

Antony ARSLANYAN

CI2019 - Profil SPID/ROB

Rapport de PFE

Interprétation des gestes de la main

Tuteur Entreprise

Eric Remilleret

Tuteur Ecole

Luc Jaulin

19 août 2019



Remerciements

Avant de commencer ce rapport, je souhaite remercier les personnes et les organisations qui ont permis que ce stage se déroule dans les meilleures conditions.

En premier lieu, je fais mes plus grands remerciements à l'ENSTA Bretagne pour le savoir que cette école a su me transmettre. Un enseignement de qualité, dispensé par des professeurs impliqués et à l'écoute pendant trois ans, m'a permis de devenir un atout compétent en entreprise.

Je souhaite plus particulièrement remercier M. Luc JAULIN, professeur principal de la spécialité robotique et mon encadrant d'école pendant ce stage, mais aussi M. Gilles Le CHENADEC pour son enseignement de grande qualité sur le machine-learning et deep-learning. Sa capacité à vulgariser les notions les plus complexes fait de lui un excellent professeur. Je souhaite aussi remercier l'Université d'Angers et plus particulièrement M. David ROUSSEAU ainsi que M. Pejman RASTI pour leurs cours de deep-learning et l'organisation du challenge intelligence artificielle AgTech. Ces projets m'ont été particulièrement utile pour la compréhension de mon sujet de stage.

J'adresse mes sincères remerciements à l'entreprise Aubay pour m'avoir accueilli pendant ce stage. Cette entreprise m'offre aussi l'opportunité de faire mes premiers pas dans la vie active et je suis touché par ce gage de confiance. Le personnel Aubay intègre avec professionnalisme ses stagiaires et leur propose de nouveaux sujets chaque année.

Je remercie aussi M. Eric REMILLERET mon maître de stage pour avoir su nous accompagner dans le projet et nous faire confiance pour présenter ce projet aussi bien en interne qu'à des partenaires extérieurs d'Aubay.

Je remercie aussi Mme. Mathilde LEPEZ et Mme. Gabrielle LAPREVOTTE pour la relecture et correction de ce rapport.

Pour finir, je remercie mon équipe projet, M. Edouard MIOR, M. Aymeric ERADES et M. Zachary JESSNER, pour le travail qu'elle a su fournir et la bonne humeur présente tout au long du projet.

Résumé

La communication entre l'homme et les outils informatiques a évolué parallèlement au développement des technologies numériques. De la souris aux écran tactiles en passant par l'aide vocale, divers moyens d'interaction facilitent et fluidifient les interactions hommes-machine.

C'est dans ce contexte que ce stage s'inscrit. L'objectif de celui-ci est de mettre en œuvre un outil permettant d'interagir de manière naturelle en utilisant nos mains. Ce projet traite des thématiques d'intelligence artificielle, de traitement d'image et d'interaction homme-machine. Ce projet est en accord avec l'émergence des nouveaux moyens de communication pour les smartphones par exemple. La différence avec les innovations proposées par les grandes entreprises est l'utilisation d'une webcam comme unique capteur. Ce projet innovant a été proposé par la société Aubay.

L'intelligence artificielle est abordée sous deux points de vue dans le projet à travers deux réseaux de neurones différents. Le premier est un réseau déjà existant, entraîné sur une très grosse base de données. Le travail lié à ce réseau consiste à un réentraînement visant à cibler le réseau sur la tâche voulue. Le second est un réseau créé de toute pièce pour le projet avec un entraînement complet.

L'outil développé durant le stage satisfait les attentes définies en début de projet. Il surpasse même les attentes dans la vitesse d'exécution. Cependant, il reste pertinent de poursuivre le travail pour l'adapter sur plus de plates-formes, ajouter plus d'interactions et améliorer la précision du travail.

Abstract

Communication between person and machine has evolved in parallel with the development of digital technologies. From the computer mouse to tactile screens and voice command, various means of interaction facilitate and fluidify human-machine interactions.

It is in this context that this internship began. Its objective was to implement a tool to interact smoothly with a computer using our hands. This project deals with the themes of artificial intelligence, image processing and human-machine interaction. This project is in line with the emergence of new means of communication, for smartphones for example. The difference with innovations suggested by large companies is the sole use of a webcam as sensor. This innovative project was suggested by the Aubay company.

Artificial intelligence is approached from two points of view in this project through two different neural networks. The first one is an existing network trained on a very large database. The work related to this network consists in a transfer train for the network to focus on the desired task. The second is a network created from scratch for the project with complete training.

The tool developed during the internship meets the expectations defined at the beginning of the project. It even exceeds expectations in speed of execution. However, it is still relevant to continue the work to adapt it to more platforms, add more interactions and improve the accuracy of the work.

Table des matières

Résumé	3
Abstract	4
Glossaire	8
Introduction	9
1 Présentation de l'entreprise	10
1.1 Description générale	10
1.2 Pôle innovation	10
1.2.1 Fonctionnement et principaux projets	10
1.2.2 Le projet DWYH ¹	11
2 Recherches existantes	12
2.1 Travaux antérieurs sur le projet	12
2.1.1 Première version (2017)	12
2.1.2 Deuxième version (2018)	13
2.2 Détection par traitement d'images	14
2.3 Utilisation de réseaux de neurones	15
2.3.1 Détection de mains	15
2.3.2 Détection de mouvements	16
3 Choix effectués et structure du projet	17
3.1 Structure du code	17
3.2 Choix effectués pour les réseaux	18
4 Détection des mains sur un flux vidéo	20

1. *Drive With Your Hands*

4.1	Le framework darknet et le modèle Yolo ²	20
4.2	Fonctionnement de Yolo	21
4.2.1	Couches du modèle : convolution, max pooling, route et upsample	21
4.2.2	La couche Yolo	23
4.2.3	Hyperparamètres	24
4.2.4	Évaluation des résultats	25
4.3	Création d'une base d'image	27
4.4	Ajout de détection de tête	28
4.5	Résultats finaux	28
5	Traitement sur les mains détectés	30
5.1	Binarisation de l'image	30
5.2	Opération sur le contour de main	31
5.3	Détection du squelette de main	32
5.4	Structure de la main et résultats	33
5.5	Soustraction du fond et tracking de main	33
6	Détection des mouvements	35
6.1	Les mouvements à détecter	35
6.2	Conversion du mouvement en une image	35
6.3	Modèle choisi et entraînement	36
6.4	Résultats	37
7	Reconnaître une action et interagir	39
7.1	Les différents types d'interactions	39
7.2	Liste des actions réalisables	40
8	Interface d'utilisation du projet et résultats	41
8.1	Première interface de test	41
8.2	Interface projet fonctionnel	43
Conclusion		44
Table des figures		46
Annexes		46

2. *You Only Look Once*

A Site web du projet et vidéos	47
B Exemples supplémentaires partie traitement d'image	48
C Information sur le modèle détectant les mouvements	49
Sigles et Acronymes	50
Bibliographie	52

Glossaire

convolution Opération principale d'un CNN³ permettant d'extraire les features de l'objet à détecter. 21, 22

Darknet Darknet est un framework codé en C permettant d'entraîner le modèle Yolo . 15, 19, 20

deep-learning Apprentissage profond de caractéristiques précises d'un objet. Le plus souvent via un réseau de neurones. 13

faux positif Détection incorrecte effectuée par le modèle. 26

features Les features d'un objet sont ses caractéristiques qui permettent de le différencier du reste sur une image. . 15

geste Un geste de la main est une action fixe dans le temps reconnaissable en une image.
Par exemple le "V de la victoire" est un geste . 11, 35

ground-truth Annotation représentant la prédiction parfaite à faire par le modèle. Celle-ci est souvent aire à la main. . 25, 26, 29

max pooling Opération permettant de réduire la taille d'une matrice en ne gardant que ces éléments les plus grand. 21, 22

mouvement Un mouvement est une action effectué par la main dans le temps. Par exemple effectuer un cercle ou agiter la main sont deux mouvements. 11, 15, 16, 18, 35

non max suppression Algorithme permettant de filtrer les prédictions faites par le modèle Yolo. 24

Qt Qt est une API orientée objet vouée à la création d'interfaces graphiques. 18, 41

Qthread Classe sous Qt permettant de gérer les threads. 42

réseau de neurones Un réseau de neurones est un système d'apprentissage permettant notamment d'effectuer une classification. 14, 15, 18, 21, 35

Tensorflow Tensorflow est un framework développé par Google permettant de créer et entraîner des réseaux de neurones. 19, 36

vrai positif Détection correcte effectuée par le modèle . 26

3. *Convolutional Neural Network, acronyme anglais pour réseau de neurones convolutif*

Introduction

Ce rapport présente les travaux réalisés dans le cadre de mon projet de fin d'études avec l'entreprise Aubay. Ce stage a été effectué sous la tutelle d'Eric REMILLERET et de Luc JAULIN. Ce stage fait suite à trois années d'études à l'ENSTA Bretagne. Son objectif est de mettre en œuvre un outil permettant d'interagir avec une machine en reconnaissant diverses actions effectuées avec les mains. Le choix de ce projet a été motivé par l'attrait innovant du projet ainsi que la possibilité de travailler sur un sujet traitant d'intelligence artificielle.

Aubay est une entreprise des services du numérique française ayant une action dans toute l'Europe. Elle tente de se démarquer des autres ESN⁴ en valorisant l'aspect recherche. C'est dans une équipe de recherche que j'ai intégré Aubay, ma mission consistant dans le développement de l'application, mais aussi dans la transmission du savoir acquis pendant six mois.

L'entreprise qui m'a accueilli sera présentée en premier dans ce rapport. Ensuite, une présentation des enjeux encadrant le sujet ainsi que les technologies liées sera effectuée. Une troisième partie décrira la structure du projet et l'organisation du développement. Par la suite, quatre parties décriront les quatre axes de recherche du projet : la détection des mains, le traitement d'image, la détection des mouvements et la gestion des interactions. Enfin, seront détaillées les interfaces graphiques liées au projet facilitant son utilisation.

4. *Entreprise des Services du Numérique*

Chapitre 1

Présentation de l'entreprise

1.1 Description générale

J'ai effectué mon stage de fin d'études au sein d'Aubay, une ESN basée à Boulogne Billancourt possédant des missions dans toute l'Europe de l'Ouest. L'entreprise Aubay existe depuis 1998 et a depuis fait l'acquisition de nombreuses entreprises.

Aubay propose un service de conseil notamment pour les banques et assurances (85% de son chiffre d'affaires en France). La majeure partie de ses 6000 collaborateurs est en mission dans diverses entreprises.

Les employés sont repartis dans différentes BU¹ en fonction de leurs compétences et envies de travail. Les différentes BU sont :

- Assurance
- Banque
- Innovation
- Finance
- IATR²

Mon stage a eu lieu avec le pôle Innovation sur le projet DWYH au sein d'une équipe de quatre personnes.

1.2 Pôle innovation

Le rôle du pôle innovation est d'effectuer un travail de recherche et développement afin d'anticiper les futures demandes de clients et d'élargir le domaine de compétences d'Aubay. Le pôle innovation est un atout permettant à Aubay de remporter de nombreux appels d'offres.

1.2.1 Fonctionnement et principaux projets

La majorité des employés et stagiaires du pôle innovation sont répartis dans un open-space séparé par îlots entre chaque projet. Des chefs d'équipe peuvent ainsi aisément gérer différentes

1. *Business Unit*

2. *Infrastructure Assurance Télécom Réseaux*

équipes de projet.

Aubay travaille actuellement sur de nombreux projets, les plus importants sont :

- E-board, un outil Agile permettant d'intégrer les fonctionnalités de la méthode Kanban sur un logiciel adapté aux besoins d'Aubay.
- I-Scan, un logiciel utilisant l'intelligence artificielle pour décrypter des documents manuscrits et les convertir en un document texte éditable.
- Holoview, un projet utilisant le casque Hololens développé par Microsoft pour créer divers jeux et formations ludiques.
- GAO, un projet utilisant le NLP³ pour analyser des appels d'offres et les lier à de potentiels CV.
- DWYH, le projet sur lequel j'ai travaillé, ayant pour but de fluidifier les interactions entre l'humain et la machine en utilisant uniquement une webcam.

1.2.2 Le projet DWYH

Le projet DWYH a été initié au sein de la cellule innovation d'Aubay en 2017. Le but de ce projet est de fluidifier les interactions entre l'humain et la machine en détectant et en interprétant les gestes et mouvements de la main.

Cette année, nous étions une équipe de quatre pour reprendre le projet. Nous avons eu le choix entre reprendre les bases déjà existantes ou tout reprendre à zéro. Nous avions pour consigne d'introduire le deep learning dans le projet. Compte tenu du travail précédemment effectué (voir état de l'art p.12) nous avons choisi de recommencer le projet à zéro afin d'utiliser les outils de notre choix.

Notre équipe était constituée de trois stagiaires de dernière année ainsi que d'un stagiaire de deuxième année. Nous étions aussi suivis par M. Remilleret mon maître de stage et M. Kranneveld. Nous avons travaillé de manière agile. Chaque mois, durée d'un "sprint", nous devions livrer une version fonctionnelle du projet plus poussée que la version précédente. Nous avions chaque semaine une réunion pour débattre de nos avancements et évoquer nos problèmes à nos encadrants.

