différents. Cette spécificité nous amène à nous demander dans quelle mesure nos résultats sont généralisables à des objets encore inconnus de la base, et ne sont pas simplement le fruit d'un apprentissage par cœur de ces objets.

Il serait intéressant de tester de nouveaux objets sur un fond uni, pas nécessairement blanc, pour voir comment se comporte le modèle.

4.2 Quantité d'informations du jeu de données et perspectives

Le jeu de données étant un ensemble fini d'images, il contient nécessairement une quantité finie d'information. En essayant de caractériser une quantité potentiellement infinie de caractéristiques d'un matériau par un nombre fini de données, il est logique que les performances du modèle ne soient pas de 100%.

De cette façon, il est probable qu'il existe un score maximal intrinsèque à la base, simplement à cause de la quantité d'information qu'elle contient. Sous cette hypothèse, retravailler les différents choix de la complexité du modèle, et comment ce dernier capte les caractéristiques des images, pourrait permettre d'augmenter ses performances que jusqu'à ce seuil théorique limite.

La base ne comportant que 2500 images, il ne parait pas absurde d'envisager que nos 90% de bonnes classifications du dernier modèle présenté soit en fait assez proche de cette limite. Il serait alors inutile d'essayer de trouver un meilleur modèle, mais il serait sans doute plus intéressant de trouver le modèle le plus synthétique et léger permettant d'atteindre cette limite. Dans notre cas, l'entraînement de ResNet50 en 5-Fold Cross Validation par exemple à pris plus de 8 heures sur un ordinateur portable avec une carte graphique de 2GB de VRAM et un CPU tournant à 2.0 GHz. Il pourrait donc être intéressant de trouver un modèle plus rapide à entraîner.

4.3 Absence de solution exacte

Comme nous venons de le voir, ce problème n'a que très peu de chance de posséder une solution parfaite. C'est un problème difficile, qui se révèle non trivial pour un humain.

En effet, même une personne réelle aurait un taux d'erreur non nul au vu de certaines images et des similarités qui peuvent exister entre deux images de deux catégories différentes.

(a) Un élément de la classe $Verres$ (b) Un élément de la catégorie $Plastiques$

Figure 34 – Exemple d'images de la base de donnée qu'il est difficile de classer avec certitude pour humain

L'illustration précédente met en avant cette particularité dans le cas d'un plastique et d'un verre, où même l'œil du consommateur expérimenté ne sait distinguer avec certitude le matériau du déchet présenté sur la photo.

Conclusion

Ce projet a été l'occasion de mettre en œuvre différents algorithmes de machine learning et de comparer leurs performances. Les résultats obtenus semblent satisfaisants, et même de façon très étonnante. En effet, l'article prit pour référence au départ du projet [1] annonce des scores de bonnes classifications culminant à 65%, largement dépassés ici.

Si ce projet touche à sa fin sur une note encourageante, le recyclage et le tri de déchets en milieux réels, lui, ne fait que commencer. Des défis des prochaines années seront d'arriver à exploiter suffisamment cette technologie de reconnaissance d'images et de la combiner avec ce qui est déjà existant aujourd'hui afin de passer de façon performante à un tri sélectif à grande échelle. C'est de cette façon que les ressources limitées dont nous disposons pourrons être réexploitées au mieux, et de sorte à ne plus tendre inévitablement vers un épuisement des matières premières qui nous sont si précieuses.

Références

[1] Technical Report, *Classification of trash for recyclability status*. Mindy Yang , Gary Thung, Stanford University, CA, USA, 2017.

[2] Bachelor's Thesis, *Waste Sorting using Neural Networks*. Jozef Marko, Masaryk University, Czech Republic Fall 2016.

[3] Journal paper, *Waste segregation using Machine Learning*. Yesha Desai, Asmita Dalvi, Pruthviraj Jadhav, Abhilasha Baphna, University Of Mumbai, India, Int. J. Research in Applied Science & Engineering Technology, Voll. 6, Issue III, 5 p., March 2018.

[4] Journal paper, *Application of automated image analysis to the identification and extraction of recyclable plastic bottles*. Edgar Scavino, Dzuraidah Abdul Wahab, Aini Hussain, Hassan Basri, Mohd Marzuki Mustafa, Universiti Kebangsaan, Malaysia, Journal of Zhejiang University - Science A, Vol. 10, Issue 6, pp. 794-799, June 2009.

[5] Conference paper, *Applying Machine Learning to the sorting of recyclable containers*. Kenneth A. Tarbell, David K. Tcheng, Michael R. Lewis, Ty A. Newell, University of Illinois at Urbana-Champaign, IL, USA, Proc. 15th National Waste Processing Conference, Detroit, MI, USA, pp. 209-215, 17-20 May 1992.

[6] Conference paper, *An application of Image Processing for automated mixed household waste sorting system*. Zol Bahri Razali, Gunasegaran Madasamy, Universiti Malaysia Perlisi, Malaysia, Proc. Int. Conference on Man Machine Systems (ICoMMS2012), February 2012.

[7] Journal paper, *Implementation of smartbin using convolutional neural networks* . Sachin Hulyalkar, Rajas Deshpande, Karan Makode, Siddhant Kajale, Sinhgad College of Engineering, India, Int. Research Journal of Engineering and Technology, Vol 5, Issue 4, pp. 3352-3358, April 2018.

[8] Journal paper, *Novel smart waste sorting system based on Image Processing algorithms : SURF-BoW and Multi-class SVM* Yijian Liu, King-Chi Fung, Wenqian Ding, Hongfei Guo, Ting Qu, Cong Xiao, Jinan University, PRC, Computer and Information Science Archives, Vol. 11, No. 3, pp 35-48, 2018.