

Détection d'objet

Danny Sauval, *Étudiant, ETS*

Résumé—Avec l'avènement des nouvelles technologies, la société actuelle cherche de plus en plus à automatiser les tâches effectuées par l'être humain. La conduite automobile en est un exemple. Les voitures autonomes sont dotées d'un grand nombre de caméra et capteurs permettant ainsi d'éviter les objets ou de suivre la route. Mais comment font-ils ? Afin d'éviter les objets, la voiture autonome doit être capable de détecter ces objets. Dans ce papier nous nous proposons d'étudier le système Faster R-CNN qui permet de détecter des objets. Nous verrons si ce système est fiable et si il est possible de lui faire confiance pour l'intégrer à un système de voiture autonome.

Mots-clés—Détection d'objet, Faster R-CNN, CNN, classification d'objets, data augmentation, augmentation de données,

I. INTRODUCTION

Àvec l'avènement des nouvelles technologies, la société actuelle cherche de plus en plus à automatiser les tâches effectuées par l'être humain. Les robots dans le domaine industriel permettent de remplacer les opérateurs qui faisaient des tâches dangereuses ou bien pénibles. De même dans le domaine automobile, des entreprises comme *Google* ou *Uber* développent des systèmes de pilotage automatique. Cependant, comment un tel système peut-il être capable d'analyser un environnement afin de pouvoir répondre instantanément à un environnement changeant et toujours différent ? Les êtres humains sont capables de faire cela simplement, mais les machines doivent être dotées d'un grand nombre de capteurs ainsi que d'un système de traitement des données très complexe afin de prendre les bonnes décisions. L'un des principaux fondements d'un système de pilotage automatique est la vision. Une voiture autonome doit être capable de "voir" son environnement et ensuite d'analyser celui-ci afin de détecter les objets pour les éviter et la route pour la suivre. Ce papier présente l'étude expérimentale qui a été effectuée afin de mettre en place un système de détection d'objets. Ce système est basé sur Faster R-CNN que j'ai pu présenter dans ma synthèse de littérature. La section suivante ([section II](#)) présente un rappel du principe de Faster R-CNN.

II. SOMMAIRE DES TECHNIQUES ÉTUDIÉES

A. Introduction, R-CNN et Fast R-CNN

Comme énoncé dans ma synthèse de littérature, Faster R-CNN est une itération de l'algorithme Fast R-CNN lui-même une itération de R-CNN.

Pour rappel, le fonctionnement de R-CNN et Fast R-CNN :

— R-CNN [1] :

- 1) Donnée une image en entrée
- 2) La recherche sélective permet de proposer 2000 régions de l'image

- 3) Un CNN prend chacune de ses régions afin de calculer leur carte de caractéristiques
- 4) Les vecteurs caractéristiques en sortie du CNN sont envoyés à des SVM qui permettent de classifier chacune des régions
- 5) En utilisant la régression, on fusionne les régions qui correspondent au même objet

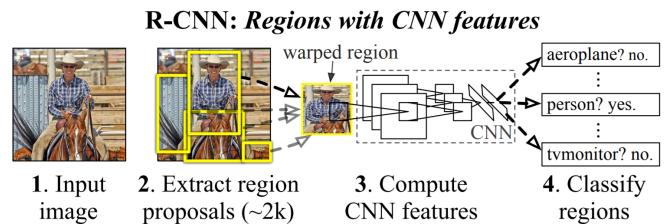


FIGURE 1: Architecture du R-CNN, [1]

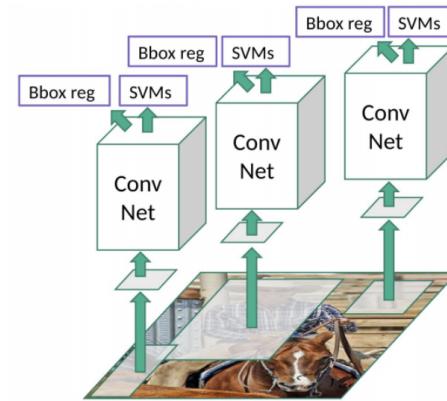


FIGURE 2: Architecture du R-CNN, [2]

— Fast R-CNN [3] :

- 1) Donnée une image en entrée et des propositions de régions (issues de la recherche sélective notamment)
- 2) L'image donnée en entrée est passée à travers un CNN (alternance de couches de convolution et de couches de maxpooling) afin de créer une carte de caractéristiques.
- 3) Pour chaque région proposée par la recherche sélective, on extrait à partir de la feature map le vecteur de caractéristique qui correspond à la région proposée. L'extraction du vecteur de caractéristique est faite avec une couche appelée "region of interest (RoI) pooling layer", en français : couche de pooling de la région d'intérêt.

