

## IMA206 - Rapport

### Etude du réseau Colornet et Applications

---

**Ecrit par :**

Philippe Liu  
Doriand Petit  
Aymeric Degroote  
David Gérard  
Yassine Hargane

**Encadrants :**

Yann Gousseau  
Raphaël Achddou

# Contents

<b>1</b>	<b>Introduction - Colornet</b>	<b>1</b>
1.1	Le réseau . . . . .	1
1.2	Les données . . . . .	2
1.2.1	Colorisation d'images naturelles . . . . .	2
1.2.2	Colorisation d'images NIR . . . . .	3
<b>2</b>	<b>Début du Projet</b>	<b>5</b>
<b>3</b>	<b>Entraînements sur le dataset de base</b>	<b>6</b>
3.1	Entrainement sur 10 classes botaniques: "VégéNet" . . . . .	6
3.2	Entrainement sur 10 autres classes majoritairement en extérieur: "OutNet" . . . . .	8
3.3	Entrainement sur 10 classes avec une plus grande palette de couleurs: mColorNet . . . . .	9
3.4	Modification de la loss en utilisant la "colorfulness" . . . . .	10
<b>4</b>	<b>Modifications des images pour améliorer nos résultats</b>	<b>13</b>
4.1	Contraste - Egalisation d'histogramme . . . . .	13
4.2	Saturation . . . . .	14
<b>5</b>	<b>Effets de la méthode de décolorisation des images sur la colorisation par l'algorithme</b>	<b>16</b>
<b>6</b>	<b>Utiliser Colornet sur des images "Near-Infra-Red"</b>	<b>19</b>
6.1	Essai sur le réseau déjà entraîné . . . . .	19
6.2	Modification du code pour le nouveau dataset . . . . .	20
6.3	Utiliser l'image NIR comme luminance . . . . .	21
6.4	Modifier Colnet . . . . .	22
<b>7</b>	<b>Conclusion</b>	<b>25</b>

# 1 Introduction - Colornet

Ce projet porte sur l'étude de la publication "Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification" de Satoshi Iizuka, Edgar Simo-Serra et Hiroshi Ishikawa publiée en 2016 pour la conférence SIGGRAPH.

L'objet de ce projet a été de prendre en main l'architecture, l'entrainer sur des données, chercher à améliorer les performances avec du pre et du post processing, étudier l'influence de la méthode de décolorisation, et enfin appliquer notre algorithme à la colorisation d'images NIR.

## 1.1 Le réseau

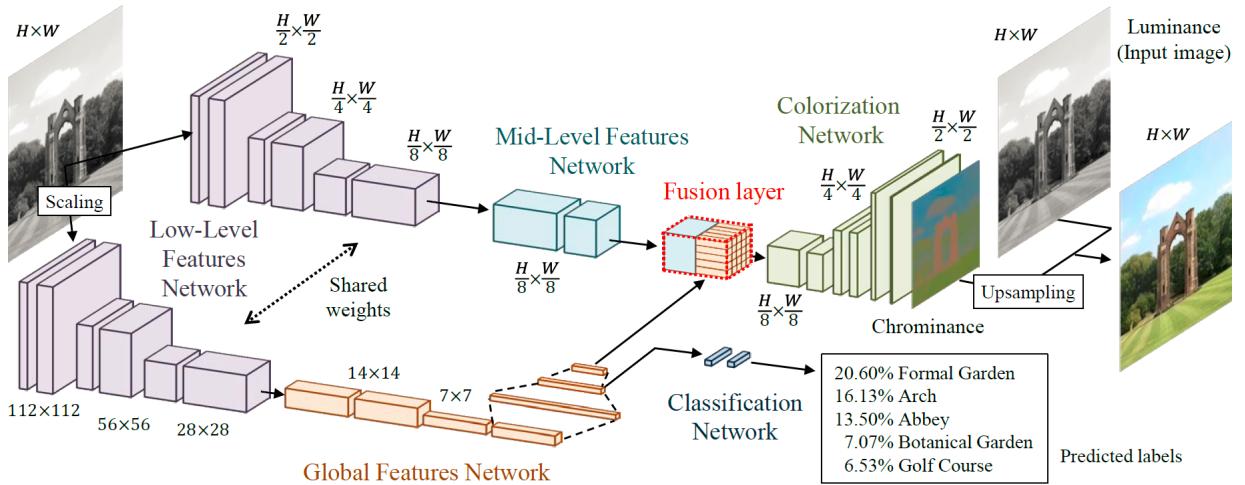


Figure 1: Architecture du réseau Colornet

L'architecture du réseau étudié comporte 5 NN :

1. **CNN d'extraction de features locales** : CNN de 6 couches convolutives permettant l'extraction de features locales. La sortie de ce CNN est branchée sur deux autres CNN, l'extracteur de features globales et l'extracteur de features de niveaux intermédiaires.
2. **CNN d'extraction de features globales** : CNN de 4 couches convolutives permettant l'extraction de features globales. La sortie de ce CNN est branchée sur une couche de fusion des données avec les données de l'extracteur de features de niveaux intermédiaires, et sur un MLP de classification.
3. **CNN d'extraction de features de niveaux intermédiaires** : CNN de 2 couches convolutives permettant l'extraction de features de niveaux intermédiaires. La sortie

de ce CNN est branchée sur une couche de fusion des données avec les données de l'extracteur de features globales.

4. **MLP de fusion des features globales et de niveaux intermédiaires** : Effectue une concaténation des données d'entrée et les fait passer dans une couche fully connected avant de les envoyer en entrée du CNN de Colorisation.
5. **CNN de Colorisation** : Upsample les images en sortie du MPL de fusion et les colorise. L'image doit encore être upsampleée à l'aide de l'image en nuances de gris d'origine pour obtenir le résultat final.

## 1.2 Les données

### 1.2.1 Colorisation d'images naturelles

Dans la publication, le RN a été entraîné et testé sur la base de donnée Places. Cette base de données comprends 365 classes de "scènes". Parmis ces classes, on trouve des scènes comme : "airport\_terminal", "bamboo\_forest", "bridge"...

Chaque classe comporte 5000 images. Les auteurs de la publication ont pris 3 semaines pour entraîner leur RN sur 11 epochs avec une batch size de 128 sur une carte graphique ayant 24 Go de mémoire.

