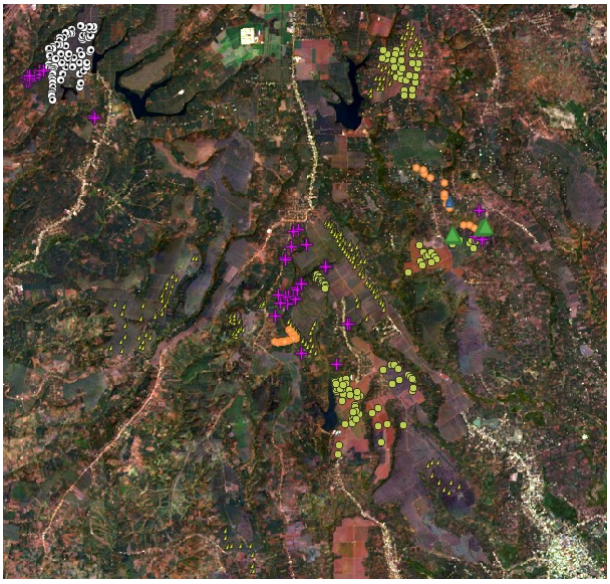
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département TIC
Filière Informatique
Orientation Logiciel

Travail de Bachelor

Une IA pour surveiller les engagements de zéro déforestation

Non confidentiel



Étudiant : **Simon Walther**

Travail proposé par : Louis Reymondin
International Center for Tropical Agriculture
CIAT - International Center for Tropical Agriculture
Vietnam

Enseignant responsable : Andres Perez-Uribe

Année académique : 2020-2021

Yverdon-les-Bains, le 15 mai 2021

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département TIC
Filière Informatique
Orientation Logiciel
Étudiant Simon, Walther
Enseignant responsable Andres, Perez-Uribe

Travail de Bachelor 2020-2021

Une IA pour surveiller les engagements de zéro déforestation

Centre International pour l'Agriculture Tropical

Résumé publiable

Étudiant : Walther Simon	Date et lieu :	Signature : .....
Enseignant responsable : Perez-Uribe Andres	Date et lieu :	Signature : .....
International Center for Tropical Agriculture : Louis Reymondin	Date et lieu :	Signature : .....

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 15 mai 2021

Authentification

Le soussigné, Simon Walther, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Aigle, le 15 mai 2021

Simon Walther

Cahier des charges

Ayant comme but l'analyse des images satellites pour détecter des champs de café au Vietnam, ce projet se déroulera de la manière suivante :

1. Programmation des scripts pour accéder aux images satellites.

2. Exploration des données disponibles, visualisation et cartographie des données géoréférencées.

3. Préparation des données pour entraîner des modèles de classification basés sur des réseaux de neurones profonds.

4. Exploitation de multiples sources de données pour entraîner les modèles. Par exemple, exploitation des divers canaux des imageries multispectrales, de différentes périodes de l'année etc.

5. Entraînement de modèles de classification à l'aide des techniques de Machine Learning, notamment des réseaux de neurones convolutifs, analyses des performances des différents modèles.

6. Développement d'outils permettant la visualisation et l'analyse de résultats.

7. Déterminer les zones à risques de déforestation en suivant par exemple l'évolution de la forêt au Vietnam, notamment dans des régions où des plantations de café sont présentes.

Table des matières

Table des matières

1 Énoncé

Dans le cadre d'un projet de collaboration avec le Centre de Recherche en Agriculture Tropicale - CIAT et le King's College London (KCL), on développe des outils exploitant des algorithmes de Machine Learning pour traiter des informations fournies par des capteurs de télédétection (e.g. par des satellites), avec l'objectif de détecter la déforestation et surveiller les changements dans l'utilisation des sols.

Dans le cadre de ce projet, nous avons l'objectif de traiter des images fournies par des satellites pour entraîner des réseaux de neurones (Deep Learning) pour détecter des champs de café ayant remplacé la forêt au Vietnam.

Beaucoup d'entreprises au Vietnam ont signé des accords de zéro déforestation dans leur "commodity chain". Or, leur problème est qu'actuellement ils ne savent pas comment vérifier si le produit qu'ils achètent vient d'une région déforestée récemment ou non [1].

En collaboration avec le centre de recherche sur l'Agriculture Tropicale (CIAT) au VietNam, nous aurons accès à des annotations de champs de café pour entraîner des réseaux de neurones et nous travaillerons sur la mise en place d'un système de vérification de non-déforestation d'une région.

Plus concrètement, il s'agit de:

- Exploiter des images satellite (Landsat 8) historiques (p.ex. à partir de 2013) pour détecter la deforestation au Vietnam, en utilisant les techniques du Machine Learning
- Comparer les résultats avec des cartes existantes et utiliser d'autres sources de données pour affiner la détection de la déforestation
- Utiliser et adapter des modèles Machine Learning de détection des champs de café et autres "commodities" pour évaluer si ces champs sont le produit d'une déforestation après 2013 ou non.

2 Contexte

Le Vietnam est le second plus gros producteur de café du monde. En tout, c'est à peu près 1.7 million de tonnes de café produit par an [2]. C'est le robusta qui est la sorte de café prédominante au Vietnam, comptant pour 97 % de la production [3].

Là-bas, la culture de café représente environ 6500 km² et le nombre de petits agriculteurs important: on estime qu'il y en a 640'000 [4].

Le Vietnam s'est engagé à la Convention-cadre des Nations unies sur les changements climatiques à réduire de 8 % les émissions de gaz à effet de serre. Pour arriver à ce but, il est primordial pour le Vietnam de réussir à limiter la déforestation [5].

Pour permettre une culture du café sans déforestation, l'agroforesterie (c.-à-d. l'introduction d'arbres dans les champs [5]) est mise en avant par le gouvernement Vietnamien [4]. Par exemple, le ministère de l'agriculture et du développement rural (MARD) a intégré des arbres fruitiers, du poivre, des avocats et encore d'autres arbres dans des champs de café [5].

Figure 1: Café robusta, anacardier, et arbres fruitiers dans la province du Dak Nong, Photo : World Agroforestry/Nong Lam University, Ho Chi Minh City, trouvé dans : Diversity of agroforestry practices in Vietnam, World Agroforestry (ICRAF) Viet Nam

3 Abréviations

API [6]: «une interface de programmation d'application ou interface de programmation applicative [...] est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.»

GEE : Google Earth Engine

CIAT [7]: «International Center for Tropical Agriculture», Centre international pour l'agriculture tropicale.

CNN : «Convolutional Neural Network», réseau de neurones convolutifs.

4 Landsat 8

Nous utilisons les images fournies par le satellite Landsat 8. Ce satellite comporte 11 bandes de fréquences différentes [8,9] :

Bande	Fréquence [µm]	Résolution
1. Coastal aersol	0.433–0.453	30 m
2. Blue	0.450–0.515	30 m
3. Green	0.525–0.600	30 m
4. Red	0.630–0.680	30 m
5. NIR	0.845–0.885	30 m
6. SWIR-1	1.560–1.660	30 m
7. SWIR-2	2.100–2.300	30 m
8. Panchromatic	0.500–0.680	15 m
9. Cirrus	1.360–1.390	30 m
10. Tir-1	10.6-11.2	100 m
11. Tir-2	11.5-12.5	100 m

Parmi celles-ci, seules les bandes de fréquences *coastal aerosol* (côtier/aérosol [10]), *blue* (bleu), *green* (vert), *red* (rouge) et *panchromatic* (panchromatique) sont visibles à l'œil nu [8]. Les données que nous avons à disposition nous permettent alors de voir plus que de simples images composées de rouge, de vert et de bleu.

Pour ce qui est de la résolution, le satellite Landsat 8 a une résolution de 30m pour la majorité des bandes. Cela signifie qu'un pixel représente une région de 30 mètres carré.

Voici une image d'un terrain de baseball permettant de mieux se rendre compte de ce que cela représente ; le carré jaune mesurant 30 mètres par 30 mètres :

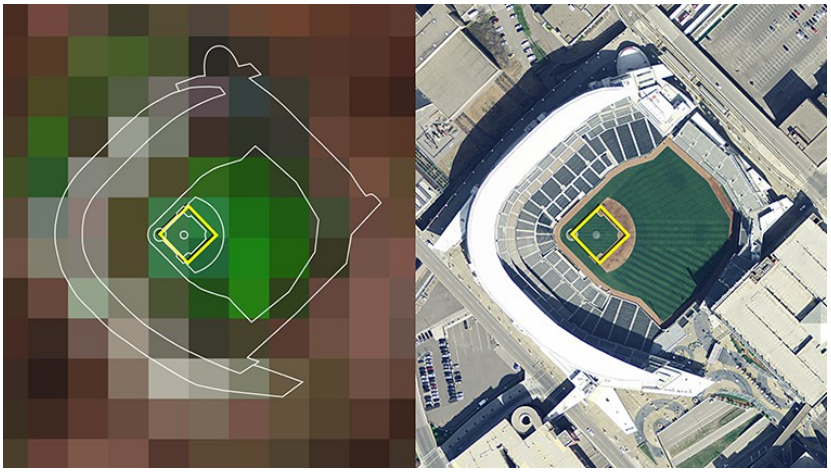


Figure 2: Comparaison entre une photo aérienne et une image Landsat, https://landsat.gsfc.nasa.gov/article/picturing-pixel

5 Introduction aux réseaux de neurones convolutifs

Qu'est-ce que sont les réseaux de neurones convolutifs, pourquoi sont-ils utilisés pour répondre à notre problématique ? En quoi aident-ils à détecter les champs de café pour observer la déforestation au Vietnam ? Et comment ?

Pour pouvoir répondre à ces questions, il faut commencer par aborder ce qu'est un réseau de neurone artificiel.

Les réseaux de neurones artificiels veulent mimer le fonctionnement du cerveau. Si l'on s'intéresse à ce qui compose les neurones, on trouvera qu'ils sont composés d'entrées, les dendrites qui reçoivent des signaux, d'une unité de traitement le soma qui va traiter les signaux qu'il reçoit, d'axons qui transmettent le résultat du soma et enfin de synapses qui permettent de connecter les neurones entre-eux [11].

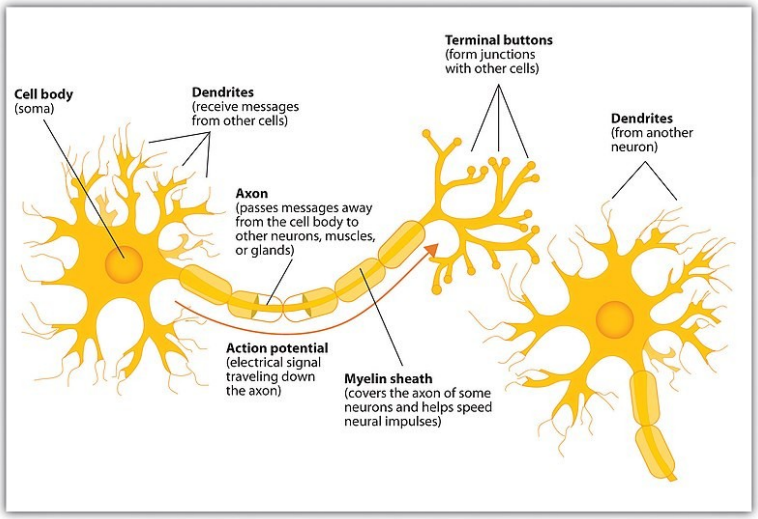


Figure 3: Représentation des composants d'un neurone, https://en.wikipedia.org/wiki/File:Components_of_neuron.jpg

5.1 Neurones artificiels

À partir de cette vue simplifiée de ce qu'est un neurone, un modèle de neurone a pu être établi par McCulloch et Walter Pitts. Ce modèle se présente ainsi: un neurone reçoit des informations par des entrées x_1, x_2, ... x_n prenant des valeurs de zéro ou un. Une fonction regroupe les entrées en faisant leur somme et l'on obtient un résultat y booléen (zéro signifie faux et un vrai). Ce résultat est à un si la somme des valeurs des entrées est supérieure ou égale à un seuil θ, sinon à zéro [11].

Ce qui donne le calcul suivant :

$$y = \begin{cases} 1 & si \sum_{k=1}^{n} x_k \geq \theta \\ 0 & sinon \end{cases}$$

Imaginons que l'on veut savoir s'il faut prendre un parapluie. On pourrait alors faire le modèle que voici :

On voit qu'on a, dans cet exemple, deux entrées représentant s'il pleut et si l'on est dehors et qu'il faut que celles-ci soient à un (soient vraies) pour que le neurone détermine qu'il faut prendre un parapluie.

Dans ce cas-ci le seuil est à deux. Ainsi il faut que les deux entrées soient vraies. S'il avait par-exemple été à un, alors il aurait suffit

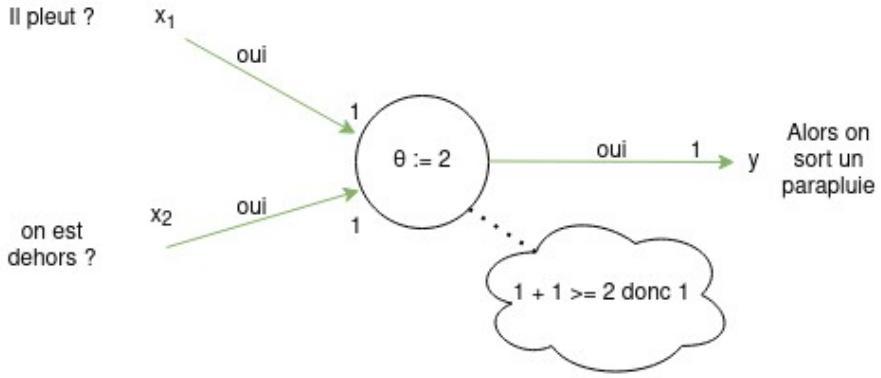


Figure 4: Exemple de condition booléene AND avec un neurone McCulloch-Pitts

qu'une des deux conditions soit vraie, car la somme des entrées aurait été supérieure ou égale au seuil de un.

Il est aussi possible d'avoir une entrée inhibitrice *i* qui, si elle a une valeur de un, alors le résultat vaut zéro quelque soit les valeurs des autres entrées :

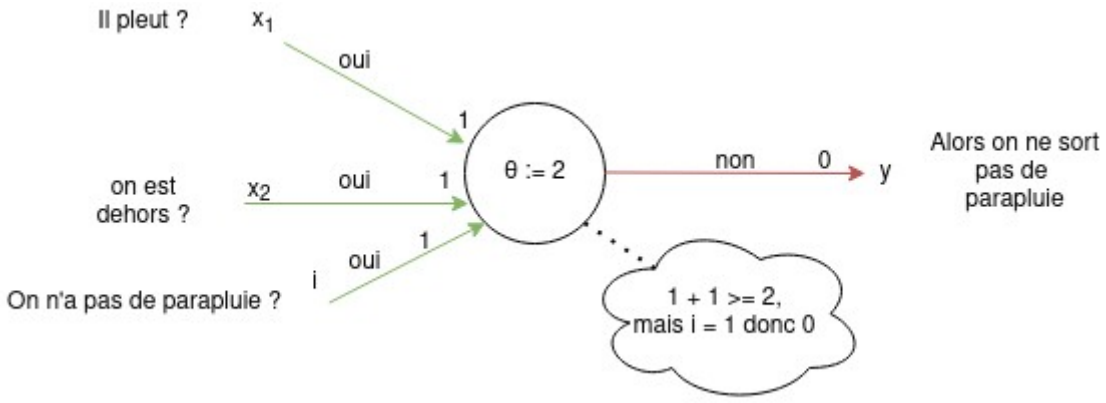


Figure 5: Exemple d'entrée inhibitrice avec un neurone McCulloch-Pitts

De nouveaux modèles se sont basés sur celui-ci pour l'améliorer, car il comporte quelques défauts : on voudrait pouvoir avoir des entrées qui soient des nombres réels, on ne voudrait pas forcément que les entrées aient toutes le même poids dans la décision et on voudrait ne pas à avoir à définir le seuil à la main [11].

Un de ces modèles est le perceptron. Il a été conçu par Frank Rosenblatt et corrige certains défauts du modèle de McCulloch-Pitts, ce qui va permettre de faire apprendre des neurones.

Ce modèle rajoute des poids pour chacune des entrées, poids qui peut être un nombre réel, ce qui permet de donner plus d'importance à certaines entrées qu'à d'autres. Une autre modification est que le seuil θ est ajouté comme une entrée constante négative *b* appelée maintenant le biais. Enfin, au lieu d'avoir une règle d'inhibition, on a maintenant la possibilité d'avoir des entrées négatives [12].

Le calcul sera alors le suivant : $\begin{cases} 1 & si(\sum_{k=1}^{n} w_k x_k)+b \geq 0 \\ 0 & sinon \end{cases}$

Les poids nous seront très utiles pour apprendre, on va pouvoir apprendre quels sont les poids qui nous donnent les meilleurs résultats.

Pour pouvoir résoudre des problèmes plus compliqués, on ne va plus seulement utiliser qu'un seul neurone, mais on va former des réseaux de neurones.

