L'ÉCLAIRAGE

LES EVOLUTIONS DE YOLO

DE LA VERSION ORIGINALE (2015) A LA V4 (2020)

Charles FRANCHOMME
DATA SCIENTIST

aquila DATA ENABLER

aquila DATA ENABLER

LES EVOLUTIONS DE YOLO

DE LA VERSION ORIGINALE (2015) A LA V4 (2020)

Par **Charles Franchomme**

YOLO (*You Only Look Once*) est devenu une référence dans la détection d'objets sur image. Cet ensemble de modèles de *Deep Learning* doit sa renommée à sa grande vitesse d'éxécution à l'inférence qui permet (sous réserve de matériel informatique/hardware suffisant) de traiter des flux vidéo en temps réel tout en assurant des détections d'une grande justesse.

Sorti en 2015, le modèle initial a évolué pour donner naissance à des versions de plus en plus performantes. Elles sont à ce jour au nombre de 4 (un YOLOv5 existe mais aucun article officiel le présentant n'est publiquement accessible à la date de rédaction de cet article).

Ce rapide constat amène quelques questionnements :

- Quelle est l'approche commune adoptée par les différentes versions de YOLO ?
- Quelles sont les évolutions et modifications qui ont contribué à l'amélioration progressive du modèle à travers les différentes versions ?

Cet article se propose d'essayer de répondre à ces questions.

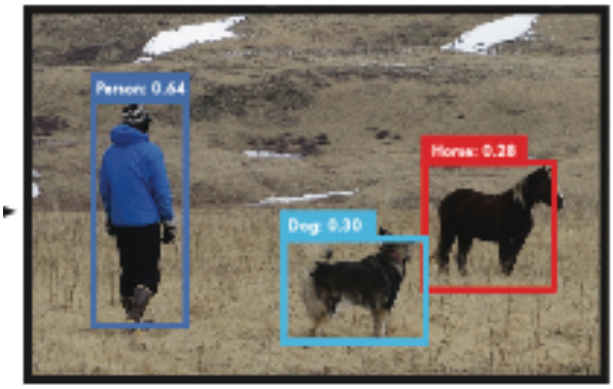
TABLE DES MATIÈRES

INTRODUCTION & DEFINITION

Avant de se concentrer sur YOLO, définissons rapidement ce qu'est la détection en Computer Vision :
Soit C classes d'objets (exemples de classes : chaise, chien, personne, casque de vélo…), pour une image donnée, si celle-ci contient des instances de ces classes, le but est de localiser et identifier la classe de ces instances.
Dans le cadre de YOLO, la localisation d'une instance de classe se fait par une *bounding box*, un rectangle dont les côtés sont horizontaux et verticaux et qui est censé encadrer au mieux l'objet détecté.
Le cadre est différent de la classification qui consiste à classer une image dans sa globalité (s'agit-il d'une image de chat ou bien d'une image représentant un vélo).
Pour une image, on attend en sortie une donnée : un nom/identifiant de classe.

Dans le cas de la détection, la sortie peut être :

- Vide (aucune instance de ces C classes détectée sur l'image)
- Un ou plusieurs objets pour lesquels est indiquée une *bounding box* et sa classe



Exemple de détection sur une image :
3 instances de classes distinctes détectées

www.aquiladata.fr - 47 Rue Louis Blanc, 92 400 Courbevoie – Tél. : 01 46 67 10 66 – Fax : 01 46 67 17 27

