Rapport projet de reconnaissance de la langue des signes

Aurore MARTORANA
Léo CHARREAU
Mael CHEVALIER
Loreena BARRERE
https://github.com/maelchvlr/projet_ia_asl

2022-2023

Table des matières

1 Présentation du sujet **3**

2 Technologies utilisées **5**

3 Développements Logiciel : Conception, Modélisation, Implémentation **7**

4 Algorithmes et Analyse **11**

5 Analyse des Résultats **14**

6 Gestion du Projet **22**

7 Bilan et Conclusions **23**

8 Bibliographie **24**

9 Annexes **25**

1 Présentation du sujet

Les langues des signes sont des langages composés de signes de la main, du visage et parfois de l'ensemble des bras et du corps. Ils sont utilisés principalement par les personnes sourdes ou malentendantes afin de faciliter leur communication. Au même titre que les langues vocales, elles se sont développées et complexifiées au cours du temps.

Il existe de nombreuses langues des signes qui correspondent aux différents pays qui l'utilisent. Parmi celles-ci, on retrouve la langue des signes américaine (American Sign Language - ASL) que nous allons adopter dans notre projet. Étant la plus répandue, de nombreuses bases de données de qualité la concernant sont disponibles, ce qui a ultimement orienté notre choix vers l'ASL.

L'objectif est d'offrir aux personnes sourdes, malentendantes, ou souffrant d'autres handicaps nécessitant une communication gestuelle, un instrument de traduction performant et précis.

Dans ce projet, nous cherchons à proposer une méthode de reconnaissance de l'ASL avec rapidité et précision. Au-delà des méthodes classiques étudiées auparavant, nous allons, dans ce projet, explorer une alternative jugée plus performante : les réseaux de neurones convolutifs.

Aujourd'hui, de nombreuses implémentations de CNN [1][2][3] voient le jour, et gagnent en popularité sur le déploiement de méthodes classiques dans des problèmes de vision par ordinateur. Dans les domaines technologiques associés à la vision par ordinateur, les CNN sont souvent privilégiés grâce à leur précision combinée à leur rapidité dans la détection de formes. C'est donc dans ce contexte que nous avons justifié le choix d'un CNN.

Tout au long de nos études en Licence Informatique, nous avons pu étudier différentes structures de données, ainsi que certains algorithmes de traitement d'images. Ces connaissances coïncident avec des éléments mis en place dans les méthodes classiques et les CNN. Notamment dans la manipulation des images et de leurs propriétés, par exemple, le calcul de l'Histogramme de gradient orienté (HOG) [4] d'une image pour la détection des contours d'un objet dans les méthodes classiques ; qui sont des notions que nous avons eu l'occasion d'étudier dans notre cursus.

De nombreuses solutions de détection d'objets classiques, souvent basées sur l'HOG, montrent leurs limite dans leur précision, le temps de préparation d'un dataset en renseignant manuellement la collection de "feature" d'un objet, et leur adaptabilité dans certaines situations (la détection pourra échouer si l'image surexposée ou sous-exposée par exemple). L'évolution de ces méthodes en méthode CNN est décrite par Xiao Youzi [5]

La rapidité et la précision étant nos principaux objectifs pour notre application, notre choix s'est tourné vers les CNN, reconnus pour leur utilisation dans des situations en temps réel. D'autres raisons secondaires justifient le choix d'un CNN :

— L'IA, et plus précisément le deep-learning offre des performances supérieures pour la reconnaissance de gestes complexes, contrairement à la programmation classique qui nécessite de coder manuellement des calculs et une représentation des formes précises pour une reconnaissance fiable. A l'inverse, certains modèles de deep-learning s'appuient sur un entraînement supervisé. Cela signifie que notre modèle est capable de reconnaître des gestes de la langue des signes qui peuvent être difficiles à décrire mathématiquement, parfois dû à des changements de lumière, contraste, ou quand l'image fournie est dans un angle difficilement analysable algébriquement.

— L'accessibilité et la possibilité d'améliorer son modèle. Le modèle peut être redéployé pour de nombreuses autres utilisations, avec de nouveaux objets à reconnaître. Cela nous a permis d'étudier plusieurs approches et d'itérer sur des résultats pour mieux comprendre le comportement du CNN.

— C'est une technologie que nous n'avions jamais eu l'occasion de développer auparavant. Nous avons vu cela comme une opportunité d'obtenir de nouvelles compétences, qui plus est, sont en adéquation avec la direction dans laquelle se dirige l'informatique de nos jours.

Dans l'ensemble, notre choix a été déterminé par plusieurs facteurs, notamment la performance supérieure pour la reconnaissance de gestes complexes, le potentiel d'amélioration continue et la possibilité de nous développer professionnellement.

Notre projet s'inscrit dans un objectif de construire une esquisse d'application permettant de traduire une partie de la langue des signes. Cette application s'avérerait utile pour les personnes sourdes et malentendantes afin qu'elle puisse se faire comprendre par d'autres individus, par exemple via des visio-conférences.

L'objectif de notre projet est de construire une esquisse d'application permettant de reconnaître une partie de la langue des signes américaine : les lettres de l'alphabet et les chiffres de 1 à 10. La caméra transmet l'image au CNN qui effectue une inférence sur celle-ci, et renvoie une bounding box et une prédiction associée qui sera annotée sur l'image finale.

Nous avons comme principal objectif d'effectuer une reconnaissance fiable et en temps réel de la langue des signes.

FIGURE 1 – exemple de reconnaissance sur la lettre V

Notre CNN, YOLO (You Only Look Once), publié par Joseph Redmon [6][7] et repose sur le framework darknet, un framework open source créé également par Joseph Redmon. YOLOv1, YOLOv2 et YOLOv3 sont des versions développées par son créateur, cependant nous avons opté pour une version plus récente, adoptée par Ultralytics, créateurs de YOLOv5, et YOLOv8. Ces versions se veulent spécialisées dans la détection d'objets en temps réel, optimisées pour être simultanément rapides et fiables. C'est donc la dernière version, YOLOV8 qui sera utilisée tout au long de ce projet.

Une fois le CNN choisi, il nous faut trouver une base de données diversifiée et suffisamment étoffée pour entraîner notre modèle. Le pré-traitement d'image et l'augmentation de données ainsi que les hyperparamètres joueront un rôle majeur dans la conception du modèle. Ces axes de recherche feront l'objet d'une attention particulière.

Enfin, notre interface graphique doit être capable d'être facilement compréhensible et doit répondre rapidement aux actions de l'utilisateur. Notre application doit être capable d'ouvrir la caméra, analyser les images à l'aide de YOLOv8 en temps réel, et l'utilisateur doit pouvoir afficher ou cacher les prédictions à l'aide de paramètres directement depuis le menu principal de l'application.