5.2 Réseaux de neurones

Voilà à quoi ressemble un réseau de neurone :

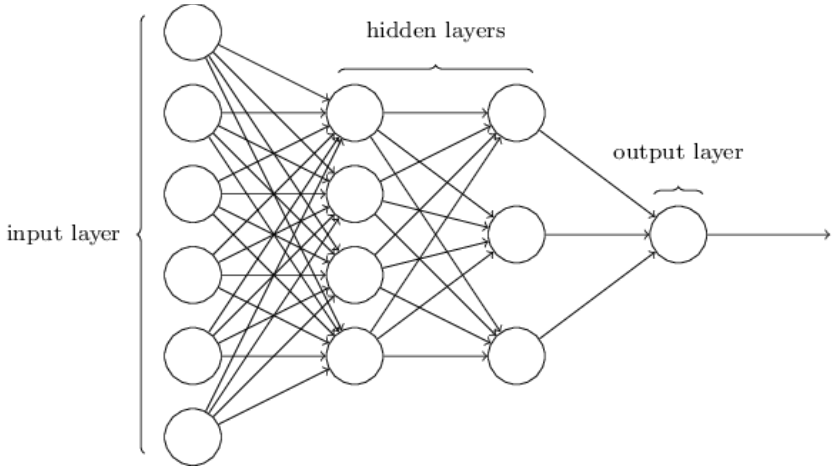


Figure 6: Exemple de réseau de neurone avec la terminologie.
http://neuralnetworksanddeeplearning.com/chap1.html

Chaque «colonne» de neurones est appelée une couche. La première couche contient les entrées ; la dernière couche est la couche de sortie, elle va nous donner les résultats ; les couches du milieu sont appelées des couches cachées.

Bien sûr, on peut modifier le nombre de neurones de chaque couche et le nombre de couches cachées en fonction du problème qu'on tente de résoudre.

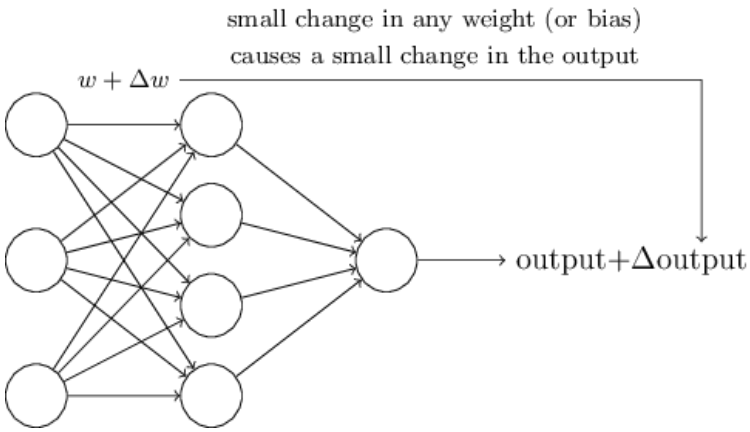


Figure 7: Un petit changement de poids doit engendrer un petit changement de sortie.
http://neuralnetworksanddeeplearning.com/chap1.html

On voudrait qu'une petite modification du poids engendre une petite modification de la sortie. Ce n'est pas le cas si l'on passe directement de zéro à un. C'est pourquoi on va arrêter de regarder si la somme pondérée de toutes les entrées est supérieure ou égale à zéro, mais on va donner le résultat de cette somme à une fonction. Cette fonction est appelée fonction d'activation et devra retourner des résultats entre zéro et un [13].

Dans un problème de classification, comme c'est le cas par-exemple lorsque l'on cherche à savoir si une image représente un champ de café, de l'eau, ou autres, on va fournir des données en entrée en spécifiant à quelles classes correspondent les données et l'on veut

que le réseau de neurones apprenne les caractéristiques de celles-ci de telle façon qu'en rencontrant une donnée qu'il ne connaît pas encore il sache la classer correctement. On va donc entraîner notre réseau de neurones avec une partie des données puis on va valider qu'il arrive bien à classer le reste des données qu'il n'a pas encore rencontrées.

Lorsqu'on entraîne un réseau de neurones, le réseau de neurones commence avec des poids aléatoires, il va donc probablement se tromper et classer faussement la donnée. En comparant la sortie obtenue et celle attendue, on va pouvoir réaliser à quel point le réseau de neurones s'est trompé. Ensuite on va petit à petit modifier légèrement les poids et ainsi apprendre de quelle façon changer les poids pour minimiser l'erreur.

En rajoutant des couches cachées et des neurones dans les couches cachées on va pouvoir apprendre des caractéristiques toujours plus complexes, mais on risque aussi d'appendre tellement bien les caractéristiques de nos données, qu'en donnant à classer une donnée inconnue au réseau de neurones, il ne la classe pas correctement car elle ne correspond pas exactement à une donnée déjà rencontrée. Cela s'appelle le sur-ajustement (*overfitting*).

5.3 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs sont prévus pour être utilisés avec des images. Contrairement à un réseau de neurones «classique», les réseaux de neurones convolutifs ont des neurones disposés en plusieurs dimensions : la hauteur, la largeur et la profondeur. La profondeur représentant les couleurs rouge, vert et bleu, dans une image avec des couleurs naturelles, mais dans le cas d'image multispectrale on peut avoir une profondeur d'image différente et des canaux différents.

Il y a trois types de couches utilisées dans les réseaux de neurones convolutifs : les couches de convolution, de sous-échantillonnage (*pooling*) et les couches entièrement connectées (f*ully Connected*) [14].

5.3.1 Couche de convolution

La couche de convolution est, comme son nom l'indique une couche qui va réaliser une convolution sur l'image. La convolution est une technique de traitement d'image qui existe aussi en dehors des réseaux de neurones.

Une convolution utilise un filtre qui va parcourir toute l'image et effectuer un produit matriciel à chaque position où se déplace le filtre. En traitement d'image, ce filtre peut par-exemple permettre de flouter une image, ou détecter les bords dans une image [15].

Dans une couche de convolution, ces filtres seront appris par le réseau de neurones et permettront de détecter des caractéristiques comme des bords, des directions, ou des taches de couleurs [14]. Comme on peut le voir sur l'image ci-dessous, des caractéristiques complexes peuvent être reconnues. Plus il y a de couches de convolutions, plus des caractéristiques complexes peuvent être trouvées.

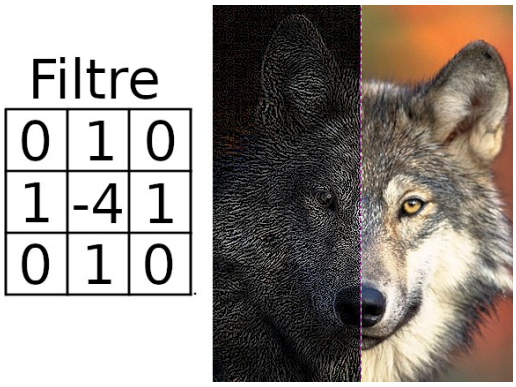


Figure 8: Exemple d'application d'un filtre de convolution à gauche de l'image. Réalisé sur Gimp à partir de https://commons.wikimedia.org/wiki/File:Front_view_of_a_resting_Canis_lupus_ssp.jpg

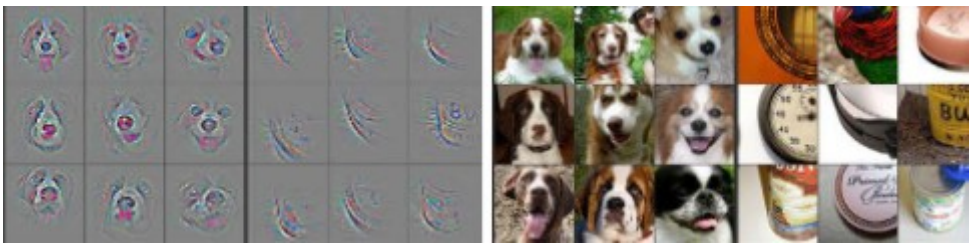


Figure 9: Exemple de caractéristiques reconnues par des filtres à la 4ème couche d'un réseau convolutif. Extrait de "Visualizing and Understanding Convolutional Network", https://arxiv.org/pdf/1311.2901.pdf

On peut ajouter des zéros aux bords de l'image de telle manière quelle garde sa taille, la convolution réduisant la grandeur de l'image dans le cas contraire [16].

5.3.2 Couche de sous-échantillonnage

Les couches de sous-échantillonnage réduisent la taille des images en parcourant l'image en prenant une zone de généralement 2x2 [14] et en gardant chaque fois une valeur par zone. Il y a plusieurs façon de faire : le plus couramment on prend la valeur maximale de cette zone ou la moyenne, mais il y a bien d'autres façons imaginables de faire. Le tout est que ce sous-échantillonnage garde les données importantes et ne prennent pas compte des détails inutiles [17].

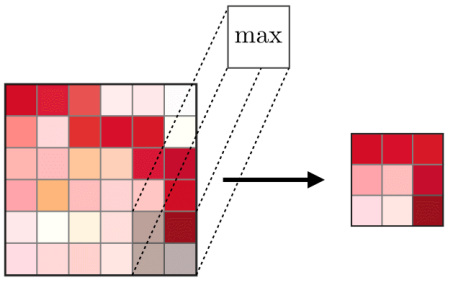


Figure 10: Exemple de sous-échantillonnage prenant la valeur maximale d'un filtre de 2x2px, https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

Ce sous-échantillonnage a deux utilités : premièrement de réduire le nombre de paramètres, ce qui permet de rendre les calculs moins coûteux, et secondement de réduire le sur-ajustement aux données d'entraînement. En effet, en réduisant ainsi les images, le modèle dépend moins de l'emplacement précis où se trouvent les caractéristiques tout en conservant la région où se trouvent les caractéristiques.

5.3.3 Couche entièrement connectée

Avant la couche entièrement connectée, on passe d'un format multidimensionnel à une seule dimension. La couche entièrement connectée pourra alors agir comme un simple réseau de neurones et tentera de réduire les erreurs de classifications [18].

5.3.4 Architecture

L'architecture d'un réseau de neurones convolutifs réside dans le choix des couches et leurs ordres et dans les paramètres choisis pour celles-ci.

Voici un exemple d'architecture de réseau de neurones convolutifs :

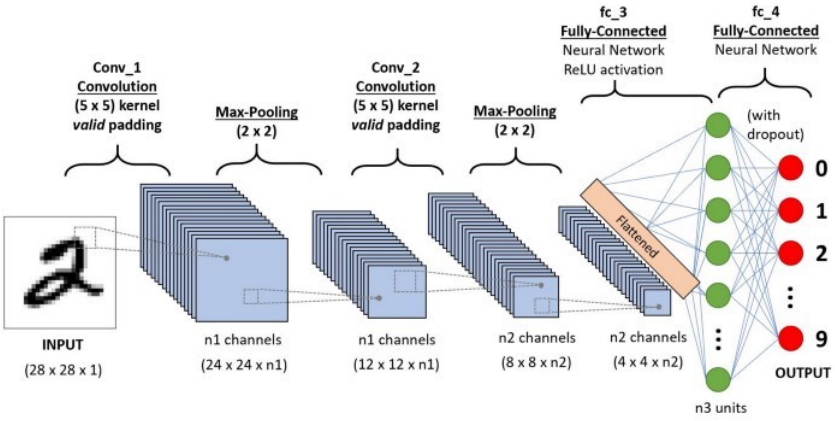


Figure 11: Exemple d'une architecture de réseau de neurones convolutif pour reconnaître des chiffres, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Habituellement, l'architecture d'un réseau de neurones convolutifs est formée d'une série de couches de convolutions (utilisant la fonction d'activation *ReLU*), d'éventuellement une couche de sous-échantillonnage et ainsi de suite jusqu'à obtenir une image de petite taille, puis enfin d'une ou plusieurs couches entièrement connectées [14].

Il faut aussi préférer plusieurs couches de convolutions avec des petits filtres qu'une seule couche de convolution avec un grand filtre. D'ordinaire, la taille du filtre ne devrait pas dépasser 5x5 pixels [14].

Pour ce qui est de la couche de sous-échantillonnage, il est courant de parcourir l'image avec un filtre d'une taille de 2x2 pixels [14,19].

5.4 Conclusion

En comprenant maintenant mieux le fonctionnement et le but de ces réseaux de neurones convolutifs, on voit alors émerger les raisons de leur utilisation dans le cadre de cette problématique. En effet, ces réseaux de neurones convolutifs permettent de dégager et combiner des caractéristiques qui auraient été bien difficiles à trouver en étudiant simplement les images que nous avons à disposition. De plus, ces réseaux de neurones convolutifs adaptent les réseaux de neurones artificiels «classiques» pour rendre leur utilisation possible avec des images. Ils sont donc conçus, dès l'origine, pour être utilisés avec des images et conservent alors les notions d'espaces, de directions et de couleurs (ou bandes de fréquences dans notre cas) sans les considérer indépendamment.

6 Technologies utilisées

6.1 Ee

Ee est la librairie de l'API python de Google Earth Engine [20]. Elle permet donc d'interagir avec GEE.

6.2 Keras

Keras est une librairie python qui fonctionne par-dessus TensorFlow [21]. On l'utilise pour créer et entraîner les réseaux de neurones convolutifs.

6.3 Rasterio

Rasterio permet de traiter des fichiers de rasters [22]. Ici elle est utilisée pour lire et fusionner des fichiers de rasters, et extraire les valeurs d'une zone.

6.4 Shapely

Shapely est une librairie permettant de travailler avec des objets géométriques, comme définis par librairie GEOS [23], et faire des analyses dans l'espace [24,25].

6.5 Scikit-learn (Sklearn)

Scikit-learn est une librairie pour le machine learning [26]. On l'utilise ici pour calculer le poids de chaque classe (par-rapport au nombre de points géoréférencés de chaque classe) et pour faire de la normalisation.

6.6 Numpy

Numpy est une librairie qui nous permet de faire des calculs et des opérations sur des matrices [27].

6.7 Pandas

Pandas et un outil permettant de faire de l'analyse et de la manipulation de données [28].

6.8 Geopandas

Geopandas se base sur pandas et étend les structure de données que pandas propose pour pouvoir traiter des données géospatiales [29].

6.9 Matplotlib

Cette librairie permet de créer différentes sortes de graphiques et de visualisations de données [30].

6.10 Seaborn

Seaborn est une librairie de visualisation de donnée basée sur Matplotlib, on l'utilise ici pour afficher les boîtes à moustaches [31].

6.11 Math

Math met à disposition des fonctions mathématiques [32].

6.12 Time

Time est une librairie pour traiter des données temporelles et propose des fonctions qui ont rapport au temps [33]. On l'utilise dans le script pour télécharger des données de GEE pour ne pas demander l'état de la tâche en cours en permanence, mais attendre un moment avant de redemander.

6.13 Glob

Glob permet de récupérer tous les chemins vers des fichiers qui correspondent à une règle [34].

6.14 Geojson

Cette librairie permet d'encoder et de décoder des fichiers GeoJSON et propose des classes pour chaque objet de la spécification GeoJSON [35,36].

6.15 Spacv

Spacv est une librairie permettant de faire de la validation croisée en créant des ensembles d'entraînement qui se situent à distance de l'ensemble de validation pour s'assurer que le modèle généralise bien [37].

6.16 Jupyter

Jupyter permet de créer des «notebooks», des fichiers permettant de créer des scripts interactifs avec les résultats, des visualisations et du texte accompagnant le code. Ils sont composés de cellules qui peuvent être exécutées indépendamment les unes des autres et dont les variables sont partagées avec les autres cellules [38].

7 Traitement des données

7.1 Obtention des données

Les données du satellite Landsat 8 sont récupérées à partir de Google Earth Engine [39].

Pour ce faire, un script python utilisant l'API de GEE a été réalisé.

Ce script «downloadMap.py» va exporter le résultat d'une requête vers GEE en un fichier de raster *GeoTiff* sur Google Drive.

Une tâche est créée et l'état de la tâche est suivi ; ainsi, on peut savoir quand l'export est fini sans passer par l'interface en ligne de GEE.

Il faut préciser quelle région du monde nous intéresse, à quelle date et à partir de quelle collection de données [40]. Cela va créer une collection d'images qui va ensuite être réduite en une seule image médiane.

Pour savoir où se situent le café, le riz, l'eau et bien d'autres, le CIAT nous a fourni des points géoréférencés avec les labels correspondants. D'après les informations que nous avons, ces labels devraient avoir été créés entre 2016 et 2018.

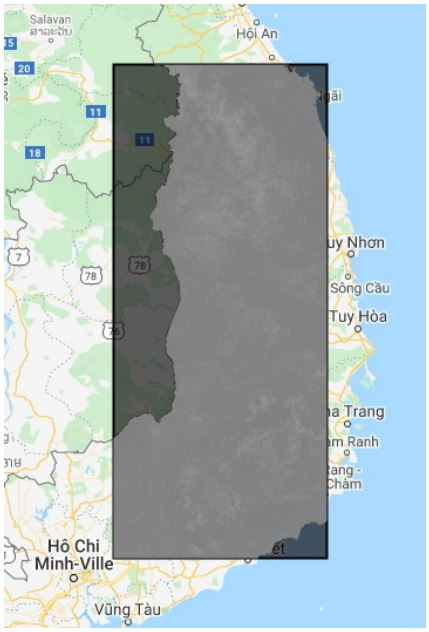


Figure 12: Visualisation de la zone récupérée, on récupère des images dans les limites du Vietnam qui se situent dans le rectangle représenté en gris

Figure 13: Visualisation avec QGIS des points géoréférencé par-dessus une carte du Vietnam

Sur l'image ci-dessus, vous pouvez voir l'entièreté des points géoréférencés qui nous ont été fournis. Les petites croix roses représentent les champs de café.

On peut voir qu'il y a des points en dehors du Vietnam. Nous n'utilisons pas ces points là.

7.2 Préparation des données

L'image de la carte du Vietnam que GEE nous donne est séparée en plusieurs fichiers de raster. Pour que le traitement des données soit plus facile, on fusionne ces fichiers en un seul.

Ensuite, on va pouvoir utiliser cette image pour récupérer les données concernant chacun des labels. On récupère les coordonnées de chaque point référencé des labels contenus dans des fichiers *shapefile* et on les classe par label.

