Université Mohammed V
de Rabat

Projet fin d'année
Conception et développement d'une application de suivi de mouvement dans une vidéo

Encadré par :
Pr. OULAD HAJ THAMI Rachid

Réalisé par :
BOUTINZER Mouhcine
OUOKKI Mohamed

Jury :
Mr. OULAD HAJ THAMI Rachid
Mme. ABIK Mounia

Année universitaire : 2019/2020

REMERCIEMENTS

Ce travail est l'aboutissement d'un dur labeur et de beaucoup de sacrifices ; nos remerciements vont d'abord à nos familles. Je souhaite également remercier toute la communauté IT, qui ont bien voulu répondre à nos différentes questions sur le domaine dans le but de permettre la facilité de l'élaboration de ce travail. Un remerciement spécial à notre encadrant Professeur OULAD HAJ THAMI Rachid qui nous a fournit les informations necéssaires pour pouvoir accomplir ce travail . En dernier lieu je remercie l'administration et le corps enseignant de l'ENSIAS qui à travers leur programme nous ont fourni des outils de qualité facilitant notre spécialisation.

Resumé

La détection et le suivi des objets en mouvement sont la tâche la plus importante et la plus difficile des applications de vidéosurveillance et de vision par ordinateur. La détection d'objets est la procédure de recherche des entités non stationnaires dans les séquences d'images. La détection est la première étape vers le suivi de l'objet en mouvement dans la vidéo. La représentation des objets est la prochaine étape importante à suivre. Le suivi est la méthode d'identification de la position de l'objet en mouvement dans la vidéo. Identifier la position est une tâche beaucoup plus difficile que de détecter l'objet en mouvement dans une vidéo. Le suivi d'objets est appliqué dans de nombreuses applications comme la vision robotique, la surveillance du trafic, la vidéosurveillance, la peinture vidéo et la simulation.

Ici, nous allons réalisé une application de suivi de mouvement dans une vidéo.

Abstract

Detecting and tracking moving objects is the most important and difficult task of video surveillance and computer vision applications. Object detection is the procedure for searching for non-stationary entities in image sequences. Detection is the first step towards tracking the moving object in the video. Representing objects is the next important step to follow. Tracking is the method of identifying the position of the moving object in the video. Identifying the position is a much more difficult task than detecting the moving object in a video. Object tracking is applied in many applications such as robotic vision, traffic monitoring, video surveillance, video painting and simulation.

Here we are going to make a motion tracking application in a video.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

L'objectif du projet est de mettre en œuvre un environnement d'apprentissage profond dans une infrastructure de calcul et de modélisation, qui doit être dynamique et sujette aux erreurs. À partir de là, nous avons implémenté un modèle YOLOv3[1] pour la détection d'objets en temps réel.

Nous avons décidé de ne pas partir du zéro, mais utiliser certains modèles communautaires déjà mis en œuvre. Notre objectif n'était pas d'obtenir les meilleurs résultats, mais un modèle qui fonctionne, dont le résultat est logique. En outre, l'une des fortes dépendances cachées du Deep Learning sont les ensembles de données, qui sont presque toujours fournis et vérifiés par les hôtes du défi, mais, en les téléchargeant et en les gérant, vérifier leur précision effective n'a pas été aussi simple que tout le monde peut le penser.

2 Contexte général

Nous expliquerons d'abord la détection d'objets, le suivi d'un mouvement et la différence entre les deux. Ainsi qu'introduire quelques applications dans le domaine du suivi d'un mouvement. Il est possible de trouver quelques connaissances de base sur le papier YOLOv3 dans l'annexe. puis la méthodologie que nous avons utilisée pour atteindre cet objectif.

Pour réaliser les objectives de ce projet, nous avons utilisé un tas d'outils, que nous allons introduire dans cette section, sous python, pour implémenter notre architecture, notamment : Keras[2] utilisant TensorFlow[3] Backend pour la partie architecture du modèle, et OpenCV[4] et NumPy[5] pour la partie prétraitement.

2.1 À quoi sert le suivi du mouvement et la détection d'un objet

La détection d'objets en mouvement est un problème courant en traitement des images dans différentes applications comme la vidéo surveillance et la surveillance du trafic routier. Plus récemment, la détection d'objets en mouvement a trouvé d'autres domaines d'applications dont certaines au sein du laboratoire L3I. On peut citer entre autres des applications dans le domaine d'interface utilisateur pour réaliser une interaction entre l'utilisateur et des applications de multimédia (E-Learning) et de jeux vidéo (EyeToys), dans la capture de mouvement sans marqueur pour pouvoir reproduire les mouvements détectés sur un personnage de synthèse (MotionCapture), dans la détection et la reconnaissance d'animaux pour des applications ludo-pédagogiques (Aqua@thèque) ou des applications scientifiques. La détection d'objets en mouvement est donc une étape importante pour tout système qui désire réaliser une interaction entre le monde réel et le monde virtuel ou étudier le comportement d'objets (personnes, animaux) à partir d'un flux vidéo.