I. L'APPROCHE YOLO

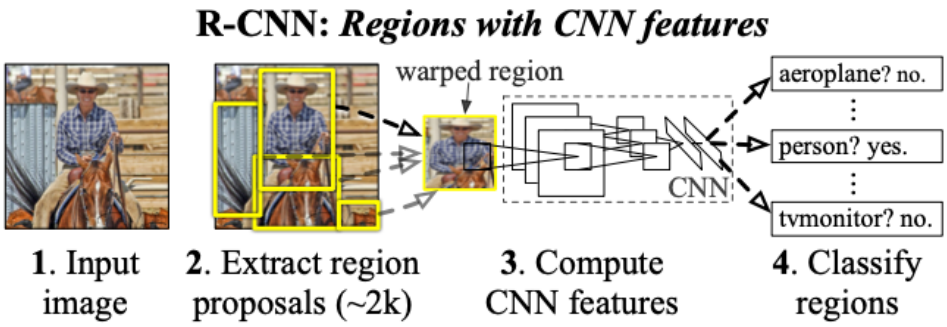
A) La détection en une étape

La réponse (du moins partielle) se trouve dans le nom de la méthode dont YOLO est l'acronyme : *You Only Look Once*, soit "vous ne regardez qu'une fois". A l'inférence, une image est traitée une unique fois par un seul réseau convolutif (CNN par la suite). Pour mieux comprendre cette spécificité, comparons cette approche avec un autre modèle de détection en vogue à la conception de la première version.

Un modèle comme R-CNN décompose le problème en 2 étapes [5] :

- générer (à l'aide d'une méthode de *region proposal*) des zones de l'image où se trouvent potentiellement les objets à détecter
- classifier ces zones générées à l'étape précédente et les affiner (pour produire des *bounding boxes* de localisation plus précises).

Les modèles en deux étapes requièrent plus de temps à l'inférence mais ont permis pendant un moment d'obtenir de meilleures prédictions que les modèles à une étape (dont YOLO fait partie).



Etapes de R-CNN

Au contraire, le modèle YOLO par l'intermédiaire d'un CNN prédit simultanément la localisation et les classes des objets.

B) Le format de l'output et le découpage en grid

Un autre point commun entre les différentes versions de YOLO est le format de l'*output* renvoyé par le réseau de neurones. C'est un 3D-tenseur de dimension : S x S x p.

Ceci correspond à un découpage de l'image initiale en une grille de S x S cellules. Pour chacune de ces portions d'image, les résultats de la détection sont stockées dans les p coordonnées disponibles par cellule.

De quelle manière ? Voyons pour cela comment les données annotées sont préparées pour l'entraînement.

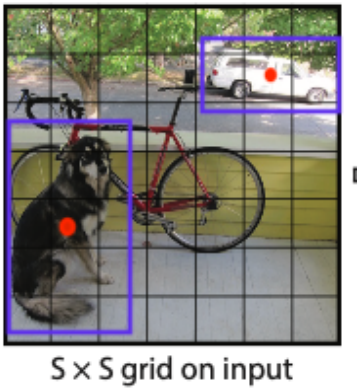


Image et découpage selon le format de l'output

Rappel : un set d'images annotées pour la détection doit contenir les informations suivantes pour chaque image: une liste (de la taille du nombre d'objets présents sur l'image) présentant pour chaque instance de classe, 1) les coordonnées (4 nombres) d'un *ground truth* (*bounding box* cadrée sur l'objet) 2) la classe à laquelle appartient l'objet.

A partir de ces annotations il faut construire les y_i (de dimension S x S x p) pour chaque image qui seront utilisés pendant l'entraînement. Pour chaque objet de l'image :

- Calculer les coordonnées du centre de la bbox
- Déterminer la cellule (x, y) de l'image où ce centre se situe ($0 \leq x, y < S$)
- Les coordonnées de la bounding box et la classe de l'objet sont ainsi renseignées dans la colonne yi[x, y, :]

Les autres parties du tenseur y_i, celles correspondant à des cellules et des *bounding box* sans objet, sont nulles.

YOLO traite ainsi la détection comme un problème de régression (prédiction des coordonnées des *bounding box*). On remarque également que p étant une dimension fixe, le nombre d'objets par cellule est limité à un nombre K. A fortiori le nombre d'objets détectables par le modèle sur une image est limité à S x S x K.

Les modèles YOLO prédisent pour chaque *bounding box* un score dit *objectness* qui modélise la probabilité qu'une instance de classe se trouve au sein de la *bounding box*, fois l'*Intersection over Union* (IoU) de cette *box* avec le *ground truth*. L'IoU permet de quantifier la justesse de la *bounding box* prédite par rapport au *ground truth* (donnée annotée).

$$objectness = Pr(object) * IOU(bb, gt)$$

C) L'inférence

Comment à partir d'un tenseur y de dimension S x S x p, les modèles YOLO construisent la liste des objets détectés ?