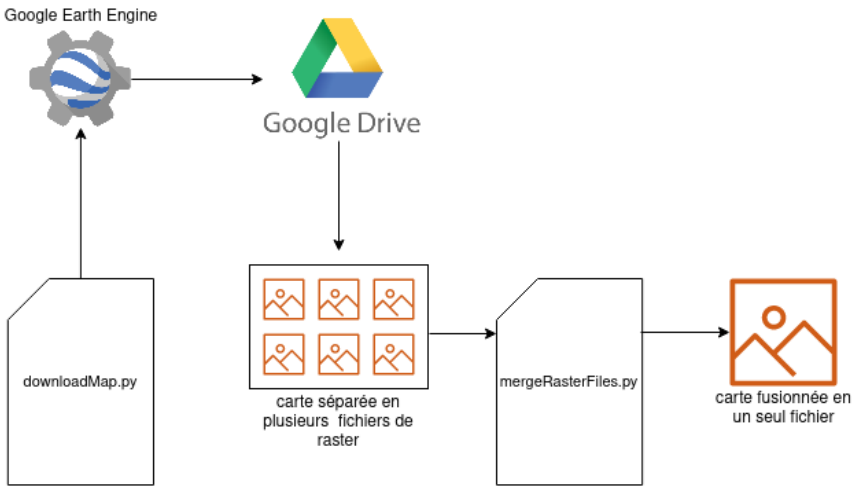


Figure 14: Schéma de la récupération et fusion des données

Ces coordonnées de labels vont ensuite permettre de savoir où récupérer chaque image des labels et à quel label chacune des coordonnées correspond.

Il est à noter que ces coordonnées sont sous la forme de latitude et longitude dans une projection EPSG:4326. On fait donc la conversion de la latitude et longitude en une ligne et colonne du pixel correspondant dans l'image.

Ensuite, on découpe des images autour de ces points géoréférencés, par-exemple de 9x9 pixels ou 11x11px.

Enfin, ces images pourront être utilisées pour l'entraînement et la validation du réseau de neurones ou bien pour analyser les valeurs des canaux par rapport aux labels.

On ne garde pas les images qui contiennent des valeurs non-définies. Cela peut être des valeurs de labels en dehors de l'image ou alors des endroits cachés par des nuages.

En effet, un problème qui se pose est la présence fréquente de nuages sur les images satellites que nous obtenons.

La visualisation à droite le montre bien. Si l'on prend des images satellites dans une période nuageuse et que l'on n'en prend pas assez, alors même en comblant certains endroits nuageux en utilisant les images avec le moins de nuages de chaque région dans cette période, il y aura des trous dans l'image finale.

Figure 15: Exemple de carte médiane nuageuse de juillet à octobre 2015

Pour remédier à cela, il faut soit choisir une période moins nuageuse, soit prendre une plus grande quantité d'images de cette période en prenant plusieurs années différentes. Suivant les années, la couverture nuageuse peut aussi être plus basse pour certaines régions. C'est pourquoi il peut être intéressant de d'abord visualiser les images sur GEE et ensuite choisir la meilleure image.

8 Analyse exploratoire des données

Une analyse exploratoire des données a été effectuée. Grâce à celle-ci on arrive mieux à comprendre l'évolution des valeurs du café à travers le temps et comment les valeurs du café se comparent aux autres labels.

Il est à noter que pour réaliser cette analyse exploratoire, la valeur médiane de chaque image des labels a été prise. Ainsi, sur une image de 9x9px seule une valeur médiane est extraite. Un autre facteur qui pourrait fausser les résultats à prendre en compte est que les images autour des points géoréférencés peuvent contenir d'autres éléments que le label seul. En effet, il peut y avoir plusieurs plantations différentes les unes à côté des autres, un cours d'eau etc.

Au Vietnam, il y a deux saisons : la saison sèche et la saison humide. La saison sèche commence en novembre et fini en avril. La saison humide, quant à elle, commence en mai et fini en octobre [40].

8.1 Comparaison du café à d'autres labels

Il semble plus facile de distinguer le café des autres labels pendant la saison sèche. En effet, en regardant les différentes boîtes à moustaches, on remarque qu'il est plus facile de différencier les labels avec les canaux *coastal aerosol*, *blue*, *green*, *red* et *nir* en saison sèche qu'en saison humide.

On remarque aussi que le café prend des valeurs proches du poivre, du thé et de l'eau. Toutefois, le café s'écarte un peu du poivre avec le canal *red* et du thé et de l'eau avec le canal *nir,* même s'il reste encore un assez grand chevauchement entre leurs données.

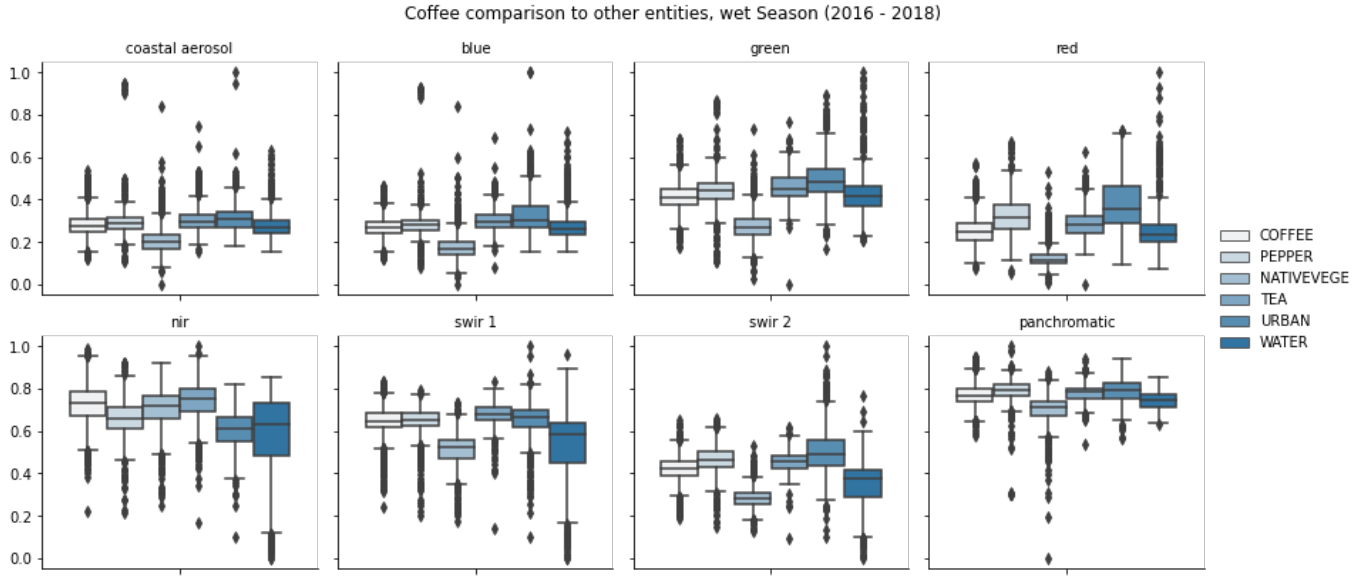


Figure 17: Comparaison du café au poivre, à la végétation naturelle, au thé, aux zones urbaines et à l'eau en saison humide

8.2 Comparaison du café par saison

Si l'on étudie la variabilité des données du café à travers la saison sèche et la saison humide, on observe que la variabilité est tantôt meilleure en saison sèche et tantôt meilleure en saison humide.

Sur les années 2014 à 2016, la saison humide à moins de variabilité à part pour le canal *nir*, alors que pendant les années 2016 à 2018, la variabilité est plus faible en saison sèche pour tous les canaux sauf *red*, *swir1*, *swir2*, et *panchromatic*.

Pour ce qui est des valeurs médianes, on remarque une nette augmentation du canal *nir*, de ~0.43 à ~0.65, en saison humide. Le canal *green* augmente lui aussi, mais en 2016-2018 plus fortement, passant de ~0.35 à ~0.45 pour les années 2014 à 2016 et augmentant jusqu'à ~0.51 en 2016 à 2018.

Tous les canaux ont tendance à légèrement augmenter à part *red*, *panchromatic* et *swir2* qui baissent légèrement en saison humide.

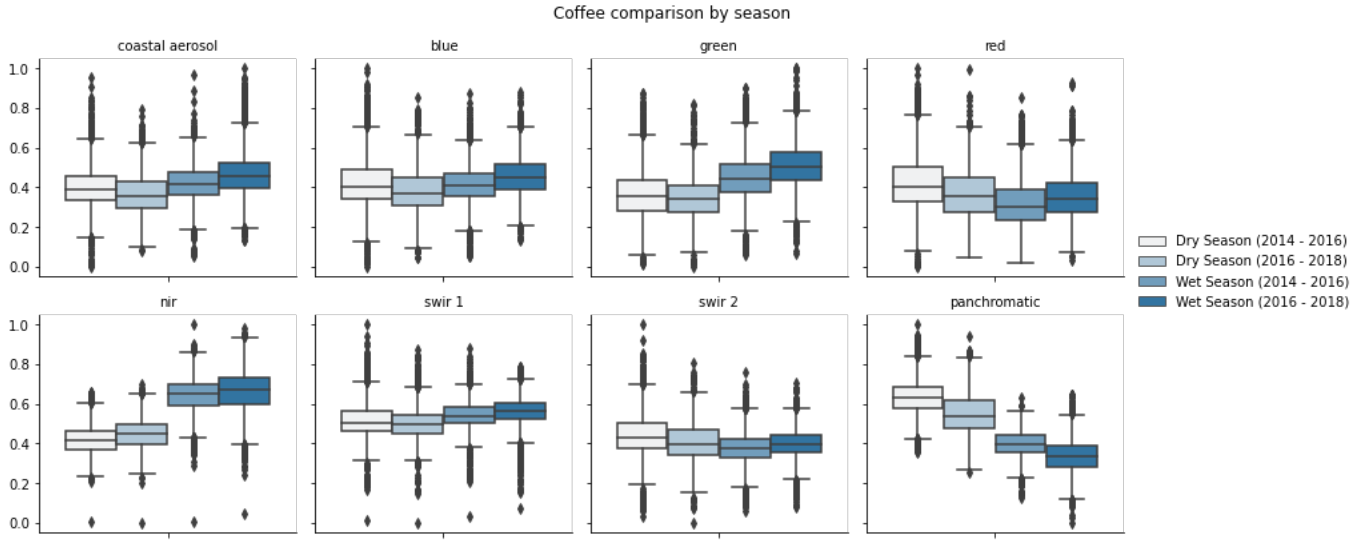


Figure 18: Comparaison des valeurs du café par saison en prenant les années 2014 à 2016 et 2016 à 2018

8.3 Comparaison du café tous les deux mois

Ici, on compare les valeurs des canaux du café deux mois par deux mois en prenant une médiane de toutes les données récoltées entre 2014 à 2020.

On remarque alors quelles sont les périodes qui ont le moins de variabilité et l'évolution des canaux à travers les mois.

8.3.1 Contexte

Au Vietnam, la récolte du café commence en octobre [41] et prend fin de décembre à janvier [42].

Après la récolte, les arbres à café sont taillés pour laisser passer plus de lumière ; et c'est de janvier à avril où le caféier est fortement arrosé [42].

En février se déroule le Têt, la fête du nouvel an lunaire, et il est possible que la café soit volontairement abondamment arrosé pour fleurir pendant le Têt.

8.3.2 Évolution

C'est à partir de mai que le canal *nir* augmente significativement, la médiane augmentant de 0.417 à 0.653.

De mars à avril, le canal rouge est au plus haut. En effet la médiane passe à 0.404 alors qu'elle était à 0.324 et qu'elle retombe ensuite à 0.333. On observe aussi que *panchromatic* augmente en mars et avril.

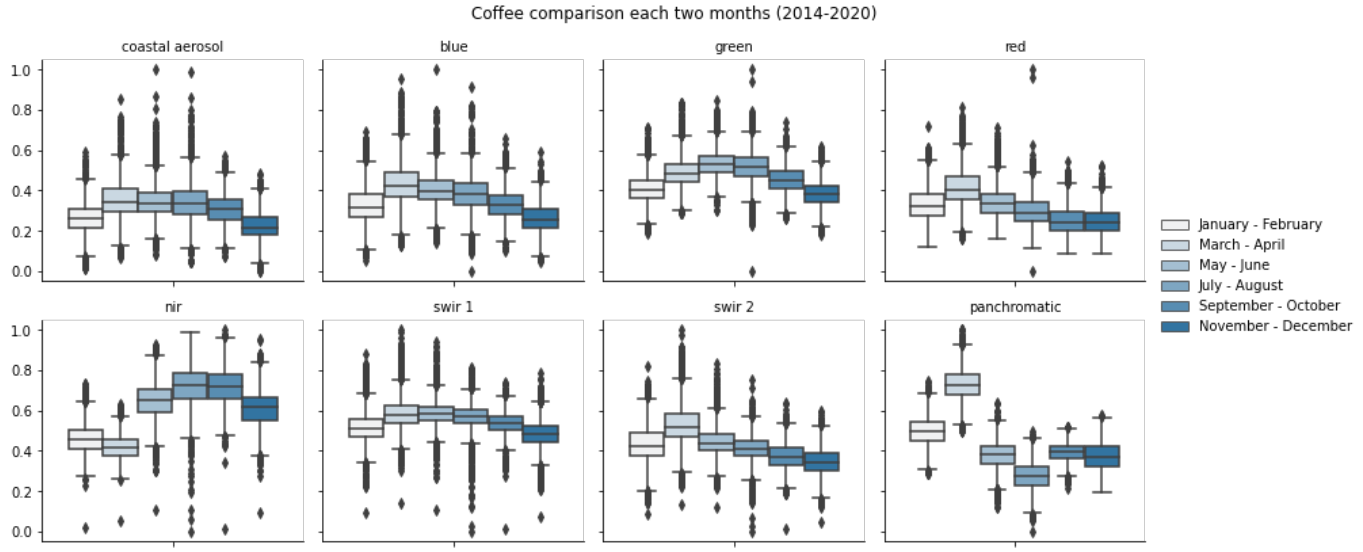


Figure 19: Comparaison du café deux mois par deux mois avec les données de 2014 à 2020

Le canal *green*, quant à lui, est à son plus haut en mai et juin, et en juillet et août.Variabilité

Voici les mois qui ont le moins de variabilité pour certains canaux :

- Septembre à octobre : coastal aerosol, blue, green, swir1, swir2, panchromatic
- Novembre à décembre : red
- Mars à avril : *nir*

Septembre à octobre est la période avec la plus petite variabilité pour toutes les bandes de fréquences sauf *red* et *nir*.

8.4 Comparaison du café par région

Les deux régions étudiées ici sont les régions où l'on a des points de café géoréférencés. En l'occurence la région «Central Highlands» et la région «South Central Coast».

Il y a moins de variabilité dans la région *«Central Highland»* , même si la région *«South Central Coast»* à une plus petite variabilité pour les bandes de fréquences *coastal aerosol* et *panchromatic*.

Dans ces deux régions, le café se dinstingue principalement par les canaux *panchromatic*, *red*, *coastal aeorosol* et *blue*. Les différences de médianes étant de 0.092 pour *panchromatic*, 0.086 pour *red*, 0.081 pour *coastal aerosol* et 0.07 pour *blue*.

Pour le graphique suivant, la région «Central Highlands» est appelée «Highland Vietnam» et la région «*South Central Coast» est appelée «Southern Vietnam».*

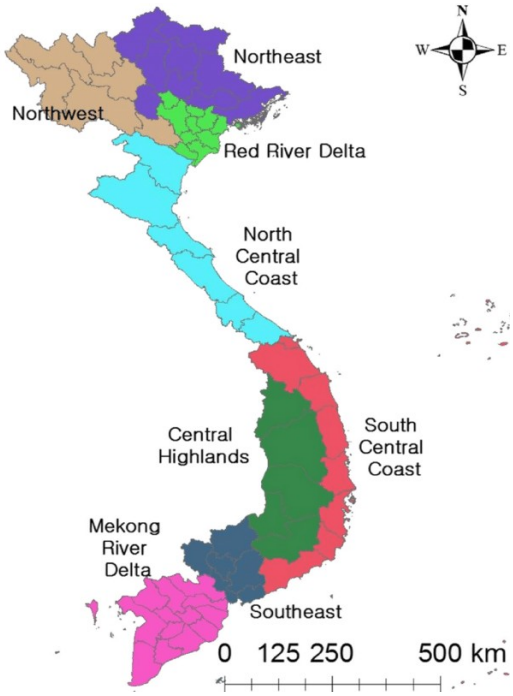


Figure 20: Carte avec les régions du Vietnam, https://www.researchgate.net/figure/List-of-regions-in-Vietnam_fig1_317913844

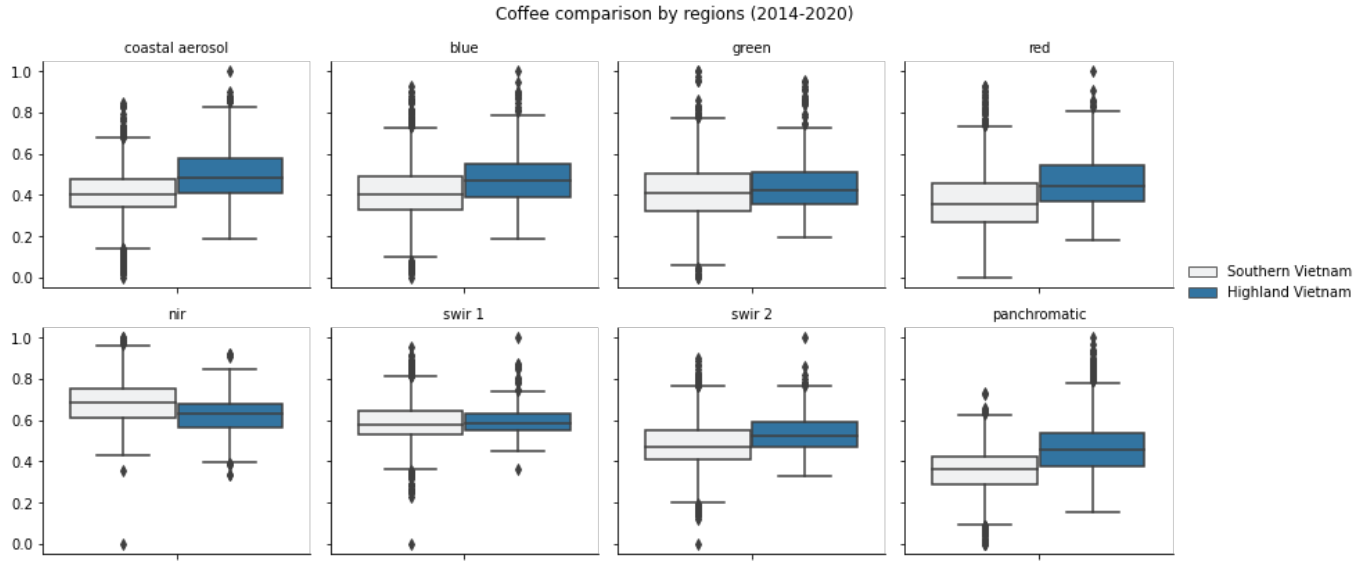


Figure 21: Comparaison du café par régions avec les valeurs du café de 2014 à 2020

8.5 Comparaison par district

Voici à quoi ressemble les valeurs des différentes bandes de fréquences du café par districts. Seuls les districts dans lesquels on avait des points de café ont été retenu.

