19/06/2020

Travail de Bachelor
Rapport Intermédiaire

L'intelligence artificielle au service de l'agriculture durable

Rod Julien
HEIG-VD

Table des matières

Contexte d'application

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

Le projet se fait en collaboration avec le Centre de Recherche en Agriculture Tropicale (CIAT) basé au Vietnam.

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

Les deux sous-chapitres ci-dessous sont l'énoncé et le cahier des charges tel qu'ils ont été définis.

Énoncé

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

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

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.

Données disponibles

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[1].

Sentinel-2

Sentinel-2[2] monitore 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 les deux approches : soit RGB, soit RG+IR qui, semblerait-il, refléterait plus la végétation.

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: Bande de sentinel-2[3]

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.

[1] https://scihub.copernicus.eu
[2] https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2
[3] https://www.satimagingcorp.com/satellite-sensors/other-satellite-sensors/sentinel-2a/

Sentinel-1

Sentinel-1[4] 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 environs. Il semblerait que Sentinel-1 soit fortement utile pour la détection de la déforestation[5].

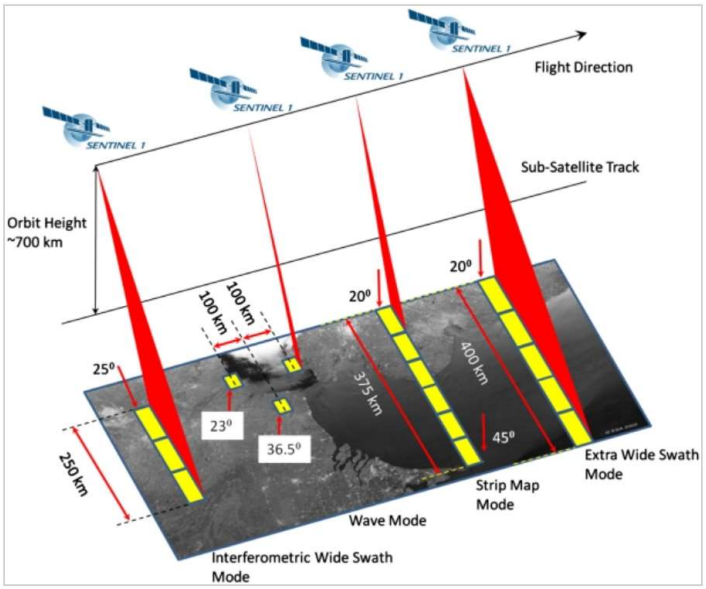


Figure 2: Sentinel-1 mode[6]

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.

QGIS

Nous utilisons le logiciel QGIS[7] 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 prédiction correspondante.

[4] https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-1
[5] https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-1-sar/applications/land-monitoring
[6] https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/acquisition-modes
[7] https://www.qgis.org/fr/site/

