



ENSAE - Institut Polytechnique de Paris

PROJET DE STATISTIQUE APPLIQUÉE

**DEEP LEARNING POUR L'EXTRACTION D'INFORMATIONS
DANS DES DOCUMENTS SCANNÉS**

Etudiants

BERREBI Nathane :
Nathane.berrebi@ensae.fr

YAPI Wilfried :
Wilfried.yapi@ensae.fr

JULIEN Nicolas :
Nicolas.julien@ensae.fr

LOUBEYRE Antoine :
Antoine.loubeyre@ensae.fr

Encadrants

AQUEREBURU Kent:
kent.aquereburu@socgen.com

JUILLARD Marc :
marc.juillard@socgen.com

Contents

1	Introduction	1
2	Prérequis	1
2.1	CNN : Réseaux de neurones convolutifs	1
2.2	Historique des architectures CNN	4
3	Modèles de détection d'objets	5
3.1	Classification et localisation d'un objet	5
3.2	Détection objets	7
3.2.1	Approche classique	7
3.2.2	Réseaux entièrement convolutifs FCN	7
3.2.3	YOLO : You Only Look Once	9
3.2.4	R-CNN	13
3.2.5	Fast R-CNN	19
3.2.6	Faster R CNN	21
4	Métriques d'évaluation	26
4.1	Indice de Jaccard (Intersection over Union IoU)	26
4.1.1	Description de l'IoU pour la détection visuelle	26
4.1.2	Pourquoi utiliser cette métrique ?	28
4.2	Moyenne de la précision moyenne : mAP	29
4.3	Introduction	29
4.3.1	Rappels sur le recall et la precision	29
4.3.2	La courbe de Precision-Recall et l'Average Precision	30
4.3.3	Du score de prédiction à la classification: recréer une métrique binaire à partir de l'IoU	33
4.3.4	La mAP	34
4.4	Résumé des métriques et de la construction de la MAP	36
5	Transfert Learning	37
5.1	Explication TL	37
6	Base de données	39
6.1	Informations sur la base de données	39
7	Démarches algorithmiques	41
7.1	Preprocessing	41

1 Introduction

Le projet de statistique appliquée porte sur l'utilisation des méthodes de Deep Learning pour l'extraction d'informations dans des documents scannés.

Pour ce projet, nous avons pour objectif avec la Société Générale Assurances, d'automatiser le traitement administratif des documents.

Pour cela, nous devons définir un modèle capable de prédire l'emplacement des tableaux dans un document scanné. Ces prédictions seront traitées afin d'extraire les informations textuelles de ces tableaux.

Nous pourrions en particulier appliquer des méthodes de NLP afin de répondre à un cas d'usage administratif où tout autre objectif fixé par la Société Générale Assurances.

Comment modéliser ce cas pratique de détection d'objets ?

2 Prérequis

Les réseaux de neurones artificiels (RNA) sont au cœur de l'apprentissage profond. Ils permettent de résoudre des tâches complexes comme la classification d'images et la détection d'objets.

Afin de bien comprendre les modèles utilisés dans ce projet, il est nécessaire d'avoir des connaissances sur le fonctionnement des RNA.

2.1 CNN : Réseaux de neurones convolutifs

Les architectures classiques de CNN consistent à empiler des couches de convolution (chacune suivie d'une couche de ReLU¹, type de fonction d'activation) puis une couche de pooling et ainsi de suite.

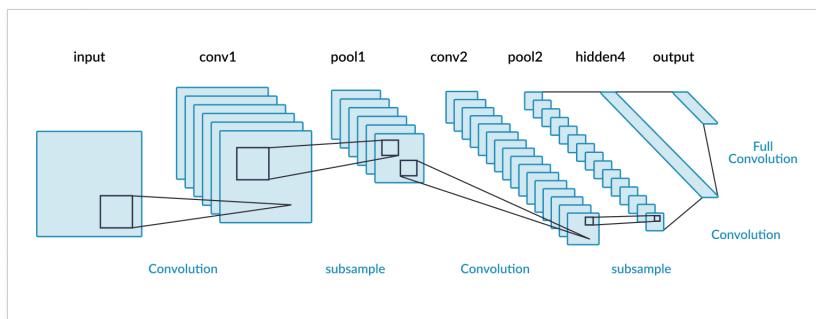


Figure 1: Architecture des réseaux de neurones convolutifs simple [2]

¹La couche d'activation ReLU (Rectified Linear Units) remplace toutes les valeurs négatives reçues en entrées par des zéros. Voir détails en référence [1]

Utilisation des réseaux de neurones convolutifs :

Etape 1 : Transformer l'image en cartes de caractéristiques (pixels). Cette étape est appelé *preprocessing*.

Pour cela, nous appliquons un filtre sur l'image. Cette méthode s'appelle le produit de convolution (kernel). On applique un kernel (matrice de taille $n \times n$) sur chaque partie de l'image en calculant le produit scalaire puis on réitère l'opération en déplaçant le kernel d'un pas p .

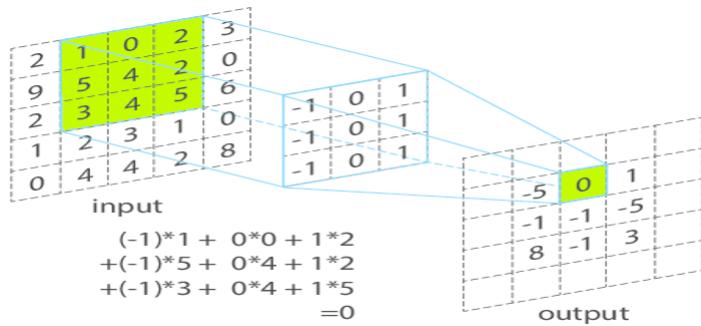


Figure 2: Produit de convolution et produit scalaire [3]

On obtient une image réduite selon la taille du kernel. L'image résultante se nomme *carte de caractéristique*.

Pour les images en couleurs, on applique un kernel sur chaque profondeur de l'image, c'est-à-dire 3 kernels pour 3 canaux d'images RVB. (ou un kernel à 3 dimensions pour une image RVB)

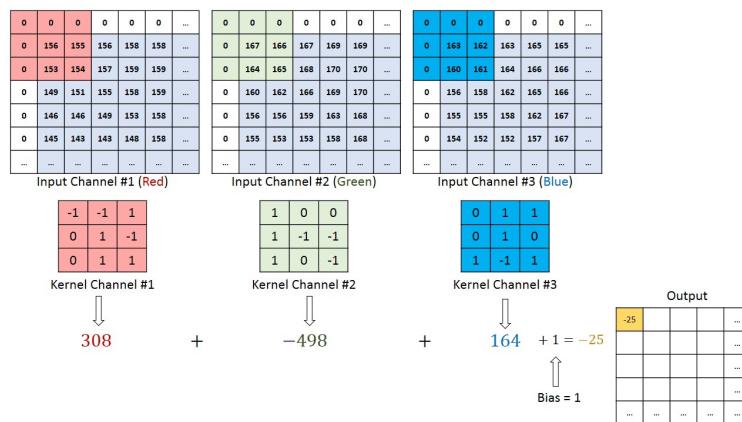


Figure 3: Produit de convolution sur une image de couleur [4]

Il est possible d'appliquer des kernels à des cartes de caractéristiques pour mettre en évidence certaines caractéristiques. Cette étape représente les couches N+1 du CNN.

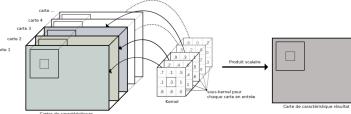


Figure 4: Produit de convolution de cartes de caractéristiques [5]

Etape 2 : Opération de Max Pooling

C'est un principe de sous-échantillonnage qui permet de réduire la dimension de l'input tout en conservant l'information la plus importante. Il existe principalement 2 types de pooling : valeur maximale et valeur moyenne.

Comme précédemment, on définit un noyau de pooling, c'est-à-dire sa taille, son nombre de pas ainsi que son type de sortie (valeur maximale ou moyenne). Ce noyau consiste à prendre les valeurs de la carte pour une localisation donnée. Si l'on choisit la valeur maximale, seule cette valeur passe à la couche suivante, les autres sont ignorées.

Remarque : Le réseau de neurones détermine lui-même les kernels de convolution par apprentissage. Au fur et à mesure de l'apprentissage, les poids des kernels s'adaptent pour faire ressortir la caractéristique la plus pertinente de l'image.

Résumé : L'image rétrécit au fur et à mesure qu'elle traverse le réseau, mais elle devient également de plus en plus profonde (nombre de cartes de caractéristiques grâce aux couches de convolutions et de pooling). Une technique consiste à ajouter une opération de *Flatten*. Cette opération transforme une matrice bidimensionnelle en un vecteur de caractéristiques.

A la fin du réseau, on ajoute souvent des couches de neurones intégralement connectées.²

²Couche Fully Connected (FC) : Ces couches sont placées en fin d'architecture de CNN et sont entièrement connectées à tous les neurones de sorties (d'où le terme fully-connected). Après avoir reçu un vecteur en entrée (Flatten), la couche FC applique successivement une combinaison linéaire puis une fonction d'activation dans le but final de classifier l'input image. Elle renvoie en sortie, un vecteur de taille d correspondant au nombre de classes dans lequel chaque composante représente la probabilité que l'image appartienne à cette classe

2.2 Historique des architectures CNN

Pour bien comprendre la classification d'images et la détection d'objets, il est utile de connaître les types d'architectures de CNN déjà existantes.

- LeNet-5 : L'architecture est constituée de plusieurs couches de convolution suivie de couches de pooling moyenne. La fonction de perte utilisée est l'entropie croisée^{3]} car elle pénalise davantage les mauvaises prédictions en produisant des gradients plus importants et en convergeant plus rapidement. [7]
- AlexNet : Cette architecture ressemble beaucoup à la précédente, la seule différence est au niveau de l'empilation des couches de convolutions. Ici, les couches sont directement empilées les unes au-dessus des autres sans couche de pooling. Pour diminuer le sur-ajustement, les auteurs ont utilisé 2 techniques de régularisation, la technique du dropout et de l'augmentation de données. [8]
- GoogLeNet : Dans cette architecture, le réseau de neurones est plus profond grâce à la présence de sous-neurones appelés "inception". Le module Inception permet de faire ressortir des motifs à des échelles différentes. [9]
- VGGNet : Cette architecture se base sur la répétition de blocs de 2 ou 3 couches de convolution et d'une couche de pooling et ainsi de suite. La dernière couche est un réseau final dense de 2 couches cachées et une couche de sortie. [10]
- ResNet : Cette architecture se base sur la notion de *réseau résiduel*. Dans ce réseau de 152 couches, les auteurs ont utilisé des *connexions de saut* pour l'entraînement. C'est ce qu'on appelle *l'apprentissage résiduel*.
- Xception : François Chollet (créateur de Keras), fusionne l'idée de GoogLeNet et de ResNet mais il remplace les modules d'inception par une couche spéciale appelée couche de convolution séparable en profondeur.
- SENet : Cette architecture regroupe celle de GoogleNet et ResNet en améliorant leurs performances. Ces réseaux sont appelés les SE-Inception et SE-ResNet.[11]

³L'entropie croisée n'est pas spécifique à l'architecture Lenet-5 mais la plupart des réseaux de neurones l'utilise. Voir référence [6]

3 Modèles de détection d'objets

Dans les sections précédentes, nous avons montré les différents types d'architecture de CNN de classification. Nous allons à présent montrer une autre fonctionnalité des CNN, à savoir la détection d'objets dans une image.

