

Réimplémentation Pytorch d'un réseau profond convolutionnel pour générer des images à grande étendue dynamique

Philippe Marcotte et Ulric Villeneuve

25 janvier 2018

Résumé

Dans cet article nous présentons une réimplémentation de l'article *Deep High Dynamic Range Imaging of Dynamic Scenes* [1]. Ils y présentent une nouvelle méthode de générer une image HDR à partir de trois images LDR avec un certain mouvement entre les photos. Nous décrirons donc le pré-traitement qui était déjà fait sur l'ensemble de données, l'étape pré-convolution constitué de l'alignement, l'augmentation et la décomposition, la fusion, les différentes architectures implémentées, soient directe, *Weight Estimator* et *Weight and Image Estimator*, l'ensemble de données, l'implémentation, suivi des résultats et d'une discussion.

1 Introduction

Générer des images *High Dynamic Range* (HDR) à l'aide de plusieurs images *Low Dynamic Range* (LDR) est une opération utilisée par beaucoup. D'autres techniques existent mais celle-ci sont généralement beaucoup plus complexe à mettre en place et donc bien moins accessible [2].

La génération d'image HDR à partir d'un ensemble d'images de différente exposition pour des scènes statiques est rendu à un point assez avancé. Cependant, il est rare de pouvoir prendre plusieurs images d'affilées sans avoir de mouvement entre elles. Les techniques pour générer des images HDR alors qu'il y a du mouvement ne sont pas encore aussi efficaces. Les auteurs de *Deep High Dynamic Range Imaging of Dynamic Scenes* [1] propose une nouvelle méthode

d'effectuer le fusionnement de plusieurs images. Ils utilisent un réseau neuronnal pour modéliser le processus. Ils ont démontré que leur méthode produit des résultats de haute qualité même meilleur que les méthodes de fine pointe.

Le but de notre projet est de réimplémenter leur méthode. Celle-ci est originalement faite avec Matlab. Nous proposons de la refaire en Python.

La prochaine section présente la méthodologie utilisée pour s'assurer de l'exactitude de notre implémentation. Puis, la section 3 décrit les transformations appliquées aux images partant du format RAW jusqu'au passage de la première phase de l'algorithme. La section suivante (4) discute de comment l'alignement des images LDR est effectuée, comment l'ensemble de données de base est augmenté et décomposé en image de plus petite taille. La section 5 décrit le réseau de convolution utilisé pour modéliser ce processus. Par la suite, la section 6 entreprend la description des trois architectures utilisées pour tester en profond le réseau convolutionnel. Suivant cette dernière section, la section 7 discute de l'ensemble de données d'entraînement et de test. La section 8 décrit l'implémentation Python utilisée. Puis, la section 9 discute des résultats obtenues avec les différentes architectures. On y compare aussi nos résultats à ceux des auteurs. Enfin, nous décrivons les problèmes et limitation rencontrés dans la section 10.

2 Méthodologie

Nous avons implémenté le prétraitement et le réseau convolutionnel simultanément, de sorte à ce que

lorsque le prétraitement est terminé, le réseau soit plus ou moins prêt à être entraîné. Une fois que ce fut le cas, nous avons exécuté chacune des étapes du prétraitement en parallèle avec l'implémentation Matlab avec les mêmes données en entrée au moins une fois afin de s'assurer de la validité de nos résultats, car on ne voulait pas perdre deux jours à entraîner le réseau avec des données faussées.

Par la suite, lors de la phase d'entraînement, nous surveillons le réseau avec des validations plus fréquentes afin d'être capable de détecter rapidement les problèmes liés à l'implémentation. ainsi nous avons pu détecter des erreurs qui faisaient en sorte que le réseau convergeait vers une valeur supérieur de PSNR que celle obtenue dans l'article très tôt dans le processus. Finalement, une fois que nos résultats faisaient du sens après quelques validations, nous l'avons laissé rouler plus librement, jusqu'à ce qu'il termine les 2 millions d'itérations.

3 Pré-traitement

Bien que les images fournit par les auteurs ont déjà subit le pré-traitement, il est important de comprendre les transformations qui ont été appliquées. Le pré-traitement prend en entrée des images en format RAW. Ce format est la représentation avec le moins de traitement possible qu'une caméra peut fournir. À ce point les images sont de dimensions 5760×3840 . Le pré-traitement consiste à premièrement à dématricier (*demosaicing*) l'image. Les images encodées selon un filtre de Bayer sont convertis à une image dite *true-color*, c'est-à-dire le format RGB [3]. Cette dernière étape linéarise les images. L'amplitude des images étant très grande, dans le domaine High Dynamic Range (HDR), une courbe gamma de valeur 2.2 est appliquée pour compresser l'amplitude à une gamme dite Low Dynamic Range (LDR). La courbe gamma s'applique ainsi [4, 5] :

$$\text{gammaRGB} = \text{linearRGB}^{\frac{1}{\gamma}} \quad (1)$$

Si l'on isole linearRGB dans l'équation 1, on peut passer du domaine LDR au domaine HDR.

$$\text{linearRGB} = \text{gammaRGB}^{\gamma} \quad (2)$$

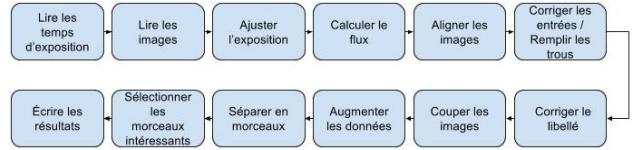


FIGURE 1 – Pipeline de pré-traitement

Le domaine LDR utilisé correspond à l'espace de couleur sRGB [6]. Cet espace de couleur est un standard pour afficher des images sur des écrans à petite gamme dynamique [7]. Les valeurs de sRGB sont exprimées entre 0 et 1.

Enfin, les images sont tronquées pour avoir les dimensions 1500×1000 .

