STATISTIQUE
SCIENCE DES DONNÉES
UNIVERSITÉ DE MONTPELLIER

Compte rendu Article YOLO

Ait Moulay Abderrahim, Fanny Chery and Adrien Simon
Département de Mathématiques, Université de montpellier

I Introduction

Dans ce document nous allons résumer et expliquer les informations principales présentées dans l'article You Only Look Once: Unified, Real-Time Object Detection de Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi. https://arxiv.org/pdf/1506.02640.pdf

Cet article présente la première version de YOLO (you only look once), un algorithme de traitement d'images utilisant un réseau de neurones pour faire de la détection d'objet. Contrairement aux méthodes plus classiques et établies qui utilisent un classifieur pour chaque potentiel objet à détecter, YOLO aborde la détection d'objet avec une approche différente, en se ramenant à un problème de régression. L'architecture de son réseau de neurones est aussi conçue différentes des autres systèmes de détection. Elle permet, entre autres une grande rapidité d'exécution, rendant la détection d'objets en temps réel possible.

Nous commencerons par expliquer comment l'architecture du réseau de neurones de YOLO est faîte et comment cette dernière permet à l'algorithme d'être efficiente en temps de calcul. Ensuite nous expliquerons comment le réseau de neurones à été entrainé et quelles ont été les données choisies. Enfin la dernière partie traitera des limites de YOLO ainsi que des améliorations suggerées par l'article.

II Architecture du réseau de neurones

Le réseau de neurones utilisé par YOLO est composé d'un unique réseau de convolution dont l'architecture est inspirée de GoogLeNet (réseau qui met à profit différentes méthodes comme la mise en commun de moyennes globales ou encore la convolution de type 1 x 1)[1].

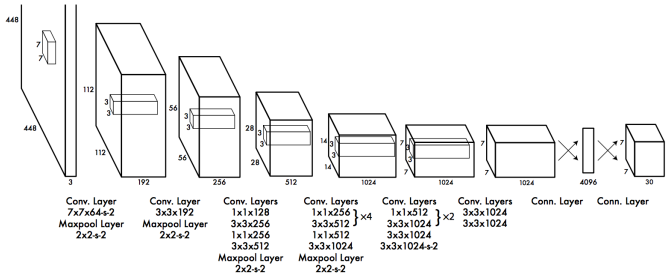

Figure (1): Architecture du réseau de neurones

Chaque couche du réseau de neurones est détaillée ci-dessus. Les 24 premières couches sont des couches de convolution et les deux dernières couches sont des couches "fully connected".

Le réseau comporte 24 couches de convolution suivies de 2 couches entièrement connectées. la figure (1) complète du réseau emprutée à l'article originel. Une version plus rapide de YOLO existe, appelée FastYOLO. Cette version n'utilise que 9 couches de convolution au lieu de 24. Cette moins grande profondeur s'accompagne d'une petite perte de performance sur la précision de la détection de chaque objet mais rend la détection d'objet possible jusqu'à 155 fps (frames per second).

YOLO fonctionne en analysant l'image dans sa globalité pour prédire des «boîtes de délimitation» associées à des classes d'objet. Chaque classe correspond à un types d'objet que YOLO est entraîné à reconnaître dans l'image traitée. Chaque image est tout d'abord redimensionnée et divisée en une grille de taille **SxS** et chaque cellule de la grille donne lieu à la prédiction de **B** boîtes de délimitation et scores de confiance. Chacune des boîtes de délimitation est déterminée par cinq coefficients (x, y, h, w, p) où (x, y) correspondent aux coordonnées du pixel au centre de la boîte, (h, w) respectivement sa hauteur et sa largeur (width). Enfin (p) reflète le niveau de confiance que l'algorithme attribue au fait que la boîte de délimitation concernée contienne effectivement un objet ou non. Le but étant que si une

boîte de délimitation ne contient pas d'objet, ou qu'elle ne prédit pas d'objet correctement en terme de localisation alors ce score vaille 0 ou s'en rapproche. Les prédictions faites par l'algorithme sont donc codées dans un tenseur de taille $S*S*(B*5+C)$.

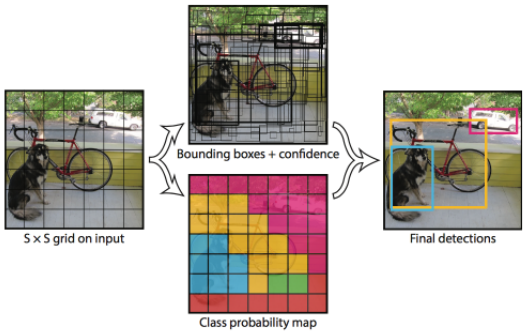

Figure (2): Fonctionnement du réseau de neurones

Chaque cellule de la grille donne lieu à la prédiction de B boîtes, mais comme on peut le voir ici, malgré le fait que le chien et le vélo soient superposés YOLO associe une seule classe à chaque cellule de la grille, il considère les cellules de la grille où les deux objets sont superposés comme des cellules associée à la classe "chien" (en bleu).

Ensuite, si $B > 1$ l'agorithme sélectionne au sein de chaque cellule de la grille, quelle boîte de délimitation il souhaite associer à ce cellule. Chaque cellule de la grille sera donc associé à un seul objet. C'est à dire que si deux objets se trouvent superposés dans une image, YOLO va faire un choix entre les deux pour n'en retenir qu'un seul et associer sa classe à la cellule entière. Ce procédé est spécifique à la première version de YOLO, cet aspect est amélioré dans les versions ultérieures.

