Détection d'objets en temps réel : Entraînement de réseaux de neurones convolutifs sur images réelles et synthétiques

T Zgheib, H Borges, V Feldman, T Guntz, C Di Loreto, O Desmaison, F Corduant

Détection d'objets en temps réel : Entraînement de réseaux de neurones convolutifs sur images réelles et synthétiques

T. Zgheib[1], H. Borges[1], V. Feldman[1], T. Guntz[1], C. Di Loreto[1], O. Desmaison[1], F. Corduant[1]

[1] KAIZEN Solutions, KZS LAB, 38330 Montbonnot-Saint-Martin

taline.zgheib@kaizen-solutions.net

Résumé

La détection des objets a prouvé son efficacité dans diverses applications industrielles. La réussite de l'entraînement d'un détecteur d'objets dépend largement de la disponibilité de jeux d'images réelles, qui sont souvent rares et dépendent de l'application visée. Cependant, il est possible de générer des images synthétiques pour l'entraînement via des modèles 3D des objets d'intérêt. Dans cet article, nous étudions les performances de deux architectures de réseaux neurones : MobileNetV2-SSD et ResNet50-SSD. Trois scénarios d'entraînement sont considérés : i) Entraînement ER sur des images réelles, ii) E1 sur des images synthétiques et iii) E2 sur des images synthétiques avec un fine-tuning sur des images réelles. Nos résultats montrent la baisse de la précision moyenne (mAP) lorsque l'entraînement est effectué sur des images synthétiques. Cette précision s'améliore lorsque l'entraînement est suivi d'un fine-tuning sur des images réelles.

Mots-clés

Détection d'objets, réseau de neurones convolutifs, smartphone, temps réel, images synthétiques.

Abstract

Object detection has been proven to be very effective in various industrial applications. Successfully training an object detector relies on real image datasets that are scarce and heavily dependent on the targeted application. However, creating synthetic image datasets for training is possible using 3D models of the objects of interest. Here, we study the performance of two Neural Network architectures : MobileNetV2-SSD and ResNet50-SSD. Three training scenarios are considered i) ER training on real images, ii) E1 training on synthetic images and iii) E2 training on synthetic images and finetuning on real images. Our results show a decrease in the mean average precision (mAP) when the models are trained on synthetic datasets compared to real image datasets. This precision improves when the training is finetuned with real images.

Keywords

Object detection, convolutional neural networks, smartphone, real time, synthetic images.

1 Introduction

La détection d'objet est une technique de vision par ordinateur qui consiste à identifier la présence et l'emplacement de toute instance d'une classe d'objets présente dans l'image (via une boîte englobante associée à une classe d'objets). Cette technique est devenue en quelques années un outil précieux dans des domaines très divers de l'industrie, comme par exemple : la classification et tri des déchets [32], la détection des défauts [31], etc. En effet, le secteur industriel peut bénéficier de la vision par ordinateur et particulièrement la détection d'objet pour automatiser plusieurs tâches. Ceci nécessite souvent le traitement d'images en temps réel sur des appareils embarqués ou mobiles qui sont plus limités en mémoire et en capacité de calcul [3].

Les algorithmes d'apprentissage automatique pour la détection d'objets sont largement utilisés dans l'apprentissage supervisé et peuvent résoudre de nombreux problèmes pratiques [20, 18]. Cependant, les réseaux neuronaux convolutifs (*CNN*) sont à l'heure actuelle plus performants dans le traitement d'images et la détection d'objets [34]. L'entraînement de modèles de réseaux de neurones est réalisé sur une base de données d'images annotées. Dans un contexte industriel, ceci pose beaucoup de contraintes liées à la disponibilité et l'accès aux données nécessaires. En effet, une base de données d'images réelles annotées est souvent inexistante, rare ou encore trop petite pour un entraînement efficace du CNN. De plus, la diversité des objets à détecter implique la génération d'une base de données spécifique pour chaque application. Ceci représente des contraintes temporelles et financières non négligeables pour l'industrie. Pour répondre à cette problématique, il est possible d'utiliser des techniques d'augmentation de données qui améliorent la taille et la qualité des ensembles de données d'entraînement de manière à pouvoir mieux entraîner les réseaux neuronaux profonds [30, 23]. Une seconde méthode consiste à générer automatiquement des données synthétiques annotées grâce aux modèles 3D des objets à détecter [25].

Nous étudions dans cet article les performances de deux détecteurs d'objets (MobileNetV2-SSD et ResNet50-SSD) entraînés sur des données réelles et synthétiques. Nous évaluons également l'influence de la qualité des données synthétiques générées sur la qualité et la performance des mo-

dèles choisis. Les modèles sont évalués sur des bases de données réelles et synthétiques que nous avons générées.

