Université de Montréal

Université de Montréal

Compétition 2 - ASCII Sign Language

Tania MENDES DIAS - 20278092

Alexia PREVOT - 20278091

Kaggle usernames : taniamendesdias et alex prvt

11 décembre 2023

Table des matières

Introduction

Ce rapport aborde le défi de classification des images du langage des signes dans le cadre de la compétition Sign Language MNIST. Notre objectif est de développer des modèles de classification, notamment un Random Forest élaboré à partir de zéro, un réseau de neurones convolutionnel (CNN), ainsi que d'autres modèles. Cette compétition offre une occasion unique d'appliquer nos connaissances à un problème concret de reconnaissance lettres du langage des signes. Ce rapport présente nos approches, nos résultats et les enseignements tirés de cette expérience enrichissante.

1 Conception des caractéristiques

Dans cette section, nous décrivons les étapes de prétraitement appliquées aux données du jeu de données Sign Language MNIST, composé de 784 pixels par image.

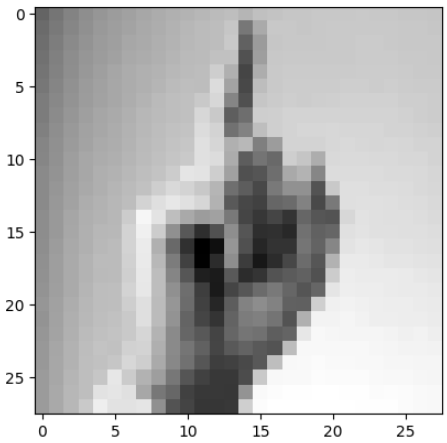

FIGURE 1 – Exemple : première image du jeu d'entraînement

1.1 Normalisation des Données

La première étape du prétraitement a consisté à normaliser les données, facilitant la convergence des modèles lors de l'entraînement.

1.2 Décalage des Labels

Afin de simplifier l'application et l'évaluation des modèles, nous avons décalé les labels à partir de 9 puisqu'il n'y a pas les lettres J et Z, correspondant aux labels 9 et 25. Ainsi, les labels vont de 0 à 24.

1.3 Images pour le CNN

Pour le modèle CNN, nous avons reformé les images en utilisant la fonction .reshape(28, 28). Cette étape était cruciale pour présenter les données sous une forme adaptée à l'architecture du CNN. En reformant les images en matrices 28x28, nous avons préservé la

structure spatiale des pixels, permettant au CNN de capturer les relations locales dans les données.

1.4 Sélection des Attributs

Concernant la sélection des attributs, aucune réduction dimensionnelle n'a été appliquée, et tous les 784 pixels ont été conservés. Chaque pixel a été préservé comme attribut afin de capturer au mieux l'information riche présente dans chaque image.

2 Algorithmes

Dans cette section, nous présentons un aperçu des algorithmes d'apprentissage utilisés pour aborder le problème de classification des images du langage des signes. Nous avons exploré une variété d'approches, comprenant la mise en œuvre d'un Random Forest from scratch, un modèle CNN personnalisé, un modèle XGBoost, ainsi qu'un modèle AdaBoost à l'aide de bibliothèques spécialisées.

2.1 Random Forest From Scratch

Pour le Random Forest, nous avons élaboré une implémentation personnalisée en créant des classes pour les nœuds, les arbres de décision et enfin l'ensemble du Random Forest. Chaque arbre de décision a été formé sur un sous-ensemble aléatoire des données d'entraînement, et les prédictions finales ont été agrégées par un vote majoritaire.

2.2 Convolutional Neural Network (CNN)

Le modèle CNN a été construit en utilisant l'architecture classique comprenant deux couches de convolution suivies de max-pooling, puis deux couches entièrement connectées. Le CNN a été formé sur les images reformées en matrices 28x28. La fonction d'activation ReLU a été utilisée pour introduire de la non-linéarité.

2.3 XGBoost et AdaBoost

En complément, nous avons exploré deux approches ensemblistes : XGBoost et AdaBoost. `XGBoost` est une bibliothèque d'implémentation de boosting qui utilise des arbres de décision en tant qu'estimateurs de base. Nous avons utilisé ses paramètres par défaut. Pour AdaBoost, nous avons utilisé la bibliothèque `scikit-learn`, exploitant des classifieurs faibles successifs pour renforcer la performance du modèle global.