2 Technologies utilisées

Pour réaliser notre projet, nous avons décidé d'utiliser le langage Python. L'application utilise Tkinter et CustomTkinter pour son interface graphique, opencv-python pour la capture caméra, et YOLOv8 pour la détection.

Pour créer nous-même une base de données il aurait été nécessaire de prendre des milliers de photos de mains, dans différentes positions, angles, et luminosité. Nous avons donc préféré récupérer une base de données déjà existante sur Kaggle, non annotée, contenant 1000 images par lettre de l'alphabet de la langue des signes américaine, ainsi que 1000 images pour chaque chiffre de 1 à 10. Cette base de données à été créée avec SeaHaven, prévu pour générer automatiquement une base de données diversifiée à partir de modèles 3D de mains, ensuite placées dans des environnements virtuels, avec des luminosités et angles différents.

Pour gérer le projet et notamment le pré-traitement d'images, nous avons utilisé Roboflow qui permet de labelliser, organiser, et exporter une base de données dans des formats spécifiques.

Dans le but de construire une application en temps réel, nous cherchons un outil qui allie précision et rapidité pour avoir un compromis idéal. Parmi l'état de l'art actuel des CNN de reconnaissance d'objets, YOLO est une implémentation idéale pour du temps réel.

Ce CNN est un algorithme de régression linéaire qui permet de prédire des cadres de délimitation des objets sur une image, appelés "bounding box", en une seule lecture. Ses versions surpassent d'autre algorithmes de détections comme SSD (Single Shot Detection), ou encore Faster R-CNN (Faster Region based Convolutional Neural Networks) [8][9], et dans certaines applications, se révèlent être dix fois plus rapide que Faster-RCNN [10]. Cette famille de CNN s'est montrée être la plus performante dans le cadre d'analyse de flux vidéo en direct.

La justification de la version 8 de YOLO s'exprime ensuite par un net gain de précision, à un coût moindre de temps de calcul comparé à ses prédécesseurs. Plusieurs tests de différentes sources renforcent ces résultats, notamment Ultralytics, Stereolabs, et Augmented Startup.

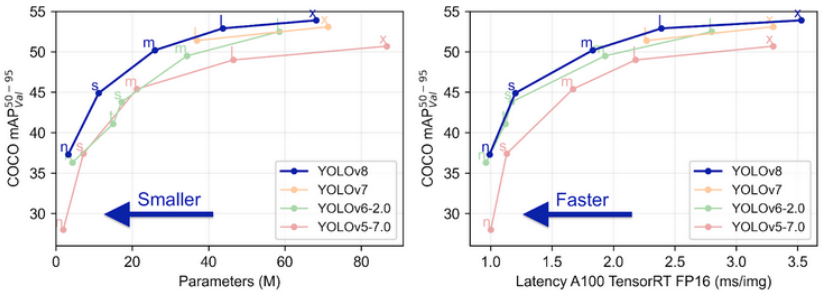

FIGURE 2 – Graphique des performances des différentes versions de YOLO

Ces résultats montrent que YOLOv8 est pertinent pour notre type d'application, puisque nous recherchons une rapidité couplée à une précision de détection. A la précision de base de YOLOv8 viennent se greffer les modifications des hyperparamètres afin d'améliorer les performances de notre modèle. Nous avons donc mené nos propres tests et ajusté les hyperparamètres pour maximiser les performances de notre modèle.

A partir de notre choix de CNN, le langage utilisé a donc été le Python. Ce langage très versatile est intuitif, polyvalent et multi-plateforme, et est associé à l'une des plus grandes bibliothèques open source tous langages confondus. Pour ce qui est de l'interface graphique, Python propose des plugins accessibles et modernes tels que Tkinter et CustomTkinter, intuitifs pour l'utilisateur, ce qui va de pair avec la vision de notre outil qui vise la simplicité et l'accessibilité.

3 Développements Logiciel : Conception, Modélisation, Implémentation

Deux programmes ont été créés, le premier étant le programme principal, comportant l'interface graphique et le nécessaire pour réaliser des inférences en temps réel, et le deuxième est un script de comparaisons de modèles, qui vise à créer une courte vidéo ou analyse de collection d'images qui présente les performances du modèle créé.

Le programme principal reçoit un flux vidéo et réalise une inférence en temps réel.

Le second programme contient une vidéo de référence déjà annotée manuellement avec des annotations "ground-truth" (c'est à dire, le résultat attendu). A l'exécution, l'application peut traiter image par image la vidéo manuellement annotée, ou traiter une bibliothèque d'image manuellement annotée. Ensuite, il effectue une prédiction sur l'image, puis récupère les résultats pour construire une nouvelle image annotée avec un seuil de confiance. Le résultat final sera une vidéo ou une collection d'image comparative entre le ground-truth et la prédiction du modèle. L'indice de performance pourra ainsi être calculé pour chaque image.

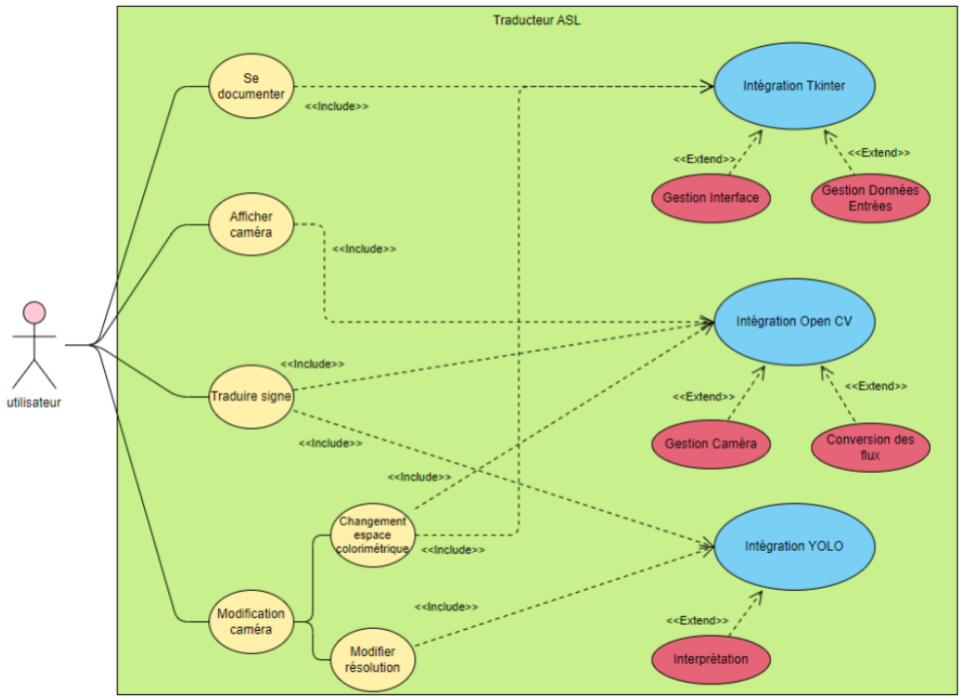

FIGURE 3 – Diagramme de cas

L'utilisateur a plusieurs choix, afficher sa caméra, appliquer des modifications, traduire un signe et se documenter par rapport à l'application, YOLOv8, et la langue des signes. D'autres fonctionnalités permettent à l'utilisateur de modifier les images envoyées au CNN.