- 4) Chacun des vecteurs de caractéristiques obtenus à l'étape précédente est envoyé à une séquence de couches entièrement connectées. Celles-ci se terminent en deux branches :
- a) La première donne la probabilité softmax estimée selon K classes, ainsi qu'une classe qui correspond à l'arrière-plan. On connaît ainsi la classe de l'objet.
 - b) La seconde retourne 4 valeurs qui donnent la position et la taille de la bounding box liée à l'objet. On connaît ainsi la bounding box liée à l'objet.

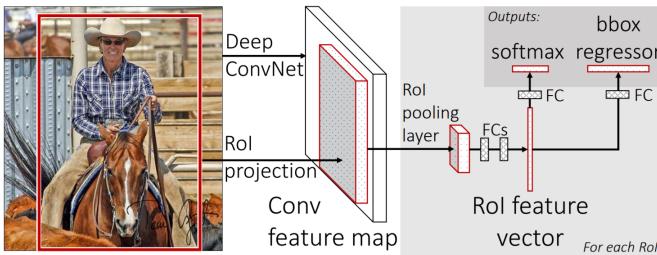


FIGURE 3: Architecture du Fast R-CNN, [3]

B. Diagramme bloc général du système

Dans le cas de notre projet, nous utilisons l'algorithme Faster R-CNN [4]. L'objectif est de remplacer l'utilisation de la recherche sélective qui prend beaucoup de temps. L'algorithme Faster R-CNN est donc identique à celui de Fast R-CNN, ce qui change est uniquement la partie proposition de région qui est gérée par un Region Proposal Network.

Shaoqing Ren et al. se sont rendu compte qu'il était possible et bien plus efficace d'utiliser la carte des caractéristiques issue du CNN en entrée du RPN. L'objectif du RPN est donc de faire passer la carte de caractéristiques à travers d'autres couches. Pourquoi cela est-il intéressant ? Tout simplement parce que les cartes de caractéristiques issues du CNN ont déjà été calculées dans le cadre de Fast R-CNN. Donner ces cartes en entrée au RPN réduit considérablement la lourdeur computationnelle du système.

RPN est en fait un CNN dont les couches sont entièrement connectées entre elles.

- Entrée du RPN : La carte de caractéristique du CNN
- Sortie du RPN : Bounding box d'un potentiel objet avec le score d'objectivité. Le score d'objectivité permet de savoir si la bounding box proposé correspond 1 plus probablement à un objet ou bien à un élément de l'arrière-plan.

Le déroulement de l'algorithme est le suivant :

- 1) On fait glisser un réseau sur la carte des caractéristiques du CNN. Ce réseau prend en entrée une fenêtre de taille $n \times n$ du réseau de neurones. Dans le papier [4] la taille est 3×3 .
- 2) La fenêtre est mappée dans un vecteur de caractéristique de dimension plus basse.

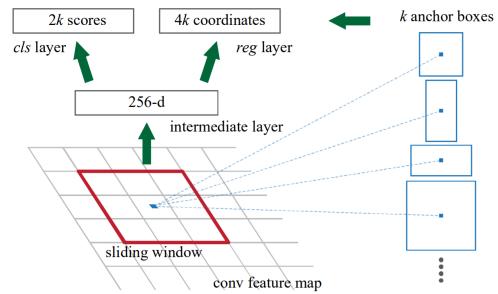


FIGURE 4: Architecture du RPN de Faster R-CNN, [4]

- 3) Le vecteur est ensuite envoyé à deux couches entièrement connectées jumelées :
 - a) Une couche de box-regression → permet d'optimiser la précision de la bounding box pour correspondre aux dimensions réelles de l'objet.
 - b) Une couche de box-classification → permet de classifier le contenu de la bounding box

