

Travail de Bachelor

L'intelligence artificielle au service de l'agriculture durable

Non confidentiel



Étudiant :

Julien Rod

Travail proposé par :

Louis Reymondin

International Center for Tropical Agriculture

CIAT - International Center for Tropical Agriculture

Vietnam

Enseignant responsable :

Andres Perez-Uribe

Année académique

2019-2020

Ropraz, le 7 août 2020

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 des diverses sources d'information pour enrichir les modèles. Par exemple, exploitation des données d'imagerie multispectrale (p.ex. diverses sources satellites, etc...).
5. Entraînement de modèles de classification à l'aide des techniques de Machine Learning, notamment des réseaux convolutifs, analyses des performances.
6. Développement d'outils permettant la visualisation et l'analyse de résultats.

Département TIC
Filière Informatique
Orientation Ingénierie Logicielle
Étudiant Julien Rod
Enseignant responsable Andres Perez-Uribé

Travail de Bachelor 2019-2020

L'intelligence artificielle au service de l'agriculture durable

Centre International pour l'Agriculture Tropical

Résumé publiable

Actuellement, la forêt vietnamienne comme beaucoup d'autres, se voit gentiment remplacée par des plantations hétéroclites. Ce projet, en collaboration avec le Centre de recherche International en Agriculture Tropicale, vise à endiguer ce phénomène.

L'objectif principal de ce projet est d'entraîner des réseaux de neurones pour qu'ils puissent distinguer les champs de café des forêts et, dans la mesure du possible, des autres cultures. Un objectif secondaire est d'arriver à faire la différence entre des champs de café et de poivre.

J'ai utilisé principalement les images satellites de Sentinel-2 (ESA) découpées en de plus petites images de 15 sur 15 pixels et les coordonnées fournies par le CIAT pour entraîner des réseaux de neurones convolutifs à reconnaître les champs de café.

Un autre programme créera une image correspondant à la prédiction d'une image de Sentinel-2. Cette image sera constituée de carré de 15 sur 15 pixels dont la couleur indique la présence supposée ou non de champs de café.

J'ai aussi exploré les images de Sentinel-1.

Les meilleurs modèles pour Sentinel-2 sont ceux utilisant une combinaison utilisant des bandes infra-rouges. Ici le rappel est proche de 90% si on utilise des images de la zone que nous souhaitons tester dans notre modèle (sinon ce dernier est variable).

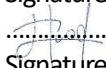
Pour Sentinel-1, une combinaison des bandes VV et VH paraissent être un bon choix. Il semble aussi qu'utiliser des bandes prises en automne dans le modèle améliore la prédiction. Cependant le rappel est assez variable (de 40 à 60% seulement).

Par la suite, nous pouvons encore tester plusieurs modèles, comme par exemple faire un modèle utilisant Sentinel-1 et Sentinel-2 en même temps ou encore inclure d'autres types de données.

Je conseille aussi d'ajouter au set d'entraînement le plus de diversité possible que ce soit au niveau du café comme par exemple mettre des champs cultivés différemment, ou au niveau du reste comme d'autres cultures, de la forêt, des villes ou des zones désertiques.

Étudiant :
Rod Julien
Enseignant responsable
Perez-Uribe Andres
Nom de l'entreprise/institution
Reymondin Louis

Date et lieu :
Ropraz, le 6 août 2020
Date et lieu :
.....
Date et lieu :
.....

Signature :

Signature :
.....
Signature :
.....

Préambule

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

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

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

HEIG-VD

Vincent Peiris
Chef du Département TIC

Yverdon-les-Bains, le 6 juillet 2020

Table des matières

Cahier des charges.....	2
Résumé publiable	3
Préambule	5
Table des matières.....	6
1 Introduction	9
1.1 Énoncé	9
2 Données utilisées.....	10
2.1 Sentinel-2.....	10
2.2 Sentinel-1.....	11
2.3 QGIS	11
2.4 Données du CIAT	12
2.5 Python.....	13
2.5.1 Numpy	13
2.5.2 Rasterio.....	13
2.5.3 Sklearn	13
2.5.4 Keras	13
2.5.5 TensorFlow	13
2.5.6 Affine	14
2.5.7 Matplotlib	14
2.5.8 GDAL	14
2.5.9 Pyproj.....	14
2.5.10 SentinelSat.....	14
2.5.11 Pyshp.....	14
2.5.12 Sympy	14
2.6 Jupyter	15
3 Ressources utilisées	15
3.1 Réseaux de neurones convolutifs.....	15
3.2 Optimiseur Adam.....	16
4 Travail effectué	16
4.1 Recherche d'information.....	16
4.2 Création des données d'entraînement.....	16
4.3 Premier réseau de neurones convolutif	17
4.4 Tentative de prédiction	19

4.5	Et Sentinel-1 ?	20
4.6	Téléchargement automatique des images	20
4.7	Nouveau réseau de neurones.....	20
4.8	Expériences annexes.....	22
4.8.1	Entraîner sur une zone et tester sur les autres	22
4.8.2	Division d'image 2.0.....	23
4.8.3	Café contre poivre	23
4.9	Au-delà des saisons	26
4.9.1	Automne	27
4.9.2	Hiver.....	28
4.9.3	Printemps.....	29
4.9.4	Conclusions.....	29
4.10	Autres expériences avec Sentinel-2.....	34
4.10.1	Entraîner sur une zone et tester sur les autres 2.0	34
4.10.2	Café contre poivre 2.0	36
4.10.3	Tester une année avant	40
4.11	Sentinel-1 dans le temps	45
5	Description des scripts réalisés.....	48
5.1	Traitement des images	48
5.1.1	DownloadImages.ipynb	48
5.1.2	DivideAndSort.ipynb.....	48
5.1.3	DivideAndSort-FallWinterSpring.ipynb.....	51
5.1.4	DivideAndSort-Final.ipynb	51
5.1.5	TranslateS1Bands.ipynb	52
5.1.6	DivideAndSortS1.ipynb.....	54
5.1.7	DivideAndSortS1-AllSeasons.ipynb.....	57
5.2	Entraînement.....	58
5.2.1	Train.ipynb	58
5.2.2	Train-FallWinterSpring.ipynb.....	60
5.2.3	Train_one_test_others-FallWinterSpring.ipynb.....	61
5.2.4	Train_two_test_others-FallWinterSpring.ipynb.....	62
5.2.5	Train-S1-AllSeasons.ipynb	62
5.2.6	CoffeeVSPepperFallWinterSpring.ipynb.....	63
5.3	Test	63
5.3.1	CompareOneYearAgo.ipynb	63

5.4	Prédiction.....	64
5.4.1	Predict.ipynb.....	64
5.4.2	Predict-FinalVersion.ipynb.....	64
5.4.3	Predict-Multi-FinalVersion.ipynb.....	65
5.4.4	PredictS1-FinalVersion.ipynb.....	65
6	Améliorations et expériences possibles	66
7	Marche à suivre	67
7.1	Entraînement.....	67
7.1.1	Télécharger des images depuis le site de l'ESA.....	67
7.1.2	Diviser les images téléchargées en plus petites images.....	69
7.1.3	Sélectionner les données de tests	69
7.1.4	Entraîner le réseau	70
7.2	Prédiction.....	70
7.3	Schématisation	70
8	Remarques et conclusion.....	71
Références	72	
Authentification.....	74	
Table des abréviations utilisées.....	75	
Table des illustrations.....	75	
Annexes	77	
Notebooks	77	
Modèles	78	
Images générées.....	79	
Autres	79	
Journal de travail	80	

1 Introduction

Ce travail de Bachelor consiste à entraîner un réseau de neurones pour qu'il puisse reconnaître des plantations de café. L'objectif final du projet global est de savoir si une plantation de café a remplacé ou non de la forêt.

Le projet se fait en collaboration avec le Centre International de recherche en Agriculture Tropicale (CIAT) basé au Vietnam. Ce centre de recherche international a déjà effectué certaines recherches dans la région montagneuse centrale du Vietnam surnommée Central Highlands [1]. Cette région possède de nombreuses plantations agricoles hétéroclites, dont du café.

Ici, nous utiliserons des images satellites. Plus précisément des images provenant des missions européennes Sentinel d'ESA (European Space Agency).

Le sous-chapitre ci-dessous est l'énoncé tel qu'il a été défini.

1.1 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 exploitants 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 récemment des forêts au Vietnam.

Beaucoup des 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 ou non.

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.

2 Données utilisées

Pour ce projet, nous utilisons les données des missions européennes Sentinel. Nous utilisons les images satellites produites par Sentinel-2 et explorons aussi les images de Sentinel-1. Nous pouvons nous procurer ces images gratuitement sur le site de l'ESA [2].

2.1 Sentinel-2

Sentinel-2 [3] surveille les changements de surfaces des paysages, mais ne couvre pas les latitudes au-dessus de 56° sud et au-dessous de 84° nord. Elle est aussi composée de 2 satellites qui partagent la même orbite, leurs positions sont opposées (un de chaque côté de la terre), ce qui nous donne une fréquence de visite d'une fois tous les 5 jours. Elle possède 13 bandes spectrales pour mesurer la radiance de la terre (de 1 à 12 avec 8a), chaque satellite de cette mission, a une bande de fréquences légèrement différente de l'autre, la plus grande différence est d'une vingtaine de nm à peine.

Dans ce projet nous n'utiliserons que 4 bandes. Celles qui représentent les composantes RGB de l'image, plus celle représentant les infra-rouges. Nous allons donc tester plusieurs approches pour savoir quelles sont les meilleures bandes à utiliser pour compléter la tâche qui nous a été donnée.

Sentinel-2 Bands	Central Wavelength (μm)	Resolution (m)
Band 1 - Coastal aerosol	0.443	60
Band 2 - Blue	0.490	10
Band 3 - Green	0.560	10
Band 4 - Red	0.665	10
Band 5 - Vegetation Red Edge	0.705	20
Band 6 - Vegetation Red Edge	0.740	20
Band 7 - Vegetation Red Edge	0.783	20
Band 8 - NIR	0.842	10
Band 8A - Vegetation Red Edge	0.865	20
Band 9 - Water vapour	0.945	60
Band 10 - SWIR - Cirrus	1.375	60
Band 11 - SWIR	1.610	20
Band 12 - SWIR	2.190	20

Figure 1: Bandes de sentinel-2 [4]

Le premier satellite de Sentinel-2 a été mis sur orbite mi 2015, la première image de la zone demandée disponible sur le site de Copernicus date du 9 décembre 2016.

2.2 Sentinel-1

Sentinel-1 [5] a des sondes radars proposant plusieurs résolutions (jusqu'à 5 m) et plusieurs couvertures (jusqu'à 400 km). Elle utilise la technologie SAR (Synthetic Aperture Radar ou radar à synthèse d'ouverture en français) lui permettant d'améliorer la résolution en azimut tout en ayant une antenne relativement petite et sans dépendre de la hauteur du porteur du radar. Cela lui permet de faire une cartographie des différences d'altitude entre un point et un autre. Un autre avantage est que la cartographie se fait indépendamment des nuages et de la luminosité. Elle est composée de 2 satellites (A et B) partageant la même orbite. Elle est surtout utilisée pour cartographier les océans, mais a aussi quelques missions terrestres. En temps normal, elle donnera des informations sur la zone voulue tous les 12 jours environ. Il semblerait que Sentinel-1 soit fortement utile pour la détection de la déforestation [6].

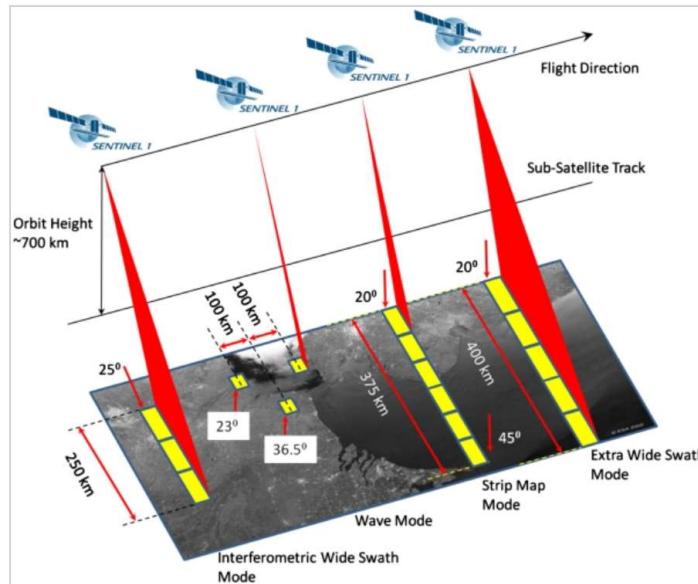


Figure 2: Modes de Sentinel-1 [7]

Le premier satellite de Sentinel-1 a été mis sur orbite en 2014, la première image de la zone demandée disponible sur le site de Copernicus date du 22 février 2015.

Sentinel-1 ne sera pas beaucoup utilisée dans ce travail de Bachelor car elle ne contient pas beaucoup d'informations comparée à Sentinel-2. De plus, même si sa capacité à voir à travers les nuages semble fortement intéressante, surtout lors de la saison des pluies, sa sensibilité au bruit et le fait que ses données deviennent chaotiques si la couverture des arbres n'est pas homogène, la rende imprécise.

2.3 QGIS

Nous utilisons le logiciel QGIS [8] qui permet non seulement de visualiser les images, mais aussi de les géolocaliser. Ainsi deux images limitrophes apparaîtront dans QGIS l'une à côté de l'autre. QGIS nous donne aussi la possibilité de mettre une image plus ou moins en transparence, ce qui nous permet notamment de superposer une image satellite avec la prédition correspondante, afin de voir à quoi ressemblent les zones qui contiennent potentiellement du café et celles qui n'en contiennent pas.

2.4 Données du CIAT

En ce qui concerne les sites sur lesquels il y aurait des plantations de café, le CIAT nous a fourni une liste des différentes observations dans la région centrale du Vietnam. Celle-ci nous permet de voir non seulement les coordonnées où du café a été planté, mais aussi d'autres coordonnées comme certaines forêts, des champs de plantations hétéroclites, etc...

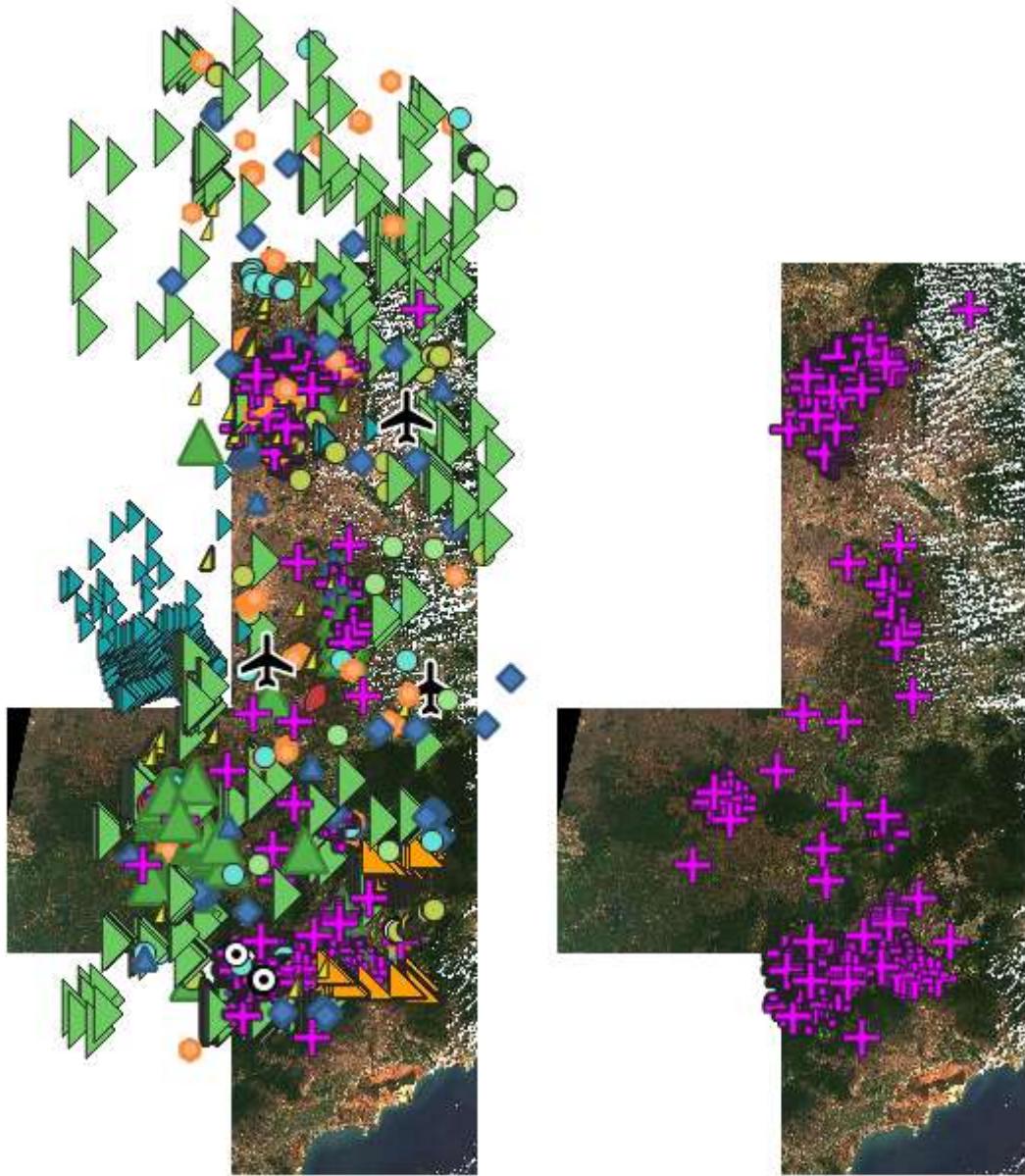


Figure 3 : À gauche, la liste de toutes les données fournies par le CIAT.
À droite, la même liste en ne gardant que les champs de café¹.

¹ Dans ce rapport, je fais souvent allusion aux différentes zones représentées par ces images. L'appellation « haut » correspond à la zone tout en haut de cette figure, « milieu » fait référence à la zone se trouvant juste en dessous, « gauche » représente la partie ressortant, « droite » est la partie à droite de gauche et « bas » symbolise la partie la plus au sud de cette figure. Il est aussi important de noter que ces zones se chevauchent un peu.

Les cartes affichées ci-dessus sont en fait 5 images de la mission Sentinel-2 affichées via le logiciel QGIS. Les icônes fournies par le CIAT, quant à elles, sont stockées dans des fichiers shp (à importer dans QGIS en tant que couche) et leurs formes sont définies dans un fichier qml (à lier à une couche en tant que style).

Le CIAT a aussi utilisé ces données pour faire leur propre réseau de neurones. Celui-ci consiste à prédire si le point central d'une image de 13 sur 13 pixels contient du café ou non.

2.5 Python

Ce travail de Bachelor étant principalement du machine learning, j'avais besoin d'utiliser un langage adapté pour ce genre de technologie. Python étant très utilisé dans ce domaine, il s'est tout naturellement imposé pour ce projet.

J'utiliserai Python 3.7 pour ce projet. Dans la suite de ce chapitre, je vous parlerai brièvement des différents modules que j'ai utilisés pour réaliser ce travail.

2.5.1 Numpy

Numpy [9] est un module python très populaire. Apportant des tableaux multidimensionnels simples à utiliser, performant et robuste. Ce module se trouve aussi sous licence open source (comme tous les modules utilisés dans ce travail).

J'utilise ce module pour la gestion des listes, surtout des données de test et d'entraînement.

2.5.2 Rasterio

Rasterio [10] est un module permettant d'ouvrir, modifier ou encore créer des fichiers de raster. Le mot raster désigne des images auxquelles on ajoute des informations de géoréférencement. Ce module me permet donc de visualiser et ouvrir les images satellites téléchargées.

Ce module est donc très utile pour travailler avec les images tiff et jp2, qui sont respectivement les extensions des images que Sentinel-1 et Sentinel-2 nous fournis. Je l'utilise aussi pour créer le fichier de sortie lors de la prédiction.

2.5.3 Sklearn

Sklearn [11] est un module pour le machine learning, utilisé pour des tâches de classifications, de régressions, de clustering, etc...

J'utilise ce module uniquement pour faire de la validation croisée pendant l'entraînement de mes modèles.

2.5.4 Keras

Keras [12] est un module servant à créer et utiliser très facilement des réseaux de neurones. Il permet aussi de créer des réseaux de neurones convolutifs, très utile pour la classification d'images. Keras est construit au-dessus de TensorFlow pour rendre la tâche de l'utilisateur plus intuitive.

J'utilise ce module pour tout ce qui est gestion de mes modèles.

2.5.5 TensorFlow

TensorFlow [13] est un module permettant de créer et gérer des réseaux de neurones de bout en bout. Il est très populaire et est aussi utilisé par Keras.

J'utilise ce module par l'intermédiaire de Keras.

2.5.6 Affine

Affine [14] est une librairie permettant de faire des calculs matriciels. Il permet notamment de faire des rotations, translations géométriques, etc... On peut aussi l'utiliser pour appliquer des transformations sur des fichiers raster.

J'utilise ce module pour pouvoir géolocaliser les membres d'une partition d'une image de Sentinel-2.

2.5.7 Matplotlib

Matplotlib [15] permet d'afficher des données sous la forme de graphique. C'est une librairie python très populaire qui collabore très bien avec Numpy.

J'utilise ce module pour afficher des éléments. Plus précisément pour afficher le résultat des entraînements de mes différents réseaux de neurones.

2.5.8 GDAL

Gdal [16] est un module qui comme Rasterio, sert à ouvrir, créer ou modifier des fichiers de type raster. J'utilise cette librairie en complément de Rasterio, car elle me permet de découper les images de Sentinel-1 de façon extrêmement simple (mais lourde).

J'utilise donc ce module pour diviser et géolocaliser correctement une image Sentinel-1 en de plus petites images en vue de l'entraînement d'un réseau de neurones.

2.5.9 Pyproj

Pyproj [17] est un module permettant de faire la conversion entre 2 systèmes de coordonnées géographiques. Nous avions déjà utilisé cette librairie lors d'un des labos de VTK et son utilisation est assez simple.

J'utilise ce module pour faire la conversion des systèmes de coordonnées ESPG (les images de Sentinel et les points donnés par le CIAT n'utilisant pas le même).

2.5.10 SentinelSat

SentinelSat [18] est un module python permettant de télécharger les images de Sentinel-1, Sentinel-2 et Sentinel-3 présentes sur le site de l'ESA [19] de façon simple et rapide. Cependant, il faut avoir un compte sur ce dit-site pour pouvoir s'y connecter (l'inscription est gratuite).

2.5.11 Pyshp

Pyshp [20] est un module permettant de travailler avec les fichiers shapefile (shp), il permet notamment de les lire ou d'en écrire de nouveau.

J'utilise ce module pour lire la liste des coordonnées labellisées donnée par le CIAT (dont les coordonnées sont au format shp) dans un script python.

2.5.12 Sympy

Sympy [21] est un module permettant de faire facilement de l'algèbre à l'intérieur d'un code python.

J'utilise ce module pour calculer les offsets utiles pour aligner plusieurs images de Sentinel-1. Plus précisément, il me sert à résoudre un système d'équations me permettant de trouver les coordonnées à une position x y donnée.

2.6 Jupyter

Dans ce projet, j'utiliserai des notebooks Jupyter [22] pour écrire et lancer mes scripts Python.

Les notebooks possédant la notion de cellule permettent notamment d'être plus rapide lors de la résolution des problèmes. Car, si nous avons bien découpé notre code, il n'y a pas besoin de relancer entièrement le programme s'il y a une erreur. Cependant, il faut bien faire attention à l'état de nos données qui dépend souvent des cellules déjà exécutées (il se peut qu'une variable déclarée en dessus d'une cellule donnée ait été modifiée par le code d'une cellule en dessous de la cellule observée). Les cellules permettent aussi d'écrire du Markdown ce qui permet une meilleure mise en page du code.

3 Ressources utilisées

3.1 Réseaux de neurones convolutifs

Dans ce projet, j'utilise des réseaux de neurones convolutifs. Un réseau de neurones convolutif est un type de machine learning par apprentissage supervisé très utilisé dans la reconnaissance d'images.

L'apprentissage supervisé consiste concrètement à entraîner un système pour qu'il puisse classer une entité selon des traits donnés. Pour schématiser nous donnons à un réseau de neurones en entrée les traits de l'entité à prédire et en sortie la classe à laquelle elle appartient, c'est-à-dire la réponse que devrait nous retourner le système. Puis, au fur et à mesure qu'on présente au système des données, il apprendra tout seul à faire la différence entre les différentes classes. Cela signifie qu'il aura trouvé des patterns permettant de décider si une entité appartient à une classe ou non, sur la base des traits passés en entrée.

Un réseau de neurones convolutif est un modèle permettant de traiter les images comme étant une matrice de pixels. En d'autres termes, le réseau de neurones est capable de reconnaître pour un pixel donné, les pixels adjacents à celui-ci et leurs voisins, contrairement à un réseau de neurones simple qui lui évaluera les pixels de l'image comme une simple liste de données perdant ainsi la notion de voisinage.

Ce type de réseau contient une partie dite de convolution servant à faire des traitements sur l'image et une seconde partie dans laquelle tous les neurones sont totalement connectés (multi-layer perceptrons).

La partie de convolution recherche des patterns dans l'image. Pour cela il utilisera ce qu'on appelle un kernel, qui est une matrice de petite taille contenant un pattern, comme par exemple, une barre verticale, horizontale, une croix, etc... Puis on parcourt tous les pixels de l'image en leur attribuant un score selon la similitude entre le kernel et le pixel observé avec son voisinage (permettant de recréer la taille du kernel). Ainsi l'image du début sera transformée en une matrice de score puis passée à la couche suivante.

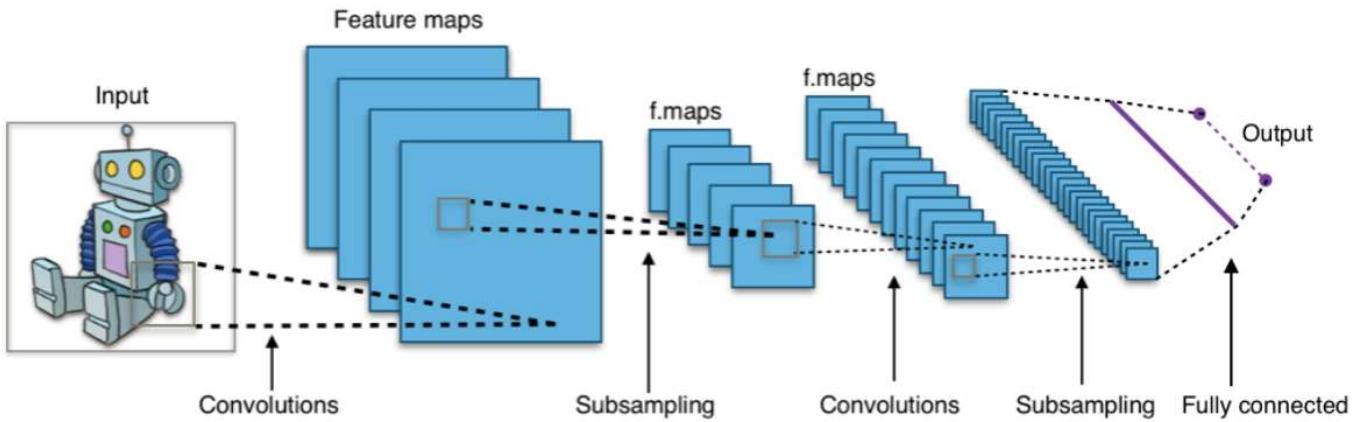


Figure 4: Réseau de neurones convolutif en image [23] (aucune modification n'a été apportée à l'image)

Généralement, on utilise une couche de « pooling » après une couche de convolution pour redimensionner l'image en une image plus petite.

Une bonne pratique consiste à faire 3 convolutions et d'utiliser une couche de dropout (couche servant à aléatoirement couper la connexion avec un neurone pour éviter que le système connaisse par cœur les données d'entraînement et soit inefficace sur de nouvelles données) avant de passer à la partie totalement connectée.

La partie totalement connectée, quant à elle, agit comme un réseau de perceptrons multicouches, c'est-à-dire qu'elle utilisera les données traitées jusqu'alors pour définir la classe de l'entité observée grâce à ses couches cachées. À noter que le système ne prend pas une matrice en entrée, mais simplement une liste de données. Pour faire la transition entre ces deux dimensions, nous utilisons une couche dite « flatten ».

3.2 Optimiseur Adam

Dans ce projet, nous utiliserons l'optimiseur Adam (adaptive moment estimation) qui permet d'adapter le taux d'apprentissage pendant l'entraînement du réseau. Il consomme aussi peu de mémoire et arrive à des bons résultats très rapidement.

Ici nous utiliserons les paramètres par défaut proposés par Keras, nous aurons donc un taux d'apprentissage de 0.001, un taux de décroissance exponentielle du premier moment à 0.9 et 0.999 pour le deuxième moment [24].

4 Travail effectué

4.1 Recherche d'information

N'ayant pas de compétence en Machine Learning au début du projet (le cours étant donné en parallèle), j'ai commencé ce travail de Bachelor en regardant la documentation concernant les missions Sentinel-2 et Sentinel-1. Puis je me suis intéressé à QGIS. Je l'ai téléchargé et ai regardé comment il fonctionnait en utilisant des images provenant des deux sentinelles utilisées dans ce projet. Je me suis donc aussi familiarisé avec le site permettant de télécharger les images des différentes missions Sentinel [19]. J'ai aussi lu un rapport montrant une mission qu'avait déjà effectuée le CIAT dans la région observée [25].

Après quelques semaines à m'informer sur ces divers sujets, j'ai commencé à lire de la documentation sur les réseaux de neurones convolutifs et ai fait quelques tutoriels [26] [27]² sur le sujet.

4.2 Création des données d'entraînement

Puis ayant reçu la liste des localisations de plantations de café et autres, j'ai créé un script python pour diviser une image de Sentinel-2 en de plus petites images, toujours géolocalisables par QGIS. Les images de Sentinel-2 étant de dimension 10980 pixels * 10980 pixels, j'ai décidé, en commun accord avec mon référent, créer des images de 90 sur 90 pour ne pas avoir de perte.

J'ai commencé par les images de Sentinel-2 car elles sont beaucoup plus simples à diviser que les images de Sentinel-1 (le type d'image retourné par Sentinel-1 et Sentinel-2 ainsi que la façon dont elles sont géolocalisées n'étant pas les mêmes).

J'ai ensuite utilisé les coordonnées des champs de café pour trouver les images qui leur correspondaient. Pour ça j'ai utilisé le logiciel QGIS, il suffit d'afficher les points sur la carte et les faire matcher avec les images créées.

² Entre autres.

Ceci m'a donné un set de données de 557 images contenant des plantations de café. J'ai donc complété mon set de données en prenant 558 images parmi celles restantes ne contenant pas de champs de café. J'ai choisi les images un peu au hasard, prenant des images dont j'étais sûr qu'elles n'en contenaient pas comme par exemple des images de mers, de forêts ou de villes et d'autres en utilisant les positions données par le CIAT comme par exemple les champs de poivres ou de riz.

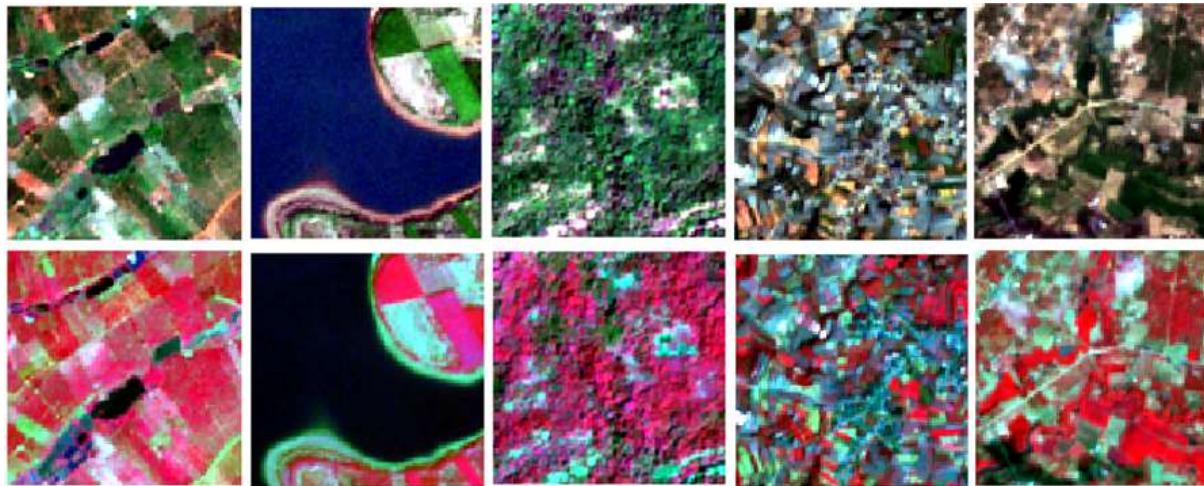


Figure 5 : De gauche à droite : Champs de café, lac, forêt, ville et image contenant des champs inconnus, le haut représente des images en RGB et le bas des images en RG+IR

4.3 Premier réseau de neurones convolutif

Après cette sélection du set de données, j'ai créé un premier réseau de neurones convolutif qui ne marchait pas très bien car il avait la même précision qu'une pièce de monnaie quand on lui mettait des couches cachées dans la partie totalement connectée du réseau (partie de fin après la convolution). S'il n'avait pas de couche cachée, le modèle arrivait à environ 75% de bonne prédiction sur les données de test.

À ce moment-là, je me suis aussi aperçu d'une chose, c'est que l'apprentissage de mon réseau de neurones était très long. Je devais donc trouver une autre solution pour accélérer le processus. Dans un premier temps, j'ai pensé à Google Colab [28] (que nous avions utilisé dans le cours de MLG), l'espace gratuit de Google nous permettant de lancer des scripts pour le machine learning, mais en lisant un peu la documentation, je m'aperçus que nous ne pouvions pas laisser le script tourner sans être connecté et que l'entraînement était limité dans le temps. Après une petite discussion avec M. Perez-Uribe, j'ai pu utiliser l'un des serveurs internes de l'école, Hyperion, qui même si je dois rester connecté en session ssh, me permet de lancer mes scripts sans limite de temps.

J'ai ensuite continué à travailler sur mon réseau de neurones convolutif. Je me suis aperçu de deux erreurs : La première était que le réseau convolutif s'attendait à recevoir des images en format « TensorFlow » c'est-à-dire qu'il devait recevoir les composantes RGB pour chaque pixel, plutôt que la liste de trois fois la même image (une par couleur) fournie par rasterio. La deuxième fut que je passais les valeurs en uint16, alors que les neurones préfèrent avoir affaire à de petites valeurs pour ne pas avoir à attribuer des petites valeurs à leur fonction d'activation. Ayant réglé ces deux problèmes, j'arrive maintenant à avoir un réseau de neurones arrivant à des prédictions au-delà de 90%.

```
def reshape(image):
    """Reformatte les données rasterio pour être utilisée par Tensorflow"""
    reshaped_image = []
    for i in range(len(image[0])):
        reshaped_row = []
        for j in range(len(image[0][0])):
            reshaped_cell = []
            reshaped_cell.append(image[0][i][j] / 65535)
            reshaped_cell.append(image[1][i][j] / 65535)
            reshaped_cell.append(image[2][i][j] / 65535)
            reshaped_row.append(reshaped_cell)
        reshaped_image.append(reshaped_row)
    return reshaped_image
```

Figure 6 : Passer une image de Rasterio à Tensorflow + normalisation des données.

