

Département génie logiciel

Rapport de projet de fin d'année :

Suivi de mouvement de main pour télécommander une Smart télé

Réalisé par:

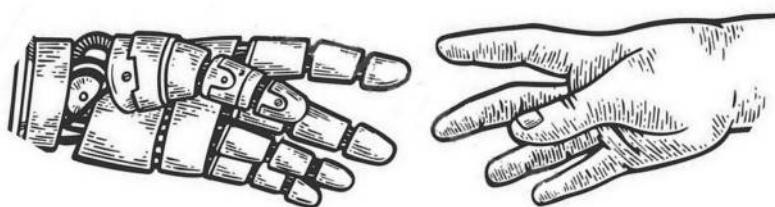
MOUMEN Reda

LALAOUI RACHIDI Omar

Encadré par le professeur: OULAD HAJ THAMI Rachid

Remerciement

. Nous adressons nos remerciements aux personnes qui nous ont aidé dans la réalisation de ce projet. En premier lieu, Nous remercions le professeur Mr Oulad Haj Thami Rachid, en tant qu'encadrant de projet, il nous a guidé dans notre travail. Nous souhaitons particulièrement remercier toute la communauté de l'intelligence artificielle marocaine et internationale pour son aide.



Résumé

Notre projet de fin d'année porte sur "*la reconnaissance des mouvements de main pour télécommander une smart télé*". Afin de le réaliser nous sommes passés par plusieurs phases : une première phase d'étude du contexte théorique du sujet, une deuxième phase de conception dans laquelle on a pu tester les différentes approches permettant de résoudre le problème, puis une troisième phase de finalisation du projet et de présentation des résultats. Le problème intègre également le fait que l'environnement dans lequel les gestes sont effectués n'est pas limité par des facteurs, c-à-d que les gestes peuvent être capturés par n'importe quel webcam standard. Nous vous présenterons donc tout au long de ce rapport, les étapes que nous avons suivis ainsi que les outils que nous avons utilisé pour réaliser ce projet.

Mots clés:

Apprentissage automatique /Apprentissage profond /Réseau de neurones.

Abstract

Our year-end project is about "*hand gesture recognition for remote control of a smart TV*". In order to achieve this we went through several phases : a first phase of study of the theoretical context of the subject, a second phase of design in which we were able to test the different approaches to solving the problem, then a third phase of project finalisation and presentation of results. The problem also integrates the fact that the environment in which gestures are made is not limited by factors, i.e gestures can be captured by any standard webcam. We are therefore presenting you throughout this report, the steps we have followed and the tools we have used to carry out this project.

Keywords:

Machine Learning/Deep Learning/Neural Network.

Liste des abréviations:

- C.N.N : Convolutional Neural Network.
- F.F.N : Feed Forward Network.
- L.S.T.M : Long Short-Term Memory.
- D.N.N : Deep Neural Network.
- 2D/3D : Two/Three Dimensional.
- ReLU : Rectified Linear Unit.
- SGD : Stochastic Gradient Descent.

Liste des tables:

- Table 1: Les cours de la spécialisation en Deep Learning.
- Table 2: Répartitions des gestes (2D-CN).
- Table 3: Récapitulatif sur la base de données 20BN-Jester.
- Table 4: Répartitions des classes (3D-CN).
- Table 5: Tableau des précisions d’entraînement top 1 & 3 (Nombre des epochs VS précision).
- Table 6: Validation d’entraînement top 1 & 3.
- Table 7: Les Gestes et leurs commandes respectives.

Liste des graphes:

- Graphe 1: Graphe de la précision d’entraînement top 1 & 3 (Nombre des epochs VS précision).
- Graphe 2: Graphe de la validation d’entraînement top 1 & 3 (Nombre des epochs VS précision).
- Graphe 3: Graphe de la perte d’entraînement & validation (Nombre des epochs VS perte).
- Graphe 4: Comparaison du modèle avec et sans la normalisation des batchs (Nombre des epochs VS précision).

Liste des figures:

- fig 2: Architecture CNN.
- fig 3: Structure générale d’un réseau de neurones à propagation avant (feed forward).
- fig 4: Exempne d’une opération de convolution.
- fig 5: Exemple de MAX Pooling.
- fig 6: Architecture 3D-CNN.
- fig 7: Logo python.

- [fig 8](#): Logo Keras+Tensorflow.
- [fig 9](#): Logo numpy.
- [fig 10](#): Logo pandas.
- [fig 11](#): Logo matplotlib.
- [fig 12](#): Logo google colab.
- [fig 13](#): Vue de l'outil colab.
- [fig 14](#): Logo OpenCV.
- [fig 15](#): Architecture 2D-CNN utilisée.
- [fig 16](#): Le modèle d'architecture 2D-CNN utilisé.
- [fig 17](#): Capture d'images.
- [fig 18](#): Capture d'images.
- [fig 19](#): Exemples d'images capturées des différentes classes.
- [fig 20](#): Classification de la classe NOTHING.
- [fig 21](#): Classification de la classe STOP.
- [fig 22](#): Classification de la classe PUNCH.
- [fig 23](#): Classification de la classe OK.
- [fig 24](#): Classification de la classe PEACE.
- [fig 25](#): Exemples des gestes performés.
- [fig 26](#): Pytorch vs TensorFlow.
- [fig 27](#): Architecture 3D-CNN utilisée.
- [fig 28 & 29](#): Le modèle d'architecture 3D-CNN utilisé.
- [fig 30-37](#): Classes des gestes utilisées.
- [fig 42](#): CNN3D + RNN.
- [fig 43](#): Unité LSTM.
- [fig 44](#): Livres utilisés.
- [fig 45](#): Vue de l'interface de démonstration.

Table des matières :

i.	Introduction.	9
ii.	Cahier des charges.	10
iii.	Cadre théorique & connaissances générales.	11
1	<i>C.N.N : Convolutional Neural Networks</i>	11
1.1	Couche de convolution	12
1.2	Couche de Pooling	12
1.3	Couche entièrement connectée	12
1.4	Couche de sortie	12
2	<i>3D-CNN</i>	13
3	Formation en apprentissage profond :	13
iv.	Outils et Frameworks utilisés.	14
v.	Méthodologie	16
1	Première approche : 2D-CNN	16
1.1	<i>Architecture</i>	16
1.2	<i>Code du modèle</i>	17
1.3	<i>Collection base de données/Dataset</i>	17
1.4	<i>Preprocessing</i>	19
1.5	<i>Résultats</i>	20
2	Deuxième approche : 2D-CNN avec concaténation d'images	23
2.1	<i>Preprocessing</i>	24
2.2	<i>Résultat</i>	24
3	Troisième approche : 3D-CNN	24
3.1	<i>Framework utilisé</i>	24
3.2	<i>Architecture</i>	25
3.3	<i>Code du modèle</i>	26
3.4	<i>Dataset/base de données</i>	26
3.5	<i>Preprocessing</i>	28
3.6	<i>Résultat</i>	28
vi.	Interface de démonstration	31
vii.	Conclusion	32
viii.	Bibliographie	33

Introduction

Le problème de la reconnaissance des gestes de main est inclus dans le domaine du traitement d’images, ce dernier consiste à donner aux ordinateurs la capacité de traiter des images et des vidéos d’une façon similaire à celle de l’humain. L’objectif donc est de permettre une vision extérieure et une compréhension basée sur des algorithmes. La richesse de ce domaine en terme de recherche et d’innovation est due au grand nombre de ses applications : marquage par images et vidéos, conduite automatique, interaction homme-machine via le gestuelle de la main etc. Dans ce même contexte de recherche on trouve la sous-catégorie de la reconnaissante et l’interprétation des gestes humains. Un geste ici est défini comme n’importe quel mouvement physique ,non-verbal, petit ou large d’une personne qui a l’intention de communiquer un message ou instruction. Dans cette optique tout les mouvements seront manuels et notre problème serait donc de concevoir un algorithme permettant la communication avec l’ordinateur via ces gestes : un problème assez populaire pour ses application telles les robots chirurgiens, interprétation du langage des signes et beaucoup d’autres. Notre application vise alors à procurer une reconnaissance efficace et efficiente utilisant des entrées sous forme de vidéos depuis le Webcam standard qui est disponible, ainsi créer une technologie disponible facilement et partout. Remplacer des images par des vidéos ajoute une dimension temporelle à notre problème, ainsi d’autres facteurs tels que l’arrière plan et la luminosité. Après avoir obtenu une architecture résolvante efficace ,ce module va être intégré dans une interface graphique simple à utiliser arrangeant les gestes possibles indisponibles pour télécommander une smart télé.

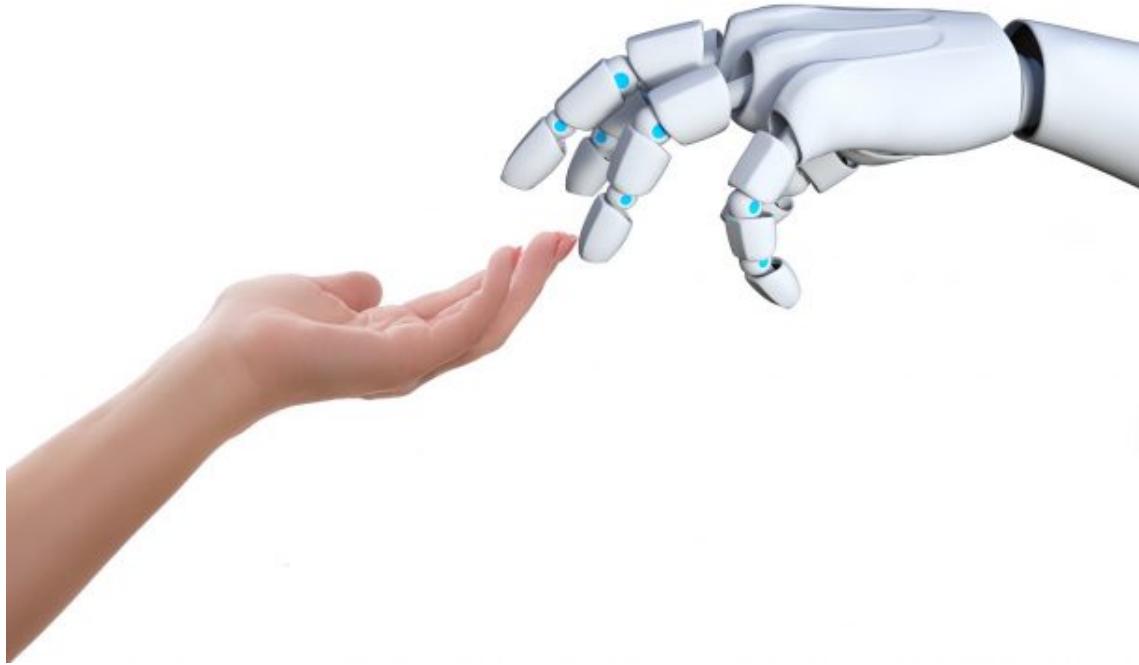


Figure 1:

Cahier des charges:

1. Maîtrise des connaissances théoriques et des outils de travail.
2. Étude du problème en profondeur et évaluation des différentes approches.
3. Recherche de la base de données.
4. Pré-traitement sur la base de donnée et préparation pour l'entraînement.
5. Conception des modèles 2d/3d.
6. Entraînement et études des résultats.
7. Création de l'interface graphique avec OpenCV.