4 Pré-convolution

4.1 Alignement

L'alignement consiste à modifier la première et la troisième image du triplet LDR afin qu'ils aient l'apparence de la deuxième image. Par exemple, si nous



FIGURE 2 – Images de basse et moyenne exposition.

prenons les figures 2(a) et 2(b). Nous allons alors tenter de modifier la figure 2(a) pour que l'homme soit accoté contre la roche. Pour se faire, il faut évidemment commencer par lire les trois images LDR ainsi que leur temps d'exposition. On peut noter que nous calculons les flux optique du couple formé de l'image de basse exposition et de moyenne exposition ainsi que celui formé de l'image de moyenne exposition et celle de haute exposition. Ensuite, nous devons ajuster le temps d'exposition des images. Nous ajustons ces temps d'exposition de sorte à ce que

les deux images aient l'exposition la plus haute des deux images. Cela a pour but de garder les caractéristiques des images mais avec un temps d'exposition plus élevé. Cet ajustement ce fait de la manière suivante :

$$Z_{i,j} = Z_i \Delta_{i,j}^{\frac{1}{\gamma}}, \text{ où } \Delta_{i,j} = \frac{t_j}{t_i} \quad (3)$$

où $Z_{i,j}$ est la valeur du pixel de l'image i ajusté à l'exposition de l'image j, où i et j sont deux images de différentes expositions et où l'image j a un temps d'exposition plus élevé.

Ensuite, à partir de ces images au temps d'exposition ajusté, nous calculons le flux optique à l'aide de l'algorithme de CeLiU. Nous avons trouvé une implémentation python utilisant les mêmes fichiers sources que celui de l'article [1]. Le flux optique est un effet qui est dû au déplacement relatif de l'image par rapport à la caméra [8]. Le flux optique obtenu sert à reconstruire la deuxième image à partir de la première, et la même chose à partir de la troisième image. Le flux optique nous indique à quel point un pixel s'est déplacé d'une image à l'autre. Donc à l'aide de ce flux il est possible d'interpoler les valeurs de pixels aux endroits où il y a eu du mouvement. Pour reprendre l'exemple utilisé plus tôt, figures 2(a) et 2(b), alors, le flux optique nous permet d'interpoler des valeurs de pixels de la première image si l'on accotait l'homme contre la roche. L'interpolation est fait par spline de troisième degré.

Par contre, lors de cette interpolation, il est possible que certaines valeurs soient impossibles à récupérer. Dans notre exemple, ces pixels sont ceux qui se trouvaient à l'endroit où était l'homme avant de se déplacer. Dans ce cas, l'article [1] mentionne que la manière utilisée pour reconstruire ces trous est quelque peu naïve et pourrait être améliorer, par exemple en utilisant un méthode *patch-based*. En effet, la méthode utilisée ne fait que prendre l'image de référence et ajuster l'exposition des pixels en question selon l'exposition recherchée de la même manière que démontré précédemment avec l'équation 3. Pour terminer cette étape d'alignement, nous corrigeons le libellé. C'est-à-dire que, aux endroits où il n'y avait pas de correspondances permettant d'interpoler, nous ajustons le libellé en prenant les valeurs de l'image de moyenne exposition et en passant au domaine HDR.

Nous appliquons ensuite la même transformation à chacunes de nos trois images alignées afin d'obtenir nos valeurs selon le domaine HDR.

4.2 Augmentation

Une fois que les images de chaque scènes sont alignées, nous rejetons les bordures de celles-ci puisque comme il y du mouvement dans notre scène, il est fort probable que les valeurs des bordures soient impossibles à interpoler. Les images que nous manipulons sont donc maintenant de 900 pixels par 1400 pixels. Puis, nous procédons à une phase d'augmentation de nos données. Nous appliquons des transformations géométriques aux scènes que nous avons lues afin d'augmenter notre ensemble de données ce qui nous permet d'éviter le surapprentissage [9]. Nous considérons les quatre cas de rotation de 90 degrés, les six combinaisons possibles pour les canaux de couleur ainsi que les deux cas possibles de miroir (valeurs inversées selon la gauche et la droite ou selon le haut et le bas), ce qui nous donne un total de 48 augmentations. Nous en appliquons dix de manière aléatoire par scènes de sorte à passer de 74 scènes à 740.

4.3 Décomposition en patch

Une fois les images alignées et les données augmentées, nous nous retrouvons avec trois images LDR, trois images HDR ainsi que le libellé. Comme mentionné plus haut, la taille de ces images est de 900 pixels par 1400 pixels par 3 canaux de couleur. Il s'agit d'une taille un peu grande pour l'entraînement du réseau, donc nous découpons ces images en morceaux de 40 par 40 pixels, en utilisant un pas de 20 pixels, ce qui nous donne 44 morceaux de larges sur 900 pixels et 69 morceaux de haut sur 1400 pixels. Cela augmente donc l'efficacité du réseau. Finalement, une fois tous les morceaux obtenus, nous voulons garder que ceux qui nous intéressent. Nous considérons un seuil de 50%, donc si le morceau contient au moins 50% de pixels possiblement sous-saturés ou sur-saturés, alors on le garde, sinon, on l'oublie. On considère que le pixel est possiblement saturé si sa valeur se trouve sous 0.2 ou au dessus de 0.8.

5 Fusion

Les auteurs décrivent le processus de fusion de la manière suivante : "Intuitively, this process requires estimating the quality of the input aligned HDR images and combining them based on their quality." En effet, l'intention est d'obtenir une image HDR où les défauts d'alignement, le bruit et la saturation n'apparaissent pas.

5.1 Fusion par convolution

Kalantari, Nima Khademi et Ramamoorthi, Ravi présente une nouvelle méthode d'effectuer le fusionnement des images LDR. Ils utilisent un réseau profond convolutionnel pour modéliser cette tâche. Ce réseau se compose de quatre couches comme présenté à la figure 3. Une première couche convolutionnelle utilisant un filtre de 7 par 7 pixels, suivit d'une deuxième avec un filtre de 5 par 5 pixels, puis une autre de 3 par 3 pixels et enfin, une dernière avec un filtre de 1 pixel. Les trois premières couches sont jumelées avec une couche d'activation composée d'unité de rectification linéaire (ReLU). ReLU est défini de la manière suivante [10] :

$$ReLU(x) = \max(0, x) \quad (4)$$

La sortie de la dernière couche, quant à elle, est normalisée entre 0 et 1 par une couche Sigmoid définie ainsi [11] :

$$S(x) = \frac{e^x}{e^x + 1} \quad (5)$$

Le réseau prend en entrée une matrice de $[Nx18xHxW]$ dimensions où N correspond à la taille d'une batch, H à la hauteur et W à la largeur des images. Les 9 premiers canaux correspondent aux images LDR et les 9 derniers aux images dans le domaine HDR. Les auteurs testent ce modèle au travers de 3 architectures décrites dans la section 6.

