Suivi de passagers de bus par apprentissage profond

Claire Labit-Bonis[1,2] Jérôme Thomas[1] Frédéric Lerasle[2,3] Francisco Madrigal[2]

[1] ACTIA Automotive, 5 rue Jorge Semprun, 31432 Toulouse
[2] CNRS, LAAS, 7 avenue du Colonel Roche, 31400 Toulouse
[3] Université de Toulouse, UPS, 118 route de Narbonne, 31400 Toulouse

{clabitbo, lerasle, jfmadrig}@laas.fr, jerome.thomas@actia.fr

Résumé

Cet article présente une étude comparative de suivi par détection appliqué au comptage passager dans les bus de ville. Un détecteur cible les passagers à chaque instant image, puis un traqueur reconstruit leurs trajectoires en associant ces détections. Nous comparons trois détecteurs par apprentissage profond peu explorés dans notre contexte applicatif, que nous couplons avec un traqueur temps réel pour une évaluation globale sur notre base large échelle d'images en situation réelle. Les résultats présentés sont très encourageants tant en terme de détection et de suivi que de rapidité pour nos perspectives embarquées.

Mots Clef

Suivi multi-cibles, détection de personnes en vue zénithale, détection d'objets par apprentissage profond.

Abstract

In this paper we present a comparative study of tracking-by-detection approaches applied to passenger counting in city buses. A detector targets passengers at each frame, a tracker then matches detections together through time to produce trajectories. We compare three deep learning detectors still under-explored in our context, and couple them with a real time tracker for global evaluation on our large scale in situ *dataset. The results we present are very encouraging in terms of detection, tracking rate and speed expected for our embedded perspectives.*

Keywords

Multi object tracking, zenithal people detection, deep learning object detection.

1 Introduction

1.1 Contexte

Suivi multi-cibles appliqué au comptage passager. Le suivi multi-cibles de personnes par détection préalable est largement investigué dans la littérature et trouve de nombreuses applications de tous types. En l'utilisant dans les transports en commun et notamment dans les bus de ville, il permet d'étudier la fréquentation des lignes du réseau grâce à du comptage de passagers, aidant ainsi l'exploitant à répartir ses flottes de véhicules de manière optimisée et à estimer le nombre de fraudes. Plusieurs études de marché montrent un réel besoin de ce type de dispositif dans les transports publics [1].

Systèmes existants et choix associés. Différents systèmes fonctionnels existent déjà pour compter les passagers dans les bus *e.g.* validateurs électroniques de tickets ou capteurs dans le plancher ; dans le cadre de nos travaux nous nous focalisons sur les systèmes de détection individualisée, placés au niveau des portes du véhicule. Plusieurs technologies capteurs ont démontré de bonnes performances *e.g.* radar [2], caméra active [3], ou scanner laser [4]. Le coût de ces capteurs et les récentes avancées en matière de vision par ordinateur ont motivé notre choix d'utiliser des caméras perspectives classiques pour notre application.

Principe du suivi par détection dans notre contexte. Le principe est alors de : (i) détecter les personnes à chaque instant image, (ii) associer leurs détections respectives dans le flux vidéo pour filtrer les artefacts de détection, enfin (iii) dans notre contexte, compter le nombre de trajectoires franchissant le pas de porte. Selon le sens de la trajectoire, la personne suivie sera comptabilisée comme « entrante » ou « sortante ». Le pas de la porte est représenté par une ligne imaginaire dans notre image. Nous décrivons notre approche de « suivi par détection » à la section 2.

Peu de systèmes par apprentissage profond. Dans ce contexte de comptage, la détection [5] et le suivi de personnes [6] par vision ont déjà été investigués dans la littérature. Cependant à notre connaissance, peu de systèmes privilégient des détecteurs récents type apprentissage profond. Kocamaz et al. [5] utilisent un classifieur en cascade basé sur des Integral Channel Features, Z. Yu et al. [6] utilisent des opérations de morphologie mathématique, B. Benfold et I. Reid [7] utilisent un détecteur HOG ; pourtant l'utilisation de réseaux profonds a démontré de réels gains en détection [8] ; il est donc légitime de l'évaluer dans notre application de comptage passager par détection préalable.

1.2 Nos contributions

Comparaison de trois détecteurs neuronaux dans notre contexte. Nous proposons une étude comparative de trois détecteurs neuronaux usuels et *opensource* couplés avec un traqueur dans notre contexte. Notre application nécessitant un traitement embarqué - et donc limité en CPU - à bord des bus (section 2), nous devons trouver un compromis temps de calcul/qualité de détection et présentons pour cela des évaluations croisées architecture réseau *vs.* détecteur.

Originalité du point de vue zénithal. Notre application sur vue zénithale se démarque de la problématique détection et suivi visuel de piétons donc en vue perspective.

Création d'un *dataset* en situation réelle. L'apprentissage de notre système, les évaluations associées en croisant détecteurs *vs.* filtre, s'appuient sur des vidéos acquises *in situ*, donc dans un bus de ville plus ou moins bondé. Les évaluations (large échelle) quantitatives associées, notamment pour des scènes très encombrées *i.e.* à heures de pointe, sont très encourageantes.

La section 2 décrit l'architecture de notre système de suivi par détection dans notre contexte de comptage, ainsi que les choix techniques et la mise en œuvre associés. La section 3 présente les résultats de nos travaux au travers d'évaluations quantitatives et qualitatives des approches choisies pour notre application. Enfin, nous concluons et présentons les perspectives futures de nos travaux dans la section 4.