Pour chaque bounding box prédite, un score de confiance est obtenu en multipliant l'*objectness* au maximum des probabilités de classe :

$$Pr(object) * IOU(bb, gt) * max(Pr(Class_i \mid object)) = max(Pr(Class_i)) * IOU(bb, gt)$$

Ce score reflète la probabilité qu'il y ait une instance de la classe $argmax(Pr(Class_i))$ dans la *box* ainsi que l'adéquation de celle-ci à l'objet.

Si ce score est supérieur à un certain seuil fixé pour l'inférence (souvent par défaut à 0.25) alors l'objet est considéré comme détecté.

Pour des grands objets s'étalant sur plusieurs cellules (de la grille de taille S x S), il se peut qu'à l'inférence plusieurs de ces cellules prédisent une bounding box pour cet objet. Il faut donc une méthode de *postprocessing* pour élaguer et conserver les meilleures prédictions sur une image. La technique du *Non-max Suppression* (NMS) est appliquée pour chaque classe. La *box* avec le score de confiance le plus élevé est sélectionnée. L'IoU entre celle-ci et les autres est calculée. Toutes les box ayant un IoU supérieur à un seuil prédéfini avec cette box sont éliminées. On réitère le processus sur les box restantes jusqu'à ce qu'ils n'en restent plus. Finalement parmi des *clusters* de *bounding box* (au sens de l'IoU) celles avec un score de confiance maximal sont sélectionnées.

D) Architecture globale

Les modèles YOLO sont constitués, comme beaucoup d'autres modèles de détection, d'un *base features extractor* (ou *backbone* par la suite). L'apprentissage du modèle se fait en 2 temps. Le réseau *backbone* est entraîné à la classification sur ImageNet (1 000 classes) [3]. Il peut dès lors être utilisé pour extraire des *features* appropriées à la reconnaissance d'objets. Est ajouté à ce base extractor d'autres couches (nous préciserons cette architeture en fonction des versions plus tard). L'ensemble du réseau est alors entraîné à la détection, les poids du *base features extractor* étant fixes.

II. COMPARAISON DES PERFORMANCES DES DIFFÉRENTES VERSIONS

Pour pouvoir comparer différents modèles, il faut que ceux-ci soient à minima testés sur un même jeu de données et que la même métrique soit évaluée. Les deux tableaux comparatifs suivants ont été construits après lecture des articles concernant les différentes versions.

La métrique affichée ici, le mAP_{50}, est la plus évoquée parmi les différents articles et permet donc d'offrir la comparaison la plus large possible.

Modèle	Taille	FPS	mAP_{50}
YOLO	448	45	63.4%
YOLOv2	288	91	69.0%
YOLOv2	352	81	73.7%
YOLOv2	416	67	76.8%
YOLOv2	480	59	77.8%
YOLOv2	544	40	78.6%

Comparaison de la vitesse et des performances de détection de YOLO et YOLOv2 sur PASCAL VOC 2007 [4] (entraînement sur PASCAL VOC 2007 – 2012) sur un GTX Titan X

Modèle	Taille	FPS	mAP_{50}
YOLOv2	608	40	48.1%
YOLOv3	320	45	51.5%
YOLOv3	416	35	55.3%
YOLOv3	608	20	57.9%
YOLOv4	416	38	62.8%
YOLOv4	512	31	64.9%
YOLOv4	608	23	65.7%

Comparaison de la vitesse et des performances de détection de YOLOv2, YOLOv3 et YOLOv4 sur COCO [9] dataset test-dev (entraînement sur COCO trainval) sur un GTX Titan X

PASCAL VOC et COCO sont des jeux de données annotées pour la détection avec respectivement 20 et 80 classes d'objets. Ceci explique que YOLOv2 affiche de moins bonnes performances sur COCO (plus le nombre de classes est important plus la tâche est complexe).

La taille fait référence à la résolution des images en entrée du modèle. Si elle vaut 416, cela signifie que les images ont été redimensionnées à 416 x 416 pixels avant l'évaluation. On remarque qu'un accroissement de la résolution entraîne une perte de vitesse logique compensée par un gain dans la justesse des détections.

