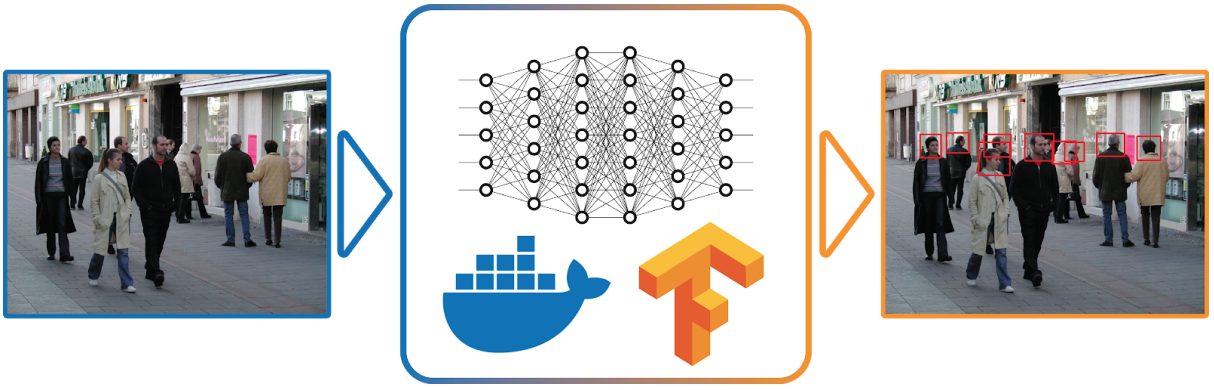
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Cliris

Stage de 4ème année

La détection de tête en temps réel dans les scènes localement dense à l'aide des dernières techniques de Deep Learning.

RONTEIX--JACQUET Flavien



Enseignant référent
HAFIANE Adel

Tuteur Entreprise
HADJERCI Oussama

Année universitaire **2017/2018**

Remerciements

Je tiens à remercier tout d'abord Oussama HADJERCI pour son encadrement et ses conseils. Sans lui, le travail et le rapport n'aurait pas été du tout les mêmes.
Je remercie également Adel HAFIANE pour m'avoir aidé à trouver ce stage dans le domaine que je voulais, l'Intelligence Artificielle.
De plus je tiens à remercier tous les collègues de Cliris, Benjamin, Benjamin, Stéphanie, Jennifer, Safa, Bruno, Marie, Muriel, Rojo, Alek, Alexandre, Patrice, Alain et plus particulièrement Antoine pour leur accueil et leur sympathie.
Un grand merci à Hugo, Corentin, Alexandre, Baptiste et Madeleine du foyer de Suresnes pour leurs soutiens et leur bonne humeur tous les soirs.
Enfin, j'adresse un grand merci à mon entourage pour tout ce qu'ils ont pu faire.

RESUME

Ce stage s'est déroulé au sein du pôle Recherche et Développement de Cliris group, entreprise experte dans la mesure de fréquentation en intérieur et en extérieur en se basant sur des méthodes de détection de mouvement et de machine learning. Cependant ces méthodes sont peu efficaces pour l'analyse de scène complexe et pour s'adapter à de nouvelles scènes. D'où le sujet du stage, la détection de tête en temps réel dans les scènes localement denses à l'aide des dernières techniques de Deep Learning.

Ce stage a été réalisé autour de deux axes principaux. D'une part une recherche bibliographique et des technologies a été mené. D'autre part, la mise en place d'une chaîne de traitement a été proposée. Cette chaîne de traitement consiste à créer un dataset de tête à partir de plusieurs datasets existant; la configuration d'un modèle et son entraînement et enfin utiliser le résultat de cet entraînement pour la prédiction.

Au bout de quatre mois, une base de connaissance a été constitué sur le Deep Learning. A partir d'une étude comparative des différents framework et des modèles existants, le framework Tensorflow et le modèle SSD/mobileNet ont été sélectionnés. Alors, un POC de détection à base de Deep Learning a été réalisé.

Les résultats de ce POC sont intéressants et ouvrent des perspectives pour Cliris mais le travail initié doit être poursuivi pour aboutir à un système de détection plus précis et plus rapide.

Mots-clés : *Intelligence Artificielle, Réseau de Neurone Profond, Vision par Ordinateur, Docker, TensorFlow, Détection d'objet,*

ABSTRACT

This internship took place within research and development team of Cliris, an expert company in the measurement of indoor and outdoor attendance based on motion detection and machine learning methods. However, these methods are less performant to analyse a complex scenes and deficient to new data. Hence the subject of the internship is real-time head detection in complex scenes using the latest Deep Learning techniques.

This internship was realized around two axes. On the one hand, bibliographic and technologic survey was conducted. On the other hand, a new workflow was proposed to the company. This workflow consists of creating a leading dataset from merging several existing datasets, tuning a model and training it, and finally using the result of this training for the prediction.

After four months, a knowledge base has been created on Deep Learning. From a comparative study of the various frameworks and models the Tensorflow framework and the SSD / mobileNet model have been selected. Then, a POC of automatical detection based on Deep Learning has been realized.

The results of this POC are interesting. However, to integrate this system into the Cliris Solution the effort must be continued to obtain a more precise and faster detection system.

Keywords : *Artificial Intelligence, Deep Learning, Computer Vision, Docker, TensorFlow, Object Detection*

Sommaire

Introduction

Une des problématiques majeure dans le domaine de la vision par ordinateur est l'automatisation des systèmes de reconnaissance et de détection. En effet, ce domaine ne cesse de se développer pour assister l'être humain dans ces tâches quotidiennes. Particulièrement depuis quelques années avec la poussée d'une nouvelle méthode d'intelligence artificielle bien connue; le **Deep Learning**.

Ce travail de stage a été effectué au sein de l'entreprise **Cliris.** Le domaine d'expertise de celle-ci est le traitement d'images et de vidéos à l'aide d'un algorithmes de détection par mouvement et plus récemment par Machine Learning.
Les techniques de machine learning ont démontré leurs puissances dans différentes thématiques, tout particulièrement pour la détection des piétons. Cependant, leurs principales difficultés reposent sur les ambiguïtés induites par la complexité de l'environnement (configuration de la caméra) et les interactions entre individus : occultations partielles, proximité et croisements de trajectoires.
D'où la volonté de Cliris d'améliorer continuellement sa solution, en se basant sur des nouvelles technologies comme le Deep Learning ou l'apprentissage profond.

A travers la mesure de flux, Cliris permet aux réseaux de magasins d'identifier le potentiel de leurs points de vente en aidant notamment à leur pilotage : adaptation des plannings des forces de vente, analyse de l'attractivité des magasins (entrées/passants), de l'impact des vitrines, valorisation des emplacements et étude de l'impact des environnements des boutiques et des actions marketing. Ces mesures sont principalement basées sur la détection et la reconnaissance des piétons et d'autres objets. Cliris montre un intérêt particulier à la détection des têtes dans un contexte de foule afin d'éviter la problématique d'occultation.

D'où le titre de stage :
la détection de tête en temps réel dans les scènes localement dense à l'aide des dernières techniques de Deep Learning.

Le travail de ce stage est basé sur deux parties; une étude bibliographique et technique du Deep Learning et une proposition d'un système complet de détection de personne.
Tout d'abord, un travail sur le Deep Learning avec de la documentation, de la recherche bibliographique et de l'étude des technologies.
Ensuite, un second travail sur la **réalisation d'un "POC[1]"** a été proposé. Pour la réalisation de ce POC il y eu la mise en place d'un workflow[2] qui permette de collecter des données, lancer l'étape d'apprentissage et finalement détecter les personnes en se basant sur une étape de prédiction.

Dans ce document, nous commencerons par présenter l'entreprise (voir chapitre I), ensuite une étude bibliographique est illustrée dans le Chapitre II & III accompagnée par une étude comparative de différentes technologies afin de prouver notre choix méthodologique. Puis le système proposé est montré dans le Chapitre IV.
Pour terminer, nous conclurons sur les perspectives qu'a ouvert ce stage et sur mon retour d'expérience.

[1] Proof of Concept, Demonstration de faisabilité
[2] Flux de travail

I. Présentation de l'entreprise

A. Présentation de l'organisation

1. Historique

Cliris a été fondé en 2006 à Courbevoie par M. Zeller. La société a toujours eu pour objectif de proposer des solutions d'analyse de fréquentation à ses clients, clients qui sont dans la majorité des cas des boutiques, des centre commerciaux ou encore des franchises. Elle est certifiée ISO9001 ce qui veut dire qu'elle vise un management de qualité dans le but de satisfaire au mieux ses clients.

2. Situation géographique

Cliris se situe à deux pas de la Défense, à Courbevoie, 6 rue de Bitches. Etant une petite structure, elle occupe un étage dans un immeuble sous forme d'un open space. Cette position au coeur d'un centre d'affaires aussi important est un atout pour la recherche de nouveau client. De plus, elle ne se limite pas à la France et propose sa solution à des clients européens comme en Angleterre ou bien au Portugal.

3. Chiffres clés

Cliris est une Société à action simplifié au capital de 136.695€ qui génère un Chiffre d'affaires de 836.000€ par an (en 2017). Le Chiffre d'affaires est en constante augmentation.
Elle a 20 clients important et répartis dans toute la France comme Alain Afflelou et possède même comme client le centre commercial Avaricum à Bourges.
Il y a 13 salariés et 6 étudiants en stage, alternance ou bien en doctorat.
L'entreprise a une politique de recherche développement poussée, notamment pour de nouvelles solutions avec les dernières innovations comme le big data ou bien l'intelligence artificielle. Elle participe également à des travaux de recherche en partenariat avec le monde académique comme avec les projets AVATAR, ORIGAMI, FUI20 ETS ou TMDSA (Fouille de données de trajectoires client pour applications d'achat dans des centres commerciaux)

B. Cadre du travail

Les effectifs sont réparties en 4 pôles, une équipe de direction, une équipe de marketing, une équipe de maintenance/support, et une équipe de recherche/développement.

Il y a notamment 4 ingénieurs dont 2 qui font également de la recherche et développement. Mon maître de stage, Oussama Hadjerci s'occupe de la recherche en vision par ordinateur et plus particulièrement en Machine Learning.
Mon apport au sein de l'équipe est la connaissance des technologies de Deep Learning et une formation généraliste en informatique, ce qui m'a permis de m'adapter.

II. Etat de l'art

A. Introduction au Machine Learning et Deep Learning

Depuis quelques années, on entend beaucoup parler de Machine Learning, de Deep Learning et d'Intelligence Artificielle. En effet, depuis la dernière décennie, les puissances de calculs ont explosé notamment grâce à l'essor des cartes graphiques. De plus la capacité de stockage des données ne cesse de s'augmenter notamment due aux nouvelles technologies de big data et le tout dans un contexte de concurrence très forte entre les grandes entreprises pour faire de l'analyse des données et l'analyse prédictive. Ces facteurs ont contribué à l'évolution rapide des méthodes de Machine et Deep Learning.

L'**intelligence artificielle** est un domaine très large qui existe depuis les tous débuts de l'informatique, puisque Alan Turing (le père de l'informatique) écrivait déjà son fameux article *"computing machinery and intelligence"* où il propose le "test de Turing" en 1950 alors que les ordinateurs n'étaient guère plus puissant que nos calculatrices de poche.
L'intelligence artificielle est une notion assez vague, notamment si l'on se réfère à une définition commune du dictionnaire, *"ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence"*. Je pense pouvoir résumer en "l'ensemble des techniques visant à résoudre des problèmes complexes que l'on croyait réservé à une intelligence humaine".
Ce domaine est donc très large et recouvre beaucoup de discipline, de l'Informatique aux Mathématiques en passant par la Philosophie.
Dans ce domaine qui est l'Intelligence Artificielle, il existe des sous-domaines assez indépendant qui visent chacun à résoudre certaines tâches, on peut prendre l'exemple du calcul formel ou bien de la synthèse expert. Pourtant, depuis quelques années, il s'agit de l'apprentissage automatique avec le Machine Learning qui semble être la piste la plus prometteuse pour la résolution de nombreux problèmes.

Le **Machine Learning** est apparu dans les années 50 et a connu un véritable essor depuis l'an 2000 notamment dans les problèmes de prise de décision, d'analyse des données et de la vision par ordinateur. Le Machine Learning est l'ensemble des techniques qui permettent à la machine "d'apprendre" à résoudre un problème.
C'est une nouvelle façon de concevoir le programme informatique puisque précédemment, le programmeur écrivait un programme puis met des données d'entrée pour obtenir des résultats en sortie. Or avec le Machine Learning, on fournit des données d'entrée, des sortie attendues et l'entraînement nous retourne un programme que l'on appelle **modèle**.
Cette apprentissage peut permettre par exemple la classification des données ou bien le clustering[3] automatique des données. Le Machine Learning a trouvé un écho dans la **vision par ordinateur** qui est une tâche complexe pour un ordinateur. Par vision on entend l'analyse, le traitement et la compréhension de l'image un peu comme le fait un être humain avec ces yeux et son cortex visuel.
Ainsi l'apprentissage automatique a mis son pied dans la vision et ça ne s'en est pas arrêté là.

Le **Deep Learning** ou apprentissage profond est apparu dans la communauté informatique depuis les années 60, mais il a explosé que très récemment. En effet son principe est simple, faire du bio-mimétisme du cerveau humain pour permettre au machine d'apprendre. Avec le perceptron et la simulation de neurone humain.
Le problème de l'apprentissage profond à l'époque était triple. D'abord, il faut une **puissance de calcul** titanesque puisque chaque neurone artificiel demande beaucoup de calcul et le cerveau humain compte 10^{10} Neurones pour 10^{15} connexions. D'autre part l'apprentissage nécessite des milliers voire des **millions d'exemple** et à l'heure où internet n'existe pas encore et le stockage était coûteux, il était impossible de faire apprendre ces systèmes. Enfin, le dernier problème est d'ordre **mathématique** puisqu'il faut concevoir des méthodes, des algorithmes pour mener ces entraînements. Finalement, il a fallu donc beaucoup de recherche, assez fondamentale, pour permettre l'apprentissage automatique en Deep Learning.
Mais depuis 2013 avec l'essor du Big Data[4], l'essor des supercalculateurs distribués à base de GPU[5] et de bases théoriques solide, le Deep Learning est devenu une réalité et a permis des améliorations

[3] Partitionnement des données
[4] Base de donnée gigantesque générée par tous
[5] Graphical Processor Unit, très efficace pour le calcul matriciel

nettes des performances dans le domaine de la vision par ordinateur (voir en annexe Z les résultats de détection d'objet sur des images issu du concours annuel **ILSVRC**[6]).

Cependant, afin de maîtriser ces technologies récentes, il faut comprendre leurs fondamentaux.
Dans le détail, comment fonctionne l'apprentissage profond ?

L'apprentissage profond possède une architecture basée sur des couches de neurone (au sens informatique du terme, Fig 1) qui sont empilées les unes sur les autres (voir Fig 2) (en Annexe A une infographie qui résume les différents types d'architecture).
On entraîne ce réseau en jouant sur les poids (W_{ji}) pour réduire l'erreur globale du réseau. L'entraînement n'est finalement que la recherche du point minimum dans un espace à plusieurs millions de dimension (une dimension par poid).

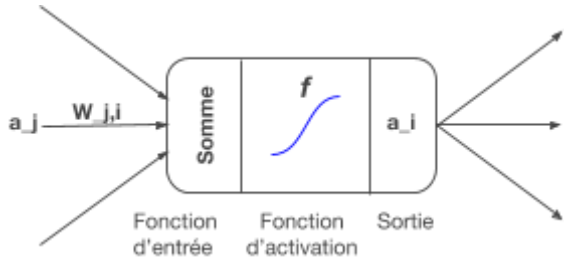


Fig 1 - Schéma d'un neurone artificiel

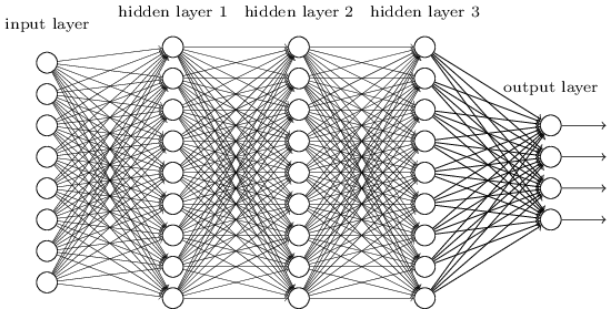


Fig 2 - Réseau de neurone

Dans le cas de la vision par ordinateur à l'aide de deep learning, il s'agit de **couche de convolution** dont on parle qui sont empilées les unes sur les autres. Les réseaux neuronaux convolutifs sont une extension du produit de convolution "classique" appliqué à un tenseur[7] de dimension 3. Le fonctionnement de la convolution "classique" est assez simple :

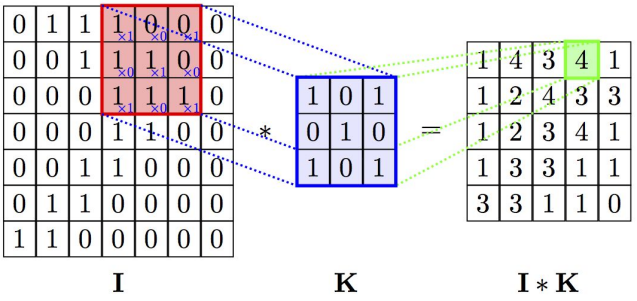


Fig 3 - Principe de la convolution classique

Il existe bien d'autres type de convolution, mais nous y reviendrons plus tard dans le rapport.
Les poids W_{ji} que l'on entraîne ici ne sont pas sur des synapses mais bien dans les noyaux de convolution ***K***, c'est eux qui sont appris au cours de l'entraînement.
L'on observe que les premières couches de convolution servent au réseau à détecter les coins, les bords, puis plus on va profondément dans le réseau et plus il détecte des motifs complexes comme des objets. Il suffit de mettre une couche de neurone "fully-connected[8]" en sortie pour trouver l'information détecter par le réseau.
Il existe des dizaines d'autres type de couche mais nous y reviendrons également plus tard lorsque nous parlerons du modèle utilisé.

Dans ce rapport nous allons utiliser beaucoup de vocabulaire assez technique du domaine et surtout un vocabulaire qui est en majorité en anglais et non-standardisé.
Ainsi en fonction des travaux de recherche, le mot "modèle" par exemple, peut recouvrir différents aspects, ce qui contribue à la confusion que l'on peut avoir à la compréhension du Deep Learning.
Nous parlions de **modèle**, ce que je vais sous-entendre par modèle sera l'architecture du réseau neuronal. Ce modèle possède des couches avec certains paramètres et agencées d'une certaines manières.

[6] Imagenet Large Scale Visual Recognition Challenge, http://www.image-net.org/challenges/LSVRC/
[7] Extension des matrices à des dimensions supérieurs
[8] Entièrement connecté, les neurones sont connectés à tous les neurones de la couche précédente

Ces modèles utilisent des types de convolutions, sont entraînés avec des algorithmes d'apprentissage, etc... Tous ceux-ci est la "base théorique", ce que nous allons appeler **algorithme de Deep Learning**.
Enfin, pour implémenter ces modèles, il faut les programmer et pour ça, nous utilisons des **Frameworks** qui sont de grosse boîte à outil informatique qui vont proposer des implémentations d'algorithmes. L'on peut résumer le rapport entre ces 3 mots par le schéma ci-dessous.

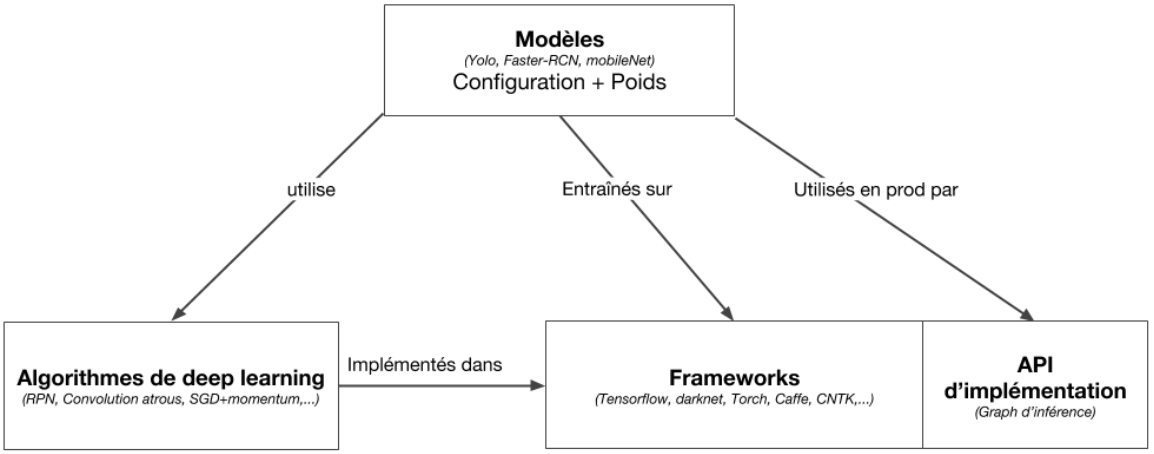


Fig 4 *- Lien entre les différents mots de vocabulaire*

Deux principales facteurs peuvent représenter la performance d'un algorithme de Deep Learning; la précision de détection et/ou reconnaissance ainsi que le temps de prédiction. Ces deux facteurs combinés sont un moteur de recherche de la communauté scientifique, d'où la multitude des publications sur ces problématiques.

B. Evolution des méthodes de détection

Le désir d'améliorer le taux de détection de personnes est présent dans plusieurs applications, comme le dénombrement, la voiture autonome, la surveillance, la robotique, et l'interaction homme machine.
Pour cette raison, des bases de données ont été bien définies, par exemple, *INRIA pedestrian* [30], *Caltech* [31]. Ces bases de données ont été utilisées pour explorer différentes méthodes pour la détection des personnes [1,2,3,4].

Pendant la dernière décennie, plusieurs travaux de recherche inspirés de la méthode de *Viola and Jones* [32,33] ont améliorer dramatiquement le taux de détection de personnes dans une scène de face. Cependant, les techniques basées machine learning sont moins performante dans des context réel et plus général comme la vue oblique ou/et zénithale avec des variations importante (luminance, occultation, etc..).