2 Notre approche

Problématique complexe. Notre problématique de détection et suivi de personnes dans les bus est complexe de par la variabilité des cibles à détecter (tenue vestimentaire, morphologie) et l'encombrement des scènes (à heure de pointe).

Contrainte de traitement à temps différé. Par ailleurs, notre contexte applicatif implique l'utilisation de calculateurs embarqués pour un traitement à bord et à temps différé *i.e.* le comptage n'est pas à inférer aux instants même de montée/descente des passagers. Le traitement des flux vidéos entre les arrêts (pendant les déplacements du véhicule) est envisagé sachant que notre finalité est d'embarquer nos algorithmes récents d'apprentissage profond sur des calculateurs de type NVIDIA Jetson TX2 *a priori* limités en ressources. Cette contrainte a largement motivé le choix des technologies explorées dans notre étude.

2.1 Synoptique du système

Architecture matérielle. Afin d'être compétitifs vis-à-vis des systèmes existants, la finalité est de compter avec une marge d'erreur faible (< 10%). Nous avons donc privilégié un placement de caméra zénithal pour limiter les occultations autant que possible. La figure 1 illustre la configuration du système complet de comptage. Il est composé d'autant de caméras que de portes dans le bus (pouvant aller de 2 à 4). Elles filment le passage des usagers aux portes

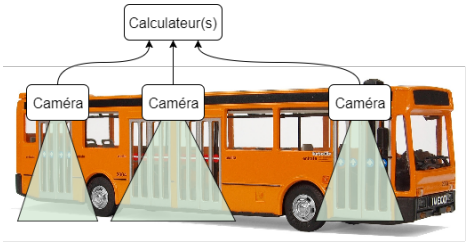


FIGURE 1 – Configuration du système de vision embarqué. Une caméra zénithale dévolue à chaque porte du bus et connectée à un même/unique calculateur pour traiter les flux associés.

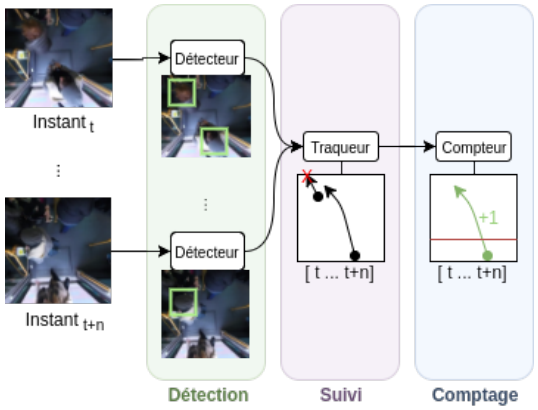


FIGURE 2 – Synoptique de notre système de comptage. En entrée, le flux de la caméra zénithale. En sortie, le comptage des cibles, *via* leurs trajectoires inférées, lors de l'ouverture des portes.

en vue zénithale et sont toutes reliées à un calculateur embarqué chargé de traiter leur flux. L'ajout d'un second calculateur est envisagé eu égard au nombre de caméras et besoin en capacité CPU.

Architecture logicielle. Pour rappel, notre approche séquence les trois modalités suivantes (figure 2) : détection image de personnes, reconstruction des trajectoires associées par association des détections inter-images, comptage des passagers en entrées/sorties du bus.

2.2 Choix des détecteurs et motivations

Les travaux sur réseaux de neurones profonds sont pléthoriques depuis AlexNet [9] en 2012 dans ILSVRC [10]. Ainsi pour le challenge ImageNet de détection d'objets en 2017, 100% des approches privilégient les réseaux de neurones profonds. Quatre détecteurs ont démontré leur efficacité : Faster R-CNN [11], R-FCN [12], SSD [13] et YOLOv2 [14]. De plus, une étude comparative [15] a validé le choix de Faster R-CNN, SSD et YOLOv2 pour nos travaux. Ces trois détecteurs sont par ailleurs rapides à l'exécution, et donc adaptés à notre contrainte de traitement à temps différé.

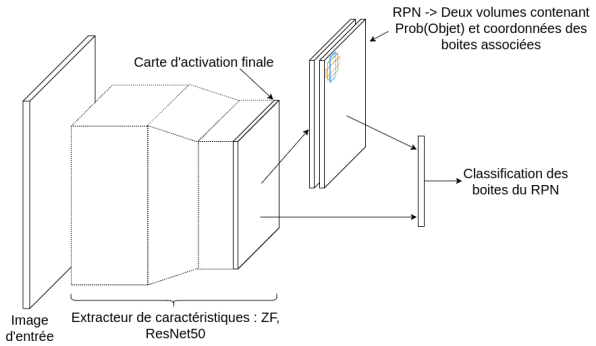


FIGURE 3 – Faster R-CNN. Un extracteur de caractéristiques produit une carte d'activation, utilisée pour prédire des boîtes et probabilités d'objet *via* un réseau de proposition de régions. Ces détections sont ensuite classifiées pour produire un vecteur de scores de classes associé aux boîtes.