D'autres réunions exceptionnelles ont aussi été programmées pour préparer diverses présentations de notre projet, notamment pour la journée des stagiaires⁴ et une grosse présentation pour un client d'Aubay.

Nous avions à disposition des PC munis d'une carte graphique K620 de 16Go de ram et d'un processeur i7. Cette configuration est amplement suffisante pour faire tourner notre projet.

Nous nous sommes concentrés sur le contrôle d'un ordinateur avec notre outil, mais ce projet a pour ambition de s'exporter sur tout type d'objets électronique tel qu'une télévision ou encore une console de jeux vidéo.

3. *Natural Language Processing*

4. La journée des stagiaires a eu lieu de 10 Juillet pour présenter les différents projets de stage au personnel d'Aubay et effectuer une démonstration de l'état du produit.

Chapitre 2

Recherches existantes

2.1 Travaux antérieurs sur le projet

Le projet DWYH a été initié en 2017. Deux versions ont vu le jour, une utilisant du traitement d'images et une tentant d'utiliser un modèle pré-entraîné de deep-learning. Je décrirai les travaux antérieurs sans rentrer dans les détails, certaines parties étant similaires avec le travail effectué cette année seront expliquées ensuite et d'autres, complètement abandonnées en 2019, ne seront pas approfondies dans ce rapport.

2.1.1 Première version (2017)

La première version tente de comprendre les mouvements de la main en utilisant uniquement des méthodes de traitement d'images [1]. La version est codée en javascript avec l'ambition de s'exporter sur navigateur.

Le traitement s'effectue en plusieurs étapes distinctes. Premièrement, l'utilisateur doit régler différents curseurs afin de pouvoir détecter la main. Le programme recherche alors la plus grande zone avec un gradient de couleur proche de la couleur choisie. La grosse limite de leur programme provient de ce réglage : il n'est pas agréable de régler un outil avant chaque utilisation et la main n'est jamais bien détectée si le fond n'est pas uni.

Lorsque la main est bien détectée, une binarisation est effectuée. L'image est transformée en une image en noir et blanc en ne gardant en blanc que les couleurs proches de celle de la main. Après filtrage du bruit, en s'inspirant de [2], les doigts sont repérés. Pour cela, le plus grand cercle inscrit dans la forme est tracé, le centre de ce cercle étant le centre de la main. En traçant des cercles de même centre avec un diamètre plus grand, il est possible d'estimer le nombre et la position des doigts (fig 2.1).

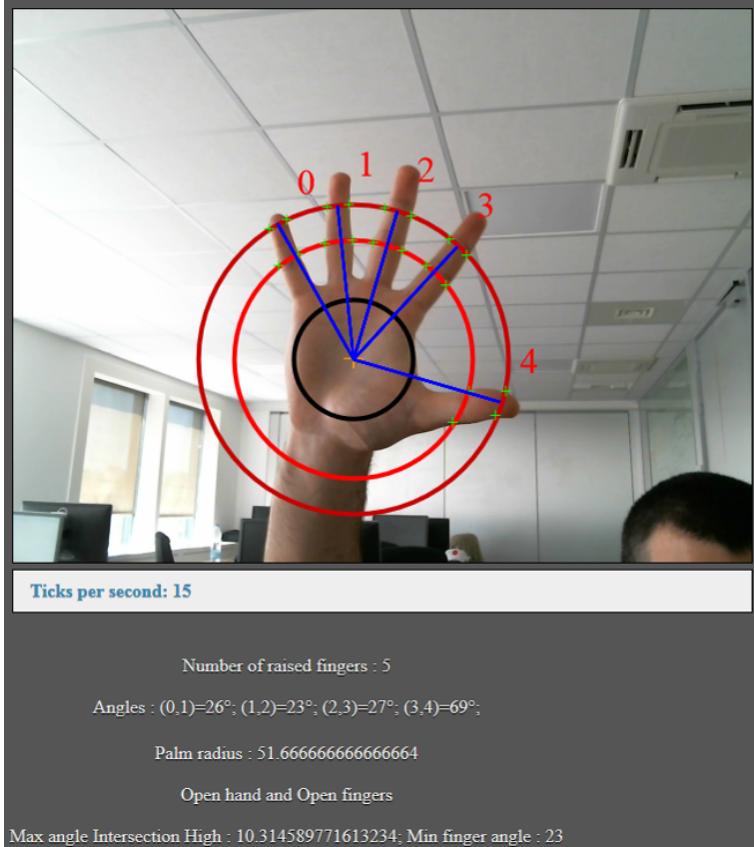


FIGURE 2.1 – Détection du nombre de doigts avec la version 2017.

Le programme est assez rapide pour être utilisé en direct, on peut voir sur fig.2.1 une détection à 15 fps¹. Cependant, cela n'est pas suffisant pour une manipulation parfaitement fluide. Seul un petit nombre d'interactions avec la machine ont été implémentées.

2.1.2 Deuxième version (2018)

Le projet de 2018 [3] vise à corriger les problèmes de détection de la main. L'idée principale est de supprimer les étapes de traitement d'image et de comprendre les actions effectuées uniquement via un modèle de deep-learning.

L'équipe de 2018 a choisi d'utiliser la librairie OpenPose qui permet via un réseau de neurones complexe de détecter les squelettes présents sur l'image (fig. 2.2)



FIGURE 2.2 – Exemple d'utilisation d'OpenPose tiré de [4].

1. Frame per second (image par secondes)

Pour améliorer la fluidité, le code d'OpenPose a été modifié pour permettre de ne détecter que les mains (fig. 2.3). Même avec cette amélioration, la fluidité du projet n'est pas assurée, le projet ne fonctionne qu'à 5 fps.



FIGURE 2.3 – Version d'OpenPose ne détectant que les mains.

Notre idée, pour poursuivre le projet, a donc été d'utiliser un réseau de neurones moins complexe pour localiser la main sans perdre trop de fluidité puis d'effectuer un traitement d'images. Nous souhaitions aussi ne pas avoir à régler les paramètres de colorimétrie à chaque utilisation.

2.2 Détection par traitement d'images

Divers articles traitent de la segmentation de peau ou plus précisément de main. Les articles [5, 6] proposent de nombreuses méthodes pour effectuer cette segmentation. Parmi celles-ci, nous avons retenu l'utilisation de deux espaces de couleurs :

- YCbCr est un espace dans lequel les différentes variations de couleurs des mains se situent dans un intervalle restreint. Effectuer une binarisation dans cet espace restreint permet d'isoler facilement la main.
- Lab est un second espace dans lequel la luminosité est codée sur un canal et la couleur sur les deux autres. Une zone d'ombre sur la main influera donc moins sur la binarisation de l'image dans cet espace. [7]

L'article [5] propose aussi une méthode permettant de différencier l'arrière-plan et le premier plan. Cette méthode sépare l'image en différentes sous-populations de l'image grâce à un algorithme de GMM². Les sous-populations correspondantes à l'arrière-plan (B) sont alors trouvées selon l'équation suivante.

$$B = \operatorname{Argmin}_k \left(\sum_{i=0}^K w_{it} > BGTHR \right) \quad (2.1)$$

où BGTHR est le seuil de segmentation de l'arrière-plan

K est le nombre de sous-populations

w est le poids de la sous-population

Cet algorithme considère que l'arrière-plan est dominant sur l'image. Dans notre cas, nous effectuons le traitement sur une image où la main est dominante ce qui empêche l'utilisation de cet algorithme.

Un autre procédé présenté notamment dans [8, 6] propose d'effectuer un tracking de la main. Une fois celle-ci repérée, sa position à l'instant suivant sera proche de sa position actuelle. Ce papier propose alors un algorithme permettant d'évaluer la probabilité qu'un pixel appartienne à la peau.

2.3 Utilisation de réseaux de neurones

Afin de grandement faciliter le traitement d'images, nous avons décidé de localiser la main à l'aide d'un réseau de neurones (gls pg-8). Nous souhaitions aussi pouvoir comprendre quel mouvement a été effectué par la main et avons opté pour détecter ces mouvements via un second réseau.

2.3.1 Détection de mains

Nous avons ensuite cherché des articles utilisant des réseaux de neurones pour détecter des mains sur un flux vidéo. La complexité de cette détection provient du fait qu'une image doit être traitée suffisamment rapidement par le réseau pour respecter la contrainte de temps réel.

L'article [9] décrit une méthode pour détecter en direct des personnes sur une vidéo. Une fois la personne détectée, le réseau est ensuite capable d'extraire les features des objets détectés, permettant de le détecter plus facilement sur l'image suivante. La partie qui extrait les features est un simple CNN et la partie qui utilise ces features sur l'image suivante est un RNN³. Le résultat à un instant donné est une combinaison linéaire du résultat trouvé par le CNN et celui trouvé par le RNN à l'instant précédent.

Nous souhaitions dans un premier lieu détecter des positions de mains fixes. Le travail effectué dans [10] vise à différencier différents signes du langage des signes. En revanche, la segmentation de la main est effectuée ici par colorimétrie. La main est ensuite séparée du bras en trouvant le poignet. Le poignet est, sur l'image segmentée, la partie la plus fine de la main. Nous avons tenté d'utiliser cet algorithme dans notre code, mais il ne s'est pas révélé suffisamment robuste. Une fois la main bien segmentée, le geste détecté en utilisant un réseau de neurone ayant en entrée la taille de l'objet, le ratio entre longueur et largeur et le ratio entre la surface segmentée et non segmentée dans la box de détection.

Nous avons finalement, notamment grâce aux articles [11, 12], découvert le modèle Yolo (You only look once). Celui-ci fonctionne initialement avec un framework codé spécialement pour le modèle nommé Darknet. Yolo permet d'effectuer de la détection d'objets de manière précise et fluide sur une vidéo. Sa version la plus évoluée possède 9000 [13] objets détectables. Un exemple est fourni en (fig 2.4). Les classes sont en fait réparties en sous-classes, par exemple un beagle est une sous-classe du chien, lui-même une sous-classe des animaux, lui-même étant une sous-classe des êtres vivants. Nous avons essayé cette version, mais du fait des limitations apportées par nos cartes graphiques le rendu n'a été que peu fluide. Cependant, il existe une

3. Recurrent Neural Network

version alternative plus légère permettant de garder la fluidité de la vidéo. Nous avons choisi le modèle Yolo avec le framework darknet pour le projet. Les raisons de ce choix sont expliquées au chapitre 3 et le fonctionnement de Yolo est développé dans le chapitre 4.

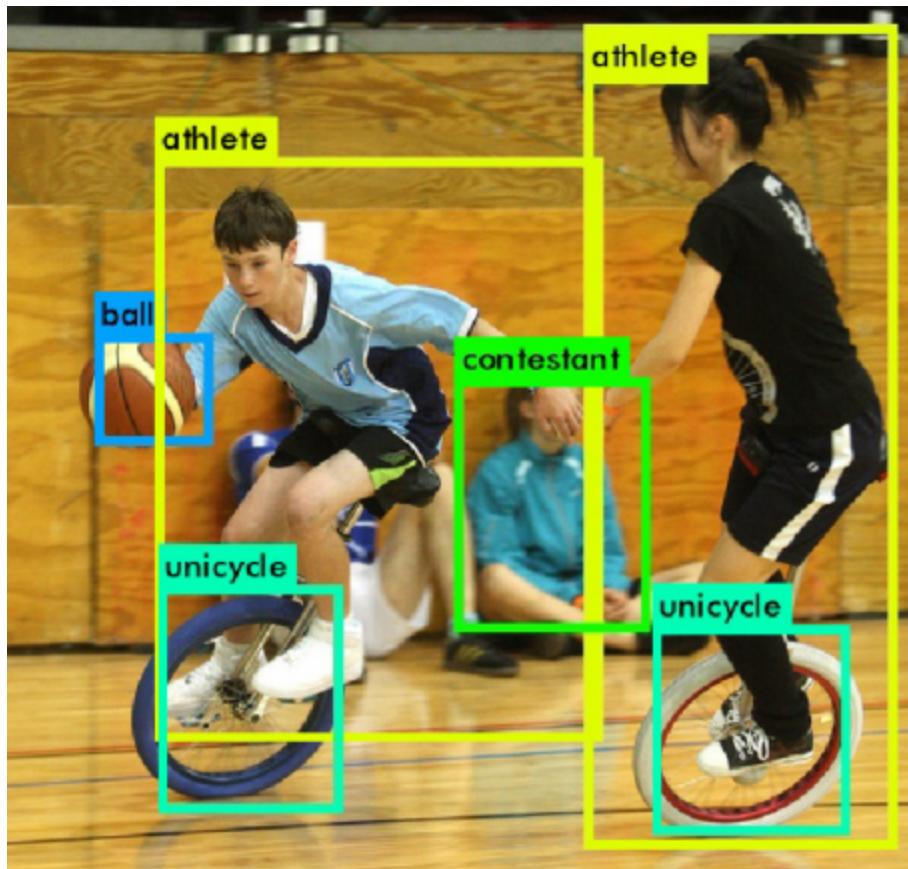


FIGURE 2.4 – Exemple d'utilisation de Yolo avec 9000 classes tiré de [13]

2.3.2 Détection de mouvements

La seconde partie de notre programme devra détecter des mouvements. Comme l'explique [9], un RNN permet de prendre en compte la notion de temps. Nous nous sommes aussi appuyés sur [14] dont le but est de détecter les comportements violents sur des caméras de surveillance. Leur modèle travaille ensuite sur la différence entre deux images consécutives pour détecter une accélération qui correspondrait à une action dangereuse.