N'ayant pas accès à autant de puissance de calcul et ayant un délais serré, nous nous sommes restreints à entraîner sur les classes :

1. botanical\_garden
2. formal\_garden
3. japanese\_garden
4. roof\_garden
5. topiary\_garden
6. vegetable\_garden
7. zen\_garden
8. lawn
9. house
10. cottage

11. bridge
12. castle
13. house
14. library-indoor
15. mountain
16. palace
17. park
18. pond
19. sky
20. clothing\_store
21. bar
22. ballroom
23. art-school
24. skyscraper
25. waterfall
26. market-outdoor

### **1.2.2 Colorisation d'images NIR**

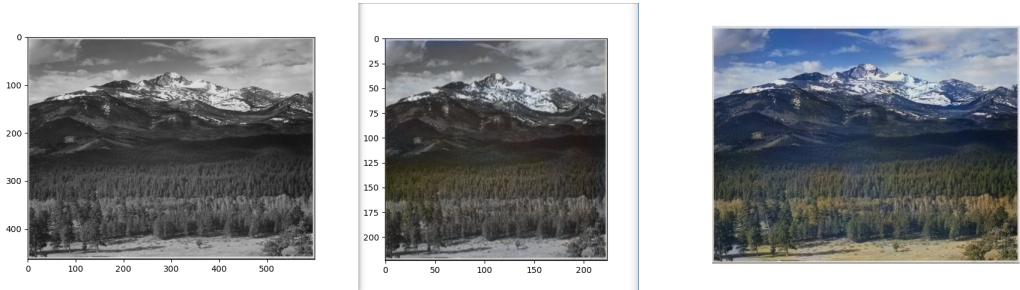
Dans une seconde partie du projet, nous avons utilisé l'algorithme pour coloriser des images NIR. La base de donnée utilisée est la "RGB-NIR Scene Dataset". Elle contient 477 images NIR réparties dans 9 classes de scènes :

1. country
2. field
3. mountain
4. oldbuilding
5. indoor
6. street

7. urban

8. water

9. forest



**Figure 2:** Gauche : Image originale, Milieu : Image obtenue après coloration avec le pré-entraînement, Droite : Image obtenue avec l’application en ligne

## 2 Début du Projet

Au commencement du projet, la première étape a été de comparer le code avec le modèle décrit dans le papier de recherche afin de mieux le comprendre. A cause de l’absence des poids de l’entraînement complet réalisé par les auteurs du papier, nous comptions d’une part effectuer notre propre entraînement, quitte à se concentrer sur un type d’image afin de ne pas avoir un entraînement trop long ; et d’autre part utiliser les poids fournis au format pickle obtenus d’un entraînement d’un ancien groupe.

Nous avons commencé par cette dernière solution afin d’obtenir rapidement des premiers résultats. Pour cela, nous nous sommes inspirés du code simplifié fourni (qui ne contient que le modèle, sans aucun pré/post processing ni notion de classe comme la code original), dans lequel nous avons ajouté les éléments manquants autour du réseau et surtout débuggé les composants du réseau de neurones (il manquait notamment un layer d’upsampling par exemple).

Après tous ces ajouts (le code peut être retrouvé dans le fichier `colornet.py`), les résultats obtenus ont été assez décevants (fig. 2). Bien que ceux-ci soient cohérents (le couleur verte apparaît au niveau de la forêt, un peu de bleu dans le ciel, etc...), l’image obtenue est loin d’être aussi qualitative que l’image colorisée par l’application en ligne des auteurs originaux. Malgré divers essais afin de trouver une solution (notamment en essayant de travailler sur la saturation en sortie du réseau et sur la normalisation), les résultats n’ont jamais été très bons, et le réseau ne colorait pas assez. Une des causes probables est que le pré-entraînement fourni était soit incomplet soit spécifique à un type d’image qui ne correspondait pas à nos images essayées.

A partir de là, nous avons préféré repartir du code original afin d’effectuer nos propres entraînements et nos propres expérimentations.

### 3 Entraînements sur le dataset de base

Dans cette partie, nous allons tout d'abord détailler les différentes démarches entreprises afin d'entraîner cette architecture sur les images du dataset de base. Les différentes classes d'images utilisées ici seront issues de Places365, qui est un répertoire d'images de résolution 256x256, subdivisé en 365 classes variées avec 5000 images chacune.



Quelques classes au hasard...

#### 3.1 Entrainement sur 10 classes botaniques: "VégéNet"

Nous avons tout d'abord entraîné ColNet sur seulement 10 classes majoritairement botaniques de chacune 4000 images, qui sont pour la plupart des jardins, sur 32 epochs avec des batchs de 32 images, avec un learning rate de 10e-3 en utilisant l'implémentation fournie.

Les 10 classes utilisées ici sont: botanical\_garden formal\_garden japanese\_garden roof\_garden topiary\_garden vegetable\_garden zen\_garden lawn house cottage

Voici ci-dessous différents résultats qu'on obtient durant l'entraînement sur des images tests



On peut aussi voir que le réseau se débrouille très bien pour les images des classes sur lesquelles le réseau est entraîné, et est capable de colorer de manière cohérente les différentes textures présentes sur l'image: lorsque la texture s'apparente à des végétaux ou de la verdure, le réseau colore en vert, lorsqu'il y a des formes géométriques, des bâtiments et des routes, la couleur est appropriée.

Cependant, comme indiqué sur le papier, on voit tout de suite que le réseau n'a pas le contexte sémantique de l'image, or on remarque la prédominance de vert dans le base d'entraînement, ce qui a pour conséquence de préférer le vert à d'autres couleurs lorsqu'une incertitude émerge: on peut le voir avec les fleur rouges du topiary garden qui sont colorées en vert, ou encore l'arbre sakura rose qui est coloré en vert à droite.

Pour le voir de manière plus évidente, on peut tester le meilleur modèle (selon la validation loss, c'est le modèle de l'epoch 23) que l'on nomme VégéNet, appliqué à des images colorées de classes différentes de celle de la base d'entraînement:



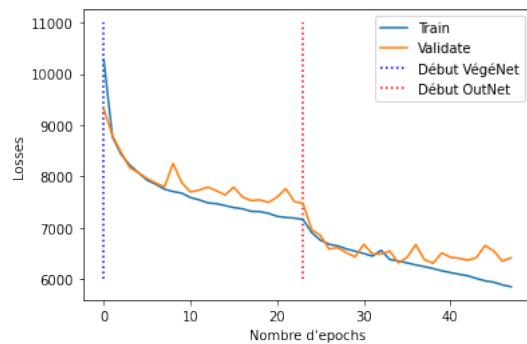
Classe de l'image bridge castle house indoor mountain palace sky  
library

On voit par exemple ci-dessus que la montagne, le palace et le ciel ont des artefacts verts, à cause du dataset de départ et la prédominance statistique du vert.