3 Méthodologie

3.1 Division Entraînement/Validation :

Nous avons séparé le jeu de données d'entraînement en deux jeux distincts : un jeu de train et un jeu de validation. Pour ce faire, nous avons utilisé "StratifiedShuffleSplit" afin

de conserver la proportion des classes et de mélanger les données.

3.2 Random Forest

Le modèle de Random Forest codé à la main à partir des bibliothèques `numpy` et `pandas` est malheureusement beaucoup trop long à entraîner et donne des résultats très mauvais (environ 0.20 de précision). Nous avons tenté d'optimiser les calculs en utilisant sparse matrix mais cela n'a pas changer énormément.
Nous avons alors utilisé la bibliothèque `sklearn` et entraîné une forêt aléatoire sur notre jeu de données en vue d'une comparaison avec notre modèle. Les résultats étant bien meilleurs (environ 0.99 de précision), et le temps de calcul bien plus raisonnable (<30min), nous avons décidé d'utiliser le random forest de cette bibliothèque pour la suite. C'est donc sur ce modèle que nous avons tenté d'améliorer nos résultats et que nous avons comparé aux autres modèles.

Cross validation :

Nous avons utilisé une approche de validation croisée Stratified K-Fold avec 5 plis. Cette approche garantit que chaque pli contient une distribution équilibrée des différentes classes de notre ensemble de données, assurant ainsi une représentation adéquate de chaque classe dans les ensembles d'entraînement et de validation.

Optimisation des hyperparamètres :

Nous avons effectué une Grid Search pour trouver les hyperparamètres optimaux qui contribuent à la régularisation du modèle. Les paramètres tels que le nombre d'estimateurs (n_estimators), la profondeur maximale (max_depth), et le nombre minimal d'échantillons pour la division (min_samples_split) ont été optimisés.

3.3 CNN

Nous avons utilisé la bibliothèque `Albumentations` pour appliquer des transformations d'augmentation d'images, augmentant ainsi la variabilité des données.
Nous n'avons pas pu exploré d'autres architectures que celle vu plus haut.

3.4 XGBoost

Cross validation :

Comme pour le random forest, nous avons adopté une approche de validation croisée Stratified K-Fold avec 5 plis pour diviser notre jeu de données en ensemble d'entraînement et de validation. Cette méthode assure une distribution équilibrée des différentes classes dans chaque pli, garantissant ainsi une représentation adéquate de chaque classe dans les ensembles d'entraînement et de validation.

Optimisation des hyperparamètres :

Pour trouver les hyperparamètres optimaux régularisant le modèle XGBoost, nous avons réalisé une recherche aléatoire (Random Search). Les paramètres tels que le taux

d'apprentissage (learning rate), la profondeur maximale de l'arbre (max_depth), et le nombre minimum d'échantillons pour la division (min_child_weight) ont été sujets à cette recherche.

3.5 Évaluation des modèles :

Pour chaque modèle, le meilleur a été entraîné sur l'ensemble des données d'entraînement et évalué sur l'ensemble de validation. Les métriques d'évaluation incluent l'accuracy et une matrice de confusion. Enfin, pour la soumission, le modèle est réentraîné sur l'ensemble de données complet.

3.6 Données équilibrées

Nous avions remarqué qu'il n'y a pas le même nombre d'images dans le jeu d'entraînement pour représenter chaque lettre de l'alphabet (voir graphique en annexes). Nous avons fait plusieurs tests afin de remédier à ce problème. D'abord nous avons essayé de rééquilibrer le jeu de données en conservant le même nombre de données par classes. Puis, en faisant quelques recherches, nous avons trouvé un autre jeu de données et il s'avère que la répartition de ce nouveau jeu permettait, en l'assemblant avec le jeu d'entraînement, d'avoir un nouveau jeu de données équilibré, tout en ayant gagné en nombre de données (voir en annexes le deuxième graphique sur les proportions). Nous avons donc recommencé le cheminement précédent avec ce jeu complet.

4 Résultats

Pour commencer, nous présentons ici des résultats des modèles avec des hyperparamètres différents. Nous regardons l'accuracy et le temps de calcul afin d'avoir une idée globale de la performance du modèle.

L'optimisation des paramètres s'est déroulé comme dit dans une partie précédente avec des grilles ; ici il n'y a donc qu'une partie des résultats de l'optimisation des hyperparamètres. C'est juste pour avoir un apperçu de leurs impactes. Nous avons aussi présenté que les cas du Random Forest et de XGBoost (mêmes constats avec Adaboost).

