Détection et classification automatique des défauts de soudures dans une image thermique en se basant sur des méthodes d'apprentissage profond

Mendo Sébastien

Février 2024

Résumé: Ce document présente l'analyse et l'application de modèles d'apprentissage dans le cadre d'un projet de détection de défauts dans des images infrarouges de soudures. Nous ferons l'état de l'art des modèles existants pour la classification et la détection d'objets dans des images, puis nous effectuerons un travail de recherche sur le modèle à choisir et comment définir certains de ses paramètres. Enfin nous montrerons les résultats résumés dans une interface graphique afin de tester nos différents modèles sur les images que l'on veut et les comparer.

Mots-clés: modèles d'apprentissage, images thermiques, classification, détection.

1. Introduction

Ce projet s'inscrit dans un contexte industriel et de recherche piloté par Hamid Ladjal au laboratoire LIRIS, sur des images thermiques de soudures d'un type de sachet (Cf figure 1) pour le transport de produits. Il vise à concevoir un système capable d'agir en temps réel pour la détection de défauts dans des images thermiques sur la soudure de ces sachets grâce à des méthodes d'apprentissage profond. Par la suite, ce système classifiera le défaut trouvé dans une des 5 catégories possibles de défauts (cf. figure 1) qui représentent chacune potentiellement la cause de la mauvaise soudure, ou indiquera qu'aucun problème n'est présent. Cela permettra de corriger plus facilement l'équipement qui s'occupe du soudage. Un défaut de soudure pourrait être lié à une surchauffe de la machine, à l'usure d'un composant etc… On pourrait corréler un type de défaut avec un type de problème sur un équipement. Finalement, notre outil permettrait de faire un diagnostic de la machine.

Ce système de détection et de classification existe et est utilisé actuellement, l'objectif serait donc d'en concevoir un qui serait encore plus performant afin de détecter plus de défauts ou de mieux les classifier en utilisant différentes architectures d'apprentissage profond et de trouver les meilleurs paramètres sur le modèle choisi qui donnent les meilleurs résultats.

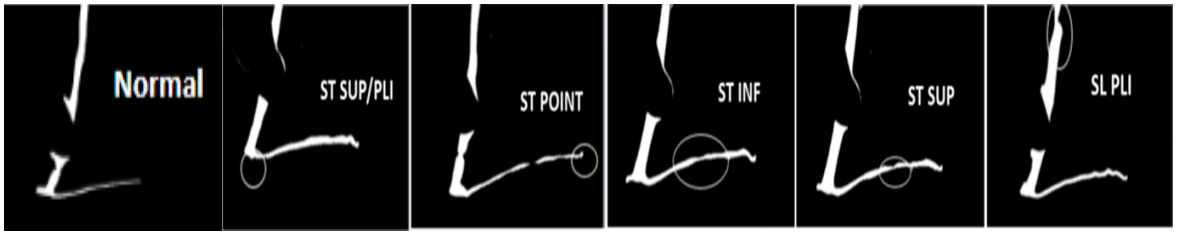


Figure 1 : Images thermiques de soudure avec et sans défaut.

2. Etat de l'art

2.1. modèle xception

Ce modèle de classification consiste en trois étapes. L'image va passer dans un bloc d'entrée, puis huit fois dans un bloc central et enfin une fois dans un bloc de sortie. Ces blocs contiennent une série de convolutions et de pooling (filtrage et réduction de la taille de l'image) permettant d'obtenir des images à traiter plus petites et contenant des caractéristiques plus intéressantes à étudier comme les contours de l'objet présent dans l'image. Développé par Google, ce modèle permet une grande précision sur la prédiction en effectuant la convolution sur chaque niveau du pixel un par un (filtre 1D sur R, G et B) au lieu de le faire en une fois comme la convolution classique (filtre 3D). L'inconvénient est que l'entraînement de ce modèle est plus long que d'autres dû à la quantité de traitement qu'il effectue sur l'image.

2.2. densenet

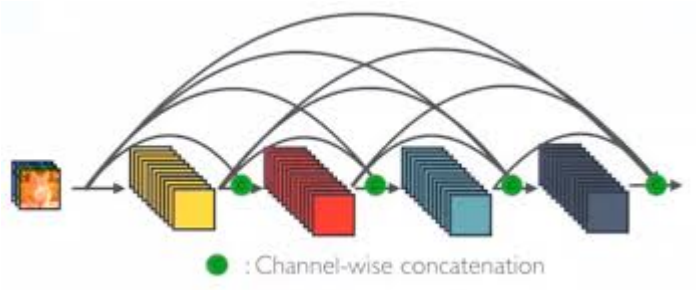


Figure 2 : Architecture DenseNet. [MRD]

Ce type d'architecture de classification là aussi vise à optimiser les performances lors de la rétropropagation c'est-à-dire de l'entraînement du réseau de neurone, mais aussi dans le nombre d'opérations nécessaires pour traiter l'image et donner un résultat. En effet, cette architecture consiste à réaliser une succession de pooling et de convolution, puis de donner le résultat à une autre couche de traitement concaténé aux sorties des couches précédentes (cf. figure 2). A chaque couche, les données reçues en entrées deviennent de plus en plus petites ce qui permet une rapidité de traitement et contiennent des données plus intéressantes car la couche finale reçoit une concaténation des sorties précédentes et connaît donc plusieurs caractéristiques différentes de l'image d'entrée en même temps (le bord d'un objet dans l'image, les éléments principaux des objets etc…).

2.3. ResNet

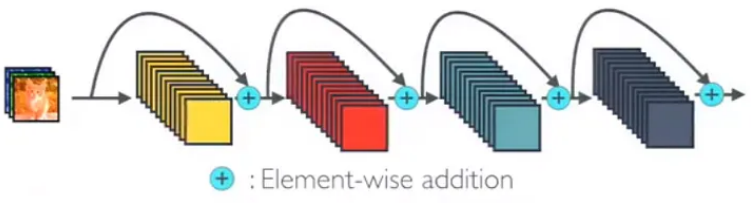


Figure 3 : Architecture ResNet. [MRD]

Ce modèle ressemble fortement au DenseNet et en effet ils sont similaires, la différence réside dans le fait que chaque couche ne reçoit pas toutes les sorties des couches qui le précède mais seulement celles des deux couches précédentes (cf. figure 3). Ce modèle est très performant sur

la classification lorsque ses paramètres sont bien choisis. La figure 4 en annexe [CMC] donne la comparaison de la précision et de la perte de différents modèles sur un jeu d'images. Nous pouvons voir que ResNet est plus performant que tous les autres modèles avec qui il est comparé, il atteint une plus grande précision plus rapidement et ne fait que peu d'erreurs sur la classification. Cette figure peut être différente selon la base d'image utilisée lors de l'entraînement mais de manière générale l'architecture ResNet donnera de meilleurs résultats.

