

# Module: Apprentissage

**SIGN LANGUAGE DETECTION**

Présenté par :

- ABA MEKONGO Pascal
- ADOSSEHOUN KOSSI Josué

*Hanoï 09/09/2022*

Encadrement : Dr. Vu Viet Vu

# Sommaire

Introduction	2
I. Contexte et justification	2
II. Choix de l'architecture	3
Le neuro flou (FNN)	3
Le K-NN	3
1) Principe de la méthode des k-NN	4
Le LSTM	4
2) Architecture de LSTM	4
III. Implémentation	6
a) Ressources	6
b) Les librairies	6
c) Process	7
3) L'extraction des valeurs des Keypoints	8
Collecte du jeu de donnée	8
4) Pré traitement des données pour la création des features et Labelles	8
Présentation des résultats:	9
IV. Difficultés	10
V. Perspectives	11
VI. Conclusion	11
VII. Bibliographie	12

## Introduction

La Machine Learning (ML) est le champ d'étude de l'Intelligence Artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'apprendre à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. C'est un module de formation enseigné pour les étudiants de Master 2 de l'IFI option informatique, afin de leur donner des connaissances nécessaires pour analyser, concevoir, optimiser, développer et implémenter des algorithmes pouvant résoudre des problèmes de certains domaines d'activité. Le présent document est le rapport initié au terme de ce module où nous avons développé un prototype de traduction de langage de signes et dont les étapes suivantes: contextes et justification, le choix de l'architecture, l'implémentation, la présentation des résultats, les limites, les projections et enfin la conclusion constitueront les principales parties.

### I. Contexte et justification

L'évolution vertigineuse de l'Intelligence Artificielle a comme impact direct, la socialisation de l'apprentissage automatique qui trouve des applications dans plusieurs domaines d'activités (Fig.1). En d'autres termes la multiplicité d'algorithmes existants en ML justifie les avancées notables observées dans les domaines comme la médecine, l'automobile, la banque, les finances la liste n'est pas exhaustive. C'est dans cette ambiance que nous avons, pour le module d'Apprentissage, sollicité résoudre le problème de la communication entre les sourds muets et le reste de la communauté, un problème effectif puisque la majeure partie de la population ignore les signes utilisés par cette frange de la société pour communiquer.

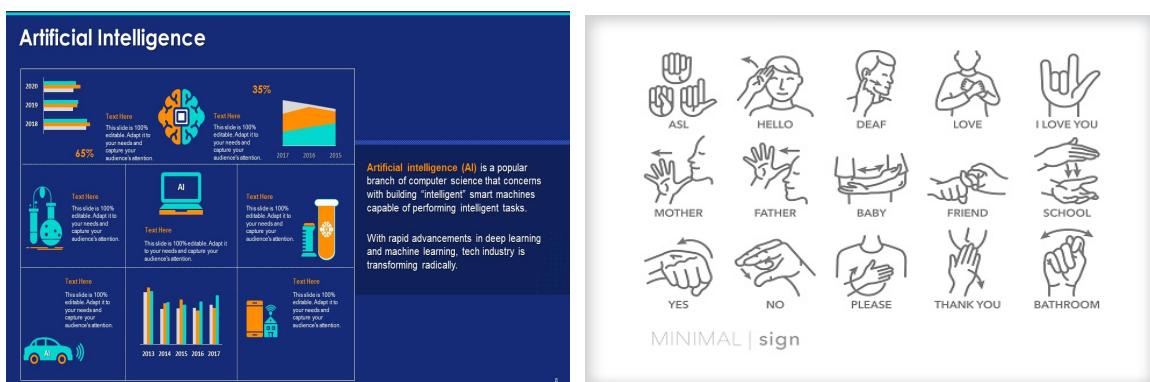


Fig. 1

## II. Choix de l'architecture

Des architectures variées ont été utilisées pour le langage de signe parmi lesquelles on peut citer:

### Le neuro flou (FNN)

Le FNN ou Fuzzy Neural Network se compose d'une couche d'entrée, d'une ou plusieurs couches cachées et d'une sous couche reliée par des poids et des fonctions d'activation. La méthode utilisée pour l'estimation du poids dans un FNN est l'algorithme d'apprentissage de la rétropropagation basé sur une approche de descente de gradient qui met à jour le réseau de manière itérative les poids dans le but de minimiser la fonction objective. L'erreur quadratique moyenne (MSE) est définie comme moyenne de différences au carré entre les valeurs observées ajustées de séries chronologiques et la fonction objective la plus courante utilisée.

