

Travail de Bachelor

La Terre vue de nuit



Auteur :

Matthieu Burguburu

Superviseur :

Prof. Andres Perez-Uribe

Table des matières

1	Préambule	1
2	Résumé	2
3	Cahier des Charges	3
3.1	Objectifs	3
3.1.1	Étapes principales	3
3.1.2	Résultats	3
3.1.3	Livrables	3
3.1.4	Calendrier du projet	4
4	Introduction	5
4.1	Problématique et contexte	5
4.2	Outils	5
4.3	Installation	6
4.4	Hypothèses et exigences	6
4.4.1	Corrélation entre la structure d'un village et la lumière émise	6
4.4.2	Modèles de prédictions	6
5	Données	7
5.1	Sources de données	7
5.1.1	Images de nuit	7
5.1.2	Images de jour	7
5.1.3	Synthèse et choix	8
5.2	Méthodes d'acquisition	8
5.2.1	Acquisition Manuelle	8
5.2.2	Acquisition par API	9
6	Préparation des données	10
6.1	Résolution spatiale	10
6.1.1	Changement de résolution par interpolation	10
6.1.2	Meilleur choix des données	11
6.2	Bruits numériques	12
6.2.1	Nuages	12
6.2.2	Reflets	13
6.2.3	Senseur	14
6.3	Format des fichiers	15
6.3.1	Fichiers .jp2	15
6.3.2	Fichiers .h5	15
6.4	Système de coordonnées	16
6.4.1	WGS4 - Données de nuit	16
6.4.2	MGRS - Données de jour	17
6.5	Découpage des données	18
6.5.1	Méthode rapide mais peu précise	18
6.5.2	Méthode lente mais précise	19
6.6	Structure du dataset	20
6.7	Normalisation des données	20
6.7.1	Images	20
6.7.2	Radiances	21
6.7.3	n% cut	21

7 Réseaux de Neurones Convolutionnels	22
7.1 Limitations	22
7.1.1 Limitations en temps	22
7.1.1.1 Arrêt de l'entraînement	22
7.1.2 Limitations physiques	22
7.1.2.1 RAM	22
7.1.2.2 GPU & VRAM	23
7.1.2.3 Machine Learning dans le Cloud	23
8 Keras	23
8.1 Hypersensibilité aux paramètres	23
8.2 Topologies des CNNs	24
8.2.1 Filtres Conv2D	24
8.2.2 Pooling	24
8.2.3 Dense layers	24
8.2.4 Dropout	25
8.2.5 Fonctions d'activation	25
8.3 Optimizers	26
8.4 Fonctions de coûts	26
8.4.1 Adam	26
8.4.1.1 Adaptive Learning rate	27
8.5 Cross-validation	27
8.5.1 Séparation Uniforme vs Aléatoire	27
8.5.2 K-Fold validation	28
9 Application des CNNs	28
9.1 Visualisation des performances & prédictions	29
9.2 Optimisation des topologies et hyper-paramètres	29
9.3 Modèle final	30
9.3.1 Topologie & Hyper-paramètres	30
9.3.2 Résultats	31
9.3.2.1 Distribution des radiances 1D	31
9.3.2.2 Correlations avec les <i>Ground Truths</i>	31
9.3.2.3 Distribution des radiances 2D	32
9.3.2.4 Différences avec les <i>Ground Truths</i>	33
9.3.2.5 Binarisation des résultats	34
10 Discussion des résultats	36
11 Améliorations possibles	38
11.1 Meilleure résolution spatiale	38
11.2 Comparaison des résultats avec d'autres jeux de données	39
11.3 Cloud Computing	39
11.4 Augmentation des données	39
12 Conclusion	41
Bibliographie	42
13 Authentification	44

1 Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 30 juillet 2021.

2 Résumé

Dans le monde, 1 personne sur 10 n'a pas accès à l'électricité. Afin de pouvoir apporter de l'aide à ces personnes, il est nécessaire d'avoir des données de recensement précises et à jour. Cependant, de telles données sont rares dans les pays défavorisés. Nous nous concentrerons sur la région du Sokoto, Nigeria.

Nous cherchons donc à générer des données équivalentes en utilisant une approche basée sur le Machine Learning, et l'abondance de données satellites librement accessibles capturées en continu. Nous nous intéressons particulièrement à des images de nuit capturant la luminosité ambiante au sol, en les utilisant comme proxy de présence humaine.

Une fois correctement traitées et préparées, ces données servent à entraîner un réseau de neurones convolutionnel prédisant la luminosité émise par différents emplacements. Le résultat souhaité prend la forme des zones habitées prédites comme émettant de la lumière par le modèle, mais n'émettant en réalité aucune lumière. Ces emplacements sont les villages défavorisés que l'on cherche à détecter.

3 Cahier des Charges

3.1 Objectifs

3.1.1 Étapes principales

Le travail est divisé en trois phases principales.

La première, consiste en une phase exploratoire visant à sélectionner les données de meilleure qualité ainsi que les algorithmes de Machine Learning les plus adaptés.

La seconde vise en la réalisation et l'entraînement des réseaux de neurones pour apprendre à reconnaître une zone habitée. Ils prendront en entrée des images satellites de jour, et s'entraîneront à prédire en sortie les images satellites de nuit comme proxy de présence humaine, et ensuite détecter les zones "potentiellement habitées" où il n'y a pas de lumière dans les images de nuit.

Finalement, la dernière phase consiste à développer des outils permettant de visualiser ces résultats, ainsi que rédiger un rapport documentant chacune de ces étapes et commentant les résultats obtenus.

3.1.2 Résultats

Les résultats attendus de ce travail sont les suivants :

Premièrement, des exemples d'applications des modèles développés sur des zones composées de villages n'ayant généralement pas accès à l'électricité, ainsi que un ou plusieurs moyens de visualiser ces résultats. Puisque le thème de ce sujet est très expérimental et tient plus d'un travail de recherche, la taux de succès de ces résultats n'est pas la partie importante, même si développer un modèle efficace serait préférable.

Deuxièmement, le travail de recherche devra être documenté afin de montrer les différentes approches envisagées et utilisées pour arriver à ces résultats, ainsi qu'expliquer pourquoi certaines approches ont été préférées à d'autres.

3.1.3 Livrables

Un certain nombre de livrables seront fournis à l'issue de ce travail. Ils incluent :

- Le rapport final
- Des instructions pour mettre en place l'environnement de développement nécessaire
- Un ensemble de scripts et notebooks pythons, permettant de :
 - Télécharger les données brutes
 - Traiter ces données
 - Construire le dataset
 - Entraîner les modèles avec ces données
 - Tester la performance des modèles entraînés et pré-entraînés
 - Visualiser les résultats
- Différents modèles pré-entraînés produisant les meilleurs résultats atteints
- Le lien vers un repository GitHub public contenant l'ensemble de ces documents.

3.1.4 Calendrier du projet

Le calendrier officiel des travaux de Bachelors pour l'année 2021 est le suivant :

Date	Tâche
17 mai 2021	Remise du rapport intermédiaire
21 juin au 30 juillet 2021	Travail à temps plein
30 juillet 2021	Remise du rapport final et du résumé publiable
30 août au 17 septembre 2021	Soutenance du travail de Bachelor

4 Introduction

4.1 Problématique et contexte

Dans le monde, 9 personnes sur 10 ont accès à l'électricité. Bien qu'étant en croissance continue depuis longtemps, passant de 72% en 1998 à 90% en 2018 [1], cela laisse près de 1 milliard de personnes n'y ayant pas accès. En Afrique subsaharienne, on estime à 500 millions le nombre de personnes n'ayant pas accès à l'électricité.

Afin de pouvoir apporter de l'aide à ces personnes, il est donc nécessaire de savoir où elles habitent, au moyen d'études de recensement. Cependant, de telles études sont effectuées bien moins souvent dans les régions qui nécessitent de suivre l'évolution de l'accès à l'électricité dans le temps que dans celles où l'électricité est disponible pour tous [2].

Ceci rend extrêmement difficile la mesure l'évolution précise de l'accès à l'électricité dans ces régions. Nous souhaitons donc produire des données permettant de combler ce manque, à partir de données accessibles à un intervalle très régulier, particulièrement les images satellites NRT (Near Real Time).

Grâce à l'apprentissage automatique (ci-après "Machine Learning" ou "ML"), il est possible d'entraîner un réseau de neurones pour générer un modèle de prédiction prenant en entrée des images satellites. En l'entraînant au moyen de données de qualité disponibles seulement à certains endroits et à certains moments, on peut généraliser les prédictions à des lieux n'ayant pas de données disponibles.

4.2 Outils

Un grand nombre d'outils seront utilisés dans le cadre du projet. Cependant, seuls certains d'entre eux seront essentiels à son bon fonctionnement et seront mentionnés à plusieurs reprises par la suite. La liste non-exhaustive suivante mentionne les plus importants :

- **Python** : Python sera le langage principal utilisé au cours de ce projet, en version 3.8.
Les paquets suivants seront les plus utilisés :
 - **JupyterLab** : Permet d'exécuter du code python dans un notebook en visualisant les résultats d'exécution intermédiaires, ainsi que d'y ajouter des commentaires au format Markdown
 - **NumPy** : Permet d'effectuer un grand nombres d'opérations efficacement sur des matrices à hautes dimensions
 - **RasterIO** : Permet de traiter toutes sortes de rasters, des images satellites dans notre cas
 - **OpenCV-Python** : Permet d'ouvrir et traiter des images dans des formats standards
 - **Matplotlib** : Permet de visualiser des données graphiquement
 - **Seaborn** : Permet aussi de visualiser des données graphiquement
 - **Tensorflow** : Framework dédié au machine learning
 - **Keras** : Framework basé sur Tensorflow permettant aussi de faire du machine learning
 - **QGIS** : Logiciel de visualisation d'images satellites. Bien que pas strictement nécessaire au fonctionnement du projet, QGIS reste un logiciel très pratique.
 - **Bash** : Shell permettant d'exécuter certains des scripts permettant de mettre en place l'environnement de développement, ou télécharger certaines données brutes.
 - **Comptes utilisateurs** : Afin de télécharger les données brutes, il est nécessaire d'être enregistré sur certains sites avant d'avoir les autorisations nécessaire pour le téléchargement :

- **Sentinel-Hub** : Nécessaire pour télécharger les images de jour manuellement en haute résolution depuis le site SentinelHub
- **Amazon Web Services** : Nécessaire pour télécharger les images de jour de SentinelHub à travers l'API depuis les scripts python. Le Free-Tier offert par Amazon suffit pour télécharger les données utilisées lors de ce projet (moins de 50% du quota de Data Transfer mensuel)
- **NASA EarthData Search** : Nécessaire pour télécharger les images de nuit manuellement ainsi qu'avec les scripts.

4.3 Installation

Afin de pouvoir reproduire les résultats obtenus au cours de ce projet, un repository GitHub a été mis en place et donne accès à l'intégralité des fichiers nécessaires. Il est disponible à l'adresse suivante : <https://github.com/matt989253/heig-TerraGazer> [3]

Pour pouvoir exécuter chacun des notebooks, un environnement Conda est recommandé. Il est aussi nécessaire d'installer pip. Le script "newenv.sh" permet d'installer l'ensemble des librairies python nécessaires, à travers pip ainsi que conda. Il est aussi techniquement possible d'installer les prérequis grâce à la commande pip install -r requirements.txt. Cependant, des erreurs peuvent être engendrées à cause des paquets uniquement disponibles sur conda.

Attention aux utilisateur n'utilisant pas Windows : Le paquet pywin32 inclus dans le script et les requirements, est exclusif à Windows, et doit être désactivé avant de poursuivre sur d'autres systèmes d'exploitation. Cependant, le projet n'a été testé que sur Windows 10.

Une fois les paquets installés, les notebooks peuvent être lancés grâce à la commande jupyter-lab. Les notebooks sont numérotés dans l'ordre logique de leur exécution au cours du projet afin de recréer les résultats obtenus.

4.4 Hypothèses et exigences

Un certain nombres d'hypothèses et exigences sont nécessaires pour permettre le bon fonctionnement de ce projet. Les hypothèses et exigences principales sont les suivantes. Toute autre hypothèse ou exigence moins importante sera mentionnée lorsque nécessaire par la suite.

4.4.1 Corrélation entre la structure d'un village et la lumière émise

Ce projet se base sur la lumière émise et mesurée par satellite la nuit. Ces données servent de proxy de la présence humaine, puisque de manière générale, lorsque suffisamment de lumière est émise la nuit pour être détectée par un satellite, cette lumière est artificielle et produite par des humains habitants à proximité.

Puisque nous souhaitons détecter des villages qui n'émettent pas de lumière, mais qui "ressemblent" à ceux qui en émettent, nous partons du principe que ces types de villages auront suffisamment de similarité pour pouvoir être détectés comme "villages qui émettent de la lumière" par les modèles de deep learning.

L'hypothèse devant être faite est donc la suivante : **La corrélation entre la structure d'un village et la lumière qu'il émet la nuit n'est pas parfaite, et ces villages ne sont suffisamment similaires pour pouvoir être confondus.**

4.4.2 Modèles de prédictions

Pour la même raison, les modèles de prédictions générés nécessitent une particularité assez peu commune dans le domaine du machine learning. Même si le modèle est capable d'avoir un

"score" parfait de prédiction de la lumière sur le jeu de données d'entraînement, il est préférable et nécessaire que les prédictions sur le jeu de données de tests (les généralisations) ne soient pas parfaites. En effet, un modèle qui recréerait parfaitement les données de nuit ne nous apporterait aucune information supplémentaire. Les données qui nous intéressent sont les zones prédites comme lumineuses, qui n'émettent en vérité pas de lumière.

L'exigence devant être posée est donc la suivante : **Les modèles de prédictions générés au cours du projet n'auront pas ou ne pourront pas avoir des capacités de généralisations parfaites.**

5 Données

5.1 Sources de données

Plusieurs sources de données différentes ont été considérées pour ce projet.

5.1.1 Images de nuit

La première source de données, pour les images de nuit, est le "Black Marble Nighttime Lights Product Suite" de la NASA. Les données peuvent être téléchargées sur le site <https://search.earthdata.nasa.gov> avec la référence "C1897815356-LAADS"¹ [4]. La NASA propose un deuxième site, <https://worldview.earthdata.nasa.gov/> permettant de visualiser ces données de manière plus simple et avec un rendu graphique plus agréable, mais il ne permet pas de télécharger les données visualisées [5].

Deux variantes similaires mais très légèrement différentes sont disponibles. La première, correspondant à la référence mentionnée, est nommée "VNP46A1". L'autre source est nommée "VNP46A2". La différence entre ces deux variantes est la correction atmosphérique appliquée sur "VNP46A2" [6]. Après inspection, la différence apportée par la correction atmosphérique entre ces deux produits semble suffisamment minime pour ne pas impacter les résultats, puisque nous n'utiliserons que des données capturées au même moment et dans une zone géographique très proche.

De ce fait, la source préférée sera celle la plus proche des données brutes possibles, "VNP46A1". Ceci nous permet d'éviter certains artéfacts qui apparaissent rarement après la correction sur l'autre suite du produit. Ces données sont disponibles à une résolution de 15 arc secondes (environ 500m) depuis Janvier 2012 [6].

5.1.2 Images de jour

Pour les images de jour, plusieurs sources de données différentes ont été envisagées.

La première source envisagée fut le "Harmonized Landsat Sentinel-2 (HLS)". Ces données, aussi trouvées sur le site <https://search.earthdata.nasa.gov> avec la référence "HLSS30"², sont un composite des données des satellites Landsat ainsi que Sentinel-2. Ces données sont disponibles à une résolution de 30m par pixels, potentiellement avec des nuages, ainsi qu'une correction atmosphérique. Il a cependant très vite été clair que cette résolution n'était pas suffisante pour obtenir de bons résultats, et que des sources avec une meilleure résolution seraient nécessaires.

