CentraleSupélec

OPENCLASSROOMS

Segmentation d'images

Contenu

Introduction

Dans l'objectif de faciliter le comptage et la détection de cellules par exemple dans le sang, il nous est demandé sur cette compétition Kaggle de faire une prédiction de la position de chaque cellule présente sur une image. Nous allons donc explorer une nouvelle partie de la data analyse qu'est la **segmentation d'images.**

Pour ce faire, nous allons aborder le pré-processing, la modélisation avec 3 différents modèles puis le post-processing. En effet, comparé aux précédents projets, une phase importante de ce projet se situe sur la phase de post-processing. Pour finir, nous parlerons des résultats obtenus, les possibilités d'améliorations puis une conclusion.

Présentation du Projet

Objectifs

Comme évoqué dans l'introduction, nous devons prédire un masque pour chaque cellule de l'image. Pour l'entrainement, nous avons accès à 670 images avec pour chacune d'elle, n-images représentant le masque de chaque cellule. A partir de celles-ci, nous devons entrainer un modèle permettant de d'extraire au mieux n-mask couvrant chaque cellule. Concernant l'évalaution, on a 65 images.

Evaluation

L'évaluation Kaggle sur ce projet un peu complexe. Dans un 1er temps, Kaggle vérifie qu'il n'y a aucune superposition de masques, sinon le score final vaut 0.

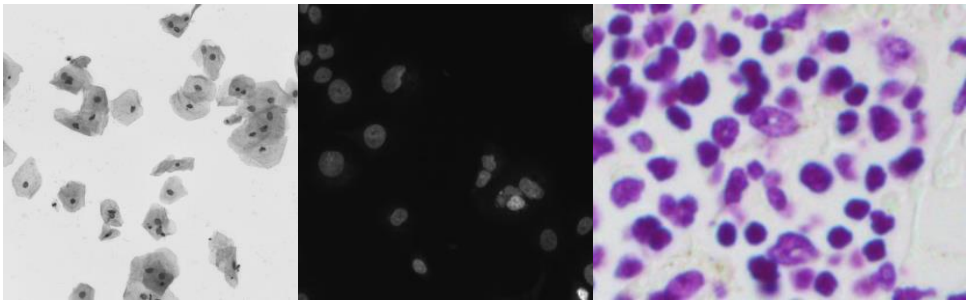
Ensuite, pour chaque image du test set, n-masques sont fournis. Ils sont comparés un par un aux autres masques en récupérant le Intersection Over Union (noté IoU par la suite). Le meilleur score est conservé car il signifie que l'on essaye de masquer cette cellule.

Pour chaque masque de chaque image, on a donc un IoU donnée. Pour tout un ensemble de seuil de IoU (allant de 0.5 à 0.95 par steps de 0.05), si celui-ci est supérieur à ce seuil, le masque est considéré comme un "hit". Pour chaque image, la moyenne des masques avec un "hit" est faite. C'est ce résultat qui est utilisé pour l'évaluation. Afin de limiter les ressources de calcul, seulement quelques masques sont testés.

Exploration

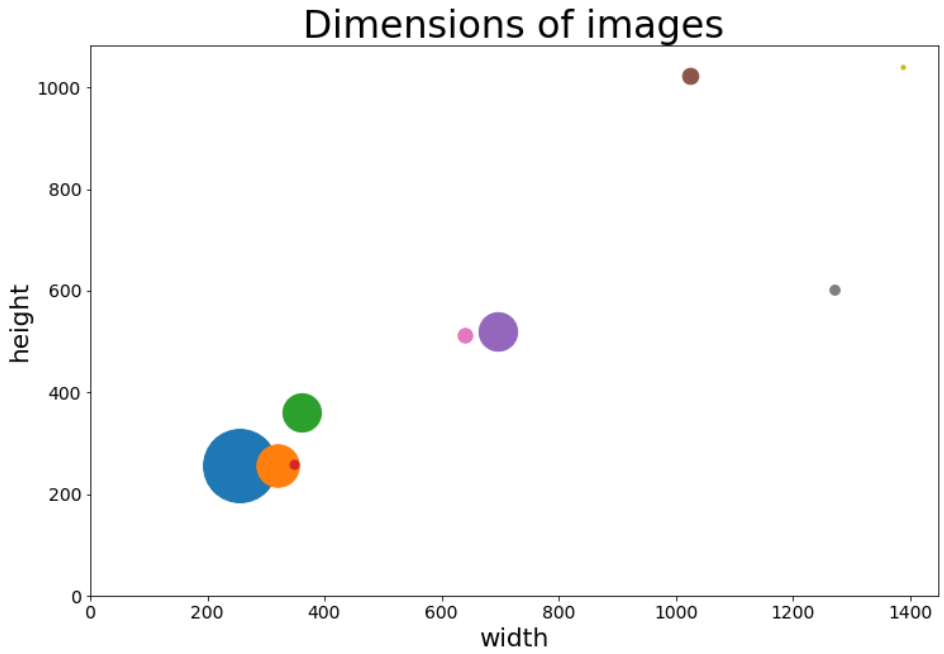
Type d'images

Lors de l'exploration, on peut remarquer que l'on a 3 types d'images. Certaines sont en couleurs, d'autres avec les cellules en blanches sur un fond noir (ce type est très majoritaire) et d'autres avec la cellule en noir sur fond blanc.



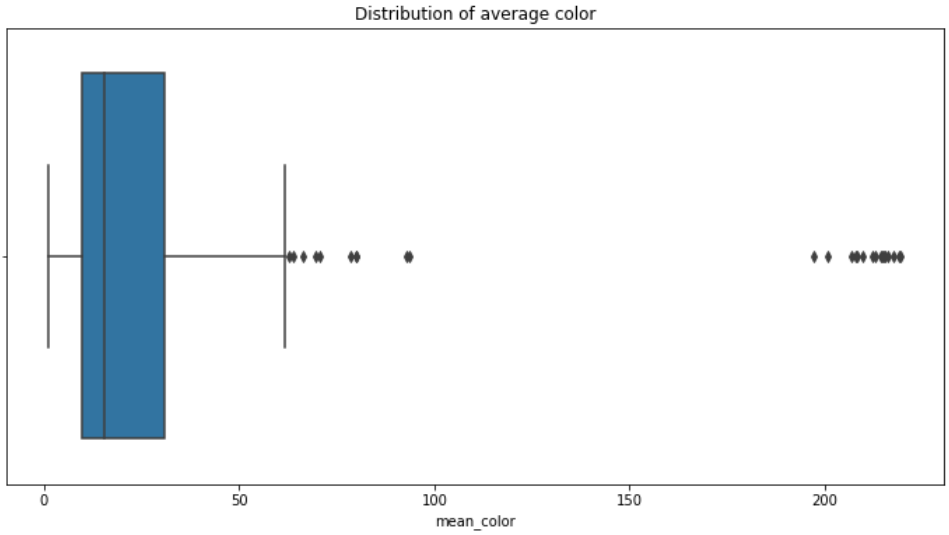
Taille des images

Concernant la taille des images, elles sont assez variables mais assez petites et avec des ratios (largeur/hauteur) corrects. On peut afficher les dimensions via un nuage de points comme ci-dessous:



Couleur moyenne

Pour la phase de pré-processing, un des objectifs était de convertir ces 3 types d'images dans un seul type (fond noir et cellules en blanc). Pour ce faire, les images en couleurs ont été converties en Noir et Blanc et la couleur moyenne a été récupérée. On peut voir le boxplot de celle-ci ci-dessous.