Nous avons tenté d'implémenter un RNN dans notre programme, cependant ces derniers se sont toujours révélés trop coûteux en temps de calcul. La solution proposée n'utilise finalement qu'un CNN et est expliquée dans 6.

Chapitre 3

Choix effectués et structure du projet

3.1 Structure du code

Ce chapitre vise à expliquer le fonctionnement global de notre projet. Je décrirai ici les blocs de traitement de l'acquisition de l'image jusqu'à l'action effectuée sur l'ordinateur et je justifierai leurs présences dans le projet. Chaque bloc sera ensuite approfondi dans les parties suivantes.

De par l'obligation de travailler sous environnement Windows nous avons utilisé Visual Studio pour créer notre projet. Cet éditeur nous permet d'exporter facilement une version exécutable utilisable par n'importe qui.

Notre code est séparé en différents fichiers. Chaque fichier correspond à une grande partie de notre processus de traitement. Les interactions entre ces fichiers sont représentées sur la fig3.1 ci-après.

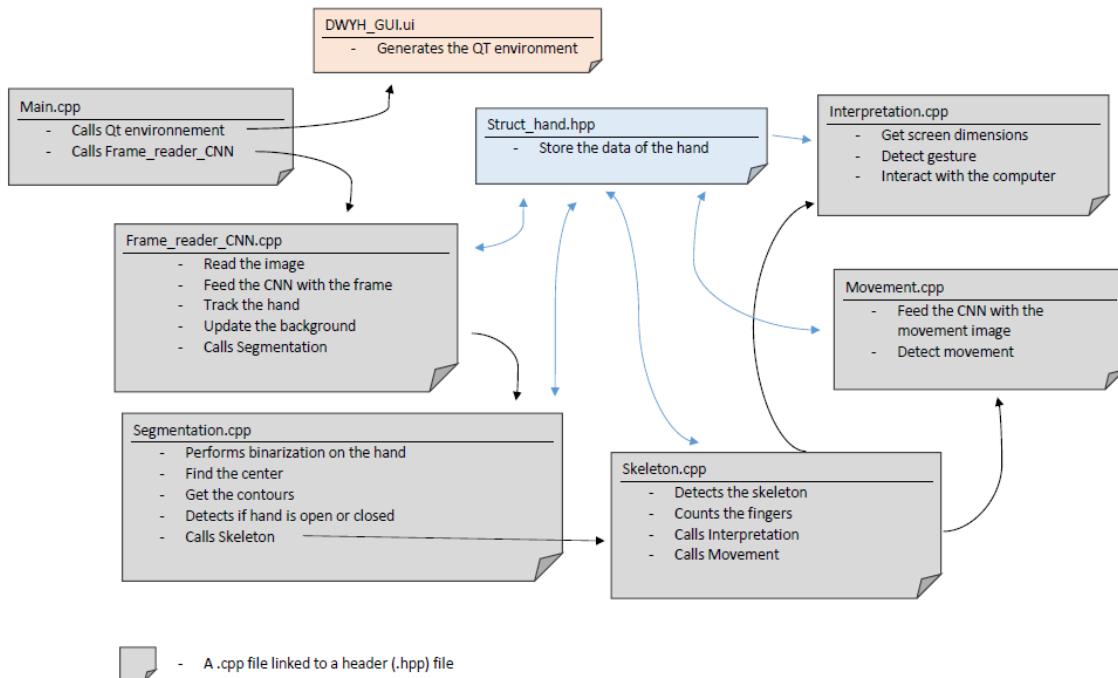


FIGURE 3.1 – Les différents fichiers de code qui composent notre projet.

Le script d'entrée de la chaîne d'information est nommé main.cpp. Celui-ci ne fait aucune opération calculatoire. Il permet juste d'initialiser la caméra et diverses variables. Ce code lance aussi l'interface Qt qui est décrite au chapitre 8.

Une fois l'interface graphique lancée, le script FrameReaderCNN a pour rôle de passer l'image lue par la caméra dans le premier réseau et de donner le nombre et la position des mains dans l'image. C'est aussi ce code qui s'occupe d'effectuer une mise à jour du fond qui sera expliquée en 5.

Une classe a été créée pour le projet, nommée struct_hand, celle-ci comporte toutes les caractéristiques utile d'une main sur une image. Lorsqu'une main est détectée, une instance de la classe struct_hand est créée. A ce stade du suivi du code, seul le numéro attribué à la main détectée et sa position sont enregistrés dans la classe.

Ensuite, le script Segmentation s'occupe d'effectuer toute la partie de traitement d'images (5). La position du centre de la main est alors ajoutée à la classe struct_hand.

Le script skeleton s'occupe ensuite d'ajouter le nombre et la position (pouce, index...) des doigts levés.

Enfin, le script mouvement analyse à nouveau l'image avec un second réseau de neurones pour détecter si un mouvement a été effectué (6). Toutes les informations stockées dans la classe main sont ensuite lues par le script interprétation qui s'occupe de gérer les interactions avec l'ordinateur (7).

L'ensemble de ces opérations doivent être réalisées suffisamment rapidement pour garder un nombre d'images traitables par secondes élevé. Le code doit donc être optimisé en utilisant par exemple des threads.

3.2 Choix effectués pour les réseaux

Pour le premier réseau de neurones, le choix c'est porté sur l'utilisation du modèle Yolo. En effet, les articles [11, 12, 13] démontrent bien l'avantage du modèle Yolo par rapport à d'autres modèles. Le fonctionnement précis du modèle sera effectué en 4. Yolo reprend les avantages de réseaux classiques tout en améliorant grandement sa vitesse.

La figure 3.2 permet de comparer les performances de Yolo avec d'autres modèles. Les différents nombres (320/416/608) sont la taille d'entrée des images. Celle-ci influe sur les performances du modèle.

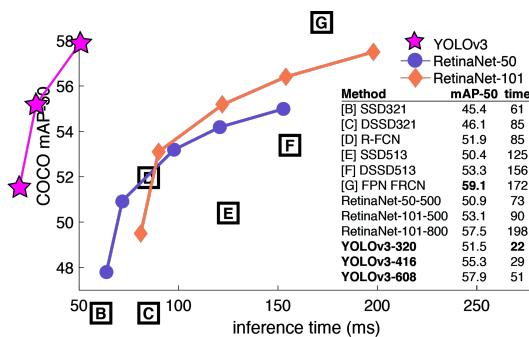


FIGURE 3.2 – Les performances de Yolo par rapport à d'autres modèles classiques.

Une fois le choix de travailler avec Yolo effectué, il a fallu décider quelle version et quel framework utiliser. Le choix s'est évidemment porté sur la dernière version du modèle (Yolov3) car plus performante. Nous avons par ailleurs, après différents tests, opté pour la version "tiny" possédant moins de couches, car elle échange une perte de précision contre un gain en vitesse d'exécution. La version normale ne nous permettait pas de travailler de manière fluide en direct. Le framework lié au modèle Yolo est Darknet. Celui-ci possède l'avantage d'avoir été codé et optimisé pour Yolo, il ne donne en revanche que peu de retour comparé à des frameworks plus classique comme Tensorflow. Il est possible d'exporter le modèle Yolo sur Tensorflow grâce à un module nommé darkflow mais celui-ci n'existe que pour la version 2 de Yolo. Au vu des résultats obtenus dans le tableau 3.1, nous avons choisi d'utiliser le modèle YoloV3_tiny avec le framework Darknet. Pour évaluer la précision des modèles, nous utilisons le mAP¹ qui sera expliqué dans la partie suivante. Un bon score de mAP est proche de 1.

Modèle	Framework	Fluidité (fps)	Précision obtenue (mAP)
Yolov3_Tiny	Darknet	30	0.89
Yolov3	Darknet	13	0.94
YoloV2_Tiny	Darknet	27	0.86
YoloV2_Tiny	Darkflow	27	0.86
YoloV2	Darkflow	11	0.91

TABLE 3.1 – Comparaison des performances selon différentes versions de Yolo.

Nous pouvons remarquer que la conversion sous Tensorflow est effectuée sans perte puisque les résultats sont les mêmes qu'avec Darknet pour un modèle équivalent.

Pour le second modèle qui détecte les mouvements, nous avons utilisé un modèle personnalisé créé via Tensorflow. Nous obtenons en effet des résultats probants avec notre propre modèle (voir partie 6) et avons donc fait le choix de ne pas utiliser une structure connue pour le modèle.

1. *Mean Average Precision*

Chapitre 4

Détection des mains sur un flux vidéo

Ce chapitre vise à expliquer plus en détail le modèle utilisé pour détecter les mains : sa structure, son fonctionnement et son entraînement. Ce chapitre développera aussi les autres idées liées au modèle comme la détection de tête par exemple.

4.1 Le framework darknet et le modèle Yolo

Comme expliqué précédemment, le modèle Yolo utilisé dans le projet est la version tiny du modèle. Celle-ci possède 14 couches de convolution ce qui est moins que la version standard. Cela permet de rendre le traitement plus rapide. C'est avec le framework Darknet que nous avons entraîné le modèle.

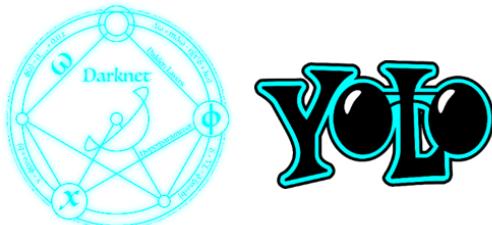


FIGURE 4.1 – Logos de Darknet et You Only Look Once.

Ce modèle a été créé en 2015 par Joseph Redmon et Ali Farhadi [11] et a subi de nombreuses évolutions afin d'arriver à sa version 3 en 2018 [12]. Les premières couches du modèle s'inspirent du Googlenet. Joseph Redmon a été embauché par Google pour améliorer son modèle.

4.2 Fonctionnement de Yolo

La version Tiny du modèle Yolo possède un total de 23 couches dont 13 convolutions et 6 max poolings. Pour l'exemple du tab.4.2 nous prenons une image de taille $832 * 832$. Comme l'image possède 3 canaux de couleurs, elle est représentée par une matrice de taille $832 * 832 * 3$. Les diverses opérations affectent le format de l'image ; le format d'entrée et de sortie de chaque couche est donné dans le tableau expliquant l'architecture du modèle (tab.4.2). Les colonnes filters et size sont les paramètres utilisés pour chaque layer.

layer	filters	size	input	output
0 conv	16	$3 \times 3 / 1$	$832 \times 832 \times 3$	$\rightarrow 832 \times 832 \times 16 0.598 \text{ BF}$
1 max		$2 \times 2 / 2$	$832 \times 832 \times 16$	$\rightarrow 416 \times 416 \times 16 0.011 \text{ BF}$
2 conv	32	$3 \times 3 / 1$	$416 \times 416 \times 16$	$\rightarrow 416 \times 416 \times 32 1.595 \text{ BF}$
3 max		$2 \times 2 / 2$	$416 \times 416 \times 32$	$\rightarrow 208 \times 208 \times 32 0.006 \text{ BF}$
4 conv	64	$3 \times 3 / 1$	$208 \times 208 \times 32$	$\rightarrow 208 \times 208 \times 64 1.595 \text{ BF}$
5 max		$2 \times 2 / 2$	$208 \times 208 \times 64$	$\rightarrow 104 \times 104 \times 64 0.003 \text{ BF}$
6 conv	128	$3 \times 3 / 1$	$104 \times 104 \times 64$	$\rightarrow 104 \times 104 \times 128 1.595 \text{ BF}$
7 max		$2 \times 2 / 2$	$104 \times 104 \times 128$	$\rightarrow 52 \times 52 \times 128 0.001 \text{ BF}$
8 conv	256	$3 \times 3 / 1$	$52 \times 52 \times 128$	$\rightarrow 52 \times 52 \times 256 1.595 \text{ BF}$
9 max		$2 \times 2 / 2$	$52 \times 52 \times 256$	$\rightarrow 26 \times 26 \times 256 0.001 \text{ BF}$
10 conv	512	$3 \times 3 / 1$	$26 \times 26 \times 256$	$\rightarrow 26 \times 26 \times 512 1.595 \text{ BF}$
11 max		$2 \times 2 / 1$	$26 \times 26 \times 512$	$\rightarrow 26 \times 26 \times 512 0.001 \text{ BF}$
12 conv	1024	$3 \times 3 / 1$	$26 \times 26 \times 512$	$\rightarrow 26 \times 26 \times 1024 6.380 \text{ BF}$
13 conv	256	$1 \times 1 / 1$	$26 \times 26 \times 1024$	$\rightarrow 26 \times 26 \times 256 0.354 \text{ BF}$
14 conv	512	$3 \times 3 / 1$	$26 \times 26 \times 256$	$\rightarrow 26 \times 26 \times 512 1.595 \text{ BF}$
15 conv	18	$1 \times 1 / 1$	$26 \times 26 \times 512$	$\rightarrow 26 \times 26 \times 18 0.012 \text{ BF}$
16 yolo				
17 route	13			
18 conv	128	$1 \times 1 / 1$	$26 \times 26 \times 256$	$\rightarrow 26 \times 26 \times 128 0.044 \text{ BF}$
19 upsample		2x	$26 \times 26 \times 128$	$\rightarrow 52 \times 52 \times 128$
20 route	19 8			
21 conv	256	$3 \times 3 / 1$	$52 \times 52 \times 384$	$\rightarrow 52 \times 52 \times 256 4.785 \text{ BF}$
22 conv	18	$1 \times 1 / 1$	$52 \times 52 \times 256$	$\rightarrow 52 \times 52 \times 18 0.025 \text{ BF}$
23 yolo				

FIGURE 4.2 – L'architecture du modèle Yolo.