L'interface graphique implémentée est simple à comprendre pour tout utilisateur, le but étant de mettre en avant l'accessibilité de l'application et la simplicité d'utilisation du transcripteur.

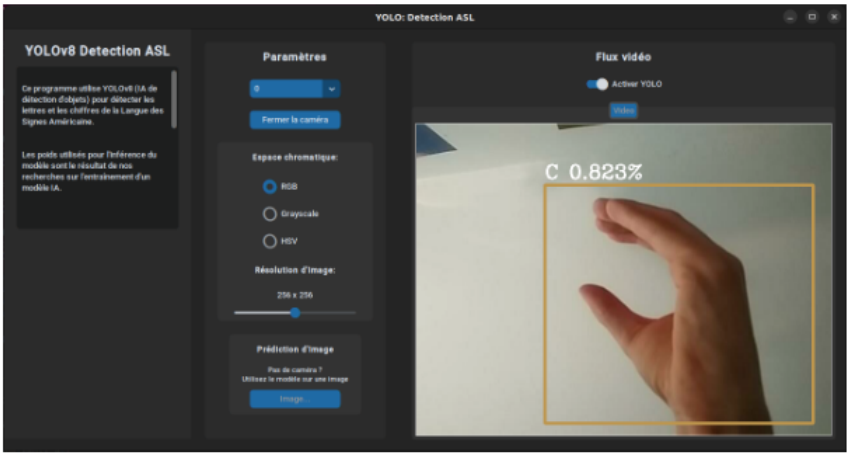

FIGURE 4 – Capture d'écran de l'application en cours de fonctionnement

L'application se divise en 3 sections :

La première partie à gauche de l'écran est une section explicative, récapitulant toutes les fonctionnalités de l'application. Une description du CNN ainsi que de nos objectifs, puis les détails de chaque paramètre et de son utilité.

Dans la partie centrale se trouve un "panneau de contrôle" permettant de choisir la caméra à utiliser, choisir de l'activer ou non et de changer l'espace chromatique (couleur, noir et blanc, Hue / Saturation / Value (HSV)), à noter cependant que le changement d'espace chromatique est uniquement visuel et ne change en rien les résultats fournis par YOLO puisque ce dernier a ses espaces propres de travail. Il est également possible de modifier la résolution de l'image donnée au CNN afin d'améliorer la vitesse de calcul (réduire la taille de l'image) ou d'augmenter sa précision (augmenter la taille de l'image). Si l'utilisateur ne dispose pas de caméra, il est possible de choisir une image depuis son ordinateur pour la faire analyser.

La dernière partie à droite est la visualisation en temps réel de la caméra de l'utilisateur, il est possible d'activer ou de désactiver la prédiction afin de faire apparaître les "bounding boxes" de détection, la classe (lettre ou chiffre) détectée ainsi que son pourcentage de fiabilité. Sur cette même visualisation s'appliqueront en direct les changements d'espace couleurs, mais également si l'utilisateur n'a pas de caméra, l'image analysée fournie au préalable.

Les structures de données utilisées sont des bases de données, collection uniforme d'images, regroupées par classes. Nous avons décidé d'utiliser des images

en couleurs au format ".jpg" et de dimension 800 x 800 pixels. Pour commencer, nous avons pris 50 images au hasard parmi les 1000 images de chaque classe.

Plus tard dans le développement du projet, nous avons décidé d'utiliser des images plus petites, d'une taille de 256 x 256 pixels. Ce choix est justifié dans le cadre de notre utilisation : les webcams sont rarement en haute définition pour de nombreux utilisateurs, de plus le CNN performe plus rapidement avec des données plus légères. Avec ce choix s'accompagne aussi une exploration de base de donnée différente pour comparer les performances entre notre premier choix (base de donnée générée synthétiquement avec SeaHaven par des scans 3D de mains, limitant ainsi la diversification des mouvements), et celle-ci. Cette nouvelle base de données comporte seulement 10 images par classe mais sont toutes de réelles photographies. Ces analyses seront détaillées dans la partie Résultats.

Concrètement, le pré-traitement d'image avec Roboflow se déroule ainsi :

- Uploader séparément chaque paquets d'images afin de faciliter l'attribution des annotations.
- Créer une classe d'annotation, ici correspondant à une lettre ou un chiffre, afin de pouvoir annoter toutes les images correspondant à cette lettre/chiffre.
- Sélectionner les images une à une, encadrer la partie qui nous intéresse (ici strictement manuellement) et sélectionner la classe d'annotation représentée.
- Vérifier que toutes les images ont bien été traitées, une fois vérification faite, uploader les images annotées dans notre base de données, en divisant les images en 3 parties : "training" du modèle, vérification et tests.
- Recommencer pour la prochaine classe, jusqu'à ce que toutes les classes soient annotées.
- Lancer un entraînement du modèle, analyser les courbes et en tester l'efficacité afin de déduire s'il est nécessaire d'ajouter d'autres images à la base de données ou de changer des hyperparamètres. On portera une attention particulière au sur-apprentissage et l'underfitting qu'il faudra éviter.

Au total, le projet compte deux scripts :

- Le premier est une classe qui englobe les différentes fonctions de l'application graphique ainsi que toutes ses propriétés d'interface. Nous avons décidé de garder une unique classe car le peu de fonctions ne justifiait pas une approche plus complexe de programmation. Au total, avec les commentaires et les indications pour préparer l'environnement de travail, le code fait 290 lignes.
- Le second est un simple script sans objets, seulement composé d'une fonction et d'autres opérations pour le traitement d'image. Au total, avec les commentaires et les indications, ce code fait 108 lignes.

4 Algorithmes et Analyse

L'Intersection over Union (IoU) [11] (voir Indice de Jaccard) est un outil de mesure utilisé pour déterminer la performance d'un algorithme de détection d'objets.