2 Travaux antérieurs

2.1 Architecture générale d'un détecteur d'objet

L'architecture typique d'un détecteur d'objet comprend trois parties : l'épine dorsale (*backbone*), le cou (*neck*) et la tête de détection (*detection head*). L'épine dorsale a pour rôle d'extraire les caractéristiques des images d'entrée. Les cartes de caractéristiques (appelées parfois cartes d'activations) générées sont ensuite introduites dans le cou du détecteur, responsable de l'extraction des caractéristiques pertinentes et de leurs fusions à multi-échelle. Les caractéristiques passent finalement par la tête de détection pour la classification finale et la régression des boîtes englobantes générées.

2.2 Extracteurs de caractéristiques

L'extraction de caractéristiques est une étape cruciale pour la détection d'objet. En effet, le choix de l'architecture utilisée a un impact sur la performance du modèle (vitesse d'inférence et précision). Les réseaux de neurones profonds peuvent apprendre et transmettre des caractéristiques complexes à plusieurs niveaux à partir de données d'entrée. Cependant, à mesure que les réseaux de neurones deviennent de plus en plus profonds, une saturation puis une dégradation de la précision se produisent. Ainsi des équipes de recherche [7] proposent les réseaux résiduels *ResNet* pour faciliter l'entraînement des réseaux de neurones très profonds et résoudre le problème de la dégradation. Ils introduisent la notion des connexions résiduelles entre les couches des réseaux neurones pour éviter le problème de disparition de gradients [7] (*vanishing gradient*). En effet, la rétropropagation du gradient dans un réseau très profond peut rendre le gradient infiniment petit. La connexion résiduelle proposée effectue des raccourcis qui permettent le passage des matrices d'activation d'une couche N à une couche plus profonde $N + k$. Cette cartographie d'identité et plus précisément les connexions résiduelles permettent la transmission du gradient à travers les couches les plus profondes, réduisant ainsi le problème de disparition de gradient [29].

Nos recherches s'intéressent aux réseaux de neurones applicables dans un contexte industriel où la vitesse d'inférence doit être proche du temps réel. Des réseaux récents ont été développés dans cette optique : Les *MobileNet* [9]. Ils offrent un compromis efficace entre vitesse et précision ainsi que la possibilité d'adapter la taille du réseau en fonction des besoins de l'utilisateur et de l'application proposée. L'architecture MobileNet introduit pour la première fois la notion de convolution séparable en profondeur qui vient replacer la convolution classique dans les réseaux de neurones. Elle permet ainsi une diminution conséquente en coût de calcul sans dégrader la précision du modèle. En 2018, MobileNetV2 [22] propose des blocs résiduels inversés (*inverted residual blocks*) et des goulots d'étranglements linéaires (*linear bottelneck*), offrant au réseau la possibilité d'apprendre sur des fonctions plus riches tout en réduisant la taille mémoire.

2.3 Détecteurs d'objets

Il existe deux approches pour la détection d'objet : l'approche à une passe et celle à deux passes. Les approches convolutives traditionnelles à deux passes (*two-stage detectors*, telles que R-CNN [5], SPP-net [6], FPN [12]) extraient les régions les plus susceptibles de contenir un objet (régions d'intérêt) [33, 35] lors d'une première passe. Ensuite, la seconde passe analyse ces régions et prédit à la fois la probabilité de présence d'un objet ainsi que sa classe d'appartenance. Ces méthodes sont certes très précises mais aussi fastidieuses et lentes. Des équipes ont alors développé des approches dites "en une passe" (*one stage detectors*, telles que SSD [14] et YOLO [21]), qui s'avèrent être plus rapides et plus adaptées pour la détection en temps réel. Ces modèles analysent l'image, localisent et identifient les objets en une seule passe [33, 35].

2.4 Entraînement sur des images synthétiques et réelles

L'entraînement des réseaux de neurones convolutifs nécessite l'utilisation d'une base de données synthétiques/réelles conséquente, or la taille de la base de données est un critère très important à considérer pour la performance du modèle. Cependant dans la pratique, ces bases de données sont le plus souvent rares ou bien ne répondent pas toujours aux exigences de taille et de qualité nécessaires pour un tel entraînement. Puisque générer un très grand nombre de données réelles engendrerait un sur-coût financier non négligeable, d'autres approches d'augmentation de données sont donc proposées pour créer des images synthétiques. Certaines méthodes reposent sur la modélisation du contexte visuel [4] et l'apprentissage automatique contradictoire[28] (*adversarial learning*) pour produire des images composites réalistes. D'autres utilisent la randomisation de domaine [24, 25] pour générer des images d'entraînement avec suffisamment de variations pour minimiser le sur-apprentissage.