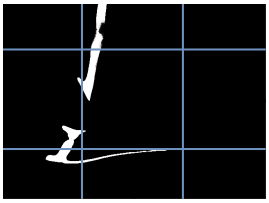
2.4. VGG

VGG, pour Visual Geometry Group, est un modèle de classification qui a été proposée avec des noyaux de convolutions plus petits que ce qui se faisait habituellement, il s'agit d'une architecture qui présente la capacité (avec de bonnes images d'entraînement) de ne faire que très peu d'erreurs. Ce fût un des premier modèle à utiliser de petits noyaux de convolution ce qui améliore les performances et cela fonctionne si bien que tous les modèles l'utilisent. Elle ne contient que peu de couches de traitements ce qui rend l'apprentissage rapide (cf. figure 5).

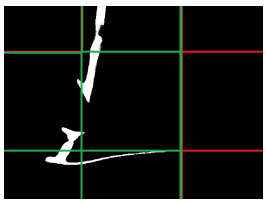
2.5. YOLO (You Only Look Once)

YOLO est une architecture qui s'occupe à la fois de la classification et de la détection d'objets, c'est-à-dire le fait d'indiquer l'emplacement de certains objets dans une image. Un autre modèle, R-CNN, permet lui aussi de faire de la détection mais YOLO ayant une plus grande vitesse d'exécution et puisque nous souhaitons un modèle capable d'agir en temps réel, nous nous concentrerons plutôt sur celui-ci.

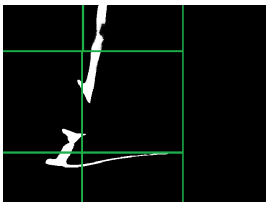
La détection avec YOLO comporte 4 étapes majeures:



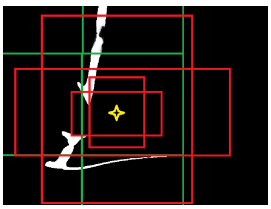
Tout d'abord, l'image est découpée en différentes cases.



L'algorithme va prédire pour chacune de ces cases la probabilité que tel ou tel objet soit présent.



Chaque case ou la probabilité de présence d'un objet est faible (inférieure à 0.5) sera enlevée de la suite de l'algorithme.



Ensuite, pour chaque centre trouvé grâce aux probabilités de chacune des cases, plusieurs "boîtes" prédéfinies vont être placées sur ce centre sur trois niveaux différents, c'est-à-dire après chaque couche de convolution pour essayer de délimiter notre objet. Une seule sera gardée selon la probabilité que l'objet soit à l'intérieur.

YOLO permet d'effectuer tous ces traitements en une fois, d'où "You Only Look Once". Grâce à cela, il peut fonctionner en temps réel à au moins 30 images par seconde. Cela permet également de n'avoir qu'une seule architecture pour faire la classification et la détection rendant le problème plus simple.

3. Outils utilisés

- Je me suis servi de Visual Studio Code pour l'édition de texte, ce dernier possède de multiples extensions pour chaque langage de programmation facilitant le développement.

- J'ai utilisé Python comme langage de programmation car la grande majorité des modèles d'apprentissage s'utilise via ce langage. Python possède des bibliothèques pour faire des interfaces graphiques tel que Pygame ce qui sera très utile pour afficher les résultats de ce projet. De plus, il est utilisé sur Google Colab, autre outil dont je me suis servi.

- En effet, mon ordinateur lors de l'apprentissage de modèles ne me permettait pas d'avoir un temps d'exécution convenable (plusieurs heures étaient nécessaires), ni d'entraîner ces modèles sur de larges bases d'images (RAM trop faible). Après avoir tenté d'utiliser mon GPU pour pallier à ces problèmes en vain, je me suis servi de Google Colab. Cet outil permet d'exécuter des scripts Python sur un serveur de Google contenant plus de RAM et un GPU facilement utilisable. La encore il a fallu être prudent puisque leur GPU est à accès limité, seul un certain temps d'exécution est accordé.

- Roboflow: Cet outil permet la labellisation et l'augmentation d'images pour le modèle YOLO.

4. Travail de recherche

4.1. Le choix du modèle

Le modèle choisi est le dernier présenté, YOLO version 8. En effet ce modèle présente plusieurs avantages: le premier étant que celui-ci effectue à la fois la classification et la détection d'objets ce que les autres modèles présentés ne font pas. YOLO comme dit précédemment fonctionne en temps réel et cela est nécessaire pour pouvoir détecter les défauts de soudure dans un cas pratique industriel. De plus il possède plusieurs modèles accessibles entraînés sur différents jeux d'images, cela permet de faire du transfer-learning, nous y reviendrons juste après. Afin d'obtenir les meilleurs configurations du modèle YOLO version 8 (la dernière version) nous allons l'entraîner de trois façons différentes et choisir la meilleure:

- Un modèle qui effectuera l'apprentissage avec tous ces poids initialisés au hasard. Cela signifie que le réseau de neurones n'a aucune connaissance au départ.

- Le deuxième lui partira d'un modèle déjà entraîné sur des millions d'images de la base COCO [UTD] tel que des voitures, personnes, objets etc… Cela devrait permettre un meilleur apprentissage car ce modèle possède des connaissances générales sur les objets du monde.

- Enfin le troisième sera utilisé pour faire du transfer-learning. Cette technique consiste à prendre un modèle déjà entraîné (ici on prendra le même modèle que pour le deuxième cas) et à figer une partie de ses poids (une partie des couches de neurones). De cette manière seul une partie d'entre eux vont être mis à jour lors de l'apprentissage et cela devrait donner le même effet que pour le deuxième cas mais de manière plus forte étant donné que cette fois-ci les connaissances vont êtres en partie sauvegardées et non écrasées, et plus rapide puisque seul une partie du modèle sera entraîné. On peut faire le parallèle avec un humain qui sait jouer de la guitare et qui voudrait apprendre le piano, il apprendra plus facilement et rapidement qu'une personne ne sachant jouer aucun instrument car il possède déjà des connaissances musicales.