Cadre théorique & connaissances générales

1 C.N.N : Convolutional Neural Networks

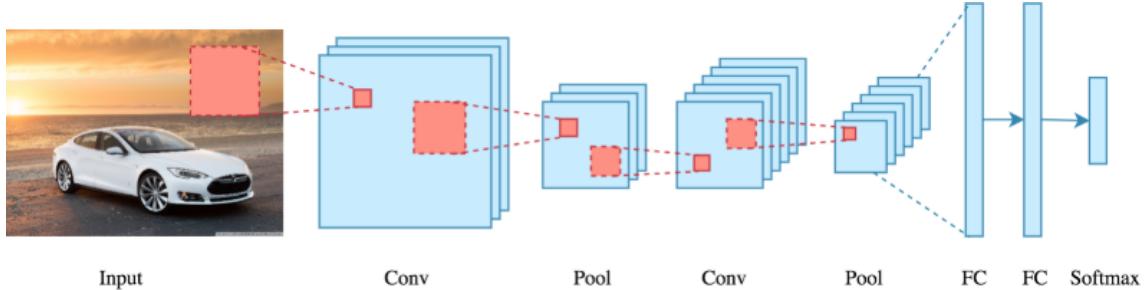


Figure 2: Architecture CNN

C'est une classe d'algorithmes d'apprentissage profond dont le réseau de neurones est de type *feed forward*, et dont les applications sont dédiées à la reconnaissance des images et vidéos. La caractéristique principale qui différencie cet algorithme et un réseau de neurones conventionnel est la couche de convolution. Une architecture CNN se compose de 4 couches :

- Couche de convolution
- Couche de Pooling
- Couche entièrement connectée
- Couche de sortie

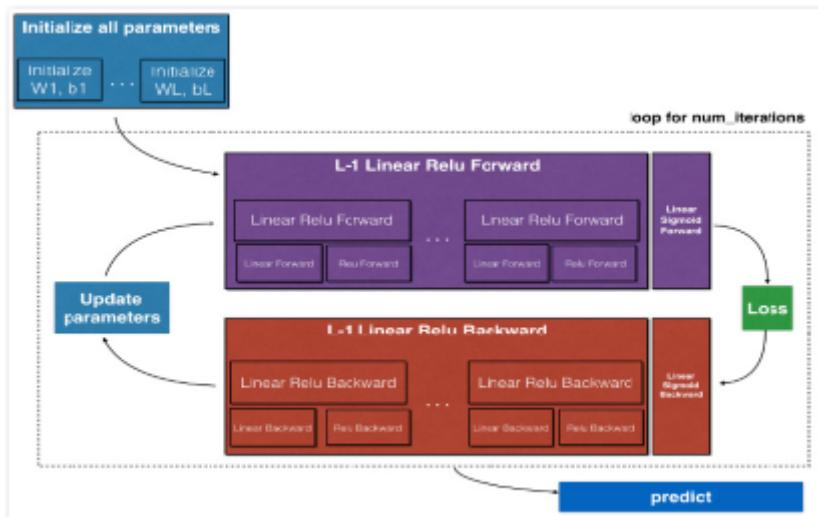


Figure 3: Structure générale d'un réseau de neurones à propagation avant (feed forward)

1.1 Couche de convolution

C'est la couche principale pour établir cette structure, elle fonctionne ainsi : Chaque opération de convolution est suivie de l'application d'une fonction d'activation i.e ReLU. Un réseau de neurones conventionnel apprend à classifier les données d'entrée, mais pratiquement il n'est pas fesable. Le nombre des neurones utlisés sera gigantesque même pour les réseaux peu profonds. Par exemple pour une image de taille 75x75, la taille des poids pour chaque neurone de la seconde couche serait 5625. La couche de convolution résout ce problème en dépit de la taille d'image, un filtre de taille 3x3 par exemple peut être utilisé pour extraire des caractéristiques avec un total de 9 paramètres (figure 1). Cette couche alors permet d'avoir un réseau profond avec peu de paramètres.

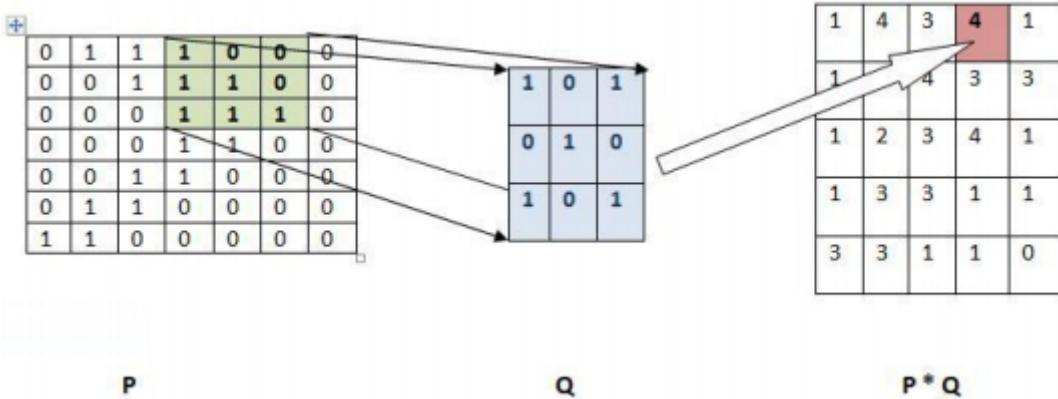


Figure 4: Exemple d'une opération de convolution.

1.2 Couche de Pooling

Cette couche réduit progressivement la taille de la sortie de la couche de convolution, ce qui en redit le nombre des paramètres et ainsi les opérations de calcul (figure 1.2).

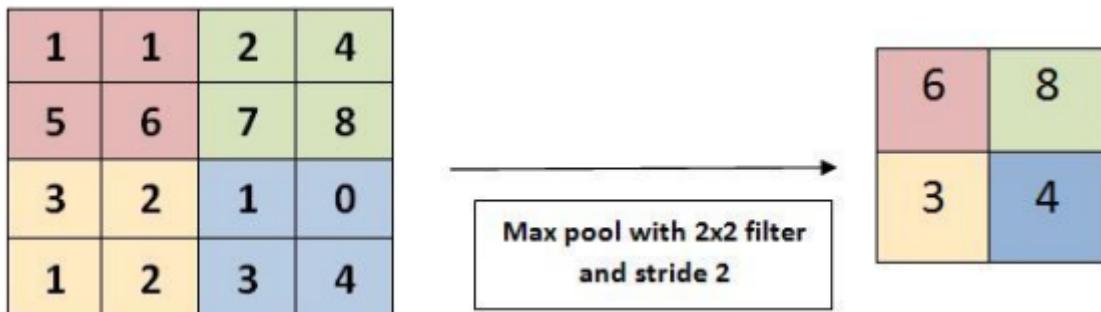


Figure 5: Exemple de MAX Pooling.

1.3 Couche entièrement connectée

C'est le réseau de neurones conventionnel où chaque neurone d'une couche est connectés à tous les autres neurones de la couche suivante.

1.4 Couche de sortie

Composée de neurones de nombre égal au nombre de classes, c'est à ce niveau qu'on applique la fonction Softmax.

2 3D-CNN

Utilisée principalement pour classifier les vidéos. La différence est que la donnée d'entrée est un tenseur 3D : comme un tenseur 2D on trouve les dimensions d'une image : la hauteur et la largeur, en ajoutant la dimension temporelle (profondeur de l'image). Les étapes pour établir l'entrée de cette couche :

1. Créer des frames d'images 2D pour chaque vidéo dans des intervalles de temps discrets.
2. Empiler ces frames en profondeur pour obtenir un tenseur 3D.

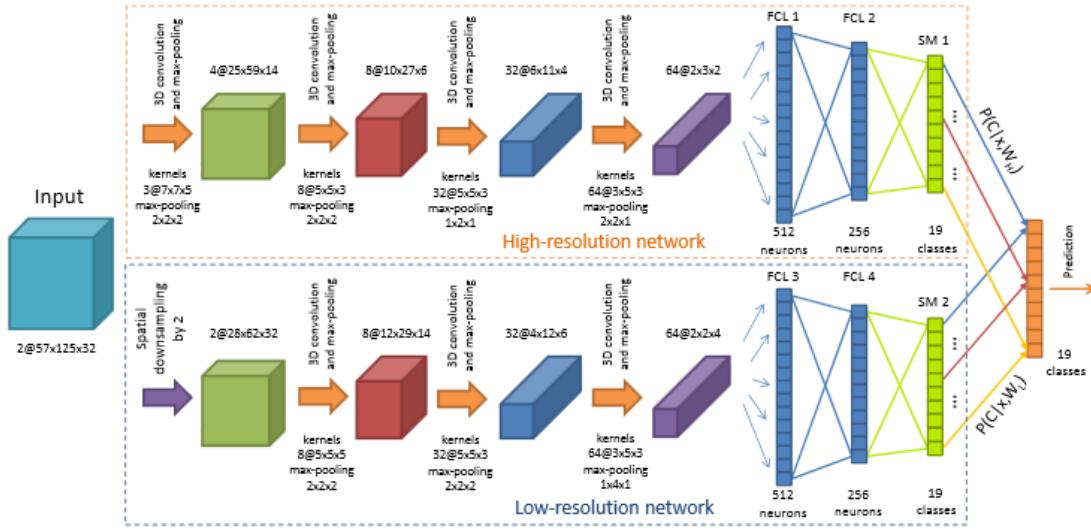


Figure 6: Architecture 3D-CNN

3 Formation en apprentissage profond :

Certifié dans les cinq cours proposés par la compagnie [deeplearning.ai](#) dirigée par *Andrew Ng*, nous avons acquéri la spécialisation en *Deep learning*.