Avoir une base de données de qualité est indispensable à l'entraînement des réseaux neurones. En effet, la littérature souligne l'importance d'utiliser un jeu de données équilibré où toutes les classes d'objets sont représentées de manière homogène. De même, la diversité de taille et de position des objets ainsi que le nombre de distracteurs impactent la qualité de l'apprentissage du modèle [24]. Par exemple, prendre en compte l'occlusion partielle des objets rend le modèle plus robuste au sur-apprentissage [1, 24]. Cependant, une étude montre que l'entraînement sur des images réelles reste nécessaire pour obtenir des performances acceptables [17].

3 Méthodologie

3.1 Architecture des modèles

Pour réaliser la détection d'objets, nous nous sommes appuyés sur le réseau de neurones Single Shot MultiBox Detector (SSD) [14]. SSD est un détecteur en une passe entièrement convolutif. Il permet une détection à multi-échelles facilitant ainsi la détection d'objets de tailles différentes [14]. SSD s'appuie sur l'architecture VGG-16 pour l'extraction de caractéristiques. Des couches supplémentaires convolutives succèdent le *backbone* et font partie du *neck* du détecteur. Elles ont pour rôle de réduire progressivement la taille des matrices d'activation. Enfin, la couche de la tête du détecteur prend en entrée des cartes de caractéristiques du neck et de l'extracteur. Dans notre application, on remplace VGG-16 par les réseaux MobileNetV2[22] et ResNet50 [7] pré-entrainés sur la base de données COCO [13]. Cette procédure est appelée "transfert d'apprentissage" et a pour avantage de réduire le temps d'exécution de l'algorithme et la quantité d'images utilisées pour *le fine-tuning* du modèle. Les modèles pré-entraînés sont fournis par le *TensorFlow 2 Détection Model Zoo* [1].

Le *neck* du modèle reste celui présenté dans l'article présentant le modèle SSD. A noter que ResNet50 désigne une architecture de réseau résiduel d'une profondeur de 50 couches.

3.2 Génération des images synthétiques et réelles

Pour entraîner les modèles de détection nous avons généré deux jeux de données réelles et synthétiques. Les deux jeux de données réelles sont composés respectivement de 100 (nommé R100) et 400 (R400) images. Ces données réelles sont des photos prises par l'équipe en conditions réelles (Figure 1b). Les deux jeux de données synthétiques SV1 et SV2 sont quant à eux respectivement composés de 2000 et 7500 images. Pour générer ces jeux de données synthétiques, nous avons adapté la technique décrite par [27, 26] qui proposent de créer des images synthétiques à partir de modèles 3D, d'images de fond réelles et en faisant varier plusieurs paramètres.

Les images synthétiques sont générées avec le moteur de jeu Unity3D en modifiant le fond et l'orientation des objets pour SV1 (Figure 1c) et l'inclinaison du fond, la lumière et la texture des objets pour SV2 (Figure 1d). Voir Table 1 pour plus de détails sur les différents paramètres choisis pour la génération de SV1 et SV2.

Les objets à détecter sont des briques Lego de trois formes différentes : 2x1, 2x2 et 4x1 (Figure 1a). Les images synthétiques sont générées en utilisant les modèles 3D des différents distracteurs et briques de Lego à positionner dans l'image. Nous avons choisi trois types de distracteurs : sphère, cube et cylindre. Leurs nombres par image varient aléatoirement et leurs textures sont modifiées en utilisant des images issues du jeu de données Flickr 30k. Le même jeu de données est utilisé pour modifier aléatoirement les textures des briques Lego et les images de fond. De même, Unity nous permet aussi de varier plusieurs paramètres de façon aléatoire : le positionnement et orientation des objets, orientation des caméras, le nombre de sources lumineuses, etc. Tous ces paramétrages ont pour objectif de limiter le sur-apprentissage du modèle lors de l'entraînement. Les Figures 1c et 1d présentent deux images synthétiques type utilisées pour l'entraînement des modèles étudiés. Les annotations (boîtes englobantes et labels) des images synthétiques ainsi produites ont été générées automatiquement au format adapté pour le framework TensorFlow.