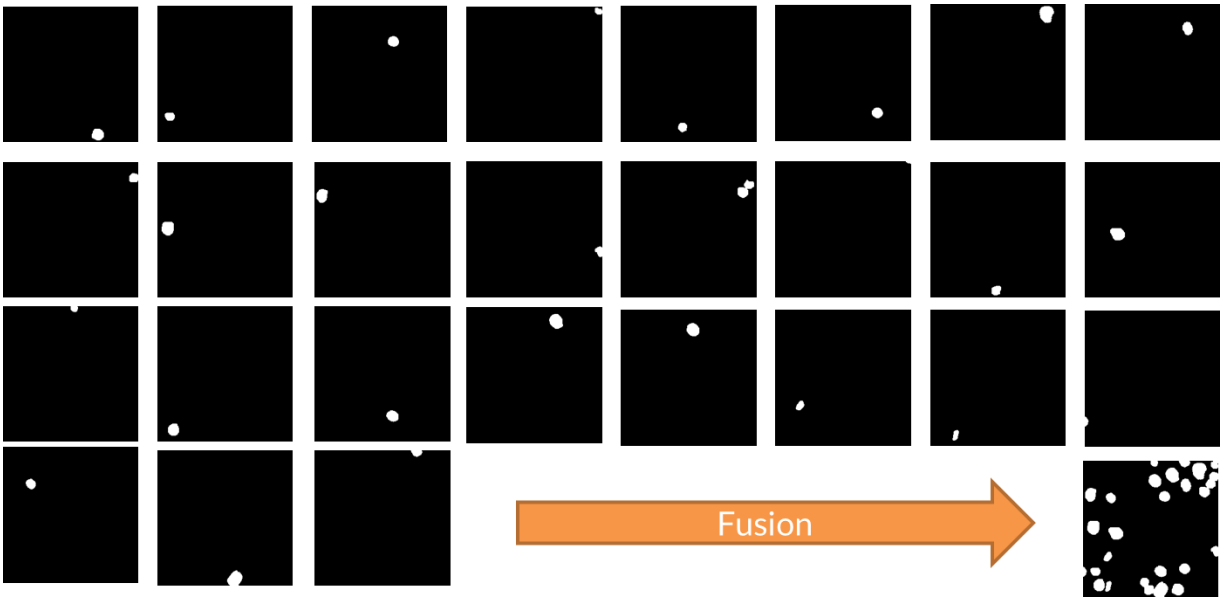


On peut remarquer que certaines images ont une moyenne très haute donc sont majoritairement blanches. A l'inverse, on a beaucoup d'images avec le noir majoritaire. On a un gap important entre ces 2 types d'image (de 95 à 190) et on peut donc faire l'inversion N&B => B&N en fonction d'un seuil mis à 150 dans notre cas.

Pré-processing

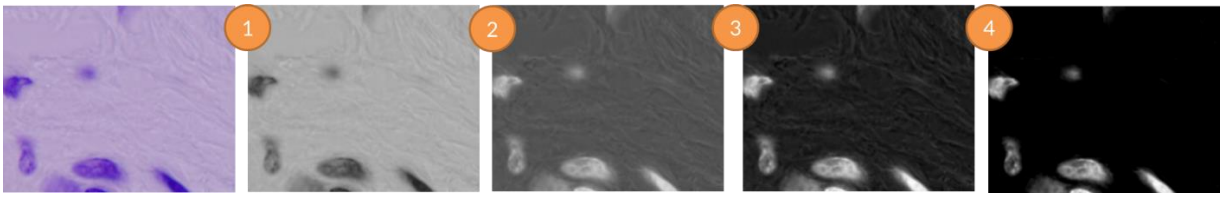
Préparation des masques

Concernant les masques fournis, on a un dossier par image avec tous les masques de celle-ci. Pour la phase de pré-processing, une simple fusion a été faite afin d'avoir un masque global.



Augmentation de Contraste

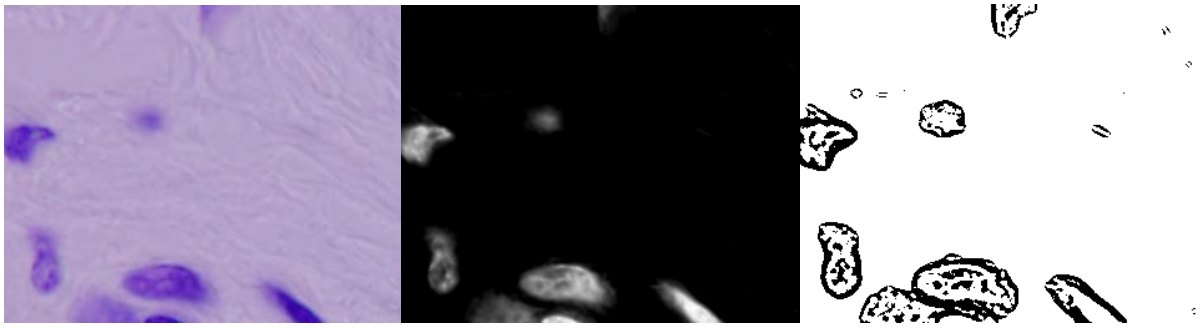
Comme on l'a vu précédemment, on a un seuil permettant de détecter si l'image est blanche sur fond noir ou noir sur fond blanc. Ajouté à cela, on va normaliser le contraste. L'objectif étant d'avoir le max à de l'image 255 et le minimum à 0. Cela a pour effet d'augmenter les contrastes. Par la suite, une seconde augmentation sera faite avec pour objectif dépasser ce maximum/minimum et ensuite clipper l'image. Sur une image test, cela donne :



La phase 1 est la conversion en Noir et Blanc, la phase 2 est l'inversion en Blanc et Noir (moyenne > 150). La phase 3 est la normalisation des contrastes et la phase 5 est le "seuillage". Cela permet d'avoir des images homogènes et mieux faire ressortir les noyaux.

Adaptative Threshold

Lors de différents pré-processing testés (disponible dans le notebook Data Préparation), un type de pré-processing m'a aussi intéressé, c'est l'adaptative Threshold. Celui-ci permet de faire ressortir les contours de cellules et permettre ainsi d'aider à séparer les noyaux qui se touchent par la suite.