2.2 Difference entre détection et suivi

2.2.1 La detection d'un objet

La détection est liée à la vision par ordinateur et au traitement d'images qui traite de la détection d'instances d'objets sémantiques d'une certaine classe (comme les humains, les bâtiments ou les voitures) dans les images et vidéos numériques.

2.2.2 Le suivi d'un mouvement

Le suivi des objets consiste à prendre un ensemble initial de détections d'objets, à créer un ID unique pour chacune des détections initiales, puis à suivre chacun des objets lorsqu'ils se déplacent dans les images d'une vidéo, en conservant l'attribution d'ID.

2.3 Applications

— Surveillance du trafic :
il peut être utilisé pour la surveillance et la gestion du trafic sur les routes. surveillance du trafic débit, détection des accidents de la circulation, comptage piéton.
— Robotique industrielle :
il peut être utilisé dans le système de contrôle de la robotique industrielle et humanoïde, par exemple, en utilisant un capteur de vision avec un algorithme de suivi dans la boucle de rétroaction, un robot humanoïde ASIMO, un contrôle visuel pour véhicule aérien sans pilote (UAV).
— Suivi des véhicules :
il peut être utilisé pour l'automobile suivi, par exemple, suivi d'un véhicule par UAV, suivi véhicules sur la route pour assister le conducteur et pilote automatique d'un UGV.
— Jeux vidéo :
il peut être utilisé pour les jeux vidéo fournir un meilleur contrôle de l'utilisateur, par exemple, suivre les mouvements des utilisateurs, suivre le visage pour jouer au jeu.
— Systèmes de diagnostic médical :
il a poussé son importance dans le domaine médical pour le diagnostic de différentes maladies, par exemple, le suivi de la paroi ventriculaire, la reconstruction de la forme des voies vocales.
— Activity recognition :
il peut être utilisé pour la reconnaissance d'activité pour la surveillance intérieure et extérieure, par ex. Modèles d'activité d'apprentissage et activité humaine reconnaissance.

2.4 Outils utilisés

2.4.1 Tensorflow

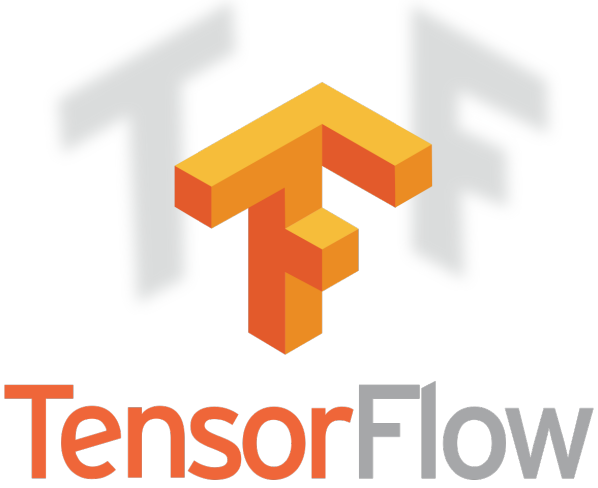

FIGURE 1 – Tensorflow

TensorFlow est un outil open source d'apprentissage automatique développé par Google. Le code source a été ouvert le 9 novembre 2015 par Google et publié sous licence Apache. Il est fondé sur l'infrastructure DistBelief, initiée par Google en 2011, et est doté d'une interface pour Python et Julia.
TensorFlow est l'un des outils les plus utilisés en IA dans le domaine de l'apprentissage machine.

2.4.2 OpenCv

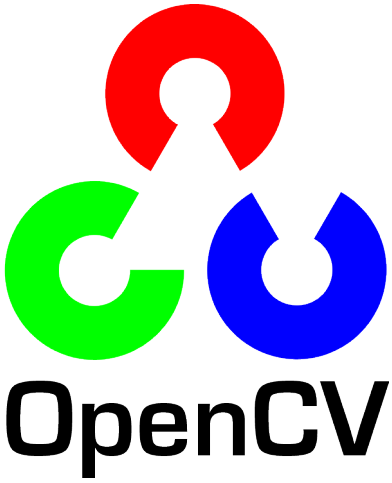

FIGURE 2 – OpenCV

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque. Depuis 2016 et le rachat de ItSeez par Intel, le support est de nouveau assuré par Intel.

2.4.3 Keras

FIGURE 3 – Keras

La bibliothèque Keras permet d'interagir avec les algorithmes de réseaux de neurones profonds et de machine learning, notamment Tensorflow3, Theano, Microsoft Cognitive Toolkit4 ou PlaidML. Conçue pour permettre une expérimentation rapide avec les réseaux de neurones profonds, elle se concentre sur son ergonomie, sa modularité et ses capacites d'extension. Elle a été développée dans le cadre du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System).