Faster R-CNN. Le détecteur Faster R-CNN [11] associe proposition de régions et classification au travers d'une architecture de réseau classique *e.g.* ZF ou ResNet qui opère un ensemble de convolutions sur l'image d'entrée. La carte d'activation générée alimente ensuite un réseau de proposition de régions (RPN) qui utilise une fenêtre glissante pour prédire, à chaque position, un ensemble de boites caractérisées à la fois par leur probabilité de contenir un objet et par leurs coordonnées relatives à des « ancres » d'aspects et ratios prédéfinis. Ces régions proposées ainsi que la carte d'activation utilisée alimentent ensuite des couches destinées à la classification et à la prédiction des coordonnées des boîtes englobantes correspondantes dans l'image d'origine. Cette architecture de détection est illustrée figure 3.

La particularité de Faster R-CNN réside dans le fait que les couches de convolution situées en amont du RPN mais également des couches de classification sont partagées et optimisées pour ces deux problématiques, en même temps, *via* un apprentissage « alterné » des deux branches.

SSD. SSD [13] est un détecteur *one shot i.e.* qui voit une seule fois l'image entière en entrée de son réseau sans passer par un mécanisme de proposition puis de classification de régions, d'où sa rapidité. Comme illustré figure 4, il utilise un réseau de convolution usuel (type Inception, ResNet) dont il tronque la couche de régression pour la remplacer par plusieurs couches de convolution successives dont la résolution décroît. En sortie de chacune, la carte d'activation générée est divisée selon une grille. A l'instar de Faster R-CNN, chaque case génère des boîtes caractérisées par un *offset* en x et y par rapport à des ancres, une hauteur h, une largeur w et un vecteur de probabilités des classes à détecter. A la fin du réseau, une couche rassemble et filtre les prédictions des différentes résolutions.

La différence avec Faster R-CNN est cette prédiction de boîtes à différents stades du réseau et dont les scores sont directement calculés sans passer par deux étapes distinctes de proposition puis de classification de régions.

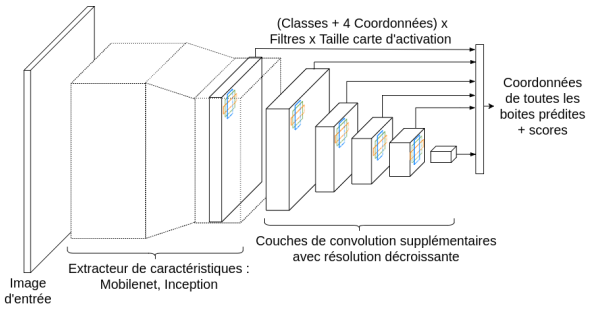


FIGURE 4 – SSD. L'image passe par un extracteur de caractéristiques convolutionnel, puis des couches de résolution décroissante y sont ajoutées. A chaque étape, un ensemble de boîtes englobantes et scores de classes sont générés.

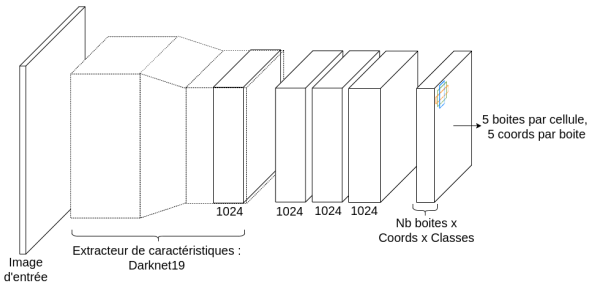


FIGURE 5 – YOLOv2. L'image d'entrée passe par un extracteur de caractéristiques « Darknet19 » ; trois couches convolutionnelles lui sont greffées. Le réseau se base ensuite sur des ancres d'aspects et ratios prédéfinis pour prédire des boîtes englobantes et scores de confiance.

YOLOv2. YOLOv2 [14] est, comme SSD, un détecteur *one shot* qui exécute une seule fois son réseau de convolution sur une image entière pour un traitement rapide.
Dans la première version de YOLO, des convolutions extraient les caractéristiques de l'image, puis une couche de neurones complètement connectés prédit directement pour chaque zone de l'image, divisée selon une grille, un ensemble de coordonnées x, y, w, h de boîtes englobantes et leur probabilité de classe associée. Dans la nouvelle version de YOLO, différents ajustements ont été réalisés parmi lesquels la suppression de la couche complètement connectée pour reprendre l'idée commune à Faster R-CNN et SSD de prédiction d'*offsets* par rapport à des ancres, l'apprentissage multi-échelles en changeant aléatoirement la taille d'entrée du réseau, ou encore la conception d'une nouvelle structure convolutionnelle d'extraction de caractéristiques plus précise et moins coûteuse en calculs. La figure 5 illustre l'architecture de YOLOv2.

2.3 Choix du traqueur et motivations

DeepSORT. DeepSORT [16] est un traqueur multi-cibles temps réel basé sur une association inter-images de détections qui prend en compte la cohérence du mouvement apparent et l'apparence des cibles traquées. Son rôle est illustré par la figure 6.

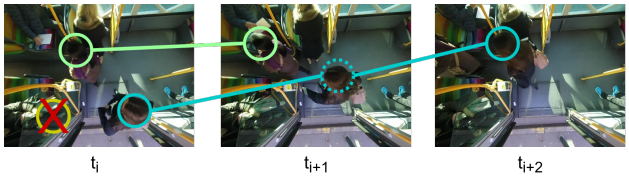


FIGURE 6 – Un traqueur multi-cibles associe les détections dans le temps. Dans l'idéal, il permet de reconstruire les trajectoires en filtrant à la fois les faux positifs (t_i, en bas à gauche) et les faux négatifs (t_{i+1}, en pointillés).

