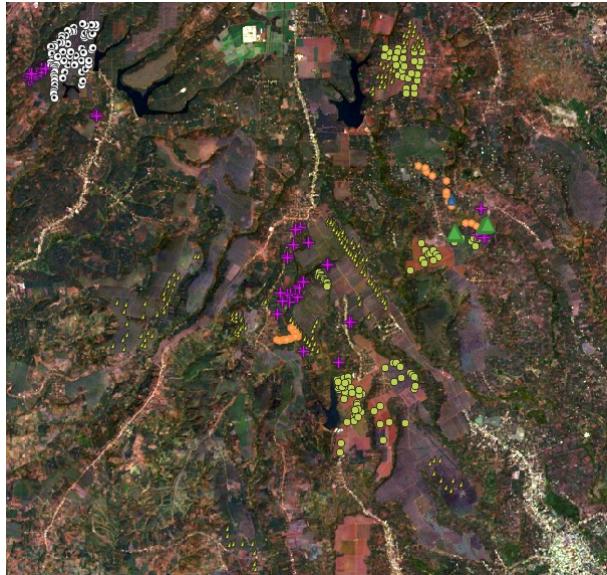


# Travail de Bachelor

Une IA pour surveiller les engagements de zéro déforestation

Non confidentiel



**Étudiant :**

**Simon Walther**

**Travail proposé par :**

Louis Reymondin

CIAT - International Center for Tropical Agriculture

Vietnam

**Enseignant responsable :** Andres Perez-Uribe

**Année académique :** 2020-2021

Yverdon-les-Bains, le 29 juillet 2021

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

---

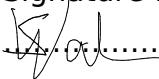
Centre International pour l'Agriculture Tropical

### Résumé publiable

Le Vietnam est le second plus gros producteur de café après le Brésil. De grandes multinationales ont signé des accords de zéro-déforestation, mais il leur est difficile de savoir si leurs fournisseurs de café au Vietnam cultivent des terrains qui ont été déforestés il y a peu. C'est là où intervient ce travail de bachelor : permettre de surveiller la déforestation et les changements d'utilisations des sols avec des outils de machine learning et des images satellites. Les images satellites ont été capturées par le satellite Landsat 8 ce qui a permis d'avoir des images remontant jusqu'à 2013, au prix d'une résolution plus faible que certains satellites plus récents.

Du point de vue pratique, il faut donner une grande quantité d'images satellites annotées par le CIAT à un réseau de neurones convolutif pour l'entraîner à reconnaître les différents types d'utilisation du sol et en particulier le café. Par ce biais, le réseau de neurones va apprendre les caractéristiques démarquant les diverses classes. On pourra valider les performances du modèle en fournissant des images dont on connaît le contenu et en vérifiant que le réseau prédise correctement la classe qui s'y trouve.

Dans un premier temps il a fallu préparer les images satellites pour que leur format soit utilisable par le réseau de neurones. Après cela, une analyse exploratoire des données a été réalisée. Ensuite, il a fallu entraîner des réseaux de neurones et rechercher des améliorations de performances en essayant différentes techniques et en choisissant judicieusement les paramètres. Enfin, des représentations graphiques ont été générées à partir des prédictions du réseau de neurones. Entre autres, des représentations permettent de visualiser les zones les plus à risque d'avoir été déforestées.

Étudiant :	Date et lieu :	Signature :
Walther Simon	Aigle, le 29 juillet 2021	 Signature :
Enseignant responsable :	Date et lieu :	Signature :
Perez-Uribe Andres	.....	.....
International Center for Tropical Agriculture :	Date et lieu :	Signature :
Reymondin Louis	.....	.....

# 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 29 juillet 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 29 juillet 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.....	9
2 Contexte.....	10
3 Abréviations.....	11
4 Landsat 8.....	12
5 Introduction aux réseaux de neurones convolutifs.....	13
5.1 Neurones artificiels.....	13
5.2 Réseaux de neurones.....	15
5.3 Réseaux de neurones convolutifs.....	16
5.3.1 Couche de convolution.....	16
5.3.2 Couche de sous-échantillonnage.....	17
5.3.3 Couche entièrement connectée.....	17
5.3.4 Architecture.....	18
5.4 Conclusion.....	18
6 Technologies utilisées.....	19
6.1 Ee.....	19
6.2 Keras.....	19
6.3 Rasterio.....	19
6.4 Tifffile.....	19
6.5 Shapely.....	19
6.6 Scikit-learn (Sklearn).....	19
6.7 Albumentations.....	19
6.8 Numpy.....	19
6.9 Pandas.....	19
6.10 Geopandas.....	20
6.11 Matplotlib.....	20

6.12 Seaborn.....	20
6.13 Math.....	20
6.14 Time.....	20
6.15 Os.....	20
6.16 Sys.....	20
6.17 Glob.....	20
6.18 Geojson.....	20
6.19 Spacv.....	20
6.20 Jupyter.....	21
6.21 Ipywidgets.....	21
7 Traitement des données.....	22
7.1 Obtention des données.....	22
7.2 Points géoréférencés.....	22
7.2.1 Jeux de données de Landsat 8.....	24
7.3 Préparation des données.....	25
8 Analyse exploratoire des données.....	27
8.1 Comparaison du café à d'autres labels.....	27
8.2 Comparaison du café par saison.....	28
8.3 Comparaison du café tous les deux mois.....	29
8.3.1 Contexte.....	29
8.3.2 Évolution.....	29
8.3.3 Variabilité.....	30
8.4 Comparaison du café par région.....	30
8.5 Comparaison par district.....	31
8.6 Comparaison du café par types de sols.....	31
8.7 Proportion moyenne de nuage.....	32
9 Entraînement des modèles.....	33
9.1 Validation croisée k-blocs.....	33
9.2 Augmentation des données.....	33
9.3 Arrêt prématué.....	34
9.4 Équilibrage des classes.....	34
9.5 Premiers réseaux et améliorations.....	34
9.6 Comparaison de combinaisons de bandes de fréquences.....	35
9.7 Comparaison d'architectures de CNN.....	37
9.7.1 Couches avancées.....	37
9.7.1.1 Dropout.....	37
9.7.1.2 Spatial dropout.....	38
9.7.1.3 Global average pooling.....	38

9.7.1.4 Convolution avec un saut de deux.....	38
9.7.2 Résultats.....	38
9.8 Seconde comparaison de combinaisons de bandes de fréquences.....	40
9.8.1 Résultats.....	41
9.9 Validation spatiale.....	42
9.9.1 Résultats.....	42
9.10 Entraîner sur une année et valider sur une autre.....	44
9.10.1 Résultats.....	44
9.11 Modèle avec tous les labels.....	45
9.11.1 Résultats.....	45
9.12 Modèle à plusieurs sorties.....	47
9.12.1 Résultats.....	47
9.13 Méta-apprentissage avec Reptile.....	50
9.13.1 Résultats.....	51
9.14 Modèle final.....	53
9.14.1 Prédictions.....	53
9.14.2 Zones à risque de déforestation.....	55
10 Organisation du travail.....	60
11 Scripts.....	61
11.1 Structure du projet.....	61
11.2 Notebooks Jupyter.....	62
11.3 Code source python.....	62
11.3.1 convNetUtils.py.....	64
11.3.2 rasterUtils.py.....	70
11.3.3 regionUtils.py.....	74
11.3.4 labelsUtils.py.....	75
11.3.5 bandUtils.py.....	78
11.3.6 statisticsUtils.py.....	78
11.3.7 visualizationUtils.py.....	79
11.3.8 downloadMap.py.....	81
11.3.9 mergeRasterFiles.py.....	82
11.3.10 config.py.....	82
12 Conclusion.....	83
13 Remerciements.....	84
14 Annexe.....	91

## 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 déforestation 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 km<sup>2</sup> et le nombre de petits agriculteurs est 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] :

Bandé	Fréquence [ $\mu\text{m}$ ]	Résolution
1. Coastal aerosol	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. Tirs-1	10.6-11.2	100 m
11. Tirs-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 :

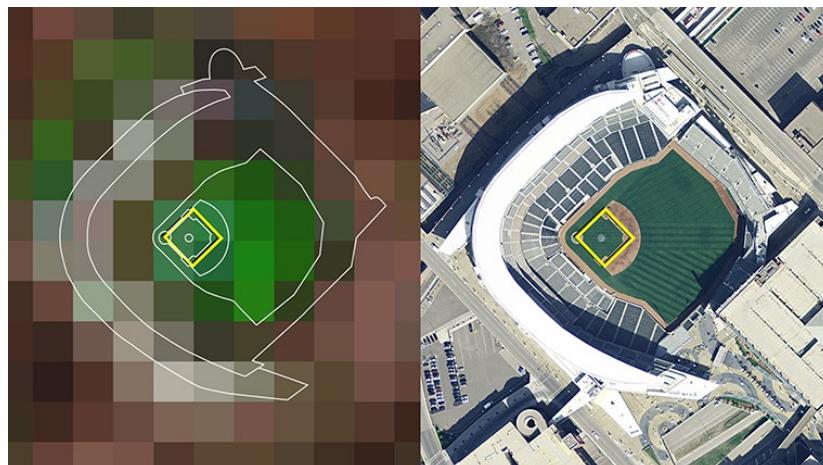


Figure 2: Comparaison entre une photo aérienne et une image Landsat,  
<https://landsat.gsfc.nasa.gov/article/picturing-pixel>

## 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].

### 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  $x_1, x_2, \dots, x_n$  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  $\theta$ , sinon à zéro [11].

Ce qui donne le calcul suivant :

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

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

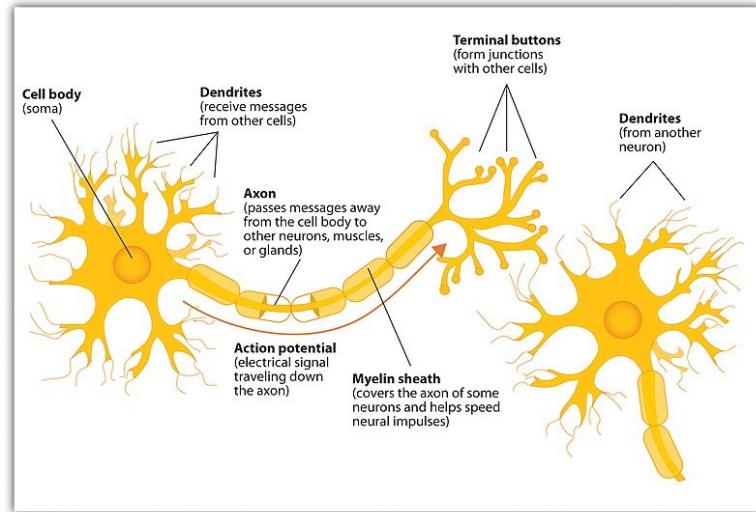


Figure 3: Représentation des composants d'un neurone,  
[https://en.wikipedia.org/wiki/File:Components\\_of\\_neuron.jpg](https://en.wikipedia.org/wiki/File:Components_of_neuron.jpg)

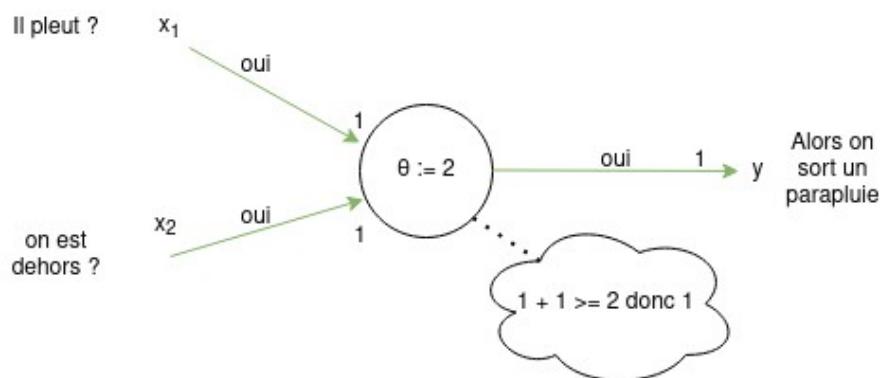


Figure 4: Exemple de condition booléene AND avec un neurone McCulloch-Pitts

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 :

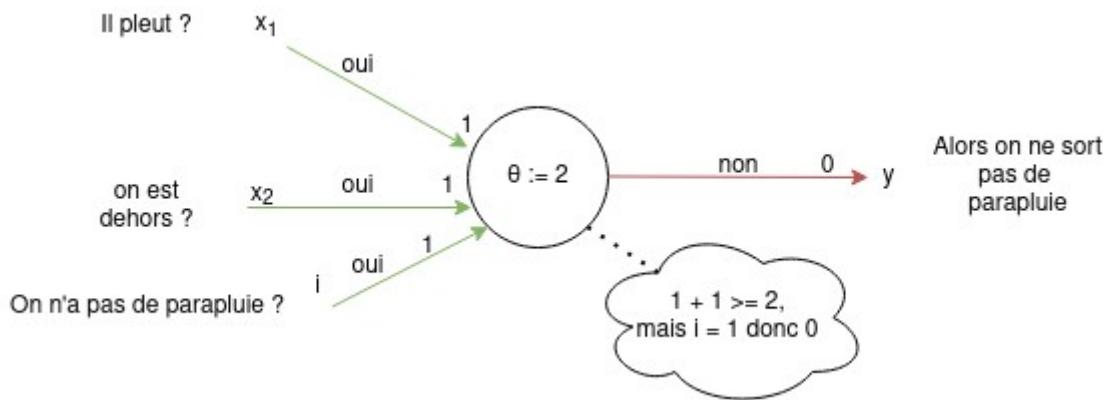


Figure 5: Exemple d'entrée inhibitrice avec un neurone McCulloch-Pitts

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  $\theta$  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 :

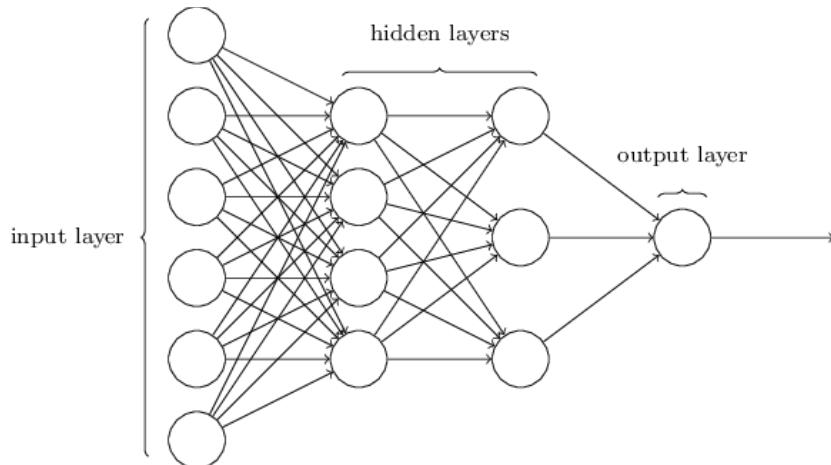
$$\begin{cases} 1 & \text{si } \left( \sum_{k=1}^n w_k x_k \right) + b \geq 0 \\ 0 & \text{sinon} \end{cases}$$

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

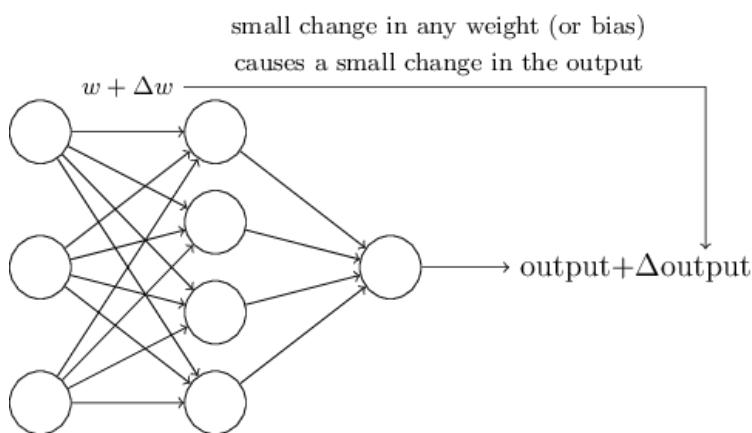
Voilà à quoi ressemble un réseau de neurone :



*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 engende 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'apprendre 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 (*fully 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.

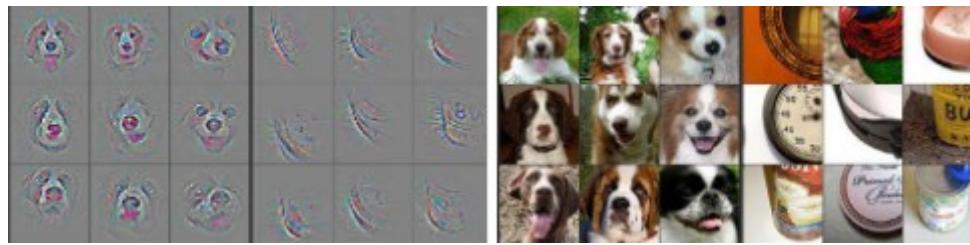
Une convolution utilise un filtre qui va parcourir toute l'image et effectuer un produit matriciel à chaque position où se déplace le filtre. En traitement d'image, ce filtre peut par-exemple permettre de flouter une image, ou détecter les bords dans une image [15].

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.

Filtre		
0	1	0
1	-4	1
0	1	0



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](https://commons.wikimedia.org/wiki/File:Front_view_of_a_resting_Canis_lupus_ssp.jpg)



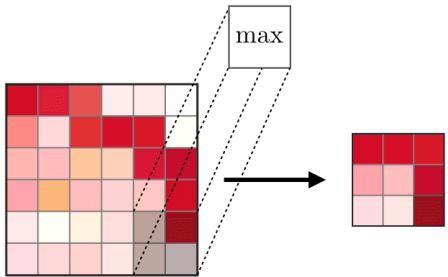
*Figure 9: Exemple de caractéristiques reconnues par des filtres à la 4ème couche d'un réseau convolutif. Extrait de "Visualizing and Understanding Convolutional Network", <https://arxiv.org/pdf/1311.2901.pdf>*

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  $2 \times 2$  [14] et en gardant chaque fois une valeur par zone. Il y a plusieurs façons 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].

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.



*Figure 10: Exemple de sous-échantillonnage prenant la valeur maximale d'un filtre de  $2 \times 2 \text{px}$ , <https://stanford.edu/~shervine/l1/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels>*

### 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 :

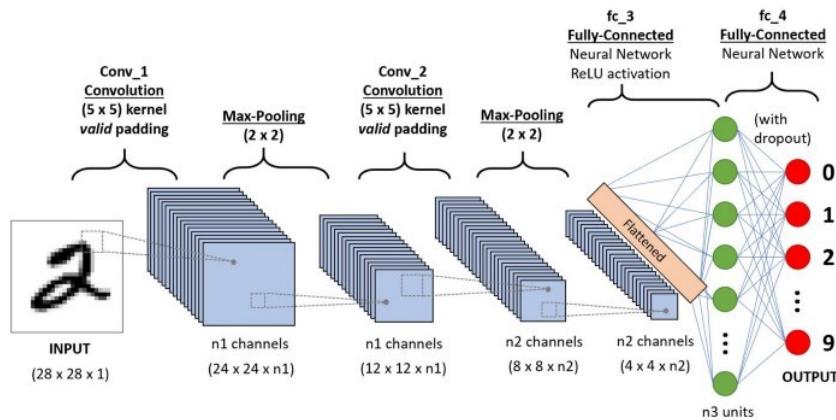


Figure 11: Exemple d'une architecture de réseau de neurones convolutif pour reconnaître des chiffres,

<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 Tifffile

Tifffile est une librairie python qui permet de lire et écrire des fichiers de raster [23]. Cette librairie est utilisée pour lire les images durant l'entraînement, car elle s'est révélée plus rapide que rasterio et qu'elle permettait de lire des fichiers de rasters de plus de quatre canaux.

### 6.5 Shapely

Shapely est une librairie permettant de travailler avec des objets géométriques, comme définis par librairie GEOS [24], et faire des analyses dans l'espace [25,26].

### 6.6 Scikit-learn (Sklearn)

Scikit-learn est une librairie pour le machine learning [27]. 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.

### 6.7 Albumentations

Albumentations permet de faire de l'augmentation d'images [28]. L'avantage d'utiliser Albumentations plutôt que l'augmentation d'images de Keras dans ce projet est que qu'Albumentations est plus rapide et permet de prendre en charge des images avec plus que quatre canaux.

### 6.8 Numpy

Numpy est une librairie qui nous permet de faire des calculs et des opérations sur des matrices [29].

### 6.9 Pandas

Pandas est un outil permettant de faire de l'analyse et de la manipulation de données [30].

## 6.10 Geopandas

Geopandas se base sur Pandas et étend les structures de données que Pandas propose pour pouvoir traiter des données géospatiales [31].

## 6.11 Matplotlib

Cette librairie permet de créer différentes sortes de graphiques et de visualisations de données [32].

## 6.12 Seaborn

Seaborn est une librairie de visualisation de données basée sur Matplotlib, on l'utilise ici pour afficher les boîtes à mustaches [33].