4.2. Choix des paramètres

4.2.1. Le choix du nombre d'epoch

Les épochs correspondent au nombre d'itérations que le modèle effectuera pour apprendre à classifier et détecter les images. Arbitrairement j'ai essayé d'entraîner le premier des trois modèles avec 100 epochs puis avec 150 afin de voir lequel donnerait les meilleurs résultats:

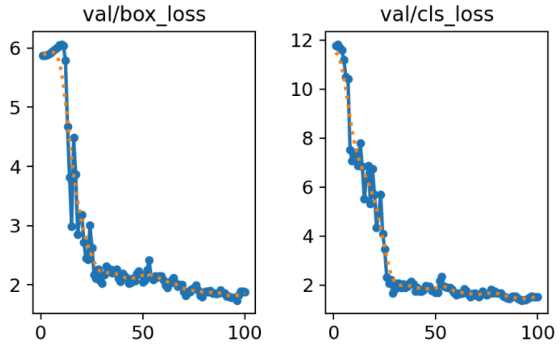


Figure 6 : Courbes de pertes (100 epochs).

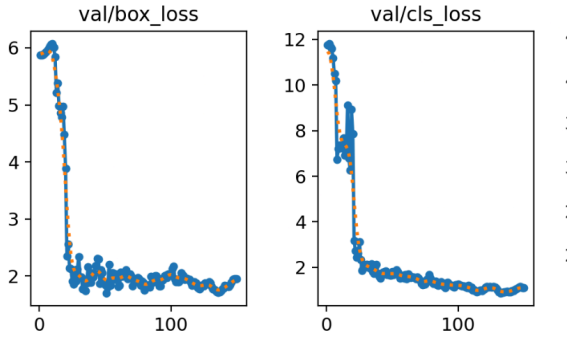


Figure 7 : Courbes de pertes (150 epochs).

Nous pouvons sur la figure 6 et 7 la perte des boîtes (ou "box_loss", à gauche de chaque figures), c'est-à-dire si une boîte (de détection) est plus ou moins bien placée autour d'un objet. A leur droite il s'agit de la perte liée à la classification ("cls_loss"), c'est-à-dire si ce qu'il y a dans les boîtes est bien classifié. Ici la différence est très légère. J'ai donc comparé d'autres courbes pour me décider.

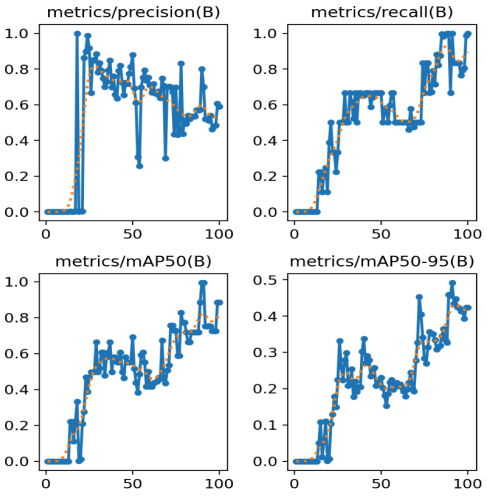


Figure 8 : Courbes de précisions (100 epochs).

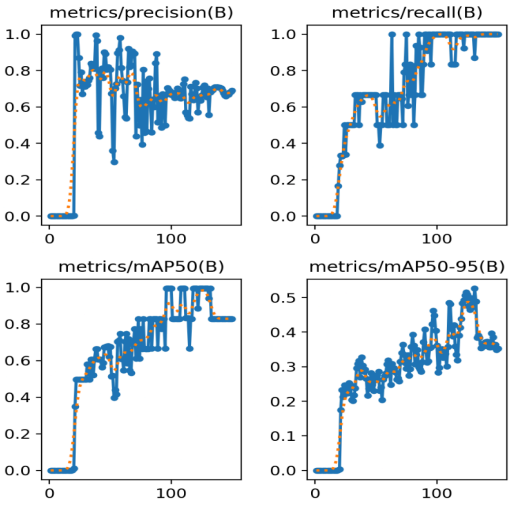


Figure 9 : Courbes de précision (150 epochs).

Nous avons sur la figure 8 et 9 la précision, le rappel, le mAP50 et mAP50-95. Sans trop entrer dans les détails, ces mesures indiquent la qualité de classification et de détection entre 0 et 1 (1 étant le meilleur). Nous voyons qu'avec 150 epochs les performances sont souvent les mêmes qu'avec 100 mais semble très légèrement meilleures. Puisque les différences sont très minimes et afin de limiter le temps d'entraînement, j'ai choisi de partir sur 100 itérations.

4.2.2. Différentes bases d'images

Afin de pouvoir utiliser YOLO, il a fallu labelliser sur chacune des images que mon encadrant m'a fourni le type de défaut et à quel endroit il se situe, ou si aucun défaut n'était présent. Pour cela j'ai utilisé l'outil Roboflow qui permet de labéliser facilement les images (cf. figure 15) et permet également de faire de l'augmentation de données, c'est-à-dire la déformation d'image tel que la rotation, l'étirement, le recadrage etc… (cf. figure 16). Cela permet au modèle de généraliser son apprentissage sur ce qu'est un objet (peu importe s'il est à l'envers ou déformé) et saura donc mieux le reconnaître. Premièrement les images ont été augmentées avec seulement l'étirement horizontal et vertical. Cela m'a permis d'en d'obtenir 189 au lieu des 70 que l'on m'a confiées au départ.

Ensuite j'ai répété l'opération en appliquant l'étirement, la rotation et le recadrage des images. Grâce à cela nous obtenons 1056 images ce qui devrait aider à faire de la généralisation et éviter au modèle de faire du surapprentissage, c'est-à-dire d'apprendre par cœur les données et de ne pas savoir prédire correctement de nouvelles images jamais vues auparavant.

Pour comparer les performances selon la quantité d'images, nous allons utiliser les deux bases d'entraînement de 189 puis de 1056 images et entraîner un même modèle dessus. Nous allons comparer ces résultats grâce à des matrices de confusion, ces graphiques indiquent pour chaque classe de notre problème (chaque type de défaut de soudure) ce que le modèle à prédit sur les images de validation (des images jamais vues par le modèle).

