

Projet DLIM: Prédiction de boîtes englobantes

Nicolas Portal, Geoffrey Jount, Camille Huet, René Louis Lemaire

Présentation du problème :

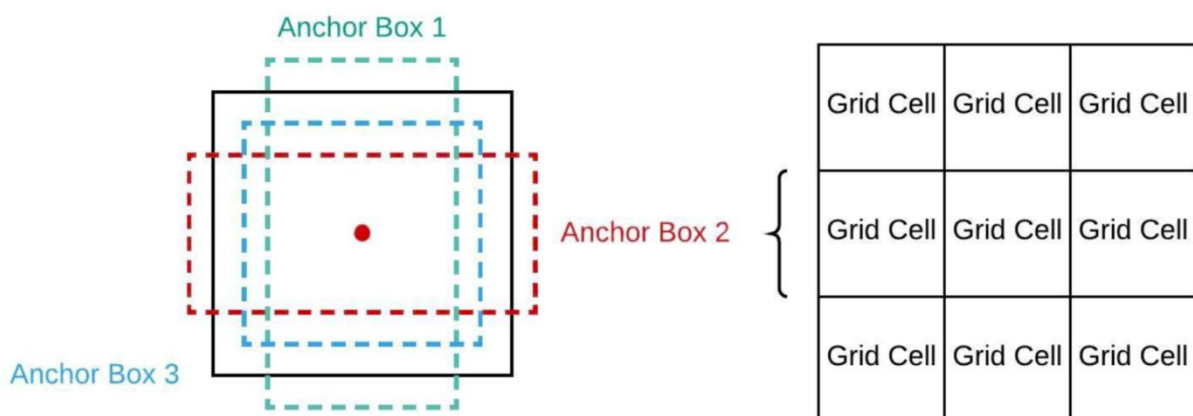
Ce projet consiste en une prolongation du tp 3. On essaye ici de pouvoir prédire automatiquement les coordonnées des boîtes englobantes en s'appuyant sur le principe 'd'anchor boxes' ou boîtes d'ancrages. Contrairement au tp 3, notre réseau ne doit plus simplement ségmenter les visages dans l'image, mais détecter automatiquement la position des visages. Cela signifie également que l'on ne s'appuie plus sur le principe de 'connected components' pour tracer nos boîtes englobantes.

Présentation de la solution choisie :

Notre projet s'inspire fortement du réseau YOLOv3. Nous avons décidé de partir sur ce réseau car étant en plusieurs versions, cela nous permettait de procéder à des améliorations incrémentales. Avant d'implémenter une nouvelle fonctionnalité de YOLOv3 ou v2, on s'assurait que les spécificités du réseau YOLOv1 étaient correctement implémentées. Par ailleurs c'est aussi la clarté des articles de YOLO qui nous ont incités à partir sur ce réseau. En outre, de prime abord, les solutions concurrentes telles que retina net nous ont semblé assez complexes en comparaison avec YOLOv1. Cela était dû au fait que YOLOv1 n'utilise pas d'analyse multi-échelle. Par la suite On a réussi à implémenter cette fonctionnalité qui est aussi présente dans YOLOv3.

Par conséquent, ce ne sont pas des considérations d'efficacité ou de précision du modèle qui nous ont incitées à s'inspirer de YOLO plutôt que de retinaNet ou un autre modèle de type R-CNN, mais plutôt des considérations liés à l'aisance apparente d'implémentation.

Création des tenseurs de vérité terrain



Notre modèle s'appuie donc sur le principe des 'anchor boxes'. Ce sont des boîtes englobantes de tailles différentes qui vont correspondre à des visages de tailles variés. Pour construire nos labels Y_{true} , on va découper les images en cases de tailles égales. Si l'on considère pour commencer que l'on utilise qu'un seul niveau d'échelle, on peut décider de découper notre image en 16×16 case (par exemple). Pour chaque image on va calculer l'iou entre les visages présents et l'ensemble de nos boîtes d'ancrages. Par exemple, si on a trois visages dans l'image et qu'on a quinze boîtes d'ancrages

on aura une matrice de valeurs de taille 15*3. Ensuite pour chaque visage, on détermine la boîte d’ancrage ayant l’iou maximum. Si cette boîte d’ancrage correspond à une boîte d’ancrage de l’échelle courante alors cette boîte est valide, sinon elle est invalide. Si la boîte d’ancrage est valide pour l’échelle considérée, on va ajouter les coordonnées de la boîte englobante du visage (la vérité terrain) dans le tenseur de sortie à l’indice de la case où se trouve le centroid du visage (c'est-à-dire le centroid de la boîte englobante de la vérité terrain) et à l’indice de la boîte d’ancrage valide. On répète cette étape pour chaque image et pour chaque échelle pour obtenir nos tenseurs des labels. Au final pour une échelle seulement, le tenseur de sortie aura la forme suivante :

[Nombre d’images, nombre de cases selon x, nombre de cases selon y, nombre de boîtes d’ancrages, 5]

Par exemple pour une échelle en seize par seize avec trois boîtes d’ancrages :

[4000, 16, 16, 3, 5]


Selon la dernière dimension du tenseur on a en effet cinq éléments : un booléen qui prend la valeur 1 si un visage est présent dans cette case et les quatre coordonnées de la boîte englobante du visage. Les coordonnées de la boîte englobante se décomposent comme suit :

[x, y, w, h]

Le centre de la boîte selon x, le centre de la boîte selon y, la largeur de la boîte, la hauteur de la boîte.