Ces districts sont les suivants: Gia Lai, Đắk Lắk, Đắk Nông et Lâm Đồng.

Les frontières des districts ont été récupérées à partir de données ouvertes [43].

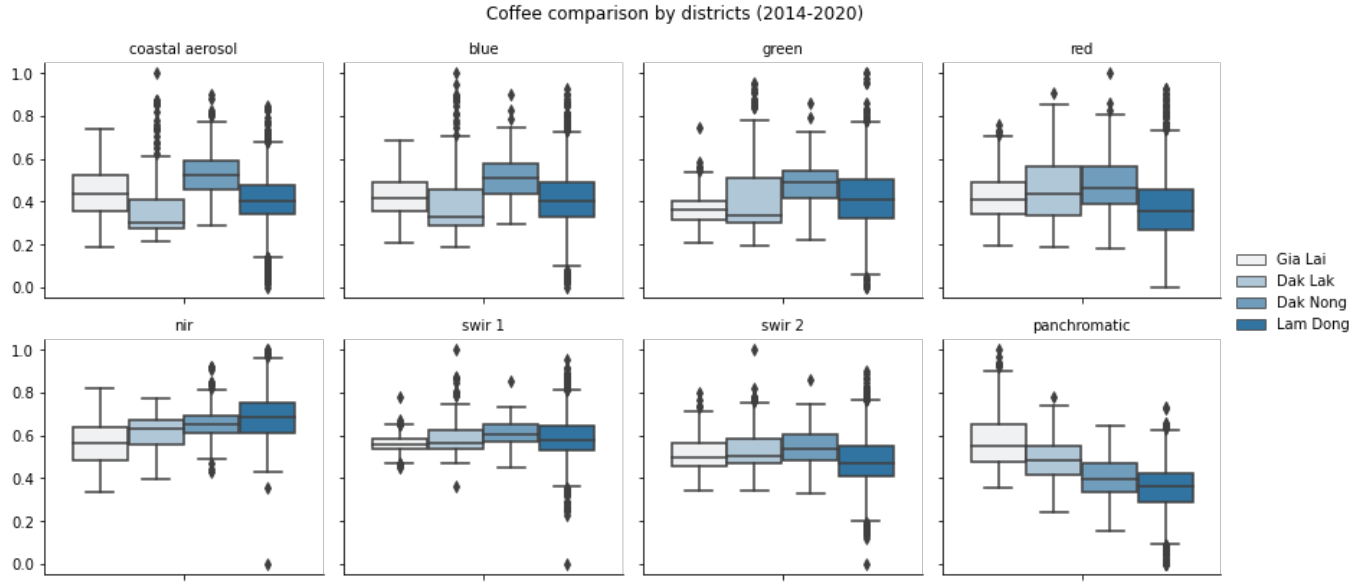


Figure 22: Comparaison des valeurs du café par district

8.6 Comparaison du café par types de sols

Voici à quoi ressemble les valeurs des différentes bandes de fréquences du café par types de sols. Seuls les types de sols dans lesquels on avait des points de café ont été retenu.

Ces types de sols sont les suivants : *Acric Ferrasols*, *Ferric Acrisols*, *Orthic Acrisols*, *Rhodic Ferrasols* et *Orthic Ferrasols*.

Les types de sols ont été récupérés à partir de données ouvertes [44].

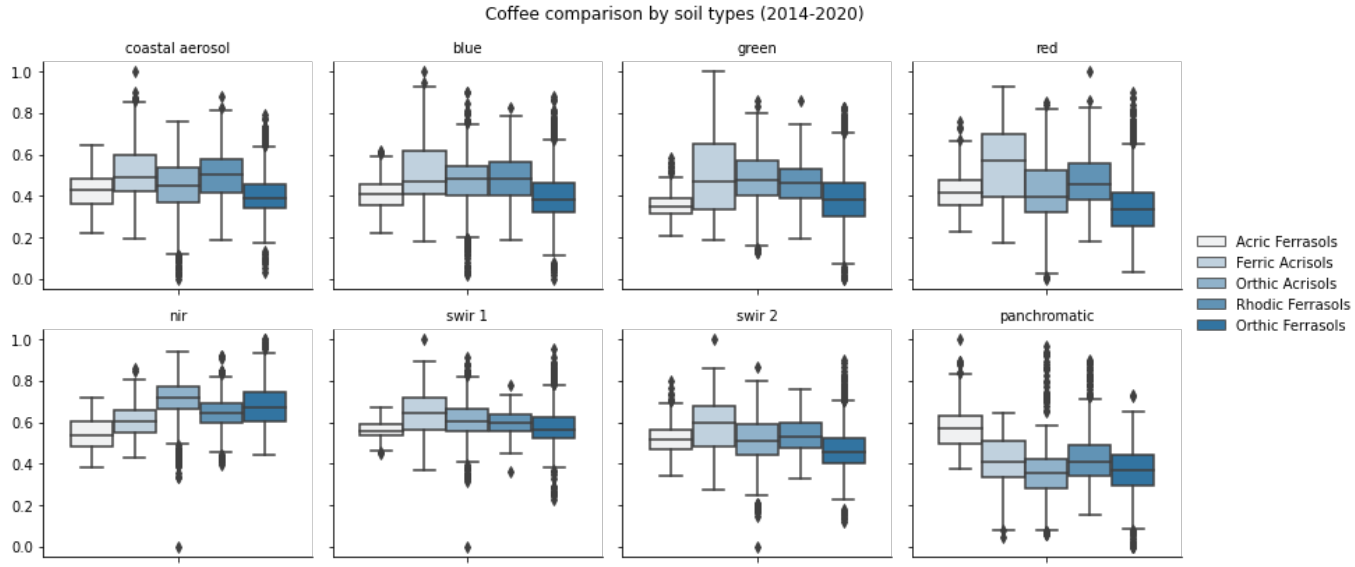


Figure 23: Comparaison des valeurs du café par types de sols

8.7 Proportion moyenne de nuage

Cette proportion a été calculée à partir du script de téléchargement avec :

```
merged_image_collections.aggregate_stats('CLOUD_COVER').getInfo()
```

merged_image_collections est la collection d'image, de la région du Vietnam qui nous intéresse, tous les deux mois de 2014 à 2020.

Les résultats sont les suivants :

Mois	Pourcentage de couverture nuageuse
Janvier-février	28.71%
Mars-avril	**23.30%**
Mai-juin	37.40%
Juillet-août	**45.11%**
Septembre-octobre	39.66%
Novembre à décembre	38.90%

9 Entraînement des modèles

9.1 Validation croisée k-blocs

Pour pouvoir rendre les résultats plus fiables, on utilise la validation croisée k-blocs (k-fold cross validation).

Cela se présente ainsi : on va répartir l'ensemble des données en k sous-ensembles. On va entraîner sur chaque sous-ensemble l'un après l'autre en prenant les autres sous-ensembles comme données de validation.

Le nombre k de blocs est couramment défini à 10, mais 3 et 5 sont aussi des valeurs fréquentes [45]. Si l'on utilise 5 blocs, on va chaque fois entraîner sur 80 % des données et valider avec 20 % des données.

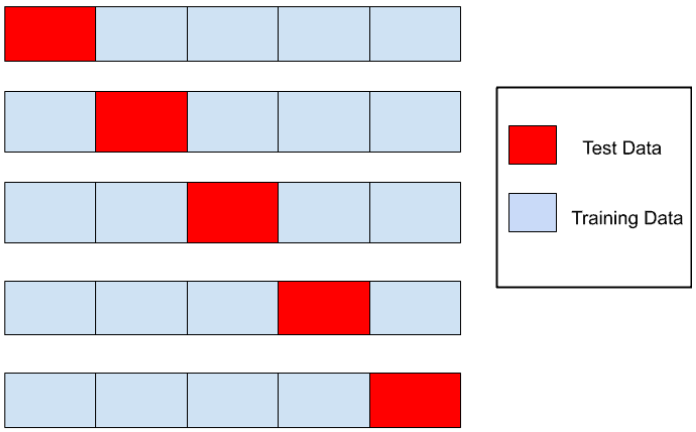


Figure 24: Représentation de la répartition des sous-ensembles avec la validation croisée, https://www.mltut.com/k-fold-cross-validation-in-machine-learning-how-does-k-fold-work/

Avant de faire ce découpage en sous-ensemble, on va arranger les données dans un ordre aléatoire. Imaginons que nous avons mis toutes les données dans l'ordre dans lesquels nous avons créé l'ensemble des données, alors peut-être que l'on aura d'abord toutes les images de café, ensuite toutes les images de thé etc. On voit bien que cela peut représenter un problème.

Mais cet ordre aléatoire peut néanmoins influencer les résultats. C'est pourquoi on va en plus faire cette validation croisée plusieurs fois avec chaque fois un ordre des données différent.

9.2 Augmentation des données

Il est important d'avoir une grande quantité de données pour pouvoir extraire les caractéristiques les plus représentatives de chaque classe. Malheureusement on n'a parfois pas une grande quantité de données et celles-ci sont dures à obtenir. Pour palier à cela, on va «augmenter les données».

Cela consiste à prendre les données que l'on a et en générer des nouvelles à partir de celles-ci, par-exemple en effectuant des rotations sur l'images, en inversant le sens des images, en décalant l'image ou encore en tordant l'image [46].

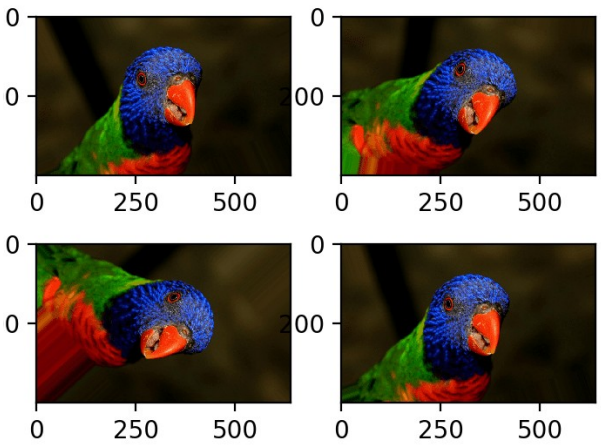


Figure 25: Exemple de rotations aléatoire d'image avec l'augmentation des données. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Imaginons un ensemble de donnée avec des animaux. Si on a par exemple que des images de chiens qui regardent vers la gauche, alors lorsque l'on rencontrera une image de chien regardant vers la droite, on risque de ne pas le reconnaître comme un chien. Augmenter les données évite ce type de problème.

9.3 Arrêt prématuré

L'arrêt prématuré (early stopping) permet d'arrêter l'entraînement du réseau de neurones au moment où les performance du réseau lors de la validation ne s'améliorent plus.

On peut peut se baser sur différentes mesures comme l'erreur ou la précision (accuracy). De plus, il est aussi possible de spécifier la patience c-à-d le nombre d'epochs sans amélioration avant de stopper l'entraînement.

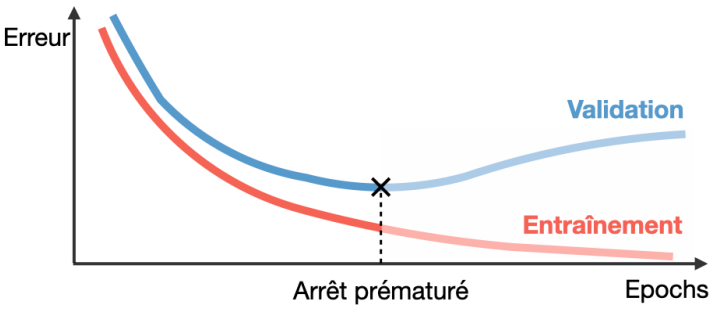


Figure 26: Démonstration de l'arrêt prématuré avec un graphe de l'erreur, https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-petites-astuces-apprentissage-profond#regularization

Cette technique permet donc de s'arrêter au moment où le modèle a les meilleures performances et ne pas faire une grande quantité d'epochs inutiles.

9.4 Équilibrage des classes

Dans l'ensemble des données que nous avons, il n'y a pas la même quantité de chaque classe. Cela pose problème, car cela déséquilibre les résultats en faveur de la classe sur-représentée. Il y a trois moyens de remédier à cela : enlever des données de telle manière qu'il y ait autant de données dans chaque classe, ajouter plusieurs fois des données des classes sous-représentées pour arriver à un même nombre de données pour chaque label ou ajouter un poids pour chaque classe en fonction du nombre de données par classe [47].

9.5 Premiers réseaux et améliorations

Les premiers réseaux de neurones convolutifs qui ont été réalisés comportaient quelques erreurs de débutant. Des améliorations ont pu alors être discutées lors de réunions à propos du travail de bachelor.

Par exemple, les couches de convolutions utilisaient une taille de filtre de 4x4 pixels ou 5x5 pixels alors que l'image faisait 9x9 pixels. Ce n'est pas un problème de divisibilité de l'image par la taille du filtre, vu que l'on rajoute des zéros sur les bords, mais cette taille de filtre était trop grande pour une si petite image. Pour améliorer cela, la taille des filtres de convolutions ont été maintenant définie à 2x2 ou 3x3 pixels.

Comme il a déjà été précisé dans l'introduction aux réseaux de neurones convolutifs, il vaut mieux avoir plus de couches de convolutions avec de petits filtres qu'une couche de convolutions avec un grand filtre.

Une autre amélioration qui a pu être trouvée est l'utilisation de la fonction d'activation softmax au lieu de sigmoid pour la couche entièrement connectée.

En effet, la fonction d'activation softmax garantit qu'il n'y ait qu'une seule classe attribuée comme résultat. La somme de l'activation de chaque neurone de sortie est alors de un et permet d'être interprété comme une probabilité. C'est le neurone qui a la plus haute probabilité qui est choisi [48].

Autrement, les couches entièrement connectées avaient 256 neurones, ce qui est trop. Le nombre de neurones a donc été réduit dans la couche entièrement connectée.

9.6 Comparaison de combinaisons de bandes de fréquences

Pour savoir quelles combinaisons de bandes de fréquences donnaient les meilleurs résultats, une série de combinaisons différentes a été testée.

Pour choisir quelles combinaisons de canaux tester, il a d'abord été testé plusieurs combinaisons de trois canaux fréquemment utilisées [49,50].

Ensuite, des combinaisons proches de celles donnant les meilleurs résultats ont été essayées.

Pour pouvoir comparer les combinaisons de canaux, les paramètres utilisés sont les mêmes entre chaque test et l'architecture du réseau de neurones convolutifs est aussi la même.

Ces paramètres utilisés sont les suivants :

Nombre d'epochs	500
Nombre de tests	4
Labels utilisés	Café, végétation naturelle, urbain, eau, poivre et thé
Nombre de pixels autour du label	4 (c-à-d une image de 9x9 pixels)

Et l'architecture :

```
model = Sequential([
    Rescaling(1./2**16, input_shape=(image_width, image_height, image_depth)),
    Conv2D(filters=8, kernel_size=(3, 3), padding="same", activation="relu"),
    Conv2D(filters=8, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=16, kernel_size=(2, 2), padding="same", activation="relu"),
    Conv2D(filters=16, kernel_size=(2, 2), padding="same", activation="relu"),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(name='flat'),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(nb_outputs, activation='softmax'),
])
```

Voici les résultats :

Combinaison	Erreur moyenne	Précision moyenne	Précision café	Rappel café	F-score café
RED, GREEN, BLUE	1.299	42.78%	61.42%	25.48%	36.02%
RED, GREEN, COASTAL_AEROSOL	1.235	45.09%	67.38%	24.50%	35.93%
SWIR1, NIR, COASTAL_AEROSOL	1.041	55.71%	79.35%	34.80%	48.38%
SWIR1, NIR, BLUE	1.030	57.43%	79.07%	37.21%	50.60%
SWIR1, SWIR2, RED	0.968	58.79%	79.56%	39.55%	52.84%
SWIR1, SWIR2, BLUE	1.038	54.75%	81.25%	33.62%	47.56%
NIR, RED, GREEN	1.061	56.72%	75.96%	44.22%	55.90%

NIR, SWIR2, COASTAL_AEROSOL	1.031	56.96%	80.64%	39.31%	52.85%
NIR, RED, SWIR1	0.960	61.23%	81.47%	**46.27%**	**59.02%**
NIR, RED, SWIR2	0.990	59.14%	80.99%	42.80%	56.01%
NIR, RED, BLUE, SWIR1	0.964	61.50%	82.29%	45.30%	58.43%
NIR, RED, SWIR1, SWIR2	**0.943**	**61.87%**	**81.58%**	45.95%	58.78%
NIR, RED, GREEN, SWIR1	1.005	58.40%	81.77%	41.00%	54.62%

La meilleure combinaison de test s'est révélée être NIR, RED, SWIR1, SWIR2 si on regarde tous les labels, mais pour ce qui est du café, la meilleure combinaison est NIR, RED et SWIR1.