Toutefois, ces techniques ne cessent d'évoluer. Par exemple 40 techniques groupées en 3 grande familles ont été étudiées dans [1]; es méthodes Forêt de Décision (**DF**) , les méthodes Détecteur à Partie Déformable **(DPM**) et récemment les réseau neuronal convolutifs (**CNN)**.

L'objectif de l'article [1] est de comprendre et de comparer les 40 méthodes afin de sélectionner les meilleurs algorithmes de détection de personnes sous divers variations (capteurs, agrégation de données, luminance, multi échelle). Les méthodes étudiées dans cette article sont appliquées sur les bases de données *Caltech* et *INRIA pedestrian*.

INRIA pedestrian est un ensemble d'image annotée de personne proposée par l'institut français INRIA[9] dans le cadre d'une thèse sur la détection des personnes. *Caltech pedestrian* est proposé par l'université américaine Caltech[10] et contient 10 heures de vidéo à 30 fps de piéton annotés capturée depuis un véhicule.

[9] Institut National de Recherche en Informatique et en Automatique
[10] California Institute of Technology

Le résultat de l'étude [1,...] est présenté dans le Tab. 23 (résultats complets issus de l'article en annexe B) . Dans ce tableau nous illustrons le taux d'erreur de détection (MR) des méthodes les plus connues pour cette problématique. L'erreur de détection est de 94.73% pour la méthode VJ [33] en 2001 et elle est passée à 34.60% en 2014. Dans ce travail nous remarquons aussi que la fusion de différentes caractéristiques (Katamari-v1 : 22.49 %) améliore nettement le résultat. Cependant pendant les deux dernières années les algorithmes de deep learning ont montré leur performance par rapport aux techniques de machine learning. Dans la Tab. 23, deux techniques de deep learning; DeepParts[22] et F-DNN2 [23] atteignent respectivement, 11.89 % et 8.12 % de MR.

Méthode	Miss Rate	Famille	Type de caractéristiques	année
Viola et Jones	94.73%	DF	Haar	2001
ConvNet	77.20%	DN	Pixels	2013
HOG	68.46%	-	HOG	2005
FPDW	57.40%	DF	HOG+LUV	2010
ACF	51.36%	DF	HOG+LUV	2014
ACF-Caltech	44.22%	DF	HOG+LUV	2014
Joint Deep	39.32%	DN	Color + Gradient	2013
Informed Haar	34.60%	DF	HOG+LUV	2014
Katamari-v1	22.49%	DF	HOG+Flow	2015
DeepParts	11.89%	DN	Pixels	2017
F-DNN2	8.12%	DN	Pixels	2018

Table 1 - Liste des méthodes de détection de personnes étudiées

D'après notre étude bibliographique, nous avons remarqué que les techniques les plus performantes sont basées sur la fusion des caractéristiques afin de représenter au mieux les contours, les textures, les formes locales. En effet, deux types de méthodes existent pour obtenir ce mélange de caractéristiques; la fusion des descripteurs en se basant sur du **machine learning** et les méthodes de **deep learning.**

L'objectif de cette section est de définir les techniques de machine learning et de deep learning les plus performantes, afin de proposer notre chaîne de traitement adapté au context de l'entreprise Cliris.

C. Machine Learning

Comme nous l'avons vu précédemment, il existe différentes familles de méthode de classification et d'extraction de caractéristiques, en effet, la chaîne de traitement des solutions de machine learning se base sur deux grandes étapes; l'extraction et sélection des caractéristiques et la classification (voir Fig. 5)

Fig. 5 - Architecture de machine learning classique

L'étape d'extraction de caractéristiques consiste à calculer des informations représentatif de bords, contours et motifs de texture. Les méthodes les plus utilisées pour le domaine de détection de personnes sont les suivantes :

- **Haar** [15] : ces méthodes sont présentées pour la première fois dans l'article de Viola et Jones;.. Son principe repose sur le calcul pour toute l'image des caractéristiques avec des zones rectangulaires de pixel noir ou blanc appelés "caractéristiques".
- **Gradients** : le gradient est calculé pour chaque pixel, afin de caractériser les contours dans une image.
- **Local binary pattern (LBP**) [16] : un codage binaire est utilisé pour caractériser les motifs répétitifs dans une image.
- **Aggregate Channel Feature** (**ACF**) [17] : Ce descripteur est basé sur différentes caractéristiques comme les canaux LUV, Gradient, etc,...
- **Optical Flow** [18] : Utilisation du tracking de certains points de l'image comme caractéristique
- **Curvature Scale Space** (**CSS)** [19] : Utilisation des courbes de l'image pour en extraire des caractéristiques.
- **Histogram Oriented gradient (HOG)** [21]: Les gradients sont calculés pour chaque pixel dans l'image, puis ils sont regroupés selon l'orientation des angles. Ces groupes d'orientations de gradients sont représentés par des histogrammes. Ces méthodes sont invariantes au changement de luminance ou d'ombre.

Une fois les caractéristiques calculées, des techniques de classification sont appliquées sur ces caractéristiques non seulement pour former un modèle représentatif des piétons mais aussi pour reconnaître (prédire) une personne. Deux classifieurs supervisés sont souvent appliqués pour détecter les personnes; **Support Vector Machine** (**SVM**) [42] et **Adaptive Boosting** (**AdaBoost**) [43]. Ces classifieurs sont utilisés pour projeter les caractéristiques dans un espace plus développé afin de trouver la bonne séparation entre les différentes classes. Cette séparation est représentée par un modèle d'hyperplan.

Les chaînes de traitement à base de machine learning sont très performantes. Cependant, leurs majeurs inconvénient est le manque de généralisation du modèle d'apprentissage. En effet, les techniques de classification fonctionnent dans un contexte bien précis. Cependant, les solutions à base de Deep Learning sont capables de produire un modèle plus généralisé, capable de s'adapter à divers scénarios [44].

D. Deep Learning

A la différence des méthodes classiques de Machine Learning, le Deep Learning cherche à apprendre de bout en bout, de l'extraction de caractéristique à la classification. Il n'y a pas de différence d'algorithme entre l'extraction et la classification.

Fig 6 - Architecture du Deep Learning

L'objectif de cette section est de présenter l'avantage et l'inconvénient de solution de Deep Learning existante, afin de sélectionner les techniques adaptées à notre problématique.

Le travail de HUANG et al. [3], *"Speed/Accuracy trade-offs for modern convolutional object detectors"*, mène une étude comparative de différent modèles de réseaux neuronaux convolutifs pour la détection des objets a été proposée.

Ils ont trouvés, comme illustré en Fig. 7, une architecture assez commune dans ces modèles, un méta modèle décomposé en trois modules; l'extraction de caractéristiques; la proposition de région; classification des régions. Dans le cadre de ce stage, nous nous sommes appuyés sur cette taxonomie qui a le mérite d'être simple et commune à beaucoup de modèle.

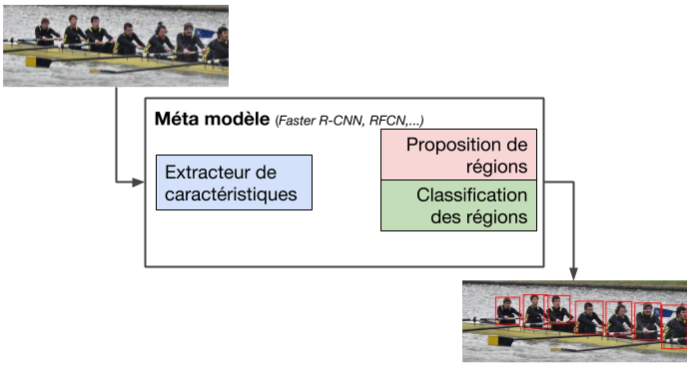


Fig. 7 - Composants d'un méta modèle

Dans le travail de [3], une dizaines de combinaisons méta-modèle/extraction de caractéristiques sont étudiées; 3 familles principales de méta-modèle et 6 modules d'extractions de caractéristiques. L'étude expérimentale a été mené sur les mêmes configuration de machine, de base de donnée et de bibliothèque (Tensorflow), afin d'obtenir des résultats comparables.

Les méta-modèle étudiés sont : Faster Region-Convolutional Neural Network (R-CNN), Region-based Fully Convolutional Network (R-FCN), Single Shot Detector (SSD). Ils remarquent également You Only Look Once (YOLO) mais étant trop particulier, nous l'étudierons dans la prochaine partie du rapport.
Les modules d'extractions de caractéristique appliqués sont : Inception Resnet V2 [11], Inception V2 [12], Inception V3 [12], MobileNet [7], Resnet 101 [13] et VGG-16 [14]. Les schémas représentant l'architecture de chacun de ces réseaux sont présentés en annexe G.

Méta-modèle:

- **SSD** [8] : La détection de chaque classe est obtenue directement à partir du module extraction de caractéristiques (voir fig. 8).

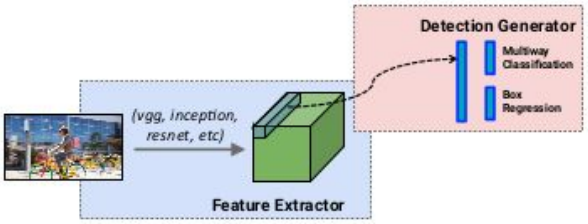


Fig. 8 - Fonctionnement de SSD

- **Faster R-CNN** [9] : La performance de ce méta modèle est dûe au rajout de deux module par rapport au SSD; une étape de proposition de régions et une autre de classification (voir fig. 9).

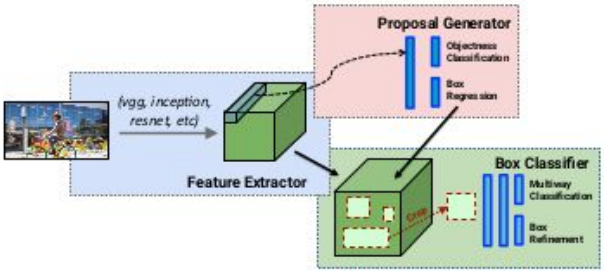


Fig. 9 - Fonctionnement de Faster R-CNN

- **R-FCN** [10] : Même principe que le R-CNN. Cependant, les caractéristiques sont modifiées par des couches supplémentaires entre les modules de proposition de régions et de classification, tout en supprimant les couches pour la classification Cette architecture réduit le nombre de calcul et la précision de détection (voir fig. 10).

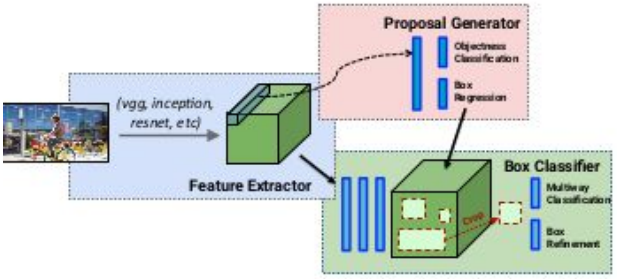


Fig 10 - Fonctionnement de R-FCN

Une étude expérimentale poussée sur les combinaisons méta-modèle et extraction de caractéristiques a été menée dans [3], afin de mesurer le coût en mémoire, la corrélation entre précision du module extraction de caractéristiques et la précision du modèle global, l'optimisation du CPU, le coût en calcul par rapport au nombre de box proposées, etc. Le résultat le plus intéressant pour nous de cette étude est montrée dans la figure suivante :

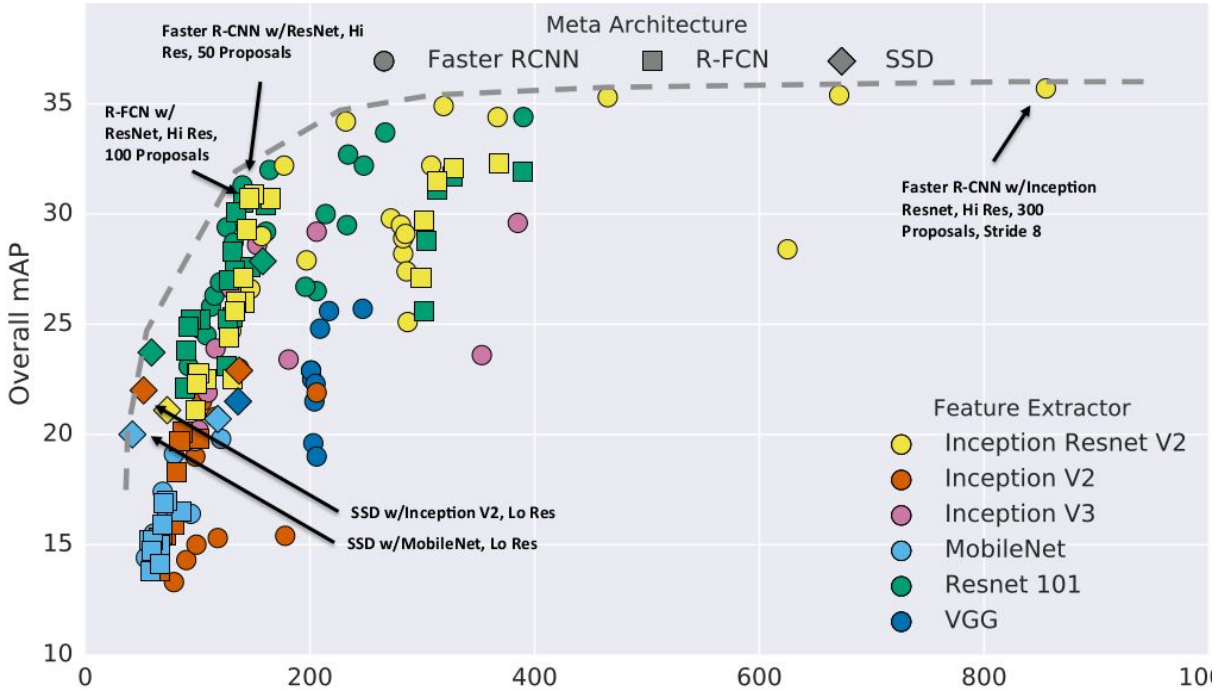


Fig 11 - Graphique représentant le rapport entre précision et vitesse de différents modèles

En Fig. 11, le rapport entre la précision du modèle et le temps de calcul nécessaire est illustré. Une courbe limite sur le "trade-off" entre précision et performance est remarquée. Plus le modèle est précis, plus il consomme des ressources en calcul. Cette expérimentation a permis non seulement de mettre en évidence les différentes vitesses et précision des détecteurs à base de CNN moderne, mais également de trouver de nouvelles méthodes pour améliorer les performances et la vitesse.
Le rapport entre la qualité et les ressources est la clé de notre problématique, car la détection précise et en temps réel est primordiale à la solution Cliris.
Suite à l'étude menée et à la problématique ci-dessus, nous pensons que le méta-modèle **SSD** utilisant comme module extraction de caractéristiques **Mobilenet** est probablement le meilleur choix.

D'autres travaux [2,4] ont essayé d'autres méthodes comme la combinaison de modèle de Deep Learning ou bien la combinaison de modèle et Deep Learning et de Machine Learning.

Dans [2], les auteurs ont étudié la détection des têtes dans un environnement stable (moins d'occultation et vue de face) avec une combinaison de modèle de Deep Learning.
Partant du constat que seul 30% des pixels en moyenne représentent une personne sur une photo, ils ont décidé d'utiliser le fond de la scène comme contexte pour améliorer la détection. De plus, il existe souvent une relation entre les mêmes classes d'objet, comme entre 2 têtes. D'où l'utilisation de 3 modèle de réseau convolutif en parallèle. Un local, un global et un pairwise[11]. En annexe C la visualisation des résultats des différents modèles sur un exemple.

[11] Par paire de relation

Le **modèle local** s'appuie sur un R-CNN à qui on donne une zone issu d'une proposition de recherche sélective (d'où le local) avec un peu de contexte. Le R-CNN choisi est issue du modèle AlexNet (architecture de ce modèle très connu en annexe D), pré-entraîné sur le dataset ImageNet.
Le **modèle global** est un CNN qui découpe l'image en plusieurs grilles à plusieurs échelles pour détecter et estimer la position et la taille des objets (Un exemple est illustré dans l'annexe E).
Le **modèle pairwise** cherche à trouver des relations entre les objets détectés pour affiner la détection. En annexe F, nous pouvons voir le fonctionnement de ce système de relation avec un exemple.

A la suite d'un entraînement assez classique du modèle sur le dataset *Hollywood Head* ils obtiennent des résultats très intéressant avec un modèle très proche de l'état de l'art étudié. Par contre le modèle est lent et à des difficultés de généralisation. De ce fait, cette technique n'est nullement adaptée à notre problématique.

Dans un autre travail de l'université technologique du Pakistan, de février 2018, *"People counting in dense crowd images"* [4], les auteurs ont étudié la problématique du dénombrement des foules à l'aide d'une combinaison de machine et de deep learning.
Dans un premier temps, les techniques de machine learning ont été appliqués pour la classification des zones de foule. Dans un second temps, le nombre de tête a été estimé dans chaque zone détectée comme foule. En annexe H le schéma de l'algorithme proposé.
Cette solution n'améliore pas les performances tout en étant plus lente par rapport à des solutions de détection en deep learning classique que l'on a vu avec les articles précédents.
Pour autant, ce n'est pas une raison pour totalement écarter les **modèles hybrides** restés fermé aux alternatives.

Dans cette section, une étude bibliographique a été menée afin de tracer une directive de travail. En effet, à la suite de cette étude, nous avons avons sélectionné le méta-modèle **SSD** et le module extraction de caractéristique **MobileNet**. Toutefois, une étude plus approfondie de ces modèles et des frameworks est essentielle afin de proposer un système de détection complet et adapté à notre problématique.

III. Etude comparative des frameworks et des modèles

Dans cette section, nous allons étudier en détail le modèle YOLO (mis de côté dans l'article de comparaison) et la combinaison SSD/MobileNet dans un premier temps. Dans un second temps, nous comparerons divers frameworks. En effet, ces framework sont primordiaux pour l'implémentation (programmation) des modèles.

A. Modèles

Nous avons vu qu'il existe de nombreux modèles pour la détection des objets. La détection des têtes n'est qu'un cas particulier de détection mais avec quelques difficultés.
La première est la **densité** de tête dans une scène. La seconde est **l'échelle** (taille) des têtes dans l'image.
Ensuite, plus particulièrement pour Cliris, il y a les problématiques d'**hétérogénéité** des caméras avec des angles, des qualités et des hauteurs très différentes d'où la nécessité d'une bonne **généralisation** du modèle, et de **vitesse** de détection.
Notre étude va chercher à comprendre le fonctionnement des modèles et comment ils peuvent répondre à ces difficultés.

Comme illustré dans la Sec.II.D "Deep learning", nous avons choisi d'étudier la combinaison SSD/mobilenet. Cependant YOLO a montré récemment de très bonnes performances. De ce fait YOLO est inclu dans cette étude.

1. YOLO9000

La famille de modèle YOLO a été proposé par **Joseph Redmon et al.** dans [5] en 2015. Cette famille de modèle est aujourd'hui à sa version 3. Nous avons étudié la version 2 car elle est la plus utilisée par la communauté.

L'objectif des auteurs dans [5] est de proposer un modèle simple de réseau neuronal convolutif qui permette des détections **précises** et surtout **rapides**, ce qui le rend adapté à notre problématique.

Pour réaliser ces exploits de rapidité et de précision, ils utilisent les dernières techniques du deep learning. Pour le modèle YOLO9000 (version 2), l'accent a été mis sur l'amélioration de la **précision**, de la **vitesse** et de la **robustesse** comme le nom de l'article l'indique.

La première version de YOLO avait de nombreux défaut par rapport aux autres technologies à la pointe et particulièrement, avec un nombre significatif d'erreur sur la localisation et un rappel[12] plus faible. Ainsi, le focus a été mis sur l'amélioration du rappel et de la localisation, tout en maintenant la même précision dans la classification.

Généralement, les modèles actuels ont tendance à être plus large et plus profond, pour obtenir des meilleurs résultats. Cependant, l'objectif de YOLO version 2 est d'obtenir un modèle plus précis mais sans perte de vitesse. De ce fait, les auteurs ont préféré ne peut plus grossir le réseau mais plutôt le simplifier et intégrer de nouvelles méthodes que l'on va voir ensuite :

1- L'amélioration de la **précision** :

- La **normalisation du batch** [6], qui a pour objectif de normaliser (c'est à dire de se rapprocher d'une loi de distribution normale) les entrées des neurones. Cette méthode permet de converger plus rapidement vers une solution, et d'éviter d'introduire une technique de dropout[13] sans pour autant faire du surapprentissage[14].
- **Classifieur de plus haute résolution**, entraîné sur ImageNet, pour extraire le plus d'information généraliste pour ensuite réalisé un "transfer learning" pour un entraînement particulier.
- **Nouveau système pour la localisation** des objets basé sur du clustering plutôt que sur des anchors boxes[15].
- **Extracteur d'information à fin-grain** pour améliorer la détection des plus petits objets, avec des maps de feature 26x26 au lieu de 13x13.
- **Entraînement multi-échelle** car grâce au redimensionnement et les couches de pooling[16] on peut adapter le réseau à la taille de l'image d'entrée, ce qui permet l'apprentissage sur différentes résolutions et donc une meilleur généralisation.

2- L'amélioration de la **vitesse**

- Un **nouveau système d'extraction** plus fin pour remplacer VGG16 comme extracteur d'informations. En effet Il est jugé selon l'article très performant et très précis mais trop complexe pour la majorité des cas. Ainsi, ils proposent l'extracteur Darknet-19 (à 19 couches, schéma en annexe I) basé sur l'architecture de GoogleNet. Il permet de faire passer le besoin en calcul de 30.69 Milliards d'opération à 8.52 (-70%) tout en ne faisant passé la précision sur image de 90% à 88% ce qui est acceptable.
- Au cours de l'entraînement de la détection, l'on **apprend peu de box** (5 et 5 coordonnées) et **peu de classe** (et si l'on apprend qu'une seule classe c'est encore mieux), ce qui permet de réduire le nombre de filtre de détection et donc diminuer le temps en calcul nécessaire.

3- L'amélioration de la **robustesse**

- Nous n'allons pas expliciter la partie "robustness" de l'article, car elle n'a pas d'intérêt pour notre problème comme avec la méthode du "wordtree" pour atteindre les 9000 classes détectables.

En conclusion, nous avons vu différentes techniques dont on peut s'inspirer pour les réseaux neuronaux convolutifs. Son implémentation est parfaite sur le framework Darknet mais implémenté sur d'autres framework comme TensorFlow, les performances ne sont pas si éloignées d'autres modèles comme MobileNet que l'on va étudier ensuite.