À partir de ce point, j'utiliserai souvent des captures d'écran de matrice de confusion. Voici une petite explication sur comment les lire.

[[1553	469]	
		95	428]	
]			

Images de café prédictent comme étant du café

Images de café prédictent comme n'étant pas du café

Autres images prédictent comme n'étant pas du café

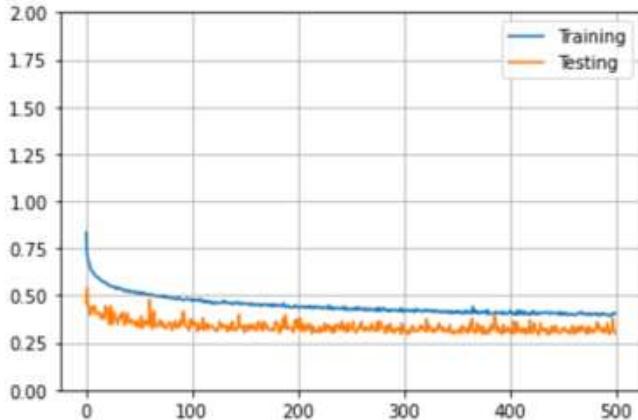
Autres images prédictent comme étant du café

Figure 7 : Comment lire les matrices de confusion pour deux classes : « café » et « pas café »

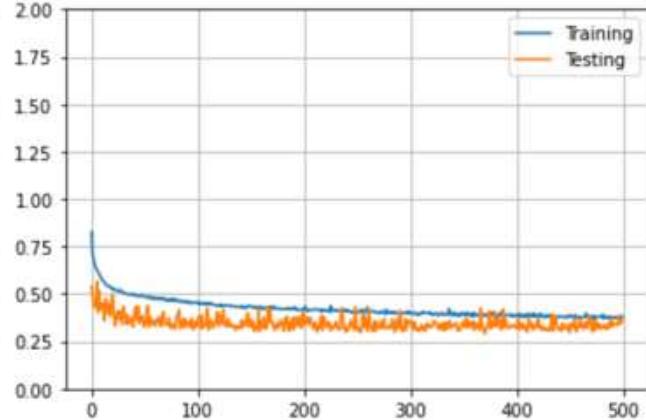
Les colonnes représentent le nombre de données prédites comme faisant partie de la classe spécifiée et les lignes représentent les classes effectives. La diagonale (haut gauche à bas droit) représente les données bien classées.

Il existe aussi des matrices de confusions plus grandes, l'ordre des classes y est toujours précisé.

Test accuracy: 0.8530055284500122



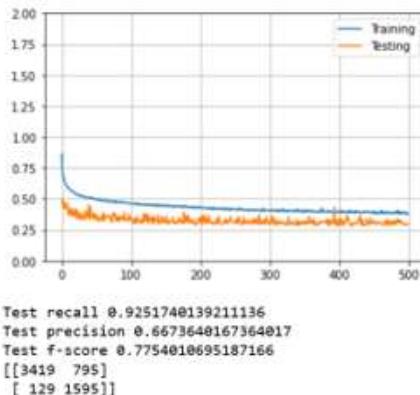
Test accuracy: 0.8045125603675842



Test recall 0.9008695652173913
Test precision 0.6888297872340425
Test f-score 0.7807083647324793
[[3512 702]
[171 1554]]

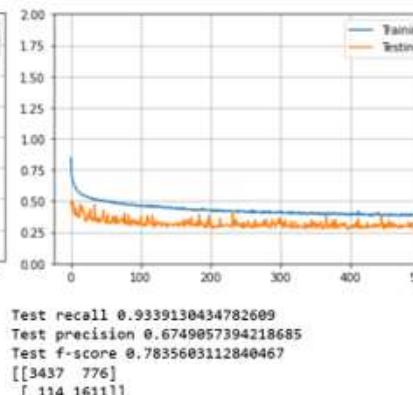
Test recall 0.9501449275362319
Test precision 0.6039056742815033
Test f-score 0.7384546068934444
[[3139 1075]
[86 1639]]

Test accuracy: 0.8443920612335205



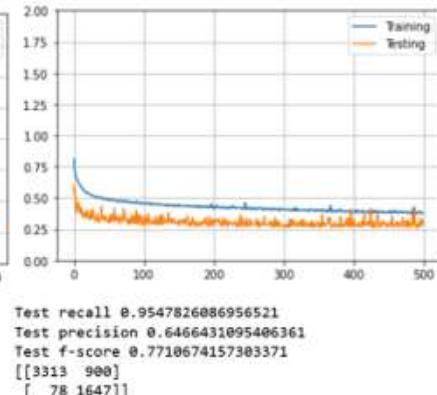
Test recall 0.9251740139211136
Test precision 0.6673648167364817
Test f-score 0.7754010695187166
[[3419 795]
[129 1595]]

Test accuracy: 0.8501178622245789



Test recall 0.9339130434782609
Test precision 0.6749057394218685
Test f-score 0.7835603112840467
[[3437 776]
[114 1611]]

Test accuracy: 0.8352980613708496



Test recall 0.9547826086956521
Test precision 0.6466431895406361
Test f-score 0.7710674157303371
[[3313 900]
[78 1647]]

Figure 8 : Visibilités, graphiques d'apprentissage et matrices de confusion d'une même validation croisée (étape 4.9)

On constate sur la figure ci-dessus que notre validation croisée est stable, c'est-à-dire que le résultat d'entraînement ne dépendra pas de l'aléatoire lors de la sélection des données d'apprentissage et de test. On peut aussi en déduire que le système arrive à trouver des différences significatives entre les images comprenant des champs et le reste du set.

4.4 Tentative de prédiction

En même temps, j'ai aussi commencé à écrire une fonction de prédiction prenant en entrée une image de Sentinel-2 et ressortant une image bicolore noir-blanc montrant le résultat de la prédiction. Utilisant mon meilleur modèle j'arrive au résultat expliqué à la page suivante.

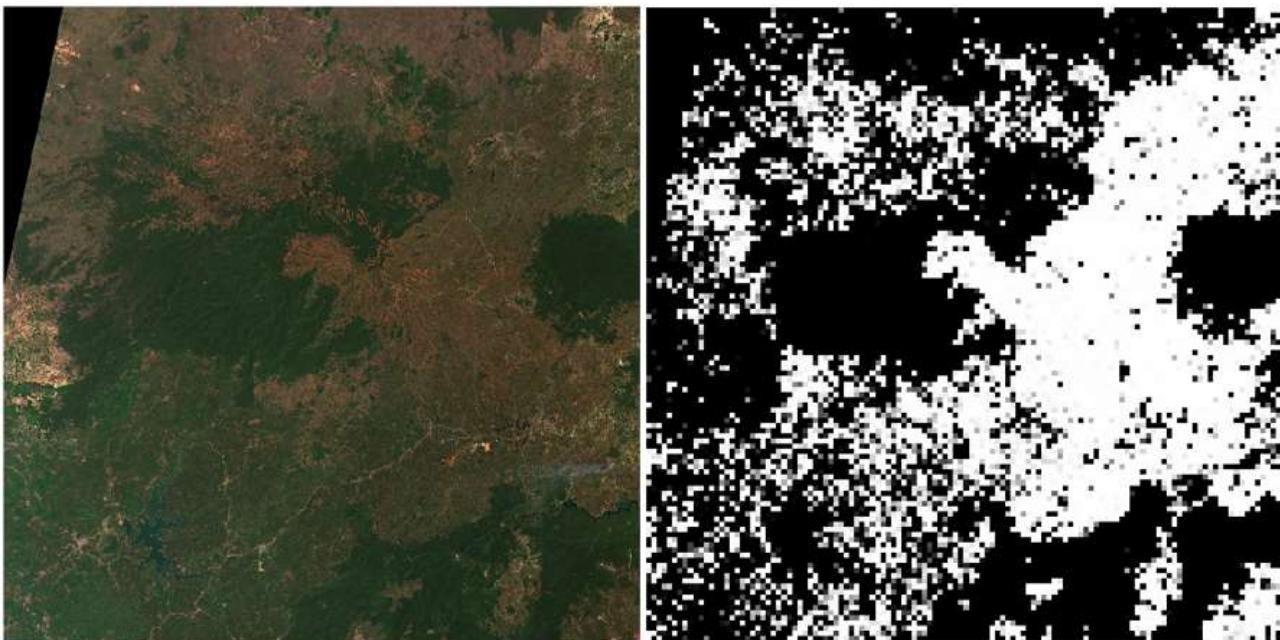


Figure 9 : Image Sentinel-2 et sa prédiction

Plus la prédiction s'approche du blanc, plus le réseau de neurones pense que le groupe de pixels observé a de chance de contenir un ou plusieurs champs de café.

On peut voir dans les images ci-dessus que notre réseau de neurones fait surtout la différence entre les parties brunes et les parties d'autres couleurs de l'image. Plus la partie observée est brune, plus il y a de chance que celle-ci soit détentrice d'un champ de café (selon notre réseau). Bien évidemment, nous ne pouvons pas savoir si c'est effectivement le cas. Toutefois, on peut conclure que ce qui est retourné comme étant potentiellement un champ n'est ni de la forêt, ni un lac, ni une ville avec une assez bonne précision.

À noter que le modèle utilisé pour faire cette prédiction était un modèle entraîné sur les données RG+IR. J'ai aussi testé mon premier réseau sur des images en RGB. Les résultats sont plus ou moins les mêmes.

4.5 Et Sentinel-1 ?

J'ai finalement réussi à diviser une image de Sentinel-1 tout en gardant la géolocalisation. J'ai aussi opté pour des images de 90 sur 90 pixels, même si nous rognons un peu l'image de base (de 8 et 58 pixels).

J'ai ensuite créé un set d'entraînement pour Sentinel-1, mais le réseau de neurones ne dépasse guère les 70% et dans la moitié des cas, il reste bloqué à 0.5 lors de l'entraînement... Cependant, une étude fournie par le CIAT m'a appris que le fait qu'il ne dépasse pas les 70% de visibilité est normal pour les données que ce satellite fournit [29].

4.6 Téléchargement automatique des images

Je me suis ensuite penché sur le point 1 du cahier des charges qui est de trouver un moyen de télécharger les données de Sentinel-2 via un script. Pour ceci j'ai simplement utilisé le module Python SentinelSat [18]. J'avais, dans un premier temps essayé de le faire via des requêtes cURL, mais ce module est plus simple d'utilisation et permet de spécifier une couverture nuageuse maximale.

4.7 Nouveau réseau de neurones

J'ai eu une vidéo-conférence avec le CIAT et M. Perez-Uribe. Lors de celle-ci j'ai pu apprendre qu'ils avaient par le passé déjà utilisé des images de Sentinel-2, mais qu'ils contrôlaient simplement si le pixel central d'une image de

13 sur 13 était un champ de café. Ils aimeraient aussi si possible que le système sache distinguer les champs de café des champs de poivres. J'ai donc modifié mon script pour qu'il découpe les images dans un format de 15 sur 15 pixels (soit une surface de 225 mètres carrés), ce qui est plus proche des expériences que le CIAT a réalisées et me permet de recouvrir la totalité d'une image de Sentinel-2. Après cela, j'ai créé un nouveau set de données d'entraînement, ce qui m'a pris à peu près une semaine.

À noter que cette fois-ci j'ai fait des images de 4 couches plutôt que de séparer les images en RGB et RG+IR, cela me permet d'économiser de la place de stockage.

Le CIAT aimerait aussi que notre réseau de neurones ait le meilleur rapport efficacité/coût. Pour cela j'ai modifié un peu mon ancien réseau de neurones pour qu'ils acceptent le nouveau format d'image, puis j'ai testé, sur un entraînement de 2000 époques, toutes les combinaisons de bandes comme noté dans le tableau ci-dessous. Le rappel, la précision et le f-score sont basés sur la classe contenant du café.

Bandes utilisées(s)	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	85.48%	80.73%	93.53%	86.66%
Bleu, Vert et Rouge	85.02%	80.62%	92.18%	85.97%
Bleu, Vert et Infra-Rouge	83.90%	80.42%	89.70%	84.68%
Bleu, Rouge et Infra-Rouge	86.14%	82.44%	90.50%	86.27%
Vert, Rouge et Infra-Rouge	85.71%	83.08%	89.61%	86.18%
Bleu et Vert	81.90%	76.59%	90.03%	82.72%
Bleu et Rouge	83.82%	77.27%	94.18%	84.87%
Bleu et Infra-Rouge	81.95%	76.28%	91.14%	82.93%
Vert et Rouge	82.07%	77.59%	90.57%	83.39%
Vert et Infra-Rouge	82.53%	77.75%	91.23%	83.85%
Rouge et Infra-Rouge	83.52%	77.23%	93.33%	84.50%
Bleu	76.50%	73.43%	84.52%	78.00%
Vert	75.07%	68.04%	93.44%	78.52%
Rouge	80.65%	75.31%	91.17%	82.41%
Infra-Rouge	76.70%	71.03%	89.96%	79.93%

Bien que le réseau utilisé ne soit pas optimal, il me permettra quand même de faire une présélection des bandes les plus utiles pour la détection.

On constate que des 4 bandes, la bande rouge est de loin la meilleure pour la prédiction. À elle seule, elle arrive environ 5% derrière la combinaison de toutes les bandes.

On peut aussi voir que la bande verte est l'une des moins utiles du lot, mais ce résultat est sans doute biaisé dû au fait que les images satellites utilisées ont été prises fin février, soit en plein hiver.

La bande bleue arrive aussi en queue de peloton, mais je suis quand même impressionné que nous arrivions à des scores aussi haut (je pensais qu'elle ne serait utilisable que pour la détection des cours d'eau et des bâtiments).

On constate aussi que le rappel tourne au minimums autour des 90% quelques soit la combinaison de bandes utilisées. Cependant, il faut minimum 3 bandes pour atteindre une précision supérieure à 80% (dans cette expérience, les classes sont bien équilibrées).

Les meilleures combinaisons de bandes sont donc rouge, infra-rouge combinées avec du vert ou du bleu. Si l'on ne souhaite que 2 bandes : bleu et rouge ou rouge et infra-rouge me paraissent être de bonnes solutions, même si les écarts avec les autres combinaisons ne sont pas très grands.

À noter que les résultats sont un peu moins bien qu'avec mon premier réseau de neurones. Ceci est dû au fait que les données d'entraînement sont plus complexes (plus petites donc forme du champ plus difficile à distinguer, présence d'autres types de champs dans le set, etc...)

4.8 Expériences annexes

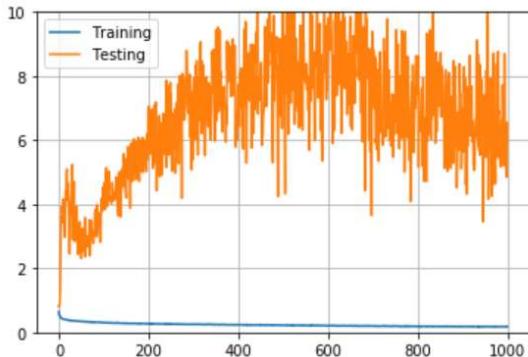
En attendant les résultats de ma sélection de bande, j'ai fait quelques expériences annexes.

4.8.1 Entrainer sur une zone et tester sur les autres

J'ai testé mon premier réseau de neurones en ne prenant que les données contenues dans une zone pour l'entraînement et en testant sur le reste des données. Comme annoncé lors de la vidéo-conférence, la visibilité descend drastiquement (même si elle reste quand même au-dessus de 80%). Ceci est dû au fait que la nature du sol n'est pas forcément le même d'une région à l'autre et au fait que les plantations de café peuvent être différentes, aussi bien au niveau de la variété de café que dans la façon de le cultiver. On constate aussi sur la capture d'écran de droite que mon système, bien qu'arrivant à séparer les différentes catégories souffre d'overfitting.

Par la suite, après avoir réglé le problème d'overfitting en rajoutant les rotations des images, j'ai adapté ce réseau pour lui faire prendre les images plus petites. Malheureusement, ce dernier n'obtient de loin pas les résultats escomptés.

Test score: 6.543635000413464
Test accuracy: 0.34645161032676697



Test recall 0.6287425149700598
Test precision 0.35443037974683544
Test f-score 0.4533189422558014
[[117 765]
[248 420]]
34.65% (+/- 0.00%)

Figure 11 : Graphique d'erreur et matrice de confusion en entraînant sur une seule zone et en testant sur les autres, avec des images de 15 sur 15 pixels

accuracy: 84.31%
Test score: 2.105391204855931
Test accuracy: 0.8430962562561035

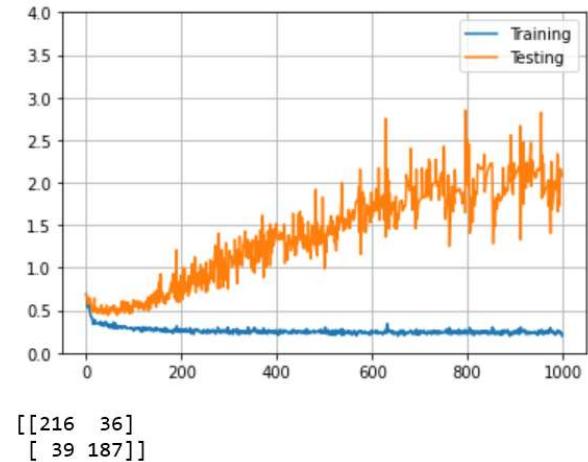


Figure 10 : Graphique d'erreur et matrice de confusion en entraînant sur une seule zone et en testant sur les autres, avec des images de 90 sur 90 pixels.

On constate qu'il n'arrive pas du tout à retrouver la bonne classe lors des tests. On remarque aussi qu'il prédit beaucoup de café alors que ce type de champs est minoritaire dans les données de test. Ces mauvais résultats sont sans doute dus à la grande quantité de champs de poivre présent dans le set d'entraînement (et absent lors de l'expérience citée plus haut), mais quasi absent dans le set de test, et inversement pour les données appartenant au milieu urbain.

Sur l'image ci-contre, je ne pense pas à un signe d'overfitting, mais plus à l'incapacité du réseau à trouver la bonne classe avec les données de test.

Je referai cette expérience plus tard sur des réseaux plus complexes.

4.8.2 Division d'image 2.0

J'ai aussi essayé de faire en sorte que mon script de division d'image fasse déjà un tri sur les images selon les points donnés par le CIAT, pour éviter de repasser une semaine si je dois encore refaire une sélection de données d'entraînement. Cette version du script marche très bien. Il n'y a donc plus besoin de passer des heures à faire matcher à la main les images déjà labellisées. Les images ne possédant pas des coordonnées présentes dans la liste donnée par le CIAT seront enregistrées dans un dossier nommé « not_labeled », celle en possédant plusieurs de types différents seront, quant à elles enregistrées dans un dossier nommé « ambiguous ».

4.8.3 Café contre poivre

J'ai essayé de faire un entraînement avec seulement des images de café et de poivre, sur toutes les bandes. Avec un set de données équilibré, on arrive à une visibilité d'environ 70%. J'ai aussi remarqué qu'il se peut que le système n'apprenne pas, dans le sens que même après quelques époques, il continue à donner toujours la même prédiction. Il faut donc ajouter plus de neurones dans la partie totalement connectée pour diminuer la fréquence d'apparition de ce problème.

À noter que selon l'aléatoire de l'entraînement, il se peut que les résultats ne soient pas aussi uniformes que les captures d'écran ci-jointes. Parfois, le réseau arrive à un rappel supérieur à 85% et une précision frôlant les 60% ou vice-versa. La capture d'écran ci-dessous montre aussi la moyenne des différentes mesures après une validation croisée de 1000 époques.

Accuracy 73.33% (+/- 3.17%)

Recall 73.50% (+/- 9.78%)

Precision 74.42% (+/- 5.74%)

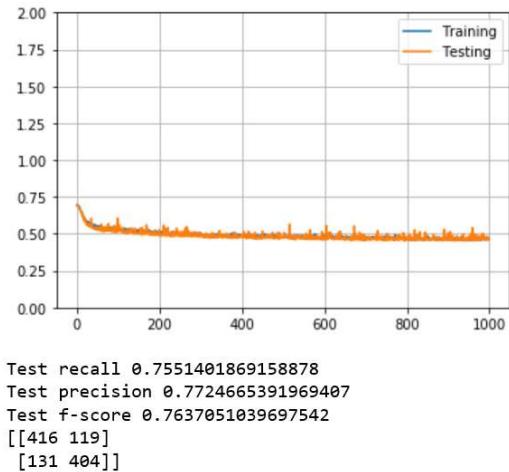
F-Score 73.20% (+/- 3.39%)

Figure 13 : Moyenne des résultats de poivre VS café

J'ai aussi essayé de faire un réseau pouvant classer les images dans différentes catégories (via une sortie en vecteur one-hot). Dans cette expérience, j'avais 4 classes : le café, le poivre, les deux en même temps et une dernière catégorie pour le reste qu'on appellera « autre ». Dans cette dernière catégorie, j'avais pris des images de forêts et de caoutchoucs. Après avoir entraîné mon système, je constate qu'il arrive bien à faire la distinction entre les classes, surtout au niveau pour la catégorie « autre ». Cependant, il n'est pas capable de dire si un champ contient les deux types de cultures et se contente de classer les images correspondantes soit dans l'un soit dans l'autre... Cela est sans doute dû au peu de données pour cette classe-ci. Il confond aussi bien souvent les champs de café et les champs de poivre.

J'ai ensuite un peu modifié le réseau ci-dessus en ne prenant plus que 3 classes (j'ai enlevé celle contenant les 2), j'ai aussi changé les données de la catégorie « autre » pour rajouter plus de diversité (en rajoutant de la mer, de la ville et d'autres champs). Après une validation croisée, j'arrive aux résultats décrits dans le tableau se trouvant au haut de la page suivante.

Test score: 0.47117915588004566
Test accuracy: 0.7663551568984985



Test recall 0.7551401869158878
Test precision 0.7724665391969407
Test f-score 0.7637051039697542
[[416 119]
[131 404]]

Figure 12: Poivre VS Café (dans cet ordre dans la matrice de confusion)

[[132	3	1	0]
[1	95	38	0]
[0	39	95	0]
[0	7	2	0]]

Figure 14: de haut en bas : autre, poivre, café, les deux.

	Rappel	Précision	F-Score
Café	71.11%	72.22%	71.47%
Poivre	77.65%	73.79%	75.47%
Autre	90.27%	94.64%	92.40%

La matrice de confusion ci-jointe (dans l'ordre : autre, poivre et café) nous montre avec quoi sont confondues les images mal classées. Sans surprise, nous constatons que la plus grande partie de l'erreur est due à l'incapacité du système à distinguer de façon certaine les champs de café des champs de poivre. On note aussi une augmentation d'erreurs d'identification au niveau de la classe autre, sans doute due au fait de la diversification des données, dont certaines sont aussi des champs.

$$\begin{bmatrix} 485 & 4 & 45 \\ 11 & 376 & 148 \\ 22 & 104 & 409 \end{bmatrix}$$

Figure 15 : Autre VS poivre VS café (dans cet ordre)

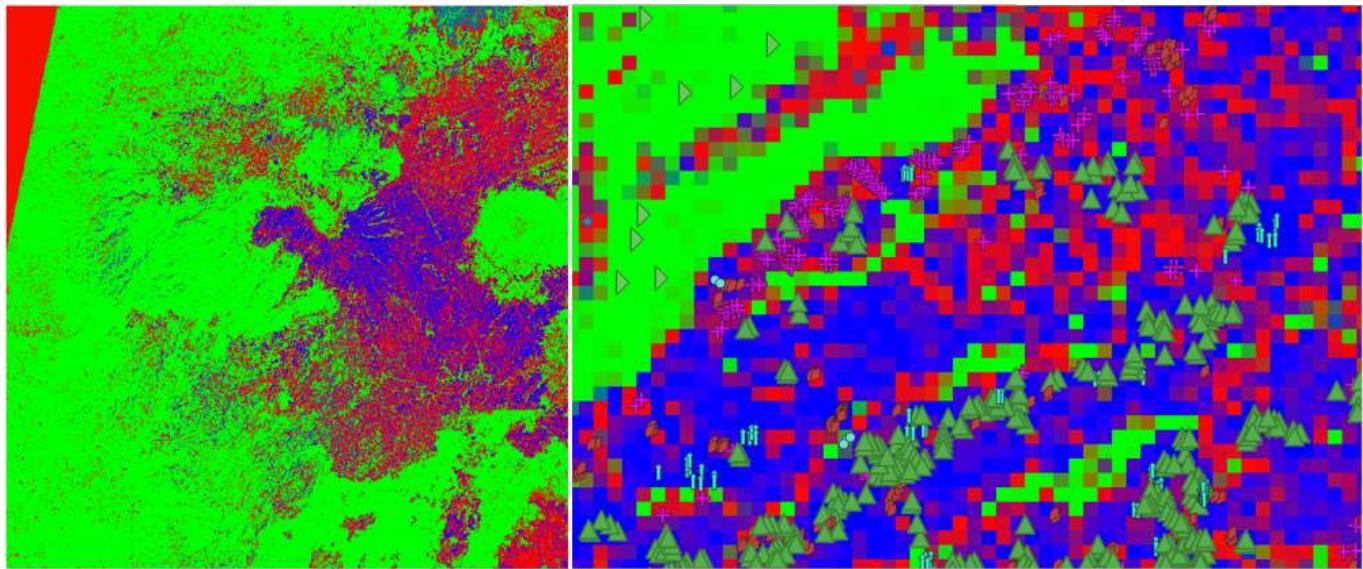


Figure 16: Image générée en prédiction multiple (total de l'image à gauche, zoom à droite avec les coordonnées du CIAT activées)

L'image de sortie à encore quelques problèmes. Par exemple, la bande noire du coin droite (sur l'image de départ voir page 18) apparaît comme un champ de café. Il faudrait rajouter des images noires (ne possédant pas de données) dans le set de données d'entraînement. Si l'on zoom sur la couche et que l'on active les points donnés par le CIAT, on voit que les champs de poivres (\blacktriangle et \uparrow) sont assez bien classés vu qu'ils correspondent à la partie bleue leur étant associée. Les champs de café (\blacksquare) quant à eux sont, hélas, bien souvent indistinguables des champs de poivres ; ils sont bien souvent dans le rouge qui est la couleur qui leur est associée, mais très proche des zones bleues. J'aurais souhaité avoir plus de nuances au niveau des couleurs pour avoir une meilleure retenue (dans cette image-ci, j'ai l'impression que la prédiction est trop catégorique, alors qu'elle devrait être plus nuancées au vu de la variété des différentes cultures de cette région), j'ai donc relancé un entraînement du réseau avec une fonction de fin en sigmoid plutôt qu'un softmax pour essayer de régler ce problème. En ce qui concerne la catégorie « autre » : la culture intercalaire (intercrop \blacksquare) et les points d'eau (\circlearrowleft) se retrouvent dans beaucoup de catégories différentes, ceci est sans doute dû à leur proximité avec des coordonnées à labels différents. Les régions n'ayant naturellement pas d'arbres (\blacklozenge) sont confondues avec des champs de café, il n'y a que les forêts (\blacktriangleright) qui obtiennent la couleur verte associée. À noter que des parties de l'image satellite utilisée pour générer ces captures d'écrans étaient des données d'entraînement.

Après avoir modifié mon réseau et mon set d'entraînement, j'arrive, en moyenne, aux résultats suivants.

	Rappel	Précision	F-Score
Café	76.87%	69.65%	73.02%
Poivre	71.38%	76.16%	73.63%
Autre	91.08%	94.98%	92.98%

On constate que les résultats des deux expériences sont assez similaires au niveau de l'entraînement. Au niveau de la prédiction, les choses sont un peu différentes.

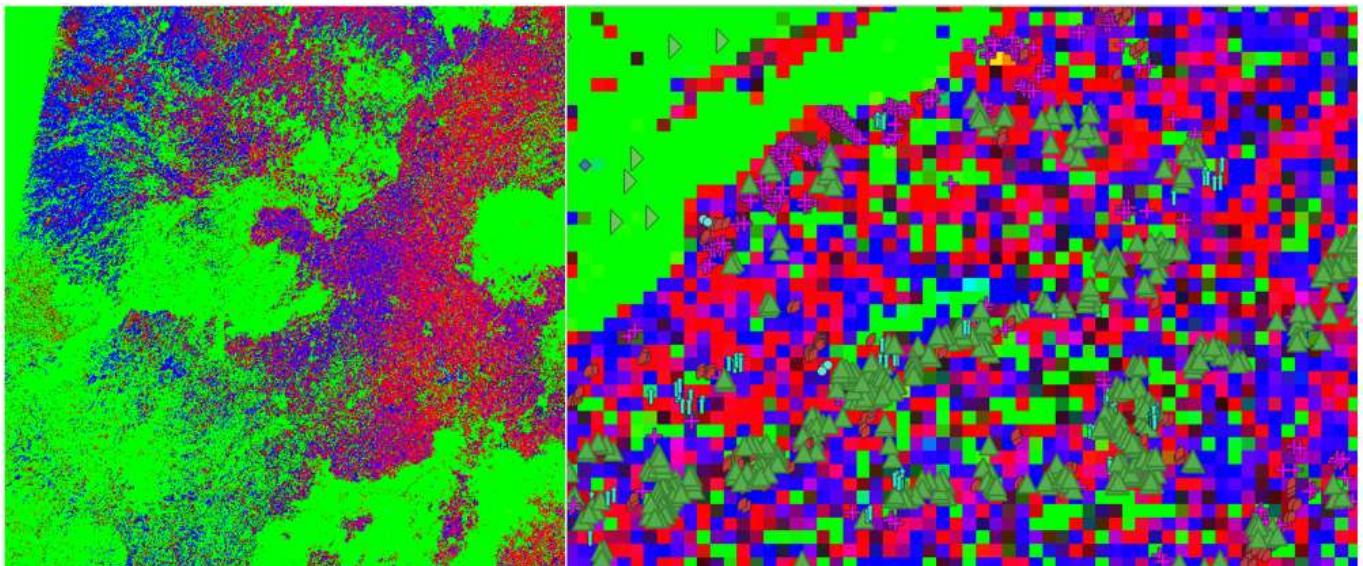


Figure 17: Image générée en prédiction multiple avec le nouveau réseau
(total de l'image à gauche, zoom à droite avec les coordonnées du CIAT activées)

On constate que la zone en haut à gauche est bien passée en vert, dû aux rajouts de matrices totalement noires dans notre base de données. Cependant, une grande partie de la zone nord-ouest est maintenant considérée comme du poivre ou du café alors qu'à l'œil nu elle partait plus désertique... La nuance de couleur n'est pas non plus celle exceptée. À l'avenir j'utiliserais donc softmax en fonction de sortie.



Figure 18 : À gauche zone du nord-ouest, à droite une zone contenant des champs de café et de poivre

4.9 Au-delà des saisons

Le principal problème de mes réseaux de neurones jusqu'à présent et que je ne le testais que sur des images de même date. Or la végétation ne reste pas fixe durant toute l'année. C'est pourquoi il est important de faire un réseau tenant compte de tout ceci.



Figure 19 : Même région prise (de gauche à droite) en automne, en hiver et au printemps

Si l'on regarde les différentes images ci-dessus, on peut voir que celle prise en automne est beaucoup plus verdoyante (sans doute grâce à la saison des pluies à laquelle elle succède) que celle prise au printemps qui elle vire sur le rouge-brun. Celle prise en hiver quant à elle fait la transition entre les deux. Il se peut donc que les bandes les plus utiles changent en fonction des saisons.

Pour cette expérience, les images utilisées n'ont pas toutes été prises à la même date entre les différentes zones testées à cause des conditions météorologiques.

Pour cette expérience, j'ai utilisé toutes les bandes des trois saisons testées et pour les données d'entraînement, je n'ai pris que des données déjà labellisées permettant un set plus hétéroclite, donc plus complexe ce qui explique les résultats un peu moins bons qu'au point 4.7.

Pour sélectionner les meilleures combinaisons de bandes, je ne vais pas tester toutes les combinaisons possibles, car ceci serait trop long, mais plutôt tester toutes les combinaisons par saison.

Les bandes utilisées ont été prise entre fin septembre et fin octobre pour l'automne, fin décembre pour l'hiver et mi-mars pour le printemps.

Accuracy 84.74% (+/- 0.37%)
Recall 86.87% (+/- 3.68%)
Precision 83.03% (+/- 2.02%)
F-Score 84.82% (+/- 0.83%)

Figure 20 : Moyenne des résultats pour un réseau avec toutes les bandes des trois saisons

4.9.1 Automne

Bandes utilisées	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	76.96%	77.77%	74.62%	76.08%
Bleu, Vert et Rouge	73.05%	78.09%	63.10%	69.65%
Bleu, Vert et Infra-Rouge	74.16%	75.45%	70.68%	72.86%
Bleu, Rouge et Infra-Rouge	76.72%	76.29%	76.40%	76.32%
Vert, Rouge et Infra-Rouge	74.45%	77.40%	67.94%	72.30%
Bleu et Vert	67.46%	65.34%	72.88%	68.70%
Bleu et Rouge	69.56%	67.91%	72.92%	70.20%
Bleu et Infra-Rouge	73.50%	74.12%	71.42%	72.54%
Vert et Rouge	72.97%	74.97%	67.85%	71.17%
Vert et Infra-Rouge	73.21%	72.92%	72.53%	72.67%
Rouge et Infra-Rouge	73.37%	73.53%	72.14%	72.66%
Bleu	59.73%	57.19%	72.36%	63.83%
Vert	63.39%	61.38%	71.39%	65.51%
Rouge	63.27%	60.79%	73.09%	66.12%
Infra-Rouge	67.56%	67.93%	64.81%	66.18%

On constate que pour l'automne aucune bande seule ne se démarque vraiment. Seule la bande bleue a des résultats légèrement inférieur aux autres.

Pour ce qui est des combinaisons de 2 bandes, les combinaisons contenant la bande infra-rouge obtiennent les meilleurs résultats. La bande verte et rouge a des résultats légèrement inférieurs. Quant aux combinaisons utilisant la bande bleue, elles obtiennent des résultats dans l'ensemble moins bon (sauf celle avec l'infra-rouge bien entendu).

Dans les combinaisons de 3 bandes, la combinaison bleu, rouge et infra-rouge se démarque particulièrement puisque ses résultats sont quasi similaires que ceux obtenus par la combinaison des 4 bandes. Le plus mauvais dans le lot est la combinaison bleu, vert et rouge, qui bien qu'ayant la meilleure précision de cette expérience, a aussi le plus mauvais rappel ce qui explique ses médiocres résultats.

On en conclut donc pour l'automne que la bande la plus utile est la bande infra-rouge, même si seule, elle ne se démarque pas totalement (si ce n'est que sa précision et son rappel sont proches l'un de l'autre) les combinaisons qu'elles engendrent sont les meilleures. Contrairement à l'observation faite au point 4.7, dans cette expérience, la bande verte est beaucoup plus utile dans cette période-ci qu'en fin février. Ses résultats sont comparables à ceux obtenus avec la bande rouge.

Pour l'automne, je recommanderais la combinaison des bandes bleu, rouge et infra-rouge ou, si le souhait est d'avoir le moins de bande possible d'utiliser l'une des combinaisons utilisant l'infra-rouge et une autre bande (surtout la verte ou la rouge).