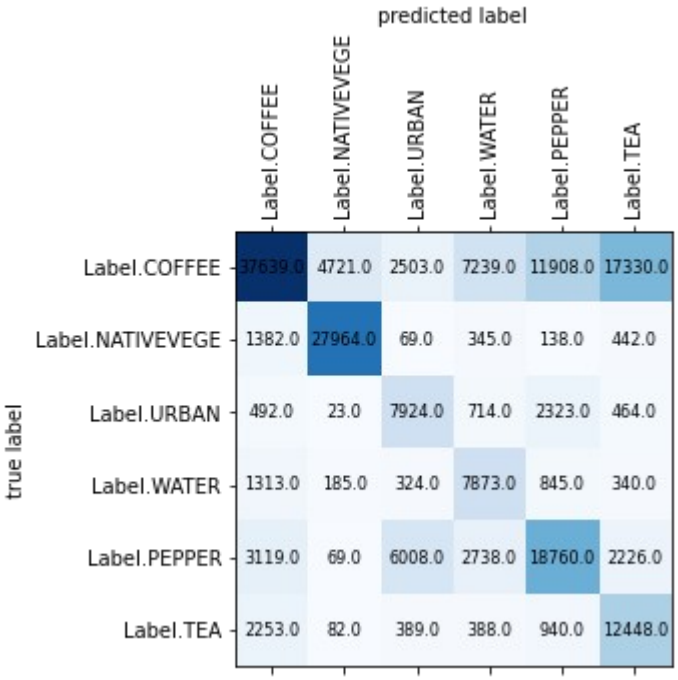


Figure 28: Matrice de confusion avec la combinaison NIR, RED, SWIR1

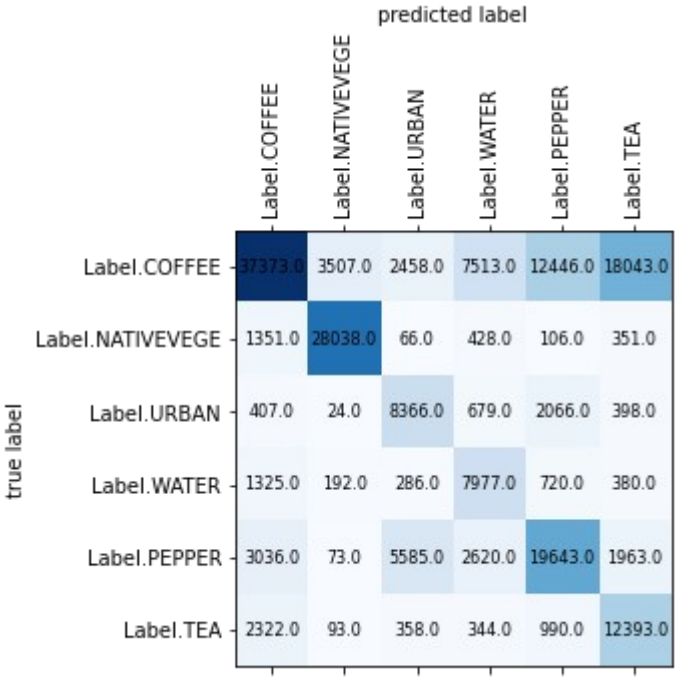


Figure 27: Matrice de confusion avec la combinaison NIR, RED, SWIR1, SWIR2

Si l'on regarde les matrices de confusion ci-dessus, on remarque que le café est confondu en majorité avec le thé et le poivre.

Le test s'est déroulé sur seulement 500 epochs. En observant le graphe de l'erreur par-rapport à l'entraînement et la validation, on remarque que l'erreur pourrait encore être optimisée et que de meilleurs résultats pourraient être obtenus en entraînant le réseau de neurones sur plus d'epochs.

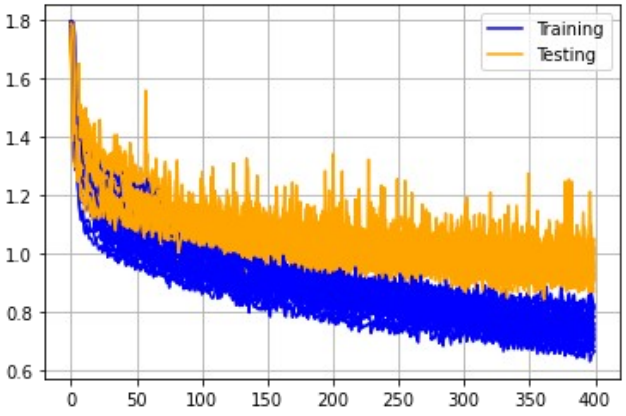


Figure 29: graphe de l'erreur avec la combinaison NIR, RED, SWIR1

Il serait aussi intéressant de continuer à tester des combinaisons différentes pour voir si on arrive à éliminer certaines confusions comme celle entre l'eau et le café.

9.7 Comparaison d'architectures de CNN

Plusieurs architectures de réseaux de neurones convolutifs ont été testés et comparés.

Pour que les résultats soient comparables, tous les tests ont été effectués avec les mêmes bandes de fréquences. Chaque architecture a été testée avec quatre validations croisées de cinq blocs et 500 epochs.

D'abord diverses architectures ont été testées, puis, suivant les performances, les meilleures modèles ont été sélectionnés et modifiés pour ensuite être retestés avec d'autres modifications qui semblaient donner de bons résultats dans d'autres modèles, ainsi que des changements pour répondre à certains problèmes (sur-ajustement, nombre trop important de paramètres...).

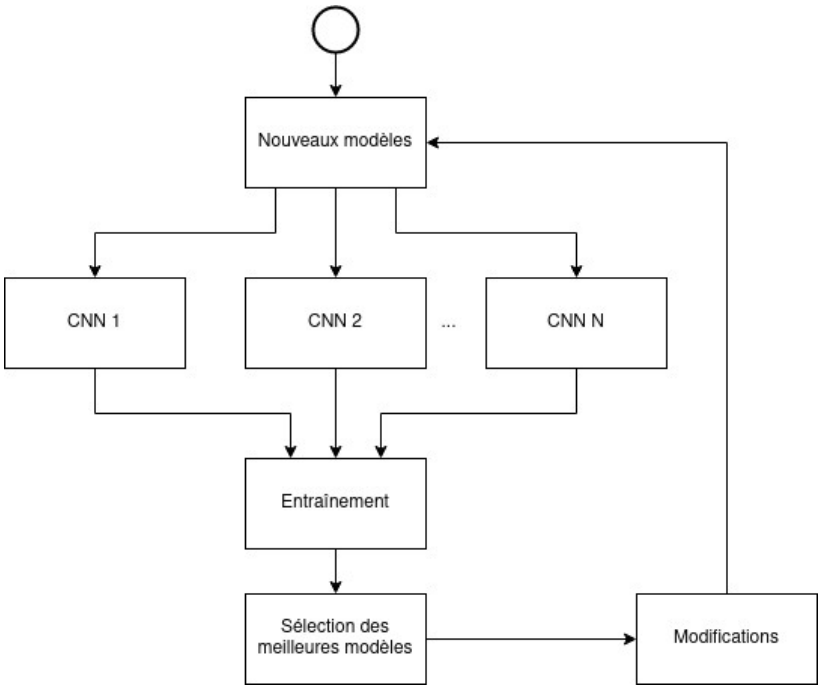


Figure 30: Processus pour décider quelles architectures tester

Les meilleures architectures ont ensuite été testée avec de l'arrêt prématuré, car 500 epochs étaient soit trop soit pas assez d'epochs suivant les modèles.

9.7.1 Couches avancées

Pour réduire le sur-ajustement, diminuer le nombre de poids synaptiques des réseaux de neurones ou encore améliorer les performances, plusieurs couches «avancées» et paramètres inhabituels ont été utilisés. Par exemple : décrochage (dropout), décrochage spatial (spatial dropout), sous-échantillonage global moyen (global average pooling), convolution avec un saut (strides) de deux.

9.7.1.1 Dropout

Les couches de *dropout*, ou en français de décrochage, sont des couches pour réduire le sur-ajustement, à chaque itération, les neurones ont une probabilité d'être retirés ; cette probabilité est paramétrable [51].

9.7.1.2 Spatial dropout

Le but du *spatial dropout* est le même que celui du décrochage : réduire le sur-ajustement.

Contrairement à ce dernier, le *spatial dropout* ne va pas enlever aléatoirement des neurones, mais il va enlever aléatoirement une carte de caractéristiques (feature map).

Le décrochage spatial répond à un problème que pose le décrochage au niveau des couches de convolutions : si les pixels adjacents sont fortement corrélés, alors le décrochage classique va juste résulter en un apprentissage plus lent [51,52].

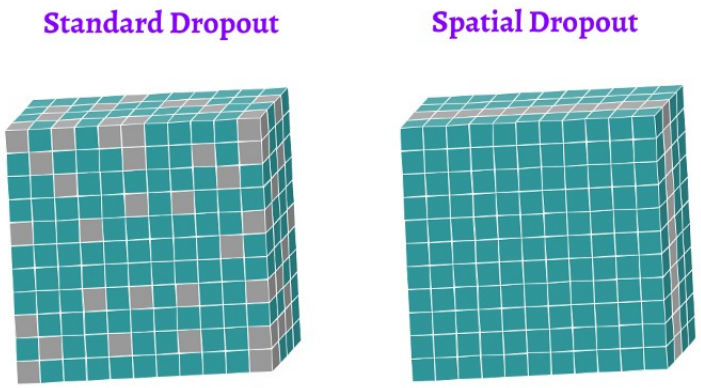


Figure 31: Différence d'approche entre le dropout et le spatial dropout, https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293

9.7.1.3 Global average pooling

À partir du constat que les couches entièrement connectées sont sujettes au sur-ajustement, le papier «Network in Network» propose de remplacer les couches entièrement connectées dans les réseaux de neurones convolutifs par une couche de *global average pooling*. Cette couche va prendre la moyenne de chaque carte de caractéristiques (feature map) que l'on va ensuite pouvoir donner à la couche de sortie. Cela va considérablement réduire le nombre de paramètres. De plus cette couche renforce la correspondance entre les cartes de caractéristiques et les classes [53].

Il est cependant possible d'ajouter une couche entièrement connectée après le *global average pooling*. C'est ce que fait le modèle ResNet-50 [54].

9.7.1.4 Convolution avec un saut de deux

Technique trouvée dans le papier «striving for Simplicity : The All Convolitonal Net» [55] qui obtient de bons résultats en remplaçant les couches de sous-échantillonnage par des couches de convolutions avec un saut de deux. Comme le sous-échantillonnage, cela va réduire la taille de l'image.

9.7.2 Résultats

Les résultats complets des comparaisons d'architectures sont disponibles en annexe.

Les paramètres qui ont été utilisés pour les dernières comparaisons d'architectures, avec de l'arrêt prématuré, sont les suivants :

Nombre de tests	4
Labels utilisés	Café, végétation naturelle, urbain, eau, poivre et thé
Nombre de pixels autour du label	4 (c.-à-d. une image de 9x9 pixels)
Canaux	Red, Blue, NIR, SWIR1
Arrêt prématuré	Sur le f1-score, patience de 150

Le modèle qui a obtenu les meilleures performances est celui-ci :

```
model = Sequential([
    Rescaling(1./2**16, input_shape=(image_width, image_height, image_depth)),
    BatchNormalization(),
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu"),
    Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=(3, 3)),
    Conv2D(filters=64, kernel_size=(3, 3), padding="same", activation="relu"),
    Conv2D(filters=64, kernel_size=(3, 3), padding="same", activation="relu"),
    MaxPooling2D(pool_size=(3, 3)),
    SpatialDropout2D(0.25),
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.25),
    Dense(nb_outputs, activation='softmax'),
])
```

Avec ces performances :

Nb paramètres	Précision moyenne	Rappel moyen	F-score moyen	F-score du café
74'966	70.785%	77.116%	73.191%	78.645%

On peut voir sur la matrice de confusion que le café est encore confondu avec par exemple l'eau. Cela ne devrait pas être le cas.

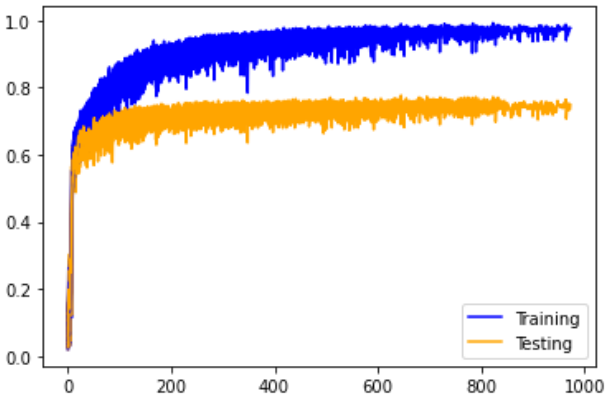


Figure 33: Graphe de l'évolution du f-score en fonction du nombre d'epochs

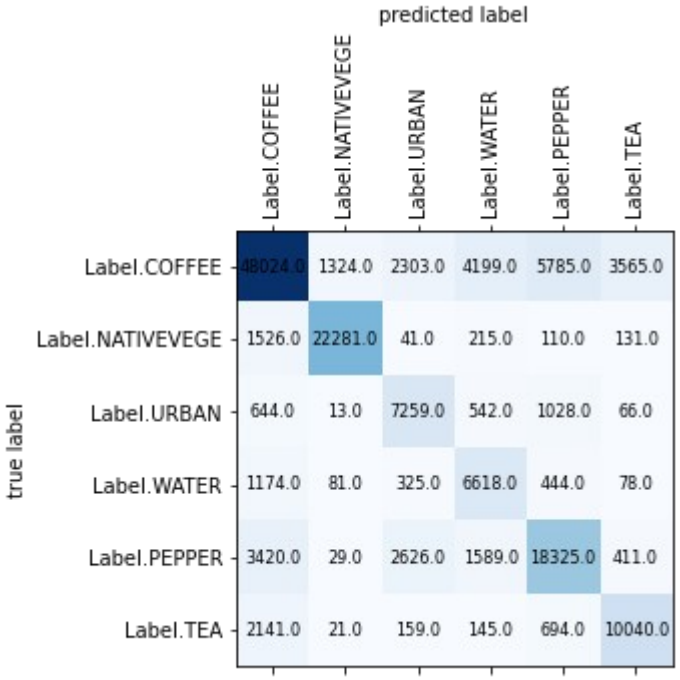


Figure 32: Matrice de confusion du modèle sélectionné

10 Organisation du travail

Chaque semaine, une réunion avait lieu pour avoir un retour sur le travail en cours et donner de nouvelles directives. Cette réunion était tenue à distance avec l'outil de vidéoconférence Microsoft Teams.

Pour réussir à bien structurer le travail, les tâches à effectuer ont été rentrées sur *Github project*. Cela donne un bon aperçu du travail qu'il reste à faire, permet de ne pas oublier ce qui a été discuté pendant les réunions et fixer des délais.

Le travail est ensuite réparti en trois colonnes : *To Do* (à faire), *In Progress* (en cours) et *Done* (terminé).

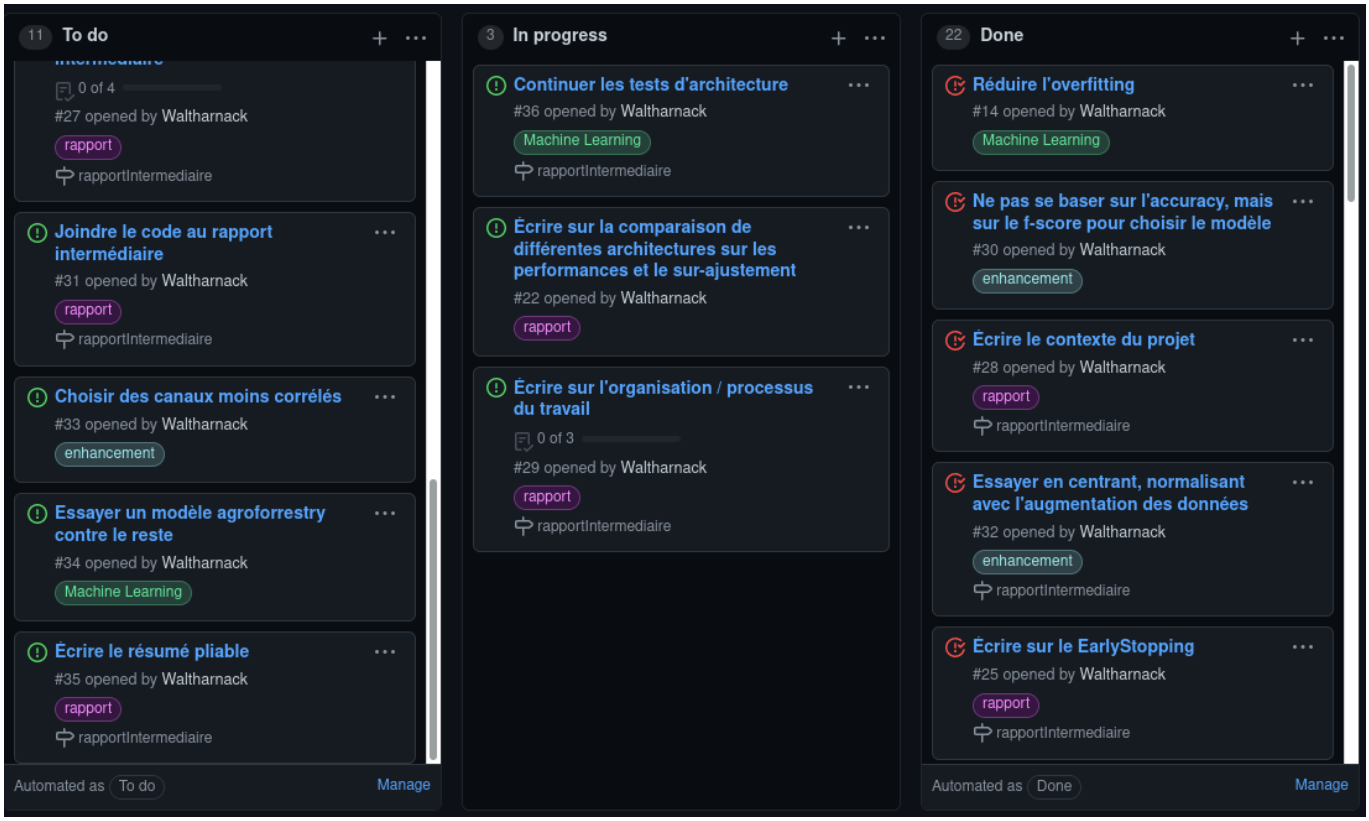


Figure 34: Organisation du TB sur Github project

11 Scripts

L'intégralité du code du projet peut être retrouvé sur GitHub : https://github.com/Waltharnack/vietnam_coffee_detection

Le code n'est pas entièrement inclus dans le rapport, seules les fonctions les plus intéressantes sont détaillées.