4.2.1 Couches du modèle : convolution, max pooling, route et up-sample

Les couches classiques du modèle Yolo sont les convolutions. Ce sont les couches principales d'un réseau de neurones. Pour expliquer ce qu'est une convolution nous prendrons exemple sur la première couche du réseau (fig.4.3).

0 conv 16 3 x 3 / 1 $832 \times 832 \times 3 \rightarrow 832 \times 832 \times 16$

FIGURE 4.3 – Première couche du réseau

Cette première couche est une convolution de **noyau** $3 * 3$ avec 16 **filtres** et un **pas** de 1.

L'opération de convolution d'une matrice M de taille $n * n * 3$ par un noyau K de taille $k * k * 3$ et un pas s résulte en une matrice de dimension $\frac{n-k+1}{s} * \frac{n-k+1}{s} * 1$.

Afin de ne pas affecter les dimensions de l'image, on effectue une opération de zéro-padding sur l'image d'entrée qui consiste à ajouter des 0 autour de la matrice pour virtuellement augmenter les dimensions de l'image. On part donc avec une matrice M de taille $n + 2 * n + 2 * 3$ pour terminer avec une matrice de taille $\frac{n}{s} * \frac{n}{s} * 1$.

L'élément de la ligne i et de la colonne j de la matrice est défini par :

$$R_{ij} = \sum_{l=1}^3 \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} K_{a,b,l} * M_{i+a,j+b,l} \quad (4.1)$$

Une convolution n'est en réalité qu'une somme pondérée des éléments de la matrice, la figure 4.4 résume ce que fait cette opération. Le nombre de filtres est le nombre de différents noyaux que l'on utilise à la suite. Ici il est de 16, on obtient donc 16 matrices de taille $\frac{n}{s} * \frac{n}{s} * 1 = 832 * 832 * 1$. La matrice de sortie de la convolution est la concaténation de ces 16 matrices, c'est-à-dire une matrice de taille $832 * 832 * 16$.

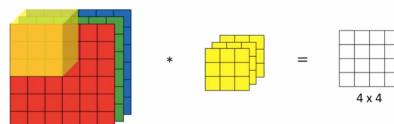


FIGURE 4.4 – Simplification de l'explication d'une convolution.

Les valeurs prises par les noyaux sont les poids que le réseau va entraîner pour que les diverses convolutions extraient les caractéristiques voulues de l'image.

La deuxième couche du modèle est un max pooling, cette couche vise à diminuer la dimension de la matrice d'entrée. Le max pooling de la deuxième couche est effectué avec un **noyau** de taille $2 * 2$ et un **pas** de 2.

$$1 \text{ max} \quad [2 \times 2] / 2 \quad 832 \times 832 \times 16 \rightarrow 416 \times 416 \times 16$$

FIGURE 4.5 – Deuxième couche du réseau Yolo.

L'opération consiste à faire glisser le noyau à travers chaque filtre (ici 16) de la matrice et à ne garder que l'élément le plus grand. L'opération est résumée par un exemple ci-après. Effectuer un max pooling sur une matrice de dimension $n * n * f$ avec un noyau de taille k et un pas p permet d'obtenir une matrice de dimension $\frac{n}{k} * \frac{n}{k} * f$.

$$Maxpool_{2*2/2}\left(\begin{bmatrix} 8 & 6 & 10 & 0 \\ 5 & 2 & 1 & 0 \\ 1 & 5 & 0 & 4 \\ 4 & 7 & 2 & 6 \end{bmatrix}\right) = \begin{bmatrix} 8 & 10 \\ 7 & 6 \end{bmatrix}$$

L'architecture du modèle possède deux autres couches qui ne sont que peu calculatoires. La couche route permet de reprendre la matrice de sortie d'une couche précédente. L'opération "Route 13" reprendra le réseau tel qu'il était à la couche n13. Un résultat a bien entendu été obtenu entre la couche 13 et le layer route.

Le layer upsample est un layer qui augmente la taille de la matrice en remplaçant chaque élément par un groupe d'éléments (exemple ci-après).

$$upsample_{2*}\left(\begin{bmatrix} 8 & 10 \\ 7 & 6 \end{bmatrix}\right) = \begin{bmatrix} 8 & 8 & 10 & 10 \\ 8 & 8 & 10 & 10 \\ 7 & 7 & 6 & 6 \\ 7 & 7 & 6 & 6 \end{bmatrix}$$

4.2.2 La couche Yolo

La couche qui effectue les calculs finaux du réseau est la couche Yolo. Ce layer est du type "Fully Connected". Il effectue une somme pondérée de tous les éléments de la matrice d'entrée pour obtenir un seul nombre en résultat. Les pondérations sont des paramètres à entraîner. Le layer Yolo consiste en $5 + c$ fully connected avec c le nombre de classes que le modèle est entraîné à détecter.

Pour une unique classe (la main) nous avons alors 6 sorties. Ces sorties correspondent aux données permettant de placer l'éventuelle box détectée. Les 5 premiers sont :

- La confiance en la détection, un nombre entre 0 et 1
- La longueur de la box
- La largeur de la box
- La position en X du centre de la box
- La position en Y du centre de la box

La 6e sortie est inutile pour une détection monoclasse. S'il y avait eu plusieurs classes, les sorties restantes auraient correspondu en un vecteur "one-hot" (un vecteur one-hot est un vecteur avec uniquement des 0 et un 1 sur l'information pertinente). L'information codée sur ce vecteur est tout simplement la classe détectée.

Les paramètres de réglage sont importants pour détecter des boxes cohérentes. Avant l'entraînement, un algorithme permet d'analyser l'ensemble des mains de la base de données pour extraire 6 tailles de box (longueur, largeur) permettant d'inclure une majorité du jeu de données. L'image est ensuite quadrillée en plusieurs mini images. Le layer Yolo trouve une box par zone du quadrillage et par taille de box. Le premier appel de la couche trouve des boxes pour les 3 premières tailles de box et le second pour les 3 autres. De nombreuses boxes sont alors détectées et il faut les filtrer (fig.4.6).

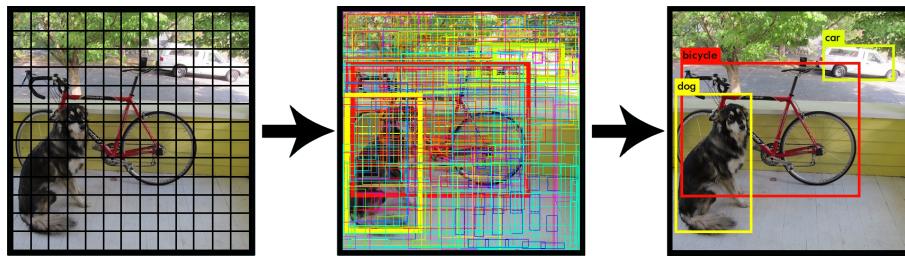


FIGURE 4.6 – Détection puis filtrage de box.

L'algorithme permettant de filtrer les boxes est nommé non max suppression (NMS¹).

Algorithm 1 Non maximum suppression

```

1: procédure NMS
2:   Définir Seuil confiance
3:   Supprimer toutes les boxes avec un score < Seuil confiance
4:   for Nombres de classes do
5:     while Des prédictions ne sont pas triés do
6:       Garder la box avec la plus grande confiance
7:       Supprimer les boxes trop proche de la box sélectionnée      ▷ Pour ne pas
   détecter deux fois le même objet ; La proximité entre deux boxes sera expliquée ensuite

```

Celui-ci traite les prédictions classes par classes (nous n'avons ici qu'une seule classe). La prédiction avec la plus grande probabilité d'être l'objet souhaité est gardée et les boxes trop proches de cette dernière ne le sont pas. En supprimant dans un premier temps toutes les prédictions avec un score trop bas, seul un nombre pertinent de prédictions sont gardées.

4.2.3 Hyperparamètres

Les hyperparamètres d'un modèle sont les paramètres qui doivent se régler à la main avant le début de l'apprentissage. Par exemple pour les deux couches Yolo du modèle il y a plusieurs hyperparamètres à fixer :

- Les "anchors", qui sont les tailles prédéfinies des boxes en fonction des annotations du jeu d'entraînement
- Le nombre de classes (pour savoir le nombre de sorties à donner à la couche)
- La valeur du seuil pour le NMS

Pour définir la taille des anchors nous utilisons une fonction intégrée à darknet qui analyse l'ensemble des boxes du jeu de données, les réparties en six sous-catégories, et compute les tailles optimales des six anchors box. Le nombre de classes est fixé ici à 1 et nous avons placé le seuil de NMS à 0.7. Pour choisir ce nombre nous avons effectué différent tests grâce à un jeu de données inconnues du modèle pour son entraînement. Nous avons ensuite fait converger la valeur vers le seuil qui donnait les meilleurs résultats.

Un autre hyperparamètre qui peut être réglé est la taille de la grille qui vient diviser l'image. Nous avons ici fait le choix de laisser la taille originale proposée avec le modèle, car toutes

1. *Non Max Suppression*

nos tentatives de changement n'ont affecté le modèle qu'en mal. Les images sont initialement traitées par le modèle pour une taille $416 * 416 * 3$. Après avoir constaté que cette dimension fait perdre trop d'informations comparée à la résolution de la webcam utilisée pour le projet, la dimension des images d'entrée du modèle a été revue pour $832 * 832 * 3$. Cela ralentit beaucoup l'entraînement mais une fois qu'il est effectué cela n'affecte que peu les performances en vitesse du modèle (moins de 1fps).

L'apprentissage du modèle se fait en affinant les divers poids vus précédemment. Pour affiner ces poids, on va chercher à minimiser pour chaque détection de la base d'entraînement une fonction appelée fonction de coût, ou *loss* en anglais. Cette fonction est calculée ainsi :

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

FIGURE 4.7 – Fonction de coût de Yolo tirée de [11]

Cette fonction peut se séparer en une somme de trois blocs. Les trois blocs servent à calculer une erreur. Pour une prédiction parfaite, le coût sera donc de 0. Le **premier bloc** calcule l'erreur de position de la box par rapport au ground-truth (gls pg-8). Les valeurs x, y, w et h sont respectivement la position, la largeur et la hauteur de la box. La constante λ_{coord} représente le poids que l'on donne à cette partie de la fonction *loss*. La fonction $\mathbb{1}_{ij}^{\text{obj}}$ renvoie 1 si un objet est détecté, 0 sinon.

La **deuxième partie** de la fonction fait augmenter la fonction de coût en fonction la confiance du réseau en sa détection pour la première double somme et en sa non-détection pour la seconde. C est la confiance donnée par le modèle dans la box. Par exemple, un vrai positif avec peu de confiance sera pénalisé en moindre mesure qu'un faux positif.

Enfin, la **dernière partie** de la fonction inflige une pénalité au modèle (en faisant augmenter la *loss*) si un objet est mal classé (par exemple un chien classé en chat). Dans le cas d'une classe unique, cette partie n'est pas réellement nécessaire.

L'optimisation de cette fonction se fait via une descente de gradient. Le pas de cette descente est appelé le learning rate. Il s'agit aussi d'un hyperparamètre qu'il est nécessaire de régler. Le pas d'apprentissage diminue au cours de l'apprentissage afin de faire une descente de plus en plus précise.

4.2.4 Évaluation des résultats

Une fois un modèle entraîné, il faut pouvoir évaluer ses performances et le comparer aux autres modèles. Le modèle affine ses poids via le jeu d'entraînement, il le fait en réduisant le résultat de la fonction de coût pour le jeu d'entraînement. Un premier moyen efficace pour vérifier de l'efficacité du modèle est de regarder la courbe de la fonction *loss* au cours de l'entraînement. Celle du jeu d'apprentissage converge vers 0 mais pas forcément celle du jeu de test fig.4.8, cela s'appelle le surapprentissage. Le modèle apprend trop, dans ce cas, les features du jeu d'entraînement et est moins capable de généraliser sur d'autres images. Dans ce cas, il est préférable d'utiliser les poids du modèle au point de divergence de la courbe.