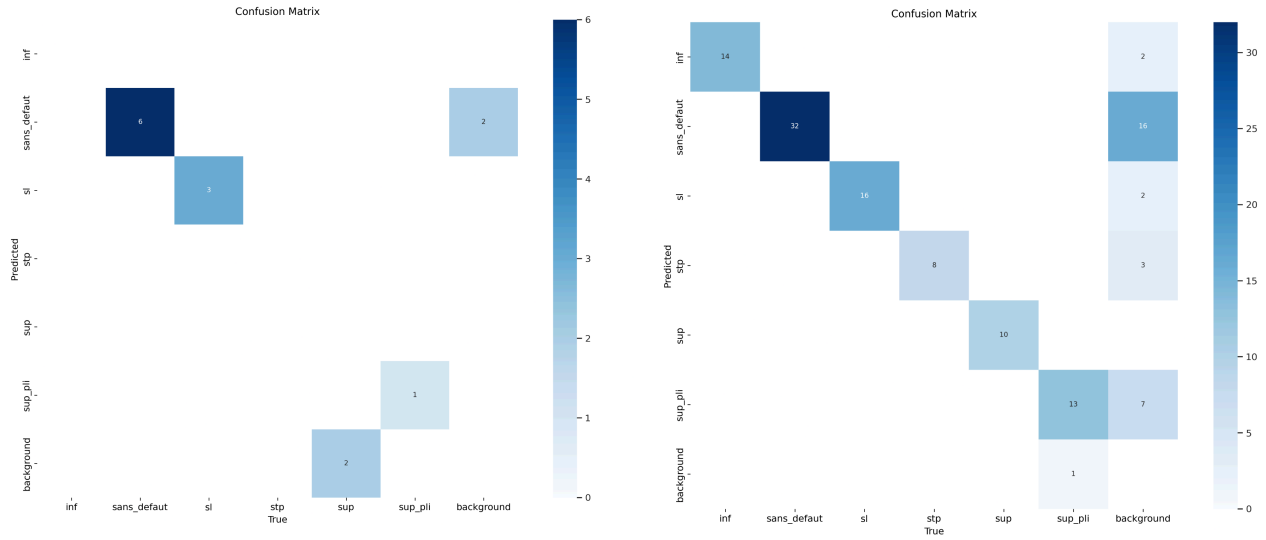


Figure 10 : Matrices de confusions de d'un modèle YOLO entraîné sur 189 images (à gauche) et 1056 images (à droite).

Ici le modèle entraîné est celui dont les paramètres sont aléatoires et ne possède donc aucune connaissance avant son apprentissage. Sur la matrice de gauche, nous voyons que deux classes sont mal prédites, de même sur la matrice de droite. Cependant le modèle de droite est en fait meilleur, en effet la base de validation pour la matrice de droite est bien plus grande que celle de gauche. Proportionnellement au nombre d'images, le modèle de gauche à donc fait plus d'erreurs. Nous allons donc préférer utiliser un modèle entraîné sur beaucoup que peu d'images. Nous verrons plus tard que cela se confirme sur l'interface créée.

De plus, nous pouvons dire que l'augmentation des images étant légèrement plus présente sur la base de 1056 images, cela participe également à l'efficacité du modèle.

4.3. Résultats

Figure 11 : Menu principal de l'outil de comparaison de modèles développé en Python.

Cette capture d'écran représente le menu de notre interface, différentes options sont possibles:

- On peut choisir une image grâce au bouton "Select file" afin de classifier et faire de la détection dans une image (cf. figure 12).
- Le bouton "Show precision" permet d'afficher la courbe de précision de chaque modèle (cf. figure 17)
- Le bouton "Show confusion" affiche les matrices de confusion (cf. figure 18)
- Le bouton show loss lui affiche les courbes de perte (cf. figure 19)

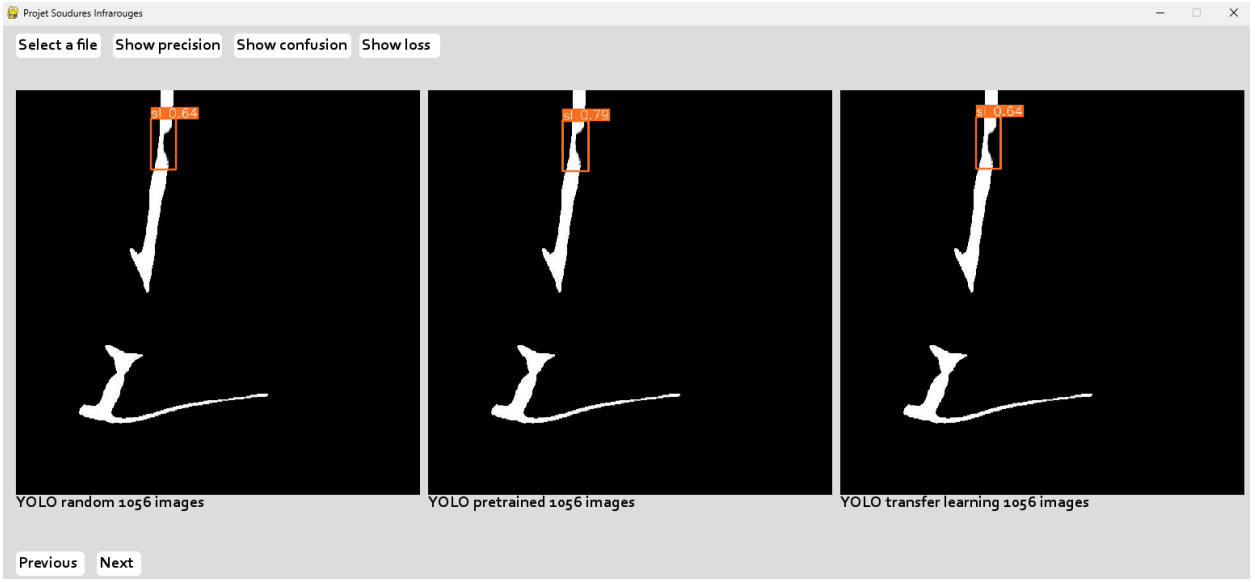


Figure 12 : Prédictions des trois modèles entraînés sur 1056 images.