11.1 Structure du projet

Voici comment se présente la structure du projet si l'on ne garde que les fichiers contenant du code :

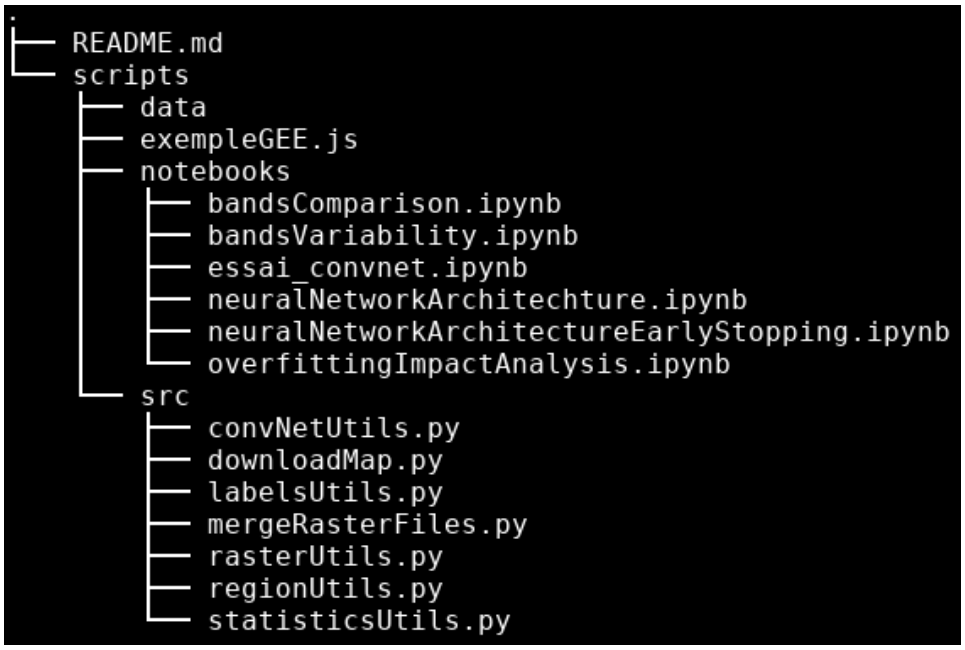


Figure 35: Aperçu de la structure du projet

Le dossier data n'est pas sur GitHub. Il contient les données utilisées par les scripts et ces données sont parfois volumineuses.

Autrement, le fichier exempleGEE.js montre un exemple en javascript permettant de télécharger des fichiers de rasters depuis l'interface en ligne de Google Earth Engine[1].

1 Disponible ici: https://code.earthengine.google.com/

11.2 Notebooks Jupyter

Le projet contient donc ces «notebooks» :

- bandsComparison.ipynb : compare les performances des combinaisons de bandes de fréquences lors de l'entraînement d'un réseau de neurones convolutifs
- bandsVariability.ipynb : compare la variabilité et les valeurs des bandes de fréquences du café et de quelques autres labels dans différentes situations
- essai_convnet.ipynb : contient des essais par-rapport aux réseaux de neurones convolutifs. Ce fichier a été créé au début du travail pour appréhender les réseaux de neurones convolutifs.
- neuralNetworkArchitecture.ipynb : c'est dans ce fichier que l'on compare différentes architectures de réseau de neurones convolutifs. Ce fichier ne test les modèles que sur 500 epochs.
- neuralNetworkArchitectureEarlyStopping.ipynb : ici on compare les modèles avec de l'arrêt prématuré se basant sur le f1-score, contrairement à neuralNetworkArchitecture.ipynb. Les résultats montrent aussi le f1-score qui est une mesure plus fiable que l'accuracy, notamment lorsque les classes sont déséquilibrées. Les derniers tests d'architectures sont donc fait dans ce «notebook».
- overfittingImpactAnalysis.ipynb : ce fichier permet de mieux comprendre comment les performances évoluent quand le réseau de neurones a du sur-ajustement.

11.3 Code source python

On pourrait classer ces fichiers en deux catégories : les fichiers utilitaires définissant des fonctions étant utilisées ailleurs et les scripts permettant de réaliser une action précise.

Dans la première catégorie il y a :

- convNetUtils.py : code qui concerne les réseaux de neurones convolutifs
- labelsUtils.py : code concernant les labels, notamment pour extraire les coordonnées des points géoréférencé de chaque labels
- rasterUtils.py : les fonctions pour traiter les fichiers de raster
- regionUtils.py : défini des régions et permet d'obtenir les coordonnées des labels qui sont contenues dans celles-ci
- statisticsUtils.py : les fonctions pour établir des statistiques

Et dans la deuxième catégorie on trouve :

- downloadMap.py : permet de télécharger des cartes depuis Google Earth Engine
- mergeRasterFiles.py : permet de fusionner plusieurs fichiers de raster exporté depuis Google Earth Engine. Attention toutefois : ce script nécessite une grande quantité de mémoire vive.

11.3.1 convNetUtils.py

Une des fonctions les plus importantes est la fonction *cross-validation* :

```
def cross_validation(model, dataset, bands, labels, epochs, nb_cross_validations=1, k=5,
early_stopping=False):
    """
    This is a function to do cross validation on a given keras model.
    :param model: the Keras neural network model
    :param dataset: the dataset, typically created with make_dataset_from_raster_files
    :param bands: an array of the position of the bands to use. ex: [3, 2, 1] will select bands Red, Green,
Blue
    if the dataset contains images with all bands. Bands positions start at zero.
    :param labels: an array of selected labels. Those labels should be entries of Label enum defined in
labelsUtils.py
    :param epochs: the number of epochs
    :param nb_cross_validations: the number of time to repeat the cross validation.
    :param k: the number of folds
    :param early_stopping: defines if early stopping is used
    :return: mean loss, mean accuracy, array of each history and the confusion matrix
    """
    labels_names = [label.name for label in labels]
    nb_labels = len(labels_names)
    histories = []
    mean_loss = 0
    mean_accuracy = 0
    total_conf_matrix = np.zeros((len(labels_names), len(labels_names)))

    model.summary()

    for validation in range(nb_cross_validations):
        np.random.shuffle(dataset)

        for fold, fold_indices in enumerate(k_fold_indices(dataset, k)):
            X_train, y_train, Y_train, X_test, y_test, Y_test = split_fold_into_train_test_sets(
                dataset, fold_indices[0], fold_indices[1], bands, labels_names, len(labels)
            )

            history, conf_matrix, accuracy, loss = train_and_evaluate_fold(
                X_test, X_train, Y_test, Y_train, y_test, y_train,
                epochs, fold, model, nb_labels,
                early_stopping
            )

            fold_size = len(y_train)

            histories.append(history)
            total_conf_matrix += conf_matrix
            mean_accuracy += accuracy * fold_size / (len(dataset) * nb_cross_validations)
            mean_loss += loss * fold_size / (len(dataset) * nb_cross_validations)

    return mean_loss, mean_accuracy, histories, total_conf_matrix
```

Elle permet de faire de la validation croisée en spécifiant le nombre de blocs (folds) et en lui donnant un modèle Keras de réseau de neurones.

Les mesures prennent en compte la taille des blocs pour le cas où ceux-ci n'auraient pas tous la même taille.

Pour obtenir une matrice de confusion représentant toutes les validations croisées, on fait simplement une somme de chaque matrice de confusion.

Keras permet de définir des classes avec des fonctions de rappels (callbacks) qui vont être appelées lors de certains évènements.

```
class Metrics(keras.callbacks.Callback):
    """
    Keras callback to provides additional metrics.
    It logs f1-score of the train and validation after each epoch. Those metrics can be used by the early stopping.
    Code inspired by: https://medium.com/@thongonary/how-to-compute-f1-score-for-each-epoch-in-keras-a1acd17715a2
    """

    def __init__(self, train, validation):
        """
        Initialize callback
        :param train: the train set as a tuple (train set values, correct labels)
        :param validation: the validation set as (test set values, correct labels)
        """

        super(Metrics, self).__init__()
        self.train = train
        self.validation = validation

    def on_epoch_end(self, epoch, logs={}):
        target_train = np.argmax(np.asarray(self.train[1]), axis=-1)
        predicted_train = np.argmax(np.asarray(self.model.predict(self.train[0])), axis=-1)
        f1_score_train = me.f1_score(target_train, predicted_train, average="macro")

        target_validation = np.argmax(np.asarray(self.validation[1]), axis=-1)
        predicted_validation = np.argmax(np.asarray(self.model.predict(self.validation[0])), axis=-1)
        f1_score_val = me.f1_score(target_validation, predicted_validation, average="macro")

        logs['f1_score_train'] = f1_score_train
        logs['f1_score_val'] = f1_score_val

        return
```

Ici on créé une classe *Metrics* qui va effectuer des mesures supplémentaires comme le f1-score. Cette mesure va être calculée après chaque epochs et va ensuite pouvoir être utilisée comme mesure repère pour l'arrêt prématuré.

On pourra aussi avoir un historique de ces mesures additionnelles.

Pour le moment *Metrics* n'ajoute que le f-score.

Pour entraîner un modèle, c'est la fonction *train_model* qui est utilisée :

```
def train_model(model, X_train, Y_train, X_test, Y_test, class_weights, epochs, steps_per_epoch,
early_stopping=False):
    """
    Train a Keras neural network model
    :param model: the Keras neural network model
    :param X_train: the image train set
    :param Y_train: the correct labels of train set in one hot encoding
    :param X_test: the image test set
    :param Y_test: the correct labels of test set in one hot encoding
    :param class_weights: weight of each class. If classes are imbalanced it will gives more importance
    to underrepresented classes
    :param epochs: the number of epochs
    :param steps_per_epoch: the number of steps per epoch
    :param early_stopping: defines if early stopping is used
    :return: the train history
    """

    # Define data generator arguments
    data_gen_args = dict(
        rotation_range=45,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
    )

    batch_size = 32

    # create data generator
    datagen = ImageDataGenerator(**data_gen_args)
    datagen.fit(X_train)
    train_datagen = datagen.flow(X_train, Y_train, batch_size=batch_size)

    callbacks = [Metrics(train=(X_train, Y_train), validation=(X_test, Y_test))]

    if early_stopping:
        callbacks.append(EarlyStopping(monitor='f1_score_val', patience=150, mode="max"))

    # Define fit arguments
    fit_args = dict(
        x=train_datagen,
        epochs=epochs,
        class_weight=class_weights,
        steps_per_epoch=steps_per_epoch,
        validation_data=(X_test, Y_test),
        callbacks=callbacks
    )

    return model.fit(**fit_args)
```

On remarque qu'actuellement l'augmentation des données prend toujours les mêmes paramètres : on décale légèrement les images horizontalement et verticalement, on les tourne jusqu'à 45 degrés et on les retourne horizontalement.

L'arrêt prématuré (early stopping) est optionnel et se base sur le f-score. Pour équilibrer les classes, un poids de classe (la variable class_weights) est utilisé. Il a été calculé préalablement, dans la fonction *train_and_evaluate_fold*, avec :

```
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
```

Il est à noté que la fonction class.weight.compute_class_weight provient de sklearn.utils.

Ensuite, dans cette même fonction *train_and_evaluate_fold*, on va évaluer le modèle sur l'ensemble de validation :

```
# Evaluate model
score = current_model.evaluate(X_test, Y_test, verbose=0)
loss = score[0]
accuracy = score[1]

# Add current confusion matrix to the global confusion matrix
pred = current_model.predict_on_batch(X_test)
pred = np.argmax(pred, axis=-1)

conf_matrix = me.confusion_matrix(y_test, pred, labels=np.arange(nb_labels))
```

Pour déterminer quelle classe est prédite, on va prendre la classe ayant la plus grande activation. Rappelons que la sortie de softmax représente une probabilité. On va donc choisir la classe qui a été prédite comme ayant la plus grande probabilité de correspondre à l'image.

11.3.2 rasterUtils.py

Dans ce fichier, la fonction *make_dataset_from_rater_files* permet de créer les jeux de données. Ces jeux de données vont ensuite pouvoir être utilisés pour entraîner et valider les modèles avec les fonctions qui se trouvent dans convNetUtils.py.

```
def make_dataset_from_raster_files(labels, raster_paths, labels_coordinates_list, nb_pixel_around):
    """
    Make a dataset by taking images around provided labels. Multiple rasters can be given and in this case images are
    taken inside every rasters.
    Dataset will not include images with 'NaN' values.
    :param labels: the labels to use
    :param raster_paths: the paths to rasters from which to take the values
    :param labels_coordinates_list: a list of dictionary of all coordinates for each labels.
    This a list to enable having multiple dictionaries created separately without having to merge them into one
    :param nb_pixel_around: the number of pixel to take around labels
    :return: the dataset
    """

    values = []

    for i, path in enumerate(raster_paths):
        with rasterio.open(path) as raster:
            for labels_coordinates in labels_coordinates_list:
                for label in labels:
                    for coordinates in labels_coordinates[label.value]:
                        out_img, out_transform = image_n_px_around_coordinates(
                            coordinates, nb_pixel_around, raster
                        )

                        lat = coordinates[0]
                        lon = coordinates[1]
                        point = geojson.Point((lat, lon))

                        # Don't add data if there is 'NaN' values
                        if not np.isnan(out_img).any():
                            values.append([label.name, out_img.tolist(), point])

    return values
```

On peut voir que les images contenant des valeurs inconnues ne sont pas gardées.

Ci-dessous, la fonction permettant d'extraire des carrés autour d'un point géoréférencé :

```
def square_of_n_px_around_coordinates(coordinate, nb_pixel_around, raster):
    """
    Create a square of a given number of pixels around a latitude and longitude tuple.
    :param coordinate: the latitude and longitude tuple
    :param nb_pixel_around: the number of pixels around a given (lat, lon)
    :param raster: the raster from which the data will be retrieved
    :return: a geojson object of the square
    """

    lat = coordinate[0]
    lon = coordinate[1]

    # Latitude and longitude are converted to row and col of pixels in the raster
    row, col = raster.index(lat, lon, precision=23)

    # To get the min and max coordinates, n pixel are taken around the row and col of given coordinates
and
    # then converted back to latitude and longitude.
    min_coord = rasterio.transform.xy(raster.transform, row - nb_pixel_around, col - nb_pixel_around)
    max_coord = rasterio.transform.xy(raster.transform, row + nb_pixel_around, col + nb_pixel_around)

    return shapely.geometry.box(min_coord[0], min_coord[1], max_coord[0], max_coord[1])
```

L'objet géométrique représentant le carré est ensuite fourni à la fonction mask de raserio pour découper l'image qui est à l'intérieur de ce carré :

```
def image_n_px_around_coordinates(coordinates, nb_pixel_around, raster):
    """
    Retrieves raster image n pixels around a given latitude and longitude
    :param coordinates: the latitude and longitude tuple
    :param nb_pixel_around: the number of pixel around the coordinates
    For example: 4px means that it will get a square of 9x9 px. (4px + 1px + 4px)^2
    :param raster: the raster from which the data will be retrieved
    :return: the raster image, the affine transform
    """

    polygon = square_of_n_px_around_coordinates(coordinates, nb_pixel_around, raster)
    return rasterio.mask.mask(raster, shapes=[polygon], crop=True, all_touched=True)
```

11.3.3 regionUtils.py

Pour utiliser regionUtils.py, il ne faut que le chemin vers les données soit juste.

Si les données se trouvent à un endroit différent, il suffit de changer les chemins d'accès qui se trouvent ici :

```
DATA_ROOT_PATH = '../data/'
DISTRICTS_PATH = DATA_ROOT_PATH + "districts/diaphantinh.geojson"
SOILMAP_PATH = DATA_ROOT_PATH + "soilmap/soilmap.geojson"
SHAPEFILE_ROOT_PATH = DATA_ROOT_PATH + 'labels/'
SHAPEFILE_PATHS = glob.glob(SHAPEFILE_ROOT_PATH + '**/*.shp')
```

Aucune donnée n'est disponible sur le github.