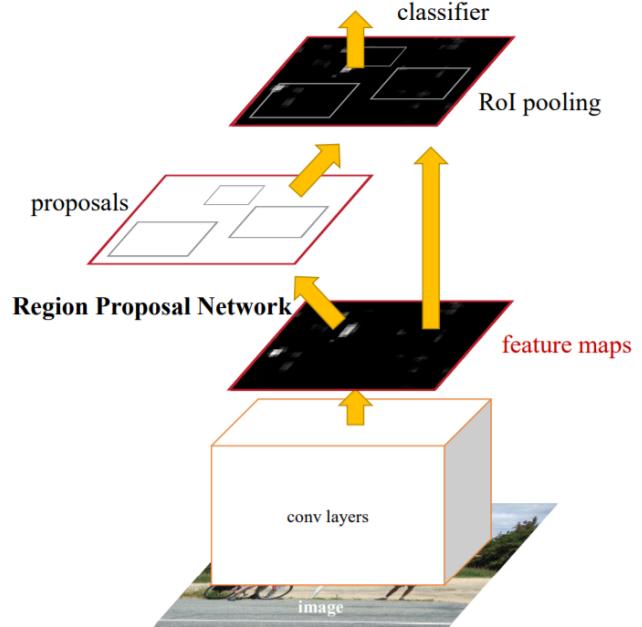


FIGURE 5: Architecture du Faster R-CNN, [4]

Le principe peut être résumé comme suit :

- 1) Donnée une image en entrée
- 2) On envoie l'image à un réseau de neurones qui prédit les régions. Le réseau de neurones est appelé un RPN, Region Proposal Network.
- 3) On envoie ensuite l'image et les régions proposées à Fast R-CNN.

C. Descriptions algorithmiques brèves des techniques qui sont étudiées

- 1) *Augmentation de données:* Ayant une petite base de données, nous avons utilisé la technique de l'augmentation

de données. Le principe est simple, donnée une image en entrée, un générateur de données renvoie en sortie une image modifiée. Dans le cadre de ce projet, le générateur est simple :

- 1) Donnée une image X
- 2) On génère aléatoirement 3 nombres binaires X1, X2, X3
- 3) Si X1 vaut true, appliquer une rotation de 90, 180 ou 270
- 4) Si X2 vaut true, appliquer un effet miroir vertical à l'image
- 5) Si X3 vaut true, appliquer un effet miroir horizontal à l'image
- 6) Retourner l'image générée

Il est possible de générer pour une seule image en entrée un grand nombre d'images. Cependant, choisir un trop grand nombre mène à un overfitting important. Dans le cas de l'algorithme utilisé dans le projet, on ne génère qu'une seule image.

D. Code source

Le code du projet entier peut-être téléchargé à partir de mon Github : https://github.com/DannySauval/Projet_SYS843. Les librairies nécessaires sont : tensorflow, tensorflow-gpu : Très important, sans cela le GPU n'est pas utilisé et l'entraînement est beaucoup trop long, Keras, Pillow, numpy, opencv-python, opencv-contrib-python, pandas, scikit-learn, scikit-image, scipy et sklearn.

En utilisant le dataset de PascalVOC que j'ai adapté à mon projet (voir [subsubsection III-2](#)) :

- 1) Entrainement : exécuter `train.py`. Si le dossier `model` avec le fichier `model_frcnn_vgg.hdf5` et l'historique `record.csv`, l'entraînement reprendra sa progression où je l'ai arrêté (100ème epoch). Dans le dossier `model` il y a également un fichier Matlab, `plotTrainRes.m` qui permet de tracer les courbes de tous les résultats de la phase d'entraînement.
- 2) Test : exécuter `test.py`.

III. MÉTHODOLOGIE EXPÉRIMENTALE

1) Protocole pour évaluer les performances: Comme on pourra le voir, l'entraînement de l'algorithme est très long. Il a donc été très intéressant pour moi de mettre en place différents indicateurs de mesures afin de pouvoir contrôler l'avancement de la performance de mon système en fonction des epochs. A chaque epoch, j'ai écrit dans un fichier `record.csv` : la précision de la classification, le coût de la régression du RPN, le coût de la classification du RPN, le cout de la régression du classificateur, le coût de la classification du classificateur ainsi que le temps d'exécution.

L'efficacité du système a été mesurée en utilisant la précision de la détection obtenue lors de la phase de test, ainsi que la mAP. Le temps d'exécution permettra également de rendre compte de l'efficacité du système.

2) Description de la base de données: La base de données initiale utilisée est la version 2012 de PascalVOC . Cette base de données contient 17.125 images annotés. La structure de la base de données est telle que donnée par la [Figure 6](#).