2. MobileNet et SSD

MobileNet est un modèle proposé dans l'article [7] par des chercheurs de Google Brain en Avril 2017. Une seconde version de ce modèle est publiée en Mai 2018.

[12] Recall, part de bons objets détectés par rapport aux objets à détecter

[13] Technique de suppression de certain neurone non-significatif au cours de l'apprentissage

[14] Problème récurrent de tous modèles d'apprentissage, il s'agit de trop coller aux données d'apprentissage et donc avoir une mauvaise généralisation

[15] Système d'ancre répartie sur toute l'image autour desquelles nous construisons les rectangles de détection

[16] Système de sous-échantillonnage pour réduire la taille d'une matrice.

L'idée derrière MobileNet comme son nom l'indique est de porter le Deep Learning sur des **appareils mobile**, donc avec des contraintes de ressource mémoire, de calcul et énergétique. Ceci rentre parfaitement dans notre problématique de la détection en temps réel.
Pour réaliser cet objectif, les auteurs ont travaillé sur la partie la plus gourmande en calcul des réseaux convolutifs, la **convolution**.
Le coût en calcul d'une convolution est représenté par $Cout_{conv} = D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f$ où D_k la taille du noyau de convolution, D_f la taille de la couche d'entrée, *M, N* représente respectivement,le nombre de canaux de la couche d'entré et de sortie.

Une méthode pour réduire le coût en calcul serait de réduire le nombre de couche et leurs épaisseurs. Cependant, la performance de l'apprentissage profonde est liée directement au nombre de couche et de filtre de représentation.

La solution adoptée par l'équipe du Google Brain lab a donc été de créer un **nouveau type de convolution** qui ne réduit pas la taille mais réduit le nombre de calcul et de paramètres. Cette nouvelle convolution s'appelle *"Depthwise separable convolution"*. Elle se présente sous la forme d'une convolution en 2 étapes.

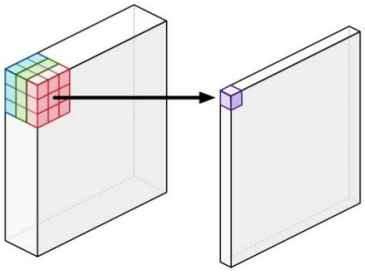


Fig 12a - Convolution classique

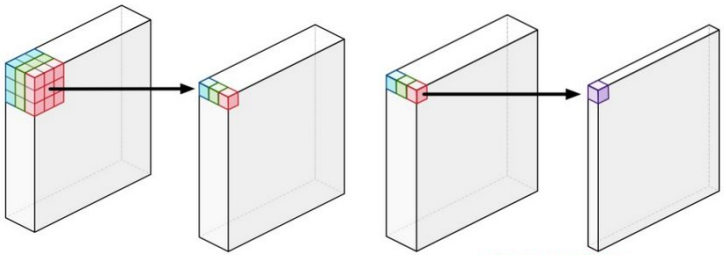


Fig 12b - Convolution Depthwise Separable

On effectue dans un premier temps une sorte de "moyenne pondérée" sur chaque canal, puis une normalisation de batch, ensuitele ReLU6[17] est appliqué et finalement une convolution de 1x1 est utilisée pour obtenir une map sur laquelle on applique encore une normalisation de batch et un ReLU6. La représentation peut ainsi être partagée entre différents filtres pour faire de la factorisation de la représentation intermédiaire. Les formules du coût en calcul sont illustrées dans les équations suivantes :

$$(1)\ Cout_{depth} = D_k \cdot D_k \cdot M \cdot D_f \cdot D_f$$ avec les mêmes signification que précédemment

$$(2)\ Cout_{point} = D_f \cdot D_f \cdot M \cdot N$$

$$(3)\ Cout_{depthwise\ separable\ conv} = D_k^2 \cdot M \cdot D_f^2 + M \cdot N \cdot D_f^2$$

Si l'on veut comparer les 2 méthodes de convolution, il suffit de calculer le rapport suivant :

$$(4)\ \frac{Cout_{separable}}{Cout_{conv}} = \frac{1}{N} + \frac{1}{D_k^2}$$

Ainsi par cette relation, on constate que plus la taille des noyaux de convolution est importante et plus le nombre de couche de sortie est élevé et plus la convolution proposée dans cet article est intéressante. Si on fixe N à une grande valeur et taille de filtre classique de 3, on diminue le coût de calcul de 1/9 donc **d'au moins 88%** .

Une autre technique qui a été introduit avec mobileNet est l'hyper paramètre ρ qui est un **coefficient pour la résolution**. De ce fait on peut réduire ou augmenter la taille de l'image d'entrée et des couches avec celui-ci. Le calcul du **coût** est ici très simple on remplace D_f^2 par ρD_f^2 donc on réduit le coût de ρ^2 .

Finalement, les auteurs ont introduit le paramètre α qui permet de jouer sur l'**épaisseur du réseau** de manière uniforme à chaque couche. Ici on remplace M et N par αM et αN. De manière équivalente on réduit le **coût** en calcul de α^2 .

[17] Rectified Linear Unit, est un type de fonction d'activation. Si x<0 -> y=0, si x>6 -> y=6 sinon y=x

Quant à l'architecture du modèle il s'agit de succession de couche de convolution classique et de cette nouvelle convolution avec des couches "fully-connected" pour la prédiction. Cette architecture se trouve en annexe J.

Suite à leurs expérimentations, l'on remarque une baisse du mAP[18] sur ImageNet par rapport à un modèle qui serait uniquement constitué de convolution classique mais comme on le voit avec le graphique ci-dessous, on diminue de plus de 90% le coût de calcul. D'une part, en trouvant les paramètres optimaux pour alpha = 0.75 et rho = 0.714 on arrive à passer d'un coût de 462 à 15.1 millions soit **-97%** (voir Fig. 13).
D'autre part, le modèle MobileNet avec alpha = 0.5 et la résolution d'image à 160*160 est aussi précis que le modèle AlexNet et 10 fois moins coûteux. En annexe K les tableaux de résultat complet issu de l'article pour les différentes configuration.

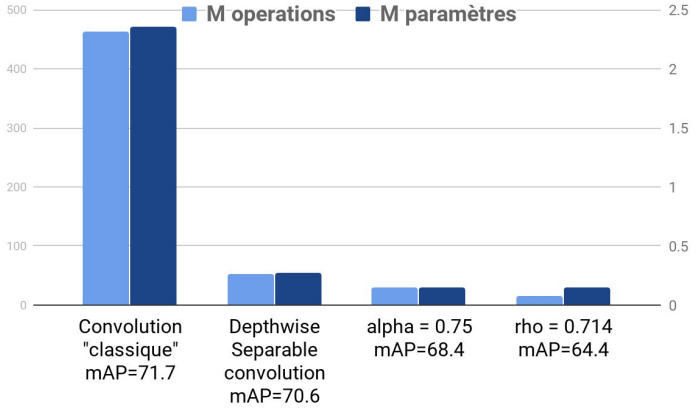


Fig. 13 - Graphique du coût en calcul et précision de détection (mAP) de la convolution en fonction des configurations

Comme on l'a dit plutôt, il existe une version 2 de MobileNet. Les améliorations de cette version sont liées à de nouvelles méthodes de convolution plus complexe. En plus de la convolution separable depthwise, un système de bloc résiduel et de goulot d'étranglement. En annexe L un schéma qui résume les méthodes utilisées. L'objectif de ces améliorations est toujours la diminution du coût de calcul mais aussi une amélioration des performances.

Pour conclure sur mobilenet, il s'agit d'un modèle en cours de développement par Google destiné aux appareils mobile avec un très bon rapport coût/performance, d'où l'intérêt que l'on peut avoir pour ce modèle surtout combiné avec un méta-modèle léger et performant comme SSD.

SSD (voir Sec. II.D) est une méta-architecture pour faire de la détection d'objet qui a vocation à être minimaliste. Combiné à un extracteur de caractéristique minimaliste comme MobileNet, on s'assure d'avoir le modèle le plus performant possible (comme vu en Fig. 11), mais aussi avec une bonne précision, donc est un bon compromis vitesse/précision.
Son fonctionnement est simple comme dit plus haut avec seulement quelques convolution supplémentaire et des combinaisons entre couche. L'architecture est présentée en annexe M.

Suite à cette étude comparative, nous soulignions et justifions notre choix de la combinaison **MobileNet/SSD.**

3. Autres

Pour autant, nous avons essayés de comprendre d'autres modèles sans entrer dans le détail, contraint par le temps.

- Ci-dessous les inconvénients de chaque modèle étudié : **Faster R-CNN avec Inception Resnet** [9] : modèle très précis mais très lourd en calcul
- **DeepLab V3+** [24]: Le modèle le plus performant dans le domaine de la segmentation d'image mais également extrêmement lourd
- **Tiny Yolo** : Version légère de Yolo avec une rapidité de calcul sans égal mais une précision vraiment médiocre après des tests que nous avons réalisés en local avec un modèle pré-entraîné.

[18] Mean Average Precision

Maintenant que nous avons choisi la combinaison MobileNet/SSD, il nous faut un framework pour l'implémenter.

B. Frameworks

Après avoir sélectionné le modèle, nous présentons dans cette section les différentes façons d'implémenter celui-ci et surtout les outils afin de lancer l'étape d'entraînement et de prédiction.

En effet, un panorama des **frameworks** et technologie est essentielle pour choisir le système le plus adéquat.

Comme illustré dans la figure 14, il existe beaucoup de framework. Certains sont open-source, d'autre pas. Nous n'avons pas eu le temps de tous les étudier en détail et nous nous sommes donc concentrés que sur quelques uns. Nous les avons choisis en fonction de leur importance dans le milieu du Deep Learning, mais pas seulement.

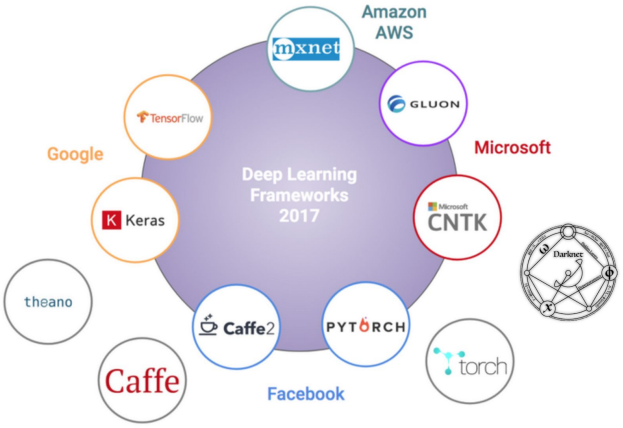


Fig 14 - Panorama des frameworks de Deep Learning en 2017

Nous avons donc choisi d'étudier en détail les frameworks **Darknet**, **Caffe**, **TensorFlow** et dans une moindre mesure **PyTorch** et **Theano**, par la réalisation de codes d'exemples, de tutoriels et d'étude du fonctionnement interne.

1. Darknet

Darknet [25] est un framework open source développé par **Joseph Redmon** en **2012**. Il est très optimisé notamment car il est codé en langage C (proche du langage machine), il utilise la technologie CUDA[19] de NVIDIA et qu'il est minimaliste. Le coeur du framework ne possède que le nécessaire pour l'entraînement et l'utilisation de réseau de neurone.

L'étude en profondeur est d'autant plus utile car il y a **très peu de documentation**. Ainsi, le site officiel ne propose que des exemples d'utilisation sans expliquer toutes les fonctions utilisables, leur rôle et leurs paramètres.
Cette étude en profondeur est rendue plus facile par le fait que le projet est entièrement open-source et en C, un langage que l'on a appris à maîtriser à l'INSA.

De ce travail d'étude nous avons réalisé et compris :

1- **L'architecture** à 3 niveaux :

- Les **algorithmes de deep learning**, il s'agit du coeur du framework.
- L'implémentation d'entraînement de modèle, de démonstration, de validation, etc.
- Les **fichiers de configurations** qui contiennent les paramètres de poids, la structure du réseau, les paramètres d'entraînement, etc.

2- La **documentation** réalisée lors de ce stage est un apport important pour l'entreprise et la communauté car il pourrait permettre à plus de monde d'utiliser ce framework et en comprendre les subtilités pour en profiter au maximum au-delà de exemples fournis.

[19] Technologie de Nvidia pour faire du calcul intensif sur carte graphique

3- L'**installation est simple** car il faut le compiler avec un fichier de configuration makefile. Cependant elle est très dépendante des logiciels et librairie liés comme OpenCV, CUDA ou Python, ce qui peut amener à des problèmes de compilation.

4- Ensuite l'utilisation est simple, notamment si l'on veut utiliser le modèle de détection YOLO car ce **framework a été développé pour YOLO** (que l'on a vu précédemment).

5- Le **modèle entraîné** se présente sous la forme de 2 fichiers, un fichier de configuration (.cfg) qui décrit la structure du réseau et un fichier de poids (.weight) qui contient les poids pour le réseau.

6- Darknet propose plusieurs **outils**, certains très utiles, d'autres très gadget. Les plus remarquable sont l'estimation du coût en calcul d'un modèle et un système pour tester seulement une partie du réseau (uniquement le classifieur ou l'extracteur de caractéristique).

7- **L'entraînement** de YOLO est simple, on peut réaliser très facilement du transfer learning[20]. Il suffit de spécifier les datasets la configuration et les poids (si l'on veut faire un transfer learning) et on obtient un fichier .weight à différentes étapes de l'entraînement.

Ce framework donne une bonne idée de l'implémentation des algorithmes de deep learning, il est très optimisé et très efficace pour le modèle YOLO. Malheureusement, il s'agit également de son point faible car il est conçu pour YOLO, et dès que l'on veut utiliser un autre modèle, cela implique le développement de la partie d'entraînement. De plus, des algorithmes sont manquant, comme le depthwise convolution de mobilenet ce qui nécessite de les coder entièrement.

Durant mon stage, j'ai **contribué** au projet car j'ai corrigé un bug de compilation lié à la librairie d'OpenCV 3.4.1 . Cette correction a été accepté par Joseph Redmon..

2. Caffe

Caffe [26] est un framework développé par **Yangqing Jia** au cours de sa thèse au laboratoire de vision de **Berkeley** et poursuivi depuis par ce même laboratoire et une communauté importante. Le framework est sous license BSD[21] et open source.

1- Il est tourné vers la **vitesse** et la **modularité**.
2- La majorité des **algorithmes** de deep learning sont parfaitement implémentés notamment en ce qui concerne les convolutions. Par contre il a de grosses lacunes[22] dans d'autre champs comme la reconnaissance de texte ou de voix à base de RNN[23].

3- Ce framework est implémenté en langage **C++**, ce qui constitue un avantage non-négligeable puisque c'est un langage puissant et optimisé mais il propose également d'excellentes **interfaces** pour les langages C, Python et l'outil Matlab. Ceux sont des langages très courant dans la recherche ce qui montre très bien l'orientation académique de ce framework, ce qui ne l'empêche pas d'être utilisé pour des applications industrielles, notamment avec le fork Caffe2 réalisé par Facebook.

4- Son principal **problème** est l'entraînement de gros modèle car il est très difficile de débugger il ne propose pas d'auto-différenciation. Cela veut dire que si l'on veut répartir l'entraînement sur plusieurs machines, on doit écrire un programme à la main pour le faire ce qui complexifie rapidement la tâche.

5- La **documentation** officielle est importante tout comme la communauté qui l'ait avec beaucoup d'exemple.
L'implémentation d'un nouvel algorithme peut s'avérer complexe car il faut écrire un nouveau module.

6- **L'installation** comme darknet se fait par une compilation avec un MakeFile. Tout comme Darknet la difficulté peut survenir avec les dépendances qui sont encore plus nombreuse.

7- **L'entraînement** prend en paramètre un modèle et ses configurations d'entraînement qui sont au format protobuf (format texte pouvant être manipulé par un éditeur de texte). L'entraînement n'est

[20] Transfert d'apprentissage, technique qui permet d'utiliser un entraînement précédent pour une nouvelle tâche
[21] Berkeley Software Distribution
[22] Selon une étude de l'INRIA, https://project.inria.fr/deeplearning/files/2016/05/DLFrameworks.pdf
[23] Recurrent Neural Network

finalement qu'un **solveur** qui va prendre ce protobuf et le lien vers les données d'entraînement afin de calculer un modèle entraîné au format *caffemodel*. (En annexe N une capture d'écran d'un entraînement)

8- Pour la **prédiction**, le modèle entraîné et les interfaces caffe sont utilisés.

En conclusion, le framework caffe permet de configurer et tester rapidement des modèles et des configurations. Malheureusement il est en perte de vitesse, notamment dans le secteur industriel.

3. TensorFlow

Le framework TensorFlow [27] est le plus en vogue[24] depuis 2017 (avec la sortie de la version 1.0), développé par l'équipe **Google Brain** à partir d'un framework interne nommé DistBelief. Il est proposé sous license open source depuis Novembre **2015**. Ce framework a su s'imposer sur le marché pour être aujourd'hui intégrer dans énormément de solution et en premier lieu avec les produits de Google.

Le fonctionnement, comme nous pouvons le voire sur le schéma ci-dessous se présente sous une forme très **modulaire**.
En effet il fonctionne sur de nombreux matériels comme les CPU[25], GPU[26], TPU[27] avec des accélérateurs de calcul comme XLA[28].

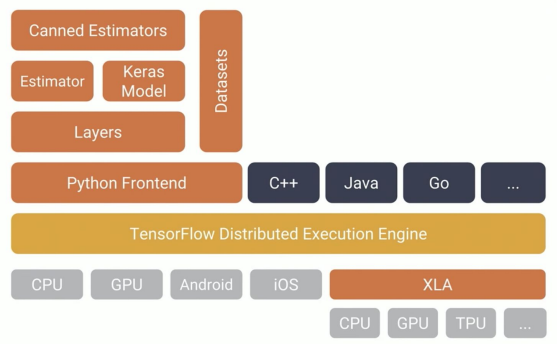


Fig 15 - Architecture de TensorFlow

Il y a beaucoup de modules, comme pour la gestion des datasets, l'interface du framework Keras (qui rend le développement plus simple), etc.
On développe les modèles avec le langage **Python** mais nous pouvons utiliser de **nombreux langages** pour implémenter les modèles entraînés TensorFlow dans des solutions comme C++, Go, Swift, Javascript,.... Enfin le coeur (TensorFlow Distributed Execution Engine) est lui développé en C++ pour une optimisation optimale, une très bonne répartition de charge et des options de compilations avancées.

Le **fonctionnement classique d'un programme** TensorFlow se base sur plusieurs étapes; la définition du modèle; l'optimisation; l'initialisation des variables (les tenseurs des poids par exemple). Après ces étapes, une session de calcul peut être lancé pour les phases d'apprentissage ou de prédiction.
Cette notion de **Session** est primordiale afin de répartir les calculs en blocs d'une part et d'autre part pour manipuler l'ordre d'exécution de chaque bloc de calcul. Cette manipulation de l'ordonnancement du calcul est primordial pour les grands modèles d'apprentissage car ces modèles sont en réalité des tenseurs[29] à calculer qui sont connectés entre eux en formant un graph.

La configuration des modèles est sauvegardée dans un format standardisé **protobuf** .pb (format binaire) ou .pbtxt (format texte) et les **modèles entraînés** sont au format .ckpt puis au format .pb lorsque les paramètres de poids sont constants. En effet ces paramètres de poids sont utilisées pour l'étape de prédiction.

---

[24] En fonction du nombre d'étoile sur github et l'activité sur stackoverflow
[25] Central Processing Unit
[26] Graphics Processing Unit
[27] Tensor Processing Unit
[28] Accelerated Linear Algebra
[29] Objet très générale de l'algèbre multilinéaire qui dans tensorflow est une sortes de matrice de dimension élevée.

En réalité l'avantage du framework TensorFlow par rapport aux autres est sa diversité **d'outils** (développé en grande partie par la communauté) qui rendent le développement de modèle facile. Nous allons en détailler quelques un que nous avons utilisés au cours de ce stage. Ces outils sont :

TensorBoard :
Cette outil sert à visualiser les étapes d'apprentissage . La procédure est:

- Mettre en place des "summary" dans le code qui vont écrire des évènements dans des fichiers "event".
- Lancer le serveur TensorBoard qui va prendre en paramètre le dossier d'événement et fournir une interface web comme présenté en Annexe O.

L'on peut y visualiser des graphiques (comme au cours de l'entraînement, la fonction de coût), des histogrammes ou bien la représentation du graph de session.
La principale utilité pour notre cas de l'entraînement d'un réseau pour de la détection d'objet est de suivre le déroulement de l'apprentissage et l'évaluation du modèle.

TFProf :
Permet de faire du **profilage** du modèle, en évaluant le coût en calcul et en mémoire. De plus il permet de mesurer l'impact de chaque couche dans la performance globale. Cet outil est utile pour évaluer le modèle après l'entraînement afin de l'améliorer, notamment en terme de rapidité de calcul.

TFDbg :
Cet outil est un débogueur sous le même format que GDB pour des programmes C. En effet, il peut arriver qu'au cours d'un entraînement, l'on ait des valeurs aberrantes dans une couche. Cet outil permet de lancer le programme en mode interactif pour visualiser le contenu des variables et au besoin les modifier pour comprendre ce qu'il se passe.

Enfin le dernier module qui a grandement motivé le choix de TensorFlow est une API pour l'entraînement simple de modèle pour la détection d'objet (**TensorFlow Object Detection API**). Cet API permet de rendre simple la configuration et l'entraînement de modèle pour ce type de problème de Deep Learning en réduisant quasiment à zéro le besoin en développement.
Elle se présente sous la forme très simple d'un programme python à lancer pour l'entraînement avec certains paramètres, mais nous verrons le fonctionnement plus en détail dans la Section IV.B-2 d'implémentation.

Pour **conclure**, Tensorflow est en plein essor grâce à une communauté de chercheur, d'entreprise, de passionné très importante. Il propose non seulement une modularité et des outils très utile notamment pour la vision par ordinateur, mais il est aussi très optimisé et facilement parallélisable.

4. Autres

Nous allons terminer notre tour d'horizon des framework de Deep Learning avec 3 frameworks assez connu mais que nous n'avons pas eu le temps d'étudier dans le détail, **Theano**, **Pytorch** et **Keras**.