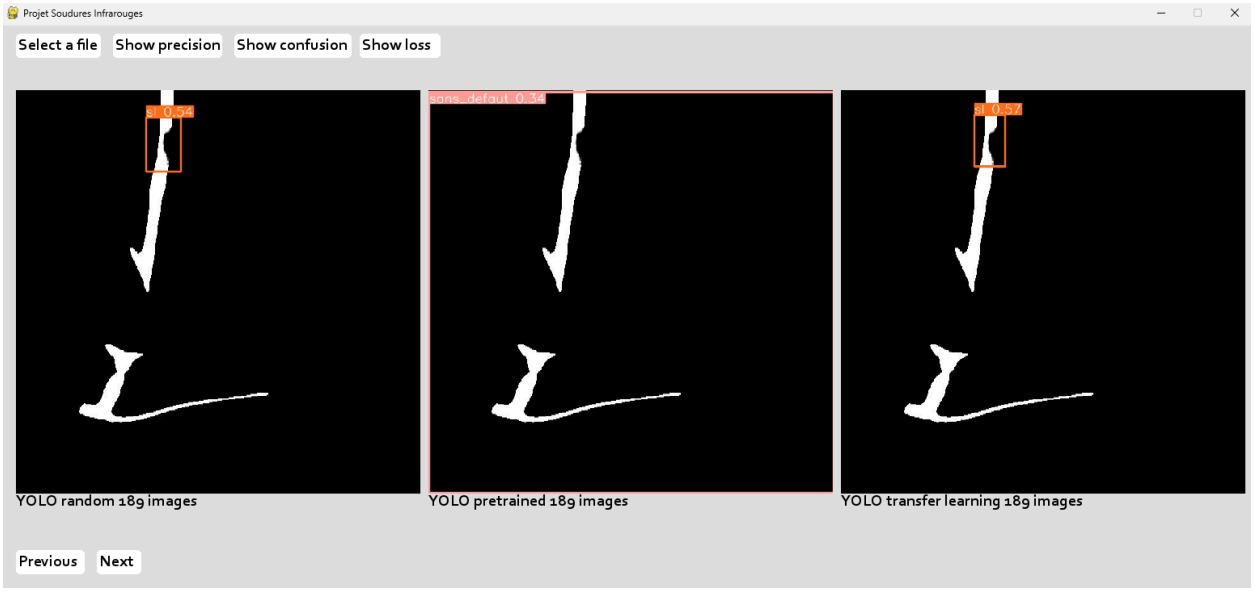


Figure 13 : Prédictions des trois modèles entraînés sur 189 images.

Une fois notre image sélectionnée via l'explorateur de fichier, nous avons trois images qui s'affichent qui correspondent respectivement de gauche à droite aux prédictions de nos trois modèles (cf. figure 12) à savoir: un non entraîné auparavant, un pré entraîné et un de transfer-learning. En bas de ces images le nom du modèle et le nombre d'images sur lequel il s'est entraîné est affiché. Nous retrouvons également deux boutons en bas à gauche: "Previous" et "Next". "Next" permet de passer à la deuxième page qui contient cette fois-ci les prédictions des mêmes modèles mais entraînés sur la base contenant 189 images (cf. figure 13).

Nous pouvons voir sur la figure 13 que l'entraînement sur 189 images ne donne pas de très bon résultats, en effet le modèle "YOLO pretrained" "indique qu'aucun défaut n'est présent dans l'image hors il y en a bien un de type "SL" que les modèles nommé "YOLO random" et "YOLO transfer learning" ont su trouver. Ce modèle devrait pourtant être meilleur que "YOLO random" puisque le réseau de neurones possède à la base des connaissances ce qui est souvent plus précis. Seulement avec une si petite base d'image cela ne semble pas être toujours le cas.

Si nous regardons maintenant la figure 12, cette-fois ci chaque prédiction est correcte, cela montre la grande efficacité d'un entraînement sur beaucoup d'images. De plus, nous remarquons que le modèle "YOLO pretrained" donne une probabilité de présence d'un défaut plus grande (ici égale à 0.79) contre 0.64 pour "YOLO random". Un modèle ayant des connaissances (étant déjà entraîné au préalable) semble donc être meilleur.

Néanmoins un premier problème apparaît, le modèle "YOLO transfer learning" indique figure 12 une même probabilité que le modèle ayant des poids aléatoires, hors nous venons de dire qu'un modèle ayant des connaissances devrait être meilleur. Le problème ici est que lors du transfer learning, une partie des couches de neurones ont été fixées arbitrairement, ici la moitié n'ont pas étés modifiées lors de l'entraînement. Par souci de capacité de calcul, je n'ai pas pu tester avec plus ou moins de couches fixées afin de déterminer par exemple si fixer 90% des couches donnerait de meilleurs résultats ou si au contraire il faudrait en fixer que 20%. Cependant ce problème n'apparaît pas à chaque fois.

Un autre problème récurrent apparaît, il s'agit de la faible probabilité des prédictions comme sur cette exemple:

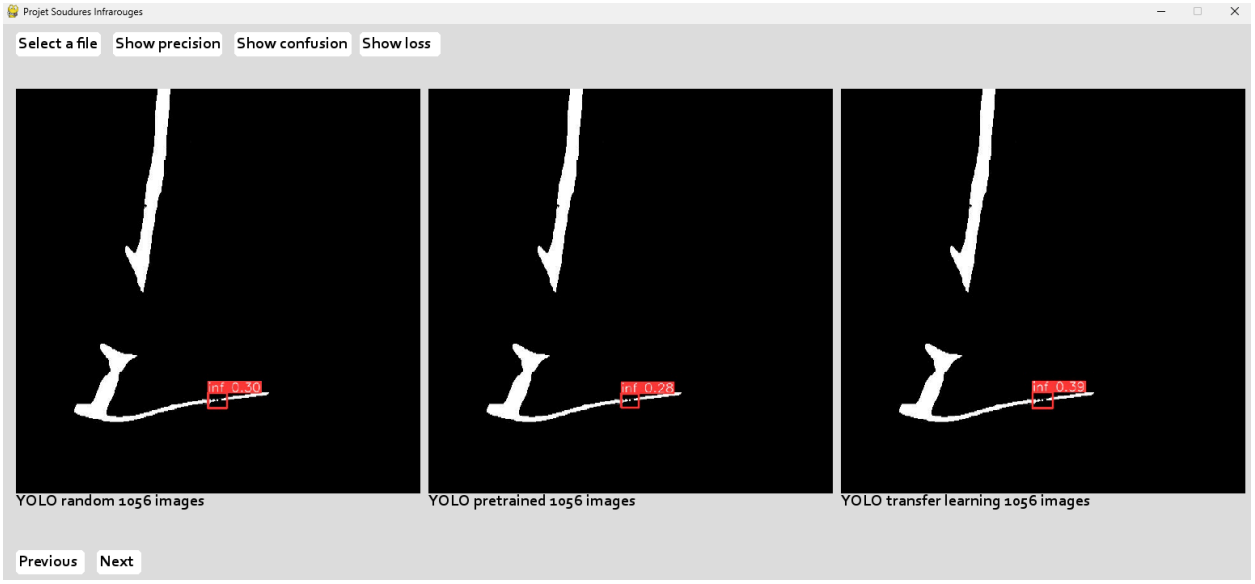


Figure 14 : Prédictions sur une autre image des trois modèles entraînés sur 1056 images.