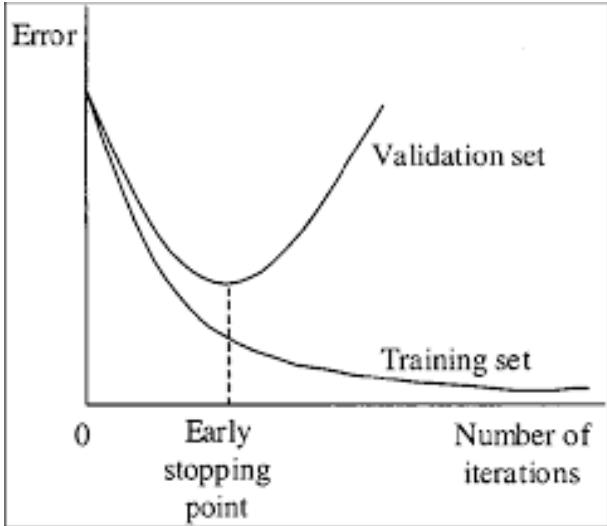


FIGURE 4.8 – Exemple de surapprentissage

Pour avoir une idée chiffrée de l’efficacité d’un modèle, il faut pouvoir qualifier ce qu’est une bonne détection. En effet, on ne considère pas comme bon uniquement les boxes qui correspondent parfaitement à la box labellisée. Premièrement, la labélisation est faite à la main et certaines boxes sont parfois un peu grandes. Ensuite, car une box légèrement différente de la ground-truth peut quand même suffire à encadrer la main.

On utilise une mesure nommée l’IoU² pour qualifier une bonne ou mauvaise détection. Cet outil permet de comparer deux boxes : la box détectée et la box de ground truth correspondante. Il s’agit du rapport entre la surface commune aux deux boxes et la surface totale occupée par les deux boxes [15]. Il est compris entre 0 et 1, un IoU de 1 correspond à deux box superposées et un IoU de 0 à deux boxes sans aucune correspondance.

$$IoU = \frac{S_{GT} \cap S_{detect}}{S_{GT} \cup S_{detect}} \quad (4.2)$$

Avec S_{GT} la surface de la box annotée et S_{detect} la surface de la box détectée. On considère une détection comme bonne pour un IoU supérieur à 0.5. C’est à dire que la moitié de la surface des boxes est commune aux deux boxes. Ce nombre correspond aux critères d’évaluation du challenge Pascal VOC (visual object classes) [16], référence dans le domaine. Un IoU de 0.75 est aussi un nombre utilisé pour qualifier une bonne détection, mais nous avons préféré avec 0.5 : ce nombre ne permet pas d’avoir de meilleurs résultats (l’entraînement est le même), mais il classifie plus de détection comme bonne ; nous considérons qu’une détection avec un IoU de 0.5 est suffisante pour localiser la main.

L’IoU permet donc de faire la différence entre vrai positif (TP³) et faux positif (FP⁴). Cette mesure ne permet en revanche pas d’attribuer un score au réseau. Pour faire cela, on utilise une mesure nommée mAP. Le mAP est un score en % qui permet d’évaluer notre modèle. Pour le calculer, il faut, dans un premier temps, calculer la précision et le rappel pour toutes les images de la base de données.

-
- 2. *Intersection over Union*
 - 3. *True Positive*
 - 4. *False Positive*

- La précision peut être interprétée comme le pourcentage de détections correctes.

$$Precision = \frac{TP}{TP + FP} \quad (4.3)$$

- Le rappel peut être interprété comme le pourcentage d'objets détectés. Il s'agit du rapport entre le nombre d'objets détectés et le nombre d'objets labelisés.

$$Rappel = \frac{TP}{TP + FN} \quad (4.4)$$

Avec FP le nombre de faux positifs (fausses détections) et FN le nombre de faux négatifs (objet non détecté).

Il est alors possible pour chacune des classes de placer les couples précision/rappel afin d'effectuer une courbe telle que celle montrée en 4.9. L'Average precision (AP) de la classe est alors l'aire sous la courbe. Le mAP est la moyenne de l'average precision de toutes les classes.

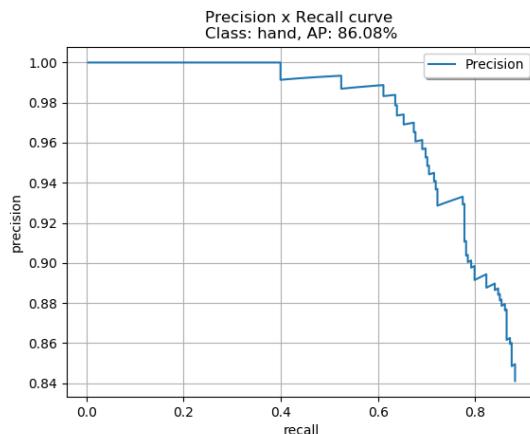


FIGURE 4.9 – Courbe précision rappel

Sur ce modèle, on obtient un mAP de 86%. En testant ensuite le modèle nous avons pu remarquer que les mains étaient bien détectées de face, mais que les mains sur le côté l'étaient un peu moins. Aussi, ce modèle avait tendance à détecter des coudes. C'est pourquoi après cette version nous avons décidé d'agrandir la base de données en ajoutant notamment des photos de coudes et de mains de côté.

4.3 Cr éation d'une base d'image

Il faut une base de données conséquente et adaptée pour entraîner le modèle. Pour commencer, nous avons rempli la base d'un grand nombre d'images prises dans notre open-space. Plusieurs centaines d'images de mains de diverses personnes travaillant pour Aubay permettent de détecter les mains en direct après entraînement. Pour entraîner, il faut non seulement la base d'image, mais aussi la base correspondante de labels : un fichier texte par image contenant autant de lignes que de mains à l'image. Chaque ligne est remplie des coordonnées et dimensions de la box correspondant à la main. Pour labelliser les images de la base de données, nous utilisons un script open source [17].

Cependant, entraîner le modèle uniquement sur des mains provenant de notre bureau a mené à un surapprentissage : le modèle n'était pas efficace dans un environnement extérieur. Nous avons donc souhaité rajouter un grand nombre d'images issues de bases de données extérieures. Après avoir ajouté des images de [18, 19, 20], le modèle est capable de détecter les mains aussi bien dans notre bureau que dans des environnements extérieurs.

Notre jeu de données est ensuite séparé en trois jeux de données distincts : entraînement, validation et test. Le jeu d'entraînement est celui utilisé pour minimiser la fonction de coût. A partir de la millième itération et toutes les 100 itérations, la *loss* et le mAP sont calculés sur le jeu de validation. Cela permet de détecter un éventuel surapprentissage. Enfin, le jeu de test permet de tester le modèle après l'entraînement. La séparation est effectuée ainsi : 80% d'images dans le jeu d'entraînement, 15% dans le jeu de validation et 5% dans le jeu de test.

4.4 Ajout de détection de tête

Dans la suite de notre projet, nous utilisons la box de détection de la main pour effectuer une binarisation de celle-ci (chapitre 5). La présence de visage pose quelques problèmes pour cette binarisation. Afin de savoir si un visage est présent et proche de la main, nous avons ajouté une seconde classe au modèle. Le modèle détecte désormais les têtes et les mains.

Cet ajout nous permet aussi de connaître le nombre de personnes à l'écran. Nous pensions exploiter cette information en avertissant l'utilisateur qu'une potentielle deuxième personne pourrait tenter d'utiliser l'application, mais n'avons pas eu le temps d'implémenter cette fonction.

Pour pouvoir détecter les visages nous avons dû re-labelliser l'ensemble de la base de données en ajoutant tous les visages. Les têtes de dos n'ont pas été labellisés car inutiles. Nous avons aussi modifié l'architecture du modèle afin d'intégrer deux classes.

4.5 Résultats finaux

Après divers entraînements, modifications des hyperparamètres et ajustement de la base d'images, nous avons obtenu un modèle avec des résultats suffisamment convenables pour détecter la majorité des mains et des têtes en temps réel. Le modèle obtient une average precision de 89% pour les mains et 81.5% (fig.4.10) pour les têtes soit un mAP de 85.2%.

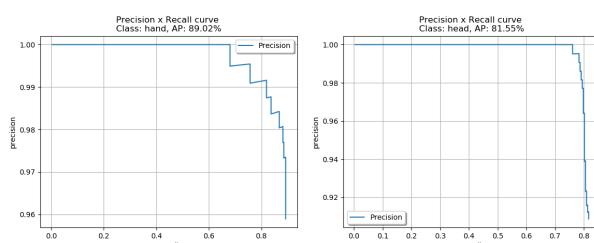


FIGURE 4.10 – Courbe précision rappel pour les classes tête et main

La figure 4.11 montre l'évolution de la fonction loss sur le jeu de validation. On peut remarquer que la courbe est décroissante ce qui signifie que l'entraînement a bien minimisé la fonction de coût. La courbe rouge est quant à elle l'average precision pour la classe main en fonction des itérations. Le modèle retenu n'est pas celui de la dernière itération car un modèle obtenu précédemment (autour de la 5000e itération) obtient un meilleur AP sur la classe main.

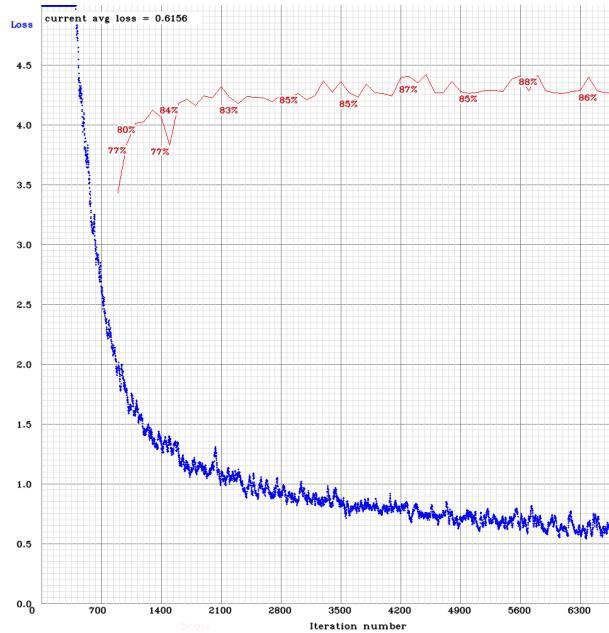


FIGURE 4.11 – Courbe de *loss* en fonction du temps et évolution liée du mAP

Enfin, la figure 4.12 montre une image de la base de test avec les boxes correspondant au ground-truth (en bleu pour les têtes et vert pour les mains) et les boxes correspondant à la détection (en violet pour les têtes et jaune pour les mains).

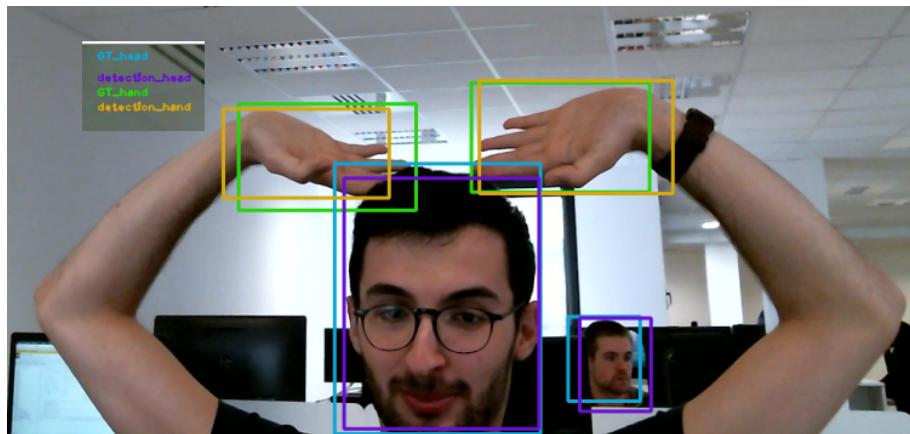


FIGURE 4.12 – Exemple de détection par le modèle sur le jeu de test

Lors de l'entraînement, le réseau n'a pas eu accès à cette image. Les boxes détectées correspondent pourtant grandement à celles annotées à la main. Les quelques erreurs restantes sont majoritairement que des faux positifs pour les têtes de dos (ce qui n'est pas dérangeant pour la suite) et des faux négatifs pour des mains dans des positions difficiles à détecter (par-dessus le visage par exemple).

Chapitre 5

Traitement sur les mains détectés

Le chapitre précédent montre comment il est possible de connaître la position de chaque main sur l'image. Dans cette partie, nous travaillons désormais avec une image de plus petite taille. L'image de travail est maintenant la box encadrant la main¹. Le travail effectué ensuite vise à extraire de cette image les caractéristiques de la main détectée. Les travaux sont effectués à l'aide de la bibliothèque OpenCV ainsi qu'en implémentant des fonctions à la main.

5.1 Binarisation de l'image

La première étape du traitement consiste à binariser la main sur l'image, c'est-à-dire obtenir une image binaire (un pixel est noir **ou** blanc) avec, en blanc, les pixels de la main et en noir, tout le reste.

Pour cela, nous faisons un travail sur la couleur. Nous ne travaillons pas avec l'image dans l'espace RGB car il ne permet pas de différencier facilement la main du reste. Nous convertissons notre image dans l'espace de couleur LAB² (fig 5.1). Cet espace permet de séparer la luminosité du reste, ce qui résout les problèmes d'ombres sur la main. La composante A comporte les couleurs de vert à magenta et B de bleu à jaune.



