Université Paris Dauphine - PSL
Master 1 Informatique, Données, Décision
Année universitaire : 2023/2024

SIGNA VERSE

“ Libérez la communication sans frontières :
découvrez notre solution de traduction des signes vers le langage parlé !”

Projet Machine Learning

Nil-Dany Mostefai
Yani LHADJ

Enseignant : Florian YGER

Mai 2024

Sommaire

1 Introduction :Pourquoi avoir choisi cette tâche?

Nous souhaitions dès le début développer une application utilitaire véritablement pratique pour l'utilisateur.

Le choix de la thématique du projet ne fut pas simple. Notre première idée était de créer une application capable de déterminer l'état émotionnel d'une personne en analysant ses expressions faciales. Cependant, nous avons réalisé que l'interprétation des expressions faciales était une tâche complexe, même sans l'utilisation d'algorithmes spécifiques. Nous avons donc décidé d'abandonner cette idée.

En parcourant les différents ensembles de données disponibles sur Kaggle, nous sommes tombés sur des données particulièrement intéressantes : un ensemble de données représentant des photos de mains effectuant des signes de la langue des signes. C'est à partir de là qu'est née notre idée de développer un traducteur de la langue des signes.

2 Tâche

Le projet consiste à concevoir une application web qui permettra aux utilisateurs de capturer des vidéos de leurs gestes en utilisant la caméra de leur appareil. Une fois la vidéo capturée, celle-ci sera divisée en plusieurs images. Ensuite, chaque signe apparaissant dans chaque image sera traduit. Ainsi, une succession de signes donnera une succession de lettres, formant ainsi un mot. L'application traduira automatiquement ces mots en texte dans les langues cibles, à savoir le français, l'anglais et l'arabe. Ce système utilisera des algorithmes de machine learning pour détecter et interpréter les différents signes.
À l'origine, notre ambition était de traduire les signes en temps réel. Cependant, nous avons rapidement réalisé que cette tâche était bien trop complexe compte tenu du laps de temps limité dont nous disposions.

3 Description du jeu de données

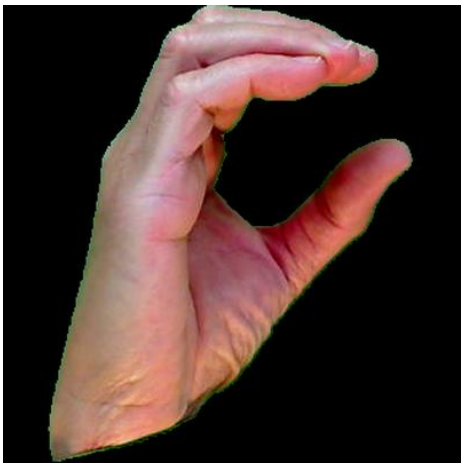
3.1 Origine du jeu de données

Le jeu de données a été obtenu à partir de Kaggle, où il est présenté sous la forme d'un dossier divisé en 36 sous-dossiers. Chaque sous-dossier est nommé selon un chiffre de 0 à 9 ou une lettre de l'alphabet de A à Z. À l'intérieur de chaque sous-dossier, on trouve exactement 70 photos représentant une main effectuant un signe

de la langue des signes correspondant au nom du dossier.
Lien vers le jeu de données sur Kaggle :
American Sign Language Dataset (kaggle.com)

3.2 Raisons du choix de ce jeu de données

Ce jeu de données a été sélectionné pour sa richesse en données, comprenant un total de 2520 images. De plus, sa diversité est un atout, car les images présentent les signes réalisés sur différents ongles, ce qui ajoute une variabilité pertinente pour la reconnaissance des gestes.



3.3 Séparation des données d'entraînement et de test

Pour diviser les données en ensembles d'entraînement et de test, nous avons initié le processus en mélangeant l'intégralité du jeu de données. Ensuite, un pourcentage approprié du jeu de données mélangé a été extrait pour constituer l'ensemble de test, tandis que le reste des données a été réservé pour l'ensemble d'entraînement. Cette méthode de séparation assure que les deux ensembles sont représentatifs de l'ensemble des données disponibles, favorisant ainsi la généralisation du modèle. De plus le data set trouvé sur kaggle fourni un ensemble de données de test.