Un réseau de convolution extrait les caractéristiques de chaque détection à chaque instant image ; celles-ci sont ensuite associées aux trajectoires existantes *via* un algorithme hongrois prenant en compte la position de la cible et le vecteur de caractéristiques généré par le réseau de convolution. Les trajectoires non associées sont prolongées par les prédictions d'un filtre de Kalman et les détections non associées génèrent de nouvelles trajectoires.
DeepSORT est simple, très peu coûteux en temps de calcul et a montré de bonnes performances dans le MOTChallenge 2016 [17] ; pour ces raisons, il paraît adapté à nos contraintes.

2.4 Mise en œuvre

Matériel utilisé. Nous avons réalisé tous nos tests et apprentissages sur un poste fixe équipé d'un processeur Intel Xeon E5-1620V4 / 3.5 GHz, de 16Go de RAM, et d'une carte graphique NVIDIA Titan Xp.

Création d'une base vidéo *in situ* large échelle. Pour nos évaluations, nous avons constitué une base d'images en plaçant une caméra GoPro au-dessus de la porte centrale d'un bus le temps d'une journée. Les vidéos sont acquises à divers moments de la journée pour capturer la variabilité des scènes perçues (jour/nuit, encombrement du bus, forme/apparence des passagers).
La base est divisée en trois parties : apprentissage, validation et test. L'optimisation des détecteurs est faite sur la partie apprentissage ; les hyperparamètres de l'application *e.g.* pas d'apprentissage de la fonction d'optimisation, seuil de classification pour le suivi sont ajustés d'après les résultats obtenus sur la partie validation. Une fois tous les paramètres fixés, les résultats finaux sont générés sur une base jusqu'alors inconnue : la base de test. Le tableau 1 caractérise notre *dataset* et les annotations réalisées. Des exemples d'images sont présentés à la section 3 avec les évaluations qualitatives.

Extracteurs de caractéristiques *vs.* détecteurs La récente implémentation d'une API de détection dans Tensorflow [15] nous a permis de tester différentes architectures internes pour Faster R-CNN et SSD. L'intérêt est alors d'évaluer diverses couches du réseau dédiées aux caractéristiques image *i.e.* en amont des couches de régression.
Nous testons Faster R-CNN avec des extracteurs de caractéristiques ZFNet [18] et ResNet50 [19] ; SSD avec

	Nom	Images	Traject. $\neq$	Labels
App.	video_1	15,382	47	15,724
	video_2	18,427	76	16,773
	video_3	29,889	96	33,947
Val.	min_clutter	9,751	45	10,017
	max_clutter	20,353	86	66,030
Test	video_4	11576	38	14,580

TABLE 1 – Caractérisation de notre base d'images : 60% dédié à l'apprentissage, 30% à la validation, 10% au test. On distingue deux scénarios : (i) min_clutter *i.e.* scène de jour et encombrement moyen (3 à 4 personnes max. par image), et (ii) max_clutter *i.e.* scène de nuit en éclairage artificiel à heure de pointe (10 personnes max. par image).

Inception-v2 [20] et MobileNet [21] ; YOLOv2 avec son extracteur Darknet19 [14]. Ces choix sont motivés par la disponibilité en ligne de modèles pré-appris sur des bases d'apprentissage publiques et qui servent d'initialisation des paramètres lors des apprentissages spécifiques sur notre base (principe du *fine-tuning*). La finalité de ce couplage est d'exhiber les associations architecture *vs.* détecteur offrant les meilleurs compromis rapidité/performances dans notre contexte. Les associations seront alors couplées avec DeepSORT [16] pour évaluations finales.

Paramètres d'apprentissage des détecteurs. Ci-après sont donnés les hyperparamètres d'apprentissage des détecteurs ainsi que les modèles qui ont servi à l'initialisation des poids des réseaux. Nous avons utilisé les valeurs par défaut pour tous les paramètres non mentionnés, et avons arrêté chaque apprentissage lorsque sa fonction de coût ne diminuait plus ou très peu :

- *Faster R-CNN + ZF* : modèle Pascal VOC, pas d'apprentissage initial de 10^{-3} divisé par 10 toutes les 50.000 itérations, 200.000 itérations au total ;
- *Faster R-CNN + ResNet50* : modèle COCO, pas initial de 10^{-5}, 200.000 itérations, *batch size* de 4 ;
- *SSD + Inception/Mobilenet* : modèle COCO, pas initial de 0.004 puis diminution d'un facteur 0.95 à l'itération 500.000 pour un total de 800.000 itérations, *batch size* de 24 ;
- *YOLOv2* : modèle Pascal VOC, pas initial de 10^{-4} pour 90.000 itérations, *batch size* de 64.

Seuil de détection pour le couplage détecteur-traqueur. Pour chaque couplage détecteur-traqueur, nous avons déterminé le seuil du score de détection qui maximisait le *Multi-Object Tracking Accuracy* (MOTA) du traqueur sur la base de validation (métriques d'évaluation à la section 3.1). Ainsi, toutes les détections dont le score de confiance est supérieur à ce seuil sont fournies en entrée du traqueur. Il est de 0.98 pour Faster R-CNN (ResNet50 et ZF), 0.8 pour SSD (Inception et Mobilenet), 0.5 pour YOLOv2.