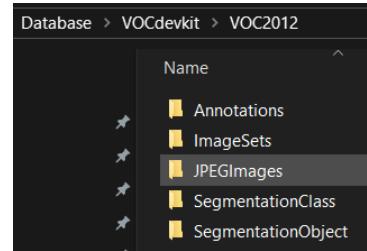


FIGURE 6: Structure du dû dataset PascalVOC 2012 issue du kit de développement

Les données qui nous intéressent sont les images en elles-mêmes ainsi que les annotations associées.

- Le dossier `JPEGImages` contient toutes les images indépendamment de leur classe ou annotation.
- Le dossier `Annotations` contient des fichiers `.xml` contenant des informations telles que le nom du fichier associé, la database associée, les bounding boxes de l'image, ect. Dans notre cas, pour de la détection d'objet avec Faster R-CNN, nous avons seulement besoin de connaître les coordonnées des bounding boxes ainsi que le nom de la classe associé à chaque bounding box. Nous avons également besoin du nom du fichier.

Comme on peut le voir sur la [Figure 7](#), les informations dont on a besoin sont difficilement accessibles. Afin de rendre plus simple la manipulation du dataset, j'ai reformaté toutes ces données à l'aide de Python. Le code que j'ai utilisé pour cela est disponible sur mon [Github](#). Le fonctionnement du code est simple :

- 1) On crée un fichier `annotations.txt`
- 2) En entrée, un fichier XML tel que celui de la [Figure 7](#)
- 3) On lit le nom de l'image associée
- 4) Pour chaque objet dans le xml, les coordonnées de la bounding box
- 5) Pour chaque objet, on écrit dans le fichier `annotations.txt` le nom de l'image associée, `xmin`, `ymin`, `xmax`, `ymax`, le nom de la classe. Les informations sont séparées par des virgules

Une partie du fichier `annotations.txt` peut-être lu sur la [Figure 8](#).

Ce fichier contient ainsi 43,211 bounding boxes associées à une classe. L'algorithme Faster R-CNN a besoin de deux datasets :

- Train dataset : Ce sont les données utilisées pour entraîner Faster R-CNN
 - Test dataset : Ce sont les données utilisées pour tester le modèle obtenus par la phase d'entraînement
- Avant de séparer les données, j'ai créé un [algorithme python](#) qui permet de changer aléatoirement l'ordre

```

<annotation>
  <folder>VOC2012</folder>
  <filename>2007_000027.jpg</filename>
  <source>
    <database>The VOC2007 Database</database>
    <annotation>PASCAL VOC2007</annotation>
    <image>flickr</image>
  </source>
  <size>
    <width>486</width>
    <height>500</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>person</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>174</xmin>
      <ymin>101</ymin>
      <xmax>349</xmax>
      <ymax>351</ymax>
    </bndbox>
    <part>
      <name>head</name>
      <bndbox>
        <xmin>169</xmin>
        <ymin>104</ymin>
        <xmax>209</xmax>
        <ymax>146</ymax>
      </bndbox>
    </part>
    <part>
      <name>hand</name>
      <bndbox>
        <xmin>278</xmin>
        <ymin>210</ymin>
        <xmax>297</xmax>
        <ymax>233</ymax>
      </bndbox>
    </part>
  </object>

```

Objet

Coordonnées de la bounding box

Objet

Objet

FIGURE 7: Structure d'un fichier d'annotation associé à une image

```

2007_000027.jpg,174,101,349,351,person
2007_000027.jpg,169,104,209,146,head
2007_000027.jpg,278,210,297,233,hand
2007_000027.jpg,273,333,297,354,foot
2007_000027.jpg,319,307,340,326,foot
2007_000032.jpg,104,78,375,183,aeroplane
2007_000032.jpg,133,88,197,123,aeroplane
2007_000032.jpg,195,180,213,229,person
2007_000032.jpg,26,189,44,238,person
2007_000033.jpg,9,107,499,263,aeroplane
2007_000033.jpg,421,200,482,226,aeroplane
2007_000033.jpg,325,188,411,223,aeroplane
2007_000039.jpg,156,89,344,279,tvmonitor
2007_000042.jpg,263,32,500,295,train
2007_000042.jpg,1,36,235,299,train
2007_000061.jpg,274,11,437,279,boat
2007_000061.jpg,184,214,281,252,boat

```

1 image
5 objets

1 image
4 objets

1 image
3 objets

1 image
1 objets

FIGURE 8: Extrait du fichier annotations.txt

des lignes du fichier d'annotations afin de sélectionner aléatoirement des bounding boxes d'image différentes. De plus, j'ai choisi d'entraîner mon réseau à ne détecter que 2 classes : voitures (car) et vélo (bicycle). A l'aide d'un [script](#) python j'ai pu ne sélectionner que les classes correspondantes à "car" ou bien "bicyclette" et stocker 80% de ces résultats dans un fichier "trainAnnotations.txt" et 20% dans un fichier "testAnnotations.txt". Un dernier [script](#) m'a permis de répartir les images correspondant aux annotations dans deux dossiers : `trainData` et `testData`.