3.4 Difficultés rencontrées

Nous avions initialement envisagé que notre jeu de données était suffisamment varié pour entraîner notre modèle de manière efficace. Cependant, au fur et à mesure du développement de notre projet, nous avons réalisé que cette diversité n'était pas aussi présente que nous l'avions espéré. En effet, la présence systématique d'un fond noir dans toutes les images posait problème lors de la traduction des signes, notamment lorsque les images de test ne présentaient pas ce même fond noir.

Pour résoudre cette problématique, nous avons exploré différentes approches. Dans un premier temps, nous avons envisagé d'adapter chaque image avant de la passer au programme en détectant la main, la découpant et la plaçant sur un fond noir. Cependant, nous avons rapidement abandonné cette idée en raison de sa complexité et de son inefficacité, car ce processus s'avérait être à la fois coûteux en termes de temps de calcul et peu fiable en termes de résultats.

Pour pallier cette lacune dans notre jeu de données, nous avons pris la décision d'enrichir celui-ci en y ajoutant de nouveaux échantillons.

Nous avons décidé d'enrichir notre jeu de données initial en le fusionnant avec un autre ensembles de données disponibles sur Kaggle : l'ensemble de données ASL (American Sign Language) Alphabet Dataset (kaggle.com)

Le jeu de données "ASL Alphabet" est composé de 29 classes, incluant les 26 lettres de l'alphabet de A à Z, ainsi que trois autres classes pour ESPACE, SUPPRIMER et RIEN. Ces nouvelles données enrichissent la variabilité de notre ensemble de données et améliorent la performance de notre modèle. Cependant, en raison de contraintes de mémoire, nous avons choisi de limiter notre utilisation de ces nouveaux ensembles de données. Nous avons conservé environ 3 à 5 pourcent du deuxième ensemble de données.

4 Méthodologies

4.1 Étapes de prétraitement utilisées

Les données dont nous disposons sont des images, et chaque donnée est représentée par une matrice de trois dimensions (la hauteur de l'image, la largeur de l'image et le nombre de canaux). Étant donné que les algorithmes étudiés en cours ne peuvent prendre que des données sous forme de vecteur en entrée, un prétraitement des données était donc nécessaire.

Le processus de réduction des dimensions n'est pas une tâche facile. Quelles variables devraient être supprimées ? Comment cette suppression affecterait-elle la qualité des prédictions ? Ces questions sont complexes et souvent sans réponse claire. Face à cette complexité, nous avons donc décidé de nous orienter vers la seconde option, soit l'utilisation d'un réseau de neurones.

Initialement, nous avons décidé de construire notre propre réseau de neurones convolutionnel, à plusieurs couches. Pour cela, nous avons utilisé la bibliothèque Keras. Cependant, les résultats obtenus n'étant pas concluants, nous nous sommes tournés vers l'utilisation d'un réseau de neurones pré-entraîné ResNet. Cette architecture est largement populaire dans les tâches de classification d'images, ce qui nous a semblé être la solution la plus adéquate pour notre problématique.

Pour pouvoir réaliser ce processus, nous avons implémenté une fonction appelée `prepare_images` qui pré-traite les images du dataset. Cette fonction prend en entrée le dataset à traiter. Elle commence par créer un réseau de neurones pré-entraîné, ensuite, chaque image du dataset est redimensionnée et normalisée selon les valeurs spécifiées pour ImageNet. Enfin, les embeddings sont calculés pour chaque image à l'aide du réseau de neurones, et ces embeddings sont stockés dans une liste. La fonction renvoie un tableau numpy contenant tous les embeddings calculés pour les images du dataset.

Les embeddings résultants du passage des images à travers le réseau de neurones, forment notre nouveau jeu de données.

Soit X une image représentée par une matrice de pixels.