Pour ce qui est des fichiers diaphantinh.geojson et soilmap.geojson, ils sont disponibles librement en ligne [43,44].

Les fichiers contenant les points géoréférencés des labels sont les fichiers avec une extension «.shp».

Ce fichier met à disposition des fonctions permettant de créer des dictionnaires avec les labels et leurs points géoréférencés.

Les fonctions vont prendre en considération des régions données comme par exemple une partie du Vietnam ou va créer un dictionnaire par région comme pour les districts ou les types de sols.

Voici la fonction qui va le faire pour la partie du Vietnam que nous utilisons :

```
def vietnam_labels_coordinates():
    """
    Get coordinates of each entries of every labels, takes only the region in Vietnam where we have labels.
    :return: the labels coordinates dictionary
    """

    district_file = gpd.read_file(DISTRICTS_PATH)
    vietnam_shape = shapely.ops.unary_union([shape for shape in district_file['geometry']])
    selected_region = shapely.geometry.box(106.9998606274592134, 10.9999604855719539, 109.0000494390797456, 15.5002505644255208)
    boundaries_shape = vietnam_shape & selected_region
    return labels_coordinates_from_files(SHAPEFILE_PATHS, boundaries_shape)
```

Et vu que l'on travaille avec des fichiers geojson, on voudrait extraire les formes après les avoir filtrées, c'est ce que fait la fonction *shapes_from_geojson* :

```
def shapes_from_geojson(file, names, name_column='Name', geometry_column='geometry'):
    """
    Extract shapes from a geojson file
    :param file: the geojson file
    :param names: names to keep
    :param name_column: name of the column where the names are
    :param geometry_column: name of the column with the shape
    :return: the shapes
    """

    return [shape for shape in file[file[name_column].isin(names)][geometry_column]]
```

11.3.4 labelsUtils.py

Ce fichier défini un enum pour les labels :

```
class Label(Enum):
    CACAO = 1
    COFFEE = 2
    COMPLEX_OIL = 3
    NATIVEVEGE = 4
    OIL_PALM = 5
    RUBBER = 6
    UNKNOWN = 7
    SEASONAL = 8
    URBAN = 9
    WATER = 10
    OTHER_TREE = 11
    OTHER_NO_TREE = 12
    NATIVE_NO_TREE = 13
    WATER_OTHER = 14
    PEPPER = 15
    CASSAVA = 16
    TEA = 17
    RICE = 18
    BANANA_JUNG = 19
    BABY_PALM = 20
    CUT_OFF_REGROW = 21
    NATURAL_WETLAND = 22
    INTERCROP = 23
    DECIDUOUS_FOREST = 24
    STICK_PEPPER = 25
    FLOODED_PLANTATION = 26
    PINE_TREES = 27
    COCONUT = 28
    BAMBOO = 29
    SAVANA = 30
    MANGO = 31
    OTHER_FRUIT_TREE_CROP = 32
    WATER_MINE = 33
```

Et voici comment sont créés les dictionnaires contenant les coordonnées de chaque points géoréférencés des labels :

```
def labels_coordinates_from_files(shapefiles_paths, boundaries):
    """
    Create a dictionary with coordinates of each entries for every labels
    :param shapefiles_paths: the path to the labels shapes
    :param boundaries: the boundaries outside which entries are excluded
    :return: the labels coordinates dictionary
    """

    # Create a dictionary which will contain all
    # points classified in the opened shapefiles
    # with their coordinates lat-lon
    labels_coordinates = {}

    # Initialize the dictionary
    for i in range(1, 34):
        labels_coordinates[i] = []

    # Add each points coordinate
    # in its corresponding class
    for path in shapefiles_paths:
        for shape_record in shapefile.Reader(path).shapeRecords():
            if shape_record.record.Class <= len(Label):
                current_list = labels_coordinates.get(shape_record.record.Class)

                # Add label coordinate only if this label is inside the boundaries
                if shapely.geometry.Point(shape_record.shape.points[0]).within(boundaries):
                    current_list.append(shape_record.shape.points[0])
                    labels_coordinates[shape_record.record.Class] = current_list

    return labels_coordinates
```

11.3.5 labelsUtils.py

Une fonction intéressante, dans ce fichier, est la fonction permettant d'obtenir des mesures à partir d'une matrice de confusion :

```
# See: https://stackoverflow.com/questions/48100173/how-to-get-precision-recall-and-f-measure-from-confusion-matrix-in-python
def recall_precision_fscore_from_confusion_matrix(conf_matrix):
    """
    compute recall, precision and f-score metrics from a confusion matrix
    :param conf_matrix: the confusion metrics
    :return: recall, precision and f-score
    """

    tp = np.diag(conf_matrix)
    fp = np.sum(conf_matrix, axis=0) - tp
    fn = np.sum(conf_matrix, axis=1) - tp
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    fscore = 2 * recall * precision / (recall + precision)

    return recall, precision, fscore
```

11.3.6 downloadMap.py

Voici comment ont peut exporter les image des cartes du Vietnam de Landsat 8 :

```
# Initialize Google Earth Engine library
ee.Initialize()

# Landsat 8 collection
SATELLITE_DATASET = "LANDSAT/LC08/C01/T1_SR"

# Rectangle region of interest
REGION_RECTANGLE = ee.Geometry.Rectangle([
    106.9998606274592134, 10.9999604855719539,
    109.0000494390797456, 15.5002505644255208
])

# Country name to retrieve country geometry
COUNTRY_NAME = 'Vietnam'

image_collection = create_image_collection('2017-03-01', '2017-05-01')
median_img = image_collection_to_median_img(image_collection)
export_task = create_export_task(median_img, '2017_march_april')

# Start the task
start_task(export_task)
```

Si l'on veut on peut bien sûr changer la région, la plage de temps et les jeux de données des satellites qui sont utilisés.

Une fonction, appelée dans create_image_collection, va masquer les données appartenant aux nuages :

```
def mask_clouds(image):
    """
    mask clouds in a landsat 8 image.
    See: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC08_C01_T1_SR?
hl=in&skip_cache=false
    :param image: the image
    :return: image without clouds
    """

    # Bits 3 and 5 are cloud shadow and cloud, respectively.
    cloud_shadow_bit_mask = (1 << 3)
    clouds_bit_mask = (1 << 5)

    # Get the pixel QA band.
    qa = image.select('pixel_qa')

    # Both flags should be set to zero, indicating clear conditions.
    mask = qa.bitwiseAnd(cloud_shadow_bit_mask).eq(0) and (qa.bitwiseAnd(clouds_bit_mask).eq(0))

    return image.updateMask(mask)
```

11.3.7 mergeRasterFiles.py

Les fichiers sont fusionnés avec rasterio comme ceci :

```
# For each folder in FOLDER_NAMES, merge all rasters to a file named 'merged.tif'
for folder_name in FOLDER_NAMES:
    folder_path = './' + folder_name + '/'

     print("Merging raster files...")
    raster_files = open_raster_files(folder_path)
    merged_result, out_transform = rasterio.merge.merge(raster_files)
    print("Raster files merged")

    metadata = raster_files[0].meta.copy()
    metadata.update({
        "driver": "GTiff",
        "height": merged_result.shape[1],
        "width": merged_result.shape[2],
        "transform": out_transform,
        "crs": raster_files[0].crs.to_proj4()
    })

    print("Writing merged raster to disk...")

    # Write merged raster to disk
    with rasterio.open(ROOT_PATH + folder_path + 'merged.tif', "w", **metadata) as dest:
        dest.write(merged_result)

    print("Finished writing to disk")
```

Cette façon de faire est coûteuse en mémoire vive. Il serait préférable de traiter le fichier de raster par blocs au lieu de charge l'entièreté des fichiers de raster en mémoire pour les traiter[2].

2 Voir : https://gis.stackexchange.com/questions/348925/merging-rasters-with-rasterio-in-blocks-to-avoid-memoryerror

12 Conclusion

Lors de cette première partie du travail de bachelor, j'ai eu la chance d'apprendre le fonctionnement des réseaux de neurones convolutifs et de les utiliser. Je me suis aussi familiarisé au système d'information géographique QGIS.

J'ai trouvé particulièrement intéressant de mieux comprendre les réseaux de neurones convolutifs et d'essayer d'améliorer les performances de mes modèles en m'inspirant d'architectures que je trouvais comme «Net In Network [53]» ou «The all convolutional net [55]».

Jusqu'à présent, j'ai exploré les données et je les ai préparées pour qu'elles soient utilisées dans les modèles. Avec QGIS, j'ai pu visualiser les cartes et les labels. Enfin, j'ai comparé les performances de diverses architectures de réseau de neurones et de multiples combinaisons de bandes de fréquences. Cependant, il me semblerait intéressant de continuer d'explorer les combinaisons de canaux, car les modèles se méprennent sur des labels qui devraient pourtant pouvoir être distingués comme le café et l'eau.

Quand un modèle final sera sélectionné, il restera encore à développer un outil permettant de visualiser et d'analyser les résultats des prédictions à l'aide d'une carte.

Il pourrait aussi être intéressant de tester des modèles avec tous les labels et d'essayer des modèles groupant le café et les autres cultures en une seule classe.

Enfin, il faudra réussir à déterminer les zones à risques de déforestation.

Une difficulté que j'ai rencontrée a été de bien choisir les hyperparamètres, car bien qu'il y ai des recommandations, il n'y a pas de règles permettant de trouver les hyperparamètres les plus adaptés à un problème.

Je me réjouis de pouvoir continuer à avancer dans ce travail de bachelor, en espérant pouvoir améliorer encore les performances.

Bibliographie

[1] « Companies' 'zero deforestation' pledges: everything you need to know ». *the Guardian* [En ligne]. 2017. Disponible sur : http://www.theguardian.com/sustainable-business/2017/sep/29/companies-zero-deforestation-pledges-agriculture-palm-oil-environment (consulté le 7 mai 2021)

[2] *po-production.pdf* [En ligne]. Disponible sur : http://www.ico.org/prices/po-production.pdf (consulté le 27 mars 2021)

[3] *icc-124-9e-profile-vietnam.pdf* [En ligne]. Disponible sur : http://www.ico.org/documents/cy2018-19/icc-124-9e-profile-vietnam.pdf (consulté le 4 mai 2021)

[4] « Toward a sustainable coffee future in Vietnam's Central Highlands ». *The Alliance of Bioversity International and the International Center for Tropical Agriculture (CIAT)* [En ligne]. Disponible sur : https://alliancebioversityciat.org/news_and_blogs/toward-a-sustainable-coffee-future-in-vietnams-central-highlands/ (consulté le 6 mai 2021)

[5] Inoguchi T. E. & A. « From driver to solution: coffee agroforestry in Viet Nam ». *un-redd-website* [En ligne]. 2019. Disponible sur : https://www.un-redd.org/post/2019/07/10/from-driver-to-solution-coffee-agroforestry-in-viet-nam (consulté le 7 mai 2021)

[6] *Interface de programmation* [En ligne]. *Wikipédia*. 16 mars 2021. Disponible sur : https://fr.wikipedia.org/w/index.php?title=Interface_de_programmation&oldid=180919106 (consulté le 26 avril 2021)

[7] « International Center for Tropical Agriculture ». *CIAT* [En ligne]. Disponible sur : https://ciat.cgiar.org/ (consulté le 27 avril 2021)

[8] « Landsat 8 Bands | Landsat Science ». Disponible sur : https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-bands (consulté le 27 mars 2021)

[9] « Landsat 8 Overview | Landsat Science ». Disponible sur : https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-overview (consulté le 23 février 2021)

[10] *Landsat 8* [En ligne]. *Wikipédia*. 14 novembre 2020. Disponible sur : https://fr.wikipedia.org/w/index.php?title=Landsat_8&oldid=176583718 (consulté le 14 mai 2021)

[11] Chandra A. L. « McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron ». *Medium* [En ligne]. 2018. Disponible sur : https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1 (consulté le 29 mars 2021)

[12] Loiseau J.-C. B. « Rosenblatt's perceptron, the very first neural network ». *Medium* [En ligne]. 2021. Disponible sur : https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a (consulté le 13 avril 2021)

[13] Nielsen M. A. « Neural Networks and Deep Learning ». 2015. Disponible sur : http://neuralnetworksanddeeplearning.com (consulté le 27 mars 2021)

[14] « CS231n Convolutional Neural Networks for Visual Recognition ». Disponible sur : https://cs231n.github.io/convolutional-networks/ (consulté le 15 avril 2021)

[15] « Understanding Convolutions - colah's blog ». Disponible sur : https://colah.github.io/posts/2014-07-Understanding-Convolutions/ (consulté le 23 avril 2021)

[16] Stanford University School of Engineering. *Lecture 5 | Convolutional Neural Networks* [En ligne]. 2017. Disponible sur : https://www.youtube.com/watch?v=bNb2fEVKeEo (consulté le 14 mai 2021)

[17] Gholamalinezhad H., Khosravi H. « Pooling Methods in Deep Neural Networks, a Review ». p. 16.

[18] « CS 230 - Pense-bête de réseaux de neurones convolutionnels ». Disponible sur : https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels (consulté le 23 avril 2021)

[19] « "Convolutional neural networks (CNN) tutorial" ». Disponible sur : https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/ (consulté le 26 avril 2021)

[20] « An Intro to the Earth Engine Python API | Google Earth Engine ». *Google Developers* [En ligne]. Disponible sur : https://developers.google.com/earth-engine/tutorials/community/intro-to-python-api-guiattard (consulté le 23 février 2021)

[21] Team K. « Keras documentation: About Keras ». Disponible sur : https://keras.io/about/ (consulté le 1 mai 2021)

[22] « Rasterio: access to geospatial raster data — rasterio documentation ». Disponible sur : https://rasterio.readthedocs.io/en/latest/ (consulté le 1 mai 2021)

[23] « GEOS ». Disponible sur : https://trac.osgeo.org/geos/ (consulté le 1 mai 2021)

[24] « The Shapely User Manual — Shapely 1.7.1 documentation ». Disponible sur : https://shapely.readthedocs.io/en/stable/manual.html (consulté le 1 mai 2021)

[25] Bora B. « An Introduction to Shapely with Python ». *Medium* [En ligne]. 2021. Disponible sur : https://towardsdatascience.com/shapely-with-python-daca70af1366 (consulté le 1 mai 2021)

[26] « scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation ». Disponible sur : https://scikit-learn.org/stable/ (consulté le 1 mai 2021)

[27] « NumPy ». Disponible sur : https://numpy.org/ (consulté le 1 mai 2021)

[28] « pandas - Python Data Analysis Library ». Disponible sur : https://pandas.pydata.org/ (consulté le 1 mai 2021)

[29] « GeoPandas 0.9.0 — GeoPandas 0.9.0 documentation ». Disponible sur : https://geopandas.org/ (consulté le 1 mai 2021)

[30] « Matplotlib: Python plotting — Matplotlib 3.4.1 documentation ». Disponible sur : https://matplotlib.org/ (consulté le 1 mai 2021)

[31] « seaborn: statistical data visualization — seaborn 0.11.1 documentation ». Disponible sur : https://seaborn.pydata.org/ (consulté le 1 mai 2021)

[32] « math — Mathematical functions — Python 3.9.4 documentation ». Disponible sur : https://docs.python.org/3/library/math.html (consulté le 1 mai 2021)

[33] « time — Time access and conversions — Python 3.9.4 documentation ». Disponible sur : https://docs.python.org/3/library/time.html (consulté le 1 mai 2021)

[34] « glob — Unix style pathname pattern expansion — Python 3.9.4 documentation ». Disponible sur : https://docs.python.org/3/library/glob.html (consulté le 1 mai 2021)

[35] Gillies S., Butler H., Daly M., Doyle A., Schaub T. « The GeoJSON Format ». Disponible sur : https://tools.ietf.org/html/rfc7946 (consulté le 4 mai 2021)

[36] Gillies S. *geojson: Python bindings and utilities for GeoJSON* [En ligne]. Disponible sur : https://github.com/jazzband/geojson (consulté le 4 mai 2021)

[37] Comber S. *spacv: Spatial cross-validation in Python* [En ligne]. Disponible sur : https://github.com/SamComber/spacv (consulté le 4 mai 2021)

[38] « Project Jupyter ». Disponible sur : https://www.jupyter.org (consulté le 11 mai 2021)

[39] « Google Earth Engine ». Disponible sur : https://earthengine.google.com (consulté le 26 avril 2021)

[40] « Vietnam - Climate | Britannica ». Disponible sur : https://www.britannica.com/place/Vietnam/Climate (consulté le 9 mars 2021)

[41] « Harvest Seasons - AmaRin Coffee - Vietnam Coffee Roaster and Supplier ». Disponible sur : http://amarin.com.vn/coffee-harvest-season (consulté le 12 mars 2021)

[42] *ap301e.pdf* [En ligne]. Disponible sur : http://www.fao.org/3/ap301e/ap301e.pdf (consulté le 12 mars 2021)

[43] « Provincial administration of Vietnam - OD Mekong Datahub ». Disponible sur : https://data.opendevelopmentmekong.net/en/dataset/a-phn-tnh?type=dataset (consulté le 27 avril 2021)

[44] « Soil types of Vietnam - Datasets - OD Mekong Datahub ». Disponible sur : https://data.opendevelopmentmekong.net/en/dataset/soil-types-in-vietnam?type=dataset (consulté le 27 avril 2021)

[45] Brownlee J. *How to Configure k-Fold Cross-Validation* [En ligne]. *Machine Learning Mastery*. 30 juillet 2020. Disponible sur : https://machinelearningmastery.com/how-to-configure-k-fold-cross-validation/ (consulté le 28 avril 2021)

