Projet INF473V-Modal : Airbus Ship Detection Challenge

Yoane Camara et Malhou Mohamed

1 Introduction et description de l'approche adoptée

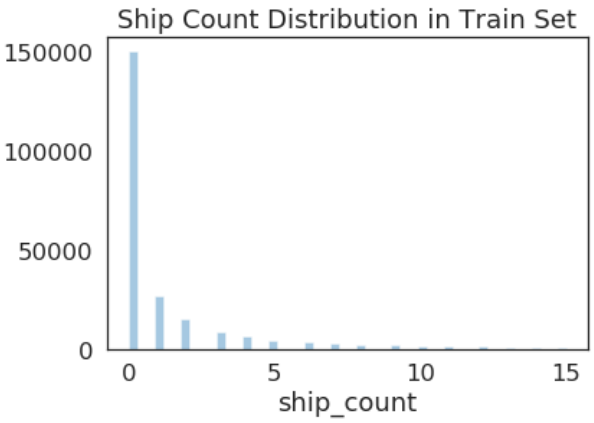
- Le projet consiste en la détection d'objets sous la forme d'une compétition postée par Airbus sur Kaggle. Nous sommes censés localiser les navires dans les images et placer un segment de boîte englobante aligné autour des navires que nous localisons. Nous abordons deux approches differentes. Dans la première approche nous implementons U-ResNet en nous inspirant du dernier TD sur la détection des chiffres, et nous l'appliquons sur le jeu de données d'Airbus. la deuxieme approche consiste à implementer le reseau R-CNN.

- Les réseaux convolutionnels sont généralement utilisés pour les tâches de classification où la sortie du réseau qui prend une image en entrée est une étiquette de classe unique. Mais U-ResNet (ou UNet) est destiné aux tâches de segmentation d'images. cela signifie qu'il produit une image de la même taille de l'entrée mais avec seulement 2 canaux sur lesquels nous appliquons soft-max pour obtenir des distributions de probabilité des deux classes: navires et arrière-plan. l'architecture se compose d'un chemin de contraction et d'un chemin expansif. le réseau expansif est un réseau convolutionnel typique qui consiste en l'application répétée de blocs ResNet. chaque bloc est une série de convolutions suivies d'une unité linéaire rectifiée (ReLU). Pendant la contraction, les informations spatiales sont réduites tandis que les informations de caractéristiques (features) sont augmentées. La voie expansive combine les caractéristiques et les informations spatiales à travers une séquence de convolutions ascendantes et de concaténations avec des caractéristiques de haute résolution provenant de la voie contractuelle.

2 Description Du Premier Modèle

2.1 pré-traitement de la base de données

Figure 1: Distribution des navires dans les images.



La base de données AirBus-Ship consiste en 200k images de taille $768 \times 768 \times 3$. Comme le montre la Figure 1, le jeu de données n'est pas calibré. en effet, la plupart des images ne contiennent pas de navire et les navires qui apparaissent dans les autres images sont souvent de petites dimensions. Rajoutons à cela que la taille des images entraine une grand nombre de paramètres dans le réseau ce qui nécessite des caractéristiques assez importantes pour la machine utilisée et limite aussi batch-size. Ainsi, il fallait trouver un compromis.
Nous avons décidé alors de mettre un classifieur en amont du modèle pour la phase des tests et ensuite proposer des régions de recherche d'objets pour le réseau UResNet.

Pour la période d'entrainement on a du calibrer le nombre d'images "vides" mais aussi recadrer les images et les avoir finalement de taille $224 \times 224 \times 3$. ainsi, les étiquettes sont de taille : $224 \times 224 \times 1$.