	Accuracy	Temps de calcul (en s)
max_depth = 5, n_estimator = 20	0.937	4
max_depth = 10, n_estimator = 20	0.54	16
max_depth = 10, n_estimator = 200	0.976	94
max_depth = 50, n_estimators = 200	0.998	1342

TABLE 1 – Comparaisons des hyperparamètres pour le **Random Forest**

	Accuracy	Temps de calcul (en s)
max_depth = 5, n_estimator = 20	0.78	53
max_depth = 10, n_estimator = 20	0.934	125
max_depth = 10, n_estimator = 200	0.976	1484
max_depth = 30, n_estimators = 200	0.978	1694

TABLE 2 – Comparaisons des hyperparamètres pour **XGBoost**

Pour le Random Forest et le modèle XGBoost, l'augmentation de n_estimators (nombre d'arbres) entraîne une amélioration des performances, au détriment d'un temps de calcul plus long. Cela s'explique par le fait que davantage d'arbres permettent généralement une meilleure généralisation et une amélioration des résultats. Cependant, il faut considérer le compromis entre la précision et le temps de calcul.
Dans le cas du Random Forest, le constat est le même : des arbres plus profonds améliorent la performance, mais entraînent une augmentation du temps de calcul.
En ce qui concerne XGBoost, bien que présentant des performances similaires, son temps de calcul reste généralement plus élevé que celui du Random Forest, en raison de la complexité de son ensemble de modèles.

Par ailleurs, les résultats suivants sont ceux obtenus avec nos meilleurs modèles trouvés avec les précédentes démarches, et l'évaluation se fait sur des données de validation. Par exemple, pour le Random Forest, nous avons pris $n\_estimators = 200$ et pour le XGBoost, aussi avec en plus une profondeur maximale de 30. Nous avons voulu présenté ces résultats finaux tout en comparant la différence en ayant utilisé uniquement le jeu d'entraînement ou toutes les données trouvées.

	Jeu d'entraînement	Jeu d'entraînement + new
RandomForest	0.998	0.998
CNN	1.0	1.0
XGBoost	0.978	0.981
Adaboost	0.188	0.445

TABLE 3 – Accuracy sur les différents modèles

Les résultats obtenus révèlent les performances des modèles sur différents ensembles de données :
- Random Forest : le modèle atteint une accuracy de 0.998, démontrant une forte capacité de généralisation.
- CNN : il affiche une performance parfaite (accuracy de 1.0) sur les deux ensembles, ayant ainsi les meilleurs résutats parmis tous les modèles.
- XGBoost : il présente une petite amélioration avec l'ajout de nouvelles données (accuracy de 0.978 puis 0.981).
- Adaboost : ce modèle affiche des performances relativement faibles avec une accuracy de 0.188 sur l'ensemble initial et de 0.445 sur l'ensemble étendu. Ces résultats soulignent les limites d'Adaboost pour cette tâche (nombre de caractéristiques élevé puisqu'il s'agit d'images).

En résumé, ces observations montre la puissance des architectures profondes comme le CNN, la robustesse des modèles ensemblistes comme Random Forest, et la sensibilité de certains modèles aux caractéristiques et à la diversité des données.

Il est à noter que nous avons ré-entraîné nos meilleurs modèles sur tout le jeu de données avant de soumettre sur Kaggle. Nous avons donc réussi à obtenir une accuracy de 1 pour cette compétition.

5 Discussion

Nous avons fait de notre mieux dans le temps imparti pour répondre au problème de classification des images du langage des signes dans le cadre de cette compétition. Cependant, nous sommes conscientes de notre démarche et de ses faiblesses, et allons voir ici des pistes d'amélioration.

Tout d'abord, l'implémentation personnalisée du Random Forest a souffert de problèmes de performance sur ce jeu de données (moins sur d'autres jeux plus simple comme le dataset iris) et de temps de calcul, conduisant à l'adoption du modèle de scikit-learn. Certainement que des changement pourraient être fait afin d'avoir un modèle utilisable. Cependant, son implémentation personnalisé a permis une compréhension encore plus approfondie du fonctionnement de cet algorithme.
Par ailleurs, l'utilisation de la validation croisée Stratified K-Fold garantit une représentation équilibrée des classes dans les ensembles d'entraînement et de validation.

