Challenge SLB
Détection de corrosion dans les conduites en acier

Tocquec Louis
Ecole Normale Supérieure Paris-Saclay
louis.tocquec@ens-paris-saclay.fr

Abdelaziz Aissa
Ecole Normale Supérieure Paris-Saclay
aissa.abdelaziz@ens-paris-saclay.fr

Abstract

Ce travail a été réalisé dans le cadre du cours "Apprentissage et génération par échantillonnage de probabilités" dispensé par Stéphane Mallat au Collège de France. Il consiste en la création d'un modèle de prédiction pour le challenge data CorroSeg proposé par SBL et plus précisément en la détection de corrosion dans les conduites en acier. Le projet a été réalisé en Python en faisant appel à la bibliothèque "PyTorch".

Code source

1 Présentation du Challenge

La corrosion des puits et des tuyaux dans l'industrie représente une grande menace à l'origine de nombreux accidents et fuites. Sa détection est alors en enjeu majeur permettant de réaliser d'importantes économies mais surtout d'augmenter le niveau de sécurité à proximité de ces puits et d'empêcher une éventuelle importante source de pollution.
Cette corrosion se manifeste par une perte importante d'épaisseur des canalisations aussi bien à l'extérieur qu'à l'intérieur des conduits.
Concernant le Dataset, il est composé d'un ensemble d'images représentant l'épaisseur des canalisations. Les images couvrent 15 puits différents et sont, à l'origine, en très grand format (jusqu'à 68580x36 pixels). De ce fait, elles ont été préalablement découpées en carré de 36x36 pixels. Les défauts liés à la corrosion sont réparties en trois catégories : la corrosion par piqûres, les défauts localisés et les rainures axiales.
L'objectif du challenge est alors de détecter ces marques de corrosion en réalisant une segmentation efficace des images de façon à localiser efficacement les parties abîmées des canalisations. La métrique évaluant la performance du modèle est évaluée selon l'intersection sur l'union (IoU) :

$$\frac{|X \cap Y|}{|X \cup Y|}$$

où X est l'ensemble des pixels prédits et Y la vérité terrain. Lorsque X et Y sont vides l'intersection sur l'union est prise comme étant égale à 1.
Un exemple d'image qui compose le Dataset ainsi que son masque associé sont illustrés dans la figure 1.

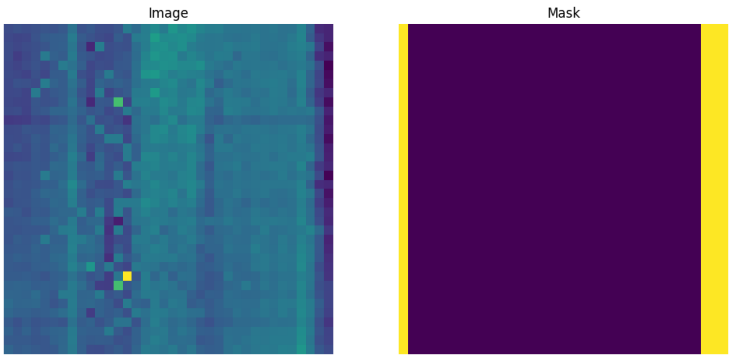


Figure 1: exemple d'image et son masque associé

2 Les modèles

La segmentation appartient au domaine de la vision par ordinateur. Il s'agit de définir sur une image les groupes de pixels appartenant à un même objet ou à une même partie de l'image comme l'arrière-plan par exemple. Formellement, on cherche à identifier les régions $E_1, ...E_n$ qui partitionnent l'image E, i.e telles que :

$$E = \bigcup_{i=1}^{n} E_i$$

Cette tâche occupe une place centrale dans un large panel de domaines allant de l'imagerie médicale jusqu'à la conduite autonome.
Avant l'avènement du Deep Learning, les principaux algorithmes utilisés étaient ceux utilisant des méthodes de seuillage classifiant en fonction des variations d'intensité entre les pixels, ou bien encore des méthodes basées sur les contours [6] ou les graphes [2].
Néanmoins, aujourd'hui l'état de l'art est largement dominé par les réseaux de neurones convolutifs. On peut notamment citer SegNet [1], DeepLab [3] et U-Net [9].
Dans le cadre du challenge, nous nous sommes uniquement intéressés à cette seconde catégorie, et tout particulièrement U-Net, de façon à pouvoir rivaliser avec le benchmark lui-même basé sur un réseau de neurones convolutif.