L'arborescence de ma database est ainsi la suivante :

```

— trainData
  | — 2007_000129.jpg
  | — 2007_000515.jpg
  | — ....
— testData
  | — 2007_000663.jpg
  | — 2007_000727.jpg
  | — ....
— trainAnnotations.txt
— testAnnotations.txt

```

1,423 images pour l'entraînement avec 2424 bounding boxes. 470 images pour les tests avec 605 bounding boxes. On a deux classes : car et bicycle.

Un [algorithme Python](#) disponible sur mon Github permet de sélectionner une image aléatoire dans la database et d'afficher ses bounding boxes.

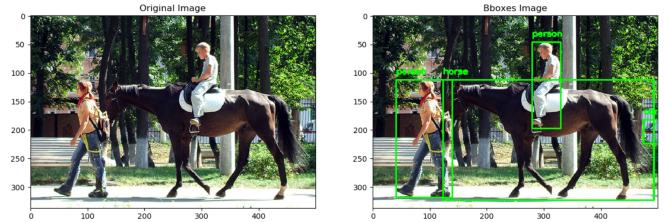


FIGURE 9: Exemple issu de la database

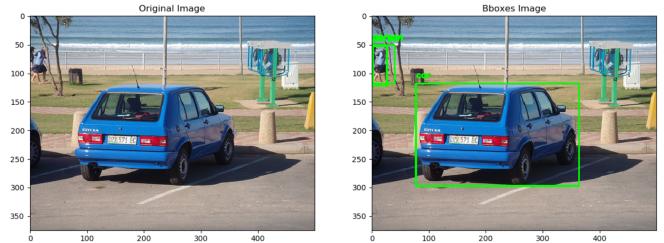


FIGURE 10: Exemple issu de la database

IV. RÉSULTATS DE SIMULATIONS

1) Représentation des résultats: Dans cette section je vais d'abord présenter les résultats que j'ai obtenus avec l'entraînement du réseau. Je parlerais ensuite des résultats obtenus lors de la phase de test.

Lors de l'exécution de l'algorithme, j'ai récupéré dans un fichier `record.csv` l'historique du résultat de chaque epoch :

- Le temps écoulé, [Figure 11](#)
- Le coût du classificateur du *RPN*, [Figure 12](#). La bounding box correspond à un objet ou non ? (classificateur binaire)
- Le coût de la régression du *RPN*, [Figure 13](#). Proposition d'une bounding box.
- Le coût de la classification du classificateur, [Figure 14](#). Quelle est la classe de l'objet ?

- Le coût du régression du classificateur, [Figure 15](#). Précision de la bounding box proposé par le *RPN* de manière approximative.
- Le coût total du modèle, [Figure 16](#)
- Précision du modèle, [Figure 17](#)

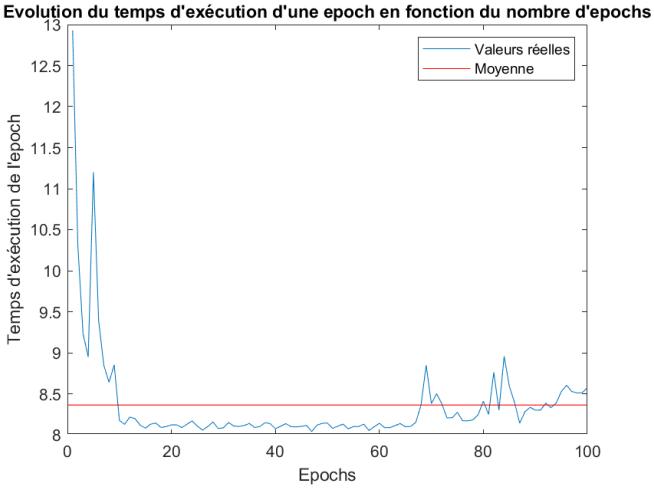


FIGURE 11: Temps d'exécution par epoch, temps total pour les 100 epochs : 14 heures