1. La moyenne μ des valeurs des pixels de l'image est calculée pour chaque canal de couleur (rouge, vert, bleu, etc.).

$$\mu = \frac{1}{N} \sum_{i=1}^{N} X_i$$

2. L'écart type σ des valeurs des pixels de l'image est calculé pour chaque canal de couleur.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \mu)^2}$$

3. Les valeurs des pixels de l'image sont ensuite normalisées en soustrayant la moyenne μ et en divisant par l'écart type σ, pour chaque canal de couleur.

$$X_{\text{normalisé}} = \frac{X - \mu}{\sigma}$$

Pour chaque algorithme de classification utilisé, nous avons suivi une procédure standard. Tout d'abord, nous avons créé le modèle en utilisant la bibliothèque `sklearn.linear_model`. Ensuite, pour chaque ensemble de données - le jeu de données d'origine "American Sign Language Dataset (kaggle.com)" et le jeu de données enrichi après fusion avec le jeu de données "ASL (American Sign Language) Alphabet Dataset (kaggle.com)" - nous avons effectué les étapes suivantes

1. **Optimisation des hyperparamètres du modèle avec Grid Search :**
   Nous avons utilisé une recherche par grille (Grid Search) pour faire varier les hyperparamètres du modèle et trouver les combinaisons qui minimisent les déviations et maximisent l'accuracy. La recherche par grille consiste à définir une grille de valeurs pour les hyperparamètres à tester, puis à évaluer chaque combinaison de manière exhaustive. Si P représente l'ensemble des hyperparamètres à optimiser et $v_1, v_2, ..., v_n$ sont les valeurs possibles pour chaque hyperparamètre, la recherche par grille teste toutes les combinaisons possibles $(p_1, p_2, ..., p_n)$ où $p_i \in v_i$. Cela permet de trouver les valeurs d'hyperparamètres qui optimisent les performances du modèle.

   Pour chaque combinaison d'hyperparamètres, le Grid Search calcule la métrique de performance (accuracy, F1-score, etc.) sur l'ensemble de validation, en utilisant la fonction de score définie. Le but est de déterminer la combinaison d'hyperparamètres qui permet de maximiser cette métrique.

2. **Entraînement du modèle avec les hyperparamètres optimaux :**
   Une fois les hyperparamètres optimaux du modèle déterminés, nous avons entraîné le modèle en utilisant ces paramètres et l'ensemble de données d'entraînement.

3. **Évaluation du modèle avec une validation croisée :**
   Nous avons évalué la performance du modèle en utilisant une validation croisée.

La validation croisée est une technique qui consiste à diviser l'ensemble de données en k sous-ensembles (appelés plis), d'entraîner le modèle sur $k-1$ plis et de le tester sur le pli restant. Cette opération est répétée k fois, en utilisant chaque pli comme ensemble de test une fois, et les performances sont ensuite moyennées. Mathématiquement, si D est l'ensemble de données, la validation croisée divise D en $D_1, D_2, ..., D_k$ et évalue le modèle M en utilisant la fonction de score S sur chaque pli D_i. La performance moyenne du modèle sur tous les plis est ensuite calculée.

$$\text{Performance} = \frac{1}{k} \sum_{i=1}^{k} S(M, D_i)$$

En outre, nous évaluons le modèle en utilisant directement la fonction `score` du modèle, et cela sur les données test fournies dans le data set récupéré sur Kaggle.

5 Algorithmes utilisés et résultats obtenus

5.1 Régression Logistique Multinomiale

5.1.1 Modèle entraîné sur les données avant enrichissement

En appliquant une validation croisée avec $cv = 8$, nous obtenons les résultats présentés dans la figure 1.