Il faut donc effectuer une conversion car dans le jeu de données x et y représentent le coin supérieur gauche de la boîte.

Il faut également noter que ces cinq coordonnées doivent avoir pour référence la taille de l’image c'est-à-dire que ces valeurs sont comprises entre 0 et 1 (on divise par la taille de l’image). Pour calculer l’iou on calcule le rapport de l’intersection entre les deux boîtes sur leur union (cf code). L’iou représente donc une mesure du chevauchement entre deux boîtes.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


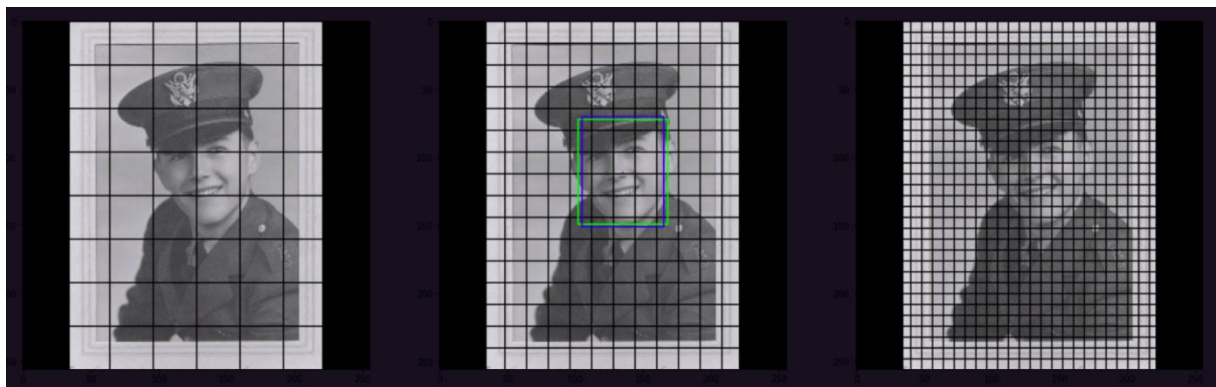
Cependant, et conformément à YOLOv3, nous avons décidé d’utiliser plusieurs échelles pour analyser les images. Cette analyse multi-échelles doit permettre de mieux repérer les plus petits visages notamment (ou les visages énormes). Concrètement, on va avoir non plus un tenseur de vérité terrain mais plusieurs car nous avons plusieurs échelles. Chaque tenseur correspond à une taille de case différente. Cela va également nécessiter d’adapter le nombre de boîtes d’ancrages. Il doit y avoir autant de boîtes d’ancrages par échelle. On répartie les boîtes d’ancrage par échelle en

fonction de leur aire. Si l'on a cinq échelles et trois boîtes d'ancrage par échelle (donc quinze boîtes d'ancrages en tout), on va allouer les trois boîtes d'ancrages ayant l'aire la plus petite à l'échelle de plus petite taille, c'est-à-dire l'échelle avec le plus grand nombre de cases qui va servir à détecter les visages les plus petits. A l'inverse les trois boîtes d'ancrages ayant l'aire la plus élevée seront attribuées à l'échelle la plus grande c'est-à-dire l'échelle ayant le plus faible nombre de cases servant à détecter les visages les plus grands. On fait de même pour les autres boîtes d'ancrages et les autres échelles. Cela va donc nécessiter de classer les boîtes d'ancrages par aire et les échelles par taille et à faire correspondre boîtes d'ancrages et échelles. En outre, comme évoqué plus haut, cela va nécessiter d'adapter la création des labels de sorte que, pour une image et une échelle particulière, le tenseur de sortie est mis à jour seulement si la boîte d'ancrage ayant l'IOU maximale avec la vérité terrain est l'une des trois réservées pour cette échelle. En effet, on souhaite éviter, pour une même image et pour un même visage, d'avoir plusieurs boîtes d'ancrages dans le tenseur créé. Pour des échelles de taille 32*32, 16*16 et 8*8, on obtient trois tenseurs de la forme :

[Nombre d'images, 8, 8, 3, 5]

[Nombre d'images, 16, 16, 3, 5]

[Nombre d'images, 32, 32, 3, 5]



Dans l'image ci-dessus on peut constater que la taille du visage correspond à l'échelle moyenne de 16*16. On a affiché en vert la boîte englobante de la vérité terrain et en bleu la boîte d'ancrage ayant l'IOU le plus élevé avec cette vérité terrain. On remarque également que, pour un même visage, on a une réponse dans seulement une des trois échelles.