C'est un ratio qui compare la "ground truth box" (celle que nous avons préalablement définie à la main), à la box prédite par l'algorithme avec la formule suivante : IoU = aire de l'intersection des 2 boxes / aire de leur union

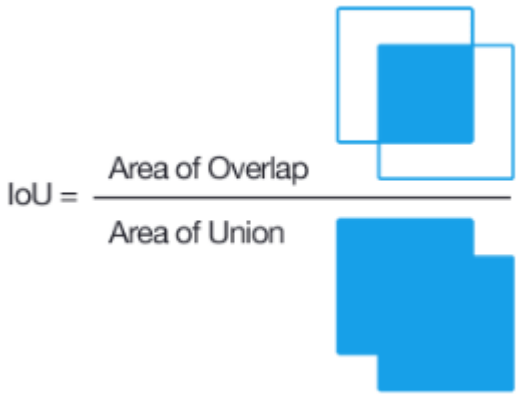

FIGURE 5 – Illustration du principe de l'IoU

N'importe quel algorithme de détection qui utilise des "bounding boxes" peut être évalué avec cet outil. L'objectif est généralement d'obtenir un score IoU $>$ 0.5.

Déroulement de l'algorithme dans notre code `''IoU_comparison_video.py''` : Nous allons extraire les "ground truth box" de la vidéo pour les comparer à celles de YOLO. Image par image, le script effectue ces opérations :

— Définitions des variables :

```
boxA = [xA, yA, xB, yB]

if res[0].boxes:
  bb = res[0].boxes[0].xyxy.numpy()[0]
  pxA = int(float(bb[0]))
  pyA = int(float(bb[1]))
  pxB = int(float(bb[2]))
  pyB = int(float(bb[3]))

  boxB = [pxA, pyA, pxB, pyB]
```

FIGURE 6 – Initialisation code

boxA est la “ground truth box” ; boxB est la box prédite par YOLO.
Les boxes sont au format :
box[0], box[1] : Xa,Ya qui correspond au coin supérieur gauche de la box
box[2], box[3] : Xb, Yb qui correspond au coin inférieur droit de la box

— Calcul de l’aire du rectangle d’intersection qui sera stockée dans “interArea” :

```
xA = max(boxA[0], boxB[0])
yA = max(boxA[1], boxB[1])
xB = min(boxA[2], boxB[2])
yB = min(boxA[3], boxB[3])
# compute the area of intersection rectangle
interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
# compute the area of both the prediction and ground-truth
```

FIGURE 7 – Calcul de l’aire du rectangle d’intersection

— Calcul de l’aire de boxA (boxAArea) et celle de boxB (boxBArea), on obtient l’aire de leur union :

```
boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
# compute the intersection over union by taking the intersection
# area and dividing it by the sum of prediction + ground-truth
# areas - the interesection area
```

FIGURE 8 – Calcul de l’aire de l’union

— Et finalement l’IoU :

```
iou = interArea / float(boxAArea + boxBArea - interArea)
# return the intersection over union value
```

FIGURE 9 – Calcul de l’IoU

Une fois l’image analysée par YOLOv8, il faut en extraire les données. La fonction predict() renvoie une collection de tensors, une structure de données composée de matrices multidimensionnelles préparées pour l’optimisation de temps de calcul.

Dans cette collection on retrouve notamment :

— Les “boxes”, qui serviront pour l’IoU, et leur affichage à l’écran, contient pour chaque objet détecté son emplacement, la classe prédite et son score de fiabilité
— Les “masks” (utile dans des travaux de segmentation d’instance, pas dans notre cas)
— Les “probs”, probabilités de détection pour chaque classe

Déroulement de l'algorithme dans nos deux scripts, extraction des données dans un premier temps, puis reconstruction d'une image annotée avec les prédictions :

— Tout d'abord, comme notre application permet de changer la résolution d'image comme donnée d'entrée à YOLO, on calcule au préalable le facteur de multiplication appliqué à l'image, pour ensuite renvoyer une image non redimensionnée sur laquelle on annotera les résultats. En convertissant les résultats de YOLO d'une image plus petite, vers des valeurs qui correspondent à notre image de taille originale, les annotations ne seront pas déformées :

```
### GROUND TRUTH DRAWING
# get image size to adjust bounding box float values to pixel values
size_x = image.shape[1]
size_y = image.shape[0]
```

FIGURE 10 – Attribution de la taille des annotations

— Ensuite, on extrait les données pour chaque objet détecté dans l'image, c'est à dire sa classe, sa probabilité, ainsi que les coordonnées de la "bounding box", qu'on va multiplier par notre facteur de redimension d'image pour convertir ces coordonnées à l'échelle d'une image plus petite, vers notre image taille réelle :

```
# open label file with bounding box values
# converting (x_center, y_center, width, height) to a (xA, yA, xB, yB) format
f = open(label_name)
bb = f.read().split()
if bb:
  label = bb[0]
  x = float(bb[1]) * size_x
  y = float(bb[2]) * size_y
  width = float(bb[3]) * size_x
  height = float(bb[4]) * size_y

  xA = int(x - width / 2)
  yA = int(y - height / 2)
  xB = int(x + width / 2)
  yB = int(y + height / 2)
```

FIGURE 11 – Extraction des données des résultats

— Une fois les coordonnées converties, on peut annoter l'image avec nos prédictions, premièrement en dessinant la "bounding box" puis en dessinant le nom de la classe et sa probabilité :

```
# draw box
cv2.rectangle(image, start_point, end_point, (50, 50, 50), 3)

# draw class name + confidence
class_list = ["L", "B", "C", "7"]

font = cv2.FONT_HERSHEY_DUPLEX
cv2.putText(
  image,
  class_list[int(label)],
  (xA + int(width) - 25, yA + 23),
  font,
  0.7,
  (50, 50, 50),
  1,
)
```

FIGURE 12 – Affichage des bounding box et de l'indice de confiance

Pour ces deux algorithmes, les temps de calculs sont négligeables, respectant le but de l'application qui se veut être en temps réel. Les librairies comme opencv ou numpy utilisent toutes les deux des fonctions implémentées dans notre code de complexité O(n).

L'Intersection over Union ne contient pas de boucles imbriquées, ni de récursion. Les opérations sont peu coûteuses :

- min(), max() et d'autres opérations élémentaires sont en O(1)
- numpy(), et les fonctions internes à YOLOv8 sont en O(n)
- Donc Intersection over Union est de complexité O(n)

Quant à l'algorithme d'extraction de données, une boucle cycle sur les objets reconnus, en effectuant des opérations de numpy comme numpy.round() de complexité O(n). Le reste étant des opérations élémentaires, cela nous donne une complexité de $O(n^2)$

5 Analyse des Résultats

L'analyse des résultats est liée à la méthode d'entraînement. Plusieurs choix étaient possibles :

- L'entraînement automatique avec Roboflow, l'inconvénient étant l'opacité des hyperparamètres et des options utilisées, qui ne sont pas accessibles.
- L'entraînement manuel sur Google Collab en empruntant la puissance de leurs ordinateurs en cloud, l'avantage étant la transparence totale et la personnalisation de l'entraînement, au coût du temps d'exécution.