Figure 1: Score de test résultant de la validation croisée

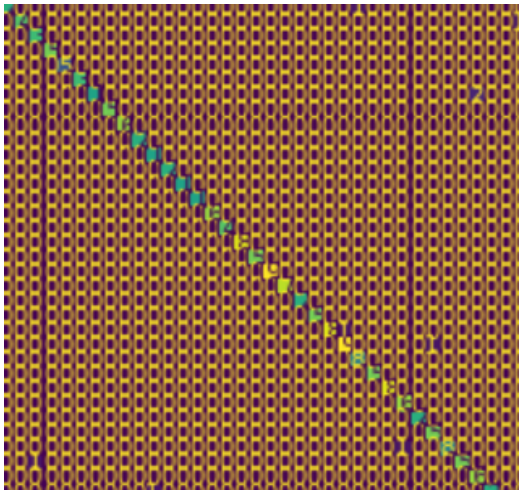


Figure 2: Matrice de confusion

L'accuracy moyenne est de 0.96. Cependant, l'écart-type est assez important. Cela pourrait indiquer un manque de robustesse du modèle, car nous observons des accuracies assez différentes en variant les ensembles d'apprentissage et de test lors de l'application de la validation croisée. Il est donc important de prendre en compte cette variation lors de l'évaluation de la performance du modèle.
En évaluant le modèle sur le jeu de données de test, nous obtenons une précision moyenne très basse de 0.05, comme illustré dans la figure 3.

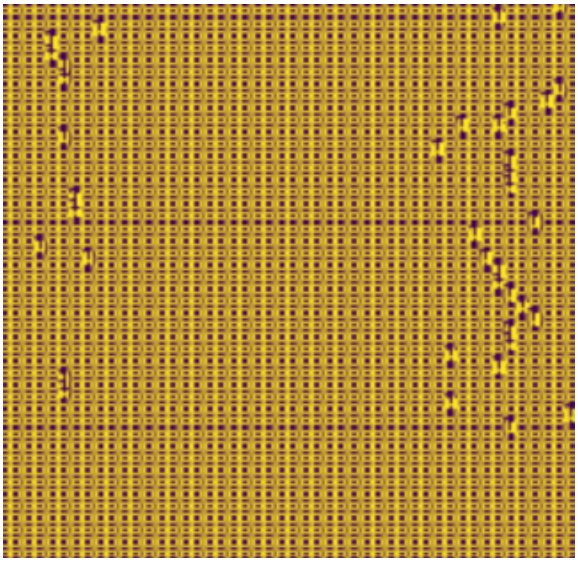


Figure 3: Matrice de confusion pour le jeu de données de test

5.1.2 Modèle entraîné sur les données enrichies

Après l'entraînement du modèle sur les données enrichies, nous avons constaté une précision moyenne de 0.93 lors de l'utilisation d'une validation croisée. Cette valeur est légèrement inférieure à celle obtenue avant l'enrichissement du dataset.
En revanche, lors de l'évaluation sur les images test fournies, nous avons observé une déviation beaucoup plus importante, avec une précision moyenne de 0.89.

5.2 SVM (Support Vector Machine)

5.2.1 Modèle entraîné sur les données avant enrichissement

En appliquant une validation croisée avec $cv = 10$, les résultats sont présentés dans la figure 5.

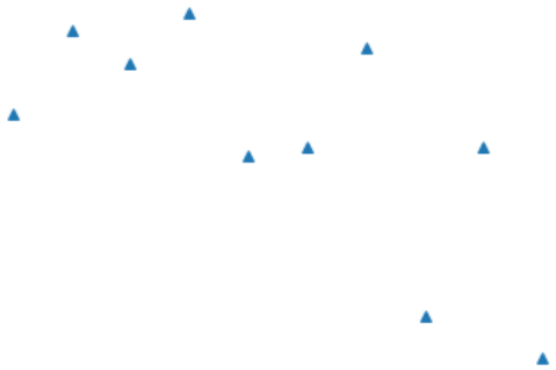


Figure 4: Score de test résultant de la validation croisée

L'accuracy moyenne est de 0.98, cependant, l'écart-type est assez important. L'évaluation de ce modèle sur le jeu de données de test est très décevante, avec une précision moyenne très faible de 0.05.

5.2.2 Modèle entraîné sur les données enrichies

Après l'entraînement du modèle sur les données enrichies, nous avons constaté une précision moyenne de 0.94 lors de l'utilisation d'une validation croisée.
Lors de l'évaluation sur les images test fournies, nous obtenons une précision moyenne de 0.9.

5.3 K plus proches voisins (KNN)

5.3.1 Modèle entraîné sur les données enrichies

Pour optimiser les performances de notre modèle, nous avons effectué une analyse approfondie en faisant varier le nombre de voisins k considérés dans l'algorithme k-NN.
Les résultats de cette analyse sont illustrés dans la figure 6.

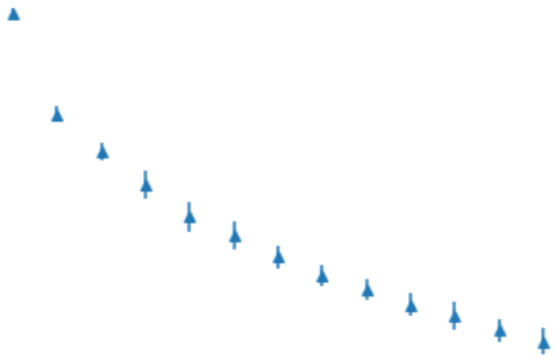


Figure 5: Accuracy en fonction du nombre de voisins k

Nous observons que la meilleure valeur d'accuracy est obtenue pour $k = 1$, ce qui signifie que le modèle ne considère qu'un seul voisin le plus proche pour effectuer sa prédiction, avec une accuracy de 0.96.
Lors de l'évaluation sur les images test fournies, nous obtenons une précision moyenne de 0.94.

5.4 Synthèse

Nos modèles étaient limités par le dataset utilisé initialement. En effet, notre dataset d'origine ne contenant que des images de mains découpées sur fond noir, introduisait nécessairement un biais lors de l'apprentissage des embeddings et, par conséquent, lors de l'apprentissage des modèles.
L'enrichissement de notre dataset a clairement amélioré les performances de nos modèles, les rendant beaucoup plus robustes. Le taux d'apprentissage correct sur l'ensemble de données test fourni a drastiquement augmenté après l'enrichissement du dataset utilisé pour l'entraînement, passant d'environ 0.05 à 0.94 pour les trois modèles.

Modèle	Avant enrichissement (validation croisée)	Après enrichissement (validation croisée)	Images test fournies
Régression Logistique	0.96	0.93	0.89
SVM	0.98	0.94	0.9
KNN	-	0.96	0.94

Table 1: Comparaison des performances des modèles

6 Visuel

Notre objectif initial était de créer une application capable de prendre une personne en vidéo qui parle en langue des signes et d'effectuer la traduction en temps réel. Nous avons donc développé une application web permettant de capturer une vidéo et de la traduire en utilisant l'un des trois modèles que nous avons entraînés : SVM, KNN ou Régression Logistique. Chaque modèle utilise ses hyperparamètres optimaux pour une meilleure performance.

Les visuels de cette application sont présentés ci-dessous :

Figure 6: Capture d'écran de l'application - Vue 1

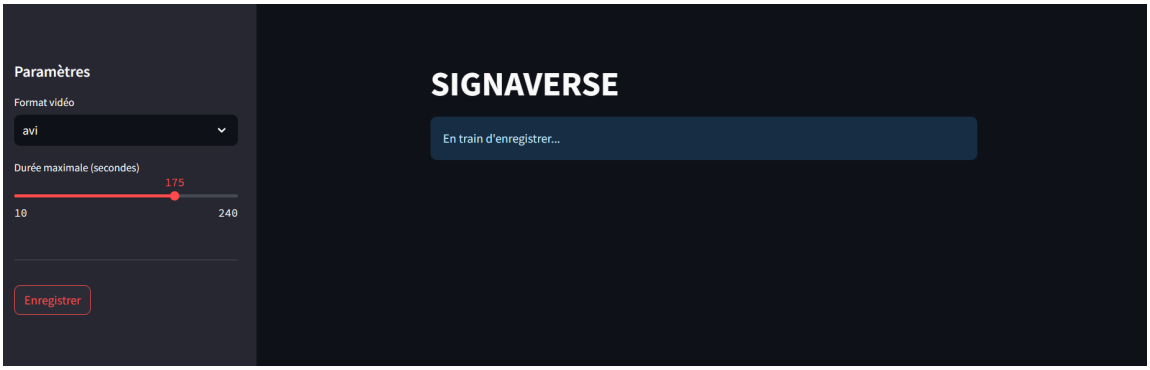


Figure 7: Capture d'écran de l'application - Vue 2

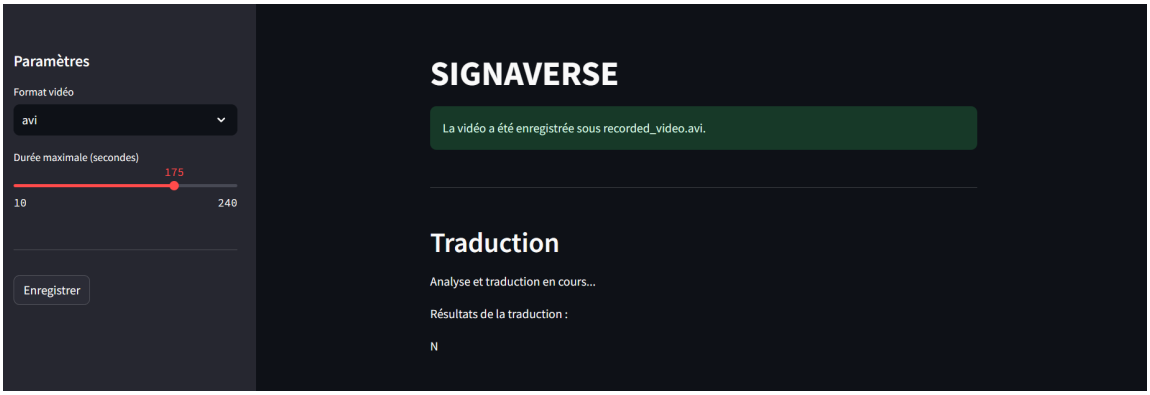


Figure 8: Capture d'écran de l'application - Vue 3

Comment ca marche ?
L'application vous permet d'enregistrer une vidéo, de l'enregistrer sur le répertoire courant. Divise la vidéo en images, passe les images au modèle de classification, traduit le signe sur l'image en lettre.

7 Conclusion

Dans le cadre de ce projet, nous avons développé une application visant à traduire la langue des signes à partir de vidéos capturées par les utilisateurs. Malgré nos efforts pour enrichir les données, optimiser les hyperparamètres et sélectionner les meilleurs modèles d'apprentissage automatique, nous n'avons pas atteint les résultats escomptés.

Les résultats obtenus ont montré des performances encourageantes lors de la validation croisée, notamment après l'enrichissement du jeu de données. Cependant, lors de l'évaluation sur des images prises par nos soin, la précision moyenne reste décevante pour tous les modèles, ce qui indique une difficulté à généraliser à de nouvelles données.

Une des principales difficultés rencontrées a été le manque de variabilité dans le jeu de données initial, avec des images de mains sur fond noir. Cela a introduit un biais dans l'apprentissage des modèles, affectant leur capacité à généraliser à de nouvelles conditions, notamment lorsque les images de test ne suivaient pas le même schéma.

Malheureusement, le temps limité dont nous disposions ne nous a pas permis d'explorer toutes les avenues pour résoudre ce problème. Nous aurions aimé avoir plus de temps pour investiguer plus en profondeur les raisons de cette difficulté de généralisation. Une piste aurait pu être d'explorer d'autres techniques de prétraitement des images ou de collecter un jeu de données plus varié et représentatif.

En résumé, bien que notre application représente une première étape vers la création d'un traducteur de langue des signes utilisant l'intelligence artificielle, il reste encore des défis à relever pour améliorer sa précision et sa fiabilité. Avec davantage de temps, nous aurions pu explorer ces défis plus en détail et développer des solutions plus efficaces.

Malgré ces obstacles, ce projet nous a permis d'acquérir une expérience précieuse dans le domaine de l'apprentissage automatique, et nous sommes reconnaissants d'avoir eu l'opportunité de travailler sur un sujet aussi stimulant et pertinent.