Theano [28] est le premier framework de Deep Learning open source, il est développé par l'université de Montréal en 2008. Il est encore très utilisé dans la recherche mais il est progressivement abandonné tant par les utilisateurs que par les développeurs. Le développement et la compilation peuvent être laborieux et il est donc conseillé d'utiliser une librairie de plus haut niveau dessus comme Keras.

PyTorch [29], comme son nom l'indique est basé sur un framework aujourd'hui quasiment disparu, **Torch**. Il est développé par **Facebook** en Python et se veut le concurrent direct de Tensorflow avec les mêmes système de module. Les principales différences que nous avons trouvé sont : une communauté moins importante et un autre paradigme que les sessions pour l'exécution du code.

Enfin **Keras** [34] qui n'est pas réellement un framework à part entière puisqu'il a besoin d'un framework de plus bas niveau pour fonctionner (comme TensorFlow, pytorch, CNTK[30] ou bien MXNet [31]). Il permet de rendre la conception de modèle très accessible notamment pour les chercheurs dont le développement n'est pas leur domaine. De plus il est minimaliste, modulaire et intégré par défaut dans TensorFlow. Par contre la communauté est faible et surtout il y a peu de modèle pré-entraîné Keras, on doit passer par des modèles TensorFlow ou Caffe.

[30] Microsoft Cognitive Toolkit
[31] Framework de Deep Learning de la fondation Apache

C. Choix des technologies

A la suite de l'étude de toutes ces technologies, nous avons donc choisi d'utiliser **TensorFlow** comme framework pour tous les avantages que nous avons déjà cité comme l'API de détection d'objet et l'outil Tensorboard.

Pour autant, Darknet pourrait être la base d'une solution avec le modèle optimisé YOLO si le temps de développement n'était pas trop long, au lieu d'utiliser un gros framework comme TensorFlow.

En ce qui concerne les modèles, il a été difficile de faire un choix mais il a été motivé par le choix du framework TensorFlow. Nous avons donc décidé d'utiliser un modèle **SSD avec MobileNet** même s'il ne faut surtout pas négliger YOLO et ses capacités, d'autant plus si l'on souhaite intégrer une solution à base de Darknet.

Nous avons désormais, à la suite de l'étude bibliographique et de l'étude comparative des technologies, un framework de Deep Learning et un modèle à entraîner. Donc nous pouvons réaliser une implémentation.

IV. Implémentation et résultats

A. Planification du workflow

Après la recherche bibliographique et l'étude des technologies, nous nous sommes entrepris à mettre en place un **workflow**[32] qui permet la réalisation d'un POC répondant à la problématique, qui est la détection des têtes en temps réel à l'aide des dernières techniques de deep learning.

Le **Flux d'entraînement** est inspiré de celui du "data scientist" classique et se décompose comme suit (voir Fig. 16) :

- La **collecte des datasets** et leur traitement pour obtenir un dataset de tête
- La **configuration** de l'API de détection avec la configuration du modèle et de l'entraînement
- L'**entraînement** à proprement parlé avec TensorFlow et l'API de détection du modèle avec le dataset de tête et la configuration.
- L'**évaluation** de l'entraînement, le **benchmark** et la **conversion** du modèle pour ensuite implémenter le modèle entraîné dans une solution de détection.
- Enfin **l'implémentation** d'une solution de détection.

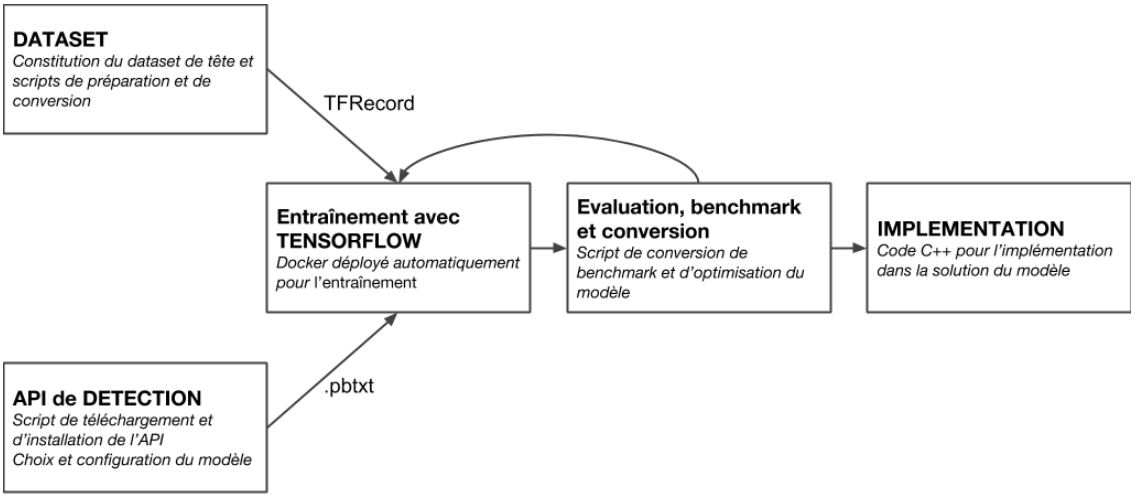


Fig. 16 - Workflow d'implémentation

Le workflow a été planifié minutieusement avec un calendrier prévisionnel et la liste de toutes les parties à développer ce qui a permis d'aller très vite sur sa réalisation.

[32] Flux de travail ou chaîne de traitement

Pour gérer au mieux les versions et le développement de différent module en parallèle, nous avons décidé d'utiliser l'outil **git** qui est un outil de versionning similaire à svn[33]. D'une part le code est versionné, donc l'on peut suivre les modifications mais il permet aussi de partager le code facilement, via l'utilisation de la plateforme Github.

B. Apprentissage

Cette partie consiste à mettre en place une infrastructure pour mener un entraînement en fonction des choix techniques qui ont été porté sur la partie de recherche et d'étude. Cela aurait pu être une succession de tâche manuelle mais nous avons opté pour une architecture **automatique** et **flexible** et **simple**.

1. La collecte et traitement des données

Ce premier module est constitué de 2 parties. Une première pour la **collecte des images** et une seconde partie pour leurs **traitements automatiques** pour générer un dataset adapté au framework Tensorflow.

Le **choix du dataset** en deep learning appliqué à la vision par ordinateur est assez simple, c'est une collection d'image annotée avec les éléments que l'on souhaite détecter. Donc dans notre cas il a fallu trouver des bases de données d'images de tête annotée dans le plus de configuration possible.
A la suite de nos recherches, nous avons trouvé des bases de données très connues et très généraliste qui propose des objets "tête" parmis les classes reconnues mais aussi des datasets plus spécifique à des personnes.

Ainsi nous avons trouvé et étudié les datasets publique suivant :

- **Pascal VOC** [36]: généraliste, une référence dans le deep learning puisque c'est avec lui que sont généralement évalués les modèles. Il contient 27.000 images étiquetées. Dans le format Pascal VOC XML.
- **Ms COCO** [37]: Common Object in Context est équivalent à pascal VOC mais avec des annotations liées à une segmentation de l'image en classe d'objet, donc pas seulement un rectangle qui encadre le mieux l'objet.
- **ImageNet** [38]: Là aussi une base de données d'images généralistes annotées, la différence est le nombre (plus de 14 millions) et le fait que la production de ces images est participative. Pour les têtes nous avons trouvés 320 images qui correspondaient à nos critères.
- **Hollywood Heads** [2]: images issues de film avec l'annotation des têtes. Elle contient près de 220.000 images au format Pascal VOC XML. Son soucis c'est que les angles de caméra sont ceux du cinéma, donc pas très naturel.
- **INRIA Person** [39]: images labellisées pour la détection des personnes. Elle contient 902 images de plus de 2.000 personnes. Le format d'annotation est très particulier mais surtout il s'agit de personne et non de tête, il a donc fallu la convertir dans un format utilisable pour nous.
- **Person In Photo Albums** [40]: proposé par le laboratoire de vision de l'université de Berkeley, il s'agit de l'annotation de tête dans des images venant d'album photo. Elle compte 30.000 images mais en réalité, il manque beaucoup d'annotation ce qui réduit drastiquement le nombre d'image utilisable. Le format d'annotation est là aussi assez particulier.
- **UcoHead** [41]: Base de donnée très particulière, destinée à la calibration des caméras. Il s'agit de plusieurs têtes prisent en photo (de taille 40x40 pixels) sous diverses angles déterminés. Il n'y a pas d'annotation puisque les images ne représentent qu'une tête, mais on peut en générer. Il y a 37.700 images mais en réalité, certaines images sont trop proche et donc il peut être intéressant d'en sélectionné qu'un sous-ensemble.

De plus, on peut compléter le dataset avec des images issues des caméras installées par l'entreprise Cliris pour obtenir un modèle adapté aux données Cliris.

- **Cliris** : Mise en place d'un système d'annotation "humain". Au moins une centaine d'image serait suffisante pour compléter le dataset.

Pour notre situation et vu le temps imparti, nous avons choisi d'utiliser, les datasets **Hollywood Head**, **INRIA person**, **PiPA** et **UcoHead** pour leur nombre d'exemple (d'image) très important et spécifiquement de tête (ou de personne).

[33] Subversion de la fondation Apache

L'objectif est d'automatiser le processus du téléchargement des datasets à la conversion finale. Cet objectif nous a mené à réaliser ce travail d'écriture d'une chaîne de script (bash et python) représentée ci-dessous.

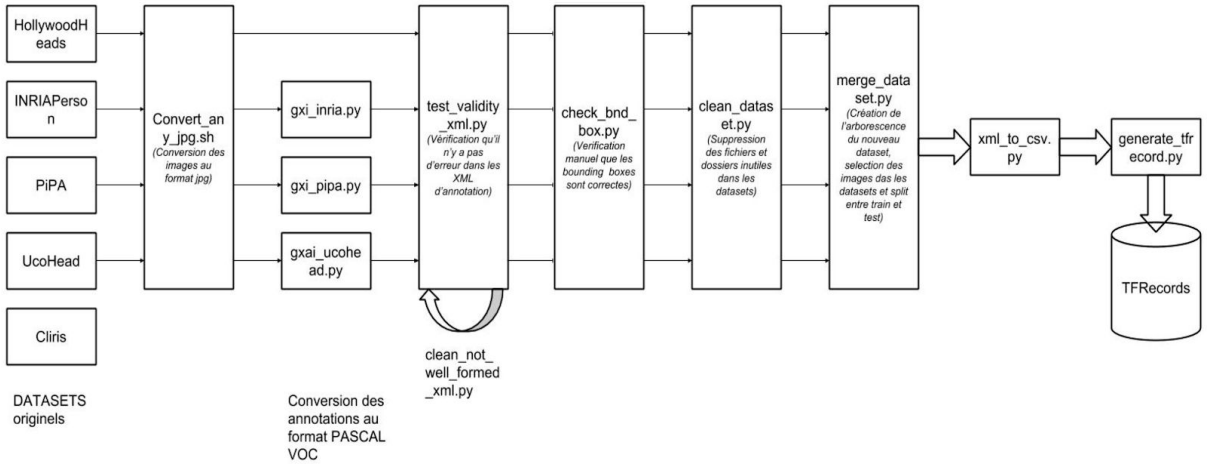


Fig 17 - Pipeline de script pour le traitement des datasets

On commence par convertir les images et ces annotations au bon format XML.
On vérifie ensuite les erreurs d'annotations et on a la possibilité de vérifier visuellement sur quelques images aléatoire les rectangles d'annotation.
Ensuite on convertit les datasets et on fait la synthèse dans un nouveau dataset **ClirisHead**. C'est au cours de cette étape que l'on sélectionne des exemples aléatoire dans les datasets traités.
Ce nouveau dataset (dont on fait la sauvegarde par une archive compressée) est ensuite convertie au format **TFRecord**, qui est un format binaire afin d'optimiser l'entraînement TensorFlow.

Nous avons donc sélectionné **40.902** images pour le dataset final répartie en 70% **d'entraînement** et 30% de **test**. Le pipeline complet (sans le téléchargement) met **3h50** environ à s'exécuter et l'on récupère un dataset en 2 parties faisant **4.2Go** au total.

A la suite de tous nos scripts on obtient donc un dataset au format TFRecord qui peut être utilisé tel quel par n'importe quelle programme TensorFlow.
Nous pouvons donc passer à la suite du workflow avec la configuration du modèle et de l'entraînement.

2. Configuration du modèle

L'**API de détection d'objet** fournie par Tensorflow va nous simplifier énormément la tâche comme nous l'avons vu dans la partie sur TensorFlow, Sec III.B-3. Elle nous permet de configurer un modèle très facilement pour faire de la détection d'objet à partir de modèle connu. Il est possible de créer son propre modèle en prototxt, mais ici on utilise un modèle implémenté.

Avec cette API, l'on doit configurer 2 fichiers, la **map d'étiquettes et** le **pipeline d'entraînement**.
La map d'étiquettes (*Item { id:1, name: 'head' }*)est un fichier avec le numéro d'identifiant (id: i) associé à une étiquette (name: 'label').
Le deuxième fichier de configuration est le "**pipeline**" du modèle qui se présente sous la forme d'un .config avec une syntaxe proche du json (voir annexe P) avec les blocs à configurer.

Nous allons attirer l'attention sur les paramètres qui vont nous permettre de raffiner le modèle (SSD/MobileNet) et son entraînement, notamment à la suite de l'analyse d'un entraînement (voir Sec. IV.B-4).

Comme montré dans la Sec III.A-2, deux paramètres sont essentiels pour la configuration du modèle MobileNet, α et ρ Ces deux paramètres représentent respectivement, le redimensionnement de la taille de l'image (image_resizer) et la profondeur du réseau (depth_multiplier).

D'autre part, les paramètres primordiaux de l'entraînement d'un modèle sont :

- **Batch_size** (la taille du batch):. le batch est le lot d'exemple que l'on va donner au réseau avant de mettre à jour les poids. Si on le met à 1, on va probablement faire un sur-apprentissage car les poids vont être modifiés pour cet exemple particulier. Inversement

si on met un nombre trop grand, il y aura beaucoup d'image en mémoire vive et il y a un risque de plantage de l'entraînement par manque de mémoire.

- **Optimizer** (l'algorithme de minimisation): il est primordial car c'est lui qui va permettre de minimiser l'erreur du réseau et donc de faire "apprendre" au réseau. Ici RMSProp[34].
- **Learning_rate** : C'est le premier paramètre à choisir et modifier pour l'entraînement car d'un mauvais réglage on fait du sur-apprentissage ou du sous-apprentissage très facilement, ce qui est visible sur la fonction de coût. On voit intervenir ce taux (α) dans l'équation de l'algorithme d'optimisation de RMSProp, $\Delta\theta_t = -\alpha g_t v_t^{-1/2}$ où $\Delta\theta_t$ est la différence de poids à appliquer.
- Les autres paramètres de l'optimiseur comme le moment, epsilon, etc… vaut mieux conserver les valeurs préconisées par les concepteurs de ces algorithmes pour avoir un fonctionnement optimal
- **Fine_tune_checkpoint** : il est très important puisque c'est ce paramètre qui permet de faire le **transfer learning**. C'est-à-dire un apprentissage antérieur pour ne pas ré-apprendre de 0. Nous préconisons d'utiliser un modèle pré-entraîné Pascal VOC pour améliorer la qualité de l'apprentissage mais aussi la rapidité.
- **Num_step** : il permet de définir le nombre d'exemple pour l'apprentissage. Il n'est pas le nombre d'itération. Pas = itération x taille du batch.
- **Data_augmentation_option**: elle permet d'augmenter virtuellement la taille du dataset en faisant des recadrages aléatoire de l'image ou bien des zooms, ces techniques peuvent permettre un meilleur entraînement en diversifiant les exemples.

Il y aurait beaucoup à dire sur la configuration du pipeline mais nous allons s'en arrêter à ces paramètres essentiels sur lesquels on va jouer pour améliorer le modèle après l'analyse des entraînements.

3. L'entraînement avec Tensorflow sur Docker

La partie d'apprentissage est le coeur d'un bonne détection. De ce fait, nous avons étudié divers contraintes, comme la flexibilité, la simplicité, la robustesse et l'évolutivité.

La première est la **flexibilité**, en effet le système doit permettre de changer facilement de modèle ou de dataset sans nouvelle implémentation et surtout sans ajout d'autres opérations.
Ce qui nous amène à la seconde contrainte, la **simplicité**. En effet, j'avais des compétences et une connaissance en Deep Learning que j'ai pu renforcer avec ce stage, mais ce ne sera pas forcément le cas de l'utilisateur du système d'entraînement. Donc ce système doit être simple d'utilisation.
La troisième contrainte est la **robustesse**, on entend surtout la résilience aux pannes. En effet si l'on a un entraînement qui dure plusieurs jours et qu'une panne survient, il faut pouvoir reprendre l'entraînement de manière automatique et à partir de la dernière sauvegarde pour éviter de le reprendre du début.
Enfin la dernière contrainte est **l'évolutivité** puisque nous voudrons déployer le système dans un futur proche sur une autre machine plus puissante voir sur un serveur pour externaliser le calcul.

Pour répondre à ces contraintes nous avons bien sûr utilisé **TensorFlow avec l'API de détection** qui rendent l'entraînement simple avec seulement un fichier de configuration et les datasets sous format TFRecord.
Pour la **flexibilité**, la **robustesse** et **l'évolutivité** nous nous sommes appuyés sur la technologie de Docker. **Docker** est un outil qui permet d'automatiser le déploiement d'application dans des "container" qui sont une sorte de VM[35] légère. Ces conteneurs utilisent également une technologie de Nvidia pour pouvoir utiliser les ressources GPU (ce qui est essentiel pour mener rapidement un entraînement en Deep Learning), **Nvidia-Docker** (voir Fig. 18a). Ces conteneurs sont des instances d'une image docker. Une image docker est créée à la compilation d'un **dockerfile** (voir annexe Q), qui est un fichier dans lequel on "code" ce que l'on veut sur le système, par exemple Ubuntu 16.04-64 avec python3 et jupyter.

Pour notre cas, l'architecture de notre système docker est représentée par la figure suivante.

[34] Root Mean Square Propagation, un optimiseur de la famille des SGD, Stochastic Gradient Descent, sûrement la famille d'optimiseur la plus utilisée en Deep Learning
[35] Machine Virtuelle

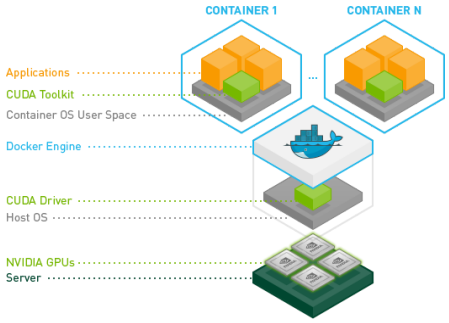


Fig 18a - Architecture de Nvidia docker

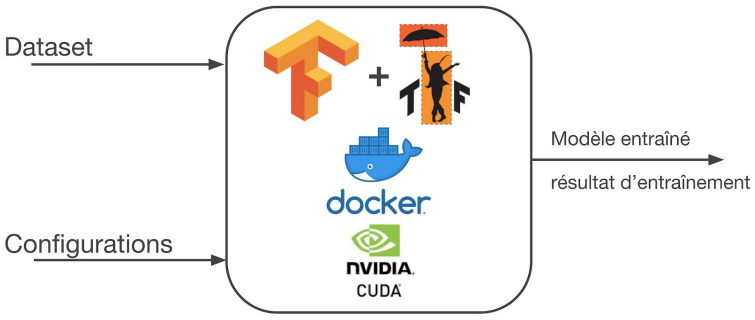


Fig 18b - Architecture du docker

Tout d'abord, les datasets et la configuration sont sauvegardés dans 2 dossiers. Ensuite, une instance de l'image (voir Figure 18b) docker est lancée afin de **mener l'entraînement** (en annexe R une capture d'écran de ce lancement). Au cours de l'entraînement, nous récupérons les informations de l'itération courante ainsi que de la fonction de coût pour l'entraînement et le dataset de validation. A la fin de l'entraînement le conteneur sauvegarde le modèle entraîné ainsi que les évènements d'entraînement dans un dossier. Nous pouvons donc passer à l'analyse de l'entraînement et la préparation du modèle pour son utilisation dans une application.

4. Analyse des résultats d'entraînement

Nous avons obtenu à la suite de l'entraînement un modèle. Il est maintenant nécessaire **d'analyser** le résultat obtenu avant de l'utiliser en production.

En annexe S et T nous pouvons voir l'interface de **tensorboard** sur l'évolution de l'entraînement, celle de la visualisation de l'architecture et de l'évaluation d'un modèle.
Ensuite l'interface du **programme** qui permet de visualiser facilement le rapport de benchmarking du modèle est aussi illustrée en annexe T.
On peut y voir le **temps d'inférence** en fonction des couches, des types de noeuds,... On peut notamment remarquer que les premières couches de convolution sont les plus coûteuses en calcul. Enfin certaines autres informations sur le modèle comme les tenseurs d'entrée, de sortie, le coût en calcul,...

Avec ces informations, nous pouvons évaluer un modèle avant de passer à la conversion. Nous allons donc ci-dessous détailler le premier entraînement.
Les **principaux paramètres** sont:

- Pour le modèle : SSD + mobilenet v2, une taille de l'entrée de 300x300, ρ de 1.
- Pour l'entraînement : batch de 10, RMSProp avec un learning rate de 0.004, un moment de 0.9 et pas decay.

L'entraînement a duré sur **100.000** pas pendant **17h55** sur une carte graphique Nvidia GTX960.
Le modèle obtenu fait 50 Mo et nécessite 1.28 milliards d'opération, nous pouvons donc espérer avec notre machine au mieux 17 Fps pour la détection.

Les résultats montrent d'abord une précision au test pascal VOC de **85**% (c'est à dire que la détection est bonne dans 85% des cas en prenant en compte de la position détectée). De plus l'on a un recall de **76**%, c'est-à-dire que 76% des têtes qui auraient dû être détectée l'on été. Ce score montre tout de même que l'on a raté une tête sur 4 ce qui n'est pas extrêmement satisfaisant.

Ensuite si l'on observe les **courbes de coût total** et de précision sur l'évaluation au cours de l'entraînement on en déduit un **sur-apprentissage** (qui se confirme avec une vérification visuelle sur les images en annexe U).
Vu la fonction de coût c'est en partie dû à un haut learning rate.