4.9.2 Hiver

Bandes utilisées(s)	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	77.80%	73.93%	85.18%	79.02%
Bleu, Vert et Rouge	73.29%	69.55%	82.11%	75.09%
Bleu, Vert et Infra-Rouge	75.96%	71.93%	83.93%	77.43%
Bleu, Rouge et Infra-Rouge	75.02%	72.47%	79.78%	75.70%
Vert, Rouge et Infra-Rouge	74.73%	70.29%	84.39%	76.61%
Bleu et Vert	68.48%	66.33%	73.99%	69.40%
Bleu et Rouge	70.29%	66.62%	79.74%	72.53%
Bleu et Infra-Rouge	70.13%	64.83%	85.61%	73.74%
Vert et Rouge	68.86%	65.51%	77.63%	70.93%
Vert et Infra-Rouge	72.91%	68.06%	84.75%	75.46%
Rouge et Infra-Rouge	72.68%	68.21%	83.38%	74.96%
Bleu	62.67%	58.79%	80.71%	67.80%
Vert	63.80%	59.19%	85.26%	69.79%
Rouge	65.57%	60.46%	86.57%	71.18%
Infra-Rouge	63.14%	60.28%	74.20%	66.38%

Pour ce qui est de l'hiver, on constate que seule la bande rouge se démarque toute seule avec les meilleurs résultats dans toutes les catégories, les autres bandes suivent sans trop se démarquer.

Pour ce qui est des combinaisons à deux bandes, ici aussi, ce sont toujours les combinaisons possédant la bande infra-rouge qui se démarquent, surtout les combinaisons avec le vert ou le rouge.

Des combinaisons à trois bandes, la combinaison bleu, vert et infra-rouge sort du lot ayant le meilleur f-score et la meilleure visibilité. Aucun de ses résultats ne dépasse toutefois la combinaison des quatre bandes, mais nous ne perdons pas plus de 2% au niveau des résultats en comparaison.

Je conseillerais la combinaison bleu, vert et infra-rouge. Pour ce qui est des combinaisons à 2 bandes seulement, les combinaisons vert et infra-rouge ou rouge et infra-rouge me paraissent être des bons choix.

4.9.3 Printemps

Bandes utilisées	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	79.37%	77.17%	82.63%	79.69%
Bleu, Vert et Rouge	75.39%	72.19%	81.38%	76.46%
Bleu, Vert et Infra-Rouge	76.76%	74.22%	80.98%	77.39%
Bleu, Rouge et Infra-Rouge	77.81%	75.57%	81.28%	78.25%
Vert, Rouge et Infra-Rouge	77.36%	75.74%	79.66%	77.59%
Bleu et Vert	69.62%	65.79%	80.02%	72.11%
Bleu et Rouge	72.78%	68.86%	81.49%	74.60%
Bleu et Infra-Rouge	75.30%	72.24%	80.90%	76.26%
Vert et Rouge	72.44%	69.13%	79.70%	73.93%
Vert et Infra-Rouge	75.96%	72.97%	81.42%	76.89%
Rouge et Infra-Rouge	76.61%	76.23%	76.33%	76.19%
Bleu	66.94%	62.74%	81.30%	70.74%
Vert	65.63%	62.22%	76.96%	68.67%
Rouge	66.59%	64.20%	73.67%	67.67%
Infra-Rouge	68.61%	65.31%	77.28%	70.73%

On constate sans beaucoup d'étonnement que la bande infra-rouge est l'une des meilleures.

Pour ce qui est des combinaisons à 2 bandes, sans étonnement non plus, ce sont les bandes ayant la bande infra-rouge qui arrivent en tête ayant toutes plus de 76% de f-score et 75% de visibilité.

Quant aux combinaisons à 3 bandes, là encore celles qui contiennent de l'infra-rouge ont de meilleurs résultats que la dernière.

Je recommanderais donc la combinaison bleu, rouge et infra-rouge pour une combinaison de 3 bandes. Pour ce qui est des combinaisons à 2 bandes, je n'ai pas de recommandation à faire si ce n'est d'en prendre une avec la bande infra-rouge, la différence entre elles étant minime.

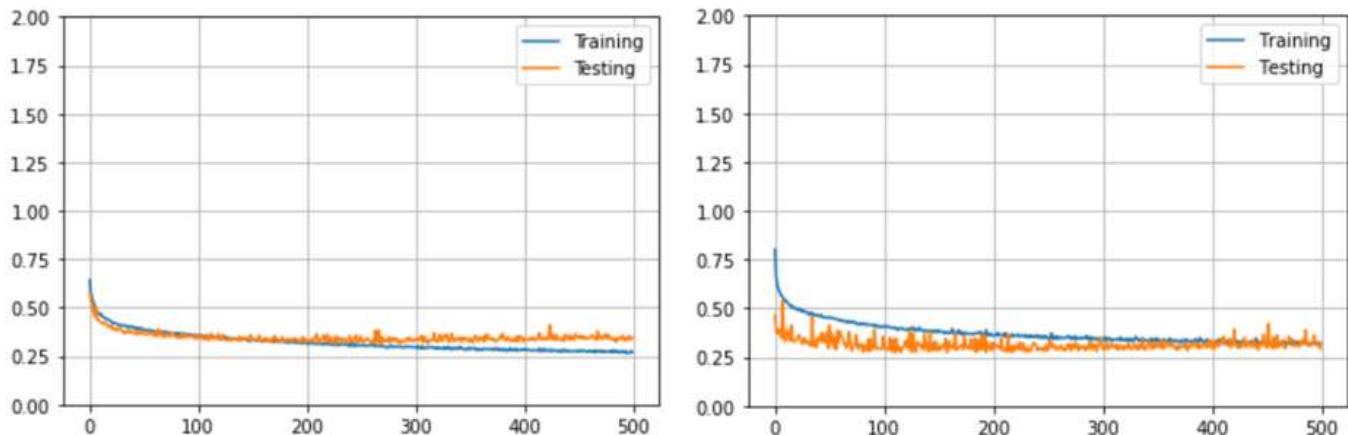
4.9.4 Conclusions

Vu la faible différence entre les résultats des différentes combinaisons, je pense qu'il est possible que mes recommandations soient différentes si j'entraînais à nouveau mes réseaux de neurones ou si je changeais un peu les données prises pour la classe « autre ». Ces recommandations ne représentent que les résultats de mes différents apprentissages à un moment donné. On peut toutefois en tirer les conclusions suivantes :

- La bande infra-rouge est la meilleure pour arriver à faire la différence entre les champs de café et le reste. Toutefois, elle seule ne suffit pas. Il faut donc la combiner avec au minimum une bande pour arriver à des résultats plutôt bons.
- On remarque que de manière générale, les entraînements sur le printemps ont un rappel et une précision assez proches. Alors que pour l'hiver, le rappel est généralement 10% plus haut que la précision. Le printemps, quant à lui, est un peu entre les deux. Si l'on regarde les f-scores, toutes les saisons se valent.

Si on entraîne un réseau en prenant mes recommandations, on obtient les résultats ci-dessous :

Pour 3 couches par saison, ici bleue et infra-rouge combinées à rouge pour le printemps et l'automne et verte pour l'hiver. On constate que les résultats obtenus sont grossièrement les mêmes que ceux obtenus lors de l'entraînement avec l'intégralité des bandes. On peut donc s'affranchir de trois bandes sans réelles pertes de performances.



```
Test recall 0.878840579710145
Test precision 0.8306849315068493
Test f-score 0.8540845070422536
[[1475  309]
 [ 209 1516]]
Accuracy 84.77% (+/- 0.89%)
Recall 86.24% (+/- 2.88%)
Precision 83.37% (+/- 0.59%)
F-Score 84.75% (+/- 1.19%)
```

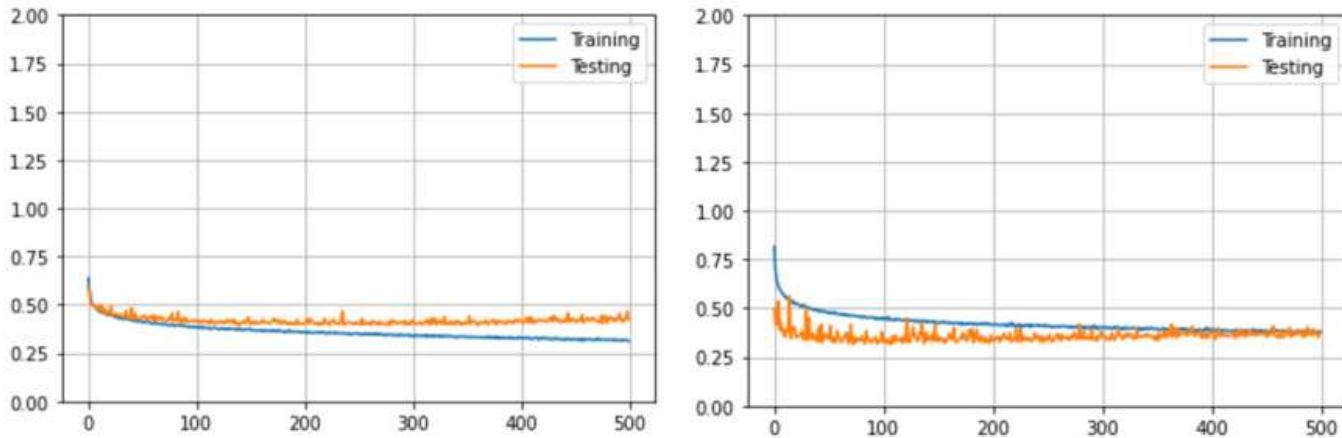
```
Test recall 0.9043478260869565
Test precision 0.729995320542817
Test f-score 0.807871569135163
[[3636  577]
 [ 165 1560]]
Accuracy 87.19% (+/- 0.78%)
Recall 90.53% (+/- 1.80%)
Precision 72.43% (+/- 2.14%)
F-Score 80.43% (+/- 0.76%)
```

Figure 21 : Entraînement sur 9 bandes, à gauche avec les deux classes plus ou moins équilibrées, à droite sur l'intégralité des images labellisées en utilisant le paramètre `class_weight` de Keras

La capture d'écran ci-dessus nous montre les résultats de 2 entraînements différents. Le premier est un entraînement fait sur un set de données plus ou moins équilibré, le deuxième utilise, quant à lui, l'intégralité de nos images couplée au paramètre `class_weight` de la méthode `fit` Keras permettant d'équilibrer les classes lors de l'entraînement. Dans cette deuxième image, le résultat le plus intéressant est le rappel. La visibilité, la précision et, par conséquent, le f-score sont biaisés à cause du manque d'équilibrage des classes lors du test du réseau.

On constate que dans les deux cas, les résultats sont assez bons.

En ce qui concerne les combinaisons de 2 couches par saison (ici verte et infra-rouge pour toutes les saisons), on obtient cette fois-ci des résultats un peu moins bons qu'avec un entraînement comprenant l'intégralité des couches. On constate que la différence la plus grande est d'un peu moins de 3%. Ce type de modèle est donc totalement utilisable vu que nous divisons par 2 le volume des données à utiliser.



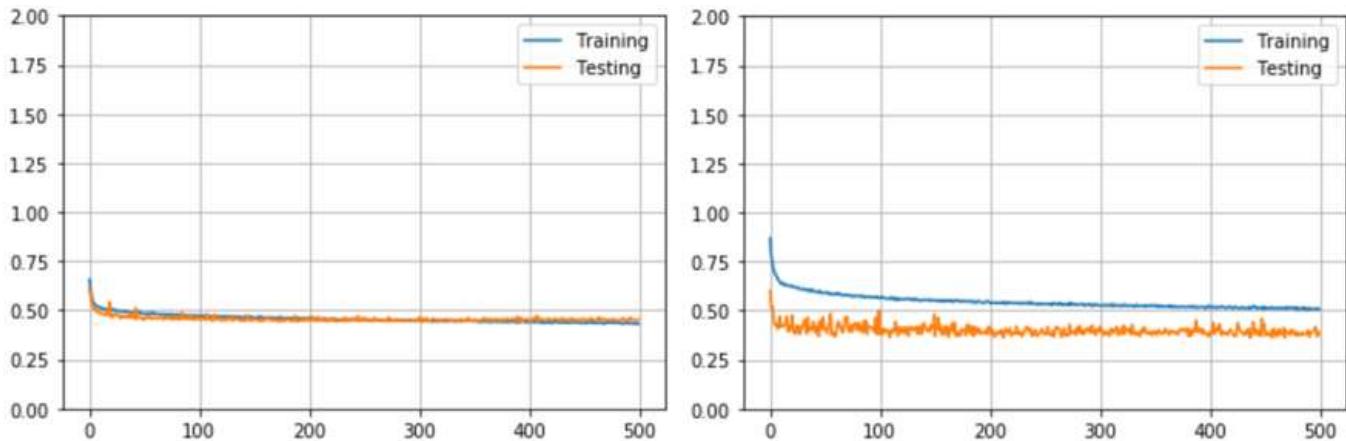
```
Test recall 0.8735498839907193
Test precision 0.7876569037656904
Test f-score 0.8283828382838284
[[1379  406]
 [ 218 1506]]
Accuracy 82.46% (+/- 0.16%)
Recall 87.37% (+/- 1.84%)
Precision 79.15% (+/- 0.92%)
F-Score 83.03% (+/- 0.36%)
```

```
Test recall 0.9135730858468677
Test precision 0.6373937677053825
Test f-score 0.7508939213349226
[[3318  896]
 [ 149 1575]]
Accuracy 84.12% (+/- 1.02%)
Recall 91.01% (+/- 0.88%)
Precision 66.64% (+/- 1.84%)
F-Score 76.92% (+/- 1.05%)
```

Figure 22 : Entraînement sur 6 bandes, à gauche avec les deux classes plus ou moins équilibrées, à droite sur l'intégralité des images labellisées en utilisant le paramètre `class_weight` de Keras

Comparé avec les modèles utilisant les combinaisons à 3 bandes, on constate que le rappel augmente, mais ce au détriment de la précision qui descend. Le f-score et la visibilité sont aussi tirés vers le bas.

Finalement, si l'on ne prend que les bandes infra-rouges, nous arrivons à un résultat où toutes les mesures sont entre 75% et 80%, ce qui est une assez bonne performance.



```
Test recall 0.7221577726218097
Test precision 0.8110749185667753
Test f-score 0.7640380484811291
[[1495 290]
 [ 479 1245]]
Accuracy 78.20% (+/- 0.83%)
Recall 76.91% (+/- 3.03%)
Precision 78.48% (+/- 2.40%)
F-Score 77.60% (+/- 0.86%)
```

```
Test recall 0.8655072463768116
Test precision 0.6123872026251025
Test f-score 0.7172711986548163
[[3268 945]
 [ 232 1493]]
Accuracy 80.16% (+/- 0.88%)
Recall 86.12% (+/- 1.57%)
Precision 61.33% (+/- 1.57%)
F-Score 71.61% (+/- 0.59%)
```

Figure 23 : Entraînement sur 3 bandes infra-rouge, à gauche avec les deux classes plus ou moins équilibrées, à droite sur l'intégralité des images labellisées en utilisant le paramètre `class_weight` de Keras

On peut aussi s'apercevoir que le rappel est toujours meilleur dans les entraînements effectués sur l'intégralité de nos images labellisées. Si l'on calcule le rappel pour la classe « pas café », on s'aperçoit (sauf dans l'expérience infra-rouge uniquement) que son résultat est aussi plus élevé, sans doute dû au fait qu'il a plus de données différentes à disposition.

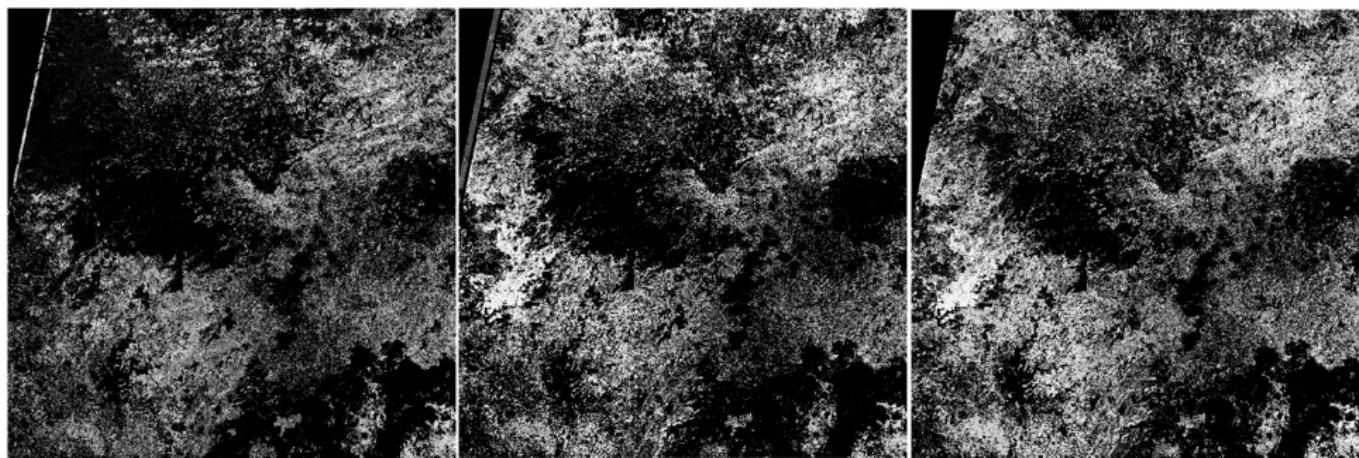


Figure 24 : Prédiction multi-saisons avec de gauche à droite 6, 9 et 12 bandes

Sur les images de base, il n'y a pas de données dans le coin supérieur droit, mais les données ne commencent pas toutes aux mêmes moments dans les différentes images. Ces petits décalages sont à l'origine des bandes blanches que l'on peut apercevoir dans ce coin sur la prédiction de gauche et du milieu. La prédiction utilisant toutes les bandes ne semble pas souffrir de ce problème.

On constate que les zones désertiques mentionnées au point 4.8.3 sont ici considérées comme étant un potentiel champ de café. Cela est peut-être dû au fait qu'aucune de ces images ne figure dans le set de données.

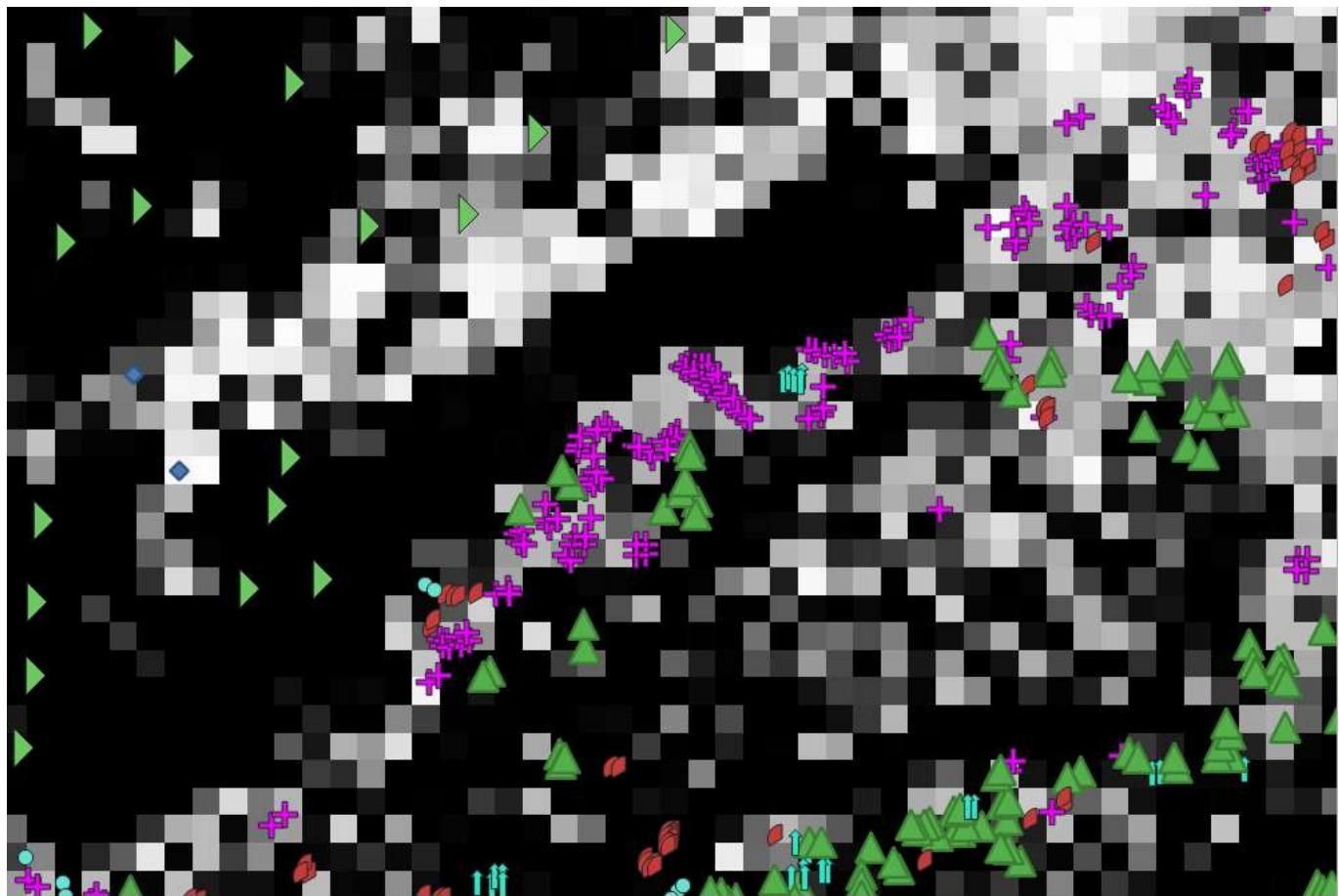


Figure 25 : Zoom sur la prédiction à 9 bandes avec les labels activés

On constate que la majorité des données que la majorité des champs de café ($\textcolor{purple}{+}$) se trouvent dans des zones claires, indiquant la présence supposée de tels champs. Cependant on constate aussi que certains champs de culture intercalaire ($\textcolor{red}{\cdot}$) se trouvent aussi dans des zones claires, ceci est sûrement dû au fait de leur proximité avec les champs de café. D'ailleurs on constate que ceux éloignés des champs de café sont souvent dans des zones sombres. Les champs de poivre (\triangle et \uparrow) quant à eux semblent bien se comporter se trouvant plutôt dans des zones foncées. Les points d'eau (\circ) et la végétation naturelle (\triangleright) semble aussi bien classés. Finalement, les endroits où il n'y a naturellement pas d'arbre (\blacklozenge) sont un peu mitigé, mais généralement proches des zones sombres.

4.10 Autres expériences avec Sentinel-2

4.10.1 Entraîner sur une zone et tester sur les autres 2.0

Dans un premier temps, j'ai fait un entraînement sur toutes les bandes des trois saisons. Les résultats varient en fonction du lieu d'apprentissage et le lieu d'entraînement.

Les résultats plus intéressants dans l'image ci-dessous sont les matrices de confusion, le rappel et la visibilité. À noter que l'entraînement et les tests ont été effectués sur l'intégralité des données labellisées d'une zone donnée.

Trained on basfws	Trained on droitefws	Trained on gauchefws	Trained on hautfws	Trained on milieufws
Tested on basfws	Tested on basfws	Tested on basfws	Tested on basfws	Tested on basfws
accuracy: 89.82%	accuracy: 75.34%	accuracy: 69.56%	accuracy: 67.06%	accuracy: 53.15%
Test score: 0.271782458624315	Test score: 51.95619739401491	Test score: 15.917751146950732	Test score: 20.276500216693258	Test score: 0.5314778089523315
Test accuracy: 0.8981571197589766	Test accuracy: 0.7533962726593018	Test accuracy: 0.695576548576355	Test accuracy: 0.6706427931785583	Test recall 0.13455093833788016
Test recall 0.8274647887323944	Test recall 0.71263404825753726	Test recall 0.6665549597855228	Test recall 0.640583109919571	Test precision 0.6761584718783162
Test precision 0.38778877887788776	Test precision 0.77116953762466	Test precision 0.7024545294013773	Test precision 0.6578902082257787	Test precision 0.6205564142194745
Test f-score 0.520898876404495	Test f-score 0.7407471915004789	Test f-score 0.6840340469435131	Test f-score [[4273 1831]]	Test f-score 0.22115119801707517
[[3469 371]]	[[4842 1262]]	[[4419 1685]]	[[1990 3978]]	[[5613 491]]
[49 235]]	[1715 4253]]	[1990 3978]]	[2145 3823]]	[5165 803]]
Tested on gauchefws	Tested on gauchefws	Tested on droitefws	Tested on droitefws	Tested on droitefws
accuracy: 59.11%	accuracy: 65.37%	accuracy: 92.10%	accuracy: 53.64%	accuracy: 74.37%
Test score: 1.8353049194772637	Test score: 50.26364370532309	Test score: 0.5675191183788585	Test score: 2.107291120793725	Test score: 0.7436954379881726
Test accuracy: 0.591077446937561	Test accuracy: 0.6537036895751953	Test accuracy: 0.9209508519595337	Test accuracy: 0.5363724827766418	Test accuracy: 0.13708920187793427
Test recall 0.727979274611399	Test recall 0.4481865284974093	Test recall 0.5211267605633803	Test recall 0.5140845070422535	Test precision 0.10562015503875968
Test precision 0.20208558072635743	Test precision 0.17501264542235712	Test precision 0.4378698224852071	Test precision 0.18568994889267462	Test f-score 0.21645663454410674
Test f-score 0.31635237827188295	Test f-score 0.2517279010549291	Test f-score 0.4758842443729904	[[1994 1846]]	[[2921 919]]
[[2949 2219]]	[[3537 1631]]	[[3650 190]]	[136 148]]	[138 146]]
[210 562]]	[426 3461]]	[136 148]]	[66 218]]	[66 218]]
Tested on hautfws	Tested on hautfws	Tested on hautfws	Tested on hautfws	Tested on gauchefws
accuracy: 73.45%	accuracy: 79.93%	accuracy: 74.85%	accuracy: 85.02%	accuracy: 82.36%
Test score: 10.706725978645785	Test score: 56.35498059636413	Test score: 232.73993472446656	Test score: 12.959703348140524	Test score: 17.15640517653841
Test accuracy: 0.734451949596405	Test accuracy: 0.79926210931396484	Test accuracy: 0.748460590839386	Test accuracy: 0.6501683592796326	Test accuracy: 0.8235689997673035
Test recall 0.46438746438746437	Test recall 0.5911680911680912	Test recall 0.5790560290598291	Test recall 0.617875647683938	Test recall 0.09196891191709844
Test precision 0.4012307692307692	Test precision 0.5320851282051282	Test precision 0.4380387931034483	Test precision 0.21106194690265487	Test precision 0.16985645933014354
Test f-score 0.43050511720039614	Test f-score 0.5600539811066126	Test f-score 0.4987730061349694	Test f-score 0.31464379947229554	Test f-score 0.11932773109243697
[[4119 973]]	[[4362 730]]	[[4849 1043]]	[[3385 1783]]	[[4821 347]]
[752 652]]	[574 830]]	[591 813]]	[295 477]]	[701 711]]
Tested on milieufws	Tested on milieufws	Tested on milieufws	Tested on milieufws	Tested on hautfws
accuracy: 81.04%	accuracy: 79.91%	accuracy: 79.06%	accuracy: 89.15%	accuracy: 78.40%
Test score: 0.7146690411387749	Test score: 17.893939337134363	Test score: 4.176420152187347	Test score: 1.7594881820791173	Test score: 26.436305658570653
Test accuracy: 0.8103773593902588	Test accuracy: 0.7998565896634241	Test accuracy: 0.7905660271644592	Test accuracy: 0.8915094137191772	Test accuracy: 0.7840209603309631
Test recall 0.7959183673469388	Test recall 0.8622448979591837	Test recall 0.6887755102840817	Test recall 0.8571428571428571	Test recall 0.0833333333333333
Test precision 0.4921135646687697	Test precision 0.476056338028169	Test precision 0.4560810810810811	Test precision 0.6588235294117647	Test precision 0.5821459227467812
Test f-score 0.6081871345029239	Test f-score 0.6134301270417423	Test f-score 0.5487804878048781	Test f-score 0.7450110864745011	Test f-score 0.1429441050702503
[[703 161]]	[[678 186]]	[[703 161]]	[[777 87]]	[[4976 116]]
[40 156]]	[27 169]]	[61 135]]	[28 168]]	[1287 117]]
Visibility 75.85% (+/- 11.27%)	Visibility 75.14% (+/- 5.94%)	Visibility 78.89% (+/- 8.33%)	Visibility 68.72% (+/- 12.86%)	Visibility 72.07% (+/- 11.28%)
Recall 70.39% (+/- 14.29%)	Recall 65.36% (+/- 15.26%)	Recall 61.39% (+/- 6.75%)	Recall 72.08% (+/- 9.72%)	Recall 20.60% (+/- 17.89%)
Precision 37.08% (+/- 10.54%)	Precision 48.86% (+/- 21.23%)	Precision 50.86% (+/- 11.22%)	Precision 41.29% (+/- 25.74%)	Precision 35.74% (+/- 20.85%)
F-Score 47.00% (+/- 10.91%)	F-Score 54.15% (+/- 17.97%)	F-Score 55.19% (+/- 8.07%)	F-Score 47.58% (+/- 23.23%)	F-Score 17.50% (+/- 4.47%)

Figure 26 : Entraîner sur une zone tester sur les autres avec toutes les bandes (à lire en colonne)

On remarque que le nombre de faux-positifs est plus élevé quand on teste sur la zone dite gauchefws, ceci est dû au fait qu'une grande majorité des champs de poivre (et d'autres cultures) se trouve dans cette zone (facilement confondable avec les champs de café et souvent très proche). Inversement, le nombre de faux-négatifs est plus élevé lors de l'entraînement avec la zone milieufws ceci est sûrement dû au fait qu'elle est la zone contenant le moins de champs de café (seulement 196 petites images), ce qui n'est sans doute pas très recommandé pour un entraînement. On constate aussi que de manière générale une majorité des images ne contenant pas de champs de café a bel et bien été classée en tant que tel. Quant aux images contenant du café, cela varie beaucoup entre la région d'entraînement et la région de test. Par exemple, si l'on entraîne sur basfws, les résultats obtenus sur la plupart des zones sont plus ou moins satisfaisant, sauf sur la zone hautfws qui a un rappel inférieur à 50%.

J'ai ensuite fait la même expérience sur les 9 bandes recommandées.

Trained on basfws	Trained on droitefws	Trained on gauchefws	Trained on hautfws	Trained on milieufws
Tested on basfws	Tested on basfws	Tested on basfws	Tested on basfws	Tested on basfws
accuracy: 89.52%	accuracy: 71.79%	accuracy: 70.91%	accuracy: 67.98%	accuracy: 61.22%
Test score: 0.2937942507571092	Test score: 27.952230293439506	Test score: 7.523366951326253	Test score: 50.90093663980662	Test score: 37.61002135837686
Test accuracy: 0.8952473402023315	Test accuracy: 0.7178595066070557	Test accuracy: 0.709078848361969	Test accuracy: 0.6798376441001892	Test accuracy: 0.6121603846549988
Test recall 0.8380281690140845	Test recall 0.6549932975871313	Test recall 0.6823056300268097	Test recall 0.6933646112600537	Test recall 0.35103887399463807
Test precision 0.3814182564182564	Test precision 0.7437214611872146	Test precision 0.7158931082981715	Test precision 0.6703385711971489	Test precision 0.7214187327823691
Test f-score 0.5242290748898678	Test f-score 0.696543121881682	Test f-score 0.6986959505833905	Test f-score 0.6816571946297669	Test f-score 0.4722723174030658
[3454 386]	[4757 1347]	[4488 1616]	[4069 2035]	[5295 809]
[46 238]	[2059 3909]	[1896 4072]	[1830 4138]	[3873 2095]
Tested on gauchefws	Tested on gauchefws	Tested on droitefws	Tested on droitefws	Tested on droitefws
accuracy: 46.63%	accuracy: 67.64%	accuracy: 92.12%	accuracy: 92.39%	accuracy: 56.98%
Test score: 2.3520406799284297	Test score: 20.958176694895684	Test score: 0.42073251330803246	Test score: 0.5844724879898509	Test score: 3.3450233279560284
Test accuracy: 0.466329962015152	Test accuracy: 0.6764310002326965	Test accuracy: 0.921193006365448	Test accuracy: 0.92386683115081787	Test accuracy: 0.5698351263999939
Test recall 0.8341968911917098	Test recall 0.36658031088082904	Test recall 0.3626760563380282	Test recall 0.778169014084507	Test recall 0.5563380281690141
Test precision 0.17471513836136734	Test precision 0.1649184149184149	Test precision 0.41700404858299595	Test precision 0.4682203389830508	Test precision 0.08748615725359911
Test f-score 0.2889187976671153	Test f-score 0.227491961414791	Test f-score 0.3879472693032015	Test f-score 0.5846560846560845	Test f-score 0.15119617224880383
[2126 3042]	[3735 1433]	[3696 144]	[3589 251]	[2192 1648]
[128 644]	[489 283]	[181 103]	[63 221]	[126 158]
Tested on hautfws	Tested on hautfws	Tested on hautfws	Tested on hautfws	Tested on hautfws
accuracy: 71.09%	accuracy: 82.00%	accuracy: 75.79%	accuracy: 64.02%	accuracy: 81.90%
Test score: 5.175057822847601	Test score: 76.59842621029569	Test score: 140.87273981110215	Test score: 58.7492203400994	Test score: 32.20016325170344
Test accuracy: 0.7108989953994751	Test accuracy: 0.8200430870056152	Test accuracy: 0.75785108460805249	Test accuracy: 0.6402356624603271	Test accuracy: 0.8190235495567322
Test recall 0.561965811965812	Test recall 0.6018518518518519	Test recall 0.665954415954416	Test recall 0.5155440414507773	Test recall 0.07901554404145078
Test precision 0.3845029239766082	Test precision 0.5807560137457045	Test precision 0.45855811672388425	Test precision 0.1841739935215178	Test precision 0.14352941176470588
Test f-score 0.4565972222222222	Test f-score 0.5911157747464149	Test f-score 0.5431309904153354	Test f-score 0.27139447664507327	Test f-score 0.18192147034252297
[3829 1263]	[4482 618]	[3988 1104]	[3405 1763]	[4804 364]
[615 789]	[559 845]	[469 935]	[374 398]	[711 61]
Tested on milieufws	Tested on milieufws	Tested on milieufws	Tested on milieufws	Tested on milieufws
accuracy: 73.77%	accuracy: 80.57%	accuracy: 88.75%	accuracy: 74.25%	accuracy: 79.34%
Test score: 0.8038069727285853	Test score: 9.816740565929774	Test score: 2.5545055942948753	Test score: 2.125183222878654	Test score: 109.57698008741065
Test accuracy: 0.7377358675003052	Test accuracy: 0.8056603679120239	Test accuracy: 0.8075471520423889	Test accuracy: 0.74245285987854	Test accuracy: 0.7934113144874573
Test recall 0.673469387755102	Test recall 0.7755102040816326	Test recall 0.4336734693877551	Test recall 0.9489795918367347	Test recall 0.11894586894586895
Test precision 0.3815028901734104	Test precision 0.4840764331210191	Test precision 0.47752808988764045	Test precision 0.4142538975501114	Test precision 0.6139705882352942
Test f-score 0.487084708487084	Test f-score 0.5960784313725491	Test f-score 0.4545454545454546	Test f-score 0.5767441860465117	Test f-score 0.1928400954653938
[650 214]	[702 162]	[771 93]	[601 263]	[4987 105]
[64 132]	[44 152]	[111 85]	[10 186]	[1237 167]
Visibility 70.26% (+/- 15.35%)	Visibility 75.58% (+/- 5.99%)	Visibility 79.89% (+/- 7.87%)	Visibility 74.66% (+/- 10.86%)	Visibility 69.85% (+/- 10.98%)
Recall 72.69% (+/- 11.61%)	Recall 59.97% (+/- 14.86%)	Recall 53.62% (+/- 14.84%)	Recall 73.48% (+/- 15.62%)	Recall 27.63% (+/- 19.22%)
Precision 33.05% (+/- 9.00%)	Precision 49.34% (+/- 21.11%)	Precision 51.72% (+/- 11.68%)	Precision 43.42% (+/- 17.31%)	Precision 39.16% (+/- 27.94%)
F-Score 43.92% (+/- 9.00%)	F-Score 52.78% (+/- 17.84%)	F-Score 52.11% (+/- 11.64%)	F-Score 52.86% (+/- 15.41%)	F-Score 23.12% (+/- 14.34%)

Figure 27 : Entraîner sur une zone tester sur les autres avec les 9 bandes recommandées (à lire en colonne)

On constate que les résultats sont généralement inférieurs (sauf dans certains cas), si on ne teste pas avec toutes les bandes.