## 6.13 Math

Math met à disposition des fonctions mathématiques [34].

## 6.14 Time

Time est une librairie pour traiter des données temporelles et propose des fonctions qui ont rapport au temps [35]. 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.

## 6.15 Os

Os est un module python qui donne la possibilité d'utiliser des fonctionnalités qui dépendent des systèmes d'opérations, comme par exemple la construction de chemins d'accès vers des fichiers, sans devoir se soucier de quel système d'exploitation est utilisé [36].

## 6.16 Sys

Sys est un module permettant d'interagir avec des variables et fonctions de l'interpréteur Python [37]. Dans ce projet, il va permettre de demander à l'interpréteur de considérer le dossier «src» comme contenant des modules pouvant être importés.

## 6.17 Glob

Glob permet de récupérer tous les chemins vers des fichiers qui correspondent à une règle [38].

## 6.18 Geojson

Cette librairie permet d'encoder et de décoder des fichiers GeoJSON et propose des classes pour chaque objet de la spécification GeoJSON [39,40].

## 6.19 Spacy

Spacy 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 [41].

## 6.20 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 [42].

## 6.21 Ipywidgets

Ipywidgets est un module qui permet d'ajouter des moyens d'interagir avec un «notebook» Jupyter [43]. On peut par exemple ajouter des boutons ou des champs texte.

## 7 Traitement des données

### 7.1 Obtention des données

Les données du satellite Landsat 8 sont récupérées à partir de Google Earth Engine [44].

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 [45]. 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.

### 7.2 Points géoréférencés

D'après les informations que nous avons, les données de «central highland» du Vietnam devraient avoir été créés entre 2017 et 2018 pour les points les plus vieux et entre 2019 et 2020 pour les points les plus récents.

Un jeu de données mis-à-jour, qui spécifie les années des points géoréférencés, qui apporte des données supplémentaires (environ quatre cent pour 2018) et avec quelques changements dans les labels (notamment «végétation naturelle» qui est renommé en «forêt dense»), a été reçu vers la fin du travail de bachelor.

En tout en 2018 c'est un peu plus de quinze mille points géoréférencés dont plus de quatre mille points labellisés comme du café.

Des points d'autres régions ont aussi pu être obtenus en fin de travail de bachelor : Sumatra sud (en Indonésie), Sumatra centre (en Indonésie), Célèbes (ou Sulawesi, en Indonésie), Bornéo, Pará centre et Pará sud (au Brésil), Ocotepeque (au Honduras) et Ghana.

Tous ces jeux de données n'ont pas forcément les mêmes labels présents dans leurs données. Toutefois, les labels sont désignés de la même façon entre chacune de ces régions.

Les labels de 2018 qui se trouvent au moins 5 exemplaires au Vietnam sont : café, forêt dense, caoutchouc, agriculture saisonnière, urbain, eau, autres arbres, nature sans arbre, poivre, thé, riz, forêt décidue, pins, brousse & fruticée (shrubland bushland), arbre épargné (spare tree), pâturages, forêt secondaire dégradée, mine sol visible (mine baresoil), bâton pour poivre, poivre et café, poivre et autres, culture mélangée (intercrop).

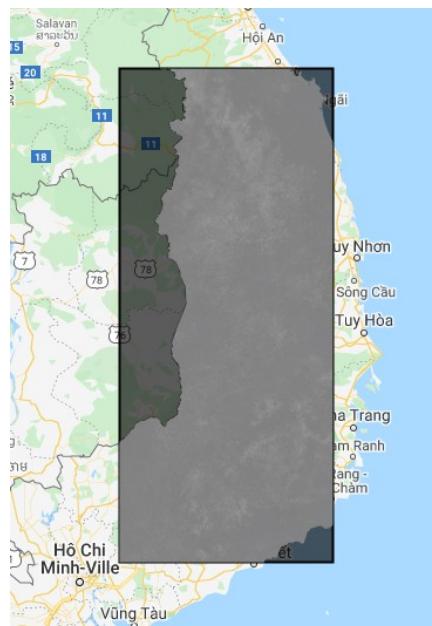


Figure 12: Visualisation de la zone récupérée, on récupère des images dans les limites du Vietnam qui se situent dans le rectangle représenté en gris



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.

## 7.2.1 Jeux de données de Landsat 8

Il y a plusieurs jeux différents de données de Landsat 8 sur Google Earth Engine. Le premier qui a été utilisé dans ce travail de bachelor est «USGS Landsat 8 Surface Reflectance Tier 1». Certains des derniers notebooks réalisés utilisent «USGS Landsat 8 Level 2, Collection 2, Tier 1» qui est une nouvelle version du jeu de données précédemment cité.

Ces deux jeux de données ont pour avantage de pré-traiter les données en tenant compte d'effets atmosphériques [46]. Par contre, les canaux suivants en sont absents : Panchromatic et Cirrus. Avec la collection 2, il n'y a qu'une seule bande de fréquence pour la température.

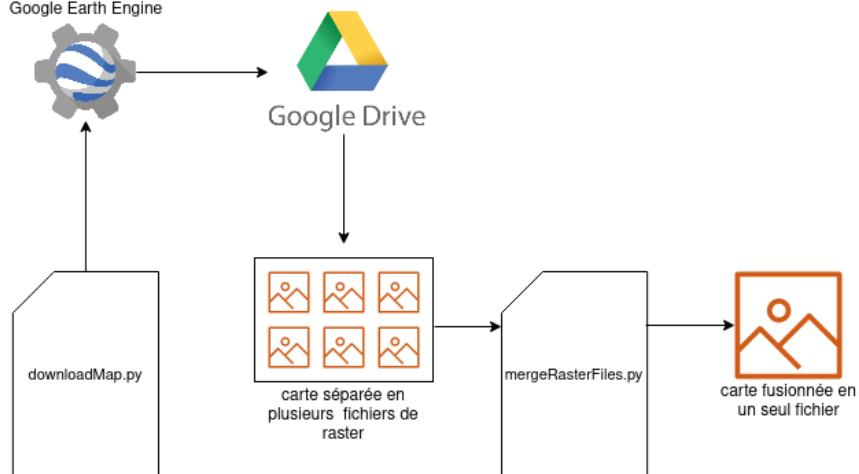
Ces deux collections fournissent un masque appelé «pixel\_qa» ou «QA\_PIXEL» qui permet, entre autres, de masquer les nuages et les ombres des nuages.

Il est conseillé de mettre à l'échelle leurs données avant de les utiliser. Pour la collection 1 il faudra multiplier par un facteur de 0.0001, et pour la collection 2 on multipliera par 0.0000275 et on soustraira 0.2 [47].

## 7.3 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.



Ces coordonnées de labels vont *Figure 14: Schéma de la récupération et fusion des données* 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.

Voici un exemple de ce à quoi peuvent ressembler les images qui sont utilisées :



Figure 16: Comparaison des images de différents labels

On peut constater que la différence peut être relativement difficile à faire à l'œil nu. Il faut néanmoins garder à l'esprit que le réseau de neurones ne va pas forcément utiliser que ces trois canaux rouge, vert et bleu, mais pourra profiter des données d'autres canaux.

## 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 [45].

### 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.

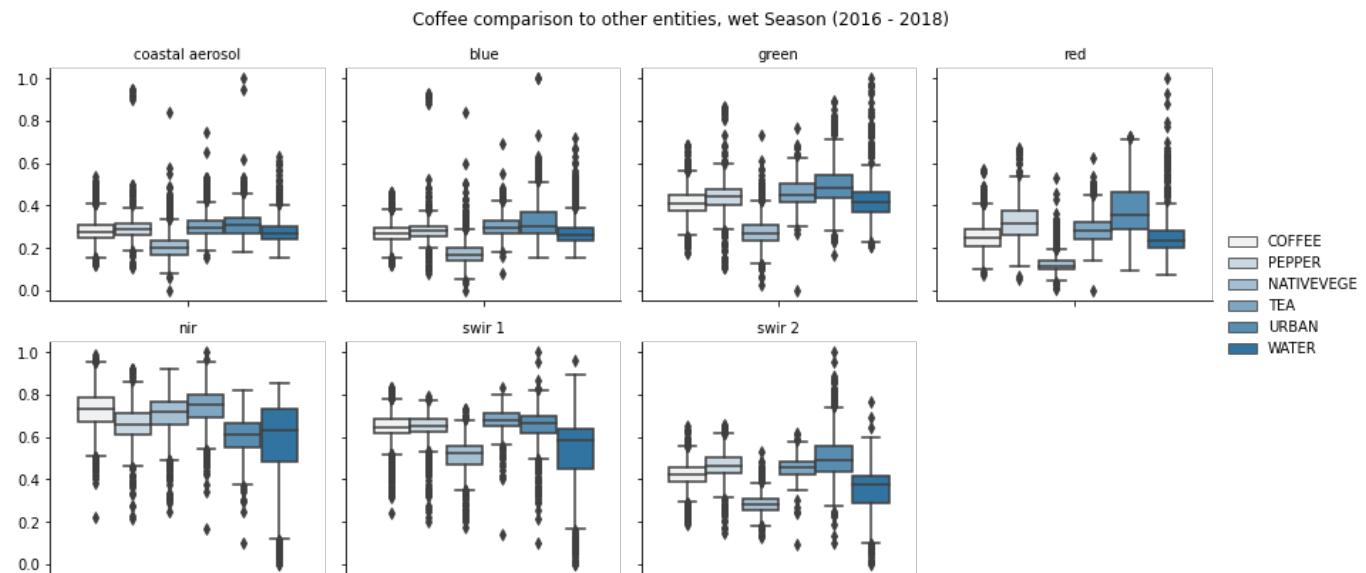


Figure 17: Comparaison du café au poivre, à la végétation naturelle, au thé, aux zones urbaines et à l'eau en saison humide

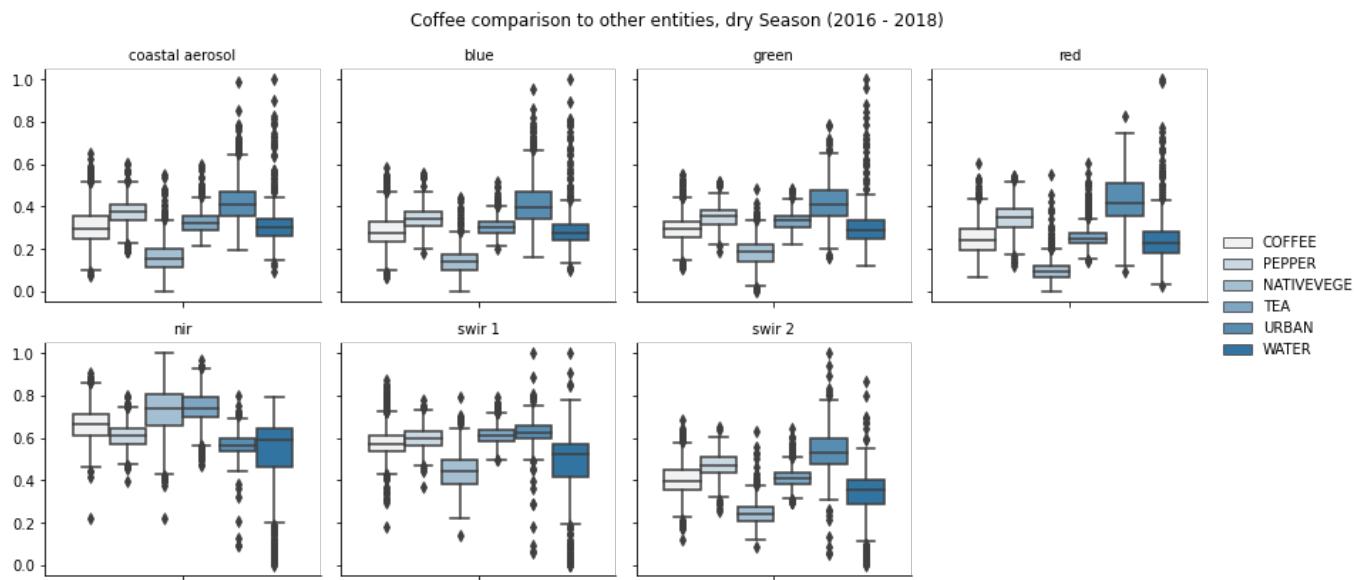


Figure 18: Comparaison du café au poivre, à la végétation naturelle, au thé, aux zones urbaines et à l'eau en saison sèche

## 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*.

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* et *swir2* qui baissent légèrement en saison humide.

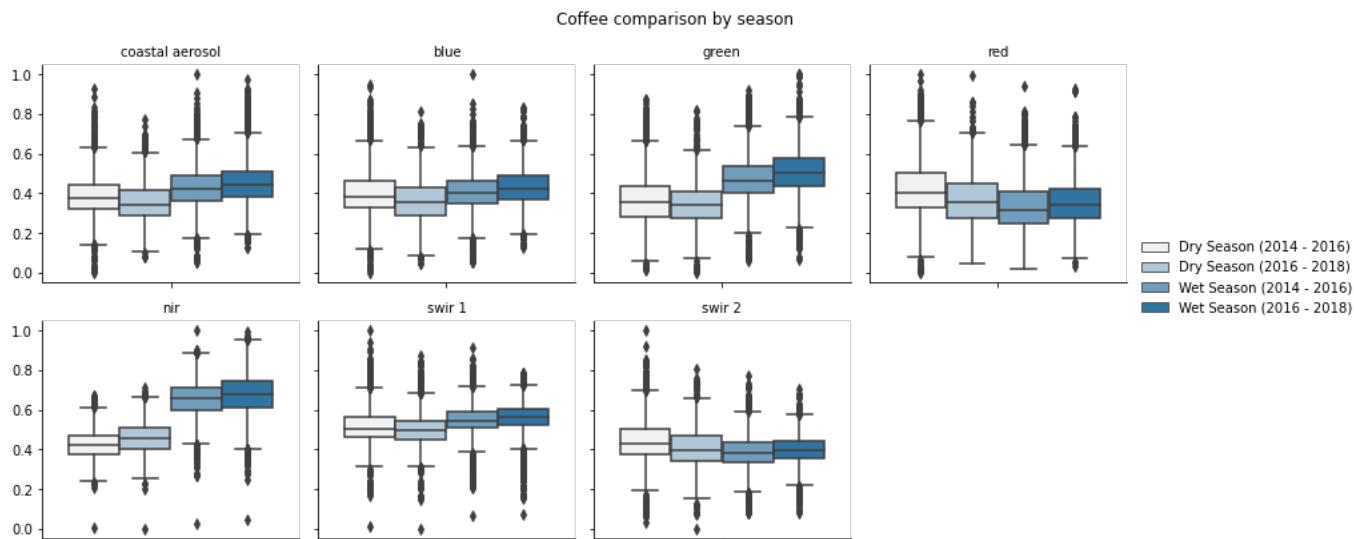


Figure 19: Comparaison des valeurs du café par saison en prenant les années 2014 à 2016 et 2016 à 2018

## 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 [48] et prend fin de décembre à janvier [49].

Après la récolte, les arbres à café sont taillés pour laisser passer plus de lumière ; et c'est de janvier à avril où le cafier est fortement arrosé [49].

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.

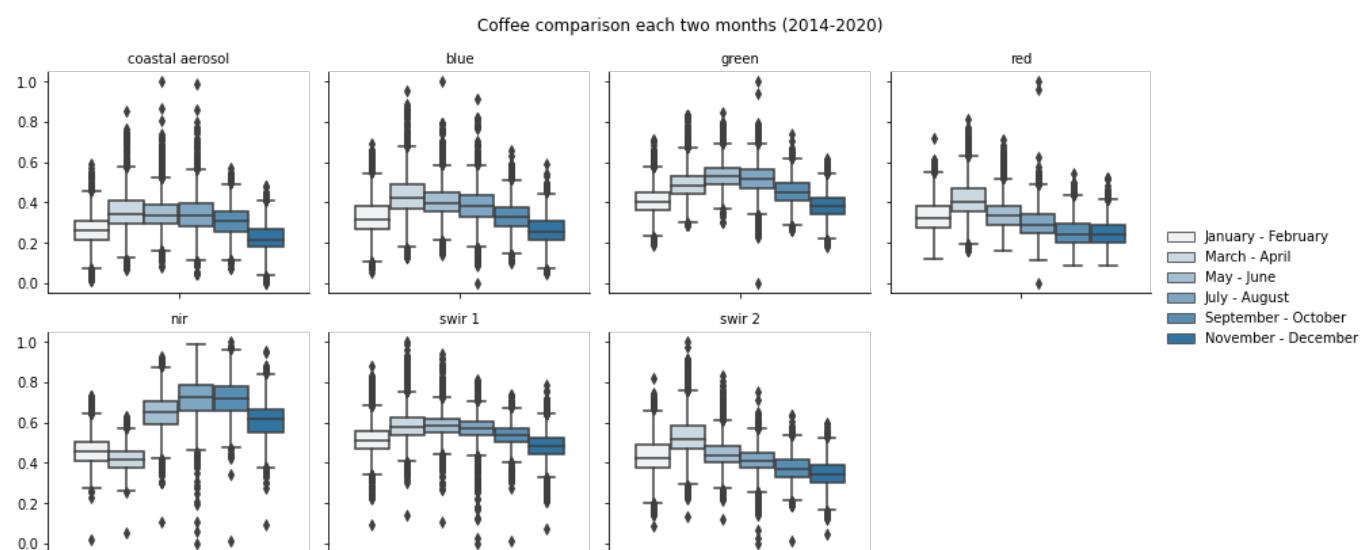


Figure 20: Comparaison du café deux mois par deux mois avec les données de 2014 à 2020

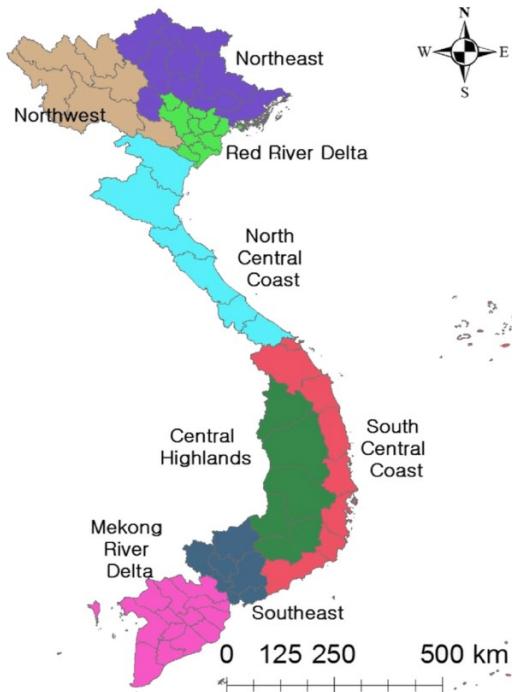
Le canal *green*, quant à lui, est à son plus haut en mai et juin, et en juillet et août.

### 8.3.3 Variabilité

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

- Septembre à octobre : coastal aerosol, blue, green, swirl1, swirl2
- 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'occurrence 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 la bande de fréquences *coastal aerosol*.

Dans ces deux régions, le café se distingue principalement par les canaux *red*, *coastal aerosol* et *blue*. Les différences de médianes étant de 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 21: Carte avec les régions du Vietnam,  
<https://www.researchgate.net/figure/ListOf-regions-in->

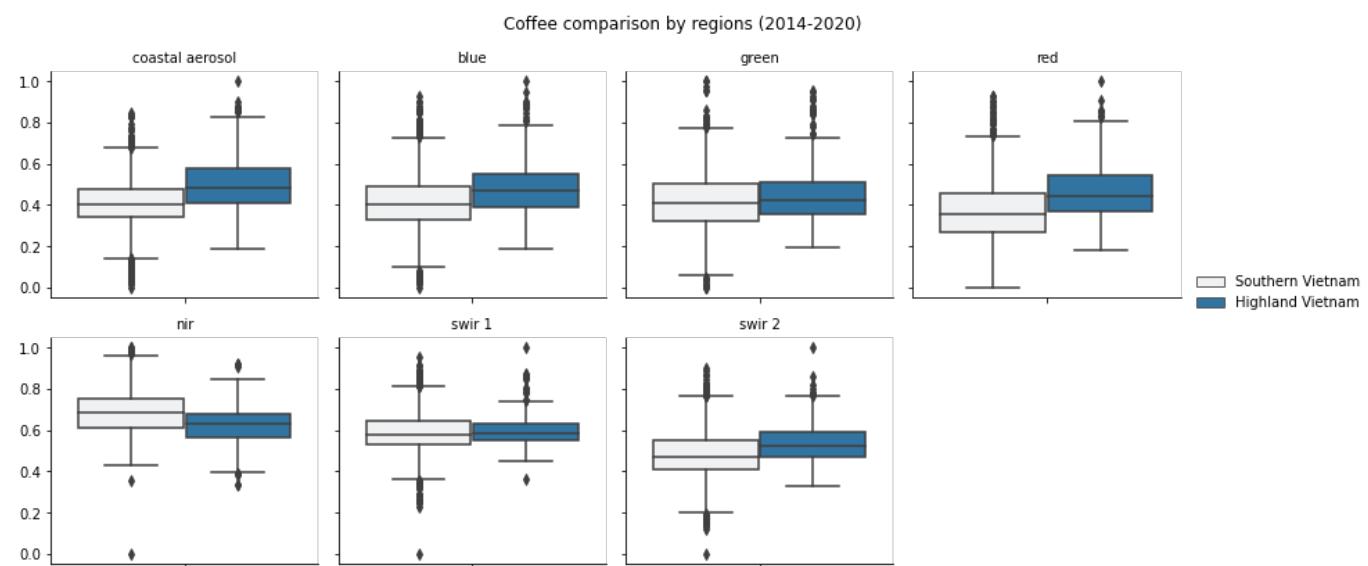


Figure 22: Comparaison du café par régions avec les valeurs du café de 2014 à 2020

## 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, Đăk Lăk, Đăk Nông et Lâm Đồng.

Les frontières des districts ont été récupérées à partir de données ouvertes [50].