On vient de le voir, pour créer nos trois tenseurs de vérité terrain on a besoin au préalable de définir trois boîtes d'ancrages par échelle. Conformément à YOLOv3, Pour trouver les neuf boîtes d'ancrages nous avons utilisé un algorithme du kmeans avec un nombre de features égal à deux et qui correspondent à la largeur et la hauteur des boîtes englobantes de chaque visage de chaque image dans le dataset. Pour l'exemple précédent, on va donc utiliser une valeur de k égale à neuf. Ensuite on trie les boîtes en fonction de leur aire de manière à pouvoir les attribuer aux trois échelles.

Fonction de cout

La fonction de cout qui est utilisé dans YOLOv3 se décompose en trois parties : Une partie de classification pour savoir si un objet est présent, une partie régression pour ajuster les coordonnées des boîtes d'ancrages et une dernière partie de classification pour déterminer quel type d'objet est

présent. Dans notre cas, et puisqu'on traite uniquement des visages la troisième et dernière partie de la fonction de cout était inutile. Par conséquent notre fonction de cout se décompose en deux parties :

$Fc = \text{Classification (un visage est il présent ?)} + \text{Régression (coordonnées de la boite englobante)}$

Les deux composantes de cette fonction de cout se décomposent elles-mêmes en deux parties :

$Fc\text{-classification} = Fc_objet_présent + Fc_objet_non_présent$

$Fc\text{-régression} = Fc_centroid_x_y + Fc_largeur_hauteur$

En fin de compte, on obtient la fonction de cout suivante :

$\lambda_o Fc_objet_présent * obj_mask + \lambda_{no} Fc_objet_non_présent * (1 - obj_mask) * ignore_mask + \lambda_{xy} Fc_centroid_x_y + \lambda_{wh} Fc_largeur_hauteur$

Les poids par lesquels on pondère les composantes de la fonction de cout permettent d'ajuster l'importance que l'on donne aux différentes composantes. Ils ont une importance cruciale pour améliorer la précision des détections. Dans la plupart des cas on va donner un poids plus faible pour $Fc_objet_non_présent$ pour s'assurer que le réseau ne sera pas dominé par des cases où il n'y a pas d'objets.

Pour les fonctions $Fc_objet_présent$ et $Fc_objet_non_présent$, on utilise la crossentropie binaire étant donné que ces fonctions ne peuvent prendre que deux valeurs : 0 si il n'y a pas de visage et 1 si un visage est présent. Pour la partie localisation des coordonnées de la boite englobante on utilise la régression via le mean square error(mse).

La partie $Fc_objet_non_présent$ est nécessaire pour pénaliser les faux positifs, autrement le modèle proposerait trop de boites englobantes. On obtient les faux positifs grâce au masque $(1 - obj_mask)$.

Une autre spécificité de la fonction de cout découle du fait que le réseau ne prédit pas directement les coordonnées de la boite englobante dans l'image mais plutôt un décalage par rapport à la case dans laquelle se trouve la boite d'ancrage. Il faut aussi tenir compte du fait que les données présentes dans les labels ont pour référence l'image. Par conséquent les données en sortie du réseau et les données présentes dans la vérité terrain n'ont pas la même référence (par rapport à la case et par rapport à l'image donc). Ainsi, il va être nécessaire d'effectuer une transformation au sein de la fonction de cout pour obtenir les coordonnées de la boite englobante dans l'image. On préfère ici prédire les coordonnées de la boite englobante par rapport à la case plutôt que l'image car cela permet d'éviter l'instabilité du réseau notamment dans les premières phases de l'apprentissage. On a utilisé les transformations suivantes (issues de l'article Yolov2/Yolov3):

$$\begin{aligned}
 b_x &= \sigma(t_x) + c_x \\
 b_y &= \sigma(t_y) + c_y \\
 b_w &= p_w e^{t_w} \\
 b_h &= p_h e^{t_h}
 \end{aligned}$$

b_x , b_y , b_w , et b_h correspondent aux coordonnées de la boîte englobante par rapport à l'image. Ce sont les coordonnées présentes dans le tenseur de vérité terrain. t_x , t_y , t_w et t_h correspondent aux coordonnées de la boîte englobante par rapport à la case où elle est présente. Ce sont les coordonnées obtenues en sortie du réseau. L'exponentielle appliquée aux coordonnées de largeur et hauteur obtenues en sortie du réseau va permettre de ne pas avoir de prédictions négatives.

On applique une sigmoïde à t_x , t_y et t_o pour garder ces données entre 0 et 1. c_x et c_y représentent les coordonnées de la case de la grille dans l'image (par exemple 1 et 1 pour la deuxième ligne et deuxième colonne). p_w et p_h représentent respectivement la largeur et la hauteur des boîtes d'ancrages.

Pour calculer la fonction de coût il va falloir ramener toutes les coordonnées par rapport à la case. Cela implique d'effectuer une transformation sur les données comprises dans le tenseur de la vérité terrain. Par exemple pour obtenir t_w à partir de b_w on effectuera la transformation inverse :

$$t_w = \log(b_w / p_w)$$

On effectue les transformations inverses pour les quatre équations ci-dessus.