La seconde source envisagée fut le "Landsat-7". Les données peuvent être téléchargées sur le site code.earthengine.google.com, sous le label "Landsat-7"³. Ces images sont disponibles à une

1. <https://go.nasa.gov/3eY12tI>

2. <https://go.nasa.gov/3BH7L4R>

3. <https://bit.ly/3ryowuC>

résolution de 10m par pixel depuis Janvier 1999 [7]. Cependant, le satellite a subi une panne partielle en 2003 et produit depuis des images contenant des artefacts, et sera bientôt remplacé par un satellite de nouvelle génération, "Landsat-9" [8].

La dernière source de données, et celle qui sera retenue, est le "Sentinel2-L2A", de Sentinel Hub avec des contributions de ESA. Les données peuvent être téléchargées sur le site apps.sentinel-hub.com/eo-browser, sous le label "Sentinel2", avec le sous-label "L2A"⁴ qui permet d'accéder à des données avec une correction atmosphérique disponibles à une résolution de 10m par pixel. De plus, le service proposé par sentinel-hub permet de filtrer les données par couverture nuageuse et par date. [9].

Une source de données potentielle aurait été les images disponibles sur "Google Maps". En effet, à première vue, ces données ont une excellente résolution spatiale pouvant descendre jusqu'à 15cm par pixel, et ne comportent en plus aucun nuages. Cependant, elles ne correspondent pas aux critères spécifiés dans le cahier des charges : Ce ne sont pas des données "Near Real Time". Elles ne permettraient donc pas de faire un suivi des évolutions jour à jour, puisqu'elles sont un composite d'images de différentes dates. Cette source de données ne sera donc pas envisagée.

5.1.3 Synthèse et choix

Les différentes sources de données mentionnées précédemment peuvent être résumées par le tableau suivant :

Satellite	Période	Résolution	Source
Black Marble	Nuit	500m	NASA
HLS	Jour	30m	NASA
Landsat7	Jour	10m	Google
Sentinel2-L2A	Jour	10m	Sentinel-Hub, ESA

TABLE 1 – Informations principales des différentes sources de données envisagées

Les sources de données finalement retenues seront le "Black Marble Nighttime Lights Product Suite VNP46A1" pour les images de nuit, et le "Sentinel2-L2A" pour les images de jour.

5.2 Méthodes d'acquisition

Plusieurs manières de télécharger les données sont proposées par chaque site, et ont divers avantages et inconvénients.

5.2.1 Acquisition Manuelle

Chacun des sites choisis pour les sources de données permettent de télécharger des données manuellement. Ils proposent une interface permettant de spécifier certains critères afin de filtrer les résultats.

Pour les images de nuit, le site <https://search.earthdata.nasa.gov>, permet de choisir une zone géographique ainsi qu'une plage entre deux dates. Un aperçu des données correspondante est alors affiché, et il est possible de sélectionner les zones souhaitées afin de les télécharger. Deux méthodes sont alors proposées : Téléchargement direct, ou génération d'un script permettant de télécharger les données sélectionnées. Les deux méthodes offrent le même résultat, mais l'utilisation du script permet de re-télécharger les mêmes données sans passer par la phase de recherche sur l'interface, qui n'est pas toujours évidente à utiliser.

4. <https://bit.ly/2WdgLhW>

Pour reproduire les résultats obtenus durant ce projet, le script "1 - blackmarble download.sh" permet de télécharger les données de nuit utilisées. Un compte enregistré sur le site <https://search.earthdata.nasa.gov> est nécessaire afin de pouvoir exécuter le script. Le fichier téléchargé doit alors être placé dans le dossier `jupyter-notebook/data/blackmarble`.

Pour les images de jour, le site <https://apps.sentinel-hub.com/eo-browser/> offre aussi une interface pour télécharger les données manuellement. Comme mentionné en section 5.1.2, on peut sélectionner différents critères permettant de filtrer les résultats, particulièrement la couverture nuageuse. Bien que très pratique pour explorer les données, la méthode manuelle n'est pas très efficace pour le téléchargement. En effet, pour télécharger les données à la plus haute résolution disponible, en plus de nécessiter un compte, il est nécessaire de "zoomer" sur la zone souhaitée. La zone couverte par les données téléchargées sera alors très petite, faisant quelques kilomètres de côté seulement. Ceci rend le téléchargement de grandes zones de données très long et fastidieux.

5.2.2 Acquisition par API

Contrairement au site <https://search.earthdata.nasa.gov>, <https://apps.sentinel-hub.com/eo-browser/> offre une API très complète permettant d'accéder à leurs données programmatiquement, grâce au package python `sentinelhub` [10]. Ce package permet d'accéder directement aux données stockées chez **Amazon Web Services**, grâce au service "S3 Requester Pays Bucket" ⁵. Ceci demande un compte **Amazon Web Services**, puisque le téléchargement de grande quantités de données peut engendrer des coûts de transferts. Cependant, la quantité de données utilisée au cours de ce projet est entièrement couverte par le "Free Tier" et n'utilise pas plus de 50% du taux de transfert mensuel maximum.

L'avantage principal de l'API offerte par Sentinel-Hub est la possibilité de télécharger des données couvrant une grande zone géographique en une fois, à la résolution maximale de 10m par pixel. Ces données sont découpées en **Tiles** de diverses formes, référencées selon le système **MGRS** (qui sera plus en détail en section 6.4), et se conformant aux longues bandes capturées par le satellite. La figure 1 illustre ce découpage.

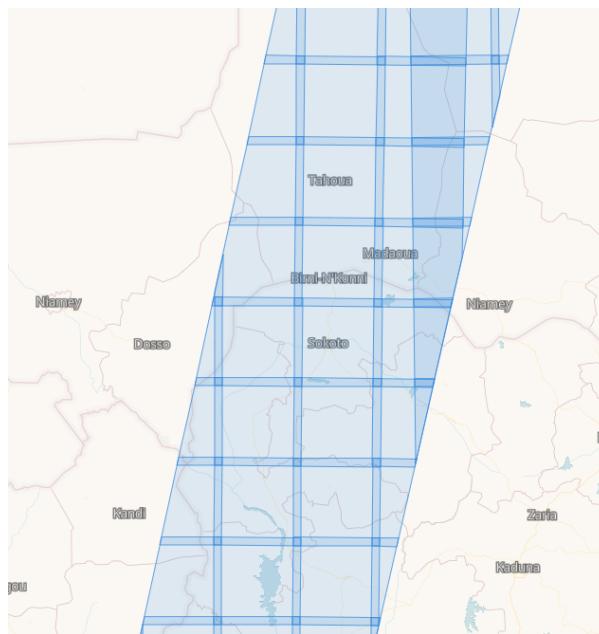


FIGURE 1 – Découpage d'une bande d'images de la région du Sokoto, capturées par le satellite Sentinel2 et visualisées le site Sentinel-Hub

5. <https://amzn.to/3y9j2ce>

Le notebook jupyter 1 - `sentinel2 download.ipynb` permet télécharger les données de jour utilisées lors de ce projet. Il nécessite de configurer les tokens d'accès pour un compte Sentinel-Hub ainsi qu'un compte AWS.

6 Préparation des données

Une fois les données brutes téléchargées, nous pouvons tourner notre attention sur la préparation de ces données en vue de leur utilisation dans les réseaux de neurones convolutionnels.

6.1 Résolution spatiale

Comme mentionné en section 5.1.2, plusieurs résolution spatiales sont disponibles pour les données de jour.

La résolution spatiale correspond à la zone couverte au sol par 1 pixel de l'image satellite produite. Cette mesure correspond généralement à la longueur de l'un des côtés du pixel en question. Elle peut être exprimée en mètres par pixels, ou en secondes (de degrés) par pixel. Si la résolution est exprimée en secondes de degrés par pixel, il est important de se rappeler que l'équivalent d'une seconde de degrés en mètres varie, en fonction de la distance à l'équateur sur l'axe des latitudes.

6.1.1 Changement de résolution par interpolation

Initialement, une résolution de 30m par pixel avait été choisie pour les données de jour, provenant de Landsat-7. Comme les données de nuit ont une résolution spatiale de 500m par pixel, il fallait un moyen de mettre ces données à la même échelle, ou du moins à une échelle compatible, puisque 30 ne divise pas 500 parfaitement.

Différentes méthodes d'interpolations avaient été envisagées afin de changer la résolution des images de jour, l'objectif principal étant de perdre le moins d'informations possible de l'image originale. Les méthodes principales envisagées étaient les suivantes :

- Interpolation par proches voisins
- Interpolation bilinéaire
- Interpolation bicubique
- Conv2DTranspose

On peut visualiser facilement les 3 premières méthodes grâce à la figure 2.

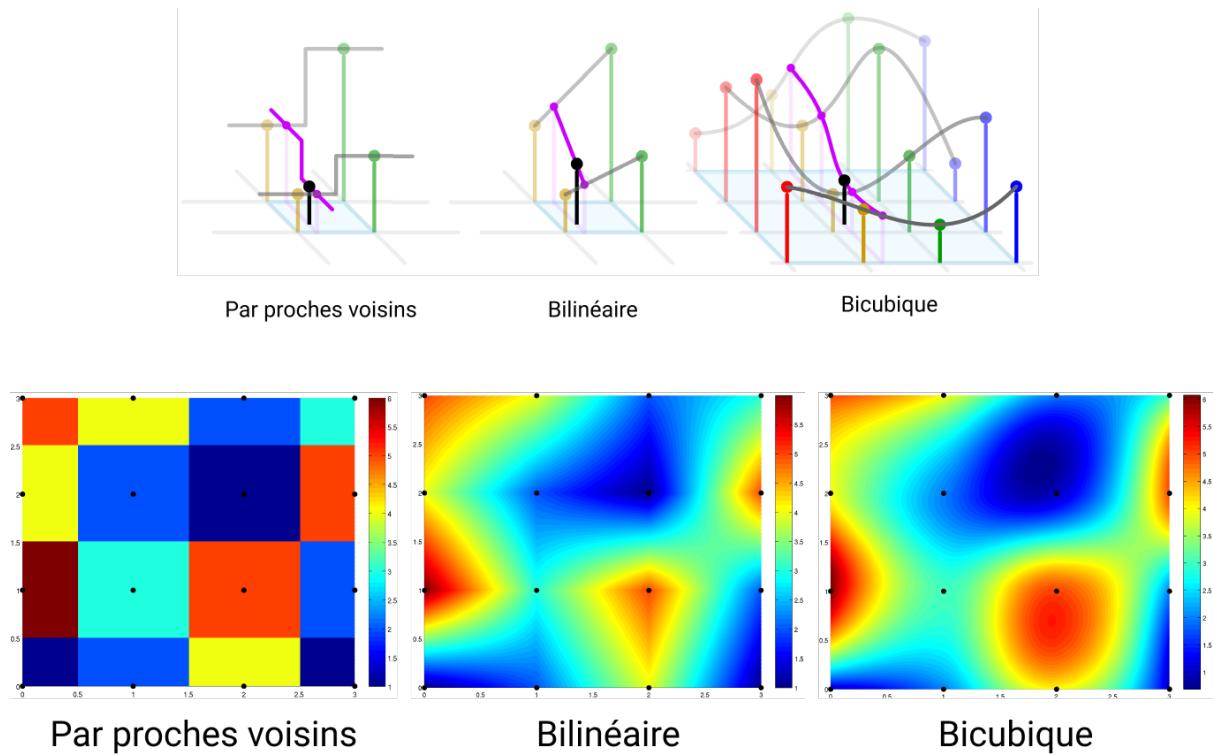


FIGURE 2 – Différentes méthodes d'interpolation [11]

La première méthode, l’interpolation par proches voisins, est la plus simple. Elle consiste à assigner au nouveau pixel la valeur du pixel original le plus proche. Cette méthode ne sera pas retenue de par sa maigre performance avec des données réelles et continues, telles que des données cartographiques.

La seconde méthode, l’interpolation bilinaire, consiste à prendre la valeur moyenne pondérée par la distance aux 4 (2x2) voisins les plus proches. Cette méthode donne des résultats bien plus satisfaisante que l’interpolation par proches voisins.

La troisième méthode, l’interpolation bic cubique, est très similaire à l’interpolation bilinaire. Elle permet cependant d’obtenir un gradient plus lisse, en prenant la valeur moyenne pondérée par la distance aux 16 (4x4) voisins les plus proches. Cette méthode est généralement préférée aux précédentes dans le traitement des images numériques.

La dernière méthode, "Conv2DTranspose", est basée sur le Machine Learning. Elle consiste en une opération convolutionnelle avec un kernel entraîné sur les données à redimensionner.

6.1.2 Meilleur choix des données

Malgré toutes ces méthodes d’interpolations très performantes, la solution finalement utilisée pour le projet est bien plus simple et efficace. Utiliser des données avec une résolution directement compatible avec la résolution des images de nuit, puisqu’aucune intervention n’est nécessaire. Ceci permet de ne pas prendre le risque de créer des données complètement erronées, en utilisant directement les données brutes.

Dans notre cas, passer des données Landsat-7 ayant une résolution de 30m par pixel à celles de Sentinel-2 ayant une résolution de 10m par pixel permet de faire correspondre parfaitement 1 pixel des données de nuit avec 50x50 pixels des images de jour. La décision ayant ammené ce changement n’avait pas été faite dans l’optique de se passer de l’étape d’interpolation, mais plutôt

pour augmenter la "densité d'information" disponible dans les images de jour, afin d'améliorer les résultats des réseaux de neurones convolutionnels.

6.2 Bruits numériques

Certains des jeux de données disponibles comportent du "bruit numérique", ou des imperfections ne représentant pas exactement les données souhaitées et optimales. Les données de jour ont assez peu de bruit numérique présent. Le seul qui peut poser problème pour nos réseaux de neurones sont les nuages qui peuvent complètement couvrir une zone habitée.

Le jeu de données des images de nuit "Black Marble" possède 3 sources de bruit notables.

6.2.1 Nuages

Premièrement, puisque de nouvelles images sont disponible chaque jour, aucun traitement de combinaison d'images n'est effectué pour retirer les zones recouvertes de nuages. Cependant, ces derniers sont détectés de manière très précise, et un "Cloud Mask" est disponible dans le jeu de données. En comparant le masque nuageux ainsi que les images de nuit, on remarque vite que les zones recouvertes de nuages sont plus "floues" que les alentours, alors que les zones sans nuages sont bien plus nettes, comme on peut le voir en figure 3. Ce flou est problématique et pourrait fausser les données si utilisé tel quel.