2.2 la classification d'images

- Il s'agit de classifier les images en deux catégories : celles qui contiennent des navires et les autres. pour cela nous implémentons l'architecture d'Alexnet avec comme entrée des images de taille $786 \times 786 \times 3$ et en sortie deux classes. Comme nous avons vu pendant les séances de TD, l'architecture d'AlexNet possède deux parties distinctes, un premier sous-réseau "features", chargé d'extraire des caractéristiques pertinentes de l'image, et un "classifier" que l'on applique par dessus. La partie "features" est réutilisable pour d'autres tâches. Nous allons importer un modèle pré-entraîné sur ImageNet pour l'architecture AlexNet. Nous utiliserons ensuite les "features" correspondants pour notre problème de classification et n'entrainer finalement que la partie "classifier".

- le but de la classification est d'éliminer les images vides, ainsi nous nous interessons pour cette partie à l'erreur de type FN (False Negative). En effet nous avons intérêt à ne pas éliminier excessivement des images étiquetées en 1 car cela rend le classifieur obsolète. Nous abordons cela dans la partie Résultats.

2.3 la segmentation d'image

- En ce qui concerne l'implementation de U-ResNet, nous adaptons la classe ResNet-Block implementée dans le TD6 pour l'utiliser comme block elémentaire pour UNet. l'architecture du réseau est illustrée dans les Figures 2 et 3.

- Nous appliquons premièrement une convolution de noyeau 3 suivie par une normalisation batchNorm et une unité linéaire rectifiée (ReLU). Ensuite, nous appliquons un premier block ResNet sans sous-échantillonner. Puis, on diminue la taille de l'image en doublant le nombre de canaux. Il faut noter que

Figure 2: L'architecture du réseau U-ResNet

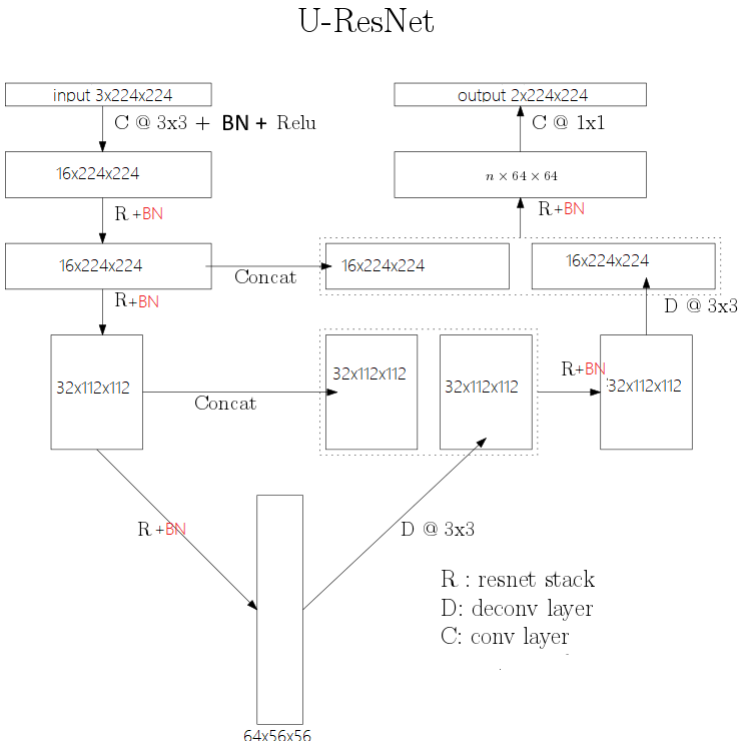
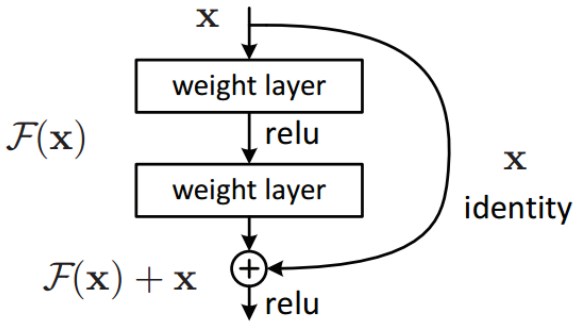


