Université de Montréal

IFT6390A - Fondements de l'apprentissage automatique

Compétition *Kaggle* 2

ASCII langage des signes

Matricules
20077521
20096040

Identifiants Kaggle
Pierre-Antoine Bernard
SimoHakim

Nom de l'équipe Kaggle
SimoHakim

Noms
Pierre-Antoine Bernard
Simo Hakim

13 décembre 2023

1 Introduction

La reconnaissance de chiffres manuscripts à partir de données visuelles est l'exemple emblématique de problème d'apprentissage machine axé sur la classification d'images. Certaines méthodes modernes comme les réseaux de neurones convolutifs ont grandement facilité ce défi, rendant difficile la distinction de leurs performances respectives. Il apparait donc nécessaire d'introduire un nouveau problème étalon poussant ces algorithmes aux limites de leur capacité et permettant d'évaluer leurs forces et faiblesses respectives.

Dans ce projet, nous avons étudier la performance de différents modèles de classification d'images affectés à la tâche suivante : reconnaitre des charactères de l'alphabet latin à partir d'images de mains représentant ceux-ci dans le langage des signes. La performance des algorithmes a été mesuré par leur taux de succès à prédire les paires de charactères associés à des paires d'images. Nous avons considéré une forêt d'arbre décisionnels, un modèle de réseau de neurone convolutif et un modèle XGBoost.

2 Traitement des données

Format des données : Les données d'entraînement utilisées dans ce travail sont composées de 27455 images de dimension 28×28. La valeur associée à chaque pixel est un entier entre 0 et 255, représentant un niveau de gris. Notons que les attributs associés à chaque pixel sont organisés comme un vecteur de dimension 728 dans les données d'origine. Ces derniers ont été réorganisés en tableaux de dimension 28×28 pour mettre de l'avant la relation locale entre les attributs associés à des pixels voisins. La valeur de chacun des attributs associés à des pixels a aussi été divisé par 255 afin qu'elle soit dans l'intervalle $[0, 1]$.

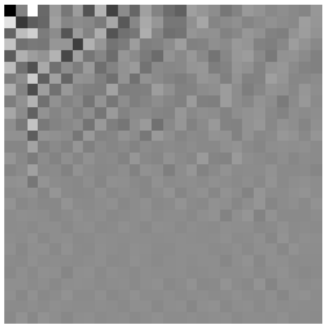
Chaque image a aussi un attribut entier entre 0 et 25 qui correspond à la cible à prédire. Les entiers 0-25 sont associés chronologiquement aux charactères de l'alphabet latin A-Z. Il n'y a pas de données pour $9 = J$ et $25 = Z$ puisque ces lettres demandent un mouvement de la main dans le langage des signes. Pour simplifier le travail, nous avons remplacé les valeurs $24 = Y$ par 9, menant à un ensemble de cibles $\{0, 1, \ldots, 23\}$ sans valeurs manquantes.

Normalisation : La plupart des algorithmes d'apprentissage machine demande de normaliser les attributs afin d'éviter l'introduction de biais associés aux différents ordres de grandeurs d'attributs distincts. Ici, puisque chaque attribut correspond à un pixel et partage la même échelle, une telle normalisation n'apparaît pas nécessaire. Elle a tout de même été implémentée sous la forme de normalisation de sous-échantillons (i.e. *batch normalisation*) entre les couches du réseau de neurone convolutif. Elle a aussi été implémenté par une soustraction à chaque pixel d'une image de la valeur moyenne des pixels pour cette image, pour le modèle de forêt aléatoire. Cette soustraction peut être interprétée comme l'ajustement de la luminosité des différentes images.

Décomposition en modes de Krawtchouk : Dans les modèles de forêt aléatoire, les attributs sont en correspondance avec les questions pouvant être posées aux noeuds des arbres. Or, l'information contenu dans un unique pixel rend difficile la distinction entre les classes cibles et informe peu sur la structure globale de l'image. Pour contrevenir à ce problème, nous avons décomposés les images dans une familles de modes basés sur les fonctions de Krawtchouk [1]. Cette transformation est analogue à une série de Fourier $2d$ de l'image. Les amplitudes des différents modes donne de l'information sur l'ensemble de l'image et sont les attributs choisis pour le modèle de forêt aléatoire.