3 Problèmatique

Dans cette section, nous parlerons des bases de la détection d'objets. Les concepts seront divisés en ce que l'on appelle le "traitement d'images fixes" qui englobe trois étapes primordiales : Classification, Localisation et Détection. Puis nous aborderons le traitement vidéo qui est lié au traitement d'images fixes.

3.1 Traitement d'images fixes

Le traitement d'images fixes s'agit de l'apprentissage dans trois objectifs spécifiques différents : la classification des objets dans les images, leur localisation et enfin leur détection.

Comme prévu, toutes ces tâches sont apprises par le modèle sans tenir compte des propriétés spatiales et temporelles, car l'hypothèse de départ est que les photos séquentielles ne sont pas corrélées dans le temps.

3.1.1 Classification

La classification est le processus de prédiction de la classe de points de données donnés. Les classes sont parfois appelées comme cibles / labels ou catégories. La modélisation prédictive de la classification consiste à approximer une fonction de mappage (f) des variables d'entrée (X) aux variables de sortie discrètes (y).

La classification appartient à la catégorie de l'apprentissage supervisé où les cibles sont également fournies avec les données d'entrée. Il existe de nombreuses applications en classification dans de nombreux domaines tels que l'approbation de crédit, le diagnostic médical, le marketing ciblé, etc.

3.1.2 Localisation

La localisation et la détection semblent vraiment similaires et la plupart de la communauté est parfois confuse à cause de la légère différence. Les deux visent à détecter de l'image d'entrée vers une sortie faite de la présence ou non d'objets. S'ils sont présents, ils doivent produire un nombre équivalent de boîtes englobantes qui mettent en évidence les objets de l'image.

En quelques mots, la différence est selon la typologie des objets : des objets mobiles et fixes pour la localisation et uniquement des objets mobiles pour la détection.

3.1.3 Détection

La détection, comme indiqué ci-dessus, est la tâche de vérifier la présence d'objets et, si elle est positive, de les mettre en évidence avec un cadre de sélection.

Voici les principaux modèles du sujet de détection :

— Fast R-CNN[6], Microsoft Research (Ross Girshick)
— Faster R-CNN[6], Microsoft Research (Faster R-CNN : Towards Real-Time Object Detection with Region Proposition, 2015).
— YOLO (You Only Look Once : détection unifiée et en temps réel des objets, 2015).
— OverFeat, NYU (OverFeat : reconnaissance, localisation et détection intégrées utilisant les réseaux convolutifs, 2014).

Method	**Test Dataset**	**mAP%**
Fast R-CNN, [14] 2014	VOC 07+12	68.4
Faster R-CNN (RPN + VGG, shared), [39] 2014	VOC 07+12	70.4
You Only Look Once, [38] 2015	VOC 07+12	64.3
YOLO and Faster R-CNN, [38]2015	VOC 07+12	75.0
Overfeat, [41] 2013	ILSVRC2013	24.3

TABLE 1 – Comparaison de certains modèles de détection [7].

4 Analyse

Dans cette section nous allons tout d'abord expliquer tout ce qui est nécessaire pour construire une architecture et un modèle Deep learning, et puis on va parler des architecture classique proposé par les pionniers de l'intelligence artificiel, pour enfin proposer une architecture pour notre problème (Le suivi de mouvement dans une vidéo) en se basant sur ce qui precède.

4.1 Architectures Classique

4.1.1 VGG

VGGNet[8] se compose de 16 couches convolutives et est très attrayant en raison de son architecture très uniforme. Seulement 3x3 convolutions, mais beaucoup de filtres. Il s'agit actuellement du choix le plus populaire dans la communauté pour l'extraction de fonctionnalités à partir d'images. La configuration de poids du VGGNet est disponible publiquement et a été utilisée dans de nombreuses autres applications et défis en tant qu'extracteur de fonctionnalités de base. Cependant, VGGNet comprend 138 millions de paramètres, ce qui peut être un peu difficile à gérer.

FIGURE 4 – Architecture VGGNet

4.1.2 Inception

Inception[9] est un réseau qui utilise un CNN inspiré de LeNet mais a mis en œuvre un nouvel élément appelé module de création. Il a utilisé la normalisation par lots, les distorsions d'image et RMSprop. Ce module est basé sur plusieurs très petites convolutions afin de réduire considérablement le nombre de paramètres. Leur architecture consistait en un réseau CNN profond de 22 couches mais réduisait le nombre de paramètres de 60 millions (AlexNet) à 4 millions.