Idées d'amélioration :
- Tester des architectures plus complexes pour le CNN ou des modèles pré-entraînés pour extraire des caractéristiques plus riches.
- Explorer d'autres méthodes ensemblistes pour combiner les forces de différents modèles.
- Rechercher des méthodes d'optimisation du temps de calcul pour les modèles gourmands en ressources, par exemple, parallélisation ou accélération matérielle (notamment pour le Random Forest from scratch).
- Expérimenter avec d'autres techniques d'augmentation de données pour avoir plus de variabilité et améliorer la généralisation.

En **conclusion**, bien que notre approche ait donné des résultats solides, il existe toujours des opportunités d'amélioration, notamment en explorant de nouvelles architectures, en optimisant les méthodes d'ensemble learning, et en mettant l'accent sur des stratégies d'augmentation de données plus avancées.
Il est aussi important de noter que la taille des données aurait pu être plus grande et dans ce cas les résultats et conclusions pourraient différer pour certains modèles.

6 Déclaration des contributions

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

Annexes

Le projet a bien entendu commencé par une analyse des données. Nous avons visualisé les images et cherché à voir la proportion de chaque lettre dans le jeu d'entraînement.

FIGURE 2 – 10 premières images pour chaque lettre de l'alphabet

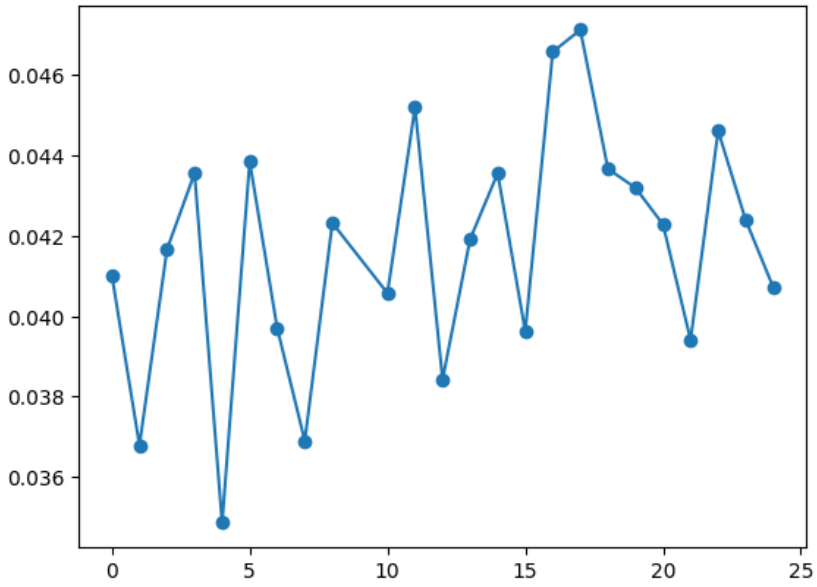

FIGURE 3 – Proportion de chaque lettre dans le jeu d'entraînement

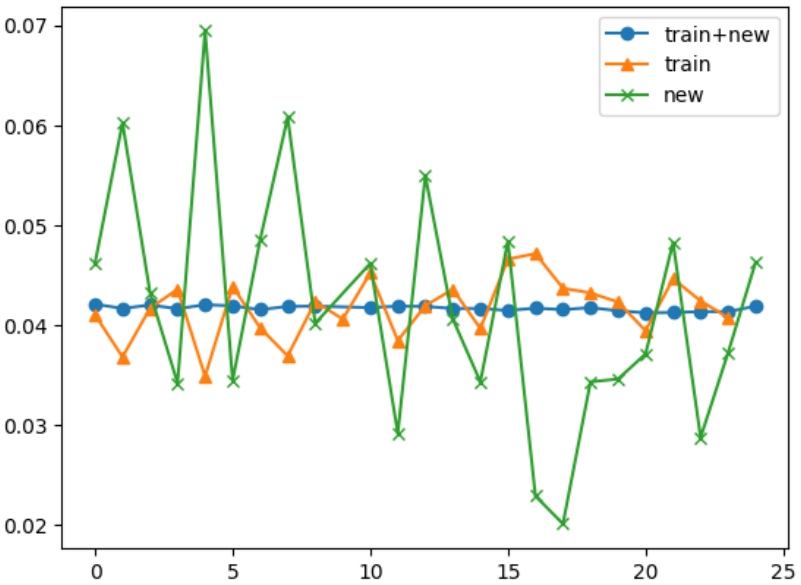

FIGURE 4 – Proportion de chaque lettre dans les différents jeux

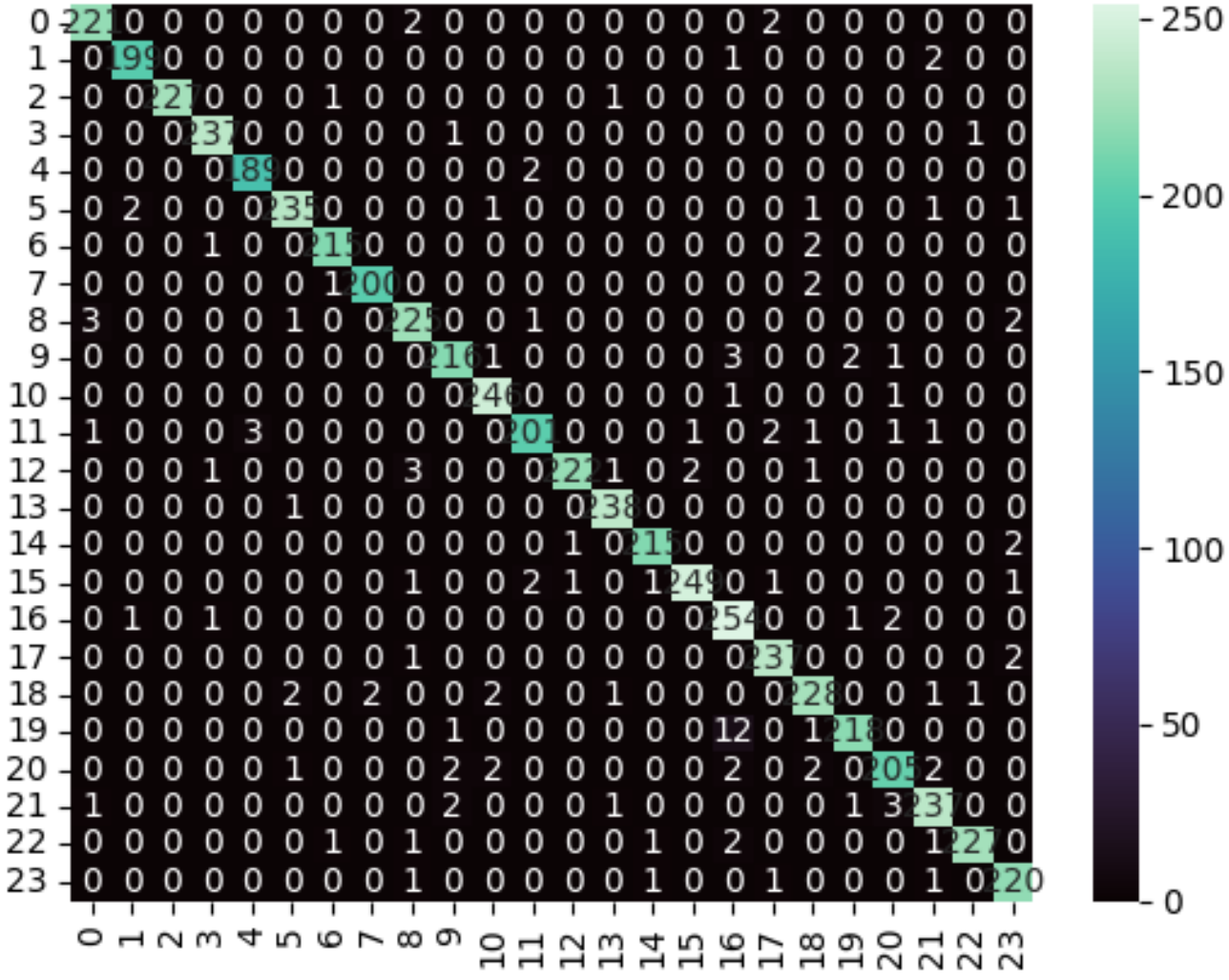

FIGURE 5 – Exemple d'une matrice de confusion pour le meilleur modèle XGBoost

Références

- Compétition Kaggle

- Modèles d'Hugging Face

- Autre compétiton Kaggle

- Inspiration Random Forest