### 3.2 Entrainement sur 10 autres classes majoritairement en extérieur: "OutNet"

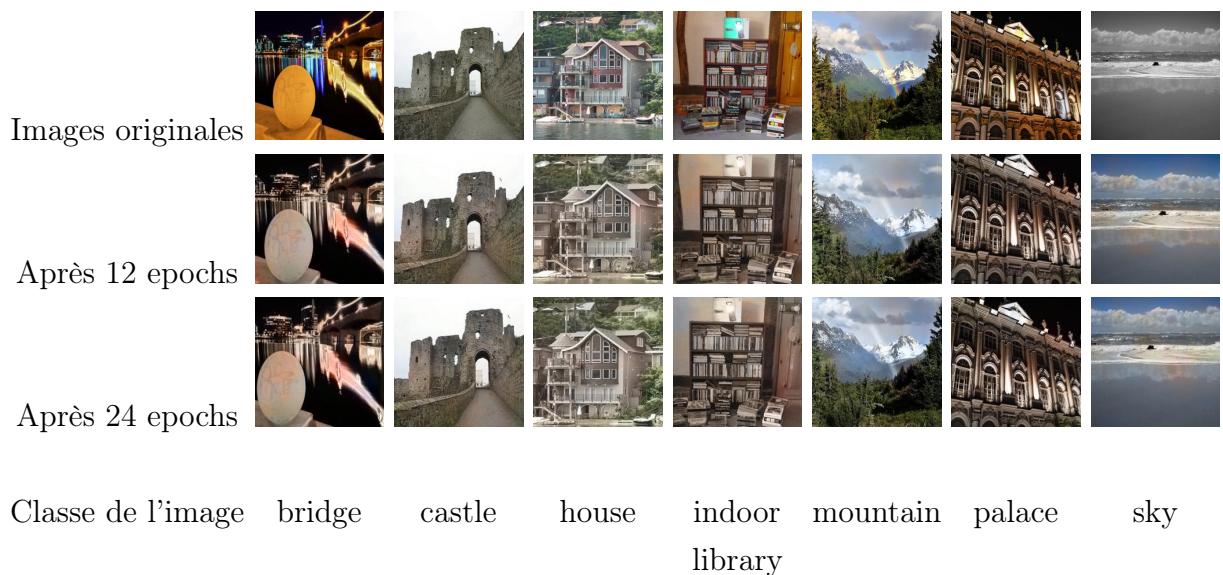
Ici, nous avons repris les poids du meilleur modèle précédent et les avons ré-entraîné sur un dataset plus varié mais qui reste en extérieur, afin d'avoir un modèle qui permet de colorer des images comme vues précédemment: des ponts, des châteaux, des montagnes, maisons, etc. Les 10 classes utilisées ici sont: bridge castle cottage house library-indoor mountain palace park pond sky

Appelons le modèle résultant OutNet. La reprise des poids précédents nous permet de conserver une partie de l'information apprise par le réseaux sur les images botaniques, et de nous faire gagner du temps et des données: c'est le principe du transfer learning. Nous pouvons ainsi voir que la loss d'entraînement et de validation qui est déjà très faible au départ de l'entraînement (qui commence ici à l'epoch 24), ce qui signifie que VénéNet se généralise pas trop mal aux nouvelles données:



## Training et validation loss

Nous pouvons voir ci-dessous les résultats au fur et à mesure de l'entraînement:



Nous pouvons voir que très rapidement le modèle entraîné obtient des résultats acceptables et stagne, les nouvelles epochs n'apportant alors pas de nouvelles informations qui

permettrait d'améliorer la coloration des images. Par exemple, la coloration de la maison ou de l'arc-en-ciel (dans mountain) reste terne car le modèle est face à une ambiguïté, elle ne tente pas d'apporter une couleur car elle risque (statistiquement parlant) d'accroître la loss sur laquelle le modélisé est entraîné (ou parce que le modèle ne connaît pas les arc-en-ciel ici). Ici, entraîner plus reviendrait à faire de l'overfitting sur les données d'entraînement, pour éviter cela, nous réalisons un early-stopping qui permet de reprendre le meilleur modèle/OutNet: celui de l'epoch 39.

Notons aussi la présences de petits artéfacts colorés comme de petites taches rouges qui n'étaient pas présent avant.

Nous pouvons vérifier que OutNet colore toujours de manière acceptable les plantes, et modère la quantité de vert en sortie:



### 3.3 Entrainement sur 10 classes avec une plus grande palette de couleurs: mColorNet

On voit que lorsqu'il y a une ambiguïté sur la couleur, les couleurs en sortie sont assez ternes, ou suivent la couleur majoritaire en sortie. Du coup, nous avons voulu corriger ce phénomène de plusieurs façons différentes. Une première tentative consistait à entraîner le réseau sur une base de donnée, avec une palette colorimétrique plus variée que les derniers dataset. Les 10 classes utilisées ici sont: ballroom bar bridge clothing\_store library-indoor market-outdoor palace sky skyscraper waterfall.



Ici, malgré la diversité des couleurs, le meilleur des réseaux après nouvel entraînement n'arrive pas du tout à deviner les couleurs éclatantes des vêtements ou des fruits, encore une fois, pour minimiser la loss statistique, le réseau pose des couleurs ternes.

### 3.4 Modification de la loss en utilisant la "colorfulness"

La partie précédente presuppose que la loss est statistiquement responsable du manque de dynamiques dans les couleurs générées par le modèle: ici, cette loss vaut la moyenne des écarts quadratiques des canaux sur chaque pixels avec un terme de classification :

$$\mathcal{L}(y^{\text{color}}, y^{\text{class}}) = \|y^{\text{color}} - y^{\text{color},*}\|_{\text{FRO}}^2 - \alpha[y^{\text{class},*} - \log(\sum_{i=0}^N \exp(y^{\text{class},i}))]$$

avec en étoile la vérité terrain, et  $\alpha$  un coefficient qui balance la loss de colorisation et la loss de classification.

On a donc voulu ajouter une loss associée à une mesure de la dynamique des couleurs, la "colorfulness" en anglais. D'après ce papier de l'EPFL: on peut mesurer la dynamique colorimétrique en utilisant les canaux  $a^*b^*$  dans l'espace  $La^*b^*$ :

$$\text{colorfulness}_1 = \sigma_{ab} + 0.37\mu_{ab}$$

avec  $\sigma_{ab}$  l'écart type des canaux  $a^*$  et  $b^*$  et  $\mu_{ab}$  la moyenne de ces dernières.