Figure 7:

Cours 1	Neural Networks and Deep Learning.
Cours 2	Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization.
Cours 3	Structuring Machine Learning Projects.
Cours 4	Convolutional Neural Networks.
Cours 5	Sequence Models.

Table 1: Les cours de la spécialisation en Deep Learning

Outils et Frameworks utilisés:

- *Python comme langage de programmation.*



Figure 8:

- *Keras, TensorFlow & PyTorch comme bibliothèque de Machine Learning et de calcul numérique.*



Figure 9: Frameworks

- *Numpy, Pandas & Matplotlib respectivement comme bibliothèque de calcul numérique et matriciel, manipulation et d'analyse des données et dessin des graphes. Ces outils seront manipulés durant tout le projet*



Figure 10: logo1

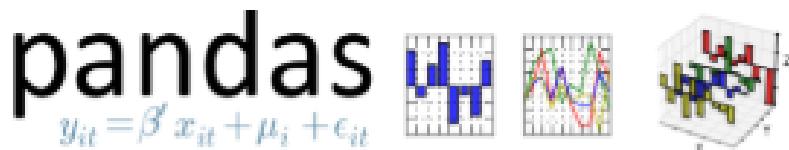


Figure 11: logo2



Figure 12: logo3

- **Google Colaboratory** : Utilisé dans tout le projet comme environnement Jupyter et en tant que machine de calcul.



Figure 13: Google colabolatory

```
[3] !nvidia-smi
Tue Jun 4 14:41:41 2019
+-----+
| NVIDIA-SMI 418.67      Driver Version: 410.79      CUDA Version: 10.0 |
| GPU  Name Persistence-MI Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
+-----+
| 0  Tesla T4           Off  00000000:00:04.0 Off   0 |
| N/A  39C   P8    15W / 70W |     0MiB / 15079MiB |      0%     Default |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU     PID  Type  Process name        Usage  |
|+-----+
| No running processes found               |
+-----+


!git clone https://github.com/JihongJu/keras-resnet3d

Cloning into 'keras-resnet3d'...
remote: Enumerating objects: 100, done.
remote: Total 100 (delta 0), reused 0 (delta 0), pack-reused 100
Receiving objects: 100% (100/100), 20.46 KiB | 6.82 MiB/s, done.
Receiving deltas: 100% (40/40), done.

[ ] from resnet3d import Resnet3DBuilder
model = Resnet3DBuilder.build_resnet_50(18, 128, 128, 31, 27)
```

Disque [] 333.66 GB disponible(s)

Figure 14: Vue de l'outil

- **OpenCV** comme bibliothèque graphique de traitement d'images, et de création d'interface graphique.



Figure 15: OpenCV

Méthodologie

L'objectif du projet est de classifier des gestes humains pour de nombreuses applications dont la notre qui est la télécommandation d'une smart télé. Pour faire, on va entamer une étude comparative entre différentes architectures.

1 Première approche : 2D-CNN

1.1 *Architecture*

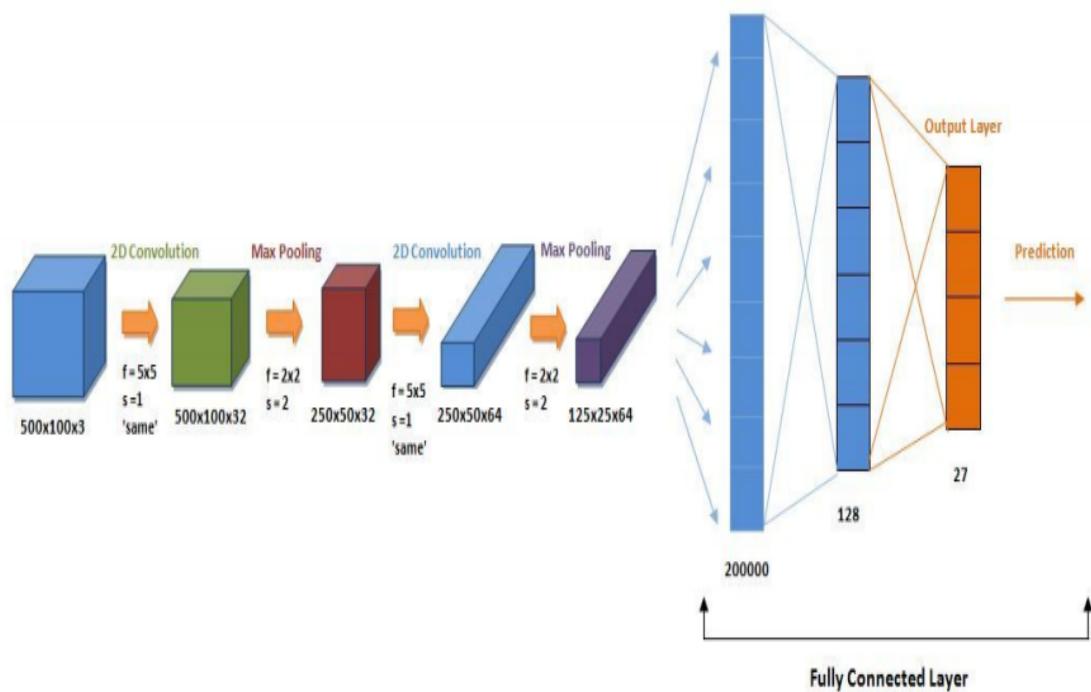


Figure 16: Architecture 2D-CNN utilisée

1.2 Code du modèle

```
def cnnModel():
    model = Sequential()
    model.add(Conv2D(nb_filters[0], (nb_conv, nb_conv),
                   strides = (2, 2),
                   padding='valid',
                   input_shape=(IMG_SIZE, IMG_SIZE, img_channels)))
    convout1 = Activation('relu')
    model.add(convout1)
    model.add(Conv2D(nb_filters[1], (nb_conv, nb_conv)))
    convout2 = Activation('relu')
    model.add(convout2)
    model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
    model.add(Conv2D(nb_filters[2], (nb_conv, nb_conv)))
    convout3 = Activation('relu')
    model.add(convout3)

    model.add(Flatten())
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(Dense(nb_classes))
    model.add(Activation('softmax'))
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy',
                  optimizer =sgd,
                  metrics=[ 'accuracy'])
    return model
```

Figure 17: le modèle d'architecture 2D-CNN utilisé

1.3 Collection base de données/Dataset

Les images ont été collectées manuellement via une application faite avec OpenCV, et ont été structurées de la façon suivante: 16200 images en total distribuées sur 5 classes :

Geste	Nombre d'images/frames
OK	3240
PEACE	3240
STOP	3240
PUNCH	3240
NOTHING	3240

Table 2: Répartition des gestes (2D-CNN)