L'observation du mAP_{50} pour différentes versions à taille similaire montre qu'il y a une hausse de la précision (au sens général et non de la métrique) du modèle à chaque version. Concernant la vitesse du modèle, celle-ci a augmenté en passant de la v1 à la v2 puis a diminué (au profit de la justesse des détections) à la v3 avant de rehausser quelque peu à la v4.

III. LE RÔLE PRIMORDIAL DU *BASE FEATURES EXTRACTOR*

Toutes les versions de YOLO utilisent à la base de leur modèle un CNN entraîné à la classification sur ImageNet (1 000 classes). Pour la première version ce *features extractor* ressemble à GoogLeNet [16]. Darknet-19 (v2) est similaire aux VGG qui ont fait leur preuve dans le domaine de la classification. Darknet-53 (v3) d'après l'auteur est un ResNet (skip connections) qui s'inspire de Darknet-19 [7]. CSPDarknet-53 vient ajouter l'idée développée pour les Cross Stage Partial Networks (architecture qui permet de réduire les calculs et mieux propager le gradient à l'entraînement) [17]. On remarque une hausse des performances de classification à chaque version qui vont de pair avec celle concernant la détection vue en préambule. Ceci montre d'une part l'importance du choix du *features extractor* à la base de l'architecture et d'autre part la pertinence d'utiliser la classification pour entraîner un *features extractor* adéquat au problème de détection.

Modèle	**Backbone**	**Nombre de *convolutional layers***	**Billions of operations**	**Top – 5 accuracy (taille de l'image)**
YOLO [12]	Similar to GoogLeNet	20	11.1	88% (224 x 224)
YOLOv2 [13]	Darknet19	19	7.29	91.8% (256 x 256)
YOLOv3 [14]	Darknet53	53	18.7	93.8% (256 x 256)
YOLOv4 [2]	CSPDarknet53	?	52 (512 x 512)	94.8% (512 x 512)

Comparaison des performances de classification des base features extractor sur ImageNet

La hausse du nombre de couches entre la base de la version 2 et de la version 3 explique majoritairement la perte de vitesse de l'une par rapport à la précédente, le *feature extractors* représentant la majeure partie de l'architecture dans ces deux cas.

IV. DETECTION MULTIPLE PAR CELLULE A PARTIR DE LA V2

Dans la première version de YOLO, un seul objet par cellule peut être prédit. Dès la v2 le modèle est implémenté pour permettre de détecter plusieurs objets par cellule. Cette fonctionnalité va permettre entre autres d'améliorer la détection dans des régions où différents objets sont proches. Chaque bounding box en plus d'avoir un score d'*objectness* possède C scores modélisant la probabilité d'appartenir à une classe conditionnée au fait que la *bounding box* contienne un objet : $Pr(Class_i \mid object)$

Ceci se traduit entre autre par un changement de dimension de l'*output* :

$S \times S \times (B * 5 + C)$ à la première version

$S \times S \times (B * (5 + C))$ à la seconde version

B étant le nombre de bounding box prédites à chaque cellule.

V. INCORPORATION DES ANCHOR BOXES A LA V2 ET MODIFICATION DES VALEURS EN SORTIE

Les *anchor boxes*, préalablement utilisées par Faster R-CNN [15], à la conception de la v2 sont un ensemble de *bounding boxes* de référence à partir desquelles les *bounding boxes* vont être prédites à l'inférence. La largeur b_w et la hauteur b_h ne sont désormais plus directement prédites. Les variables qui les remplacent sont t_w et t_h qui sont reliées aux précédentes par les relations suivantes :

$$b_w = p_w \exp(t_w)$$

$$b_h = p_h \exp(t_h)$$

où p_w et p_h sont la largeur et la hauteur d'une *anchor box*. L'introduction de l'exponentielle permet de contraindre les largeurs et hauteurs de la bounding box à des valeurs positives.

L'idée est comme on le voit dans ces formules de prédire des écarts ou *offsets* par rapport à des *bounding boxes* modèles, les *anchor boxes*. Ces dernières doivent donc être représentatives dans leur ensemble des *bounding boxes* à prédire. C'est pourquoi les *anchor boxes* sont obtenues à partir d'un k-means appliqué sur l'ensemble des *ground truths* présentes dans le jeu de données d'entraînement. La métrique utilisée pour ce clustering est l'IoU.