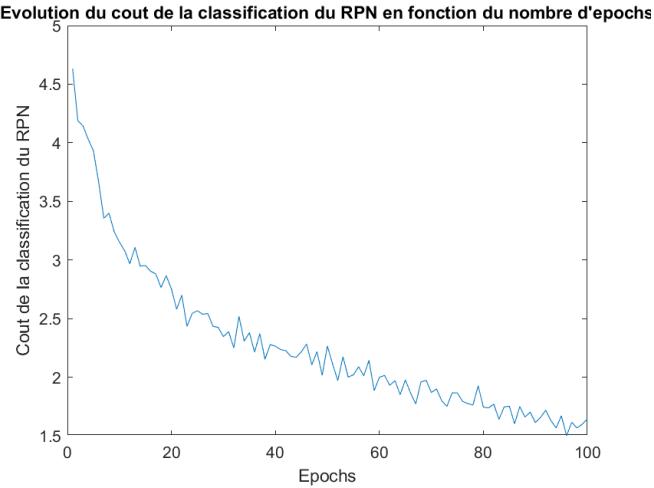


FIGURE 12: Le coût du classificateur du *RPN* en fonction des epochs

Après la phase de test du modèle, j'ai pu obtenir un mAP de 0.15 et une confidence de 71% (pour les bonnes détections) pour la classe car, et un mAP de 0.26 avec une confidence de 86% (pour les bonnes détections) pour la classe bicycle.

Les figures 18 à 24 montrent quelques exemples de résultats, bons ou mauvais que j'ai pu relever grâce à l'[algorithme](#) disponible sur mon Github.

2) *Interprétation des résultats:* Je suis plutôt content de mes résultats. Le modèle ne détecte pas toujours une voiture ou un vélo alors qu'il devrait et les bounding boxes sont assez

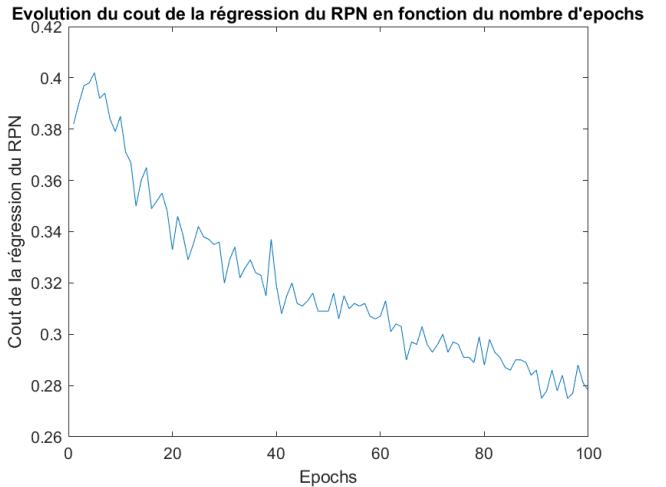


FIGURE 13: Le coût de la régression du *RPN* en fonction des epochs

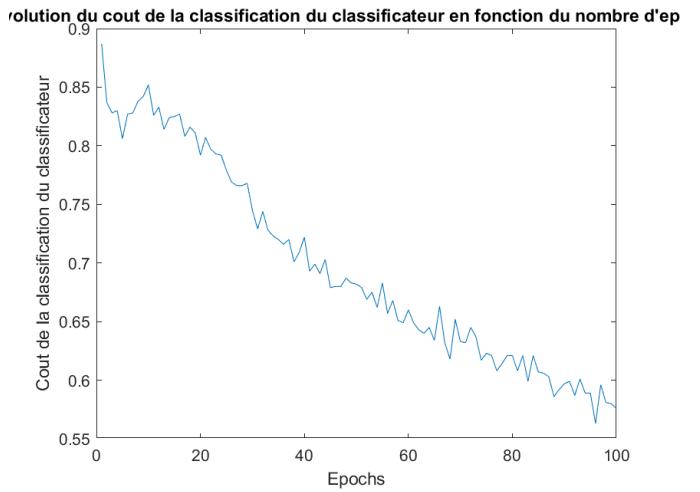


FIGURE 14: Le coût de la classification du classificateur en fonction des epochs

peu précises, mais c'est en accord avec ce que j'ai pu lire dans la littérature. Pour un entraînement de 14h sur un petit dataset comme celui que j'ai utilisé, je trouve que mes résultats sont très bons. De plus, j'ai arrêté l'algorithme (qui était en train de tuer mon PC..) à la 100e epoch alors que comme on peut le voir sur la [Figure 16](#) le coût total décroissait de plus en plus.