Nous avons choisi l'entraînement avec Google Collab pour la maîtrise et la transparence des hyperparamètres.

Dans cette section, deux grands axes de recherches seront explorés : le traitement d'image, et les hyperparamètres. Nous comparerons ensuite leurs effets sur la performance de nos modèles dans le cadre d'une analyse quantitative. Enfin, nous aborderons la question de la pertinence des bases de données choisies dans notre projet.

Premièrement, le traitement d'image se décompose en deux parties : le pré-traitement, et l'augmentation de données.

Le pré-traitement et l'augmentation sont des processus dans lesquels un ensemble de transformation s'applique à un ensemble d'images.

Le pré-traitement vise à conformer les images à des normes. Les transformations disponibles sont les suivantes :

- `Orientation automatique`
- `Mise à l'échelle`

L'augmentation de données est reconnu pour son éfficacité [12][13], et crée des images supplémentaires modifiées pour artificiellement augmenter le nombre d'images et simuler des conditions différentes. Les transformations disponibles sont les suivantes :

- `Miroir horizontal`
- `Rognage maximum de 20%`
- `Rotation +- 20 degrés`
- `Distortion +- 7 degrés horizontal et vertical`
- `Niveau de gris pour 10% des images`
- `Luminosité +- 25%‘`
- `Exposition +- 25%`
- `Flou jusqu'à 1,25 pixel`

Enfin, les hyperparamètres supervisent le modèle en modifiant son comportement pendant l'entraînement. L'ensemble des hyperparamètres utilisés sont les suivants :

- `Epochs`
- `Dropout`
- `Rotation`
- `Taille de l'image`

L'interprétation des résultats est itérative. A chaque entraînement on associe un indice de performance en comparant sa prédiction sur des images déjà annotées.

Le nombre d'"epochs" joue un rôle important. Le passage de 10 à 25 itérations augmente considérablement la précision.

FIGURE 13 – Entraînement n°1 - epochs=10 | M :064%

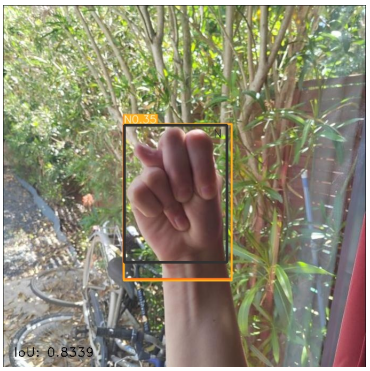

FIGURE 14 – Entraînement n°2 - epochs=10 | N :035%

On observe un net gain de précision dans la détection de la lettre N, faussement classifiée pour un entraînement de 10 epochs. Ce gain de précision peut aussi être observé dans les résultats quantitatif du modèle.

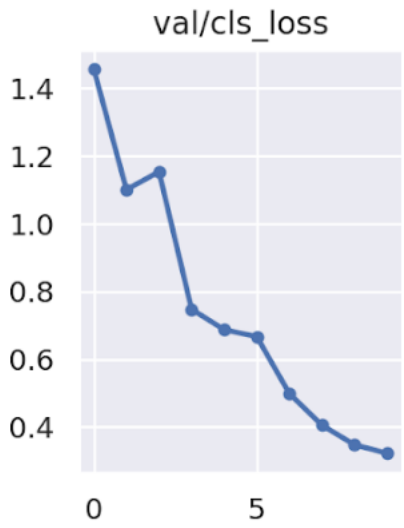

FIGURE 15 – Entrainement n°1 - Indice d'erreur de labellisation de classe (Y : epochs, X : perte)

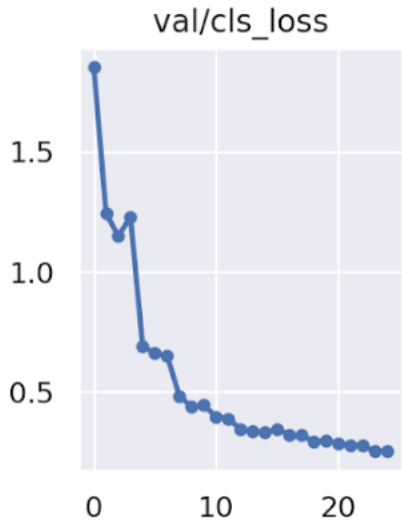

FIGURE 16 – Entrainement n°2 - Indice d'erreur de labellisation de classe (Y : epochs, X : perte)

La perte d'information est supérieure quand le modèle est entraîné moins longtemps, 0.32 pour 10 epochs contre 0.25 avec 25 epochs. Ces comparatifs

sont effectués pour chaque entraînements successifs.

Avec ces résultats, nous avons approfondis le prochain axe de recherche, les hyperparamètres. Après plusieurs tentatives, la progression s'est arrêtée. Le modèle était limité par l'over-fitting, quand un modèle possède plus de paramètres que nécéssaire [14]. L'erreur de prédiction stagne, puis augmente. Ce phénomène est observé en calculant la différence entre le taux d'erreur pendant la phase d'entraînement, et le taux d'erreur pendant la phase de validation. Pour valider cette théorie, nous avons construit une base de donnée ne comprenant que 10 images par classe.

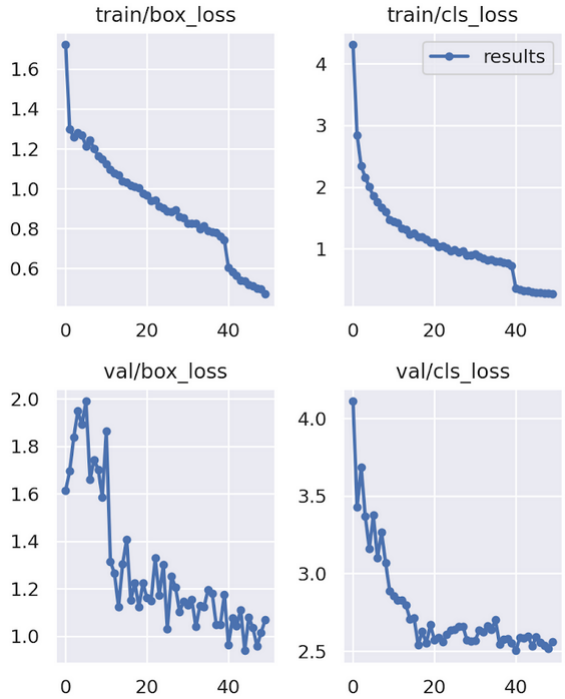

FIGURE 17 – 10 images par classes, over-fitting

L'une des solutions explorée est de réduire la complexité des données, avec un hyperparamètre "dropout" [15]. Cependant, avec plusieurs combinaisons d'hyperparamètres dont "dropout", les résultats ont recommencé à stagner.