On a ainsi construit une simple loss à partir de cette métrique, qui va être à un facteur près égal à la différence des colorfulness des images (on aurait pu aussi choisir  $|1 - \text{rapport des deux}|$  ) :

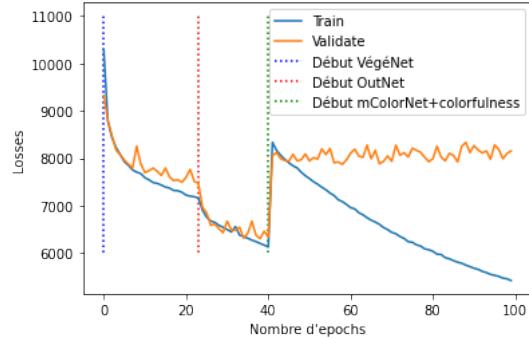
$$\mathcal{L}_{\text{colorfulness}_1}(y^{\text{color}}) = \Delta \text{colorfulness}_1(y^{\text{color}}, y^{\text{color},*})$$

Nous avons ainsi implémenté la loss totale de cette manière, et de la multiplier par un certain facteur multiplié par la partie MSE de la loss, afin d'avoir un impact important (nous avons pris  $\beta = 5$ ):

$$\begin{aligned} \mathcal{L}(y^{\text{color}}, y^{\text{class}}) &= \|y^{\text{color}} - y^{\text{color},*}\|_{\text{FRO}}^2 (1 + \beta \Delta \text{colorfulness}_1(y^{\text{color}}, y^{\text{color},*})) \\ &\quad - \alpha[y^{\text{class},*} - \log(\sum_{i=0}^N \exp(y^{\text{class},i}))] \end{aligned}$$

Nous avons ainsi essayé de ré-entrainer mColorNet et VégéNet avec cette nouvelle loss, mais il s'avère que les résultats obtenus sont aussi très similaires au résultats obtenus

précédemment, si ce n'est plus imprécis, en effet on peut remarquer que ce changement ne fait qu'augmenter le plateau de validation loss et n'apporte pas grand chose au modèle:

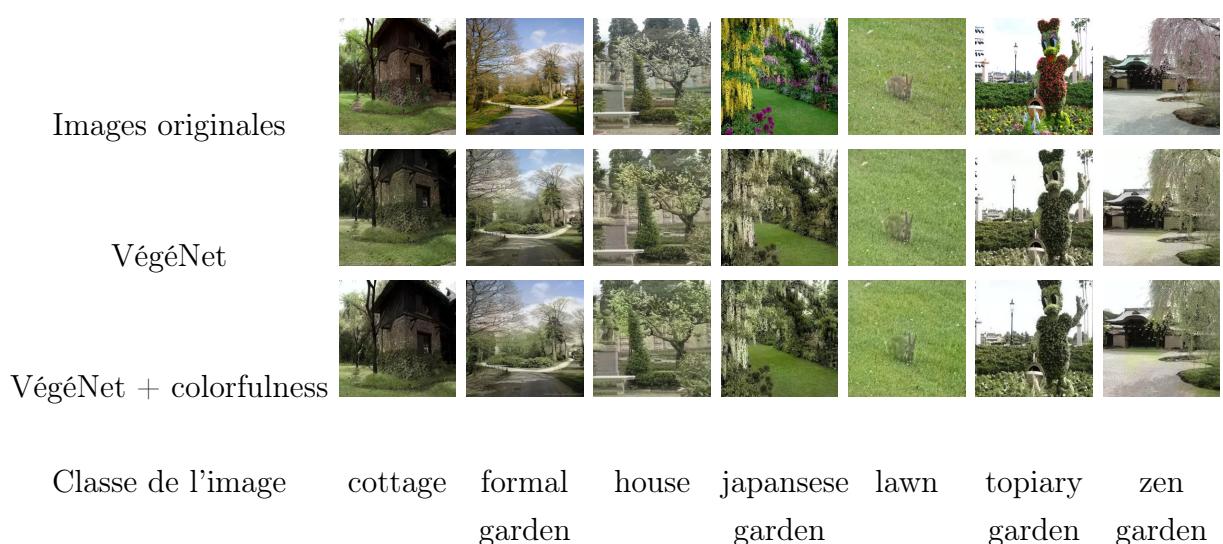


Training et validation loss de mColorNet

On peut aussi remarquer que le modèle overfit très rapidement, et se généralise très mal pour les images de validation, on peut aussi le voir ci-dessous:



Même chose pour VégéNet :



On peut en revanche voir que par exemple sur zen garden, les couleurs débordent plus que prévu, car le modèle a tendance à vouloir écarter les nuances grises vers des couleurs où il est plus certain: ici, le vert est plus présent.

## 4 Modifications des images pour améliorer nos résultats

Ainsi, nos résultats, bien que pas trop mauvais, restent assez éloignés des images originales. Nous avons donc essayé de rajouter du post-processing afin d'obtenir des résultats plus fidèles à nos données.

Afin d'améliorer les résultats en sortie du réseau de neurones, l'idée principale était de travailler sur la saturation et le contraste des images de sortie du réseau.



**Figure 6:** Image nuance de gris



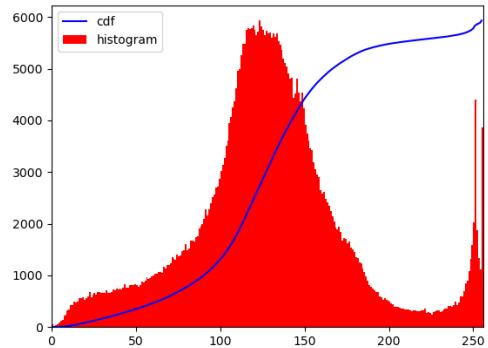
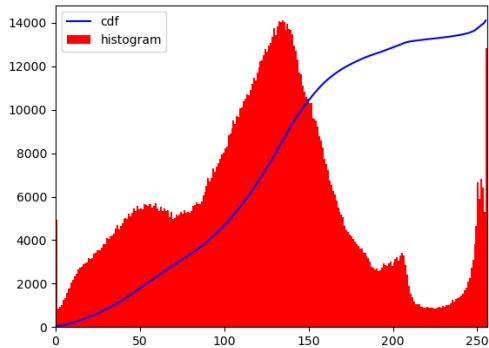
**Figure 7:** Image restaurée par l'algorithme



**Figure 8:** Image couleur originale

### 4.1 Contraste - Egalisation d'histogramme

Pour cela, il est intéressant de regarder les histogrammes des images originales contre celles colorisées (fig. 9).



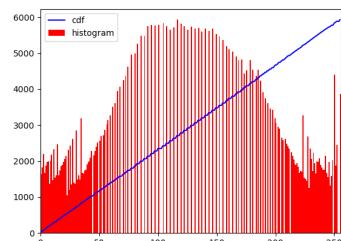
**Figure 9:** Histogrammes Original et de l'image colorisée

Nous pouvons voir une perte remarquable de dynamique dans l'image avec la colorisation. Une idée d'amélioration serait alors d'appliquer une méthode d'égalisation d'histogramme. Cela permettrait de redonner une certaine dynamique à l'image. Une autre possibilité serait d'appliquer une méthode CLAHE à l'image, comme celle que certains d'entre nous

ont vu en projet d'IMA201 (dans le cas des images sous-marines), mais appliquer cet algorithme paraissait trop puissant par rapport à ce qu'on espérait obtenir.



**Figure 10:** Image Colorisée



**Figure 11:** Histogramme égalisé



**Figure 12:** Image restaurée avec histogramme égalisé

Ce nouveau contraste donne des résultats mitigés. D'une certaine façon, nous nous approchons des originaux mais nous en éloignons aussi dans le même temps. Il serait plus pertinent de modifier l'histogramme cumulé de manière plus libre, plutôt que d'en faire une droite. L'égalisation d'histogramme n'est sans doute pas un post-processing adapté ici car apporte une trop grande modification et ne l'utiliserons donc pas ensuite. Cependant, cela nous a permis de comprendre que la différence principale avec l'originale vient de la couleur et du manque de saturation. Plutôt que de continuer à travailler sur le contraste, ce sera donc la prochaine étape afin d'obtenir les résultats les plus fidèles possibles.