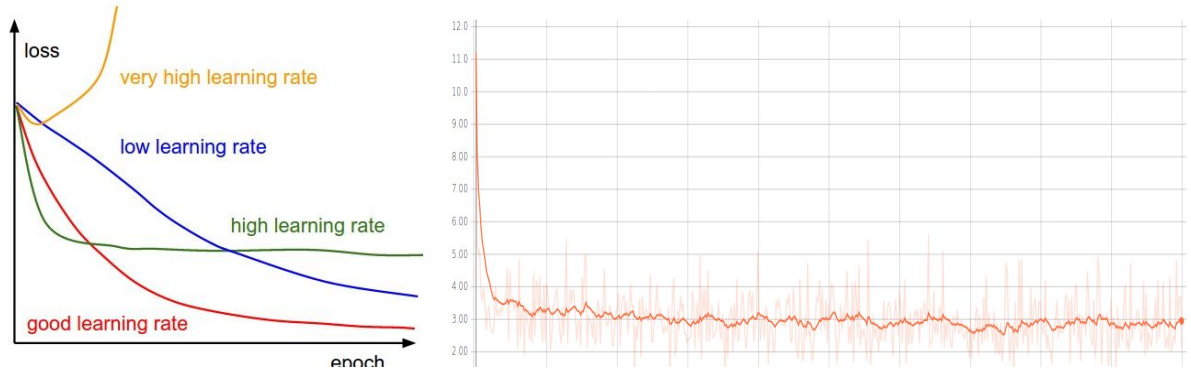


Fig 19a (gauche)- Evolution de la fonction de coût en fonction du learning rate et courbe de coût obtenue
Courbe de gauche extraite de http://cs231n.github.io/neural-networks-3/ cours de réseau de neurone de Stanford
Fig19b (droite) - Evolution de la fonction de coût au cours de l'apprentissage obtenue au premier entraînement

Donc on peut régler ce problème en **diminuant le learning rate** voir un meilleur apprentissage avec un learning rate dégressif au cours de l'entraînement. Aussi, Le dataset n'est pas très bon, il faudrait des têtes annotées issues d'image caméra et de caméra type de Cliris en particulier car on remarque que pour les images similaires à celles que l'on peut trouver dans les datasets d'entraînement l'on a de bonne détection mais pas pour les images de caméra trop différentes.

5. conversion et optimisation du modèle

Nous sommes toujours dans le même module d'évaluation et de **conversion** mais nous allons passer à la conversion à proprement dit. C'est-à-dire que l'on va transformer le modèle obtenu par l'entraînement en modèle "gelé" importable dans n'importe quel programme avec l'API d'implémentation de TensorFlow.

En effet, nous n'en avons pas parlé, mais à la sortie de l'entraînement, le contenu du dossier ressemble à ceci :

- **Checkpoint** : document donnant les liens vers les checkpoint
- **Graph.pbtxt** : décrit tous les noeuds du modèle
- **Model.ckpt.meta-xxx** : les méta données du graph à l'étape xxx
- **Model.ckpt.index-xxx** : les informations à l'indice de fin d'entraînement pour pouvoir reprendre l'entraînement plus tard
- **Model.ckpt.data-xxx** : Les poids pour l'itération xxx
- **Event-...** : le fichier d'event pour tensorboard

Avec des programmes de conversion et d'optimisation de graph fournis par tensorflow, nous concevons un script qui convertit ce dossier en modèle gelé .pb .

Par **optimisation** on entend en réalité la suppression des noeuds inutiles et la factorisation de certains ce qui accélère l'inférence.
Tensorflow propose également un programme "**learn2compress**" pour réduire la profondeur du graph et donc diminuer le temps d'inférence avec un modèle au format tensorflow lite, prévu pour le mobile. Malheureusement nous n'avons pas eu le temps de le mettre en place.

Fig 20 - Architecture du Learn2Compress

Son fonctionnement est simple, au cours d'un entraînement l'on va apprendre à un plus petit modèle à copier le modèle original pour avoir les mêmes réponses mais avec moins de noeud, c'est le principe de la distillation. Le schéma explicatif en annexe V.

Cette partie conclut l'apprentissage de notre modèle. Nous avons donc constitué un dataset de tête, ClirisHead. Nous avons aussi configuré un modèle SSD/MobileNet avec ses paramètres d'entraînement. Nous avons ensuite procédé à l'entraînement de notre modèle avec Tensorflow à l'aide de Docker. Enfin nous avons évalué le modèle entraîné obtenu pour ensuite le convertir pour la suite qui est l'implémentation d'un POC de détection .

C. Prédiction

1. Implémentation

Nous avons désormais un modèle entraîné pour la détection des têtes, issu de notre workflow d'entraînement. L'objectif est maintenant de réaliser un POC qui permet de montrer la viabilité de l'utilisation du Deep Learning pour la détection de personne.

Cette partie est finalement assez indépendante de la première. En effet, en téléchargeant un modèle entraîné sur de la détection d'objet généraliste, l'on n'aurait très bien pu faire des solutions d'implémentation. C'est ce que l'on a fait dans un premier temps pour tester les frameworks, comprendre le fonctionnement des modèles,... Mais pour notre problème particulier de la détection de tête et encore plus particulièrement de tête depuis des caméras, il nous fallait un modèle dédié.

Nous avons réalisé une **première implémentation** en python avec la bibliothèque OpenCV[36]. Son fonctionnement est très simple mais finalement répond au problème de la détection, l'algorithme est présenté en annexe W.
Son problème est son manque total d'optimisation et langage Python alors que toutes les solutions de Cliris sont en C++.
Pour profiter des optimisations de Tensorflow, nous choisissons donc d'utiliser **l'API d'implémentation C++** fournie. Elle est très bien documentée sur le site officiel et possède de nombreux exemples. La compilation du code C++ se fait à l'aide de l'outil **Bazel** qui est un compilateur cross-plateform et open-source. Il permet notamment de compiler le C++ avec un fichier de config, il est similaire à Maven ou Make mais avec des fonctionnalités comme la compilation partagée.

Pour pouvoir **simuler l'environnement** dans lequel cette solution pourrait être déployée nous avons utilisé une machine virtuelle avec le même système d'exploitation que les "box" Cliris (Debian 8), 2 coeurs, 4Go de RAM, pas de GPU. Sur cette machine virtuelle, nous avons mis le binaire et simulé un flux vidéo à l'aide d'une vidéo.

2. Résultats

L'analyse des résultats s'appuie sur d'une part les performances et la précision. Les performances sont appréciées avec le nombre d'image par seconde. La précision est estimée visuellement

Ces résultats ne sont pas mauvais mais sont loin d'être idéaux car on tourne autour de 8 fps ce qui est loin du temps réel. De plus, comme nous avons pu le voir après analyse de l'entraînement, la précision n'est pas encore là même si ce n'est pas mauvais. (en annexe X une capture d'écran).

Ce POC montre qu'**il est possible de faire de la détection de tête avec du Deep Learning** en quasi temps réel sur une machine de Cliris, ce qui prouve une industrialisation possible dans les années à venir. Ceci conclut l'implémentation comme solution au problème.

V. Conclusions techniques et perspectives

A. Conclusion sur le résultat

Comme nous l'avons vu avec la partie résultat de l'implémentation (voir Sec. IV.B-2), le POC **prouve** la possibilité **d'industrialisation** d'une solution à base de Deep Learning pour la détection des têtes.
Malgré tout, ces résultats ne sont pas encore satisfaisant en terme de performance et de résultat, surtout lorsqu'ils sont mis en rapport avec les résultats issus de la solution à base de Machine Learning en test actuellement.
Ceci nourrit des pistes de réflexions et des perspectives pour la suite de ce travail que nous exposons dans les paragraphes suivant.

[36] Open Computer Vision, librairie largement utilisée dans le domaine de la vision par ordinateur

B. Matériel dédié

Une possibilité d'avenir et plus particulièrement pour Cliris c'est l'utilisation d'un matériel dédié à l'inférence, c'est à dire à la prédiction par un modèle.
En effet, la configuration actuelle des "box" Cliris déployées chez les clients ne permet pas l'utilisation de carte graphique ne serait-ce que pour une question de coût. De ce fait, le calcul de l'inférence se fait sur le CPU sauf que ce n'est pas adapté à ce genre de calcul et surtout la box n'a pas que cette tâche à réaliser, donc le modèle ne peut pas s'accaparer toutes les ressources.
Ainsi nous avons pensé à un matériel dédié low cost qui n'aurait comme tâche que de faire l'inférence. Nous avons testé le dongle "Neural Compute Stick" de **Movidius**, une filiale d'Intel.

Fig 21 - Movidius Neural Compute Stick

Sous son aire de clef USB, il embarque tout de même une puce neuronale Myriad avec une puissance de calcul de 27 GFLOPs[37]/s et le tout pour une consommation électrique de quelques centaines de mA pour un coût de $80.
Son fonctionnement est très simple, on établit la connexion avec un programme en C ou Python, afin de charger le modèle entraîné Caffe ou TensorFlow, puis on met la donnée en entrée du réseau (une image par exemple) et le dongle nous renvoie un vecteur contenant les valeurs de sorties du réseau. En annexe Y, un schéma qui résume le fonctionnement.
Suite à nos tests, nous n'avons pas remarqué d'amélioration de performance par rapport à l'utilisation sur CPU (Intel i5) classique. Pour autant, il faut garder à l'esprit qu'avec la popularité du Deep Learning et la poussée de l'Intelligence Artificielle (Apple, Android), il y aura de plus en plus de matériel dédiés qui répondent aux exigences de notre problématique de détection en temps réel low cost.

C. Amélioration du modèle

Bien sûr, on ne peut pas proposer des perspectives pour ce stage sans parler de l'amélioration du modèle que ce soit en terme de précision ou de vitesse puisque c'est le combat permanent que mène le Deep Learning. Plus rapide et mieux. Cette amélioration du modèle passe par 2 grands axes.

D'une part, continuer les **entraînements** en ajoutant des données, notamment des données Cliris, de nouveaux paramètres d'entraînement, une meilleure visualisation des résultats de sortie avec plus d'indicateur possible pour faire les bons choix d'entraînement.
Grâce à l'infrastructure mise en place, cette tâche peut être facilement réalisée et déployable sur un serveur de calcul avec à la clef un entraînement plus rapide et donc plus d'entraînement.

D'autre part, faire de la **veille technologique** car comme nous l'avons vu au cours de l'étude bibliographique, le secteur du Deep Learning évolue très vite avec de nouveaux modèles et de nouvelles méthodes qui apparaissent tous les mois. Ainsi, MobileNet est à sa version 2 aujourd'hui mais il n'est pas impossible qu'une nouvelle version arrive dans les 9 prochains mois sans compter des modèles concurrents qui pourraient avoir les mêmes objectifs de précision et de rapidité.

Il ne faut pas non plus négliger l'arrivée de nouvelles techniques complètement différentes de Deep Learning qui peuvent révolutionner comme l'on fait les réseaux convolutifs en leur temps. Nous prendrons l'exemple des réseaux profond à base de "capsule" comme **CapsuleNet** apparus ces derniers mois et dont l'objectif est de remplacer les réseaux neuronaux convolutifs d'aujourd'hui.

Ainsi, nous voyons ce stage vraiment comme un déclencheur d'une nouvelle thématique qui va être poursuivie.

[37] Milliard d'opération par seconde

D. Intégration dans la solution existante

Le sujet de ce stage doit être poursuivi car il fait partie d'une volonté de Cliris d'intégrer les dernières technologies pour toujours proposer les **meilleurs services** à ses clients.
Comme nous l'avons vu dans l'introduction, la solution de Cliris se base actuellement sur la détection des mouvements. Ils sont en phase de test pour une mise en production d'une solution à base de Machine Learning.

La prochaine étape est donc **l'intégration** d'une solution Deep Learning pour tous les avantages que cette nouvelle méthode apporte comme la très bonne généralisation, le suivi, l'étude de scène complexe, etc.
Cette intégration devra garantir un niveau de qualité au moins aussi élevé que la solution existante et surtout une détection proche du temps réel. Le travail mené pendant ce stage ne garantie ni l'un ni l'autre.

E. Questions éthique et juridique

Nous souhaitons conclure ce rapport de stage avant de passer à des points plus personnel, sur une ouverture sur **l'éthique** et les **aspects juridiques** qui sous-tendent ce stage.

Tout d'abord, nous n'allons pas débattre de la dangerosité de l'intelligence artificielle mais de l'**importance des données**.
En effet ces données sont des images de tête, donc de personne qui n'ont pas nécessairement donnée leur accord et pourtant ces têtes vont être données à une intelligence artificielle qui va ensuite certes généraliser pour réaliser sa tâche de reconnaissance mais qui aura inscrit quelques part des traits de ces personnes. C'est une question très vaste mais qu'il faut se poser lorsque l'on utilise des images de personnes pour ce type d'application.
Cette question se pose d'autant plus avec l'entrée en vigueur le 25 mai 2018 de la RGPD[38] et la protection des données personnelles.

Une autre question qui s'est soulevée au cours du stage est notre **dépendance** à certaines entreprises. En effet, comme nous avons pu le remarquer, en terme de technologie, de recherche, de communauté, Google et Facebook ont toujours sous-tendu ce stage.
Google en particulier, avec MobileNet, TensorFlow,... Ce mouvement de concentration auprès de quelques groupes est particulièrement flagrant depuis quelques années et notamment dans le Deep Learning. Ceci doit nous interroger sur la dépendance de cette nouvelle technologie prometteuse et bientôt omniprésente, à certaines grandes entreprises aux intérêts particuliers. Ceci sera la conclusion de ce rapport de stage.

[38] Règlement Général sur la Protection des Données

Conclusions personnelles

Synthèse du stage

Ce stage chez Cliris m'a donc permis comme nous l'avons vu de lui ouvrir un travail sur le Deep Learning. Dans ce rapport, j'ai voulu retranscrire au mieux ma réflexion de manière chronologique pour justifier ma démarche. Ces étapes sont présentées dans le Gantt ci-dessous :

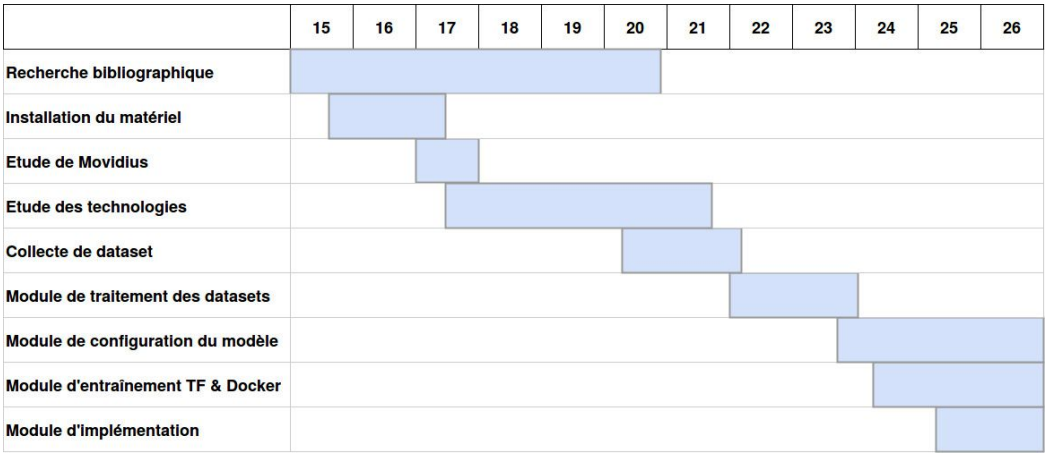


Fig 22 - Gantt du travail effectué

Dans une première partie, une recherche bibliographique qui m'a aidé à tracer une certaine méthodologie. Dans ce sens, une étude des technologies a été effectuée, ce qui m'a permis de comprendre les technologies utilisées dans le domaine et faire un choix stratégique, notamment, du framework TensorFlow et la combinaison MobileNet/SSD.
La seconde partie a été portée sur l'implémentation de tout une chaîne de traitement automatique pour obtenir un modèle entraîné et réaliser un POC de détection de tête à base des techniques de Deep Learning. C'est une partie qui a été réalisée assez rapidement grâce à une planification et des outils efficaces.
Enfin, en marge du travail de stage, j'ai pu également assisté à des conférences comme celle organisée par Matlab ou bien l'exposition Vivatech Paris pour m'ouvrir à d'autres méthodes et en apprendre plus. De plus j'ai pu assister à des entretiens pour le futur thésard de Cliris, qui reprendra mon travail de stage.

Mon retour d'expérience

Ce stage m'a permis de développer un vrai goût pour l'Intelligence Artificielle, avec un réel intérêt à poursuivre dans ce domaine.
De plus, j'ai également développé un goût pour la recherche via la lecture d'article scientifique, de compte rendu et de discussion avec mon maître de stage.
Enfin, bien sûr, ce stage m'a permis de mettre en place les techniques de l'ingénieur, que j'ai pu acquérir au cours de ces 4 dernières années à l'INSA Centre Val de Loire.

Tables des illustrations

Bibliographie

[1] BENENSON, Rodrigo, MOHAMED OMRAN, , HENDRIK HOSANG, Jan y BERNT SCHIELE, . "Ten Years of Pedestrian Detection, What Have We Learned?". CoRR. 2014, vol abs/1411.4304

[2] VU, Tuan-Hung, ANTON OSOKIN, y IVAN LAPTEV, . "Context-aware CNNs for person head detection". CoRR. 2015, vol abs/1511.07917

[3] HUANG, Jonathan, VIVEK RATHOD, , CHEN SUN, , MENGLONG ZHU, , ANOOP KORATTIKARA, , ALIREZA FATHI, , IAN FISCHER, , ZBIGNIEW WOJNA, , YANG SONG, , SERGIO GUADARRAMA, y KEVIN MURPHY, . "Speed/accuracy trade-offs for modern convolutional object detectors". *CoRR*. 2016, vol abs/1611.10012

[4] SHAMI, M., S. MAQBOOL, , H. SAJID, , Y. AYAZ, y . "People Counting in Dense Crowd Images using Sparse Head Detections". *IEEE Transactions on Circuits and Systems for Video Technology*. 2018, p. 1-1.

[5] REDMON, Joseph y ALI FARHADI, . "YOLO9000: Better, Faster, Stronger". *CoRR*. 2016, vol abs/1612.08242

[6] LOFFE Sergey, SZEGEDY Chrisitan, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. CoRR. 2015, vol abs/1502.03167

[7] G. HOWARD, Andrew, MENGLONG ZHU, , BO CHEN, , DMITRY KALENICHENKO, , WEIJUN WANG, , TOBIAS WEY, , , MARCO ANDREETTO, y HARTWIG ADAM, . "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". *CoRR*. 2017, vol abs/1704.04861.

[8] LIU, ANGUELOV, ERHAN, SZEGEDY, REED, FU, BERG, . “SSD: Single Shot Multibox Detector”. CoRR. 2015, vol abs/1512.02325

[9] REN, Shaoqing, KAIMING HE, , B. GIRSHICK, Ross y JIAN SUN, . "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". *CoRR*. 2015, vol abs/1506.01497

[10] DAI, LI, HE, SUN, . “R-FCN: Object Detection via Region-based Fully Convolutional Network”. CoRR vol abs/1605.06409

[11] SZEGEDY, IOFFE, VANHOUCKE, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”, CoRR vol abs/1602.07261

[12] SZEGEDY, IOFFE, VANHOUCKE, SHLENS, “Rethinking the inception architecture for Computer Vision”, CoRR vol abs/1512.00567

[13] HE, ZHANG, REN, SUN, . “Deep residual learning for image recognition”, CoRR vol abs/1512.03385

[14] SIMONYAN, ZISSERMAN, . “Very deep convolutional networks for large-scale image recognition”, CoRR vol abs/1409.1556

[15] VIOLA, JONES, . “Rapid Object Detection using a Boosted Cascade of Simple Features”, CVPR

[16] Ojala T, Pietikïnen M & Harwood D (1996) A comparative study of texture measures a with classification based on feature distributions. Pattern Recognition 29: 51–59

[17] DOLLAR, TU, PERONA, BELONGIE, . “Integral Channel Features”, BVMC 2009

[18] HORN, SCHUNCK, . “Determining Optical Flow”, Artificial Intelligence 17 185-203, 1981

[19] MOKHTARIAN, SUOMELA, . “Robust Image Corner through Curvature Scale Space”, IEEE transactions on pattern analysis and machine intelligence, vol. 20, no. 12, 1998.

[20] YANG, YAN, LEI, LI, . “Aggregate Channel Features for Multi-view Face Detection”, CoRR vol abs/1407.4023

[21] DALAL, TRIGGS, . “Histograms of Oriented Gradients for human detection”, CVPR 2005

[22] Y. Tian, P. Luo, X. Wang, and X. Tang, “Deep learning strong parts for pedestrian detection,” in ICCV, 2015.

[23] X. Du, M. El-Khamy, V. Morariu, J. Lee, L. Davis,. “Fused Deep Neural Networks for efficient pedestrian Detection”, CoRR vol abs/1805.08688, 2018

[24] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, “Encoder-Decoder with atrous separable convolution for semantic image segmenation”, CoRR vol abs/1802.02611, 2018.

[25] https://pjreddie.com/darknet/

[26] http://caffe.berkeleyvision.org/

[27] https://www.tensorflow.org/

[28] http://deeplearning.net/software/theano/

[29] https://pytorch.org/

[30] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in CVPR, 2005.