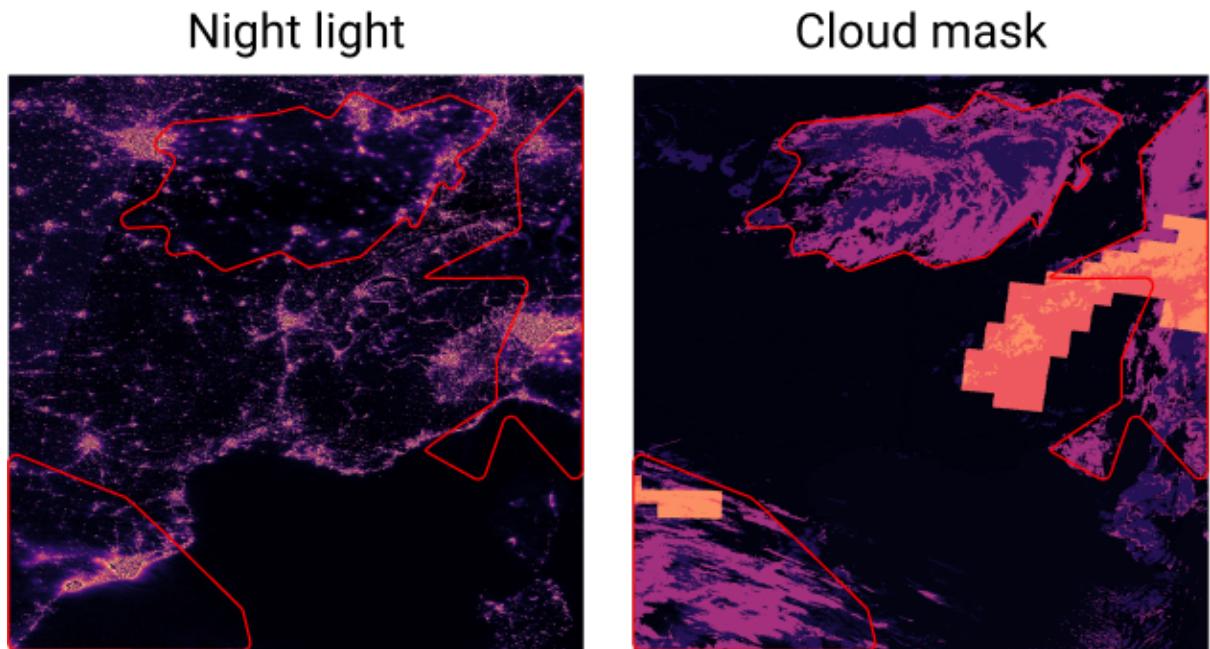


FIGURE 3 – Masque de nuages du satellite Black Marble

On remarque aussi que certaines parties du masque nuageux sont étrangement colorées, avec de nombreux angles droits qui semblent peu naturels. En se référant au manuel de la suite de produits "Black Marble" [6], on remarque que la valeur affichée est divisée en bits, et que chaque groupe de bits indique différentes informations sur les nuages, comme le montre la table 2 :

Bit	Flag description key	Results
0	Day / Night	0=Night 1=Day
1-3	Land / Water Background	000=Land & Desert 001=Land no Desert 010=Inland Water 011=Sea Water 101=Coastal
4-5	Cloud Mask Quality	00=Poor 01=Low 10=Medium 11=High
6-7	Cloud Detection Results & Confidence Indicator	00=Confident Clear 01=Probably Clear 10=Probably Cloudy 11=Confident Cloudy
8	Shadow Detected	1=Yes 0=No
9	Cirrus Detection (IR)(BTM15-BTM16)	1=Cloud 0=No Cloud
10	Snow/ice surface	1=snow/ice 0=no snow/ice

TABLE 2 – Table des bits du Cloud Mask du satellite Black Marble

Deux solutions sont alors envisageables pour régler le problème des nuages.

Premièrement, il serait possible d'effectuer un agrégat sur plusieurs dates des données de nuit, en ne sélectionnant que les zones sans nuages. Le problème de cette méthode est que l'on s'éloigne du concept des données que l'on souhaitait utiliser, "Near Real Time". Choisir une plage de date relativement courte permettrait de minimiser ce problème, mais n'est pas parfait.

La deuxième solution, plus simple à mettre en place et celle qui a été utilisée durant le projet, est de n'utiliser que des données qui n'ont naturellement pas de nuages. Cette solution a le même problème que la première, qui s'éloigne du "Near Real Time", mais possède un avantage distinct : Utiliser des données de jour provenant de la même date permet d'avoir des données de jour et de nuit sans nuages, ayant une correspondance temporelle exacte. Ceci n'est pas le cas avec la première solution d'aggrégation envisagée.

6.2.2 Reflets

Une autre source de bruit, moins visible au premier coup d'oeil, est le reflet de la lune et/ou du soleil dans le capteur du satellite. Encore une fois, le jeu de données "Black Marble" propose différents masques qui indiquent l'emplacement et intensité de ces reflets.

On peut cependant voir l'effet du reflet comme illustré en figure 4. En mettant en évidence la coupure nette visible dans le coin supérieur gauche de "Glint_Angle", on peut le comparer aux images de nuit "DNB_At_Sensor_Radiance". La luminosité des images de nuit a été fortement augmentée afin de rendre l'effet plus marqué.

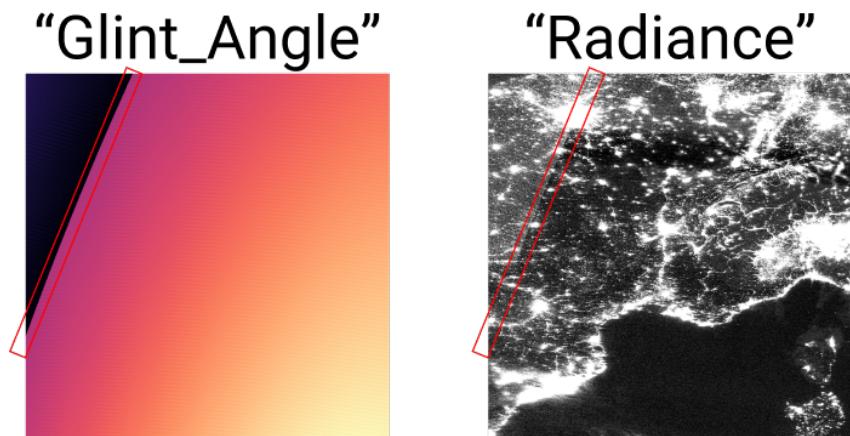


FIGURE 4 – Comparaison de l'effet du reflet

Bien que visible en augmentant fortement la luminosité des données de nuit, cet effet n'est pas suffisamment marqué pour justifier le temps qui serait nécessaire à le corriger. Les données seront donc utilisées telles quelles.

6.2.3 Senseur

La dernière source de bruit numérique, plus difficile à détecter que les précédentes, nécessite à nouveau d'augmenter la luminosité des images de nuit. De plus, elle requiert de zoomer suffisamment sur les données afin de pouvoir distinguer chaque pixel de manière individuelle.

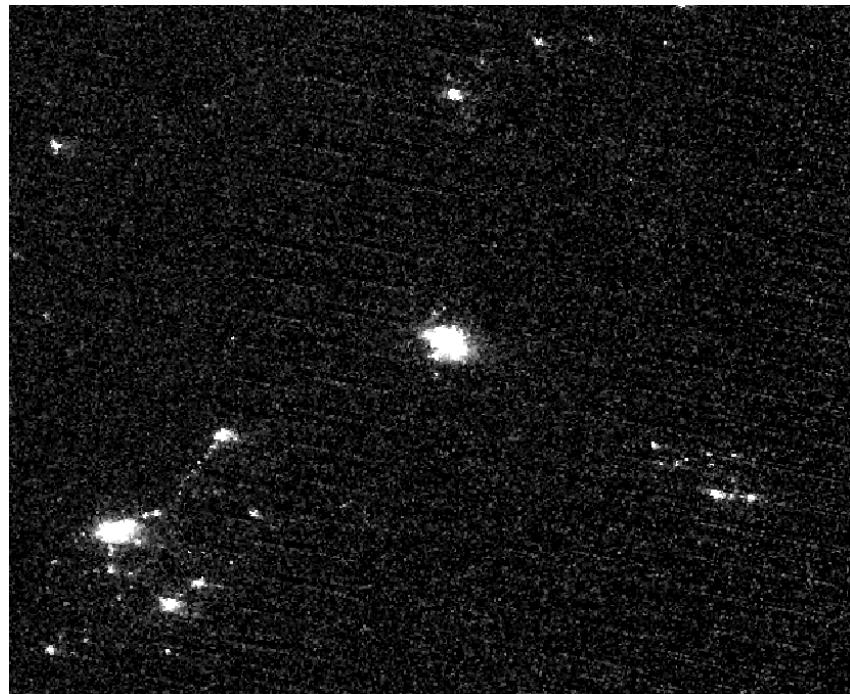


FIGURE 5 – Bruit numérique provenant directement du senseur de lumière, illustré sur les données du Sokoto

Comme le montre la figure 5, en augmentant suffisamment la luminosité des données, on peut observer un effet ressemblant beaucoup au grain présent sur les photos numériques prises en

basse luminosité.

La zone lumineuse au centre de l'image correspond à la ville du Sokoto, Nigeria. à part les autres taches lumineuses, les zones sombres correspondent à des zones complètement inhabitées et ne produisent donc absolument aucune lumière. Cependant, la lumière détectée dans ces zones semble aléatoirement assignée entre des valeurs de radiance allant de 0 à 3. Durant la suite des expériences, ces valeurs seront considérées comme une zone n'émettant aucune lumière.

6.3 Format des fichiers

Les sources de données utilisées lors du projet sont fournies sous différents formats de fichiers, fournissant plus ou moins de données concernant le contenu en plus des données brutes

6.3.1 Fichiers .jp2

Les images de jour, provenant du satellite Sentinel2-L2A et fournies par Sentinel-Hub, sont sauvegardées au format ".jp2". Ce format de fichier contient des metadata incluant les informations géographiques des données, permettant de les géo-localiser facilement [12].

Les données contenues correspondent à une image "True Color Image" contenant les bandes RGB du satellite. De cette manière, il n'est pas nécessaire de faire de traitement manuel afin de combiner les bandes de couleurs, ce qui aurait été nécessaire avec certains autres formats de fichiers tels que GeoTIFF.

De plus, un fichier annexe nommé "tileInfo.json", aussi fourni par Sentinel-Hub avec chaque Tile de données. Ce fichier contient d'autres metadata décrivant les données en question.

6.3.2 Fichiers .h5

Les images de nuit, provenant du satellite Black Marble et fournies par la NASA, sont quant à elles sauvegardées au format ".h5". Ce format de fichier peut contenir de nombreux jeux de donnée hiérarchiques [13]. Dans notre cas, les différents jeux de données proposés sont les suivants :

- BrightnessTemperature_M12
- BrightnessTemperature_M13
- BrightnessTemperature_M15
- BrightnessTemperature_M16
- DNB_At_Sensor_Radiance_500m
- Glint_Angle
- Granule
- Lunar_Azimuth
- Lunar_Zenith
- Moon_Illumination_Fraction
- Moon_Phase_Angle
- QF_Cloud_Mask
- QF_DNB
- QF_VIIRS_M10
- QF_VIIRS_M11
- QF_VIIRS_M12

- QF_VIIRS_M13
- QF_VIIRS_M15
- QF_VIIRS_M16
- Radiance_M10
- Radiance_M11
- Sensor_Azimuth
- Sensor_Zenith
- Solar_Azimuth
- Solar_Zenith
- UTC_Time
- StructMetadata.0

Comme expliqué en section 6.2, certains de ces jeux de données auraient pu nous être utiles afin de traiter ou visualiser les différentes sources de bruit numérique. Cependant, comme nous avons une solution pour les nuages et avons décidé de ne pas traiter le reflet du senseur, seul le jeu de données nommé "DNB_At_Sensor_Radiance_500m" nous intéresse.

Ce jeu de données contient les valeurs de radiance mesurées par le satellite. La radiance correspondant à l'intensité lumineuse mesurée. Les termes "luminosité", "illumination" et "radiance" seront donc utilisés de manière interchangeable par la suite. L'unité de la radiance est le $nW \cdot sr^{-1} \cdot cm^{-2}$ (nano-watts par stéradian par centimètre carré), et peut aller de 0 à 65534. Cependant, les valeurs mesurées vont très rarement au-delà des 1000. Par exemple, la piste d'atterrissement de l'aéroport de Genève a une radiance aux alentours des 1500, et est de loin la zone la plus lumineuse de Suisse Romande. La zone géographique principalement utilisée pour ce projet, le Sokoto (Nigeria), a une radiance maximale aux alentours des 350 et dépasse rarement les 150.

6.4 Système de coordonnées

Les données téléchargées sont représentées selon différents systèmes de coordonnées, et nécessitent certaines transformations pour passer d'un système à l'autre.

6.4.1 WGS4 - Données de nuit

Les données de nuit sont représentées selon le système WGS84 (World Geodetic System). C'est le système de coordonnées le plus communément utilisé de nos jours, notamment par le système GPS. De par sa compatibilité avec la majorité des services cartographiques tels que Google Maps, c'est le standard sur lequel nous baserons nos données.

Comme le montre la figure 6 extraite du manuel de documentation "NASA's Black Marble Nighttime Lights Product Suite Algorithm Theoretical Basis Document" [6], les images nocturnes sont découpées en carrés de 10° de longitude/latitude de côté, que l'on peut localiser dans l'espace au moyen du nom du fichier :

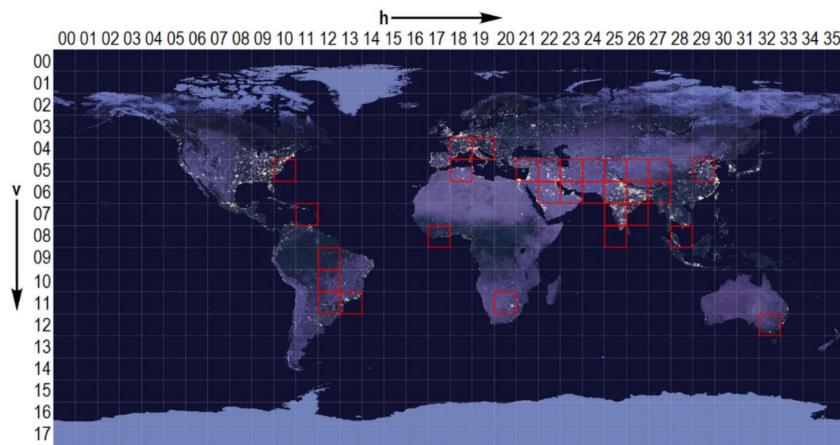


FIGURE 6 – Extrait du manuel des cartes "Black Marble" [6] représentant le découpage du jeu de données disponibles

Un fichier nommé "VNP46A1.A2015001.h08v05.001.2017012234657.h5" indique : [6]

- VNP46A1 - Nom court du produit
- .A2015001 - Date julienne d'acquisition (A-AAAAJJJ)
- .h08v05 - Identificateur de tuile (horizontalXXverticalYY)
- .001 - Version de la collection
- .2017012234657 - Date julienne de production (AAAAJJJHHMMSS)
- .h5 - Format de données HD5

On peut donc extraire la longitude et latitude de la 3ème partie du nom de fichier ("h08v05"), représentant l'emplacement du coin gauche supérieur de la sous-carte, représenté en WGS84.

6.4.2 MGRS - Données de jour

Les données de jour, quant à elles, sont représentées selon un système de coordonnées très différent. Les **Tiles** de données satellites sont découpées selon le standard MGRS ("Military Grid Reference System"), lui-même basé sur le standard UTM [14].

Ce standard représente les coordonnées d'une manière très différente que le standard WGS84 : Les coordonnées ne sont pas en degrés de longitudes et de latitudes, mais en mètres. De plus, l'inclinaison du système est propre à chaque case du système **MGRS**, mesurant environ 100km de côté. Cela implique que toute donnée étant séparée de 100km ou plus n'est pas "compatible" avant d'appliquer une transformation.

Afin de reprojeter les données de jour dans le bon système de coordonnées, on peut soit utiliser l'outil officiel du groupe EPSG [15] afin d'effectuer une transformation manuelle, ou utiliser la fonction `warp.reproject` de `rasterio` afin de projeter automatiquement les données depuis leur système d'origine (qui change tous les 100km dans une certaine direction) en WGS84.

La figure 7 illustre deux cases du système MGRS côte à côte se chevauchement partiellement. Quant inspectées individuellement avant reprojection, chaque case apparaît parfaitement carrée. Cependant, en les reprojetant en WGS84 (équivalent à EPSG4326), on s'aperçoit de l'inclinaison de chaque case par rapport à l'autre, ainsi que par rapport au système WGS84 qui suit parfaitement les bords verticaux et horizontaux de la figure.

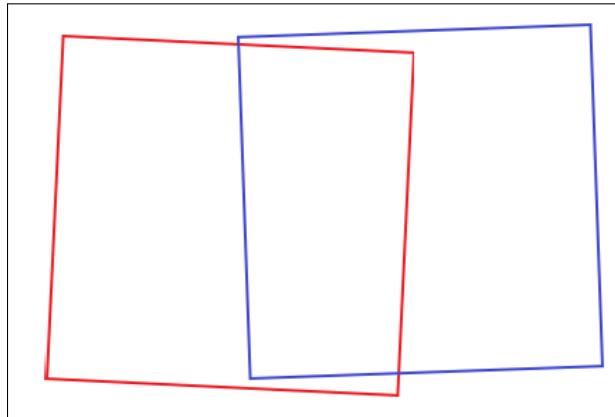


FIGURE 7 – Deux cases Sentinel-2 adjacentes avec différentes inclinaisons

Le notebook jupyter 2 - `reproject_sentinel2.ipynb` permet d'effectuer cette reprojection sur toutes les données de jour téléchargées depuis le satellite Sentinel-2.