Pour obtenir 'ignore_mask' évoqué plus haut dans la fonction de coût il est nécessaire de calculer l'IOU entre les boîtes englobantes prédites par le réseau et la vérité terrain. En effet, on souhaite pénaliser les boîtes englobantes prédites par le réseau dont l'IOU avec un visage est inférieur à un seuil. Cela s'explique par le fait que l'on ne souhaite pas pénaliser les prédictions relativement proches de la vérité terrain. Pour calculer cet IOU, on aura cette fois-ci besoin que toutes les coordonnées aient pour référence l'image. Comme les coordonnées présentes dans le tenseur de vérité terrain ont déjà pour référence l'image, c'est cette fois-ci les coordonnées obtenues en sortie du réseau qu'il va falloir transformer en suivant les quatre premières équations ci-dessus.

On a donc au final deux étapes dans la fonction de coût qui vont nécessiter des transformations différentes : le calcul de l'IOU et le calcul de la fonction de coût en elle-même.

Le réseau

Notre réseau est découpé en deux parties : une partie extracteur qui va servir à obtenir des 'feature embedding', c'est-à-dire une représentation sous forme de tenseur des caractéristiques des images pour chacune des échelles (ici les boîtes englobantes, cf partie 'création des tenseurs de vérité terrain') ; et une partie détecteur à laquelle on va donner nos tenseurs obtenus en sortie de

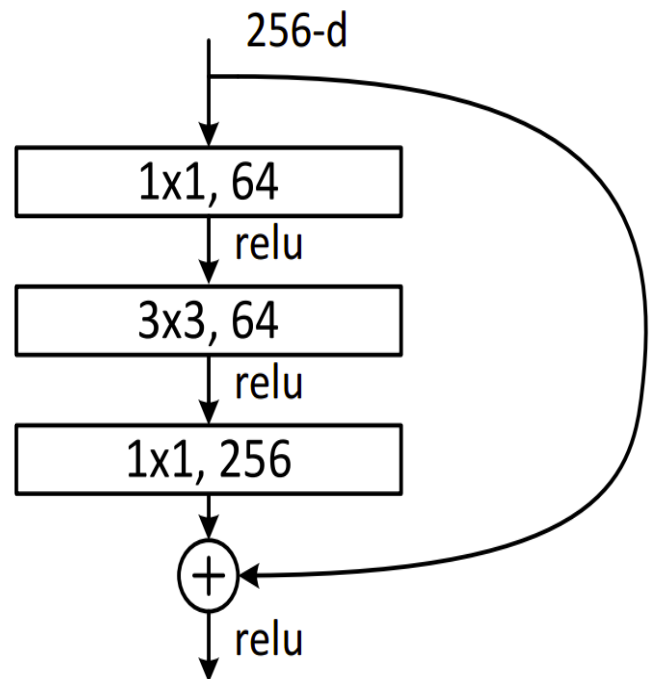
l'extracteur et qui va nous donner en sortie de nouveaux tenseurs dont la forme correspond à celle spécifiée dans la partie 'création des tenseurs de vérité terrain'.



Notre extracteur de caractéristiques s'inspire du réseau 'Darknet 53' utilisé par YOLOv3. Ce réseau s'appuie sur les principes établis par Resnet de manière à conserver un nombre important de couches sans provoquer de sur-apprentissage. Dans notre cas on a utilisé un peu moins de 53 couches car nos images ont une taille de 256*256 et non 416*416 comme dans YOLOv3.

	Type	Filters	Size	Output
1x	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	128 × 128
	Convolutional	64	3 × 3	
	Residual			
2x	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	64 × 64
	Convolutional	128	3 × 3	
8x	Residual			
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	32 × 32
	Convolutional	256	3 × 3	
	Residual			
8x	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	16 × 16
	Convolutional	512	3 × 3	
	Residual			
	Convolutional	1024	3 × 3 / 2	8 × 8
4x	Convolutional	512	1 × 1	8 × 8
	Convolutional	1024	3 × 3	
	Residual			