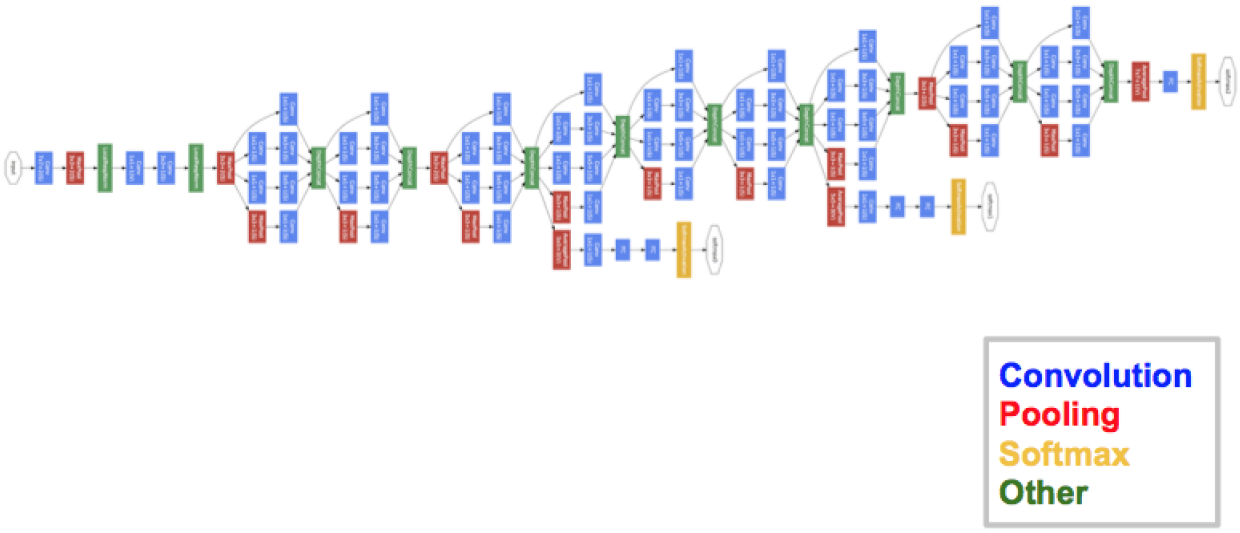

FIGURE 5 – Architecture Inception

4.1.3 ResNet

L'architecture avec «saute les connexions» présente une normalisation lourde des lots. Ces connexions de saut sont également appelées unités gated ou unités récurrentes gated et présentent une forte similitude avec les éléments réussis récents appliqués dans les RNN. Grâce à cette technique, ils ont pu former un NN avec 152 couches tout en ayant une complexité inférieure à celle de VGGNet. Il atteint un taux d'erreur de 3,57ce qui est supérieur à la performance humaine de cet ensemble de données.

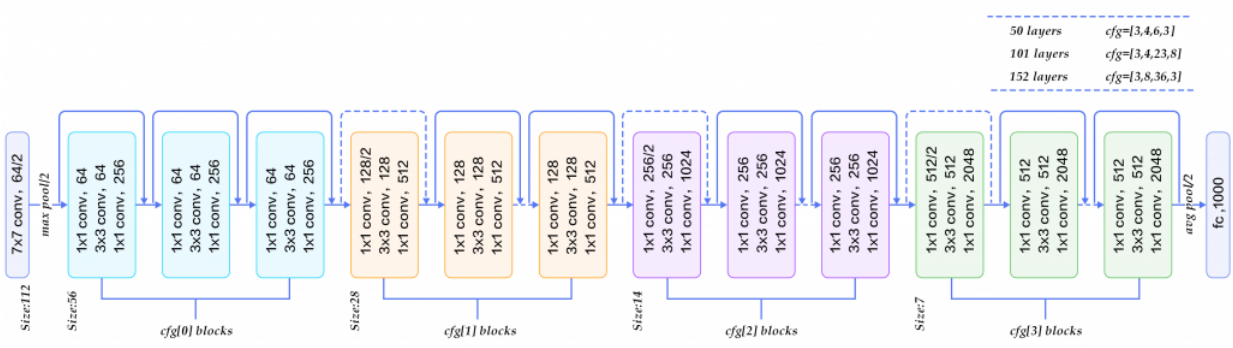

FIGURE 6 – Architecture ResNet[10]

4.1.4 Faster RCNN

Le Faster R-CNN, est composé de deux modules. Le premier module est un réseau entièrement convolutif qui propose des régions, et le deuxième module est le détecteur Fast R-CNN qui utilise les régions proposées. L'ensemble du système est un réseau unique et unifié pour la détection d'objets.