L'algorithme prédit ensuite une boîte par groupe de cellules associées au même objet.
Une image empruntée à l'article détaille ce processus en figure (2). Sur cet exemple on voit qu'un vélo et un chien sont superposés dans l'image. YOLO tranche donc pour chaque cellule où les deux objets sont superposés, et prédit la cellule comme appartenant à la classe chien ou vélo.

III ENTRAÎNEMENT DU RÉSEAU

Comme expliqué précédemment, le réseau de neurones de YOLO comporte 24 couches de convolution. 20 d'entre elles ont subi un pré-entraînement, effectué sur un échantillon d'images appartenant au jeu de données de la compétition 1000-classes de ImageNet, pour la localisation d'objet. Cet entrainement a duré pendant environ une semaine et YOLO a obtenu un taux de précision de 88% sur l'échantillon de validation de ImageNet ILSVRC 2012.
Notons que l'indicateur utilisé pour déterminer la précision de chaque prédicition est appelé IOU (intersection over union) mesure le rapport entre l'intersection et l'union de la boîte prédite par YOLO et la vraie boîte annotée dans l'échantillon. C'est un indicateur de précision de localisation variant entre 1 et 0.

Quelques mots sur les données d'entraînement :
"ImageNet est une base de données d'images annotées produit par l'organisation du même nom, à destination des travaux de recherche en vision par ordinateur. [...] Le jeu de données ImageNet le plus utilisé, ILSVRC 2012-2017, est composé d'environ 1.5 millions d'images, réparties en environ 90% d'images d'entraînement, 3% de validation et 7% de test." [2]

Le modèle entier a ensuite été entrainé à la détection d'objet sur les données de PASCAL VOC 2012 et 2017, en ajoutant les quatre couches de convolution supplémentaires et les deux couches fully connected. Notons que les poids initiaux de chaque couche du réseau de neurones ont été choisis au hasard.

Toutes les couches du réseau à part la dernière utilisent des fonctions d'activation linéaire rectifiées de type leaky c'est à dire une type de fonction linéaire d'activation ReLU mais ayant une pente non nulle pour les valeurs négatives au lieu d'une pente nulle. [3]

Par simplicité l'optimisation est faite sur l'erreur quadratique du modèle. Le problème lié à ce choix est que malgré sa rapidité d'execution, il attribue la même importance au erreurs de localisation des objets prédits qu'à l'erreur de classification. Or les concepteurs de YOLO souhaitaient maximiser la MAP (mean average precision) c'est à dire la précision de la localisation de chaque prédiciton. Ce choix d'optimisation induit aussi une certaine instabilité du modèle à cause d'une différence de rapport trop grande entre le gradient des cellules contenant un objet et celles qui n'en contiènent pas. Pour ces raisons YOLO utilise une fonction de perte qui cherche à corriger ce problème en pénalisant d'avantage l'erreur de localisation.

loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Figure (3): Fonction de perte utilisée par YOLO

Les deux paramètres λ_{coord} et λ_{noobj} utilisés pour pénaliser la fonction sont définis comme valant 5 par défaut. L'indicatrice $\mathbb{1}_i^{obj}$ indique l'appartenance de l'objet apparait dans la boîte i. Quant à l'indicatrice $\mathbb{1}_{ij}^{obj}$ elle informe du prédicteur responsable de la classe associé à la cellule i.

IV Limites de YOLO et ouverture sur des améliorations possibles

Le fait de ne prédire qu'un objet par cellule de la grille de décomposition de chaque image, rend YOLO moins efficace pour prédire les petits objets proches les un des autres. Le modèle rencontre des difficultés a prédire des nuées d'oiseaux par exemple.
La base de données sur laquelle YOLO a été entraîné cause aussi une difficulté à prédire correctement des objets dans un cadre inhabituel. À la fin de l'article un exemlpe est donné, d'un parachutiste confondu avec un avion et non reconnu comme une personne par YOLO. La source principale d'erreur est la localisation des objets dans l'image.

L'article détaille une comparaison de YOLO avec d'autres méthodes de détection d'objet. Nous ne nous attarderons pas sur ces comparaisons ici. Nous relèverons quand même que l'article met en avant les qualités de la combinaison de YOLO avec Fast R_CNN (Une autre méthode de détection d'objet). La combinaison de ces deux méthodes permet entre autres d'obtenir un taux très faible de faux positifs (c'est à dire de prédire des objets dans l'arrière plan qui n'existent pas) tout en maintenant une précision (MAP) élevée sur les données de l'ensemble de test de VOC 2017.

V Conclusion

YOLO est une méthode de détection d'objet dont les principaux avantages sont la rapidité de l'évaluation de chaque image grâce à un réseau de neurones de convolution et l'adaptabilité à des nouvaux jeux de données et donc a différents types d'objets. Son approche par problème de régression permet de prendre en compte l'image dans son intégralité lors de l'évaluation de cette dernière ce qui implique un taux de faux positifs plus faibles que ses concurents.

VI Application pratique

Un répositoire git, sur lequel nous mettons en pratique certains éléments mentionnés dans ce résumé, est disponible au lien suivant : Dans cette application nous faisons varier les paramètres *iou_thresh* et *nms_thresh*. Les effets de ces paramètres sur le code ainsi que les instructions pour recréer l'expérience que nous avons faite sont disponibles dans le ReadMe du répositoire.

Github : `https://github.com/FannyChery/YOLO`

VII BIBLIOGRAPHIE

[1]: Article disponible au téléchargement via le lien suivant:
https://arxiv.org/abs/1409.4842v1
[2]: Texte extrait de l'article wikipédia disponible au lien suivant : https://fr.wikipedia.org/wiki/ImageNet
[3]: Informations issues de l'article suivant:
https://deeplylearning.fr/
cours-theoriques-deep-learning/
fonction-dactivation/