Détails d'implémentation. Nous avons annoté notre base d'images avec l'outil VATIC [1] au format Pascal VOC. Les apprentissages de Faster R-CNN + ResNet50, SSD + Inception et SSD + Mobilenet ont été faits avec l'API de détection d'objets de Tensorflow [2], ceux de Faster R-CNN + ZF avec *py-faster-rcnn* [3], et ceux de YOLOv2 avec *darknet* [4]. Les modèles pré-entraînés évoqués auparavant sont issus de ces trois dernières références.

3 Évaluations et discussion

3.1 Métriques d'évaluation

Toutes les évaluations sont présentées sur la base de test `video_4` qui n'a eu aucune influence sur le choix des paramètres d'apprentissage ou de couplage détecteur-traqueur. Nous présentons également des évaluations sur la base de validation ; malgré des résultats biaisés par son influence sur le choix des paramètres, elle permet d'appréhender l'impact des deux scénarios `min_clutter` et `max_clutter` présentés dans le tableau 1.
Les métriques d'évaluation présentées ci-après sont suivies du symbole ↑ lorsque la valeur la plus haute est la meilleure et de ↓ dans le cas inverse.

Évaluation des détecteurs. Les métriques d'évaluation des détecteurs sont :

- la précision **Prcn** ↑ et le rappel **Rpl** ↑, fondés sur les vrais/faux positifs (VP et FP) et sur les faux négatifs (FN) : $Prcn = VP/(VP + FP)$ et $Rpl = VP/(VP + FN)$. Une précision faible indique un détecteur bruité par les fausses alarmes ; un rappel faible caractérise un détecteur manquant des cibles ;
- le score **F1** ↑ : moyenne harmonique de la précision et du rappel qui exprime la capacité du détecteur à limiter à la fois les faux positifs et les faux négatifs : $F1 = 2 \times (Prcn \times Rpl)/(Prcn + Rpl)$
- le nombre de *frames* par seconde **FPS** ↑ que traite le détecteur.

Nous présentons également des courbes mettant en regard le **taux de non-détection** ↓ *vs.* le nombre de faux positifs par image **FPPI** ↓ en faisant varier le seuil de détection.

Évaluation du couplage détecteur-traqueur. Nous utilisons les métriques du défi MOT [5] pour évaluer le couplage du traqueur DeepSORT, à savoir :

- le *Multi-Object Tracking Accuracy* **MOTA** ↑ calculé à partir des faux positifs **FP** ↓, faux négatifs **FN** ↓ et changements d'identité **IDSwitches** ↓ ;
- le *Multi-Object Tracking Precision* **MOTP** ↑ calculé à partir du score de recouvrement entre les cibles traquées et la vérité terrain ;
- le score **F1** ↑ qui prend en compte la précision et le rappel du traqueur ;
- le nombre de fragmentations **FM** ↓ *i.e.* le nombre de fois que des trajectoires ont été coupées ;
- le pourcentage de vraies trajectoires couvertes au moins à 80% : ***Mostly Tracked*** **MT** ↑ ;
- le pourcentage de vraies trajectoires couvertes au plus à 20% : ***Mostly Lost*** **ML** ↓.

3.2 Évaluation des détecteurs

Détecteurs peu bruités. D'après la figure 7, tous les détecteurs produisent globalement peu de bruit : Faster R-CNN atteint au maximum 1.5 faux positifs par image (FPPI) sur le scénario `max_clutter` ; les autres détecteurs sont entre 0.06 et 0.4 FPPI toutes bases confondues.

Faux positifs *vs.* faux négatifs. Cependant même s'il produit plus de faux positifs comme le montrent ses valeurs de précision (tableau 2), Faster R-CNN permet d'augmenter son taux de détection en faisant varier son seuil de prédiction. Au contraire, YOLOv2 et SSD ratent plus de cibles et limitent le taux de détection atteignable. Par exemple, SSD + Inception atteint au plus 76.6% de taux de détection aux alentours de 0.4 FPPI sur `max_clutter`, là où Faster R-CNN + ZF va jusqu'à 94.8% à 1.5 FPPI.

Difficulté du scénario max_clutter. La situation à heure de pointe est plus difficile que les scénarios peu encombrés (figure 7) : en se plaçant à 0.1 FPPI, les intervalles des taux de détection vont de [85-95]% sur `min_clutter` et [70-90]% sur `test` à [45-60]% sur `max_clutter`.

YOLOv2 *vs.* SSD. YOLOv2 génère environ autant de détections que SSD mais est plus performant (figure 7) : sur la base de test `video_4` à 0.06 FPPI, YOLOv2 atteint 81.4% de taux de détection, là où SSD + Inception atteint 70.8% et SSD + Mobilenet 64.1%. Cette observation reste vraie pour les deux autres bases même si l'écart est moins marqué sur la base `max_clutter`.

Extracteurs de caractéristiques *vs.* détecteurs. La figure 8 met en regard le nombre de *frames* par seconde et le score F1 des différents couples extracteur-détecteur.
Faster R-CNN : ResNet50 et ZF ont un F1 quasi équivalent face à `test` et `max_clutter` ; cependant ResNet50 génère plus de faux positifs sur la base `min_clutter` et son F1 s'en trouve impacté négativement (73.9% pour ResNet50 contre 82.2% pour ZF). Par ailleurs, ZF est presque quatre fois plus rapide (37 FPS contre 10 pour ResNet50).
SSD : l'extracteur Inception montre de meilleures performances que Mobilenet : son score F1 est supérieur, mais l'écart entre `min_clutter` et `max_clutter` est également bien moindre (écart de 9.1% pour Inception, et de 26.2% pour Mobilenet) et semble donc plus robuste face à une situation difficile.
YOLOv2 : présente de meilleures performances que SSD sur la base de validation mais un plus gros écart avec la base de test, en étant un peu plus rapide que SDD + Inception (81 FPS contre 75).