[46] Brownlee J. *How to Configure Image Data Augmentation in Keras* [En ligne]. *Machine Learning Mastery*. 11 avril 2019. Disponible sur :

https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/ (consulté le 4 mai 2021)

[47] « Learning from imbalanced data. » *Jeremy Jordan* [En ligne]. 2018. Disponible sur : https://www.jeremyjordan.me/imbalanced-data/ (consulté le 28 avril 2021)

[48] « Softmax Function ». *DeepAI* [En ligne]. 2019. Disponible sur : https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (consulté le 28 avril 2021)

[49] « The Many Band Combinations of Landsat 8 ». *L3Harris Geospatial* [En ligne]. Disponible sur : https://www.l3harrisgeospatial.com/Support/Self-Help-Tools/Help-Articles/Help-Articles-Detail/ArtMID/10220/ArticleID/15691/The-Many-Band-Combinations-of-Landsat-8 (consulté le 23 février 2021)

[50] *Landsat 8 Bands and Band Combinations* [En ligne]. *GIS Geography*. 18 octobre 2019. Disponible sur : https://gisgeography.com/landsat-8-bands-combinations/ (consulté le 23 février 2021)

[51] Thevenot A. « Main Dropout Methods : Mathematical and Visual Explanation ». *Medium* [En ligne]. 2021. Disponible sur : https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293 (consulté le 7 mai 2021)

[52] Team K. « Keras documentation: SpatialDropout2D layer ». Disponible sur : https://keras.io/api/layers/regularization_layers/spatial_dropout2d/ (consulté le 7 mai 2021)

[53] Lin M., Chen Q., Yan S. « Network In Network ». *arXiv:1312.4400 [cs]* [En ligne]. 4 mars 2014. Disponible sur : http://arxiv.org/abs/1312.4400 (consulté le 7 mai 2021)

[54] Cook A. « Global Average Pooling Layers for Object Localization ». Disponible sur : https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/ (consulté le 14 mai 2021)

[55] Springenberg J. T., Dosovitskiy A., Brox T., Riedmiller M. « Striving for Simplicity: The All Convolutional Net ». *arXiv:1412.6806 [cs]* [En ligne]. 13 avril 2015. Disponible sur : http://arxiv.org/abs/1412.6806 (consulté le 7 mai 2021)

13 Annexe

Table 1: Comparaison d'architectures sur 500 epochs

Architecture	Nb paramètres	erreur moyenne	Accuracy moyenne	F-score du café	Rappel du café
2x8 conv, MaxPooling k=(2, 2), 4x8 conv, MaxPooling k=(2, 2), 1x256 FC with 0.5 dropout	11'426	0.967	62.53%	62.60%	50.30%
2x8 conv, MaxPooling k=(2, 2), 0.25 dropout, 4x8 conv, MaxPooling k=(2, 2), 0.25 dropout, 1x256 FC with 0.5 dropout	11'426	0.950	62.32%	61.14%	48.88%
2x16 conv, MaxPooling k=(2, 2), 2x16 conv, MaxPooling k=(2, 2), 1x256 FC with 0.5 dropout	21'522	0.938	67.06%	68.44%	57.10%
2x16 conv, MaxPooling k=(2, 2), 4x16 conv, MaxPooling k=(2, 2), 1x256 FC with 0.5 dropout	23'602	0.981	65.98%	67.39%	56.09%
2x16 conv, MaxPooling k=(2, 2), 0.25 SpatialDropout, 2x16 conv, 2x64 FC with 0.5 dropout	24'338	0.849	67.06%	66.53%	54.82%
1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 0.5 SpatialDropout, MaxPooling2D k=(2, 2) s=(1, 1), GlobalAveragePooling	27'394	0.892	70.29%	71.59%	62.02%
2x32 conv, 2x64 conv, MaxPooling2D k=(2, 2), GlobalAveragePooling	29'650	1.008	70.68%	73.55%	65.93%
2x32 conv, 2x64 conv, GlobalAveragePooling	29'650	0.960	69.89%	72.88%	65.22%
2x16 conv, MaxPooling k=(2, 2), 0.25 SpatialDropout, 2x16 conv, 128, 64 FC with 0.5 dropout	44'882	0.868	67.33%	66.76%	55.21%
1x16 conv, 0.25 SpatialDropout, 2x16 conv, 1x16 conv s=(2, 2) 1x128 FC with 0.5 dropout	55'442	0.799	69.67%	70.16%	59.82%
1x8 conv, 0.25 SpatialDropout, 1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 1x128 conv, GlobalAveragePooling	61'098	0.817	71.20%	74.05%	69.94%
1x8 conv, 1x16 conv, MaxPooling k=(2, 2) s=(1, 1), 1x32 conv, 1x64 conv, MaxPooling k=(2, 2), 1x64 conv, 1x128 conv, 0.25	61'098	1.055	70.62%	74.12%	67.40%

SpatialDropout, MaxPooling k=(2, 2) s=(1, 1), GlobalAveragePooling					
1x8 conv, 0.25 SpatialDropout, 1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 0.25 SpatialDropout, 1x128 conv, GlobalAveragePooling	61'098	0.795	70.23%	72.05%	64.45%
2x16 conv, MaxPooling k=(2, 2), 0.25 SpatialDropout, 2x16 conv, 1x256 FC with 0.5 dropout	70'674	0.824	68.69%	69.10%	58.41%
2x16 conv, 0.25 SpatialDropout, 2x16 conv, 3x64 FC with 0.2, 0.2, 0.5 dropout	95'058	0.845	66.81%	66.04%	53.67%
2x16 conv, 0.25 SpatialDropout, MaxPooling k=(2, 2) s=(1, 1), 2x32 conv, 0.25 SpatialDropout, MaxPooling k=(2, 2), 2x64 conv, 1x128 FC with 0.5 dropout	164'146	0.827	68.16%	68.12%	57.60%
6x8 SeparableConv, 1x256 FC with 0.5 dropout	168'298	1.265	44.06%	49.67%	37.25%
6x8 conv, 1x256 FC with 0.5 dropout	169'122	0.904	67.45%	69.19%	59.10%
4x16 conv, BatchNorm -> ReLU, 2x128 FC with 0.5 dropout	186'898	1.116	71.24%	75.50%	70.15%
4x16 conv, BatchNorm -> ReLU, 3x128 FC with 0.5 dropout	203'410	1.164	71.01%	74.93%	69.58%
4x16 conv with BatchNorm -> ReLU, 3x128 FC with 0.25, 0.25, 0.5 dropout	203'410	0.895	70.20%	73.95%	66.33%
2x16 conv, 0.25 SpatialDropout, 2x32 conv, 1x32 conv s=(2, 2), 0.25 SpatialDropout, 2x64 conv, 1x128 FC with 0.5 dropout	242'002	0.808	68.90%	69.00%	58.44%
4x16 conv, 1x256 FC with 0.5 dropout	336'914	0.910	72.75%	75.34%	68.78%
1x16 conv, 0.25 SpatialDropout, 3x16 conv, 1x256 FC with 0.5 dropout	336'914	0.791	72.04%	73.48%	65.54%
4x16 conv, 0.25 dropout, 1x256 FC with 0.5 dropout	336'914	0.862	71.19%	73.44%	64.92%
4x16 conv, BatchNorm -> ReLU, 1x256 FC with 0.5 dropout	337'170	0.947	72.27%	76.07%	70.86%
8, 0.25 SpatialDropout, 16, 32, 32 conv, 2x128 FC with 0.25 dropout	356'042	0.920	71.87%	75.09%	71.78%
1x16 conv, 0.25 SpatialDropout,	468'498	0.834	71.31%	73.31%	66.18%

3x16 conv, 3x256 FC with 0.2, 0.2, 0.2 dropout					
1x16 conv, 0.25 SpatialDropout, 3x16 conv, 3x256 FC with 0.25, 0.25, 0.5 dropout	468'498	0.813	70.92%	71.68%	62.68%
4x16 conv with BatchNorm -> ReLU, 3x256 FC with 0.25, 0.25, 0.5 dropout	468'754	0.895	71.49%	75.09%	69.04%
4, 8, 16, 32 conv, BatchNorm -> ReLU, 1x256 FC with 0.5 dropout	668'398	0.980	69.99%	74.32%	68.10%
8, 0.25 SpatialDropout, 16, 32, 0.25 SpatialDropout, 64 conv, 1x128 FC, 0.2 dropout, 1x64 FC, 0.5 dropout	683'306	0.809	68.34%	67.36%	56.06%
8, 0.25 SpatialDropout, 16, 32, 32 conv, 2x256 FC with 0.2 and 0.5 dropout	737'994	0.852	72.10%	74.65%	70.14%
8, 0.25 SpatialDropout, 16, 32, 64 conv, 2x256 FC with 0.5 dropout	1'405'674	1.010	72.95%	76.23%	75.08%

Table 2: Comparaison d'architectures avec arrêt prématuré

architecture	nb paramètres	précision moyenne	rappel moyen	f-score moyen	f-score du café
2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(3, 3), MaxPooling2D k=(3, 3), 0.5 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout	74'966	70.785%	77.116%	73.191%	78.645%
2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.5 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout	44'246	69.978%	77.268%	72.685%	77.537%
2x32 conv, 1x32 s=(3, 3), 2x64 conv, 1x64 s=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout	121'142	70.404%	75.514%	72.575%	78.112%
3x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 3x64 conv k=(3, 3), MaxPooling2D k=(3, 3), 0.5 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout	121'142	70.02%	75.58%	72.15%	77.97%

2x32 conv k=(2, 2), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.5 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout	38'486	69.256%	77.670%	71.985%	76.738%
1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 0.25 SpatialDropout, MaxPooling2D k=(2, 2) s=(1, 1), GlobalAveragePooling	27'462	69.098%	76.992%	71.897%	75.656%
1x16 conv k=(3, 3), 1x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x32, 1x32 FC with 0.25 dropout	33'286	68.912%	77.044%	71.706%	76.702%
1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 0.25 SpatialDropout, MaxPooling2D k=(2, 2) s=(1, 1), GlobalAveragePooling, 1x128 FC	36'166	68.644%	77.186%	71.492%	75.151%
2x32 conv, 2x64 conv, MaxPooling2D k=(2, 2), GlobalAveragePooling	29'782	68.885%	76.807%	71.425%	76.299%
1x16 conv, 0.25 SpatialDropout, 3x16 conv, 1x256 FC with 0.5 dropout	336'982	68.402%	76.155%	71.321%	75.174%
1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 0.25 SpatialDropout, MaxPooling2D k=(2, 2) s=(1, 1), GlobalAveragePooling, 1x32 FC	29'350	67.995%	77.394%	71.051%	74.181%
1x16 conv k=(3, 3), 1x32 conv k=(2, 2), 1x64 conv k=(2, 2), 1x64 conv k=(1, 1), 0.25 SpatialDropout, MaxPooling2D k=(2, 2) s=(1, 1), GlobalAveragePooling, 1x32 FC - without padding	17'382	67.278%	76.134%	70.189%	74.995%
1x16 conv k=(3, 3), 1x32 conv k=(3, 3), 1x64 conv k=(2, 2), 1x64 conv k=(1, 1), 0.25 SpatialDropout, MaxPooling2D k=(2, 2), GlobalAveragePooling, 1x32 FC with 0.25 dropout	19'942	66.244%	76.397%	69.413%	72.413%
1x8 conv, 0.25 SpatialDropout, 1x16 conv, 1x32 conv, 1x64 conv s=(2, 2), 1x64 conv, 1x128 conv,	61'134	66.630%	71.895%	68.612%	74.146%

GlobalAveragePooling					

Table 3: Journal de travail

Date	activité
23.02.21	Lire TB Mr. Rod, Regarder des informations sur Landsat8, Suivre le tutoriel d'installation de l'API python de GEE et Faire quelques essais de script pour interagir avec GEE. Lire de la documentation QGIS Recherche d'informations sur les différentes bandes de fréquences
24.02.21	Quelques tests de visualisation de combinaisons de bandes de fréquences sur GEE
26.02.21	Continuer à faire des essais avec GEE pour télécharger des données
02.03.21	Regarder la vidéo sur Terra-i et les vidéos GEE recommandées par Pr. Perez-Uribe Récupérer cartes 2013 et labels depuis le serveur atlas Afficher les labels situant le café par-dessus un raster virtuel formé des rasters de 2013 sur QGIS. Déterminer la région du Vietnam qui est récupérée dans les images satellites de 2013 sur le serveur atlas. Améliorer le script de téléchargement de carte depuis GEE
03.03.21	Améliorer le script de téléchargement de carte depuis GEE, Script utilisant rasterio pour fusionner plusieurs fichiers tif en un seul et l'écrire sur le disque, Script utilisant PyShp pour lire les fichiers de shapefile contenant les points labelisés et lister les coordonnées.
04.03.21	Réussir à découper un carré de quelques px autour d'un emplacement de café, il reste à corriger les valeurs min-max des bandes qu'on obtient
05.03.21	Correction de la taille des carrés coupés. Créer un notebook Jupyter pour analyser les résultats. Améliorer le script pour découper la partie écriture des carré labelisé sur le disque et celle du calcul des statistiques par-rapport à une date. Utilisation de pandas pour rentrer les données, possibilité de comparer les valeurs des classes pour les différentes bandes de fréquence.
09.03.21	Recherche d'informations sur le climat au Vietnam. Amélioration du script de téléchargement de carte pour pouvoir prendre plusieurs plage de temps. Comparaison du café à travers des saisons («hiver», «été»). Amélioration de la visualisation des résultats.
10.03.21	Essayer de trouver une découpe de l'année qui permet d'avoir assez de donnée pour ne pas avoir trop de nuage, mais être plus précis que seulement deux saisons. Je me suis renseigné et j'ai appris que les labels fournis devraient daté de 2016-2018. Téléchargement des cartes médiane de tout les deux mois sur une plage de 2014 à 2020.

	Début d'analyse exploratoire des données.
11.03.21	Normalisation des données, améliorer visualisation des données
12.03.21	Analyse exploratoire des données
16.03.21	Correction de deux erreurs : j'utilisais la moyenne au lieu de la médiane qui est meilleure pour comparer ces valeurs et les valeurs nodata étaient pris en compte. Ajout de fonctions pour traiter les labels par région, découpage des labels de café du Vietnam en deux région.
17.03.21	Réusinage du script pour analyser les données.
23.03.21	Réunion pour parler du travail de bachelor d'un étudiant en master et de mon travail de bachelor. Lecture et recherche sur le machine learning et les réseaux de neurones convolutifs.
24.03.21	Regarder plus de théorie sur les réseaux de neurones convolutifs et commencer à se familiariser avec Keras
25.03.21	Commencer à faire un dataset qui peut-être utiliser avec Keras
26.03.21	Faire un dataset des données satellites en json, premiers essais avec Keras sur les données, essayer d'ajouter de l'augmentation de données avec ImageDataGenerator, essayer d'équilibrer les classes avec compute_class_weight
27.03.21	Avancer sur le rapport, commence à explique les réseaux de neurones artificiels
29.03.21	Continuer à essayer avec des hyperparamètres différents, essais avec jusqu'à 5000 epochs, avancer sur le rapport, réussir à lancer le notebook sur google colab (mais moins rapide actuellement que sur ma machine)
30.03.21	Réunion avec Prof. Perez-Uribe et un étudiant en master, essai d'utiliser flow_from_directory de Keras sans succès pour le moment (PIL ne supporte pas les images en float64)
31.03.21	Améliorer la structure du projet
12.04.21	Implémentation de la validation croisée, avancer sur l'explication des réseaux de neurones dans le rapport
13.04.21	Réunion, finir l'explication des neurones artificiels et des réseaux de neurones, faire la cross validation plusieurs fois pour valider les modèles, réaliser une matrice de confusion sommant celle de chaque validation
15.04.21	Avancer le rapport, calcul de la précision, du rappel et du f-score à partir de matrices
16.04.21	Tester quelles combinaisons de canaux de fréquences donnent les meilleurs résultats
17.04.21	Tester quelles combinaisons de canaux de fréquences donnent les meilleurs résultats, tester la meilleure combinaison de canaux sur 5000 epochs (11x11px)
19.04.21	Tester la meilleure combinaison de canaux sur 5000 epochs (9x9px)
20.04.21	Réunion, validation croisée en prenant en compte la situation géographique des labels

23.04.21	Avancer sur le rapport
26.04.21	Avancer sur le rapport (notamment finir l'explication des réseaux de neurones convolutifs), Analyser l'impact du sur-ajustement sur les performances de classifications en regardant périodiquement la différences de performances de classification Après un certain nombre d'epochs.
27.04.21	Avancer le rapport, assister à distance à une conférence de l'ISPRS (qui a du être interrompue à cause de participants empêchant le bon déroulement de la conférence)
28.04.21	Avancer le rapport, faire différents tests d'architecture de CNN
29.04.21	faire différents tests d'architecture de CNN
01.05.21	Avancer le rapport, et continuer à faire des tests d'architecture de CNN
04.05.21	Avancer le rapport, réunion, continuer à faire des tests d'architecture de CNN (réduire la couche entièrement connectée, essais avec pooling, différent essais avec augmentation des données)
06.05.21	tests d'architecture et reporter les résultats des performances des architectures dans un tableur
07.05.21	Avancer le rapport, continuer les tests d'architectures
08.05.21	Ajouter un «callback» pour calculer le f-score après chaque epochs et pouvoir l'utiliser avec l'early stopping et imprimer un graphe du f-score
11.05.21	Réunion, continuer le rapport, commenter le code, lancer des entraînements de modèles avec de nouvelles architectures
12.05.21	Correction de fautes de français dans le rapport, commenter le code,
14.05.21	Réaliser la bibliographie avec Zotero, écrire la conclusion, relecture
15.05.21	Finaliser le rapport intermédiaire