2.1 U-Net

L'architecture U-Net a, à l'origine, été développé pour des segmentations dans le cadre de l'imagerie médicale. Cependant, sa flexibilité et son efficacité s'exporte à un bien plus large panel d'applications. Il se compose de deux structures : un encodeur et un décodeur.
La partie encodeur du réseau ressemble à un réseau neuronal convolutionnel typique. Comme son nom l'indique, elle encode l'information de départ, ici des images, en un ensemble de caractéristiques qui représente efficacement les données d'entrées. La partie décodeur, quant à elle, tient le rôle opposé. A partir des données codées, l'encodeur cherche à reconstruire une image cohérente par rapport à celle utilisée comme entrée.

Là où réside la particularité de U-Net, et qui lui vaut de ce fait son nom, est sa forme particulière en U dû à des chemins de connexions entre chaque étapes symétriques de l'encodeur et du décodeur. En combinant les caractéristiques apprises dans les couches profondes de l'encodeur avec les informations de bas niveau dans le décodeur, les chemins de connexion permettent une segmentation plus précise.
Cette architecture est illustrée dans le schéma 2.

Plusieurs architectures ont été testées. Finalement, nous avons choisis :

- Encodeur :

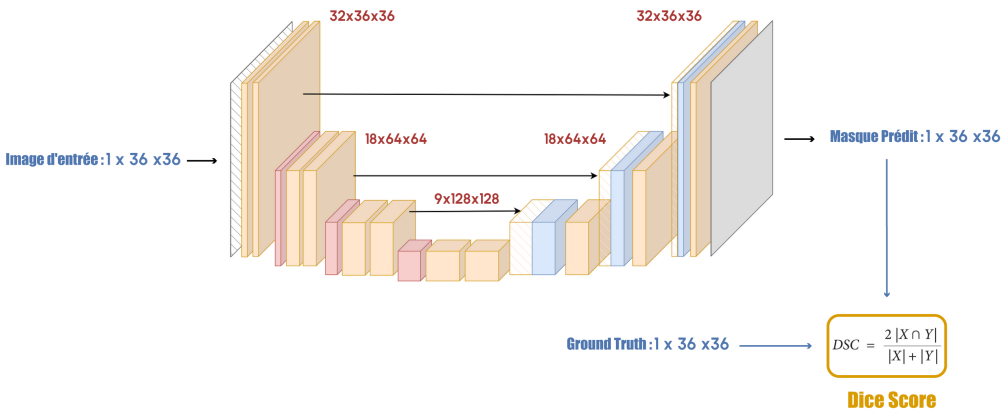


Figure 2: Architecture U-Net

  - une couche de convolution qui produit 32 cartes de caractéristiques en sortie. Elle utilise un noyau de taille 3x3 avec un padding de 1 pour maintenir la résolution spatiale. Elle est suivie par une fonction d'activation ReLU ainsi qu'un max-pooling.
  - une deuxième couche de convolution qui produit 64 cartes de caractéristiques en sortie. Elle utilise un noyau de taille 3x3 avec un padding de 1 pour maintenir la résolution spatiale. Elle est suivie par une fonction d'activation ReLU ainsi qu'un max-pooling.
  - une deuxième couche de convolution et produit 128 cartes de caractéristiques en sortie. Elle utilise un noyau de taille 3x3 avec un padding de 1 pour maintenir la résolution spatiale. Elle est suivie par une fonction d'activation ReLU.
- Décodeur :
  - le première couche est une de déconvolution et se restreint à 64 cartes de caractéristiques en sortie. La fonction ReLU est appliquée après la déconvolution pour introduire de la non-linéarité.
  - Après la déconvolution, les caractéristiques obtenues sont concaténées avec les caractéristiques de la couche correspondante de l'encodeur. Cette concaténation permet de combiner les informations de bas niveau de l'encodeur avec les caractéristiques reconstruites par le décodeur. Les caractéristiques concaténées sont ensuite soumises à une couche de convolution, qui affine les caractéristiques pour la segmentation.