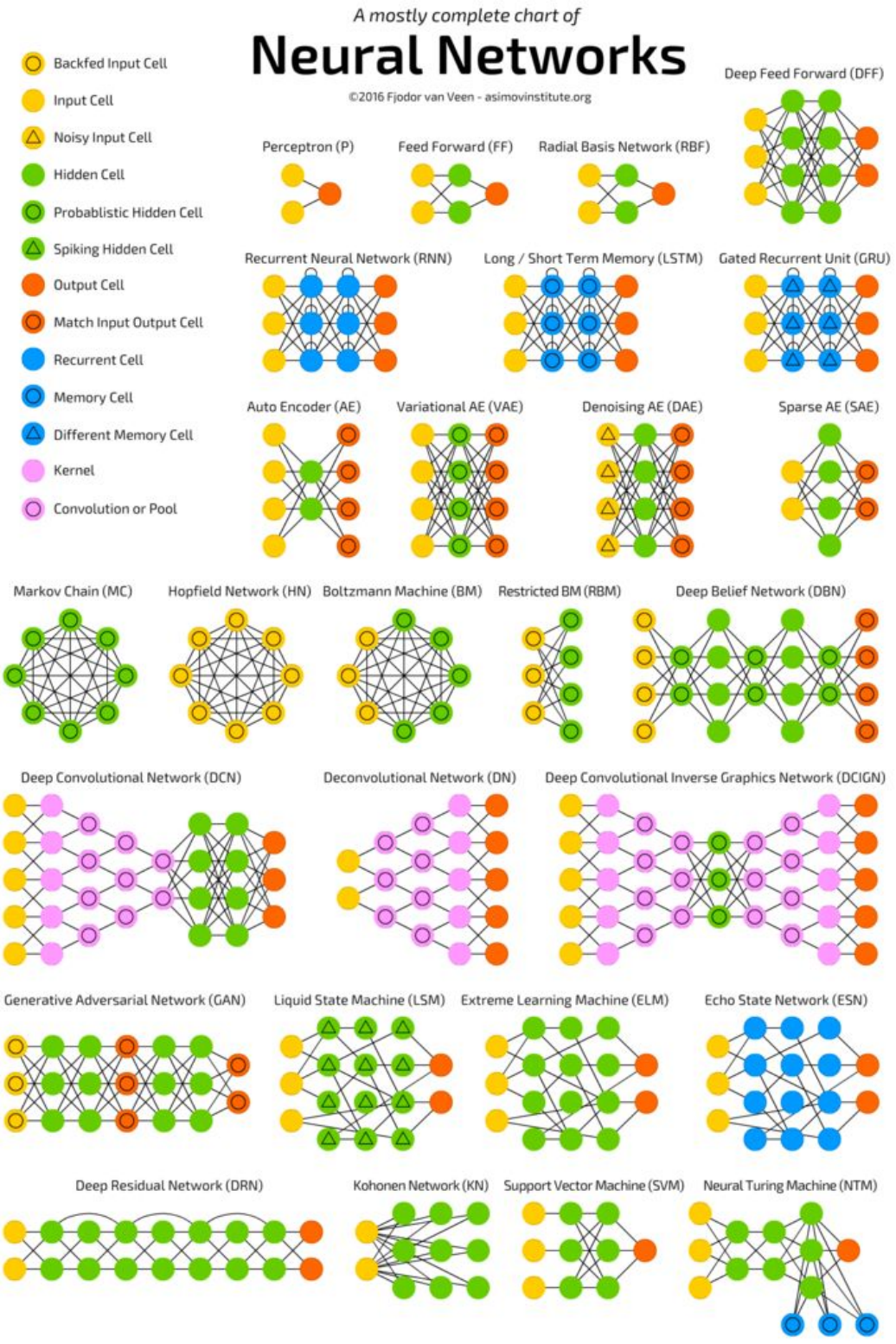
[31] P. Dollár, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: An evaluation of the state of the art,” PAMI, vol. 99, 2011.

[32] P. Viola, M. Jones, and D. Snow, “Detecting pedestrians using patterns of motion and appearance,” IJCV, vol. 63(2), 2005.

[33] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in CVPR, 2001.
[34] https://keras.io/
[35] https://resources.github.com/downloads/development-workflows-data-scientists.pdf
[36] M. Everingham, L.V. Gool, C. Williams, J. Winn, A. Zisserman, . “The PASCAL Visual Object Challenge”, Springer 2009
[37] P. Dollar, T. Lin, M. Maire, S. Belongie, L. Bourdev, . “Microsoft COCO : Common Objects in Context”, vol abs/1405.0312, 2015
[38] J. Deng, W. Dong, R. Socher, K. Li, L. Fei-Fei, . “ImageNet : A large-scale hierarchical image database”, CVPR 2009
[39] M. Taiana, J. Nascimento, A. Bernardino, . “An improved labelling for the INRIA Person Data set for pedestrian detection”, Springer 2013
[40] N. Zhang, M. Paluri, Y. Taigman, R. Fergus, L. Bourdev, “Beyond frontal faces: Improving person recognition using multiple cues”, vol abs/1501.05703, 2015
[41] Muñoz-Salinas R, Yeguas E., Saffiotti A., Medina-Carnicer R., "Multi-Camera Head Pose Estimation", Machine Vision and Applications, Volume 23, Number 3 (2012), 479-490.
[42] B. Boser, I. Guyon, V. Vapnik,. “A training algorithm for optimal margin classifiers”, COLT '92 Proceedings of the fifth annual workshop on Computational learning theory Pages 144-152, 1992
[43] Y. Freund, R. Schapire,. “Experiments with a new boosting algorithm” AT&T research, 1996
[44] K. Kawaguchi, L. Kealbling, Y. Bengio,. “Generalization in Deep Learning”, vol abs/1710.05468v3 2018

Annexes

A. Différent type d'architecture



towarddatascience.com

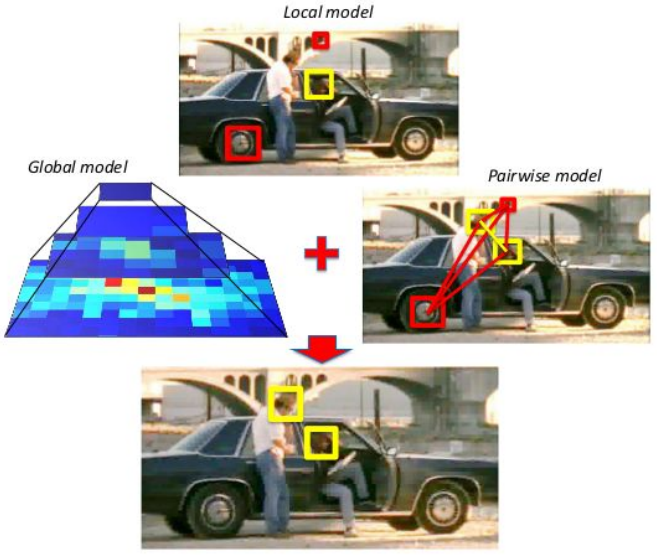
B. Résultats de l'article "Ten years of pedestrian…"

Method	MR	Family	Features	Classifier	Context	Deep	Parts	M-Scales	More dat	Feat. typ	Training
VJ [9]	94.73%	DF	✓	✓						Haar	I
Shapelet [10]	91.37%	-	✓							Gradients	I
PoseInv [11]	86.32%	-					✓			HOG	I+
LatSvm-V1 [12]	79.78%	DPM					✓			HOG	P
ConvNet [13]	77.20%	DN				✓				Pixels	I
FtrMine [14]	74.42%	DF	✓							HOG+Color	I
HikSvm [15]	73.39%	-		✓						HOG	I
HOG [1]	68.46%	-	✓	✓						HOG	I
MultiFtr [16]	68.26%	DF	✓	✓						HOG+Haar	I
HogLbp [17]	67.77%	-	✓							HOG+LBP	I
AFS+Geo [18]	66.76%	-			✓					Custom	I
AFS [18]	65.38%	-								Custom	I
LatSvm-V2 [19]	63.26%	DPM		✓			✓			HOG	I
Pls [20]	62.10%	-	✓	✓						Custom	I
MLS [21]	61.03%	DF	✓							HOG	I
MultiFtr+CSS [22]	60.89%	DF	✓							Many	T
FeatSynth [23]	60.16%	-	✓	✓						Custom	I
pAUCBoost [24]	59.66%	DF	✓	✓						HOG+COV	I
FPDW [25]	57.40%	DF								HOG+LUV	I
ChnFtrs [26]	56.34%	DF	✓	✓						HOG+LUV	I
CrossTalk [27]	53.88%	DF			✓					HOG+LUV	I
DBN-Isol [28]	53.14%	DN					✓			HOG	I
ACF [29]	51.36%	DF	✓							HOG+LUV	I
RandForest [30]	51.17%	DF		✓						HOG+LBP	I&C
MultiFtr+Motion [22]	50.88%	DF	✓						✓	Many+Flow	T
SquaresChnFtrs [31]	50.17%	DF	✓							HOG+LUV	I
Franken [32]	48.68%	DF		✓						HOG+LUV	I
MultiResC [33]	48.45%	DPM			✓		✓	✓		HOG	C
Roerei [31]	48.35%	DF	✓					✓		HOG+LUV	I
DBN-Mut [34]	48.22%	DN			✓		✓			HOG	C
MF+Motion+2Ped [35]	46.44%	DF			✓				✓	Many+Flow	I+
MOCO [36]	45.53%	-		✓	✓					HOG+LBP	C
MultiSDP [37]	45.39%	DN		✓	✓		✓			HOG+CSS	C
ACF-Caltech [29]	44.22%	DF		✓						HOG+LUV	C
MultiResC+2Ped [35]	43.42%	DPM								HOG	C+
WordChannels [38]	42.30%	DF		✓			✓		✓	Many	C
MT-DPM [39]	40.54%	DPM					✓		✓	HOG	C
JointDeep [40]	39.32%	DN			✓					Color+Gradient	C
SDN [41]	37.87%	DN				✓	✓			Pixels	C
MT-DPM+Context [39]	37.64%	DPM			✓		✓		✓	HOG	C+
ACF+SDt [42]	37.34%	DF	✓						✓	ACF+Flow	C+
SquaresChnFtrs [31]	34.81%	DF	✓							HOG+LUV	C
InformedHaar [43]	34.60%	DF	✓							HOG+LUV	C
Katamari-v1	22.49%	DF	✓		✓				✓	HOG+Flow	C+

Table 1: Listing of methods considered on Caltech-USA, sorted by log-average miss-rate (lower is better). Consult sections 3.1 to 3.9 for details of each column. See also matching figure 3. "HOG" indicates HOG-like [1]. Ticks indicate salient aspects of each method.

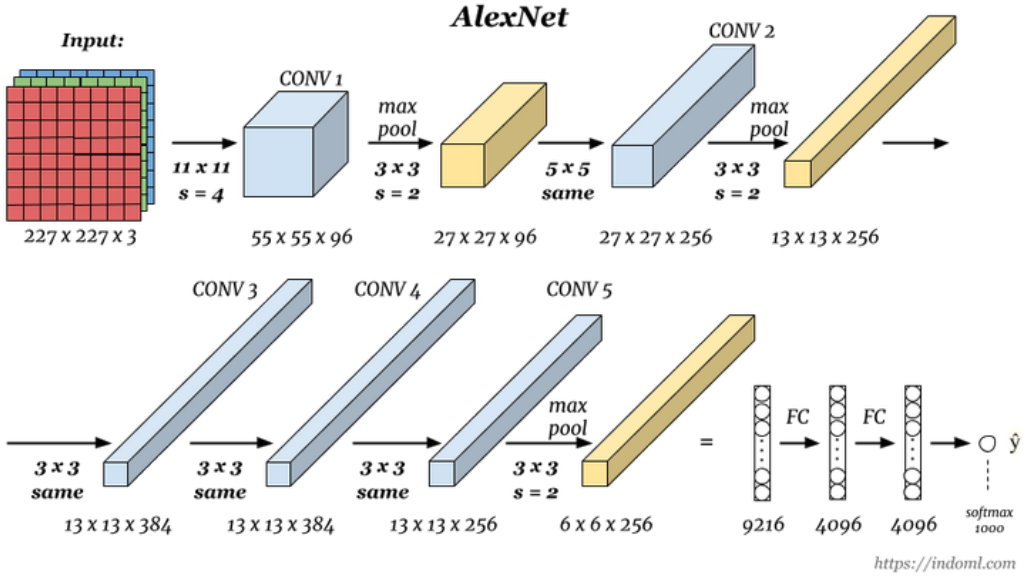
"Ten years of pedestrian detection, what have we learned ?"

C. Visualisation des résultats des différents modèles



"Context-aware CNNs for person head detection"

D. Alexnet pour le modèle local

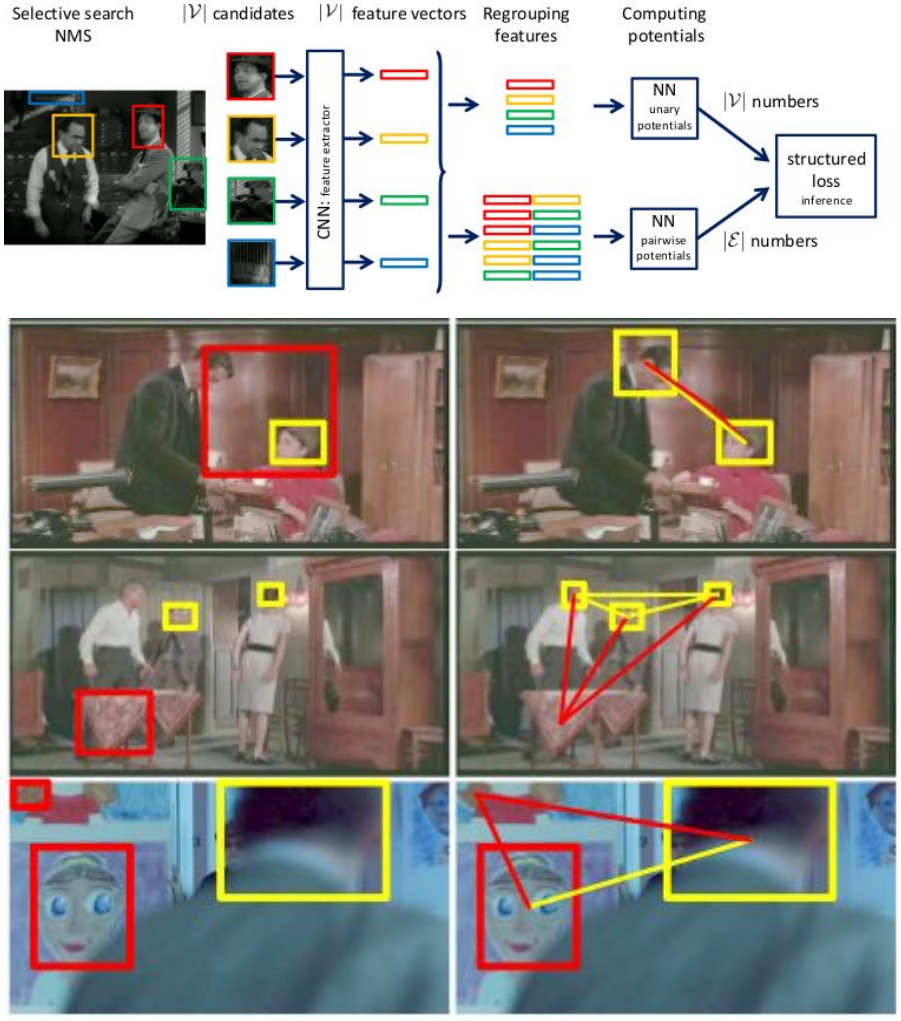


“Context-aware CNNs for person head detection”

E. Le modèle global

“Context-aware CNNs for person head detection”

F. Le modèle pairwise

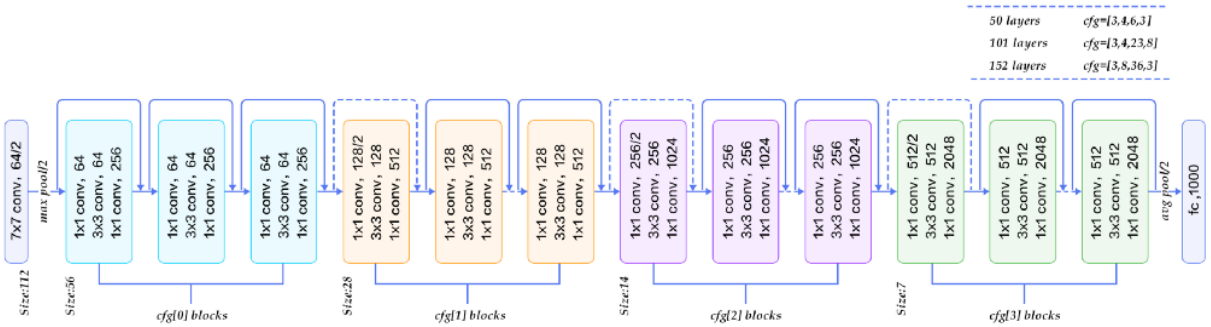


"Context-aware CNNs for person head detection"

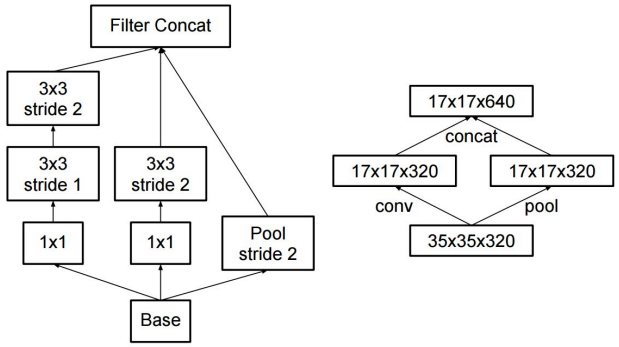
G. Les modèles d'extracteur de caractéristique

VGG-16

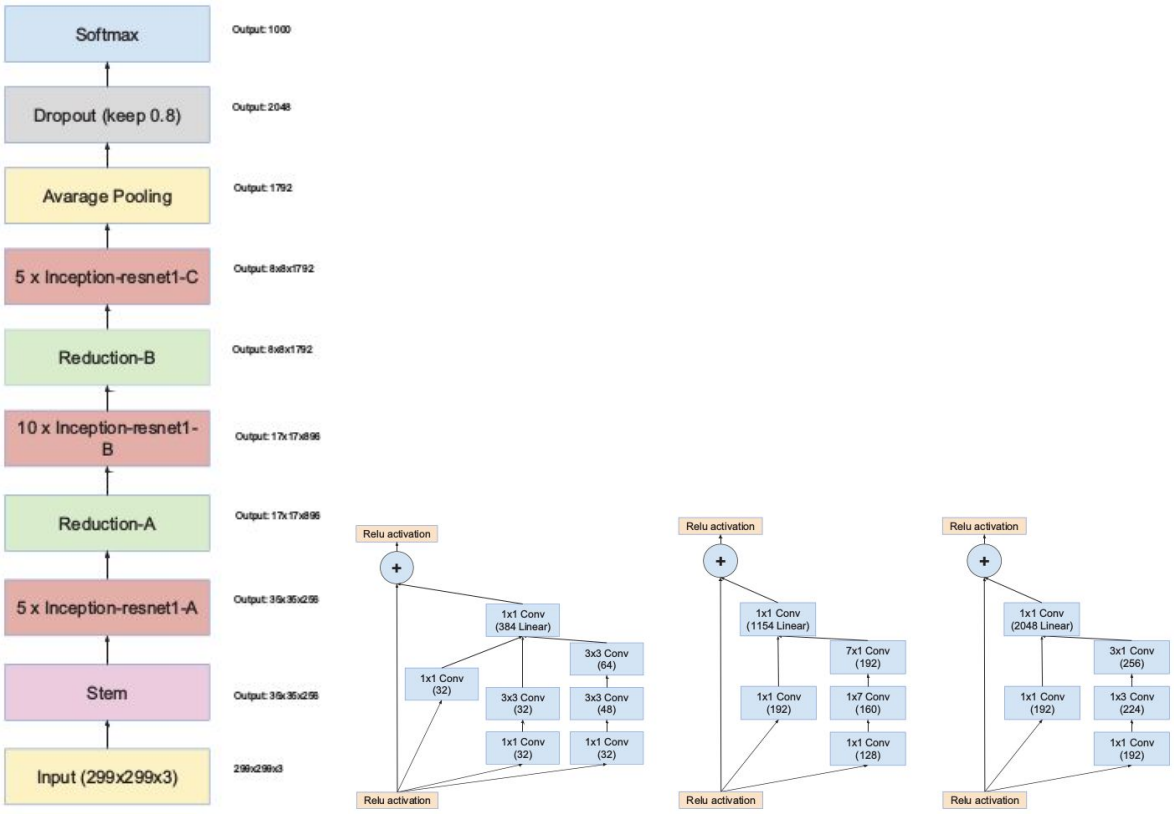
Resnet-101



Inception v2



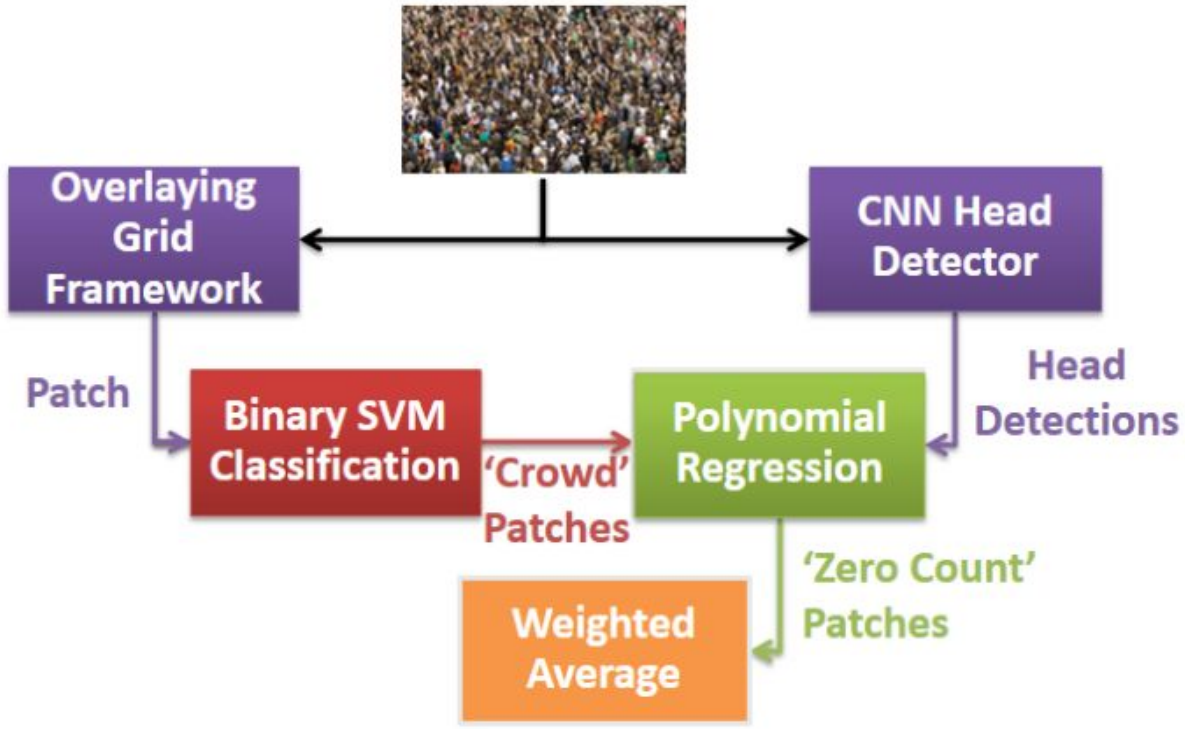
Inception Resnet



MobileNet

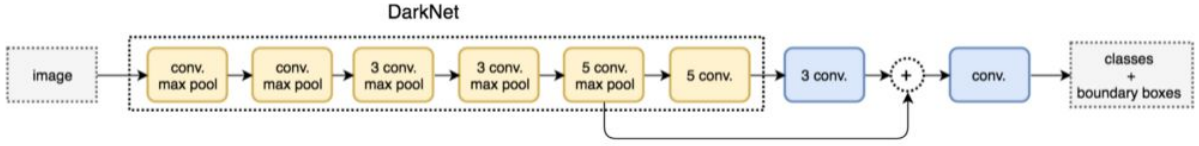
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