Le réseau convolutionnel a pour objectif de minimiser la perte calculée par la distance l2 entre l'image HDR résultante et celle de référence. La distance l2

est utilisée pour déterminer cette perte :

$$E = \sum_{k=1}^3 (\hat{T}_k - T_k)^2 \quad (6)$$

T représente les images une fois *tonemapped*. En effet, les auteurs ont jugé qu'il était préférable de déterminé la perte des images une fois *tonemapped*. Bien qu'il est possible de simplement compresser l'amplitude d'un signal HDR dans un interval LDR, les résultats peuvent être décevants. La compression fait perdre beaucoup des nuances que le signal original présente. *Tonemapping* est un processus pour transformer une image HDR tout en tentant de conserver une partie des nuances du signal original. Il existe plusieurs algorithmes pour *tonemap* une image [12]. Cependant, les auteurs font remarquer que les algorithmes de *tone mapping* ne sont en général pas différentiable. Or, cela cause problème pour un réseau profond. En effet, un réseau neuronal atteint son objectif en ajustant ses paramètres. Ces paramètres sont ajustés par *backpropagation* [13]. Pour effectuer ce processus, il faut que chaque étape jusqu'au calcul de la perte soit différentiable. Ainsi, pour *tone mapped* les images, les auteurs proposent d'utiliser la loi μ qui en fait ne correspond qu'à une compression de l'étendue du signal HDR. Elle est définie ainsi [14] :

$$F(x) = \frac{\operatorname{sgn}(x) \ln(1 + \mu|x|)}{\ln(1 + \mu)}, \quad 0 \leq |x| \leq 1 \quad (7)$$

$\operatorname{sgn}(x)$ représente le signe de x . x dans ce cas-ci correspond à une image HDR résultante du réseau convolutionnel. Nous savons donc qu'elle sera toujours positive. L'équation peut alors se réécrire de la manière suivante :

$$T = \frac{\ln(1 + \mu H)}{\ln(1 + \mu)} \quad (8)$$

T représente l'image HDR *tonemapped*. La loi μ est défini ainsi : "A standard analog signal compression algorithm, used in digital communications systems of the North American digital hierarchy, to optimize, i.e., modify, the dynamic range of an analog signal prior to digitizing" [15]. Comme la définition

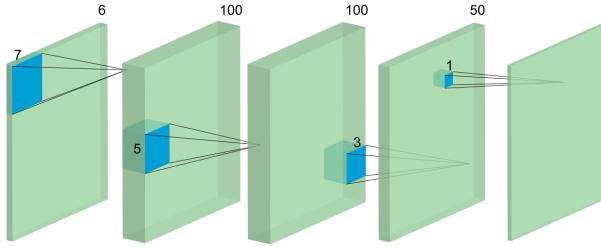


FIGURE 3 – Réseau convolutionnel [1]

l'indique, elle peut servir à modifier, dans ce cas-ci compresser, l'amplitude d'un signal. Ainsi, elle peut s'appliquer au signal d'une image HDR pour réduire son domaine à d'une image LDR. Il est à noter qu'une valeur de 5000 a été utilisée pour μ .

6 Architectures

Trois architectures ont été implémentées pour mesurer l'efficacité du réseau convolutionnel. La figure 3 contient n_o canaux de sortie. Le nombre change dépendamment de l'architecture.

6.1 Directe

L'architecture directe est le modèle le plus simple. Il modélise le fusionnement au complet. C'est-à-dire qu'il produit directement des images HDR. Le résultat du réseau a donc 3 canaux de sorties. La *backpropagation* requiert de résoudre la dérivée en chaîne suivante :

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{T}} \frac{\partial \hat{T}}{\partial \hat{H}} \frac{\partial \hat{H}}{\partial w} \quad (9)$$

Le terme $\frac{\partial E}{\partial \hat{T}}$ représente la dérivée partielle de l'équation 6 par rapport à l'image HDR résultante une fois *tonemapped*. Cette dérivée s'exprime ainsi :

$$\frac{\partial E}{\partial \hat{T}} = 2 \sum_{j=1}^3 \hat{T}_j - T_j \quad (10)$$

Par la suite, le terme $\frac{\partial \hat{T}}{\partial \hat{H}}$ représente la dérivée partielle de l'équation 8 par rapport à l'image HDR résultante. Elle s'exprime ainsi :

$$\frac{\partial \hat{T}}{\partial \hat{H}} = \frac{\mu}{\ln(1 + \mu)} \frac{1}{\ln(1 + \mu \hat{H})} \quad (11)$$

Enfin, le troisième terme $\frac{\partial \hat{H}}{\partial w}$ correspond à la dérivée des poids du réseaux par rapport au résultat de celui-ci. Ce calcul est fait par *backpropagation* au travers des différentes couches [13].

Bien que l'architecture soit la plus simple en terme d'implémentation, c'est celle qui a la tâche de modélisation la plus complexe. Elle doit modéliser la fusion au complet. Cela lui donne un degré de liberté plus grand que les deux autres architectures vis-à-vis ce qu'elle peut estimer. Cependant, ceci représente un couteau à double tranchant. En effet, d'un côté, ce degré de liberté de plus peut permettre au réseau de détecter des caractéristiques qu'un algorithme plus simple ne verrait pas. D'un autre côté, il augmente les chances que le réseau n'apprenne pas exactement ce qui est voulu. Alors, c'est l'architecture qui est le plus à risque de laisser passer des artefacts non-désirés.