  - Une nouvelle couche de déconvolution est appliquée.
  - On réalise une nouvelle opération de concaténation avec la couche correspondante de l'encodeur. Cela permet de fusionner les informations de bas niveau restantes de l'encodeur avec les caractéristiques reconstruites par le décodeur. On a ensuite une nouvelle couche de convolution qui affine davantage les caractéristiques (32 cartes de caractéristiques) pour améliorer la précision de la segmentation.
  - Une dernière couche de convolution a lieu, suivie d'une fonction d'activation sigmoïde. Cette fonction est utilisée pour normaliser les valeurs de sortie entre 0 et 1 qui correspondent aux probabilités de chaque pixel d'appartenir ou non au masque.

2.2 U-Net et mécanismes d'attention

Attention UNet [8] est une extension d'UNet qui intègre des mécanismes d'attention afin de mieux se concentrer sur les régions importantes de l'image d'entrée. Ces mécanismes d'attention peuvent aider le réseau à apprendre des caractéristiques plus fines et à améliorer la qualité globale de la segmentation.

Étant donné que les marques de corrosions représentent le plus souvent une partie assez réduite des images, l'utilisation de mécanismes d'attention en plus de l'architecture d'U-Net semble être une approche raisonnable dans notre cas. En effet, les mécanismes d'attention vont permettre de

souligner les caractéristiques importantes transmises par les connexions de saut. Un autre avantage est l'augmentation de l'interpretabilité du modèle. En effet, ces derniers permettent une meilleure compréhension de ce sur quoi le modèle se concentre.
On intègre ce processus d'attention à U-Net par le biais d'Attention Gate qui filtre les éléments non pertinents des images. Un exemple d'architecture U-Net avec des mécanismes d'attentions est donné dans la figure suivante 3 tirée de [8].

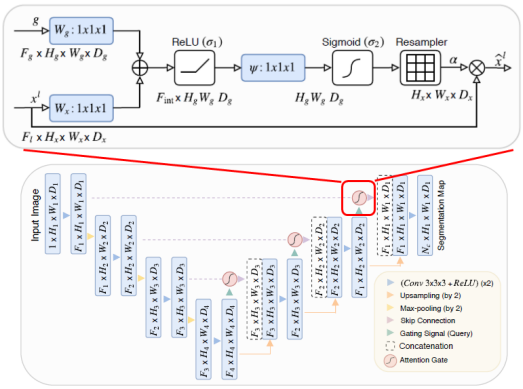


Figure 3: Architecture UNet avec mécanismes d'attention

2.3 U-Net avec blocs résiduels

U-Net avec des blocs résiduels [5] est une extension d'U-Net qui implique le remplacement des CNN standards dans chaque niveau de U-Net par des blocs résiduels (figure 4) . Les blocs résiduels , inspirés de ResNet et dotés de connexions de saut, ont largement contribué à créer des réseaux de neurones convolutifs de plus en plus profonds et ont permis d'obtenir de remarquables résultats.