Nous avons donc exploré le dernier axe de recherche, l'augmentation de données. Roboflow permet d'appliquer cet ensemble de transformations rapidement, après quelques itérations, nous avons trouvé une combinaison de transformations qui résultent en un modèle plus performant.

Cette augmentation de données, exploitée à partir du 6ème entraînement, donne des résultats plus précis et stables, avec moins de fluctuations. A ce stade, nous avons décidé que la taille de l'image serait de 256 pixels contrairement à 800 pixels, initialement utilisé mais plus lent. Cette baisse en résolution offre

un avantage considérable en temps de calcul, avec des inférences moyennes de quelques dizaines de millisecondes comparées aux centièmes de millisecondes initiales.

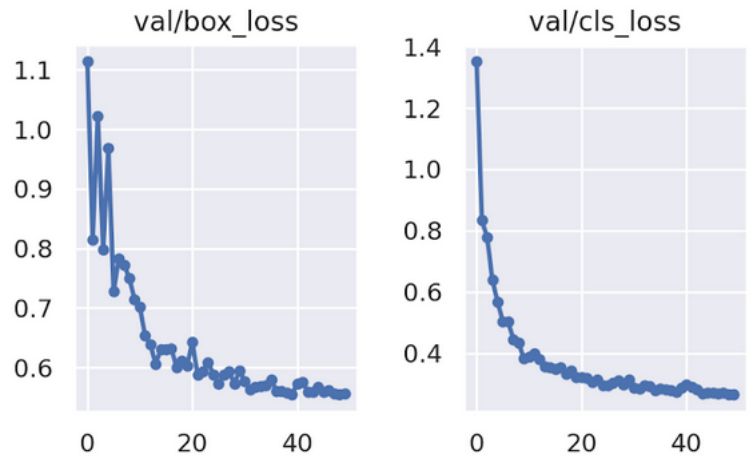

FIGURE 18 – Entrainement n°5, epochs=50, dropout=0.2, imgsize=800, sans augmentation

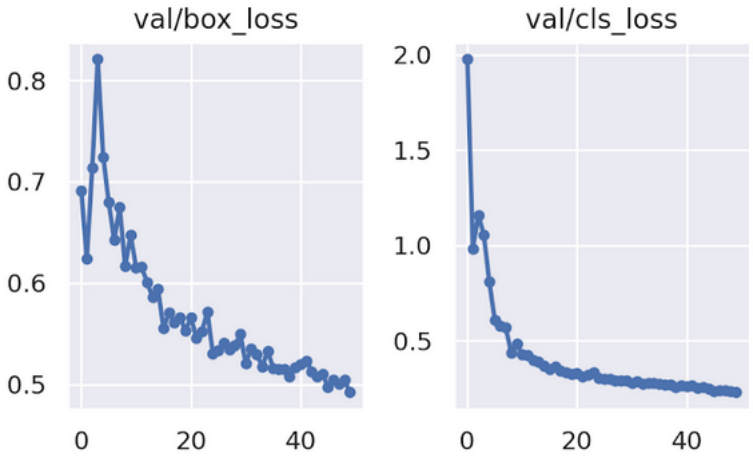

FIGURE 19 – Entrainement n°6, epochs=50, dropout=0.2 rotation=20, imgsize=256, avec augmentation (application de rognage et miroir horizontal)

Par curiosité nous avons également exploré comme cité précédemment une nouvelle base de donnée plus réaliste, qui ne serait pas générée automatiquement

par des scans de mains en 3D. Cependant cette base de donnée s'est révélée trop petite pour la taille de notre projet, résultant à de l'over-fitting :

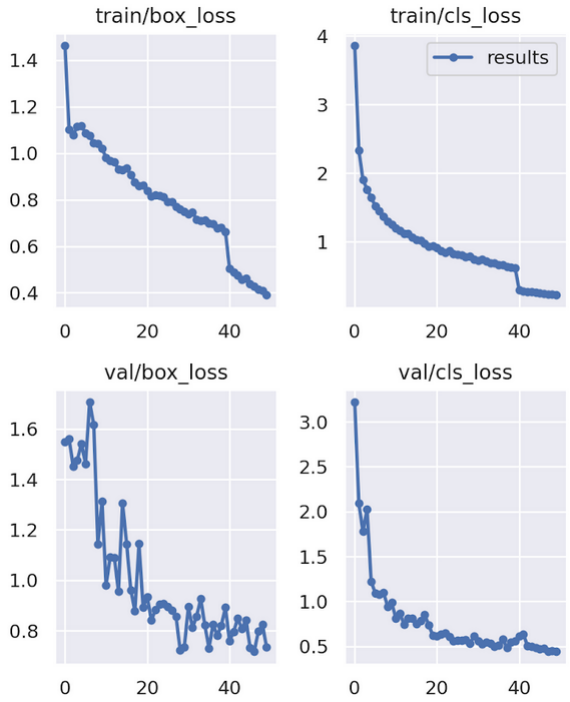

FIGURE 20 – Base de donnée alternative, epochs=50, dropout=0.2, rotation=20, imgsize=256, plus toutes les augmentations

Après d'autres recherches sur de potentiels axes d'amélioration, nous en avons déduit que l'élément limitant était le temps d'entraînement. En effet sur la figure 19, un nombre plus important d'epochs aurait produit un modèle plus performance. Cependant le temps requis pour entraîner les modèles dépassent les limites de Google Collab. Le modèle final, sur sept entraînements, est le résultat final de l'exploration de ces axes de recherches pour améliorer au maximum la précision en restant rapide.

FIGURE 21 – Entrainement n°7 epochs=50, dropout=0.2 rotation=0.2 imgsize=256, plus toutes les augmentations

Analyse des résultats du modèle final :

— Les lettres J et Z sont des mouvements continus, notre modèle n'étudie que des images fixes, donc la détection est faussée pour ces lettres.

— En ASL, les signes “6” et “W” sont les mêmes, tous les modèles développés sont donc induits en erreur.

— En comparaison, le premier modèle détecte 21 classes correctement sur 36 au total.

— Notre septième et dernier modèle détecte 29 classes correctement sur 36 au total, avec un taux de précision plus élevé pour chaque lettre.