Rotation/Augmentation des données : Il est attendu que la (vraie) fonction prenant les images en arguments et redonnant le charactère latin associé est invariante sous certaines transformations des images. En particulier, le recadrement et les rotations des images de faible amplitudes ne devraient pas affecter le charactère associé. Cette

(a) Exemple d'image



(b) Décomposition en mode de Krawtchouk

observation permet de générer de nouvelles données à partir des données initiales en appliquant ces transformations et en conservant la même cible. Nous avons utilisé cette augmentation des données pour le réseau de neurone convolutif. Elle a été réalisée avec la méthode *ImageDataGenerator* de la librairie *Tensorflow*.

Données de test : Les données de test ont le même format que les données d'entraînement, à l'exception qu'ils sont organisés en paires d'images et que l'attribut correspondant à la cible est absent. Ainsi, le même traitement a été appliqué à ces données pour chaque modèle. Notons aussi qu'en visualisant la première image de chaque paire composant les images de test, nous avons observé un bruit aléatoire dans la première colonne de pixels. Nous avons corrigé ce bruit en translatant l'image d'une colonne (voir Annexe B).

3 Algorithmes d'apprentissage

Forêt d'arbres décisionnels : Ce modèle est basé sur une approche ensembliste où l'apprentissage par la construction d'un arbre de décision est répétée N fois sur des sous-ensembles des données d'entraînement (*bootstrapping*), avec chaque fois un sous-ensemble aléatoire des attributs. Chaque arbre construit est conservé en mémoire et contribue aux prédictions faites par le modèle : étant donné une nouvelle donnée x, le modèle prédit la classe majoritaire parmi l'ensemble des prédictions de chaque arbre.

La construction d'un arbre est exécuté en déterminant la partition linéaire binaire de l'espace des attributs (i.e. *noeud)* maximisant le gain d'information, i.e. minimisant l'entropie conditionnelle associée à la distribution des données d'entraînement après ladite partition. Cette construction est itérative : chaque sous-espace d'une partition binaire est soumise à la recherche d'une nouvelle partition optimale pour celle-ci. Un arbre de décision est charactérisé par sa profondeur p, c.-à-d. le nombre de noeuds séparant l'espace total des attributs de ses composantes minimales après la partition engendrée par l'arbre. La construction d'un arbre cesse lorsque la partition totale divise parfaitement les données de chaque classe ou lorsqu'une profondeur maximale $p = p_{max}$ est atteinte.

Réseau de neurones convolutif (CNN) : Les réseaux de neurones sont formés par la composition de transformations linéaires et non-linéaires appliquées de manière consécutive aux données. À chaque étape les données existent dans un espace vectoriel dont la taille varie, et qui est représenté par une couche de neurones.

Parmi les transformations pouvant être utilisées, on compte les couches connectés (application linéaire), les couches d'activation (application d'une fonction non-linéaire sur chaque neurone), les couches de *pooling* (compression des données en regroupant plusieurs neurones en un) et les couches de *dropout* (retrait d'un sous-ensemble aléatoire de neurones). Les réseaux dit convolutifs utilise un type particulier de couches connecté où la transformation linéaire

possède la structure d'une convolution : la valeur des neurones de la couche suivante sont obtenus en appliquant une même transformation linéaire sur différents sous-ensemble de neurones voisins.

XGBoost : Cette approche est similaire aux forêts d'arbres décisionnels, puisqu'elle constitue également une méthode ensembliste basée sur la combinaison d'arbre de décision. Toutefois, plutôt que d'avoir une série d'arbres indépendants, cet algorithme utilise le *boosting*, c.-à-d. chaque nouvel arbre est entraîné pour corriger les erreurs de l'ensemble des arbres précédents. Le type de correction requise est déterminé en approximant à l'ordre 2 (méthode Newton-Raphson) la fonction de coût (*softmax* dans notre cas) à chaque itération. On peut associer à chacune de ces corrections un poids appelé *taux d'apprentissage* modulant son impact sur les prédictions finales du modèle.