Ici les prédictions sont correctes pour chaque modèle mais ne sont respectivement de gauche à droite que de 30%, 28% et 39%. Ces modèles ne sont pas très certains de ce que contient l'image. Dans un cas pratique, le sachet soudé pourrait être identifié comme bien soudé et mis en vente même s'il contient en réalité un ou plusieurs défauts, il pourrait être un faux négatif. Prenons le premier modèle figure 14 comme exemple, si une autre détection de probabilité supérieure à 0.30 existait, elle s'afficherait à la place de celle-ci hors elle ne serait pas forcément juste. Entre une prédiction de probabilité 0.30 et 0.32 par exemple, il est difficile de dire laquelle est juste puisque la marge d'erreur est fine.

En conclusion, ces résultats montrent en pratique que l'entraînement sur beaucoup d'images donne de meilleures performances. Cela indique de plus que les modèles ayant étés entraînées sur des images de toute sorte auparavant donnent bien de meilleur résultats qu'un modèle n'ayant aucune connaissance au préalable. Il faut tout de même faire attention car ces modèles n'étant pas très certains de leur classification, peuvent mal prédire une image de temps en temps.

5. Conclusion

Ce projet pose une base fonctionnelle permettant de répondre au problème de classification et de détection de défauts de soudures. Le modèle qui semble être le meilleur est YOLO version 8 pré-entraîné sur la base COCO puis entraîné sur 100 itérations sur 1056 images de soudure augmentées avec l'étirement, la rotation et le recadrage. Il a montré une plutôt grande efficacité en sachant détecter dans la grande majorité des cas le bon défaut de soudure au bon endroit. Cependant le défaut principal est la qualité de la prédiction. En effet, celle-ci est correcte mais de faible probabilité dans la plupart des cas. Ce que nous voudrions est un modèle permettant de dire de manière plus certaine qu'un défaut est présent ou non. Cela permettrait d'éviter des erreurs de prédictions et donc de pouvoir corriger la machine s'occupant des soudures plus efficacement et seulement si nécessaire. J'ai pu apporter grâce à ce projet de recherche des éléments de réponse sur la manière de construire un bon modèle d'apprentissage et sur la façon de continuer ce projet afin d'obtenir des résultats on l'espère meilleurs.

Une suite à ce projet pourrait consister à entraîner le modèle YOLOv8 utilisant le transfer-learning sur beaucoup plus d'images, par exemple quelques milliers ou dizaine de milliers. Cela n'a pas pu être possible dû à la capacité de calcul disponible pour ce projet et semble être

une bonne piste pour corriger les faibles probabilités des prédictions. Les différences montrées précédemment sur les deux bases d'images étaient minimes mais il y avait seulement 850 images de plus sur la deuxième base pour essayer de montrer une différence nette et cela reste relativement peu même si les résultats restent meilleurs.

Deuxièmement nous pourrions tester différents apprentissages du modèle de transfer learning en fixant à chaque fois un nombre différent de couches de neurones afin de voir quelle configuration est la meilleure.

Enfin, une autre piste consisterait en l'apprentissage d'autres modèles uniquement dédiés à la classification comme ceux présentés lors de l'état de l'art, dans l'espoir qu'un autre modèle ait une plus grande précision que le modèle de classification YOLO pour déterminer s'il y a un défaut dans une image de soudure ou non. La détection est utile pour savoir ou est le problème dans la soudure mais dans la grande majorité des cas ces défauts sont largement visibles à l'œil nu et sont très souvent au même endroit. Il suffirait donc de prédire quel type de défaut est présent dans l'image et nous verrions nous même d'où vient le problème. Cette solution permettrait de pouvoir tester de multiples classifieurs et peut-être gagner en précision sur la classification mais si un défaut est présent et est très peu visible, un œil humain pourrait avoir du mal à localiser le problème et pourrait ne pas savoir quoi corriger sur la machine en conséquence.

6. Références

[UTD] Modèle de détection YOLOv8 pré-entrainé https://docs.ultralytics.com/fr/tasks/detect/
[CMC] Comparaison entre différents modèles de classification
https://www.researchgate.net/figure/Comparison-of-training-results-between-ResNet-50-and-other-CNN-models-a-training_fig2_363422694
[MRD] Modèles ResNet et DenseNet
https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

7. Annexes

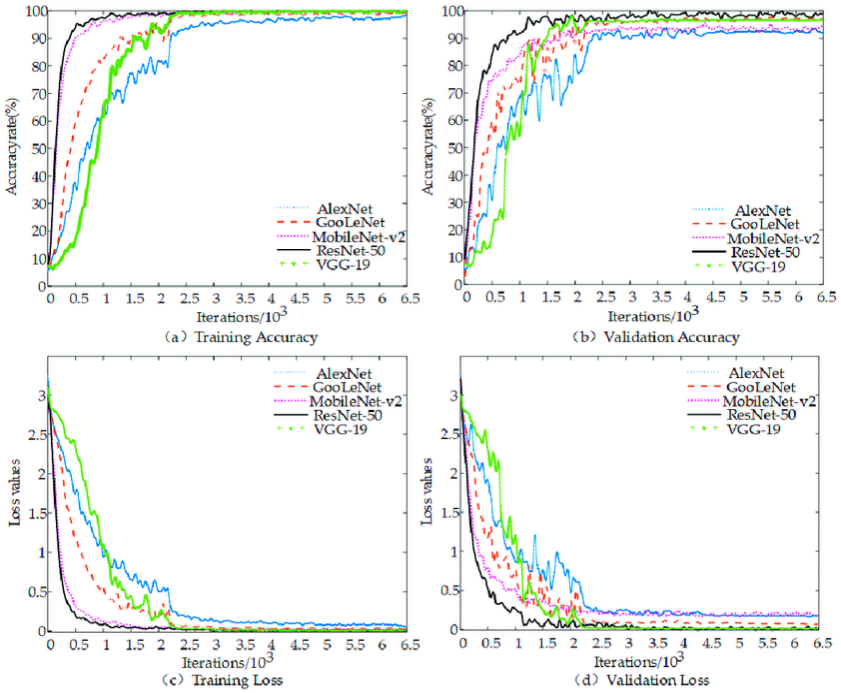


Figure 4 : Courbes de précision et de pertes de différents modèles lors de l'apprentissage.