6.5 Découpage des données

Reprojeter les données de jour et de nuit sur le même système de coordonnée nous permet de les aligner parfaitement, et de s'atteler à la tâche suivante qui consiste à découper les images de jour pour les faire correspondre aux images de nuit. Ceci correspond à un découpage de 50x50 pixels pour chaque image de jour et à 1x1 pixel pour les images de nuit.

Deux méthodes ont été utilisées au cours du projet.

6.5.1 Méthode rapide mais peu précise

La première méthode mise en pratique était l'approche la plus simple. Découper équitablement chaque case (mesurant 100x100km) des images de jour en sous-images de 50x50 pixels, et récupérer la valeur de radiance de l'image de nuit correspondant au pixel central de l'image de jour. Bien qu'extrêmement rapide, cette méthode a un gros défaut : Elle est extrêmement peu précise, de deux manières différentes :

Premièrement, afin de maintenir un alignement parfait, il faudrait avoir la certitude que chaque pixel des images de nuit mesure exactement 500m de côté. Or, ce chiffre n'est pas toujours décrit de manière aussi précise. Il en est de même pour les images de jour. Pour les images de nuit, certaines sources décrivent la résolution comme valant 15 arc secondes [16], ce qui correspond à entre 450m et 500m. Une erreur allant de 0 à 10% peut devenir gigantesque. Cela pourrait représenter jusqu'à 10km d'erreur pour chaque case mesurant 100km de côté.

Deuxièmement, il y a le problème de l'alignement initial. En démarrant le découpage des images de jour dans le coin supérieur gauche, il faut avoir la certitude que le coin supérieur gauche des images de nuit est placé exactement au même endroit, ce qui n'est généralement pas le cas. Bien que moins significatif que la première imprécision, ceci peut représenter jusqu'à 499m de décalage, et "décaler" l'entièreté du dataset de 1 pixel. La figure 8 illustre ce problème. Chaque carré noir représente un pixel des images de nuit, et chaque carré rouge représente les images de jour découpées en sous images, à une échelle très réduite.

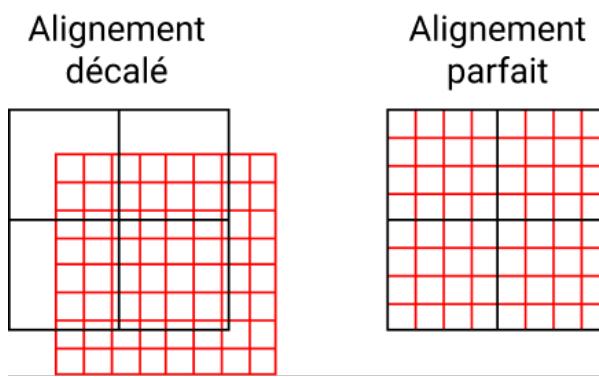


FIGURE 8 – Illustration simplifiée du problème de l'alignement initial des données

6.5.2 Méthode lente mais précise

La deuxième méthode, mise en place une fois les imprécisions de la première méthode détectées, permet d'avoir une très bonne précision, au prix de la vitesse de traitement.

Cette méthode utilise de nombreux outils de package python `rasterio`, l'outil principal étant `windows.Window`. De cette manière, pour chaque pixel des données de nuit, il est possible d'exporter une "fenêtre" des images de jour, centrée sur une coordonnée GPS et mesurant précisément 50x50 pixels autour de cette coordonnée centrale. Les fenêtres "voisines" sont complètement indépendantes, et peuvent se chevaucher ou être plus éloignées, en fonction des dimensions des pixels des images de nuit.

Cependant, cet export d'une "fenêtre" demande un nombre conséquent d'opérations mathématiques lourdes, telles que des changements de coordonnées, appartenance complète de polygones dans d'autres polygones, lecture répétée des metadata, etc... De ce fait, le temps de traitement du jeu de données utilisé de la région du Sokoto passe alors de quelques minutes à plusieurs heures. Cependant, ce traitement n'est à effectuer qu'une seule fois pour l'entièreté du projet, et le gain en précision est absolument essentiel au bon fonctionnement des réseaux de neurones.

Le découpage des données téléchargées par le notebook "1 - sentinel2 download.ipynb" se fait grâce au notebook "3 - split_tile.ipynb", et génère près de 450'000 images de 50x50 pixels au format png, occupant près de 2.5 GB sur le disque. Un fichier XML comportant des metadata est également généré pour chaque image, et ils peuvent être supprimés. Ne pas les supprimer n'a pas de conséquences, mais ces fichiers occupent de l'espace qui pourrait être libéré. Il est conseillé d'effectuer cette suppression en plusieurs fois, car la commande `rm *.xml`, supprimant tous les fichiers xml dans le dossier courant, peut effectuer un stack overflow et échouer lorsqu'appliquée sur les 450'000 fichiers d'un coup.

Chacune des images générées comporte quelques informations supplémentaires, placées dans le nom du fichier pour faciliter le traitement durant les prochaines étapes. Ces informations sont les suivantes (séparées par un "_" dans le nom du fichier) :

- La position GPS du centre de l'image
- Les coordonnées du pixel équivalent dans l'image de nuit originale
- La valeur de radiance correspondante

Finalement, un dernier traitement est nécessaire. Certaines des images exportées étaient situées en dehors de la zone de données capturées par le satellite, et sont complètement ou partiellement remplies par la couleur noire. Le notebook "4 - delete_black.ipynb" permet de supprimer les images comportant au moins 0.5% des pixels complètement noirs, ce qui ne semble jamais arriver

naturellement et permet donc la suppression de toutes les cases voulues et aucune des cases non-voulues.

6.6 Structure du dataset

Une fois les données exportées, on peut les charger dans un dataset NumPy afin de faciliter leur utilisation pour l'entraînement et la phase de test des réseaux de neurones convolutionnels.

Formellement, le type de donnée du dataset structuré NumPy est décrit par la ligne de code suivante :

```
dtype=[('image', 'float32', (50, 50, 3)),
      ('radiance', 'float32'),
      ('radiance_predicted', 'float32'),
      ('gps', 'float32', (2,)),
      ('pixels', 'int32', (2,))]
```

Cette ligne de code représente une array numpy dans laquelle chaque élément contient :

- Une image de 50x50 pixels avec 3 channels de couleur (R, G, et B), pour laquelle les valeurs des pixels sont des float32
- Une valeur de radiance correspondante en float32
- Une valeur de radiance prédictée (optionnelle, permet de ne pas dupliquer les datasets) en float32
- Un tableau contenant 2 éléments correspondant aux coordonnées GPS du centre de l'image, chaque coordonnée étant un float32
- Un tableau contenant 2 éléments correspondant aux pixels de l'image de nuit originale, chaque valeur en int32. Ces pixels permettent de facilement reconstruire l'image de nuit avec les prédictions des modèles.

Afin de pouvoir charger l'entièreté du dataset et effectuer les opérations qui suivent, il est conseillé d'avoir 32GB de RAM sur la machine exécutant le code. Les limitations du hardware seront discutées plus en détail en section 7.1

6.7 Normalisation des données

Une fois les données chargées en mémoire dans le dataset NumPy, plusieurs opérations de normalisation sont nécessaire avant de pouvoir utiliser les données pour l'entraînement des réseaux de neurones convolutionnels. De manière générale, les réseaux de neurones arrivent mieux à "apprendre" des données proches de la même valeur. Nous normaliserons donc l'ensemble de nos données à des valeurs entre 0 et 1.

6.7.1 Images

Afin de normaliser les données, on peut simplement diviser chaque bande de l'image par la valeur maximale pouvant y être trouvée, qui vaut 255 puisqu'une image PNG comporte 8 bits par bande de couleur ($2^8 - 1 = 255$) ainsi que 8 bits pour la bande alpha, que nous n'utilisons pas.

6.7.2 Radiances

Pour normaliser les radiances, on pourrait d'abord penser les diviser par la valeur maximale possible de 65534. Cependant, les valeurs de radiance ne sont jamais aussi grandes. Dans le dataset du Sokoto, la valeur maximale est proche de 350, et la deuxième plus grande valeur est de moins de 200. Comme l'objectif est d'obtenir des valeurs distribuées entre 0 et 1, on peut donc diviser les radiances par la valeur maximale apparaissant dans le dataset.

La figure 9 illustre la distribution des radiances avant et après normalisation.

Un point qui mérite d'être mentionné est qu'en utilisant simplement la valeur maximale comme facteur de normalisation, la capacité de généralisation à d'autres emplacements géographiques diminue. Cependant, les modèles développés n'ont pas vraiment l'objectif d'être applicables à d'autres zones, cette normalisation n'est donc pas un problème.

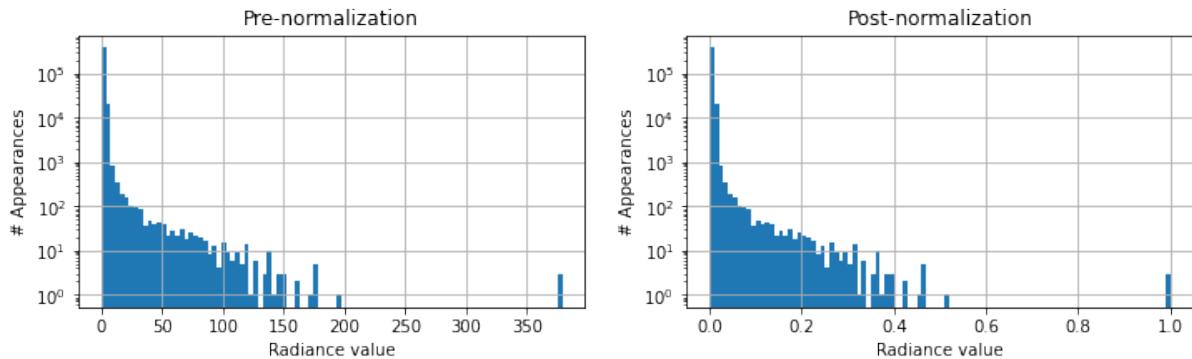


FIGURE 9 – Distribution des valeurs de radiance avant et après normalisation

6.7.3 n% cut

En observant la figure 9, on observe que à part une valeur aberrante, la quasi-totalité des données normalisées se trouvent en dessous de 0.5, ce qui "compresse" la majorité des valeurs sur la moitié inférieure, plutôt que de les distribuer de manière aussi équitable que possible entre 0 et 1. Afin de pallier ce problème, on peut simplement faire diminuer ces valeurs aberrantes à une nouvelle valeur de radiance maximale. Ceci peut par exemple être le n-ième pourcentile. Afin de ne modifier aussi que de valeurs de radiances que possible, le 99.999 ième pourcentile sera choisi comme nouvelle valeur maximale, correspondant l'ancienne valeur normalisée de 0.49.

La nouvelle distribution des radiances est illustrée par la figure 10

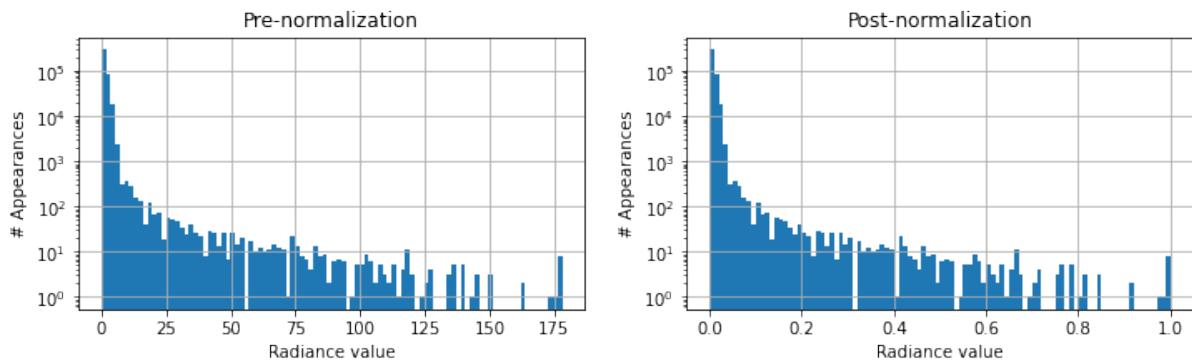


FIGURE 10 – Distribution des valeurs de radiance avant et après normalisation après rectification de la valeur maximale

7 Réseaux de Neurones Convolutionnels

Comme mentionné dans le cahier des charges, nous utiliseront les réseaux de neurones convolutionnels afin de générer les résultats souhaités. Sans rentrer dans les détails, les réseaux de neurones convolutionnels sont une technique de deep learning permettant entre autres d'extraire diverses features depuis des images et effectuer des prédictions dessus. Ces réseaux peuvent être relativement complexes à configurer et mettre en place pour une personne non-initiée, particulièrement l'approche "Data-Driven" que l'on ne trouve pas vraiment hors du monde du Machine Learning. Cependant, les possibilités offertes par ces réseaux justifient largement la courbe d'apprentissage difficile. De plus, de nombreuses ressources d'apprentissage excellentes sont disponibles gratuitement sur internet.

7.1 Limitations

Malgré les clairs avantages provenant de l'utilisation des réseaux de neurones convolutionnels, nombreuses de leur caractéristiques imposent des limitations sur leur utilisation.

7.1.1 Limitations en temps

Entraîner un réseau de neurones est quelque chose qui prend passablement de temps. De plus, plus la complexité du réseau augmente, plus ce temps devient long. Ceci rend la recherche d'une topologie ainsi que des hyper-paramètres les plus adaptés un processus très lent, et demande de faire un choix : tester peu de réseaux avec une topologie complexe, ou en tester beaucoup plus avec une topologie relativement simple.

Dans le cadre de ce projet, la seconde approche testant des topologies relativement simples sera utilisée. La recherche de topologies complexes, comme celle d'un réseau tel que VGG16 [17], pourrait être explorée dans un futur projet.

7.1.1.1 Arrêt de l'entraînement

Une deuxième limitation des réseaux de neurones provient de la nature même du problème à résoudre, trouver l'ensemble des paramètres permettant de minimiser une fonction de coût. Même si l'algorithme d'optimisation utilisé peut facilement trouver un minimum local, il n'existe aucun moyen permettant de garantir de trouver le minimum global en un temps polynomial. Evidemment, brûler de l'ensemble des combinaisons des paramètres entraînables n'est pas une option au delà de quelques paramètres.

De ce fait, nous devons prendre une décision de stopper l'entraînement du réseau à un certain moment, sans avoir la certitude d'avoir attendu assez longtemps. En effet, il est possible qu'un réseau qui "stagne" se "déboule" et trouve un nouveau, meilleur minimum local. Ce problème implique soit de devoir perdre plus de temps pour augmenter les chances d'avoir attendu suffisamment pour avoir la meilleure solution trouvable pour la configuration actuelle, ou d'accepter que le réseau aurait pu produire de meilleurs résultats.

7.1.2 Limitations physiques

En plus des limitations temporelles qui s'appliquent à tous les problèmes de réseaux de neurones, des limitations physiques dépendantes du matériel à disposition s'ajoutent au problème.

7.1.2.1 RAM

Pour pouvoir effectuer l'entraînement du réseau de neurone, il est nécessaire de pouvoir charger l'entièreté du jeu de données en RAM. Il serait probablement possible de charger les données une à une depuis le disque, mais ceci serait 10 à 100x plus lent et n'est donc pas envisageable.

De plus, une attention particulière doit être portée afin de ne pas dupliquer le jeu de données lorsque ce n'est pas nécessaire. Pour cette raison, comme expliqué en section 6.6, la valeur optionnelle de "radiance prédictive" est stockée dans le jeu de données initial, afin de ne pas avoir à le dupliquer entièrement pour simplement y stocker les valeurs de radiance prédictive.