Ces résultats sont basés sur une collection d'images et une vidéo que nous avons construite et annotée manuellement. Elle sert de référence et est générée automatiquement par notre script ‘‘`IoU_comparison_video.py`’’.

6 Gestion du Projet

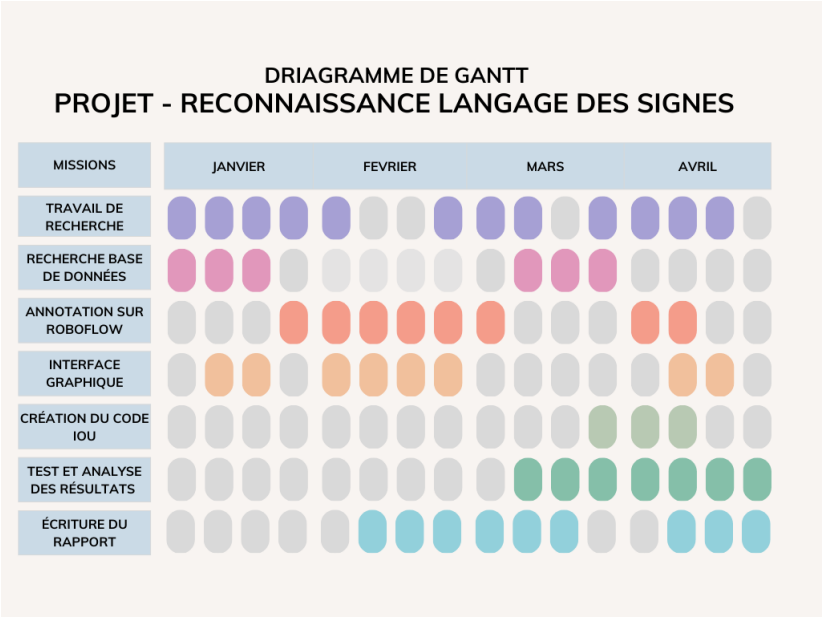

FIGURE 22 – Diagramme de Gant

Nous avons commencé à travailler sur notre projet durant le mois de janvier. A cette période, nous avons principalement fait des recherches sur le deep learning afin de mener à bien notre projet. Nous nous sommes répartis les tâches afin d'être plus efficaces, nous avons décidé de scinder les objectifs en deux : commencer à rechercher une base de données qui correspond à nos besoins et développer une interface graphique. Après avoir trouvé une base de données exploitable, nous avons commencé un travail de grande ampleur : l'annotation. Cette tâche est très répétitive et chronophage, nous avons donc passé plus d'un mois sur cette étape ; en parallèle du développement de l'interface graphique.

Nous avons commencé l'écriture de notre rapport relativement tôt, lors du mois de février. Au commencement, il s'agissait de traces sur ce que nous avions réussi à faire, ou bien des questionnements et réflexions sur la suite de notre projet. Au fur et à mesure de notre avancement, nous avons rédigé et structuré le rapport, tout en ajoutant des références, des figures et des annexes.

Nous avons effectué nos premiers tests durant le mois de mars. Cependant, nous pensions que les résultats obtenus avec notre première base de données pouvaient être poussés plus loin. Nous avons donc voulu changer d'approche en recherchant une autre base de données plus "organique", et pas générée auto-

matiquement par un outil, ce qui a engendré un nouveau travail d'annotation sur celle-ci. Nous avons également créé différentes versions de nos données, avec des tailles plus petites, pour observer le comportement du modèle pendant l'entraînement.

De ce fait, afin de quantifier la précision de nos tests, nous avons développé le code IOU fin mars, début avril.

Lors de la recherche d'une base de données composés d'images de mains représentant des chiffres et des lettres en langues des signes, plusieurs modifications ont dû être faites. Nous avons trouvé des bases de données déjà annotées, cependant celles-ci n'étaient pas assez imposantes pour nous, qui souhaitions avoir environ 1000 images au total (100 images pour le A, 100 images pour le B...). Nous nous sommes donc tournés vers des bases non-annotées, trouvées sur le site Kaggle. Nous avons trouvé deux bases de données assez complètes et diverses, une pour les lettres de l'alphabet et une pour les chiffres de 1 à 10. Elles disposaient de beaucoup d'images, nous avons donc créé un nouveau projet, composé de 50 des images trouvés dans les bases de données de Kaggle. Nous avons dû créer plusieurs bases de données différentes afin d'arriver à la base de données qui nous satisfait : 100 images de chaque classe (par classe, on entend la classe A, la classe B...).

7 Bilan et Conclusions

Au final, nous avons atteint nos objectifs, c'est-à-dire, une application simple et très accessible, tenant compte des différents problèmes techniques des utilisateurs (manque de webcam/caméra par exemple), où toutes les fonctionnalités sont expliquées ou intuitives.

Notre réseau neuronal est rapide et précis et est capable d'interpréter les signes dans de nombreuses conditions, capable de reconnaître la majorité des lettres de l'alphabet et les chiffres de 1 à 10 sans trop de difficultés. Notre base de donnée générée synthétiquement par des scans de mains en 3D s'est révélée être suffisante et adaptée à notre utilisation. Notre limite de temps et de puissance de calcul pour produire des modèles élaborés confirment que les autres axes de recherches ont étés correctement exploités.

Au-delà du réseau neuronal, l'interface graphique en elle même est épurée, simpliste mais agréable, avec toutes les options disponibles en un clic et une actualisation en temps réel à chaque modification.

Avec le recul et l'expérience obtenue de par la création de l'application, il aurait été intéressant d'implémenter la reconnaissance de simple signes basiques tels que Bonjour ou Merci, impliquant une détection du mouvement, mais il aurait fallu une base de donnée plus conséquente pour détecter un mouvement

continu, et nous aurions été limités par Roboflow en terme du nombre d'images. Il aurait également été intéressant de permettre aux applications de communication tierce (Skype, Microsoft Teams, Discord) d'exploiter les informations fournies par notre application et de les utiliser comme flux d'informations vidéos lors d'un appel, permettant une transcription intégrée en temps réel.

En conclusion, le développement de cette application aura été un réel challenge technique et de réflexion. Nous avons découvert l'intelligence artificielle ainsi que le deep learning et nous avons dû trouver un moyen de l'implémenter et de l'adapter à notre sujet.

8 Bibliographie

Références

[1] Keiron O'Shea. "An Introduction to Convolutional Neural Networks". In : (2015), p. 10. DOI : https://doi.org/10.48550/arXiv.1511.08458.

[2] Rikiya Yamashita. "Convolutional neural networks : an overview and application in radiology". In : (2018), p. 19. DOI : https://doi.org/10.1007/s13244-018-0639-9.