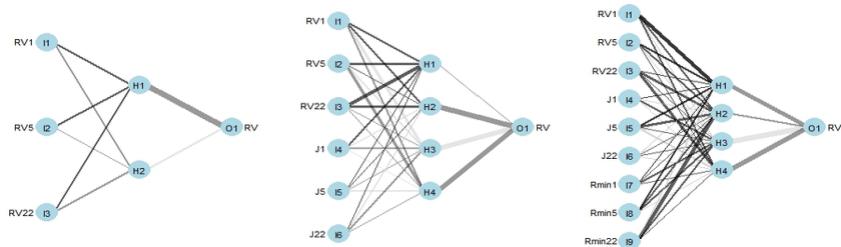


fig 2 l'architecture des FNN

Les architectures sont reprochées par leur nombre de paramètres à estimer et leur sensibilité au surajustement ce qui limite le nombre de neurones cachés à utiliser pour éviter le surajustement et pour avoir des prévisions avec plus de précision.

### Le K-NN

L'algorithme des  $k$  plus proches voisins, en anglais k-NN pour «  $k$  Nearest Neighbors » est un algorithme simple à implémenter. C'est un algorithme d'apprentissage supervisé qui permet à la fois de résoudre un problème de classification et de régression. On parle d'apprentissage supervisé lorsqu'un algorithme, pour pouvoir apprendre, a besoin de données d'entrée (c'est-à-dire, des entrées ayant une étiquette) et des données de sortie (c'est-à-dire que l'algorithme va devoir prédire par la suite) labellisées.

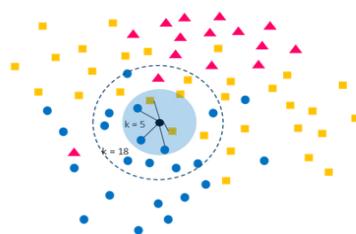


fig 2 K-NN

## 1) Principe de la méthode des k-NN

Une fois la phase d'apprentissage de l'algorithme réalisée, pour faire une prédiction à partir d'une nouvelle observation inconnue, l'algorithme trouve dans le jeu de données d'apprentissage, l'observation qui lui est la plus proche.

Ensuite, l'algorithme assigne l'étiquette de cette donnée d'apprentissage à la nouvelle observation qui était inconnue.

Le  $k$  dans la formule "k plus proches voisins" signifie qu'à la place de se contenter du seul voisin le plus proche de l'observation inconnue, nous pouvons prendre en compte un nombre fixé  $k$  de voisins du jeu d'apprentissage. Enfin, nous pouvons faire une prédiction en nous basant sur la classe majoritaire dans ce voisinage.

Le principal inconvénient de la méthode est son coût en termes de complexité. En effet, l'algorithme cherche les  $k$  plus proches voisins pour chaque observation. Ainsi, si la base de données est très grande (quelques millions de lignes), le temps de calcul peut devenir extrêmement long. Il faut donc porter une attention particulière à la taille du jeu de données d'entrée.

## Le LSTM

C'est un type spécial de réseau neuronal récurrent (RNN). De manière générale, les RNN mémorisent les informations précédentes et les utilisent pour traiter les entrées actuelles. Le défaut de ces réseaux est qu'ils ne peuvent pas se souvenir des dépendances à long terme due au gradient de disparition. Les LSTM (**Long Short Term Memory**) sont un type de RNN conçus pour gérer les dépendances à long terme.

Maraqa et al. Dans leurs travaux, ont comparé deux systèmes basés sur les réseaux de neurones : feedforward et récurrents et ils ont trouvé que le système ayant une architecture entièrement récurrente offre les meilleures performances avec une précision de 95% pour la reconnaissance des signes statiques. Ces conclusions ont beaucoup influencé le choix du modèle d'architecture LSTM pour notre travail.