Figure 3: L'architecture d'un block de ResNet



dans la fonction Forward, il faut avoir conservé une copie de la sortie de chaque block ResNet pour les concaténer avec leurs paires équivalents du "upsampling". Après une autre couche ResBlock, nous nous retrouvons avec une sortie de taille $64 \times 56 \times 56$. pour revenir aux dimensions principales, nous appliquons une serie de convolutions transposées et toujours des block ResNet. Notons aussi que nous avons rajouté entre chaque block une couche batchNorm2d car nous avions des difficultés à entrainer le modèle avec des non-linéarités sui saturent et ainsi la perte ne diminuait pas. Nous avons essayé différentes valeurs pour le paramètre learning-rate avant de faire cette solution.

- La fonction de perte est calculée en appliquant CrossEntropyLoss. Ce critère combine LogSoftmax et NLLLoss et permet de définir des poids pour chacune des deux classes. Nous avons au début essayé de calibrer la perte en fonction du pourcentage moyen de chaque classe dans les images mais nous avons finalement opté pour des poids par défaut après quelques tentatives. Nous abordons cela dans la partie suivante.

3 Résultats de la première approche

- Commençons par exposer les résultats obtenus pendant la classification. On introduit la notion de matrice de confusion vue en INF442 pour analyser les performances du classifieur.
- Nous appliquons la classification sur 3000 images avec la moitié pour chaque classe. le classifieur a un taux de succès de 82.2%. Cependant, comme on peut le voir dans le tableau, le classifieur marche moins bien pour les signaux faux négatifs, c'est à dire qu'il rejette 354 images parmi les 1500 qui contiennent effectivement un navire, ce qui limite l'utilisation de ce classifieur. Notons aussi qu'on n'a pas exploité toute la dataset pour l'entrainement.

Confusion Matrix	Classe éstimée arriere plan	Classe éstimée navire
Classe réelle arriere plan	1320 vrais négatifs	180 faux positifs
Classe réelle navire	354 faux négatifs	1146 vrais positifs

- Passons à la segmentation d'images. Avant le pré-traitement mentionné avant, nous avons essayé au lieu des crops d'utiliser juste des resizes ce qui fait qu'on perde d'informations et par conséquent les features que le réseau ne sont pas assez claires et spécifiques aux navires. Ce qui nous donne les sorties dans la Figure 4.

- Finalement, après le pré-traitement, nous obtenons les résultats suivants dans la figure 5. Nous calculons la précision par la fonction IoU (Intersection over Union) qui est la métrique d'évaluation standard utilisée pour mesurer la précision d'un détecteur d'objet sur un ensemble de données particulier. La durée d'entrainement pour les 30 epoches est quelques dizaines d'heures.