TABLE 1 – Récapitulatif des paramètres de génération des jeux de données d'images synthétiques SV1 et SV2

Paramètres	**SV1**	**SV2**
Nombre d'objets d'intérêts : 3 briques de Lego (e.g. 2x1,2x2,2x4)	X	X
Position et orientation aléatoire de l'objet d'intérêt dans la scène	X	X
Type de distracteurs : Sphère, cylindre, cube. Variation aléatoire de la position/rotation	X	X
Variation de la position (Z fixe) et de la rotation (X,Y,Z) de l'angle de la caméra	X	X
Variation du nombre de lumières (entre 0 et 12), leur intensité, portée et angle		X
Variation de la texture de l'objet d'intérêt en utilisant une image aléatoire de Flickr 30k		X
Inclinaison et translation de l'image de fond		X

3.3 Entraînement et évaluation des modèles

Pour le transfert de connaissance nous choisissons les modèles MobineNetV2-SSD (2.5 millions de paramètres) et ResNet50-SSD (31 millions de paramètres) pré-entraînés avec une résolution d'image de 640x640 pixels. Les performances des modèles choisis sont disponible dans la Table 2.

Pour chacun des modèles, nous proposons huit entraînements différents répartis sur trois scénarios (Table 3) :

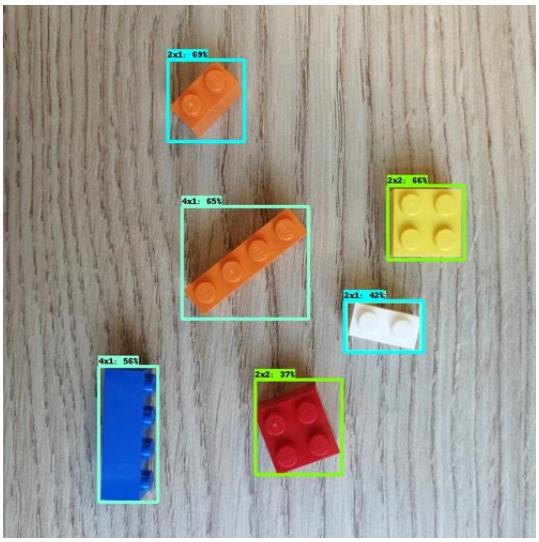
i) Scénario de référence (ER) : Deux entraînements exclusivement sur des images réelles. Les jeux de données utilisés sont R100 et R400.

ii) Scénario E1 : Deux entraînements sur uniquement des images synthétiques. Les jeux de données utilisés sont SV1 et SV2.

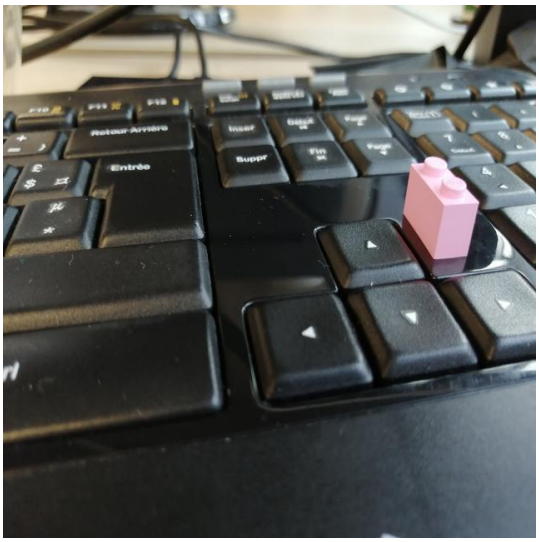
iii) Scénario E2 : Quatre entraînements sur images synthétiques suivi d'un *fine-tuning* sur images réelles. Pour ce scénario quatre combinaisons de jeux de données sont utilisées : SV1+R100, SV2+R100, SV1+R400 et SV2+R400.

Les deux architectures sont optimisées par l'algorithme de descente de gradient stochastique (SGD, *stochastic gradient descent)* avec une taille de *batch* de 4 images, un *lear-*

1. shorturl.at/hmpKQ

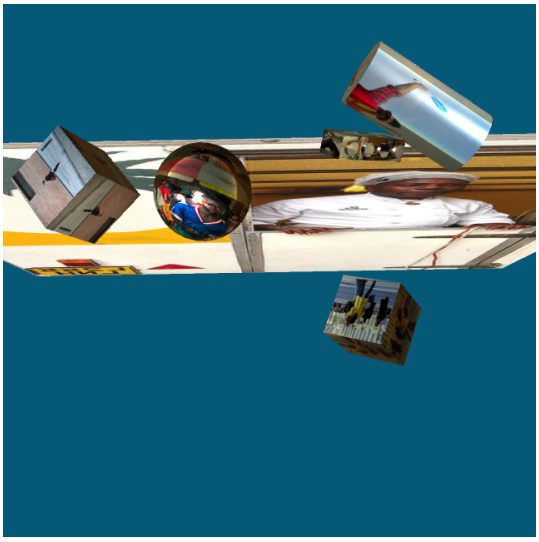


(a) Image réelle avec les différentes boîtes englobantes de 3 types de briques Lego (2x2, 2x1 et 4x1).