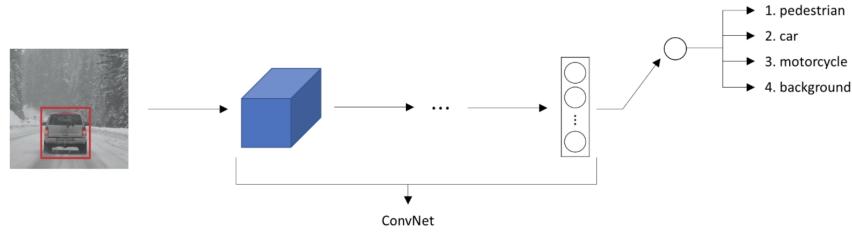


Figure 5: CNN Classification multiclass [12]

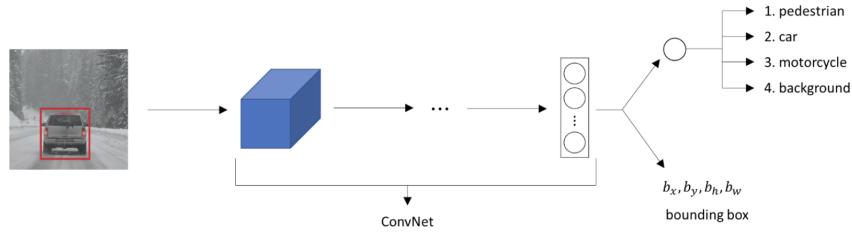


Figure 6: CNN classification multiclass et bounding box

3.1 Classification et localisation d'un objet

L'objectif de la localisation est de prédire un rectangle d'encadrement autour de l'objet, appelé *bounding box*.

L'approche classique consiste à prédire 4 paramètres :

x, y : centre du rectangle

w, h : largeur et hauteur du rectangle

ou

x, y, z, t : les coordonnées des 4 côtés du rectangle

La première étape consiste à créer des bounding box à la main ou à l'aide d'outil open source d'étiquetage d'images.

La seconde étape consiste simplement à ajouter une deuxième couche de sortie dense avec 4 unités et l'entrainer avec la fonction de perte MSE ou l'indice de Jaccard (IoU).

Pour résumer, notre output comportera 8 sorties pour une classification 3 classes :

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Figure 7: Output classification multiclass et bounding box

$$P_c(\text{objectness}) = \begin{cases} 1 & \text{si présence d'un objet appartenant à une des classes} \\ 0 & \text{sinon} \end{cases}$$

b_x, b_y, b_h, b_w : les paramètres de la bounding box détectée

$$c_i = \begin{cases} 1 & \text{si objet appartient à la classe i} \\ 0 & \text{sinon} \end{cases}$$

Remarque : Lorsqu'une image ne contient pas d'objet, la fonction de perte prend en compte ce facteur dans son calcul⁴.

⁴Voir les fonctions de perte dans les papiers de recherche des modèles Yolo[13] et Faster RCNN[14]

3.2 Détection objets

3.2.1 Approche classique

Il apparaît donc qu'utiliser un réseau de convolution serait la bonne approche pour détecter un objet dans une image. Le problème est que le nombre d'objets à détecter varie d'une image à une autre, or la variable d'output d'un CNN est statique. Une solution naïve serait d'appliquer le CNN sur différentes fenêtres de l'image de manière à chercher tous les objets, mais ce n'est pas suffisant car nous ne connaissons pas la proportion de l'objet. Nous pourrions être dans une fenêtre où il y a déjà plusieurs objets mais aussi dans une fenêtre incluse dans la superficie de l'objet.

Pour pallier ce problème, nous pourrions essayer de parcourir l'image en appliquant un CNN sur des fenêtres de tailles fixes et de faire varier la taille de la fenêtre. Cette solution pourrait fonctionner mais celle-ci est chronophage et nécessite une puissance de calcul trop importante. C'est dans ce cadre que des modèles comme FASTER R CNN et YOLO ont été conçus.

3.2.2 Réseaux entièrement convolutifs FCN

Avant d'étudier le modèle Yolo, nous allons analyser le modèle FCN (réseaux entièrement convolutifs). L'architecture du modèle de Yolo est très proche de ce modèle.

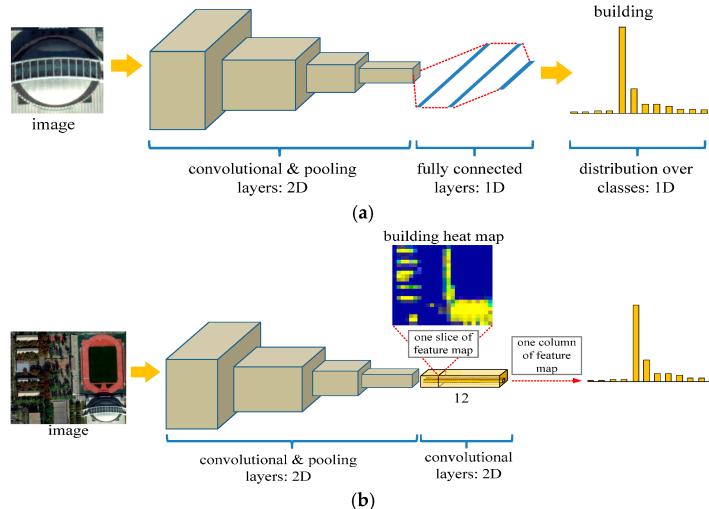


Figure 8: Architecture CNN (a) vs FCN (b) [15]

L'idée du FCN est de montrer qu'on pouvait remplacer les dernières couches denses d'un réseau de neurones convolutifs par des couches de convolutions.

Explication

Supposons qu'un réseau de neurones possède une couche dense de 200 neurones placé au-dessus d'une couche de convolution produisant 100 cartes de caractéristiques de taille 7x7. Le résultat de la couche dense sera donc un tenseur de forme [taille de lot⁵. Si nous remplaçons cette couche dense par une couche de convolution qui utilise 200 kernels de taille 7*7. Le résultat sera 200 cartes de caractéristiques de taille 1*1 (car la kernel a la même taille que les cartes de caractéristiques d'entrées). Le résultat de la sortie sera un tenseur de forme [taille de lot, 1, 1, 200].

L'inconvénient d'une couche dense se situe au niveau des entrées, en effet, elle nécessite une taille spécifique puisqu'elle accorde un poids par entrées. A l'inverse, une couche de convolution peut traiter n'importe quel type de taille. Le premier avantage d'un FCN est qu'il peut traiter des images de tailles différentes.

Supposons que nous ayons une image de taille 448x448 comme input qui passe dans un CNN qui génère 1 carte de caractéristique de taille 14x14. L'ajout d'une couche de convolution (1 kernel 7*7, pas de 1) permet de transformer cette carte de caractéristiques (1 kernel donc 1 dimension) en 1 grille de taille 8*8 (14 - 7 +1=8).

Le FCN va donc traiter l'intégralité de l'image en une seule fois et produire une grille de taille 8*8. Chaque *cellule* de cette grille contient 10 valeurs :

- 5 probabilités de classe
- 1 score d'*objectness*
- 4 coordonnées de bounding box

Cela revient à prendre l'approche classique et déplacer un CNN sur l'image sauf qu'ici on ne regarde l'image qu'une seule fois.

⁵La taille du lot (ou batch size) représente le nombre d'échantillons utilisés pendant l'entraînement.

3.2.3 YOLO : You Only Look Once

L'architecture de YOLO a été proposé par Joseph Redmon dans un article publié en 2015. A ce jour, il existe 5 améliorations du modèle.

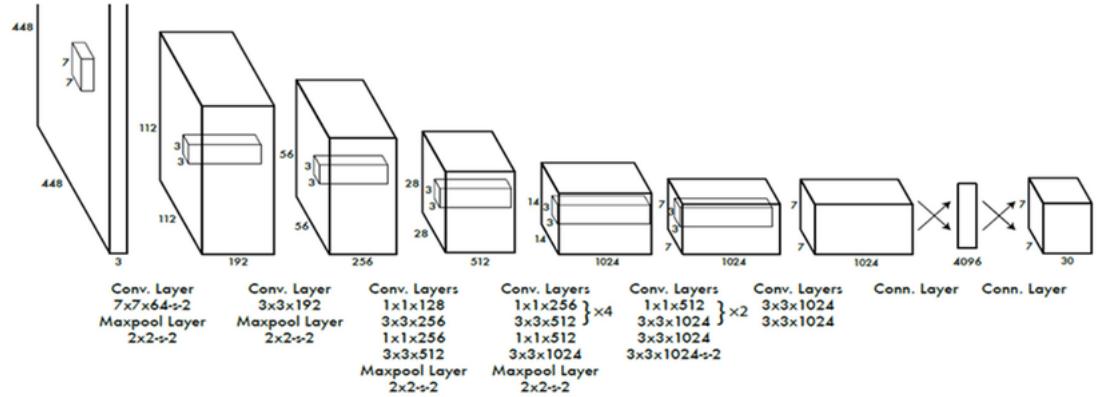


Figure 9: Architecture YOLO [16]

Explication

L'architecture du modèle YOLO est assez proche du modèle de FCN à quelques détails près.

- Précédemment, nous avions un rectangle par cellule qui prédisait 10 valeurs. Le modèle de YOLO produit quand à lui B rectangles par cellule de la grille de taille S*S. Chacun de ces rectangles possède un score *d'objectness*.
- L'output est donc le suivant :
 - B : nombre de bounding box
 - Pour chaque bounding box nous avons 5 sorties (4 coordonnées de rectangle + 1 score objectness)
 - C : Classes de probabilités
 - $\text{dim(Output)} = S \times S (B \times 5 + C)$
- En plus de prédire les coordonnées du centre du rectangle (x,y), YOLOv3 prédit "un décalage par rapport à la cellule". Yolo est entraîné pour prédire uniquement les bounding box qui ont un centre présent dans cette cellule (le cadre peut dépasser).

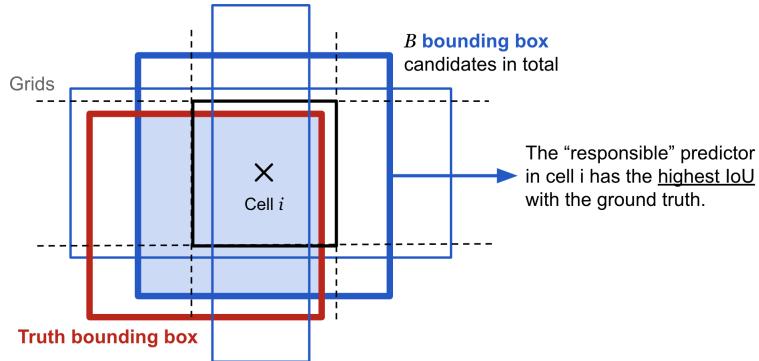


Figure 10: Bounding box YOLO [17]

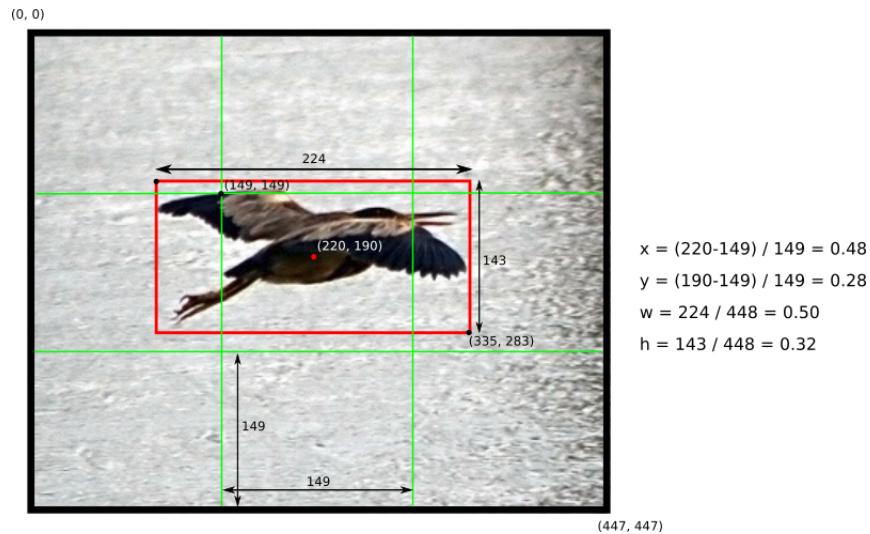


Figure 11: Calcul des coordonnées d'un bounding boxes [18]