Figure 4: Trained model on resized images

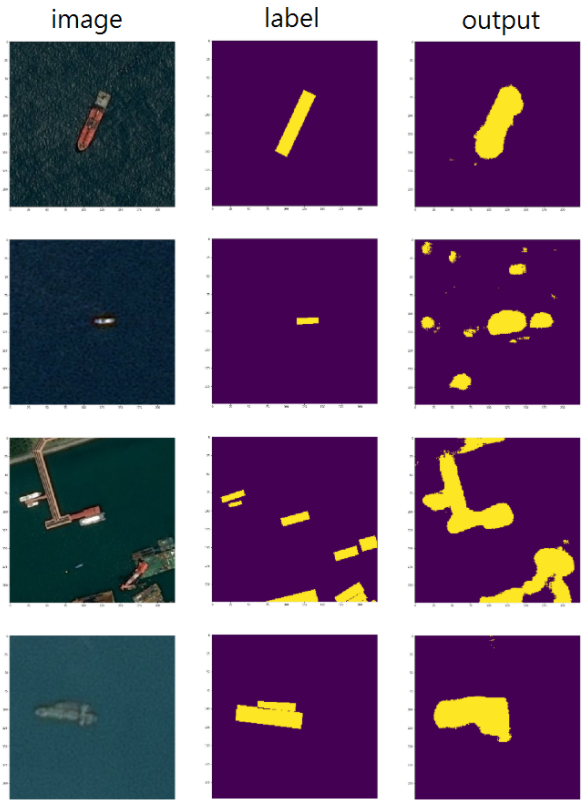
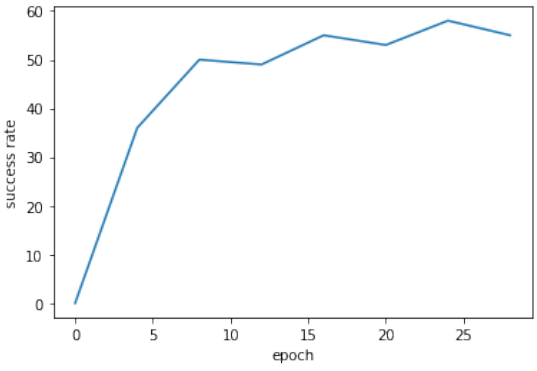
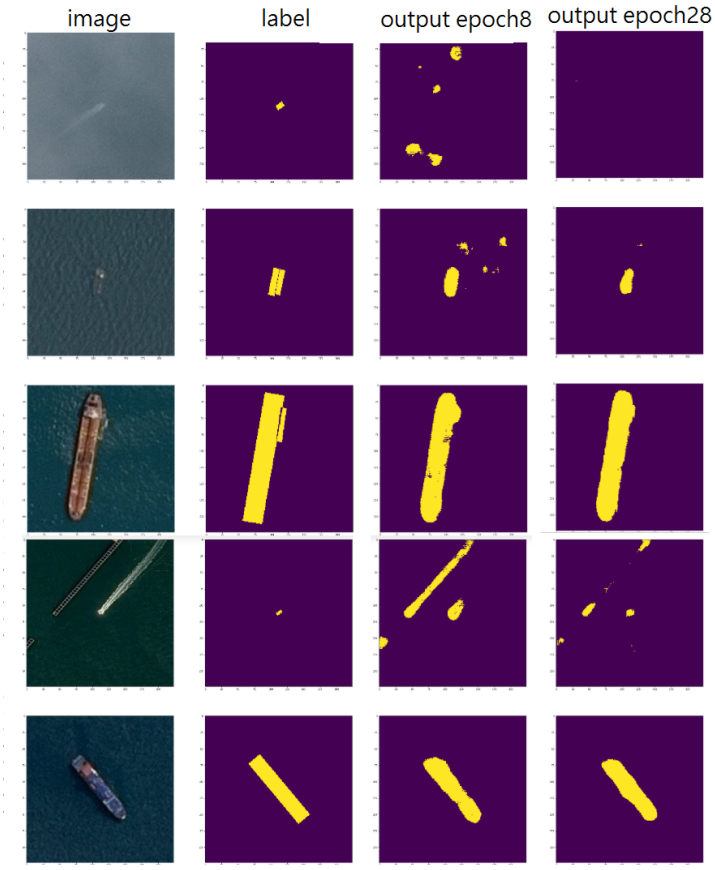


Figure 5: performances de UResNet au long de l'entrainement



Nous avons essayé de comparer les résultats de deux versions obtenues après 8 epochs vs 28 epochs:

Figure 6: Trained model on resized images

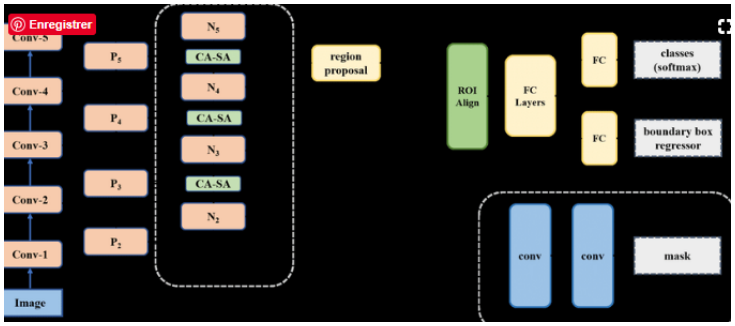


– On peut constater l'évolution de la capacité du réseau à distinguer entre les navires et les autres objets et constructions présentes dans l'image. Cependant on peut aussi voir que dans quelques images, notamment la première dans la Figure 6, il est impossible pour l'oeil humain de détecter le navire présent à cause de sa taille ou meme parfois les conditions climatiques comme la présence des nuages.

4 Modèle complémentaire

Nous avons dans la deuxième partie du projet essayé de développer un second modèle que nous n'avons pas pu terminer. Nous jugeons cependant, intéressant de vous présenter le descriptif de ce modèle ainsi que l'état d'avancement dans notre implémentation. Notre choix de second modèle s'est porté sur une version améliorée de l'architecture Mask R-CNN décrite dans l'article [1]. La raison principale est sa forte similarité l'architecture R-CNN, vue en TD. En effet, les réseaux Fast CNN, Faster CNN et Mask R-CNN découlent toutes de R-CNN et sont successivement des améliorations, chacune de son précédent. En revanche, à la différence de R-CNN, Mask R-CNN produit en plus des bounding box et labels pour chaque objet de l'image en entrée, un masque binaire, c'est à dire une image globalement nulle avec uniquement les pixels éléments d'un objet qui valent 1 : On parle alors de segmentation pixel par pixel. Aussi de manière générale, Mask R-CNN a une utilisation beaucoup plus intéressante des features extraits par convolution qui sont réutilisés pour générer grâce à un RPN des propositions de Regions of Interest(ROI) au lieu d'utiliser la selective search qui est beaucoup plus couteuse en temps.
Pour ce qui est de sa composition, le réseau commence par une partie convolutive (backbone) qui produit en sortie des Feature maps. Ceux-ci sont de dimensions et canaux différents et représentent des outputs à différents niveaux d'évolution dans la convolution. Ces différences traduisent le fait qu'au fur et à mesure qu'on avance dans l'extraction, d'un coté, on obtient des Features de plus en plus pertinentes en doublant à chaque fois le nombre de canaux mais de l'autre on perd de l'information spatiale, ce qui dans notre cas est essentielle vu que l'objectif final est de faire de la segmentation au niveau pixel. A ce niveau, 4 propose comme solution d'adjoindre à la Feature pyramid un chemin top-bottom avec des connexions latérales vers les features de même niveau dans la pyramide précédente, ce qui permet d'associer d'une part des éléments avec une faible résolution mais une forte sémantique et d'autre part des éléments dans le cas inverse. L'utilisation de Features pyramid pour la détection est plutôt répandue,

Figure 7: Architecture Mask R-CNN



mais l'innovation dans 4 consiste à rajouter un autre chemin bottom-top cette fois, ce qui crée un raccourci entre les low-maps et les high-maps de la pyramide et facilite la propagation. Par ailleurs, l'article suggère de remplacer les convolutions dans les connexions latérales et verticales par des Attention Mechanism. L'idée derrière cela, est d'affecter des coefficients différents aux canaux et aux positions dans la Feature map en fonction des degrés d'importance.