6.2 Weight Estimator

L'architecture Weight Estimator(WE) diffère de l'architecture directe par le fait qu'elle ne produit pas une image HDR. L'architecture utilise plutôt une moyenne pondérée des images dans le domaine HDR données en entrées. L'utilisation de cette technique est répandue pour fusionner trois images en une seule de type HDR [16]. Le réseau convolutionnel sert à déterminer la pondération de chacune des images du triplet. Il produit donc un résultat avec 9 canaux de sorties soit 3 par matrice de pondération. Voici l'équation de la moyenne pondérée :

$$\hat{H}(p) = \frac{\sum_{j=1}^3 \alpha_j(p) H_j(p)}{\sum_{j=1}^3 \alpha_j(p)} \quad (12)$$

où

$$H_j(p) = \frac{I^\gamma}{t_j} \quad (13)$$

$\alpha_j(p)$ représente la pondération du pixel p pour la j-ième image. L'intention de la pondération est

de faire ressortir les caractéristiques importantes de chaque exposition. Ainsi, la dérivée en chaîne à résoudre s'exprime de la manière suivante :

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{T}} \frac{\partial \hat{T}}{\partial \hat{H}} \frac{\partial \hat{H}}{\partial \alpha} \frac{\partial \alpha}{\partial w} \quad (14)$$

où α représente les trois matrices pondération. Les deux premiers termes résultent aux mêmes dérivées partielles que dans la première architecture. Le troisième terme $\frac{\partial \hat{H}}{\partial \alpha}$ représente la dérivée partielle de l'équation 12 par rapport aux trois pondérations. Elle s'exprime ainsi :

$$\frac{\partial \hat{H}}{\partial \alpha_i} = \frac{H_i(p) - \hat{H}}{\sum_{j=1}^3 \alpha_j(p)} \quad (15)$$

L'obtention de cette dérivée est décrite dans l'annexe A.

Le quatrième terme $\frac{\partial \alpha}{\partial w}$ représente la différentiation du résultat du réseau par rapport à ses paramètres. Celle-ci s'obtient aussi par *backpropagation* comme pour la première architecture [13].

Cette deuxième architecture est la plus contrainte des trois. En effet, elle ne modélise que la pondération accordée aux pixels des différentes expositions. Celle-ci devrait permettre d'ignorer les sections des images de basse et haute exposition contenant des artefacts d'alignement. Elle devrait aussi permettre d'ignorer les sections saturées ou contenant du bruit.

6.3 Weight and Image Estimator

La dernière architecture consiste à premièrement raffiner les images LDR puis, déterminer la pondération accordée à chacune pour obtenir une image HDR par l'équation 12. Il est possible de forcer un réseau profond à reproduire l'entrée qui lui a été donnée. Ce type de réseau correspond à un Autoencodeur. Il s'agit d'une forme d'apprentissage non-supervisé où le but du réseau est d'apprendre la fonction identité pour un ensemble de données [17]. Dans notre contexte, comme décrit dans la section 4.3, l'entrée est un ensemble de *patches* d'une taille de 40x40. L'application en chaîne des filtres du réseau convolutionnel sur ces *patches* résultent en de nouvelles de taille 28x28 (annexe B). Le fait que le résultat est

plus petit force le réseau à apprendre quelles sont les caractéristiques les plus importantes pour représenter l'image LDR en entrée. Ainsi, le résultat est une version "concentrée" de ces caractéristiques. L'objectif est de minimiser la distance l2 entre les images LDR alignées et celles résultantes du réseau. La relation que le réseau représente s'exprime ainsi $\tilde{I} = I$. Cette architecture produit donc un résultat avec 18 canaux de sorties. 9 sont pour les images LDR raffinées et 9 autres pour les matrices de pondérations. Elle convertit les trois images LDR raffinées en images HDR raffinées grâce à l'équation 13 puis, applique l'équation 12 avec la pondération qu'elle produit. La dérivée en chaîne à résoudre est la suivante :

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{T}} \frac{\partial \hat{T}}{\partial \hat{H}} \frac{\partial \hat{H}}{\partial \{\alpha, \tilde{I}\}} \frac{\partial \{\alpha, \tilde{I}\}}{\partial w} \quad (16)$$

La seule différence avec la deuxième architecture est qu'il faut effectuer une deuxième dérivée en chaîne pour obtenir $\frac{\partial \hat{H}}{\partial \tilde{I}_i}$.

$$\frac{\partial \hat{H}}{\partial \tilde{I}_i} = \frac{\partial \hat{H}}{\partial \tilde{H}_i} \frac{\partial \tilde{H}_i}{\partial \tilde{I}_i} \quad (17)$$

Le premier terme $\frac{\partial \hat{H}}{\partial \tilde{H}_i}$ correspond à la dérivée partielle de l'équation 12 par rapport aux images HDR raffinées.

$$\frac{\partial \hat{H}}{\partial \tilde{H}_i} = \frac{\alpha_i(p)}{\sum_{j=1}^3 \alpha_j(p)} \quad (18)$$

Le deuxième terme $\frac{\partial \tilde{H}_i}{\partial \tilde{I}_i}$ correspond à la dérivée partielle de l'équation 13 par rapport à l'image LDR raffinée.

$$\frac{\partial \tilde{H}_i}{\partial \tilde{I}_i} = \frac{\gamma}{t_i} \tilde{I}_i^{\gamma-1} \quad (19)$$

L'architecture profite de plus de liberté que la deuxième mais moins que la première. En effet, le fait qu'elle génère des images LDR raffinées lui donne plus de possibilités de trouver des détails que la deuxième architecture ne pourrait pas voir. Cependant, il reste qu'elle ne modélise pas la fusion au grand complet.

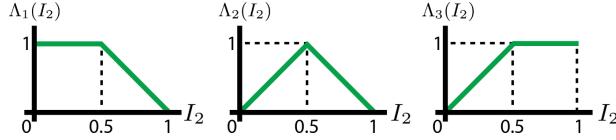


FIGURE 4 – Les fonctions triangles utilisées pour déterminer les poids utilisés dans la moyenne pondérée afin de générer les images HDR de référence [1].

7 Ensemble de données

Nous reprenons l'ensemble de données fourni par l'article de recherche. Il y a un ensemble d'entraînement et un de test. L'ensemble d'entraînement contient 74 scènes alors que celui de test en contient 15. Chaque scène consiste en trois images Low Dynamic Range (LDR), le temps d'exposition de chacune et une image HDR. Les trois images LDR serviront d'entrée au à la pré-convolution. Ces images ont une résolution de 1000x1500 et sont exprimées dans un espace de couleur sRGB grâce au pré-traitement. La première image du triplet correspond à une image d'exposition plus basse que la normale, la deuxième correspond à une exposition normale et la troisième à une plus haute exposition. Le temps d'exposition de chacune est contenu dans un fichier texte nommé *expo.txt*. Ces trois images représentent une certaine scène contenant une ou plusieurs personnes effectuant une certaine action causant du mouvement visible. Ainsi, la position de la personne change d'une image à l'autre. L'image HDR quant à elle représente environ la même scène mais diffère par le fait que le sujet ne bouge pas. Elle servira de référence pour entraîner et tester le réseau profond. En effet, cette image a été créée à partir de trois images LDR de la scène sans mouvement. Dans une telle situation, il est plus simple de créer une image HDR satisfaisante puisqu'il n'y a pas d'alignement nécessaire. Les auteurs ont utilisés l'équation 12 avec des poids calculés ainsi :

$$\alpha_1 = 1 - \Lambda_1(I_2), \alpha_2 = \Lambda_2(I_2), \alpha_3 = 1 - \Lambda_3(I_2) \quad (20)$$

Les fonctions $\Lambda_1, \Lambda_2, \Lambda_3$ sont définies à la figure 4. Enfin, toutes les images LDR ont été prises avec une caméra Canon EOS-5D Mark III.