Réseau resnet 53 utilisé dans Yolov3

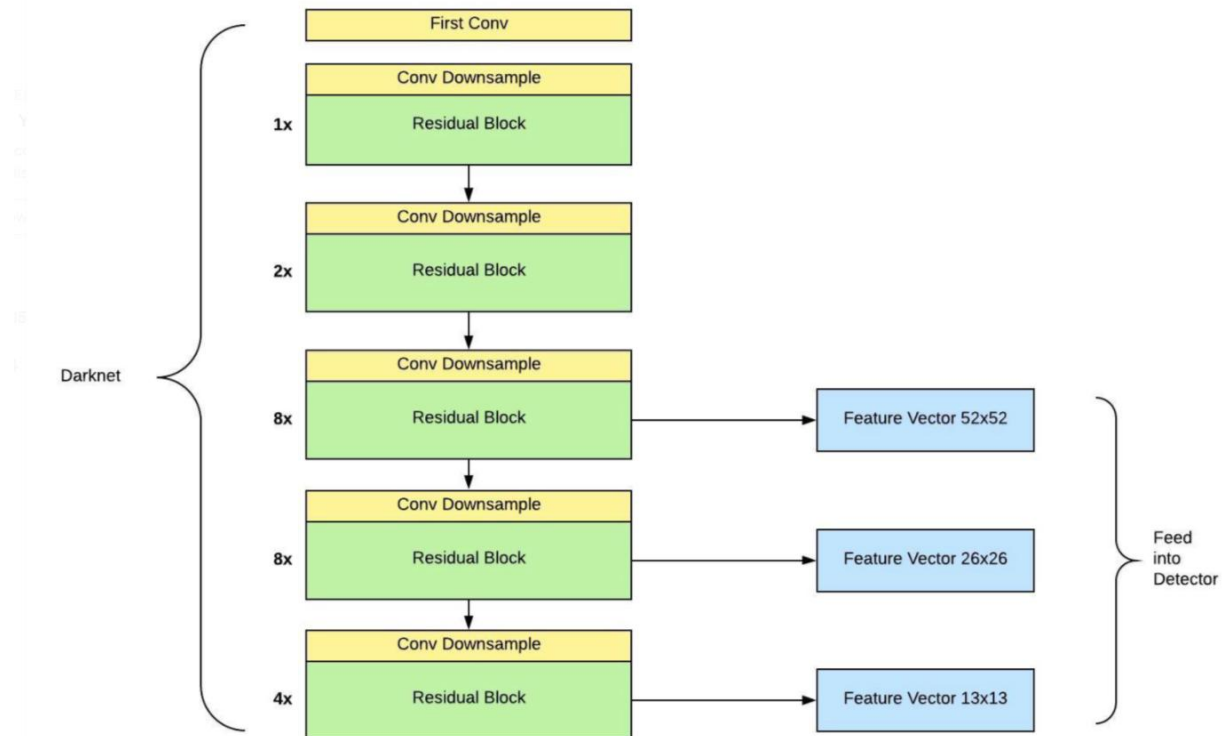


Bloc 'bottleneck' du réseau Resnet

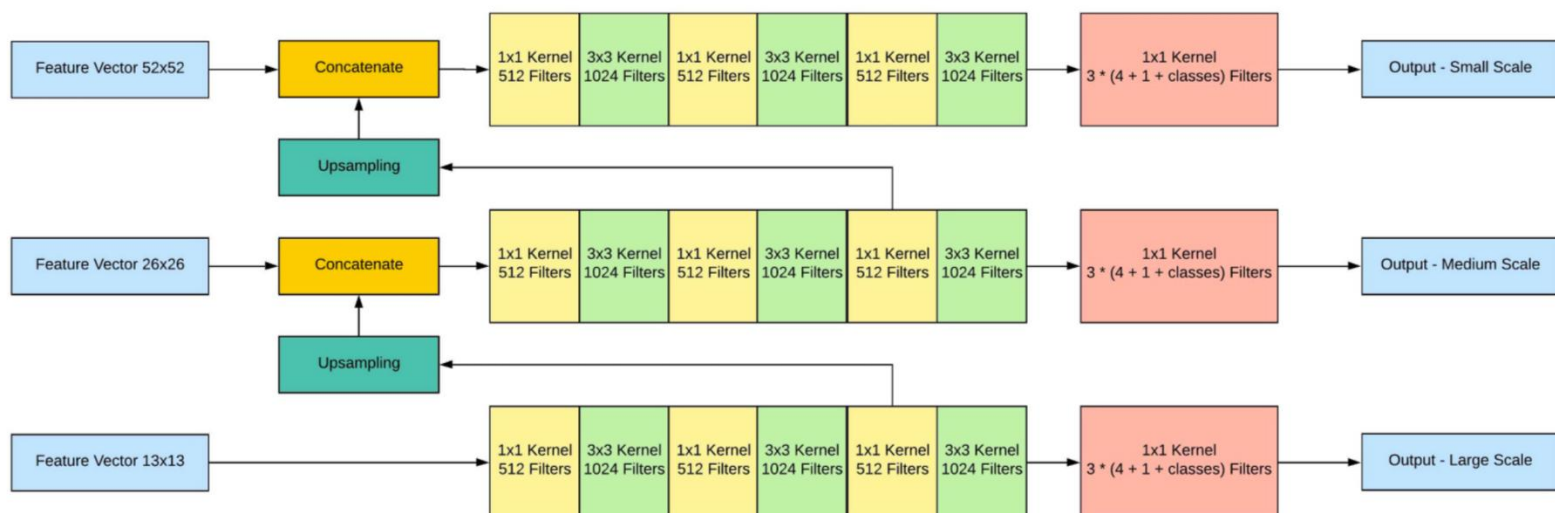
Les parties encadrées dans le schéma ci-dessus correspondent à des blocs de type resnet. La couche 'residual' consiste à ajouter l'input du bloc avec l'output de la dernière couche de convolution du bloc. De cette manière l'input du bloc effectue un 'saut' au dessus des deux couches de convolution du bloc. La colonne 'size' correspond à la taille du noyau pour la convolution. Au sein d'un bloc de type resnet on a donc une couche avec un noyau de taille un et une couche avec un noyau de taille trois. Cela correspond à ce qui est appelé une structure 'bottleneck' dans l'article resnet. Entre ces blocs on a des couches avec des strides de deux pour diviser la taille de l'image par deux. On n'utilise donc pas de maxpooling. Chaque étape de convolution est constituée en plus d'une couche de batch normalisation de manière à rendre l'apprentissage plus stable et améliorer la généralisation sans

avoir recours au dropout. La fonction d'activation utilisée est leaky relu avec un paramètre alpha de 0.1 comme conseillé dans l'article.