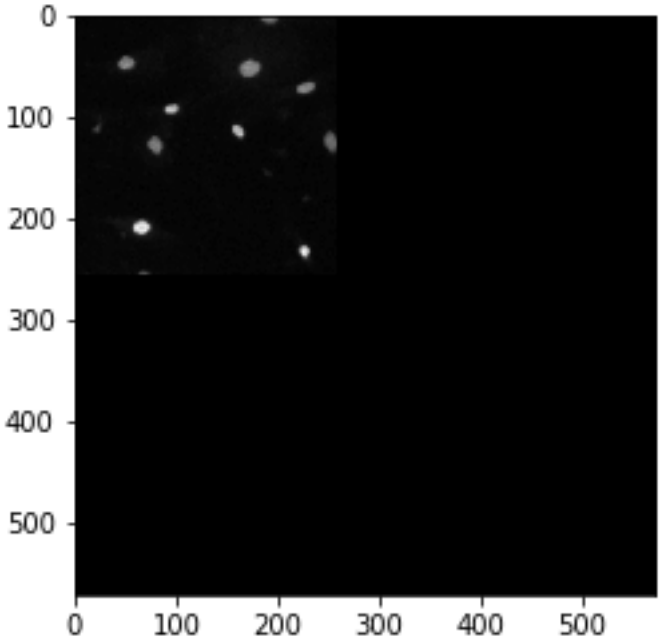
Ci-dessus, se trouve à gauche l'image de base, au milieu l'image pré-processée et à droite le résultat de l'adaptative Threshold. Ce résultat étant intéressant pour un dernier modèle.

Redimensionnement

Pour les CNN que l'on verra par la suite, on a besoin d'une image d'une dimension fixe. Selon les modèles, des images de 512 x 512 étaient trop lourdes en mémoire donc les images ont été redimensionné en 256 x 256. Pour ce redimensionnement, 2 approches ont été pensées.

Version non déformée

Afin de ne pas impacter la forme des cellules, le redimensionnement imaginé au départ était de ne redimensionner que les images plus grandes que la taille voulue. Sinon elle sera positionnée dans un coin. Quant au redimensionnement, le facteur est constant pour avoir le plus grand coté égal à la taille souhaitée.



Ci-dessus, on a le cas d'une image de 256 x 256 pour une image voulue de 512x512. Malheureusement le fond n'étant pas toujours parfaitement noir, cela a créé beaucoup de bruits lors de l'entrainement et performait moins bien.

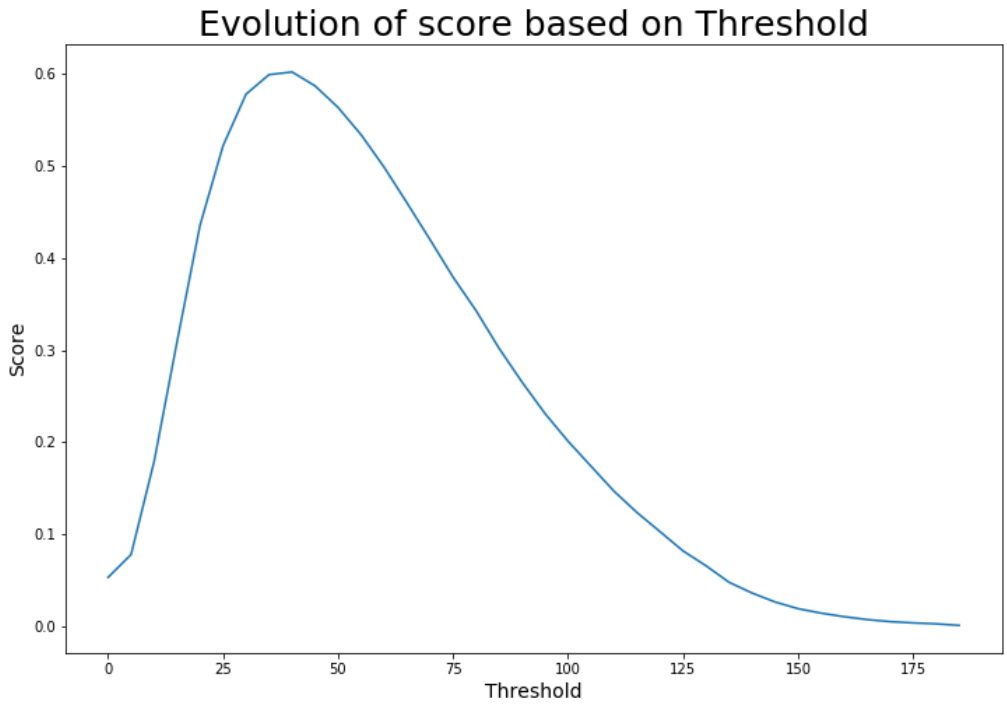
Version classique

Suite à l'échec du précédent redimensionnement, et au vu des ratios (hauteur/largeur) qui ne sont pas extrêmes, un scaling classique a été utilisé. Dans un premier temps, afin de tester le modèle en 128 x 128 puis en 256 x 256.