- Le temps écoulé, [Figure 11](#). Le temps est stable. On peut tout de même voir des pics où le temps grimpe. On peut expliquer ces pics par le fait que lorsque je lance l'algorithme, mon ordinateur prend un peu de temps avant de vraiment démarrer les calculs et allouer la mémoire nécessaire à l'algorithme. Les pics que l'on retrouve pour les epochs ~ 70 et ~ 82 sont dus à des arrêts de l'algorithme de ma part (et donc suivi d'une reprise).
- Le coût du classificateur du *RPN*, [Figure 12](#).
- Le coût de la régression du *RPN*, [Figure 13](#). Proposition

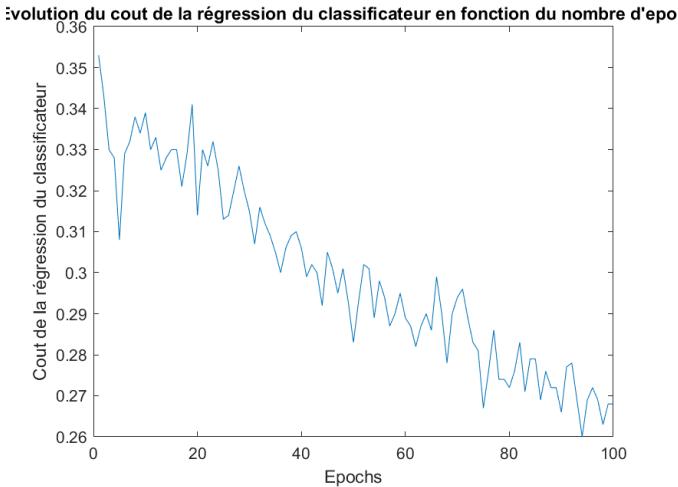


FIGURE 15: Le coût de la régression du classificateur en fonction des epochs

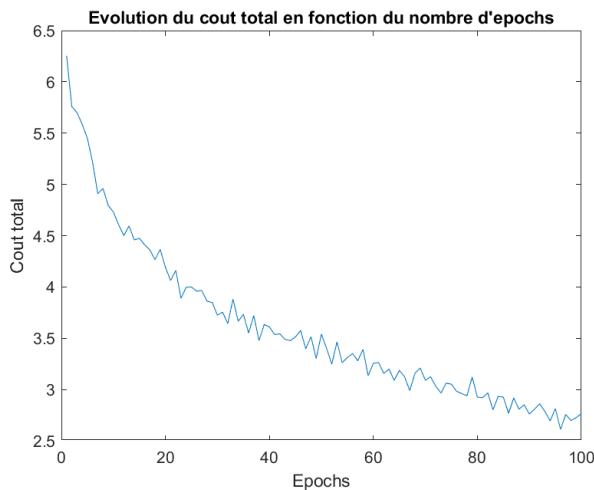


FIGURE 16: Le coût de la régression du classificateur en fonction des epochs

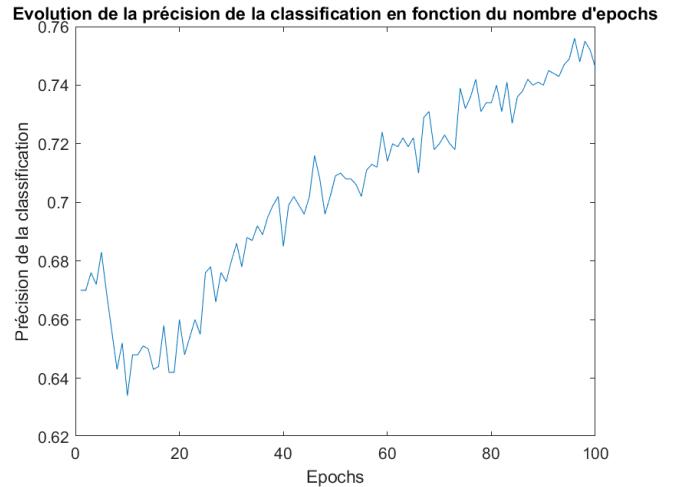


FIGURE 17: Précision de la classification en fonction des epochs



FIGURE 18: Exemple de détection 1



FIGURE 19: Exemple de détection 2

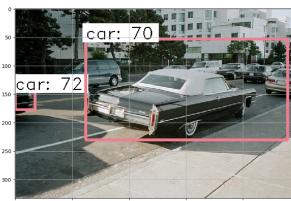


FIGURE 20: Exemple de détection 3



FIGURE 21: Exemple de détection 4

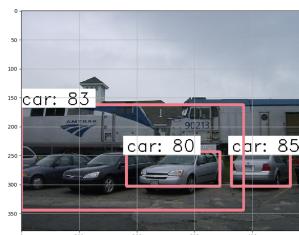


FIGURE 22: Exemple de détection 5



FIGURE 23: Exemple de détection 6

d'une bounding box.