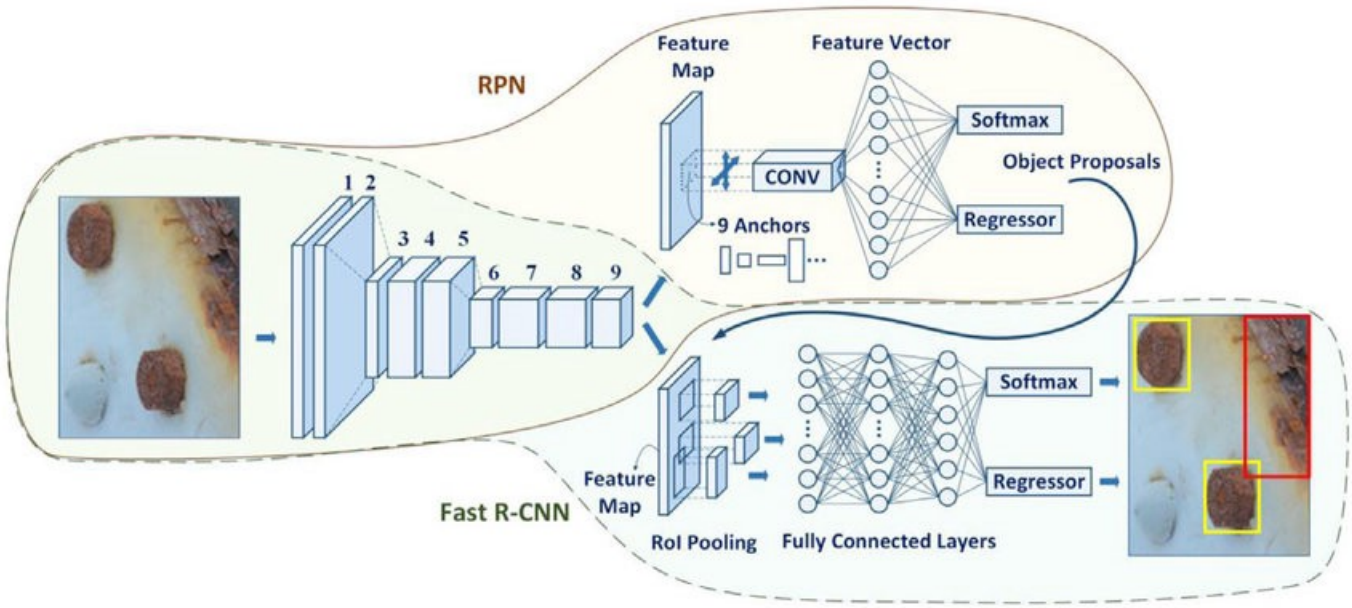

FIGURE 7 – Architecture FASTER-RCNN

4.2 Comparatif Fast RCNN et Faster RCNN

Fast RCNN VS Faster RCNN			
Algorithme	Caractéristiques	Temps/image de prédiction	Limites
Fast RCNN	Chaque image n'est transmise qu'une seule fois au CNN et les cartes d'entités sont extraites. La recherche sélective est utilisée sur ces cartes pour générer des prédictions.	2 secondes	La recherche sélective est lente et donc le temps de calcul est encore élevé.
Faster RCNN	Remplace la méthode de recherche sélective par un réseau de propositions de régions, ce qui a rendu l'algorithme beaucoup plus rapides	0.2 secondes	La proposition d'objet prend du temps et comme différents systèmes fonctionnent l'un après l'autre, les performances des systèmes dépendent de la performance du système précédent.

Table 2 – Comparaison Fast RCNN et Faster RCNN.