VI. REFORMULATION DES SORTIES A PREDIRE (SUITE)

A) Version 2

Les coordonnées b_x et b_y du centre de la *bounding box* ainsi que son *objectness* sont également reformulées à la v2.

$$b_x = \sigma(tx) + c_x$$

$$b_y = \sigma(ty) + c_y$$

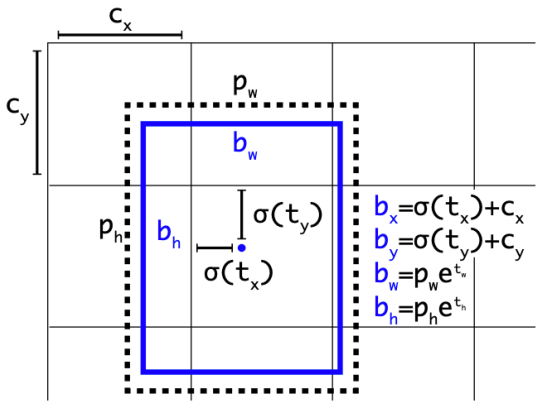
$$Pr(object) * IOU(b, object) = \sigma(t_o)$$

σ est la fonction d'activation sigmoïde : $\sigma(x) = 1/(1+\exp(-x))$ $\quad 0 < \sigma(x) < 1$

c_x et c_y ($0 \leq c_x, c_y \leq S-1$) sont les indices permettant d'identifier la cellule

A l'instar de l'exponentielle pour la largeur et la hauteur, l'utilisation de la fonction sigmoïde contraint les valeurs d'intérêt (les coordonnées de la *bounding box* et l'*objectness*) à rester dans un intervalle adéquat. Par exemple le score d'objectness doit être entre 0 et 1.

Les nouvelles variables directement prédites par le CNN, t_x, t_y, t_w, t_h et t_o peuvent dès lors prendre n'importe quelle valeur sans pour autant contredire la logique du modèle.



Anchor box (pointillé) et bounding box (trait plein) à partir de la v2

B) Version 3

A partir de la v3, la même transformation logistique est utilisée pour exprimer les probabilités d'appartenance aux classes :

$$Pr(Class_i \mid Object) = \sigma(t_i)$$

C) Version 4

Avec cette nouvelle formulation de b_x et b_y utilisant la fonction sigmoïde, il faut que t_x et t_y soient extrêmes afin que b_x et b_y atteignent les valeurs 0 ou 1, qui correspondent à une configuration où le centre de l'objet se situe sur le contour/bord de la cellule. Un coefficient $\alpha > 1$ est ainsi ajouté devant la sigmoïde permettant aux extrémités des cellules d'être plus facilement atteintes.

$$b_x = \alpha.\sigma(tx) + c_x$$

$$b_y = \alpha.\sigma(ty) + c_y$$

www.aquiladata.fr - 47 Rue Louis Blanc, 92 400 Courbevoie – Tél. : 01 46 67 10 66 – Fax : 01 46 67 17 27

VII. LA DETECTION A DIFFERENTES ECHELLES, UNE MODIFICATION MAJEURE DE LA V3

La faiblesse du modèle à bien détecter les objets de petite taille (petit relativement à la taille de l'image) est une des limites adressées aux deux premières version de YOLO. Pour renforcer le modèle sur ce point-ci, à partir de la v3 la prédiction se fait à 3 échelles différentes.

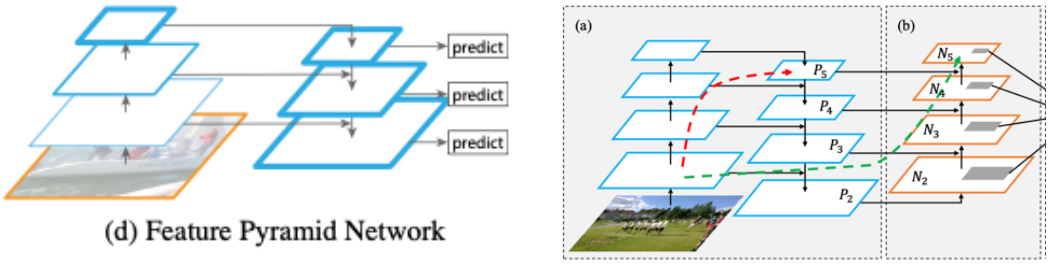
Les CNNs de la v1 et v2 renvoient un tenseur de dimension S x S x p. Pour la v3 et la v4 ce n'est pas un mais trois tenseurs de dimensions S x S x p, 2S x 2S x p et 4S x 4S x p qui se trouvent en sortie du CNN. Ces trois tenseurs correspondent chacun à un découpage spécifique de l'image d'entrée. La détection des objets les plus petits se fait donc au niveau du tenseur de dimension 4S x 4S x p qui correspond au découpage le plus fin.