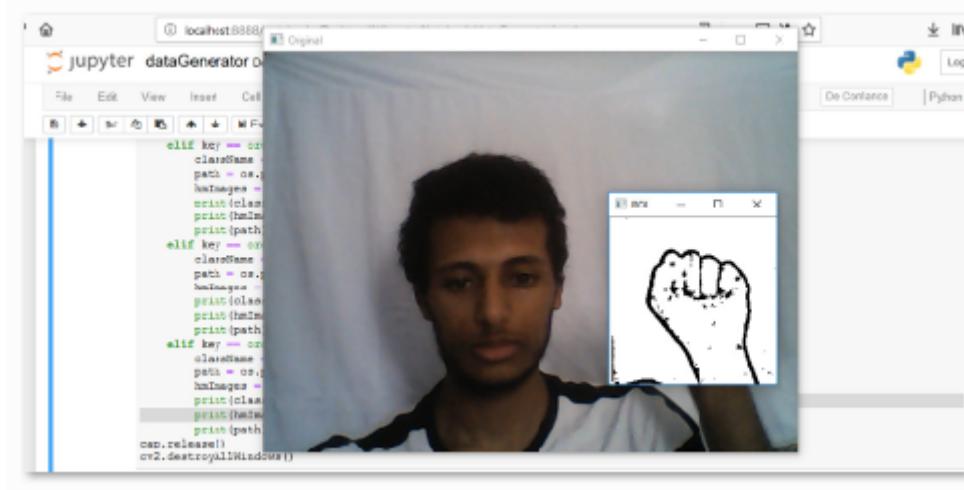


Figure 18: Capture d'images

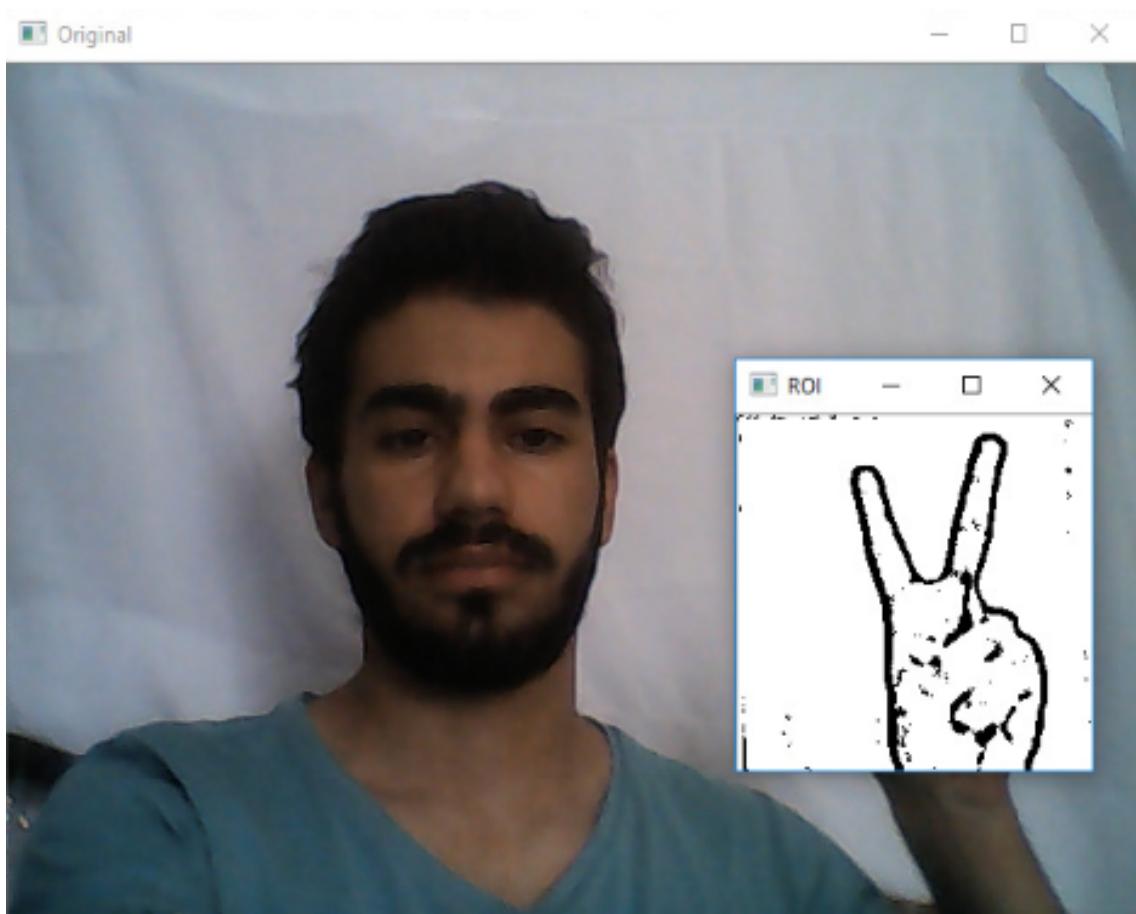


Figure 19: Capture d'images

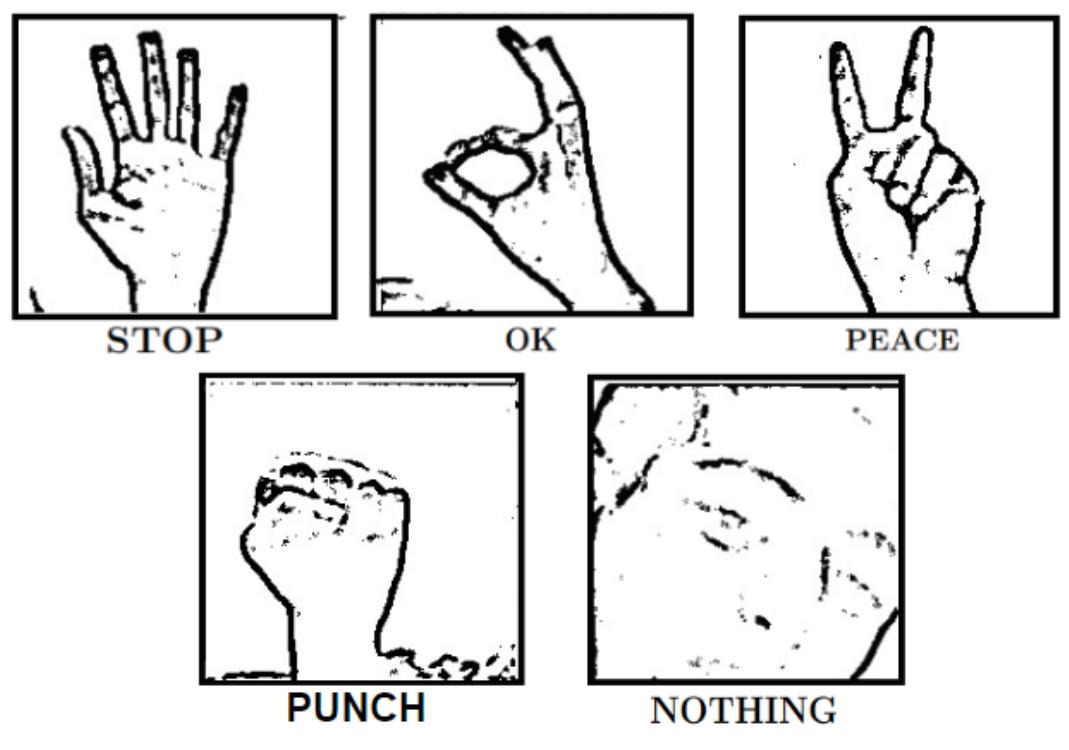


Figure 20: Exemples d’images capturées des différentes classes

1.4 *Preprocessing*

Le modèle prend comme input un tenseur 4D de dimension $(N, 200, 200, 3)$. Donc les images constitutants la base de donnée et qui sont de dimension $(200, 200)$ sont transformés en un tenseur de dimension $(200, 200, 3)$ puis concaténés pour former N images prêtes à être passées dans le modèle. Le nombre N représente la taille du lot.

1.5 Résultats

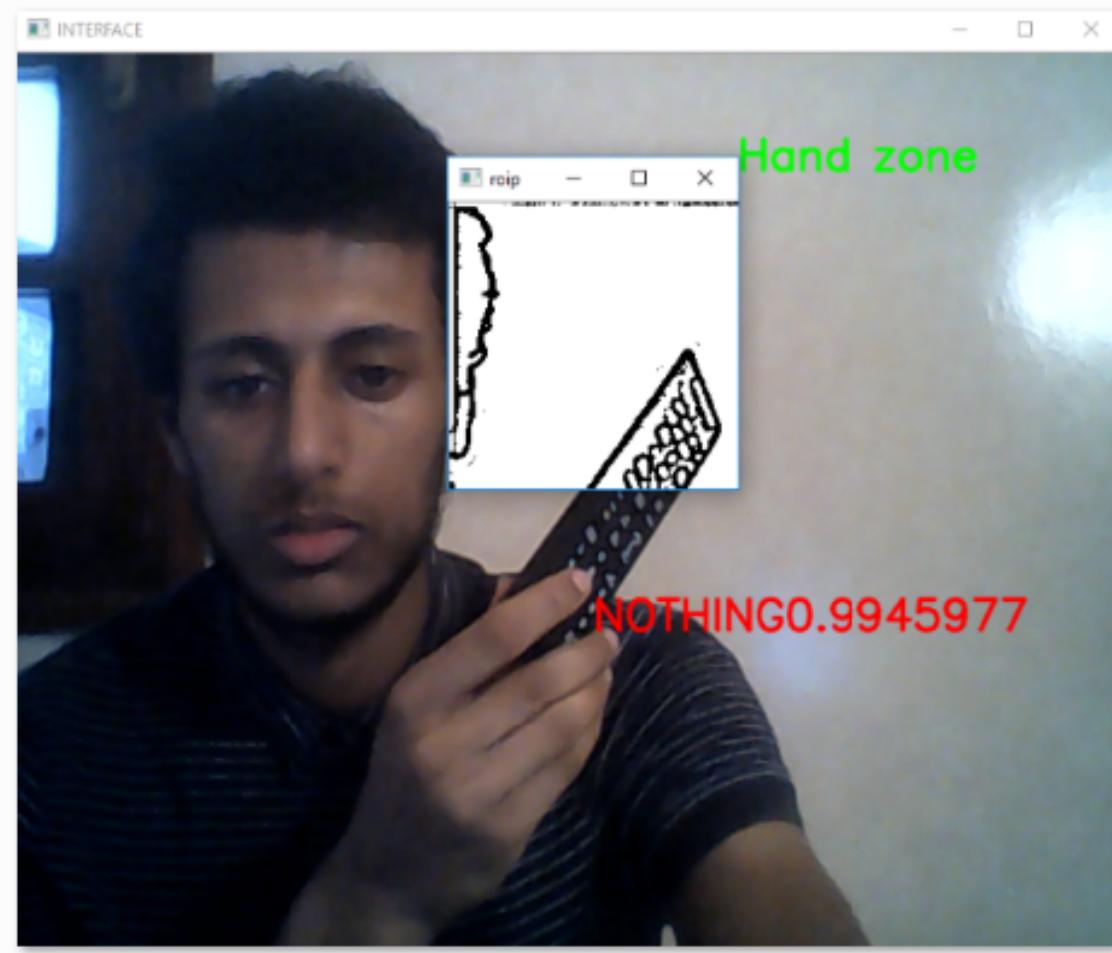


Figure 21: Classification de la classe NOTHING

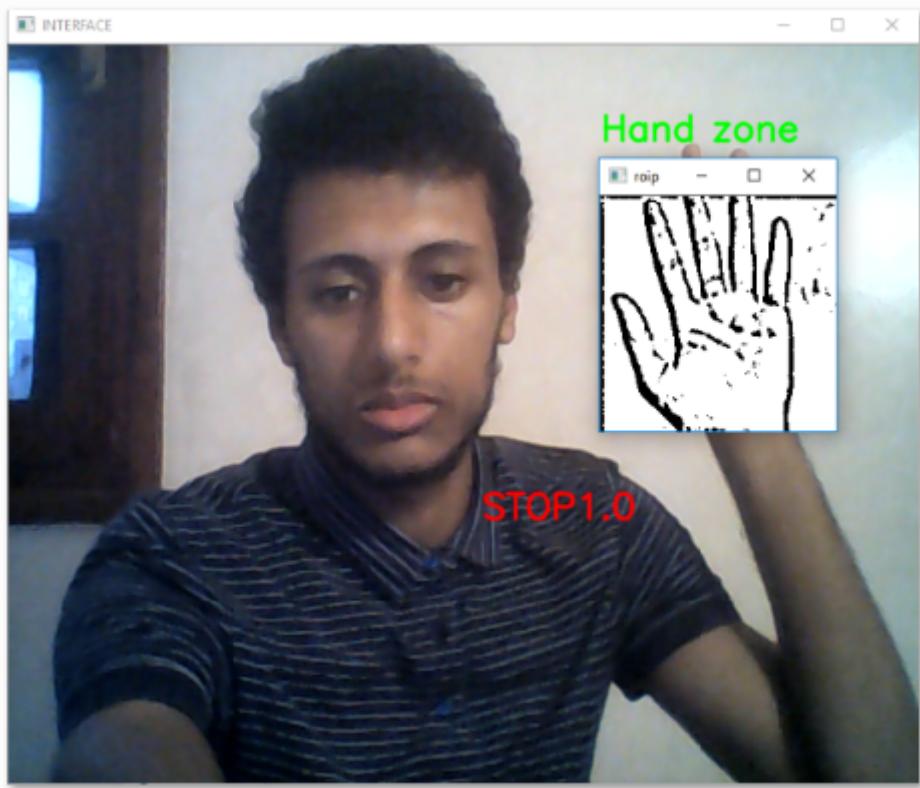


Figure 22: Classification de la classe STOP



Figure 23: Classification de la classe PUNCH.