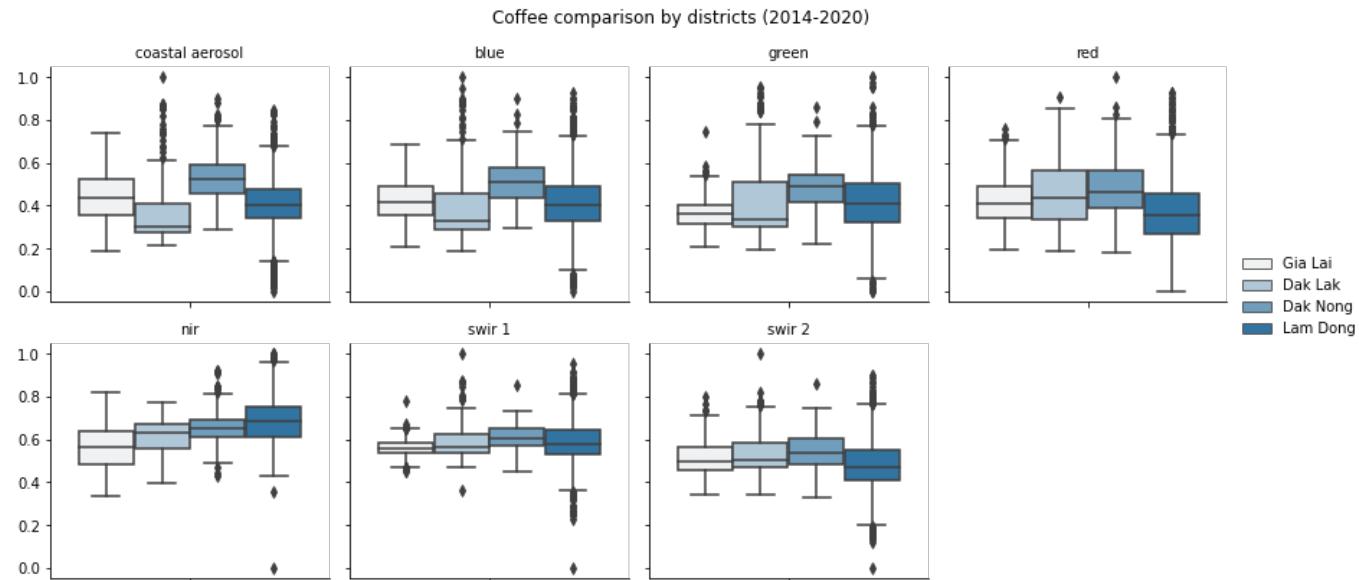


Figure 23: Comparaison des valeurs du café par district

## 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 [51].

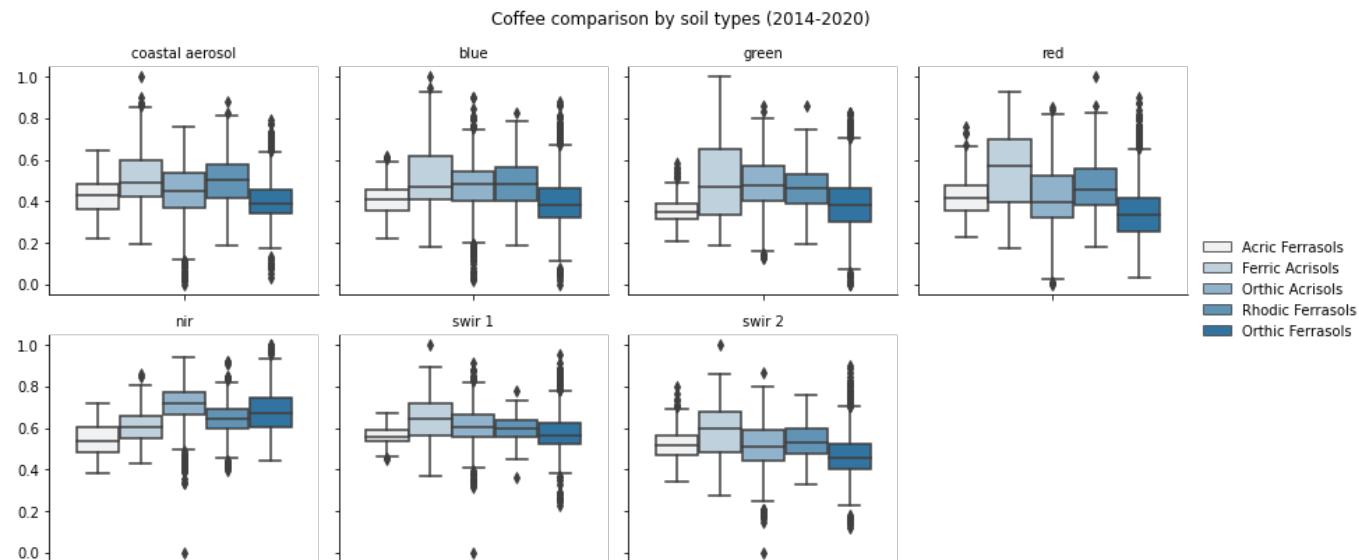


Figure 24: Comparaison des valeurs du café par types de sols

## 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	<b>23.30%</b>
Mai-juin	37.40%
Juillet-août	<b>45.11%</b>
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 [52]. Si l'on utilise 5 blocs, on va chaque fois entraîner sur 80 % des données et valider avec 20 % des données.

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 [53].

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.

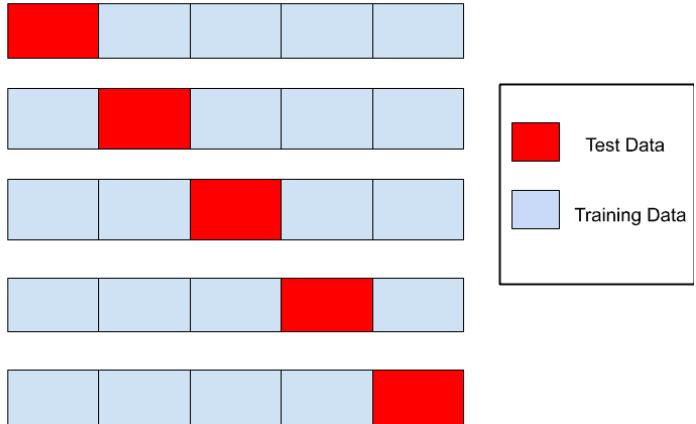


Figure 25: Représentation de la répartition des sous-ensembles avec la validation croisée,  
<https://www.mltut.com/k-fold-cross-validation-in-machine-learning-how-does-k-fold-work/>

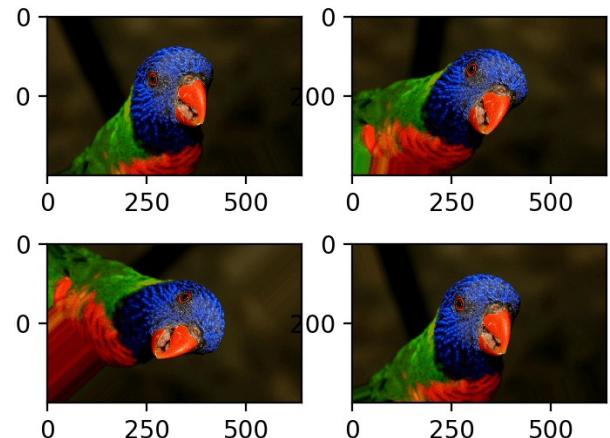


Figure 26: Exemple de rotations aléatoire d'image avec l'augmentation des données.  
<https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>

### 9.3 Arrêt prématué

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

On 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.

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.

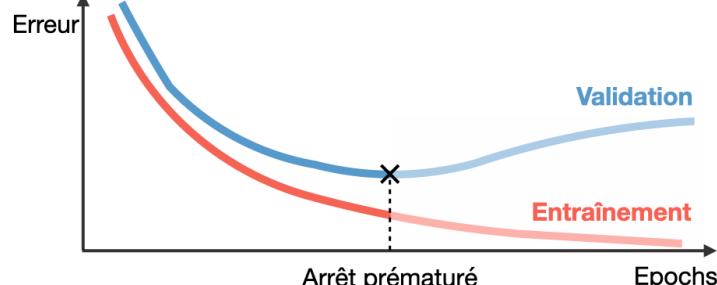


Figure 27: Démonstration de l'arrêt prématué avec un graphe de l'erreur,  
<https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-petites-astuces-apprentissage-profound#regularization>

### 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 [54].

### 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 [55].

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.

Les données qui ont été utilisées dans cette comparaison sont celles de janvier et février 2017. Pour choisir quelles combinaisons de canaux tester, il a d'abord été testé plusieurs combinaisons de trois canaux fréquemment utilisées [56,57].

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%	<b>46.27%</b>	<b>59.02%</b>
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	<b>0.943</b>	<b>61.87%</b>	<b>81.58%</b>	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.

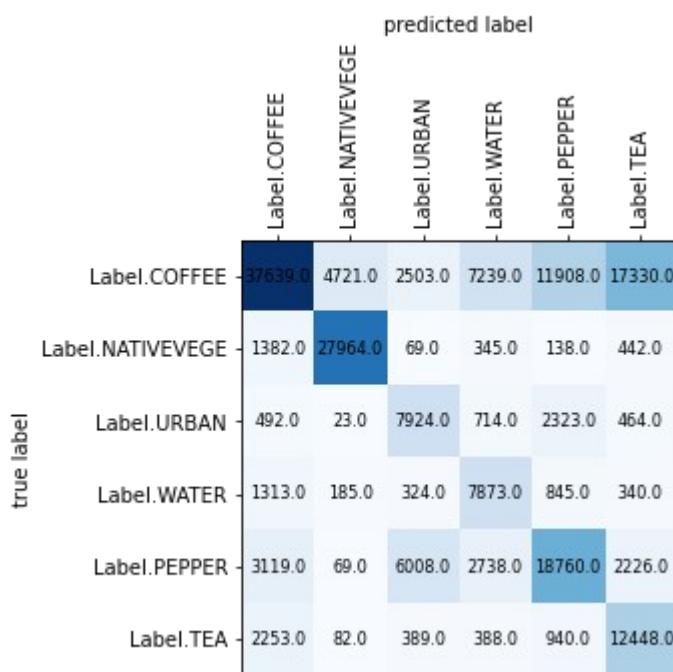


Figure 29: Matrice de confusion avec la combinaison NIR, RED, SWIR1

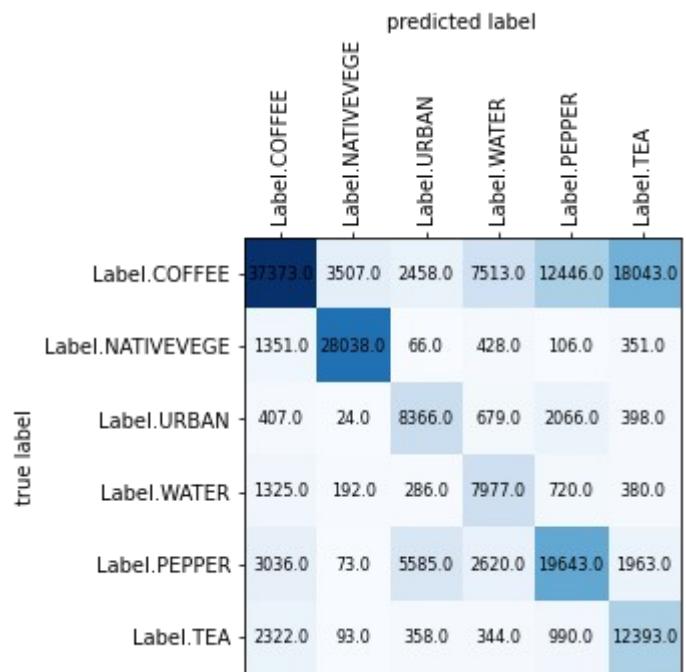


Figure 28: Matrice de confusion avec la combinaison NIR, RED, SWIR1, SWIR2

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

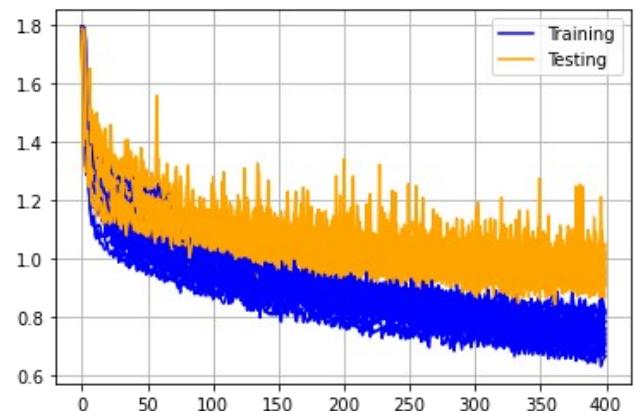


Figure 30: graphe de l'erreur avec la combinaison NIR, RED, SWIR1

## 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...).

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.

Les comparaisons d'architectures ont été effectuées sur les données de janvier et février 2017.

### 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 [58].

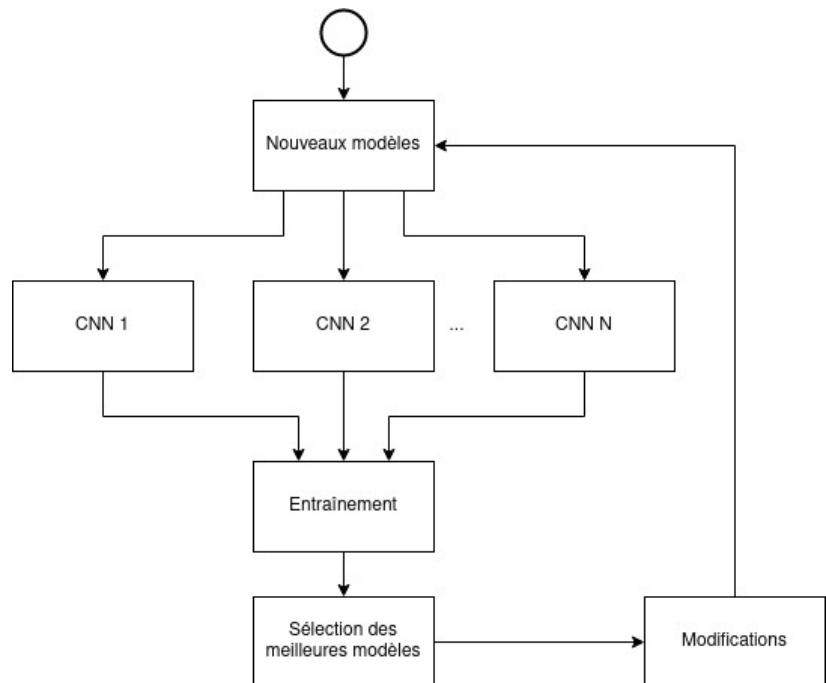


Figure 31: Processus pour décider quelles architectures tester

### 9.7.1.2 Spatial dropout

Le but du *spatial dropout* est le même que celui du décrochage : réduire le surajustement.

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 [58,59].

### 9.7.1.3 Global average pooling

À partir du constat que les couches entièrement connectées sont sujettes au surajustement, 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 [60].

Il est cependant possible d'ajouter une couche entièrement connectée après le *global average pooling*. C'est ce que fait le modèle ResNet-50 [61].

### 9.7.1.4 Convolution avec un saut de deux

Technique trouvée dans le papier «striving for Simplicity : The All Convolutonal Net» [62] 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.

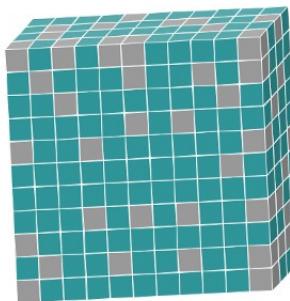
## 9.7.2 Résultats

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

Standard Dropout



Spatial Dropout

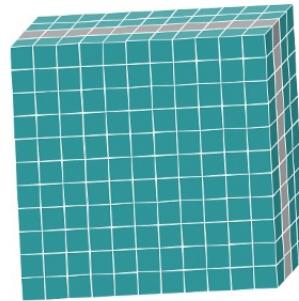


Figure 32: Différence d'approche entre le dropout et le spatial dropout, <https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293>

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.

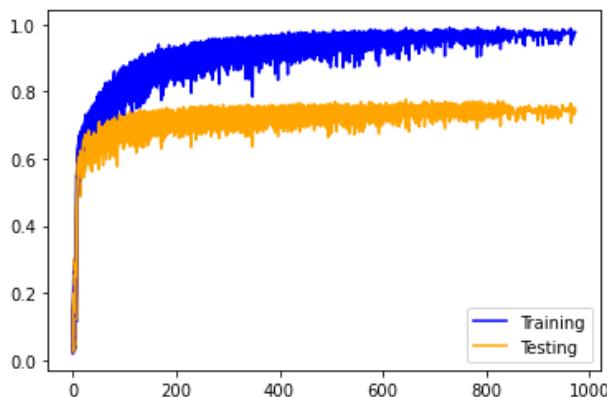


Figure 34: Graphe de l'évolution du f-score en fonction du nombre d'epochs

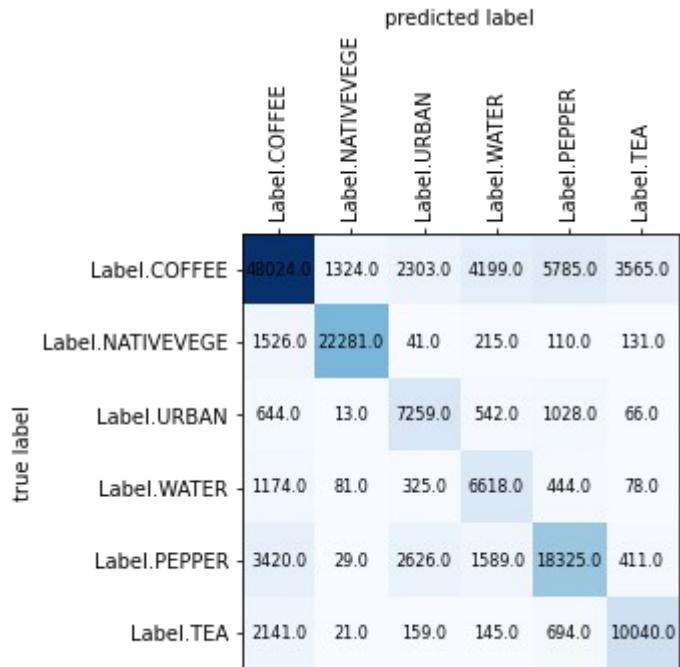


Figure 33: Matrice de confusion du modèle sélectionné

## 9.8 Seconde comparaison de combinaisons de bandes de fréquences

À la comparaison précédente, on utilisait jusqu'à quatre canaux. Une comparaison comportant plus de canaux a été réalisée ici pour voir si les résultats seraient meilleurs. L'augmentation de données de Keras ne permettant pas d'avoir plus que quatre canaux, la librairie Albumentations, qui permet de faire de l'augmentation de données avec plus que quatre canaux, est utilisée.

Un autre changement a été de modifier les paramètres de l'augmentation de données, notamment de ne plus utiliser une rotation de 45 degrés, mais des rotations de 90°.

Les mêmes données que celles de la comparaison d'architectures précédentes ont été utilisées.

Dans cette expérience, les données sont augmentées comme suit :

```
AUGMENTATIONS = Compose([
    HorizontalFlip(p=0.5),
    VerticalFlip(p=0.5),
    ShiftScaleRotate(
        shift_limit=0.0625,
        scale_limit=0,
        rotate_limit=0,
        p=0.8
    ),
    RandomRotate90(p=0.5),
])
```

Cela veut dire que l'on va faire prendre les images miroirs verticalement et horizontalement, que l'on va décaler les images et que l'on va tourner les images de 90° un nombre aléatoire de fois.

Les autres paramètres ont été définis comme ceci :

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)
Architecture	2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout
Arrêt prématué	Sur le f1-score, patience de 100

De plus, des canaux calculés ont été intégrés dans la comparaison. Ces canaux sont NDVI [63], MNDWI [64], EVI2 [65] et BU [66]. Ceux-ci n'ont malheureusement pas significativement amélioré les résultats. En effet, les améliorations que l'on peut noter en les ajoutant aux autres canaux, sans retirer les canaux qui sont utilisés dans le calcul, sont légères et au prix d'un nombre plus important de paramètres.