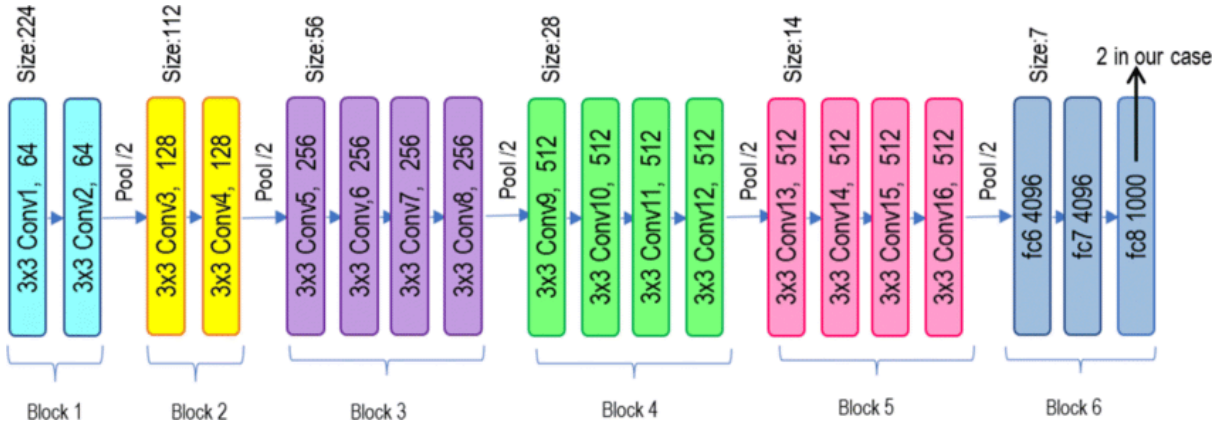


Figure 5 : Architecture VGG-19.

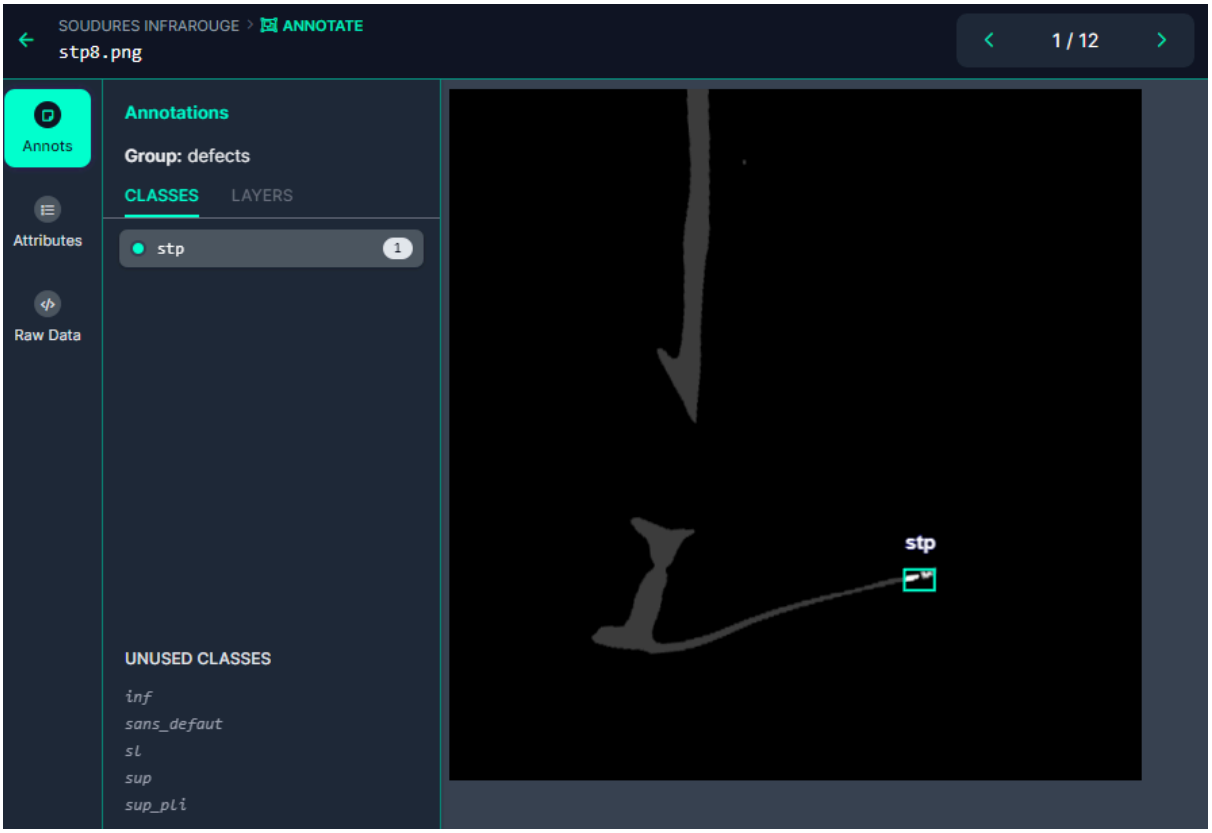


Figure 15 : Labellisation d'une image de soudure contenant un défaut de type "stp" via l'outil Roboflow.