- Avant d'entrainer, le modèle de YOLOv3 recherche les meilleures formes de bounding box pour la prédiction (appelé Anchor Boxes). Pour cela, il applique l'algorithme des **k-means** :
 - 1 : Tout d'abord, il suffit d'extraire les coordonnées de chaque bounding box du training data.
 - 2 : Nous calculons la largeur et la longueur de chaque bounding box par une simple soustraction.
Largeur = abscisse coin inférieur droit - abscisse coin inférieur gauche
Longueur = ordonnée coin inférieur droit - ordonnée coin inférieur gauche

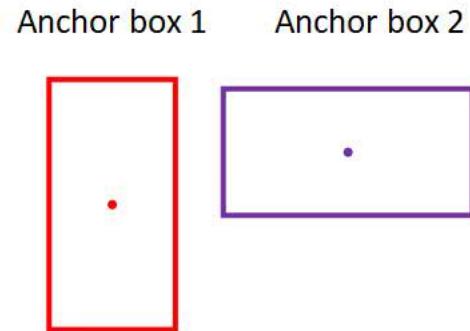


Figure 12: type d'Anchor boxes [19]

- 3 : Initialiser k boîtes d'ancrage
- 4 : Calculer la valeur IoU de chaque boîte d'ancrage et de bounding box
- 5: Après le calcul de l'étape 4, nous pouvons obtenir l'erreur $D = 1 - \text{IOU}$ pour chaque bounding box. Ensuite, l'algorithme compare les erreurs et affecte chaque bounding box à une anchor box.

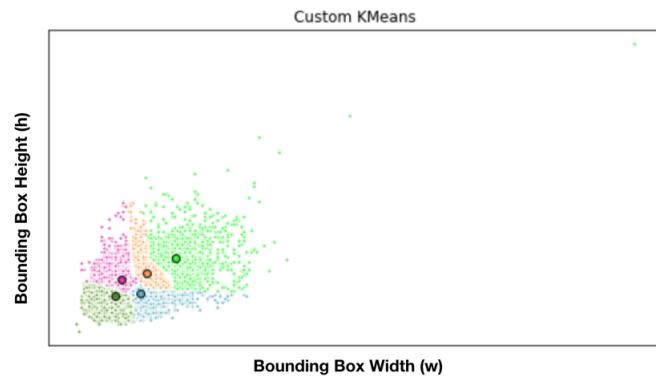


Figure 13: K-means Bounding box [20]

- L'algorithme prédit non pas la vraie valeur des bounding box mais un redimensionnement (dû à chaque couche de convolution).

Plus précisément, le réseau prédit le logarithme des facteurs de redimensionnement vertical et horizontal du rectangle d'ancrage.

Pourquoi utiliser cette méthode ?

Cela permet à l'algorithme de prédire les rectangles les plus probables de se trouver dans l'image et donc augmentera la rapidité du modèle.

- Le réseau est entraîné sur des images de tailles différentes. Cela lui permet de détecter des objets de tailles différentes.

Avant d'introduire le modèle *Faster R-CNN*, nous proposons de suivre l'évolution des approches le précédent.

3.2.4 R-CNN

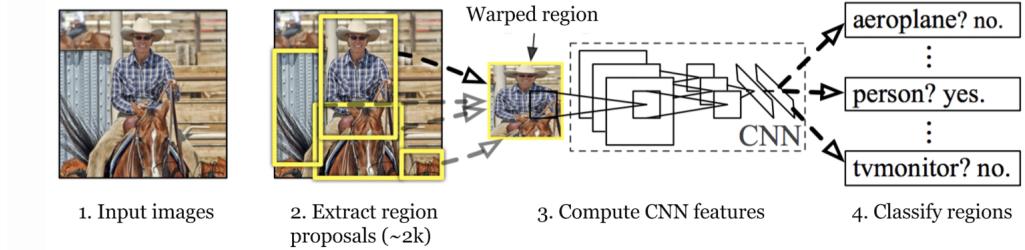


Figure 14: Architecture R-CNN [21]

Contrairement à l'algorithme YOLO, le R-CNN utilise un CNN sur **une partie de l'image** et non sur l'image entière.

Pour sélectionner les parties/régions de l'image à donner dans le CNN, l'algorithme utilise en amont un autre algorithme, appelé *Selective Search*.

L'algorithme selective search[22] renvoie 2000 régions d'intérêts pour chaque image ; ce sont les *ROI*. Souvent ces ROI sont indiquées par des bounding boxes. Chaque région d'intérêt contient potentiellement un objet. Nous en détaillons le fonctionnement.

Voici un exemple des étapes de fonctionnement de la selective search :

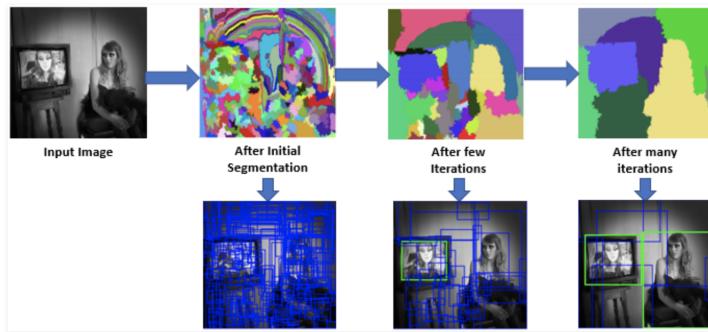


Figure 15: On entre en input une image. Puis l'algorithme *selective search* génère une segmentation de l'image initiale. L'algorithme combine ensuite les régions similaires pour en former une plus grande, via la similitude des couleurs, formes, tailles etc.. On obtient donc ensuite nos ROI

Etape 2 : Utilisation des ROI

Au sujet de la recherche selective il existe plusieurs méthodes pour définir des *ROI*, mais il faut noter que le R-CNN ne dépend pas de la méthode choisie. De plus il n'y a pas de méthode d'apprentissage pour faire la selective search, c'est plus une méthode qui segmente l'image. Le nombre de faux positifs est donc assez important, mais tous les vrais positifs sont présents, c'est tout l'intérêt de l'algorithme.

Comparé à la méthode naïve, nous avons donc diminué significativement le nombre de fenêtres à analyser. Cependant 2000 fenêtres reste un nombre assez important et ce type d'algorithhme ne peut être utilisé en temps réel (sur une vidéo par exemple).

R-CNN utilise donc l'algorithme *selective search* pour définir 2000 ROI (bien que toujours très grand, le nombre de fenêtre a beaucoup diminué grâce à cette méthode). Les régions obtenues peuvent potentiellement contenir des objets à détecter.

La suite consiste à appliquer un CNN (de type Alex-Net ou Conv-Net) sur toutes ces *ROI*. Cependant, les ROI sont de tailles différentes, il faut donc les normaliser via une méthode de *warping* afin que le même CNN puisse être utilisé sur tous les inputs. Un paramètre *p* est ajouté pour indiquer la quantité de dilatation de la région proposée initiale afin d'évaluer la prédiction finale.

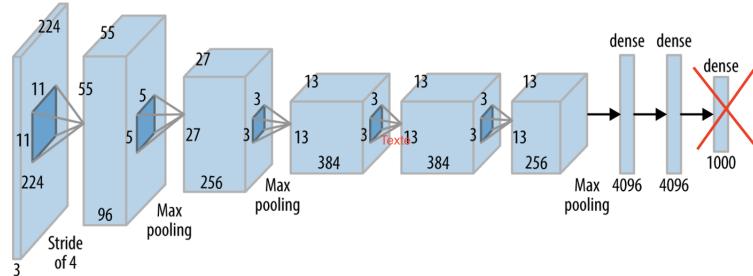


Figure 16: Struture AlexNet [23]

Les figures 22 et 23 montrent le CNN utilisé (Alex Net). Il possède 5 couches de convolutions et 2 couches entièrement connectées. Comme vous pouvez le voir sur la figure 22, la dernière couche de classification est remplacée par une couche de classification softmax initialisée aléatoirement pour les N classes d'objets + 1 classe pour la détection d'objet.

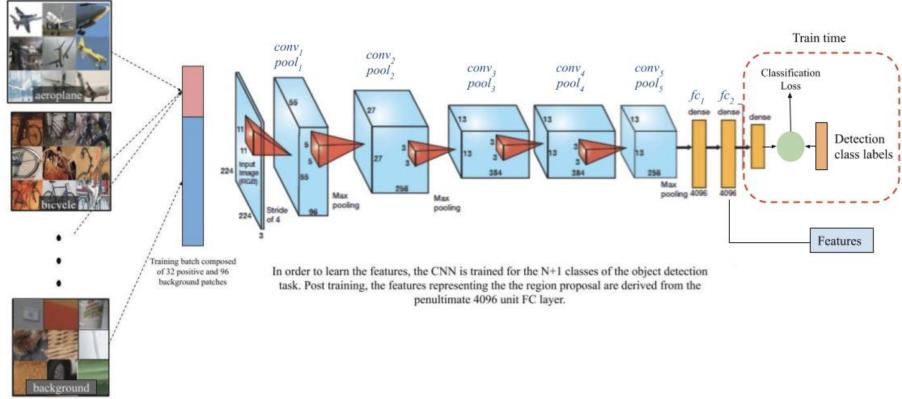


Figure 17: Struture R-CNN

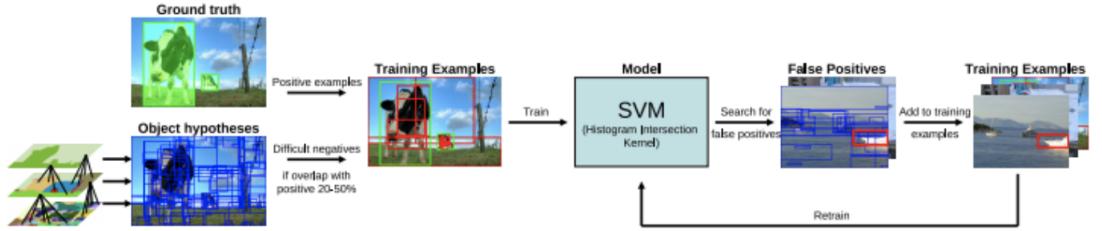


Figure 18: Classification des ROI [24]

Chaque ROI est ensuite classé en fonction de son IOU par rapport à la bounding boxes initiale.

$$\text{"ClassificationROI"} = \begin{cases} \text{positive} & \text{si } IoU \geq 0.5 \\ \text{negative} & \text{si } IoU < 0.5 \end{cases}$$

Pour chaque itération de l'algorithme, on utilise un lot de 32 ROI positive et 96 négatives, ce qui représente un lot de 128 ROI.

L'output final représente un vecteur de 4096 caractéristiques pour chacune des 2000 ROI proposées (voir figure 23).