### 9.8.1 Résultats

Les performances se sont révélées bien meilleures en utilisant une grande partie des canaux qu'en en utilisant peu.

Pour comparaison, voici les résultats de la meilleure combinaison de trois bandes de fréquences, que l'on avait obtenue, celle comprenant tous les canaux sauf les canaux de températures, et celle comprend tout les canaux :

canaux	nb paramètres	précision moyenne	rappel moyen	f-score moyen	f-score du café
nir, red, swir1	74'674	68.00%	74.70%	70.68%	76.70%
coastal_aerosol, blue, green, red, nir, swir1, swir2	75'842	75.26%	79.44%	77.14%	81.68%
Tous les canaux	76'426	76.84%	80.31%	78.45%	82.90%

C'est cette dernière combinaison de bande de fréquences qui est la plus prometteuse : c'est celle qui obtient les meilleures performances si on exclut les canaux de températures.

En effet, prendre les canaux de températures peut légèrement améliorer les résultats, mais la température peut changer d'une année à l'autre. Il est donc important que notre modèle ne se fasse pas influencer par la température.

En essayant de rajouter des calculs calculés, l'amélioration est de seulement 0.12 % pour le f-score moyen en rajoutant les canaux mndwi, evi2 et bu. Par contre, cela rajoute 876 paramètres supplémentaires.

Dans le cas où l'on retire les canaux participant aux calculs des bandes mndwi, evi2 et bu, la performance chute, le fscore moyen passant de 76.58% à seulement 62.28%.

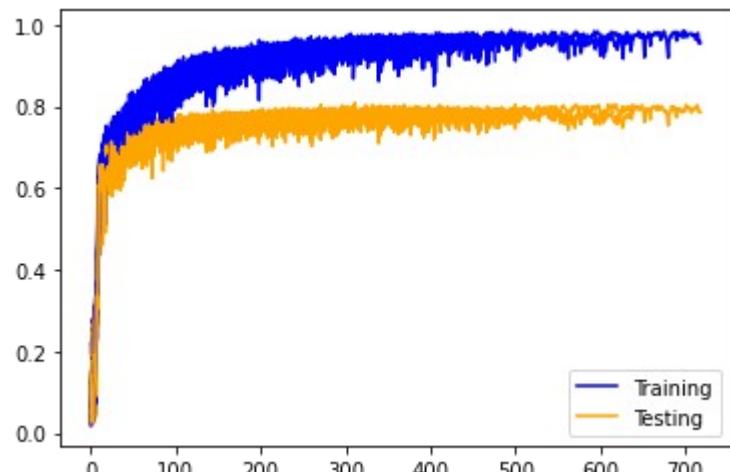


Figure 35: Évolution du f-score en fonction des epochs avec le modèle comprenant les canaux coastal\_aerosol, blue, green, nir, swir1 et swir2

## 9.9 Validation spatiale

Dans les expériences précédentes, la validation a été effectuée en prenant les images au hasard. Cette façon de faire peut donner des résultats trop optimistes, car des images situées à des endroits géographiquement très proches pourront alors se retrouver, certaines dans l'ensemble d'entraînement et d'autres dans l'ensemble de test. De par leur proximité elles risquent d'être très similaires et les résultats seront alors biaisés.

C'est pourquoi une expérience avec de la validation croisée spatiale a été réalisée. Dans cette expérience, on essaie que les points qui se trouvent dans l'ensemble de validation respectent une certaine distance avec les points de l'ensemble d'entraînement.

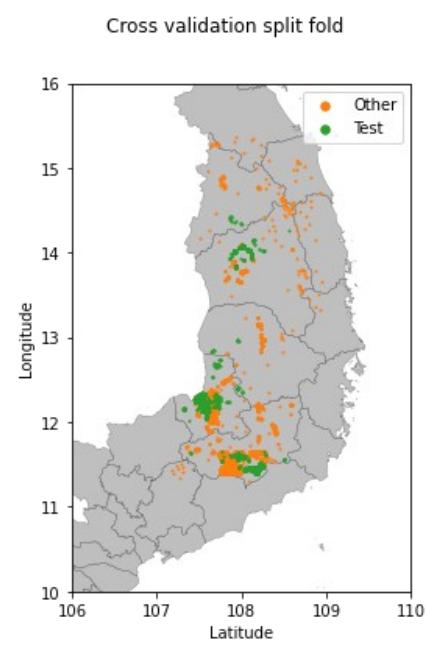
Pour effectuer la séparation des données en ensembles de tests, validations et entraînements, on va réaliser, pour chaque label, dix blocs (folds) devant être séparés par au moins 1km. Ensuite on va grouper deux à deux ces blocs, en ne gardant que leur ensemble de validation, pour en obtenir cinq. On se retrouve donc avec cinq groupes avec des images différentes et ayant de la distance les séparant. Un groupe parmi ces cinq fera office de groupe de test, les autres seront utilisés pour l'entraînement et la validation. Ces deux ensembles sont sauvegardés sur le disque. Au moment de l'entraînement, on peut recommencer ce processus pour séparer les images d'entraînement des images de validation.

La difficulté avec cette expérience est de trouver une manière de séparer convenablement les données tout en gardant à la fois de la distance entre les images et à la fois en s'assurant que les ensembles ne soient pas déséquilibrés par rapport à leur taille et à la quantité d'images de chaque classe qu'ils contiennent.

### 9.9.1 Résultats

Les paramètres qui ont été utilisés sont les suivants :

Distance minimale entre les ensembles d'entraînements et validations	1 km
Canaux	coastal_aerosol, blue, green, red, nir, swir1, swir2
Nombre de tests	4
Labels utilisés	Café, végétation naturelle (forêt dense), urbain, eau, poivre et thé
Nombre de pixels autour du label	4 (c.-à-d. une image de 9x9 pixels)
Architecture	2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout



Arrêt prématuré

Sur le f1-score, patience de 100

Et les résultats obtenus sont :

Avec validation spatiale ?	précision moyenne	rappel moyen	f-score moyen	f-score du café
Avec	56.02%	66.88%	59.35%	60.02%
Sans	75.26%	79.44%	77.14%	81.68%

Si l'on compare avec les résultats que l'on obtenait avec le même modèle et de la validation croisée avec séparation aléatoire on remarque, comme on s'y attendait, que la performance est bien moins bonne.

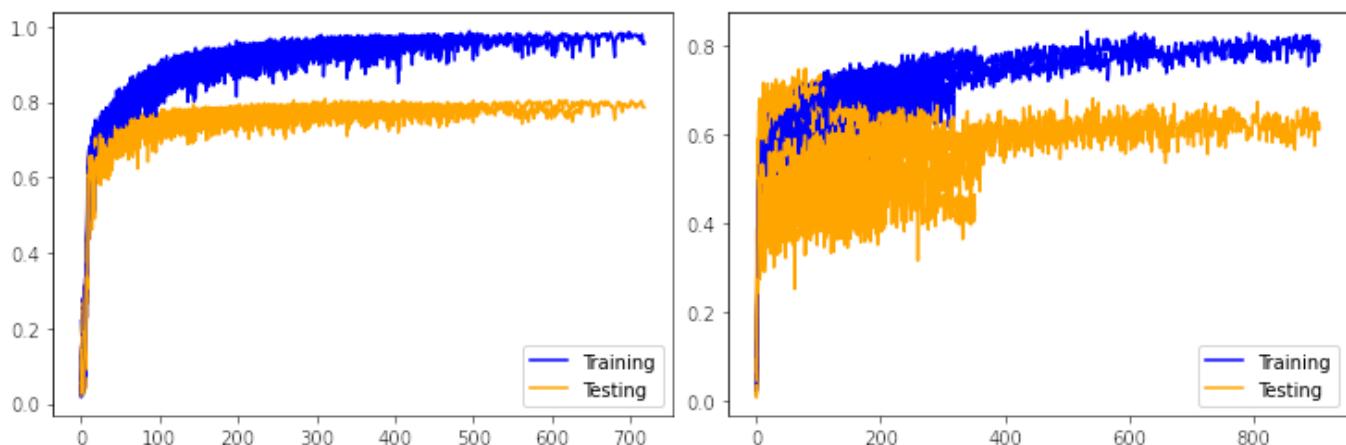


Figure 36: Comparaison de l'évolution du f-score avec le modèle qui utilise tous les canaux sauf ceux de température (à gauche) et le modèle avec de la validation spatiale (à droite)

## 9.10 Entraîner sur une année et valider sur une autre

Pour valider le modèle, une autre façon de faire qui a été essayée est d'entraîner un modèle avec toutes les données d'une année, ce qui permet d'avoir un maximum de données pour l'entraînement, et de valider avec les données d'une autre. Il faut par contre garder à l'esprit que les points se situent au même endroit et peuvent être assez similaires d'une année à l'autre. Il faut aussi noter que certaines erreurs peuvent avoir lieu si ce qui se trouve à un endroit a changé entre les deux années. Dans cette expérience on part du principe que les points n'ont pas changé entre temps.

### 9.10.1 Résultats

Dans cette expérience, on n'utilise pas d'arrêt prématué, car on ne valide pas pendant l'entraînement et qu'on n'a donc pas de mesure fiable pour savoir quand s'arrêter.

Nombre d'epochs	500
Canaux	coastal_aerosol, blue, green, red, nir, swirl1, swirl2
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)
Architecture	2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout
Arrêt prématué	Sans arrêt prématué

L'expérience a été réalisée avec 2017 en validant sur 2018 et avec 2018 en validant sur 2019. Les données des différentes années ont été prises de janvier à avril.

Voici les résultats qui ont été obtenus :

Années	précision moyenne	rappel moyen	f-score moyen	f-score du café
2017 validé sur 2018	73.78 %	79.75 %	75.48 %	80.53 %
2018 validé sur 2019	84.24 %	80.94 %	82.28 %	84.98 %

On peut voir qu'en prenant les années 2017 et 2018 les résultats sont moins bons qu'avec les années 2018 et 2019. Cette différence peut être due à ce que les points géoréférencés datent de 2018 et correspondent donc mieux au contenu des images utilisées pour l'entraînement, mais d'autres facteurs comme par exemple la différence entre les images des deux années et la quantité de nuages des années choisies peuvent intervenir.

Si l'on compare ces résultats avec ceux obtenus avec cette combinaison de canaux, cette plage de temps de janvier à avril et cette architecture, on peut constater que les performances sont encore plus optimistes qu'avec une validation croisée et une séparation aléatoire. En effet, là où le f-score moyen était de 77.14 %, il est à présent de 82.28 %.

## 9.11 Modèle avec tous les labels

Les expériences précédentes s'intéressaient à un nombre restreint de labels. Dans cette expérience, on inclut tous les labels de 2018 que nous avons à disposition, qui apparaissent au moins 5 fois au Vietnam. On exclut néanmoins les labels trop difficiles à déterminer : culture mélangée, bâton pour poivre, poivre et café, poivre et autres, arbre épargné ; ces labels ressemblaient trop à d'autres labels existants.

En tout, c'est dix-sept labels différents qui devront être prédits correctement.

### 9.11.1 Résultats

Les paramètres utilisés, dans cette expérience, sont ceux-ci :

Canaux	coastal_aerosol, blue, green, red, nir, swir1, swir2
Nombre de tests	4
Labels utilisés	Café, forêt dense, caoutchouc, agriculture saisonnière, urbain, eau, autres arbres, nature sans arbre, poivre, thé, riz, forêt décidue, pins, brousse & fruticée (shrubland bushland), pâturages, forêt secondaire dégradée, mine sol visible (mine baresoil).
Nombre de pixels autour du label	4 (c.-à-d. une image de 9x9 pixels)
Architecture	2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout
Arrêt prématué	Sur le f1-score, patience de 100

Ces paramètres sont les mêmes que pour la seconde comparaison des canaux à la différence que d'autres labels sont utilisés. Une autre différence est l'utilisation du jeu de données «collection 2» de Landsat 8.

Voici les performances :

précision moyenne	rappel moyen	f-score moyen	f-score du café
50.41 %	70.95 %	53.02 %	47.92 %

Il y a donc une véritable diminution des performances. Cette diminution était prévisible : plus il y a de labels à prédire, plus le réseau de neurones a de chance de se tromper.

Des expériences ont ensuite été réalisées dans le but d'améliorer ces résultats tout en conservant tous les labels.

Pour ce qui est des confusions, on remarque que le café est confondu principalement avec le thé, le poivre, la forêt secondaire dégradée, les pâturages et le caoutchouc.

Le poivre, lui, est confondu avec les territoires urbains et le café. Le caoutchouc est confondu avec le thé et le café. Et le thé est confondu avec le café et le poivre.

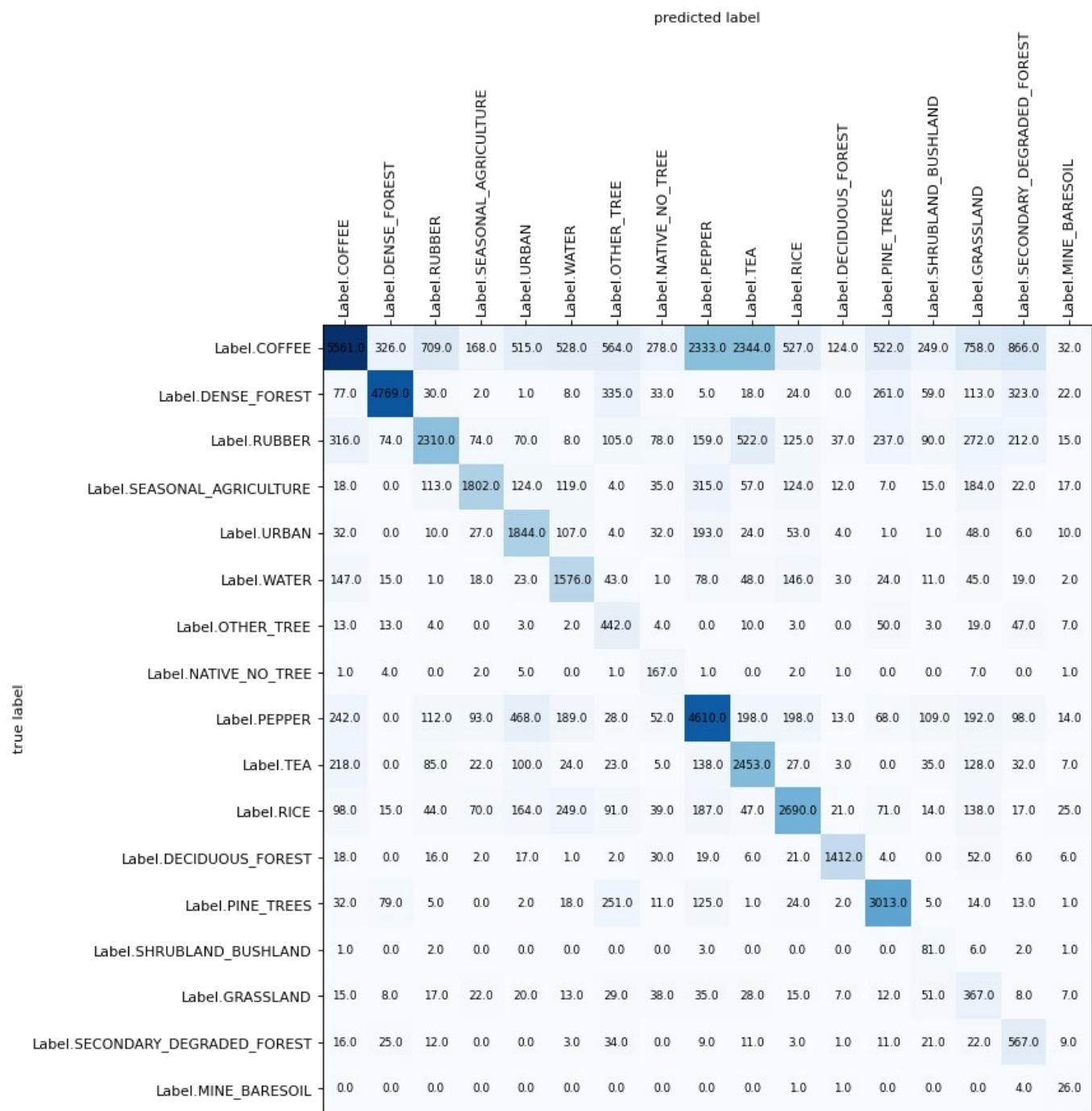


Figure 37: Matrice de confusion de l'expérience avec tous les labels

## 9.12 Modèle à plusieurs sorties

Jusqu'à cette expérience, tous les modèles testés n'avaient qu'une seule sortie. Il est possible d'avoir des modèles avec plusieurs sorties [67-69] et dans cette expérience-ci on va créer un modèle avec une sortie prédisant le label comme auparavant et une autre prédisant une catégorie.

La catégorie est déterminée par rapport au label, ces catégories sont celles-ci :

- Culture : toutes les plantations comme café, thé, poivre, huile de palme, banane etc
- Forêt : les forêts
- Urbain : les villes, routes, mais aussi les mines, fermes et serres.
- Eau : point d'eau, lacs, rivière, ...
- Autres arbres : les arbres qui ne font pas partie d'une forêt. Les arbres d'ombres qui peuvent abriter des plantations ou les arbres comme les acacias qui sont utilisés pour leur bois en font partie aussi.
- Autre nature : tous les endroits naturels mais qui ne concernent pas les catégories précédentes (pâturages, savanes, terres brûlées...). Certains de ces terrains pourraient toutefois contenir des arbres.

Pour plus de détails sur la répartition des labels en catégories, voir le fichier «labelsUtils.py».

### 9.12.1 Résultats

Voici quels paramètres ont été utilisés :

Canaux	coastal_aerosol, blue, green, red, nir, swir1, swir2
Nombre de tests	4
Labels utilisés	Café, forêt dense, caoutchouc, agriculture saisonnière, urbain, eau, autre arbres, nature sans arbre, poivre, thé, riz, forêt décidue, pins, brousse & fruticée (shrubland bushland), pâturages, forêt secondaire dégradée, mine sol visible (mine baresoil).
Catégories	Culture, forêt, urbain, eau, autres arbres, autre nature
Nombre de pixels autour du label	4 (c.-à-d. une image de 9x9 pixels)
Architecture	2x32 conv k=(3, 3), MaxPooling2D k=(3, 3), 2x64 conv k=(2, 2), MaxPooling2D k=(3, 3), 0.25 SpatialDropout, GlobalAveragePooling, 1x128 FC with 0.25 dropout
Arrêt prématuré	Sur le f1-score, patience de 100

Ce sont les données de janvier à avril de la collection 2 en 2018 qui ont été prises.

Il faut aussi préciser que cette expérience utilise la fonction de perte «focal loss». C'est une fonction de perte qui vise à palier aux problèmes de déséquilibre entre les classes lorsque la quantité d'images pour chacune d'elle varie beaucoup [70,71].

Les performances sont meilleures qu'avec le modèle à une seule sortie qui utilise tous les labels, et la performance pour la classification en catégories est meilleure que celle pour la classification en labels :

	<b>précision moyenne</b>	<b>rappel moyen</b>	<b>f-score moyen</b>	<b>f-score café/culture</b>
<b>Labels</b>	77.61 %	73.93 %	75.52 %	82.46 %
<b>Catégories</b>	85.21 %	80.46 %	82.39 %	95.45 %

En observant la matrice de confusion, on peut constater que le café est beaucoup moins confondu qu'auparavant avec certains labels d'autres catégories comme «forêt secondaire dégradée» et «pâturages».