8 Implémentation

L'annexe C démontre le diagramme classe représentant l'implémentation du système utilisée pour construire et entraîner les trois architectures. L'implémentation a été faite principalement à l'aide de Pytorch et numpy. Le diagramme montre l'utilisation des classes `torch.util.Dataset` et `torch.nn.Module`. La première sert à définir les ensemble de données et la manière de les récupérer. La deuxième sert à définir les modèles.

Une classe nommée `ModelDeepHDR` a été implémentée pour initialiser le réseau convolutionnel et les paramètres du réseau (selon la méthode de Xavier [18]). De plus, cette classe définit une fonction *post_convolutions_steps* qui permet de prendre avantage du patron de conception *Template Method* [19, p. 325-330]. Ce patron consiste à définir un squelette d'algorithme et des fonctions qui peuvent être redéfinies par des classes héritantes de la classe mère. Ainsi, `ModelDeepHDR` définit *post_convolutions_steps* comme étant le *tonemapping*. La classe `WEDeepHDR` redéfinie la fonction pour effectuer la moyenne pondérée et le *tonemapping*. Alors que `WIEDeepHDR` la redéfinie pour convertir les images LDR raffinées en images HDR raffinées et faire la moyenne pondérée en plus du *tonemapping*.

Les entrées données aux modèles sont toutes convertis au type `torch.autograd.Variable` avant d'être traitées par le réseau. Cela permet de déléguer le calcul du gradient au module autograd. Pour ce faire, il faut que toutes les opérations effectuées durant la *forward phase* soient faites par des fonctions Pytorch. La classe `Variable` maintient un historique de toutes les opérations Pytorch qu'on a effectué sur elle. Ainsi, lorsque la fonction `backward` est appelée, elle traverse l'historique des opérations dans le sens inverse pour calculer la dérivée en chaîne. Les dérivées calculées par autograd sont décrites à la section 6.

La classe abstraite `TrainerDeepHDR` permet de définir la base de l'algorithme pour entraîner un modèle. Pour permettre son abstraction, celle-ci hérite de `ABC` qui est une classe Python permettant de définir des fonctions abstraites. Une fonction abstraite

doit être obligatoirement redéfinie par les classes qui en héritent. Ainsi, la fonction `build_model` est redéfinie par trois classes chacune initialisant le modèle correspondant à leur nom.

Il est important de noter que la relation entre les classes *trainers* et les classes modèles présente un problème, un anti-patron, nommé *Parallel Inheritance Hierarchies* [20]. Ce problème survient lorsque l'ajout d'une sous-classe (une classe héritante d'une autre) requiert l'ajout d'une autre dans une hiérarchie différente. Dans notre contexte, l'ajout d'un modèle requiert l'ajout d'un *trainer*. Par manque de temps, le problème n'a pas été réglé puisqu'il n'a pas de répercussion grave dans la situation présente.

Enfin, il est possible de constater que la classe `WIEDeepTrainer` semble plus complexe que les deux autres *trainers*. Cela s'explique par le fait que son entraînement se fait en deux phases. La première consiste à seulement raffiner les images LDR. Puis, la deuxième consiste à générer les images HDR à partir des images LDR raffinées et des pondérations. La combinaison de `WIEDeepTrainer` et `WIEDeepHDR` représente une machine à état. Le *trainer* met le modèle en phase 1 et l'entraîne. Puis, lorsque la phase est terminé, il le met en phase 2 et l'entraîne. De plus, les classes `RefinerScenesDeepHDR` et `ScenesDeepHDR` servent à charger les différentes scènes dépendamment de la phase. L'algorithme ne change autrement que pour la classe qui charge les scènes. Dans ce sens, il s'agit d'une implémentation du patron de conception *Strategy* [19, p. 315-323]. Ce patron a pour principe de permettre de changer, en cours d'exécution, le fonctionnement de certaines parties d'un algorithme.

9 Résultats

L'optimiseur Adam a été utilisé comme décrit dans l'article sur lequel l'expérience est basé. Cela correspond à un taux d'apprentissage de 0.0001, ainsi que $\beta_1 = 0.9$ et $\beta_2 = 0.999$. L'entraînement a été performé sur 2 millions d'itérations avec des *batches* de 20. Deux machines ont été utilisées. Une première ayant 24 cpu Intel Xeon 5600, une GTX Titan X et 32Go de RAM. La deuxième avait un cpu i5-4670K, une GTX 1070 et 16Go de RAM. Pour les deux ma-

chines, un entraînement était d'environ 22h.

Les auteurs ont eu l'aimabilité de nous fournir les fichiers de paramètres pour obtenir le même résultat de *tonemapping* qu'eux en utilisant Photomatix. De plus, ils nous ont fournis les paramètres utilisés pour calculer la mesure HDR-VDP-2.

On observe certains artéfacts dans les résultats obtenus, surtout dans les scènes où il y a d'amples mouvements. Par exemple, dans la figure 5, on peut voir que l'homme a les bras écartés dans une des photos, mais pas dans l'image de moyenne exposition. Dans ce cas, on observe que les artéfacts qui ressortent dans l'image HDR à la sortie de notre réseau convolutionnel contient des artéfacts là où les bras de l'homme se trouvait. Ceci peut être expliquer par le fait que l'alignement des trois images n'est pas parfait. Par contre, avec la deuxième architecture ces artéfacts sont absents. En effet, comme cette architecture est plus contrainte, elle permet de détecter plus rigoureusement ces artéfacts puisqu'elle peut y consacrer plus d'attention.