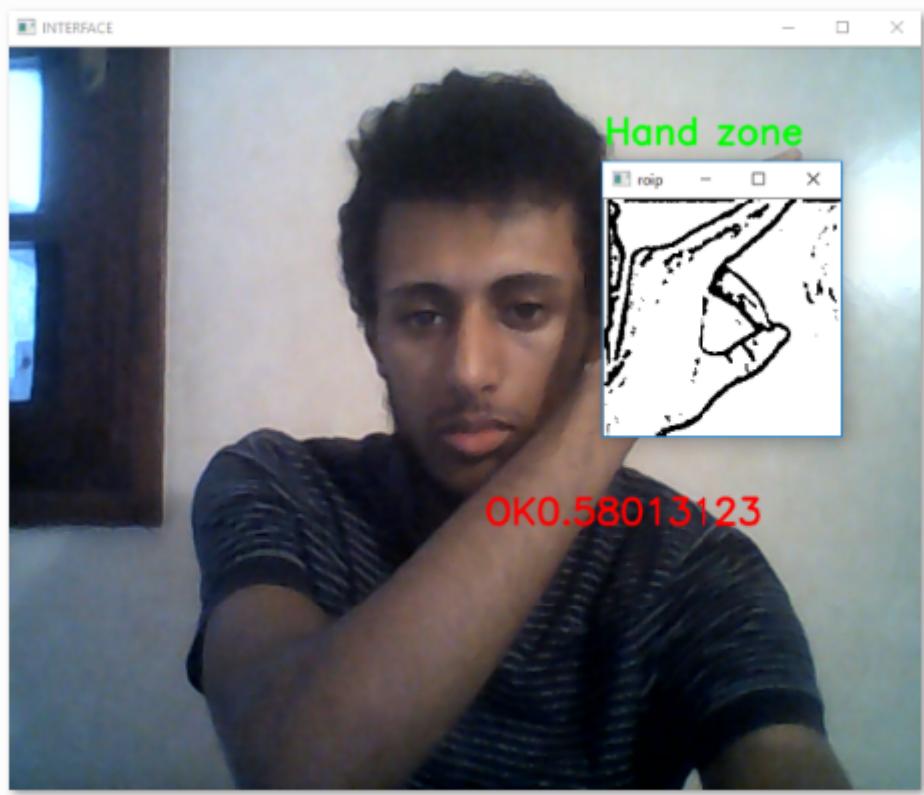


Figure 24: Classification de la classe OK.

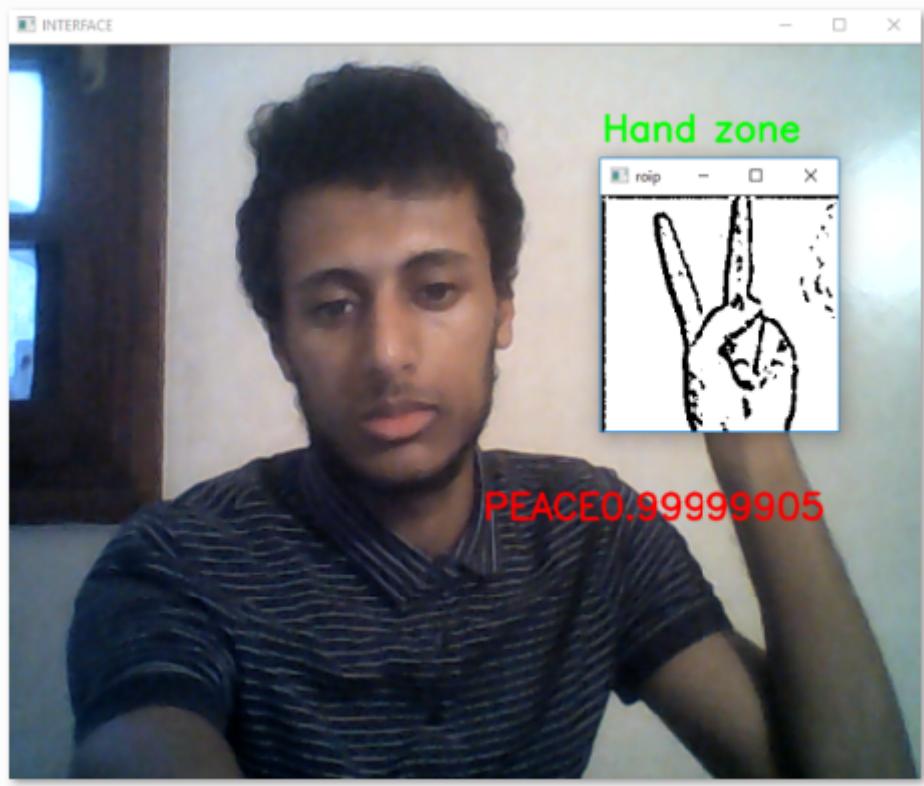


Figure 25: Classification de la classe PEACE.

Problème de surapprentissage/Overfitting

Après la phase d'entraînement ,on peut remarquer que le modèle a remarquablement

performé pour classifier les données d'entraînement (99% comme précision après 5 epochs). Il a bien classifié *peace*, *stop* et *nothing* mais a mélangé entre *punch* et *stop* et entre *ok* et *peace*, ce phénomène peut être expliqué par l'insuffisance des données et par les masques binaires (binary-masks) qui font perdre à ces images leurs détails et caractéristiques.

Une des solutions est d'entraîner plus de data.

2 Deuxième approche : 2D-CNN avec concaténation d'images

20BN-Jester dataset

Le Dataset 20BN-JESTER est une grande collection de clips vidéo densément étiquetés qui montrent des humains effectuant des gestes de main pré-définis devant une caméra portable ou webcam. Cet ensemble de données a été créé par un grand nombre de contributeurs parmi une large communauté de l'AI. Il permet de former des modèles d'apprentissage pour reconnaître les gestes de main humains. Il est disponible gratuitement pour la recherche universitaire. Les licences commerciales sont disponibles sur demande.

Format

Les données vidéos sont fournies sous la forme d'une grande archive de type TGZ, divisée en parties de 1 Go maximum. La taille totale du téléchargement est de 22,8 Go. L'archive contient des répertoires numérotés de 1 à 148092. Chaque répertoire correspond à une vidéo et contient des images JPG avec hauteur 100px et largeur variable. Les images JPG ont été extraites des vidéos originales à 12 images par seconde. Les noms des fichiers des JPG commencent à 00001.jpg. Le nombre de JPGs varie selon la longueur des vidéos originales.

20BN-Jester dataset	
Nombre total des vidéos	148,092
Ensemble entraîné	118,562
Ensemble validé	14,787
Ensemble de test (w/o)	14,743
Labels	27

Table 3: Recap sur la base de données 20BN-Jester

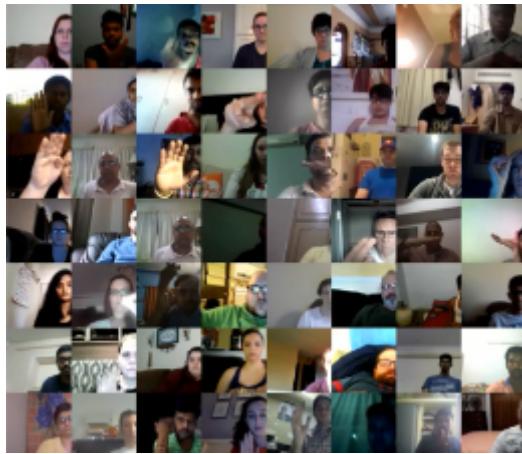


Figure 26: Exemples des gestes performés

2.1 *Preprocessing*

On va garder les mêmes outils de travail, architecture et classes. L'entraînement est fait sur un dataset plus large en concaténant un nombre de frames fixé à 30, Si ce nombre y est inférieur on ajoute des frames de zeros. Toutes les 30 frames/images sont concaténées pour obtenir une seule de taille 3000x100,et puisque ces images sont de format RGB on aura 3000x100x3. Entrée finale de notre architecture est obtenue après redimensionnement en 500x100x3.

2.2 *Résultat*

Problème de sousapprentissage/underfitting

Les résultats obtenus n'étaient pas proméants, comme la précision a stagné entre deux bornes (8% - 15%) après 15-20 epochs même après plusieurs tests dont le benchmark est 96,6%. Ceci est due à l'architecture peu profonde et le nombre de paramètres limités qui ont entraîné un apprentissage moins efficient. Une autre cause est que les frames ont été redimensionnées en le **1/6** éme de sa taille d'origine donc une perte de détails ainsi une fausse classification.

Il faut explorer d'autres architectures plus profonde.

3 Troisième approche : 3D-CNN

3.1 *Framework utilisé*

PyTorch comme bibliothéque de machine learning et outil de calcul tensoriel.

pytorch VS TensorFlow:

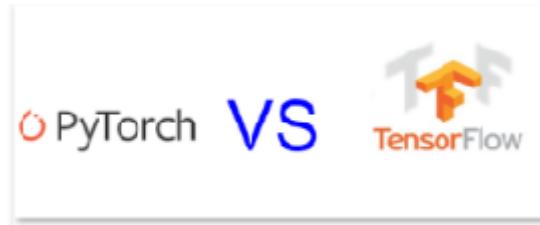


Figure 27:

TensorFlow utilise des graphes statique , après avoir configurer le graphe de calcul il exécute ceci plusieurs fois. Tandis que Pytorch utilise des graphes dynamiques. On peut voir cela comme «définir en exécutant» et «définir puis exécuter». Ainsi TensorFlow est populaire dans le domain d'industrie et production, et Pytorch dans le domain de la recherche. Tensorflow est difficile en terme de réglage et debogage mais, mais stable dans l'utilisation une fois qu'il est prêt, tandis que Pytorch est facile à expérimenter mais nécessite quelques efforts pour le mettre en production. En global Pytorch est caractérisé par la simplicité et l'accessibilité. Nous voulons aussi non découvrir plusieurs frameworks pour pouvoir tester le meilleur et le plus adaptable à notre projet et pour nos futurs projets .