4.3 Architecture utilisée

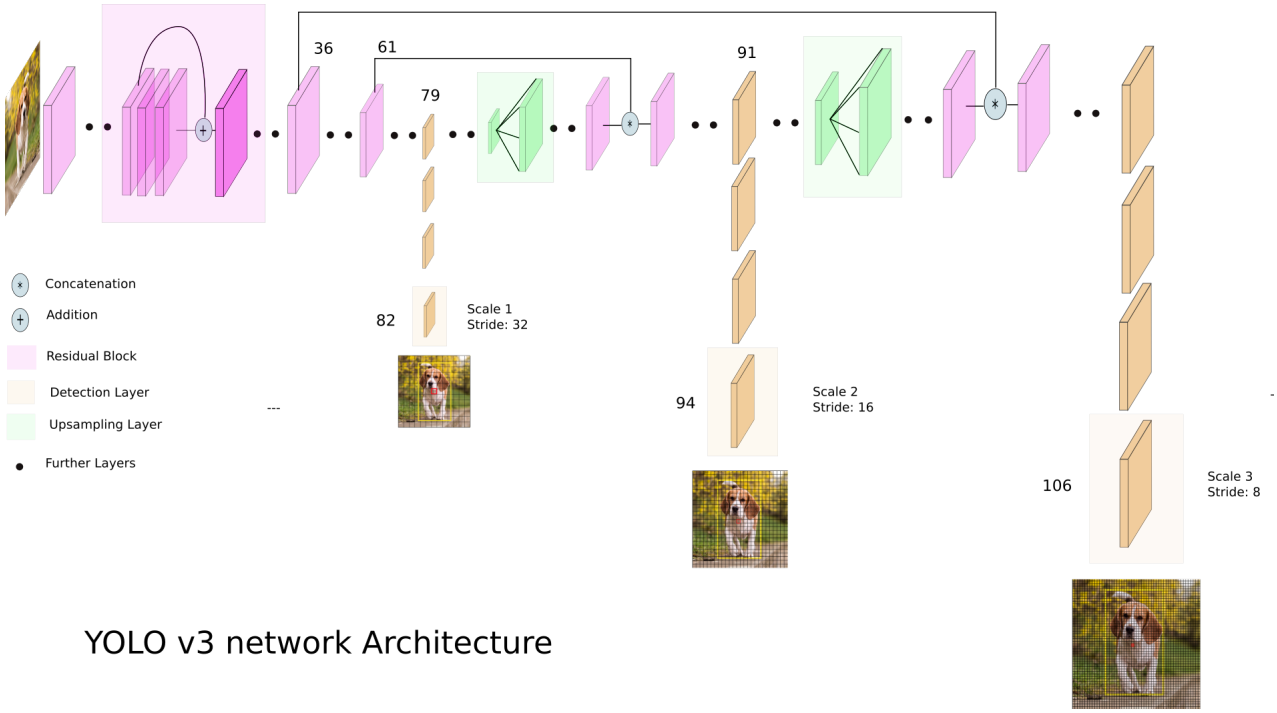

FIGURE 8 – Architecture YOLOv3

— Residual block

Les blocs résiduels sont fondamentalement un cas particulier des réseaux routiers sans aucune porte dans leurs connexions de saut. Essentiellement, les blocs résiduels permettent le flux de mémoire (ou d'informations) des couches initiales aux dernières couches.

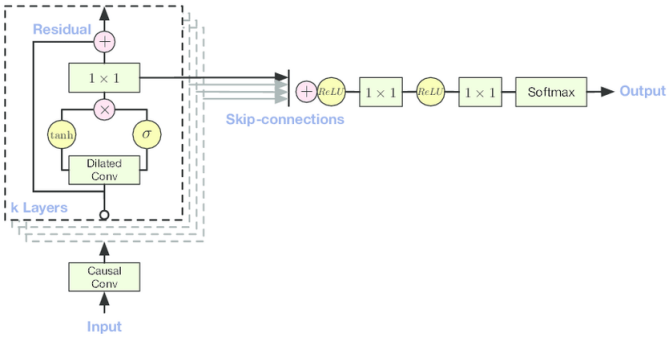

FIGURE 9 – Architecture Residual Block

— DarkNet

Darknet est un framework de réseau neuronal open source écrit en C et CUDA. Il est rapide, facile à installer et prend en charge le calcul CPU et GPU.

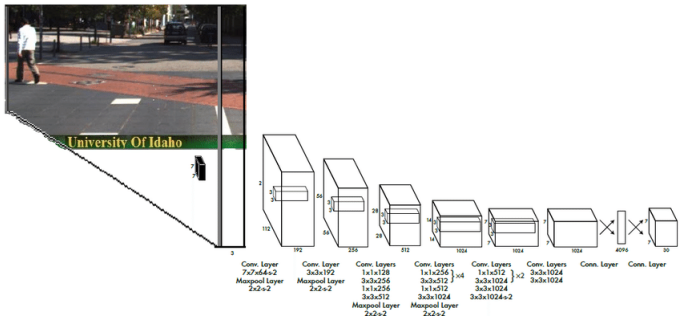

FIGURE 10 – Architecture DarkNet[11]

— ResNet

Un réseau neuronal résiduel est un réseau neuronal artificiel qui s'appuie sur des constructions connues des cellules pyramidales du cortex cérébral. Les réseaux de neurones résiduels le font en utilisant des connexions de saut ou des raccourcis pour sauter par-dessus certaines couches.