## 2) Architecture de LSTM

Comme n'importe quel neurone, les neurones LSTM sont généralement utilisés en couches. Dans ce cas, **les sorties de tous les neurones sont réinjectées en entrée de tous les neurones.**

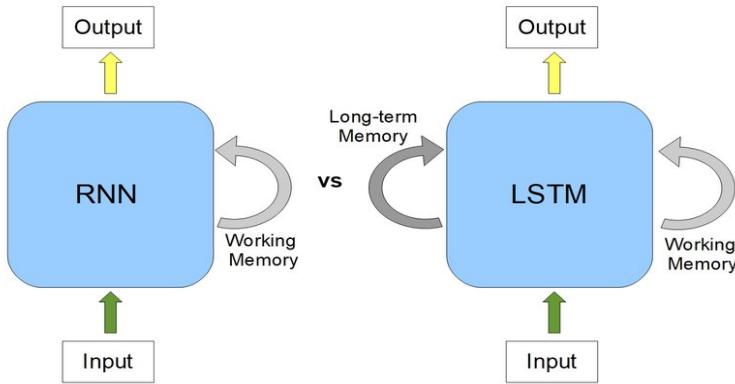


Fig. 4 architectures du RNN et LSTM

Une cellule d'un réseau LSTM est principalement composée d'un **input gate**, un **output gate** et un **forget gate**

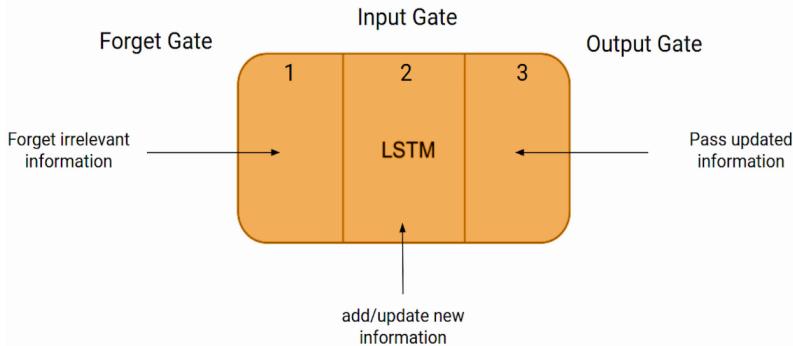


Fig.5 cellule simplifiée du LSTM

La principale idée derrière un LSTM est de diviser le signal qui traverse notre réseau en deux parties bien distinctes :

- *Court Terme* à travers le **hidden state**
- *Long Terme* à travers le **cell state**

## Étapes LSTM

Un réseau LSTM effectue durant chaque passage les 5 étapes suivantes :

- Détection des informations passées dans le **cell state** via le **forget gate**
- Choix des informations pertinentes à *long terme* à travers l'**input gate**
- Ajout des informations choisies au **cell state**
- Détection des informations importantes à *court terme* dans le **cell state**

- Génération du nouveau **hidden state** à travers l'**output gate**

La **relation de récurrence** d'un **LSTM** comprend donc une variable **h** pour le *hidden state* et une variable **c** pour le *cell state* :  $h_t, c_t = f(x_t, h_{t-1}, c_{t-1})$

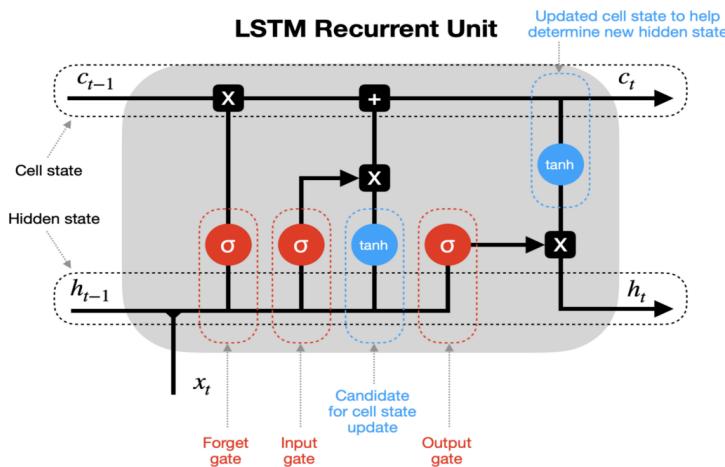


Fig.6 cellule LSTM

## Limites du modèle

Malgré le fait que les LSTM règlent en partie le **Vanishing Gradient Problem**, le modèle n'est pas pour autant parfait. En effet, il comprend de nombreux défauts parfois assez compliqués à surpasser. Parmi ces défauts, on peut citer le problème d'overfitting (quand le modèle colle trop aux données d'entraînement) ou encore le fait que le modèle soit fortement affecté par les initialisations de poids aléatoires.