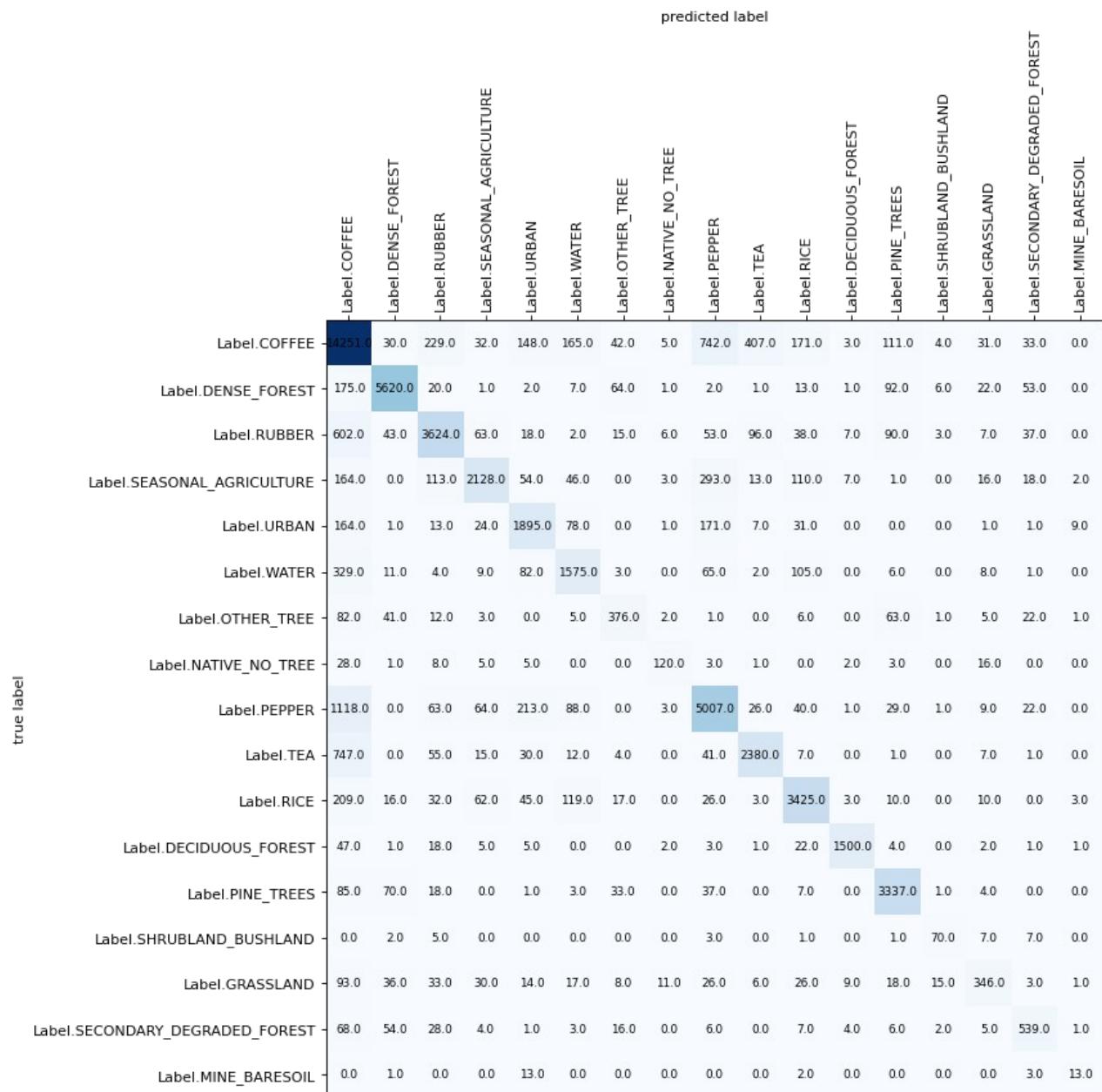


Figure 38: Matrice de confusion des labels du modèle à plusieurs sorties

La matrice de confusion des catégories montre que la difficulté se situe surtout entre autres arbres et culture (peut-être parce que des arbres ont été mis dans culture comme les bananiers ou les palmiers à huile). On peut voir aussi que toutes les autres catégories que «culture» sont surtout confondues avec de la culture.

La catégorie «culture» est quant à elle principalement confondue avec des territoires urbains, de l'eau et «autre nature».

L'évolution du f-score en fonction des epochs à été réalisée en prenant la moyenne des f-score pour la sortie «labels» et pour la sortie «catégorie» à chaque epochs.

On peut voir sur ce graphe une différence de performance entre l'entraînement et la validation. Ce qui est aussi à relever est que l'amélioration des performances est très rapide dans les cinquante premières epochs et demande de plus en plus d'epochs pour gagner de petites améliorations.

		predicted label					
		CULTURE	FOREST	URBAN	WATER	OTHER_TREE	OTHER_NATURE
true label	CULTURE	36568.0	202.0	453.0	417.0	78.0	322.0
	FOREST	441.0	7757.0	6.0	12.0	33.0	187.0
URBAN	444.0	5.0	1900.0	77.0	2.0	0.0	
WATER	549.0	11.0	68.0	1559.0	4.0	9.0	
OTHER_TREE	323.0	65.0	15.0	10.0	537.0	30.0	
OTHER_NATURE	257.0	117.0	2.0	6.0	7.0	3827.0	

Figure 39: matrice de confusion des catégories du modèle à plusieurs sorties

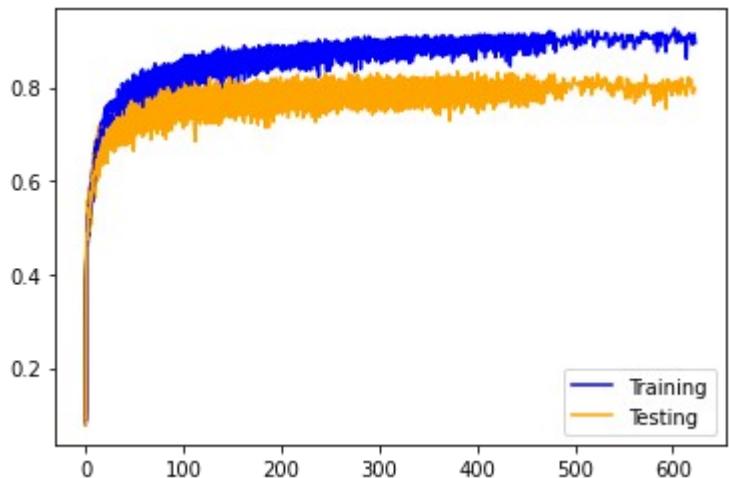


Figure 40: Évolution du f-score du modèle à plusieurs sorties

## 9.13 Méta-apprentissage avec Reptile

Reptile est une technique de méta-apprentissage. Son but n'est pas d'obtenir un modèle qui va pouvoir résoudre une tâche spécifique comme classer des images, mais d'arriver à un modèle qui apprend des caractéristiques utiles pour de multiples tâches et ainsi avoir un modèle qui apprend plus rapidement à résoudre des tâches qu'il n'a pas rencontrées auparavant, et qui est prévu pour s'adapter rapidement à de nouvelles tâches ne demandant alors que peu d'exemples de données pour apprendre [72,73]. Si l'on observe la façon dont un humain apprend, par-exemple une nouvelle langue, on remarque qu'il ne va pas devoir chaque fois réapprendre les bases communes (comme certains concepts de grammaire etc) et qu'il va donc apprendre plus rapidement des langues similaires. Ici l'objectif est d'arriver à un modèle qui ait appris les bases communes de différentes tâches pour arriver à un résultat un peu semblable au lieu d'entraîner un modèle qui ne sera utile que pour une tâche très spécifique et qui devra tout réapprendre s'il doit résoudre une tâche différente.

Le fonctionnement de Reptile se présente ainsi : on va d'abord commencer avec un modèle qui a des poids aléatoires. Ensuite, on va entraîner le modèle sur une tâche aléatoire (dans notre cas les tâches sont de classer les images d'une région du monde). On répétera généralement moins de dix epochs l'entraînement sur cette tâche. Après cela, on va regarder la différence entre les poids obtenus et ceux que l'on avait avant pour pouvoir mettre à jour les poids du modèle en les modifiant en conséquence. Enfin, on recommence le même processus sur une autre tâche choisie aléatoirement et ainsi de suite [73,74].

Ce qu'il faut prendre en considération est que la sortie du modèle changera dans notre cas à chaque tâche. En effet, les labels et les catégories présentes dans chaque région étant différentes, la sortie est elle aussi différente à chaque fois. On conservera donc les poids de chacune de ces couches de sortie pour pouvoir les réutiliser la prochaine fois que l'on s'entraînera sur la même tâche.

Le but de cette expérience est de voir si en pré-entraînant un modèle avec Reptile sur diverses régions autres que le Vietnam on peut obtenir un modèle qui apprenne rapidement à classer correctement les données du Vietnam. Si cela fonctionne, cela montre que l'on peut ensuite changer les labels et la région concernée et obtenir des résultats rapidement avec un modèle pré-entraîné avec Reptile.

### 9.13.1 Résultats

L'architecture du modèle a été choisie parmi les modèles n'incluant pas de couche «global average pooling». Cette couche ne semblait peut-être pas adaptée à cette technique de pré-entraînement. Cette architecture est une des architectures avec les meilleurs résultats et n'a pas un nombre démesuré de paramètres.

```
inputs = Input(shape=(image_width, image_height, image_depth))
layers = Rescaling(0.0000275, offset=0.2)(inputs)
layers = BatchNormalization()(layers)
layers = Conv2D(filters=16, kernel_size=(2, 2), padding="same")(layers)
layers = BatchNormalization()(layers)
layers = Activation(activations.relu)(layers)
layers = Conv2D(filters=16, kernel_size=(2, 2), padding="same")(layers)
layers = BatchNormalization()(layers)
layers = Activation(activations.relu)(layers)
layers = Conv2D(filters=16, kernel_size=(2, 2), padding="same")(layers)
layers = BatchNormalization()(layers)
layers = Activation(activations.relu)(layers)
layers = Conv2D(filters=16, kernel_size=(2, 2), padding="same")(layers)
layers = BatchNormalization()(layers)
layers = Activation(activations.relu)(layers)
layers = Conv2D(filters=16, kernel_size=(2, 2), padding="same")(layers)
layers = BatchNormalization()(layers)
layers = Activation(activations.relu)(layers)
layers = Flatten(name="last_pretrained_layer")(layers)
layers = Dense(128, activation='relu')(layers)
layers = Dense(128, activation='relu')(layers)
layers = Dropout(0.5, name="last_input_layer")(layers)

label_output = Dense(len(labels), activation='softmax', name="label")(layers)
category_output = Dense(len(LabelCategory), activation='softmax', name="category")(layers)

model = Model(inputs=inputs, outputs=[label_output, category_output])
```

Comme on peut le remarquer dans le code ci-dessus, c'est un modèle à plusieurs couches de sorties qui sera pré-entraîné dans cette expérience. La fonction de perte est dans cette expérience la fonction de perte «focal loss».

Lors du pré-entraînement avec Reptile, les résultats étaient meilleurs en utilisant les données de toute une année plutôt qu'en utilisant seulement celles de janvier à avril.

On peut en effet le remarquer sur le graphe ci-dessous, la perte est plus petite sur certains jeux de données. Ce n'est pas étonnant, car certaines régions sont très nuageuses et un autre facteur est que les différentes régions ont des saisons et des climats différents durant les mois de janvier à avril.

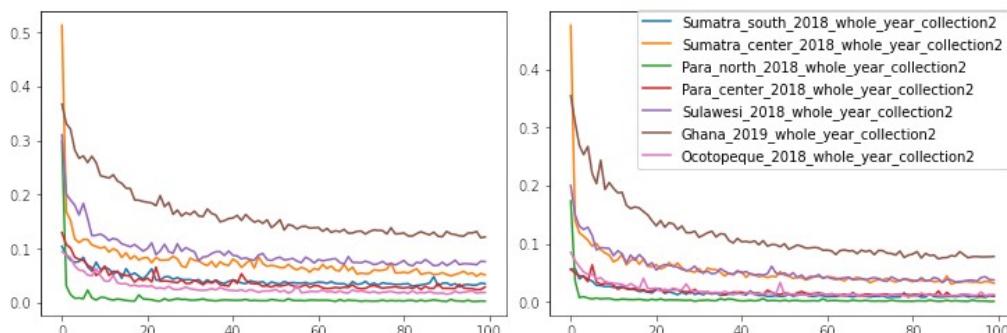


Figure 41: Comparaison de l'évolution de la perte en fonction des epochs de chaque tâche. À gauche avec les données de janvier à avril et à droite les données sur une année.

C'est donc cette plage de temps d'une année qui a été choisie pour le pré-entraînement. Ce graphe montre aussi que certaines régions sont plus difficiles à classifier que d'autres.

Après avoir pré-entraîné le modèle avec Reptile, le modèle a été entraîné avec les données du Vietnam. Pour pouvoir comparer les résultats, un modèle à plusieurs sorties avec la même architecture et utilisant aussi les données de toute l'année 2018 a été entraîné sans partir d'un modèle pré-entraîné avec Reptile.

Il a aussi été testé de geler les couches pré-entraînées, mais les résultats étaient mauvais.

Le modèle qui a été pré-entraîné avec Reptile a demandé moins d'epochs pour arriver à des performances légèrement inférieures. En effet, chaque bloc de la validation croisée avec le modèle pré-entraîné a demandé en moyenne 349 epochs contre 426 pour un modèle sans l'utilisation de Reptile.

Voici la comparaison des performances des deux modèles :

	<b>précision moyenne</b>	<b>rappel moyen</b>	<b>f-score moyen</b>	<b>f-score café/culture</b>
<b>Sans Reptile</b>				
<b>Labels</b>	77.81 %	69.97 %	73.06 %	81.56 %
<b>Catégories</b>	85.46 %	80.22 %	82.40 %	95.39 %
<b>Avec Reptile</b>				
<b>Labels</b>	76.74 %	69.29 %	72.19 %	81.49 %
<b>Catégories</b>	85.68 %	79.94 %	82.44 %	95.61 %

On peut voir sur les graphes ci-dessous la comparaison de l'évolution du f-score et de la perte d'un modèle sans ou avec le pré-entraînement. Les graphes se ressemblent beaucoup, mais il faut prendre garde que l'échelle des epochs va de 0 à 700 pour les graphes de gauche alors que les graphes de droite vont de 0 à presque 600.

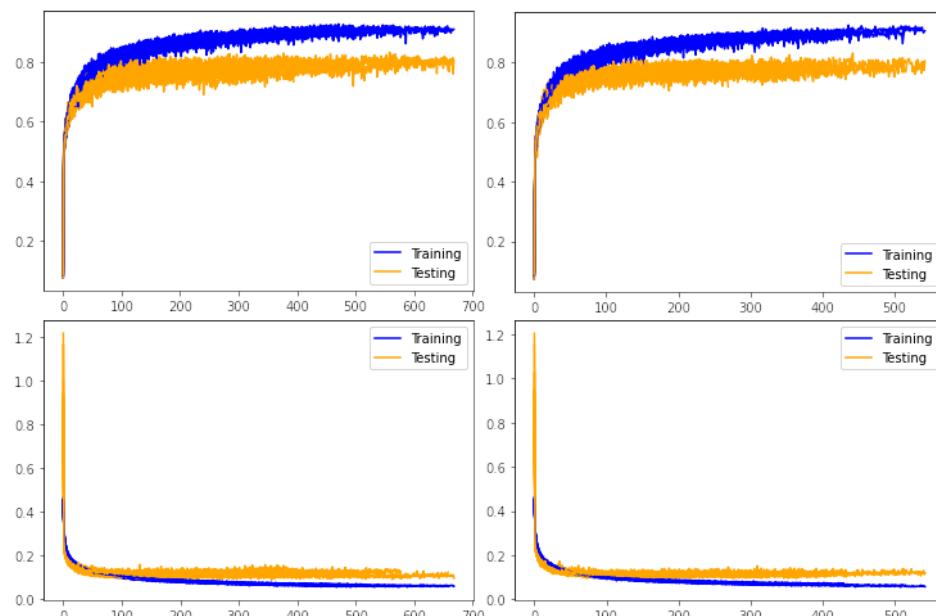


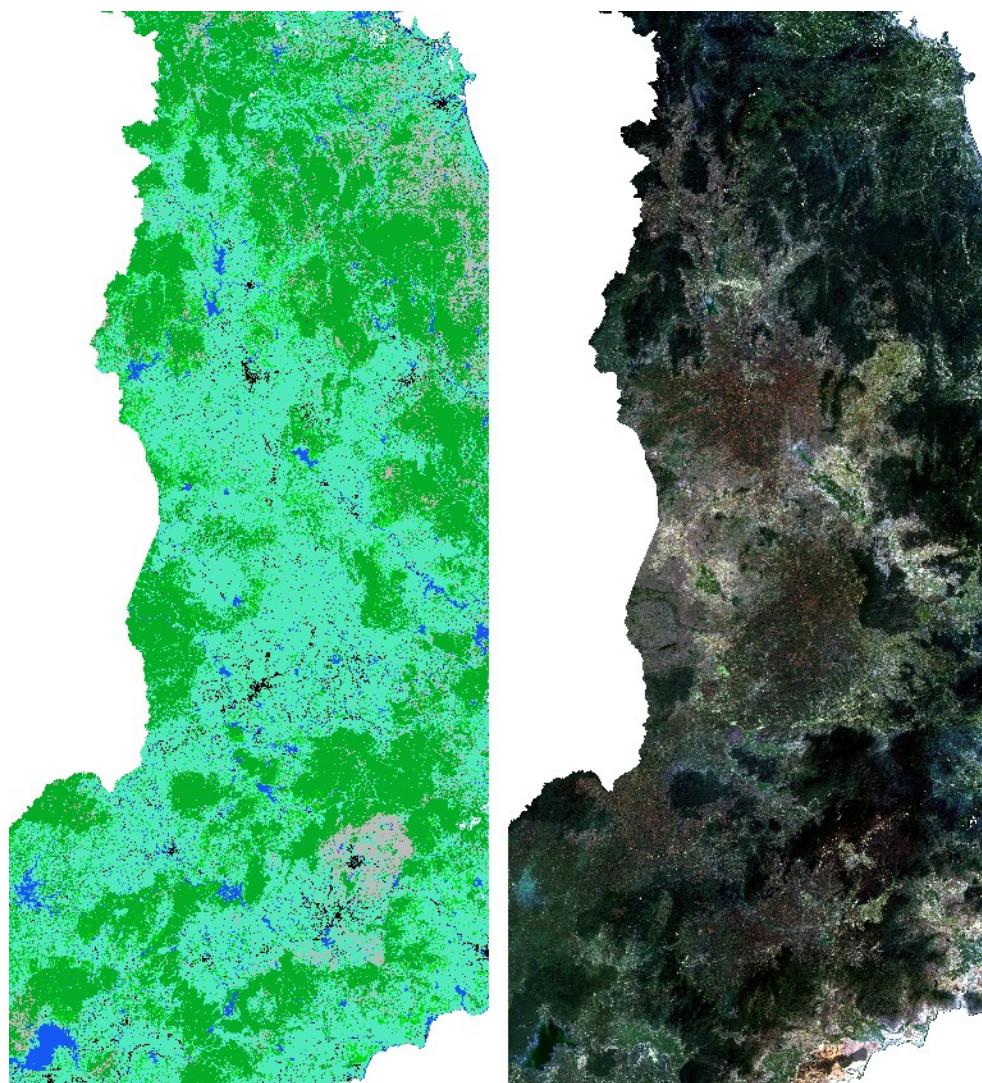
Figure 42: Évolution du f-score et de la perte d'un modèle sans (à gauche) et avec (à droite) Reptile

## 9.14 Modèle final

Le modèle final choisi est celui de l'expérience de modèle à plusieurs sorties. Ce modèle a été choisi, car c'est celui qui a eu les meilleurs résultats. En effet, ses performances sont meilleures que celles du modèle pré-entraîné avec Reptile et l'architecture est celle qui a été sélectionnée après les différentes comparaisons d'architectures. De plus, le modèle avec Reptile demandait d'utiliser les données de toute une année alors qu'avec ce modèle-là, on peut utiliser la plage de temps de janvier à avril qui semble donner de meilleurs résultats<sup>1</sup>.

### 9.14.1 Prédictions

Voici à quoi ressemblent la prédiction des catégories pour 2018 de la partie montagnes centrales du Vietnam (central highland):



Les catégories sont :

- vert : forêt
- turquoise : culture
- eau : bleu
- urbain : noir
- autre arbre : vert clair
- autre nature : gris

Globalement, la prédiction semble cohérente. On remarque des rivières, des lacs et des forêts groupées et non des couleurs aléatoirement réparties. Il est intéressant de regarder l'image satellite de 2018 pour se rendre compte du degré de ressemblance de la prédiction. Une erreur que l'on voit est que des territoires urbains sont confondus avec de la culture.

Figure 43: Prédiction des catégories en 2018 et carte satellite de 2018

<sup>1</sup> Voir l'expérience dans le notebook «timeRangeComparison.ipynb»

Toujours en 2018, voici où le café a été prédit :



Figure 44: Prédiction du café en 2018

Les couleurs ont été choisies comme ceci :

- vert : le café
- rouge : les points connus de café
- noir : le reste

Les points connus sont, comme on pouvait le souhaiter, placés à des endroits où du café est prédict.

Si l'on compare avec la carte de la prédiction des catégories, on voit que c'est une partie non négligeable des cultures qui sont prédictes comme du café, mais non l'entièreté. On pouvait s'y attendre, car dans la phase de validation du modèle on a vu que le poivre, le thé et le caoutchouc étaient fréquemment confondus avec le café.

### 9.14.2 Zones à risque de déforestation

Pour déterminer les zones à risque de déforestation, une carte comptabilise le nombre de fois qu'une parcelle prédictée comme de la forêt en 2014 a été ensuite été prédictée comme du café. Pour ce faire, c'est bien la sortie de label qui a été utilisée, mais les labels de forêt ont été regroupés en un seul label forêt de la même manière que cela avait été fait pour classer les labels en catégories. Malheureusement, en comparant la déforestation prédictée et les images satellites à différentes dates disponibles avec Google Earth Pro (images satellites de meilleure qualité que celles fournies au réseau de neurones), on constate que les résultats ne sont pas toujours corrects. En effet, à de nombreux endroits où du café a été prédicté comme ayant remplacé de la forêt, se trouve en fait encore de la forêt. D'autres fois, de la forêt avait été prédictée alors que du café s'y trouvait depuis 2014 déjà.