(b) Image réelle (jeu de données R100) prise par l'équipe contenant un Lego 2x1.

(c) Image synthétique (jeu de données SV1) générée avec notre approche contenant 2 distracteurs et un Lego 4x1.



(d) Image synthétique (jeu de données SV2) générée avec notre approche contenant 4 distracteurs et un Lego 4x1.

FIGURE 1 – Images réelles (a,b) et synthétiques (c, d) générées et utilisées pour l'entraînement des différents modèles étudiés.

ning rate de 0.01 et un total de 30 epochs pour les scénarios ER et E1. Pour le scénario E2 les modèles sont entraînés sur 20 epochs d'images synthétiques et 10 epochs d'images réelles avec respectivement un *learning rate* de 0.01 et de 0.001. La fonction perte utilisée pour la localisation est celle proposée par défaut, c'est-à-dire une fonction Smooth L1 (voir Hubert [10]). Tous les autres hyperparamètres des deux architectures sont ceux fournis par *TensorFlow 2 Détection Model Zoo*, nous conservons leur valeur par défaut.

L'entraînement et la validation des modèles sont réalisés sur une machine locale avec 32GB de RAM, un processeur Intel Xeon E5-1650 v2 et une carte Nvidia GTX1080. Les modèles sont validés sur 500 images réelles distinctes de celles utilisées pour les différents entraînements (R100 et R400). La précision moyenne (mAP, *mean Average Precision*) est utilisée comme métrique d'évaluation des performances pour les différents scénarios.

TABLE 2 – Performances des modèles sur le Dataset COCO 2017 pour des images de dimensions 640x640 pixels (obtenus sur TFHub)

Modèle	**Inférence (ms)**	**mAP**	**Nombre de paramètres (M)**
MobileNetV2-SSD	39	28.2	2.5
ResNet50-SSD	46	34.3	31

TABLE 3 – Récapitulatif des scénarios d'entraînement pour MobileNetV2-SSD et ResNet50-SSD.

Scénarios	**Jeux de données**
(ER) : Entraînement de référence	R100 et R400
(E1) : Entraînement sur images synthétiques	SV1 et SV2
(E2) : Entraînement sur images synthétiques + fine-tuning sur images réelles	SV1+R100, SV2+R100, SV1+R400, SV2+R400

4 Résultats

La figure 2 présente la précision moyenne (mAP) de la détection pour les différentes analyses. Les résultats montrent que, tant pour MobilenetV2-SSD que pour ResNet50-SSD, la précision moyenne diminue lorsque l'on utilise des données entièrement synthétiques au lieu de données réelles. Pour Resnet50-SSD, la précision passe de 43% en moyenne pour des données réelles à 30% pour des données synthétiques. En revanche, pour Mobilenetv2-SSD, la baisse de la précision moyenne n'est que de 3 % (de 46%(scénario ER) à 41%(scénario E1)).

L'entraînement sur des données synthétiques suivi d'un ajustement fin sur des données réelles (scénario E2) augmente la précision de MobileNetV2-SSD à 47% (contre 41% pour scénario E1). Ainsi, l'utilisation de données réelles pour le *fine-tuning* augmente la mAP de MobileNetV2-SSD à des niveaux comparables à ceux de l'entraînement ER. De même, pour ResNet50-SSD, la mAP augmente de 31% pour le scénario E1 à 43% pour le scénario E2 (entraînement sur images synthétiques + *fine-tuning* sur images réelles). Notre résultat sur MobileNetv2-SSD et Resnet50-SSD est conforme à celui de Nowruzi *et al.* [17] qui suggèrent une augmentation de la précision moyenne quand l'entraînement sur des données synthétiques est suivi d'un *fine-tuning* sur les images réelles. De plus, Nowruzi *et al.* [17] montrent qu'en diminuant la proportion de données réelles utilisées pour le *fine-tuning*, nous dégradons la performance du modèle. Contrairement à cette dernière hypothèse, nous n'avons constaté aucun changement majeur dans les performances de notre modèle. L'ajustement sur 100 ou 400 images réelles n'a pas d'impact sur la précision de ResNet50 et augmente celle de MobileNet de seulement 3% (45% (SV2+R100) à 48% (SV2+R400)).