## III. Implémentation

Pour implémenter notre solution nous avons eu besoin de:

### a) Ressources

- Un Mac i5 de 16Giga RAM sous CPU
- Une distribution libre et open source des langages de programmation Python et R
- Un éditeur Web Open Source : Jupyter Notebook

### b) Les librairies

Les dépendances installées sont :

- tensorflow version 2.4.1
- la librairie OpenCV de computer vision qui nous a permis de travailler avec notre webcam pour la détection en temps réel.

- le Médiapipe Holistic qui a permis l'extraction de Keypoints
- Sklearn pour évaluer, séparer le jeu de données en données d'entraînement et en données de test et aussi pour la labellisation.
- Matplotlib pour la visualisation facile de nos données.

## c) Process

Nous avons réalisé une configuration qui nous permettait d'extraire en premier les données à utiliser:

Pour cela nous avons eu accès à notre Webcam en utilisant OpenCV. En deuxième plan nous avons fait une détection en utilisant le Médiapipe. A la fin nous avions eu des landmarks représentés en des valeurs x, y , z et visibilité

Le x étant l'axe des abscisses

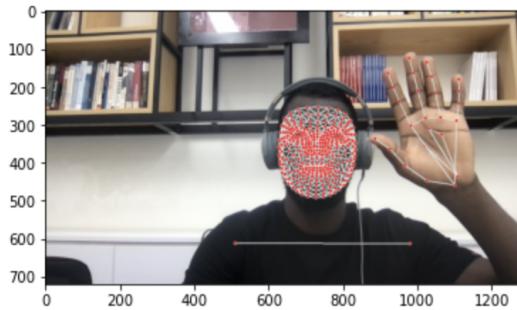
Y l'axe des ordonnées et z la distance relative au caméra. Tout ceci pour chaque partie la face les mains et la pose.

```
Entrée [126]: | results.pose_landmarks
  }
  landmark {
    x: 0.4644082486629486
    y: 0.9904323816299438
    z: 0.07747134566307068
    visibility: 0.15853455662727356
  }
  landmark {
    x: 0.593107283115387
    y: 0.9469484686851501
    z: -0.008362109772861004
    visibility: 0.17166057229042053
  }
  landmark {
    x: 0.4551478922367096
    y: 0.9965652227401733
    z: 0.01976127177476883
    visibility: 0.17772164940834045
  }
```

Nous avions enfin catapulté les résultats sur une image en dessinant les landmarks sur chaque frame pour pouvoir les visualiser au niveau des mains (droite et gauche) de la face et de la pose.

```
Entrée [33]: plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```

```
Out[33]: <matplotlib.image.AxesImage at 0x7ff6efc4f370>
```



### 3) L'extraction des valeurs des Keypoints

Nous avons ici trouvé un moyen d'extraire des valeurs des Keypoints en les mettant sur un format qui sera facilement réutilisable. Donc pour ce faire nous avons concaténé les valeurs des keypoints (pose, face, main gauche et droite) dans un tableau Numpy et à chaque fois que nous n'avons pas de valeurs on crée un Numpy zeros avec la même dimension pour compenser. Toutes ces données sont ensuite aplatis pour être utilisées plus tard par notre modèle LSTM.

### Collecte du jeu de donnée

Pour le programme notre modèle a besoin de keypoints pour l'entraînement et le Test. Nous avons décidé de collectionner notre jeu de données nous-même. Nous avons donc défini pour ce prototype la reconnaissance de 3 signes.

- I love you
- Thank You
- Hello

Pour l'ensemble de ces actions nous avons collectionné une séquence de 30 vidéos de 30 frames chacune qui sont stockées dans un dossier.

Ces séquences sont ensuite traitées, transformées en un tableau Numpy.

### 4) Pré traitement des données pour la création des features et Labelles

Nous avons travaillé avec des dépendances spéciales. la fonction `train_test_split` de Sklear pour partitionner notre dataset en données d'entraînement et en données de test. Puis les labelles en donnée catégorielle avec la fonction `to_categorical` de Keras. Ensuite un map de labelles a été créé pour représenter chacune de nos actions à déterminer.

Un dictionnaire est donc créé avec nos actions et un label a été ajouté

- hello : 0
- thanks : 1

- I love you : 3

Puis encodé sous l'encodage one hot.

Nous avons donc séparé nos données de telle sorte à avoir 5% des données en test et 95% en données de train.

### Formation et entraînement du réseau de neurone LSTM:

Nous avions utilisé le modèle séquentiel, la couche LSTM et dense pour former un réseau de neurones.

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 30, 64)	442112
lstm_7 (LSTM)	(None, 30, 128)	98816
lstm_8 (LSTM)	(None, 64)	49408
dense_6 (Dense)	(None, 64)	4160
dense_7 (Dense)	(None, 32)	2080
dense_8 (Dense)	(None, 3)	99
<hr/>		
Total params: 596,675		
Trainable params: 596,675		
Non-trainable params: 0		

Nous avons donc entraîné notre modèle avec 30 vidéos de chaque action de 30 frames chacune pour un total de 1662 keypoints pour 2000 epochs.

### Présentation des résultats:

Les prédictions du modèle sur les données de test ont donné de bons résultats, la fonction Softmax de la dernière couche dense de la sortie nous donne divers probabilité des 5 données test pour chaque action particulière.