7.1.2.2 GPU & VRAM

Un autre facteur physique limitant l'entraînement des réseaux de neurones est l'utilisation ou non d'un GPU, qui peut réduire le temps d'entraînement drastiquement (par un facteur 4, avec le matériel utilisé lors de ce projet). Cependant, un nouveau problème similaire à celui de la RAM arrive avec l'utilisation d'un GPU. La quantité de VRAM, ou Video RAM. A nouveau, plus cette quantité est grande, plus on accélère l'apprentissage en permettant de charger de jeu de données dans cette mémoire très rapide.

7.1.2.3 Machine Learning dans le Cloud

Une solution potentielle à ces limitations physiques est d'effectuer l'entraînement des réseaux de neurones dans le cloud. L'avantage principal du cloud est qu'il permet de paralléliser l'entraînement de nombreuses configurations, plutôt que de devoir le faire de manière séquentielle. Cette approche avait été envisagée dès le début du projet, mais après inspection, les coûts engendrés peuvent rapidement augmenter. De plus, les plateformes telles que Amazon Sagemaker⁶ offrent des outils extrêmement avancés, ce qui réduirait beaucoup l'intérêt d'un travail de Bachelor.

Pour ces raisons, le machine learning dans le cloud ne sera pas utilisé pour ce projet.

8 Keras

Comme mentionné en section 4.2, le framework utilisé pour le machine learning durant ce projet sera Keras [18]. Cette section a pour but de brièvement décrire les divers outils proposés par keras qui seront utilisés pour construire les réseaux de neurones convolutionnels (ci-après "CNN"), ainsi que les choix qui ont été faits par rapport à ces outils. Cependant, les résultats obtenus par ces choix ne seront pas illustré dans cette section, mais le seront en section 9.

8.1 Hypersensibilité aux paramètres

Avant de discuter de la topologie des CNNs, il est nécessaire de mentionner un problème récurrent durant ce projet afin de pouvoir y faire référence par la suite.

Assez rapidement durant la phase initiale de développement des CNNs, il est apparu que les réseaux étaient extrêmement sensibles à certains hyperparamètres mais pas à d'autres. On pourrait s'attendre à ce que modifier ces hyperparamètres au delà du seuil optimal fasse diminuer la performance du modèle. Cependant, au lieu de légèrement modifier les prédictions comme on aurait pu s'y attendre, le changement se fait de manière abrupte et le modèle ne prédit plus que des radiance à 0.

Ces hyperparamètres problématiques seront détaillés dans les sections suivantes, mais pour rapidement les résumer, les paramètres en question sont les suivants :

- "Dropout"
- "Learning Rate"
- "Filter Depth"
- Nombre de neurones dans une couche "Dense"

6. <https://aws.amazon.com/sagemaker/>

8.2 Topologies des CNNs

Les CNNs sont composés de plusieurs "blocs" effectuant diverses opérations sur les données, afin de les raffiner en un résultat, sous forme de regression dans notre cas. Les "blocs" principaux envisagés pour ce projet sont les suivants.

8.2.1 Filtres Conv2D

Les filtres Conv2D permettent d'extraire des "features" des données passées en entrée, grâce à des noyeaux de convolution dont les valeurs correspondent aux paramètres entraînés [19].

Ces filtres font partie des éléments essentiels aux CNNs. Deux paramètres principaux sont à déterminer afin d'extraire des features des données : La taille du noyeau, ainsi que la profondeur des filtres.

Pour la taille du noyeau, les valeurs 3x3, 5x5 et 7x7 sont couramment choisies. Pour la suite des expériences du projet, nous utiliserons un noyeau de 3x3, puisqu'après plusieurs tests, il n'y avait pas de différence significative de résultats avec une taille de noyeau plus grande.

Pour la profondeur des filtres, diverses valeurs ont été essayées. Un problème assez récurrent s'est assez vite présenté : Utiliser plus de filtres que la largeur de l'image (50 pixels) résultait systématiquement en le problème décrit en section 8.1, et produisait uniquement des prédictions de 0 radian. La profondeur maximale de filtres utilisée durant la suite du projet sera donc 48.

8.2.2 Pooling

Toujours utilisés après les filtres Conv2D, les opérations de Pooling permettent de réduire la taille des données en extrayant les features détectées par les filtres Conv2D. Ces opérations de pooling sont effectuées sur un "pool size", correspondant à la dimension de la zone qui sera agrégée en une seule valeur. Durant la suite du projet, nous utiliserons la valeur la plus courante, 2x2, qui semble donner les meilleurs résultats.

Différents types de Pooling existent, et les deux auxquels nous nous intéresserons sont "MaxPooling" et "AveragePooling". MaxPooling, comme son nom l'indique, produit comme valeur de sortie la valeur maximale parmi les 2x2 valeurs en entrées. AveragePooling produit comme valeur la moyenne de la fenêtre de 2x2 passée en entrée. Après plusieurs tests, il est ressorti que MaxPooling est l'opération de pooling générant les meilleurs résultats.

Utiliser une dimension de 2x2 divise la taille des images passées en entrée à l'opération de Pooling par 2 en ne gardant que la valeur produite par l'opération de pooling sur les 2x2 pixels. Ceci signifie qu'il n'est pas possible d'effectuer une quantité illimitée de ces opérations les unes à la suite des autres, puisque l'image d'entrée sera rapidement réduite à 1x1 pixel de large. Pour une image de 50x50 pixels, cela correspondrait à $\log_2(50) = 5.64$ opérations de pooling. Cependant, puisque chacune de ces opération est précédée d'une opération de Conv2D qui réduit la largeur des images par 2 (quand aucun padding n'est utilisé, ce qui est notre cas), le nombre maximal d'opérations Conv2D + MaxPooling combinées s'élève à 4.

8.2.3 Dense layers

Une fois les features extraites par les 4 couches Conv2D + MaxPooling, on souhaite pouvoir extraire un résultat sous forme de régression. Pour ce faire, on doit transformer les données produites par les diverses couches de Conv2D + MaxPooling, qui sont encore multi-dimensionnelles à ce stade, à une liste uni-dimensionnelle. Ceci est fait grâce à la fonction `flatten` de keras.

Une fois les données "applatis", on peut les utiliser comme valeurs d'entrées dans un réseau de neurones "Fully-Connected", indiquant que chaque neurone est connecté à chacun des neurones

de la couche suivante.

Finalement, un dernier neurone est nécessaire pour produire une valeur de sortie unique.

Les paramètres devant être choisis pour cette étape sont :

- le nombre de couche de neurones
- le nombre de neurones dans chaque couche cachée

Par expérience, les valeurs qui permettaient d'obtenir les meilleurs résultats étaient une à deux couches cachées de neurones (en plus du neurone de sortie), ayant chacune entre 8 et 32 neurones.

8.2.4 Dropout

Un des problème qui arrive souvent dans le domaine des réseaux de neurones est "l'overfitting", ou "surapprentissage". C'est un phénomène qui apparaît lorsque le réseau "mémorise" le jeu de données d'entraînement, sans extraire de features. Ceci mène généralement à de mauvaises performances de généralisation sur de nouvelles données.

Une manière commune d'éviter le surapprentissage est d'appliquer du "Dropout". Le Dropout est une opération qui s'effectue entre deux couches cachées de neurones, et permet de simuler un entraînement avec de nombreuses topologies différentes en parallèle, en "supprimant" certaines des connexions entre deux couches de neurones. Dans le cas des CNNs, cette opération est généralement appliquée au sein de la couche dense du réseau plutôt que dans les phases Conv2D + MaxPooling, puisque la couche dense, ou "pleinement connectée" est celle qui comporte souvent le plus grand nombre de connexions qui mène au surapprentissage.

Le seul paramètre à déterminer pour le dropout est le pourcentage des connexions entre les deux couches cachée qui devraient être supprimées. Des valeurs entre 0.1 et 0.5 sont communément utilisées.

Malheureusement, les configurations de CNNs utilisés lors de ce projet étaient telles qu'utiliser plus que quelques pourcents de dropout mènerait au problème décrit en section 8.1 et prédirait toujours une radiance de 0. Pour cette raison, aucun dropout n'a été utilisé lors de ce projet.

8.2.5 Fonctions d'activation

Concernant la topologie des CNNs, un dernier élément reste à envisager. La fonction d'activation, qui transforme les résultats de chaque opération **Conv2D** ou **Dense** avant de le passer à la couche suivante.

Ici, assez peu de choix ont besoin d'être faits. La fonction d'activation "relu" est presque toujours le bon choix dans les couches cachées du réseau. Pour la couche de sortie, un peu plus de choix s'offre à nous. Puisque nous souhaitons obtenir un résultat de régression entre 0 et 1, la fonction "sigmoid" semble très adaptée.

La figure 11 illustre 4 des fonctions d'activation les plus utilisées.

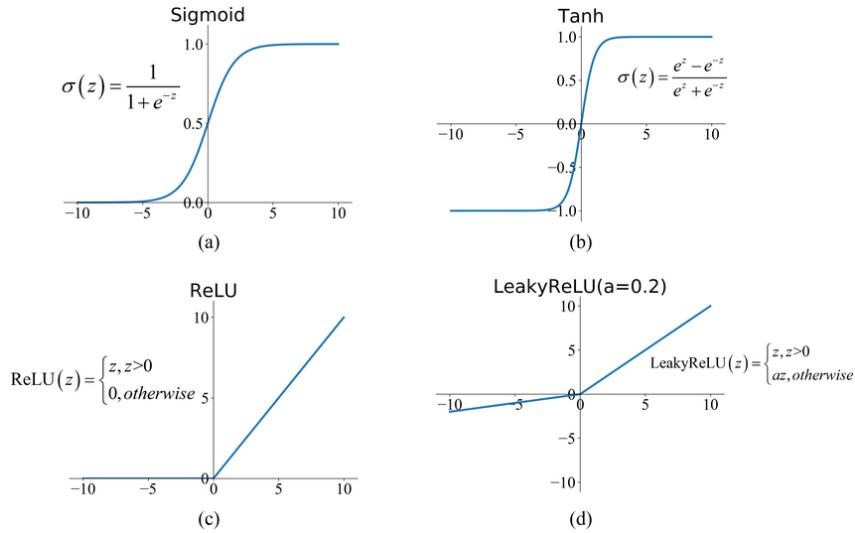


FIGURE 11 – Comparaison de 4 fonctions d’activations les plus utilisées [20]

8.3 Optimizers

Un autre choix qu’il faut faire lors de la construction d’un réseau de neurones convolutionnel est l’optimizer qui sera utilisé afin de minimiser une fonction de coût représentant à quel point les prédictions du réseau sont bonnes.

8.4 Fonctions de coûts

Différentes fonctions de coûts sont proposées par Keras. Les deux qui nous intéressent le plus sont "MSE", ou "Mean Square Error" ainsi que "MAE", ou "Mean Absolute Error".

Comme indiqué par leur nom, ces fonctions calculent l’erreur moyenne entre les prédictions et les vraies valeurs de radiance. Afin de toujours avoir un résultat positif, MSE met cette valeur au carré et MAE prend sa valeur absolue.

Durant le reste du projet, nous utiliserons la fonction MSE pour la raison suivante :

Nos données sont composées d’une majorité de zones n’émettant pas de lumière (illustré par la figure 10 avec son échelle logarithmique). Ceci implique qu’effectuer des prédictions proche de zéro permettra à notre réseau d’obtenir de bons scores.

De plus, notre objectif est de reconnaître les features des zones qui émettent de la lumière, mais ne pas "mémoriser" les zones habitées qui n’en émettent pas (afin de prédire qu’elles émettent de la lumière par la suite sur les données de test). De ce fait, utiliser MSE qui décuple l’erreur au carré permet de mettre plus d’importance sur les zones habitées qui émettent de la lumière, et "d’oublier" les zones habitées qui n’en émettent pas.

Cette justification peut sembler peu conventionnelle, mais de manière générale, utiliser MSE semblait produire de meilleurs résultats que MAE.

8.4.1 Adam

Pour minimiser cette fonction de coût, choisir un "optimizer" est nécessaire. Les deux plus populaires sont "Adam" ainsi que "SGD", ou "Stochastic Gradient Descend". Après différents tests, il est apparu que les deux offraient des résultats similaires. De ce fait, le choix se porta assez facilement sur "Adam" de par sa popularité et la quantité de ressources disponibles en ligne illustrant son utilisation.

8.4.1.1 Adaptive Learning rate

Une des options offertes par Keras est de pouvoir changer la learning rate de manière dynamique. Ceci permet généralement d'affiner les résultats en diminuant la learning rate au cours de l'entraînement. Différentes méthodes pour déterminer le changement que subira la learning rate. Dans notre cas, nous utiliserons "Exponential Decay", nous permettant de réduire la learning rate selon la fonction suivante : $initial_learning_rate * decay_rate^{(step/decay_steps)}$.

8.5 Cross-validation

Afin d'avoir une meilleure idée de la qualité des prédictions effectuées par les modèles générés, une méthode communément utilisée est la "Cross-validation". Elle consiste à diviser son jeu de données en deux : Un "training set" qui sera utilisé pour entraîner le réseau, et un "testing set" qui sera utilisé une fois le modèle terminé pour tester les capacités de généralisation du modèle.

8.5.1 Séparation Uniforme vs Aléatoire

Pour le cas d'utilisation de notre projet, deux méthodes de séparation des données peuvent être utilisées. Une séparation aléatoire ou uniforme.

La séparation uniforme consiste à choisir une ligne séparant les données, et assigner les données de part et d'autre de cette ligne au training set ainsi qu'au testing set, respectivement. Bien sûr, cette séparation pourrait prendre d'autres formes qu'une ligne, mais c'est la manière la plus simple de faire. Cette méthode peut cependant causer une distribution non équitable des données entre le training set et le testing set, par exemple en ayant toutes les grandes villes dans le testing set mais aucune dans le training set. Ceci peut être considéré comme un avantage, simulant l'entraînement des modèles sur une région et les appliquant sur une région complètement différente (différents pays ayant le même climat). Ce type de séparation sera nommée "Split" durant la suite du projet

La séparation aléatoire, comme son nom l'indique, consiste à placer chaque élément du jeu de données aléatoirement entre le training set et le testing set. Cette méthode permet d'augmenter les chances d'avoir une distribution équitable des features entre les deux datasets, mais diminue légèrement la clarté des données lorsque visualisées, puisque une partie non négligeable des données est manquante (dans notre cas, nous utiliserons 50% pour le training set et 50% pour le testing set). Ce type de séparation sera nommée "Sparse" durant la suite du projet.

La figure 12 illustre la visualisation des deux types de séparations sur le jeu de données du Sokoto. Dans le cas de la séparation "Sparse", les données du training set sont affichées. Les données manquantes (appartenant au testing set) sont affichées en noir. Dans le cas de la séparation "Split", la ligne blanche sépare les deux jeux de données (dont la taille n'est pas égale).