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…

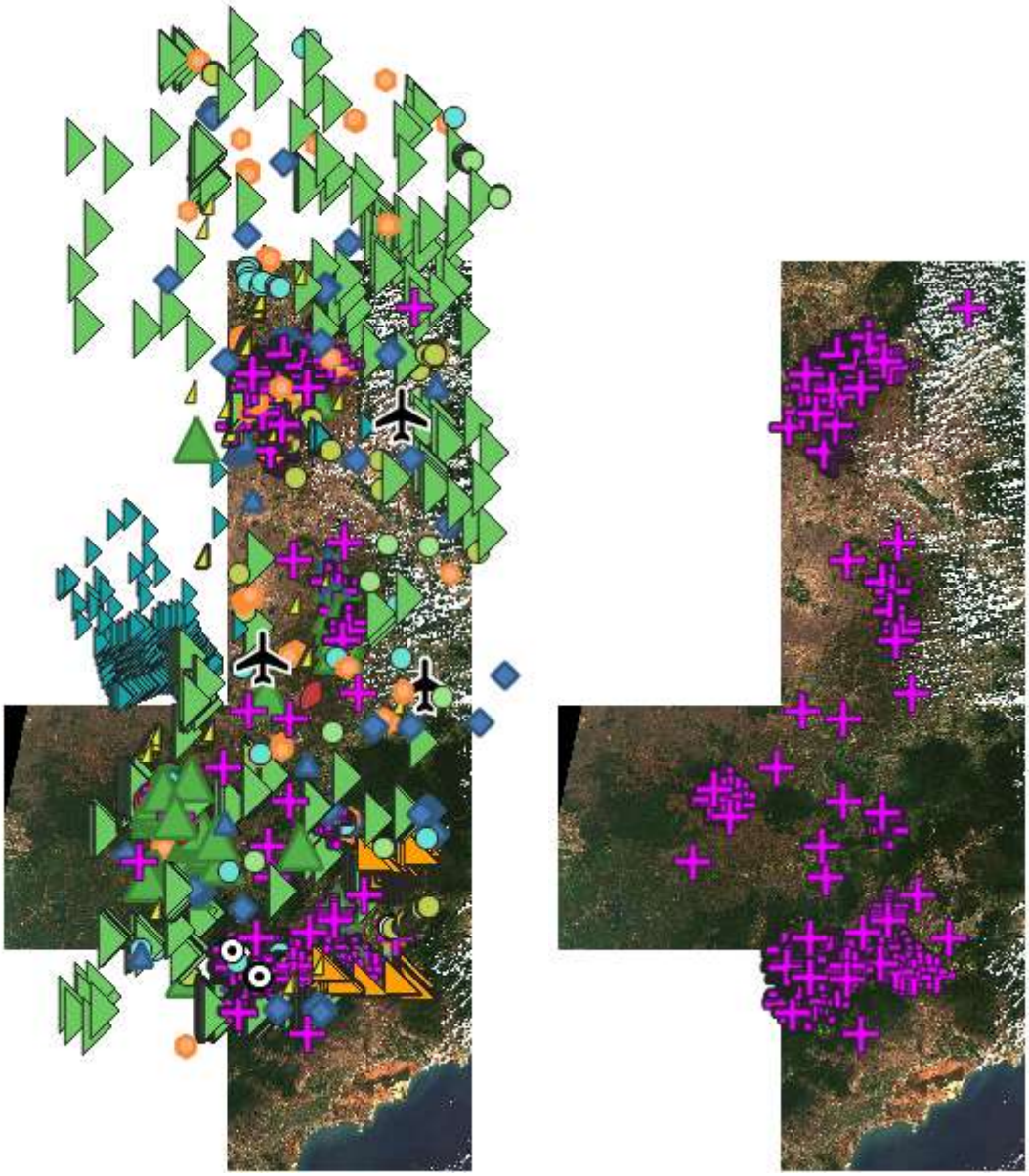


Figure 3 : À gauche, la liste des de toutes les données fournies par le CIAT.
À droite, la même liste en ne gardant que les champs de café.

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).

Modules utilisés

Les scripts sont réalisés en python 3.7. En plus de toutes les données citées plus haut, voici une liste des modules python que j'utilise dans ce projet :

- Numpy[8] pour tout ce qui est gestion de liste.
- Rasterio[9] pour travailler sur les images tiff et jp2.
- Sklearn[10] pour faire de la validation croisée pendant l'entraînement de mes modèles.
- Keras[11] pour créer et utiliser mes modèles.
- TensorFlow[12] par l'intermédiaire de Keras.
- Affine[13] pour pouvoir géolocaliser les membres d'une partition d'une image de Sentinel-2
- Matplotlib[14] pour afficher des éléments
- Gdal[15] pour diviser et géolocaliser correctement une image Sentinel-1 en de plus petites images en vue de l'entraînement d'un réseau de neurones.

J'utilise des notebooks Jupyter pour lancer et stocker les scripts python.

Ressources utilisées

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 entrainer 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 pixel. En d'autres termes, le réseau de neurones est capable de reconnaitre 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.

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 perceptron).

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

[8] https://numpy.org/
[9] https://rasterio.readthedocs.io/en/latest/intro.html
[10] https://scikit-learn.org/stable/
[11] https://keras.io/
[12] https://www.tensorflow.org/?hl=fr
[13] https://pypi.org/project/affine/
[14] https://matplotlib.org/
[15] https://gdal.org/

la taille du kernel). Ainsi l'image du début sera transformée en une matrice de score puis passée à la couche suivante.

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'entrainement 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'il 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 ».

Optimiseur Adam

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

Ici nous utiliserons les paramètres par défaut proposé 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[16].

[16] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Travail effectué

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 Sentinels utilisées dans ce projet. Je me suis donc aussi familiarisé avec le site permettant de télécharger les images[17] des différentes missions Sentinel. J'ai aussi lu un rapport montrant une mission qu'avait déjà effectuée le CIAT dans la région observée[18].

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[19] sur le sujet.

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 correspondait. 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.

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.

[17] https://scihub.copernicus.eu/dhus/#/home
[18] https://www.sciencedirect.com/science/article/pii/S0303243419307202?via%3Dihub
[19] https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 et https://www.kdnuggets.com/2018/04/building-convolutional-neural-network-numpy-scratch.html entre autres.