Comme on peut le voir avec la figure 6, certains artéfacts sont corrigés par la deuxième et troisième architecture, mais pas par la première. Effectivement, on voit qu'avec la première architecture, le fait que les trois images ne soient pas cadrées de la même façon fait en sorte qu'un artéfact apparaît dans le coin supérieur gauche de l'image, là où dans deux des trois photos initiales se trouvent du feuillage alors que dans la troisième le feuillage laisse place au ciel. De plus, un autre résidu apparaît, mais cette fois sur la jambe droite de la dame. Il est, encore une fois dû, au mouvement relatif à la caméra dans cette partie de l'image, et donc un résultat de l'alignement, mais seule l'architecture directe ne réussit pas à la corriger. Comme mentionné plus haut, cette architecture est la moins contrainte, c'est-à-dire qu'elle se concentre sur plusieurs aspects, elle divise ses ressources en quelques sortes, donc elle est la moins susceptible de corriger ces erreurs.

Pour avoir des résultats plus significatifs, nous évaluons plusieurs métriques sur les images obtenues et comparons ces résultats dans le tableau 1. Le PSNR est une méthode objective de calculer la perte de qualité ou d'information à l'aide du libellé [21]. Il s'agit d'un calcul du ratio entre la puissance maximal d'un



FIGURE 5 – Comparaison des résultats des sections en rouge et vert pour les trois architectures de l’article et pour notre architecture directe et WE avec celle de référence.

signal et le bruit qui altère ce signal. Nous comparons aussi le HDR-VDP-2, qui consiste en une prédiction de la différence entre deux images, comme son nom l’indique [22]. En effet, le calcul de cette métrique prend en compte une image, une référence ainsi que quelques caractéristiques de l’écran utilisé. Pour calculé ce score, nous avons utilisé l’image en sortie de notre réseau convolutionnel, le libellé et les paramètres fournis par les auteurs. On constate que les PSNR obtenus par l’architecte directe et WE sont proches de ceux obtenus par les auteurs. Ce n’est pas du tout le cas de l’architecture WIE. Une explication

possible est donnée dans la section suivante. Il est intéressant de noter le fait que malgré que les PSNR sont proches pour les deux premières architectures, il y a une importante différence entre les valeurs des HDR-VDP-2.

Ensuite, nous présentons les pondérations obtenues par les architectures WE et WIE. Lorsqu’on regarde les trois matrices de pondérations de l’architecture WIE, on voit clairement que les résultats ne sont pas cohérents avec ce qu’on attend. Comme le montre la figure 8, on s’attendrait à des poids plus significatifs aux endroits les plus lumineux de l’image de basse

Tableau 1 – Comparaison des métriques de leurs architectures aux nôtres. Les valeurs de PSNR-T sont celles obtenues sur les images *tonemapped* tandis que les valeurs de PSNR-L ont été obtenues sur les images linéaires.

	Les leurs			Les nôtres		
	Direct	WE	WIE	Direct	WE	WIE
PSNR-T	42.92	42.74	43.26	39.65	42.71	7.55
HDR-VDP-2	67.45	66.63	67.50	49.54	48.36	33.42
PSNR-L	41.69	41.25	41.60	40.50	40.75	7.00

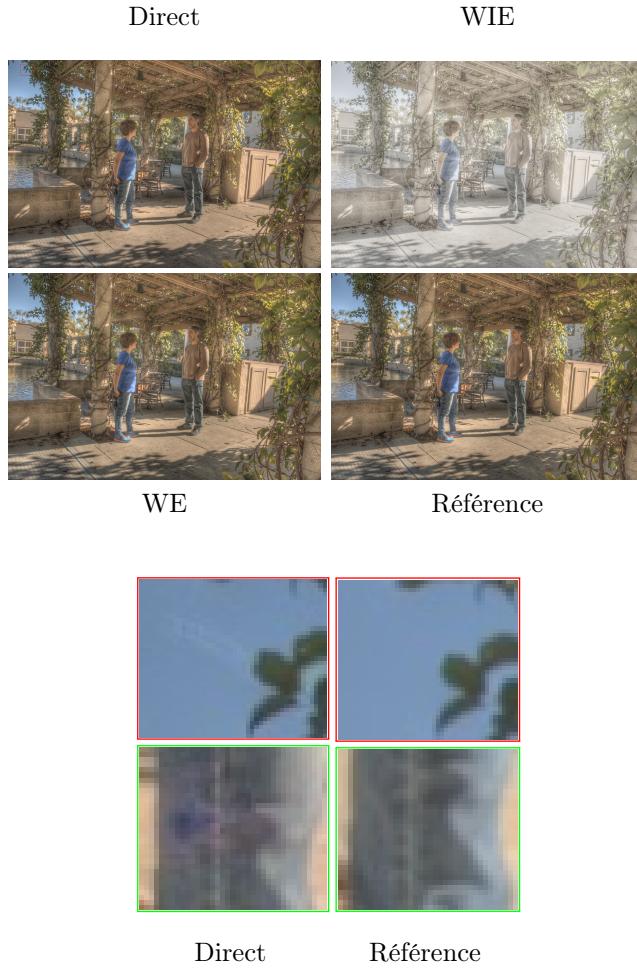


FIGURE 6 – Comparaison de quatres images HDR *tonemapped*. De plus, démonstration d'un défaut d'alignement qui n'a pas été résolu par l'architecture directe.



FIGURE 7 – Les trois matrices de pondération pour la scène BarbequeDay produites par notre WIE, puis les images raffinées de cette architecture et l'image HDR produite.

exposition, alors qu'on obtient plutôt des poids qui semblent beaucoup trop élevés partout dans l'image, figure 7. Contrairement à la dernière architecture, WE produit des pondérations intéressantes. En effet, celles-ci semblent bien relever les caractéristiques que chaque exposition amène. Les parties plus lumineuses sont mises en importante pour l'image de basse exposition, alors que les plus sombres le sont pour l'image de haute exposition. Enfin, la pondération de l'image de moyenne exposition semble relever les sections qui ne sont ni sous ni trop exposées. Par exemple, pratiquement tout autour de la table est blanc ce qui correspond à partie peu lumineuse dans l'image originale. À l'inverse, le ciel est noir dans la pondération. La cuisse droite de la dame n'est pas complètement blanche. Il est possible que le réseau est jugé cette section sous exposée.