Avec cette expérience, je pensais qu'il était important d'entraîner le réseau avec des champs de café et d'autres choses venant de différentes régions. J'ai donc refait la même expérience en entraînant non sur une seule région, mais sur deux différentes. J'ai obtenu les résultats suivants. Cette expérience se base aussi sur 9 bandes.

Trained on basfwsmilieufws	Trained on basfwsgauchefws	Trained on basfwshautfws	Trained on basfwsmilieufws	Trained on basfwsdroitefws
Tested on basfws	Tested on droitefws	Tested on basfws	Tested on basfws	Tested on basfws
accuracy: 46.0%	accuracy: 89.6%	accuracy: 92.7%	accuracy: 94.18%	accuracy: 76.71%
Test score: 2.2887884820109667	Test score: 0.24174252808614335	Test score: 0.15376417154209573	Test score: 0.18630437634442232	Test score: 32.07394236994252
Test accuracy: 0.4609427750110626	Test accuracy: 0.896702229976654	Test accuracy: 0.9270126223564148	Test accuracy: 0.941804051399231	Test accuracy: 0.767064273357914
Test recall 0.8290155440414507	Test recall 0.8450704225352113	Test recall 0.954225352112676	Test recall 0.8380281690148845	Test recall 0.8944369973190348
Test precision 0.1725067385444744	Test precision 0.3858520900321543	Test precision 0.4847942754919499	Test precision 0.5509259259259259	Test precision 0.7098404255319148
Test f-score 0.285586791610888	Test f-score 0.5290013245033112	Test f-score 0.6429418742586003	Test f-score 0.664804469273743	Test f-score 0.7915183867141163
[2089 3070]	[3458 382]	[3552 288]	[3646 194]	[3922 2182]
[132 640]	[44 240]	[13 271]	[46 238]	[630 5338]
Tested on hautfws	Tested on hautfws	Tested on gauchefws	Tested on gauchefws	Tested on hautfws
accuracy: 74.12%	accuracy: 74.71%	accuracy: 39.18%	accuracy: 48.05%	accuracy: 74.97%
Test score: 6.288054985979508	Test score: 8.902047894564756	Test score: 1.9277259166794594	Test score: 2.187264725495669	Test score: 419.4198777152987
Test accuracy: 0.741253618240356	Test accuracy: 0.7478751404762268	Test accuracy: 0.39175084223283997	Test accuracy: 0.48047137268043701	Test accuracy: 0.7496921420097351
Test recall 0.5868945868945868	Test recall 0.7008547088547008	Test recall 0.9248074663212435	Test recall 0.73575129533787575	Test recall 0.7393162393162394
Test precision 0.4288519480519481	Test precision 0.4458541085898349	Test precision 0.16725228390723823	Test precision 0.1646376811594203	Test precision 0.4516971279373368
Test f-score 0.4950435566236187	Test f-score 0.5450013846579895	Test f-score 0.2832771275340567	Test f-score 0.2696667929891047	Test f-score 0.5607779578606159
[3991 1101]	[3869 1223]	[1613 3555]	[2286 2882]	[3832 1260]
[580 824]	[420 984]	[58 714]	[204 568]	[366 1038]
Tested on milieufws	Tested on milieufws	Tested on milieufws	Tested on milieufws	Tested on milieufws
accuracy: 89.94%	accuracy: 79.15%	accuracy: 75.85%	accuracy: 75.00%	accuracy: 76.42%
Test score: 1.080497829198295	Test score: 1.2096597704122651	Test score: 0.46097829960427195	Test score: 4.85236529558163	Test score: 12.88512420665093
Test accuracy: 0.8094339370727539	Test accuracy: 0.7915094494819641	Test accuracy: 0.7584905624389648	Test accuracy: 0.75	Test accuracy: 0.7641509175309598
Test recall 0.7244897959183674	Test recall 0.7704081632653061	Test recall 0.8571428571428571	Test recall 0.569088190883191	Test recall 0.9285714285714286
Test precision 0.4896551724173931	Test precision 0.4617737080358104	Test precision 0.42424242424242425	Test precision 0.434939493949395	Test precision 0.4354066985645933
Test f-score 0.5843612399176956	Test f-score 0.5774378585806042	Test f-score 0.5675675675675677	Test f-score 0.49596523898199885	Test f-score 0.5928338762214984
[716 148]	[688 176]	[636 228]	[4073 1019]	[628 236]
[54 142]	[45 151]	[28 168]	[605 799]	[14 182]
Visibility 67.05% (+/- 15.08%)	Visibility 81.18% (+/- 6.27%)	Visibility 69.24% (+/- 22.35%)	Visibility 72.41% (+/- 18.92%)	Visibility 76.03% (+/- 8.76%)
Recall 71.35% (+/- 9.92%)	Recall 77.21% (+/- 5.89%)	Recall 91.21% (+/- 4.07%)	Recall 71.43% (+/- 11.08%)	Recall 85.41% (+/- 8.24%)
Precision 36.34% (+/- 13.73%)	Precision 43.12% (+/- 3.27%)	Precision 35.88% (+/- 13.77%)	Precision 38.50% (+/- 16.23%)	Precision 53.23% (+/- 12.57%)
F-Score 45.50% (+/- 12.52%)	F-Score 55.07% (+/- 1.99%)	F-Score 49.79% (+/- 15.49%)	F-Score 47.66% (+/- 16.21%)	F-Score 64.84% (+/- 10.21%)
Trained on droitefwsdroitefws	Trained on droitefwsmilieufws	Trained on gauchefwsdroitefws	Trained on gauchefwsmilieufws	Trained on hautfwsmilieufws
Tested on basfws	Tested on basfws	Tested on basfws	Tested on basfws	Tested on basfws
accuracy: 73.10%	accuracy: 76.94%	accuracy: 73.57%	accuracy: 74.82%	accuracy: 70.78%
Test score: 16.123156170840293	Test score: 123.29450610893188	Test score: 3.4661409944676067	Test score: 9.866712659276464	Test score: 67.67178195181708
Test accuracy: 0.731030461723633	Test accuracy: 0.7356693148612976	Test accuracy: 0.7481778978959924	Test accuracy: 0.7070879445838928	Test accuracy: 0.7070879445838928
Test recall 0.713869705093834	Test recall 0.75037533120644	Test recall 0.7882037533521064	Test recall 0.6853217158176944	Test recall 0.6853217158176944
Test precision 0.7346054154164511	Test precision 0.7720437457279562	Test precision 0.725925925925926	Test precision 0.71142807444773	Test precision 0.71142807444773
Test f-score 0.7240588805323382	Test f-score 0.7644670050761421	Test f-score 0.7557840616966581	Test f-score 0.698130928841854	Test f-score 0.698130928841854
[4465 1539]	[4778 1334]	[4493 1411]	[4445 1659]	[3656 184]
[1708 4260]	[1458 4518]	[1780 4188]	[1878 4090]	[97 187]
Tested on gauchefws	Tested on gauchefws	Tested on droitefws	Tested on droitefws	Tested on gauchefws
accuracy: 67.78%	accuracy: 64.04%	accuracy: 94.42%	accuracy: 92.58%	accuracy: 59.60%
Test score: 13.839130746796476	Test score: 59.348116955291545	Test score: 0.3888935164551449	Test score: 0.27361162955340657	Test score: 0.5623898126336698
Test accuracy: 0.677777671813965	Test accuracy: 0.64840440455818176	Test accuracy: 0.9442288875759834	Test accuracy: 0.9318622946739197	Test accuracy: 0.9318622946739197
Test recall 0.555699481865285	Test recall 0.6101836269430051	Test recall 0.4507042253521127	Test recall 0.721830859515493	Test recall 0.6584507842553251
Test precision 0.2145	Test precision 0.20424978317432785	Test precision 0.63366336633666337	Test precision 0.47453703703703703	Test precision 0.5040431266846361
Test f-score 0.3095238095238095	Test f-score 0.3060428849902534	Test f-score 0.5267489711934157	Test f-score 0.5709923664122137	Test f-score 0.5709923664122137
[3597 1571]	[3333 1835]	[3766 74]	[3613 227]	[3656 184]
[343 4291]	[301 4711]	[156 128]	[79 2051]	[97 187]
Tested on milieufws	Tested on hautfws	Tested on milieufws	Tested on hautfws	Tested on gauchefws
accuracy: 86.42%	accuracy: 85.39%	accuracy: 81.60%	accuracy: 77.76%	accuracy: 59.60%
Test score: 4.7082405902585895	Test score: 1049.658585059452	Test score: 4.902545177711631	Test score: 199.04719551303998	Test score: 84.43900155700015
Test accuracy: 0.8641509413719177	Test accuracy: 0.8539100885391235	Test accuracy: 0.7775554060935974	Test accuracy: 0.7775554060935974	Test accuracy: 0.7775554060935974
Test recall 0.80612248489795918	Test recall 0.5990028490028491	Test recall 0.5969387755102041	Test recall 0.6695156695156695	Test recall 0.68081314715026
Test precision 0.5984848484848485	Test precision 0.6854115729421353	Test precision 0.5021459227467812	Test precision 0.48932847475273294	Test precision 0.1960418222554145
Test f-score 0.6869565217391305	Test f-score 0.6393006461421513	Test f-score 0.545454545454545455	Test f-score 0.5654135338345865	Test f-score 0.3043478260869565
[758 106]	[4706 386]	[748 116]	[4111 981]	[3015 2153]
[38 158]	[563 841]	[79 117]	[464 940]	[247 525]
Visibility 75.77% (+/- 7.84%)	Visibility 75.46% (+/- 8.78%)	Visibility 83.28% (+/- 8.59%)	Visibility 81.72% (+/- 7.77%)	Visibility 74.49% (+/- 13.97%)
Recall 69.19% (+/- 10.34%)	Recall 65.54% (+/- 7.26%)	Recall 58.31% (+/- 10.30%)	Recall 72.65% (+/- 4.86%)	Recall 67.46% (+/- 1.16%)
Precision 51.59% (+/- 22.02%)	Precision 55.39% (+/- 24.98%)	Precision 62.79% (+/- 10.04%)	Precision 56.33% (+/- 11.52%)	Precision 47.05% (+/- 21.17%)
F-Score 57.35% (+/- 18.73%)	F-Score 56.99% (+/- 19.35%)	F-Score 59.88% (+/- 8.98%)	F-Score 63.13% (+/- 8.81%)	F-Score 52.45% (+/- 16.41%)

Figure 28 : Entrainer sur deux zones et tester sur les autres (à lire en colonne)

On constate que la visibilité ne change pas vraiment. Les résultats sont plus ou moins les mêmes. Cependant, le fait d'entraîner sur deux régions semble améliorer légèrement le rappel général, car il peut combler les lacunes de certaines zones (comme par exemple milieufws). On remarque aussi que la zone la plus dure à classer est toujours basfws.

Au vu des résultats très variables selon le lieu d'entraînement et le lieu de test, je conseillerais donc de toujours utiliser quelques données, que ce soient des champs de café ou autres, provenant de la zone étudiée pour améliorer nos modèles. Il faudrait aussi ajouter des images de plusieurs variétés de champs de café, que ce soit de l'espèce de café ou de la façon de le cultiver, car il se peut qu'on ne cultive pas la même espèce de café, ni de la même façon dans des régions différentes.

4.10.2 Café contre poivre 2.0

Dans cette expérience, je refais l'expérience du modèle multi classes mais cette fois en prenant les images des 3 saisons. J'utilise toujours les trois classes : café, poivre et autre.

Ici, les réseaux ont été entraînés sur toutes les images labellisées.

4.10.2.1 Toutes les bandes

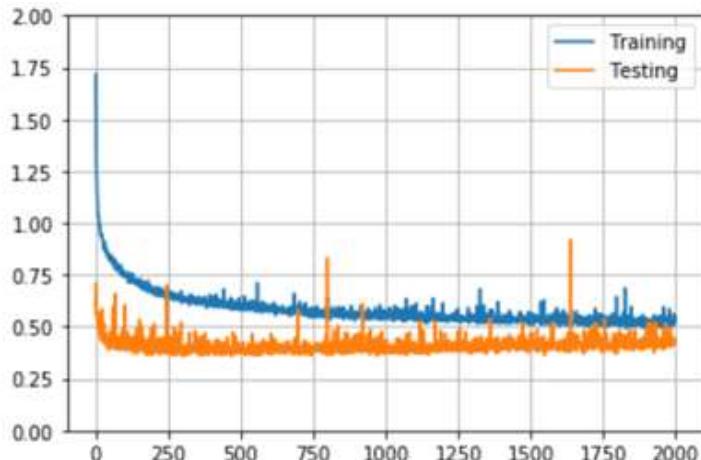
En utilisant toutes les bandes, j'ai obtenu les résultats suivants :

	Rappel	Précision	F-Score
Café	86.71%	74.59%	80.16%
Poivre	90.09%	66.74%	76.61%
Autre	83.50%	96.34%	89.44%

On constate que pour le café, les résultats sont tous meilleurs que lors de l'expérience précédente. Néanmoins, on peut voir que la précision du poivre a diminué, pareil pour le rappel de la classe « autre ». Ces pertes sont sans doute dues à l'augmentation du nombre de données « autre », peut-être trop proche des champs, et à la stagnation du nombre de champs de café.

En regardant la matrice de confusion, on peut s'apercevoir que la confusion d'un champ de café avec une autre classe se fait maintenant de manière plus égale entre les champs de poivre et le reste. On voit aussi que pas mal d'images de catégorie « autre » ont été confondues avec du poivre, ceci est possiblement dû au fait que certaines images peuvent être proche des champs de poivre.

On constate aussi qu'il n'y a pas de problème d'overfitting dans cet entraînement.



$$\begin{bmatrix} [3118 & 97 & 427] \\ [15 & 526 & 30] \\ [125 & 123 & 1477] \end{bmatrix}$$

Figure 29 : Graphique d'erreur et matrice de confusion d'un entraînement multi classes à 12 bandes

4.10.2.2 9 bandes

En utilisant les 9 bandes utilisées lors de la première expérience de la conclusion de la sélection des bandes par saison, nous obtenons les résultats suivants :

	Rappel	Précision	F-Score
Café	85.40%	73.60%	79.02%
Poivre	89.32%	66.41%	76.14%
Autre	83.21%	95.78%	89.04%

On constate qu'ils sont un peu moins bons qu'avec toutes les bandes, mais la différence n'excède pas 1.14% alors que nous n'avons utilisé que 75% des bandes à notre disposition.

La matrice de confusion et le graphique d'erreur sont plus ou moins similaires que celui fait avec toutes les bandes.

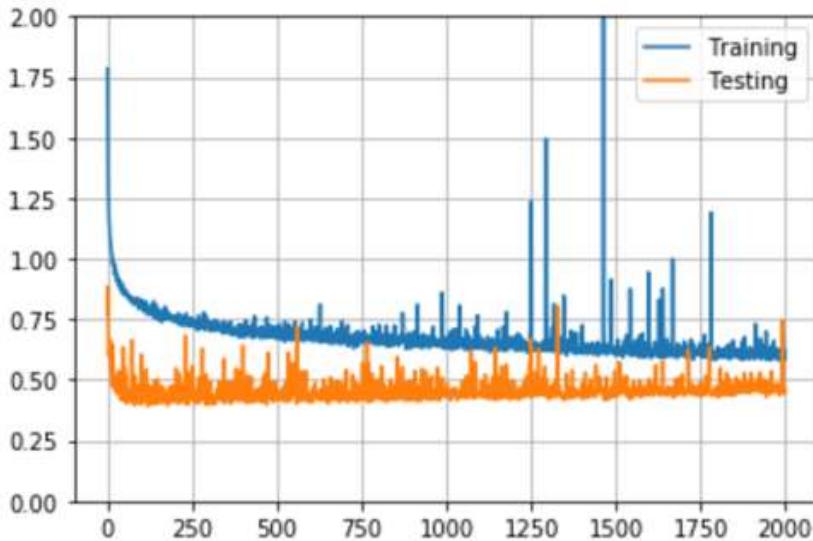


Figure 30 : Graphique d'erreur et matrice de confusion d'un entraînement multi classes à 9 bandes

4.10.2.3 6 bandes

Dans cette expérience-ci, on n'utilisera que les bandes infra-rouges et vertes de chacune de nos saisons.

	Rappel	Précision	F-Score
Café	81.77%	70.12%	75.48%
Poivre	84.45%	51.68%	64.07%
Autre	78.60%	95.71%	86.31%

On constate ici que nos résultats sont bien inférieurs aux résultats obtenus par les expériences similaires. Les résultats obtenus pour les classes « café » et « autre » sont plutôt bons avec tous les indicateurs au-dessus de 70%. Le principal problème vient au niveau du poivre même si son rappel est plutôt bon, sa précision a rudement chuté entraînant avec elle son f-score.

Si l'on regarde la matrice de confusion sur la page suivante, on s'aperçoit que cette perte de précision provient tout autant du café que de la classe « autre » avec une augmentation des faux-positifs d'environ 25% dans chacune de ces classes lors de la prédiction de poivre. La sélection des bandes faites jusqu'alors ne prenant pas en compte le poivre en tant que classe, il est possible que les certaines bandes utiles à sa distinction aient été supprimées lors du passage de 9 à 6 bandes.

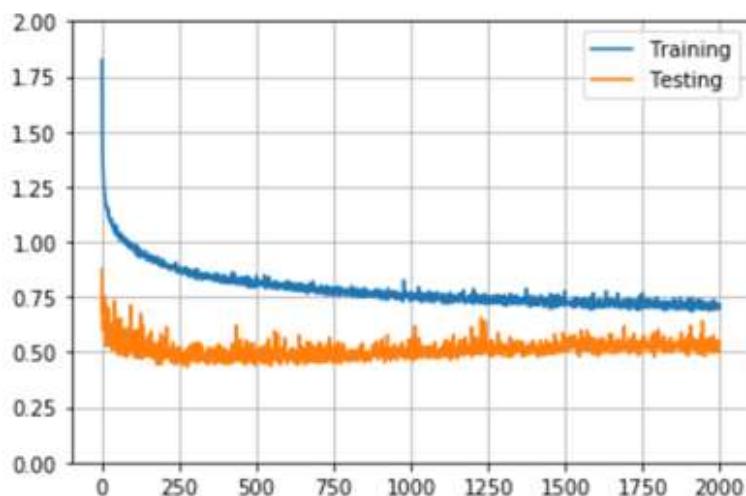


Figure 31 : Graphique d'erreur et matrice de confusion d'un entraînement multi classes à 6 bandes

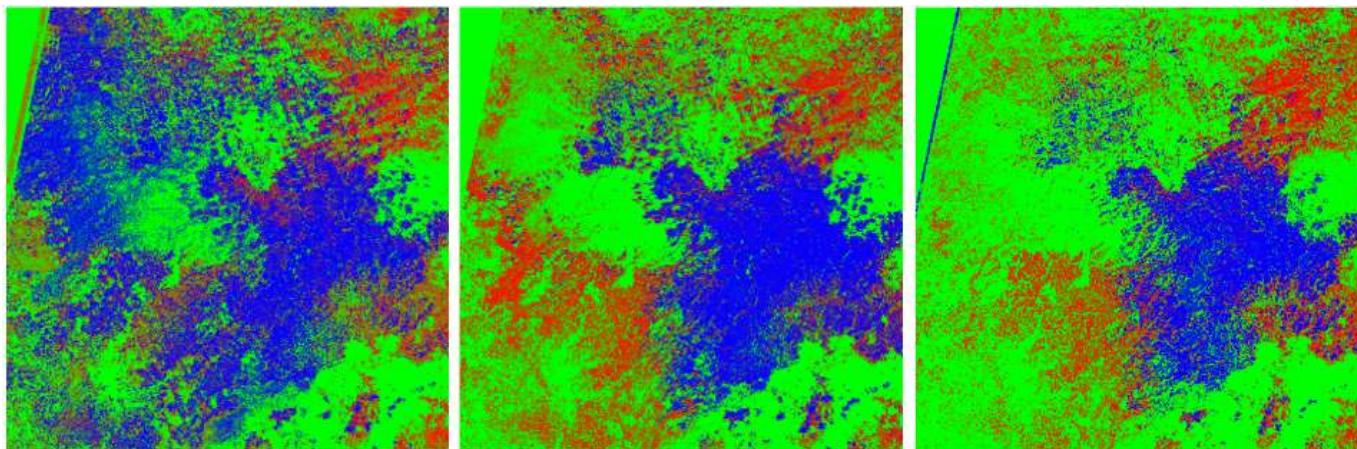


Figure 32 : Prédiction multi-saisons multi classes avec de gauche à droite 6, 9 et 12 bandes

Contrairement au test sur une seule bande, on constate que le café est souvent confondu avec le poivre. Ce problème est sans doute dû au fait que la plupart des bandes n'ont pas d'information permettant de les différencier. Il semble toutefois moins fréquent dans l'expérience où l'on n'utilise que les bandes vertes et infra-rouges (à gauche), par contre, comme annoncé précédemment, ce modèle à la fâcheuse tendance à voir du poivre un peu partout...

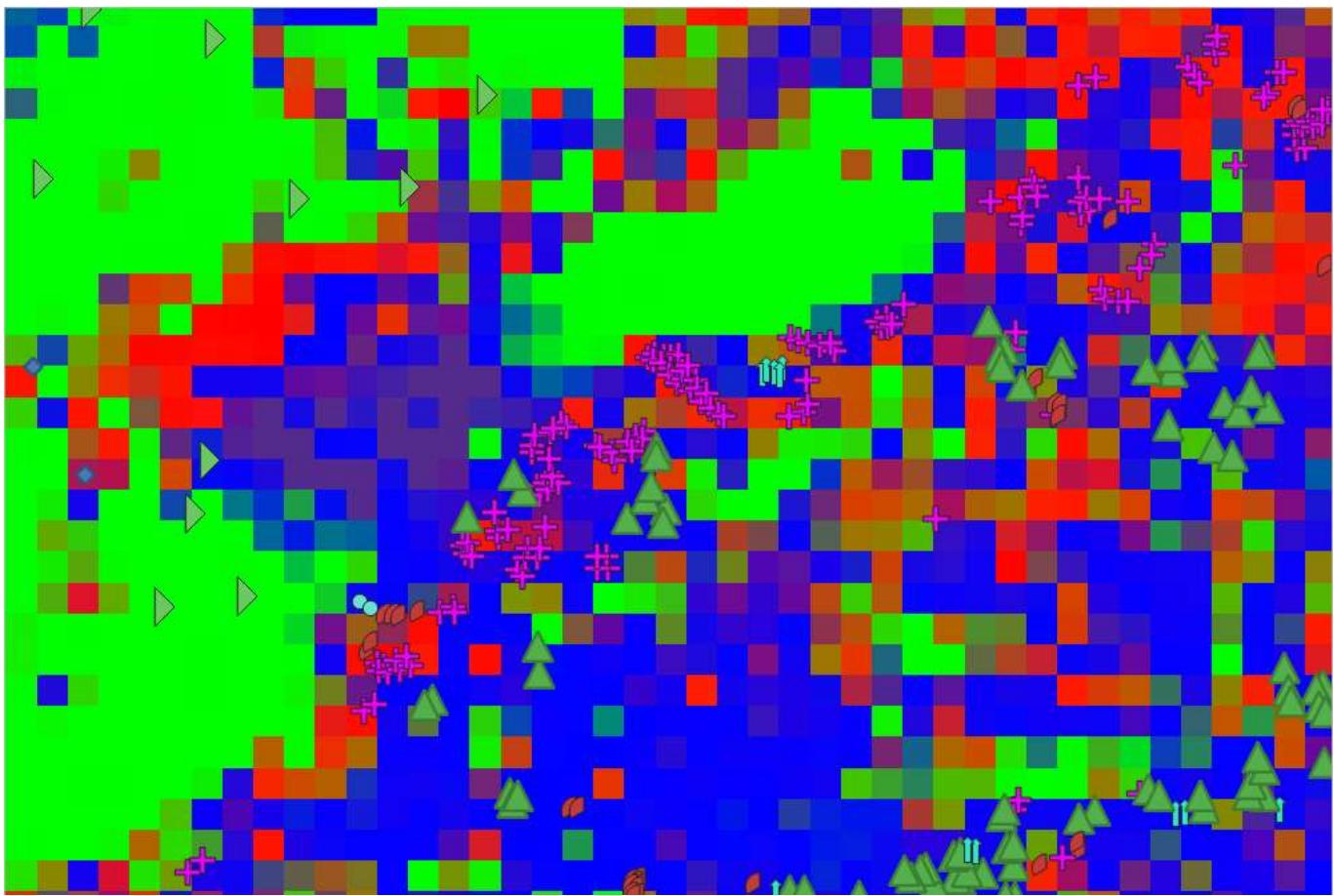


Figure 33 : Zoom sur la prédiction multi classes à 6 bandes avec les labels activés

Hormis le fait que le rappel des champs de café ($\textcolor{purple}{+}$) est moins bon que son équivalent noir et blanc (voir page 31), mes conclusions sur cette image restent les mêmes qu'au point 4.9.4.

J'ai ensuite essayé de faire un entraînement simple ne prenant que les données contenant du poivre ou du café en prenant toutes les bandes disponibles et ai obtenu les résultats ci-contre. On constate donc que le système arrive pourtant assez bien à séparer le café du poivre. C'est donc sans doute la présence d'autres champs qui perturbe la différenciation.

```
Test recall 0.9292343387470998
Test precision 0.9720873786407767
Test f-score 0.9501779359430605
[[ 526 46]
 [ 122 1602]]
Accuracy 92.70% (+/- 0.35%)
Recall 95.04% (+/- 1.27%)
Precision 95.26% (+/- 1.17%)
F-Score 95.14% (+/- 0.24%)
```

Figure 34 : Entraînement binaire poivre vs café

4.10.3 Tester une année avant

Dans cette expérience, j'ai simplement pris certains modèles entraînés avec des images allant de mi-2019 à 2020 et les tester sur l'intégralité des données obtenues en découplant et classant des images allant d'automne 2018 au printemps 2019.

J'ai donc essayé de tester mes différents réseaux sur des données de Sentinel-2 sur les données susmentionnées. Petit hic, les images du type 2A (type de produit délivré par Sentinel-2, plus précisément il s'agit de la réflexion en basse atmosphère) sont inaccessibles avant décembre 2018. J'ai donc pris à la place les images de l'autre type de produit délivré par Sentinel-2 : 1C (réflexion en haute atmosphère). Les résultats ne sont pas à la hauteur de mes attentes comme le montre la capture d'écran ci-contre.

```
Now
accuracy: 87.03%
Test accuracy: 0.8702681064605713
Test recall 0.8933289647495362
Test precision 0.724332455810455
Test f-score 0.7999999999999999
[[4534 733]
 [ 230 1926]]
A year ago
accuracy: 73.91%
Test accuracy: 0.739054262638092
Test recall 0.2166048237476809
Test precision 0.6531468531468532
Test f-score 0.32532218739115293
[[5019 248]
 [1689 467]]
```

Figure 35 : Tester un réseau entraîné sur 2A avec des images contenant du 1C, ça ne marche pas...

Au vu de ces résultats très bas, j'ai donc décidé d'entraîner un nouveau réseau totalement fait avec de type 1C. Après avoir eu quelques problèmes avec Hyperion (qui apparemment avait un espace stockage trop chargé), je suis finalement parvenu à entraîner mes modèles et j'ai obtenu les résultats suivants :

Now	Now	Now
accuracy: 82.19%	accuracy: 86.21%	accuracy: 85.40%
Test accuracy: 0.821945309638977	Test accuracy: 0.8621382713317871	Test accuracy: 0.853965699672699
Test recall 0.9494765589440146	Test recall 0.9394629039599454	Test recall 0.9522075557578517
Test precision 0.6313559322033898	Test precision 0.6972972972972973	Test precision 0.6798830029249269
Test f-score 0.7584075622614077	Test f-score 0.8004653868528214	Test f-score 0.7933257489571482
[[4049 1218]	[[4371 896]	[[4282 985]
[111 2086]]	[133 2064]]	[105 2092]]
A year ago	A year ago	A year ago
accuracy: 78.04%	accuracy: 80.47%	accuracy: 81.79%
Test accuracy: 0.7804126739501953	Test accuracy: 0.8046624064445496	Test accuracy: 0.8179260492324829
Test recall 0.8711879836140192	Test recall 0.8124715521165226	Test recall 0.7910787437414656
Test precision 0.5853211009174312	Test precision 0.6305192511480042	Test precision 0.6588324488248674
Test f-score 0.7002012072434608	Test f-score 0.7100238663484487	Test f-score 0.7189245087900725
[[3911 1356]	[[4221 1046]	[[4367 900]
[283 1914]]	[412 1785]]	[459 1738]]

Figure 36 : Tester une année plus tôt de gauche à droite avec 6, 9 et toutes les bandes

On constate que généralement, les données utilisées pour entraîner le réseau (haut de la capture d'écran) ont de meilleurs résultats que le set d'images plus vieilles. Cela peut s'expliquer par les conditions météorologiques qui ne sont pas les mêmes, les images prises en 2018 étant plus nuageuses que celles du set d'entraînement utilisé. De plus, les images ne datent pas forcément de la même période de l'année. Il se peut qu'il y ait un petit décalage d'un mois au plus entre deux bandes représentant la même saison ce qui pourrait affecter un peu la prédiction.

Outre, ces quelques différences, les résultats retournés sont positifs avec plus de 74% de rappels pour toutes les classes et quel que soit le nombre de bandes utilisées. On peut même en conclure que nous pouvons faire de bonne prédiction sur des images un peu décalées dans le temps.

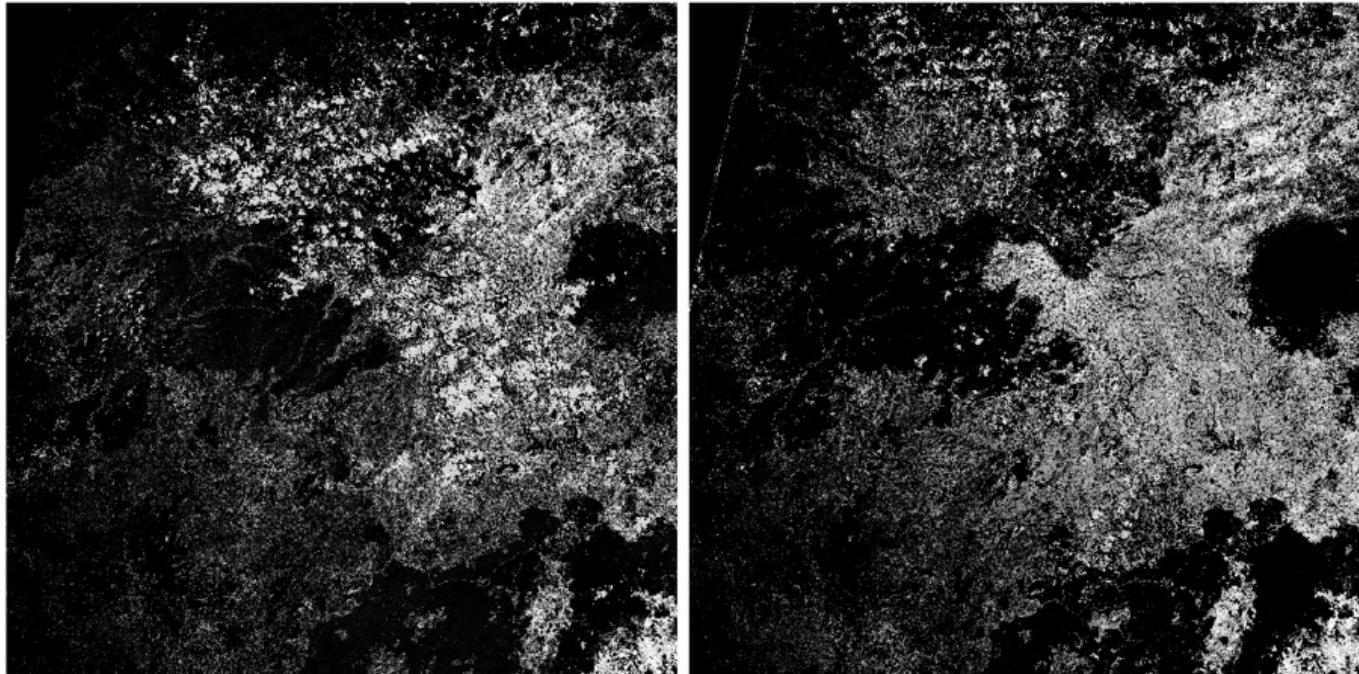


Figure 37 : Prédiction sur toutes les bandes à gauche en 2018-2019 et à droite 2019-2020

On le voit sur les images au bas de la page précédente, les prédictions faites entre 2018 et 2019 ne sont pas les mêmes. Cependant, on remarque que de manière générale, les deux prédictions tendent vers un pattern commun. Les quelques changements peuvent être expliqués par la couverture nuageuse et le moment où les différentes bandes ont été prises, mais aussi par une modification de la nature du sol.

Si l'on compare ces images-ci avec les images obtenues dans la conclusion du point 4.9, on remarque que celles-ci sont plus sombres. La différence de couleur étant notable, il se peut que les bandes les plus utiles soient différentes, c'est pourquoi, je vais refaire une sélection des bandes avec le produit 1C.

4.10.3.1 Automne

Bandes utilisées	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	75.75%	74.00%	78.21%	75.99%
Bleu, Vert et Rouge	70.59%	71.19%	68.23%	69.50%
Bleu, Vert et Infra-Rouge	73.74%	72.72%	75.16%	73.67%
Bleu, Rouge et Infra-Rouge	75.93%	74.38%	78.13%	76.10%
Vert, Rouge et Infra-Rouge	73.32%	73.07%	72.94%	72.78%
Bleu et Vert	69.34%	73.27%	59.81%	65.59%
Bleu et Rouge	67.20%	65.23%	71.71%	68.15%
Bleu et Infra-Rouge	71.93%	71.04%	72.56%	71.75%
Vert et Rouge	70.92%	71.66%	68.01%	69.67%
Vert et Infra-Rouge	71.67%	69.23%	76.69%	72.63%
Rouge et Infra-Rouge	72.65%	71.90%	72.87%	72.35%
Bleu	61.47%	59.38%	69.40%	63.82%
Vert	60.78%	57.17%	80.47%	66.81%
Rouge	61.69%	58.00%	82.59%	67.98%
Infra-Rouge	67.89%	66.30%	70.52%	68.33%

On remarque que les résultats sont à peu près les mêmes que son équivalent 2A, bien qu'ils soient un peu inférieurs, sauf le rappel qui est plus élevé à certains endroits.

La bande infra-rouge reste la meilleure au niveau des combinaisons. Mes recommandations concernant l'automne sont donc les mêmes que pour les produits de type 2A.