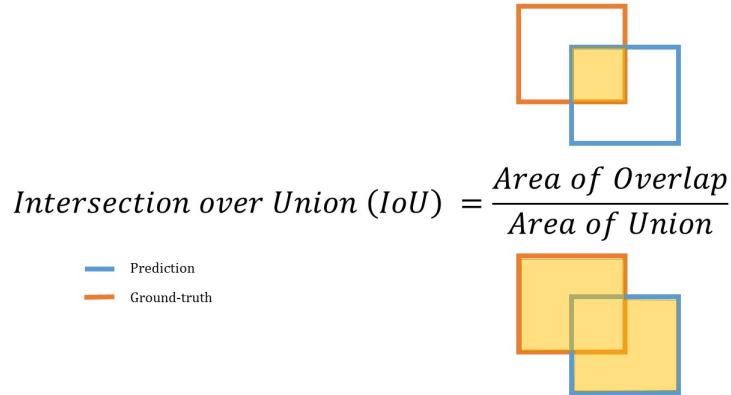


Figure 19: IoU overlap [25]

Etape 3 : Utilisation du SVM pour la classification

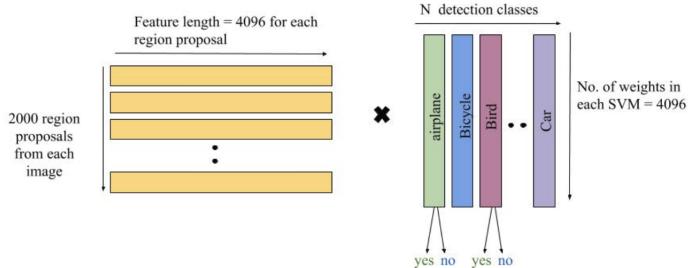


Figure 20: Struture R-CNN SVM

Nous donnons la *convolutionnal feature map* en entrée d'un SVM dans l'objectif de classifier la zone d'intérêt (*ROI*); l'idée est que le SVM classe les features de façon binaire; soit oui la zone correspond à l'objet à détecter, soit c'est l'arrière plan. Il y a donc autant de SVM que de classes d'objets à détecter.

L'input est donc le vecteur de 4096 caractéristiques pour chaque région.

Chaque région ayant un IoU inférieur à 0.3 par rapport à la bounding box initiale sont classées négatives pendant l'entraînement. Les positives sont les bounding boxes elles-mêmes. Les autres régions sont ignorées.

Par la suite, nous avons un produit matriciel avec d'un côté, une matrice de taille 2000*4096 et de l'autre un SVM avec une matrice de 4096 *N (= nombre de classe).

L'output final représente l'ensemble des ROI positive pour chaque classe

d'objets.

Bounding box régression

L'algorithme inclut une étape de régression dans le but d'améliorer les performances de la localisation d'objets. En effet, l'algorithme inclut une étape afin d'apprendre les corrections de la localisation et de la taille de la bounding box prédite (par la sélective search).

$\begin{aligned} P^i &= (P_x^i, P_y^i, P_w^i, P_h^i) \\ G &= (G_x, G_y, G_w, G_h) \end{aligned}$ <p style="text-align: center;">(1)</p>	$\begin{aligned} t_x &= (G_x - P_x)/P_w \\ t_y &= (G_y - P_y)/P_h \\ t_w &= \log(G_w/P_w) \\ t_h &= \log(G_h/P_h). \end{aligned}$ <p style="text-align: center;">(2)</p>	$\begin{aligned} \hat{G}_x &= P_w d_x(P) + P_x \\ \hat{G}_y &= P_h d_y(P) + P_y \\ \hat{G}_w &= P_w \exp(d_w(P)) \\ \hat{G}_h &= P_h \exp(d_h(P)). \end{aligned}$ <p style="text-align: center;">(3)</p>	$\boxed{\begin{aligned} d_{\star}(P) &= \mathbf{w}_{\star}^T \phi_5(P) \\ \mathbf{w}_{\star} &= \operatorname{argmin}_{\tilde{\mathbf{w}}_{\star}} \sum_i^N (t_{\star}^i - \tilde{\mathbf{w}}_{\star}^T \phi_5(P^i))^2 + \lambda \ \tilde{\mathbf{w}}_{\star}\ ^2 \end{aligned}}$ <p style="text-align: center;">(4)</p>
---	--	--	--

Figure 21: Bounding box régression

- L'équation 1 indique les coordonnées des régions prédites P et les coordonnées initiales de la bounding boxes G .
- L'équation 2 représente les transformations de la bounding box initiale.
- L'équation 3 représente l'estimation des transformations. \hat{G} représente la prédiction de la bounding box initiale corrigée calculé à l'aide de la prédiction de la box P et la transformation prédite $d_k(P)$
- L'équation 4 représente un modèle linéaire utilisé dans la couche 5 (figure 23), d'où ϕ_5 .

L'output final représente pour chaque ROI positive de chaque classe prédite par le SVM, une bounding boxes corrigées de l'objet recherché.

Les étapes du R-CNN en résumé :

- On prend un CNN pré-entraîné pour détecter N catégories d'objets
- Une image est donnée en input d'un algorithme de type selective search qui renvoie K Region Of Interest *ROI*
- Nous appliquons une méthode de *warping* pour redimensionner la taille des K *ROI* afin que K CNN puisse les prendre en input
- Chaque CNN est ré-entraîné et renvoie sa *convolutionnal feature map*
- Pour chaque catégorie d'objets, un SVM par catégorie d'objets classifie l'output du CNN (donc les features détectés) de façon binaire. N *SVM* sont donc utilisés par *ROI*

- Enfin nous entraînons un modèle de régression linéaire pour améliorer les bounding box initiales

Inconvénients toujours présents:

- 2000 ROI reste un nombre trop important de fenêtres et donc le temps d'exécution est trop long.
- Appliquer un CNN par ROI, et donc K CNN, est encore trop long

Le temps d'exécution moyen pour une image est de 50 secondes. On ne peut donc pas l'utiliser en temps réel.

3.2.5 Fast R-CNN

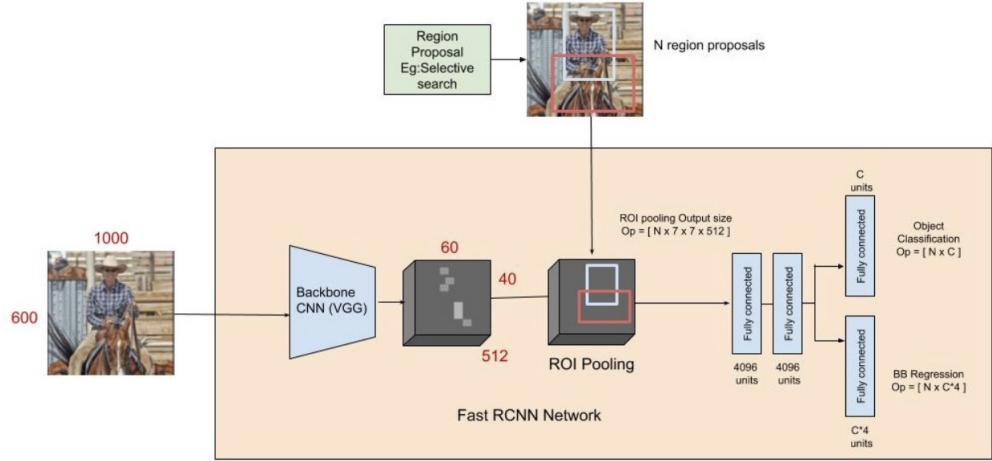


Figure 22: Structure d'un algorithme Fast R-CNN [26]

Cette approche est similaire à celle du R-CNN, et reprend la méthode de SPP-net. Mais au lieu d'appliquer d'abord la selective search sur l'image, et d'entrer les ROI dans un CNN, on applique directement le CNN sur l'image initiale ce qui génère une convolutional feature map grâce à laquelle nous trouvons les ROI via la selective search.

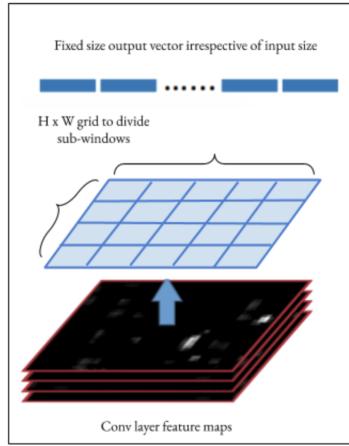


Figure 23: Architecture ROI Pooling

Comme le montre la figure 28, le CNN se compose d'une couche de ROI pooling suivi de 2 couches de réseaux de neurones intégralement connectés puis d'une séparation entre 2 branches. D'un côté, nous avons une couche de neurones intégralement connectés pour la classification (Softmax pour K+1 catégories d'objets) et de l'autre pour la régression.

Comme le montre la figure 28, l'image entre dans le CNN afin d'obtenir des cartes de caractéristiques. Chaque carte de caractéristiques est envoyé à la couche de ROI pooling. En parallèle, des ROI sont proposés via l'algorithme selective search. La couche de ROI pooling divise les cartes de caractéristiques ou sous fenêtres en fonction des ROI proposées par le selective search. Nous obtenons des sous-fenêtres de taille HxW. L'algorithme effectue ensuite une opération de pooling dans chacune des sous-fenêtres. A la fin de cette opération, nous obtenons une sortie de taille Nx7x7x512, où N est le nombre de ROI proposées.

Ensuite, nous appliquons les différentes couches successives de classification et BB-régression. La branche Softmax classification nous renvoie la probabilité que la ROI contienne un objet appartenant aux classes et la BB régression nous renvoie les bounding boxes prédites corrigées.

La fonction de régression est donnée par la formule suivante :

$$L_{loc}(t^u, v) = \sum_{i \in x, y, w, h} smooth_{L_1}(t_i^u - v_i)$$

v_i : représente la vraie valeur de la bounding boxes initiale

La fonction de classification est donnée par la formule suivante :

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \leq 1]L_{loc}(t^u, v)$$

L'interet est donc de n'avoir appliqué qu'une seule fois le CNN au lieu de 2000 fois.

Le temps d'exécution du Fast R CNN est de 2 secondes par image. Bien que 25 fois plus rapide que le R-CNN il est toujours impossible de l'appliquer en temps réel. La cause est l'algorithme *selective search*.