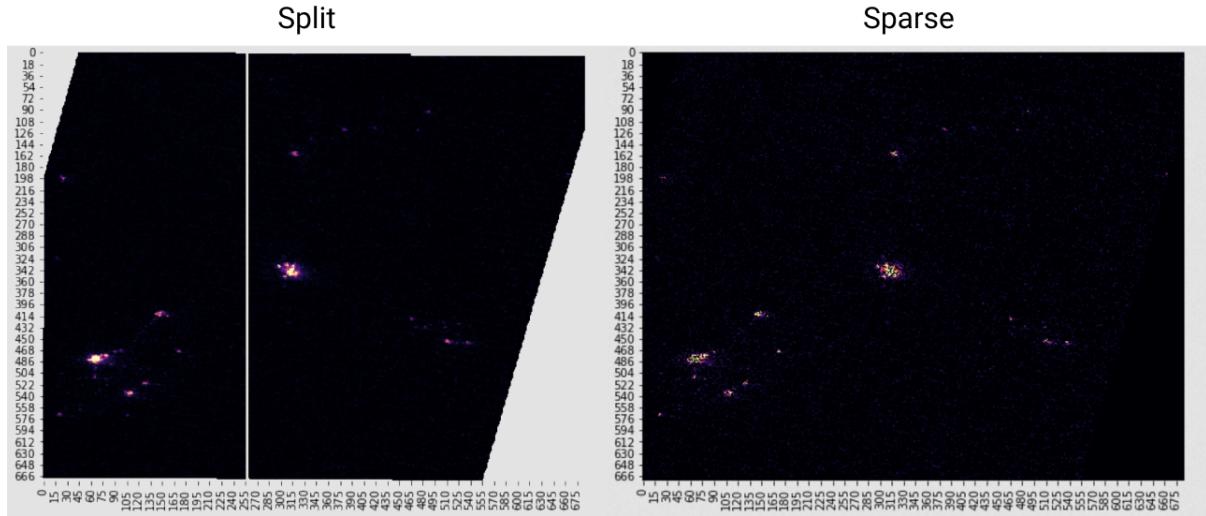


FIGURE 12 – Comparaison des séparations de cross-validation "Split" vs "Sparse"

8.5.2 K-Fold validation

Une sous catégorie de Cross-Validation que l'on peut appliquer est le "K-Fold Validation". Cela consiste à tester plusieurs combinaisons de séparation des jeux de données train/test différents, afin de s'assurer que l'on a pas eu de la "chance" lors de la séparation initiale. K-Fold validation a l'avantage de donner une meilleure certitude sur les capacités de génération du modèle produit, mais rallonge le temps d'entraînement par un facteur K. La figure 13 illustre une validation "5-Fold".



FIGURE 13 – 5-Fold Validation [21]

Dans le cadre de ce projet et à cause des contraintes en temps, nous n'effectuerons qu'un 2-Fold validation sur les modèles les plus prometteurs.

9 Application des CNNs

Une fois les divers outils et blocs de construction utilisables pour définir un CNN bien définis, ainsi que les différentes raisons justifiant le choix de leurs paramètres bien expliquées, on peut se pencher sur des exemples concrets de réseaux de neurones convolutionnels, leur topologie, le choix de leurs paramètres, la séparation des données, et les résultats bruts produits.

Une discussion plus fine de ces résultats sera faite en section 10.

Le notebook jupyter "5 - cnn.ipynb" permet de faire des expériences en modifiant la topologie et les hyper-paramètres, d'entraîner les réseaux décrits dans cette section, ainsi que d'en visualiser les résultats. Le notebook "6 - model testing.ipynb" permet de directement charger un modèle pré-entraîné et en évaluer les performances.

9.1 Visualisation des performances & prédictions

Avant de pouvoir comparer les modèles générés, il est important de pouvoir mesurer et visualiser leur performances. Les prédictions effectuées par les modèles peuvent être visualisées sous différentes formes. Les principales sont la distribution des radiances (sur une dimension), la distribution des radiances (sur deux dimensions), la différence entre les ground truths et les prédictions (sur deux dimensions), ainsi que la corrélation de Pearson entre les prédictions et les ground truths. Un dernier outil de visualisation utile mais pas en lien direct avec les prédictions est l'historique d'évolution de la fonction de coût à travers les epochs.

Ces outils seront présentés plus en détails en temps voulu.

9.2 Optimisation des topologies et hyper-paramètres

Lors de la phase de "développement" de ce projet, de nombreuses topologies et combinaisons d'hyper-paramètres différents ont été testés afin de trouver celles qui fonctionnaient le mieux. En plus des simples tests manuels, une méthode efficace consistait à définir une liste des paramètres à tester, et effectuer l'entraînement séquentiel de chacun de ces paramètres.

Cependant, cette méthode test chaque paramètre de manière indépendante. La méthode optimale, qui prendrait malheureusement trop de temps à appliquer, consisterait à entraîner un modèle pour chaque combinaison possible de ces paramètres. Pour faire un compromis entre ces deux méthodes, diverses combinaisons des paramètres ont été testées manuellement afin de voir l'effet combiné de certains paramètres.

Les caractéristiques principalement comparées entre les différents modèles générés sont les suivantes :

- Profondeur des filtres Conv2D
- Taille des noyeaux de convolution
- Type de pooling
- Taille de la fenêtre de pooling
- Nombre de couches Conv2D + Pooling
- Nombre de couches denses
- Nombre de neurones dans une couche dense
- Fonctions d'activation utilisées
- Utilisation ou non de dropout
- Type d'optimizer
- Learning rate constante vs adaptive
- Séparation des données pour Cross-validation
- Normalisation des données

9.3 Modèle final

Après de nombreux tests en tous genres (certains illustrés en section 9.2), le modèle final généré pour ce projet peut être reconstruit selon la topologie (Table 3) et les hyper-paramètres (Table 4) suivants :

9.3.1 Topologie & Hyper-paramètres

Type de couche	Fonc. Activ.	Dimension	# paramètres
Input		$50 \times 50 \times 3$	0
Conv2D (3x3, 48)	ReLU	$48 \times 48 \times 48$	1344
MaxPooling (2x2)		$24 \times 24 \times 48$	0
Conv2D (3x3, 32)	ReLU	$22 \times 22 \times 32$	13856
MaxPooling (2x2)		$11 \times 11 \times 32$	0
Conv2D (3x3, 16)	ReLU	$9 \times 9 \times 16$	4624
MaxPooling (2x2)		$4 \times 4 \times 16$	0
Flatten		256	0
Dense (32)	ReLU	32	8224
Dense (32)	ReLU	32	1056
Dense (1)	Sigmoid	1	33
Paramètres totaux			29137

TABLE 3 – Topologie du modèle final

Paramètre	Valeur
Optimizer	Adam
Adam Momentum parameters	Default
Loss function	MSE
Learning rate decay function	ExponentialDecay
Initial Learning rate	$5 * 10^{-5}$
Decay steps	2000
Decay rate	0.99
Batch size	64
Epochs	250

TABLE 4 – Hyper-paramètres du modèle final

En appliquant cette topologie ainsi que ces paramètres, on génère un modèle relativement "simple" (moins de 30k paramètres entraînables). Cependant, ce modèle apprend et généralise aussi bien, voire mieux que des modèles plus complexes testés allant jusqu'à plus de 1 million de paramètres. La figure 14 illustre l'évolution de la fonction de coût au cours des epochs. La version de droite zoom sur les valeurs autres que celles de la première epoch, puisqu'une amélioration d'un facteur 20 entre la première et deuxième epochs "écrase" le graphique.

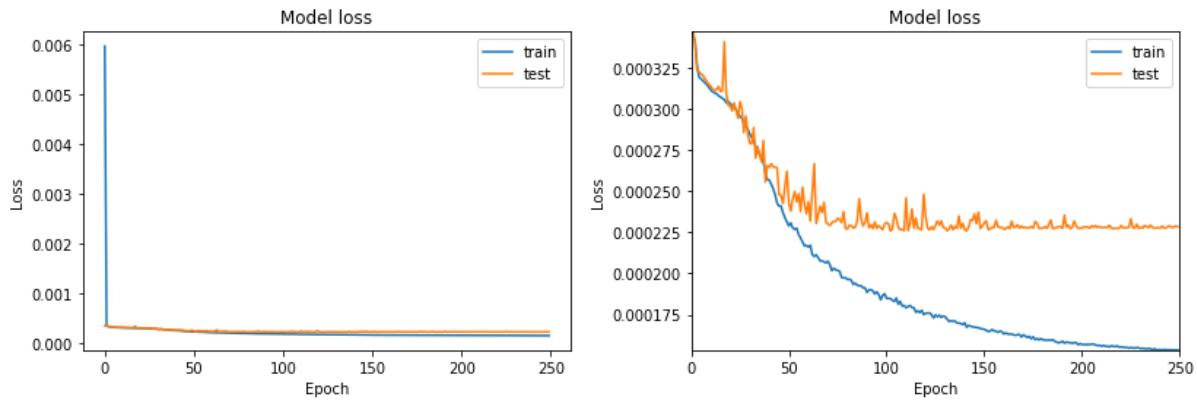


FIGURE 14 – Evolution de la fonction de coût selon les epochs. Training set en bleu, Testing set en orange. Le graphe de droite est une version "zoomée" de celui de gauche

On voit que après une centaine d'epochs, la fonction de coût pour le testing set stagne, alors que celle du training set continue à s'améliorer. Malgré cela, on ne voit pas les signes typiques de surapprentissage qui feraient "remonter" la fonction de coût au delà de son minimum atteint.

9.3.2 Résultats

9.3.2.1 Distribution des radiances 1D

Une première manière de visualiser les prédictions effectuées consiste à faire un histogramme des valeurs prédites. Bien que très simple, cette méthode permet d'avoir une première idée de leur distribution ainsi que de les comparer aux "ground truths". La figure 15 illustre cette visualisation.

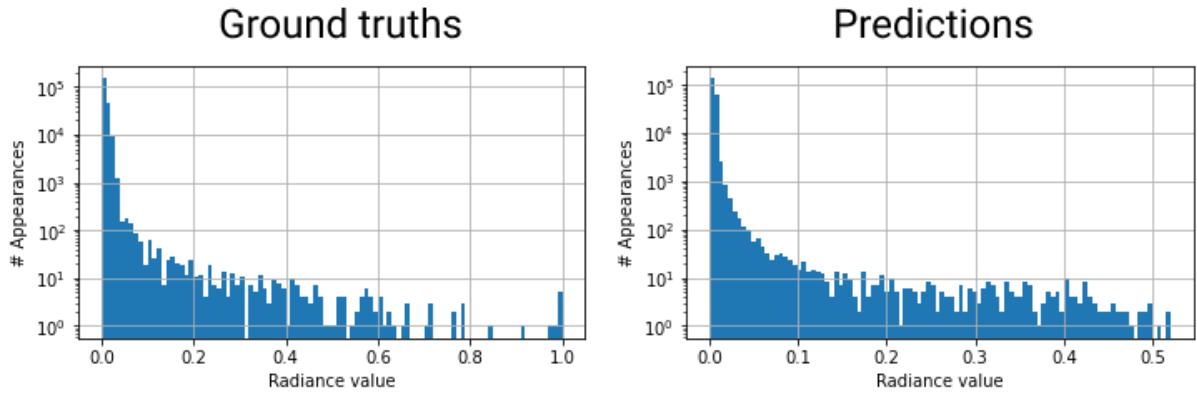


FIGURE 15 – Comparaison de la distribution uni-dimensionnelle des radiances du training set, ground truths vs predictions

A première vue, la "forme" de la distribution semble très similaire. Cependant, on se rend compte que la valeur maximale des prédictions ne dépasse pas 0.6. Il faut cependant bien remarquer que l'échelle de l'axe des # d'apparitions est logarithmique, et que les radiances supérieures à 0.6 dans les ground truths se comptent sur les doigts de la main et représentent une infime portion du jeu de données complet.

9.3.2.2 Correlations avec les *Ground Truths*

Une autre méthode similaire permettant de visualiser les résultats, mais qui offre bien plus d'informations consiste à calculer la correlation de Pearson entre l'ensemble des ground truths et les prédictions. Pour accompagner cette correlation, un graphique ayant sur l'axe x la valeur ground truth et sur l'axe y la valeur prédictive permet de bien visualiser les prédictions de manière concise. La figure 16 illustre cette méthode de visualisation.

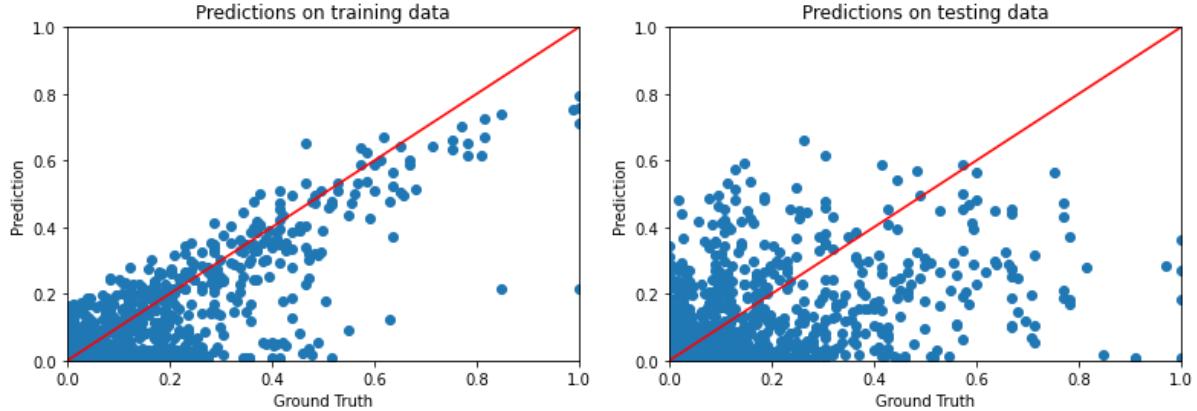


FIGURE 16 – Graphe comparant les valeurs prédites avec les valeur réelles. La ligne rouge indique les prédictions parfaites.

On voit très rapidement que les prédictions effectuées sur le set de training sont bien plus proches de la ligne rouge représentant les prédictions parfaites que celles du set de testing. Cependant, cela n'est pas un problème. En effet, comme nous recherchons des villages qui semblent émettre de la lumière mais n'en émettent pourtant pas, avoir des prédictions parfaites serait indésirable. Les données qui nous intéressent particulièrement sont donc celles situées sur la partie tout à gauche du graphe, qui sont prédites comme lumineuses mais ne le sont en vérité pas. Les données sous la barre rouge des prédictions parfaites correspondent simplement aux emplacements lumineux prédits comme peu lumineux. Même si l'on espère minimiser ces prédictions, elles ne posent pas de vrai problème pour les résultats espérés.

Il est donc possible de calculer la corrélation des prédictions avec chacun des jeux de données, et de mesurer à quel point les mesures sont liées. Une correlation parfaite vaudrait +1 ou -1. Plus la correlation s'approche de 0, moins les données sont similaires et liées. Le package python `scipy` offre une telle méthode : `scipy.stats.pearsonr`. Les corrélations pour les deux graphes de la figure 16 sont les suivantes :

- Training set : Correlation de Pearson = 0.77
- Testing set : Correlation de Pearson = 0.57

Ces résultats semblent en accord avec la méthode visuelle illustrée en figure 16

9.3.2.3 Distribution des radiances 2D

Une autre manière de visualiser les prédictions, bien plus représentative de la réalité, consiste à reconstruire la portion de l'image constituante les données de nuit à partir des prédictions, en les comparant avec la même reconstruction cette fois-ci effectuée avec les ground truths. La figure 17 illustre cette visualisation.

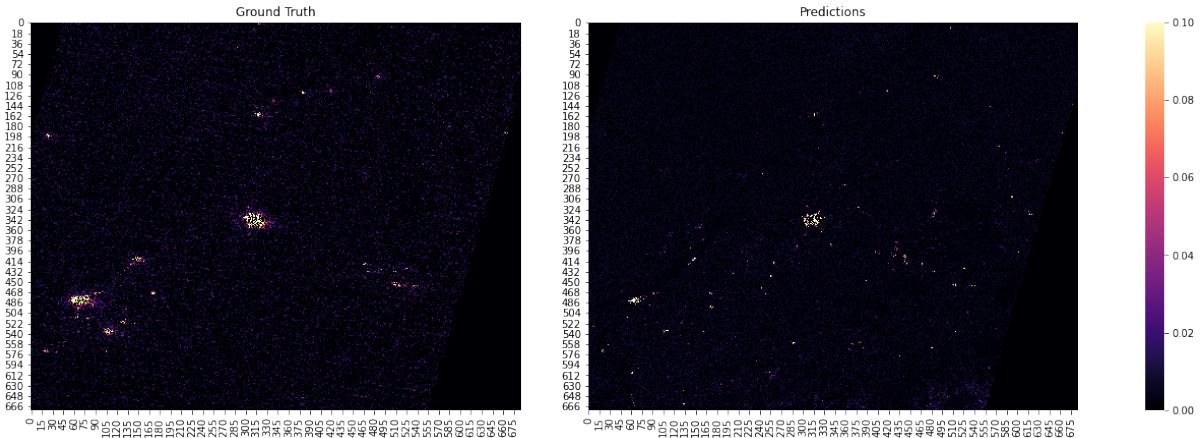


FIGURE 17 – Comparaison de la distribution bi-dimensionnelle des radiances du training set, ground truths vs predictions

Comme expliqué en section 8.5.1, à cause de la cross-validation, les données apparaissent parsemées. Il serait évidemment possible d'afficher les prédictions du training set ainsi que du testing set sur la même image afin de produire une image uniforme, mais cela aurait en vérité assez peu de sens puisqu'il ne serait plus possible de distinguer quel pixel de l'image de nuit a été prédit selon de training set ou le test set.