## 4.2 Saturation



**Figure 13:** Image colorisée



**Figure 14:** Image colorisée et saturée



**Figure 15:** Image originale

Malheureusement, même si les résultats continuent de s'améliorer, nous atteignons ici une limite infranchissable avec de simples post-processings. Tandis que l'herbe (qui est le composant principal de l'image) est beaucoup plus proche de l'image originale, les autres éléments comme les roches et le chemin s'éloignent de l'image originale (le vert saturé de

l'herbe va déteindre notamment). Et pourtant, l'image saturée a été obtenue en modifiant séparément les saturations des différents canaux à l'aide du logiciel de traitement d'images Photoshop. Une manière d'automatiser ce processus serait d'utiliser un modèle de machine learning, qui minimiseraient une loss (mse par exemple) entre l'image originale et l'image modifiée en changeant la saturation. Cependant, même avec ce genre de système, nous n'obtiendrions jamais des images exactement comme les images originales avec ces post-processings, en démontre notamment la présence d'artefacts imprévus dans l'image colorisée ( par exemple la présence de tâches bleues). Cependant, en fusionnant un bon entraînement du réseau de colordnet avec ces pré et post processings, nos résultats finaux restent très satisfaisants.

## 5 Effets de la méthode de décolorisation des images sur la colorisation par l'algorithme

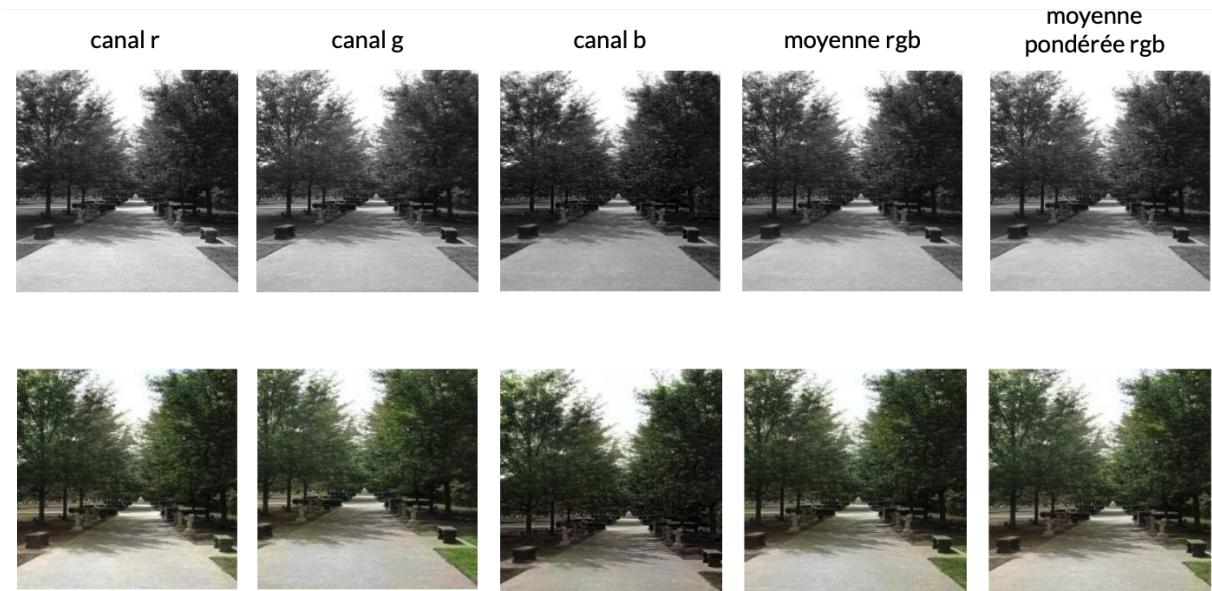
Notre algorithme est à ce stade capable de coloriser avec un certain succès des images proches des images issues des classes utilisées à l'entraînement. Nous nous interrogeons dans cette partie à l'effet de la méthode de décolorisation des images de validation.

En effet, il y a beaucoup de méthodes différentes pour obtenir une image en nuance de gris. Voici celles que nous avons étudié :

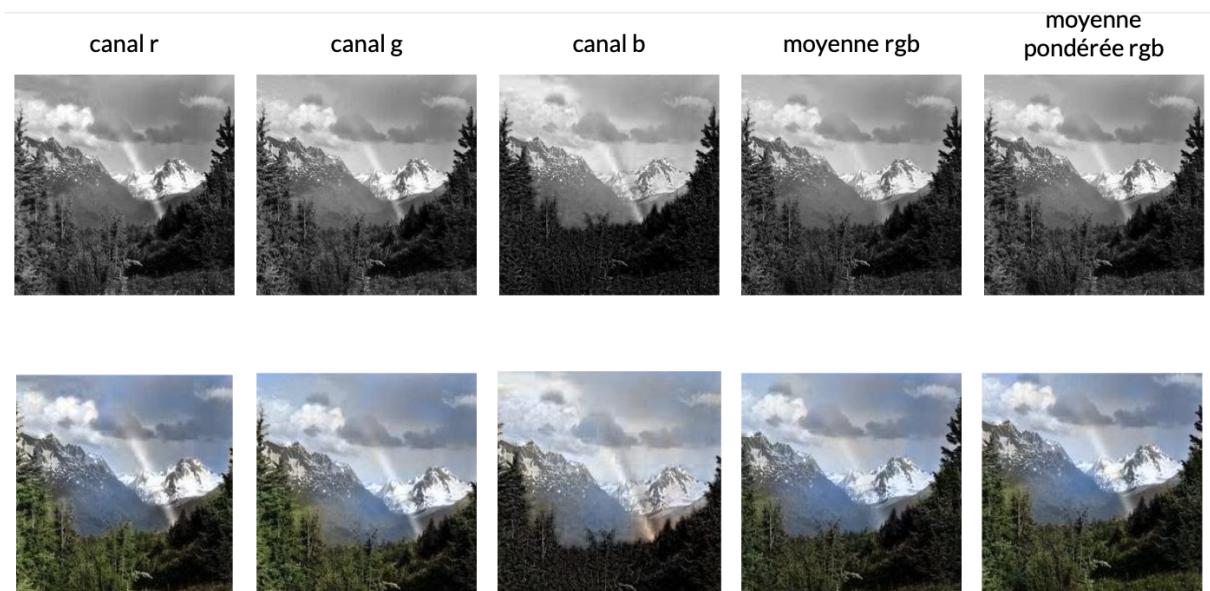
1. Conserver uniquement le canal rouge d'une image RGB
2. Conserver uniquement le canal vert d'une image RGB
3. Conserver uniquement le canal bleu d'une image RGB
4. Faire la moyenne des 3 canaux d'une image RGB
5. Faire la moyenne pondérée des 3 canaux d'une image RGB selon la formule :

$$x = (0.2125 * r + 0.7154 * g + 0.0721 * b)$$

Nous avons recolorisé les différentes images en nuances de gris que nous avons créée à l'aide de l'algorithme préalablement entraîné. Voici nos résultats :