1. VATIC : `https://github.com/cvondrick/vatic/`
2. Tensorflow : `https://github.com/tensorflow/models/tree/master/research/object_detection`
3. py-faster-rcnn : `https://github.com/rbgirshick/py-faster-rcnn`
4. Darknet : `https://pjreddie.com/darknet/yolov2/`
5. MOT Benchmark : `https://motchallenge.net`

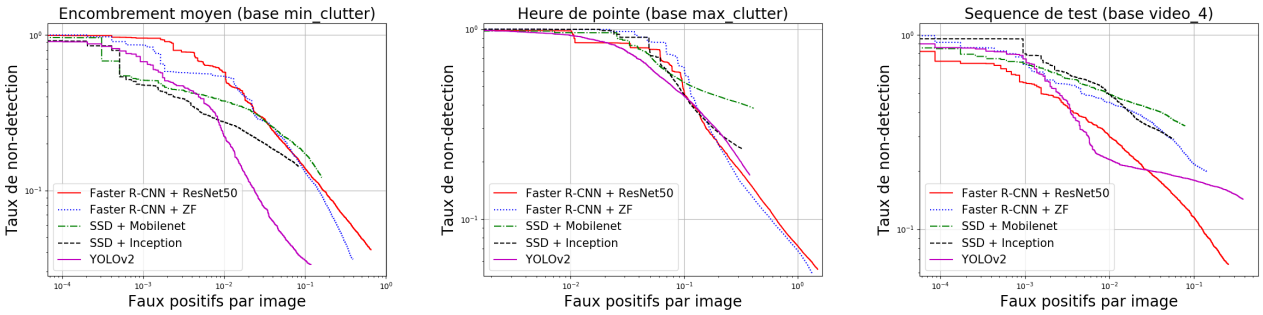


FIGURE 7 – Taux de non-détection *vs.* faux positifs par image (FPPI). (i) FRRCNN génère globalement plus de détections que les détecteurs *one shot* ; (ii) le scénario à heure de pointe montre sa difficulté : plus de FP pour un taux de non-détection plus élevé ; (iii) à nombre de détections équivalent, YOLOv2 se démarque de SSD avec de meilleures performances.

Détecteur	*min_clutter*		*max_clutter*		*video_4 (test)*	
	Prcn	**Rpl**	**Prcn**	**Rpl**	**Prcn**	**Rpl**
FRRCNN + ResNet 50	60.2	95.8	67.2	94.6	77.4	**93.4**
FRRCNN + ZF	71.6	96.4	69.8	**94.8**	83.8	80.3
SSD + Inception	**91.4**	85.7	88.3	76.6	**92.0**	70.8
SSD + Mobilenet	84.7	87.9	83.0	61.7	88.7	66.0
YOLOv2	89.4	**96.7**	**87.7**	82.9	67.9	85.7

TABLE 2 – Performances architecture réseau *vs.* détecteur. **Faster R-CNN** (i) génère plus de FP que les autres ; (ii) l'extracteur ResNet50 est stable sur les trois bases, là où ZF rate plus de cibles sur la base de *test*. **SSD** (i) produit plus de FN sur les bases *max_clutter* et *test* que sur *min_clutter* (cf. Rpl) ; (ii) Mobilenet est plus faible qu'Inception tant sur les FP que sur les FN. **YOLOv2** présente une précision et un rappel équilibrés, à l'exception d'une baisse de la précision sur la base de test.

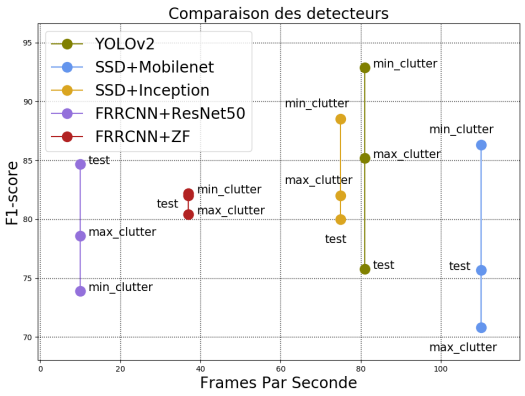


FIGURE 8 – F1 *vs.* FPS. YOLOv2 et SSD sont les plus rapides (> 75 FPS), mais FRRCNN présente un écart plus faible entre les deux scénarios extrêmes. SSD+Mobilenet est le plus rapide, mais le moins bon à heure de pointe.

Évaluation qualitative des détecteurs. La figure 9 présente trois images extraites des bases de validation et de test, appliquées aux cinq détecteurs. Nos observations précédentes sont corroborées par cette visualisation :

- sur la base `min_clutter` (ligne 1), les détecteurs se valent et détectent les deux cibles de l'image ;
- sur la base `max_clutter` (ligne 2), les deux Faster R-CNN détectent le plus de cibles. SSD Inception et Mobilenet ratent 3 et 4 cibles. YOLOv2 en rate moins que SSD, mais plus que Faster R-CNN.
- sur la base `test` (ligne 3), Faster R-CNN et YOLOv2 détectent les deux cibles (YOLOv2 et ZF génèrent resp. un et deux faux positifs). Les deux architectures SSD sont sujettes à la non-détection.