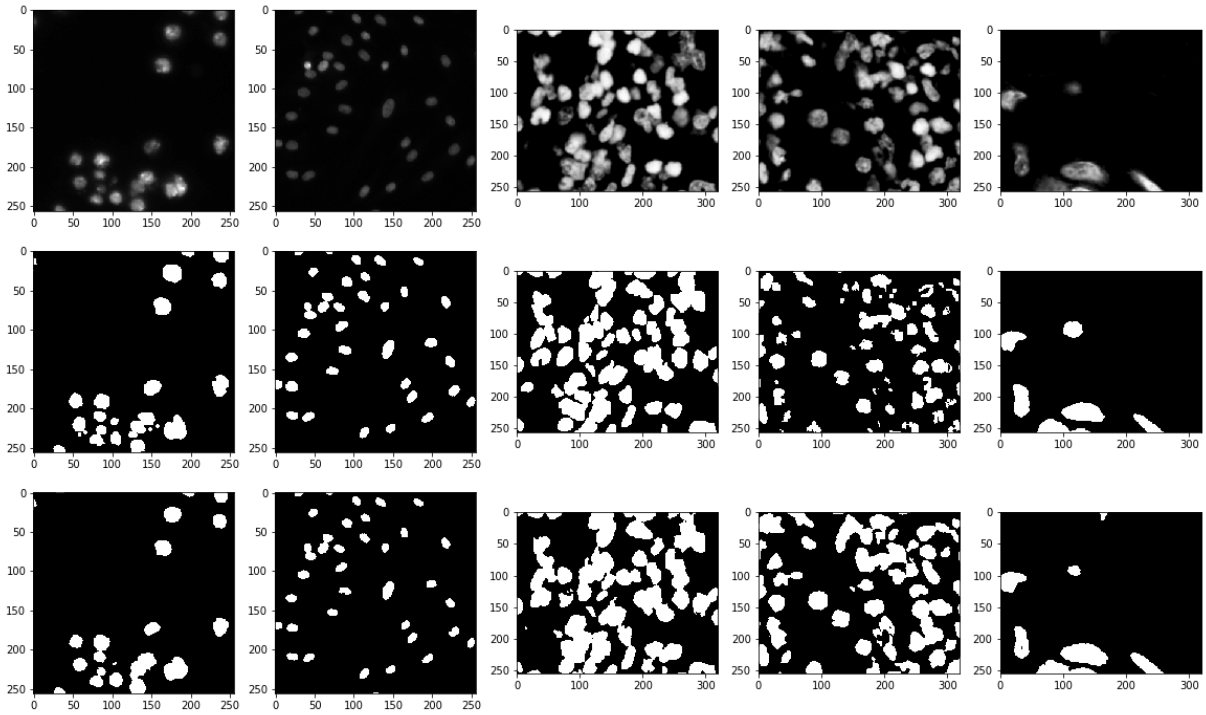
Modèles

Modèle Simple

Le premier modèle que l'on peut tester afin d'avoir une baseline, est un modèle classique n'utilisant que de l'*image processing* avec openCV. Pour ce faire, nous avons déjà les images pré-processée précédemment. Sur celle-ci, un simple threshold sera appliquée sur la valeur de chaque pixels. Si on prend un seuil trop bas, l'image aura trop de blanc. L'union sera très grande, résultant à un faible IoU. Si on prend un seuil trop haut, on aura trop peu d'intersection et donc un score faible aussi. De ce faite, une mesure du IoU **sur le masque complet** a été faite suivant tous les seuils possible. On trouve les résultats suivants :



Le seuil optimal est à 40. Basé la dessus, on peut prédire les masques et les comparer à ceux fournis. Sur l'image ci-dessous, la 1ère ligne montre les images pré-processées, la 2nde ligne est la prédiction et la dernière ligne est le vrai masque.



On peut remarquer que les résultats sont assez similaires.

U-net Classique

En 2015, un modèle de segmentation d'image dans le milieu médical a gagné le ISBI 2015 loin devant les autres modèles. Ce projet étant similaire, c'est le modèle le plus utilisé par les compétiteurs. C'est donc aussi un modèle que j'ai voulu testé.

Un des bémols sur ce projet avec le U-net classique (architecture présentée en Annexe 1), c'est qu'il coupe les bordures. Sur une image de 572x572, l'image traitée est de 388x388 tout comme le masque est de 388x388. Afin d'avoir une taille identique en entrée qu'en sortie, j'ai décidé de prendre une taille multiple de 2 afin de ne pas avec de soucis lors du Max Pooling ainsi que de passer tous les padding en "same" (L'architecture est fournie en Annexe 2).

Ce modèle a été entrainé avec les images pré-processées pour avoir un nombre de layer fixe (à 1) redimensionné en 256 x 256. La data-augmentation a été testée mais ne donne pas de bons résultats pour les mêmes raisons que le redimensionnement sans déformations (bordures blanches). Quelques images et leurs masques augmentés sont présents en annexe 4.

Un callback a été mis en place aussi pour enregistrer à chaque Epoch la prédiction sur une image de test. En fin de training, un gif est généré avec l'ensemble des prédictions. Les gifs sont présent dans le dossier training sur github (https://github.com/Coni63/OC_DS/tree/master/P9/img/training). Le numéro à la fin correspond à l'ID de l'image de test. J'ai utilisé 4 images ayant de des masques très différents.

Si on regarde ces gif, on peut remarquer que la prédiction n'est pas stable du tout avec souvent des masques qui deviennent tout blanc en 1 seule Epoch. Cependant, le modèle sauvegardé (après 50 Epoch) est **par chance** sans ce défaut. Selon les entrainements, on a parfois des cellules très fusionnées. Pour aider à les décomposer, un modèle extended a été testé.

U-net Extended (ou multi-arm)

Sur ce modèle, l'objectif était d'utiliser l'image pré-processées mais aussi l'image de base et l'adaptative threshold vu au départ. Pour ce faire, la partie "convolution/encodage" a été dupliquée pour prendre ces 3 images en inputs. Et pour le transfert des tenseurs lors de la dé-convolution, c'est pareil sauf que la matrice de concaténation est plus grande. Le modèle est aussi représenté en Annexe 3.

Concernant l'entrainement, la data augmentation a aussi été supprimée pour les même raisons. Le callback a aussi été mis en place (mais un peu plus complexe) afin d'avoir les même gif en comparaison. Les gifs des entrainements sont aussi présent sur github à la même adresse. Dans les dossiers, les images des entrainements, ainsi que l'image de base et le masque sont disponibles.

Post-processing

Comparé aux précédents projets, une phase de post-processing est nécessaire. En effet, nous prédisons un masque global pour l'image mais la compétition demande un masque par cellule.

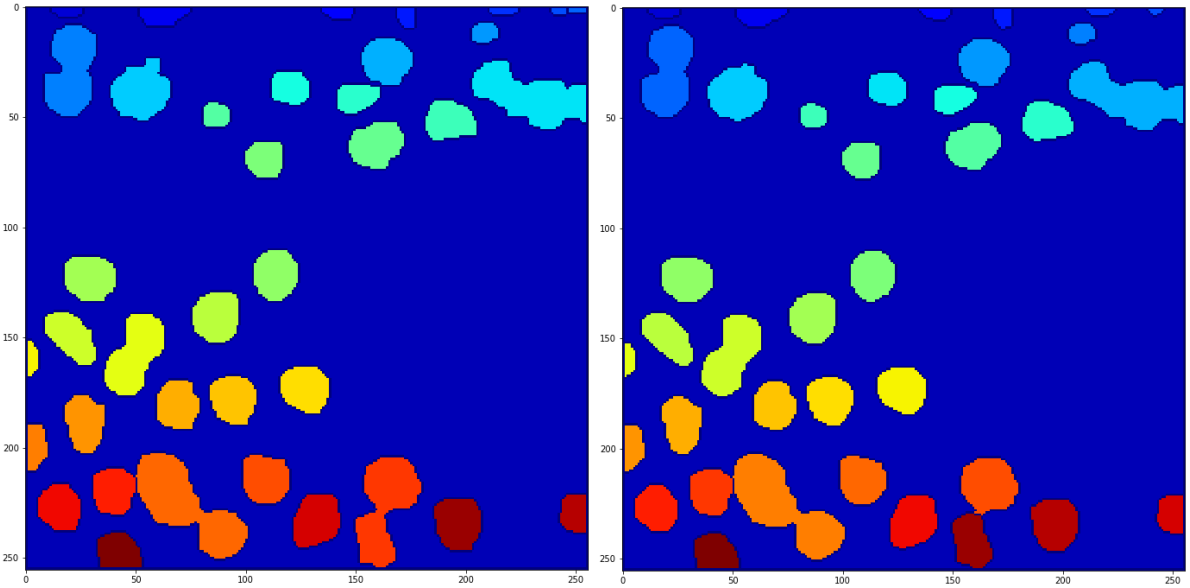
Dans un 1er temps, pour les CNN, il a fallu aussi déterminer le seuil à partir de laquelle on considère un pixel blanc ou noir (comme on l'a fait pour le modèle simple). En effet, on a en sortie une image basé sur une sigmoïde pour chaque pixel. Comme on l'a fit pour le modèle simple, l'étude du IoU en fonction du seuil a été fait sur le masque complet. Les résultats seront présentés dans la prochaine partie.