4.10.3.2 Hiver

Bandes utilisées	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	74.90%	71.43%	82.29%	76.21%
Bleu, Vert et Rouge	68.97%	63.61%	86.99%	73.29%
Bleu, Vert et Infra-Rouge	73.37%	69.54%	81.97%	75.09%
Bleu, Rouge et Infra-Rouge	72.58%	67.52%	85.39%	75.36%
Vert, Rouge et Infra-Rouge	73.57%	68.73%	85.10%	75.99%
Bleu et Vert	63.49%	59.38%	83.50%	69.24%
Bleu et Rouge	67.59%	62.55%	85.53%	72.16%
Bleu et Infra-Rouge	68.79%	64.65%	81.23%	71.81%
Vert et Rouge	66.33%	61.77%	83.69%	70.67%
Vert et Infra-Rouge	70.70%	65.82%	85.41%	74.15%
Rouge et Infra-Rouge	70.32%	66.66%	79.22%	72.35%
Bleu	61.59%	57.16%	87.20%	69.05%
Vert	60.16%	56.42%	85.18%	67.57%
Rouge	64.06%	59.24%	87.84%	70.61%
Infra-Rouge	60.89%	57.93%	75.66%	65.47%

Comme pour l'automne, on constate que les résultats sont à peu près les mêmes qu'avec les images de types 2A, bien que les résultats soient un peu inférieurs à ce dernier, sauf dans certains cas.

Ici aussi, sans réelle surprise, les combinaisons utilisant la bande infra-rouge sont les meilleures.

4.10.3.3 Printemps

Bandes utilisées	Visibilité	Précision	Rappel	F-Score
Bleu, Vert, Rouge et Infra-Rouge	71.88%	67.61%	83.13%	74.40%
Bleu, Vert et Rouge	-	-	-	-
Bleu, Vert et Infra-Rouge	68.56%	64.33%	82.48%	72.06%
Bleu, Rouge et Infra-Rouge	69.80%	66.22%	80.37%	72.30%
Vert, Rouge et Infra-Rouge	66.65%	63.81%	84.85%	71.71%
Bleu et Vert	-	-	-	-
Bleu et Rouge	-	-	-	-
Bleu et Infra-Rouge	66.32%	62.62%	78.44%	69.42%
Vert et Rouge	-	-	-	-
Vert et Infra-Rouge	67.24%	61.74%	87.84%	72.49%
Rouge et Infra-Rouge	69.42%	64.39%	84.68%	73.10%
Bleu	-	-	-	-
Vert	-	-	-	-
Rouge	-	-	-	-
Infra-Rouge	63.37%	59.78%	78.87%	67.81%

Contrairement à son homologue 2A qui montre des résultats en moyenne plus élevés que les autres saisons, le printemps du type 1C a du mal à s'entraîner, voire même ne s'entraîne pas du tout lorsque la bande infra-rouge est absente... J'ai pourtant essayé de rajouter des neurones, de rajouter des couches cachées, de formater les données d'entrée, d'enlever une couche de convolution, de regarder si quelque chose clochait au niveau des entrées, regarder si la cross-validation prenait les deux classes en égal, etc... rien n'y a fait. Les coordonnées d'entraînement étaient pourtant les mêmes qu'à l'expérience [4.9.3](#).

De manière générale, les résultats obtenus par le produit 1C sont inférieurs au produit 2A. Cela vient sans doute du fait que les images de 1C sont plus floues que celles de 2A.



Figure 38 : À gauche une image de type 2A, à droite une image de type 1C

4.11 Sentinel-1 dans le temps

Dans cette expérience, je vais faire la même expérience sur Sentinel-1.

Cette fois les images d'été seront aussi utilisées les nuages ne gênant pas les radars du satellite. Contrairement à la dernière expérience avec Sentinel-1, nous utiliserons ici les images de Sentinel-1A, la 1B n'ayant pas beaucoup d'images GRD avant 2020 sur la zone observée. Ces images sont un peu plus petites, nous aurons donc un peu moins de données.

Contrairement à Sentinel-2, les images de Sentinel-1 ne sont pas toujours prises au même endroit, il y a donc des décalages plus ou moins grands entre 2 images prises à des moments différents. Si l'on souhaite regrouper plusieurs de ces images en une seule, il est donc important que toutes les positions tabulaires des différentes couches correspondent plus ou moins aux mêmes coordonnées géographiques.

Pour y parvenir, nous devons faire 2 choses : se mettre d'accord sur un point de départ et trouver des offsets pour faire coïncider les positions des cellules avec les coordonnées que nous voulons, en d'autres termes, la première cellule de chaque couche doit coïncider avec le point de départ.

Comme point de départ, j'ai simplement pris la latitude et la longitude les plus petites parmi les positions initiales (correspondant à la première cellule) de nos couches. Ceci me paraît une bonne approximation. Une autre solution aurait été de calculer la coordonnée engendrant le moins de perte entre toutes les couches, mais la complexité de cette solution me paraît trop élevée pour le temps qu'il me reste à disposition, pour finalement ne pas gagner beaucoup plus de données si les images sont proches.

Nous utilisons ensuite les offsets pour recalculer une nouvelle matrice en faisant une « translation » des données de la bande. Toutes les données manquantes (hors de la bande) seront comptabilisées en tant que 0.

Lors de l'entraînement, je me suis aperçu qu'il peut arriver parfois que le système n'arrive pas à trouver une façon de résoudre le problème³. Pour cette raison, j'ai donc mis beaucoup de neurones dans les différentes couches, ceci permet de diminuer l'apparition de ce problème.

Les bandes de Sentinel2 GRD (l'un des différents produits de Sentinel1) que nous utilisons contiennent 2 bandes, VV et VH. « V » pour vertical et « H » pour horizontal, l'ordre des lettres indique de gauche à droite l'émission et la réception [30]. À noter que j'utilise aussi une troisième bande qui a été créée à partir des deux bandes susmentionnées [31]. Les mesures prises dans le tableau de la page suivante ont été faites sur un entraînement de 750 époques.

Les bandes utilisée pour cette expérience ont été prises mi-janvier, mi-avril, mi-juillet et mi-octobre (2019-2020).

Test score: 0.5378505598468256
Test accuracy: 0.8117878437042236

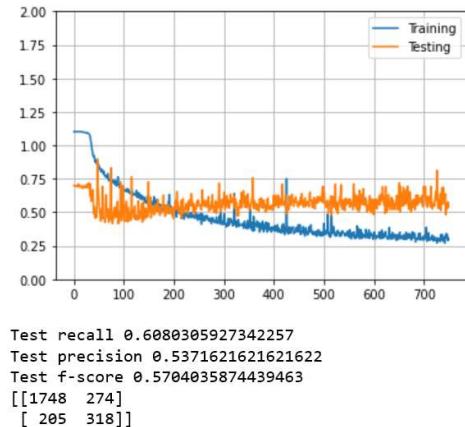


Figure 39 : Résultat d'une itération de Sentinel-1 avec toutes les saisons

³ On constate que sur le set d'entraînement, le système essaye de trouver les bons poids mais n'y arrive pas. La visibilité peut avoir de grands écarts entre deux époques voisines. Quant au set de test, on constate que toutes les prédictions (ou presque) sont faites dans la même classe. Dans les deux cas, la fonction d'erreur ne semble pas bouger.

Bandes utilisées	Visibilité	Précision	Rappel	F-Score
Toutes	83.45%	62.15%	51.11%	55.83%
VV et VH	83.27%	61.58%	51.61%	55.90%
VV et VV / VH	79.46%	50.91%	45.11%	47.44%
VH et VV / VH	81.03%	55.81%	49.24%	51.41%
VV	79.15%	49.40%	41.67%	44.70%
VH	81.17%	55.45%	48.09%	51.02%
VV / VH	77.98%	45.93%	34.25%	38.73%

Les points les plus importants dans ce tableau sont la visibilité et le rappel (les données « autre » étant quatre fois plus nombreuses que les champs de café, la précision et le f-score sont donc tirés vers le bas en comparaison avec un test totalement équilibré).

Lors de l'entraînement, j'ai pu constater que si nous ne prenons qu'une seule bande, le rappel peut varier beaucoup (allait de 25 à 50% si l'on ne prenait que la bande VV).

Il semblerait aussi que la bande |VV| / |VH| ne soit pas utilisable si l'on la normalise en la divisant par la valeur maximale d'un uint16. Après avoir vérifié, la valeur la plus haute obtenue pour cette bande étant 42, les valeurs seraient donc trop proches pour que le réseau arrive à faire quoi que ce soit. Je l'ai donc divisée par 100 lors de cet entraînement. J'ai aussi essayé les deux façons pour les bandes combinées, les résultats sont à peu près les mêmes.

Nous pouvons remarquer que le rappel du café n'est pas très haut. Au mieux, il peut atteindre les 60% si l'on utilise VV et VH en même temps, mais ceci est au détriment du rappel de la classe « autre », en plus d'être assez rare. En revanche, nous avons une assez bonne visibilité. Ceci est dû à une classification de la classe « autre » correcte, étant parfois supérieur à 90%, ce qui rend un tel réseau utilisable en complément d'un autre entraîné sur Sentinel-2.

Je conseillerai de prendre la combinaison de bande VV et VH, car elle est la seule qui arrive à avoir des résultats similaires à la combinaison des autres bandes. Pour ce qui est du découpage, on peut donc supprimer la création de la bande |VV| / |VH|. Cependant, une telle mesure impactera l'affichage des petites images dans QGIS.

Dans l'expérience ci-dessous, nous utiliserons seulement la combinaison de bande VV et VH. Nous allons voir quelles sont les saisons les plus pratiques pour la reconnaissance.

Saison(s) utilisée(s)	Visibilité	Précision	Rappel	F-Score
Printemps, été et automne	81.54%	55.47%	51.49%	53.26%
Eté, automne et hiver	81.48%	55.75%	55.12%	55.11%
Automne, hiver et printemps	81.93%	57.26%	53.41%	54.70%
Hiver, printemps et été	78.81%	48.38%	38.42%	42.57%
Printemps et été	-	-	-	-
Printemps et automne	79.09%	49.90%	49.92%	49.33%
Printemps et hiver	77.50%	43.94%	31.77%	36.44%
Eté et automne	81.41%	56.23%	45.53%	50.01%
Eté et hiver	77.33%	43.20%	30.81%	35.73%
Automne et hiver	81.17%	55.12%	48.96%	51.59%

Premièrement, on remarque que les combinaisons n'utilisant pas les bandes automnes sont derrière celles qui l'utilisent. On peut donc en déduire que les bandes automnales sont les meilleures pour réussir à trouver du café (même si le rappel reste faible).

Je n'ai cependant pas réussi à entraîner printemps et été seulement, et ce même en rajoutant des neurones et des couches supplémentaires dans mon réseau, ou en reformatant les données d'entrée. Très rarement, il arrive à apprendre lors de certaines parties de la validation croisée, mais les résultats n'excèdent pas ceux obtenus avec les bandes automnales. Mais dans la majorité des cas, il se contente de donner toujours la même prédiction.

Ma seule recommandation est donc d'utiliser une combinaison utilisant des bandes automnales, la différence entre les différentes combinaisons n'étant pas très importante.

On peut expliquer le rappel plutôt mauvais aux cultures assez hétéroclites qui sont assez fréquentes dans la région (je le rappelle Sentinel-1 n'aime pas trop la différence de végétation dans une même zone) et aussi à l'imprécision de la géolocalisation.

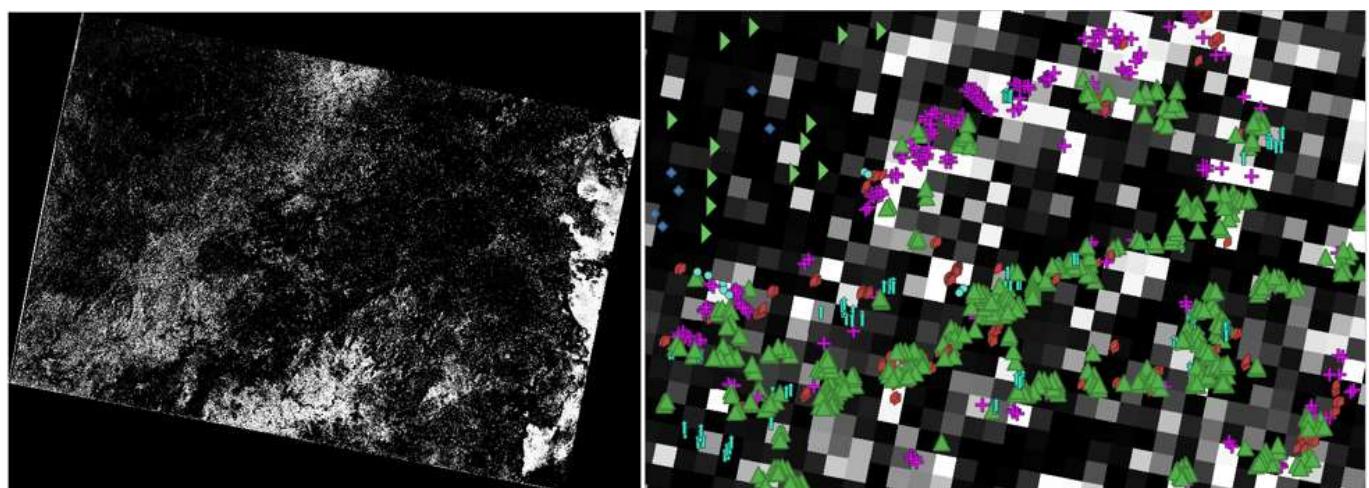


Figure 40 : : Image générée en prédiction de Sentinel-1 sur toutes les saisons avec les bandes VV et VH (total de l'image à gauche, zoom à droite avec les coordonnées du CIAT activées)

Les images ci-dessus montrent le résultat d'une prédiction de Sentinel-1. Dans l'image de gauche, on peut voir en bordure une sorte de cadre blanc. Cela est sûrement dû au manque de données occasionné par le rabotage des bandes pour pouvoir les superposer. Autre petit problème, on constate à droite une grande partie de cette image qu'il y a beaucoup de prédiction de café. Il s'agit en fait de la mer.

Si on regarde dans l'image de droite, on s'aperçoit que cette fois-ci les endroits où il n'y a naturellement pas d'arbre (diamond) et la forêt (triangle) sont très bien classés. On peut voir aussi, avec surprise, que les champs de café (cross), de poivre (triangle) et la culture intercalaire (square) sont très bien classés, étant soit dans la couleur qui lui est associée (blanc pour le café et noir pour le reste), soit dans la case d'à côté.

Bien que le modèle utilisé pour faire cette prédiction ait un rappel inférieur à 60% lors des tests, l'image de sortie me paraît totalement utilisable.

5 Description des scripts réalisés

Dans cette section, je ne parlerai pas des tutoriels faits, mais des scripts servant directement au projet.

5.1 Traitement des images

5.1.1 DownloadImages.ipynb

Ce notebook utilise le module SentinelSat pour télécharger les images des missions Sentinels.

Dans la capture d'écran ci-dessous, on voit la partie principale de cette application. Il suffit de commenter la partie de Sentinel-2 et de décommenter la partie de Sentinel-1 pour passer de l'un à l'autre. À noter que la fréquence de passage de Sentinel-2 sur une région donnée est de 5 jours et que celle-ci monte à 12 pour Sentinel-1. Il se peut aussi que sentinel-2 ne télécharge pas d'image au vu des conditions météorologiques (vous pouvez toujours changer les paramètres pour permettre plus de couverture nuageuse).

```
api = SentinelAPI(credential[0], credential[1], 'https://scihub.copernicus.eu/dhus')

# search by polygon, time, and SciHub query keywords
footprint = geojson_to_wkt(read_geojson('coordinates.geojson'))
products = api.query(footprint,
                     date=["NOW-5DAY", "NOW"],
                     #Pour sentinel-1
                     platformname='Sentinel-1',
                     producttype='GRD',
                     #Pour Sentinel-2
                     platformname='Sentinel-2',
                     producttype='S2MSI2A', #ou 'S2MSI1C'
                     cloudcoverpercentage=(0, 30)
                     )

# download all results from the search
if len(products) > 0:
    api.download_all(products, directory_path="todo")
else:
    print("No image")

Downloading:  0%|          | 0.00/1.08G [00:00<?, ?B/s] | 0.00/1.19G [00:00<?, ?B/s]
Downloading:  0%|          | 1.05M/1.19G [00:00<10:41, 1.86MB/s] | 2.10M/1.19G [00:01<10:14, 1.94MB/s]
```

Figure 41 : Importer des images avec SentinelSat

De plus amples détails sur le fonctionnement de ce notebook seront fourni dans la partie « Marche à suivre » de ce rapport.

5.1.2 DivideAndSort.ipynb

Ce script permet de diviser les images de Sentinel-2 en une multitude de petites images de 15 sur 15 pixels (si on ne change pas les paramètres par défaut). Cela nous est utile pour la création de données d'entraînement pour notre réseau de neurones.

Il faut lui donner le chemin et la partie du nom en commun des différentes images jp2 fournies par Sentinel-2. Pour entrer un peu plus dans les détails, ce script se sert des bandes 2, 3, 4 et 8 (bleu, vert, rouge et infra-rouge proche).

```
imagePath = '../Données/S2A_MSIL2A_20190227T030651_N0211_R075_T48PYU_20190227T083207/' + \
            'S2A_MSIL2A_20190227T030651_N0211_R075_T48PYU_20190227T083207.SAFE/GRANULE/' + \
            'L2A_T48PYU_A019234_20190227T031435/IMG_DATA/R10m/T48PYU_20190227T030651_'
band2 = rasterio.open(imagePath+'B02_10m.jp2', driver='JP2OpenJPEG') #blue
band3 = rasterio.open(imagePath+'B03_10m.jp2', driver='JP2OpenJPEG') #green
band4 = rasterio.open(imagePath+'B04_10m.jp2', driver='JP2OpenJPEG') #red
band8 = rasterio.open(imagePath+'B08_10m.jp2', driver='JP2OpenJPEG') #near impact
```

Figure 42 : DivideAndSort importer les bandes

Vous avez aussi la possibilité de lui passer des chemins vers d'éventuels shapefiles contenant des coordonnées géographiques déjà labellisées pour trier les différentes images de sorties dans les bons dossiers.

```
IMAGE_PIXEL = 15
IMAGE_WIDTH = 732
LIST_SHAPEFILES = ["../Données/heigvd/central_highlands_2_other/central_highlands_2_other.shp",
                   "../Données/heigvd/central_highlands_2_test/central_highlands_2_test.shp",
                   "../Données/heigvd/central_highlands_1_other/central_highlands_1_other.shp"]
CORRESPONDANCES = {"1": "cacao", "2": "coffee", "3": "complex_oil", "4": "nativevege", "5": "oil_palm", "6": "rubber", "7": "unknown",
                    "8": "seasonal", "9": "urban", "10": "water", "11": "other_tree", "12": "other_no_tree", "13": "native_no_tree",
                    "14": "water_other", "15": "pepper", "16": "cassava", "17": "tea", "18": "rice", "19": "banana", "20": "baby_palm",
                    "21": "cur_off-regrow", "22": "natural_wetland", "23": "intercrop", "24": "deciduous_forest", "25": "stick_pepper",
                    "26": "flooded_plantation", "27": "pine_trees", "28": "coconut", "29": "bamboo", "30": "savanna", "31": "mango",
                    "32": "other_fruit_tree_crop", "33": "water_mine", "0": "not_labeled", "-1": "ambiguous"}
SHAPEFILE_ESPG = 4326
```

Figure 43 : DivideAndSort, shapefiles et autres paramètres

Dans la capture d'écran ci-dessus, on peut voir la liste des shapefiles et la correspondance entre les valeurs des tags et leurs représentations.

La partie du code ci-dessous permet de découper une bande en bandes plus petites correspondant à un pixel de taille IMAGE_PIXEL. Par défaut, IMAGE_PIXEL est à 15 et IMAGE_WIDTH de 732. Il faudra découper 4 bandes pour créer les images plus petites, qui seront donc en 4 couches.

```
def split_band(band):
    """Split une bande de Sentinel-2 en bande plus petite
    IMAGE_WIDTH^2 images de IMAGE_PIXELxIMAGE_PIXEL"""
    result = []
    my_band = band.read(1)
    for x in range(IMAGE_WIDTH):
        columns = []
        for y in range(IMAGE_WIDTH):
            lines = []
            for i in range(IMAGE_PIXEL):
                cells = []
                for j in range(IMAGE_PIXEL):
                    cells.append(my_band[i + IMAGE_PIXEL * x][j + IMAGE_PIXEL * y])
            lines.append(cells)
        columns.append(lines)
    result.append(columns)
    return result
```

Figure 44 : Découper une bande en bandes plus petites

Ce code est aussi présent dans les autres fichiers de découpages et dans les scripts de prédiction.

Le bout de code suivant est utilisé pour pouvoir spécifier une bonne localisation géographique à notre fragment d'image. Cela permet de l'afficher à la bonne place dans QGIS.

```
for x in range(IMAGE_WIDTH):
    for y in range(IMAGE_WIDTH):
        begin = blue.transform * (y * IMAGE_PIXEL, x * IMAGE_PIXEL)
        end = blue.transform * ((y + 1) * IMAGE_PIXEL, (x + 1) * IMAGE_PIXEL)
        transform = affine.Affine(10.0, 0, begin[0], 0.0, -10.0, begin[1])
```

Figure 45 : Calcul des coordonnées de début et de la fin d'une image découpée de Sentinel-2

La fonction suivante permet de savoir si des coordonnées taguées sont dans le bout d'image que nous traitons et dans l'affirmative quel est leur type. Il utilise la liste des shapefiles pour tout stocker toutes les coordonnées dans une liste.

```
points = []
for path in LIST_SHAPEFILES:
    sf = shapefile.Reader(path)
    shapes = sf.shapes()
    for point in sf.records():
        points.append((shapes[point.oid].points[0], point[0]))

def get_labels(begin, end):
    """Donne tous les labels contenu entre 2 positions géographiques"""
    labels = set()
    long1, lat1 = convert_to_format(begin)
    long2, lat2 = convert_to_format(end)
    for point in points:
        if long1 <= point[0][0] <= long2 and lat2 <= point[0][1] <= lat1:
            labels.add(point[1])
    return labels
```

Figure 46 : Savoir quels labels se trouvent dans une image de Sentinel-2

Les images de Sentinel-2 n'ayant pas le même système de coordonnées que les données fournies par QGIS, nous les convertissons via la fonction suivante.

```
transformer = Transformer.from_crs(band2.gcps[1], SHAPEFILE_ESPG)
def convert_to_format(coords):
    """Sert à convertir le format de géolocalisation de l'image pour matcher avec celui des points des shapefiles"""
    conversion = transformer.transform(coords[0], coords[1])
    return (conversion[1], conversion[0])
```

Figure 47 : Conversion des systèmes de coordonnées géographiques (réutilisation d'un labo de VTK)

Le bout de code suivant montre comment on s'y prend pour savoir dans quel fichier nous devons mettre l'image de sortie courante. On regarde simplement les labels qui lui sont attribués (en utilisant la liste des labels fournie). S'il en a 0, le fichier sera enregistré dans « not_labeled », s'il en a plus d'un il sera enregistré dans « ambiguous » et dans les autres cas, il sera enregistré dans le dossier correspondant au label qui lui est attribué. Ceci permet de gagner pas mal de temps lors de la sélection des données d'entraînement.

```

if len(labels) == 0:
    if not os.path.exists(path_name + "/" + CORRESPONDANCES['0'] + "/" + str(x)):
        os.mkdir(path_name + "/" + CORRESPONDANCES['0'] + "/" + str(x))
    image_name += CORRESPONDANCES['0'] + "/" + str(x) + "/"
elif len(labels) > 1:
    image_name += CORRESPONDANCES['-1'] + "/"
else:
    image_name += CORRESPONDANCES[str(next(iter(labels)))] + "/"
image_name += file_name
smaller_image = rasterio.open(image_name + '_' + str(x) + '_' + str(y) +
                             '.tiff', 'w', driver='GTiff',
                             width=IMAGE_PIXEL, height=IMAGE_PIXEL, count=4, crs=blue.crs,
                             transform=transform,
                             dtype=blue.dtypes[0]
)

```

Figure 48 : Classer automatiquement les images Sentinel-2

Le dossier suivant devrait être généré.

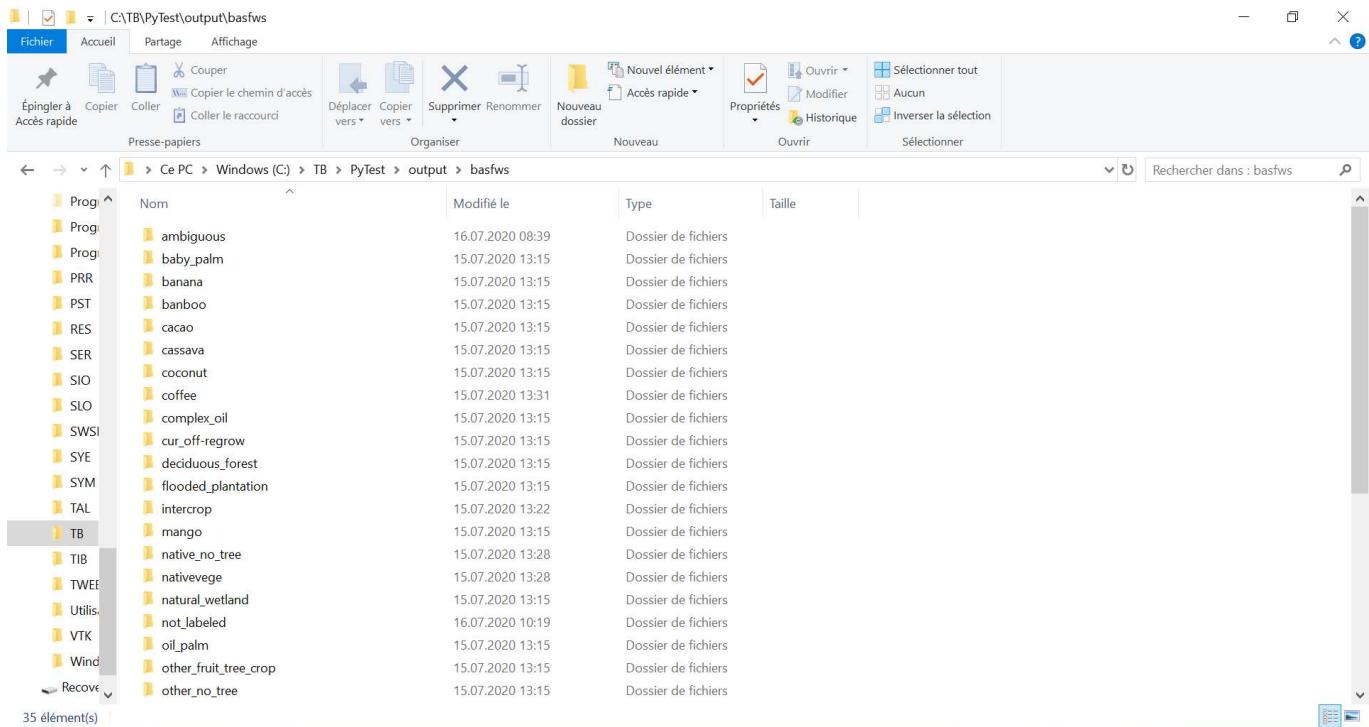


Figure 49 : Dossier généré par DivideAndSort

La première composante de l'image est l'infra-rouge, suivie du rouge, puis du vert et finalement du bleu.

5.1.3 DivideAndSort-FallWinterSpring.ipynb

Version de DivideAndSort à 12 bandes. Ce script permet de créer des images contenant 3 saisons (automne, hiver et printemps, l'été étant un peu trop nuageux). L'ordre des composantes ne change pas par rapport au code de DivideAndSort, cependant on met d'abord les bandes appartenant à l'automne, puis celles qui appartiennent à l'hiver avant de finir par le printemps.

5.1.4 DivideAndSort-Final.ipynb

Version de DivideAndSort à bandes modulables. Il suffit de mettre toutes les bandes que nous souhaitons avoir pour entraîner notre modèle dans une liste nommée « bands ». L'ordre des bandes sera respecté dans les images ainsi créées.

De manière générale, je recommande de mettre toutes les bandes possibles (de même résolutions) si vous souhaitez faire une sélection des bandes les plus utiles dans le système. Il vous suffira ensuite de commenter les bandes que vous ne souhaitez pas tester dans le script d'entraînement. Comme ceci, vous n'aurez pas besoin de générer un set d'entraînement pour chaque combinaison testée.

5.1.5 TranslateS1Bands.ipynb

Ce script permet de faire une translation de plusieurs bandes de Sentinel-1 pour faire coïncider une coordonnée de départ avec la première cellule de leur matrice. À la base, je voulais faire ceci directement dans les notebooks de découpage et de prédiction de Sentinel-1, mais en faisant ceci je perdais toujours la connexion avec le serveur lorsque je lançais les programmes. J'ai quand même laissé les fonctions utilisées dans les notebooks susmentionnés.

Ce fichier prend en entrée une liste d'images de Sentinel-1 (VV ou VH) et sortira une version « déplacée » de chacune de ces bandes.

Il est recommandé de mettre les fichiers ainsi générés en entrée de DivideAndSortS1 pour créer un set d'entraînement ou à PredictS1-FinalVersion en plus d'un modèle entraîné sur Sentinel-1 si vous souhaitez faire une prédiction.

```
def get_vectors(gcp_top_right, gcp_top_left, gcp_bottom_right, gcp_bottom_left):
    norm_v1 = (gcp_top_left.col - gcp_top_right.col)
    v1 = np.array([(gcp_top_right.x - gcp_top_left.x + gcp_bottom_right.x - gcp_bottom_left.x) / (2 * norm_v1),
                  (gcp_top_right.y - gcp_top_left.y + gcp_bottom_right.y - gcp_bottom_left.y) / (2 * norm_v1)])
    norm_v2 = (gcp_bottom_right.row - gcp_top_right.row)
    v2 = np.array([(gcp_top_right.x - gcp_bottom_right.x + gcp_top_left.x - gcp_bottom_left.x) / (2 * norm_v2),
                  (gcp_top_right.y - gcp_bottom_right.y + gcp_top_left.y - gcp_bottom_left.y) / (2 * norm_v2)])
    return v1, v2
```

Figure 50 : Fonction donnant les vecteurs (approximatifs) pour passer d'un GroundControlPoint à ses voisins

Nous avons besoin de calculer les vecteurs permettant de passer d'un GroundControlPoint à un autre pour trouver de combien de cases il est nécessaire de déplacer les matrices de nos différentes bandes.

Il est aussi important que les coordonnées GCPS commence au même point. Pour ce faire, il faut en recréer des nouvelles avec une position de départ précise et commune à toutes les bandes. Les autres positions seront calculées en moyennant les différences de positions. Cette façon de faire est assez imprécise, mais permet quand même une assez bonne approximation, si les bandes sont assez proche les unes des autres.

```
from osgeo import gdal
def create_gcps(list_gcps):
    """Fait une approximation des gcps en moyennant les gcps de toutes les bandes de saisons (prend comme point de départ le plus petit x et le plus petit y trouvé)"""
    firsts_x = []
    firsts_y = []
    for band in list_gcps:
        firsts_x.append(band[0].x)
        firsts_y.append(band[0].y)
    x_min = min(firsts_x)
    y_min = min(firsts_y)
    gcps = [rasterio.control.GroundControlPoint(row=0.0, col=0.0, x=x_min, y=y_min, id='1')]
    for i in range(1, len(list_gcps[0])):
        previous_x_average = 0
        previous_y_average = 0
        actual_x_average = 0
        actual_y_average = 0
        for band in list_gcps:
            previous_x_average += band[i-1].x
            previous_y_average += band[i-1].y
            actual_x_average += band[i].x
            actual_y_average += band[i].y
        x = gcps[-1].x + (actual_x_average - previous_x_average) / len(list_gcps)
        y = gcps[-1].y + (actual_y_average - previous_y_average) / len(list_gcps)
        gcps.append(rasterio.control.GroundControlPoint(row=list_gcps[0][i].row, col=list_gcps[0][i].col,
                                                        x=x, y=y, id=str(i+1)))
    return gcps
```

Figure 51 : Créer des coordonnées GCPS communes à toutes les bandes de sorties

Après avoir trouvé cette position de départ commune, il faut trouver à combien de cases correspond le décalage pour faire matcher la première position de notre bande à cette position. La capture d'écran au haut de la page suivante montre comment nous avons procédé.

```
from sympy import symbols, Eq, solve
def get_offset(gcps, x_min, y_min):
    gcp_top_right, gcp_top_left, gcp_bottom_right, gcp_bottom_left = None, None, None, None
    for gcp in gcps:
        if gcp.row == 0.0 and gcp.col == 0.0:
            gcp_top_right = gcp
        elif gcp.row == 0.0 and gcp_top_left == None:
            gcp_top_left = gcp
        elif gcp.col <= 0.0 and gcp_bottom_right == None:
            gcp_bottom_right = gcp
        elif gcp_bottom_right != None:
            gcp_bottom_left = gcp
            break
    v1, v2 = get_vectors(gcp_top_right, gcp_top_left, gcp_bottom_right, gcp_bottom_left)
    origin = np.array([gcp_top_right.x, gcp_top_right.y])
    u1, u2 = symbols('u1 u2')
    eq1 = Eq(gcp_top_right.x - u1*v1[0] - u2*v2[0] - x_min, 0)
    eq2 = Eq(gcp_top_right.y - u1*v1[1] - u2*v2[1] - y_min, 0)
    sol = solve((eq1, eq2), (u1, u2))
    return round(sol[u1]), round(sol[u2])
```

Figure 52 : Trouver les de combien de cases il faut décaler la matrice d'une bande donnée

Il suffit donc maintenant de faire une translation de l'image en utilisant les offsets trouvé à l'étape précédente en remplaçant les données manquantes par des zéros.

```
def translate_image(image, offset_x, offset_y):
    result = []
    for i in range(len(image)):
        if (i % 1000) == 0:
            print(i)
        if 0 <= (i + offset_y) < len(image):
            result_row = []
            for j in range(len(image[0])):
                if 0 <= (j + offset_x) < len(image[0]):
                    result_row.append(image[i + offset_y][j + offset_x])
                else:
                    result_row.append(0)
            result.append(result_row)
        else:
            result.append([0 for i in range(len(image[0]))])
    return np.array(result, dtype=np.uint16)
```

Figure 53 : Faire une translation d'une bande de Sentinel-1

5.1.6 DivideAndSortS1.ipynb

Ce script est la version Sentinel-1 des scripts ci-dessus. Ici, nous prenons les bandes vv et vh données par Sentinel-1 (mission GRD). Et on génère une image en RGB grâce à elle [31]. Ensuite, nous découpons cette image en images plus petites. Même si ce script à des parties de son code en commun avec son équivalent Sentinel-2, nous ne pouvions pas utiliser ce dernier pour les images de Sentinel-1 car la géolocalisation est gérée différemment et nous avions besoin de générer une troisième bande à partir des deux données. À noter que ce découpage ne recouvre pas entièrement les images de bases.

Les chemins vers les deux bandes de Sentinel-1 sont définis ainsi :

```
IMAGE_PIXEL = 30
DEFAULT = "not_labeled"
S1DIRECTORY = "output/sentinel1Simple"
output_filename = 'tile_'

vh = rasterio.open('S1Spring/measurement/s1a-iw-grd-vh-20200408t223649-20200408t223714-032042-03b3ce-002.tif')
vv = rasterio.open('S1Spring/measurement/s1a-iw-grd-vv-20200408t223649-20200408t223714-032042-03b3ce-001.tif')
```

Figure 54 : DivideAndSortS1, importer les bandes

Pour la génération des images plus petites, le fait que les images de Sentinel-1 utilisent des GroundControlPoints pour placer correctement les pixels, il est impossible d'utiliser le code de DivideAndSort.