Nous notons également que YOLOv2 et Faster-RCNN ont tendance à générer des faux positifs sur les sacs à dos (constat fait sur plusieurs autres exemples en plus de ceux présentés).

3.3 Couplage détecteur - traqueur

Le tableau 3 montre les résultats du couplage détecteur-traqueur sur nos images. À titre de comparaison, la dernière ligne du tableau indique les résultats MOT16 de DeepSORT sur le suivi de piétons en vue perspective.

Bonne localisation des trajectoires. Le premier constat concerne le MOTP exprimant l'alignement des trajectoires avec la vérité terrain : il est supérieur à 78% sur les trois bases et traduit une bonne localisation du traqueur DeepSORT de manière générale.

Meilleure gestion des FP que des FN. Nous constatons par ailleurs que les détecteurs qui produisent le moins de faux négatifs sont les plus performants lors du couplage : sur `min_clutter`, YOLOv2 qui avait le plus fort rappel (96.7%) obtient les meilleurs résultats (IDF1 = 93.8%, MOTA = 87.3%). Sur `max_clutter` FRRCNN + ZF avait un rappel de 94.8% malgré une précision de 69.8% ;



(a) FrRCNN + ZF (b) FrRCNN+ResNet50 (c) SSD + Inception (d) SSD + Mobilenet (e) YOLOv2

FIGURE 9 – Exemples de détections : faible encombrement (haut), heure de pointe (milieu), cas particulier (bas). Les boîtes vertes (resp. noires) correspondent aux détections (resp. à la vérité terrain).

	Détecteur	IDF1	Prcn	Rpl	GT	MT	ML	FP	FN	IDS	FM	MOTA	MOTP
min_clutter	FRRCNN + ResNet50	88.2	90.3	86.1	45	64.4	13.3	929	1388	11	87	76.8	**83.8**
	FRRCNN + ZF	88.1	86.6	89.8	45	73.3	11.1	1396	1022	18	72	75.7	80.7
	SSD + Inception	87.3	**96.9**	79.4	45	68.9	13.3	**251**	2062	23	99	76.7	81.3
	SSD + Mobilenet	86.1	89.8	82.7	45	64.4	15.6	938	1736	14	66	73.2	81.6
	YOLOv2	**93.8**	91.9	**95.8**	45	**86.7**	**2.2**	845	**420**	**8**	**18**	**87.3**	81.0
max_clutter	FRRCNN + ResNet50	86.1	89.4	83.1	86	53.5	15.1	6491	11171	**82**	709	73.1	**79.7**
	FRRCNN + ZF	**87.4**	88.8	**86.1**	86	**64.0**	**12.8**	7179	**9183**	95	**625**	**75.1**	77.0
	SSD + Inception	79.1	**93.5**	68.5	86	27.9	23.3	**3171**	20800	136	1098	63.5	79.3
	SSD + Mobilenet	67.0	93.0	52.4	86	18.6	44.2	2598	31444	146	1081	48.2	78.7
	YOLOv2	84.2	89.4	79.6	86	46.5	15.1	6234	13487	108	704	70.0	78.8
video_4 (test)	FRRCNN + ResNet50	**82.5**	**98.7**	70.9	38	57.9	10.5	**101**	3156	20	170	**69.8**	81.3
	FRRCNN + ZF	77.9	94.9	66.0	38	63.2	13.2	388	3684	22	105	62.3	79.7
	SSD + Inception	77.5	96.6	64.7	38	47.4	18.4	245	3832	17	165	62.3	**83.2**
	SSD + Mobilenet	71.2	97.0	56.3	38	31.6	21.1	192	4742	28	177	54.3	82.5
	YOLOv2	78.8	74.3	**83.8**	38	**71.1**	**5.3**	3150	**1753**	**14**	**59**	54.7	80.6
	DeepSORT MOT16	62.2	72.1	54.7	759	32.8	18.2	12852	56668	781	2008	61.40	79.10

TABLE 3 – Performances du couplage détecteur-traqueur. YOLOv2 offre des performances équivalentes à Faster R-CNN, peu de fragmentations et de faux négatifs, avec des valeurs IDF1 et MT parmi les plus élevées. SSD + Mobilenet génère des MOTA, IDF1 et MT faibles comparativement aux autres, ainsi que beaucoup de fragmentations et de faux négatifs.

son couplage avec le traqueur se situe parmi les meilleurs sur cette base (meilleur IDF1 à 87.4%, MT et MOTA les plus hauts, FN et FM les plus bas, IDS en deuxième place).

Au contraire, les détecteurs avec un plus faible rappel offrent des performances de couplage moindre : SSD + Mobilenet avait un rappel de 61.7% sur `max_clutter`, et malgré une précision de 83.0%, il obtient les MOTA, IDF1, MT les plus bas (48.2%, 67.0% et 18.6%) pour des fragmentations, faux négatifs et changements d'identité élevés.

Cela peut s'expliquer par le fonctionnement de DeepSORT qui tue ses trajectoires après trois temps sans association ; un détecteur ratant ses cibles dans la durée donnera des trajectoires régulièrement tuées et plus de fragmentations.