4 Méthodologie

Division des données : Les modèles définis dans la section précédente possèdent de nombreux hyperparamètres affectant leur architecture et leur capactié d'apprentissage et de généralisation. Afin de mesurer ces capacités, nous avons divisé les données d'entraînement en deux ensembles : données d'entraînement et données de validation. Ces dernières représentent respectivement 80% et 20% des données avec une cible fournies. La division a été effectué de manière aléatoire en utilisant la fonction *train_test_split* de la librairie *sklearn*. Les données d'entraînement et leurs cibles sont fournies au modèle pendant la phase d'entraînement. Le taux de succès du modèle à prédire les cibles des données de validation servira ensuite à mesurer la capacité du modèle à prédire la classe de nouvelles images.

Forêt d'arbres décisionnels : Notre modèle de *Random Forest* a été implémenté de zéro à l'aide de la librairie *numpy* de python. Il basé sur l'implémentation du modèle d'apprentissage par arbre de décision utilisé dans la septième scéance de travaux pratiques du cours [5] et dépend de 4 hyperparamètres : le nombre d'arbres de décision N, la profondeur maximale de chaque arbre p_{max}, le nombre de points des données N_d et le nombre d'attributs N_a considérés par chaque arbre.

Afin de trouver les hyperparamètres offrant les meilleures performances sur les données de validation, nous avons entraîné le modèle pour différentes valeurs sur une grille d'hyperparamètres (*gridsearch*) et avons conservé celui donnant le meilleur taux de succès sur les données de validation. La grille est fournie à l'Annexe A.

Réseau de neurones convolutif (CNN) : Le réseau de neurone a été implémenté avec la méthode *Sequential* de la librairie *pytorch* [4]. Pour le choix d'architecture, nous avons utilisé une structure ayant déjà démontré ces capacités pour la reconnaissance d'image. En effet, nous avons légèrement modifié celle utilisée dans un travail disponible en ligne [3]. Cette dernière est composée en alternance de couches convolutives avec 32 filtres, des couches de *pooling* avec des noyaux 2×2 et des couches de *dropout*. Notre architecture finale est composée de :

(1) *Couche convolutive* : 32 filtres de noyau 3 ;

(2) *Couche d'activation* : *ReLU* ;

(3) *Batch Normalization* ;

(4) *Couche convolutive* : 32 filtres de noyau 3 ;

(5) *Couche d'activation* : *ReLU* ;

(6) *Batch Normalization* ;

(7) *Max Pooling* de 2×2 ;

(8) *Dropout* : ratio de 0.25 ;

(9) Répétition des couches (1) à (8) avec des convolutions à 64 filtres ;

(10) *Couche dense complétement connecté* : 512 neurones ;

(11) *Dropout* : ratio de 0.5 ;

(12) *Couche dense complétement connecté* : 24 neurones associés aux classes à distinguer.

Pour optimiser les paramètres du réseau, nous avons utilisé l'algorithme Adam [2] basé sur la descente par (moyenne mobile exponentielle) du gradient. Cet algorithme possède trois paramètres α, β_1 et β_2 relié au pas de la descente et aux paramètres dans la moyenne exponentielle prise sur le gradient. Nous avons utilisé les paramètres suggérés dans le papier [2], soit $\alpha = 0.001$, $\beta_1 = 0.9$ et $\beta_2 = 0.999$. Le modèle a été entraîné sur un nombre standard de 50 époques.