En tête de la Feature Pyramid Network, se trouve un Region Proposal Network(RPN) chargé de fournir des régions d'intérêts ainsi que des scores indiquant la probabilité de contenir un objet ou d'être un élément du background. Ce réseau indépendant se constitue d'une convolution de kernel 3 suivi de deux convolutions de kernel 1 parallèles, l'une pour la classification et l'autre pour la régression vers la bounding box de l'objet identifié. Notons à cet effet, que les convolutions représentent en réalité des couches FC avec sliding sur chaque Features maps de la pyramide. Le reste du réseau a exactement la même qu'un modèle Mask R-CNN de base. Une couche Non Maximum Suppression (NMS) permet de ne conserver que les prédictions des régions les plus

4.1 Implémentation

– Backbone: Pour la partie convolutive, on récupère celle du modèle de base de pytorch qui est un réseau ResNet50 pré-entrainé sur le dataset COCO. Nous implémentons comme indiqué plus haut, en complément à ce réseau les chemins bottom-up et top-down et nous choisissons comme Attention Mechanism, une combinaison de Channel-wise Attention et de Spatial-Attention. Channel-wise Attention permet de diminuer le nombre de chaines prises en compte en annulant celles qui sont inutiles. Quant au choix de Spatial Attention, il permet de réduire la confusion entre le bruit dans l'image et des objets de petites dimensions comme il en existe dans le dataset. A titre de comparaison, on passe une image à notre backbone modifié et à celui du modèle de base de pytorch et on affiche le premier channel du dernièr niveau de la Feature pyramid.

On remarque déjà rien qu'avec les features pré-entrainés, des couleurs et des formes plus vives au niveau de la zone d'intérêt chez lebackbone pré-entrainé.

– RPN
RPN est la deuxième et la dernière partie du réseau que nous avons implémenté. Mais avant cela, il fallait d'abord mettre en forme les données à passer au réseau. Pour cela, on se réfère au concept d'anchor qui permet d'avoir les coordonnées de zones au sein d'une image donnée. Un anchor correspond en effet à un cadre défini par une échelle(scale) et un ratio donné et qui, centré en un point de la Feature map permet de faire le lien avec l'image d'origine. En général, on définit 9 anchors par point de la Feature map pour 3 scales et 3 ratios. Cependant, nous définissons une échelle fixe pour chaque niveau de la pyramide des features, ce qui nous donnent donc 3 anchors par point. Le principe est donc de classifier un anchor donné entre les catégories objet et non objet en fonction du point du vecteur (1,C) dans la feature map, auquel est associé ce anchor et de faire correspondre par un procédé de régression, les box qui ont été classées comme étant objet à la bounding box de cet objet considérée alors comme ground truth.

[1]: Attention Mask R-CNN for ship detection and segmentation from remote sensing images XUAN MIE-MENGYANG DUAN-HAOXUAN DING-BINGLIANG HU-AND ADWARD K.WONG

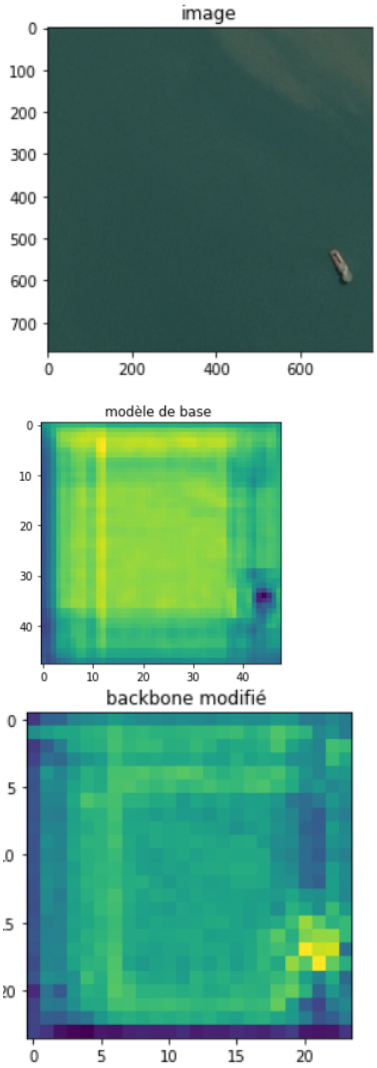


Figure 8: Images des dernières niveaux des backbones