Pour obtenir les différents tenseurs de sortie correspondant aux différentes échelles, il va être nécessaire de sauvegarder l'output de certaines couches. Cette sauvegarde aura lieu plus ou moins tôt dans le réseau en fonction de l'échelle considérée. La sortie correspondant à l'échelle la plus petite sera sauvegardée le plus tôt dans le réseau tandis que la sortie correspondant à l'échelle la plus grande sera sauvegardée en dernier. Cela est lié au fait qu'en descendant dans le réseau, on réduit progressivement la taille de l'image par deux.



L'image ci-dessus représente un exemple d'extracteur de caractéristiques avec trois échelles en 52*52, 26*26 et 13*13. Une fois que l'on a obtenu les trois tenseurs (si on utilise trois échelles) en sortie de l'extracteur de caractéristiques, on doit donner ces tenseurs au détecteur. Notre détecteur se présente comme ceci :



La particularité de ce détecteur est que pour les échelles medium et small, on va concaténer le résultat de l'échelle supérieur avec le feature vector obtenu en sortie de l'extracteur pour cette échelle. De cette manière, la détection à une échelle donnée peut bénéficier du résultat de la détection obtenu à une échelle plus large. Il sera donc nécessaire de faire de l'upsampling car les couches sont de tailles différentes. En particulier, pour une échelle donnée (sauf la plus grande), le vecteur de caractéristiques est de taille deux fois supérieur au résultat de la couche précédente. Il est donc nécessaire de multiplier la taille du résultat de la couche précédente par deux.

Au final après l'extracteur et le détecteur on obtient autant de tenseurs qu'il y a d'échelles. Ces tenseurs seront utilisés dans la fonction de cout avec la vérité terrain. Pour être plus précis, on aura autant de fonctions de couts que d'échelles. Chaque fonction calculera le cout associé à un tenseur/échelle, puis ces couts seront additionnés avant d'effectuer la backpropagation.

Non max suppression

Une fois que l'on a effectué nos prédictions des boites englobantes sur des images, on se retrouve avec énormément de boites prédites. Il va donc être nécessaire d'effectuer un tri entre ces boites pour affiner les prédictions. Pour cela, on a décidé d'implémenter une étape de non max suppression. On va d'abord supprimer toutes les boites dont le score de confiance est inférieur à un seuil, par exemple 0.5. Ensuite, de manière itérative, on va sélectionner la boite avec la probabilité de détection la plus élevée (score de confiance correspondant au premier élément de notre tenseur selon la dernière dimension). Cette boite sélectionnée va être stockée en mémoire. Toutes les boites dont l'IOU avec cette boite sélectionnée est supérieur à un seuil (par exemple 0.2) vont être éliminées. Cela va permettre d'éviter d'avoir plusieurs boites pour un même visage. On répète cette opération jusqu'à ce qu'il n'y ait plus de boite pour l'image en veillant à chaque fois à stocker dans une liste les boites ayant la probabilité de détection la plus élevée. Voici un exemple imagé issu du notebook où on affiche pour une case sur quatre (de manière à ce que l'image ne soit pas toute jaune) :



A gauche l'image avant l'étape de non max suppression. A droite, l'image après l'étape de non max suppression

Présentation du protocole expérimental

Il a été décidé de garder uniquement les images dont tous les champs autres que les champs correspondant aux coordonnées des boîtes englobantes étaient à zéro dans le fichier 'wider_face_train_bbx_gt.txt' et le fichier 'wider_face_val_bbx_gt.txt'. De cette manière on élimine les images où les visages sont plus difficiles à détecter du fait par exemple de l'occlusion ou du flou. On a aussi choisi de garder uniquement les images qui comportent au moins un visage car cela rend plus facile le calcul de la précision et du rappel. On a tout de même gardé les images contenant un nombre important de visages ou les images comportant des très petits visages. Les images dans le jeu de données étaient très grandes ce qui aurait consommé trop de mémoire et rendu l'apprentissage trop long, on a donc décidé de les redimensionner pour atteindre une taille de 256*256. Du padding est rajouté le long des lignes ou des colonnes pour préserver le format de l'image lors du redimensionnement.

Dans un premier temps, on a utilisé trois échelles de détection : 32*32, 16*16, 8*8. Pour chaque échelle on a utilisé trois boîtes d'ancrages.

Pour évaluer nos détections on a utilisé un score de précision, de rappel, ainsi que le score F1.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Les vrais positifs correspondent aux boites dont l'IOU maximale avec les boites appartenant à la vérité terrain est supérieur ou égal à 0.5. La somme des vrais positifs et des faux positifs pour une image correspond au nombre de visage prédits par le réseau. La somme des vrais positifs et des faux négatifs pour une image correspond au nombre de visages réellement présents dans l'image.

Ces mesures sont aussi utilisées pendant l'entraînement en tant que métrique pour évaluer les performances sur les données d'entraînement et de validation. Par ailleurs, ces métriques sont calculées après avoir effectué l'étape de non-max suppression de manière à éviter d'avoir des boites englobantes prédites qui se chevauchent sur plus de 50% de leur aire.