3.2.6 Faster R CNN

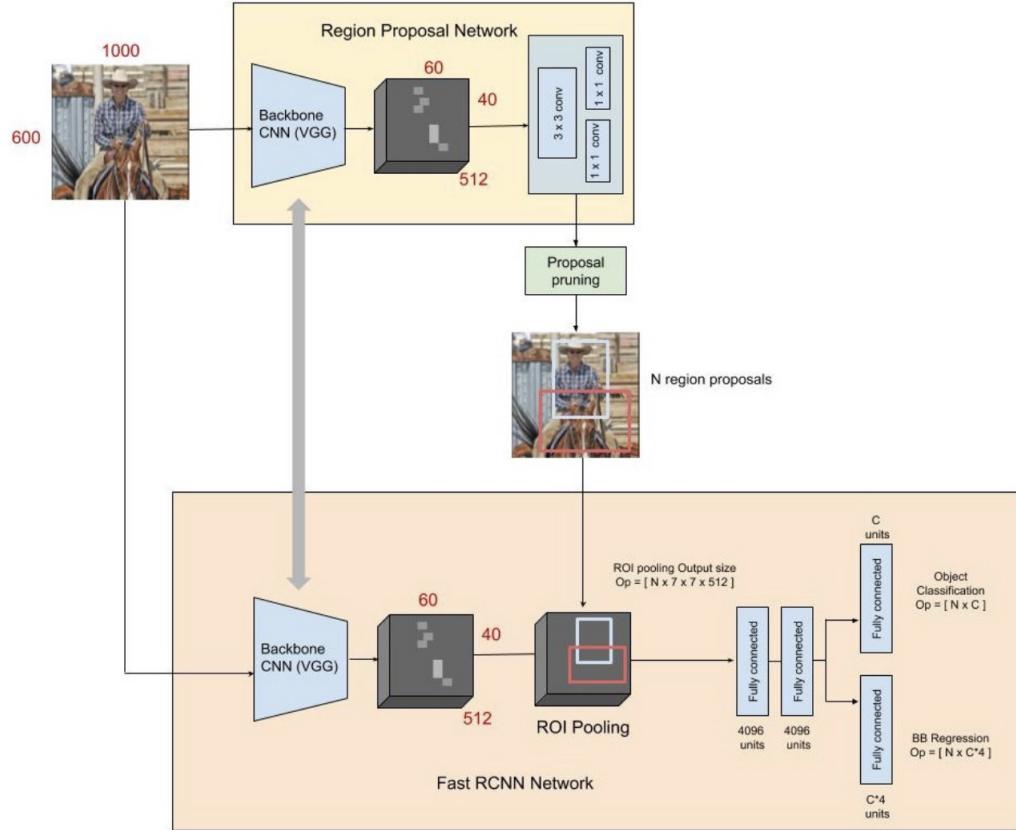


Figure 24: Structure Faster R-CNN [27]

Le modèle *Faster R-CNN* remplace l'étape de selective search du modèle Fast R-CNN par une étape de RPN, *Region Proposal Network*.

L'idée est d'utiliser l'algorithme RPN pour proposer des régions et de l'autre le Fast R-CNN pour détecter des objets dans ces régions.

Dans un premier temps nous présenterons le fonctionnement du RPN, puis nous expliquerons le fonctionnement des deux algorithmes au complet.

Explication RPN

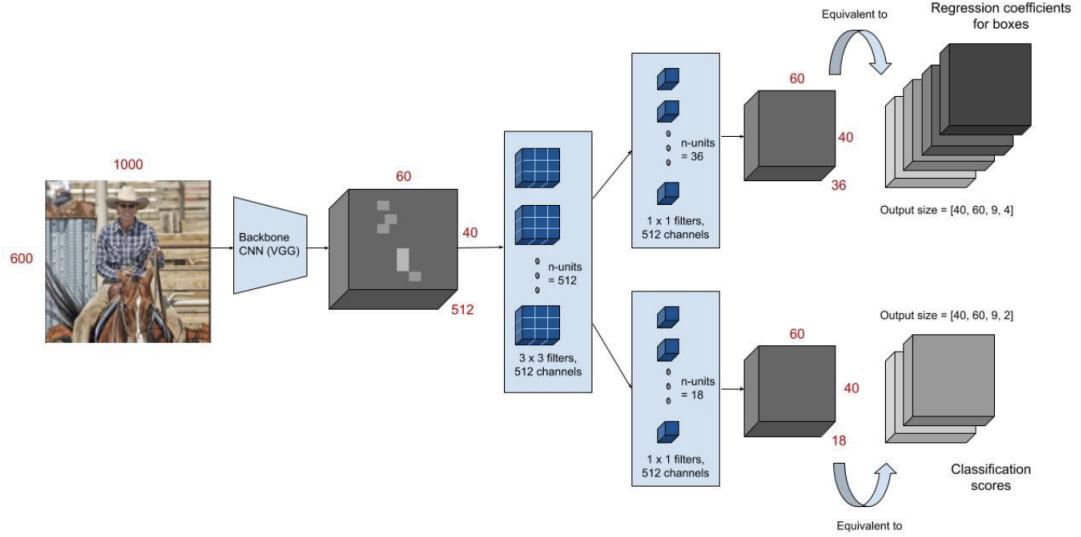


Figure 25: Structure RPN

- L'input du RPN est une image redimensionnée de taille 600x1000 pixels, qui est entrée dans un CNN. Les cartes de caractéristiques qui ressortent du CNN sont en général plus petite que la taille en input à cause des différentes couches de convolution présentes dans le CNN.
- Sur chaque cellule des cartes de caractéristiques, l'algorithme place des "anchors box". Ces "Anchors box" ont en général 3 tailles différentes et 3 ratios différents (donc 9 anchors box).

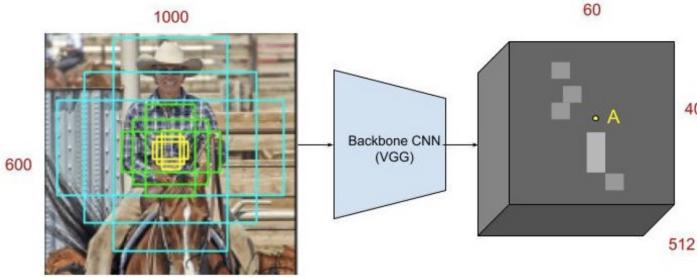


Figure 26: Ici, pour la cellule A, nous avons 9 anchors box

Une fois les "Anchors box" créées, l'algorithme doit vérifier si elles contiennent des objets ou non.

- Tout d'abord, l'algorithme applique une couche de convolution contenant 512 blocs de kernel de taille 3x3 sur les 512 canaux. Puis, il divise en deux le processus. D'un côté, il y a une couche de convolution contenant 36 blocs kernel et de l'autre 18 blocs kernel, chaque bloc contient des kernels de taille 1x1.
- L'output de la couche du haut (36 blocs de kernel) représente 36 cartes de caractéristiques de taille (H,L,36). Cette sortie est ensuite utilisée dans une régression pour donner les 4 coefficients prédicts des "Anchors box" présentes dans le CNN de base. Ces coefficients sont utilisés pour améliorer la précision des anchors box contenant des objets.
- L'output de la couche du bas (18 blocs de kernel) représente 18 cartes de caractéristiques de taille (H,L,18). Ces output sont ensuite utilisés pour prédire si un objet se trouve dans les "Anchors box" ou non.

Les cartes de caractéristiques de sortie se composent de 40x60 localisations possibles. Comme nous avions 9 anchors box par emplacement, alors cela nous fait un total de 40x60x9 soit environ 20 000 anchors box possible dans l'image.

Chaque anchors box est ensuite classé en classe positive/négative (comme dans le R-CNN). Une anchor box est positive si elle satisfait une des deux conditions suivantes : (voir figure 15)

- IoU de l'ancrage prédict est supérieure ou égale à 0.7 par rapport toutes les bounding box initiales.
- Une anchor box est negative si l'IoU est inférieur à 0.3. Les ancrues ni positives, ni negatives sont retirées.

La suite du processus consiste à sélectionner 128 anchors box positives et 128 négatives. Cela permet de réduire le biais d'échantillonnage de toutes les anchors box.

La fonction de coût de chaque lot d'anchors box est donnée par la formule suivante :

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*).$$

Figure 27: Fonction de coût RPN pour chaque lot d'anchors box

L'index i représente l'index de l'anchor box du lot. La partie classification du RPN représente le 1er terme de la formule.

- L_{cls} : représente le log de la perte entre les deux classes (objet ou non objet)
- p_i : représente le score de l'output de la classification du lot
- p_i^* : représente le label de l'anchor box
- L_{reg} : représente la fonction de régression, elle est activée seulement si l'anchor box contient un objet, c'est-à-dire si $p_i^* = 1$. Si c'est le cas, la régression se réalise et calcule t_i^* :

- $t_x^* = (x^* - x_a)/w_a$
- $t_y^* = (y^* - y_a)/h_a$
- $t_w^* = \log(w^*/w_a)$
- $t_h^* = \log(h^*/h_a)$
- t_x et t_y représentent les coordonnées du centre de la box , t_h la hauteur et t_w la largeur.
- x^* : correspond à l'abscisse du centre de la bounding box initiale
- x : correspond à l'abscisse du centre de l'anchor box

Les coefficients de la régression sont ensuite appliqués aux anchors box pour améliorer la précision. Cela nous renvoie des "nouvelles" bounding boxes avec une meilleure précision.

Pour chaque nouvelle bounding box, une suppression non maximale est appliquée : On classe les bounding box en fonction de leur score de classification de manière décroissante puis pour chaque bounding box qui se chevauche, on garde celle avec l'IoU le plus élevé et on supprime les autres.

A la fin de cette opération, nous arrivons à environ 2000 propositions de régions.

Si une bounding box dépasse de l'image, alors dans ce cas, on découpe la partie hors de l'image.

Fast R-CNN + RPN

Tout comme le RPN, le Fast R-CNN réalise lui aussi un traitement d'images afin de sélectionner les meilleures bounding boxes. L'image est d'abord passé dans un CNN pour avoir comme le RPN 512 cartes de caractéristiques de taille 60x40.

Le lien entre les 2 algorithmes est de partager les poids trouvés dans le RPN et dans le Fast R-CNN. Comme dans le Fast RCNN, une couche de ROI pooling est appliquée pour mettre en commun les cartes de caractéristiques et les régions proposées par le PRN.

L'output est de taille ($N, 7, 7, 512$) comme dans le Fast R-CNN. L'output est ensuite envoyé aux 2 couches de réseaux de neurones intégralement connectés puis est divisé entre la couche de classification et la couche de régression.

4 Métriques d'évaluation

4.1 Indice de Jaccard (Intersection over Union IoU)

4.1.1 Description de l'IoU pour la détection visuelle

L'Intersection over Union est une métrique d'évaluation utilisée pour évaluer la précision d'une prédiction faite par un modèle de détection d'objet. Ainsi, cette métrique porte bien sur une prédiction particulière faite par le modèle, et non le modèle en général.

Cette métrique d'évaluation est la plus utilisée dans les concours de détection d'objet comme le populaire PASCAL VOC challenge. Cette métrique est notamment la plus utilisée pour évaluer les détecteurs CNN (Convolutional Neural Network), comme ceux que vous avez vu précédemment (R-CNN et YOLO).

Cependant, l'IoU est une métrique qui peut s'appliquer à n'importe quel algorithme qui renvoie des bounding box prédictes comme output.

Afin d'utiliser l'IoU pour évaluer une prédiction faite par un détecteur d'objet, nous avons besoin :

- De la bounding box réelle, c'est-à-dire une zone de l'image labélisée à la main dans le dataset d'entraînement qui spécifie où se situe réellement l'objet à détecter sur l'image.
- De la bounding box prédictée par notre modèle.

Dès lors que nous disposons de ces deux éléments, nous pouvons appliquer une Intersection Over Union. Illustrons cela avec un exemple visuel : prenons par exemple, un document à scanner à la société générale, dans lequel nous cherchons à détecter les tableaux.



Figure 28: Le document qui fera office d'exemple pour illustrer le concept d'IoU

Nous créons alors la bounding box réelle à la main, qui correspond à la position réelle du tableau dans le document, puis nous regardons la prédiction de

notre modèle sur ce document

Nous obtenons: Il suffit alors, pour calculer l'IoU, de calculer le ratio de l'intersection



Figure 29: Un exemple de détection de tableau dans une image. La bounding box prédictée est affichée en vert tandis que la bounding box réelle est affichée en rouge. Notre objectif est alors de calculer l'IoU entre ces deux bounding boxes

entre nos deux bounding boxes et de l'union de nos deux bounding boxes:

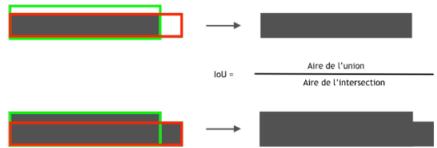


Figure 30: Calculer l'IoU réside dans un simple calcul de quotient entre deux aires, obtenues grâce à nos deux bounding boxes.