3.2 Architecture

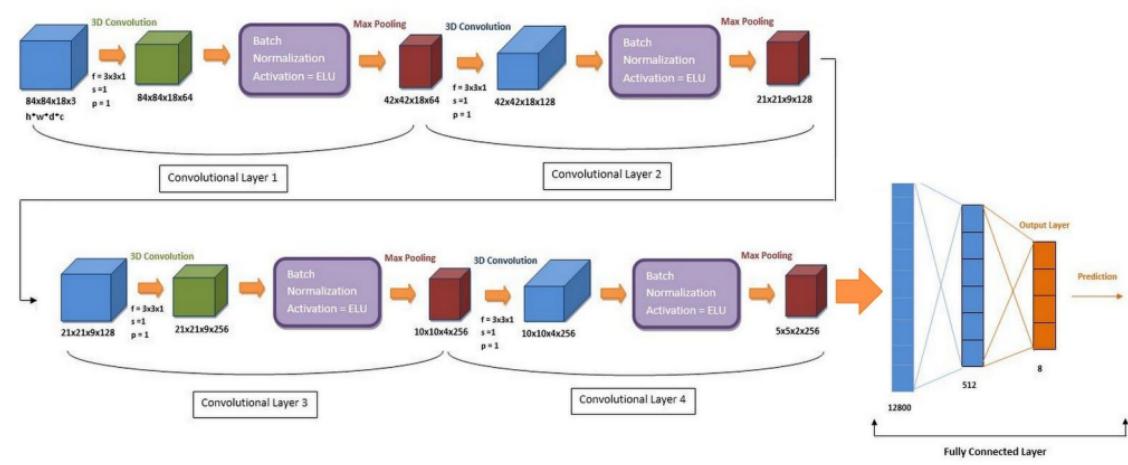


Figure 28: Architecture 3D-CNN utilisée

3.3 Code du modèle

```
class ConvColumn(nn.Module):

    def __init__(self, num_classes):
        super(ConvColumn, self).__init__()
        #print("in init")
        self.conv_layer1 = self._make_conv_layer(3, 64, (1, 2, 2), (1, 2, 2))
        self.conv_layer2 = self._make_conv_layer(64, 128, (2, 2, 2), (2, 2, 2))
        self.conv_layer3 = self._make_conv_layer(
            128, 256, (2, 2, 2), (2, 2, 2))
        self.conv_layer4 = self._make_conv_layer(
            256, 256, (2, 2, 2), (2, 2, 2))

        self.fc5 = nn.Linear(12800, 512)
        self.fc5_act = nn.ELU()
        self.fc6 = nn.Linear(512, num_classes)

    def _make_conv_layer(self, in_c, out_c, pool_size, stride):
        conv_layer = nn.Sequential(
            nn.Conv3d(in_c, out_c, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm3d(out_c),
            nn.ELU(),
            nn.MaxPool3d(pool_size, stride=stride, padding=0)
        )
        return conv_layer
```

Figure 29:

```
def forward(self, x):
    x = self.conv_layer1(x)
    x = self.conv_layer2(x)
    x = self.conv_layer3(x)
    x = self.conv_layer4(x)

    x = x.view(x.size(0), -1)

    x = self.fc5(x)
    x = self.fc5_act(x)

    x = self.fc6(x)
    return x
```

Figure 30: Le modèle d'architecture 3D-CNN utilisé

3.4 Dataset/base de données

8 classes ont été choisies pour leurs pertinences à décrire des mouvements de télécommandation parmi 27 classes.

Geste	Nombre des Vidéos
Tirer deux doigts	5315
Secouer la main	5314
Glissement de deux doigts en bas	5410
Glissement de deux doigts en haut	5345
Glissement de deux doigts à gauche	5244
Glissement de deux doigts à droite	5262
Stop	5413
Aucun geste	5344

Table 4: Répartitions des classes (3D-CNN)

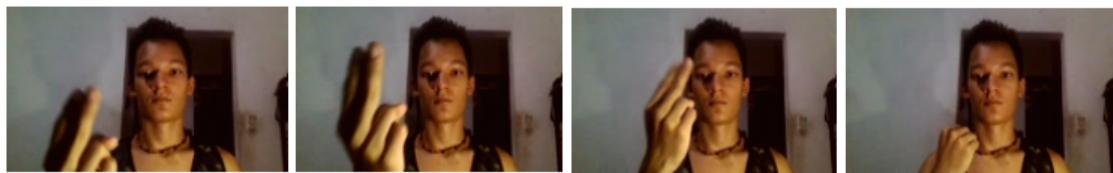


Figure 31: classe 1 : Tirer deux doigts



Figure 32: classe 2 : Secouer la main

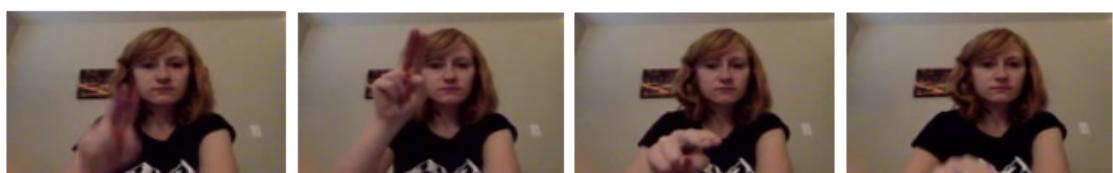


Figure 33: classe 3 : Glissement de deux doigts en bas

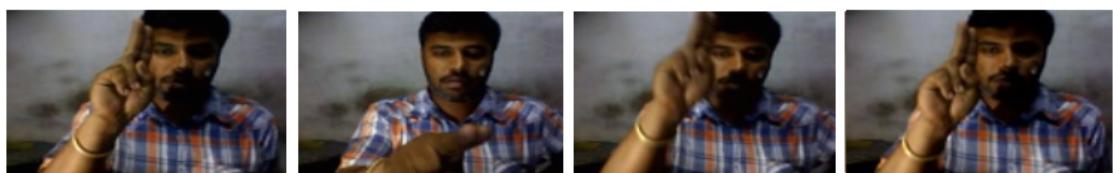


Figure 34: classe 4 : Glissement de deux doigts en haut



Figure 35: classe 5 : Glissement de deux doigts à gauche

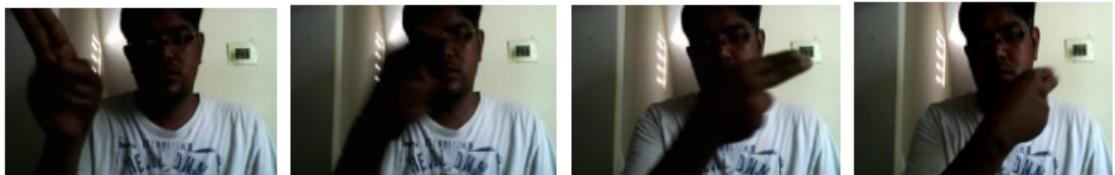


Figure 36: classe 6 : Glissement de deux doigts à droite

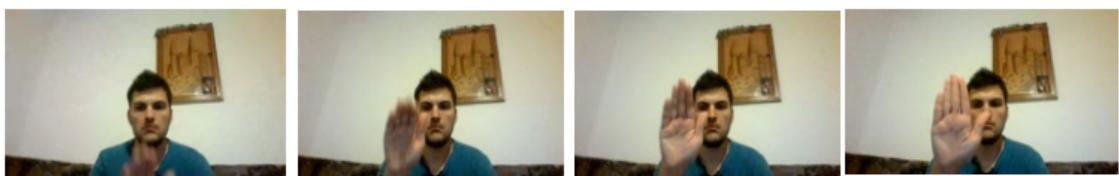


Figure 37: classe 7 : Stop

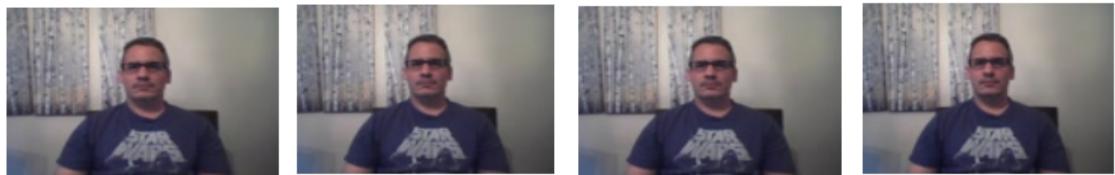


Figure 38: classe 8 : Aucun geste

3.5 Preprocessing

Similaire à celui du 2D-CNN, on a pris des vidéos sous forme de frames dans des intervals temporels discrets. Chaque frame est enregistré sous taille 176x100 qui est ensuite découpé au milieu pour obtenir une image de taille 84x84. 18 frames de chaque vidéo sont empilés séquentiellement le long de la profondeur donnant ainsi un tenseur 5D de taille 84x84x18. Ce dernier est normalisé après par une moyenne nulle et une déviation standard de 1.

3.6 Résultat

Après les pauvres résultats du convolution 2D, l'architecture 3D-CNN a montré des performances plus satisfaisantes. L'entraînement pour cette dernière a duré 3 heures en utilisant *Tesla4 16GO*, pour 12 epochs avec un coefficient d'apprentissage $\alpha = 0,01$ et la fonction de perte est de type *Cross-Entropy*:

$$L(y, p) = - (y \log(p) + (1 - y) \log(1 - p))$$

La mise à jour des poids est assurée par l'optimiseur de type *gradient stochastique* (*SGD*). Les résultats obtenus sont ci-dessous:

Epoch	Top 1(Accuracy)	Top 3 (Accuracy)	Perte (Loss)
0	15.000	50.000	2.0500
1	75.129	92.330	0.7332
2	87.750	97.561	0.3868
3	90.880	98.357	0.2868
4	92.715	98.810	0.2275
5	94.199	99.202	0.1775
6	95.348	99.488	0.1430
7	96.012	99.580	0.0970
8	97.237	99.766	0.0824
9	97.547	99.857	0.0713
10	98.538	99.909	0.0478
11	98.898	99.968	0.0363

Table 5: Tableau des précisions d'entraînement top 1 & 3 (Nombre des epochs VS précision)