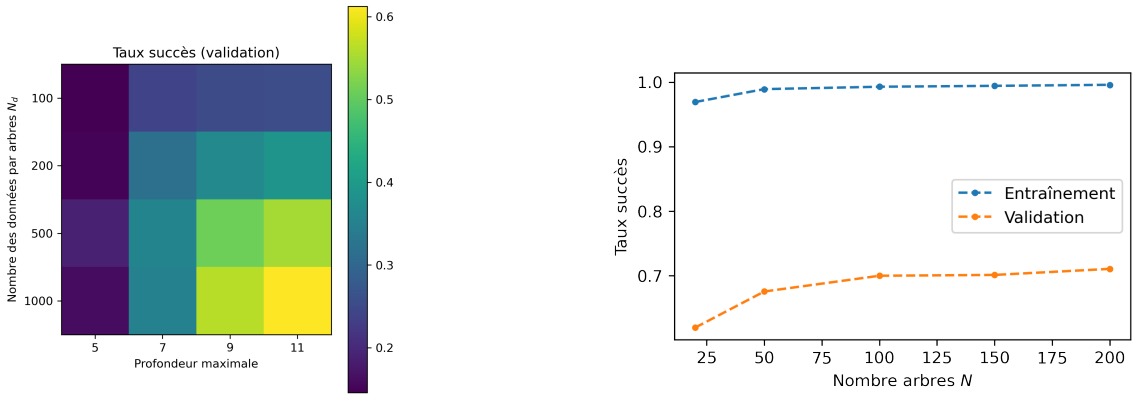
XGBoost : Ce modèle possède sa propre librairie python nommé *xgboost*. Il est donc implémenté en utilisant un objet *Dmatrix* de cette librairie et est entraîné en appelant la méthode *train*. Puisque ce modèle est basé sur l'assemblage d'arbre de décision, il est nécessaire déterminer l'hyperparamètre p_{max} associé à leur profondeur maximale. Puisque l'algorithme utilise le *boosting* et associe un poids à chaque arbre, il y a aussi un hyperparamètre α lié au taux d'apprentissage à fixer.

Afin de trouver les hyperparamètres optimaux, nous avons utilisé la même approche que pour la forêt d'arbre décisionnels : une recherche sur une grille d'hyperparamètres donnée dans l'Annexe A Cette grille a été choisie en considérant les valeurs proches des hyperparamètres par défaut du modèle XGBoost.

Manipulation ASCII : Les algorithmes implémentés associent aux images un nombre n entre 0-24 en correspondance avec les charactères latins A-Z. Pour transformer ce nombre en l'étiquette ASCII associé à la lettre, nous ajoutons 65, i.e. $n_{\text{ASCII}} = n + 65$. Ensuite, pour obtenir le charactère associé à une paire d'image, nous additionons leur nombre n_{ASCII} puis soustrayons 65 jusqu'à ce que le résultat soit dans l'intervalle $65 - 122$.

5 Résultats

Forêt d'arbres décisionnels : En entraînant notre modèle sur une grille d'hyperparamètres, nous avons trouvé que les performances sur les données d'entraînement et de validation convergent à un plateau vers $N = 100$ et augmentent de manière monotone avec la profondeur p_{max} et la taille du sous-échantillonage pour chaque arbre N_d (voir Figure 2). Cela suggère que la grille d'hyperparamètres considérée ne couvre pas le régime ou le modèle est en surentraînement.



(a) Taux de succès pour différents N_d et p_{max} avec 25 arbres

(b) Taux de succès en fonction du nombre d'arbres N avec $N_d = 1000$ et $p_{max} = 11$

FIGURE 2 – Taux de succès sur les données de validation pour différents choix d'hyperparamètres

Bien que cela inspire à étendre la grille de recherche pour obtenir de meilleures performances, cette option n'était pas concrétisable dû aux contraintes de temps de calcul. En effet, la complexité temporelle de l'entraînement augmente

avec le nombre d'arbres à entraîner, le nombre de couches à construire et la taille du sous-échantillonage (qui est associé au nombre de questions à investiguer à chaque noeud).

Au terme de l'optimisation des hyperparamètres, nous avons obtenu avec ($N = 100$, $p_{max} = 11$, $N_d = 1000$) un taux de succès de 99.6% sur les données d'entraînement, 71.0% sur les données de validation et 35.5% sur les données de test. Puisque le taux de succès sur les données de test correspond au succès à identifier une paire d'image, cela correspond à un taux de succès moyen d'environ 59% pour l'identification d'une image.

Cette performance est significativement inférieure à celle sur les données de validation et implique que les données de test et de validation ne sont pas issues de la même distribution. Cette différence est possiblement dû à l'existence de doublons dans les données fournies pour l'entraînement. En effet, plusieurs images des données d'entraînement semblent avoir été générées à partir de transformations mineures sur une même image.