L'IoU est donc un simple ratio:

- Pour le numérateur on calcule l'aire commune à la boundig box prédicté et à la bounding box réelle.
- Pour le dénominateur, on utilise l'aire de l'union entre ces deux bounding boxes.
- La division nous donne alors notre score final, l'Intersection over Union.

Il faut donc bien noter que l'IoU contiendra toujours une partie empirique et subjective: la bounding box réelle est toujours déterminée "à la main", soit par un humain, soit par un algorithme de prétraitement des données.

On en déduit aussi qu'un data-set de computer vision doit non seulement contenir des images mais aussi les bounding boxes réelles associées à chaque image, afin que le modèle sache quoi chercher dans une image et pour pouvoir évaluer

ses performances.

Généralement, une IoU supérieur ou égale à 0.5 est considérée comme étant bonne, c'est-à-dire que la prédiction du modèle est bonne, celui-ci a bien détecté l'élément ciblé dans l'image.

4.1.2 Pourquoi utiliser cette métrique ?

Cette métrique est utilisée car le problème d'OCR est un problème bien plus complexe qu'une traditionnelle classification. Ici, notre output (bounding box prédite) est une zone, c'est-à-dire un objet en deux dimensions, ce qui implique des métriques d'évaluation très différentes de ce qui se fait habituellement.

En effet, dans les problèmes de classification, le système de métrique d'évaluation est presque toujours binaire : une prédiction est considérée comme exacte ou inexacte. Nous obtenons alors des métriques d'évaluation propres à un modèle (et à un test set) comme la précision et le recall. Cependant, dans un problème de computer vision, difficile d'établir un système binaire : les prédictions faites par notre modèle ne correspondront jamais exactement à la réalité. Ainsi, pour pouvoir évaluer une prédiction, nous devons avoir recours à un système de score, qui se rapproche de 1 quand la prédiction se rapproche de son objectif, et de 0 quand elle s'en éloigne.

Dans ce cadre, l'Intersection over Union est alors le client parfait pour fournir une bonne métrique d'évaluation : celle-ci répond à tout le cahier des charges pour être une métrique efficace d'évaluation. Comme c'est un ratio de deux valeurs, avec un dénominateur toujours plus grand que le numérateur (une intersection est toujours plus petite qu'une union correspondante), c'est une valeur comprise entre 0 et 1. Si la prédiction du modèle n'a aucun pixel en commun avec la vraie bouding box, la prédiction obtiendra une IoU de 0. Si notre prédiction contient tous les pixels recherchés, mais qu'elle est trop large, alors le score ne sera pas de 1 : l'aire de l'union serait d'autant plus grande que la prédiction est vaste, ce qui réduirait le score d'IoU.

L'IoU semble donc être un excellent compromis afin de récompenser la précision d'une prédiction : elle punit à la fois les erreurs de détection (car alors l'intersection devient petite, et donc l'IoU petite) et à la fois les prédictions trop vastes (car alors l'union devient grande, et donc l'IoU petite). Nous pouvons illustrer l'IoU à travers l'image suivante, qui permet de bien se rendre compte que l'IoU fournit des résultats très instinctifs, pour un problème qui ne l'est pas tant que ça.

Comme on peut le remarquer, les prédictions qui se rapprochent beaucoup des bounding boxes réelles obtiennent de meilleurs scores que celles qui sont trop vastes. Cela fait de l'IoU une excellente métrique d'évaluation pour les détecteurs d'objets.

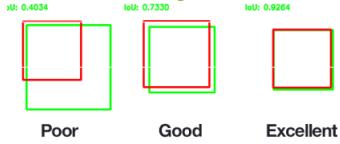


Figure 31: Un exemple de calcul d'IoU pour différentes bounding boxes.

Dans la pratique, un algorithme ne pourra jamais détecter à la perfection une bounding box réelle, au pixel près. Ainsi, l'IoU est une métrique qui permet de quantifier la qualité de la prédiction : l'IoU permet de prendre en compte le fait qu'une bounding box est un objet à coordonnées discrètes.

4.2 Moyenne de la précision moyenne : mAP

4.3 Introduction

La mean Average Precision est une métrique d'évaluation utilisée dans des problèmes de détection visuelle pour évaluer les performances d'un modèle. La mAP est donc une métrique de nature différente de l'IoU : cette dernière portait sur une prédiction spécifique d'un modèle, tandis que la mAP évalue le modèle à un niveau macroscopique, sur ses performances en général. L'IoU permet d'évaluer une prédiction, la MaP se base sur l'IoU pour évaluer un modèle. Wikipedia nous indique que la formule de calcul de la mAP est la suivante:

$$\text{MAP} = \frac{\sum_{q=1}^Q \text{AveP}(q)}{Q} \quad (1)$$

Cette formule est le fruit d'une analyse et du mélange de différentes métriques classiques de Machine Learning, appliquées au problème particulier de détection visuelle.

Afin de comprendre le fondement et le fonctionnement de cette métrique, nous allons donc devoir passer par plusieurs étapes de réflexion.

4.3.1 Rappels sur le recall et la precision

Tout d'abord, il est nécessaire de maîtriser ces deux notions de base afin de pouvoir comprendre le fonctionnement de la mAP. Ces deux métriques sont des métriques classiques afin d'évaluer les performances de modèles de classification. Ces métriques sont des métriques associées à chaque classe, et peuvent donc se décliner en moyennes (classique ou pondérées), on parle alors de *avg precision* ou de *weighted recall*, par exemple.

De manière plus précise, on décrit la précision et le recall pour une classe k donnée:

- **precision:** elle est définie par $p = \frac{TP}{TP+FP}$ où TP = "true positives" = nombre d'éléments de la classe k qui ont effectivement bien été prédits dans

la classe k, et FP = "false positives" = nombre d'éléments qui n'étaient pas de la classe k mais qui ont à tort été prédits en classe k.

La précision est donc le rapport entre le nombre d'éléments correctement prédits comme classe k et le nombre total d'éléments prédict comme k. Elle correspond à la pertinence de l'information remontée, mais ne s'intéresse pas à la quantité d'information remontée : on aura peut-être raté certains exemples de la classe k, qui auront été prédit dans une autre classe, mais tout ce qui a été prédit en classe k est effectivement de la classe k, et donc pertinent.

- **recall:** il est défini par $r = \frac{TP}{TP+FN}$ où TP = "true positives" = nombre d'éléments de la classe k qui ont effectivement bien été prédit dans la classe k, et FN = "false negatives" = nombre d'éléments qui étaient vraiment de la classe k mais qui ont été prédits comme une autre classe.

Le rappel est donc le rapport entre le nombre d'éléments correctement prédits comme classe k et le nombre total d'éléments qui sont vraiment de la classe k et qui auraient dû être prédits en k. Cela correspond à la quantité d'information effectivement remontée : même si l'on fait beaucoup d'erreurs en prédisant en classe k des exemples qui n'appartiennent pas à cette classe, on n'aura raté aucun exemple de la classe k.

Maintenant que nous maîtrisons ces notions, nous allons voir comment nous pouvons décliner celles-ci afin d'obtenir des indicateurs plus précis sur les modèles de reconnaissance visuelle.

4.3.2 La courbe de Precision-Recall et l'Average Precision

Il est désormais temps d'introduire un concept central pour la compréhension de la mAP: la courbe de Precision-Recall. La création de cette courbe part d'un constat simple: la précision et le rappel d'un modèle sont fondamentalement liés. (Il ne s'agit pas forcément de modèles de détection visuelle ici, ceci est valable dans le cas général).

Plus précisément, il est clair que quand le rappel d'un modèle augmente, alors la précision de celui-ci diminue: plus le rappel va être grand, plus le modèle ne loupe pas les positifs, donc plus le modèle parvient à ne pas oublier de positifs, mais alors la précision baisse car le nombre de faux positifs augmente.

A l'inverse, quand le modèle parvient à bien à être précis dans l'identification des positifs, c'est-à-dire que ses prédictions de positifs sont vraies, alors le nombre de faux négatifs augmente car beaucoup de positifs sont classés négatifs afin d'avoir une excellente précision et ne jamais se tromper, ce qui réduit le rappel. Ainsi, la courbe de la précision est décroissante avec le rappel.

De très nombreux modèles (y compris ceux de détection d'objet) ne renvoient

pas simplement une bounding box prédite comme nous l'avons vu plus avec l'IoU: ils renvoient en réalité une bounding box associée à un niveau de confiance: plus celui-ci est proche de 1, plus notre modèle estime qu'il est sûr de sa prédiction. Nous pouvons alors fixer un seuil, ou threshold en anglais: si le niveau de confiance associée à une prédiction faite par notre modèle est en dessous de ce seuil-clé de confiance, alors on considère que le modèle ne fait pas de prédiction.

Ainsi, plus ce seuil est élevé, moins le modèle a tendance à se tromper quand il fait des prédictions, donc sa précision augmente, et plus ce seuil baisse, plus notre modèle parvient à faire identifier de bonnes prédictions et de n'en oublier aucune (quitte à se tromper) : le rappel augmente alors. On retrouve alors bien notre corrélation décroissante évoquée précédemment entre la précision et le rappel.

De très nombreux modèles (y compris ceux de détection d'objet) ne renvoient pas simplement une bounding box prédite comme nous l'avons vu plus avec l'IoU : ils renvoient en réalité une bounding box associée à un niveau de confiance : plus celui-ci est proche de 1, plus notre modèle estime qu'il est sûr de sa prédiction. Nous pouvons alors fixer un seuil, ou threshold en anglais : si le niveau de confiance associée à une prédiction faite par notre modèle est en dessous de ce seuil-clé de confiance, alors on considère que le modèle ne fait pas de prédiction.

Comme la précision et le rappel de notre modèle sont dépendants du seuil que l'on a fixé, la courbe de Precision-Recall consiste à représenter le couple précision-rappel en fonction de différentes valeurs de seuil. Nous avons alors une vue d'ensemble des performances de notre modèle, qui ne dépendent pas de la valeur de seuil que l'on a fixé.

En voici un exemple :

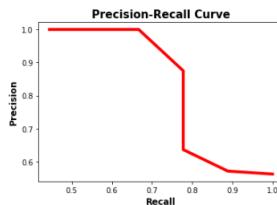


Figure 32: Un exemple de courbe Precision-Recall, où l'on remarque notamment la relation inverse entre précision et rappel.

Maintenant que nous avons une courbe, l'idéal serait alors de pouvoir obtenir une valeur, un seul chiffre qui représente cette courbe, afin d'obtenir une métrique sur notre modèle. (Attention, ici la précision et le rappel portent encore sur une seule classe d'objet associée au modèle. Quand on parle d'évaluer notre modèle, on raisonne encore classe par classe).

Le plus intuitif serait de tout simplement mesurer l'aire sous la courbe (AUC, area under the curve). Cependant, il arrive que cette métrique ne soit pas un excellent indicateur, car il arrive que la courbe contienne quelques portions où la précision monte lorsque le rappel augmente, en particulier pour les valeurs de rappel basses. Nous introduisons alors un nouvel indicateur, l'Average Precision, qui permettra de ne pas sanctionner les légères fluctuations de précision.

Ainsi, là où l'AUC permet de mesurer la précision que le classificateur offre avec un rappel de x%, l'AP est bien plus flexible car il mesure la précision que le classificateur peut offrir avec un rappel d'au moins x%: cela permet d'obtenir une décroissance complète entre précision et rappel, et ce, pour tout rappel. L'AP est alors une moyenne de la précision maximale que l'on peut obtenir avec un rappel d'au moins k%.