Avant d'être passées dans l'extracteur de caractéristiques, les images sont normalisées en déduisant la moyenne du jeu de données par canal et en divisant par l'écart type de ce même jeu de données par canal également.

Analyse des résultats

Le calcul de la précision, du rappel et du score F1 pour nos différents essais portent sur un jeu de données de test constitué de 0.1% de l'ensemble des images soit à peu près 500 images. De plus la durée d'entraînement est fixée à dix epochs pour tous les tests de manière à avoir une idée de la performance sans que cela ne soit trop long à entraîner. La taille du batch est de 32 et l'optimizer utilisé est Adam avec les paramètres par défaut (On a essayé d'utiliser d'autres learning rates mais les résultats étaient moins bons).

Notre test de référence auquel les autres tests vont être comparés a été obtenu en effectuant une normalisation des images de type 'standardisation' c'est-à-dire en soustrayant la moyenne et en divisant par l'écart type et ce, par canal. On a utilisé trois échelles en 32*32, 16*16 et 8*8 sur des images en 256*256. On a de plus utilisé trois ancres par échelle. Les poids affectés à la composante centroid x_y, la composante largeur_hauteur, la composante 'présence d'un objet' et la composante 'pas de présence d'objet' de la fonction de cout sont respectivement de 5, 5, 5 et 0.5. Voici les résultats obtenus :

Precision: 0.7926962625075834

Recall: 0.7634102511704396

F1 score: 0.7687627156828047

Changement de la méthode de normalisation

Nous avons également essayé de conduire un test avec les mêmes paramètres mais seulement en changeant la méthode de normalisation. On applique ici une normalisation min_max par canal. On effectue donc le calcul suivant par canal :

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Où x est la valeur du pixel du canal. On a obtenu les résultats suivants :

Precision: 0.6423185508877332

Recall: 0.6950315573385458
F1 score: 0.6536300330904773

Il semblerait donc que la 'standardisation' par canal produise de bien meilleur résultats que la normalisation min/max.

Changement du nombre d'ancres

Nous avons également essayé de faire tourner le code avec les mêmes paramètres et la standardisation par la moyenne et l'écart type, mais en changeant le nombre d'ancres par échelle. Dans le test suivant nous utilisons cinq ancres par échelle plutôt que trois. Voici les résultats :

Precision: 0.49960006560080583
Recall: 0.8093705682982704
F1 score: 0.5833975831654992

On remarque que les résultats sont assez significativement inférieurs à notre test de référence comportant trois ancres. Seul le résultat du rappel semble satisfaisant et comparable au rappel obtenu dans le test de référence.

Changement du nombre d'échelles

Un autre test effectué consiste à garder trois ancres par échelle mais à augmenter le nombre d'échelles. En utilisant quatre échelles en 32*32, 16*16, 8*8 ainsi que 4*4 on obtient les résultats suivants, toujours pour dix epochs:

Precision: 0.692782270140761
Recall: 0.7614591567353609
F1 score: 0.7072837308644996

En utilisant cinq échelles (64, 32, 16, 8, 4) toujours avec trois ancres par échelle et un entraînement de dix epochs on obtient les résultats suivants :

Precision: 0.7829215973870062
Recall: 0.79432960495912
F1 score: 0.7784470668352974

On remarque qu'il s'agit du meilleur score obtenu jusqu'à présent avec notamment le score F1 le plus élevé de nos tests pour dix epochs.

Changement des poids de la fonction de cout

On peut également faire un test où l'on garde les mêmes paramètres que le test de référence mais on change les poids de la fonction de cout de la façon suivante :

Poids $xy = 1$

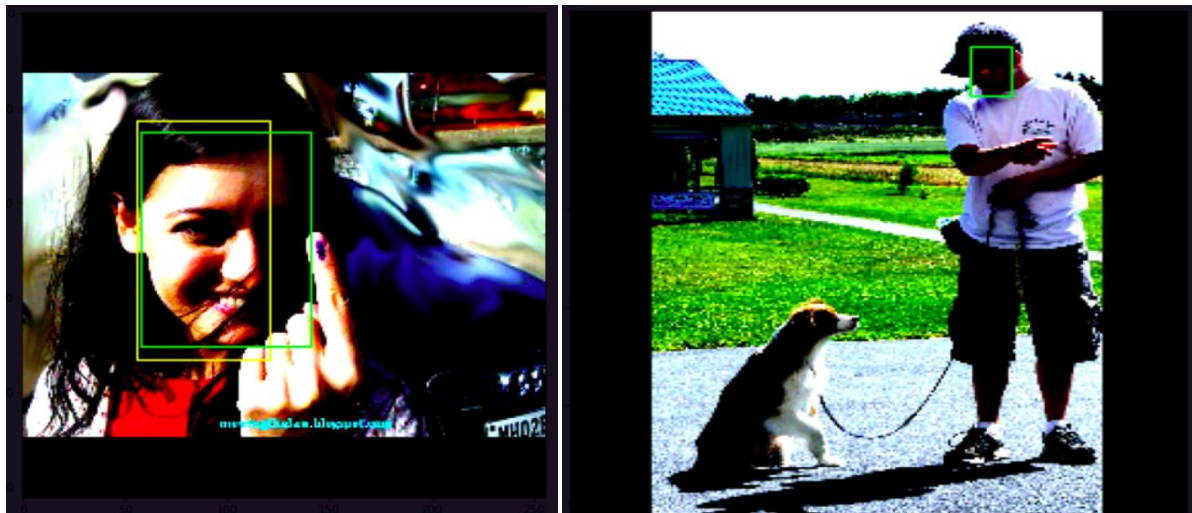
Poids wh = 1
Poids objet = 1
Poids non objet = 0.5