Réseau de neurones convolutif (CNN) : Avec l'architecture sélectionnée, nous avons obtenu une performance quasi parfaite du modèle après seulement 5 époques. Au terme des 50 époques, le modèle a atteint une précision de 99.95% sur les données de test et n'a commit aucune erreur sur les données de validation (voir Figure 3).
Contrairement au modèle de forêt d'arbres décisionnels, ce modèle a aussi bien performé sur les données de test avec

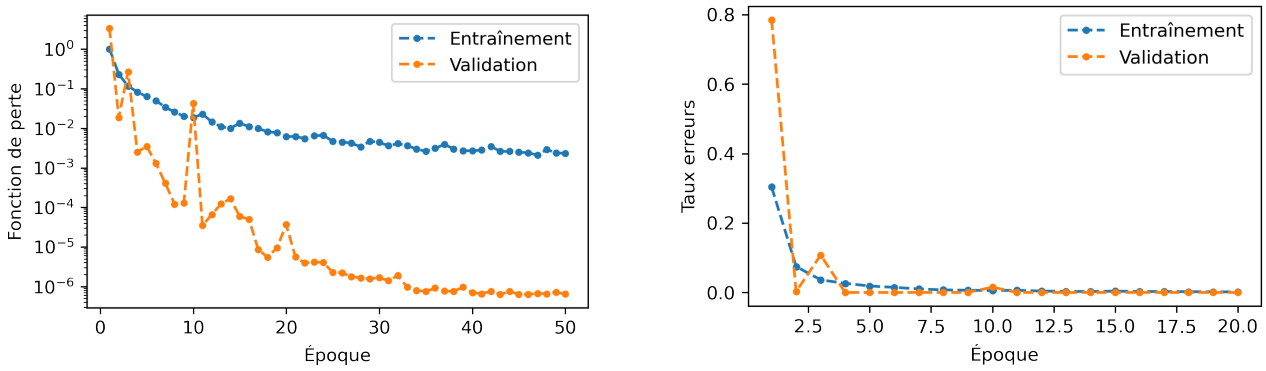


FIGURE 3 – Fonction de perte et taux d'erreurs d'entraînement et de validation en fonction du nombre d'époques d'entraînement, pour le réseau de neurones convolutif.

un taux de succès de 100%. On en déduit que sa capacité de généralisation est très supérieure à celle des arbres de décision. Elle est probablement dû à l'invariance sous translation et sous rotation encodé dans la structure convolutive du modèle et par l'augmentation des données discutée dans la section 2.

XGBoost : Pour le troisième modèle, la recherche sur la grille d'hyperparamètre a permis d'identifier qu'un taux d'apprentissage $\alpha \geq 0.1$ mène à un taux de succès de plus de 99% sur la validation (voir Figure 4). En particulier, le meilleur résultat de 99.54% a été obtenu avec une profondeur de $p_{max} = 7$ et un taux d'apprentissage de $\alpha = 0.1$. Pour ce choix d'hyperparamètre, le modèle a atteint une précision de 62.5% sur les données de test, et donc un taux de succès de 79.0% à l'identification d'une unique image. La divergence entre ce résultat et celui obtenu à la validation est similaire à la divergence observée pour la forêt d'arbre décisionnels. Elle est aussi possiblement dû à l'existence de doublons dans l'ensemble des données d'entraînement.

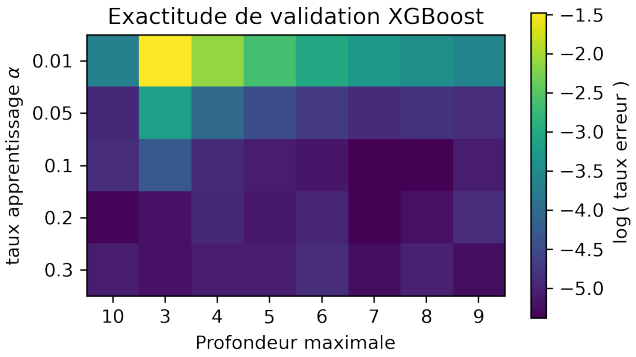


FIGURE 4 – log du taux d'erreurs de validation pour le modèle XGBoost pour différents taux d'apprentissage et profondeur maximales.

6 Discussion

Après l'implémentation et le test de trois modèles, il apparaît clair que le réseau de neurones convolutif offre les meilleures performances pour la classification d'images dû à sa très grande capacité de généralisation. Ce dernier a atteint une précision de 100% en validation et sur les données de test. En contraste, les deux autres modèles considérés et qui sont basés sur l'assemblage d'arbres de décision montraient des performances sur les données de test (35.5% et 62.5%) significativement inférieures à celle sur les données de validation. Puisque ces modèles n'ont pas été conçu en considérant l'invariance sous la translation/rotation en classification d'image, il semble que leur capacité à extrapoler pour des images issues d'une distribution légèrement différente demeure faible.