Notons que l'entraînement sur un plus grand jeu de données synthétiques pénalise la précision des modèles. En effet la mAP est diminuée de 39% (SV1) à 23% (SV2) pour Resnet50-SSD et de 45% (SV1) à 38% (SV2) pour MobilenetV2-SSD. Ceci est directement lié au choix des paramètres à faire varier lors de la création des données synthétiques. En effet, la complexité des variations des paramètres choisis pour la génération de SV2 par rapport à SV1 (Table 1) complique l'apprentissage et dégrade la qualité du jeu de données. Nous suspectons que l'inclinaison du fond (Figure 1c) (source de zones uniformes de couleur aléatoire dans les images synthétiques), la texture ajoutée au Lego ainsi que les reflets lumineux ajoutent une trop grande variabilité des objets d'intérêts et rend l'apprentissage difficile. Toutefois nous obtenons des résultats surprenants pour Resnet50-SSD, dont la précision diminue de 44% à 42% pour un entraînement sur 400 images réelles. Des analyses supplémentaires devraient être menées pour déterminer les causes plausibles qui permettraient de justifier un tel résultat.

De façon générale, MobilenetV2-SSD a une meilleure précision que Resnet50-SSD pour les trois scénarios que nous avons examinés dans cette étude. Cette différence entre les deux modèles indique que MobilenetV2 a une meilleure capacité de généralisation aux images réelles que Resnet50. Ceci peut s'expliquer par le nombre de paramètres de MobilenetV2-SSD qui est inférieur à celui de ResNet50-SSD (Table 2), ce qui limite l'apprentissage de caractéristiques trop spécifiques aux images synthétiques lors de l'entraînement[3].

Les objets étudiés au cours de cette étude ont des caractéristiques semblables et relativement simples. Nous nous interrogeons ainsi sur la généralisation de nos résultats à différents types d'objets, aux textures plus complexes et aux formes irrégulières. En général, l'apparence réaliste d'un objet dépend de ses caractéristiques de bas niveau (forme, pose, texture, arrière-plan, *etc.*). L'étude de *peng et al.* [19] montre que la sensibilité à la pose, texture et l'arrière-plan de l'objet peut être surmontée lorsque l'entraînement du réseau de neurones (sur les données synthétiques) est suivi d'un *fine-tuning* sur des images réelles. Ainsi, si l'objet à détecter possède une texture complexe, il ne sera pas nécessaire de reproduire la complexité de cette texture lors de la création du jeu de données synthétiques. Cependant, si le réseau n'est pas affiné sur des images réelles, l'invariance des indices de bas niveau n'est plus valable. Néanmoins, si la texture est un facteur décisif dans la classification/détection de l'objet, une représentation réelle de la texture devrait être considérée. Par ailleurs, la forme de l'objet reste un critère important et indispensable pour la majorité des modèles qui se focalise sur les formes et les limites des objets pour l'apprentissage [16]. En conséquence, le résultat ne dépend pas de l'objet à détecter, mais *i)* du modèle choisi, *ii)* de la qualité de l'ensemble de données synthétiques utilisé et *iii)* du degré auquel l'objet est représenté. Il est alors préférable d'expérimenter avec le modèle de détection choisi afin d'identifier les aspects des objets qui méritent l'investissement nécessaire pour les représenter dans un jeu de données synthétiques de bonne qualité.

5 Conclusion et futurs travaux

L'approche proposée dans cet article permet de réaliser l'entraînement d'un détecteur d'objet sur trois différents type de briques Lego, avec très peu de données réelle annotées en notre possession. Nos premiers résultats montrent que nous pouvons entraîner un détecteur d'objet unique-