De manière plus rigoureuse, on obtient alors la formule suivante:

$$AP = \sum_{k=0}^{n-1} [\text{Recalls}(k) - \text{Recalls}(k + 1)] \times \text{Precisions}(k)$$

$$\text{Recalls}(n) = 0, \text{Precisions}(n) = 1$$

$n = \text{Number of Thresholds}$

Très exactement, l'Average Precision est alors la moyenne des précisions mesurées à chaque seuil, pondérées par l'augmentation de rappel entre ce seuil et le précédent. Ici, n, le nombre de seuils analysés pour effectuer la courbe est arbitraire. n prend des valeurs discrètes et doit être assez grand pour pouvoir renvoyer une courbe, car il correspond aux nombres de points de rupture de celle-ci. Le choix de n n'impacte pas beaucoup les comparaisons entre les modèles, tant que celui-ci est assez grand. Généralement, on prend entre 10 et 20 valeurs de seuils, uniformément réparties sur [0,1].

Pour conclure, l'intérêt de l'AP est donc qu'elle permet de capter l'entièreté de la forme de la courbe (contrairement au f1-score par exemple), tout en écartant les cas particuliers où la précision croît avec le rappel, ce qui rend cette métrique plus précise que l'AUC.

4.3.3 Du score de prédiction à la classification: recréer une métrique binaire à partir de l'IoU

Désormais, maintenant que nous avons bien introduit les concepts de courbe précision/rappel et d'AP, nous allons voir comment nous pouvons les appliquer aux problèmes de détection visuelle: souvenons-nous de notre analyse faite sur l'IoU. Nous avions observé que le fondement de cette métrique venait de l'impossibilité d'appliquer les métriques habituelles de classification que sont le rappel et la précision.

En réalité, il est possible d'utiliser l'IoU pour retrouver une précision et un rappel créés artificiellement, ce qui nous permettra alors d'appliquer les concepts de courbe précision/rappel et d'AP vus précédemment. Cette partie va détailler comment procéder.

Pour retrouver nos métriques classiques de classification, il faut tout simplement transposer le problème de détection visuelle à un problème de classification. Pour cela, nous allons encore utiliser un seuil, appelé seuil d'IoU, ou IoU threshold. Nous avons vu dans la description de l'IoU que celle-ci était une métrique qui variait de manière discrète entre 0 et 1. Ainsi, intuitivement, pour transformer ces données en 0 et 1, il suffit de fixer un seuil qui déterminera si la prédiction est bonne ou mauvaise, ce à quoi correspond le threshold.

Il faut voir le threshold comme la valeur minimale d'IoU pour laquelle on estime notre prédiction bonne : nous avons vu précédemment que cette valeur était souvent fixée à 0.5. Ainsi, quand une prédiction obtient une IoU supérieure à 0.5, on estime que notre prédiction est bonne, donc nous avons bien repéré l'objet à détecter, nous avons donc un Vrai Positif. A l'inverse, quand notre IoU est inférieur ou égale à 0.5, alors le modèle a échoué à détecter l'image avec assez de précision, c'est comme s'il avait estimé que l'objet en question n'était pas présent sur l'image, nous avons donc un Faux Négatif. Nous supposons en effet que nous n'aurons jamais de Vrais Négatifs: dans chaque image de notre data-set, l'objet que nous cherchons à détecter sera présent.

$$class(IoU) = \begin{cases} \text{Positive} & \rightarrow \text{IoU} \geq \text{Threshold} \\ \text{Negative} & \rightarrow \text{IoU} < \text{Threshold} \end{cases} \quad (2)$$

Cela pourrait cependant avoir du sens d'inclure des features de classe 0, c'est-à-dire des images qui ne contiennent pas l'objet que l'on cherche à détecter visuellement: dans notre exemple d'application concrète, à la Société Générale, nous devrons peut-être traiter des documents qui ne contiennent pas de tableau. Notre modèle, s'il n'est entraîné que sur des images qui contiennent de stableaux, comme ça sera le cas avec le dataset avec lequel nous travaillerons, ne saura que trouver où se situe le tableau à détecter, et sera très limité pour déterminer s'il y en a un sur l'image ou non. Ainsi, sur les données d'entraînement, le cas des Vrais Négatifs n'existera pas car le data-set de test ne contient que des objets de classe 1 (des images qui contiennent l'objet à détecter visuellement), ce qui

pourra poser problème pour l'utilisation concrète du modèle. Une solution serait d'enrichir le dataset avec des images de classe 0, et bien spécifier au modèle qu'il effectue une classification, en plus d'une détection visuelle.

De plus, l'IoU ne serait tout simplement pas pertinente pour des objets de classe 0 : sans bounding box réelle, nous obtenons forcément un IoU de 0 (car l'intersection serait nulle), donc notre IoU est forcément inférieure au threshold et donc tous les features de classe 0 seront considérées comme Vrais Négatifs, donc la métrique n'a ici pas beaucoup de sens. Ainsi, pour rester en accord avec l'utilisation de l'IoU, les data sets d'entraînement de modèles ne contiennent en général pas d'images de classe 0.

Enfin, les Faux Positifs sont eux aussi un cas particulier: ils apparaissent quand problème de détection visuelle est un problème multi-classes, c'est-à-dire que le modèle doit détecter des objets de classes différentes, et donner leur classe. Cela ne sera pas utile non plus dans notre cas, étant donné que notre modèle devra se concentrer sur la détection de tableaux uniquement. Si jamais celui-ci devait détecter les tableaux et les photos d'identité par exemple, un faux positif correspondrait au cas où notre modèle aurait très bien déterminé la zone d'un tableau ou d'une photo d'identité ($\text{IoU} > \text{threshold}$), mais aurait associé la mauvaise classe à sa prédiction. Les Faux Positifs sont donc un cas relativement rare.

Finalement, à partir de l'IoU des prédictions et d'un threshold donnés, nous pouvons recréer des métriques d'évaluation semblables à la précision et au rappel des modèles de classifications traditionnels.

NB: Nous pouvons par ailleurs prédire certains effets propres aux précisions et aux rappels de modèles d'OCR : nous obtiendrons notamment toujours une précision de 1 pour les objets de classe 1 dans le cas où nous ne détectons qu'un seul type d'objet. Ainsi, utiliser l'IoU pour se ramener à un problème de classification n'a de sens que dans des problèmes de détection visuelle multi-classes.

4.3.4 La mAP

Maintenant que nous avons réussi à recréer un rappel une précision à partir de l'IoU, nous pouvons retrouver notre Average Precision détaillée précédemment. Cependant, l'obtention de cette nouvelle Average Precision peut se faire de différentes manières, en fonction de nos modèles de détection visuelle :

- Soit notre modèle n'associe pas de niveau de confiance à ses bounding boxes prédites et dans ce cas, on utilise l'IoU threshold comme le seuil à faire varier pour obtenir les différents couples (précision, rappel).
- Soit notre modèle peut associer un niveau de confiance à ses prédictions, et dans ce cas on distingue bien le seuil utilisé pour obtenir la courbe de précision/rappel et le seuil d'IoU pour faire la classification. Alors, pour

obtenir l'AP, on calcule les AP obtenues pour des IoU thresholds variant entre 0 et 1, puis on fait la moyenne des AP ainsi obtenues. L'AP devient alors déjà une moyenne de moyenne, mais il ne s'agit pas à proprement parler de la mAP.

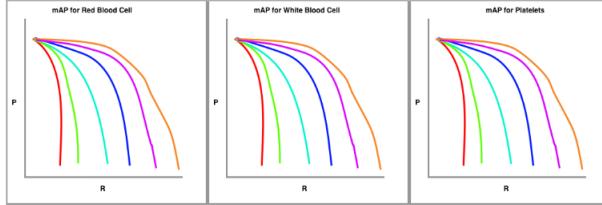


Figure 33: Exemple de différentes courbes précision/rappel obtenues pour différentes valeurs d'IoU threshold, pour différentes classes, pour un modèle de détection visuelle de cellules sanguines.

Nous pouvons alors obtenir la mAP: il suffit de faire la moyenne des AP en fonction de chaque classe associée au modèle (par exemple, l'AP des tableaux + l'AP des photos d'identité, le tout divisé par 2). En effet, n'oublions pas que la précision et le rappel sont propres à des classes, et donc l'AP aussi. Ainsi, la mAP permet d'obtenir une valeur unique pour évaluer notre modèle sur toutes les classes.

Cet indicateur, bien que complexe ("deux moyennes", cela peut sembler redondant), répond bien au cahier des charges d'une bonne métrique d'évaluation, et c'est pourquoi elle est aujourd'hui la métrique d'évaluation la plus utilisée dans les problèmes de détection visuelle. Celle-ci a notamment l'avantage de très peu dépendre de la fixation du n dans son calcul, ce qui permet in fine de bien comparer différents modèles sans que le paramétrage de tous ces seuils soit trop sensible.

Métriques	
<i>Intersection over Union</i>	Va de 0 à 1, permet d'évaluer la qualité d'une prédiction.
<i>Precision/recall curve</i>	Courbe se basant sur l'IoU de différentes prédictions afin de pouvoir visualiser la relation entre la précision et le rappel, en fonction de différents seuils fixés.
<i>Average Precision</i>	Moyenne pondérée sur les valeur de la courbe de précision/rappel, permettant d'obtenir une valeur unique pour évaluer la qualité des prédictions d'un modèle sur une classe d'objet. Dépend du nombre n de seuils fixés.
<i>Mean Average Precision</i>	Moyenne des Average Precision du modèle sur toutes les classes. Permet d'évaluer un modèle dans son entiereté.

4.4 Résumé des métriques et de la construction de la MAP

5 Transfert Learning

Le transfert learning consiste à utiliser un modèle déjà entraîné (souvent très profond, beaucoup de couche) afin de résoudre différents problèmes :

- Manque de données d'entraînement
- Entraînement trop lent
- Sur-ajustement

5.1 Explication TL

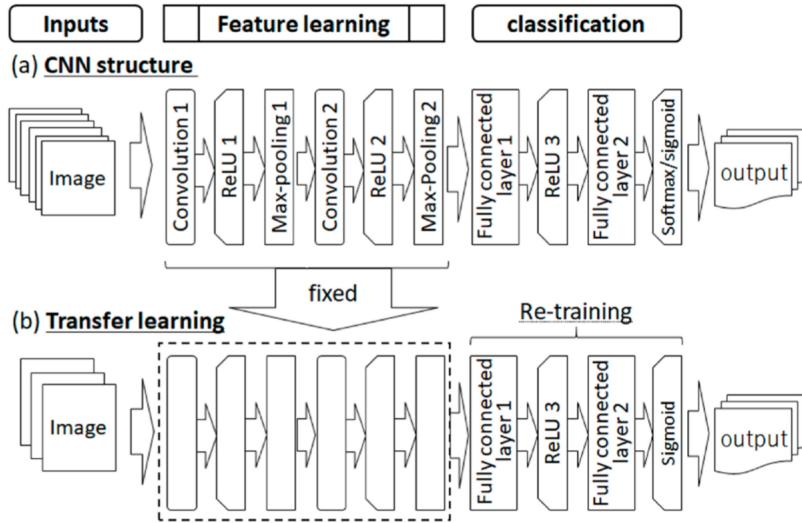


Figure 34: Architecture Transfert Learning d'un CNN

Dans notre cas, l'utilisation du transfert learning nous permettra de réutiliser une partie d'une architecture d'un modèle de détection d'objets déjà entraîné sur nos images. Les poids des réseaux de neurones seront donc déjà calculés grâce à l'architecture utilisée.

En général, nous réutilisons la couche d'entrée ainsi que les premières couches de convolutions du modèle existante.

Avant d'utiliser cette méthode, il est important de redimensionner les images de notre DataSet en fonction de la taille des images attendus du modèle transféré.