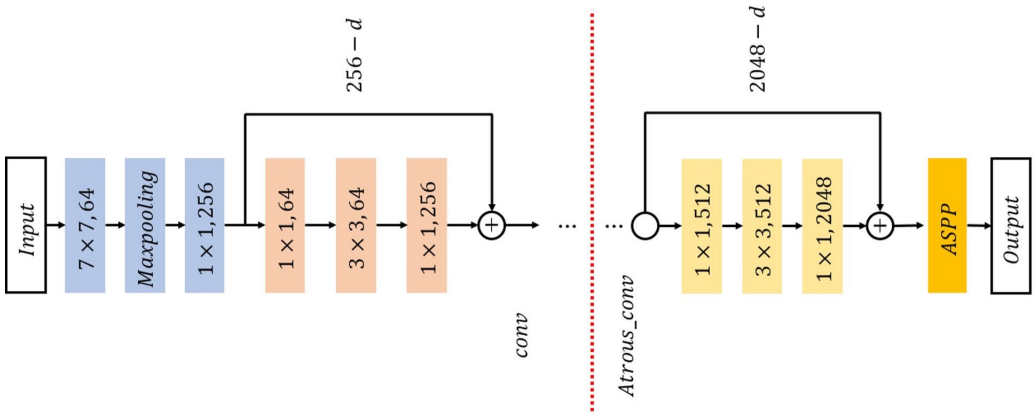

FIGURE 11 – Architecture ResNet

5 Développement du traitement

Cette section passe par le développement du modèle que nous avons implémenté. Dans la première section, nous expliquerons les détails du pipeline d'une image ou d'une vidéo à prévoir, décrivant chaque étape qu'il traverse le modèle. Ensuite, dans la deuxième section, nous décrirons les détails et les hyper-paramètres utilisés pour la formation ainsi que l'ensemble de données utilisé, qui étaient les principaux problèmes que nous avons rencontrés et comment nous avons essayé de les surmonter.

5.1 Traitement d'une image

1. **Image en entrée**

   L'algorithme accepte en entrée une matrice représentant les pixels de l'image en entrée de 3 canaux.

2. **Retailler l'image**

   Adapter et retailler l'image en entrée à l'expectation de l'algorithme : Hauteur et Longueur égaux respectivement à 416, 416 avec 3 canaux.

3. **Prédiction des rectangles**

   — Faire passer l'image retaillée dans le modèle et extraire la liste de prédictions contenant les trois prédictions associées aux 3 niveaux de prédictions.
   — Comme indiqué dans le Paper de YOLOv3, extrairons les coordonnées prédites à l'aide des fonctions Sigmoid et exp et multiplions ces coordonnées par les coordonnées des boîtes anchors.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

FIGURE 12 – Equations d'extraction des coordonnées.[1]

   — À l'aide de la fonction Sigmoid, extrairons la probabilité de l'existence d'un objet dans chaque boîte prédite ainsi que les probabilités de l'appartenance aux classes.
   — Répartir les boîtes résultantes au niveau de la grille et mettre à l'échelle les coordonnées selon la grille, la largeur et la longueur selon la taille de l'image traitée de chaque boite.

4. **Filtrer les réctangles par le seuil : *Classthreshold***

   Garder seulement les rectangles qui ont une probabilité supérieure ou égale au seuil *Classthreshold* égal à 0.6.

5. **Filtrer les réctangles avec l'algorithme Non max suppression**

   Filtrer les boites résultantes de l'étape précédente à l'aide de l'algorithme NMS en utilisant un seuil de Intersection Over Union (**IOU**) égal à 0.5.

6. **Traçage des rectangles**

   Réadapter les tailles des boîtes de l'étape précédente selon la taille de l'image originale et les tracer en écrivant les labels à l'aide de OpenCV.

5.2 Traitement d'une vidéo

Le traitement d'une vidéo s'agit de répéter le traitement d'une image sur autant de frames extraites de la vidéo. Les étapes suivantes illustrent ce traitement :

1. Lire et extraire les frames d'une vidéo en entrée à l'aide de OpenCV.

2. Parcourir la liste des frames et appliquer la démarche de traitement d'une image expliquée ci-dessus sur chaque frame.

3. Écrire chaque frame résultante dans un buffer de vidéo en utilisant OpenCV.

4. Faire flush du buffer dans un fichier vidéo.

5.3 Entraînement du modèle

5.3.1 MS COCO Dataset

MS COCO[12] est un ensemble de données de détection, de segmentation et de sous-titrage d'objets à grande échelle. Cette version contient des images, des boîtes englobantes " et des labels pour la version 2014.

Remarques :

— Certaines images du train et des ensembles de validation n'ont pas d'annotations.
— Le fractionnement de test n'a pas d'annotations (seulement des images).
— COCO définit 91 classes mais les données n'utilisent que 80 classes.
— La taille du Dataset : 37.57 GiB.

5.3.2 Paramètres et Hyper-paramètres de l'entraînement

Après avoir réalisé l'architecture du modèle représentée dans la Figure 9 avec Keras, on définit les paramètres de l'entraînement suivants :

— Taux de division du Dataset : 20% Pour le lot de validation et 80% pour le lot de l'entraînement.
— Utilisation des poids pré-entraînés et faire freeze à toutes les layers sauf les trois dernières layers qui sont destinées à entraîner.
— Appliquer la régularisation sur la plupart des layers de convolutions via l'application du Batch Normalization de paramètre epsilon = 0.001.
— Utiliser LeakyReLU au lieu de ReLU sur la plupart des layers de convolutions avec le paramètre alpha = 0.1 pour éviter l'amortimssement des poids durant l'entraînement.
— Utiliser l'optimisateur *Adam* avec un *learning_rate = 0.001*, un moment *beta = 0.9* et un *epsilon = 1e-7*.
— Utiliser un batch_size = 32 avec 750 epochs.
— Utiliser une fonction d'erreur particulière basée sur la fonction d'erreur *binary_crossentropy* disponible dans Keras (utile pour éviter les débordements de exp()) :

  — Calculer le coût associé aux coordonnées (X, Y) à l'aide de *binary_crossentropy*.
  — Calculer le coût associé à la Longueur et la Largeur par le carré de la différence entre la prédiction et les vraies valeurs.
  — Calculer le coût de l'existence d'un objet *(*confidence_loss) à l'aide de *binary_crossentropy*.
  — Calculer le coût de l'appartenance à une classe *class_loss* à l'aide de *binary_crossentropy*.