**Figure 16:** Images en nuances de gris d'une photo de parc colorisées



**Figure 17:** Images en nuances de gris d'une photo de montagnes

### Observations :

Tout d'abord, il semble que la méthode utilisée par les créateurs du dataset pour décoloriser les images était celle de la moyenne pondérée. En effet, les images colorisées à partir des images en nuances de gris obtenues avec la moyenne pondérée des canaux RGB

sont identiques aux prédictions que Colornet fait avec les images en nuances de gris de la base de donnée. Ceci n'est pas surprenant puisque c'est la méthode qu'utilise la fonction `rgb2gray` de la librairie `skimage.color`.

Ensuite, on note quelque chose de très intéressant : on pourrait s'attendre à ce que la colorisation des images issues de la singularisation des canaux chromatiques soit de mauvaise qualité car elle ne contiennent pas d'informations sur les autres canaux chromatiques, mais il n'en est rien. Notre algorithme donne des images colorisées très proches de celles réalisées à partir des images créées par la méthode moyenne pondérée.

### **Interprétation :**

Les images que nous avons décolorisé sont des images naturelles et obéissent plutôt bien à l'hypothèse du monde gris. Ainsi, les valeurs des canaux rouges, verts et bleus sont en général assez corrélées, et les différentes images en nuances de gris ne sont pas très différentes. L'algorithme est donc capable d'obtenir de l'information contextuelle et de bien coloriser les images, sans avoir besoin de l'information directement issue des autres canaux.

Cela laisse la porte ouverte à une possibilité : puisqu'un seul canal chromatique suffit à coloriser une image, alors pourquoi ne pas essayer avec un canal chromatique correspondant à une longueur d'onde hors du domaine du visible?

**C'est ce que nous allons à présent étudier avec l'application de l'algorithme à des images NIR.**

## 6 Utiliser Colornet sur des images "Near-Infra-Red"

L'utilité de Colornet s'étend plus loin que son but initial. Dès le départ, nous avions plusieurs idées de d'extension du projet, de domaines dans lesquels Colornet serait un véritable ajout. Nous pensions par exemple à la colorisation de vidéo et du problème de cohérence entre les images consécutives d'un film. Bien que nous n'ayons pas développé cette idée, c'est un autre domaine qui nous a intéressé : la colorisation d'images NIR (Near-Intra-Red). La partie précédente nous a introduit l'intérêt de cette idée que nous allons essayer de développer maintenant.

Résoudre ce problème reviendrait à créer un système de vision nocturne, ce qui serait très utile pour des caméras de surveillances par exemple. De plus, Colornet semble être une véritable solution car les images NIR sont constituées d'un seul canal à la manière des images noires et blanches que le réseau de neurones prend habituellement en entrée. Après quelques recherches, nous avons découvert qu'une majorité des projets de recherches sur le sujet faisait intervenir le Deep Learning, et l'architecture complexe de Colornet semble être particulièrement adaptée à ce type de problème (parfois même plus que l'architecture de certains papiers).

### 6.1 Essai sur le réseau déjà entraîné

Avant toute chose, nous avons essayé de coloriser nos images NIR à partir de notre colornet déjà entraîné. Les résultats sont souvent satisfaisants, ce qui montre qu'un canal NIR est souvent équivalent à un canal RGB dans le domaine du visible. Cependant, il y a plusieurs situations où cette similarité semble atteindre ses limites et fausse les résultats (fig. 18). C'est pourquoi nous allons par la suite chercher à adapter notre code pour ré-entraîner un modèle prenant des images NIR en entrée.



**Figure 18:** Exemples d'images faussement prédites sorties du réseau depuis des images NIR

Afin d'adapter notre code à cette application, de nombreuses modifications ont dû être réalisées, et pour cela, plusieurs idées différentes nous sont venues.

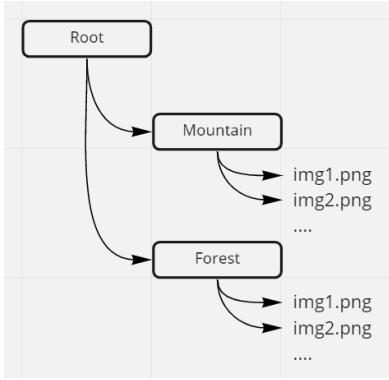
## 6.2 Modification du code pour le nouveau dataset

Une idée était de considérer les images chargées comme des images à 4 canaux, les L,a et b classiques ainsi que le canal correspondant à l'image NIR en canal  $\alpha$ . Après l'avoir entièrement codé, nous avons obtenu des résultats unicolors et n'arrivions pas à obtenir autre chose. Ce n'est que pendant la relecture finale du code, après rédaction du rapport et fin des expérimentations que nous nous sommes rendus compte que c'était une simple erreur de normalisation... Pour la suite, nous avons donc utilisé une autre idée, plus complexe, expliquée ci-dessous, mais avec une grande valeur pédagogique.

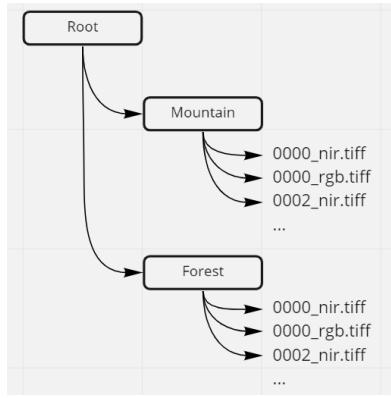
Dans le code original, les datasets sont chargés à partir des "ImageFolder datasets" et "DataLoader" de pytorch, qui sont des classes que nous ne connaissons pas très bien. Notre idée originale ne modifiait que très légèrement le dataset d'image, et ne prenait donc pas du tout en compte le changement d'organisation du dataset. La classe de Pytorch sait uniquement gérer les organisations du type de la figure 19.