H. L'algorithme d'estimation du nombre de tête dans une foule

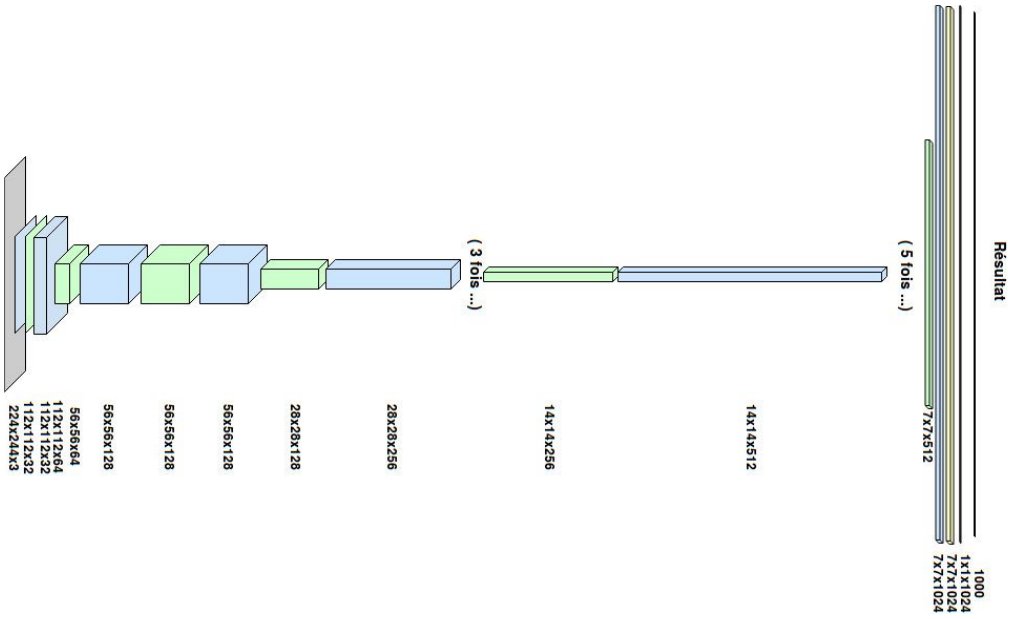


"People counting in dense crowd images using sparse head detections"

I. L'extracteur de caractéristique Darknet-19



J. Modèle MobileNet



K. Résultats d'expérimentation de mobileNet

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

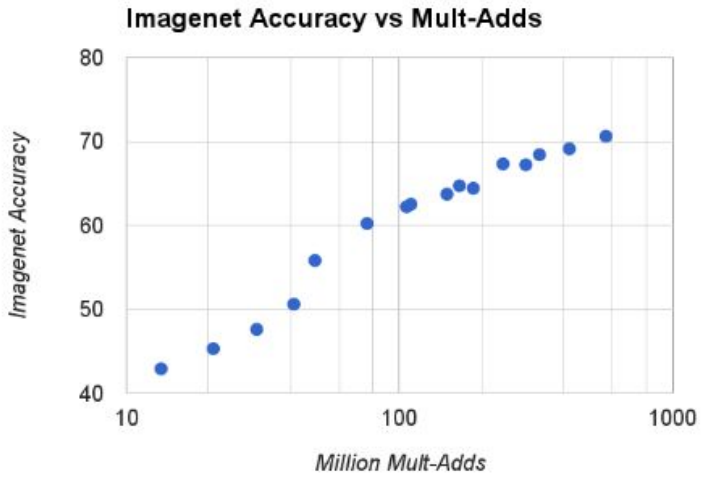
Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

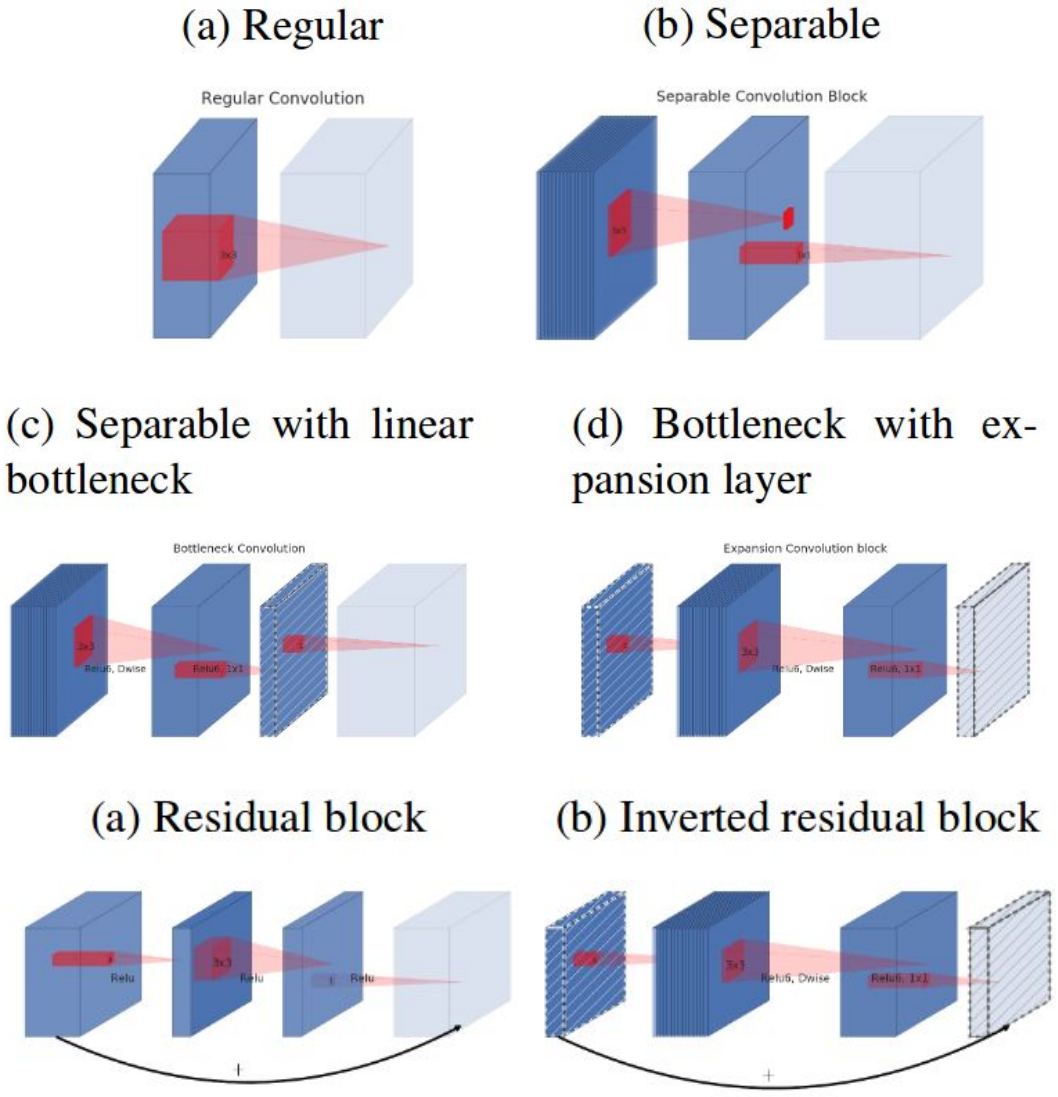
Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"



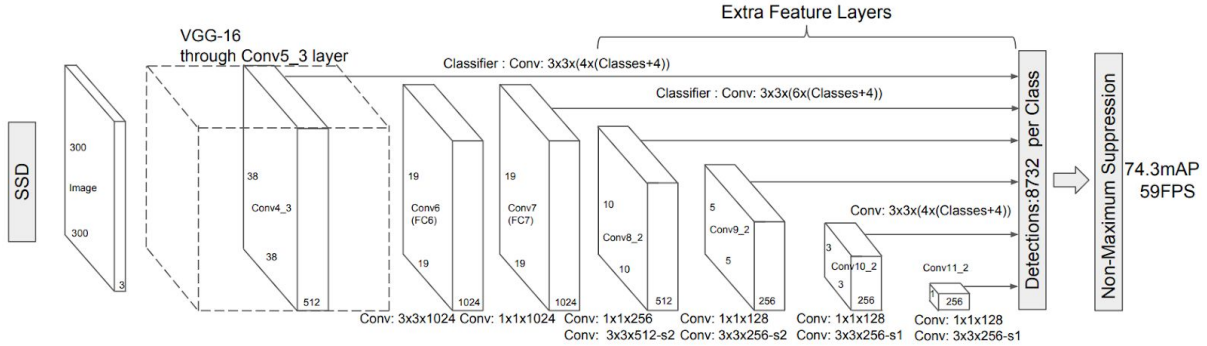
"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"

L. Méthodes de convolution pour MobileNet v2



"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"

M. Architecture du modèle SSD



"SSD: Single Shot MultiBox Detector"

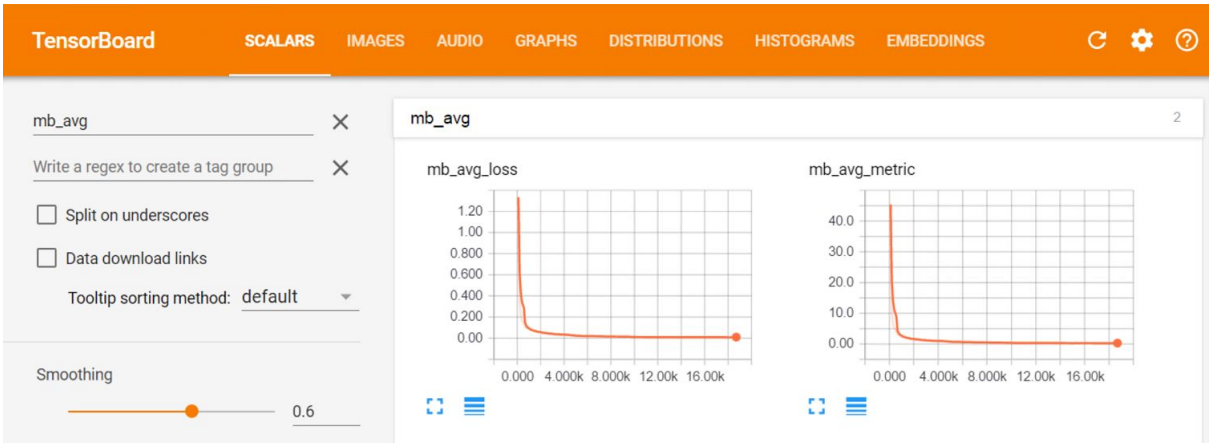
N. Exemple d'entraînement caffe

```
name: "convolution"
input: "data"
input_dim: 1
input_dim: 1
input_dim: 100
input_dim: 100
layer {
	name: "conv"
	type: "Convolution"
	bottom: "data"
	top: "conv"
	convolution_param {
		num_output: 3
		kernel_size: 5
		stride: 1
		weight_filler {
			type: "gaussian"
			std: 0.01
		}
		bias_filler {
			type: "constant"
			value: 0
		}
	}
}
```

```
./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt
```

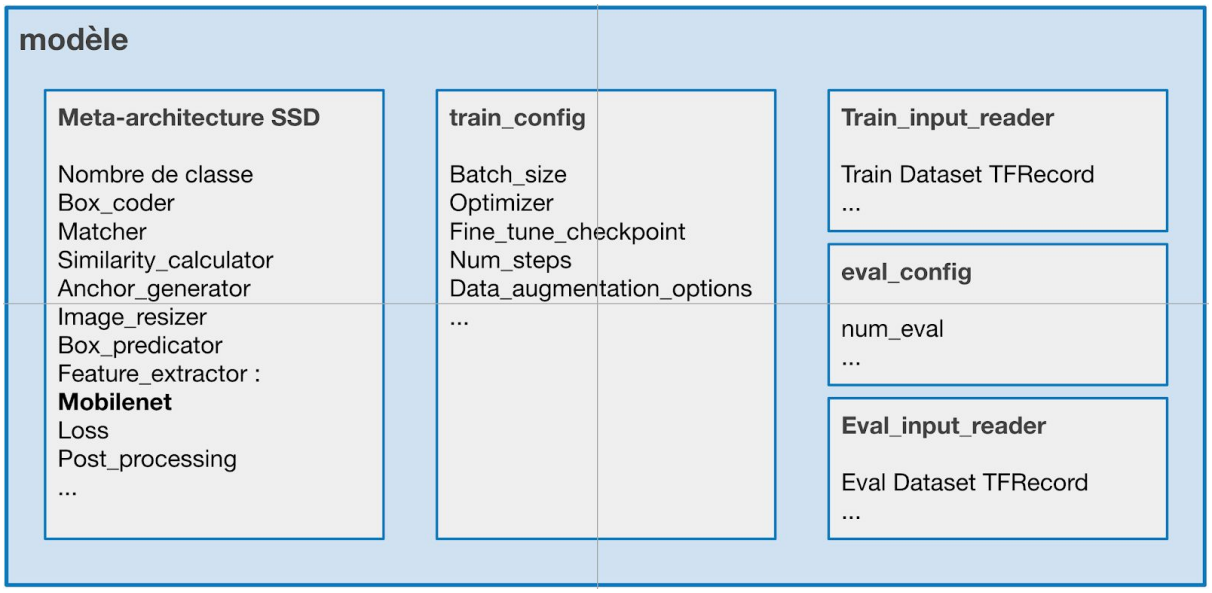
```
I0502 15:34:34.262904 22660 solver.cpp:239] Iteration 9900 (103.201 iter/s, 0.968982s/100 iters), loss = 0.00885846
I0502 15:34:34.262948 22660 solver.cpp:258]     Train net output #0: loss = 0.00885863 (* 1 = 0.00885863 loss)
I0502 15:34:34.262954 22660 sgd_solver.cpp:112] Iteration 9900, lr = 0.00596843
I0502 15:34:35.212528 22660 solver.cpp:468] Snapshotting to binary proto file examples/mnist/lenet/lenet_solver_iter_10000.caffemodel
I0502 15:34:35.219802 22660 sgd_solver.cpp:280] Snapshotting solver state to binary proto file examples/mnist/lenet/lenet_solver_iter_10000.solverstate
I0502 15:34:35.223816 22660 solver.cpp:331] Iteration 10000, loss = 0.00269747
I0502 15:34:35.223837 22660 solver.cpp:351] Iteration 10000, Testing net (#0)
I0502 15:34:35.695889 22666 data_layer.cpp:73] Restarting data prefetching from start.
I0502 15:34:35.714939 22660 solver.cpp:418]     Test net output #0: accuracy = 0.9903
I0502 15:34:35.714964 22660 solver.cpp:418]     Test net output #1: loss = 0.0283635 (* 1 = 0.0283635 loss)
I0502 15:34:35.714979 22660 solver.cpp:336] Optimization Done.
I0502 15:34:35.714983 22660 caffe.cpp:250] Optimization Done.
```

O. Tensorboard



P. Configuration d'un modèle de l'API de détection d'objet de TensorFlow

```
model {
 ssd {
   num_classes: 1
   box_coder {
    faster_rcnn_box_coder {
      ...     }
   }
   matcher {
    argmax_matcher {
      matched_threshold: 0.5
      ...     }
   }
   similarity_calculator {
    iou_similarity {
    }
   }
   anchor_generator {
    ssd_anchor_generator {
      num_layers: 6
      ...     }
   }
   image_resizer {
    fixed_shape_resizer {
      height: 300
      width: 300
    }
   }
   box_predictor {
    convolutional_box_predictor {
...
      conv_hyperparams {
       activation: RELU_6,
       ...
```

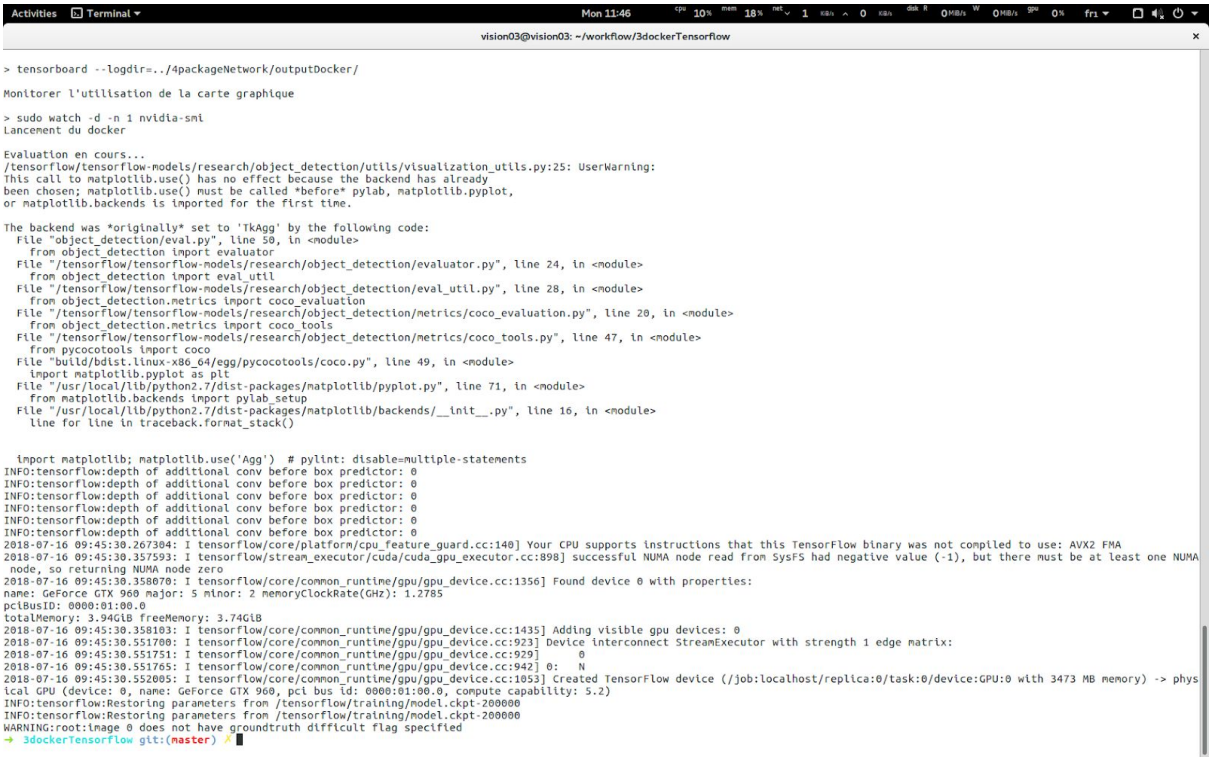


Q. Dockerfile

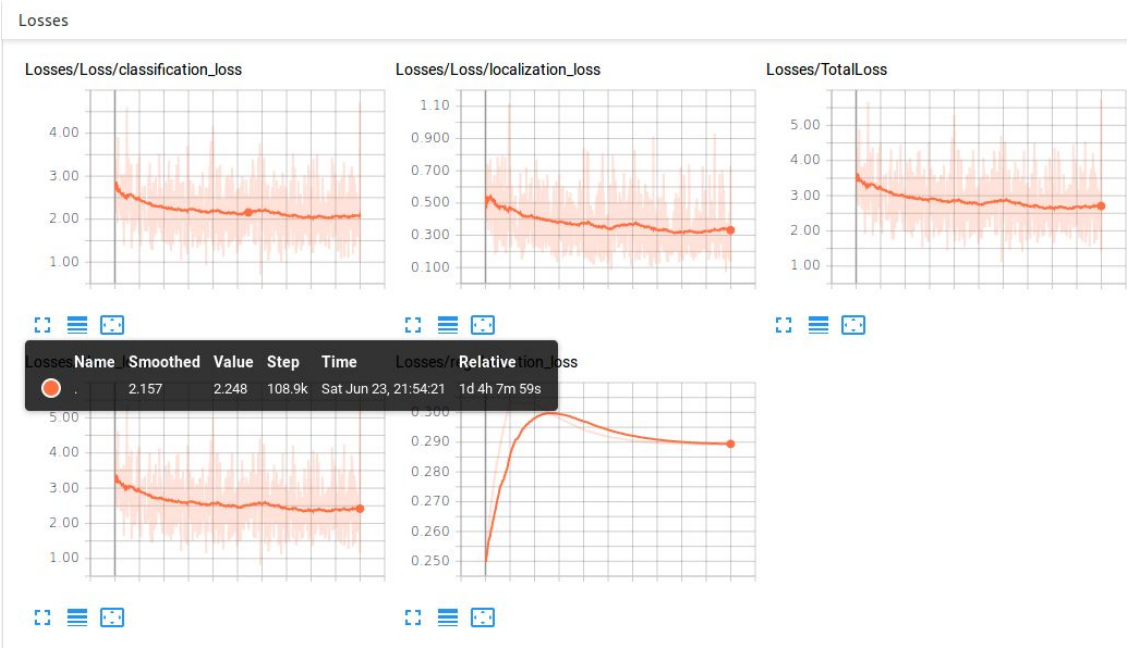
```
FROM nvidia/cuda:9.0-base-ubuntu16.04

LABEL maintainer="Flavien Ronteix--Jacquet
<flavien.ronteix-jacquet@clirisgroup.com>"

# Dependances tensorflow
RUN apt-get update && apt-get install -y --no-install-recommends \
       build-essential \
       cuda-command-line-tools-9-0 \
       cuda-cublas-9-0 \
       cuda-cufft-9-0 \
       cuda-curand-9-0 \
       cuda-cusolver-9-0 \
       cuda-cusparse-9-0 \
       curl \
       libcudnn7=7.1.4.18-1+cuda9.0 \
       libfreetype6-dev \
       libhdf5-serial-dev \
       libpng12-dev \
       libzmq3-dev \
       pkg-config \
       python3 \
       python \
       python-dev \
       python3-dev \
       python-setuptools \
       cython \
       rsync \
       software-properties-common \
       unzip \
       git \
       python-tk \
       libssl-dev \
       libffi-dev \
       libxml2-dev \
       libxslt1-dev \
       zlib1g-dev \
       sudo \
       && \
   apt-get clean && \
   rm -rf /var/lib/apt/lists/*

# Installation de pip
RUN curl -O https://bootstrap.pypa.io/get-pip.py && \
   python get-pip.py && \
   rm get-pip.py
```

```
#     python2 get-pip.py && \

# Installation des packages pip python pour tensorflow
RUN pip --no-cache-dir install \
       Pillow \
       h5py \
       ipykernel \
       jupyter \
       matplotlib \
       numpy \
       pandas \
       scipy \
       sklearn \
       Cython \
       && \
   python -m ipykernel.kernelspec

#  Installation de tensorflow depuis les repos pip, on peut aussi le compiler à
partir des sources
RUN pip install tensorflow-gpu

# For CUDA profiling, TensorFlow requires CUPTI.
ENV LD_LIBRARY_PATH /usr/local/cuda/extras/CUPTI/lib64:$LD_LIBRARY_PATH


# Copie de tous les scripts nécessaires
ADD tensorflow-models /tensorflow/tensorflow-models

# Installation de pycocotools
RUN git clone https://github.com/pdollar/coco.git && \
   cd coco/PythonAPI && \
   python setup.py build_ext --inplace && \
   python setup.py build_ext install && \
   python setup.py install && \
   cp -r pycocotools /tensorflow/tensorflow-models/research/

# Installation de protobuf et de l'API
RUN                                     curl                                    -OL
https://github.com/google/protobuf/releases/download/v3.3.0/protoc-3.3.0-linux-x86_
64.zip
RUN unzip protoc-3.3.0-linux-x86_64.zip -d protoc3 && \
   mv protoc3/bin/* /usr/local/bin/ && \
   mv protoc3/include/* /usr/local/include/ && \
   ln -s /protoc3/bin/protoc /usr/bin/protoc

# Creation des VOLUME où mettre les données
# Le dataset
RUN ["mkdir", "/tensorflow/data"]
# Les paramètres d'entraînement
RUN ["mkdir", "/tensorflow/training"]
# La sortie d'entraînement
RUN ["mkdir", "/tensorflow/output"]

# copie du script principal
COPY entrypoint.sh /
RUN ["chmod", "+x", "/entrypoint.sh"]
ENTRYPOINT ["/entrypoint.sh"]
```