- Le coût de la classification du classificateur, Figure 14. Quelle est la classe de l'objet ?
 - Le coût de la régression du classificateur, Figure 15. Précision de la bounding box proposé par le RPN de manière approximative.
- L'évolution des différents coûts semble se dérouler correctement. Le coût est décroissant bien que légèrement instable. À partir de la 100e epochs on commence à converger. Je pense tout de même qu'il aurait été intéressant d'atteindre les 200 epochs pour obtenir davantage de stabilité.
- Le coût total du modèle, Figure 16. Le coût total démontre notamment que le pas des epochs et le learning rate sont correctement définis. Malheureusement, l'entraînement prenant beaucoup de temps je ne peux pas comparer avec d'autres valeurs. J'avais essayé avec un pas de 100 étapes par epoch (au lieu de 1000 sur ces

résultats) et le système était divergent.

- Précision du modèle, Figure 17. La précision du modèle est correcte, elle augmente constamment. Pour 100 epochs on atteint une précision de 75%.

Les résultats obtenus avec la phase de test sont cohérents par rapport aux courbes précédemment décrites. En effet, la



FIGURE 24: Exemple de détection 7

confidence de la classification tourne autour de la précision donnée par la [Figure 17](#). Le coût de la régression du classificateur est convergent mais instable selon les epochs, ce qui se ressent dans la phase de test. Les bounding boxes ne sont généralement pas de très bonne qualité. De plus, le mAP est très bas. Je pense que cela est dû à la taille du dataset qui mène à un overfitting trop important.

V. CONCLUSION

1) Recommandations pour la suite du projet: Le projet en l'état actuel présente de nombreuses approches afin d'améliorer la précision du système. L'un des principaux problèmes que j'ai pu évoquer précédemment est le côté restreint du dataset. Comme on l'a mentionné dans la [subsection III-2](#), j'ai entraîné et testé le réseau avec un total de 3029 bounding boxes associées à leur classe ce qui est très peu. J'avais au départ choisi d'utiliser le dataset de PascalVOC puisqu'il est utilisé dans tous les papiers que j'ai pu lire vis-à-vis de la détection d'objets.

Mon souhait pour une amélioration potentielle de ce système serait d'avoir le temps de tester avec plusieurs paramètres les résultats du système. Avec 14h d'entraînement pour 100 epochs je n'ai pas eu l'occasion de clairement établir l'influence des paramètres choisis. De plus, il serait intéressant d'augmenter la taille du dataset afin d'éviter l'overfitting mentionné précédemment.

2) Conclusion principales: Ce projet a été très intéressant à mener. J'ai passé beaucoup de temps à coder sur Python afin de manipuler et tester avec le code de Faster R-CNN, ainsi qu'avec la database. Cela m'a permis de comprendre l'influence notamment des différents paramètres d'un modèle. J'ai beaucoup appris dans la réalisation de ce projet qui ne me donne qu'envie d'aller plus loin.

REMERCIEMENTS

Je souhaite remercier Ismail Ben Ayed, chargé du cours SYS843 à l'ETS pour m'avoir enseigné et fait comprendre beaucoup de techniques de machine learning. Ces connaissances acquises m'ont permis de traiter au mieux le problème de mon projet et ont été de très bonnes pistes pour développer mon savoir.

RÉFÉRENCES

- [1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.
- [2] “R-cnn, fast r-cnn, faster r-cnn, yolo - object detection algorithms,” <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>, accessed : 2019-04-21.
- [3] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.
- [4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn : Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.
- [5] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, “3d object representations for fine-grained categorization,” in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.
- [6] “A comprehensive hands-on guide to transfer learning with real-world applications in deep learning,” <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>, accessed : 2019-04-21.
- [7] “Faster r-cnn (object detection) implemented by keras for custom data from google's open images dataset v4,” <https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a>, accessed : 2019-04-21.