Ceci fait, on peut décomposer l'image par zones blanches qui se touchent. Cela peut se faire avec diverses librairies comme skimage (équivalent de sklearn mais pour les images) ou encore OpenCv. Cependant, cela n'est pas optimal car si on a 2 cellules très proche, il se peut qu'une ou 2 pixels les connectent et donc réduisent fortement le score (car il manque un des masques et l'union est plus grande). Pour améliorer la décomposition, on va appliquer différentes méthodes de la librairie OpenCv.

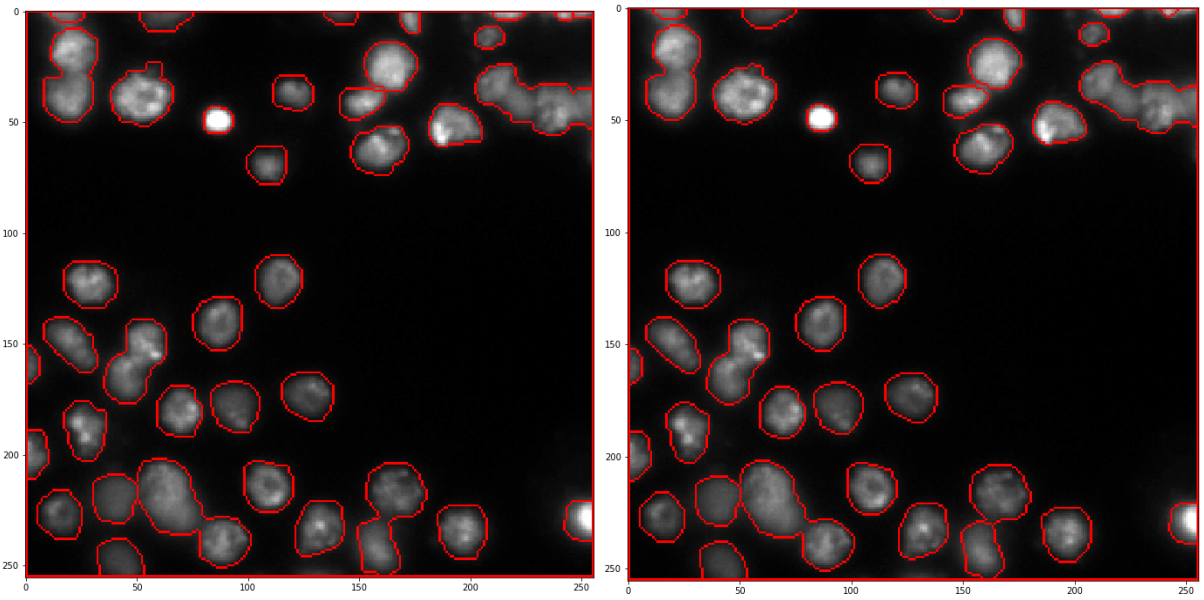
La 1ère étape est de supprimer le bruit (des pixels isolés qui seraient considérer comme des masques). Cela peut se faire avec un érode puis dilate. Si la cellule est plus grande que ce que l'on érode, on la garde à l'identique (avec le dilate), sinon elle disparait.

On peut ensuite appliquer un filtre oval pour lisser les prédictions sur des formes ovales. Cela n'est pas forcément nécessaire mais aide bien à la décomposition (ce gain sera montré à la fin).

Ensuite, il faut calculer les distances inter-pixels pour trouver les centres des cellules. Par rapport aux distances, on peut décomposer 2 cellules qui se touchent avec la méthode 'connected Components' de OpenCv. On a donc maintenant 1 masque par cellule (on peut voir une décomposition ci-dessous avec à gauche le traitement sans le filtre oval et à droite avec celui-ci). On peut remarquer qu'avec, la cellule en bas à gauche se trouve aussi décomposée.



Ou encore en superposition de l'image de base:



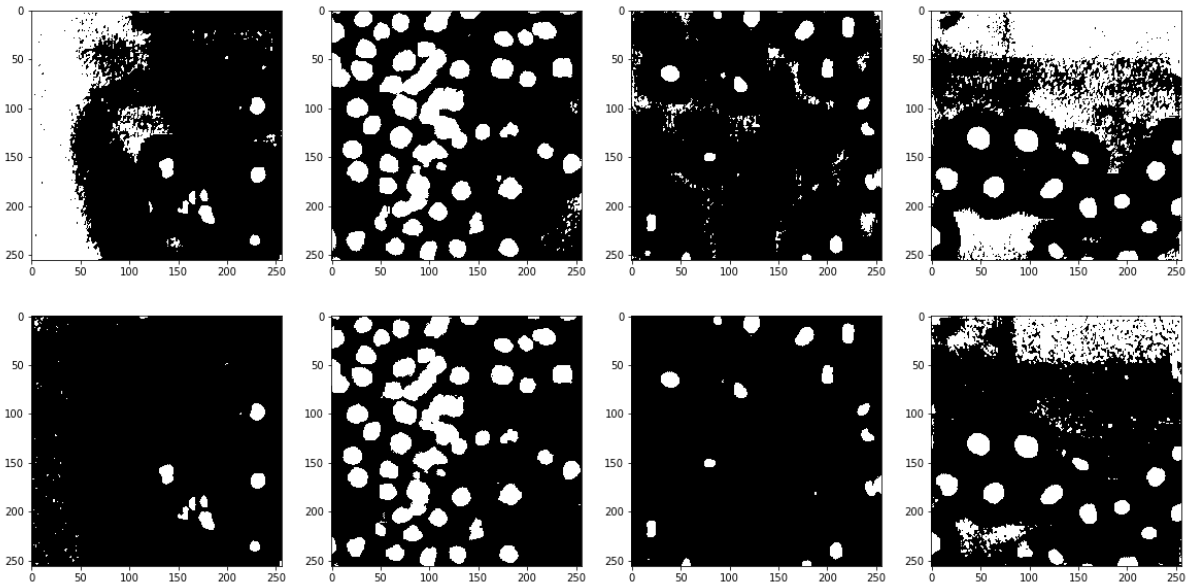
Sur Github, vous trouverez aussi les différentes prédictions faite sur l'image de base avec les contours des cellules avec les différentes méthodes dans les dossiers extractions présents à [cette adresse.]