A chaque échelle, 3 *box* sont prédites. La 3ème dimension des tenseurs p vaut donc : 3 * (5 + C) où C est le nombre de classes. Les 3 * 3 = 9 *anchor boxes* sont obtenues par k-means comme évoqué plus haut.

Comment se traduit cette modification au niveau de l'architecture du réseau de neurones ?

Dans la v3, un structure similaire aux *feature pyramid networks* est mise en place [8]. Une série de couches convolutives parmi lesquelles 2 couches de *upsample* (pour augmenter la résolution des *features maps*) insérées aux emplacements adéquats viennent s'ajouter au *backbone* (Darknet-53). Deux connexions avec des *features map* du *base features extractor* permettent d'enrichir les informations avant la prédiction aux deux échelles les plus fines.

La v4 ajoute un troisième bloc, de *downsampling* après la *backbone* (*downsampling*) et l'*upsampling* introduit dans la v3. Cette architecture tire son inspiration des *path aggregation networks* [10]. Sont opérées des concaténations entre des features issues de *layers* du *base features extractor* et des *features* plus en aval dans le réseau, l'idée étant de faire remonter de l'information obtenue en amont pour la prédiction.



Feature Pyramid Network (gauche) Path Aggregation Network (droite)

VIII. L'EVOLUTION DE LA FONCTION DE COUT OU *LOSS FUNCTION*

A) Version initiale

Initialement, la loss function est un MSE (mean squared error) sur les différents composants du tenseur final :

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Les 5 termes dont elle est la somme représentent respectivement :

- L'erreur de localisation des bounding box responsables de la détection d'un objet
- L'erreur sur les dimensions de la bounding box responsables de la détection d'un objet
- L'erreur sur l'objectness des bounding box responsables de la détection d'un objet
- L'erreur sur l'objectness des autres bounding box
- L'erreur de classification

λ_{coord} et λ_{noobj} permettent de contrôler le poids attribué respectivement à l'erreur concernant les coordonnées (localisation et dimensions) des *bounding box* contenant un objet et à l'erreur d'*objectness* des *bounding box* ne contenant pas d'objet. La valeur de 0.5 pour λ_{noobj} est mentionnée par l'auteur pour minimiser l'influence des cellules sans objet (souvent nombreuses et majoritaires) sur le gradient.

Remarques à propos de cette *loss function* :

Elle pondère également l'erreur de localisation indépendamment de la taille des objets alors qu'intuitivement un même écart de position devrait être moins préjudiciable pour une grande *bounding box*.

Par contre l'erreur sur la dimension des bounding box porte sur les écarts des racines carrées de la largeur et hauteur et attribue bien donc moins de poids à écart égal aux grandes *bounding box* (la dérivée de la fonction racine carrée étant décroissante).

B) Versions 2 et 3

L'erreur sur les *bounding box* est toujours évaluée via un MSE et porte maintenant sur les nouvelles sorties: t_x, t_y, t_w, t_h.

Les probabilités d'appartenance aux classes sont optimisées elles par l'intermédiaire d'une *binary cross-entropy*.