Comparons maintenant les résultats graphiques obtenus par nos deux architectures et celles que les auteurs de l'article [1] ont obtenus, figure 5. On peut voir qu'ils sont très semblables. Les deux architectures directes présente le même artefact près du pneu arrière de la voiture la plus à gauche. Les architectures WE présentent aussi des résultats semblables, puisque les images contiennent un artefact vraiment

moins visible à cet endroit, mais on peut quand même observer une distortion comparativement au libellé. Puis, si l'on prend la joue gauche de l'homme, on peut constater qu'il y a du bruit sur nos résultats aussi et il semble être en quantité similaire, on peut conclure que nos résultats sont plutôt similaires à ceux des auteurs.

10 Discussion

Comme on peut le voir avec ces résultats, la troisième architecture semble avoir été sabotée par des bogues logiciels que nous n'avons pu déceler, principalement en raison du temps restreint et du fait que l'entraînement prend 2 jours. En effet, les scores obtenus pour les deux autres architectes sont semblables à ceux obtenus par les auteurs, et ne diffère que de manière raisonnable. La troisième architecture obtient des valeurs de 7.54 et de 7.00 pour les PSNR-T et PSNR-L respectivement, alors qu'on s'attend à des valeurs dans les alentours de 40. Une différence aussi importante est anormale ce qui nous amène à conclure que la cause est un bogue d'implémentation.

Pour calculer le flux optique de deux images, nous avons utilisé un *wrapper* python des mêmes fichiers



FIGURE 8 – Les trois matrices de pondération pour la scène BarbequeDay produites par notre WIE, puis les images raffinées de cette architecture et l'image HDR produite.

sources que la version Matlab utilisée par les auteurs. Par contre, nous nous sommes rendu compte qu'en passant exactement les mêmes données aux deux *wrappers*, les résultats différaient. Cela peut donc expliquer certaines différences dans nos résultats. Cependant, nous avons remarqué que même après l'interpolation, la reconstruction et la sélection des patches les résultats étaient toujours près de ceux qui découlent de l'implémentation en Matlab.

De plus, avec un peu plus de temps, nous aurions pu entraîner un réseau à l'aide du code source des auteurs. Cela nous aurait permis de pouvoir passer en entrée des scènes de notre choix dans les deux réseaux et de pouvoir comparer les artefacts obtenus, au lieu de devoir se contenter des images présenter dans leur article seulement. De cette manière, l'analyse des résultats aurait pu être faite plus en profondeur.

11 Conclusion

En conclusion, nous avons entrepris de réimplémenter la méthode de génération d'images HDR présentée dans *Deep High Dynamic Range Imaging of Dynamic Scenes* [1]. Cette méthode consiste à fusionner des images LDR de différente exposition en une seule image HDR. L'algorithme se sépare en deux

phases principales. La première s'agit d'aligner les images LDR. Puis, la deuxième consiste à fusionner les images alignées. La nouveauté présentée par l'article est de modéliser la fusion par un réseau convolutionnel. Les auteurs ont décidé de tester le réseau au travers de trois architectures. L'expérience a été développée en Matlab. Nous avons décidé de la réimplémenter en Python. Les résultats obtenus sont partiellement concluant. Les valeurs PSNR pour les deux premières architecture sont similaires mais ce n'est pas le cas pour la valeur HDR-VDP-2. Enfin, La troisième architecture semble souffrir d'un bogue d'implémentation puisqu'elle ne donne pas du tout de résultats concluants. Il serait intéressant de reprendre l'expérience mais dans une optique d'améliorer le réseau. Plusieurs méthodes seraient possible. Par exemple, on pourrait ajouter du *dropout* [23] durant l'entraînement.

Références

- [1] N. K. Kalantari and R. Ramamoorthi, “Deep high dynamic range imaging of dynamic scenes,” *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

- [2] M. D. Tocci, C. Kiser, N. Tocci, and P. Sen, “A versatile hdr video production system,” *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4, p. 41, 2011.
- [3] MATLAB, version 9.3 (R2017b). Natick, Massachusetts, 2017. Disponible à <https://www.mathworks.com/help/images/ref/demosaic.html>.
- [4] M. Levoy, A. Adams, K. Dektar, and N. Willett, *CS 178 - Digital Photography : Gamma correction*, 2014. Disponible à <http://graphics.stanford.edu/courses/cs178/applets/gamma.html>.
- [5] J. L. McKesson, *Learning Modern 3D Graphics Programming : Linearity and Gamma*, 2012. Disponible à <https://web.archive.org/web/20130718042406/http://www.arc synthesis.org/gltut/Illumination/Tut12%20Monitors%20and%20Gamma.html>.
- [6] w3c, *How to interpret the sRGB color space (specified in IEC 61966-2-1) for ICC profiles*, 2014. Disponible à <https://www.w3.org/Graphics/Color/srgb>.
- [7] I. E. Commission *et al.*, “Multimedia systems and equipment-colour measurement and management-part 2-1 : Colour management-default rgb colour space-srgb,” *IEC 61966-2-1*, 1999.
- [8] K. R. Aires, A. M. Santana, and A. A. Medeiros, “Optical flow using color information : preliminary results,” in *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1607–1611, ACM, 2008.
- [9] J. Wang and L. Perez, “The effectiveness of data augmentation in image classification using deep learning,”
- [10] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.
- [11] E. W. Weisstein, “Sigmoid function,” 2002.
- [12] Y. Salih, A. S. Malik, N. Saad, *et al.*, “Tone mapping of hdr images : A review,” in *Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on*, vol. 1, pp. 368–373, IEEE, 2012.
- [13] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, “Learning representations by back-propagating errors,” *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.
- [14] “Waveform Coding Techniques.” Disponible à <https://www.cisco.com/c/en/us/support/docs/voice/h323/8123/waveform-coding.html>.
- [15] “Federal standard 1037c : Telecommunications : Glossary of telecommunication terms,” 1996. Disponible à <https://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm>.
- [16] A. K. Johnson, “High dynamic range imaging-a review,” *International Journal of Image Processing (IJIP)*, vol. 9, no. 4, p. 198, 2015.
- [17] A. Ng, “Sparse autoencoder,”
- [18] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.
- [19] E. Gamma, *Design patterns : elements of reusable object-oriented software*. Reading, Mass : Addison-Wesley, 1995.
- [20] M. Fowler and K. Beck, *Refactoring : improving the design of existing code*. Addison-Wesley Professional, 1999.
- [21] A. Hore and D. Ziou, “Image quality metrics : Psnr vs. ssim,” in *Pattern recognition (icpr), 2010 20th international conference on*, pp. 2366–2369, IEEE, 2010.
- [22] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich, “Hdr-vdp-2 : A calibrated visual metric for visibility and quality predictions in all luminance conditions,” in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 40, ACM, 2011.
- [23] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout : a simple way to prevent neural networks from overfitting.,” *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