Analyse des résultats

Au vu de la complexité de l'évaluation sur Kaggle, j'ai basé mon évaluation sur le IoU des masques complets dans un 1er temps. Pour toutes les images que j'avais dans mon test set, j'attribuais un hit si le IoU du masque complet avec la prédiction était au-dessus de chaque seuil (de 0.5 à 0.95 par step de 0.05). Ensuite, une moyenne, était effectuée et on trouve :

- 60.2 % avec le modèle classique (t = 40)
- 42.1 % avec le U-Net Classique (t = 0.9999)
- 71.9 % avec le U-Net Extended (t = 0.4)

Concernant les valeurs de "t", pour le modèle classique c'est l'intensité du pixel entre 0 et 255. Pour les CNN, c'est une valeur qui doit être entre 0 et 1. On peut remarquer que le modèle extended performe bien mieux que le modèle classique par contre les seuils sont totalement différents (0.4 pour le extended contre 0.9999 pour le modèle classique). Cela s'explique par le bruit de prédiction sur le modèle classique. Le masque prédit avec le U-net sur 4 images avec un seuil de 0.3 (ligne 1) et 0.9999 (ligne 2) sont présents ci-dessous:

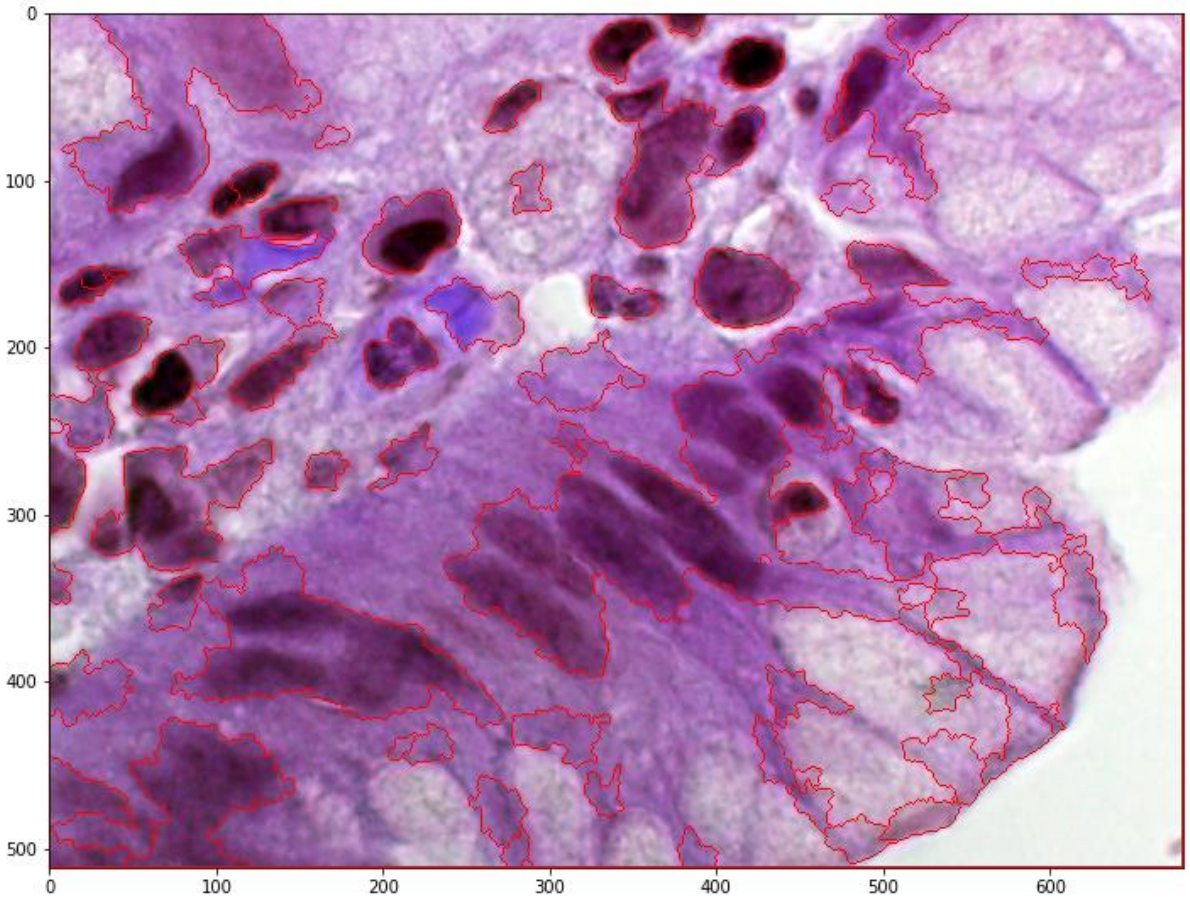


Suite au post-processing vu précédemment, les résultats sont beaucoup plus faibles :

- 0.226 avec le modèle classique
- 0.189 avec le U-net classique entrainé actuellement (le top score à 0.301 a été atteint lors d'un précédent entrainement, c'est très variable car le modèle a du mal à converger comme on le voit dans le gif d'entrainement)
- 0.243 avec le U-net extended (un score de 0.208 a été attribué aussi à ce modèle)

Cette différence s'explique par 2 points principaux :

1. Il faut mieux privilégier des masques qui débordent sans overlap car dans le calcul du IoU, l'intersection impacte plus que l'Union. En effet, si on prend une cellule qui fait 100 pixels de surface et que l'on prédit 80 pixels tous à l'intérieur du vrai masque on aura, une intersection de 80 et une union de 100 soit 0.8 de IoU. Si maintenant, on prédit 20 pixels de trop à l'extérieur, l'intersection fera 100 et l'union 120 soit un score de 0.83 (+4 % pour la même erreur).
2. L'évaluation n'est faite que sur certaines images et surement les plus complexes. Dans mon cas, certaines prédictions sont très mauvaises comme l'image suivante. On a tellement de variation de contraste que le modèle considère que ce sont des noyaux. De plus, comme beaucoup de noyaux se touchent, ça élargie d'autant plus l'Union et on a rapidement un score de 0 (si ce n'est pas une cellule) ou inférieur à 0.5 (si on prédit une cellule trop grande).



Pistes d'évolutions

Pour finir ce dernier projet, abordons les pistes d'évolutions possibles.

1. Malheureusement on ne peut pas utiliser le U-Net de 2015 car il supprime 97 pixels de chaque bordure de par son architecture. De ce fait on ne pourrait prédire les cellules en bords d'images. Cependant, il doit être possible de redimensionner l'image d'entrée à 388 x 388 et la placer au centre d'une image à fond noir de 572 x 572 pour faire la prédiction du masque. Cependant comme on l'a vu, cela crée beaucoup de bruit et nuirait surement à l'entrainement.
2. Un entrainement sur des images de 512 x 512 permettrait surement une meilleure segmentation. Malgré une estimation mémoire nécessaire pour l'entrainement de 246 Mo / images, l'entrainement déclenche une erreur mémoire de la carte graphique même avec un batch de 3 images. Les calculs sont fournis en Annexe 5.
3. Comme on le voit dans le gif d'entrainement, le bruit varie d'une Epoch à l'autre. On pourrait imaginer entrainer de multiples modèles, faire les prédictions de masque et faire l'intersection de ceux-ci. On aurait un système d'ensemble et cela réduirait surement fortement le bruit. Le bémol, je ne pense pas qu'il soit possible de lancer plusieurs modèles en parallèles sur Keras car lors de la compilation, l'ensemble de la mémoire est allouée. De ce faite il faudrait soit partager les calculs sur plusieurs GPU ou alors faire les prédictions en série mais le temps serait beaucoup plus grand.