C) Version 4

Les auteurs de la v4 évoquent la limite que représente le MSE pour optimiser la prédiction des *bounding box*. Ils considèrent que cela revient à considérer les 4 coordonnées de la *bounding box* comme des données indépendantes alors qu'elles sont intrinsèquement liées les unes aux autres et définissent dans leur ensemble la *bounding box*. C'est pourquoi ils introduisent la CIoU-loss (Complete IoU loss) qui comme son nom le suggère va mettre à profit à l'IoU pour évaluer l'erreur sur une *bounding box* prédite [18].

$$LCIoU = 1 - IoU + \rho^2(b, b_{gt})/c^2 + \alpha v$$

avec

$$v = 4/\pi^2 * (\arctan(w_{gt}/h_{gt}) - \arctan(w/h))^2$$

$$\alpha = v/((1-IoU) + v)$$

Cette loss est l'addition de trois termes :

- αv

Ce terme mesure l'écart entre la forme de la bounding box et la ground truth. Autrement dit ce terme positif est minimal (vaut 0) lorsque la proportion largeur-hauteur de la bounding box est la même que pour le *ground truth*.

- ρ2(b, bgt)/c2

Ce terme mesure la distance entre les centres de la bouding box et du ground truth ($\rho^2(b, b_{gt})$) relativement à la diagonale (c) du rectangle minimal les contenant. Ici on vient donc évaluer l'erreur de localisation tout en prenant en compte la taille des objets.

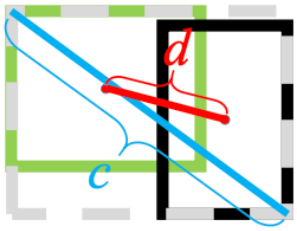


Figure illustrant la distance d = ρ2(b, bgt) entre les centres
et c la longueur de la diagonale de la plus petite enclosing box

- 1 – IoU

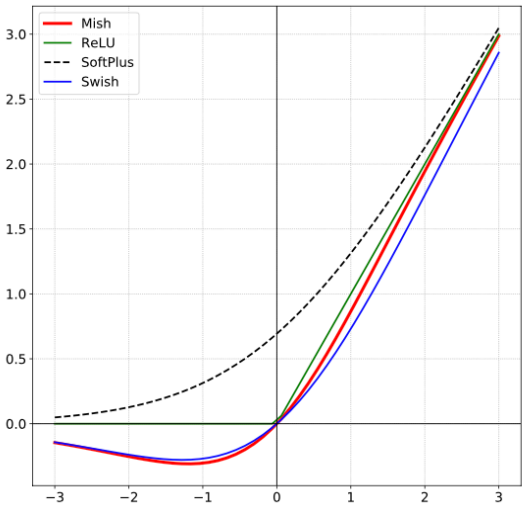
Ce dernier terme permet de s'assurer que l'IoU est maximisé pendant l'entraînement. Si les 2 termes précédents sont nuls, alors la minimisation de ce terme permettra à la *bounding box* d'avoir la même taille que le *ground truth*.

IX. AUTRES NOUVEAUTES DE LA VERSION 4

L'article présentant la v4 détaille la liste des ajouts par rapport aux versions précédentes. Parmi celles-ci :

- L'utilisation de la fonction d'activation Mish dans le réseau de neurones [11] :

$$Mish(x) = x.\tanh(\ln(1 + \exp(x)))$$



Graphe de Mish (en rouge)

- L'utilisation des méthodes de *data augmentation* CutMix, Mosaic et Self-Adversarial Training (SAT) :

CutMix (à gauche) Mosaic (à droite)

La première consiste à introduire le *crop* d'une image au sein d'une deuxième image. Cette méthode est utilisée à l'entraînement du *classifier* seulement. La deuxième permet d'agencer 4 images pour en former une nouvelle. Ces deux techniques sont intéressantes car elles mélangent dans une même image des contextes différents et entraîne le modèle à détecter des objets en dehors de leur contexte habituel. Le SAT consiste à utiliser le réseau comme générateur afin d'ajouter du bruit à une image et utiliser la nouvelle obtenue dans l'apprentissage.

CONCLUSION

Les innovations apportées successivement aux différentes versions de YOLO ont contribué à fournir un modèle de plus en plus performant. Elles sont de différents types et ne concernent pas un aspect particulier du modèle : architecture, *data augmentation*, *loss function*… La recherche est intense sur les réseaux de neurones en général et de nouvelles innovations concernant leur architecture ou leur entraînement voient fréquemment le jour. Les auteurs de YOLO ont su intégrer certaines avec succès.