[3] Ming Liang. "Recurrent Convolutional Neural Network for Object Recognition". In : (2015), p. 9. DOI : 10.1109/CVPR.2015.7298958.

[4] Navneet Dalal. "Histograms of Oriented Gradients for Human Detection". In : (2005), p. 9. DOI : 10.1109/CVPR.2005.177.

[5] Xiao Youzi. "A review of object detection based on deep learning". In : (2020), unknown. DOI : https://doi.org/10.1007/s11042-020-08976-6.

[6] Josepth Redmon. "You Only Look Once : Unified, Real-Time Object Detection". In : (2016), p. 10. DOI : https://doi.org/10.48550/arXiv.1506.02640.

[7] Peiyuan Jiang. "A Review of Yolo Algorithm Developments". In : (2022), p. 8. DOI : https://doi.org/10.1016/j.procs.2022.01.135.

[8] Shrey Srivastava. "Comparative analysis of deep learning image detection algorithms". In : (2021), p. 27. DOI : 10.21203/rs.3.rs-132774/v1.

[9] Bilel Benjdira. "Car Detection using Unmanned Aerial Vehicles : Comparison between Faster R-CNN and YOLOv3". In : (2019), p. 6. DOI : 10.1109/UVS.2019.8658300.

[10] Turpi Mahendrakarr. "Performance Study of YOLOv5 and Faster R-CNN for Autonomous Navigation around Non-Cooperative Targets". In : (2022), p. 12. DOI : 10.1109/AERO53065.2022.9843537.

[11] Hamid Rezatofighi. "Generalized Intersection over Union : A Metric and A Loss for Bounding Box Regression". In : (2019), p. 9. doi : https://doi.org/10.48550/arXiv.1902.09630.

[12] Connor Shorten. "A survey on Image Data Augmentation for Deep Learning". In : (2019), p. 48. doi : https://doi.org/10.1186/s40537-019-0197-0.

[13] Luis Perez. "The Effectiveness of Data Augmentation in Image Classification using Deep Learning Regression". In : (2017), p. 8. doi : https://doi.org/10.48550/arXiv.1712.04621.

[14] Douglas M. Hawkins. "The Problem of Overfitting". In : (2004), p. 1. doi : https://doi.org/10.1021/ci0342472.

[15] Nitish Srivastava. "Dropout : A Simple Way to Prevent Neural Networks from Overfitting". In : *Journal of Machine Learning Research* (2014), p. 30. url : http://jmlr.org/papers/v15/srivastava14a.html.

9 Annexes

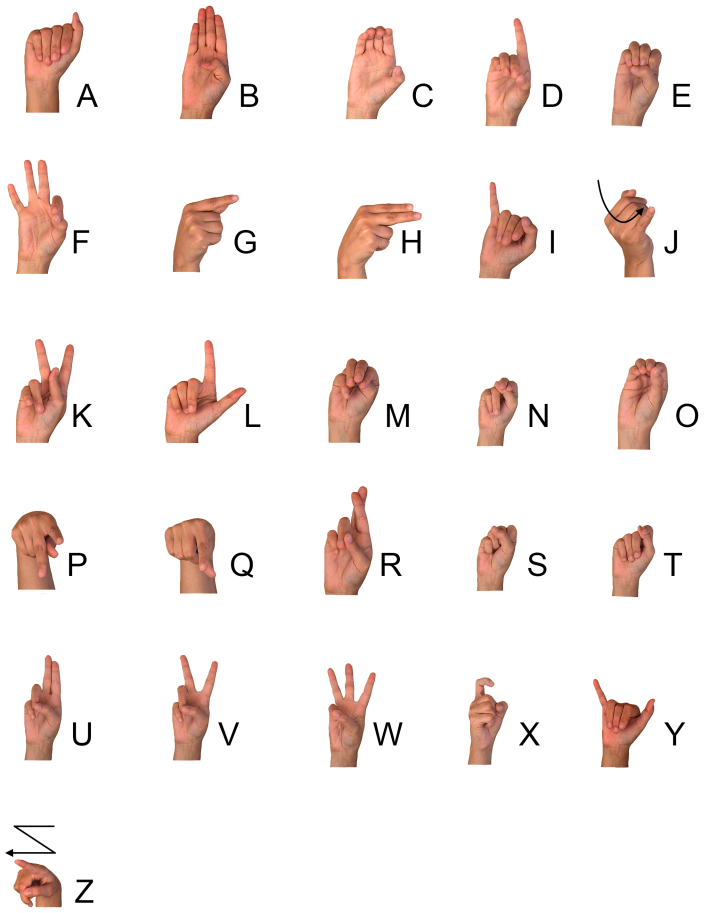

Figure 23 – ASL - Alphabet

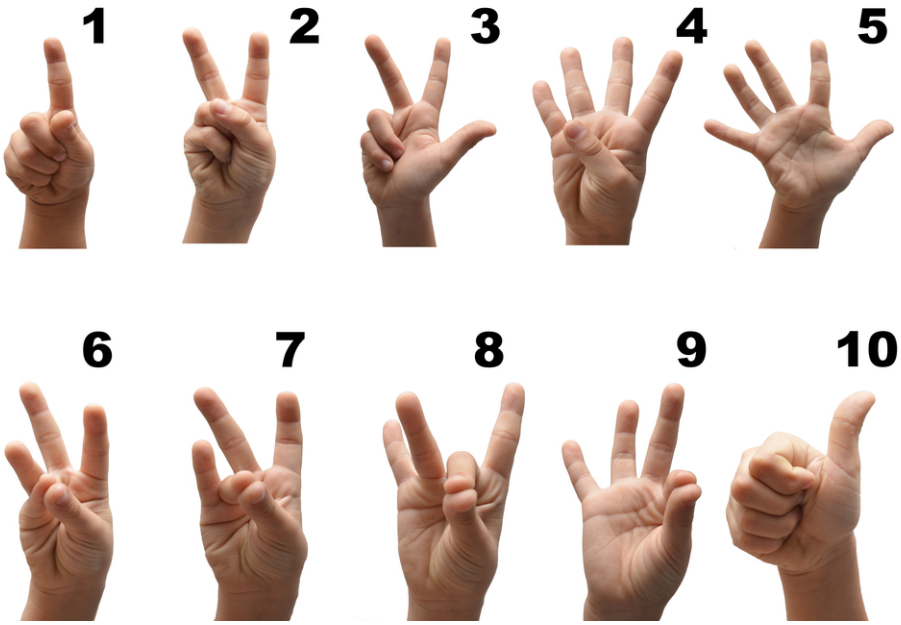

Figure 24 – ASL - Nombres de 1 à 10