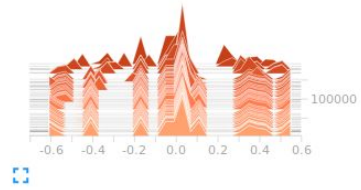
R. Lancement d’une image docker



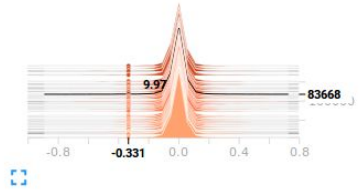
S. Interface web Tensorboard de l’entraînement



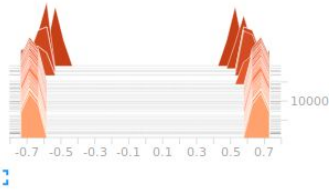
ModelVars/BoxPredictor_0
/BoxEncodingPredictor/biases



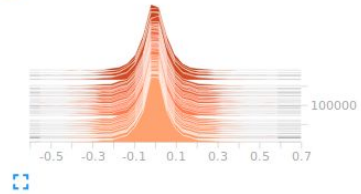
ModelVars/BoxPredictor_0
/BoxEncodingPredictor/weights



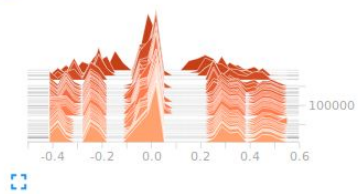
ModelVars/BoxPredictor_0/ClassPredictor
/biases



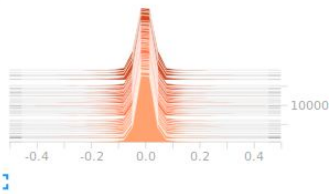
ModelVars/BoxPredictor_0/ClassPredictor
/weights



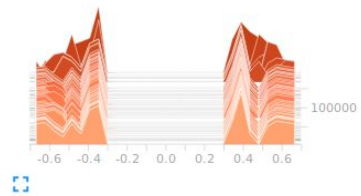
ModelVars/BoxPredictor_1
/BoxEncodingPredictor/biases



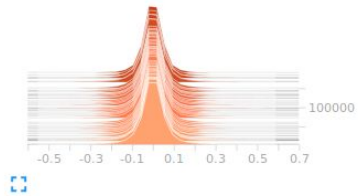
ModelVars/BoxPredictor_1
/BoxEncodingPredictor/weights



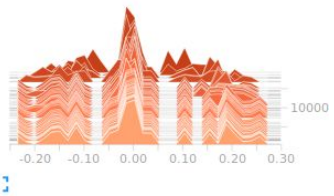
ModelVars/BoxPredictor_1/ClassPredictor
/biases

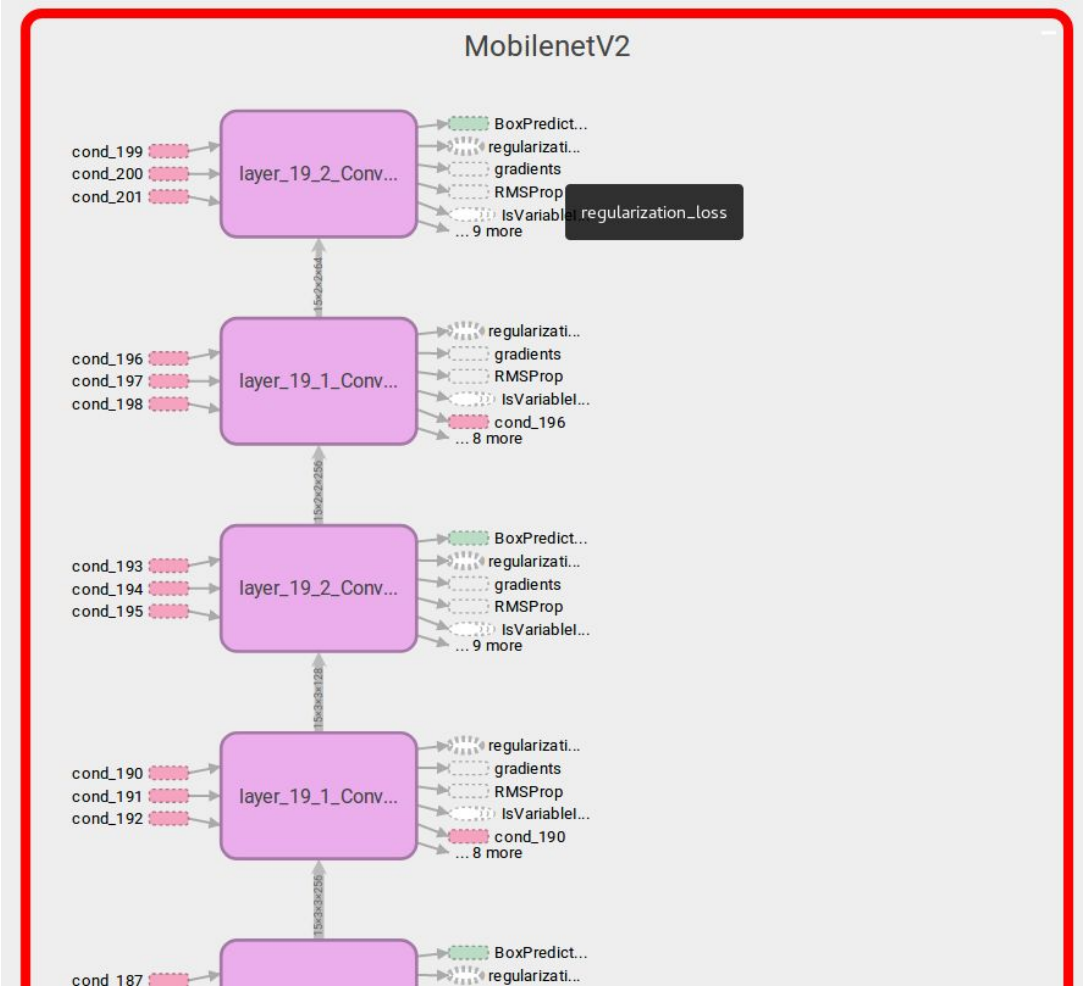


ModelVars/BoxPredictor_1/ClassPredictor
/weights

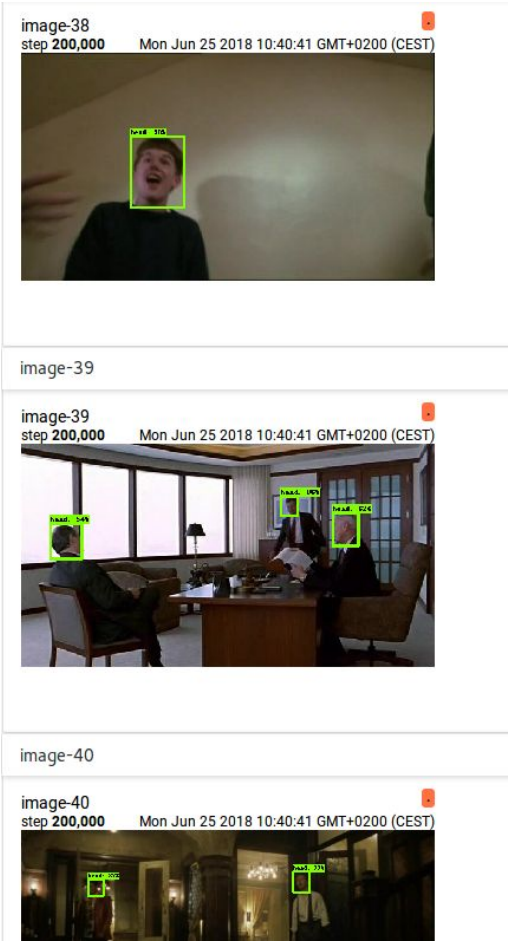


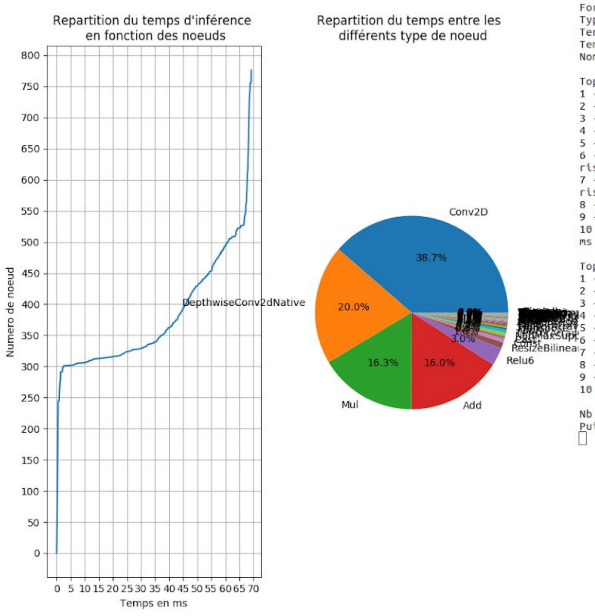
ModelVars/BoxPredictor_2
/BoxEncodingPredictor/biases





T. Visualisation du Benchmark

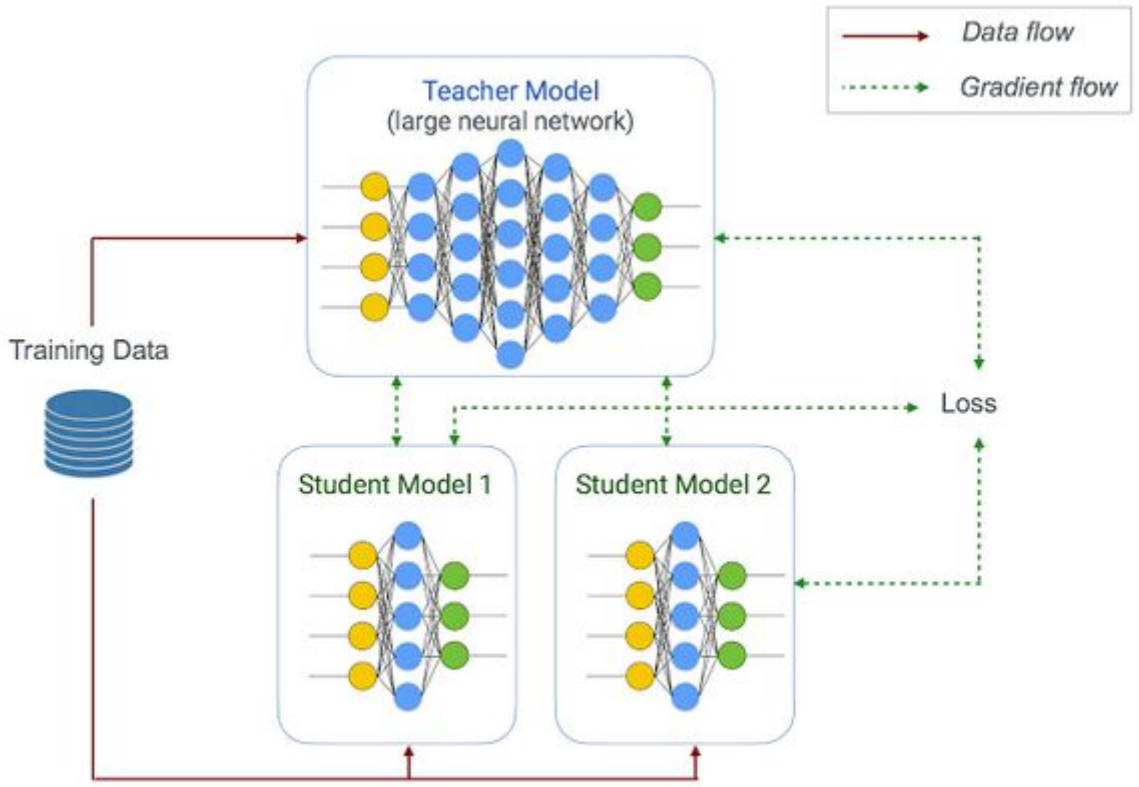





```
→ 4packageNetwork git:(master) python3 make_benchmark_readable_again.py benchmark_report.t
xt
Tenseur d'entrée : image_tensor:0
Format du tenseur d'entrée : 1,224,224,3
Type du tenseur d'entrée : uint8
Tenseur de sortie : num_detections:0
Temps d'inférence : 61.73 ms
Nombre de noeud du graph : 777

Top du temps de calcul pris
1 - Noeud FeatureExtractor/MobilenetV2/Conv/Conv2D a pris 3.011 ms
2 - Noeud FeatureExtractor/MobilenetV2/expanded_conv/depthwise/depthwise a pris 2.846 ms
3 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_2/depthwise/depthwise a pris 2.474 ms
4 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_1/expand/Conv2D a pris 2.434 ms
5 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_1/depthwise/depthwise a pris 2.142 ms
6 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_1/expand/BatchNorm/batchnorm/add_1 a p
ris 1.917 ms
7 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_1/expand/BatchNorm/batchnorm/mul_1 a p
ris 1.890 ms
8 - Noeud FeatureExtractor/MobilenetV2/Conv_1/Conv2D a pris 1.214 ms
9 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_2/expand/Conv2D a pris 1.121 ms
10 - Noeud FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_2_3x3_s2_512/Conv2D a pris 1.105
ms

Top de la mémoire prise
1 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_1/expand/Conv2D utilise 8.64 Mo
2 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_2/depthwise/depthwise utilise 3.63 Mo
3 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_2/expand/Conv2D utilise 3.24 Mo
4 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_3/expand/Conv2D utilise 3.24 Mo
5 - Noeud FeatureExtractor/MobilenetV2/expanded_conv/depthwise/depthwise utilise 2.94 Mo
6 - Noeud FeatureExtractor/MobilenetV2/Conv/Conv2D utilise 2.88 Mo
7 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_1/depthwise/depthwise utilise 2.23 Mo
8 - Noeud FeatureExtractor/MobilenetV2/expanded_conv/project/Conv2D utilise 1.44 Mo
9 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_5/depthwise/depthwise utilise 1.24 Mo
10 - Noeud FeatureExtractor/MobilenetV2/expanded_conv_4/depthwise/depthwise utilise 1.24 Mo

Nb de calcul pour le graph d'inférence : 1.28 GFlops
Puissance de la machine : 20.74 GFlops/s
```

U. Visualisation du Benchmark

V. Learn2Compress par Distillation



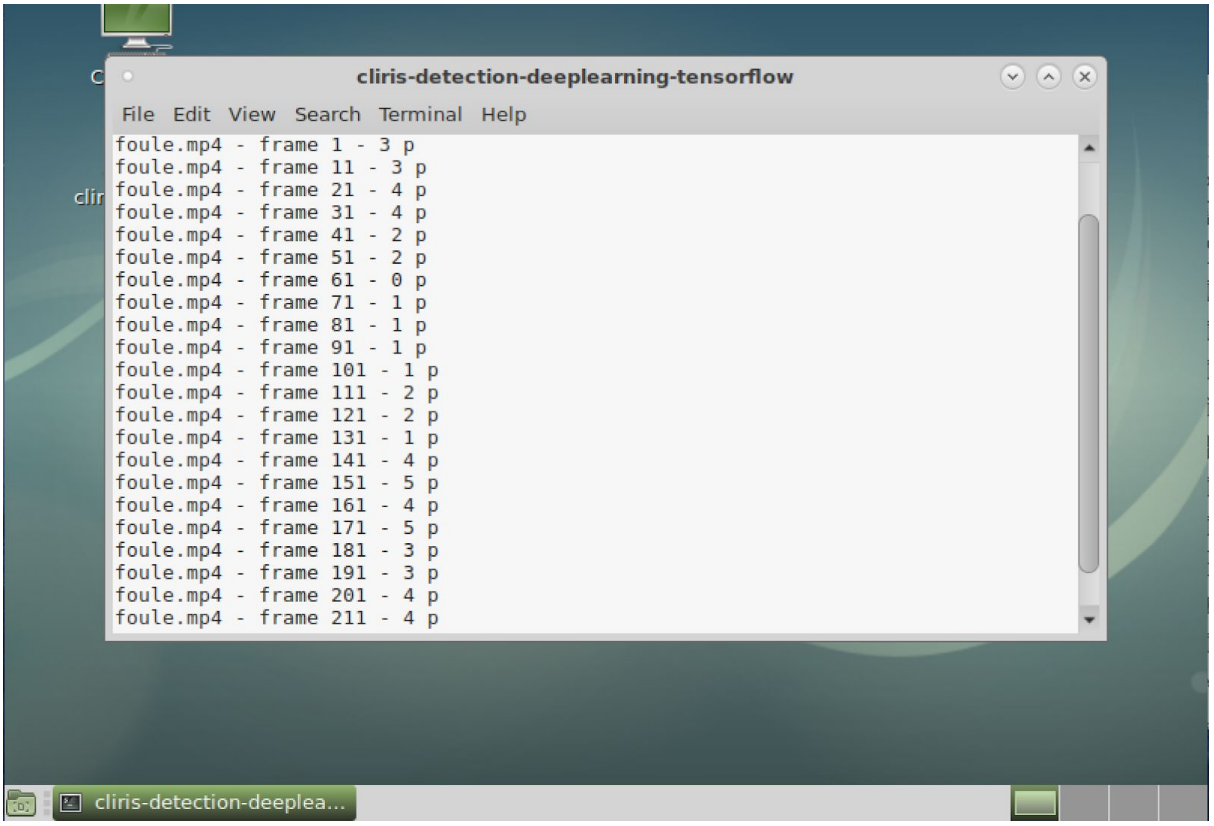
Joint training and distillation approach to learn compact student models.

Tensorflow.org

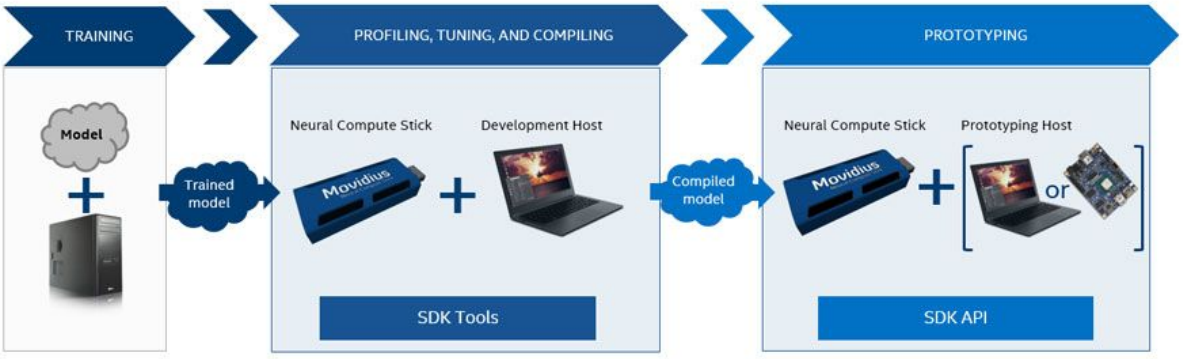
W. Algorithme de détection

```
Algorithme detection:
	Graph = ouvrir_graph(frozen_graph)
	demarrer_session():
		charger(graph)
		demarrer_ecriture_evenement(chemin_evenement)
		flux = ouvrir_flux_video(chemin_flux)
		tant_que (flux):
			frame = obtenir_image(flux)
			entree = recadrer(frame)
			resultats = inference(graph, entree)
			comptage_tete(resulats)
		fin tant_que
	fin_session
fin_algorithme
```

X. Capture d’écran de la machine virtuelle



Y. Fonctionnement de movidius



movidius.com

Z. Evolution des performances de détection et de classification