On peut aussi parler de ce qui ne marche pas :

1. La data augmentation crée trop de bruit, ce fût aussi le cas lors de l'entrainement du U-net en 2015. La seul data-augmentation valable est la déformation élastique.
2. Les pre-trained modèles performent très mal sur ce type de segmentation.

Conclusion

En conclusion, nous avons découvert pendant ce projet la segmentation d'images. Très utilisé notamment dans le milieu médical, mais aussi en vision (pour les voitures autonomes par exemple), c'est une technologie assez récente. De très bons résultats ont été obtenus avec U-net en 2015 cependant la tâche n'était pas vraiment la même, dans leur cas, il fallait juste séparer les cellules et non extraire des masques.

Dans notre cas, les résultats visuels sont plutôt bons sauf sur certaines images. Du fait que Kaggle n'utilise que quelques images et des masques séparés, le score est beaucoup plus faible sur celui-ci. Le modèle simple fonctionne assez bien voir mieux que le modèle U-net de base. Le modèle U-net Extended, semble être plus performant et plus stable mais avec des fortes restrictions de mémoire.

Concernant l'apprentissage, nous avons découvert le principe des auto-encodeurs mais pour les images. Ce projet, m'a permis aussi d'approfondir bien plus en détail le pré-processing et post-processing avec OpenCV comparé au projet 7.

Sources

Kernels :

https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277

https://www.kaggle.com/ahassaine/pure-image-processing-lb-0-274

Sites externes :

https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

https://arxiv.org/abs/1505.04597

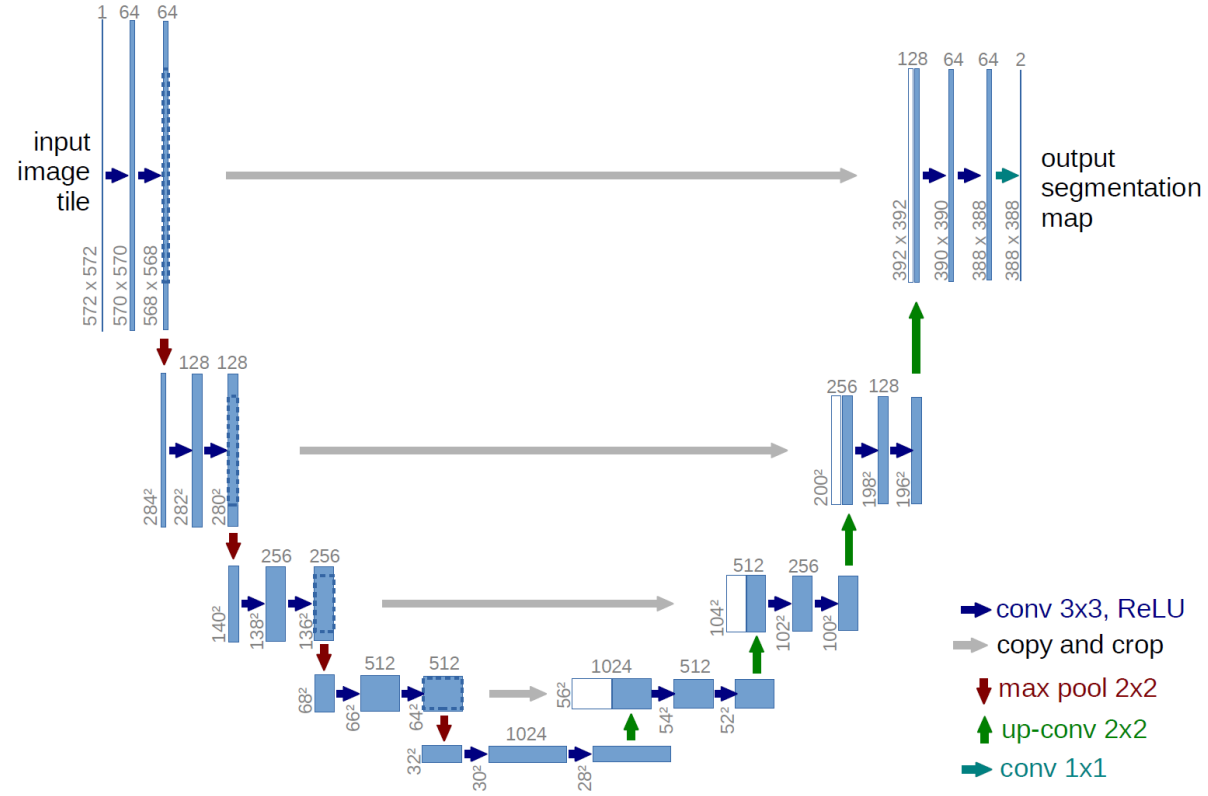
https://docs.opencv.org/3.3.1/d3/db4/tutorial_py_watershed.html

https://stackoverflow.com/questions/17389098/opencv-image-preprocessing-for-object-detection

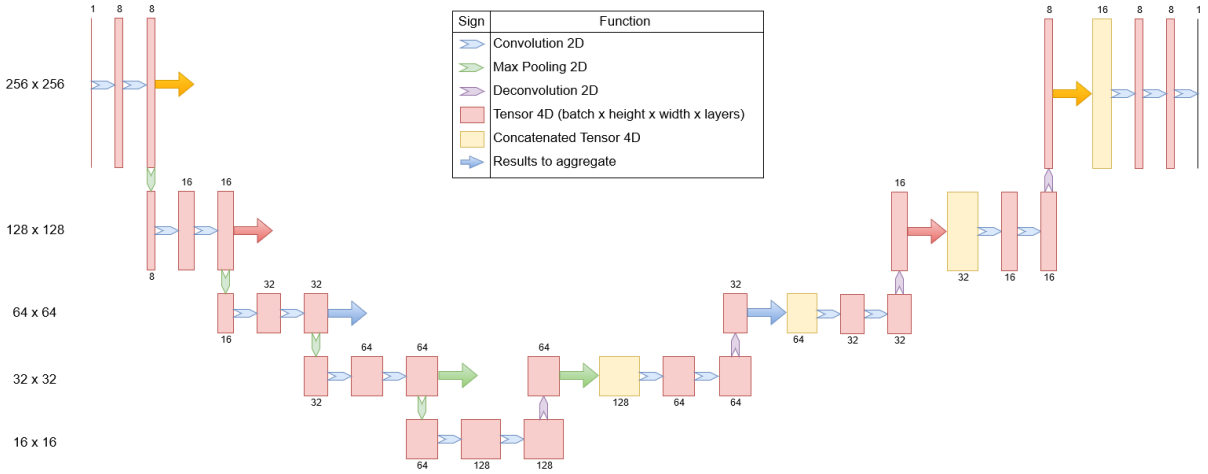
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html