3.4 Discussions relatives à notre contexte

Cohérence des résultats. L'étude comparative menée dans [15] place SSD comme le meilleur compromis temps de calcul/performance de détection, et qui reste cependant compétitif face aux autres détecteurs convolutionnels actuels ; par ailleurs la présentation de YOLOv2 à CVPR17 [6] qui se compare à cette même étude le place à des performances équivalentes à SSD tant en détection qu'en temps de calcul. Faster R-CNN est présenté comme moins rapide, mais plus performant. Ces études corroborent les résultats que nous présentons dans nos travaux.

Meilleur compromis pour notre application. De par nos contraintes d'embarquabilité et de performances de suivi par détection, le meilleur candidat pour notre applica-

6. `https://pjreddie.com/publications/`

tion semble être YOLOv2, suivi par FRRCNN + ZF même s'il est moins rapide. Ils offrent tous deux de bons résultats sur les scénarios `min/max_clutter` et génèrent moins de fragmentations et de FN, problématiques dans notre contexte de comptage, que les détecteurs SSD.
SSD + Inception pourrait être envisagé à condition de réduire le nombre de fragmentations et de faux négatifs lorsqu'on le couple au traqueur. Il présente l'avantage d'être aussi rapide que YOLOv2, et a montré de bons résultats tant en validation qu'en test.
FRRCNN + ResNet50 offre de bonnes performances tant sur la détection que sur le suivi mais semble peu exploitable de par sa lenteur. À l'inverse, SSD + Mobilenet est très rapide mais paraît peu robuste face à des situations encombrées.

4 Conclusion et perspectives

Dans notre cadre applicatif de comptage passager dans les bus de ville, nous avons présenté une étude comparative croisée de trois détecteurs neuronaux peu étudiés dans notre contexte et les avons couplés à un traqueur temps réel pour une approche de suivi par détection préalable. Par ailleurs toutes nos évaluations ont été réalisées sur une base d'images large échelle réalisée en situation réelle.
Malgré la problématique complexe liée à la variabilité de nos cibles (tenue vestimentaire, morphologie) et à l'encombrement du véhicule, les résultats obtenus à la fois en situation peu encombrée et à heure de pointe sont encourageants et nous confortent dans notre capacité à embarquer ces algorithmes récents d'apprentissage profond à bord des bus.
Nos travaux futurs consistent à inférer et évaluer le comptage des passagers à partir de leurs trajectoires, ainsi qu'à porter les algorithmes de notre étude sur une plateforme embarquée pour une validation *in situ*.

Remerciements

Je remercie l'ANRT pour son support financier dans le cadre de ma convention de thèse CIFRE chez ACTIA Automotive et au LAAS-CNRS.

Références

[1] V. Letshwiti et T. Lamprecht, Appropriate technology for automatic passenger counting on public transport vehicles in South Africa, *Proc. of the* 23^{rd} *Southern African Transport Conference (SATC)*, 2004.

[2] J. He et A. Arora, A regression-based radar-mote system for people counting, *Int. Conf. on Pervasive Computing and Communications (PerCom)*, 2014.

[3] M. Rauter, Reliable human detection and tracking in top-view depth images, *Int. Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2013.

[4] Z. Chen et al., SVM based people counting method in the corridor scene using single-layer laser scanner, *IEEE* 19^{th} *Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2016.

[5] M. K. Kocamaz et al., Vision-based counting of pedestrians and cyclists, *Winter Conf. on Applications of Computer Vision (WACV)*, 2016.

[6] Z. Yu et al., Pedestrian counting based on spatial and temporal analysis, *Int. Conf. on Image Processing (ICIP)*, 2014.

[7] B. Benfold et I. Reid, Stable multi-target tracking in real-time surveillance video, *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[8] C. Gao et al., People counting based on head detection combining Adaboost and CNN in crowded surveillance environment, *Neurocomputing*, 2016.

[9] A. Krizhevsky et al., ImageNet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[10] O. Russakovsky et al., ImageNet Large Scale Visual Recognition Challenge, *Int. Journal of Computer Vision (IJCV)*, 2014.

[11] S. Ren et al., Faster R-CNN : towards real-time object detection with region proposal networks, *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[12] J. Dai et al., R-FCN : object detection via region-based fully convolutional networks, *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[13] W. Liu et al., SSD : Single Shot Multibox Detector, *European Conference on Computer Vision (ECCV)*, 2016.

[14] J. Redmon et A. Farhadi, YOLO9000 : Better, Faster, Stronger, *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[15] J. Huang et al., Speed/accuracy trade-offs for modern convolutional object detectors, *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] N. Wojke et al., Simple Online and Realtime Tracking with deep association metric, *Int. Conf. on Image Processing (ICIP)*, 2017.

[17] A. Milan et al., MOT16 : a benchmark for multi-object tracking, *arXiv :1603.00831v2 [cs.CV]*, 2016.

[18] M.D. Zeiler et R. Fergus, Visualizing and understanding convolutional networks, *European Conference on Computer Vision (ECCV)*, 2014.

[19] K. He et al., Deep residual learning for image recognition, *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[20] C. Szegedy et al., Rethinking the Inception architecture for computer vision, *Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[21] A.G. Howard et al., MobileNets : efficient convolutional neural networks for mobile vision applications, *arXiv :1704.04961v1 [cs.CV]*, 2017.