Pour ce projet, il a fallu trouver une base de données d'images NIR couplées à leur image RGB. Assez étonnamment, le choix de base de données n'était pas très large et nous avons alors utilisé le dataset de l'EPFL "*RGB-NIR dataset CVPR11*". Cette base de données possédait une architecture incompatible avec celle décrite précédemment (fig.20). Le but a alors été d'abandonner la classe ImageFolder de Pytorch pour créer une classe personnalisée prenant en charge notre dataset. Il a fallu pour cela apprendre l'utilisation de ce genre de classe avec notamment le fonctionnement des fonctions `__getitem__` ou encore `__len__`. Sans rentrer dans les détails (le code est disponible dans le dossier `code_nir_2`, dans le fichier `dataset.py`), les principales difficultés ont résidé dans la lecture conjointe de 2 images pour chaque index dans le dataset, ainsi que dans la modification de nombreuses fonctions afin d'effectuer les transformations sur les 2 images conjointes à chaque fois (notamment le cropping). Il a aussi fallu changer l'organisation du dataset à l'aide de la fonction `modif_dataset` (fig.21).

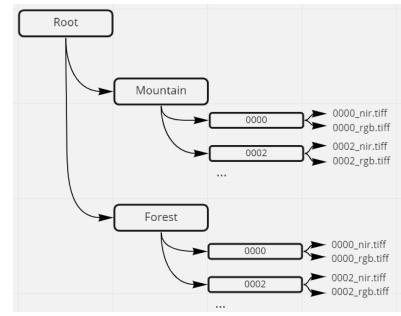
Un autre point important de cette classe personnalisée de prise en charge du dataset était la prise en charge des labels. La classe originale utilisait le système d'héritage et interprétrait directement les noms des dossiers comme labels. Plus important encore la méthode de Pytorch s'occupait même de transformer les mots en vecteurs afin de pouvoir appliquer les calculs de loss ensuite sur les labels. Ainsi, après avoir développé notre



**Figure 19:** Arborescence Acceptée par la classe ImageFolderDataset de Pytorch



**Figure 20:** Arbo du dataset nir-rgb



**Figure 21:** Nouvelle Arbo du dataset

propre classe, le système de labels n'était plus utilisable car même en extractant ceux-ci, la cross-entropy loss ne pouvait pas se calculer sur les chaînes de caractères. Initialement, nous avons enlevé cette partie du calcul de la loss, afin d'obtenir des résultats. Après beaucoup d'essais à base de one-hot-encoding et même de word embeddings, nous avons décidé de laisser de côté ce problème pour se concentrer sur l'obtention de résultat à force d'enchaîner les erreurs et de ne pas avancer.

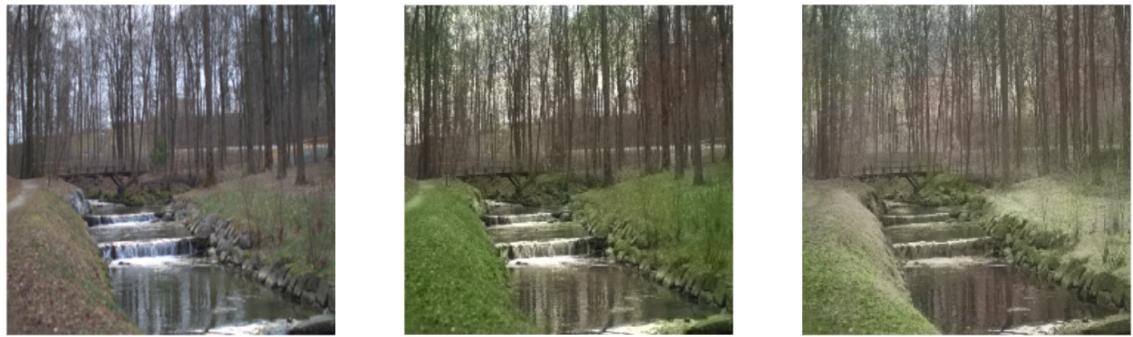
Remarque : Il est important de noter que la base de données utilisée est relativement petite avec en tout 469 images pour 9 classes (soit une bonne cinquantaine d'images par classe), ce qui est vraiment peu pour espérer obtenir des bons résultats.

### 6.3 Utiliser l'image NIR comme luminance

Ensuite, après avoir réussi à faire fonctionner le nouveau programme de prise en charge de ce nouveau dataset NIR, il a fallu adapter ce nouveau Colornet pour obtenir des résultats. Plusieurs difficultés et questions se sont alors posées. La plus importante était sur la luminance. Le colornet original donnait en sortie du réseau neuronal les canaux a et b de l'image, et il suffisait de concaténer ces canaux à la luminance L qui correspondait à l'image en noir et blanc. Cependant, dans le cas des images NIR, nous n'avons pas accès à ce canal et plusieurs possibilités s'offraient alors à nous.

Tout d'abord, nous avons utilisé l'image NIR comme canal L dans l'image finale. Nous savions que ce n'était pas la bonne solution car, bien qu'il y ait souvent un lien entre les images NIR et les images originales (comme nous l'avons dit précédemment), ces deux canaux ne sont clairement pas les mêmes. Cependant, cela nous a permis d'apercevoir nos premiers résultats avec ce code adapté et de ainsi de le débugger. Voici quelques

exemples obtenus après un entraînement sur des classes botaniques (fig. 22). Il semble y avoir une certaine cohérence dans les couleurs (malgré une colorisation générale très pauvre). Cependant, les résultats ne sont pas toujours bons, avec certains artefacts ou couleurs étonnantes, et **c'est tout à fait normal** puisque nous avons utilisé l'image NIR comme luminance. Il est important de noter que cette étape est **différente des images précédentes** en 6.1, qui avaient été obtenues en donnant une image NIR au colonet entraîné avec des images RGB. Ici, l'entraînement a été réalisé avec la combinaison des images NIR et RGB et non seulement les images RGB (c'est-à-dire qu'à chaque étape de l'entraînement, l'algorithme essayait de minimiser la loss entre l'image  $L_{RGBab}$  originale et la concaténation de  $L_{NIR}$  et ab de sortie du réseau.



**Figure 22:** Gauche : Image Originale, Milieu : Colnet depuis l'im RGB, Droite : Colnet depuis l'im NIR

## 6.4 Modifier Colnet

La deuxième solution, plus complexe mais bien plus logique, a été de modifier le réseau afin de prédire aussi le canal L. Nous avons essayé de simplement modifier le nombre de channels en sortie du réseau de neurones, mais les résultats sont étrangement totalement gris... Après avoir modifié le réseau de telle manière à sortir la luminance de sa propre "branche" du réseau (en utilisant le modèle du réseau de prédiction de a et b), on obtient des résultats très intéressants. Avec un entraînement très simple (30 epochs, batch size de 32, pour les 469 images évoquées précédemment) mais donc aussi incomplet, le résultat obtenu est "cohérent" mais très flou (fig. 23). Nous arrivons cependant à voir les points communs entre l'image originale et cette image floue, ce qui nous assure qu'il y ait bien une logique dans le processus de colorisation du réseau neuronal.