Annexes

A Dérivée partielle de la moyenne pondérée par rapport à une pondération

Règle de la dérivée d'un quotient :

$$\left(\frac{f}{g}\right)' = \frac{f'g - g'f}{g^2}$$

Ainsi :

$$\begin{aligned} \frac{\partial \hat{H}}{\partial \alpha_i} &= \frac{\frac{\partial(\sum_{j=1}^3 \alpha_j(p) H_j(p))}{\partial \alpha_i} \sum_{j=1}^3 \alpha_j(p) - \frac{\partial(\sum_{j=1}^3 \alpha_j(p))}{\partial \alpha_i} \sum_{j=1}^3 \alpha_j(p) H_j(p)}{(\sum_{j=1}^3 \alpha_j(p))^2} \\ &= \frac{H_i(p) \sum_{j=1}^3 \alpha_j(p) - 1 \cdot \sum_{j=1}^3 \alpha_j(p) H_j(p)}{(\sum_{j=1}^3 \alpha_j(p))^2} \\ &= \frac{H_i(p) \sum_{j=1}^3 \alpha_j(p)}{(\sum_{j=1}^3 \alpha_j(p))^2} - \frac{\sum_{j=1}^3 \alpha_j(p) H_j(p)}{(\sum_{j=1}^3 \alpha_j(p))^2} \\ &= \frac{H_i(p)}{\sum_{j=1}^3 \alpha_j(p)} - \frac{\hat{H}}{\sum_{j=1}^3 \alpha_j(p)} \\ &= \frac{H_i(p) - \hat{H}}{\sum_{j=1}^3 \alpha_j(p)} \end{aligned}$$

B Taille d'une *patch* résultante du réseau convolutionnel

Le changement de taille d'une image sur laquelle on applique une convolution s'exprime ainsi :

$$\hat{d}_i = d_i - f_i + s_i, i = \{1, 2\}$$

où d_i est la taille de la dimension i avant et \hat{d}_i la taille de la dimension après la convolution. f_i est la taille du filtre pour la dimension i et s_i est la grandeur du pas effectuée dans la dimension i.

Première couche de convolution

$$\begin{aligned} & 40x40 \\ & 40 - 7 + 1 = 34 \\ & 34x34 \end{aligned}$$

Deuxième couche de convolution

$$\begin{aligned} & 34x34 \\ & 34 - 5 + 1 = 30 \\ & 30x30 \end{aligned}$$

Troisième couche de convolution

$$\begin{aligned} & 30x30 \\ & 30 - 3 + 1 = 28 \\ & 28x28 \end{aligned}$$

Quatrième couche de convolution

$$\begin{aligned} & 28x28 \\ & 28 - 1 + 1 = 28 \\ & 28x28 \end{aligned}$$

C Diagramme de classe

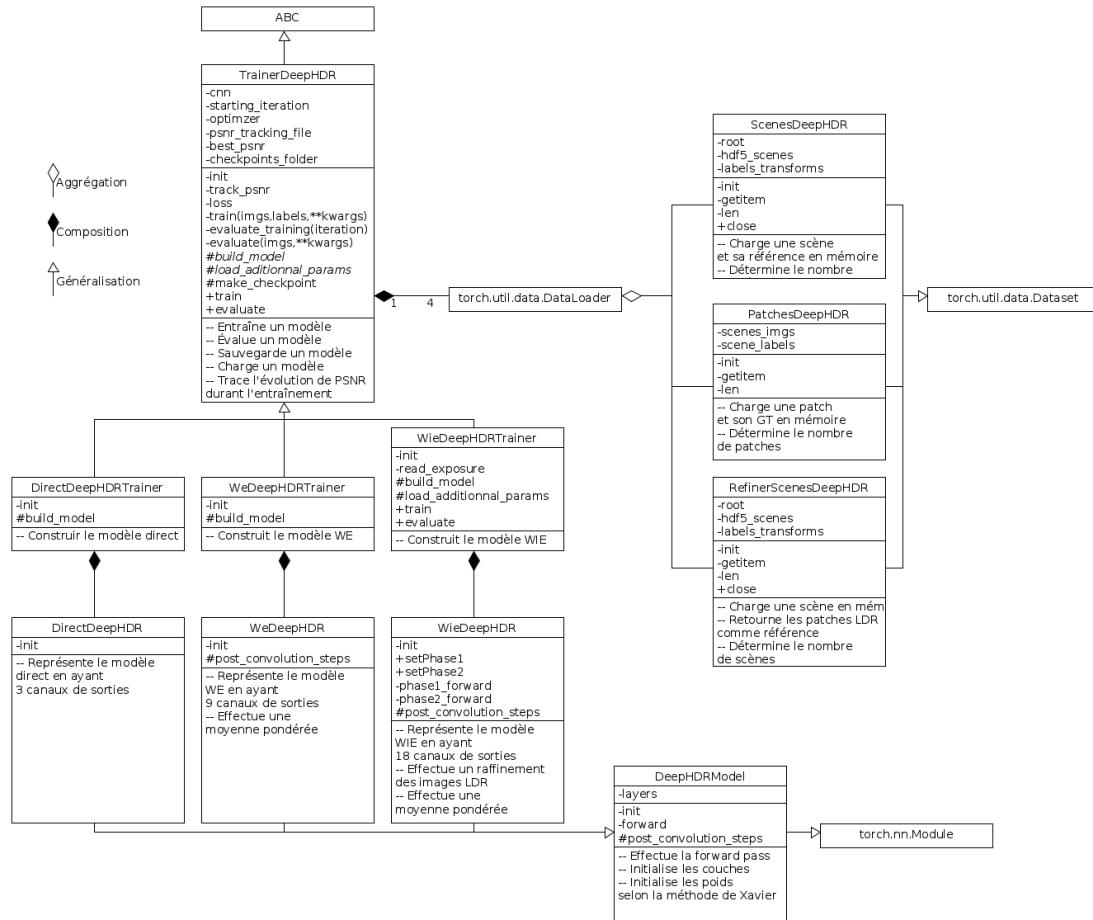


FIGURE 9 – Diagramme de classe sur lequel l'implémentation est basée