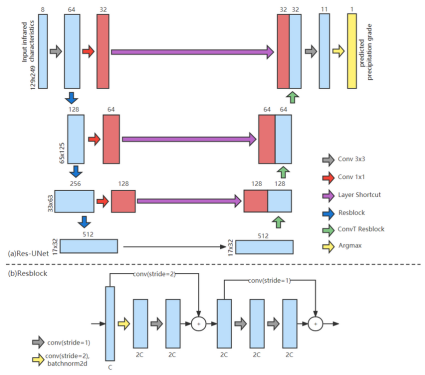


Figure 4: Architecture UNet avec des Blocs Résiduels

3 Fonctions de perte

3.1 Binary Cross Entropy

La première fonction de perte que nous avons choisi pour l'entraînement des modèles était la Binary Cross Entropy qui était également le choix du Benchmark. L'utilisation de cette fonction pour des tâches de segmentation est très fréquente, notamment pour les tâches de classification binaire comme la nôtre.
Elle se définie comme :

$$\begin{aligned} BCE(y, y_{pred}) &= -(y \log y_{pred} + (1-y) \log(1-y_{pred})) \\ &= -y \log(y_{pred} - y_{pred}^2) - \log(1-y_{pred}) \end{aligned}$$

où :

- y est le vrai masque de l'image.
- y_{pred} est le masque prédit par le modèle.

Durant le processus d'optimisation du modèle, la perte BCE est calculée comme la moyenne des pertes pour l'ensemble des pixels.

3.2 Focal Loss

Pour définir la fonction de perte focale, introduisons tout d'abord p_{target}. Si le label d'un pixel $y = 1$ alors $p_{target} = p$ où p est la probabilité prédite par le modèle que l'exemple appartienne à la classe cible. Inversement, si $y = 0$ alors $p_{target} = 1 - p$. On a alors :

$$Foc(p_{target}) = -\alpha_t(1 - p_{target})^{\gamma} \log(p_{target})$$

où :

- α_t est un paramètre de régularisation tel que $\alpha_t = \alpha$ si $y = 1$ et $\alpha_t = 1 - \alpha$ sinon, avec $0 < \alpha < 1$.
- γ est un paramètre de régularisation permettant d'accorder plus d'importance aux pixels tels que p_{target} est petit.

Cette fonction, plus complexe que la précédente, permet de donner plus d'importance aux pixels difficiles à classer, typiquement aux frontières des deux classes. On peut également remarquer que lorsque $p \to 1$ alors $Foc(p_{target}) \to 0$.
Là encore, durant le processus d'optimisation du modèle, la perte focale est calculée comme la moyenne des pertes de tous les pixels.

3.3 Dice Loss

Une autre fonction de perte largement répandue dans la littérature pour des travaux de segmentations est la perte de Dice. Pour la définir convenablement il faut préalablement définir le coefficient éponyme :

$$\textit{Dice} = \frac{2|X \cap Y|}{|X| + |Y|}$$

où X est l'ensemble des pixels prédits et Y l'ensemble des pixels correspondants à la réalité. La perte de Dice se définit alors comme $1 - \textit{Dice}$. On peut facilement vérifier que pour une prédiction parfaite, i.e $X = Y$, $|X \cap Y| = |X| = |Y|$ et donc $Dice = 1$.
Cette perte prenant en compte l'ensemble des pixels, elles favorisent de ce fait les prédictions globales. Ainsi, lorsque la zone d'intérêt est petite, la perte est moindre.

4 Entraînement du modèle

4.1 Prétraitement des données

Nous avons effectué quelques opérations de prétraitement des données. Tout comme le Benchmark, nous avons éliminé des données d'entraînement toutes les images contenant des valeurs aberrantes de façon à obtenir un jeu de données cohérent.
Encore une fois comme le benchmark, et de façon à obtenir un jeu de données plus solide, nous avons appliqué, avec une probabilité de 0.5, un flip horizontal aux images et aux masques, puis toujours avec la même probabilité, nous avons également appliqué un flip vertical. Cette pratique permet d'enrichir les données en créant des variantes supplémentaires. On obtient alors un jeu de données plus diversifiés ce qui permet d'éviter le sur-apprentissage et donc d'obtenir un modèle plus stable et résistant.
Enfin, de façon à s'assurer que les marques de corrosions étaient "régulières", au sens qu'un pixel entouré de pixels de valeur 1 doit avoir comme valeur 1, nous avons combler les trous des marques de corrosion en utilisant la fonction "binary_fill_holes" du package "scipy.ndimage".