Sur la droite de la figure 12, une échelle indique la couleur qu'a un pixel en fonction de la radiance. Afin de mieux faire ressortir les zones détectées comme lumineuses, cette échelle a été bornée sur un interval [0, 0.1], plutôt qu'un interval de [0, 1] qui produirait une image très sombre.

9.3.2.4 Différences avec les *Ground Truths*

Similaire à la méthode précédente, on peut afficher de la même manière une carte, mais cette fois-ci de la différence entre les ground truths et les prédictions. La figure 18 illustre cette méthode.

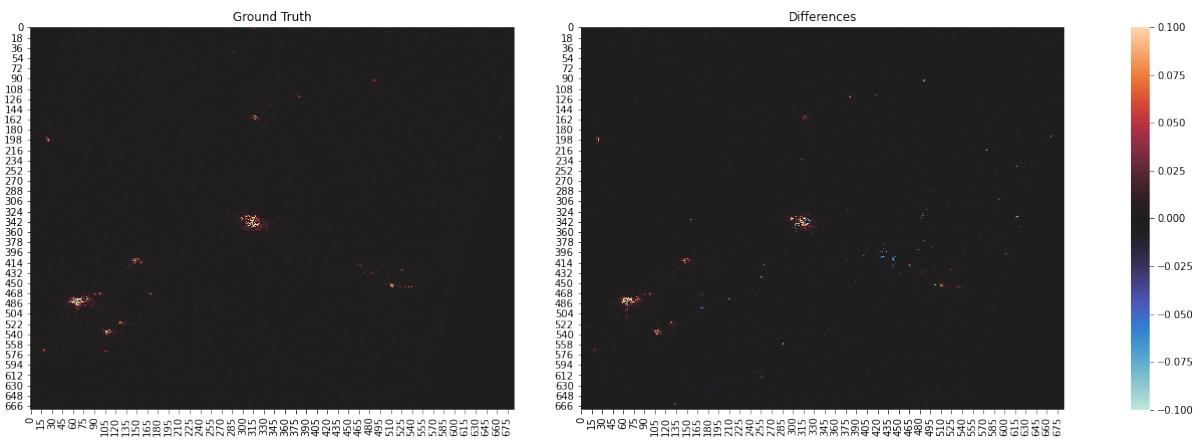


FIGURE 18 – Comparaison des différences entre ground truths et prédictions sur le testing set

L'échelle de couleur de cette visualisation nous indique 3 types de différences principales :

- En noir, les différences valent proche de 0. Cela signifie que la prédiction est presque parfaite.

- En rouge-orange, les différences positives. Cela signifie que la prédiction était plus basse que la valeur réelle. Comme on a pu le voir en section 9.3.2.1, nos prédictions sur les zones lumineuses ont tendance à être plus basse qu'en vérité. Ce biais est d'autant plus évident en observant que chaque grande ville illuminée dans les ground truths apparaît aussi relativement illuminée sur l'image des différences (elle n'apparaîtrait pas du tout avec des prédictions parfaites). Cependant, cela n'est pas un vrai problème, puisque nous nous intéressons pas vraiment à ce genre de prédictions.
- En bleu-cyan, les différences négatives. Ce sont celles qui nous intéressent, correspondant à des zones non illuminées mais que notre modèle prédit comme lumineuses. Nous espérons donc que ce sont les villages non éclairés que nous cherchons à détecter.

9.3.2.5 Binarisation des résultats

Finalement, nous pouvons utiliser une dernière méthode afin de visualiser les résultats obtenus. Cette méthode, bien plus facile pour évaluer la performance du modèle pour détecter les villages n'émettant pas de lumière, consiste à "binariser" les résultats obtenus. Comme discuté en section 6.2.3, toute radiance (absolue, non normalisée) inférieure à 3 est équivalente à n'émettre aucune lumière.

On peut alors décider d'une deuxième valeur représentant une zone lumineuse, et extraire l'ensemble des emplacements prédits lumineux mais ne l'étant pas en vérité. Dans notre cas, nous utiliserons une valeur de radiance (absolue, non normalisée) de 8 pour représenter ces zones. On peut alors exporter ces emplacements sous divers formats afin de pouvoir les visualiser. Nous les exporterons au format CSV (Comma-Separated Values) afin de les rendre compatibles avec de nombreux logiciels, chaque emplacement détecté prenant la forme `{Emplacement GPS, Radiance prédite}`.

Le service gratuit "Google My Maps" [22] nous permet d'afficher des marqueurs sur les cartes fournies par Google, ainsi que de modifier leur couleur d'affichage facilement. Il nous permet aussi d'importer nos emplacements générés au format CSV et afficher les marqueurs correspondants, en rajoutant un layer et en cliquant sur "importer". Il suffit alors de spécifier quelle colonne correspond aux coordonnées GPS, et quelle colonne correspond au label du marqueur. Dans notre cas, ce sera la radiance prédite.

On peut aussi importer plusieurs fichiers à la fois. Nous importerons donc celui généré avec les données "training", ainsi que celui généré avec les données "testing" afin de pouvoir les comparer facilement.

Le résultat obtenu est illustré en figure 19 :

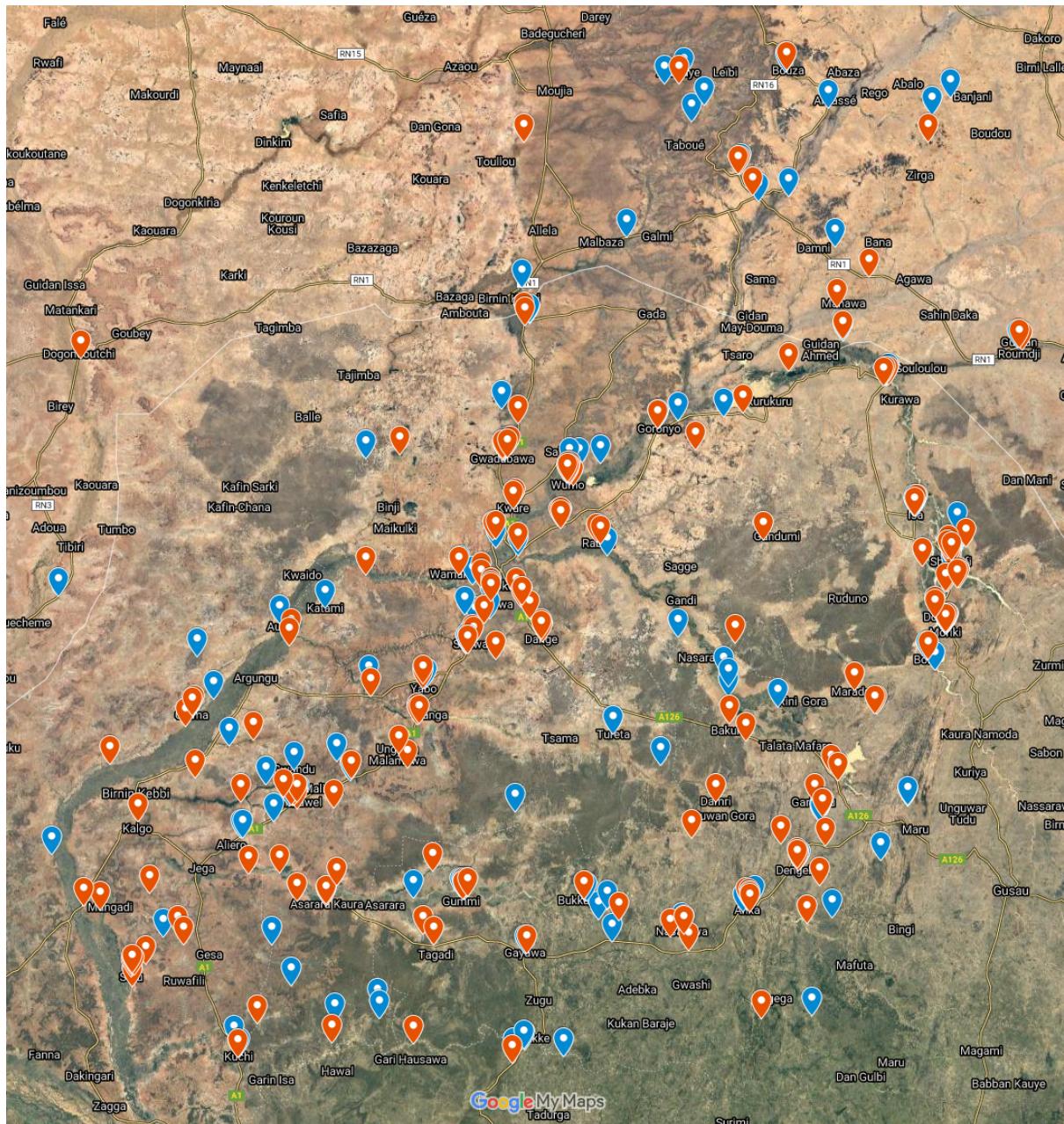


FIGURE 19 – Lieux détectés comme lumineux n'émettant en vérité pas de lumière, visualisé sur Google My Maps. En bleu, les emplacements provenant du training set, et en rouge les emplacements provenant du testing set

En observant les résultats, on observe qu'ils correspondent en grande majorité à des villages n'émettant pas de lumière. Cependant, de manière assez surprenante, on observe aussi que les marqueurs bleus, correspondant aux données provenant du training set, sont presque aussi nombreux que les marqueurs rouges provenant du testing set, ce qui est en contradiction avec nos attentes. Cependant, ce n'est pas une mauvaise surprise. En effet, cela nous permet de détecter plus de villages n'émettant pas de lumière que simplement ceux provenant du training set.

10 Discussion des résultats

Les résultats obtenus, particulièrement ceux observés sur "Google My Maps" (figure 19), semblent assez prometteurs et en accord avec ceux décrits dans le cahier des charges. On observe une grande quantité de villages qui sont prédisits comme émettant de la lumière par le modèle final, alors qu'ils n'émettent en vérité pas de lumière.

Avec ces résultats binarisés, on pourrait maintenant essayer d'appliquer les métriques généralement utilisée dans les problèmes de classification : La précision ainsi que le rappel. Malheureusement, ces deux mesures sont relativement compliquées à calculer avec les données que nous avons à disposition.

La précision demanderait qu'un humain passe sur chaque marqueur et indique s'il indique ou non un village. Il serait possible d'automatiser ces processus à l'aide d'une nouvelle source de données, par exemple une grille de densité de population telle celle fournie par Facebook pour le Nigeria en 2020 [23]. Cependant, ces cartes elles-mêmes sont générées grâce à du Machine Learning. Peuvent-elles vraiment être considérées comme des "Ground Truths" ? Evidemment, la question n'est pas de savoir si ces données sont bonnes ou non. La méthodologie décrite et les résultats obtenus donnent même une bonne confiance en leur performance. La question est plutôt de savoir si l'on peut valider des prédictions basées sur des données Near Real Time avec des données datant de plusieurs mois, ou encore valider des prédictions de Machine Learning avec des données générées similairement ?

Pour le rappel, une problématique similaire mais encore plus compliquée nous fait face : Qu'est-ce qui peut être considéré comme un village à détecter, et qu'est-ce qui ne l'est pas ? Le problème principal de cette question est lié à la taille du village. Nos prédictions se font sur des zones de 500m par 500m, ou $0.25km^2$. De ce fait, un "village" beaucoup plus petit que cela aura évidemment plus de difficultés à être détecté. La figure 20 illustre un petit "village" mesurant moins de 100m de côté. Devrait-il être considéré comme un village à détecter ?



FIGURE 20 – Village ne produisant pas de lumière mesurant moins de 100m de côté

De plus, contrairement à la précision qui pourrait être déterminée à la main assez facilement (quelques centaines de prédictions à confirmer), déterminer le rappel à la main est pratiquement impossible : Il faudrait passer sur chacun des 417'000 images et déterminer si elle correspond à une zone habitée ou non.

Malgré ces questions sans réponses, on peut quand même jeter un coup d'oeil aux données sur Google My Maps pour se faire une idée de leur qualité. La carte est disponible dans le repo GitHub du projet [3]. En passant rapidement sur la majorité des marqueurs, on se rend compte que leur précision est très bonne. Les quelques marqueurs positionnés sur des zones non habitées ressemblent généralement à celui illustré en figure 21 : Une zone couverte de buissons ras du sol, parsemée de zones clairs sans végétation.

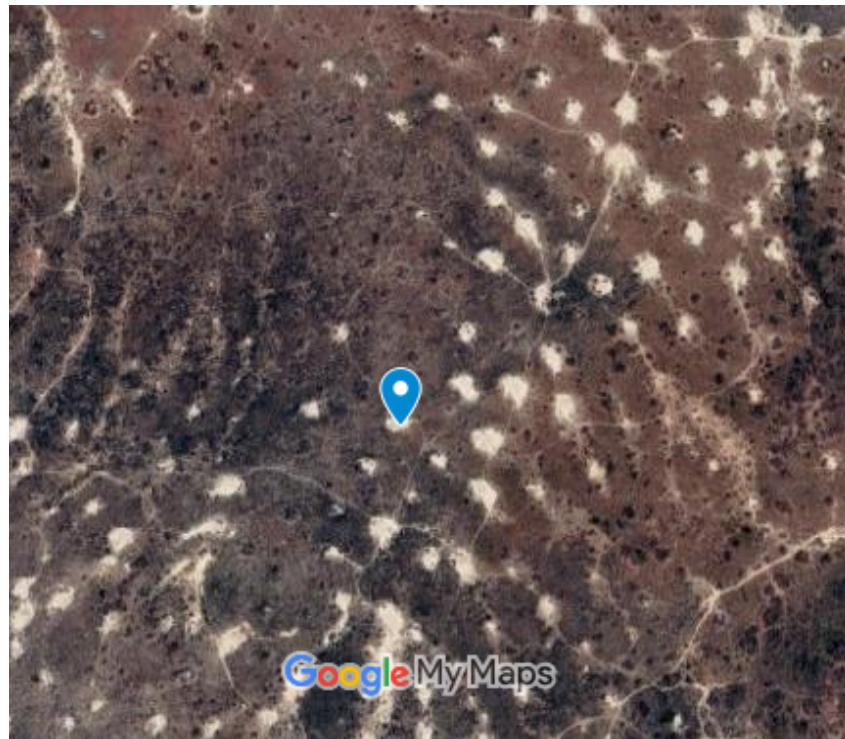


FIGURE 21 – Zone non habitée faussement détectée comme émettant de la lumière

Il n'est pas difficile de comprendre la raison pour laquelle le modèle s'est trompé : Les zones claires peuvent facilement être prises pour des maisons, et elles sont même reliées par des sortes de chemins.

Heureusement, leur nombre reste très faible, pas plus de quelques pourcents du total des prédictions. Le faible taux de détection de ce genre de zones est l'un des facteurs principal ayant mené au choix du modèle final présenté en section 9.3. La majorité des modèles générés avant celui-là détectaient beaucoup plus ce genre de zones comme émettant de la lumière, comme l'illustre la figure 22 provenant d'un ancien modèle de prédiction. Pour rappel, chacun des marqueurs bleus représente une zone de 500m par 500m.