Nous commençons par prendre la couche d'entrée et les premières couches de convolutions du modèle entraînés en gardant les poids (filtres). Nous entraînons et analysons les performances du modèle complet appliqué sur nos données.

Ensuite, nous essayons de retirer peu à peu les couches supérieures pour que la rétropropagation les ajuste à nos données. Nous comparons les performances du premier entraînement avec ce dernier. Si la précision du modèle augmente, cela signifie que les couches supérieurs sont bien entraînés et que nous pouvons continuer à enlever des couches du modèles existants. En cas de mauvaise performance, il est préférable de remettre les couches du modèles.

6 Base de données

6.1 Informations sur la base de données

Les données utilisées pour ce projet proviennent d'une base de données open source.

Cette base de données est disponible sur ce lien : <https://doc-analysis.github.io/tablebank-page/index.html>

Cette base de données est constitué de 2 principaux fichiers, *Detection* et *Recognition*. Chacun de ces fichiers contient 2 sous-fichiers, *annotations* et *images*.

Dans le fichier *images* se trouve toutes les images utilisées pour le training set.

Dans le fichier *annotations* se trouve 6 fichiers au format Json :

- tablebank latex test
- tablebank latex train
- tablebank latex val
- tablebank word test
- tablebank word train
- tablebank word val

Ces fichiers contiennent les informations liées à l'image, c'est-à-dire, son nom, les coordonnées des bbox, le nombre de BBOX par image, ...

Pour pouvoir manipuler les images avec leurs étiquettes, nous avons décidé de rassembler les 6 fichiers dans une liste, sous le nom de *List Dict*. Chacun de ces 6 fichiers contient un dictionnaire de 5 clés (info, licenses, images, catégories et annotations).

Dans les clés *info* et *licenses*, se trouve les informations concernant la création de la base de données. Ces clés ne sont pas utilisé dans notre projet. De même que la clé *Categories*.

Dans la clé *images* se trouve les étiquettes des images, c'est-à-dire leurs noms, leur numéro, leur hauteur et leur largeur.

Dans la clé *annotations* se trouve les coordonnées des BBOX et l'identifiant de l'image associé.

Chacune de ces clés a pour valeur soit une liste contenant 1 élément, soit une liste contenant plusieurs élément. Chaque élément de ces listes correspond à un sous-dictionnaire.

Remarque : Chaque dictionnaire correspond à 1 seule BBOX.

Pour des raisons de compréhension, nous avons décidé de représenter la base données sous forme de tableau.

	Clé	Valeur
Elément i	Clé	Sous-Dictionnaire
"info"	"contributor"	'MSRA NLC Group'
	"description"	"TableBank Dataset"
	"version"	"1.0"
	"url"	""
	"year"	2019
	'date_created'	2019/02/28
"licenses"	Clé	Valeur unique
(Liste de 1 élément)	"id"	1
	"url"	https://creativecommons.org/licenses/by-nd/4.0/
	"name"	"Attribution-NonCommercial-NoDerivs License"
<u>Fichier Annotation</u> (Chaque élément est un dictionnaire) :	Clé	Sous-Dictionnaire
	"images"	"file_name"
	(Liste)	"image_id"
		"license"
		"width"
		"height"
		"width" largeur de l'image "height" hauteur de l'image
"Categories"	Clé	Valeur unique
(Liste de 1 élément)	"id"	1
	"supercategory"	"table"
	"name"	"table"
"annotations"	Clé	Sous-Dictionnaire
(Liste)	"segmentation"	Double Liste avec l'élément dans la première liste : [[1,2,3]]
	"area"	superficie image
	"image_id"	numéro de l'image
	"category_id"	1
	"id"	numéro
	"iscrowd"	0
	"bbox"	Liste simple : [1,2,3]

7 Démarches algorithmiques

7.1 Preprocessing

Dans la partie preprocessing, nous avons crée des fonctions permettant de récupérer les données nécessaire à la création du modèle Faster R-CNN. Les données récupérés sont le noms, l'id et les BBOX associées de chaque images. (Voir zones jaunes dans la Base de données ci-dessus)
Nous avons regroupé les informations dans un dictionnaire.

	clé (id de l'image)	Valeur	Cle 2	valeur 2
Dictionnaire_Complet	2334	Sous-Dictionnaire	image	image en array
			bboxes	[[48, 110, 559, 508]]
	4434	Sous-Dictionnaire	im_image	image en array
			bboxes	[[117, 447, 506, 555], [118, 204, 513, 318]]

Figure 35: Dictionnaire créé

La suite a constitué à créer une *class* nécessaire à l'utilisation d'un modèle de détection d'objets. Dans cette classe se trouve des *méthodes* permettant de :

- charger l'image
- transformer les annotations en *tensor*
- définir la target comme étant la bbox initiale à prédire au mieux (Cela permettra de calculer l'erreur de prédiction)

Comme en Machine Learning classique, nous avons ensuite réparti les données de manière à avoir un training set, une partie validation et une partie Test set.

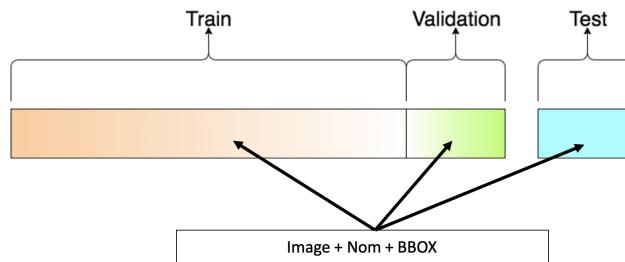


Figure 36: Répartition du DataSet

Travailler avec une quantité de données assez élevé, nous a contraint de travailler avec la classe DataLoader. Cela permet de paralléliser le processus de chargement des images.

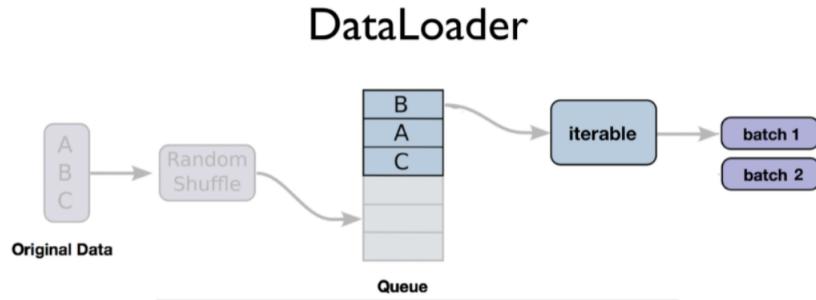


Figure 37: Architecture Data loader [28]

Chaque échantillon (batch size) se compose de N images, et chaque itération de l'algorithme sur ces échantillons est appelé *epoch*

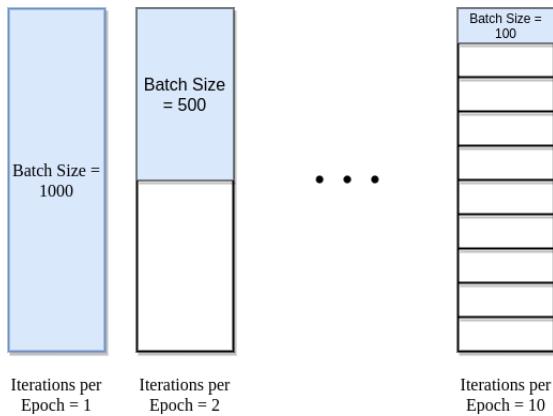


Figure 38: Epoch Explication [28]

References

- [1] Détails de la fonciton d'activation ReLu. *Machine Learning Mastery*. <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>.
 - [2] Figure 1: Architecture des réeseaux de neurones convolutifs simple. *Real Time Object Detection using Deep-Learning and OpenCV*. <https://www.irjet.net/archives/V7/i4/IRJET-V7I4621.pdf>.
 - [3] Figure 2: Produit de convolution et produit scalaire. *Benjamin Perret*. <https://perso.esiee.fr/~perretb/I5FM/TAI/convolution/index.html>.
 - [4] Figure 3: Produit de convolution sur une image de couleur. <https://www.aspexit.com/reseau-de-neurones-on-va-essayer-de-demystifier-un-peu-tout-ca-3/>.
 - [5] Figure 4: Produit de convolution de cartes de caractéristiques. *Paul Bretton*. <https://blog.betomorrow.com/les-r\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{e\global\mathchardef\accent@spacefactor\spacefactor}\let\begingroup\def{}\endgroup\relax\let\ignorespaces\relax\accent19e\egroup\spacefactor\accent@spacefactorspaceaux-de-neurones-de-convolutions-pour-les-n\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{e\global\mathchardef\accent@spacefactor\spacefactor}\let\begingroup\def{}\endgroup\relax\let\ignorespaces\relax\accent19e\egroup\spacefactor\accent@spacefactorophytes-2b36a59cf648>
 - [6] Entropie croisée. *Jason Brownlee*. <https://machinelearningmastery.com/cross-entropy-for-machine-learning/>.
 - [7] Architecture : LeNet-5. *Vaibhav Khandelwal*. <https://pub.towardsai.net/the-architecture-implementation-of-lenet-5-eef03a68d1f7>.
 - [8] Architecture : AlexNet. *Jerry Wei*. <https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951>.
 - [9] Architecture : GoogLeNet. *googlenet*. <https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/>.
 - [10] Architecture : VGGNet ResNet Inception Xception. *Adrian Rosebrock*. <https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>.
 - [11] Architecture SeNet. *Hu et al.* <https://paperswithcode.com/method/senet>.
 - [12] Figure 5 et 6. *datahacker.rs*. <http://datahacker.rs/deep-learning-object-localization/>.
 - [13] Papier de recherche modèle YOLO. <https://arxiv.org/pdf/1506.02640.pdf>.

- [14] Paper de recherche modèle Faster RCNN. <https://arxiv.org/abs/1506.01497>.
- [15] Figure 8: Architecture CNN vs FCN. *Gang Fu*. <https://www.mdpi.com/2072-4292/9/5/498>.
- [16] Figure 9: Architecture YOLO. *ODSC - Open Data Science*. <https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0>.
- [17] Figure 10: Bounding box YOLO. *Vandit Jain*. <https://towardsdatascience.com/forget-the-hassles-of-anchor-boxes-with-fcos-fully-convolutional-one-stage-object-detection-fc0e25622e1c>.
- [18] Figure 11: Calcul des coordonnées d'un bounding boxes. *Mauricio Menegaz*. <https://hackernoon.com/understanding-yolo-f5a74bbc7967>.
- [19] Figure 12: type d'Anchor boxes. *Chengwei Zhang*. <https://heartbeat.fritz.ai/gentle-guide-on-how-yolo-object-localization-works-with-keras-part-2-65fe59ac12d>.
- [20] Figure 13: K-means Bounding box. *Virginia Ng, Daniel Hofmann*. http://conference.scipy.org/proceedings/scipy2018/pdfs/virginia_ng.pdf.
- [21] Figure 14: Architecture R-CNN. *Rohith Gandhi*. <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.
- [22] Algorithme selective search et figure 15. *J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders*. https://www.researchgate.net/publication/262270555_Selective_Search_for_Object_Recognition.
- [23] Figure 16: Struture AlexNet. *O'Reilly*. <https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch01.html>.
- [24] Figure 18: Classification des ROI. <https://www.cnblogs.com/gongxijun/p/7097630.html>.
- [25] . <https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles/overview/evaluation>.
- [26] Figure 22 et 23. *Shilpa Ananth*. <https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022>.
- [27] 25 et 26 Figure 24. <https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46>.
- [28] Dataloader. <https://www.journaldev.com/36576/pytorch-dataloader>.