Pour se rendre compte de comment sont arrangés les pixels dans une image de Sentinel-1, voici un exemple avec à gauche une image de Sentinel-2 et à droite une image de Sentinel-1 RGB créée par le programme. Les deux images ont le même nombre de valeurs stockées qui est ici de 90 fois 90.

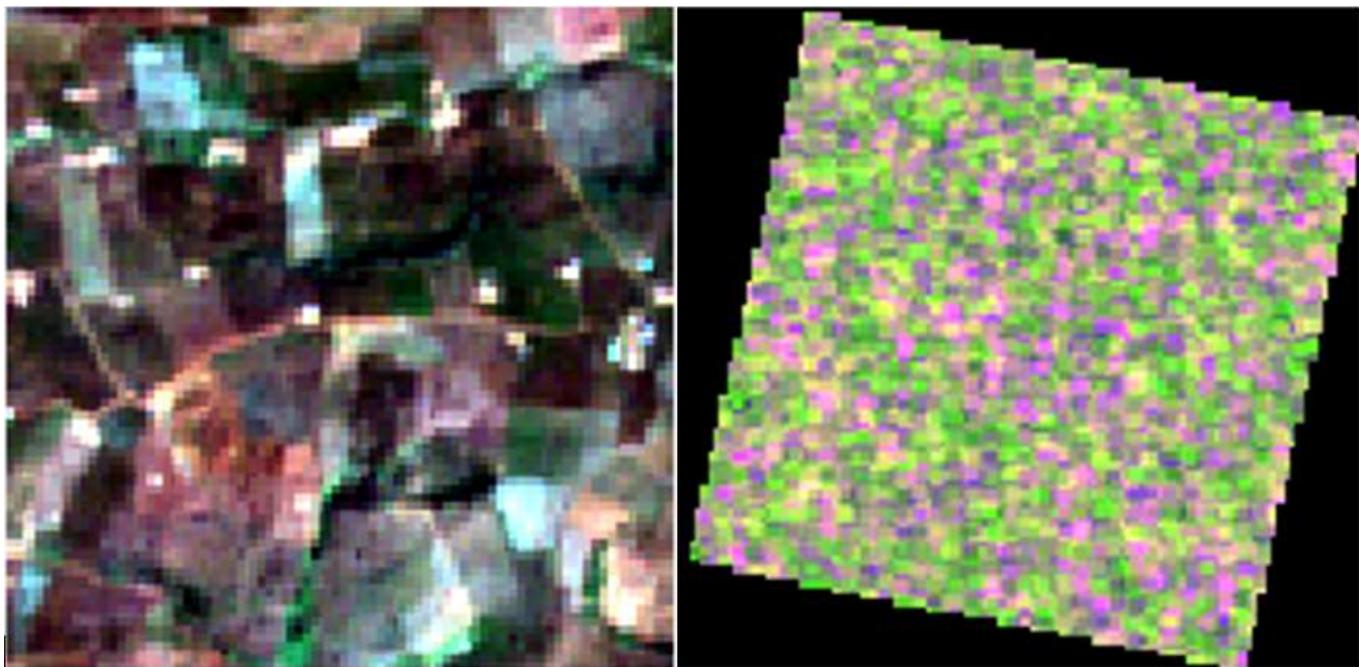


Figure 55 : Différence entre le placement des cellules entre Sentinel-2 (gauche) et Sentinel-1 (droite)

On voit que les pixels sont déplacés, un peu comme si on utilisait une rotation. Les parties noires ne représentent pas des valeurs, mais peuvent être considérées comme étant une sorte de cadre. Si on ouvrait ces différentes images avec Rasterio (ou autres), la première donnée du premier vecteur de Sentinel-2 correspond au pixel en haut à gauche de l'image. En ce qui concerne celle de Sentinel-1, elle correspond au pixel le plus à droite, ce qui n'est pas intuitif.

Pour diviser les images, je dois passer par GDAL. Pour envoyer une commande directement au système. J'avais dans un premier temps essayé de recréer des GroundControlPoints en interpolant ceux présents dans l'image de base, mais cela ne marche malheureusement pas.

```
import os, gdal

out_path = 'output/sentinel1/'
output_filename = 'tile_'

in_file = 'output/TempRGB.tif'

ds = gdal.Open(in_file)
band = ds.GetRasterBand(1)

xsize = band.XSize
ysize = band.YSize

for i in range(0, xsize, IMAGE_PIXEL):
    for j in range(0, ysize, IMAGE_PIXEL):
        com_string = "gdal_translate -of GTIFF -srcwin " + str(i) + ", " + str(j) + ", " + str(IMAGE_PIXEL) + \
                    "\n", " + str(IMAGE_PIXEL) + " " + str(in_file) + " " + str(out_path) + str(output_filename) + \
                    "\n", str(int(i/IMAGE_PIXEL)) + "_" + str(int(j/IMAGE_PIXEL)) + ".tif"
        os.system(com_string)
```

Figure 56 : Découper une image de Sentinel-1 en images plus petites

Pour ce faire, nous sommes obligés de d'abord créer une image entière contenant toutes les bandes avant de la découper.

Quant à la génération de la bande bleue, j'utilise simplement le code ci-dessous. Numpy me permet de retourner 0 au cas où l'une des valeurs de vh est de 0 (ce qui est commun), ceci évite de retourner une erreur lors de la division.

```
def make_blue(vv, vh):
    return (vv + vh // 2) // vh
```

Figure 57 : Créer la bande bleue

À noter qu'au vu des tests fait sur Sentinel-1, la bande bleue ne paraît pas être utile pour entraîner/tester un réseau de neurones. Elle est cependant utile pour compléter une image si l'on souhaite la regarder.

Pour ce qui est de la classification automatique, cela est beaucoup plus compliqué qu'avec Sentinel-2. Ici nous sommes obligés de connaître les GroundControlPoints de l'image, puis d'approximer quelles seraient les coordonnées d'un GroundControlPoint dont la colonne et la ligne valent zéro, puis d'approximer les trois autres coins de notre quadrilatère. Après ceci, il faut utiliser l'interpolation pour savoir si un label se trouve ou non dedans. Seul problème, la précision des floats ne permet pas toujours (voir même presque jamais) de faire cette interpolation. Dans ce cas, nous faisons une moyenne des bords pour pouvoir faire le tri parmi nos images. Ceci reste imprécis, mais les labels prédits se trouvent au pire à 200 mètres de l'image ainsi générée.

```
def XY_to_LM_init(px, py):
    """Crée deux vecteurs servant à voir si une coordonnée se trouve dans l'image ou non (réutilisation du labo5 de VTK)"""
    A = np.array([[1, 0, 0, 0],
                  [1, 1, 0, 0],
                  [1, 1, 1, 1],
                  [1, 0, 1, 0]])

    AI = np.linalg.inv(A)
    a = np.dot(AI, px)
    b = np.dot(AI, py)
    return a, b

def XY_to_LM(latitude, longitude, a, b):
    """Retourne l et m, si l et m sont entre 0 et 1 alors la coordonnée donnée est dans le
    carré (réutilisation du labo5 de VTK)"""
    aa = a[3] * b[2] - a[2] * b[3]
    bb = a[3] * b[0] - a[0] * b[3] + a[1] * b[2] - a[2] * b[1] + longitude * b[3] - latitude * a[3]
    cc = a[1] * b[0] - a[0] * b[1] + longitude * b[1] - latitude * a[1]
    det = math.sqrt(bb * bb - 4 * aa * cc)
    m = (-bb + det) / (2 * aa)
    l = (longitude - a[0] - a[2] * m) / (a[1] + a[3] * m)
    return l, m
```

Figure 58 : Fonctions servant à l'interpolation dans un quadrilatère quelconque

```
LIST_OF_DIRECTORIES = ["output/cafe_bas_mini", "output/cafe_droite_mini", "output/cafe_gauche_mini",
                      "output/cafe_milieu_mini", "output/cafe_haut_mini", "output/cafe_poivre_haut_mini",
                      "output/cafe_poivre_milieu_mini", "output/cafe_poivre_gauche_mini", "output/mer_bas_mini",
                      "output/forêt_bas_mini", "output/rubber_bas_mini", "output/ville_bas_mini",
                      "output/poivre_bas_mini", "output/poivre_haut_mini", "output/poivre_milieu_mini",
                      "output/poivre_droite_mini", "output/poivre_gauche_mini", "output/rubber_gauche_mini"]

COUNTAIN_START = "output/cafe_"
PIXEL_LENGTH = 15
BAND_WIDTH = 4
NB_SPLIT = 5
EPOCH = 2000
```

Figure 59 : Définition des hyperparamètres et des dossiers servant de données d'entraînement

```

def get_corners(image):
    gcps, _ = image.gcps
    #Calcul des gcps voisins
    gcp_top_right, gcp_top_left, gcp_bottom_right, gcp_bottom_left = None, None, None, None
    for gcp in gcps:
        if gcp.row <= 0 and gcp.col <= 0:
            gcp_top_right = gcp
        elif gcp.row <= 0 and gcp.col >= IMAGE_PIXEL and (gcp_top_left == None or gcp_top_left.col >= gcp.col):
            gcp_top_left = gcp
        elif gcp.row >= IMAGE_PIXEL and gcp.col <= 0 and (gcp_bottom_right == None or gcp_bottom_right.row >= gcp.row):
            gcp_bottom_right = gcp
        elif gcp.row >= IMAGE_PIXEL and gcp.col >= IMAGE_PIXEL:
            gcp_bottom_left = gcp
            break

    #Pour les cas en bordure
    if gcp_bottom_left == None:
        gcp_bottom_left = gcps[-1]
    if gcp_bottom_right == None or gcp_top_left == None:
        for gcp in gcps:
            if gcp.row == gcp_top_right.row and gcp.col == gcp_bottom_left.col:
                gcp_top_left = gcp
            elif gcp.row == gcp_bottom_left.row and gcp.col == gcp_top_right.col:
                gcp_bottom_right = gcp
            if gcp_bottom_right != None and gcp_top_left != None:
                break
    #Calcul des vecteurs
    v1, v2 = get_vectors(gcp_top_right, gcp_top_left, gcp_bottom_right, gcp_bottom_left)
    #Calcul des coins
    top_right = (np.array([gcp_top_right.x, gcp_top_right.y]) + gcp_top_right.col * v1 + gcp_top_right.row * v2)
    top_left = top_right - IMAGE_PIXEL * v1
    bottom_right = top_right - IMAGE_PIXEL * v2
    bottom_left = bottom_right - IMAGE_PIXEL * v1
    return {"tr":top_right, "tl":top_left, "br":bottom_right, "bl":bottom_left}

```

Figure 60 : Fonction permettant d'avoir les coins d'une image

```

def get_labels(image):
    """Donne tous les labels contenu dans une image de Sentinel-1"""
    corners = get_corners(image)
    labels = set()
    px = np.array([corners["bl"][0], corners["br"][0], corners["tr"][0], corners["tl"][0]])
    py = np.array([corners["bl"][1], corners["br"][1], corners["tr"][1], corners["tl"][1]])
    a, b = XY_to_LM_init(px, py)
    #A cause de la précision des floats, il arrive bien souvent que le dernier vecteur soit mis à 0
    #ce qui rend la localisation infaisable (retournera nan au lieu d'un float)
    if a[3] != 0 and b[3] != 0:
        for point in points:
            l, m = XY_to_LM(point[0][1], point[0][0], a, b)
            if 0 <= l <= 1 and 0 <= m <= 1:
                labels.add(point[1])
    else: #Si problème de précision, on fait une approximation
        for point in points:
            if ((corners["bl"][0] + corners["tl"][0]) / 2) < point[0][0] < ((corners["br"][0] + corners["tr"][0]) / 2) and \
               ((corners["bl"][1] + corners["br"][1]) / 2) < point[0][1] < ((corners["tl"][1] + corners["tr"][1]) / 2):
                labels.add(point[1])
    return labels

```

Figure 61 : Fonction de labellisation des images de Sentinel-1

5.1.7 DivideAndSortS1-AllSeasons.ipynb

Version améliorée de DivideAndSortS1, mais pour les quatre saisons.

Il est important de savoir que contrairement à Sentinel-2, Sentinel-1 ne fait pas toujours ses radars au même endroit, il y a donc un petit décalage entre chaque image de Sentinel-1. Il est donc important de faire en sorte que chaque case des bandes que nous utilisons représente plus ou moins la même position géographique. C'est pourquoi j'ai créé une fonction permettant de décaler une bande de Sentinel-1 avec des offsets définis (voir TranslateS1Bands).

Dans un premier temps j'ai dû « décaler » mes bandes sur mon ordinateur avant de créer l'image totale sur Hyperion, car je perdais la connexion quand j'essayais de faire le tout dans un seul notebook. Les images de Sentinel-1 étant beaucoup plus grandes que celles de Sentinel-2, je pense qu'il s'agissait d'un problème de mémoire allouée.

5.2 Entraînement

5.2.1 Train.ipynb

Ce notebook est le script permettant d'entraîner un réseau de neurones sur des images découpées de Sentinel-2.

Il prendra tous les fichiers se trouvant dans la liste des dossiers donnés comme ensemble d'entraînement. Ici on se base sur le chemin relatif de l'image (nom du dossier + nom de l'image) pour savoir à quelle classe elle appartient.

Dans l'image ci-dessus, on voit comment sont représentés les chemins des dossiers contenant les données d'entraînement. La variable COUNTAIN_START servira à attribuer la réponse attendue de la part du réseau, selon si le nom commence ou non par sa valeur. PIXEL_LENGTH indique la longueur/hauteur des images. BAND_WIDTH le nombre de bande que l'image contient (ici 4 pour bleu, vert, rouge et infra-rouge). NB_SPLIT représente le nombre de partitions des données d'entraînement pour la validation croisée. Finalement EPOCH représente le nombre de fois que l'on doit présenter les données d'entraînement à un modèle.

Dans sa configuration actuelle, ce réseau de neurones convolutif prend des images de 15 sur 15 pixels et fait de la validation croisée en divisant la base de données en 5.

La validation croisée a été réalisée comme ceci :

```
kfold = StratifiedKFold(n_splits=NB_SPLIT)
```

```
for train, test in kfold.split(images_x, images_y):
```

Figure 62 : Création d'une validation croisée avec Sklearn

Pour équilibrer les différentes classes, nous utiliserons la partie ci-dessous.

```
class_weight = None
pas_cafe = np.count_nonzero(images_y < 1)
cafe = len(images_y) - pas_cafe
print(pas_cafe, len(images_y))
print(pas_cafe / len(images_y))
if pas_cafe < cafe:
    class_weight = {0: cafe / pas_cafe, 1: 1.}
else:
    class_weight = {0: 1., 1: pas_cafe / cafe}
```

Figure 63 : Définir le poids des classes

Ceci permet d'utiliser le paramètre class_weight de la fonction fit de Keras pour montrer les données appartenant à la classe minoritaire plus souvent lors de l'entraînement.

Pour rajouter des données et diminuer l'effet d'overfitting, nous tournons aussi nos images. Cela se fait grâce à la fonction se trouvant au haut de la page suivante.

```
def add_all_rotations(images):
    result = []
    for image in images:
        result.append(image)
        for i in range(1, 4):
            rotated_image = []
            for j in range(BAND_WIDTH):
                rotated_image.append(np.rot90(image[0][j], i))
            result.append((rotated_image, image[1]))
    return result
```

Figure 64 : Tourner les images

Nous les tournons via la fonction rot90 de Numpy.

Pour passer les images de type raster en images traitables par Tensorflow, j'utilise le code ci-dessous :

```
def reshape(image):
    """Reformatte les données rasterio pour être utilisée par Tensorflow"""
    reshaped_image = []
    for i in range(len(image[0])):
        reshaped_row = []
        for j in range(len(image[0][0])):
            reshaped_cell = []
            reshaped_cell.append(image[0][i][j] / 65535) #Infra-rouge
            reshaped_cell.append(image[1][i][j] / 65535) #Rouge
            reshaped_cell.append(image[2][i][j] / 65535) #Vert
            reshaped_cell.append(image[3][i][j] / 65535) #Bleu
            reshaped_row.append(reshaped_cell)
        reshaped_image.append(reshaped_row)
    return reshaped_image
```

Figure 65 : Formater les données de type raster pour qu'elles soient utilisables par Tensorflow

Dans cet exemple, nous mettons toutes les bandes dans notre image finale. Nous pouvons aussi commenter les bandes que nous ne souhaitons pas utiliser pour pouvoir tester l'efficacité de certaines (ne pas oublier de changer aussi la variable BAND_WIDTH).

Il est important de d'abord rajouter les images, puis de les reformater, car si nous faisions l'inverse, nous devrions faire tourner une matrice dont chaque case contient une liste de valeurs, ce qui ne semble pas possible avec la fonction rot90.

Ces données d'entrée seront ensuite stockées dans la variable images_x et les réponses attendues en sortie dans image_y.

Le script enregistrera tous les modèles créés lors de l'exécution au même niveau que lui.

Le code ci-dessous montre la configuration des couches.

```
# create model
model = Sequential()
model.add(Conv2D(16, (3,3), activation='relu', input_shape=(PIXEL_LENGTH,PIXEL_LENGTH,BAND_WIDTH)))
model.add(MaxPooling2D(pool_size = (2 ,2)))
model.add(Conv2D(8, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2 ,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
history = model.fit(images_x[train], images_y[train], epochs=EPOCH, batch_size=32,
                      validation_data=(images_x[test], images_y[test]), class_weight=class_weight)
```

Figure 66 : Réseau de neurones de base

La sortie sera proche de 1 si le réseau pense que l'image présentée contient un champ de café. Dans le cas contraire, la réponse sera proche de 0.

Il crée aussi 5 modèles par exécution (1 pour chaque validation croisée) au format h5.

Nous utilisons que 2 couches de convolutions car c'est le maximum utilisable avec des images de 15 sur 15 pixels (si plus la couche flatten ne recevrait aucune donnée).

5.2.2 Train-FallWinterSpring.ipynb

Version dérivée de Train. Il prend juste 12 bandes en entrées au lieu de 4.

5.2.3 Train_one_test_others-FallWinterSpring.ipynb

Version dérivée de Train-FallWinterSpring. Ici on ne fait pas de validation croisée. On ne fait qu'entraîner nos données sur une zone et on teste sur toutes les autres. La capture d'écran ci-dessous montre comment cette tâche a été réalisée.

```
i = 0
for key in data:
    # balance classes
    class_weight = None
    coffee = np.count_nonzero(data[key]["y"] == 1)
    other = len(data[key]["y"]) - coffee
    if other >= coffee:
        class_weight = {0: 1., 1: other / coffee}
    else:
        class_weight = {0: coffee / other, 1: 1.}
    # create model
    model = Sequential()
    model.add(Conv2D(32, (3,3), activation='relu', input_shape=(PIXEL_LENGTH,PIXEL_LENGTH,BAND_WIDTH)))
    model.add(MaxPooling2D(pool_size = (2 ,2)))
    model.add(Conv2D(16, (3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size = (2 ,2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # Fit the model
    history = model.fit(data[key]["x"], data[key]["y"], epochs=EPOCH, batch_size=32, class_weight=class_weight)

    print("Trained on", key)
    avg_scores = []
    avg_recall = []
    avg_precision = []
    avg_fscore = []
    model.save("model_" + key + ".h5")
    for key2 in data:
        if key2 != key:
            # evaluate the model
            print("Tested on", key2)
            scores = model.evaluate(data[key2]["x"], data[key2]["y"], verbose=0)
            print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
            avg_scores.append(scores[1] * 100)

            print('Test score:', scores[0])
            print('Test accuracy:', scores[1])

            pred = model.predict_classes(data[key2]["x"])
            f, r, p = get_fscore_recall_precision(data[key2]["y"], pred)
            print('Test recall', r)
            print('Test precision', p)
            print('Test f-score', f)
            avg_recall.append(r * 100)
            avg_precision.append(p * 100)
            avg_fscore.append(f * 100)
            print(me.confusion_matrix(data[key2]["y"], pred))

    print("Visibility %.2f%% (+/- %.2f%%)" % (np.mean(avg_scores), np.std(avg_scores)))
    print("Recall %.2f%% (+/- %.2f%%)" % (np.mean(avg_recall), np.std(avg_recall)))
    print("Precision %.2f%% (+/- %.2f%%)" % (np.mean(avg_precision), np.std(avg_precision)))
    print("F-Score %.2f%% (+/- %.2f%%)" % (np.mean(avg_fscore), np.std(avg_fscore)))
```

Figure 67 : Réseau de neurones « entraîner sur une zone, tester sur les autres »

Ici les images et les réponses sont stockées dans un dictionnaire nommé « data » ayant une entrée par zone.

On affiche le résultat des tests après chaque zone testée, ceci pour pouvoir émettre des hypothèses, en fonction de la distance géographique ou la différence des données de la zone testée et de la zone d'entraînement.

Dans cette expérience, class_weight doit être recalculé pour chaque zone d'entraînement.

5.2.4 Train_two_test_others-FallWinterSpring.ipynb

Ce notebook est dérivé de Train_one_test_others-FallWinterSpring. Ici nous entraînerons avec 2 zones plutôt qu'une. Les remarques sont les mêmes que pour le notebook précédent.

```

for i in range(len(ZONES)):
    for j in range(i + 1, len(ZONES)):
        train_x = np.append(data[ZONES[i]]["x"], data[ZONES[j]]["x"], axis=0)
        train_y = np.append(data[ZONES[i]]["y"], data[ZONES[j]]["y"], axis=0)
        # balance classes
        class_weight = None
        coffee = np.count_nonzero(train_y == 1)
        other = len(train_y) - coffee
        if other >= coffee:
            class_weight = {0: 1., 1: other / coffee}
        else:
            class_weight = {0: coffee / other, 1: 1.}
        # create model
        model = Sequential()
        model.add(Conv2D(32, (3,3), activation='relu', input_shape=(PIXEL_LENGTH,PIXEL_LENGTH,BAND_WIDTH)))
        model.add(MaxPooling2D(pool_size = (2 ,2)))
        model.add(Conv2D(16, (3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size = (2 ,2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(32, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        # Fit the model
        history = model.fit(train_x, train_y, epochs=EPOCH, batch_size=32, class_weight=class_weight)

        print("Trained on", ZONES[i] + ZONES[j])
        avg_scores = []
        avg_recall = []
        avg_precision = []
        avg_fscore = []
        model.save("model_" + key + ".h5")
        for key2 in data:
            if key2 != ZONES[i] and key2 != ZONES[j]:
                # evaluate the model
                print("Tested on", key2)
                scores = model.evaluate(data[key2]["x"], data[key2]["y"], verbose=0)
                print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
                avg_scores.append(scores[1] * 100)

                print('Test score:', scores[0])
                print('Test accuracy:', scores[1])

                pred = model.predict_classes(data[key2]["x"])
                f, r, p = get_fscore_recall_precision(data[key2]["y"], pred)
                print('Test recall', r)
                print('Test precision', p)
                print('Test f-score', f)
                avg_recall.append(r * 100)
                avg_precision.append(p * 100)
                avg_fscore.append(f * 100)
                print(me.confusion_matrix(data[key2]["y"], pred))

        print("Visibility %.2f%% (%/- %.2f%%)" % (np.mean(avg_scores), np.std(avg_scores)))
        print("Recall %.2f%% (%/- %.2f%%)" % (np.mean(avg_recall), np.std(avg_recall)))
        print("Precision %.2f%% (%/- %.2f%%)" % (np.mean(avg_precision), np.std(avg_precision)))
        print("F-Score %.2f%% (%/- %.2f%%)" % (np.mean(avg_fscore), np.std(avg_fscore)))

```

Figure 68 : Réseau de neurones « entraîner sur deux zones, tester sur les autres »

5.2.5 Train-S1-AllSeasons.ipynb

Version dérivée de Train-FallWinterSpring, mais pour Sentinel-1. La seule grosse différence est qu'il a plus de neurones car l'entraînement de Sentinel-1 peut ne pas démarrer. Ceci ne permet seulement que de diminuer l'apparition de ce problème.

5.2.6 CoffeeVSPepperFallWinterSpring.ipynb

Ce notebook est une version multi classes servant à classer le café, le poivre et le reste.

Le seul changement notable est qu'il possède 3 neurones de sortie. Pour ce faire, je dois donc utiliser un vecteur one-hot dans les données à présenter en tant que sortie. Or, lorsque je fais ma validation croisée avec Sklearn, ce dernier n'aime pas avoir autre chose que des nombres en tant que données de sortie. Pour contourner ceci, je lui passe des nombres en entrée (2 pour café 1 pour poivre et 0 pour le reste) et puis, je les transforme en vecteur one-hot juste après avoir fait la séparation des données d'entraînement et de test.

```
correspondances = {'0':[1,0,0], '1':[0,1,0], '2':[0,0,1]}
def transform_output_value(liste):
    """StratifiedKFold n'aimant pas les valeurs de sortie en vecteur, cette fonction fera la correspondance entre
    une valeur et un vecteur"""
    result = []
    for value in liste:
        result.append(correspondances[str(value)])
    return np.array(result)

for train, test in kfold.split(images_x, images_y):
    train_y = transform_output_value(images_y[train])
    test_y = transform_output_value(images_y[test])
```

Figure 69 : Création des vecteurs one-hot

5.3 Test

5.3.1 CompareOneYearAgo.ipynb

Dans ce notebook, le but est de prendre un set d'entraînement d'une année donnée et un set équivalent d'une année antérieure. Puis, nous prenons un modèle et évaluons nos deux sets selon ce modèle.

```
from sklearn import metrics as me
# evaluate the model now
print("Now")
scores = model.evaluate(images_now, classes_now, verbose=0)
print("%: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print('Test accuracy:', scores[1])
pred = model.predict_classes(images_now)
f, r, p = get_fscore_recall_precision(classes_now, pred)
print('Test recall', r)
print('Test precision', p)
print('Test f-score', f)
print(me.confusion_matrix(classes_now, pred))

# evaluate the model a year ago
print("A year ago")
scores = model.evaluate(images_ago, classes_ago, verbose=0)
print("%: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print('Test accuracy:', scores[1])
pred = model.predict_classes(images_ago)
f, r, p = get_fscore_recall_precision(classes_ago, pred)
print('Test recall', r)
print('Test precision', p)
print('Test f-score', f)
print(me.confusion_matrix(classes_ago, pred))
```

Figure 70 : Comparer avec une année antérieure

Sauf s'il y a eu de gros changements au niveau du terrain ou si la couverture nuageuse n'est pas favorable, nous devrions avoir des résultats plus ou moins similaires.

5.4 Prédiction

5.4.1 Predict.ipynb

Ce script permet de faire une prédiction sur la base des images de Sentinel-2. Comme pour le script d'entraînement, on doit lui passer le chemin et la partie du nom en commun des différentes images correspondant aux bandes, cette fois-ci on lui passe en plus un modèle déjà entraîné.

Il fera la prédiction prenant des parties de 15 sur 15 pixels. Une fois finis, il générera une image de la même dimension que l'image de base. Plus une partie de l'image se rapproche du blanc, plus il y a de chance qu'un ou plusieurs champs de café se trouvent à l'endroit indiqué.

Le code ci-dessous sert à faire la prédiction sous-image par sous-image.

```
for x in range(IMAGE_WIDTH):
    row_prediction = []
    for y in range(IMAGE_WIDTH):
        smaller_image = []
        smaller_image.append(smaller_infra_red[x][y])
        smaller_image.append(smaller_red[x][y])
        smaller_image.append(smaller_green[x][y])
        smaller_image.append(smaller_blue[x][y])
        value = model.predict(np.expand_dims(reshape(smaller_image), axis=0))
        prediction_len.add(value[0][0] * 65535)
        for i in range(IMAGE_PIXEL):
            row_prediction.append(value[0][0] * 65535)

    for i in range(IMAGE_PIXEL):
        prediction.append(np.array(row_prediction).astype('uint16'))
output = rasterio.open(output_path + '_prediction.tif', 'w', driver='Gtiff',
                      width=red.width, height=red.height, count=1, crs=red.crs,
                      transform=red.transform,
                      dtype=red.dtypes[0])
output.write(prediction, 1)
output.close()
```

Figure 71 : Création du fichier de sortie

À noter que cette image-ci est aussi géolocalisée, ce qui permet en la superposant sur l'image de base et en la rendant transparente dans QGIS de mieux interpréter les résultats.

5.4.2 Predict-FinalVersion.ipynb

Version de Predict pour Sentinel-2 avec un modèle comportant n'importe quel nombre de bandes. Il faut juste faire attention à l'ordre des bandes lorsque vous les entrez dans le programme pour qu'il soit le même que celui utilisé pour l'entraînement du modèle utilisé.

```
def predict(bands, model, output_path="output", output_name="prédiction"):
    """Crée l'image de sortie en noir et blanc"""
    images = []
    width = None
    height = None
    crs = None
    transform = None
    dtype = None
    for band in bands:
        image = rasterio.open(band, driver='JP2OpenJPEG')
        images.append(split_band(image, IMAGE_WIDTH, IMAGE_PIXEL))
        if width == None:
            width = image.width
            height = image.height
            crs = image.crs
            transform = image.transform
            dtype = image.dtypes[0]
        image.close()
        print("Fin", band)
    prediction = []
    for x in range(IMAGE_WIDTH):
        row_prediction = []
        for y in range(IMAGE_WIDTH):
            smaller_image = []
            for image in images:
                smaller_image.append(image[x][y])
            value = model.predict(np.expand_dims(reshape(smaller_image, BAND_WIDTH), axis=0))
            for i in range(IMAGE_PIXEL):
                row_prediction.append(value[0][0] * 65535)

            for i in range(IMAGE_PIXEL):
                prediction.append(np.array(row_prediction).astype('uint16'))
    if not os.path.isdir(output_path):
        os.mkdir(output_path)
    output = rasterio.open(output_path + '/' + output_name + '.tiff', 'w', driver='GTiff',
                          width=width, height=height, count=1, crs=crs,
                          transform=transform,
                          dtype=dtype)
    output.write(prediction, 1)
    output.close()
    print("Le fichier de prédiction a été créé.")
```

Figure 72 : Prédiction avec n'importe quel nombre de bandes

5.4.3 Predict-Multi-FinalVersion.ipynb

Ce notebook fait la même chose que le précédent mais avec un modèle entraîner pour séparer le café du poivre du reste. Dans l'image de sortie le rouge indiquera une présence possible de café, le bleu une présence possible de poivre et le vert l'absence possible des deux.

5.4.4 PredictS1-FinalVersion.ipynb

Ce notebook sert à faire une prédiction sur des images de Sentinel-1. Il fonctionne comme Predict-FinalVersion. Il est cependant recommandé de d'abord faire une translation des bandes d'entrée pour harmoniser leurs superpositions.

6 Améliorations et expériences possibles

- Ayant beaucoup de code en commun entre mes différents notebooks, il pourrait être intéressant de sortir ces parties communes (bien souvent des fonctions) et de les mettre dans un fichier python commun. J'avais l'intention de le faire, pendant la dernière semaine, mais quelques petits problèmes avec Hyperion, m'ont retardé.
- Essayer d'avoir un pool de plusieurs modèles, où chaque modèle couvrirait une période de l'année (comme par exemple un mois ou quelques semaines). Cela permettrait d'avoir un suivi de l'évolution du sol tout au long de l'année.
- Améliorer le script de découpage d'image de Sentinel-1. En essayant d'améliorer la qualité du tri en utilisant des Decimals au lieu des floats dans tous les calculs en rapport avec la géolocalisation ou en essayant de se rapprocher de DivideAndSort en décalquant à chaque fois les colonnes et lignes des GCPS.
- Faire un script permettant d'utiliser Sentinel-1 et Sentinel-2 en même temps, cela nécessiterait cependant de savoir les coordonnées géographiques de chaque pixel de chaque image, puis de reconstruire une nouvelle image en associant à chaque pixel de Sentinel-2 le pixel de Sentinel-1 ayant les coordonnées géographiques les plus proches, ce qui est une opération compliquée (surtout pour la géolocalisation des pixels de Sentinel-1).

7 Marche à suivre

7.1 Entraînement

7.1.1 Télécharger des images depuis le site de l'ESA.

7.1.1.1 Manuellement

Vous pouvez télécharger manuellement les images satellites grâce au site de l'ESA.

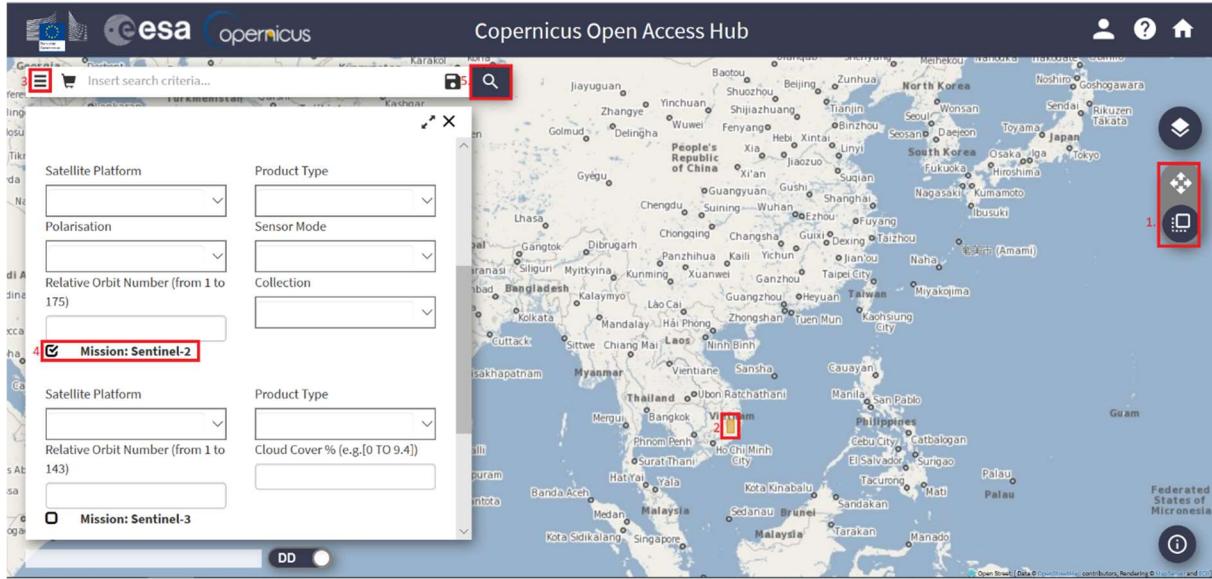


Figure 73 : Site de Copernicus (1)

Sert à changer l'action lors du cliquer/déplacer de la souris. Permettra soit de se déplacer sur la carte, soit de spécifier un rectangle montrant la surface voulue lors de la recherche.

Exemple de rectangle obtenu via la deuxième option du point 1.

Sert à ouvrir le menu des options (qui est affiché en dessous).

Ici, nous avons sélectionné Sentinel-2 à savoir que nous pouvons aussi avoir les images prises par Sentinel-1 (et Sentinel-3, qui n'est pas utilisée dans ce projet). Nous pouvons aussi choisir d'autres critères de recherche, comme une plage de date ou des autres paramètres spécifiques aux satellites.

Effectuer une recherche.