FIGURE 5.1 – Passage dans l'espace de couleur LAB

Pour binariser l'image, nous utilisons ensuite la méthode d'Otsu qui permet d'effectuer un seuillage automatique de l'image. Une fois l'image binarisée, il faut améliorer la binarisation en

-
1. Nous prenons en réalité une image un peu plus grande que la box pour ne pas perdre d'information
 2. *Luminosité A B, espace de couleur*

supprimant le bruit restant (5.2). Pour supprimer le bruit, nous effectuons une érosion suivie d'une dilatation (les deux opérations à la suite portent le nom d'ouverture).



FIGURE 5.2 – Binarisation de la main et filtrage

Le défi rencontré pour l'érosion a été la taille du noyau. En effet, si la main est loin de la caméra, elle sera uniquement représentée par un petit nombre de pixels. Un trop grand noyau l'effacerait. A l'inverse, si la main est proche de la caméra, un trop petit noyau n'aurait pas d'influence sur le bruit. La solution a alors été de faire évoluer la taille du noyau en fonction de la taille de la box encadrant la main.

5.2 Opération sur le contour de main

Une fois l'image binarisée, il est simple d'obtenir le contour de la main (fig.5.3). Nous l'obtenons à l'aide d'une fonction directement implémentée dans OpenCV. Nous calculons aussi le contour de Hull de l'image. Il s'agit du plus petit contour convexe incluant l'objet (fig.5.3).

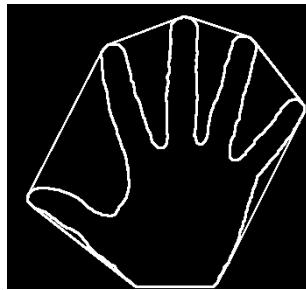


FIGURE 5.3 – Contour et contour de Hull d'une main

A l'aide des moments de l'image, il est ensuite possible de trouver le centre de la main. La matrice des moments est une matrice de taille $2 * 2$ représentant le poids des pixels sur l'image.

Pour une image de taille $l * L$, l'élément $M_{i,j}$ de la matrice des moments vaut :

$$M_{ij} = \sum_{x \in L} \sum_{y \in l} x^i y^j * I(x, y) \quad (5.1)$$

Avec $I(x,y) = 0$ si pixel noir, 1 sinon.

Le centre de l'objet a alors pour coordonnées $(x, y) = (\frac{M_{1,0}}{M_{0,0}}, \frac{M_{0,1}}{M_{0,0}})$

Le contour de Hull nous permet de savoir si la main est fermée ou ouverte. En effet, une main ouverte aura un contour de Hull peu similaire à son propre contour (voir fig.5.3). A l'inverse,

un main fermée est une forme convexe et celle-ci sera donc proche de son contour de Hull. Des exemples sont fournis en Annexe.B.

Par la suite, nous utilisons les courbes convexes du contour pour compter les doigts. En traçant le **début** de la courbe en bleu, la **fin** en rouge et le **point le plus éloigné du contour de Hull** en vert, il est facile de comprendre que les doigts peuvent être comptés ainsi (voir fig.5.4). Le cercle bleu est le centre de la main. Il est bleu si la main est ouverte et vert dans le cas contraire.

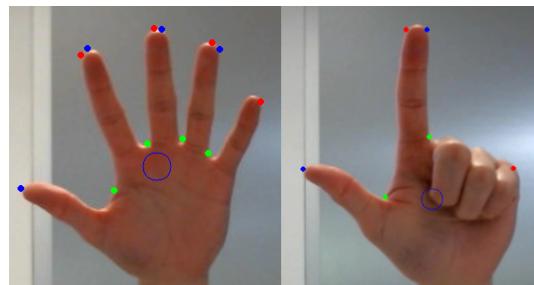


FIGURE 5.4 – Points caractéristiques de la main

Lorsqu'il y a deux points (rouge et bleu) proches, nous savons que nous ne faisons pas face à un doigt extérieur (pouce ou auriculaire pour une main ouverte par exemple). Il suffit de choisir l'un des deux points pour extrémité du doigt. Pour vérifier la proximité de deux points, on regarde leur distance sur le contour et non la distance dans l'espace. En effet, deux extrémités de doigts différents sont proches dans l'espace mais éloignées sur le tracé du contour.

Un point unique n'est en revanche pas nécessairement un doigt (exemple : le point de droite sur la deuxième image de la fig.5.4). Pour vérifier si un point unique appartient bien à un doigt, on utilise la distance au centre pour vérifier s'il s'agit bien de l'extrémité de l'un d'eux.

5.3 Détection du squelette de main

Étant désormais capables de compter le nombre de doigts, nous avons souhaité différencier une variété de gestes avec le même nombre de doigts levés. Par exemple, pour deux doigts, différencier le geste "V" du signe "Deux". Pour cela, nous nous sommes inspirés d'Openpose et avons tracé une estimation du squelette de la main (voir fig.5.5).

Pour pouvoir tracer ce squelette, nous parcourons le contour des deux côtés d'une distance égale au tiers de celle entre le centre de la main et l'extrémité du doigt. La valeur d'un tiers est le rapport entre les différentes phalanges sur un doigt. Une fois ces deux points obtenus, le milieu du segment créé par ceux-ci devient une jonction du squelette. En répétant cette étape une seconde fois puis en reliant le dernier segment au centre de la main, on obtient les squelettes de la figure 5.5.

Les angles formés par les segments du squelette sont différents si deux gestes distincts sont effectués : c'est en analysant ces différents angles que nous sommes capables de différencier le geste "V" du geste "Deux".

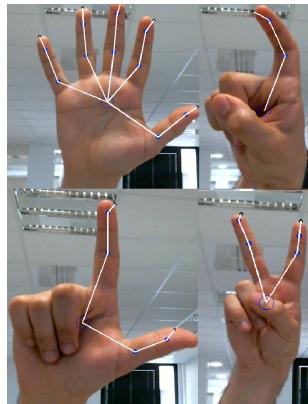


FIGURE 5.5 – Différentes images de squelette de la main

5.4 Structure de la main et résultats

Le travail effectué permet donc de savoir si la main est ouverte ou fermée, de compter le nombre de doigts levés ainsi que d'identifier le geste effectué et de connaître la position du centre de la main. Toutes ces informations sont stockées dans une instance d'un objet de type "main". En plus de ces informations, nous renseignons aussi dans cette structure un numéro unique à la main pour pouvoir travailler avec plusieurs mains à l'écran.

Un total de dix gestes différents, représentés en figure 5.6, sont donc détectables à ce stade du projet. Je vous invite à regarder la vidéo disponible sur le site proposé en Annexe-A pour avoir un aperçu en direct de la détection de ces gestes. Nous pourrions avoir plus de gestes car nous sommes capables de différencier la main gauche de la main droite, mais un total de dix gestes est déjà suffisant. Nous n'avons pas eu le besoin d'exploiter cette idée durant le projet.



FIGURE 5.6 – Les dix gestes compris par notre programme

5.5 Soustraction du fond et tracking de main

Plusieurs améliorations ont ensuite été ajoutées pour rendre plus efficace cette détection, notamment un suivi de main et une soustraction du fond. La soustraction du fond a pour but de rendre meilleure la binarisation. Son fonctionnement est simple mais n'est possible qu'avec une caméra fixe. Une matrice de taille égale à l'image de sortie de la caméra est remplie par les pixels de la caméra, excepté la zone de la main (la zone est déterminée grâce à la box). Au moment d'effectuer la binarisation pour les images suivantes, tous les pixels qui étaient déjà présents sur la matrice de fond sont supprimés. Cela permet donc d'effacer tous les objets fixes du décor (fig.5.7).

La figure B.2 en annexe B donne des exemples de binarisation avec des fonds difficiles.

La seconde amélioration est guidée par deux ambitions. La première est de pouvoir implémenter convenablement le travail avec deux mains et l'autre est de ne pas perdre la main, par

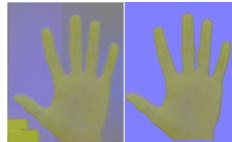


FIGURE 5.7 – LAB avant et après soustraction du fond

exemple si une deuxième personne passe derrière l'écran. Cette amélioration est le tracking de main. La première main qui apparaît à l'écran sera la main à suivre. Si l'on travaille avec deux mains, seules les deux premières mains détectées seront les mains à suivre, même si d'autres peuvent se trouver dans l'arrière-plan. Leurs positions sont enregistrées. Une main détectée sur la frame suivante ne sera traitée que si elle se situe suffisamment proche de la position d'une des deux mains à l'instant précédent. Lorsqu'une main disparaît de l'écran, il faut attendre un certain laps de temps avant de considérer une nouvelle main comme étant la main à suivre. Notre seul problème a été de savoir quelle main est celle à suivre lorsque deux mains se placent au même endroit de l'image. Actuellement, l'algorithme sélectionne l'une des deux mains de manière aléatoire. Cela fait partie des axes d'amélioration de notre programme.

Chapitre 6

Détection des mouvements

La partie précédente permet de détecter 10 gestes différents. Cette partie vise à analyser le déplacement de la main pour détecter des mouvements. Pour cela nous utilisons à nouveau un réseau de neurones entraîné pour détecter les mouvements.

6.1 Les mouvements à détecter

Nous nous sommes fixés cinq mouvements à détecter. Des versions plus évoluées du projet pourraient, en théorie, reconnaître plus de mouvements. Les mouvements doivent être faciles à réaliser et l'action qui leur est liée doit être logique. Ces cinq mouvements sont :

- Le cercle \circ
- Les flèches dans les quatre directions possible $\wedge \vee < >$

6.2 Conversion du mouvement en une image

Pour détecter les mouvements de manière fluide et rapide nous avons trouvé un moyen de les convertir en images. En effet une image est moins longue à traiter par un CNN qu'un extrait vidéo par exemple. Notre idée a donc été d'enregistrer la position du centre de la main sur une période et d'afficher la courbe effectuée par ce centre. Ainsi les mouvements à détecter ont pu être convertis en images (fig.6.1).

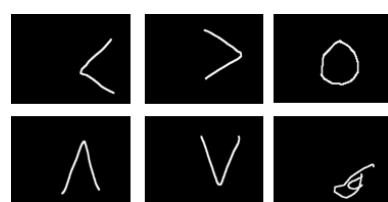


FIGURE 6.1 – Aperçu de la translation en image des cinq mouvements détectables ainsi que d'un mouvement parasite.

Dans l'utilisation réelle (post-entraînement, sur le logiciel), la courbe du mouvement est implémentée par une file. La liste des points à tracer est de taille constante afin de ne pas avoir tout l'historique des déplacements de la main sur l'image.

6.3 Modèle choisi et entraînement

Notre modèle ne doit pas être trop lourd pour ne pas trop affecter la fluidité du programme. Nous nous sommes donc limités à cinq couches de convolution et une couche complètement connectée (6.2). Nous avons d'abord tenté de travailler avec quatre couches mais les résultats avec cinq couches étaient bien meilleurs comparés à ceux obtenus sans perte de vitesse (voir Annexe.C). Nous avons ensuite des couches de zero-padding entre chaque convolution, pour ne pas perdre d'informations sur l'image, ainsi que de max-pooling, pour réduire la dimension traitée (voir 4.2 pour informations). Nous avons aussi ajouté du dropout qui consiste à ne pas prendre en compte une partie des noeuds du réseau à chaque itération. Ici, nous supprimons environ 20% des noeuds à chaque itération : l'estimation à 20% provient du fait que chaque poids a une probabilité de 0.2 de ne pas compter pour chaque itération. Cette action permet de réduire le surapprentissage qui a posé de gros problèmes sur ce modèle.

Notre base d'images est assez limitée (environ 250 images par classe) et nous avons choisi de traiter le problème avec un réseau classifieur. Chaque objet est attribué à une classe avec une certaine probabilité en sortie du réseau. Nous n'avons pas de classe pour les mouvements parasites car cette classe était trop dure à entraîner et détériorait les résultats du modèle.

Le modèle est créé sous Keras, une surcouche de Tensorflow. Keras propose plus d'options que Darknet, comme par exemple l'utilisation de Tensorboard pour comparer différents modèles et choisir le meilleur. Il permet aussi d'établir facilement la fonction *loss*, les hyperparamètres et la métrique utilisée pour connaître la précision du modèle.

Le modèle utilise une fonction de coût classique qui augmente si une image est mal classifiée. Nous utilisons l'optimiseur Adam pour la descente de gradient. On peut, grâce au tableau, compter le nombre de paramètres du réseau. Le réseau entraîne un total de 20021 paramètres, ce qui est très peu.