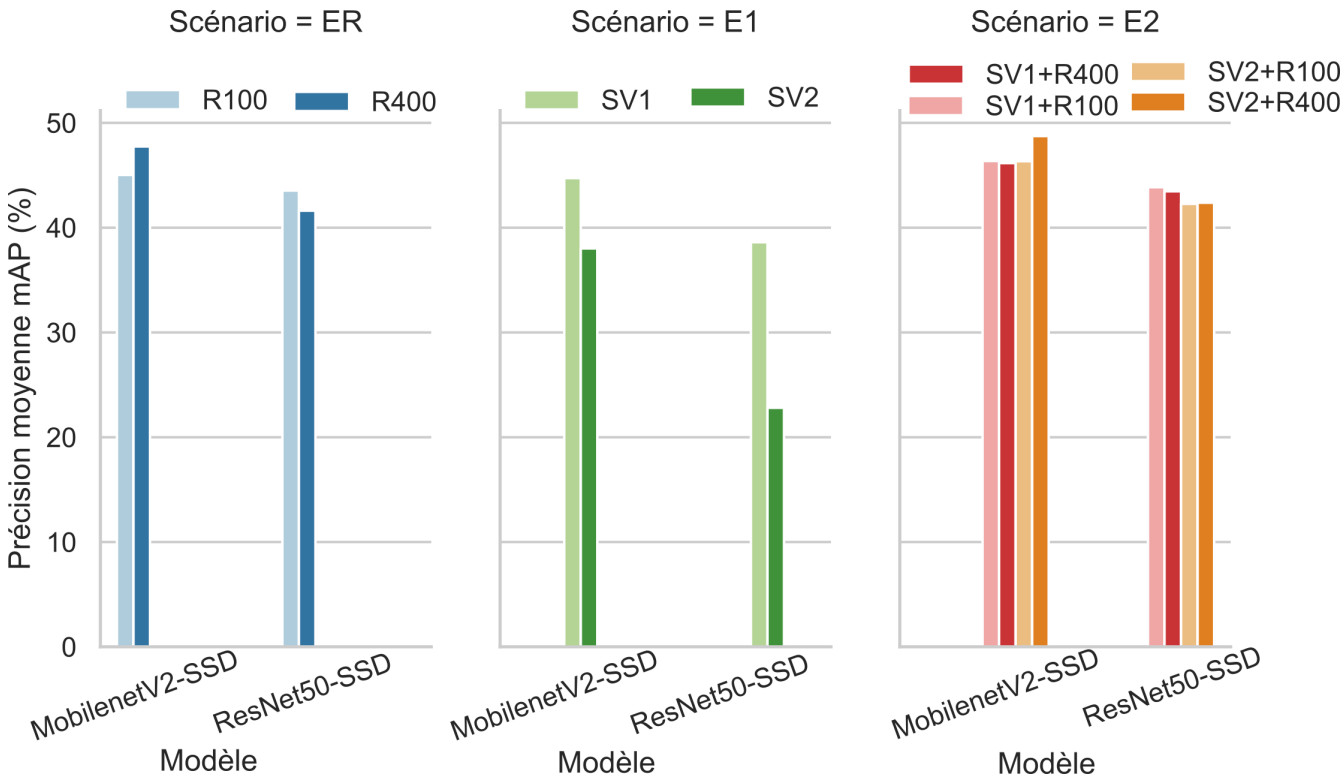


FIGURE 2 – Performance des modèles MobileNetV2-SSD et ResNet50-SSD pour les scénarios *i)* ER : entraînement sur images réelles, *ii)* E1 : entraînement sur images synthétiques et *iii)* E2 : entraînement sur images synthétiques suivi d'un *fine-tuning* sur images réelles. R100 et R400 sont les jeux de données d'images réelles avec respectivement 100 et 400 images. SV1 et SV2 sont les jeux de données synthétiques avec respectivement 2000 et 7500 images.

ment sur des données synthétiques et obtenir des performances satisfaisantes (39% pour MobileNetV2 et 45% pour ResNet-SSD, sur SV1), ce qui confirme le résultat de plusieurs autres études, comme Tremblay *et al.* [25]. Si nos résultats laissent à penser qu'utiliser seulement R100 ou R400 semble suffisant, nous pensons que l'utilisation d'une base de données hybride, combinant données réelles et synthétiques, est une approche très intéressante pour les petites équipes de recherche ou les industriels. En effet, la génération d'une telle base, par exemple 75% de données synthétiques et 25% de données réelles, est beaucoup moins chronophage et coûteuse. Cependant, de nouvelles expériences doivent être menées, cette fois-ci avec des objets à forme plus complexes que celle des Lego. Cela permettrait de confirmer la viabilité de cette approche à généraliser sur des cas plus proches des problématiques rencontrées en environnement industriel. Ces premiers résultats nous encouragent à poursuivre nos travaux en explorant plusieurs axes, notamment sur la qualité des données générées, l'optimisation du modèle et l'amélioration des performances de détection.

Tout d'abord, nous souhaitons étudier plus en détail l'impact du ratio données synthétiques/réelles dans une base de données hybride sur les performances des modèles de détection, de la même manière que cela est présenté dans les travaux de Nowruzi *et al.* [17] et de Borkman *et al.* [2]. Cela nous permettrait de déterminer la proportion minimale de données réelles nécessaires pour obtenir de bonnes performances.

Ensuite, pour expliquer les différences de performances entre les différentes stratégies de générations de données (entre SV1 et SV2), nous souhaitons mieux contrôler les variations de paramètres de ces stratégies. Une première piste est l'utilisation de la librairie Perception de Unity [2] pour générer des images synthétiques. En comprenant mieux l'influence de la variation des paramètres, nous espérons réduire l'écart de domaine entre les données réelles et synthétiques. Enfin, il serait intéressant d'étudier l'influence du photo-réalisme sur les performances de détection, dont l'efficacité reste une question ouverte dans la communauté [1, 15].