Cependant, cette approche de vérification reste assez délicate, la teneur du sol n'étant pas facile à déterminer avec un œil peu entraîné, d'autant plus que même si de la déforestation a eu lieu, encore faut-il réussir à déterminer si c'est bel et bien du café qui a remplacé la forêt.

Quelques zones, où de la déforestation avait été prédictée, ont néanmoins pu être vérifiées par ce biais.

La prédiction se trouve ci-contre, avec en fond la prédiction de la catégorie pour voir si de la forêt se trouve proche d'une zone où de la déforestation a été prédictée.

C'est dans un rouge de plus en plus foncé qu'est représenté le nombre de fois (de deux à sept) qu'une parcelle, prédictée comme de la forêt en 2014, a été prédictée comme du café par la suite.

Sont entourées en noir les zones qui ont été vérifiées avec Google Earth Pro et qui semblent en effet avoir été déforestées. Néanmoins, il est possible que cette déforestation ne soit pas à cause du café.

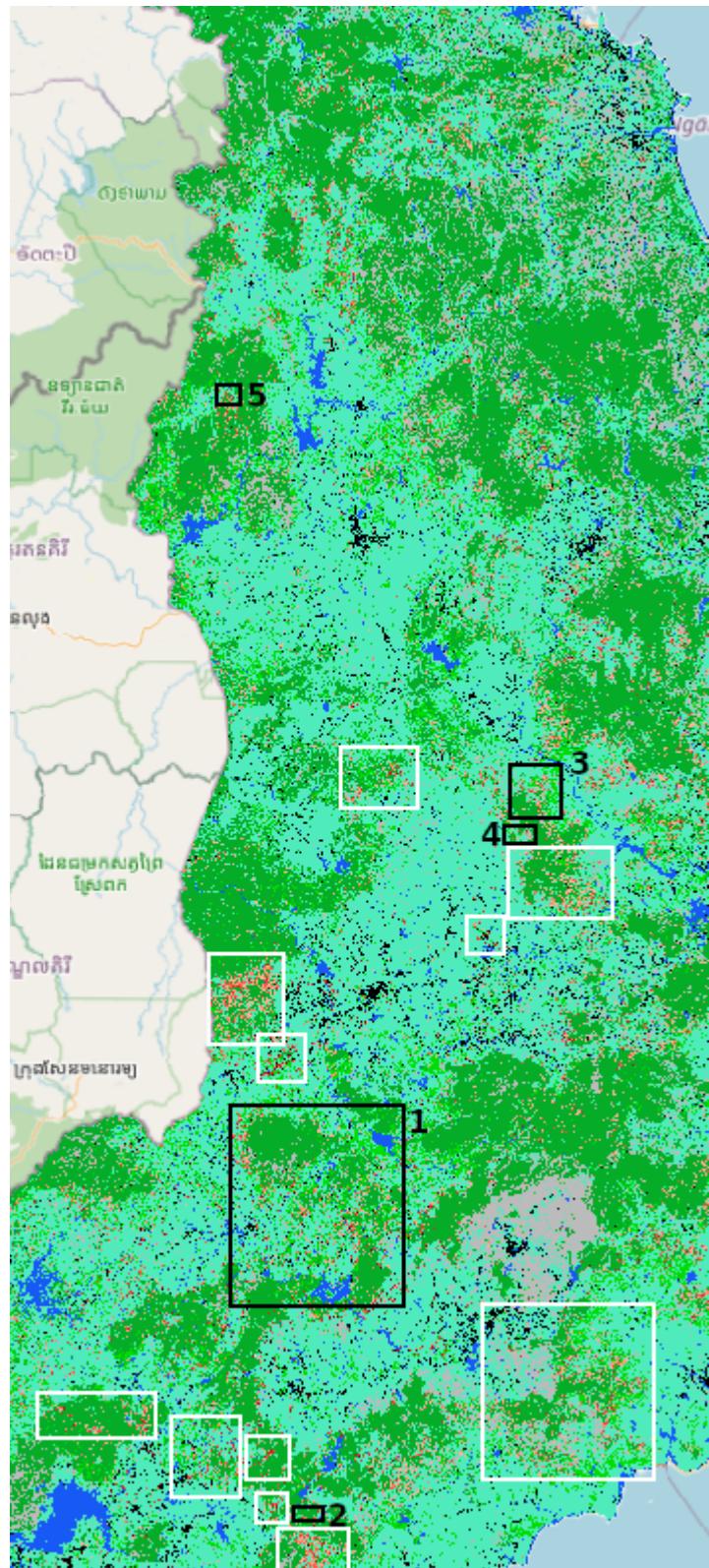


Figure 45: Carte de la prédiction de la déforestation avec en noir des zones semblant correctes (en noir) et incorrectes (en blanc).

À l'inverse, les zones entourées en blanc paraissent avoir été prédite à tort comme de la déforestation provenant du café.

La plus grande région où de la déforestation paraît avoir eu lieu, et qui a été contrôlée à l'aide d'images satellites, est la zone une. Voici une comparaison avant/après de cette région :



Figure 46: Comparaison 2014 - 2020 de la zone 1. Images provenant de Google Earth Pro

En zoomant un peu plus, on peut en effet confirmer que la déforestation a eu lieu au profit de la culture, comme on peut le voir sur l'image ci-contre.



Figure 47: Exemple de zone déforestée, 2014 - 2019, images obtenues avec Google Earth Pro.

La deuxième zone présente un tronçon dans la forêt où de la culture, probablement de café, est apparue. Même si le chemin dans la forêt était déjà en partie là, il semble s'être étendu et des cultures s'y sont apparemment installées.

Dans la troisième zone, on aperçoit clairement que la culture (partiellement sous les nuages sur l'image), qui se trouvait déjà à cette emplacement en 2014, s'est propagée en partie sur de la forêt.

Difficile à dire quelle culture occupe maintenant la zone déforestée.

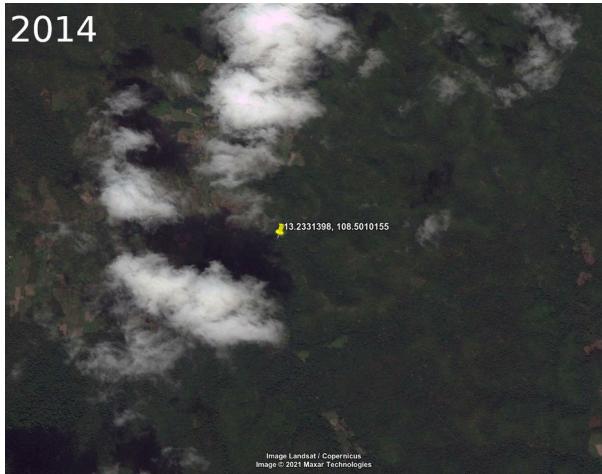


Figure 49: Zone 2, 2014 - 2021, image provenant de Google Earth Pro.

Figure 48: Zone 3, 2014 - 2021, images obtenues avec Google Earth Pro.

Ci-dessous une carte prédisant la déforestation, mais cette fois on s'intéresse plus largement à la culture, et non plus seulement au café, qui a remplacé de la forêt.

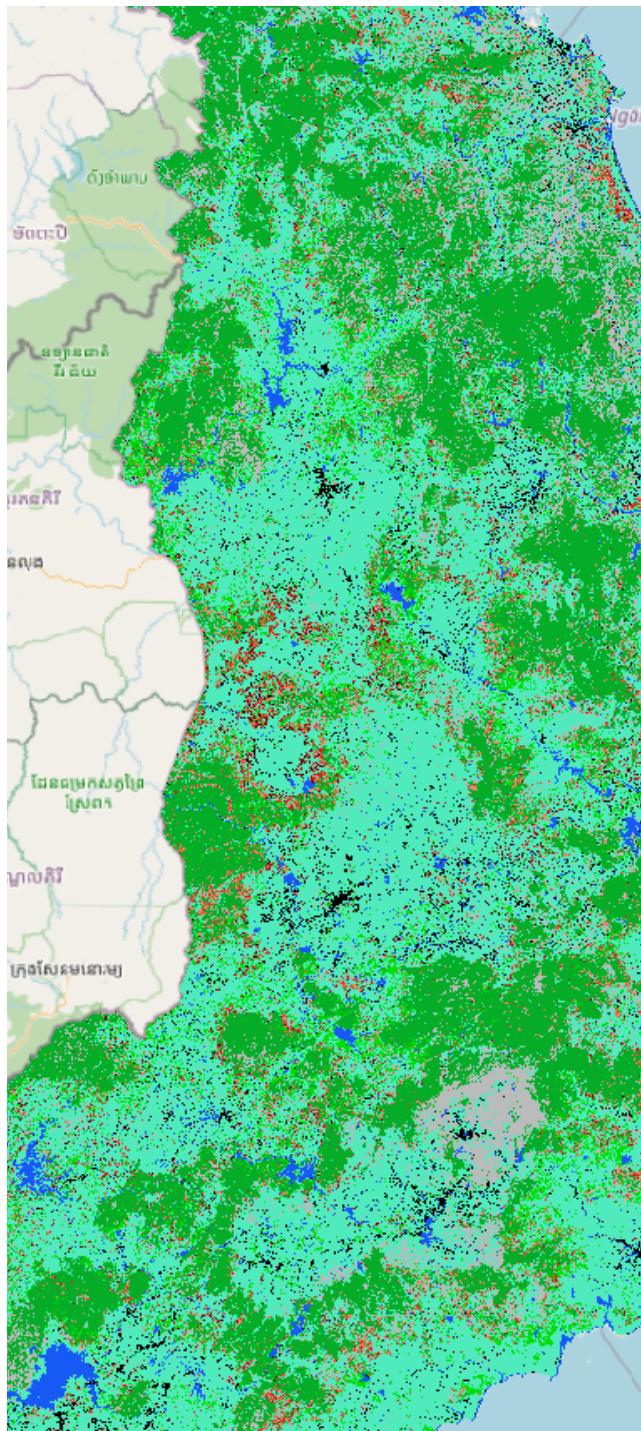


Figure 50: Prédiction de la déforestation par de la culture

Enfin, voici une image comparant la première et la dernière prédiction des catégories :

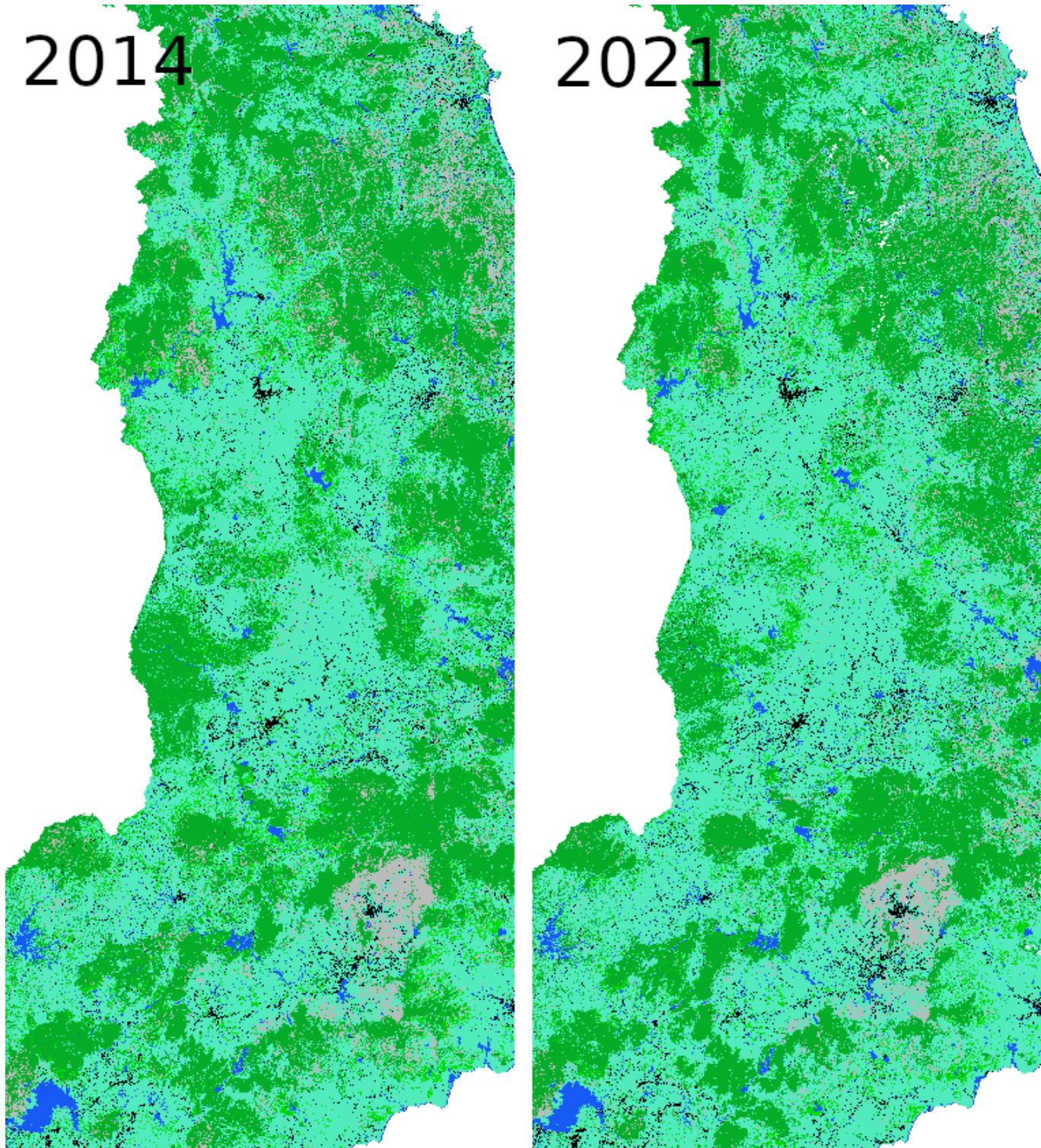


Figure 51: Comparaison de la première et dernière prédiction des catégories

## 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).

The screenshot shows a GitHub Project board with three columns: "To do", "In progress", and "Done".

- To do:** 11 items
  - Continuer les tests d'architecture
  - Joindre le code au rapport intermédiaire
  - Choisir des canaux moins corrélés
  - Essayer un modèle agroforestry contre le reste
  - Écrire le résumé pliable
- In progress:** 3 items
  - Continuer les tests d'architecture
  - Écrire sur la comparaison de différentes architectures sur les performances et le sur-ajustement
  - Écrire sur l'organisation / processus du travail
- Done:** 22 items
  - Réduire l'overfitting
  - Ne pas se baser sur l'accuracy, mais sur le f-score pour choisir le modèle
  - Écrire le contexte du projet
  - Essayer en centrant, normalisant avec l'augmentation des données
  - Écrire sur le EarlyStopping

At the bottom, there are buttons for "Automated as To do" and "Manage".

Figure 52: Organisation du TB sur Github project

## 11 Scripts

L'intégralité du code du projet peut être retrouvé sur GitHub : [https://github.com/SimWalther/vietnam\\_coffee\\_detection](https://github.com/SimWalther/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 :

```
. 
├── README.md
├── requirements.txt
└── scripts
    ├── data
    ├── exempleGEE.js
    ├── models
    └── notebooks
        ├── bandsComparison.ipynb
        ├── bandsComparisonV2.ipynb
        ├── bandsVariability.ipynb
        ├── finalModel.ipynb
        ├── multiOutputModel.ipynb
        ├── neuralNetworkArchitechture.ipynb
        ├── neuralNetworkArchitectureEarlyStopping.ipynb
        ├── overfittingImpactAnalysis.ipynb
        ├── rasterPrediction.ipynb
        ├── reptile.ipynb
        ├── spatialValidation.ipynb
        ├── timeRangeComparison.ipynb
        ├── trainingWithOneYearAndTestingWithAnother.ipynb
        └── withEveryLabels.ipynb
    └── src
        ├── bandUtils.py
        ├── config.py
        ├── convNetUtils.py
        ├── downloadMap.py
        ├── labelsUtils.py
        ├── lossesUtils.py
        ├── mergeRasterFiles.py
        ├── rasterUtils.py
        ├── regionUtils.py
        └── statisticsUtils.py
            └── visualizationUtils.py
    └── visualizations
        └── data
```

Figure 53: 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<sup>2</sup>.

Le dossier «visualizations» contient des projets QGIS contenant les résultats des prédictions.

2 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.
- neuralNetworkArchitecture.ipynb : c'est dans ce fichier que l'on compare différentes architectures de réseau de neurones convolutifs. Ce fichier ne teste 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 faits dans ce notebook.
- overfittingImpactAnalysis.ipynb : ce fichier permet de mieux comprendre comment les performances évoluent quand le réseau de neurones a du sur-ajustement.
- withEveryLabels.ipynb : ce notebook contient l'expérience avec tous les labels.
- trainingWithOneYearAndTestingWithAnother.ipynb : dans ce notebook on entraîne un réseau de neurone avec toutes les données d'une année puis on valide avec les données d'une autre année.
- timeRangeComparison.ipynb : ce notebook contient une comparaison de performances de modèles entraînés avec des plages de temps différentes.
- spatialValidation.ipynb : c'est dans ce fichier qu'a été réalisée l'expérience de validation croisée spatiale.
- reptile.ipynb : ce notebook contient l'expérience de méta-apprentissage avec reptile.
- rasterPrediction.ipynb : c'est ici qu'est effectué la prédiction des labels et des catégories sur tout le Vietnam. Des visualisations sont générées à la fin du notebook.

## 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 géoréférencés de chaque label et pour définir à quelle catégorie appartient chaque label.
- bandUtils.py : c'est là que se situe le code concernant les canaux. On y retrouve le code utilisé pour ajouter des canaux calculés à un jeu de données chargé en mémoire.
- rasterUtils.py : les fonctions pour traiter les fichiers de raster.
- regionUtils.py : définit des régions et permet d'obtenir les coordonnées des labels qui sont contenues dans celles-ci.

- `statisticsUtils.py` : les fonctions pour établir des statistiques.
- `visualizationUtils.py` : le code permettant de générer les graphiques ou toute autre représentation graphique.
- `lossesUtils.py` : définit des fonctions de perte. Ces deux fonctions de perte ont été tirées de code disponible sur GitHub [75,76].

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 rasters exportés depuis Google Earth Engine. Attention toutefois : ce script nécessite une grande quantité de mémoire vive.
- `config.py` : définit les chemins d'accès aux données. C'est particulièrement utile pour pouvoir avoir la possibilité de changer rapidement les chemins d'accès entre l'ordinateur personnel et le serveur de calcul.

### 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, with_model_checkpoint=False, model_name="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  
    :param model_name: name of the model, used to name the model file  
    :param with_model_checkpoint: defines if model checkpoint should be 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()  
  
    model_checkpoint_cb = None  
  
    if with_model_checkpoint:  
        # Model checkpoint callback should be created here to avoid resetting the  
        # 'best' property of the model checkpoint.  
        # By doing so, we ensure that model checkpoint is the best across all cross validations.  
        model_file = os.path.join(MODEL_ROOT_PATH, model_name + ".hdf5")  
        model_checkpoint_cb = ModelCheckpoint(  
            model_file, monitor='f1_score_val',  
            verbose=0, save_best_only=True, mode='max'  
        )  
  
    images = images_from_dataset(dataset, bands)  
    true_classes = labels_from_dataset(dataset, labels_names)
```

```
for nth_cross_validation in range(nb_cross_validations):
    fold = 0

    for train_index, validation_index in StratifiedKFold(n_splits=k, shuffle=True).split(images,
true_classes):
        X_train, X_validation = images[train_index], images[validation_index]
        y_train, y_validation = true_classes[train_index], true_classes[validation_index]
        Y_train = to_categorical(y_train, num_classes=len(labels_names))
        Y_validation = to_categorical(y_validation, num_classes=len(labels_names))
        fold_size = len(y_train)

        # create data generators
        train_datagen = ImageSequence(
            X_train, Y_train, batch_size=32, augmentations=AUGMENTATIONS
        )
        validation_datagen = ImageSequence(
            X_validation, Y_validation, batch_size=32,
            augmentations=VALIDATION_AUGMENTATIONS
        )
        class_weights = compute_class_weights(y_train)

        print(f"\nValidation {nth_cross_validation + 1}, fold {fold + 1} :\n-----\n")

        # clone given model without keeping the layers weights
        current_model = clone_model(model)

        # Specify optimizer and loss function
        current_model.compile(loss='categorical_crossentropy', optimizer='adam',
                               metrics=['accuracy'])

        history, trained_model = train_model(
            model=current_model,
            train_datagen=train_datagen,
            validation_datagen=validation_datagen,
            class_weights=class_weights,
            epochs=epochs,
            steps_per_epoch=fold_size / 32,
            early_stopping=early_stopping,
            model_checkpoint_cb=model_checkpoint_cb,
        )

        conf_matrix, accuracy, loss = evaluate_model(trained_model, X_validation, Y_validation,
                                                      y_validation, nb_labels)

        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)

        fold += 1

    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.