4.2 Paramètres

Les paramètres choisis pour l'entraînement sont :

- taux d'apprentissage : 0.001
- nombre d'époques : 50
- weight decay : $10e-5$
- beta : (0.9,0.99)

Le taux d'apprentissage est pris volontairement petit de façon à augmenter la précision de notre convergence, quitte à réduire sa vitesse.
L'optimiseur choisi est Adam, introduit en 2014 dans [7] par Kingma et Ba. Ce choix est largement répandu dans la littérature notamment pour les tâches de Deep Learning car il utilise des moyennes mobiles pour les gradients et leurs carrés pour adapter dynamiquement la taille des pas de mise à jour des poids du modèle. Cette méthode permet de prendre en compte à la fois la direction et la magnitude des gradients, ce qui peut être particulièrement efficace dans des espaces de paramètres complexes et hautement non linéaires.
Weight Decay est un paramètre de régularisation de l'optimiseur qui permet d'éviter le sur-apprentissage en ajoutant une pénalité sur la magnitude des poids. La première valeur de beta contrôle la décroissance de la moyenne mobile du premier moment des gradients tandis que la seconde valeur contrôle la décroissance de celle du second moment des gradients. Les valeurs choisies ici sont classiques et correspondent le plus souovent à des valeurs par défauts.

5 Post-traitements des prédictions

Après avoir entraîné les différents modèles, nous nous sommes intéressés aux prédictions incorrectes de façon à comprendre d'où les sources d'erreurs les plus fréquentes provenaient.
Dans un premier temps, nous avons remarqué que le modèle avait tendance à prédire plus de marques de corrosions que la réalité 5. De ce fait, nous avons décidés d'appliquer un seuillage aux prédictions de façon à ne conserver que les pixels avec une grande probabilité d'appartenir à une marque de corrosion (> 0.75).

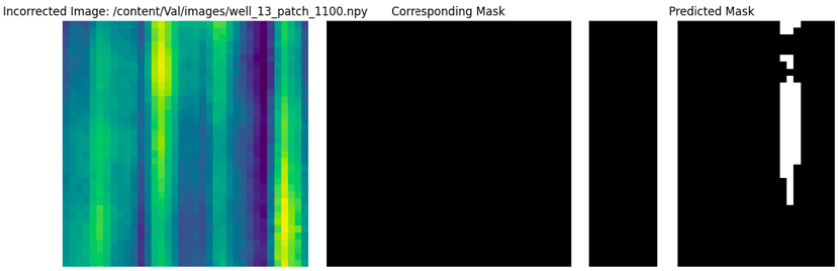


Figure 5: surplus de prédictions

Néanmoins ce seuillage, bien qu'efficace pour faire disparaître les prédictions indésirables du modèle, était à l'origine d'une certaine érosion des prédictions. A l'image de la figure 6, nos prédictions étaient souvent trop fines en comparaison de la réalité terrain. Ainsi, pour parer à ce phénomène, nous avons décidé d'appliquer une dilatation pour les pixels blancs de façon à élargir nos marques de corrosions. Pour cela nous avons fait appel à la fonction "binary_dilation" du package "skimage.morphology" avec comme paramètre disk(1). Cela signifie que pour chaque pixel de valeur 1, tous les pixels qui touchent ce pixel dans un rayon de un pixel seront considérés comme appartenant à la marque de corrosion. L'effet est illustré dans la figure 7.

Pour résumer, nous avons cherché à éliminer les prédictions erronées et ensuite à reconstruire celles correspondant aux vraies marques de corrosion.