  Après avoir calculer tous ces élèments principaux du coût, le coût global est calculé, donc, par sommation des coûts élémentaires. La figure dans la page suivante illustrera la formule mathématiques de la fonction de coût.

$$\lambda_{coord} \times \sum \Omega_{obj} \times BinaryCrossEntropy(box_{xy}, boxTrue_{xy}) +$$

$$\lambda_{coord} \times \sum \Omega_{obj} \times (box_{wh} - boxTrue_{wh})^2 +$$

$$\sum \times \Omega_{obj} \times BinaryCrossEntropy(box_{class}, boxTrue_{class}) +$$

$$\lambda_{noobj} \times \sum \Omega_{noobj} \times BinaryCrossEntropy(box_{class}, boxTrue_{class}) +$$

$$\sum \Omega_{obj} \times BinaryCrossEntropy(box_{confidence}, boxTrue_{confidence})$$

$\Omega_{obj} \in [0,1] : L'existence\ de\ \ l'objet.$
$\Omega_{noobj} \in [0,1] : Complementaire\ de\ \Omega_{obj}$
$\lambda_{coord} = 5$
$\lambda_{noobj} = 0.5$

FIGURE 13 – Fonction de coût.

(λ_{coord}) est utilisé ici pour mettre d'avantage l'accent sur la précision du box, nous multiplions la perte, donc, par (λ_{coord}) (par défaut : 5).
(λ_{noobj}) est utilisé car la plupart des box ne contiennent aucun objet. Cela provoque un problème de déséquilibre de classe, c'est-à-dire que nous entraînons le modèle à détecter l'arrière-plan plus fréquemment que la détection d'objets. Pour y remédier, nous pondérons cette perte d'un facteur (λ_{noobj}) (par défaut : 0,5).

En outre, Le problème du débordement de l'exp() mentionné au-dessus a été corrigé dans *binary_crossentropy*[13]. Il vient de :

$$z \times -log(sigmoid(x)) + (1-z) \times -log(1 - sigmoid(x))$$
$$z \times -log(1/(1+exp(-x))) + (1-z) \times -log(exp(-x)/(1+exp(-x)))$$
$$\ldots$$
$$\ldots$$
$$x - x \times z + log(1+exp(-x))$$

FIGURE 14 – exp() overflow : Equation (1) Adam.

Pour ($x < 0$) , pour éviter un débordement dans exp (-x), nous reformulons ce qui précède :

$$\begin{aligned} & x - x \times z + log(1+exp(-x)) \\ &= log(exp(x)) - x \times z + log(1+exp(-x)) \\ &= -x * z + log(1+exp(x)) \end{aligned}$$

FIGURE 15 – exp() overflow : Equation(2) Adam.

Par conséquent, pour assurer la stabilité et éviter les débordements, la mise en œuvre utilise cette formulation équivalente :

$$max(x,0) - x \times z + log(1+exp(-abs(x)))$$

FIGURE 16 – exp() overflow : Equation(3) Adam.

6 Conclusion

Ce projet était un grand défi personnel, rien de facile à réaliser. Au départ, nous étions plutôt inquiet pour un sujet comme le Machine Learning, qui était pour nous une nouvelle approche. Nous avons trouvé des difficultés - au niveau d'entrainement du modèle - dû à la configuration disponisble insuffisante.

En définitive, à travers ce travail nous avons pu vérifier la faisabilité de traiter un sujet de nouvelles approches, voire avoir la chance de déployer un modèle sur une plateforme de déploiement Deep Learning.

7 Bibliographie

[1] https://pjreddie.com/media/files/papers/YOLOv3.pdf

[2] https://keras.io/api/

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

[4] https://opencv.org/

[5] https://numpy.org/doc/stable/

[6] https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-o

[7] http://imatge.upc.edu/web/sites/default/files/pub/xFerri16.pdf

[8] http://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-incep

[9] https://towardsdatascience.com/a-simple-guide-to-the-versions-of-th

[10] https://towardsdatascience.com/an-overview-of-resnet-and-its-varia

[11] https://pjreddie.com/darknet/yolo/

[12] http://cocodataset.org/

[13] https://www.tensorflow.org/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits/