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

Dany MOSTEFAY
Yani LHADJ

Enseignant : Florian YGER

Mai 2024

Sommaire

1 Introduction :Pourquoi avoir choisi cette tâche?

Nous souhaitions dès le début développer une application applicative et 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 trop complexe. Nous avons donc décidé d'abandonner cette idée.

En parcourant les différents ensembles de données disponibles sur Kaggle, nous sommes tombés sur des jeux de d'images destinées à la classification des lettres dans la langue des signes. Nous avons donc décider 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 les lettres vers la langue écrite. Ce système utilisera des algorithmes de machine learning pour classfier les signes.

3 Description du jeu de données

3.1 Origine du jeu de données

Notre jeu de données original American Sign Language Dataset (kaggle.com) 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.

Nous avons également utilisé deux autres jeux de données dont nous parlerons plus en détail dans la partie Difficultés rencontrées:

- Le jeu de données Sign Language Digits Dataset structuré de la même manière que le dataset ci-dessus, mais ne contenant que des images de mains sur fond blanc signant les chiffres de 0 à 9.
- Le jeu de données ASL(American Sign Language) Alphabet Dataset structuré en 29 dossiers, 26 correspondant aux lettres de l'alphabet, et 3 correspondant à des labels spéciaux : nothing, del (delete) et space.

3.2 Raisons du choix de ce jeu de données

Ce jeu de données a été sélectionné car il possède un nombre de fichiers parfaitement adaptés, comprenant un total de 2520 images, ce qui nous permettra d'entraîner rapidement nos algorithmes. De plus, sa diversité en terme d'emplacements des mains est un atout, ce qui ajoute une variabilité pertinente pour la reconnaissance des gestes.

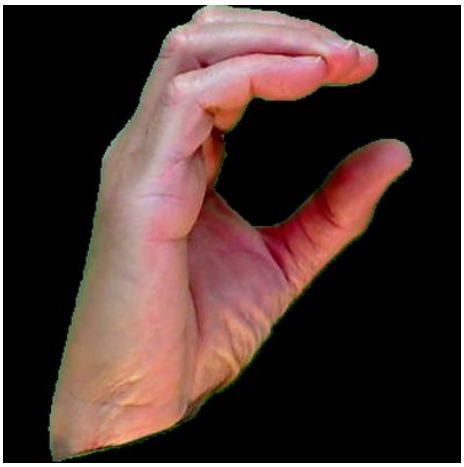

Figure 1: La main signe la lettre "C"

Figure 2: La main signe également la lettre "C" sous un angle et une position différents

3.3 Visualisation des données

Pour avoir une idée de la structure des données, et des relations qui existent entre elles, nous avons décidé d'opérer une réduction de dimension avec Umap après prétraitement des données.

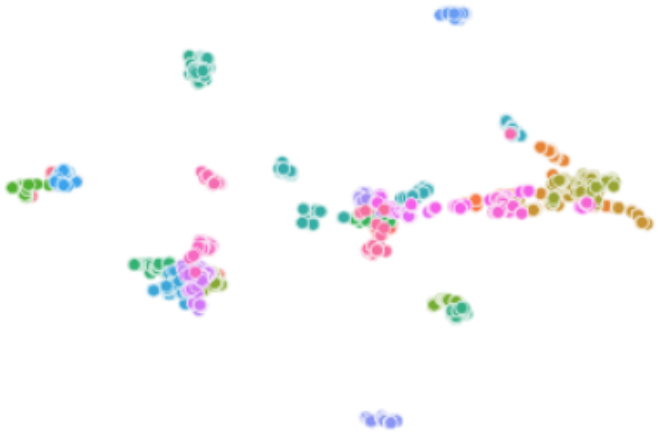

Figure 3: Visualisation des images en deux dimensions

Nous constatons que les clusters d'image sont relativement homogènes et assez bien séparées. Nous pouvons supposer que les clusters confondus entre eux correspondent à des signes qui se ressemblent, par exemple la lettre "O" avec le chiffre "0".

3.4 Difficultés rencontrées

Nous avions initialement envisagé que notre jeu de données était suffisamment varié pour entraîner notre modèle de manière efficace. Cependant, nous avons sous-estimé l'impact que pouvait avoir le fond noir systématique sur nos modèles d'apprentissage. En effet, cela posait problème lors de la traduction des signes, notamment lorsque les images de test ne présentaient pas ce même fond noir. Nos modèles étaient donc biaisés, et ne pouvaient traduire des signes que sur fond noir.

Pour résoudre cette problématique, nous avons exploré différentes approches. Dans un premier temps, nous avions envisagé d'adapter chaque image avant de la passer au modèle en détectant la main, puis 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 le processus était coûteux en temps de calcul et assez peu fiable dans ses performances.

Pour pallier cette lacune dans notre jeu de données, nous avons pris la décision de l'enrichir de deux manières, puis de conserver le dataset issu de la deuxième manière:

- En fusionnant avec les deux nouveaux jeux de données de chiffres et de lettres de l'alphabet. Etant donné que le nouveau dataset des lettres de l'alphabet pèse environ 4 Go, nous avons décider de le réduire d'environ 95 % pour pouvoir utiliser les données.

- Puis en fusionnant la moitié de notre dataset original avec uniquement le dataset ASL alphabet, en retirant les labels correpondant aux chiffres, et aux caractères spéciaux (nothing, del, space), ce qui représente notre dataset final, sur lequel nous allons entraîner nos modèles.

Nous avons tout d'abord opéré de la première manière avec l'espoir de pouvoir supprimer le biais de nos modèles sur nos fonds noirs. Si le biais a bien été supprimé pour les lettres de l'alphabet, il a néanmoins été conservé pour les images correspondant à des chiffres, la raison étant que nos images relatives aux chiffres possèdent seulement deux fonds : un fond blanc et un fond noir.

Figure 4: Image sur fond noir

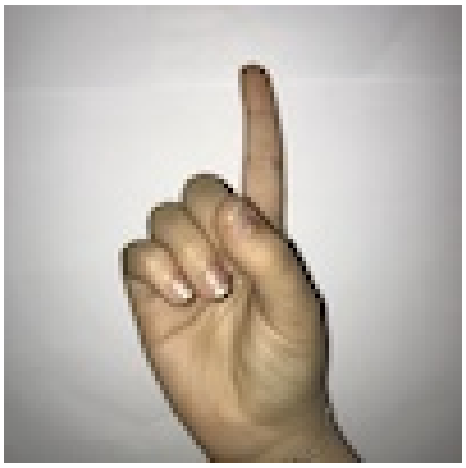

Figure 5: Image sur fond blanc

De plus, il se peut que le modèle ait également pu confondre les lettres et les chiffres, par exemple la lettre "O" et le chiffre "0" se ressemblent énormément.

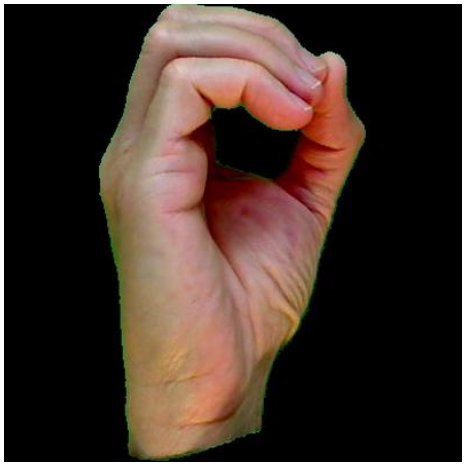

Figure 6: Chiffre 0

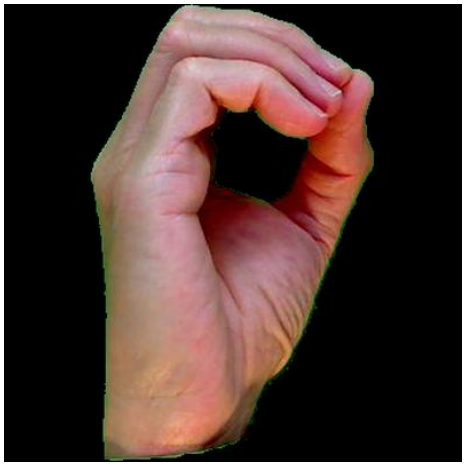

Figure 7: Lettre O

Nous avons donc décider de retirer les chiffres de nos labels, ainsi que les caractères spéciaux.

Nous avons un dataset riche et variable, qui peut s'avérer efficace pour généraliser nos modèles, car il possède des images de signes effectués dans des environnements et contextes différents (avec même l'apparition de visages).

Figure 8: Lettre A

Figure 9: Lettre A

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

Nous avons par la suite fait une séparation en ensembles d'entraînement et de tests pour entraîner nos algorthmes. Nous possédons également un jeu de "déploiement" décorrélé des deux jeux de données précédents, et contenant notamment quelques images prises par nos soins.

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`. Ensuite, 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.

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 soit une validation

croisée, soit une simple prédiction sur les jeux de test et de déploiement. 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.

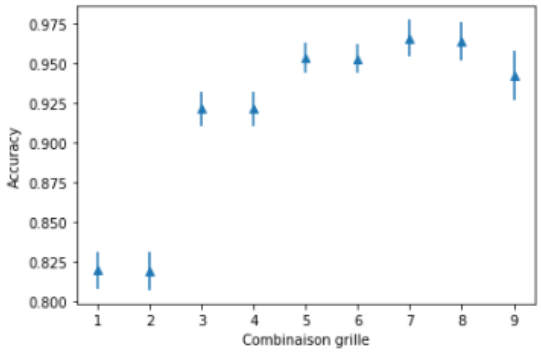

Figure 10: Score de test résultant de GridSearchCV

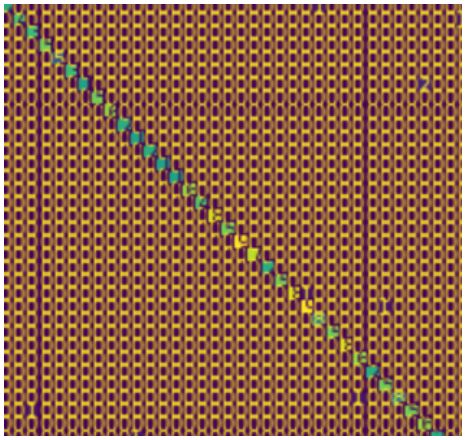

Figure 11: Matrice de confusion pour le jeu de données de test : une ligne représente un label réel, une colonne représente le label prédit

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.

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 0.84 lors de l'utilisation d'une validation croisée avec grid-Search. 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 0.86.

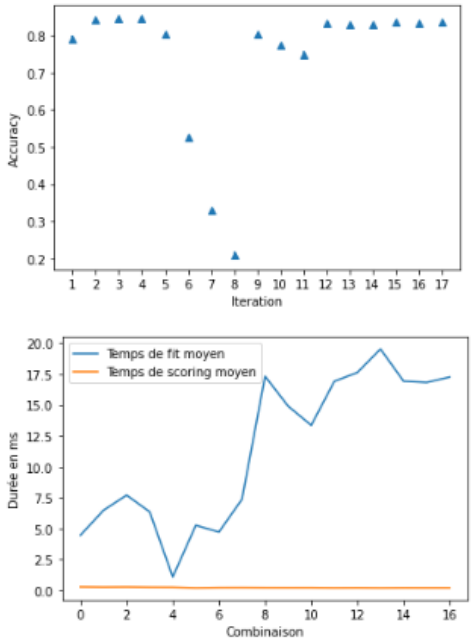

Figure 12: Score de la régression logistique, ainsi que le temps de fit

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.

A noter que pour certains algorithmes assez lourd à optimiser, nous avons recherché la bonne combinaison d'hyperparamètres sur un jeu de données réduit pour augmenter la vitesse de GridSearchCV.

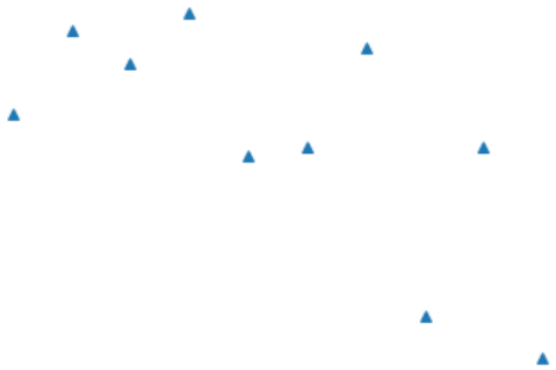

Figure 13: Score de test résultant de la validation croisée

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.