Des fonctions similaires à «cross\_validation» ont été réalisées pour des cas d'utilisation légèrement différents. Ces fonctions sont : «cross\_validation\_multi\_output\_model» pour les modèles à plusieurs sorties, «spatial\_cross\_validation\_from\_csv\_files» pour la validation croisée spatiale et «cross\_validation\_with\_metrics\_evolution» pour avoir des mesures, comme la matrice de confusion, à intervalle régulier.

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

```
class Metrics(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_datagen=None, validation_datagen=None):  
    """  
    Initialize callback  
    :param train: the train generator  
    :param validation: the validation generator  
    """  
  
    super(Metrics, self).__init__()  
    self._supports_tf_logs = True  
    self.train_datagen = train_datagen  
    self.validation_datagen = validation_datagen
```

Ici on crée 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 prématûr. On pourra aussi avoir un historique de ces mesures additionnelles. Pour le moment *Metrics* n'ajoute que le f-score.

Dans cette même classe, il faut définir ce qui se produit à la fin d'une epoch. Ici, le calcul du f-score. Un point qui pourrait être amélioré est l'utilisation de `isinstance` pour déterminer si le générateur de données à plusieurs sorties ou une seule et calculer alors le f-score pour une ou plusieurs sorties. Il serait peut-être préférable de séparer le code en deux sous-classes différentes. Il est à noter que le f-score de cette classe (donc pas celui calculé à partir de la matrice de confusion, mais l'historique du f-score) pour un modèle à plusieurs sorties a été pris comme la moyenne des f-score des sorties.

```
def on_epoch_end(self, epoch, logs={}):
    # Warning: This will load every training data in memory

    if self.train_datagen is not None:
        train_images, train_classes = elements_inside_data_generator(self.train_datagen)

        if isinstance(self.train_datagen, ImageMultiOutputSequence):
            target_train = [np.argmax(train_classes[output_name], axis=-1)
                            for output_name in train_classes.keys()]
            pred = self.model.predict(train_images)
            predicted_train = [np.argmax(pred[i], axis=-1) for i in range(len(pred))]

            # Here we define f1-score of the multi-output model
            # as the mean of f1-score of each output
            f1_score_train = np.mean([me.f1_score(target_train[i], predicted_train[i],
                                                   average="macro") for i in range(len(predicted_train))])
        else:
            target_train = np.argmax(train_classes, axis=-1)
            predicted_train = np.argmax(np.asarray(self.model.predict(train_images)), axis=-1)
            f1_score_train = me.f1_score(target_train, predicted_train, average="macro")

        logs['f1_score_train'] = f1_score_train

    if self.validation_datagen is not None:
        validation_images, validation_classes = elements_inside_data_generator(
            self.validation_datagen
        )

        if isinstance(self.validation_datagen, ImageMultiOutputSequence):
            target_validation = [np.argmax(validation_classes[output_name], axis=-1)
                                 for output_name in validation_classes.keys()]
            pred = self.model.predict(validation_images)
            predicted_validation = [np.argmax(pred[i], axis=-1) for i in range(len(pred))]

            # Here we define f1-score of the multi-output model
            # as the mean of f1-score of each output
            f1_score_val = np.mean([me.f1_score(target_validation[i], predicted_validation[i],
                                                 average="macro") for i in range(len(predicted_validation))])
        else:
            target_validation = np.argmax(validation_classes, axis=-1)
            predicted_validation = np.argmax(np.asarray(
                self.model.predict(validation_images)), axis=-1)
            f1_score_val = me.f1_score(target_validation, predicted_validation, average="macro")

        logs['f1_score_val'] = f1_score_val

    return
```

Pour entraîner un modèle, c'est la fonction *train\_model* qui est utilisée :

```
def train_model(model, train_datagen, validation_datagen, class_weights, epochs, steps_per_epoch,
                early_stopping=False, model_checkpoint_cb=None):
    """
    Train a Keras neural network model
    :param model_checkpoint_cb: the checkpoint callback
    :param model: the Keras neural network model
    :param train_datagen train data generator
    :param validation_datagen validation data generator
    :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 and the trained model
    """

    callbacks = [Metrics(train_datagen=train_datagen, validation_datagen=validation_datagen)]

    if early_stopping:
        callbacks.append(EarlyStopping(monitor='f1_score_val', patience=100, mode="max"))

    if model_checkpoint_cb:
        callbacks.append(model_checkpoint_cb)

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

    history = model.fit(**fit_args)
    return history, model
```

L'arrêt prématuré (early stopping) est optionnel et se base sur le f-score de la validation. Pour équilibrer les classes, un poids de classe (la variable «*class\_weights*») est utilisé. Il a été calculé préalablement, avec la fonction «*compute\_class\_weights*» :

```
def compute_class_weights(y_train):
    """
    Compute each class weight
    :param y_train: the true label set
    :return: dictionary of class weight
    """

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

    return dict(enumerate(class_weights))
```

Il est à noter que la fonction «`class_weight.compute_class_weight`» provient de «`sklearn.utils`».

Ensuite, après avoir entraîné le modèle on évalue ses performances avec «`evaluate_model`» :

```
def evaluate_model(model, X_test, Y_test, y_test, nb_labels):
    """
    Evaluate a given model and compute a confusion matrix
    :param model: the model
    :param X_test: images set
    :param Y_test: true labels set
    :param y_test: true labels as one hot encoding set
    :param nb_labels: the number of labels
    :return: confusion matrix, accuracy and loss
    """

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

    # Predict labels on batch
    pred = model.predict_on_batch(X_test)
    pred = np.argmax(pred, axis=-1)

    # Confusion matrix
    conf_matrix = me.confusion_matrix(y_test, pred, labels=np.arange(nb_labels))

    return conf_matrix, accuracy, loss
```

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.

Pour réaliser les prédictions sur une carte avec le modèle que l'on a entraîné, on peut utiliser «predict\_label\_category\_on\_raster» et «predict\_on\_raster» :

```
def predict_label_category_on_raster(trained_model, raster_path, bands, square_size=9):
    """
    Predict label and category on a given raster.
    :param trained_model: the trained model
    :param raster_path: the path to the raster
    :param bands: bands to use
    :param square_size: the size of images to predict. Currently also used as a step size.
    :return: the labels and category predictions and their image indices (row, col)
    """

    label_predictions = []
    category_predictions = []
    image_indices = []

    for batch_images, batch_indices in square_chunks(raster_path, bands, square_size):
        pred = trained_model.predict_on_batch(batch_images)
        label_pred = np.argmax(pred[0], axis=-1)
        category_pred = np.argmax(pred[1], axis=-1)

        label_predictions.extend(label_pred)
        category_predictions.extend(category_pred)
        image_indices.extend(batch_indices)

    return label_predictions, category_predictions, image_indices
```

### 11.3.2 rasterUtils.py

Dans ce fichier, la fonction «make\_dataset\_from\_raster\_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».

Cette fonction donne la possibilité de sauver les images sur le disque ou de garder les images dans un jeu de données en mémoire.

Les images contenant des valeurs inconnues ne sont pas gardées.

```
def make_dataset_from_raster_files(labels, raster_paths, labels_coordinates_list, nb_pixel_around,  
save_on_disk=False, dataset_folder_name=""):
```

*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

**:param** *save\_on\_disk*: defines if images need to be saved on disk

**:param** *dataset\_folder\_name*: name of the folder where to save images to

**: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:  
                folder_path = None  
  
                if save_on_disk:  
                    folder_path = os.path.join(DATASET_IMAGES_PATH, dataset_folder_name,  
                                              label.name.lower())  
  
                    # Create directories if they don't exists  
                    os.makedirs(folder_path, exist_ok=True)  
  
                for label_image_index, coordinates in enumerate(labels_coordinates[label.value]):  
                    out_img, out_transform = image_around_coordinates(  
                        raster, coordinates, nb_pixel_around  
                    )  
  
                    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():  
                        if save_on_disk:  
                            filepath = os.path.join(folder_path, str(label_image_index) + '.tiff')  
  
                            # append filepath in dataset where raster is saved not the whole raster  
                            values.append([label.name, filepath, point])  
  
                            metadata = create_image_metadata(out_img, out_transform, raster)  
  
                            # Write merged raster to disk  
                            with rasterio.open(filepath, "w", **metadata) as dest:  
                                dest.write(out_img)  
                        else:  
                            # append whole rasters to dataset if we don't save rasters on the disk  
                            values.append([label.name, out_img.tolist(), point])  
  
return values
```

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

```
def square_around_coordinates(raster, coordinate, nb_pixel_around):
    """
    Create a square of a given number of pixels around a latitude and longitude tuple.
    :param raster: the raster from which the data will be retrieved
    :param coordinate: the latitude and longitude tuple
    :param nb_pixel_around: the number of pixels around a given (lat, lon)
    :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 rasterio pour découper l'image qui est à l'intérieur de ce carré :

```
def image_around_coordinates(raster, coordinates, nb_pixel_around):
    """
    Retrieves raster image n pixels around a given latitude and longitude
    :param raster: the raster from which the data will be retrieved
    :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
    :return: the raster image, the affine transform
    """

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

La fonction «square\_chunks» permet de découper un raster en images d'une taille donnée.

Cette fonction ne donne pas la possibilité de définir le saut entre chaque image. Actuellement les images sont découpées en décalant entre chacune de celles-ci de la taille d'une image. Ce serait une amélioration possible pour cette fonction.

Les images contenant des valeurs inconnues sont ignorées.

```
def square_chunks(raster_path, bands, square_size, batch_size=32):
    """
    Split raster into chunks of a specified size
    :param raster_path: the raster path
    :param square_size: the size of images to predict. Currently also used as a step size.
    :param batch_size: the batch size
    :param bands: bands to filter
    :return: batch of images and their images indices
    """

    with rasterio.open(raster_path) as raster:
        width = raster.shape[0]
        height = raster.shape[1]
        nb_images_row = width // square_size
        nb_images_col = height // square_size

        print(f"Image width: {width}")
        print(f"Image height: {height}")
        print(f"Nb row of images: {nb_images_row}")
        print(f"Nb col of images: {nb_images_col}")

        if width % square_size != 0:
            print(f"Width is not dividable by {square_size}, some px will be ignored...")

        if height % square_size != 0:
            print(f"Height is not dividable by {square_size}, some px will be ignored...")

        img_count = 0
        batch_images = []
        batch_images_indices = []

        for i in range(nb_images_row):
            for j in range(nb_images_col):
                out_img, out_transform = image_square_at_px(
                    i * square_size, j * square_size, square_size, raster
                )
                image = convNetUtils.prepare_image(out_img, bands)

                # Don't add if there is 'NaN' values
                if np.isnan(image).any():
                    continue

                batch_images.append(image)
                batch_images_indices.append((i, j))
                img_count += 1

        if img_count == batch_size:
            yield np.asarray(batch_images), batch_images_indices
            # reset batch
            batch_images = []
            batch_images_indices = []
            img_count = 0

    # if there is a leftover of images (less remaining images than the batch size)
    # yield them
    if len(batch_images) > 0:
        yield np.asarray(batch_images), batch_images_indices
```

### 11.3.3 regionUtils.py

Pour utiliser regionUtils.py, il 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 dans le fichier «config.py».

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 [50,51].

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

«regionUtils.py» 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(CENTRAL_HIGHLANDS_SHP_PATHS, boundaries_shape)
```

Et vu que l'on travaille avec des fichiers geojson, on voudrait extraire les objets géométriques après les avoir filtrés, 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 une énumération des labels :

```
class Label(Enum):
    COCOA = 1
    COFFEE = 2
    COMPLEX_OIL_PALM = 3
    DENSE_FOREST = 4
    OIL_PALM = 5
    RUBBER = 6
    # UNKNOWN = 7
    SEASONAL_AGRICULTURE = 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 = 19
    PALM_GROWING = 20
    CUT_OFF_REGROW = 21
    NATURAL_WETLAND = 22
    INTERCROP = 23
    DECIDUOUS_FOREST = 24
    STICK_FOR_PEPPER = 25
    FLOODED_PLANTATION = 26
    PINE_TREES = 27
    COCONUT = 28
    BAMBOO = 29
    SAVANA = 30
    MANGO = 31
    ORCHARD = 32
```

MINE = 33  
SHRUBLAND\_BUSHLAND = 34  
SPARE\_TREE = 35  
BARE\_SOIL = 36  
GRASSLAND = 37  
SECONDARY\_DEGRADED\_FOREST = 38  
MINE\_BARESOIL = 39  
MINE\_WATER = 40  
PEPPER\_AND\_COFFEE = 41  
PEPPER\_AND\_OTHER = 42  
SHADE\_TREE = 43  
FARM\_HOUSE = 44  
BURNED\_LAND = 45  
ORCHARD\_SMALL = 46  
LAKE = 47  
RIVER = 48  
COFFEE\_FULL\_SUN = 49  
COFFEE\_SHADED = 50  
RUBBER\_GROWING = 51  
CASHEW\_AND\_COCOA = 52  
CASHEW = 53  
ACACIA = 54  
ROAD = 55  
TEAK\_PLANTATION\_FOREST = 56  
PASTURE = 57  
PASSION\_FRUIT = 58  
COCOA\_SHADED = 59  
TRANSITION = 60  
LOTUS = 61  
GREENHOUSE = 62  
COFFEE\_GROWING = 63  
MACADAMIA = 64  
MACADAMIA\_GROWING = 65

Le label «unknown» est commenté, car il ne présente pas beaucoup d'utilité pour ce projet.  
C'est donc en tout 64 labels qui sont définis. Ces labels et leur nomenclature, a été tiré des jeux de données qui nous ont été fournis par le CIAT.

Les catégories sont aussi déclarées dans ce fichier :

```
class LabelCategory(Enum):
    CULTURE = 0
    FOREST = 1
    URBAN = 2
    WATER = 3
    OTHER_NATURE = 4
    OTHER_TREE = 5
```

Et voici comment sont créés les dictionnaires contenant les coordonnées de chaque point géoréférencé 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 = {label.value: [] for label in Label}

    # 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 in labels_coordinates:
                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
```

Et pour déterminer la catégories d'un label ou d'un ensemble de labels on utilise :

```
def category_from_label(label):
    """
    Get the category of a given label
    :param label: the label
    :return: the category
    """

    for labelCategory in LabelCategory:
        if label in CATEGORIES_LABELS[labelCategory]:
            return labelCategory

    return None

def categories_from_label_set(labels, images_labels):
    """
    Get the categories for a label set
    :param labels: the labels used
    :param images_labels: labels for each images
    :return: the categories for each images
    """

    return np.asarray([
        category_from_label(labels[label]).value for label in images_labels
    ])
```

### 11.3.5 bandUtils.py

Dans ce fichier se trouve un enum des canaux :

```
# With Landsat 8 Level 2, Collection 2, Tier 1
class Band(Enum):
    COASTAL_AEROSOL = 0
    BLUE = 1
    GREEN = 2
    RED = 3
    NIR = 4
    SWIR1 = 5
    SWIR2 = 6
    TIRS1 = 7
    NDVI = 8
    MNDWI = 9
    EVI2 = 10
    BU = 11
```

Il s'y trouve aussi les fonctions permettant de rajouter des canaux calculés aux jeux de données.

### 11.3.6 statisticsUtils.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
```

### 11.3.7 visualizationUtils.py

Les fonctions en rapport avec l'affiche graphique sont présentes dans ce fichier.

Par exemple la fonction pour imprimer à l'écran une matrice de confusion :

```
def plot_confusion_matrix(confmatrix, labels_names, ax=None):
    """
    This function generates a colored confusion matrix.
    Code has been taken and adapted from plot_confusion_matrix of MLG course
    :param confmatrix: the confusion matrix
    :param labels_names: labels names
    :param ax: matplotlib axes object to plot to
    """

    if ax is None:
        ax = plt.subplot(111)

    ax.matshow(confmatrix, interpolation='nearest', cmap=plt.cm.Blues)

    for i in range(confmatrix.shape[0]):
        for j in range(confmatrix.shape[1]):
            ax.annotate(str(confmatrix[i, j]), xy=(j, i),
                        horizontalalignment='center',
                        verticalalignment='center',
                        fontsize=8)

    ax.set_xticks(np.arange(confmatrix.shape[0]))
    ax.set_xticklabels([labels_names[label] for label in range(confmatrix.shape[0])],
                      rotation='vertical')
    ax.set_yticks(np.arange(confmatrix.shape[1]))
    _ = ax.set_yticklabels([labels_names[label] for label in range(confmatrix.shape[1])])
    ax.set_xlabel('predicted label')
    ax.xaxis.set_label_position('top')
    ax.set_ylabel('true label')
```

C'est aussi dans ce fichier que se trouvent certaines fonctions en rapport avec les représentations graphiques des prédictions ; entre autres, la fonction permettant de déterminer la première détection des points géoréférencés d'un certain type de label :

```
def label_first_detections(raster_path, dataframe, label_index, label_coordinates, image_size=9):
    """
    Find a specific label first detection at a given coor
    :param raster_path:
    :param dataframe:
    :param label_index:
    :param label_coordinates:
    :param image_size:
    :return:
    """

    with rasterio.open(raster_path) as raster:
        row_col_coffee_labels = [
            find_image_row_col_label_coordinate(raster, coord[0], coord[1], image_size) for coord
            in label_coordinates
        ]

        label_df = dataframe[dataframe['label_predicted'] == label_index]

        first_detections = []

        for row, col in row_col_coffee_labels:
            detections = label_df.loc[(label_df['image row'] == row) & (label_df['image col'] == col)]
            first_detections.append((row, col, detections.year.agg("min")))

    return first_detections
```

### 11.3.8 downloadMap.py

Voici comment on peut exporter les images des cartes du Vietnam de Landsat 8 à l'aide de ce script :

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

# Landsat 8 collection
SATELLITE_DATASET = "LANDSAT/LC08/C02/T1_L2"

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

# Exemple : Whole year, 2014
image_collection = create_image_collection('2014-01-01', '2015-01-01')

median_img = image_collection_to_median_img(image_collection, "Vietnam")
export_task = create_export_task(median_img, 'Vietnam_2014_whole_year_collection2')

# 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 «mask\_clouds», 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,
    and here for the collection 2 QA_PIXEL bits:
    https://developers.google.com/earth-engine/datasets/catalog/LANDSAT\_LC08\_C02\_T1\_L2
    :param image: the image
    :return: image without clouds
    """

    # Bits 3 and 4 are cloud shadow and cloud, respectively.
    clouds_bit_mask = (1 << 3)
    cloud_shadow_bit_mask = (1 << 4)

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

    # 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)
```

### 11.3.9 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:
    print("Merging raster files...")
    raster_files = open_raster_files(folder_name)
    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
    })

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

    # Write merged raster to disk
    with rasterio.open(
        os.path.join(DATA_ROOT_PATH, folder_name, 'merged.tif'), "w", **metadata
    ) as dest:
        dest.write(merged_result)

    print("Finished writing to disk")

    time.sleep(45) # sleep a little to let memory usage go down
```

Cette façon de faire est coûteuse en mémoire vive. Il serait préférable de traiter le fichier de raster par blocs au lieu de charger l'entièreté des fichiers de raster en mémoire pour les traiter<sup>3</sup>. C'est d'ailleurs pour cela qu'un délai de 45 secondes est ajouté, cela permet de faire redescendre l'utilisation de la mémoire entre chaque fusion.

Il serait aussi possible de traiter les fichiers de rasters sans les fusionner. Avoir des fichiers fusionnés est juste plus agréable à utiliser que de devoir chaque fois spécifier tous les fichiers de rasters et de devoir les traiter un par un.

### 11.3.10 config.py

Ce fichier définit les chemins d'accès qui sont ensuite utilisés dans les scripts python et les notebooks. Voici quelle configuration a été utilisée dans ce projet :

```
DATA_ROOT_PATH = os.path.abspath("../data/")
DATASET_PATH = os.path.join(DATA_ROOT_PATH, 'datasets')
DATASET_IMAGES_PATH = os.path.join(DATA_ROOT_PATH, DATASET_PATH, 'images/')
RASTER_CHUNKS_PATH = os.path.join(DATA_ROOT_PATH, 'raster_chunks/')

MODEL_ROOT_PATH = os.path.abspath('../models/')
DISTRICTS_PATH = os.path.join(DATA_ROOT_PATH, 'districts', 'diaphantinh.geojson')
SOILMAP_PATH = os.path.join(DATA_ROOT_PATH, 'soilmap', 'soilmap.geojson')
SHAPEFILE_ROOT_PATH = os.path.join(DATA_ROOT_PATH, 'labels/')
```

<sup>3</sup> Voir : <https://gis.stackexchange.com/questions/348925/merging-rasters-with-rasterio-in-blocks-to-avoid-memoryerror>

## 12 Conclusion

Durant ce 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 [60]» ou «The all convolutional net [62]». Découvrir le méta-apprentissage avec Reptile a aussi été un point intéressant, même si j'espérais à obtenir de meilleurs résultats.