Trois hypothèses au flou nous sont venues : d'un côté, l'entraînement n'était pas assez complet (c'est une certitude). De plus, le dataset est probablement trop petit. Enfin, nous avons remarqué que notre loss, non modifié, prenait en compte les canaux L, a et b avec la même importance, ce qui peut être un problème car L est censé avoir une importance



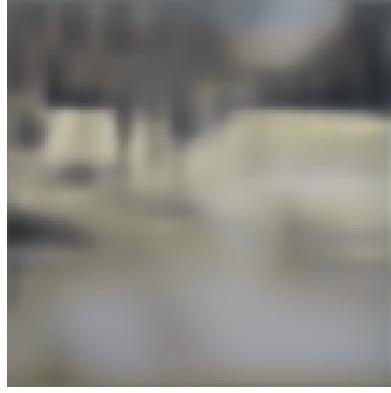
**Figure 23:** Image Colorisée avec le canal NIR comme luminance

particulière par rapport à a et b.

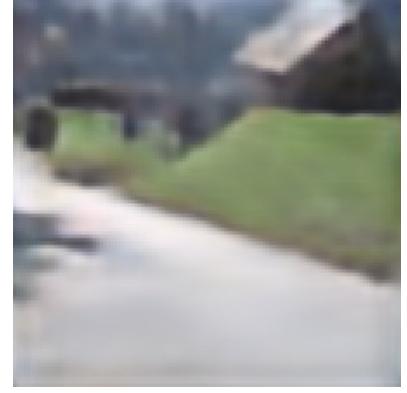
En affichant séparément les loss des différents canaux, nous nous sommes rendus compte que la loss de L était déjà prépondérante par rapport à celles des canaux a et b. Le flou ne venait donc pas de là, et la prochaine étape était donc logiquement de lancer un meilleur entraînement. C'est donc ce que nous avons fait et les résultats obtenus sont, comme prévus meilleurs (fig. 26).



**Figure 24:** Image Originale



**Figure 25:** Image prédite avec entraînement rapide



**Figure 26:** Image prédite avec bon entraînement

Bien que les images soient encore floues, la colorisation est clairement cohérente et très prometteuse. Nous pouvons voir notamment que les couleurs de la route, de l'herbe et de la cabane sont bien respectées. Nous pouvons supposer que, avec un meilleur entraînement et un meilleur dataset, il y aurait convergence. Cependant, puisque la prédiction de L est particulièrement importante pour obtenir une image proche de l'originale, la convergence est beaucoup plus longue et nous n'avons pas réussi à obtenir de meilleurs résultats que ceux présentés ici, malgré des entraînements relativement longs et complexes. En regardant les loss au cours des epochs de cet entraînement, celle-ci semblait bien continuer à décroître. Il faut aussi noter que l'ajout de la branche de prédiction L a rajouté de nom-

breux paramètres et que chaque epoch est plus longue que pour le colornet original. Mais malgré le flou, les artefacts qu'on observait en utilisant les images NIR comme luminance semblent disparaître. Notamment, certains artefacts apparaissaient dans l'eau et certains ciels à cause de la mauvaise luminance, ce qui n'est pas le cas ici, c'est donc un a priori très positif sur cette technique de modification de colornet (fig. 27). Les seuls artefacts apparaissants sont ceux de l'image en bas à gauche de la figure 27, qui sont difficilement explicables. Il est cependant probable que sur un meilleur entraînement, ces artefacts disparaîtront afin de minimiser la loss.

Nous pouvons déduire de cette supposée convergence, que Colornet peut avoir bien plus d'utilité que de simplement coloriser les images noires et blanches. A partir d'images infra-rouges, nous pouvons retrouver des images couleurs. Ainsi, en continuant à supposer une convergence vers des images nettes, nous pouvons alors imaginer énormément de nouveaux scénarios pour lequel ce nouveau Colornet serait utile, comme par exemple, dans le cas de colorisation d'images astronomiques obtenues par rayons X, voir même la colorisation d'images médicales dans un futur proche, le problème majeur étant comme souvent la récupération d'un dataset d'entraînement.



**Figure 27:** Quelques exemples de résultats

## 7 Conclusion

Au cours de ce projet, nous avons donc pu voir de nombreux aspects du réseau de neurones Colornet. Nous avons commencé par essayer de comprendre le réseau, ce qui s'est étonnamment révélé plus difficile que prévu, de par l'absence de poids pré-entraînés. Cela nous a alors forcés à utiliser les poids pickle fournis, et donc de recoder de manière simplifiée le réseau (et uniquement le réseau). Cela nous a permis de bien comprendre le modèle. L'étape suivante a donc été d'effectuer nos propres entraînements et d'expérimenter sur le modèle, avec notamment le changement de loss à partir de la "colorfulness". De cette étape nous avons pu obtenir nos premiers résultats satisfaisants. Ensuite, nous avons voulu améliorer ces résultats justement pour les approcher des images originales en utilisant différentes techniques de post-processings. Finalement, nos résultats étaient assez satisfaisants et nous sommes donc passés à la deuxième grande partie du projet : l'application aux images NIR.

Cette étape a été assez difficile sur le plan technique, nous avons rencontré beaucoup de problèmes à gérer des datasets différents de ce qui nous avait été initialement fournis, mais nous avons finalement réussi à obtenir des résultats satisfaisants. La méthode consistant à utiliser les images NIR comme luminance donne des résultats relativement correctes, mais qui restent faux de par leur nature. Au contraire, l'idée de modifier Colnet pour prédire L est intrinsèquement la meilleure solution, mais demande beaucoup plus de puissance de calcul et de temps, comme nous avons pu le voir avec nos résultats flous. Nous restons cependant assez convaincus que cette méthode, avec un meilleur entraînement et un dataset plus complet donnerait des résultats très bons et serait une bonne piste vers de nombreuses autres utilisations très intéressantes.

Voici quelques pistes d'améliorations afin d'obtenir de meilleurs résultats avec le réseau Colornet :

- Nous avons vu qu'un des problèmes étaient que les couleurs prédites étaient souvent plus ternes que dans l'image originale. Une solution évoquée serait de modifier de nouveau notre fonction de loss afin de favoriser une plus grande dynamique de couleurs.
- Un autre problème était celui de l'overfitting. Bon nombre d'images prédites possédaient des tâches bleues plus ou moins visibles dues à une tendance à prévoir trop de ciel dans les images. Nous pourrions essayer de modifier de nouveau notre réseau Colnet en ajoutant des couches de DropOut et de BatchNormalization afin de réduire l'overfitting.

- Pour éviter d'avoir à entraîner Colornet sur toutes les 365 classes dans le but d'obtenir des résultats similaires à ceux de la publication, on pourrait étudier la colorimétrie des classes d'images pour dégager des "superclasses". Par exemple, on pourrait s'attendre à ce que un jardin et une foret soient dans la même superclasse nature, mais pas dans la même superclasse qu'une scène urbaine. On pourrait alors dégager les classes les plus représentatives de ces superclasses, et entraîner Colornet sur ces classes. On s'attend alors à avoir un algorithme ayant des capacités de généralisation importantes, pour un moindre effort d'entraînement.