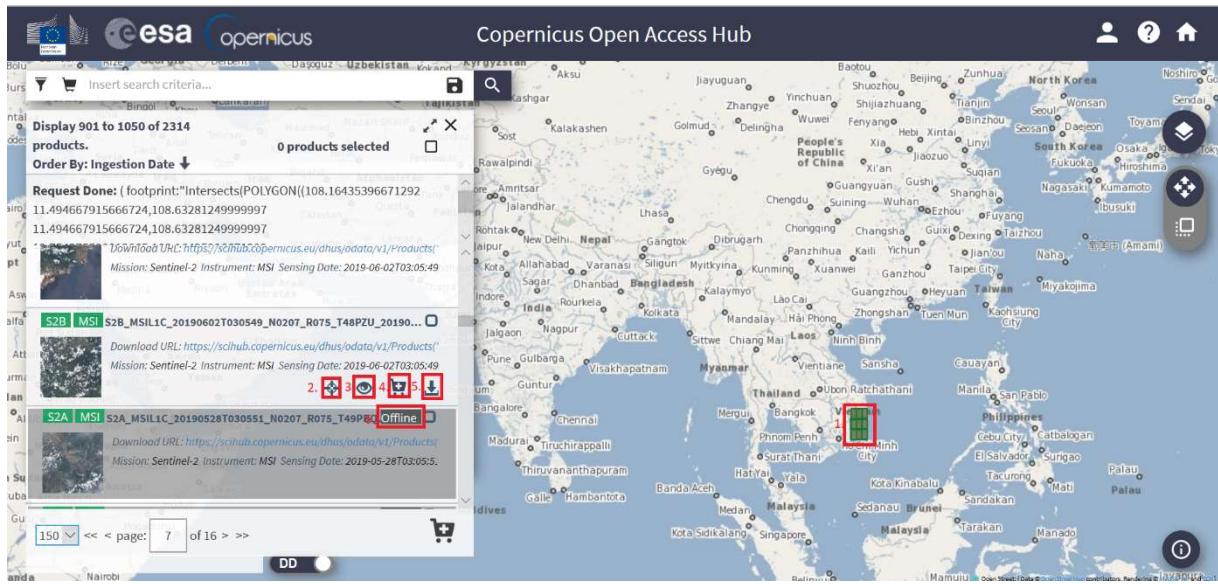


Figure 74 : Site de Copernicus (2)

1. Zone contenant toutes les images⁴ correspondant à la recherche.
 2. Sert à zoomer sur la zone que l'image recouvre.
 3. Sert à afficher le détail de l'image.
 4. Sert à ajouter l'image au panier.
 5. Sert à télécharger directement l'image.
 6. Indique que l'image est archivée. Les images archivées ne sont pas disponibles dans l'immédiat. Si nous souhaitons les télécharger il faut en faire une demande. Pour cela, il suffit simplement de cliquer sur le bouton de téléchargement, il sera automatiquement ajouté au panier. Après quelque temps (dans les 24 heures), l'image devrait être disponible dans notre panier. À noter que les images sont archivées si elles datent de plus d'une année environ et qu'il y a aussi une limite de demande par heure et par utilisateur (si la limite est dépassée mettez l'image dans votre panier et attendez quelque temps avant de faire la demande de restauration) [32].

7.1.1.2 Automatiquement

Vous pouvez aussi utiliser le notebook DownloadImage. Ce script téléchargera par défaut toutes les images de Sentinel-2 prises les 5 derniers jours dans une zone donnée ayant une couverture nuageuse inférieure à 30%.

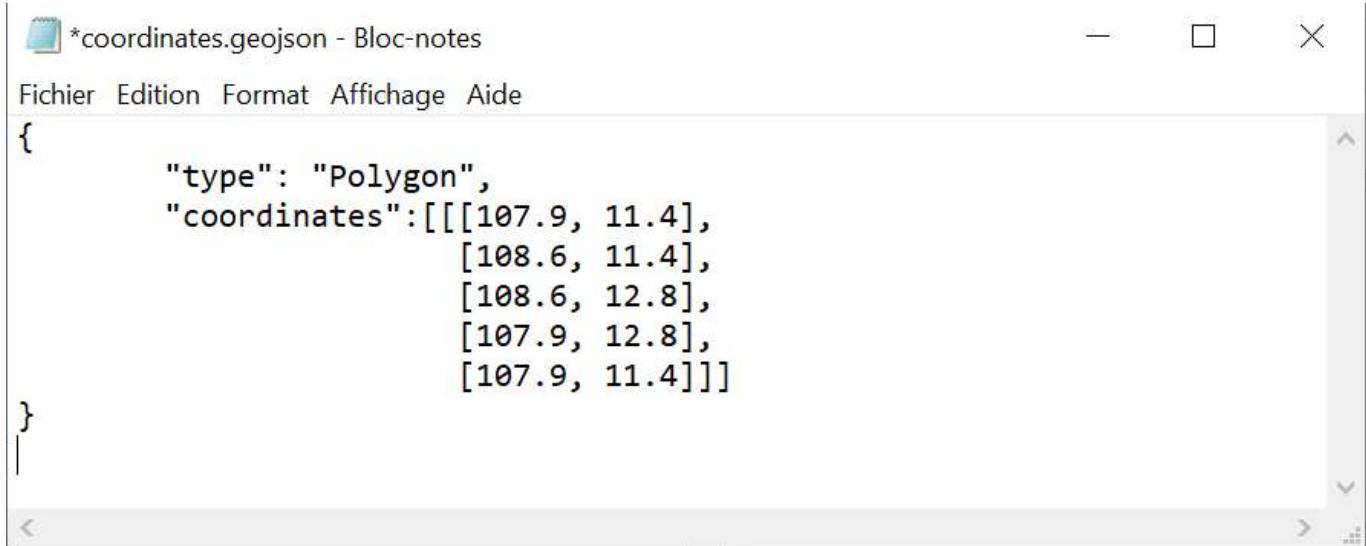
Vous devez cependant avoir un compte sur le site de l'ESA, car le script doit obligatoirement se connecter sur ce site pour pouvoir télécharger les images. Les credentials (nom d'utilisateur et mot de passe) doivent être stockées dans un fichier nommé « credentials » (sans extension). Pour spécifier la région que vous souhaitez recouvrir, il faut le spécifier les coordonnées des points définissants les différents polygones des régions à recouvrir au format geojson dans un fichier nommé « coordinates.geojson ». Les deux fichiers susmentionnés doivent être stockés sur le même niveau que le notebook.

⁴ Dans ce paragraphe, le mot image fait référence à la totalité des mesures effectuées. Il s'agit souvent de plusieurs bandes et d'autres informations.



```
*credentials - Bloc-notes
Fichier Edition Format Affichage Aide
username:password
```

Figure 75 : Exemple de fichier credentials



```
*coordinates.geojson - Bloc-notes
Fichier Edition Format Affichage Aide
{
    "type": "Polygon",
    "coordinates": [[[107.9, 11.4],
                    [108.6, 11.4],
                    [108.6, 12.8],
                    [107.9, 12.8],
                    [107.9, 11.4]]]
}
```

Figure 76 : Exemple de fichier coordinates.geojson

7.1.2 Diviser les images téléchargées en plus petites images

Utiliser un des notebooks dérivés de DivideAndSort pour découper l'image téléchargée en images plus petites. Ce script fait aussi plusieurs fichiers de sortie selon si une position déjà labellisée se trouve ou non dans une image générée ou non. S'il y a plusieurs positions avec des labels différents, l'image sera enregistrée dans le dossier « ambiguous » généré. Pour une image de Sentinel-1 multi saison, je conseille de d'abord harmoniser les positions de départ des différentes images. Cela peut prendre quelques heures.

7.1.3 Sélectionner les données de tests

Utiliser QGIS pour sélectionner les meilleures images partitionnées pour créer les données d'entraînement. Dans l'idéal, il faudrait prendre autant d'image contenant la classe voulue que d'image qui ne la contient pas, mais nous pouvons rééquilibrer les classes avec l'attribut class_weight de Keras.

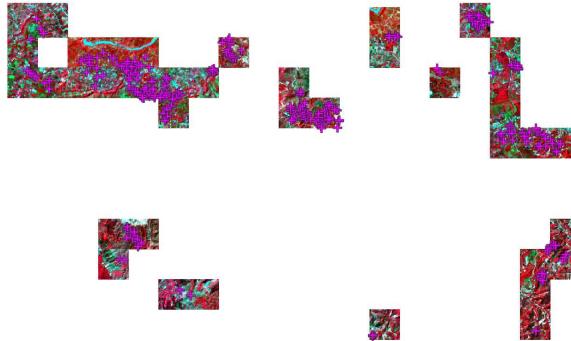


Figure 77 : Faire correspondre des images RG+IR avec les champs de café dans QGIS.

7.1.4 Entrainer le réseau

Entraîner un réseau de neurones avec les données d'entraînements en utilisant l'un des notebooks dérivés de Train. Cela prendra quelques heures, suivant le nombre d'époques définies.

Prendre l'un des modèles en sortie pour faire la prédiction.

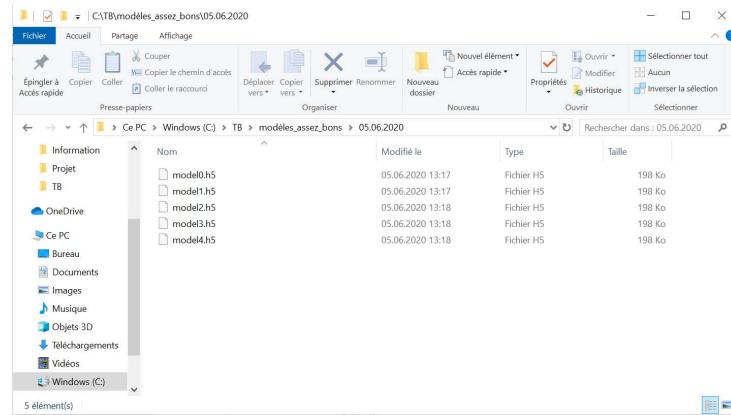


Figure 78 : Modèles générés

7.2 Prédition

Utiliser l'un des notebooks dérivés de Predict en lui spécifiant une image Sentinel et un modèle créé grâce à la marche à suivre ci-dessus.

Attendre que l'image de sortie se crée. Cela prendra quelques heures.

7.3 Schématisation

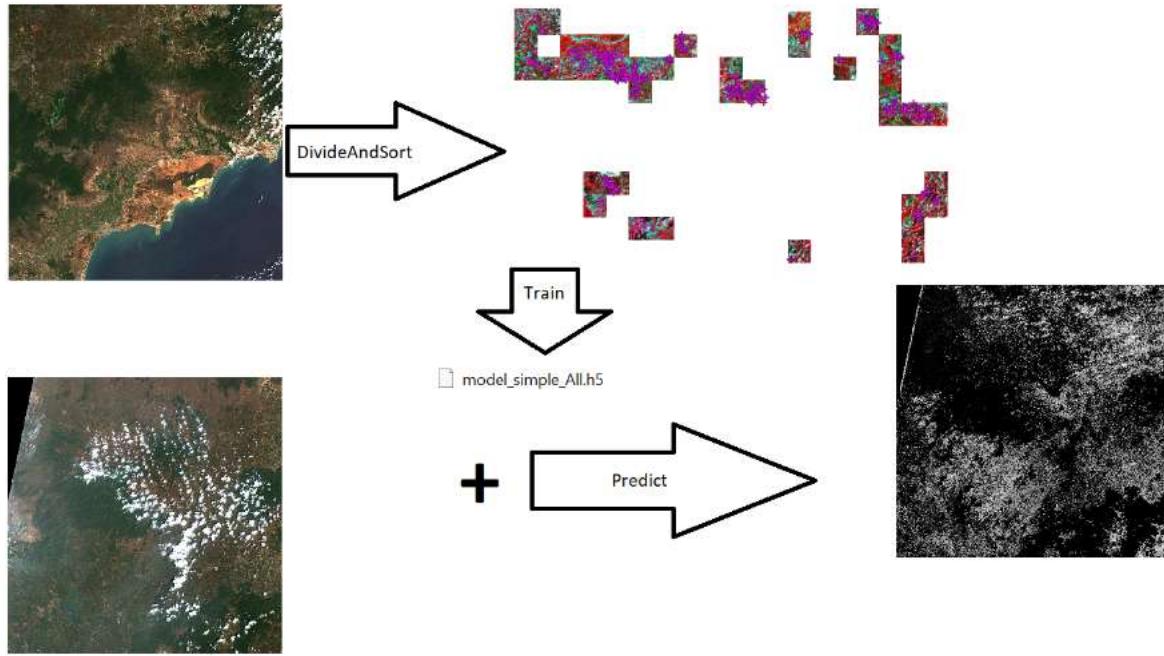


Figure 79 : Schématisation de l'entraînement à la prédiction

8 Remarques et conclusion

Comme l'on peut le constater dans la partie « travail effectué », beaucoup d'expériences ont été faites dans ce travail de Bachelor.

Des expériences faites sur Sentinel-2, on peut retenir que de toutes les bandes, la plus utile est celle utilisant les infra-rouges, même si elle ne semble pas briller avec des données simples (qui ne contenaient que des champs de café, de la ville, de la forêt, de la mer et des champs de poivre, voir point 4.7) on constate que si l'on utilise une grande partie de données à disposition par le CIAT, c'est-à-dire des données d'entraînement plus diversifiées, nous arrivons à de bonnes performances pour toutes les combinaisons utilisant cette bande-ci.

En ce qui concerne Sentinel-1, c'est la combinaison de VV et VH, soit les deux bandes fournies par le type GRD qui nous donne les meilleures combinaisons. De plus les données prises en automne semblent être les meilleures pour trouver les champs de café. Néanmoins, le rappel d'un tel système ne gravite qu'autour de 50% et ne semble atteindre 60% que dans de très rares cas. Étonnamment, l'image prédictive par un tel modèle est plutôt bonne.

Il me semble avoir touché à tous les points du cahier des charges. Cependant je ne peux pas dire que je l'ai complété, car une fois arrivé à la fin de la dernière étape, j'ai toujours dû recommencer au début pour faire de nouvelles expériences et je pense qu'il y a encore beaucoup d'expériences à faire pour arriver à améliorer mes modèles. Malheureusement, le temps me manque pour continuer à en faire d'autres.

J'ai pris beaucoup de plaisir à faire ce travail de Bachelor et suis assez fier des résultats de mes expériences même si certains (comme le rappel de Sentinel-1) sont assez frustrants. J'espère que mes diverses recherches et conclusions serviront au ralentissement de la disparition de la forêt vietnamienne.

Pour finir je tiens à remercier le professeur responsable de mon travail de Bachelor, M. Andres Perez-Uribe, pour son encadrement, sa disponibilité et ses conseils qui m'ont beaucoup aidé dans mon travail. Je remercie aussi M. Louis Reymondin et M. Thibaud Vantanon pour les données fournies et pour les réponses à mes questions fournies lors de notre entretien du 26 juin courant. Finalement je remercie M. Hector Fabio Satizabal Mejia pour m'avoir permis l'accès au serveur Hyperion.

Références

- [1] Wikipedia, Contributeurs, «Central Highlands (Vietnam),» 2020. [En ligne]. Available: [https://en.wikipedia.org/wiki/Central_Highlands_\(Vietnam\)](https://en.wikipedia.org/wiki/Central_Highlands_(Vietnam)). [Accès le 06 07 2020].
- [2] ESA, «Copernicus Open Access Hub,» 2020. [En ligne]. Available: <https://scihub.copernicus.eu/>. [Accès le 17 02 2020].
- [3] ESA, «Sentinel-2,» 2020. [En ligne]. Available: <https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2>. [Accès le 17 02 2020].
- [4] Satellite Imaging Corporation, «Sentinel-2A (10m) Satellite Sensor,» 2017. [En ligne]. Available: <https://www.satimagingcorp.com/satellite-sensors/other-satellite-sensors/sentinel-2a/>. [Accès le 19 06 2020].
- [5] ESA, «Sentinel-1,» 2020. [En ligne]. Available: <https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-1>. [Accès le 17 02 2020].
- [6] ESA, «Land-monitoring,» 2020. [En ligne]. Available: <https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-1-sar/applications/land-monitoring>. [Accès le 19 02 2020].
- [7] ESA, «Acquisition-modes,» 2020. [En ligne]. Available: <https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/acquisition-modes>. [Accès le 19 06 2020].
- [8] QGIS, «QGIS Système d'Information Géographique Libre et Open Source,» 2020. [En ligne]. Available: <https://www.qgis.org/fr/site/>. [Accès le 20 02 2020].
- [9] numpy, «numpy,» 2020. [En ligne]. Available: <https://numpy.org/>. [Accès le 06 07 2020].
- [10] MapBox, «Introduction,» 2020. [En ligne]. Available: <https://rasterio.readthedocs.io/en/latest/intro.html>. [Accès le 06 07 2020].
- [11] scikit-learn, «scikit-learn Machine Learning in Python,» 2020. [En ligne]. Available: <https://scikit-learn.org/stable/>. [Accès le 06 07 2020].
- [12] Keras Special Interest Group, «Keras Simple.Flexible. Powerful.,» 2020. [En ligne]. Available: <https://keras.io/>. [Accès le 06 07 2020].
- [13] Google, «Une plate-forme Open Source de bout en bout dédiée au machine learning,» 2020. [En ligne]. Available: <https://www.tensorflow.org/?hl=fr>. [Accès le 06 07 2020].
- [14] S. Gillies, «affine 2.3.0,» 2020. [En ligne]. Available: <https://pypi.org/project/affine/>. [Accès le 25 03 2020].
- [15] J. Hunter, D. Dale, E. Firing, M. Droettboom and the Matplotlib development team, «Matplotlib: Visualization with Python,» 2020. [En ligne]. Available: <https://matplotlib.org/>. [Accès le 09 07 2020].
- [16] F. Warmerdam, E. Rouault, and others, «GDAL,» 22 04 2020. [En ligne]. Available: <https://gdal.org/>. [Accès le 2020].
- [17] J. Whitaker, «pyproj 2.6.1.post1,» [En ligne]. Available: <https://pypi.org/project/pyproj/>. [Accès le 07 07 2020].
- [18] K. C. Marcel Wille, «SentinelSat stable,» 2016. [En ligne]. Available: <https://sentinelsat.readthedocs.io/en/stable/api.html>. [Accès le 23 06 2020].
- [19] ESA, «Copernicus Open Access Hub,» 2020. [En ligne]. Available: <https://scihub.copernicus.eu/dhus/#/home>. [Accès le 19 02 2020].

- [20] J. Lawhead, «pyshp 2.1.0,» 2020. [En ligne]. Available: <https://pypi.org/project/pyshp/>. [Accès le 07 07 2020].
- [21] SymPy Development Team, «sympy,» 2020. [En ligne]. Available: <https://www.sympy.org/en/index.html>. [Accès le 21 07 2020].
- [22] Project Jupyter, «Jupyter,» 2020. [En ligne]. Available: <https://jupyter.org/>. [Accès le 09 07 2020].
- [23] Aphex34 (contributeur wikipedia), «Fichier:Typical cnn.png,» 2015. [En ligne]. Available: https://fr.wikipedia.org/wiki/Fichier:Typical_cnn.png. [Accès le 14 07 2020].
- [24] J. Brownlee, «Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,» 2017. [En ligne]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>. [Accès le 21 06 2020].
- [25] C. Bourgoin, J. Oszwald, J. Bourgoin, V. Gond, L. Blanc, H. Dessard, T. Van Phan, P. Sist, P. Läderach, L. Reymondin, «Assessing the ecological vulnerability of forest landscape to agricultural frontier expansion in the Central Highlands of Vietnam,» 2020. [En ligne]. Available: <https://www.sciencedirect.com/science/article/pii/S0303243419307202?via%3Dihub>. [Accès le 20 02 2020].
- [26] E. Allibhai, «Building a Convolutional Neural Network (CNN) in Keras,» 2018. [En ligne]. Available: <https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5>. [Accès le 19 03 2020].
- [27] K. C. Ahmed Gad, «Building Convolutional Neural Network using NumPy from Scratch,» 2018. [En ligne]. Available: <https://www.kdnuggets.com/2018/04/building-convolutional-neural-network-numpy-scratch.html>. [Accès le 19 03 2020].
- [28] Google, «Welcome,» 2020. [En ligne]. Available: <https://colab.research.google.com/notebooks/welcome.ipynb?hl=fr>. [Accès le 15 04 2020].
- [29] D. A. Hunt, Karyn Tabor, Jennifer H. Hewson, M. A. Wood, L. Reymondin, K. Koenig, M. Schmitt-Harsh and F. Follett, «Review of Remote Sensing Methods to Map Coffee Production Systems,» 2020. [En ligne]. Available: <https://www.mdpi.com/2072-4292/12/12/2041/htm>. [Accès le 26 06 2020].
- [30] Gouvernement du Canada, «Polarization in radar systems,» 13 11 2014. [En ligne]. Available: <https://www.nrcan.gc.ca/earth-sciences/geomatics/satellite-imagery-air-photos/satellite-imagery-products/educational-resources/9567>. [Accès le 29 07 2020].
- [31] ESA, «Polarimetry,» 2020. [En ligne]. Available: <https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/product-overview/polarimetry>. [Accès le 23 04 2020].
- [32] ESA, «Long Term Archive (LTA) Access,» 14 07 2020. [En ligne]. Available: <https://scihub.copernicus.eu/userguide/LongTermArchive>. [Accès le 26 07 2020].

Authentification

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

Ropraz, le 6 août 2020

Julien Rod

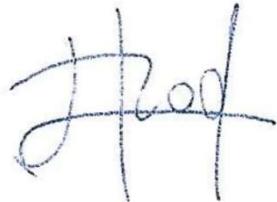


Table des abréviations utilisées

CIAT

Centre de Recherche en Agriculture Tropicale, 3, 9, 12, 13, 14, 16, 17, 20, 21, 23, 24, 25, 47, 71, 80, 81, 83, 84

ESA

European Space Agency, 3, 9, 10, 14, 67, 68

GRD

Ground Range Detected, 45, 54, 71, 78, 82

KCL

King's College London, 9

RG+IR

Red Green + Infra Red (couleurs composants l'image), 17, 20, 21, 69, 84

RGB

Red Green Blue (couleurs composants l'image), 10, 17, 20, 21, 54, 82, 83, 84

Table des illustrations

Figure 1: Bandes de sentinel-2 [4]	10
Figure 2: Modes de Sentinel-1 [7]	11
Figure 3 : À gauche, la liste de toutes les données fournies par le CIAT. À droite, la même liste en ne gardant que les champs de café.....	12
Figure 4: Réseau de neurones convolutif en image [23] (aucune modification n'a été apportée à l'image)	15
Figure 5 : De gauche à droite : Champs de café, lac, forêt, ville et image contenant des champs inconnus, le haut représente des images en RGB et le bas des images en RG+IR.....	17
Figure 6 : Passer une image de Rasterio à Tensorflow + normalisation des données	18
Figure 7 : Comment lire les matrices de confusion pour deux classes : « café » et « pas café »	18
Figure 8 : Visibilités, graphiques d'apprentissage et matrices de confusion d'une même validation croisée (étape 4.9).....	19
Figure 9 : Image Sentinel-2 et sa prédiction	20
Figure 10 : Graphique d'erreur et matrice de confusion en entraînant sur une seule zone et en testant sur les autres, avec des images de 90 sur 90 pixels.....	22
Figure 11 : Graphique d'erreur et matrice de confusion en entraînant sur une seule zone et en testant sur les autres, avec des images de 15 sur 15 pixels	22
Figure 12: Poivre VS Café (dans cet ordre dans la matrice de confusion).....	23
Figure 13 : Moyenne des résultats de poivre VS café	23
Figure 14: de haut en bas : autre, poivre, café, les deux.....	23
Figure 15 : Autre VS poivre VS café (dans cet ordre)	24
Figure 16: Image générée en prédiction multiple (total de l'image à gauche, zoom à droite avec les coordonnées du CIAT activées)	24
Figure 17: Image générée en prédiction multiple avec le nouveau réseau (total de l'image à gauche, zoom à droite avec les coordonnées du CIAT activées).....	25
Figure 18 : À gauche zone du nord-ouest, à droite une zone contenant des champs de café et de poivre	25
Figure 19 : Même région prise (de gauche à droite) en automne, en hiver et au printemps.....	26
Figure 20 : Moyenne des résultats pour un réseau avec toutes les bandes des trois saisons.....	26
Figure 21 : Entraînement sur 9 bandes, à gauche avec les deux classes plus ou moins équilibrées, à droite sur l'intégralité des images labellisées en utilisant le paramètre class_weight de Keras.....	30
Figure 22 : Entraînement sur 6 bandes, à gauche avec les deux classes plus ou moins équilibrées, à droite sur l'intégralité des images labellisées en utilisant le paramètre class_weight de Keras.....	31

Figure 23 : Entraînement sur 3 bandes infra-rouge, à gauche avec les deux classes plus ou moins équilibrées, à droite sur l'intégralité des images labellisées en utilisant le paramètre class_weight de Keras	32
Figure 24 : Prédiction multi-saisons avec de gauche à droite 6, 9 et 12 bandes	32
Figure 25 : Zoom sur la prédiction à 9 bandes avec les labels activés	33
Figure 26 : Entraîner sur une zone tester sur les autres avec toutes les bandes (à lire en colonne).....	34
Figure 27 : Entraîner sur une zone tester sur les autres avec les 9 bandes recommandées (à lire en colonne)	35
Figure 28 : Entraîner sur deux zones et tester sur les autres (à lire en colonne).....	36
Figure 29 : Graphique d'erreur et matrice de confusion d'un entraînement multi classes à 12 bandes.....	37
Figure 30 : Graphique d'erreur et matrice de confusion d'un entraînement multi classes à 9 bandes.....	38
Figure 31 : Graphique d'erreur et matrice de confusion d'un entraînement multi classes à 6 bandes.....	39
Figure 32 : Prédiction multi-saisons multi classes avec de gauche à droite 6, 9 et 12 bandes	39
Figure 33 : Zoom sur la prédiction multi classes à 6 bandes avec les labels activés	40
Figure 34 : Entraînement binaire poivre vs café.....	40
Figure 35 : Tester un réseau entraîné sur 2A avec des images contenant du 1C, ça ne marche pas.....	40
Figure 36 : Tester une année plus tôt de gauche à droite avec 6, 9 et toutes les bandes	41
Figure 37 : Prédiction sur toutes les bandes à gauche en 2018-2019 et à droite 2019-2020.....	41
Figure 38 : À gauche une image de type 2A, à droite une image de type 1C.....	44
Figure 39 : Résultat d'une itération de Sentinel-1 avec toutes les saisons	45
Figure 40 : : Image générée en prédiction de Sentinel-1 sur toutes les saisons avec les bandes VV et VH (total de l'image à gauche, zoom à droite avec les coordonnées du CIAT activées)	47
Figure 41 : Importer des images avec SentinelSat	48
Figure 42 : DivideAndSort importer les bandes.....	49
Figure 43 : DivideAndSort, shapefiles et autres paramètres.....	49
Figure 44 : Découper une bande en bandes plus petites	49
Figure 45 : Calcul des coordonnées de début et de la fin d'une image découpée de Sentinel-2	50
Figure 46 : Savoir quels labels se trouvent dans une image de Sentinel-2	50
Figure 47 : Conversion des systèmes de coordonnées géographiques (réutilisation d'un labo de VTK).....	50
Figure 48 : Classer automatiquement les images Sentinel-2	51
Figure 49 : Dossier généré par DivideAndSort.....	51
Figure 50 : Fonction donnant les vecteurs (approximatifs) pour passer d'un GroundControlPoint à ses voisins	52
Figure 51 : Créer des coordonnées GCPS communes à toutes les bandes de sorties.....	53
Figure 52 : Trouver les de combien de cases il faut décaler la matrice d'une bande donnée	53
Figure 53 : Faire une translation d'une bande de Sentinel-1	54
Figure 54 : DivideAndSortS1, importer les bandes.....	54
Figure 55 : Différence entre le placement des cellules entre Sentinel-2 (gauche) et Sentinel-1 (droite).....	55
Figure 56 : Découper une image de Sentinel-1 en images plus petites	55
Figure 57 : Créer la bande bleue.....	56
Figure 58 : Fonctions servant à l'interpolation dans un quadrilatère quelconque	56
Figure 59 : Définition des hyperparamètres et des dossiers servant de données d'entraînement.....	56
Figure 60 : Fonction permettant d'avoir les coins d'une image	57
Figure 61 : Fonction de labellisation des images de Sentinel-1	57
Figure 62 : Crédit d'une validation croisée avec Sklearn	58
Figure 63 : Définir le poids des classes	58
Figure 64 : Tourner les images.....	59
Figure 65 : Formater les données de type raster pour qu'elles soient utilisables par Tensorflow	59
Figure 66 : Réseau de neurones de base	60
Figure 67 : Réseau de neurones « entraîner sur une zone, tester sur les autres »	61
Figure 68 : Réseau de neurones « entraîner sur deux zones, tester sur les autres »	62
Figure 69 : Crédit des vecteurs one-hot	63

Figure 70 : Comparer avec une année antérieure.....	63
Figure 71 : Création du fichier de sortie	64
Figure 72 : Prédiction avec n'importe quel nombre de bandes	65
Figure 73 : Site de Copernicus (1).....	67
Figure 74 : Site de Copernicus (2).....	68
Figure 75 : Exemple de fichier credentials.....	69
Figure 76 : Exemple de fichier coordinates.geojson.....	69
Figure 77 : Faire correspondre des images RG+IR avec les champs de café dans QGIS.....	69
Figure 78 : Modèles générés	70
Figure 79 : Schématisation de l'entraînement à la prédiction	70

Annexes

Notebooks

- CoffeeVSPepperFallWinterSpring.ipynb : Entraine un modèle multi classes sur trois saisons.
- CompareOneYearAgo.ipynb : Sert à comparer deux sets sur des données de mêmes coordonnées géographiques mais séparé dans le temps.
- DivideAndSort.ipynb : Découpe une image de Sentinel-2 en images plus petites et les classes selon un shapefile spécifié.
- DivideAndSort-FallWinterSpring.ipynb : Version de DivideAndSort.ipynb sur trois saisons.
- DivideAndSort-Final.ipynb : Version finale de DivideAndSort.ipynb.
- DivideAndSortS1.ipynb : Découpe une image de Sentinel-1 en images plus petites et les classes approximativement selon un shapefile spécifié.
- DivideAndSortS1-AllSeasons.ipynb : Version de DivideAndSortS1.ipynb sur quatre saisons.
- DownloadImages.ipynb : Sert à télécharger les nouvelles images de Copernicus sans passer par le site.
- Predict.ipynb : Fait une prédiction sur un modèle créé par Train.ipynb et sur une image de Sentinel-2.
- Predict-FinalVersion.ipynb : Version finale de Predict.ipynb.
- Predict-Multi-FinalVersion.ipynb : Fait une prédiction multi classes sur un modèle créé par CoffeeVSPepperFallWinterSpring.ipynb et sur une image de Sentinel-2.
- PredictS1-FinalVersion.ipynb : Fait une prédiction sur un modèle créé par Train-S1-AllSeasons.ipynb et sur une image de Sentinel-1.
- Train.ipynb : Sert à entraîner un réseau de neurones sur une image de Sentinel-2 découpée au préalable par DivideAndSort.ipynb.
- Train_one_test_others-FallWinterSpring.ipynb : Entraîne un réseau sur une zone puis le teste sur toutes les autres.
- Train_two_test_others-FallWinterSpring.ipynb : Entraîne un réseau sur deux zones puis le teste sur toutes les autres.
- Train-FallWinterSpring.ipynb : Version de Train.ipynb sur trois saisons.

- Train-S1-AllSeasons.ipynb : Sert à entraîner un réseau de neurones sur une image de Sentinel-1 découpée au préalable par DivideAndSortS1-AllSeasons.ipynb.
- TranslateS1Bands.ipynb : Sert à coordonner la position de départ d'images prises par Sentinel-1.

Modèles

- model_multi_All.h5 : Modèle multi classes utilisant toutes les bandes de trois images Sentinel-2 2A prisent en automne, en hiver et au printemps.
- model_multi_Recommandation6.h5 : Modèle multi classes utilisant les bandes vertes et infra-rouges de trois images Sentinel-2 2A prisent en automne, en hiver et au printemps.
- model_multi_Recommandation9.h5 : Modèle multi classes utilisant les bandes bleue, rouge et infra-rouge d'une image prise en automne, les bandes bleue, verte et infra-rouge d'une image prise en hiver et les bandes bleue, rouge et infra-rouge d'une image prise au printemps (toutes les images étant de Sentinel-2 2A).
- model_simple_All.h5 : Modèle binaire utilisant toutes les bandes de trois images Sentinel-2 2A prisent en automne, en hiver et au printemps.
- model_simple_Recommandation6.h5 : Modèle binaire utilisant les bandes vertes et infra-rouges de trois images Sentinel-2 2A prisent en automne, en hiver et au printemps.
- model_simple_Recommandation9.h5 : Modèle binaire utilisant les bandes bleue, rouge et infra-rouge d'une image prise en automne, les bandes bleue, verte et infra-rouge d'une image prise en hiver et les bandes bleue, rouge et infra-rouge d'une image prise au printemps (toutes les images étant de Sentinel-2 2A).
- model1c6Bands.h5 : Modèle binaire utilisant les bandes vertes et infra-rouges de trois images Sentinel-2 1C prisent en automne, en hiver et au printemps.
- model1c9Bands.h5 : Modèle binaire utilisant les bandes bleue, rouge et infra-rouge d'une image prise en automne, les bandes bleue, verte et infra-rouge d'une image prise en hiver et les bandes bleue, rouge et infra-rouge d'une image prise au printemps (toutes les images étant de Sentinel-2 1C).
- model1cAll.h5 : Modèle binaire utilisant toutes les bandes de trois images Sentinel-2 1C prisent en automne, en hiver et au printemps.
- models1.h5 : Modèle binaire utilisant les bandes VV et VH de quatre images Sentinel-1 GRD prisent en automne, en hiver, au printemps et en été.

Images générées

- multi_6_prédiction.tif : Image générée par une prédiction multiple de Sentinel-2 avec 6 bandes (vert et infra-rouge seulement).
- multi_9_prédiction.tif : Image générée par une prédiction multiple de Sentinel-2 avec 9 bandes (bleu, rouge et infra-rouge pour l'automne et le printemps et bleu, vert et infra-rouge pour l'hiver).
- multi_All.tif : Image générée par une prédiction multiple de Sentinel-2 avec les 12 bandes.
- prédictionS1AllSeasons.tif : Image générée par une prédiction binaire de Sentinel-1 avec les bandes VV et VH de chaque saisons.
- simple_6_prédiction.tif : Image générée par une prédiction binaire de Sentinel-2 avec 6 bandes (vert et infra-rouge seulement).
- simple_9_prédiction.tif : Image générée par une prédiction multiple de Sentinel-2 avec 9 bandes (bleu, rouge et infra-rouge pour l'automne et le printemps et bleu, vert et infra-rouge pour l'hiver).
- simple_All.tif : Image générée par une prédiction binaire de Sentinel-2 avec les 12 bandes.

Autres

- coordinates.geojson : Exemple d'un fichier coordinates.geojson utilisé par DownloadImages.ipynb.
- credentials : Exemple d'un fichier credentials utilisé par DownloadImages.ipynb (ces credentials-ci ne devraient pas exister).
- Model_julien.rod1.doc : Affiche de mon TB au format doc (le « Bachelor 2020 » ayant du mal à s'afficher correctement lors de la conversion au format pdf).

Journal de travail

17.02.2020 : Lire la documentation sur les satellites que l'on utilisera.

19.02.2020 : Continué à lire la documentation (et les documents que le secrétariat nous a envoyés) + voire les images prises par les satellites. J'ai aussi créé un compte pour avoir accès aux images des satellites. Pour l'instant, je peux voir les images quand je suis sur le site, mais ne sais pas où se trouve l'image quand je télécharge les dossiers de S2...