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. J'ai aussi comparé les performances de diverses architectures de réseaux de neurones, de multiples combinaisons de bandes de fréquences, de modèles avec un ensemble de labels différents, de modèles à plusieurs sorties et en utilisant la technique Reptile de méta-apprentissage. Enfin, j'ai mis le modèle final à l'épreuve en effectuant des prédictions sur la région montagneuse centrale du Vietnam et en tentant de montrer où de la déforestation pourrait avoir eu lieu.

Les performances du modèle final ne permettent pas d'utiliser les prédictions telles quelles. Toutefois, les résultats obtenus peuvent aider à détecter de la déforestation.

Je pense qu'il doit être difficile d'améliorer nettement les résultats avec cette résolution d'image. Peut-être qu'avec encore plus de points géoréférencés les performances s'en verraient améliorées.

Ce travail de bachelor répond aux différents points du cahier des charges. Des améliorations seraient encore possibles. En effet, il serait intéressant d'utiliser les quelques nouveaux points géoréférencés du Vietnam récemment reçus, car actuellement le jeu de donnée du Vietnam a été inchangé entre le début et la fin du projet. Une approche plus orientée objet pourrait être bénéfique pour réusiner le code, notamment pour ce qui est de la différenciation entre les différentes méthodes utilisant soit un modèle à une sortie soit un modèle à plusieurs sorties. Enfin, une prédiction de la carte du Vietnam en décalant d'un seul pixel à la fois les images pourrait donner une meilleure résolution pour la prédiction.

J'espère que les expériences réalisées dans ce travail de bachelor ont pu être utiles pour faire avancer la recherche sur la détection de la déforestation et que ce rapport pourra aussi aider pour de futur projets.

## 13 Remerciements

J'aimerais particulièrement remercier toutes les personnes qui m'ont aidé durant ce projet ; Pr. Andres Perez-Uribe qui a supervisé ce travail de bachelor et qui m'a apporté son aide tout le long du projet, Pr. Hector Fabio Satizabal Mejia qui a lui aussi été présent pendant tout le projet, m'apportant de précieux conseils, ainsi qu'Axel Brian Fahy qui a aussi participé aux réunions. Je voudrai aussi remercier M. Louis Reymondin et M. Thibaud Vantalon de l'équipe du Vietnam qui se sont tenus disponibles pour répondre à mes questions. Je tiens aussi à remercier l'étudiant en master Romain Capocasale qui a participé à une partie des réunions et a pris le temps de répondre à mes questions, sur son implémentation de la validation croisée spatiale, et qui m'a fait connaître la fonction de perte «focal loss». Enfin, je voudrais remercier ma famille qui m'a soutenu et qui m'a aidé à relire ce travail de bachelor.

# 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/](https://alliancebioversityciat.org/news_and_blogs/toward-a-sustainable-coffee-future-in-vietnams-central-highlands/) (consulté le 6 mai 2021)
- [5] Thomas Enters & Akiko Inoguchi. « 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](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](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] Gohlke C. *tifffile: Read and write TIFF files* [En ligne]. Disponible sur : <https://www.lfd.uci.edu/~gohlke/> (consulté le 15 juillet 2021)
- [24] « GEOS ». Disponible sur : <https://trac.osgeo.org/geos/> (consulté le 1 mai 2021)
- [25] « The Shapely User Manual — Shapely 1.7.1 documentation ». Disponible sur : <https://shapely.readthedocs.io/en/stable/manual.html> (consulté le 1 mai 2021)
- [26] 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)
- [27] « scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation ». Disponible sur : <https://scikit-learn.org/stable/> (consulté le 1 mai 2021)
- [28] « Albumentations ». Disponible sur : <https://albumentations.ai/> (consulté le 25 mai 2021)
- [29] « NumPy ». Disponible sur : <https://numpy.org/> (consulté le 1 mai 2021)

- [30] « pandas - Python Data Analysis Library ». Disponible sur : <https://pandas.pydata.org/> (consulté le 1 mai 2021)
- [31] « GeoPandas 0.9.0 — GeoPandas 0.9.0 documentation ». Disponible sur : <https://geopandas.org/> (consulté le 1 mai 2021)
- [32] « Matplotlib: Python plotting — Matplotlib 3.4.1 documentation ». Disponible sur : <https://matplotlib.org/> (consulté le 1 mai 2021)
- [33] « seaborn: statistical data visualization — seaborn 0.11.1 documentation ». Disponible sur : <https://seaborn.pydata.org/> (consulté le 1 mai 2021)
- [34] « math — Mathematical functions — Python 3.9.4 documentation ». Disponible sur : <https://docs.python.org/3/library/math.html> (consulté le 1 mai 2021)
- [35] « 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)
- [36] « os — Miscellaneous operating system interfaces — Python 3.9.6 documentation ». Disponible sur : <https://docs.python.org/3/library/os.html> (consulté le 15 juillet 2021)
- [37] « sys — Paramètres et fonctions propres à des systèmes — Documentation Python 3.9.6 ». Disponible sur : <https://docs.python.org/fr/3/library/sys.html> (consulté le 15 juillet 2021)
- [38] « 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)
- [39] 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)
- [40] Gillies S. *geojson: Python bindings and utilities for GeoJSON* [En ligne]. Disponible sur : <https://github.com/jazzband/geojson> (consulté le 4 mai 2021)
- [41] Comber S. *spacv: Spatial cross-validation in Python* [En ligne]. Disponible sur : <https://github.com/SamComber/spacv> (consulté le 4 mai 2021)
- [42] « Project Jupyter ». Disponible sur : <https://www.jupyter.org> (consulté le 11 mai 2021)
- [43] Team IP. D. *ipywidgets: IPython HTML widgets for Jupyter* [En ligne]. Disponible sur : <http://ipython.org> (consulté le 16 juillet 2021)
- [44] « Google Earth Engine ». Disponible sur : <https://earthengine.google.com> (consulté le 26 avril 2021)
- [45] « Vietnam - Climate | Britannica ». Disponible sur : <https://www.britannica.com/place/Vietnam/Climate> (consulté le 9 mars 2021)
- [46] « Landsat Surface Reflectance ». Disponible sur : [https://www.usgs.gov/core-science-systems/nli/landsat/landsat-surface-reflectance?qt-science\\_support\\_page\\_related\\_con=0#qt-science\\_support\\_page\\_related\\_con](https://www.usgs.gov/core-science-systems/nli/landsat/landsat-surface-reflectance?qt-science_support_page_related_con=0#qt-science_support_page_related_con) (consulté le 16 juillet 2021)

- [47] « How do I use a scale factor with Landsat Level-2 science products? ». Disponible sur : [https://www.usgs.gov/faqs/how-do-i-use-a-scale-factor-landsat-level-2-science-products?qt-news\\_science\\_products=0#qt-news\\_science\\_products](https://www.usgs.gov/faqs/how-do-i-use-a-scale-factor-landsat-level-2-science-products?qt-news_science_products=0#qt-news_science_products) (consulté le 16 juillet 2021)
- [48] « Harvest Seasons - AmaRin Coffee - Vietnam Coffee Roaster and Supplier ». Disponible sur : <http://amarin.com.vn/coffee-harvest-season> (consulté le 12 mars 2021)
- [49] *ap301e.pdf* [En ligne]. Disponible sur : <http://www.fao.org/3/ap301e/ap301e.pdf> (consulté le 12 mars 2021)
- [50] « Provincial administration of Vietnam - OD Mekong Datahub ». Disponible sur : <https://data.opendvelopmentmekong.net/en/dataset/a-phn-tnh?type=dataset> (consulté le 27 avril 2021)
- [51] « Soil types of Vietnam - Datasets - OD Mekong Datahub ». Disponible sur : <https://data.opendvelopmentmekong.net/en/dataset/soil-types-in-vietnam?type=dataset> (consulté le 27 avril 2021)
- [52] 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)
- [53] 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)
- [54] « Learning from imbalanced data. » Jeremy Jordan [En ligne]. 2018. Disponible sur : <https://www.jeremyjordan.me/imbalanced-data/> (consulté le 28 avril 2021)
- [55] « Softmax Function ». DeepAI [En ligne]. 2019. Disponible sur : <https://deeppai.org/machine-learning-glossary-and-terms/softmax-layer> (consulté le 28 avril 2021)
- [56] « 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)
- [57] *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)
- [58] 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)
- [59] Team K. « Keras documentation: SpatialDropout2D layer ». Disponible sur : [https://keras.io/api/layers/regularization\\_layers/spatial\\_dropout2d/](https://keras.io/api/layers/regularization_layers/spatial_dropout2d/) (consulté le 7 mai 2021)

- [60] 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)
- [61] 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)
- [62] 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)
- [63] « Broadband Greenness ». Disponible sur : [https://www.l3harrisgeospatial.com/docs/broadbandgreenness.html#enhanced\\_VI](https://www.l3harrisgeospatial.com/docs/broadbandgreenness.html#enhanced_VI) (consulté le 24 mai 2021)
- [64] « Modified Normalized Difference Water Index (MNDWI) | Space4Water Portal ». Disponible sur : <http://space4water.org/taxonomy/term/1246> (consulté le 25 mai 2021)
- [65] « IDB - Index: Enhanced Vegetation Index 2 ». Disponible sur : <https://www.indexdatabase.de/db/i-single.php?id=237> (consulté le 25 mai 2021)
- [66] He C., Shi P., Xie D., Zhao Y. « Improving the normalized difference built-up index to map urban built-up areas using a semiautomatic segmentation approach ». *Remote Sensing Letters* [En ligne]. décembre 2010. Vol. 1, n°4, p. 213-221. Disponible sur : <<https://doi.org/10.1080/01431161.2010.481681>>
- [67] Baranovskij A. « Multi-Output Model with TensorFlow Keras Functional API ». *Medium* [En ligne]. 2020. Disponible sur : <https://towardsdatascience.com/multi-output-model-with-tensorflow-keras-functional-api-875dd89aa7c6> (consulté le 18 juin 2021)
- [68] Bressan R. « Building a multi-output Convolutional Neural Network with Keras ». *Medium* [En ligne]. 2020. Disponible sur : <https://towardsdatascience.com/building-a-multi-output-convolutional-neural-network-with-keras-ed24c7bc1178> (consulté le 18 juin 2021)
- [69] « Keras: Multiple outputs and multiple losses - PyImageSearch ». Disponible sur : <https://www.pyimagesearch.com/2018/06/04/keras-multiple-outputs-and-multiple-losses/> (consulté le 18 juin 2021)
- [70] Lin T.-Y., Goyal P., Girshick R., He K., Dollár P. « Focal Loss for Dense Object Detection ». *arXiv:1708.02002 [cs]* [En ligne]. 7 février 2018. Disponible sur : <http://arxiv.org/abs/1708.02002> (consulté le 24 juin 2021)
- [71] « What is Focal Loss and when should you use it? ». *Committed towards better future* [En ligne]. 2020. Disponible sur : <https://amaarora.github.io/2020/06/29/FocalLoss.html> (consulté le 24 juin 2021)
- [72] Goldblum M., Reich S., Fowl L., Ni R., Cherepanova V., Goldstein T. « Unraveling Meta-Learning: Understanding Feature Representations for Few-Shot Tasks ». *arXiv:2002.06753 [cs, stat]* [En ligne]. 1 juillet 2020. Disponible sur : <http://arxiv.org/abs/2002.06753> (consulté le 6 juillet 2021)

- [73] « Reptile: A Scalable Meta-Learning Algorithm ». *OpenAI* [En ligne]. 2018. Disponible sur : <https://openai.com/blog/reptile/> (consulté le 1 juillet 2021)
- [74] Team K. « Keras documentation: Few-Shot learning with Reptile ». Disponible sur : <https://keras.io/examples/vision/reptile/> (consulté le 23 juillet 2021)
- [75] « umbertogriffo/focal-loss-keras: Binary and Categorical Focal loss implementation in Keras. » *GitHub* [En ligne]. Disponible sur : <https://github.com/umbertogriffo/focal-loss-keras> (consulté le 24 juillet 2021)
- [76] Mike Clark. « Keras weighted categorical\_crossentropy ». *Gist* [En ligne]. Disponible sur : <https://gist.github.com/wassname/ce364fddfc8a025bfab4348cf5de852d> (consulté le 24 juillet 2021)

## 14 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 prématué

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: Seconde comparaison de combinaisons de canaux

Canaux	nb paramètres	précision moyenne	rappel moyen	f-score moyen	f-score du café
All bands	76'426	76.84%	80.31%	78.45%	82.90%
COASTAL_AEROSOL, BLUE, GREEN, RED, NIR, SWIR1, SWIR2, MNDWI, EVI2, BU	76'718	75.29%	79.67%	77.27%	81.58%
COASTAL_AEROSOL, BLUE, GREEN, RED, NIR, SWIR1, SWIR2	75'842	75.26%	79.44%	77.14%	81.68%
BLUE, GREEN, RED, NIR, SWIR1, SWIR2, MNDWI, EVI2, BU	76'426	75.01%	78.75%	76.70%	81.55%
BLUE, GREEN, RED, NIR, SWIR1, SWIR2	75'550	74.77%	78.74%	76.58%	80.96%
BLUE, SWIR2, NDVI, MNDWI	74'966	71.46%	75.41%	73.25%	77.40%
NIR, RED, SWIR1	74'674	68.00%	74.70%	70.68%	76.70%
BLUE, SWIR2, MNDWI, EVI2, BU	75'258	59.35%	67.36%	62.28%	61.67%

Table 4: 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  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.
02.03.21	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 expliquer 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, essay 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 tableau
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
18.05.21	Réunion, sauvegarde des modèles, refactor
19.05.21	Refactor, sauvegarde des modèles en s'assurant que seul le meilleur modèle est

	sauvegardé même entre les différentes validations croisées, ajouter albummentations pour avoir de l'augmentation d'image qui accepte les images multispectrales (et amélioration de la rapidité).
21.05.21	Comparaison de canaux, essai avec NDVI et MNDWI
24.05.21	Recherche sur les modèles permettant d'inclure une notion de temps (CNN-LSTM, CNN3D, ...), continuer à comparer les canaux et ajout de nouveaux canaux calculés
25.05.21	Reporter les résultats des comparaisons des canaux dans un tableau, tester avec des rotations de 90 degrés dans l'augmentation des données, Avancer le rapport, commencer à essayer de mettre en place la validation spatiale de Romain
26.05.21	Commencer à essayer de mettre en place la validation spatiale de Romain
27.05.21	Commencer à essayer de mettre en place la validation spatiale de Romain
28.05.21	Arriver à séparer en trois ensemble entraînement, validation et test en suivant la même procédure que Romain, mais avec plus que deux classes
31.05.21	Commencer à adapter la pipeline pour avoir les images sur le disque et les chemins d'accès dans le dataset
01.06.21	Réparer l'erreur de sélection de bandes des rasters téléchargés avec la collection Surface Reflectance (ordre des bandes après panchromatic et cirrus différent, car ils sont absents)
02.06.21	Réparer l'erreur de sélection de bandes des rasters téléchargés avec la collection Surface Reflectance (ordre des bandes après panchromatic et cirrus différent, car ils sont absents)
04.06.21	Mettre à jour la pipeline pour pouvoir avoir des jeux de données sauvegardés sur le disque ne contenant que les chemins d'accès vers les fichiers et non les valeurs des images
05.06.21	Mettre à jour la pipeline pour pouvoir avoir des jeux de données sauvegardés sur le disque ne contenant que les chemins d'accès vers les fichiers et non les valeurs des images
07.06.21	Faire l'entraînement avec la validation spatiale
08.06.21	Réunion, corriger l'erreur dans le callback du f1-score adapté pour prendre des générateurs de données, réaliser l'expérience de valider avec les données de 2019 un modèle entraîné sur toutes les données de 2018
09.06.21	Assister à la présentation du travail de master de Romain Capocasale
10.06.21	Téléchargements de données de différentes années, refaire la validation spatiale maintenant que le callback de f1-score est corrigé
11.06.21	Comparaison d'un même modèle entraîné sur différentes plages de temps, commencer la mise en place du procédé pour obtenir des résultats sur des cartes entières
12.06.21	Remplacer le système de génération de fold par StratifiedKFold qui génère des folds en essayant de préserver la proportion de chaque classes dans les folds
14.06.21	Mise en place des prédictions de 2014 à 2021 sur janvier à avril et récupération des résultats. Affichage de graphe pour le total de chaque label en fonction des

	années.
15.06.21	Réunion, dessin de la carte avec les endroits où le café est prédit et le endroits connus de café
16.06.21	Dessin de la carte avec la première prédiction des labels de café connus en différentes couleurs suivant l'année
17.06.21	Commencer le modèle de prédiction à plusieurs sorties (labels, catégories)
21.06.21	Problème de dépendances python, avancer sur le modèle à plusieurs sorties, problème avec la pondération des classes avec un modèle à plusieurs sorties (pas supporté dans la version de tensorflow utilisée, il faut définir une fonction de «loss» maison)
22.06.21	Réunion, continuer à réparer les problèmes de dépendances
23.06.21	Ajouter une fonction de loss «categorical cross entropy» qui prend les poids des classes au modèle de multi-sorties
24.06.21	Entraîner le modèle à plusieurs sorties avec la fonction de loss «focal loss», adapter la prédiction sur rasters pour le modèle à plusieurs sorties
25.06.21	Adapter la prédiction sur rasters pour le modèle à plusieurs sorties et lancer une nouvelle fois la prédiction sur les rasters de 2014 à 2021, dessin de la cartes avec les catégories et choix de l'année avec un widget
28.06.21	Utiliser imshow de matplotlib au lieu d'ipyccanvas pour afficher les visualisation de cartes
29.06.21	Faire la visualisation de la forêt remplacée par de la culture en prenant comme référence la prédiction de 2017
30.06.21	Courte réunion, faire la visualisation de la forêt remplacée par du café en prenant comme référence la prédiction de 2017
01.07.21	Réunion avec l'équipe du Vietnam, peaufiner les projets QGIS et compressions des fichiers de rasters (method=zstd, predictor=3) pour les envoyer à l'équipe du Vietnam, Lire sur la méthode de pré-entraînement «REPTILE»
02.07.21	Commencer à regarder les jeux de données des autres régions et à les télécharger.
05.07.21	Réunion avec Thibaud Vantan de l'équipe du Vietnam, essai du modèle à plusieurs sorties avec la correction par rapport à la remise à l'échelle, essai avec la collection 2 de Landsat 8 (les résultats se ressemblent)
06.07.21	Réunion, continuer à lire sur Reptile et le modèle agnostique de mét-apprentissage (MAML), télécharger des données des différentes régions (le premier trimestre)
07.07.21	Commencer l'implémentation de Reptile
08.07.21	Implémentation de Reptile.
12.07.21	Corriger des erreurs dans l'implémentation de Reptile, continuer sur le rapport
13.07.21	Réunion avec Pr. Perez-Uribe et Ludovic Delafontaine, qui va devoir faire un projet similaire, où l'on a discuté de mon TB, Discussion avec Pr. Satizabal Mejia Hector Fabio pour trouver l'erreur qui pouvait se trouver dans mon implémentation de Reptile,

	Faire un graphe des erreurs de chaque dataset pour Reptile et essayer d'améliorer les performances.
14.07.21	Continuer à entraîner le modèle Reptile, avancer sur le rapport
15.07.21	Continuer à entraîner le modèle Reptile, avancer sur le rapport, mettre à jour la classification des labels en catégories pour que les labels dans les nouveaux jeux de données puissent être catégorisés
16.07.21	Continuer à entraîner le modèle Reptile, avancer sur le rapport
17.07.21	Relancer l'expérience de validation spatiale, relancer l'expérience avec tous les labels (avec les labels mis-à-jour), relancer l'expérience avec le modèle à plusieurs sorties (avec les labels et catégories mis-à-jour)
19.07.21	Avancer sur le rapport, relancer l'expérience Reptile avec une architecture différente, accès au serveur de calcul de l'HEIG
20.07.21	Réunion pour faire le point, faire un essai avec Reptile en gelant les couches pré-entraînées
21.07.21	Continuer de faire des expériences avec Reptile (geler les couches par exemple) et regrouper les chemins d'accès dans un fichier config.py
22.07.21	Changer tous les endroits où les chemins d'accès dépendaient du système d'exploitation, utiliser le fichier config.py dans tous les notebooks au lieu de définir dans chaque notebooks les chemins d'accès, essayer de faire tourner un notebook depuis un serveur de calcul, avancer sur le rapport
23.07.21	Avancer sur le rapport, relancer certaines expériences avec un ensemble de label légèrement différent, commenter et améliorer le code
24.07.21	Avancer sur le rapport, commenter le code
26.07.21	Avancer sur le rapport
27.07.21	Avancer sur le rapport, corriger une erreur qui contenait la prédiction des rasters, lancer les prédictions finales
28.07.21	Avancer sur le rapport, regarder si la prédiction de la déforestation correspond à de la réelle déforestation en comparant à différentes dates avec google earth pro.
29.07.21	Corriger et relire le rapport, finir l'affiche, mettre en ordre le dossier visualizations, signer et rendre le TB