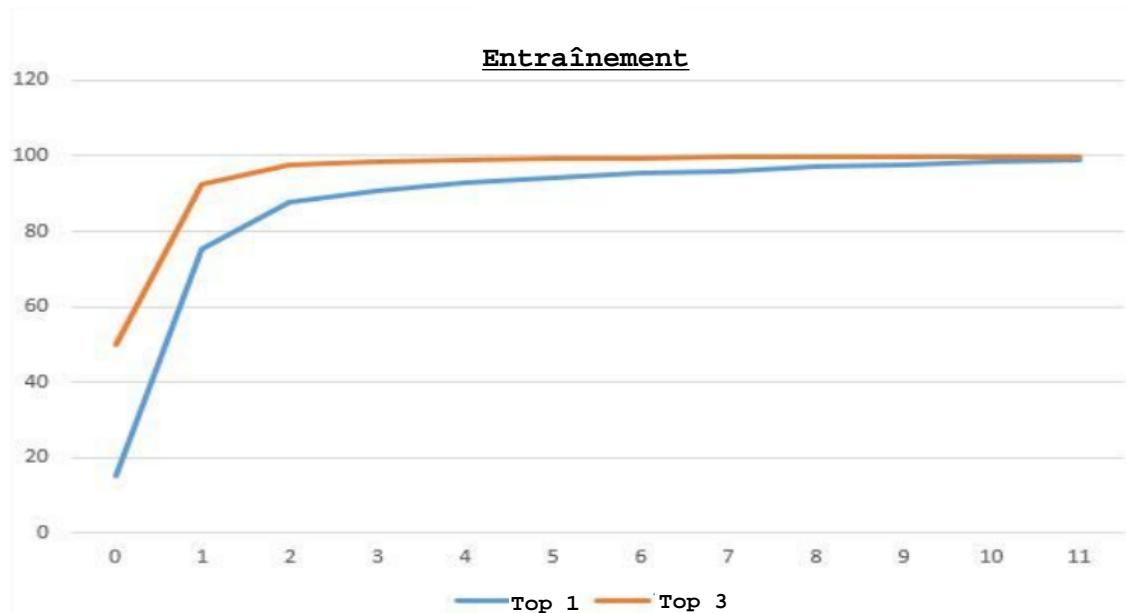


Figure 39: Graphe de la précision d'entraînement top 1 & 3 (Nombre des epochs VS précision)

Epoch	Top 1(Precision)	Top 3 (Precision)	Perte (Loss)
0	60	90.000	0.9927
1	86.779	97.688	0.4020
2	89.908	98.341	0.3179
3	89.989	98.096	0.3180
4	90.790	98.556	0.3040
5	90.968	98.060	0.3240
6	91.444	98.065	0.3280
7	92.103	98.430	0.3170
8	92.900	98.776	0.2693
9	93.145	98.803	0.2869
10	92.383	98.504	0.3082
11	91.975	98.585	0.3800

Table 6: Validation d'entraînement top 1 & 3.

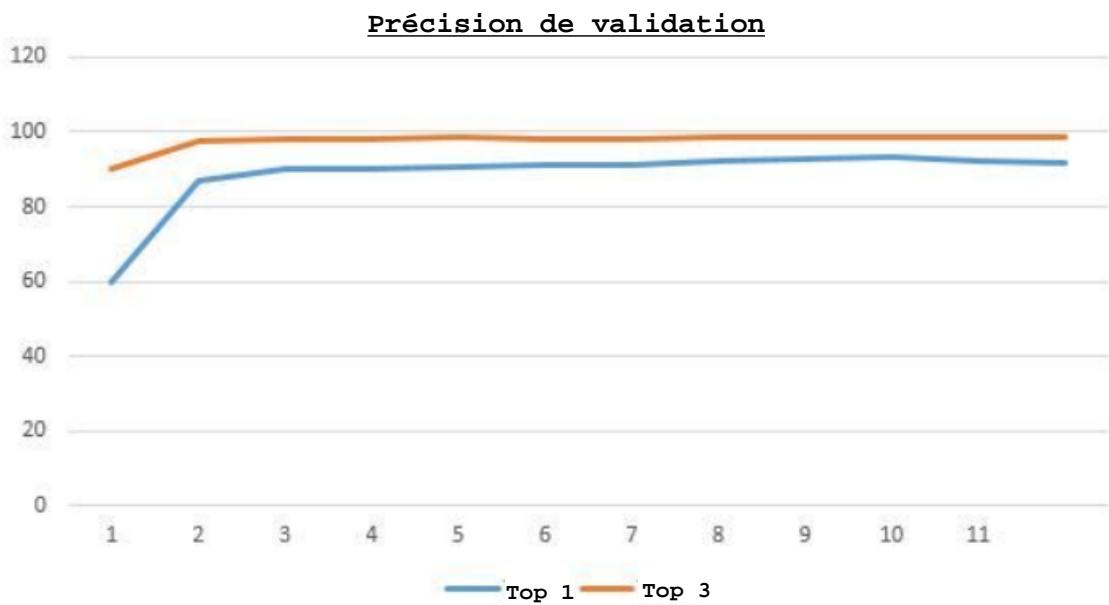


Figure 40: Graphe de la validation d'entraînement top 1 & 3 (Nombre des epochs VS précision)

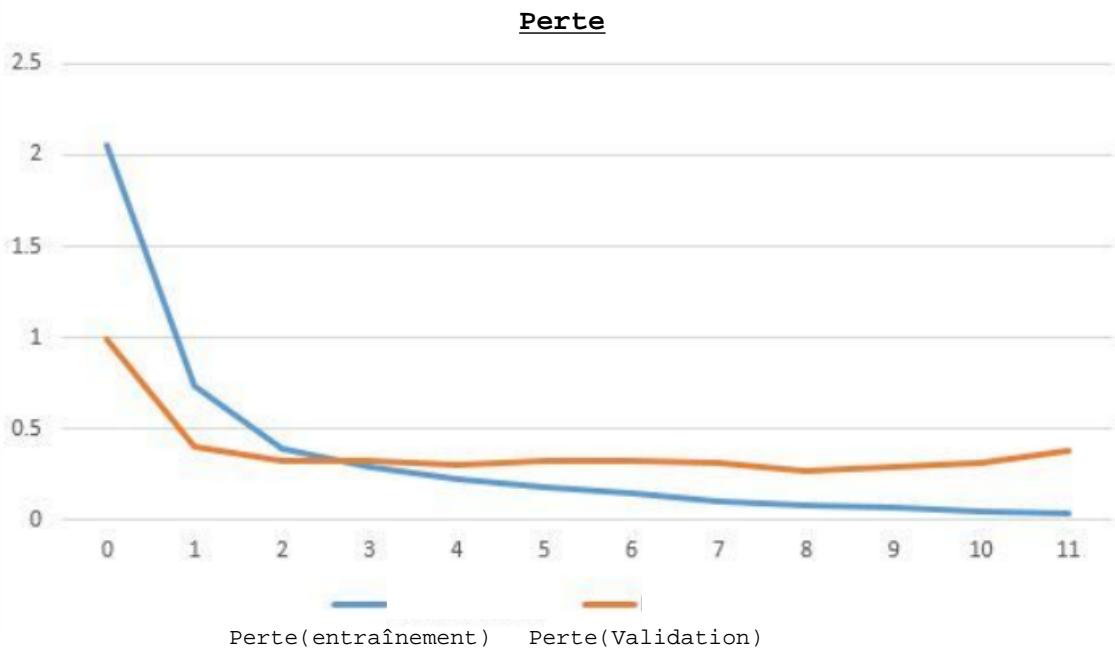


Figure 41: Graphe de la perte d’entraînement & validation (Nombre des epochs VS perte)

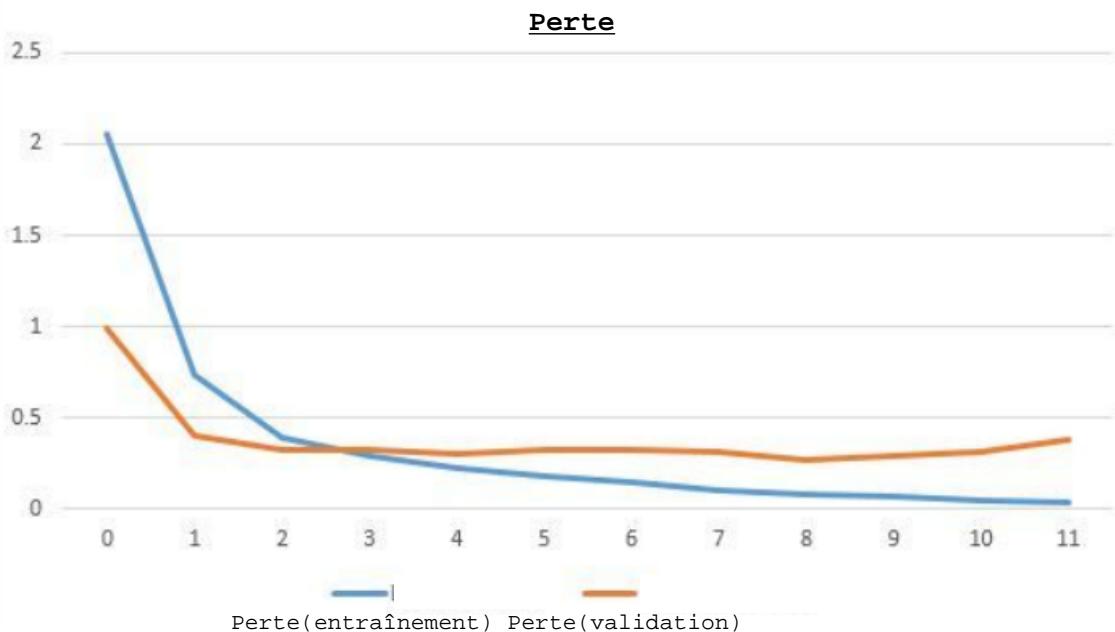


Figure 42: Comparaison des modèles avec et sans la normalisation des batchs (Nombre des epochs VS précision)

Il est clair que notre entraînement est plus rapide avec la normalisation !

Le modèle 3D présente des résultats satisfaisants avec 92% de précision, il peut maintenant être utilisé et appliqué à travers une interface graphique. La précision peut être améliorée en ajoutant une couche RNN en incluant la dimension temporelle dans la classification. Ainsi on aura un modèle *CNN3D+RNN* fusionnant la dimension spatiale et temporelle à la fois. Cela peut faire augmenter la précision jusqu'à 98%.