20.02.2020 : Téléchargé et commencé à lire la documentation de QGIS (<https://docs.qgis.org/3.4/pdf/fr/QGIS-3.4-PyQGISDeveloperCookbook-fr.pdf>). J'ai aussi eu une réunion avec monsieur Uribe pour avoir quelques explications sur le travail.

21.02.2020 : Lu la documentation donnée par M. Perez-Uribe. Il semblerait que les images apportées par Sentinel-2 pendant mi-juin à début octobre soient presque totalement inutilisables (saison des pluies oblige).

26.02.2020 : Continué la lecture.

27.02.2020 : Continué la lecture de la documentation de QGIS.

28.02.2020 : Pareil qu'hier.

02.03.2020 : Pareil que vendredi.

04.03.2020 : J'ai enfin fini de lire la documentation de QGIS pour développeur. J'ai aussi regardé plus en détail les images sur QGIS. Pour Sentinel-1, nous pouvons utiliser les images vv et vh se trouvant dans le dossier « measurement ». Pour Sentinel-2, nous pouvons utiliser les images AOT, TCI et VWP, du dossier « IMG_DATA\R10m ». Pour les BX, on remarque que les images sont presque totalement noires (pas utile pour un être humain en tout cas).

05.03.2020 : Je n'arrive pas à me connecter ni sur le site d'anaconda, ni sur le site de QGIS (il semblerait que le réseau de l'école ait un problème). Du coup, j'ai lu le guide d'utilisation de Sentinel-1.

09.03.2020 : J'ai lu la documentation sur les réseaux de neurones (je ne sais pas si le temps passé dessus doit être mis en tant que TB ou MLG, mais dans le doute...). J'ai aussi fait un petit programme python qui change des fichiers de places (cela sera sans doute utile à l'avenir lors du traitement).

11.03.2020 : J'ai regardé les points passés par le CIAT, vu que nous n'avons que les points, je me suis servi de QGIS et google map pour trouver à quoi elles correspondaient. Les coordonnées sont à peu près une longitude de 107.1 à 108.6 et une latitude entre 11.3 à 14.3, ce qui correspond à peu près à la partie frontalière au Cambodge, couvrant aussi la partie étudiée dans leur rapport. On distingue facilement à l'œil nu la forêt du reste, mais cela est plus compliqué pour reconnaître les champs de cafés du reste. J'ai essayé de suivre certains tutos sur python et Sentinel-2, mais je me suis confronté à certaines erreurs de configurations.

12.03.2020 : J'ai fait un tutoriel sur python et Sentinel-2 (j'ai résolu mes problèmes de configuration en utilisant conda install au lieu de pip install). Je me suis aussi amusé un peu avec les images. Je suis d'ailleurs tombé sur ce [lien](#) intéressant pour connaître les différentes combinaisons de bandes, les plus intéressantes étant sans doute les couleurs infrarouges (B8, B4 et B3), l'agriculture (B11, B8 et B2) et l'index de végétation ((B8 -B4)/(B8+B4)). Contrairement à ce que je pensais quelques jours plus tôt, il semblerait que les BX soient très utiles dans le traitement des images. Il sera peut-être utile d'utiliser B11, même si sa précision est moins bonne que les autres BXs cités plus haut dans ce paragraphe (5490^2 données contre 10980^2 pour les autres (soit 4 fois moins)).

13.03.2020 : J'ai regardé et lu les articles et vidéos proposés par le cours de MLG.

18.03.2020 : J'ai fait des recherches sur les réseaux de neurones convolutifs et j'ai essayé de faire un tutoriel sur ceux-ci.

19.03.2020 : J'ai fait d'autres tutoriels sur les réseaux de neurones convolutifs (cette fois-ci sans soucis de compilation).

20.03.2020 : J'ai fini encore quelques tutoriels. J'ai envoyé un mail à nos correspondants du CIAT pour savoir quels étaient les images utilisées pour trouver les points qu'ils nous avaient envoyés. J'ai commencé à regarder comment faire pour faire matcher les images de Sentinel-1 et Sentinel-2.

25.03.2020 : J'ai fait un code qui split une grande image de 10980^2 en images plus petites (200^2). J'ai aussi fait en sorte que les images s'affichent aux bonnes coordonnées dans QGIS (même si cette partie me paraît un peu fait salement, il doit sans doute y avoir une fonction plus simple permettant de le faire, mais je ne l'ai pour l'instant pas trouvée). Cela prend en revanche pas mal de temps (prend bien 1 minute pour faire 9 carrés (sur plus de 3000 par image))...

26.03.2020 : J'ai commencé à faire matcher les images ainsi générées avec les champs de café rapportés par le CIAT.

01.04.2020 : J'ai continué à faire matcher mes images (cette fois-ci en 90^2) avec les champs de café rapportés par le CIAT.

02.04.2020 : Pareil.

08.04.2020 : Fini de faire matcher mes images avec les champs de café. Puis j'ai commencé à essayer de trouver là où il n'y avait pas de champs de café (ville, forêt, océan, etc...).

09.04.2020 : Continué à chercher là où il n'y avait pas de café.

11.04.2020 : Fini de chercher là où il n'y avait pas de café, ce qui me donne un dataset avec 50% de champs de café et 50% d'autres choses. J'ai commencé à faire la première version de mon réseau de neurones convolutif.

13.04.2020 : Pas très convaincant, après plus de 24 heures d'entraînement, mon réseau a un résultat de prédiction équivalant à une pièce de monnaie.

15.04.2020 : J'ai commencé à essayer Google Colab, il prend beaucoup de temps (pas plus que ma machine) et vu que j'aurai besoin de le faire run plus que 12 heures, Google Colab ne semble pas être une option. Après quelques longs essais, j'ai remarqué que si on ne prenait que la première couche (convolution 2D) et la dernière couche (sortie) j'obtiens un résultat d'environ 75~80% pour chaque k-fold. Cependant, si j'ai plusieurs layers de type dense (comme le layer de sortie), il semblerait que mon réseau reste bloqué entre 0.48 et 0.51 lors de l'entraînement pour une raison que j'ignore (et ce même si je ne mets qu'un seul neurone dans chaque layer) ...

16.04.2020 : Pour pouvoir runner mon réseau plus longtemps, je vais utiliser l'un des serveurs de l'école. Il est plus rapide que Google Colab et n'a pas de limite de temps.

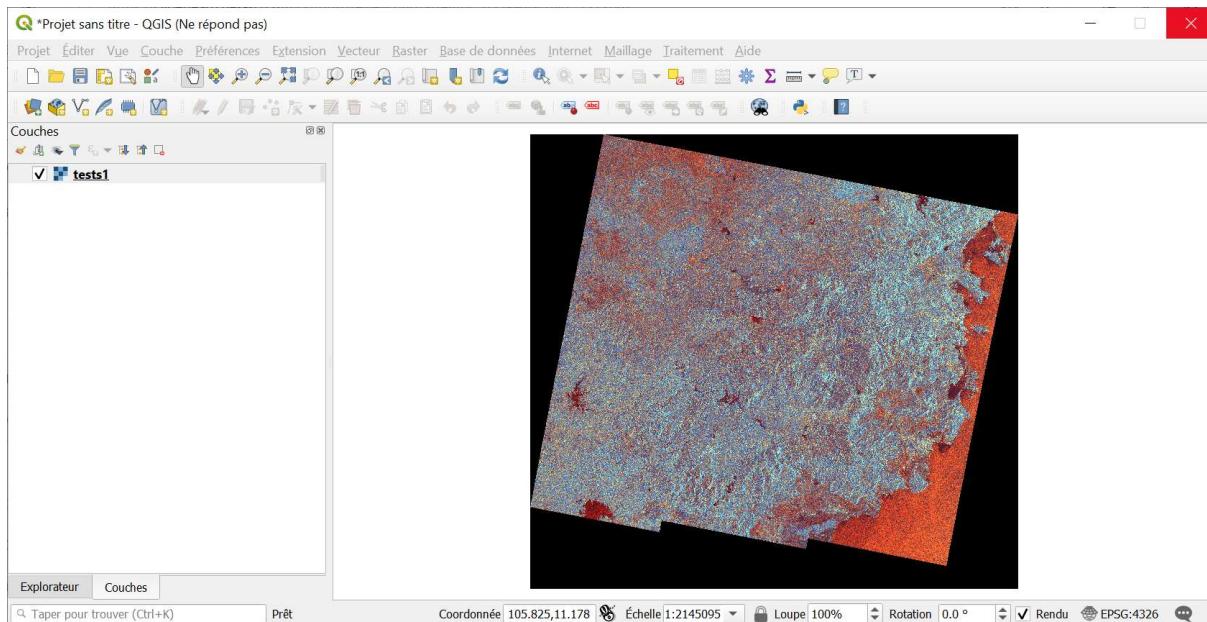
19.04.2020 : Faisant toujours des tests un peu à tâtons pour trouver la meilleure solution pour mon réseau de neurones, je commence à essayer de faire des trucs avec les images de Sentinel-1, cependant, elles ne semblent pas être géolocalisées pareil ce qui est un problème pour pouvoir les placer sur un plan (et surtout pour faire correspondre les positions du café). De plus leurs données sont en nombre complexe ce qui pose un problème pour l'affichage via rasterio.

20.04.2020 : J'ai continué de faire des choses avec les images de Sentinel-1. J'ai remarqué que le nombre de lignes et de colonnes quand j'ouvre un fichier avec rasterio n'est pas le même que ceux indiqués par QGIS lorsque j'ouvre ses propriétés. Quand j'enregistre un fichier son nombre de colonnes et de lignes est celui indiqué par rasterio. Il semble donc qu'il me manque de l'information. J'ai réussi à avoir un peu

les coordonnées géographiques de tels fichiers. Mais cela ne semble pas marcher pour le moment. J'obtiens le crs et des gcps mais le fichier ne s'affiche pas et il n'a pas d'emprise dans QGIS.

22.04.2020 : J'ai essayé d'ouvrir mes fichiers tiffs avec un autre module que rasterio (GDAL), j'ai toujours le même problème de dimension. J'ai essayé ensuite de passer mes coordonnées gcps à l'images enregistrée avec les bandes de GRD, cela marche et pour je ne sais quelle raison, les dimensions marquées dans QGIS quand j'ouvre les propriétés du fichier sont les mêmes que les autres fichiers dans QGIS (il faut croire que QGIS compte les coordonnées gcps comme faisant partie de la bande). Je suis arrivé à afficher l'image dans QGIS, une seule fois (le reste du temps lag et n'affiche rien...).

23.04.2020 : J'ai continué de faire mon image avec Sentinel-1. J'ai lu [sur le net](#) qu'il était possible de générer une image RGB en faisant $|VV|/|VH|$. Après m'être battu contre des problèmes de mémoire, je suis enfin parvenu à faire sortir cette image. J'ai même réussi à l'afficher un moment dans QGIS (j'ai joint une capture d'écran pour prouver mes dires).



30.05.2020 : J'ai repris le TB après un mois d'absence (protection civile + rattrapage des autres cours). J'ai fait une fonction pour avoir les GroundControlPoints interpolés à certains pixels de nos images.

31.05.2020 : J'ai testé la fonction. Il me semble qu'elle marche mais le résultat attendu n'est pas celui obtenu. J'ai exploré d'autres possibilités, comme créer un vecteur à partir des coordonnées fournies (comme pour sentinel-2), mais cela ne semble pas marcher (les pixels deviennent trop gros).

03.06.2020 : J'ai continué à essayer de faire fonctionner la fonction pour découper les images de S1 et en parallèle ai commencé une fonction pour avoir une carte des prédictions.

04.06.2020 : Me rendant compte que Tensorflow prenait les pixels plutôt que des couches (je ne sais pas comment l'expliquer, autrement qu'il prend des images comme des tableaux de 90 90 3 alors que les images rasterio sont de type 3 90 90), j'ai écrit une fonction pour traduire mes images dans le bon format. J'ai aussi continué mes recherches sur les images de Sentinel-1 et mon projet de prédiction. J'ai aussi eu un call avec M. Perez-Uribe. Après avoir normalisé les valeurs d'entrées de mon réseau de neurones convolutif pour Sentinel-2 entre 0 et 1 (en divisant les valeurs par max uint16) j'arrive à des modèles de prédiction supérieure à 90%. Je me demande comment il pouvait arriver quand même à 75% avant...

10.06.2020 : J'ai testé ma fonction de prédiction et une méthode pour découper une image de sentinel-1 en image plus petites, cette dernière semble fonctionner, mais pour je ne sais quelle raison, il semble y avoir des

trous entre les images créées. Pour ce qui est de ma fonction de prédiction, j'ai remarqué sur son output qu'elle arrive à faire la différence entre forêt/ville/eau et le reste (la partie brune sur nos images satellites). Cela est déjà un bon début. Il faudrait que je puisse améliorer mes données d'apprentissage pour avoir un meilleur résultat, mais cela impliquerait d'être sûre qu'il n'y a pas de champs de café là sur cette donnée-ci (ce que je ne peux faire pour l'instant).

11.06.2020 : J'ai finalement réussi à faire des images de Sentinel-1 en plus petite (et en RGB), j'ai donc lancé le programme pour qu'il crée un set de données que je pourrai ensuite utiliser pour créer un set d'entraînement. En revanche, au vu des dimensions des images, nous ne pouvons pas les diviser en images de 90 / 90 sans perte de pixel, sa hauteur étant de 22738 on ne peut la diviser que par 2 et 11369... Quant à sa profondeur de 25118, elle n'est divisible que par 2, 19 et 661... Avoir des images de 11369 / 661 ne serait pas utilisable, c'est pour ça que j'utilise des images de 90 / 90 qui ne nous font perdre que 58 pixels de hauteur et 8 de profondeur. J'ai aussi commencé à rédiger le rapport intermédiaire.

17.06.2020 : La date de remise arrivant à grands pas, j'ai continué à rédiger mon rapport intermédiaire.

18.06.2020 : Comme hier.

19.06.2020 : Pareil et je suis passé à la HEIG pour apporter ma clause de confidentialité.

20.06.2020 : Comme hier mais sans la HEIG.

22.06.2020 : J'ai fini mon rapport intermédiaire et l'ai envoyé.

23.06.2020 : J'ai implémenté le téléchargement des images satellite. J'ai passé tout mon matin à essayer de faire des requêtes CURL. Pour télécharger un fichier, tout va bien si l'on connaît l'UUID. Cependant, pour connaître l'UUID des images nous intéressantes, cela est beaucoup plus complexe. Spécifier la date et le satellite est assez facile, par contre, comment spécifier des coordonnées géographiques voulues, n'est pas spécifié dans la documentation (<https://scihub.copernicus.eu/userguide/ODataAPI...>). En cherchant sur le net comment faire, je suis tombé sur le module python SentinelSat (<https://sentinelsat.readthedocs.io/en/stable/api.html>). Après quelques essais, nous pouvons maintenant télécharger toutes les images d'une zone donnée prises pendant la semaine par l'une des missions qui nous intéressent. À noter que nous pouvons aussi spécifier la couverture nuageuse voulue. Ce qui nous sera fort utile pour ne pas inonder notre stockage d'images blanches, lors de la saison des pluies.

24.06.2020 : J'ai sélectionné les images pour faire un set de données avec les images de Sentinel-1. J'ai remarqué que les images de Sentinel-1 étaient un peu décalées en comparaison avec les images de Sentinel-2 (de quelques pixels sur la gauche). Du coup, je ne sais pas si mes sets de données sont vraiment justes... En parallèle j'ai lancé une prédiction sur les données de Sentinel-2 en RGB.

25.06.2020 : J'ai essayé de faire run mon réseau de neurones avec les données de Sentinel-1. Ce n'est pas fameux... Au pire le système n'apprend pas (littéralement), au mieux, il a une précision d'environ 70%.

26.06.2020 : Prise de notes de ma conversation avec le CIAT : Ils utilisent de petites images de 13 x 13 10m par pixel. Les données géographiques pas forcément précises à 100%. Sentinel-1 peut être chaotique si la couverture d'arbre n'est pas homogène. Sentinel-2 a plus d'informations sur le terrain. Sentinel-1 tout seul n'a pas beaucoup d'informations. Sentinel-1 a beaucoup de bruit, en prendre plusieurs dans le temps et les lisser (médiane). Un champ de café reste très stable dans le temps, contrairement à d'autres cultures. Challenge, zones données à plein de champs de café qui se suivent. Là où ça devient le plus difficile est quand il y a des arbres faisant de l'ombre et d'autres cultures en même temps. Prendre des points assez séparés. Les points sont biaisés (sans doute). Une validation propre en ordre prend beaucoup de temps. Lier cela à déforestation ou non avec une année butoir précise ; avant ou après 2008. Dire depuis quand il y a du café sur ce point et si ça a remplacé des forêts. Trouver les

réseaux de neurones les plus résistants (si on s'éloigne des points choisis pour l'entraînement). Trouver les meilleures variables. Utiliser des images dans le temps, d'autres bandes, créer des cartes (avec plusieurs années), entraîner les réseaux avec que des points d'une certaine région (changement de type de sol, qui fait que c'est plus dur de reconnaître le café). En fleurs, le café est blanc (ce qui le distingue du poivre). Faire café poivre. Trouver le plus petit réseau de neurones possibles au niveau des bandes ou des couches, sans trop péjorer la prédiction. Regarder les bandes qui sont les moins utiles et les enlever. Prochaines étapes : Tester les petites images, entraîner sur une zone et appliquer sur une autre, voir quelles sont les bandes qui trigger le réseau de neurones pour prendre une décision, café poivre, éventuellement, si le temps le permet, dans le temps (le café doit être constant). Après cela j'ai modifié mon script pour diviser les images de Sentinel-2 j'ai fait en sorte de les partitionner en image de 15 sur 15 pixels pour être plus proche des tests du CIAT et pouvoir en même temps utiliser l'entièreté de l'image de Sentinel-2 sans perte. Contrairement à avant, je n'utilise plus d'un côté des images en RGB et de l'autre en RG+IR de faire des images en utilisant ces 4 couches car cela me permet d'économiser de la place. J'ai aussi mis ces images dans des sous-dossiers selon la ligne à laquelle elles appartiennent pour éviter que QGIS ne réponde pas lorsque l'on ouvre le dossier (732^2 images en tout). Après quelques résolutions de bugs (surtout lié à la création de dossiers), le programme semble s'exécuter correctement.

27.06.2020 : J'ai commencé à sélectionner les données contenant du café dans l'image de gauche (avec celles qui contiennent aussi du poivre).

28.06.2020 : J'ai fini le travail commencé la veille.

29.06.2020 : J'ai sélectionné les données contenant du poivre dans l'image de gauche.

30.06.2020 : J'ai sélectionné les données contenant du poivre et du café dans les images de droite, du haut et centrale.

01.07.2020 : J'ai commencé à sélectionner les données contenant du café dans l'image du bas.

02.07.2020 : J'ai fini de sélectionner les données contenant du café dans l'image du bas (avec la seule donnée contenant du poivre dans cette zone-ci).

03.07.2020 : J'ai sélectionné des données contenant autre chose dans l'image du bas (mer, forêt, ville, caoutchouc, etc...).

04.07.2020 : J'ai adapté mon réseau de neurones avec les nouvelles données. Je l'ai testé avec 5000 itérations (sur les 4 bandes). J'arrive à peu près à une visibilité oscillante entre 82 et 86%, mais il semble que le réseau peut encore apprendre (n'était pas totalement à 0.9 lors de l'apprentissage). J'ai aussi essayé de tester mon ancien réseau de neurones en entraînant sur les données d'une seule région et en testant sur les autres, mais je n'arrive pas à un résultat excédant 60%. Je me suis par la suite aperçu que les données d'entraînement avaient beaucoup plus de données contenant du café que de données n'en contenant pas.

06.07.2020 : J'ai relancé mon nouveau réseau de neurones sur cette fois 10000 itérations. On constate que la visibilité de l'apprentissage augmente, mais que la visibilité des tests reste grosso modo la même. J'en déduis que le système n'arrive pas très bien à faire la distinction entre les champs de café et les champs de poivre. Il faudrait sans doute tester ce réseau sur une autre période, comme par exemple lors de la floraison (les images satellites utilisées datant de fin février 2019). J'ai aussi changé le set de données de l'autre expérience de la veille pour mieux équilibrer les classes. Je me suis aussi rendu compte d'une petite erreur dans mon code (oublié de renommer une variable, ce qui faisait qu'il ne taguait pas correctement les données de test), après ceci, mon réseau arrive à une visibilité d'environ 84%, ce qui comme expliqué lors de la conférence avec le CIAT est normal. J'ai aussi commencé à rédiger le rapport final.

07.07.2020 : J'ai relancé mon réseau de neurones sur 1000 itérations pour tester quelles sont les meilleures combinaisons de bandes. Cela me prendra sans doute quelques jours. En parallèle, j'ai modifié mon programme de partitionnement des images de Sentinel-2 pour pouvoir les mettre dans des dossiers spécifiques selon si l'image contient une position taguée auparavant dans des fichiers shapefile. Cela permettra de rendre beaucoup plus simple et rapide la sélection des données lors d'une éventuelle future sélection des données d'entraînement. J'ai aussi continué la rédaction de mon rapport.

08.07.2020 : J'ai continué à lancer mon réseau pour savoir quels sont les combinaisons de bandes les plus pertinentes. En parallèle j'ai essayé de faire tourner un réseau avec d'un côté des champs de café et de l'autre des champs de poivre. Au départ, cela ne marchait pas ; le réseau n'apprenait rien (mettait tout dans pas café), j'ai rajouté des neurones dans la partie de la couche totalement connectée et cela semble mieux marcher, j'obtiens des scores d'environ 70% (75 dans le meilleur des cas). En revanche, la majorité des champs de poivre ne vient que d'une seule zone. Je précise aussi que les images ont été prise en fin février. Je devrais peut-être prendre plusieurs images de la même zone à des dates différentes pour voir si on arrive à avoir une meilleure visibilité. Pendant que mes scripts tournaient, j'ai continué mon rapport.

09.07.2020 : J'ai fait un modèle multi classes avec d'un côté les champs de café, de l'autre les champs de poivre et deux autres classes qui sont les champs de poivre et de café et le reste (pour ce modèle-ci je n'ai utilisé que des images de forêt et de caoutchouc pour cette dernière classe). Il semble qu'il arrive plus ou moins à distinguer les différentes classes. Tout ce qui n'est pas champs de café ni de poivre est très bien classé. En revanche en ce qui concerne les champs de café et de poivre, le réseau a du mal à faire la différence même si dans la majorité des cas les images sont bien classées. Je remarque aussi que les images qui contiennent à la fois du café et du poivre sont classées soit dans l'un soit dans l'autre et jamais dans la classe à part qui leur est attribuée. Ceci est dû à la faible représentation de ces dernières images (seulement 47 en tout). J'ai aussi eu un entretien avec M. Perez-Uribe. Il m'a conseillé d'utiliser les f-mesures en plus de la visibilité et de diminuer l'overfitting du réseau. J'ai modifié mon réseau en rajoutant une fonction calculant le rappel, la précision et le f-score, en faisant une rotation des images pour avoir plus de données, en réduisant le nombre de neurones présents dans les convolutions et en diminuant le nombre d'époques.

10.07.2020 : J'ai lancé mes scripts et ai continué la rédaction de mon rapport. J'ai aussi adapté mon script de prédiction pour qu'il soit capable de donner une image de sortie en rgb en fonction d'une prédiction donnée avec de multiples classes (café, poivre et autre). Je ne l'ai pas encore testé (autant mon ordinateur qu'Hyperion était occupé à entraîner des réseaux pour savoir quelles sont les meilleures combinaisons de bandes). J'ai aussi téléchargé 3 fois la même zone, une fois prise en septembre, une autre en décembre et une dernière en mars, les ayant ouverts dans QGIS, je constate que nous pouvons bien les superposer, les pixels semblent bien correspondre. J'ai aussi dérivé mon notebook de découpage pour qu'il puisse aussi faire une division en images de 12 bandes (j'ai laissé tomber pour l'être car trop de nuages). Cependant je pense que l'entraînement d'un tel réseau me prendra quelques jours (je n'en suis pas encore là).

11.07.2020 : J'ai juste lancé mes scripts et relevé les résultats une fois ceux-ci finis.

12.07.2020 : Comme hier.

13.07.2020 : En attendant les derniers résultats, j'ai continué le rapport et j'ai aussi commencé le résumé publiable.

14.07.2020 : J'ai analysé les résultats tant attendus et j'en ai tiré les conclusions suivantes : La bande la plus utile est la bande rouge, suivie de la bande infra-rouge. Les bandes bleue et verte arrivent en queue de peloton. Pour la bande verte, il se peut que suivant la saison elle soit plus ou moins utile, ayant fait cette expérience sur des images hivernales, ce résultat peut être fortement biaisé pour cette bande-ci. J'ai aussi regardé mon script étant de classification café/poivre/autre. Le fichier de sortie semble

être assez bien quoiqu'il confonde bien souvent les endroits sans arbres naturels avec des champs de café. Cela est sans doute dû à la faible présence de ces derniers dans les données d'entraînement ou/et du fait que les images sont trop petites pour voir la géométrie des champs. J'ai aussi continué mon rapport.

15.07.2020 : J'ai un peu modifié et entraîné une nouvelle fois mon réseau de neurones de classifications multiples, en changeant quelques trucs selon les observations d'hier. Le fichier de sortie me semble un peu mieux, même si la précision, le rappel et autres f mesure sont à peu près les mêmes (toujours environ 75%) (sauf pour la zone désertique). J'ai aussi essayé de faire un entraînement sur une seule zone et tester le reste sur les autres zones, le résultat était pitoyable... Du coup j'ai commencé à voir découper des images et modifier mon script d'entraînement pour faire un réseau de neurones multi-saison.

16.07.2020 : J'ai fini de créer le set de données et ai essayé mon nouveau réseau sur toutes les bandes (12 au total). Le résultat est à peu près le même qu'avec le réseau de neurones à 4 couches que j'ai testé jusqu'à présent, mais il se peut qu'il soit plus performant dû au set de données qui contient plus de diversité au niveau des « pas cafés » (et sur 500 époques seulement). Je n'ai pas pris d'image en été car il y a beaucoup trop de nuage dans cette période (pire qu'en automne qui était déjà pas mal nuageux).

17.07.2020 : Je vais faire 3 sélections de bandes ici, une par saison. Je commence par tester celles d'automnes. En attendant les résultats, je repasse sur Sentinel-1 et lance le script de découpage d'image.

18.07.2020 : J'ai continué à tester mes bandes. Les petites images de Sentinel-1 ne sont toujours pas générées.

19.07.2020 : J'ai fini de tester les bandes pour l'automne, je passe aux bandes d'hiver. J'attends toujours les images de Sentinel-1 pour mon set de données.

20.07.2020 : Je passe au printemps. Je me suis aussi aidé d'un ancien labo de VTK (le 5) pour savoir si une coordonnée se trouve dans une image de Sentinel-1 ou non. Cela me permettra de ne pas passer plus d'une semaine à faire de la sélection des données, pour l'instant il ne marche pas très bien car les sous-images prennent quand même tous les GroundControlPoints de l'image de base... J'ai aussi effectué les modifications pour un entraînement de Sentinel-1 sur toutes les saisons (en introduisant aussi le paramètre class_weight). J'ai aussi continué la rédaction de mon rapport. Quant à la division de l'image Sentinel-1, elle devrait se finir ce soir si tout va bien...

21.07.2020 : J'ai remarqué que contrairement aux images de Sentinel-2, les images de Sentinel-1 n'étaient pas toujours prises au même endroit, il y a donc un petit décalage plus que probable entre plusieurs images de la même zone prises à des dates différentes. J'ai fait quelques fonctions pour calculer un point de ralliement, une fonction pour calculer les offsets utiliser pour arriver à ces points et une fonction pour translater le tableau d'entier pour que ça première position coïncide environ (il est probable que l'offset ne soit pas un entier, donc on l'arrondit au plus proche) avec la coordonnée calculée. En ce qui concerne le point de ralliement, je prends simplement la longitude et la latitude les plus petites. Si un offset est négatif, on remplira le tableau décalé avec la valeur 0 jusqu'à arriver à un point contenant des valeurs. J'ai aussi fait des fonctions pour avoir les quatre coins des petites images de Sentinel-1. En les ouvrants dans QGIS, j'ai constaté qu'elles étaient affichées en décaler comparé à l'image de base (de quelques centaines de mètres). Leurs GroundControlPoints, quant à eux sont les mêmes que dans l'image de base, si ce n'est les attributs col et row qui changent selon la position de l'image.

22.07.2020 : J'ai essayé de lancer plusieurs fois mon script, mais à chaque fois que je le fais sur Hyperion, je perds la connexion (après deux heures d'attente), cela est sans doute dû à une trop grande utilisation de la mémoire. Je vais donc générer mes fichiers tiff utilisés directement sur mon ordinateur avant de les lui passer (ça prend plus de 2 heures par fichiers). J'ai aussi continué mon rapport et finalement fini toute la saison du printemps.

23.07.2020 : Ayant fini les saisons, je lance une fois mon script sur les meilleures combinaisons possibles. En même temps, je l'ai aussi lancée sur l'intégralité des données labellisées comprises dans les images, dans ce cas, la précision est moins haute, à cause du fait qu'il y a plus de données sans café. J'ai aussi lancé le script de découpage de Sentinel-1 en 30 sur 30 pixels, je devrai avoir le résultat du découpage demain.

24.07.2020 : J'ai lancé mon script multi classes sur toutes les bandes, le résultat est en moyenne pareil un peu meilleur que celui de base. Ses quelques erreurs peuvent s'expliquer par le fait que les données utilisées sont aussi plus nombreuses. J'ai aussi fait l'expérience de tester dans une région et tester sur les autres avec toutes les bandes.

25.07.2020 : J'ai continué de lancer mes expériences d'hier. Je me suis aussi aperçu que mon code pour classer les images de Sentinel-1 ne marchait pas. À cause de la précision des floats en python, dans certains cas je me retrouve avec des vecteurs ayant comme dernière composante 0 ce qui me renvoie un nan quand j'essaie de savoir si une position labellisée se trouve ou non dans l'image. Ce problème semble dépendre de la position de l'image et de la taille de cette dernière. Pour « résoudre » le problème, j'ai modifié mon code pour que s'il trouve la dernière composante du vecteur nulle, on regarde la moyenne des bordures de l'image (approximation). J'ai relancé mon script de tri.

26.07.2020 : J'ai vu le résultat de mon tri des images automatique. Les labels donnés ne sont pas forcément dans l'image elle-même, mais peuvent aussi se trouver un peu à côté. Ceci sans doute dû au fait de la précision des calculs. Quoiqu'il en soit avoir une précision d'à peine une centaine de mètres me semble une approximation assez correcte. Je commence à checker mes images. J'ai aussi analysé les résultats d'entraîner sur une zone et de tester sur les autres, les résultats ne me semblent pas spécialement bons, mais relativement meilleurs que ceux obtenus précédemment.

27.07.2020 : J'ai lancé mon script d'entraînement sur Sentinel-1, les résultats de la validation croisée me semble plutôt bon (plus de 60% de précision pour le café, le reste est très bien classé), j'ai aussi commencé à découper une image seule de Sentinel1 pour pouvoir comparer les résultats avec l'image comportant toutes les saisons. J'ai aussi essayé d'entraîner un réseau sur deux zones et de le tester sur les autres, le résultat semble plus ou moins égal à l'expérience où l'on n'entraînait que sur une seule zone.

28.07.2020 : J'ai téléchargé toutes les images de 2018 utiles pour tester notre réseau sur des données anciennes (voir le résultat). Je me suis aperçu que parfois, mon réseau de neurones n'arrivait pas très bien à démarrer avec les images de Sentinel-1, j'ai donc ajouté des neurones dans les différentes couches pour diminuer l'apparition du problème. J'ai fini de checker les petites images de Sentinel-1.

29.07.2020 : J'ai continué d'entraîner le réseau de neurones sur Sentinel-1. Ici, je vais aussi analyser toutes les bandes. Cependant, j'ai quelques petits problèmes avec mon ordinateur (sans doute qu'il est trop chargé), qui fait que quelquefois je perds la connexion avec Hyperion (et l'avancée de mes notebooks au passage)... J'ai continué le rapport.

30.07.2020 : Bien qu'ayant fait de la place sur mon PC, je ne peux pas l'utiliser pour faire des prédictions avec plus de 4 bandes, car le kernel meurt pendant qu'il charge la cinquième (sans doute un problème de mémoire allouée)... Du coup je ne peux utiliser que Hyperion en ce moment. J'ai donc continué mon rapport.

31.07.2020 : J'ai continué de faire tourner mes entraînements pour les images de Sentinel-1. En attendant j'ai essayé de faire le notebook pour la prédiction des dites images. J'ai aussi continué le rapport.

01.08.2020 : J'ai fini les entraînements de Sentinel-1, sauf une seule combinaison de bande qui refuse d'apprendre. J'ai donc commencé à découper mes images de 2018-2019 pour tester mes réseaux sur des données plus vieilles. J'ai bien entendu aussi continué mon rapport.

02.08.2020 : J'ai créé un script pour tester mes modèles sur les images de 2018-2019. Le résultat n'étant pas celui espéré, je vais télécharger du coup des images 1C pour refaire des modèles à partir de ces données-ci

en espérant avoir de meilleurs résultats (les produits 2A n'étant pas disponibles pour la période d'automne 2018...). J'essaie d'upload le tout sur Hyperion, mais celui-ci semble aussi arriver au bout de sa mémoire...

03.08.2020 : J'ai principalement continué mon rapport et mon résumé publiable. J'ai aussi téléchargé sur Hyperion toutes les images dont j'avais besoin pour faire marcher le test d'une année avant, j'ai aussi fait des entraînements sur ces données pour avoir des modèles à tester. J'ai aussi sorti toutes les prédictions sur mes modèles 2A simples (toujours sur l'image de gauche). Pour les ajouter à mon rapport. Le problème d'hier a aussi refait surface, mais pas longtemps.

04.08.2020 : J'ai principalement continué mon rapport. J'ai aussi lancé quelques scripts dont la comparaison entre cette année et l'année précédente, cette fois les résultats sont ceux attendus. J'ai aussi lancé pas mal de scripts de prédiction pour mettre des images dans mon rapport final (une prédiction me prenant 3 heures, le « pas mal » est relatif).

05.08.2020 : Me rendant compte que les images de sortie de ma prédiction avec le type 1C étaient beaucoup plus sombres que les images faites à partir du type 2A, j'ai décidé de refaire une sélection des bandes en 1C, pour pouvoir comparer. Je devrai tout juste avoir le temps de faire ces expériences. J'attends aussi que mon script de prédiction pour S1 me sorte quelques choses, j'avais essayé de le lancer pendant la nuit, mais il s'est déconnecté d'Hyperion, il s'agit sans doute du même problème que lors du découpage. Du coup, je pense faire un script générant des images translatées à côté, même si cela est plus gourmand en mémoire. La prédiction prendra beaucoup de temps, j'espère avoir une image de sortie avant de rendre ce rapport...

06.08.2020 : J'ai enfin pu générer l'image de sortie de Sentinel-1 qui me semble même être un meilleur résultat que les images de prédiction de Sentinel-2 (même si elle a été générée sur les données d'entraînement). Du coup j'ai fait une correction complète de mon rapport et j'attends encore quelques-uns des résultats d'entraînement sur 1C.

07.08.2020 : J'ai encore mis quelques résultats attendus dans le rapport, fait le tri sur les fichiers à envoyer ou non, fait mon affiche et ai envoyé le tout.