Figure 16 : Augmentation d'une image

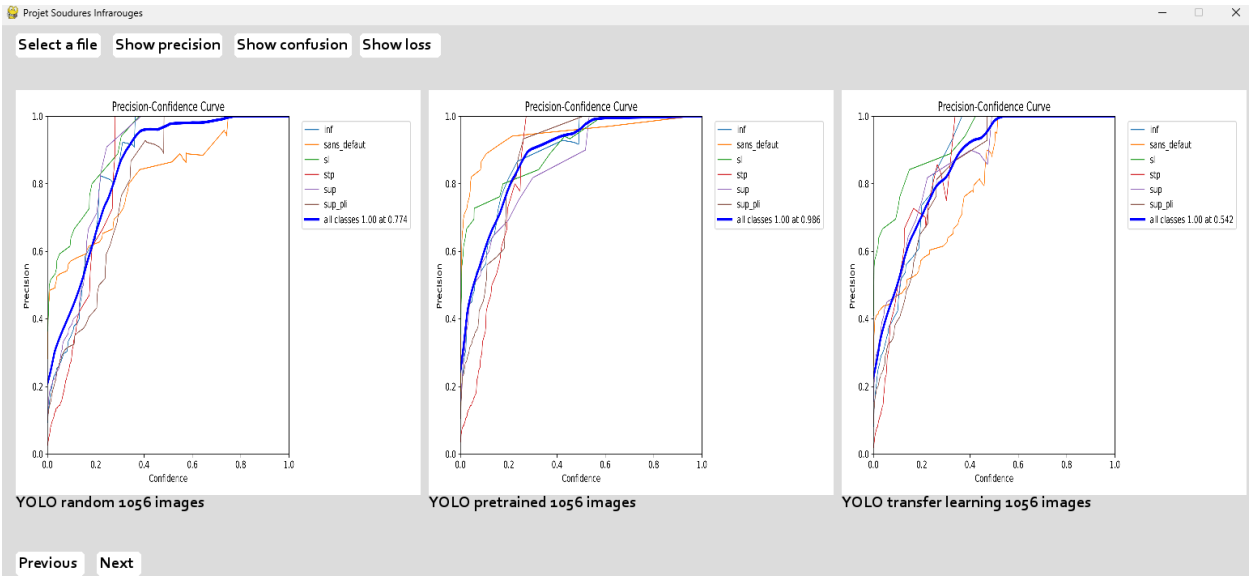


Figure 17 : Courbes de précision des modèles entraînés sur 1056 images

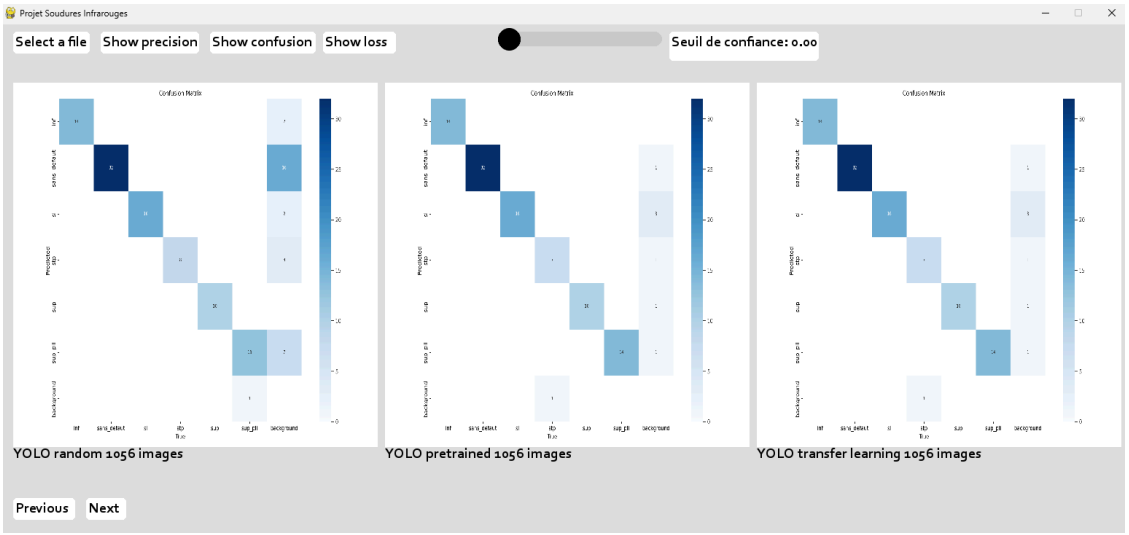


Figure 18 : Matrices de confusion des modèles entraînés sur 1056 images

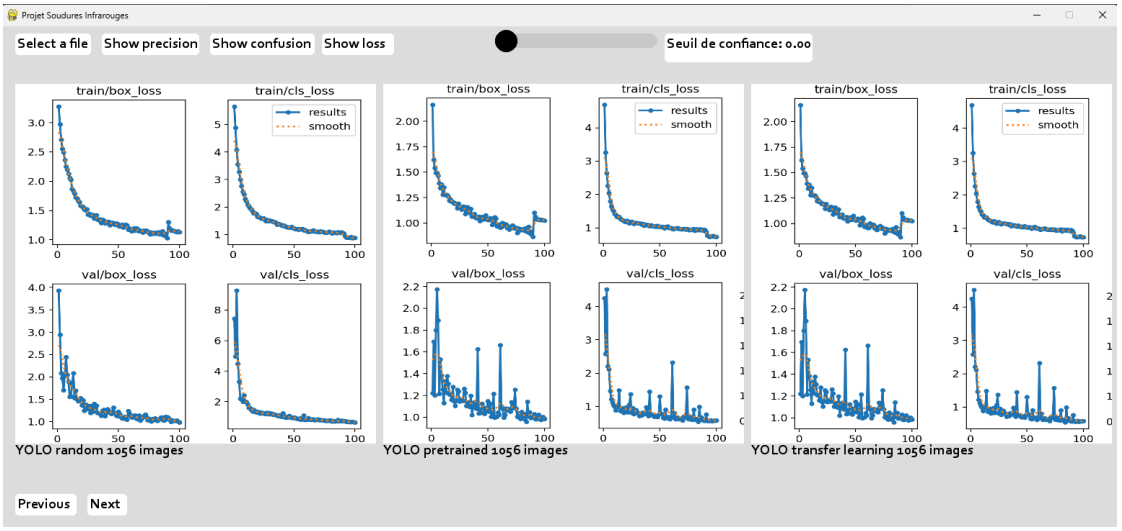


Figure 19 : Courbes de pertes des modèles entraînés sur 1056 images