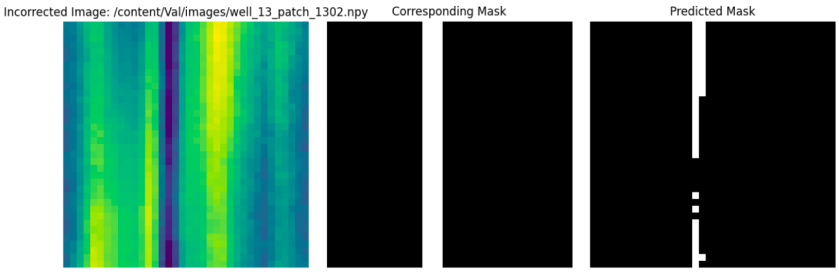


Figure 6: prédiction erodée du masque

Figure 7: Effet de la dilatation sur les masques

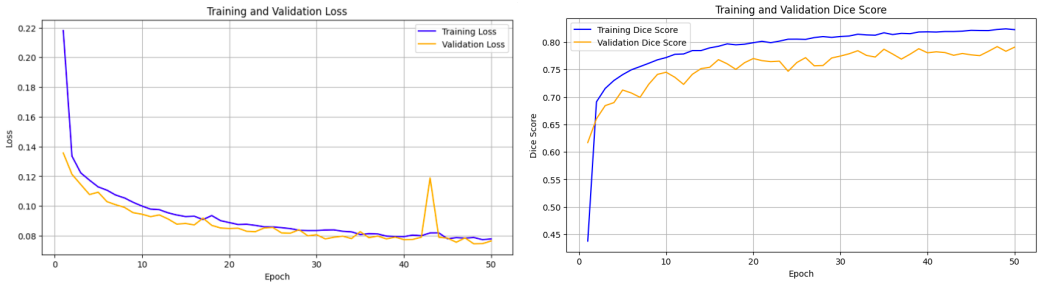
Remark 5.1 *En pratique, bien qu'efficace sur nos données, ces opérations se sont révélées pénalisantes pour l'évaluation du modèle sur les données de test : score de 0.35 pour un seuil à 0.80 et une dilatation de rayon 1 pixel contre un score de 0.81 sur nos donnée.*

6 Résultats

6.1 U-Net et Binary Cross Entropy Loss

Le premier modèle entraîné a donc été U-Net avec comme fonction de perte la Binary Cross Entropy. Dans les figures suivantes nous avons reporté les performances de ce modèle. Tout d'abord, sur la figure 8a, nous pouvons observer une décroissance exponentielle de la fonction de perte à la fois sur les données d'entraînement et sur celles de validation. Il semble qu'à partir de 20 époques, la fonction de perte commence à se stabiliser.

La figure 8b quant à elle montre le Dice score à la fois sur les données d'entraînement et de test. Encore une fois la convergence à une allure exponentielle et semble se stabiliser autour de 20 époques à des valeurs se situant autour de 0.80 sur les données d'entraînement et de 0.77 pour celles de test.



(a) Loss (b) Score

Figure 8: Le perte et le score de Dice pour l'ensemble d'entraînement et de validation

6.2 U-Net avec d'autres fonctions de perte

Nous avons également expérimenté U-Net avec d'autres fonctions de perte. Pour cela nous avons combiné la Focal Loss, présentée précédemment, avec la Binary Cross entropy. Nous avons également associé la perte de Dice avec la Binary Cross entropy. Les résulats sont présentés dans la figure 9. On peut notamment remarqué que ces versions hybrides, bien qu'efficaces, ne parviennent pas à atteindre les performances de la Binary Cross Entropy.

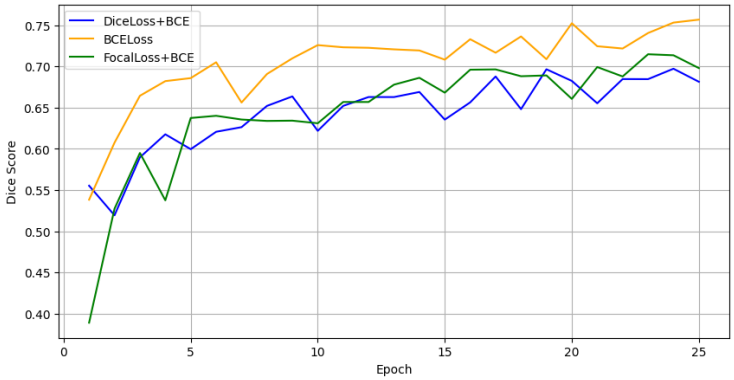


Figure 9: Résultats du score de Dice sur l'ensemble de validation pour l'entraînement de U-Net avec différentes pertes

6.3 U-Net avec mécanismes d'attention

Nous avons également entrâiné un modèle U-Net avec des Attention Gates. Nous montrons les performances de ce modèle dans la figure suivante 10 avec trois fonctions de perte différentes décrites précédemment.