Il serait intéressant de quantifier le gain (en points de mAP_{50}) apporté par chacune de ces modifications. L'article présentant la v2 détaille un à un la contribution de chacune des innovations apportées à la version originale. L'article de la v4 énumère dans un premier temps l'ensemble des fonctionnalités susceptibles d'êtres mises en place avant de les comparer et de choisir les plus intéressantes dans l'optique de la détection.

La plupart des modifications et ajustements exposées ici ne sont pas restreintes à des versions particulières. Autrement dit il est envisageable d'appliquer certaines fonctionnalités sur des versions antérieures. On pourrait par exemple utiliser la fonction d'activation Mish ou la CIoU *loss* sur les versions 1, 2 ou 3.

Quelle suite pour YOLO ?

Le modèle, tout en gardant sa philosophie peut toujours gagner en justesse de détection et en vitesse. Améliorer ce deuxième point pourrait s'avérer primordial s'il existe une volonté de déployer ces modèles sur des infrastructures plus petites (la détection en temps réel se fait avec GPU). A chaque version un petit modèle (tiny YOLO) est conçu. Ils peuvent être utilisés sur du CPU mais la détection est bien moins performante (40% de mAP_{50} pour Tiny-YOLOv4 contre plus de 60% pour YOLOv4).

D'autre part, les *bounding box* actuelles présentent leurs limites. Pour certaines dispositions d'objets un rectangle incliné serait plus adéquat. Mais cela signifie qu'il faut rajouter dans l'*output* une coordonnée par *box* à prédire et l'annotation des données sera plus complexe également. Les masques sont plus précis.

Réaliser de la *class* ou *instance segmentation* tout en essayant de préserver au maximum l'approche *one-stage* suivie par YOLO est un sujet déjà traité et sera probablement l'objet de nombreux développements dans les prochaines années [1].

LEXIQUE

1) **IoU :** Intersection over Union

L'IoU est une mesure beaucoup utilisée en *computer vision* permettant de quantifier la similarité entre deux *box*. C'est le rapport entre l'aire commune de ces 2 *box* et l'aire totale couverte par l'ensemble des 2.

$$IoU = \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|}$$

2) **mAP50** : mean Average Precision at 0.5 IoU threshold

50 signifie qu'une prédiction (*bounding box* + classe) est considérée comme juste lorsque le IoU entre la *box* prédite et le *ground truth* est supérieur à 50% (et que la classe de l'objet est correctement prédite).

Pour chaque classe du jeu de données (sur lequel est évalué le modèle), on va calculer l'*Average Precision*, nombre situé entre 0 et 1, puis déduire la moyenne de ces valeurs d'où le *mean Average Precision*. Il ne reste plus qu'à définir l'*Average Precision* pour comprendre cette métrique.

Le principe est très similaire à celui employé pour obtenir la courbe ROC. Pour une classe donnée, on fait varier le seuil de classification et on calcule pour ce seuil la précision et la sensibilité du modèle. Dans un deuxième temps on trace la précision en fonction de la sensibilité puis on calcule l'aire sous cette courbe (qui est nécessairement comprise entre 0 et 1).

REFERENCES

[1] N. Araslanov, and S. Roth. Single-Stage Semantic Segmentation from Image Labels. In Proceedings of the IEEE *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4253-4262, 2020.

[2] A. Bochkovskiy, C. Y. Wang and H. Y. M. Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. 2020.

[3] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[4] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[6] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[8] T. Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017.

[9] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[10] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8759–8768, 2018.

[11] D. Misra. Mish: A self regularized non-monotonic neural activation function. arXiv preprintarXiv:1908.08681, 2019.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[13] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *Computer Vision and Pattern Recognition (CVPR)*, 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017.

[14] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

[15] 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 (NIPS)*, pages 91–99, 2015.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[17] C. Y. Wang, H. Y. M. Liao, Y. H. Wu, P. Y. Chen, J. W. Hsieh, and I. H. Yeh. CSPNet: A new backbone that can enhance learning capability of cnn. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop)*, 2020.

[18] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren. Distance-IoU Loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

DATA SCIENCE • R&D

aquila

DATA ENABLER

aquiladata.fr

47 rue Louis Blanc · 92400 Courbevoie