```
In [195]: model.predict(X_test)

Out[195]: array([[5.5324531e-04, 9.9747699e-01, 1.9697514e-03],
 [1.0218851e-03, 9.9749714e-01, 1.4809018e-03],
 [9.9449158e-01, 6.3415279e-04, 4.8742960e-03],
 [9.7045529e-01, 8.5832064e-05, 2.9458890e-02],
 [9.9614292e-01, 5.7845027e-04, 3.2787006e-03]], dtype=float32)
```

Mais avec la fonction np.argmax nous ressortons la plus grande probabilité de l'action prédite.

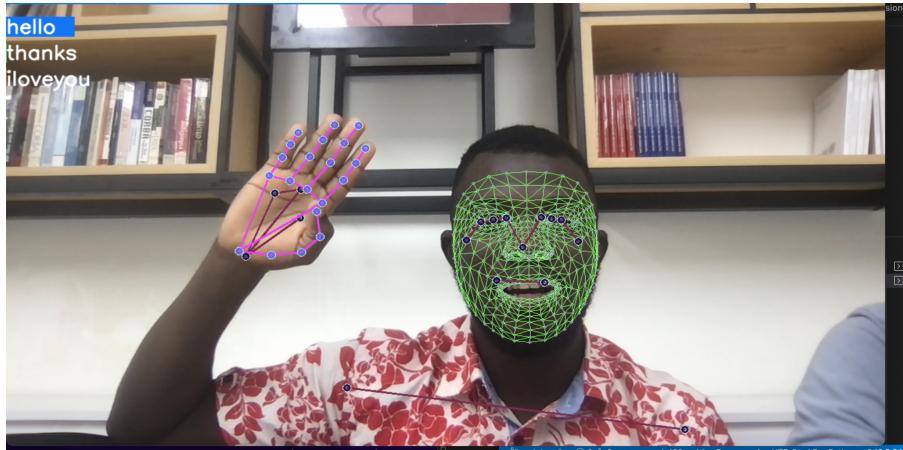
```
Entrée [66]: actions[np.argmax(res[4])]
```

```
Out[66]: 'iloveyou'
```

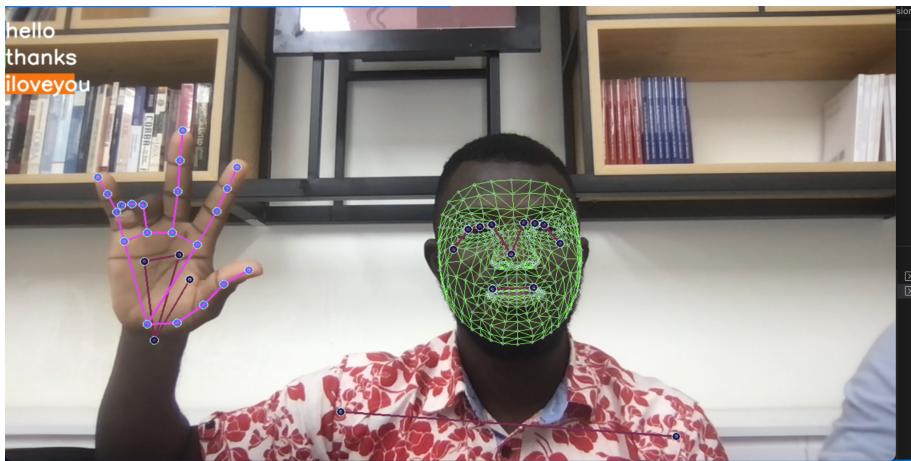
```
Entrée [67]: actions[np.argmax(y_test[4])]
```