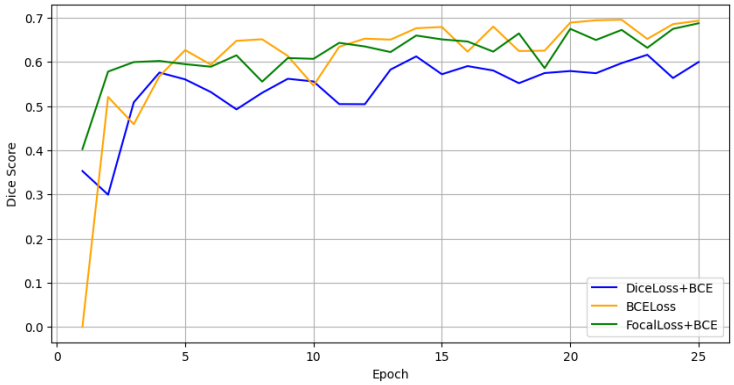


Figure 10: Résultats du score de Dice sur l'ensemble de validation pour l'entraînement de Attention U-Net avec différentes pertes

7 Travaux Futurs

Nous envisageons d'explorer différentes avenues pour améliorer les résultats obtenus. Parmi les approches à investiguer, il y a notamment l'amélioration potentielle des images et des masques en utilisant diverses techniques de traitement d'images.

De plus, il serait intéressant d'explorer la possibilité d'améliorer la segmentation en combinant les prédictions de plusieurs modèles entraînés (Ensemble Learning) .

Table 1: Comparison of Dice Score for Different Loss Functions

Dice Score									
	BCE			Focal Loss + BCE			Dice Loss + BCE		
Architecture	Train	Val	Test	Train	Val	Test	Train	Val	Test
U-NET	0.85	0.81	0.61	0.83	0.76	0.57	0.79	0.74	0.54
Att-Unet	0.81	0.76	0.50	0.72	0.68	-	0.73	0.68	-
Res-Unet	0.79	0.55	-	0.75	0.45	-	0.76	0.52	-

8 Conclusion

Finalement, nous nous positionnons troisième au classement académique avec un score de 0.589 avec un benchmark à 0.503 et le score de l'équipe gagnante à 0.658. Nous terminons également à la $5^{\text{ème}}$ position du classement général avec 0.61. Nous n'avons pas observé une baisse de score significative entre le classement public et celui privé témoignant ainsi d'une certaine robustesse de notre méthode. Quant à la méthode retenue, celle qui s'est avérée être la plus performante dans notre cas est également la plus élémentaire de celles testées, à savoir U-Net avec comme fonction de perte la Binary Cross Entropy.
Ce challenge nous aura permis de découvrir et d'appliquer de nombreuses méthodes de segmentation à un problème concret avec de fort enjeux sà la fois économiques et sociétaux.

Code source: `https://github.com/aissa-abdelaziz/Challenge-Data-CorroSeg.git`

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient nd image segmentation. *International journal of computer vision*, 70(2):109–131, 2006.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[4] Nan Ding, Hélène Urien, Florence Rossant, Jérémie Sublime, Paul Bastelica, and Michel Paques. Attention u-net pour la segmentation des pores de la lame criblée. In *GRETSI*, 2022.

[5] Peter Caccetta Chen Wu Foivos I. Diakogiannis, François Waldner. Resunet-a: a deep learning framework for semantic segmentation of remotely sensed data. *arXiv preprint arXiv:1904.00592*, 2019.

[6] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.