Layer (type)	Output Shape	Param #
zero_padding2d_36 (ZeroPaddi	(None, 518, 518, 1)	0
conv2d_36 (Conv2D)	(None, 516, 516, 8)	80
activation_38 (Activation)	(None, 516, 516, 8)	0
max_pooling2d_36 (MaxPooling)	(None, 172, 172, 8)	0
dropout_15 (Dropout)	(None, 172, 172, 8)	0
zero_padding2d_37 (ZeroPaddi	(None, 178, 178, 8)	0
conv2d_37 (Conv2D)	(None, 176, 176, 16)	1168
activation_39 (Activation)	(None, 176, 176, 16)	0
max_pooling2d_37 (MaxPooling)	(None, 58, 58, 16)	0
zero_padding2d_38 (ZeroPaddi	(None, 64, 64, 16)	0
conv2d_38 (Conv2D)	(None, 62, 62, 16)	2320
activation_40 (Activation)	(None, 62, 62, 16)	0
max_pooling2d_38 (MaxPooling)	(None, 20, 20, 16)	0
zero_padding2d_39 (ZeroPaddi	(None, 26, 26, 16)	0
conv2d_39 (Conv2D)	(None, 24, 24, 32)	4640
activation_41 (Activation)	(None, 24, 24, 32)	0
max_pooling2d_39 (MaxPooling)	(None, 8, 8, 32)	0
zero_padding2d_40 (ZeroPaddi	(None, 14, 14, 32)	0
conv2d_40 (Conv2D)	(None, 12, 12, 32)	9248
activation_42 (Activation)	(None, 12, 12, 32)	0
max_pooling2d_40 (MaxPooling)	(None, 4, 4, 32)	0
dropout_16 (Dropout)	(None, 4, 4, 32)	0
reshape_4 (Reshape)	(None, 512)	0
dense_3 (Dense)	(None, 5)	2565
activation_43 (Activation)	(None, 5)	0
<hr/>		
Total params:	20,021	
Trainable params:	20,021	
Non-trainable params:	0	

FIGURE 6.2 – Structure du modèle pour détecter les mouvements.

6.4 Résultats

Le modèle réussit à détecter les mouvements avec une importante précision de 97.5% sur le jeu de validation. Ce résultat n'est pas très surprenant car les formes à détecter sont assez basiques et les mouvements parasites ne sont pas pris en compte. La figure 6.3 montre l'évolution de la fonction de coût et de la précision des jeux d'entraînement et de validation au cours de l'apprentissage. On peut remarquer que la fonction de coût s'annule à la fin de l'entraînement sur le jeu d'entraînement, ce qui mène à une précision parfaite pour ce jeu de données. En revanche, elle est un peu en dessous sur le jeu de validation ce qui explique la précision de 97.5%.

Pour régler le problème des mouvements parasites qui sont eux aussi classifiés dans l'une des cinq classes, nous avons pris en compte la confiance qu'a le réseau en sa détection. En effet, si l'image à détecter est un mouvement parasite, elle ne rentre dans aucune classe et la confiance du réseau en sa prédiction sera faible. A l'inverse, une image n'étant pas un mouvement parasite sera détectée par le réseau avec une confiance proche de 1. Nous avons donc choisi comme seuil 0.9. Ainsi, si une image est classée avec une probabilité supérieure à 0.9, alors elle ne sera pas

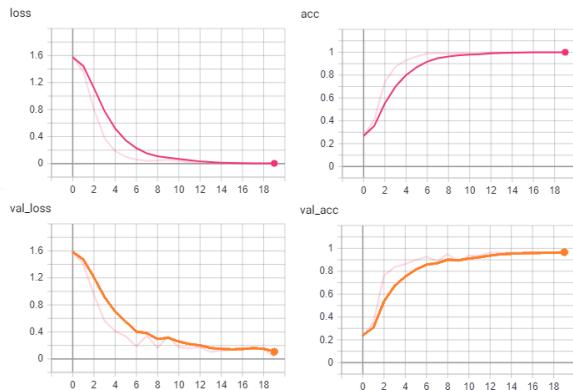


FIGURE 6.3 – Évolution de la fonction de coût et de la précision au cours de l'apprentissage pour les jeux d'entraînement et de validation.

considérée comme un mouvement parasite. En prenant en compte ce nouveau paramètre et en ajoutant de nombreux mouvements parasites dans une base de test, le modèle réussit à obtenir une précision de 95.5%. Je vous invite à regarder une nouvelle fois la vidéo disponible sur le site proposé en Annexe-A pour avoir un aperçu en direct de la détection des mouvements.

Chapitre 7

Reconnaître une action et interagir

Les parties précédentes ont montré que notre algorithme est capable de détecter 10 gestes et 5 mouvements. Cette partie vise à prouver comment la combinaison de ces gestes et mouvements entraîne une grande diversité d'actions réalisables.

7.1 Les différents types d'interactions

La première action que peut réaliser le programme n'est pas liée à la détection d'un geste ou d'un mouvement, mais uniquement à la détection de la main. Il s'agit du contrôle de la souris. En effet, la souris est contrôlée uniquement à l'aide de la position du centre de la box de détection de la main (plus stable que le centre de la main). Le déplacement du curseur est fidèle au déplacement de la main.

Le clic est l'interaction évidente pour suivre le contrôle du curseur de la souris. L'action du clic est réalisée en effectuant le chiffre deux (qui bloque le curseur de la souris) puis en fermant la main. Ce type d'action consiste en une succession de gestes.

Une autre manière de réaliser une action est de combiner un geste et un mouvement. Par exemple, si le programme détecte le mouvement cercle et que le mouvement est effectué le poing fermé, une actualisation de page est réalisée (sur navigateur par exemple).

Enfin, la dernière façon d'effectuer une action est la combinaison de deux mains. La seule action implémentée est le zoom arrière. Si le mode deux mains est activé et que les deux mains font le geste "deux", rapprocher les mains implique un zoom avant et les éloigner implique un zoom arrière.

Pour implémenter ces gestes, nous simulons des raccourcis clavier via la librairie Windows de C++.

7.2 Liste des actions réalisables

Interaction	Action réalisée
	Contrôle de la souris
	Contrôle de la souris en mode précision (déplacement léger)
puis	Clic
	Changement de fenêtre (Alt-tab)
et	Actualisation de page
et >	Diapositive suivante (en mode diapo)
et >	Suivant (Ctrl Y)
et <	Diapositive précédente (en mode diapo)
et <	Précédent (Ctrl Z)
et	Zoom
et ✓	Réduit la page
	Défilement
	Ferme la fenêtre
et ^	Active ou désactive le mode diapo
et ✓	Active ou désactive le contrôle de la souris
et	Active ou désactive les interactions (autres que la souris)

TABLE 7.1 – Les actions réalisables par le programme

Chapitre 8

Interface d'utilisation du projet et résultats

Lorsque les premières fonctionnalités du projet ont commencé à être implémentées, nous avons développé deux interfaces graphiques sous Qt (gls pg-8). La première a pour ambition de faciliter le développement des prochaines versions alors que la seconde est l'interface finale d'utilisation. Ce chapitre décrit ces deux interfaces.

8.1 Première interface de test

La première interface (fig 8.1) a pour but l'aide au débogage du code. Celle-ci permet d'afficher aisément les différentes images clefs du projet. L'interface se compose de deux cadres d'images dans lesquels on peut afficher au choix : l'image classique (webcam), les boxes détectées grâce à Yolo, le contour détecté de la main, l'extrémité des doigts ou encore le squelette de la main. La présence de deux fenêtres permet d'afficher deux de ces images en même temps.

Nous pouvons aussi cocher une case afin d'activer ou non les interactions avec l'ordinateur. Enfin, un troisième cadre permet d'afficher le geste détecté. Cette version ne prend pas encore en compte les mouvements ni les interactions avec deux mains. Les images sont actualisées toutes les 30ms soit une capacité de 33 fps. Ce temps d'actualisation est suffisant en comparaison avec temps de calcul de notre logiciel.

La deuxième version (fig 8.2) prend en compte ces ajouts. Nous avons toujours les deux cadres permettant d'afficher les différentes images. Le cadre pour détecter les gestes est lui aussi toujours présent, un plus grand nombre de gestes sont cependant implémentés dans cette version. Un second cadre permet d'afficher les divers mouvements lorsqu'ils sont détectés.

Cette interface fonctionne aussi avec deux mains, une case cliquable permet de choisir le nombre de mains devant être prises en compte. Si deux mains sont présentes quand une seule doit être prise en compte, le tracking de main permet de se focaliser uniquement sur la première main apparue à l'écran. La souris est désormais dissociée du reste des interactions, il est possible d'activer la souris sans activer le reste des mouvements (zoom, changement de fenêtre ...) et vice versa.



FIGURE 8.1 – Aperçu de la première version de l’interface de tests.

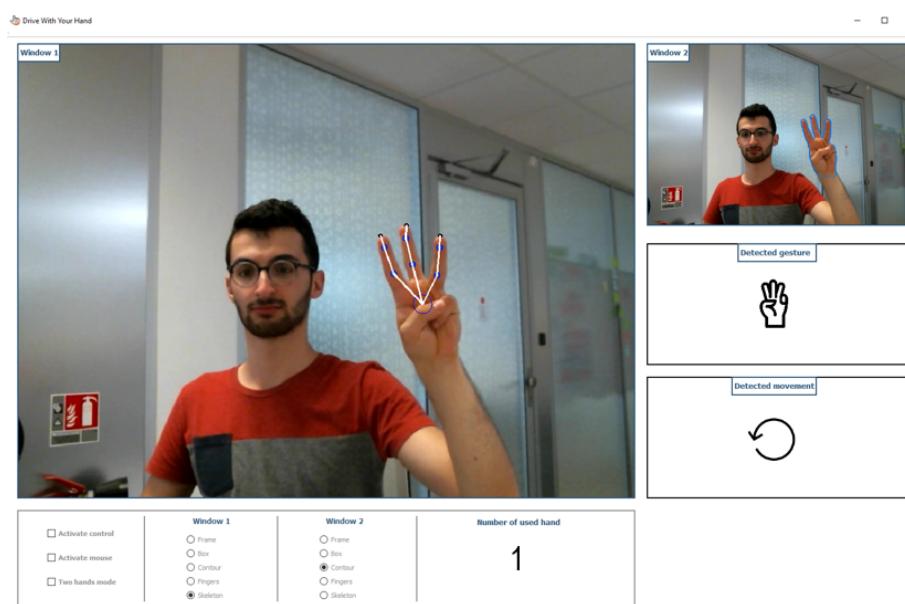


FIGURE 8.2 – Aperçu de la deuxième version de l’interface de tests.

L'affichage en temps réel de l'interface cumulé au temps de calcul du traitement ne permettait pas un rendu fluide. L'affichage des fenêtres de manière linéaire est trop coûteux en temps pour un rendu fluide. Pour corriger cela, l'interface utilise des Qthread (gls pg-8) afin de paralléliser les différents affichages.

8.2 Interface projet fonctionnel

La seconde interface du projet possède une ambition d'interface utilisateur. Il s'agit d'une fenêtre « always on top », c'est-à-dire qu'elle reste à l'écran même si elle n'est pas active. Cette fenêtre est beaucoup plus petite et a pour but d'informer l'utilisateur du bon fonctionnement de l'application. Elle permet de vérifier que la main est bien dans le cadre de la caméra et qu'elle est bien segmentée. Elle affiche aussi les gestes détectés

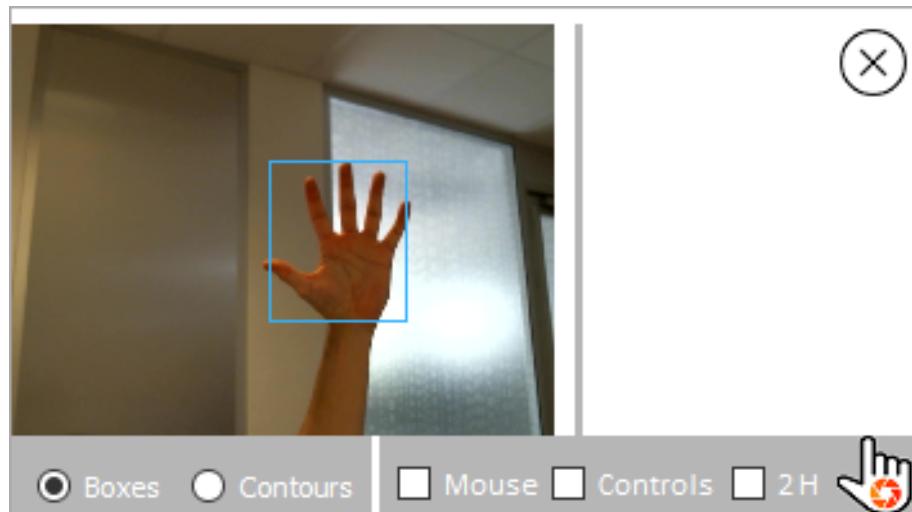


FIGURE 8.3 – Aperçu de la première version de l'interface utilisateur.

La première version (fig.8.3) permet d'afficher la boîte détectée et les contours. A ce stade, les mouvements ne sont pas encore détectés et les gestes ne s'affichent pas en bas de l'écran.