```
Out[67]: 'iloveyou'
```

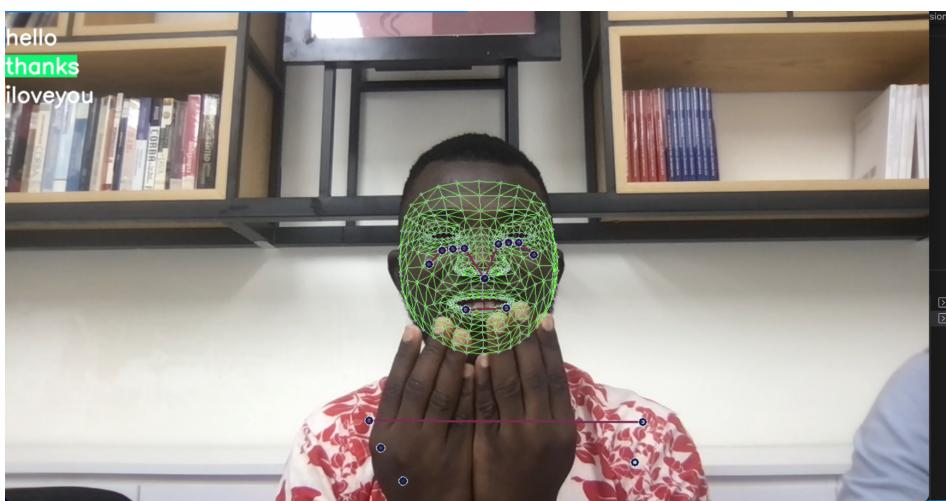
Pour la prédiction et la détection en temps réel on s'en penché sur une probabilité pour donner l'action prédictive les autres n'étant pas ignorée aussi faible qu'il en est. Ici en dessus l'action Hello a été détectée



Ici c'est plutôt I Love You



Et enfin c'est l'action Thanks.



## **IV. Difficultés**

Notre plus grande difficulté au cours de ce projet a été au niveau de la collecte de données d'entraînement. La raison première est que nous maîtrisons mal le langage des signes. La deuxième est que nous n'avions pas sous la main une personne ressource capable de nous fournir la majorité de signes à utiliser. Ceci peut prêter à équivoque la faiblesse de notre modèle mais cela n'en est pas le cas. Car avec le peu de signes que nous avons pu collecter, nous avons un meilleur comportement de notre modèle.

Le deuxième problème auquel nous avons fait face est la pauvreté des travaux connexes sur la toile. Le peu de papiers disponibles est plus spécifiques à des communautés bien précises et renvoient le plus souvent aux caractères alphabétiques et numériques à la base de du langage d'une communauté or notre orientation était trouver des signes plus universels pour des expressions usuelles.

## **V. Perspectives**

Compte tenu des difficultés rencontrées nous souhaitons modifier la méthode de collecte de données d'entraînement. Nous réfléchissons pour un module qui va s'appuyer sur une vidéo existante pour extraire un ensemble de données utiles.

Nous nous proposons également une version multilingue. Le prototype est une version anglaise or nous pensons proposer un module pour un grand nombre.

## **VI. Conclusion**

Nous avions pour objectif de proposer un modèle de détection de langage de signes, un outil social important qui faciliterait la communication entre les personnes handicapées (sourds muets) et les personnes normales. Nous avons réussi à produire un prototype qui produit des résultats intéressants. Ce prototype n'est certes pas excellent mais nous allons continuer de travailler pour rendre son fonctionnement meilleur et surtout étendre son adaptation sur plusieurs langues afin qu'il atteigne le niveau sociabilité que nous avons pensé. Sous un autre angle, ce projet a renforcé en nous le travail en équipe, la prise en compte de l'avis des équipiers et surtout le choix de l'information pertinente dans ce flux indéterminé d'informations partagées sur la toile.

## VII. Bibliographie

[Système automatique de reconnaissance de la langue de signes arabe basé sur les CNNs \(archives-ouvertes.fr\)](#)

<https://www.codespeedy.com/lstm-in-deep-learning/>

[https://www.researchgate.net/figure/FNN-type-models-architecture\\_fig1\\_326810403](https://www.researchgate.net/figure/FNN-type-models-architecture_fig1_326810403)

<https://openclassrooms.com/fr/courses/4011851-initiez-vous-au-machine-learning/4022441-en-trainez-votre-premier-k-nn>

<https://datascience.eu/fr/apprentissage-automatique/comprendre-les-reseaux-lstm/>