La force de notre approche a été de baser les prémisses de l'architecture de notre modèle de réseau de neurones sur un travail ayant déjà fait ses preuves en reconnaissance d'images [3]. Cela a permis d'obtenir rapidement un excellente performance sur les données de test, malgré qu'ils semblent différer des données de validation.

Pour améliorer notre approche et augmenter les performances de la forêt d'arbres décisionnels et du modèle XGBoost, il serait essentiel d'étudier les distributions des données d'entraînement et de test afin de comprendre entre les taux de succès pour chacun. Une avenue possible serait d'identifier les groupes d'images similaires dans les données d'entraînement et éviter de les séparer dans la division entraîenement/validation. Cela aurait le potentiel de réduire le biais dû aux doublons. Il serait aussi intéressant d'étudier l'impact de l'augmentation de données sur les modèles à base d'arbres de décision. En effet, le fait que nous avons uniquement implémenté l'augmentation pour le réseau de neurone est une faiblesse importante de notre approche.

Finalement, un travail supplémentaire aurait pu être réalisé pour déterminer le choix d'attributs pour le modèle XGBoost et la forêt d'arbres décisionnels. En effet, notre analyse s'est limité à un unique type d'attributs pour chaque modèle (l'intensité des pixels pour XGBoost et l'intensité des modes de Krawtchouk pour la forêt d'arbres décisionnels). Or, il est possible qu'un autre choix d'attributs basé sur l'invariance des cibles sous les rotations/translations des images ait mené à de meilleures performances.

Déclaration des contributions : Nous déclarons par la présente que tout le travail présenté dans ce rapport est celui des auteurs.

Références

[1] Roelof Koekoek et Rene F Swarttouw. "The Askey-scheme of hypergeometric orthogonal polynomials and its q-analogue". In : *arXiv preprint math/9602214* (1996).

[2] Diederik P Kingma et Jimmy Ba. "Adam : A method for stochastic optimization". In : *arXiv preprint arXiv :1412.6980* (2014).

[3] brendanartley. *Medium-Article-Code*. `https://github.com/brendanartley/Medium-Article-Code/blob/main/code/mnist-keras-cnn-99-6.ipynb`. 2022.

[4] *PyTorch : Sequential*. `https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html`. Accessed : 2023-12-07.

[5] *TP7 : Arbres de décision*. `https://colab.research.google.com/drive/1djfzz7iRIVeIT7WtDZ2OXgOPaEtL1k55`. Accessed : 2023-12-07.

A Grilles d'hyperparamètres

Pour l'optimisation du modèle de forêt d'arbres décisionnels, nous avons considéré la grille d'hyperparamètres définie par

$$N \in \{10, 25, 50, 100, 150, 200\}, \quad N_a \in \{15\}, \tag{1}$$

$$p_{max} \in \{5, 7, 9, 11\}, \quad N_d \in \{100, 200, 500, 1000\}, \tag{2}$$

où N, N_a, p_{max} et N_d sont respectivement le nombre d'arbres, le nombre d'attributs par arbre, la profondeur maximale et le taille du sous-échantillon pour chaque arbre. Pour le modèle XGBoost, nous avons considérés la grille d'hyperparamètres définie par

$$\alpha \in \{0.01, 0.05, 0.1, 0.2, 0.3\}, \quad p_{max} \in \{3, 4, 5, 6, 7, 8, 9, 10\}, \tag{3}$$

où α et p_{max} représentent respectivement le taux d'apprentissage et la profondeur des arbres.

B Translation de l'image 1 des données de test

Nous avons observé que la première rangée de pixel de la première image dans les paires d'image de test semblait avoir un problème. Nous avons corrigé ce bruit en translatant l'image de 1 pixel vers la gauche. Un exemple est illustré ci-dessous dans la Figure 5.

(a) Avant la correction (b) Après la correction

Figure 5 – Example de la première image de l'ensemble des données de text.