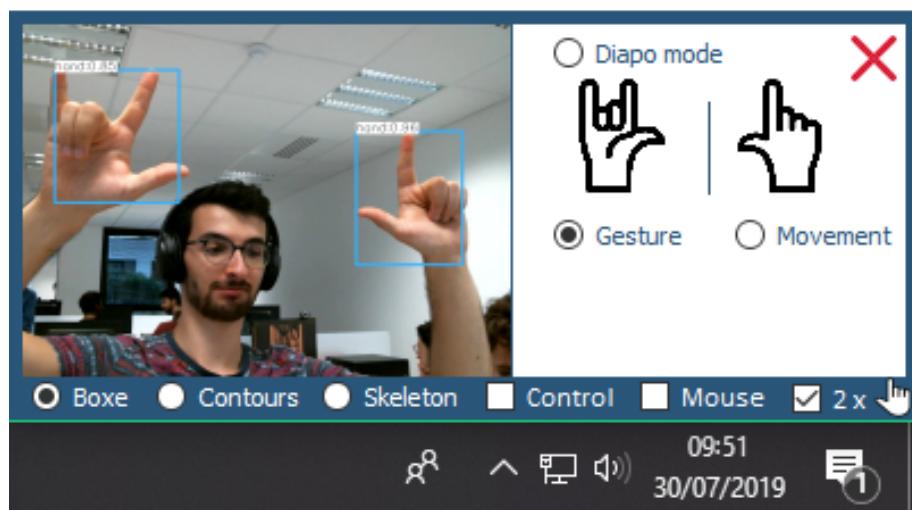


FIGURE 8.4 – Aperçu de la deuxième version de l'interface utilisateur.

La seconde version (fig.8.4) permet d'afficher les gestes ou mouvements détectés, de changer rapidement de mode (souris, contrôle total ou mode diaporama) et de travailler avec deux mains. Les icônes ont été mises à jour pour différencier la main gauche de la main droite. La barre de tâches sous l'application permet de se donner une idée de la taille de l'interface.

Conclusion et perspectives

L'objectif de ce projet était de permettre une communication entre l'Homme et la machine grâce à une compréhension des interactions faites avec les mains. Ce stage se termine avec un outil permettant de lier dix gestes et cinq mouvements, soit un grand nombre d'interactions, à des actions sur la machine. L'outil final tourne en analysant 15 images par seconde avec un CPU classique et 18 avec un petit GPU (QUADRO K620). Cette fluidité est amplement suffisante pour faire fonctionner l'outil et travailler avec celui-ci. Des améliorations sont bien évidemment réalisables sur l'outil. Par exemple, le code peut encore être optimisé pour être plus rapide. Il est aussi envisageable d'améliorer les performances des deux réseaux de neurones bien que ces derniers soient déjà suffisamment précis.

Un prolongement de ce travail pourrait s'axer sur plusieurs points. L'idée qu'Aubay aimerait suivre est l'ajout d'une seconde caméra au projet afin appréhender la notion de profondeur. Dans la continuité du projet, il faudrait aussi adapter l'outil pour que celui-ci fonctionne sur d'autres supports tels qu'une télévision ou un téléphone. Enfin, il pourrait être intéressant de remplacer le premier réseau ainsi qu'une partie de l'étape de traitement d'image par un réseau effectuant de la segmentation sémantique. Cela demande, en revanche, beaucoup plus de puissance et nécessiterai un GPU plus performant pour fonctionner.

Ce stage de fin d'études a représenté pour moi l'opportunité de suivre un projet innovant jusqu'à sa présentation aussi bien en interne pour la journée des stagiaires que pour d'éventuels clients lors d'une présentation et démonstration pour le crédit lyonnais (LCL). Je remercie encore Aubay pour la confiance apportée tout au long du projet. Aubay m'offre aussi la possibilité de continuer à travailler avec eux et je leur en suis particulièrement reconnaissant.

Table des figures

2.1	Détection du nombre de doigts avec la version 2017	13
2.2	Exemple d'utilisation d'OpenPose tiré de [4]	13
2.3	Version d'OpenPose ne détectant que les mains.	14
2.4	Exemple d'utilisation de Yolo avec 9000 classes tiré de [13]	16
3.1	Les différents fichiers de code qui composent notre projet.	17
3.2	Les performances de Yolo par rapport à d'autres modèles classiques.	18
4.1	Logos de Darknet et You Only Look Once.	20
4.2	L'architecture du modèle Yolo.	21
4.3	Première couche du réseau	21
4.4	Simplification de l'explication d'une convolution.	22
4.5	Deuxième couche du réseau Yolo.	22
4.6	Détection puis filtrage de box.	24
4.7	Fonction de coût de Yolo tirée de [11]	25
4.8	Exemple de surapprentissage	26
4.9	Courbe précision rappel	27
4.10	Courbe précision rappel pour les classes tête et main	28
4.11	Courbe de <i>loss</i> en fonction du temps et évolution liée du mAP	29
4.12	Exemple de détection par le modèle sur le jeu de test	29
5.1	Passage dans l'espace de couleur LAB	30
5.2	Binarisation de la main et filtrage	31
5.3	Contour et contour de Hull d'une main	31
5.4	Points caractéristiques de la main	32
5.5	Différentes images de squelette de la main	33
5.6	Les dix gestes compris par notre programme	33
5.7	LAB avant et après soustraction du fond	34

6.1	Aperçu de la translation en image des cinq mouvements détectables ainsi que d'un mouvement parasite.	35
6.2	Structure du modèle pour détecter les mouvements.	37
6.3	Évolution de la fonction de coût et de la précision au cours de l'apprentissage pour les jeux d'entraînement et de validation.	38
8.1	Aperçu de la première version de l'interface de tests.	42
8.2	Aperçu de la deuxième version de l'interface de tests.	42
8.3	Aperçu de la première version de l'interface utilisateur.	43
8.4	Aperçu de la deuxième version de l'interface utilisateur.	43
B.1	Centre de main	48
B.2	Binarisation sur fond difficile	48
C.1	Comparaison du modèle avec quatre ou cinq couches (courbes lissées)	49

Annexe A

Site web du projet et vidéos

Dans le cadre de mon master de recherche avec l'université d'Angers, j'ai été amené à créer un site web disponible **ici**¹.

Ce site présente rapidement le projet et inclut deux vidéos tournées durant le projet. La première est une présentation du projet effectuée pour la communication d'Aubay et la seconde est une vidéo d'exemple d'utilisation de la version de démonstration du projet.

1. pour la version imprimée : <https://arslanan.github.io/DWYH/>

Annexe B

Exemples supplémentaires partie traitement d'image

Voici trois exemples du centre de la main et de la détection de main ouverte (bleu) ou fermée (vert)

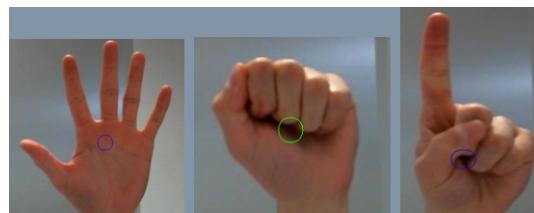


FIGURE B.1 – Centre de main

Voici deux exemples de binarisation sur fond difficile. La première représente un défi du fait du fond possédant des couleurs similaires à la main. La seconde représente un défi pour le changement de fond sur le bout du doigt.

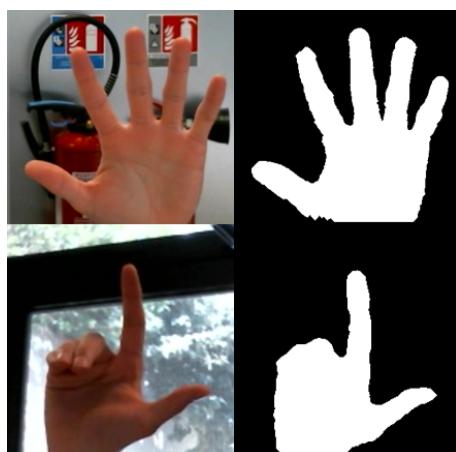


FIGURE B.2 – Binarisation sur fond difficile

Annexe C

Information sur le modèle détectant les mouvements

Sur cette image on peut voir à gauche l'évolution de la fonction de coût au fil de l'entraînement pour les modèles de 4 et 5 couches sur le jeu de validation. On peut voir que la fonction *loss* pour le modèle à 5 couches diminue plus qu'avec 4 couches au fil de l'entraînement ce qui mène à une précision plus élevée (image de droite).

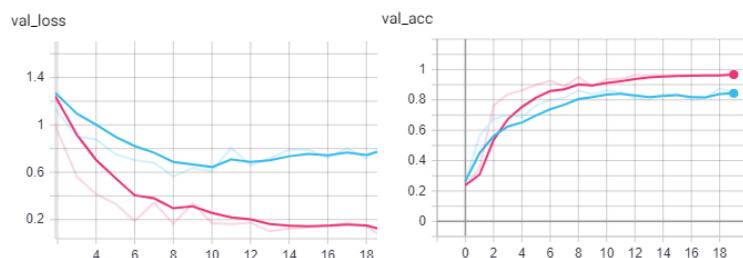


FIGURE C.1 – Comparaison du modèle avec quatre ou cinq couches (courbes lissées)

Liste des sigles et acronymes

DWYH	<i>Drive With Your Hands</i>
Yolo	<i>You Only Look Once</i>
mAP	<i>Mean Average Precision</i>
IoU	<i>Intersection over Union</i>
CNN	<i>Convolutional Neural Network, acronyme anglais pour réseau de neurones convolutif</i>
ESN	<i>Entreprise des Services du Numérique</i>
BU	<i>Business Unit</i>
IATR	<i>Infrastructure Assurance Télécom Réseaux</i>
fps	<i>Frame per second (image par secondes)</i>
NLP	<i>Natural Language Processing</i>
GMM	<i>Gaussian Mixture Modelling</i>
RNN	<i>Recurrent Neural Network</i>
NMS	<i>Non Max Suppression</i>
TP	<i>True Positive</i>
FP	<i>False Positive</i>
LAB	<i>Luminosité A B, espace de couleur</i>

Bibliographie

- [1] Aubay, *Drive With Your Hand 2017 : Work report*, 2017.
- [2] M. Hasan and P. K Mishra, “Real time fingers and palm locating using dynamic circle templates,” *International Journal of Computer Applications*, vol. 41, 03 2012.
- [3] Aubay, *Drive With Your Hand 2018 : Work report*, 2018.
- [4] A. Solano, “Human pose estimation using openpose with tensorflow.” <https://arvrjourney.com/human-pose-estimation-using-openpose-with-tensorflow-part-1-7dd4ca5> 2017.
- [5] K. Sheth, N. Gadgil, and P. Futane, “A hybrid hand detection algorithm for human computer interaction using skin color and motion cues,” *International Journal of Computer Applications*, vol. 84, pp. 14–18, 12 2013.
- [6] H. S. Hasan and S. A. Kareem, “Human computer interaction for vision based hand gesture recognition : A survey,” in *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pp. 55–60, Nov 2012.
- [7] M. V. S. Rathore, M. M. S. Kumar, and M. A. Verma, “Colour based image segmentation using $l * a * b$ * colour space based on genetic algorithm,” in *International Journal of Emerging Technology and Advanced Engineering*, 2012.
- [8] P. Xu, “A real-time hand gesture recognition and human-computer interaction system,” *CoRR*, vol. abs/1704.07296, 2017.
- [9] N. McLaughlin, J. Martinez del Rincon, and P. Miller, “Recurrent convolutional network for video-based person re-identification,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [10] T.-N. Nguyen, H.-H. Huynh, and J. Meunier, “Static hand gesture recognition using artificial neural network,” *Journal of Image and Graphics*, vol. 1, pp. 34–38, 01 2013.
- [11] J. Redmon, “Darknet : Open source neural networks in c.” <http://pjreddie.com/darknet/>, 2013–2016.
- [12] J. Redmon and A. Farhadi, “Yolov3 : An incremental improvement,” *CoRR*, vol. abs/1804.02767, 2018.
- [13] J. Redmon and A. Farhadi, “YOLO9000 : better, faster, stronger,” *CoRR*, vol. abs/1612.08242, 2016.
- [14] P. Zhou, Q. Ding, H. Luo, and X. Hou, “Violent interaction detection in video based on deep learning,” *Journal of Physics : Conference Series*, vol. 844, p. 012044, jun 2017.
- [15] M. A. Rahman and Y. Wang, “Optimizing intersection-over-union in deep neural networks for image segmentation,” in *Advances in Visual Computing* (G. Bebis, R. Boyle, B. Parvin,

- D. Koracin, F. Porikli, S. Skaff, A. Entezari, J. Min, D. Iwai, A. Sadagic, C. Scheidegger, and T. Isenberg, eds.), (Cham), pp. 234–244, Springer International Publishing, 2016.
- [16] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *Int. J. Comput. Vision*, vol. 88, pp. 303–338, June 2010.
 - [17] Cartucho, “Openlabeling.” <https://github.com/Cartucho/OpenLabeling>, 2016.
 - [18] S. Bambach, S. Lee, D. J. Crandall, and C. Yu, “Lending a hand : Detecting hands and recognizing activities in complex egocentric interactions,” in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.
 - [19] A. Mittal, A. Zisserman, and P. H. S. Torr, “Hand detection using multiple proposals,” in *British Machine Vision Conference*, 2011.
 - [20] A. U. Khan and A. Borji, “Analysis of hand segmentation in the wild,” *CoRR*, vol. abs/1803.03317, 2018.