On a donc simplement fait passer les trois premiers poids de cinq à un. Voici les résultats obtenus :

Precision: 0.7835912282453164
Recall: 0.7719569027896034
F1 score: 0.7680876561278269

On constate que les résultats sont très similaires au test de référence. Néanmoins, visuellement, en regardant les images, on remarque qu'un nombre trop important de visages ne sont pas détectés. Il serait donc peut être utile d'augmenter le poids de la composante liée à la détection des objets dans la fonction de cout (poids objet).

Visuellement, on remarque également que les parties ombragées des visages provoquent un mauvais dimensionnement des boites englobantes ou pire une absence de détection.



Les deux cas ci-dessus sont problématiques : sur la photo de gauche on voit que la boîte trouvée en jaune est mal dimensionnée. En particulier elle ne couvre pas la partie droite du visage qui est dans l'ombre. Sur l'image de droite tout le visage de la personne est dans l'ombre ce qui pourrait expliquer l'absence de détection.

En augmentant le poids de la composante de la fonction de cout liée à la présence d'un objet on remarque visuellement que moins de visages ne sont pas détectés. Par ailleurs les résultats numériques sont également satisfaisants :

Poids objet : 3
Poids xy = 1
Poids wh = 1
Poids pas d'objet = 0.5

Voici les résultats:

Precision: 0.7594535063916066
Recall: 0.819123200054232
F1 score: 0.7729408915353684

On a donc obtenu un meilleur rappel mais une précision inférieure. Le score F1 a légèrement augmenté. Il est assez logique que le rappel ait augmenté puisqu'on a augmenté le poids de la détection des visages. En effet le rappel est le nombre de cas identifiés comme positif divisé par le nombre total de cas vraiment positif. En détectant davantage de visages, on augmente donc le rappel. Il est aussi assez logique que la précision ait légèrement diminuée car forcément si on détecte plus facilement les visages, on va aussi détecter des choses qui ne sont pas des visages. En effet la précision étant le nombre de cas positifs détectés divisé par le nombre de cas identifiés comme positifs, on est en train d'augmenter le dénominateur et donc on fait baisser la précision.

Il est donc difficile de faire augmenter à la fois la précision et le rappel. Les poids de la fonction de cout doivent être fixés en fonction des priorités que l'on se fixe. Si la priorité est de détecter absolument tous les visages, peu importe les erreurs, alors on peut fixer un poids élevé pour la composante de détection d'objets ce qui aura pour conséquence de faire augmenter le rappel mais également de faire baisser la précision. A l'inverse, si on est très attaché à ne détecter que des visages et qu'on accepte pour cela de ne pas détecter certains visages, alors on peut augmenter le poids de la composante de pénalité (poids non obj) ce qui aura pour conséquence d'augmenter la précision au détriment du rappel.

Dans le dernier test effectué, on constate tout de même que le rappel a augmenté de presque six points par rapport au test de référence (le tout premier), là où la précision n'a diminué que d'un peu plus de trois points. C'est ce que montre le score F1 qui est une synthèse de la précision et du rappel. Celui-ci a bel et bien augmenté. On peut en déduire que ces derniers paramètres sont plus intéressants que ceux du test de référence.

Tests sur trente epochs

Il a été décidé de mener deux tests en entraînant le réseau sur trente epochs.

Voici les paramètres du premier test :

- Trois échelles en 32*32, 16*16, 8*8
- Trois ancres par échelle
- Taille du batch de 32
- Taille de l'image de 256*256
- Images normalisées en utilisant la 'standardisation' par canal (soustrait la moyenne et divise par l'écart type)
- Poids centroid xy de 1
- Poids hauteur largeur wh de 1
- Poids présence d'un visage de 3
- Poids absence de visage de 0.5

Voici les résultats obtenus pour ce test :

Precision: 0.8486218882445299

Recall: 0.8291729514839312

F1 score: 0.8313319745709685

Le second test utilise presque les mêmes paramètres que le test précédent. Le seul changement est le nombre d'échelles qui passe à cinq avec une échelle de 64 par 64 cases, une autre de 32 par 32 cases, une de 16 par 16, une de 8 par 8, et enfin la dernière de 4 cases par 4 cases. Voici les résultats obtenus :

Precision: 0.8633024927142577

Recall: 0.8705740545041907

F1 score: 0.8584516160727783

On remarque que ce dernier test est le plus performant. Le modèle semble avoir bénéficié du nombre d'échelle supérieur.