Annexe

Annexe 1 : Structure du U-net Standard

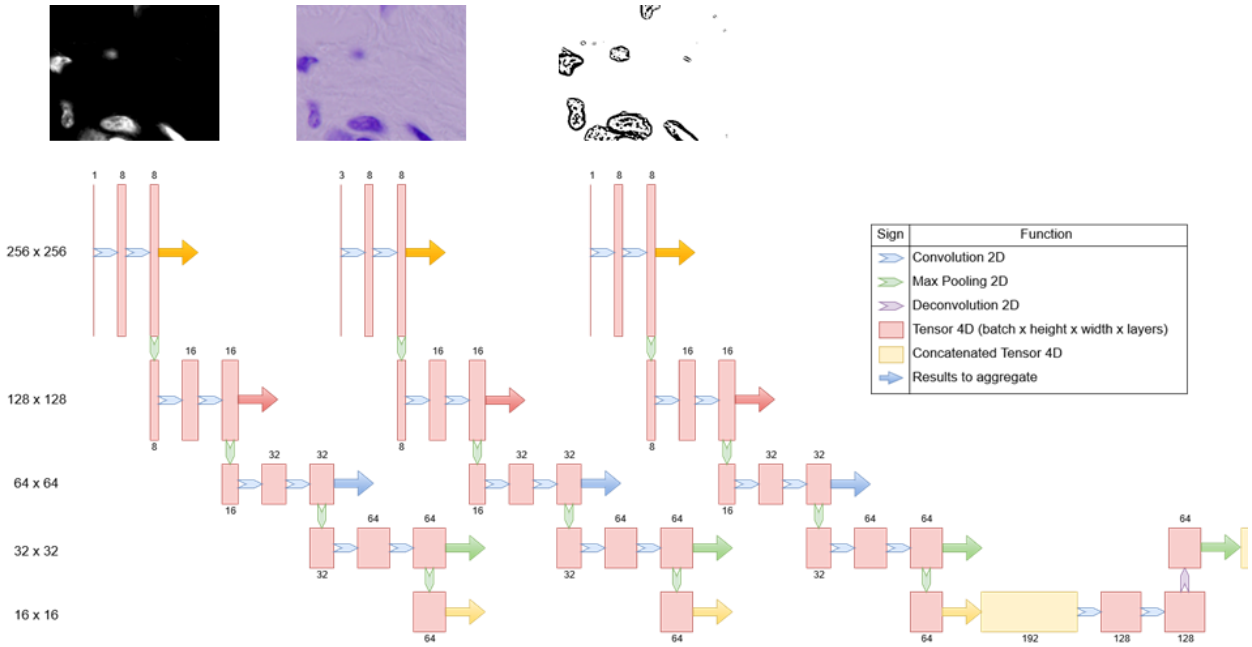


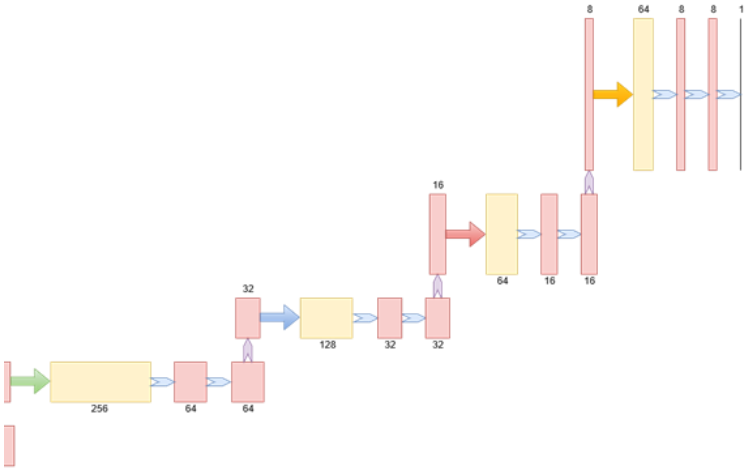
Annexe 2 : Structure du U-net modifié



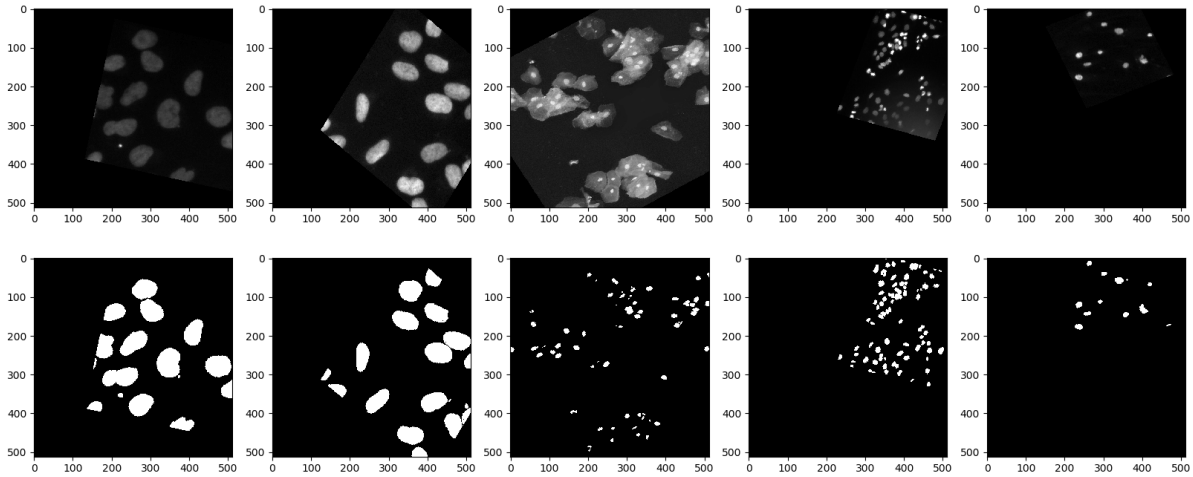
Annexe 3 : Structure du U-net extended

Pour des raisons de lisibilité, le modèle a été coupée. Chaque bras de convolution sont représentés en haut et la partie déconvolution est en dessous.





Annexe 4 : Data Augmentation



On peut remarquer avec cette data augmentation les bords d'images qui apportent trop de bruits lors du training.

Annexe 5 : Mémoire nécessaire par image

Pour ce calcul, le nombre de float 32 par place dans les tenseurs ont été calculé. Pour un float32, on a besoin de 4 octets pour le stocker, on peut donc savoir la mémoire nécessaire pour remplir l'ensemble des tenseurs du modèle. Cependant, il y a une part de mémoire nécessaire aussi pour la back propagation, la compilation, etc…

Modèle 256 x 256				
Largeur de l'image	Nombre de layers par "bras"			
	input1	input2	input3	output
256	17	19	17	89
128	40	40	40	112
64	80	80	80	224
32	160	160	160	448
16	64	64	64	448
Nombre de float32	16121856			
Mémoire nécessaire par image	61,5	Mo		

Modèle 512 x 512				
Largeur de l'image	Nombre de layers par "bras"			
	input1	input2	input3	output
512	17	19	17	89
256	40	40	40	112
128	80	80	80	224
64	160	160	160	448
32	64	64	64	448
Nombre de float32	64487424			
Mémoire nécessaire par image	246	Mo		