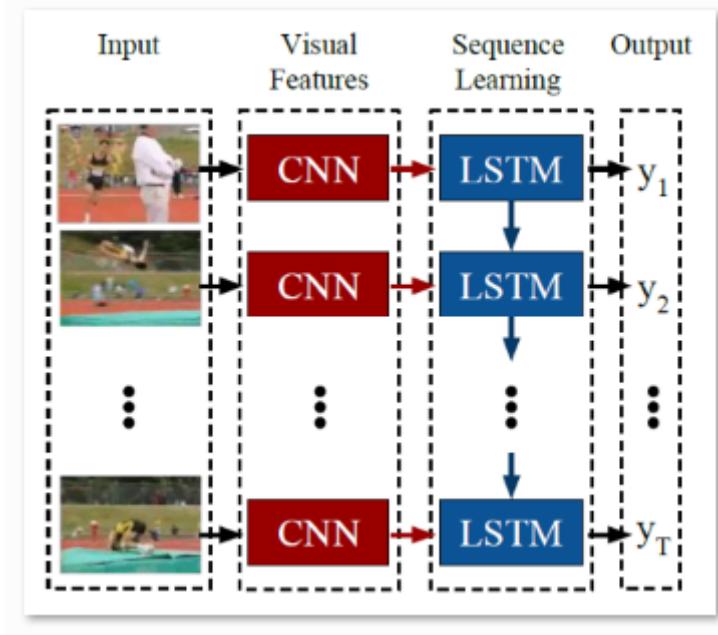


Figure 43: CNN3D + RNN

L'architecture LSTM :

Un réseau de neurones récurrents est un réseau de neurones artificiels présentant des connexions récurrentes. Un réseau de neurones récurrents est constitué d'unités (neurones) interconnectés interagissant non-linéairement et pour lequel il existe au moins un cycle dans la structure. Les unités sont reliées par des arcs (synapses) qui possèdent un poids. La sortie d'un neurone est une combinaison non linéaire de ses entrées.

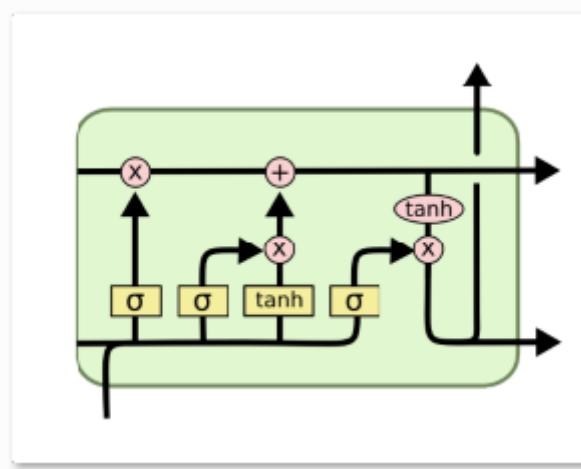


Figure 44: Unité LSTM

Conclusion

Dans ce projet nous avons essayé d'établir une architecture de réseaux de neurones qui résout la problématique de la reconnaissance efficace et efficiente des gestes de la main. Cette architecture sera après le cerveau décidant de notre interface d'application de télécommandation de notre Smart télé. Après avoir présenter notre cadre théorique de base, nous avons proposé la méthodologie de résolution suivante : notre démarche a débuté par un modèle de type **2D-CNN** qui a mal performé. Ceci est causé d'un côté par un surapprentissage vue l'insuffisance des données et la pertes des details et d'un autre côté par un sousapprentissage lorcequ'on a augmenter la taille des données. Nous avons après développé un modèle plus profond de type **3D-CNN** conforme à la dataset de 20BN ,ceci a donné une précision top-1 d'entraînement de **98.898%** et une précision top-1 de validaton de **91.975%**. Ce modèle a été implémenté après dans une interface graphique de démonstration en utilisant l'outil OpenCV de traitement d'images. En global le sujet de notre projet est très riche en terme de possibilités d'améliorations. On peut le prolonger via la création d'une interface plus élaborée ou l'intégration sous forme logicielle dans une Smart télé spécifique,pour améliorer les performances des réseaux de type 3D-CNN, l'architecture **LSTM** est une bonne piste à suivre vue que l'ajout d'une couche récurrente améliore la précision. En effet cette dernière aide la couche de convolution précédente à apprendre plus de caractéristiques.



Figure 45:

Interface de démonstration

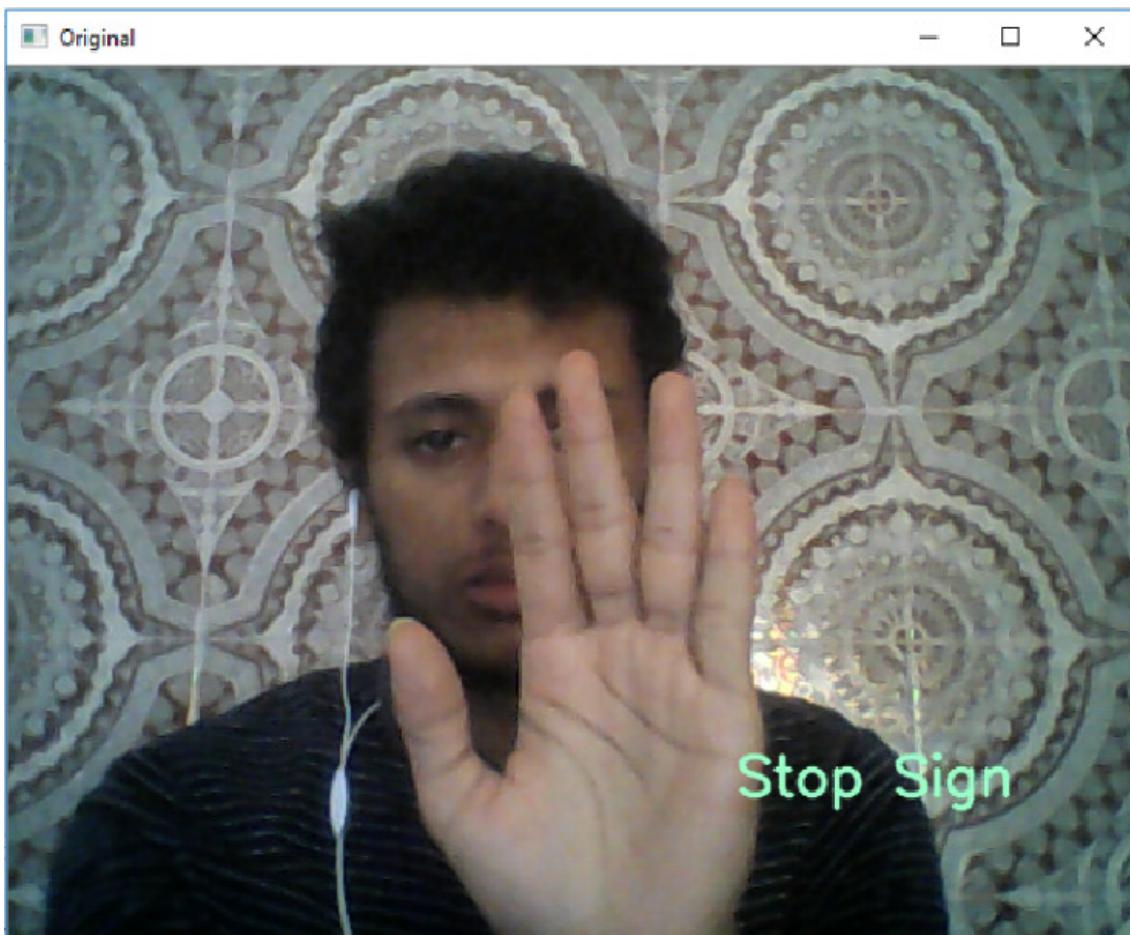


Figure 46: Interface de démonstration

Mode d'usage :

- Appuyez sur *espace*
- Performez le geste.
- Reappuyez sur *espace* pour afficher la classe du geste parmi les huits classes (STOP SIGN dans l'exemple si-dessus).

Geste	Commande
Tirer deux doigts	Zoomer
Secouer la main	Revenir en arrière
Glissement de deux doigts en bas	Diminuer le volume
Glissement de deux doigts en haut	Augmenter le volume
Glissement de deux doigts à gauche	Feuilleter les choix vers la gauche
Glissement de deux doigts à droite	Feuilleter les choix vers la droite
Stop	Sélectionner
Aucun geste	Actualiser/Afficher

Table 7: Gestes / Commandes

Bibliographie

Livres:

- *DEEP LEARNING with Python, François Chollet.*
- *Hands-On Machine Learning with Scikit-Learn & TensorFlow, Aurélien Géron, édition O'REILLY.*
- *Python deep learning projects, Matthew Lamons, Rahul Kumar, Abhishek Nagaraja.*

Articles:

- *Motion fused frames (MFFs) ,Okan Köpüklü,*
<https://arxiv.org/pdf/1804.07187.pdf>
- *A Simple Comparison of Convolutional Neural Networks, Will Burton*
<https://towardsdatascience.com/2d-or-3d-a-simple-comparison-of-convolutional-neural-networks-for-automatic-segmentation-of-625308f52aa7> .
- *Overfitting and Underfitting With Machine Learning Algorithms, Jason Brownlee,*
<https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>

Webographie:

- <https://pythonprogramming.net>
- <https://www.tensorflow.org/tutorials>
- <https://pytorch.org/tutorials/>
- <https://pandas.pydata.org/pandas-docs/>
- <https://docs.scipy.org/doc/numpy/>
- <https://docs.opencv.org>
- <https://20bn.com/datasets/jester>
- https://research.nvidia.com/publications/NVIDIA_R3DCNN_cvpr2016.pdf

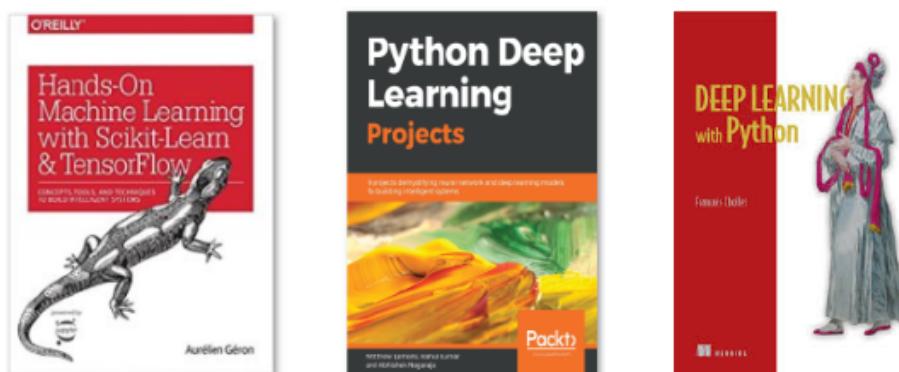


Figure 47: Livres utilisés