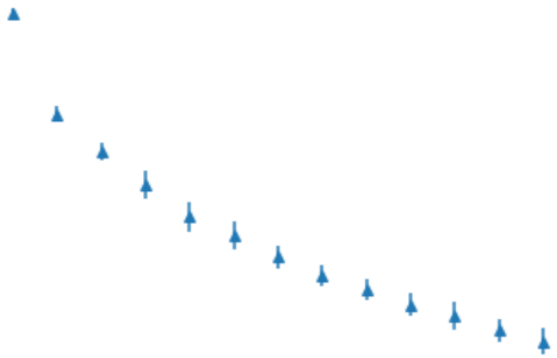

Figure 14: Accuracy en fonction du nombre de voisins k

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 D'autres modèles

Nous avons testé d'autres modèles par simple curiosité : DecisionTree qui s'est avéré non performant, RandomForest et MLP qui se sont avérés plus performants avec une accuracy de 0.90.

5.5 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 15: Capture d'écran de l'application - Vue 1

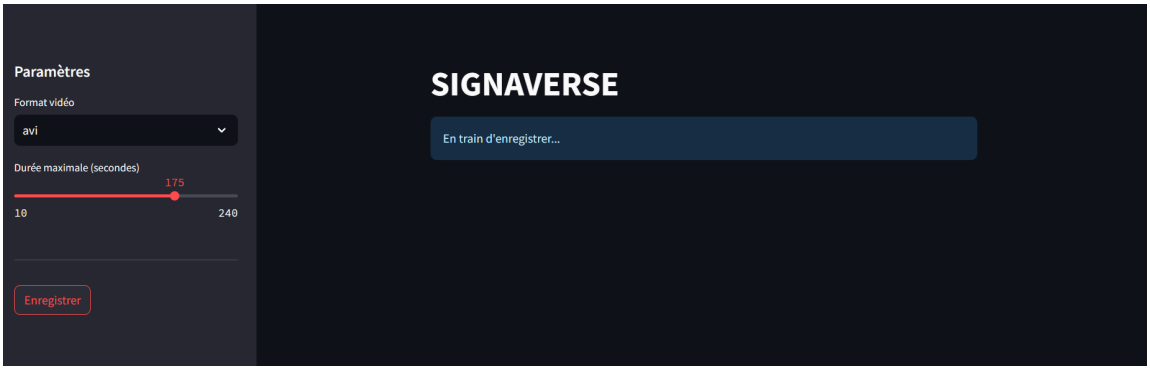

Figure 16: Capture d'écran de l'application - Vue 2

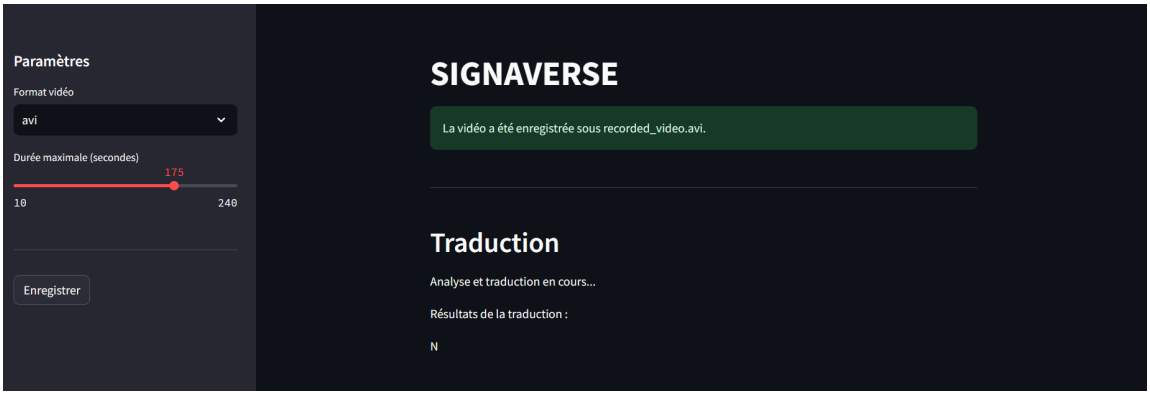

Figure 17: Capture d'écran de l'application - Vue 3

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.