Concernant le modèle de détection, nous envisageons de tester les performances d'autres architectures potentiellement plus adaptées pour la détection d'objet en temps réel (par exemple, une architecture MobileNet-V3 [8] et YOLO-V5 [11]).

Références

[1] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision : Efficient data generation for urban driving scenes. *International Journal of Computer Vision*, 126(9) :961–972, 2018.

[2] Steve Borkman, Adam Crespi, Saurav Dhakad, Sujoy Ganguly, Jonathan Hogins, You-Cyuan Jhang, Mohsen Kamalzadeh, Bowen Li, Steven Leal, Pete Parisi, et al. Unity perception : Generate synthetic data for computer vision. *arXiv preprint arXiv :2107.04259*, 2021.

[3] Julia Cohen, Carlos Crispim-Junior, Jean-Marc Chiappa, and Laure Tougne. Mobilenet ssd : étude d'un détecteur d'objets embarquable entraîné sans images réelles. In *ORASIS 2021*, 2021.

[4] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. Modeling visual context is key to augmenting object detection datasets. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 364–380, 2018.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[6] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37 :1904–1916, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[9] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets : Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv :1704.04861*, 2017.

[10] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, page 492—518. Springer, 1992.

[11] Glenn Jocher, Alex Stoken, Jirka Borovec, A Chaurasia, and L Changyu. ultralytics/yolov5. *Github Repository, YOLOv5*, 2020.

[12] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco : Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[14] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd : Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[15] Nikolaus Mayer, Eddy Ilg, Philipp Fischer, Caner Hazirbas, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. What makes good synthetic training data for learning disparity and optical flow estimation ? *International Journal of Computer Vision*, 126(9) :942–960, 2018.

[16] Sergey I Nikolenko et al. *Synthetic data for deep learning*. Springer, 2021.

[17] Farzan Erlik Nowruzi, Prince Kapoor, Dhanvin Kolhatkar, Fahed Al Hassanat, Robert Laganiere, and Julien Rebut. How much real data do we actually need : Analyzing object detection performance using synthetic and real data. *arXiv preprint arXiv :1907.07061*, 2019.

[18] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International journal of computer vision*, 38(1) :15–33, 2000.

[19] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1278–1286, 2015.

[20] Dominik Maximilián Ramík, Christophe Sabourin, Ramon Moreno, and Kurosh Madani. A machine learning based intelligent vision system for autonomous object detection and recognition. *Applied intelligence*, 40(2) :358–375, 2014.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[22] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2 : Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[23] Uday Shankar Shanthamallu, Andreas Spanias, Cihan Tepedelenlioglu, and Mike Stanley. A brief survey of machine learning methods and their sensor and iot applications. In *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pages 1–8. IEEE, 2017.

[24] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[25] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To,

Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data : Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018.

[26] Jonathan Tremblay, Thang To, and Stan Birchfield. Falling things : A synthetic dataset for 3d object detection and pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2038–2041, 2018.

[27] Jonathan Tremblay, Thang To, Artem Molchanov, Stephen Tyree, Jan Kautz, and Stan Birchfield. Synthetically trained neural networks for learning human-readable plans from real-world demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5659–5666. IEEE, 2018.

[28] Shashank Tripathi, Siddhartha Chandra, Amit Agrawal, Ambrish Tyagi, James M Rehg, and Visesh Chari. Learning to generate synthetic data via compositing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 461–470, 2019.

[29] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

[30] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification : when to warp ? In *2016 international conference on digital image computing : techniques and applications (DICTA)*, pages 1–6. IEEE, 2016.

[31] Jing Yang, Shaobo Li, Zheng Wang, Hao Dong, Jun Wang, and Shihao Tang. Using deep learning to detect defects in manufacturing : a comprehensive survey and current challenges. *Materials*, 13(24) :5755, 2020.

[32] Qiang Zhang, Xujuan Zhang, Xiaojun Mu, Zhihe Wang, Ran Tian, Xiangwen Wang, and Xueyan Liu. Recyclable waste image recognition based on deep learning. *Resources, Conservation and Recycling*, 171 :105636, 2021.

[33] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning : A review. *IEEE transactions on neural networks and learning systems*, 30(11) :3212–3232, 2019.

[34] Haiying Zhou, Xiangyu Yu, Ahmad Alhaskawi, Yanzhao Dong, Zewei Wang, Qianjun Jin, Xianliang Hu, Zongyu Liu, Vishnu Goutham Kota, Mohamed Hasan Abdulla Hasan Abdulla, et al. A deep learning approach for medical waste classification. *Scientific reports*, 12(1) :1–9, 2022.

[35] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years : A survey. *arXiv preprint arXiv :1905.05055*, 2019.