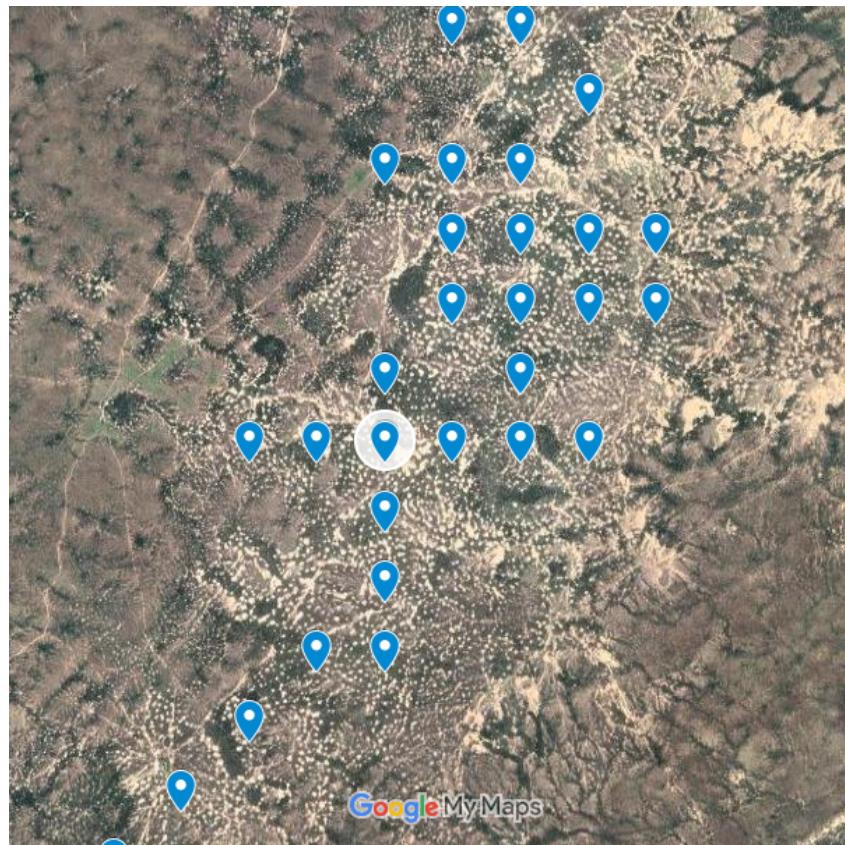


FIGURE 22 – Grande zone mal interprétée comme un grand village émettant de la lumière par un mauvais modèle de prédiction

Quant à une estimation visuelle du rappel, cela est bien plus difficile à faire. Il est cependant facile de voir que nombreux villages qui n'émettent pas de lumière (une rapide comparaison avec les données de nuit suffit) ne sont pas détectés. Différents facteurs peuvent en être la cause. Premièrement, le valeur inférieure de 8 radiance non normalisée peut être trop haute pour détecter ces villages. Cependant, abaisser cette valeur plus bas que 8, qui est déjà un chiffre très bas, permet de ne détecter que quelques villages supplémentaires, mais décuple le nombre de mauvaises détections par plusieurs dizaines de fois.

11 Améliorations possibles

Bien que les résultats obtenus semblent assez bons, de nombreuses améliorations pourraient être apportées au projet. Ces améliorations n'ont pas été abordées pour ce projet, principalement par question de temps, ou par limitations physiques.

11.1 Meilleure résolution spatiale

Une première amélioration possible assez évidente est d'augmenter la résolution spatiale du jeu de données de jour. Actuellement, les données possèdent une résolution de 10m par pixel, ce qui implique que tout objet ou forme mesurant moins de 10m de côté n'apparaîtra pas clairement sur l'image satellite. La figure 23 illustre la différence de résolution entre une case de 500m par 500m produite par Sentinel-2 et une case de la même région, disponible sur Google Maps⁷.

7. <https://bit.ly/3zJnbns>

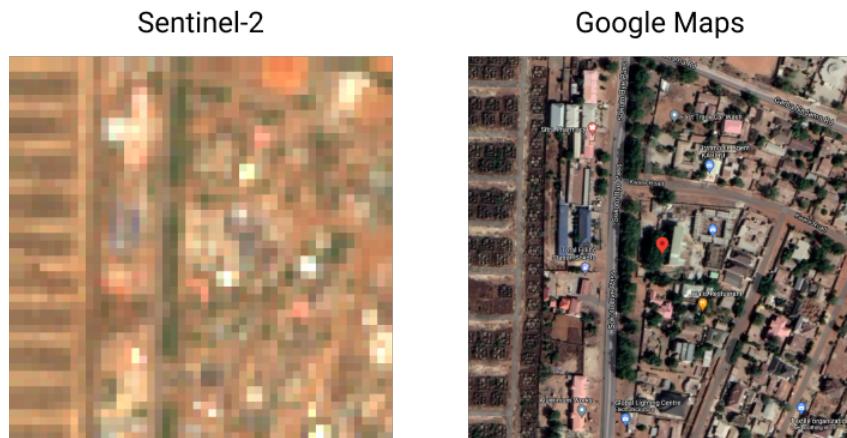


FIGURE 23 – Comparaison de la résolution de Sentinel2 et Google Maps

Cependant, comme mentionné auparavant, Google Maps ne propose pas de données Near Real Time et ne sont donc pas adaptées pour ce projet. Une autre source de données plus précise devrait être trouvée.

11.2 Comparaison des résultats avec d'autres jeux de données

Comme expliqué en section 10, il serait assez intéressant de comparer les résultats obtenus de manière scientifique afin de pouvoir tirer de nouvelles conclusions. Ceci n'a pas été abordé durant ce projet par manque de temps, en plus des raisons mentionnées en section 10.

En particulier, avoir une métrique telle que le rappel des villages sans lumière détectés, même imparfaite, permettrait de grandement automatiser le processus de test des topologies et paramètres optimaux en maximisant cette métrique plutôt que de comparer les résultats de prédiction binarisés visuellement.

11.3 Cloud Computing

Une autre amélioration possible est celle mentionnée en section 7.1.2.3. Bien que pas forcément la plus intéressante pour un travail de bachelor tel que celui-ci, cette amélioration serait extrêmement intéressante à explorer dans un autre contexte.

Par exemple, on pourrait implémenter la même pipeline que pour ce projet sur AWS Sagemaker⁸ et comparer le temps de mise en place ainsi que les temps d'exécution des différentes étapes. On pourrait aussi prendre avantage des outils de tuning proposés qui permettent de tester beaucoup de combinaisons de paramètres en parallèle et ainsi beaucoup gagner en temps et perfectionner le modèle de prédiction.

11.4 Augmentation des données

Un dernier facteur qui pourrait être bénéfique serait d'effectuer de l'augmentation sur les données d'entraînement. Augmenter les données est une manière communément acceptée de réduire l'overfitting, ainsi que de générer plus de données que celles existant réellement, par exemple en effectuant diverses rotations et inversions des images.

L'effet qu'aurait le fait d'augmenter les données sur les prédictions est difficile à prédire : Le modèle généré ne souffrant pas d'overfitting, cela reviendrait principalement à augmenter la taille du jeu de données. Pour rappel, le jeu de données du Sokoto contient près de 450'000

8. <https://aws.amazon.com/sagemaker/>

images de 50x50 pixels. L'utilité qu'aurait cette technique est difficile à prédire, mais aurait une chance d'augmenter le taux de détection de villages n'émettant pas de lumière. L'augmentation des données vaut donc la peine d'être essayée.

12 Conclusion

Au début de ce projet, nous avons décidé de trouver une solution au manque de données de recensement dans les pays pauvres. Particulièrement, nous nous sommes intéressés aux zones habitées n'ayant pas accès à l'électricité. Nous avons décidé de centrer notre solution sur le Machine Learning et les données satellites Near Real Time.

La première partie du travail, l'exploration et la préparation des données, a été l'une des étapes les plus chronophages du projet. De nombreux changements de sources de données ainsi que des changements de "frameworks" ont demandé la ré-écriture de grandes parties du code, et ce plusieurs fois. Cependant, chacun de ces changements fut au final extrêmement positif, améliorant grandement la qualité des données pour la suite du projet.

La deuxième partie du travail, centrée sur les réseaux de neurones, était d'abord exploratoire. En effet, j'ai découvert le monde du Machine Learning et des réseaux de neurones au cours de ce projet. Avec une bonne base théorique, la phase d'optimisation des modèles de prédictions a pu démarrer. A nouveau, cette phase fut très chronophage. Tester une configuration de réseau peut prendre plusieurs heures, et une combinaison presque infinie de ces configurations existe. Ainsi, le modèle final produisant les meilleurs résultats n'aura été "découvert" que quelques jours avant la fin du projet, puisque la phase de test des configurations ne s'est pas arrêtée avant la toute fin du projet.

Grâce à l'utilisation d'images de nuit du satellite Black Marble comme proxy de présence humaine ainsi que d'images de jour du satellite Sentinel-2, nous avons pu entraîner des modèles de prédictions permettant de nous indiquer l'emplacement de villages ne produisant pas de lumière la nuit, et n'ayant donc probablement pas accès à l'électricité. Bien que les résultats ne soient parfaits, particulièrement concernant le taux de rappel difficile à mesurer, ils démontrent tout de même une réelle capacité de prédiction sur les villages recherchés, ainsi qu'une très bonne précision. En mettant en pratique les divers éléments mentionnés en section 11, il sera sûrement possible de générer de bien meilleures prédictions.

Je pense donc que l'ensemble des objectifs initiaux mentionnés dans le cahier des charges en section 3 ont été atteints. D'un point de vue plus personnel, j'ai pu beaucoup apprendre au cours de ce projet. Allant de l'organisation d'un travail personnel à la gestion des imprévus et bien sûr le machine learning, ce travail m'aura permis de faire l'expérience de nombreuses situations qui me seront sûrement bénéfiques durant la suite de mon parcours.

Bibliographie

- [1] *La Banque Mondiale*. https://donnees.banquemoniale.org/indicator/EG.ELC.ACCS.ZS?end=2018&most_recent_value_desc=true&start=1990&view=chart. Accessed : 2021-05-17.
- [2] CHRISTOPHER YEH, ANTHONY PEREZ, ANNE DRISCOLL, GEORGE AZZARI, ZHONGYI TANG, DAVID LOBELL, STEFANO ERMON & MARSHALL BURKE. « Using publicly available satellite imagery and deep learning to understand economic well-being in Africa ». In : *Nature Communications* 2583 (2020). DOI : <https://doi.org/10.1038/s41467-020-16185-w>.
- [3] Matthieu BURGUBURU. *La Terre vue de nuit*. <https://github.com/matt989253/heig-TerraGazer>. 2021.
- [4] *NASA Earthdata Search*. <https://search.earthdata.nasa.gov/search>. Accessed : 2021-05-17.
- [5] *NASA Earthdata Worldview*. <https://worldview.earthdata.nasa.gov>. Accessed : 2021-07-27.
- [6] DR. MIGUEL O. ROMÁN. *NASA's Black Marble Nighttime Lights Product Suite Algorithm Theoretical Basis Document (ATBD)*. NASA Goddard Space Flight Center. Greenbelt, Maryland, 2020.
- [7] *Google Earth Engine*. <https://code.earthengine.google.com>. Accessed : 2021-05-17.
- [8] *USGS Landsat-7*. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-7?qt-science_support_page_related_con=0#qt-science_support_page_related_con. Accessed : 2021-07-26.
- [9] *Sentinel Hub EO Browser*. <https://apps.sentinel-hub.com/eo-browser>. Accessed : 2021-05-17.
- [10] *Documentation Sentinel-Hub*. <https://sentinelhub-py.readthedocs.io/en/latest/>. Accessed : 2021-07-27.
- [11] *Images Wikipedia illustrant différentes méthodes d'interpolation*. <https://bit.ly/2RZzxY2>, <https://bit.ly/2RXFGUP>, <https://bit.ly/3olvq4N>, <https://bit.ly/2SQxANX>. Accessed : 2021-05-17.
- [12] *Documentation du format de fichier JP2*. <https://fileinfo.com/extension/jp2>. Accessed : 2021-07-27.
- [13] *Documentation du format de fichier H5*. <https://fileinfo.com/extension/h5>. Accessed : 2021-07-27.
- [14] *MGRS vs UTM*. https://www.usna.edu/Users/oceano/pguth/md_help/html/mgrs_utm.htm. Accessed : 2021-07-27.
- [15] *EPSG Translator*. <https://epsg.io/>. Accessed : 2021-07-27.
- [16] *Black Marble Satellite Information page*. <https://ladsweb.modaps.eosdis.nasa.gov/missions-and-measurements/products/VNP46A1/>. Accessed : 2021-07-27.
- [17] Xiangyu ZHANG et al. *Accelerating Very Deep Convolutional Networks for Classification and Detection*. 2015. arXiv : [1505.06798 \[cs.CV\]](https://arxiv.org/abs/1505.06798).
- [18] *Keras*. <https://keras.io>. Accessed : 2021-05-17.
- [19] Axel THEVENOT. *Conv2d : Finally Understand What Happens in the Forward Pass*. <https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148>. Accessed : 2021-07-28.
- [20] *Commonly used activation functions*. <https://bit.ly/3xe7cMu>. Accessed : 2021-07-28.
- [21] *K-fold validation in machine & deep learning*. <https://bit.ly/3rDghxm>. Accessed : 2021-07-28.
- [22] *Google My Maps*. <https://www.google.com/mymaps>. Accessed : 2021-07-27.

- [23] FACEBOOK. *Nigeria : High Resolution Population Density Maps + Demographic Estimates.* <https://data.humdata.org/dataset/highresolutionpopulationdensitymaps-nga>. Accessed : 2021-07-29.

13 Authentification

Je soussigné, Matthieu Burguburu, atteste par la présente avoir réalisé individuellement ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Yverdon-les-Bains, le 30 juillet 2021.

A handwritten signature in black ink, appearing to read "Burguburu".

Matthieu Burguburu

Table des figures

1	Découpage d'une bande d'images de la région du Sokoto, capturées par le satellite Sentinel2 et visualisées le site Sentinel-Hub	9
2	Différentes méthodes d'interpolation [11]	11
3	Masque de nuages du satellite Black Marble	12
4	Comparaison de l'effet du reflet	14
5	Bruit numérique provenant directement du senseur de lumière, illustré sur les données du Sokoto	14
6	Extrait du manuel des cartes "Black Marble" [6] représentant le découpage du jeu de données disponibles	17
7	Deux cases Sentinel-2 adjacentes avec différentes inclinaisons	18
8	Illustration simplifiée du problème de l'alignement initial des données	19
9	Distribution des valeurs de radiance avant et après normalisation	21
10	Distribution des valeurs de radiance avant et après normalisation après rectification de la valeur maximale	21
11	Comparaison de 4 fonctions d'activations les plus utilisées [20]	26
12	Comparaison des séparations de cross-validation "Split" vs "Sparse"	28
13	5-Fold Validation [21]	28
14	Evolution de la fonction de coût selon les epochs. Training set en bleu, Testing set en orange. Le graphe de droite est une version "zoomée" de celui de gauche .	31
15	Comparaison de la distribution uni-dimensionnelle des radiances du training set, ground truths vs predictions	31
16	Graphe comparant les valeurs prédites avec les valeur réelles. La ligne rouge indique les prédictions parfaites.	32
17	Comparaison de la distribution bi-dimensionnelle des radiances du training set, ground truths vs predictions	33
18	Comparaison des différences entre ground truths et prédictions sur le testing set .	33
19	Lieux détectés comme lumineux n'émettant en vérité pas de lumière, visualisé sur Google My Maps. En bleu, les emplacements provenant du training set, et en rouge les emplacements provenant du testing set	35
20	Village ne produisant pas de lumière mesurant moins de 100m de côté	36
21	Zone non habitée faussement détectée comme émettant de la lumière	37
22	Grande zone mal interprétée comme un grand village émettant de la lumière par un mauvais modèle de prédiction	38
23	Comparaison de la résolution de Sentinel2 et Google Maps	39

Liste des tableaux

1	Informations principales des différentes sources de données envisagées	8
2	Table des bits du Cloud Mask du satellite Black Marble	13
3	Topologie du modèle final	30
4	Hyper-paramètres du modèle final	30