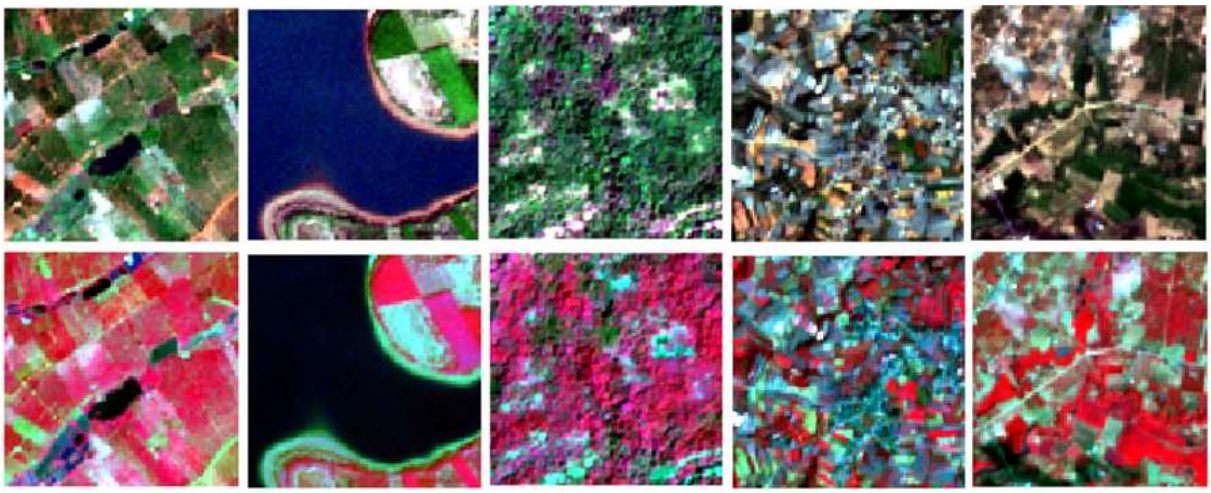


Figure 4 : De gauche à droite : Champs de café, lac, forêt, ville et image contenant des champs inconnus, le haut représente des images en RGB et le bas des images en RG+IR

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.

Je n'ai pas avancé sur le projet pendant le mois de mai, car j'étais convoqué par la protection civile.

Revenant de ma période de mobilisation et ayant fini de rattraper les autres cours, j'ai 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 même image dans les 3 couleurs (ce que fait 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%.

```python
def reshape(image):
    """Reformate 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 5 : Passer une image de Rasterio à Tensorflow + normalisation des données.

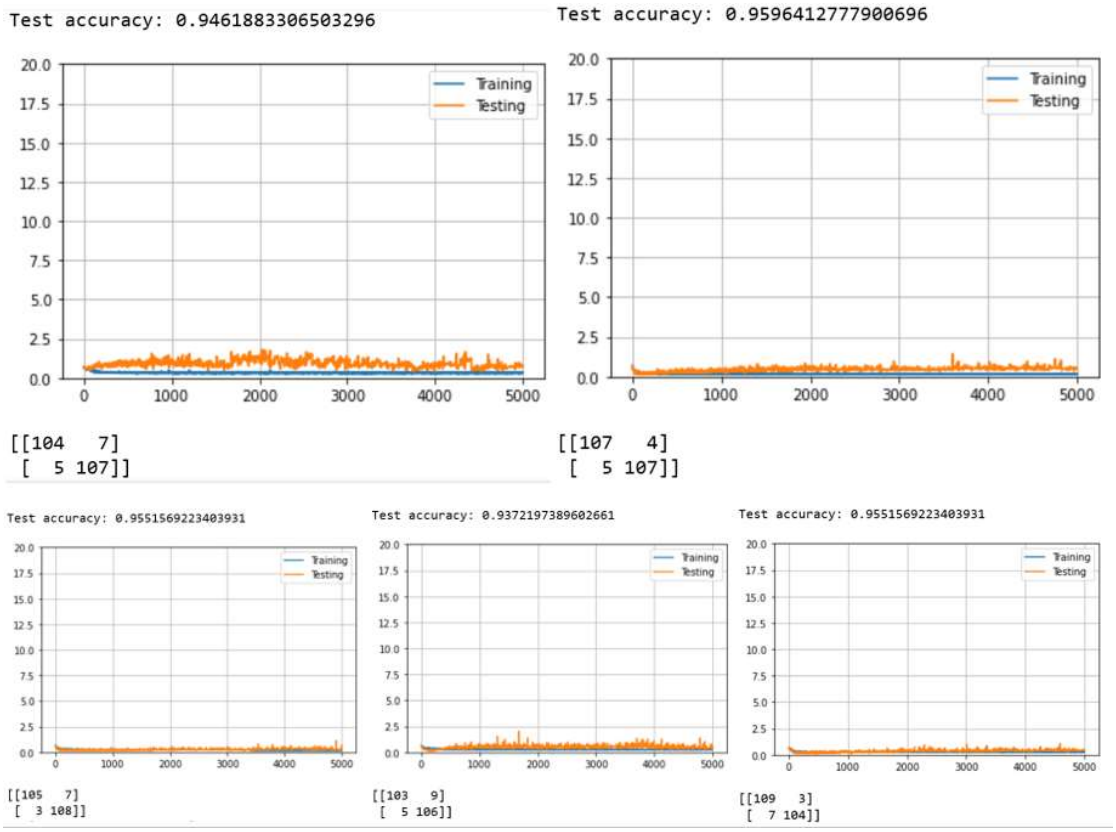


Figure 6 : Visibilités, graphiques d'apprentissage et matrices de confusion d'une même validation croisée

On constate sur la figure ci-dessus que notre validation croisée est stable, c'est-à-dire que le résultat d'entrainement 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.

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 ci-dessous.

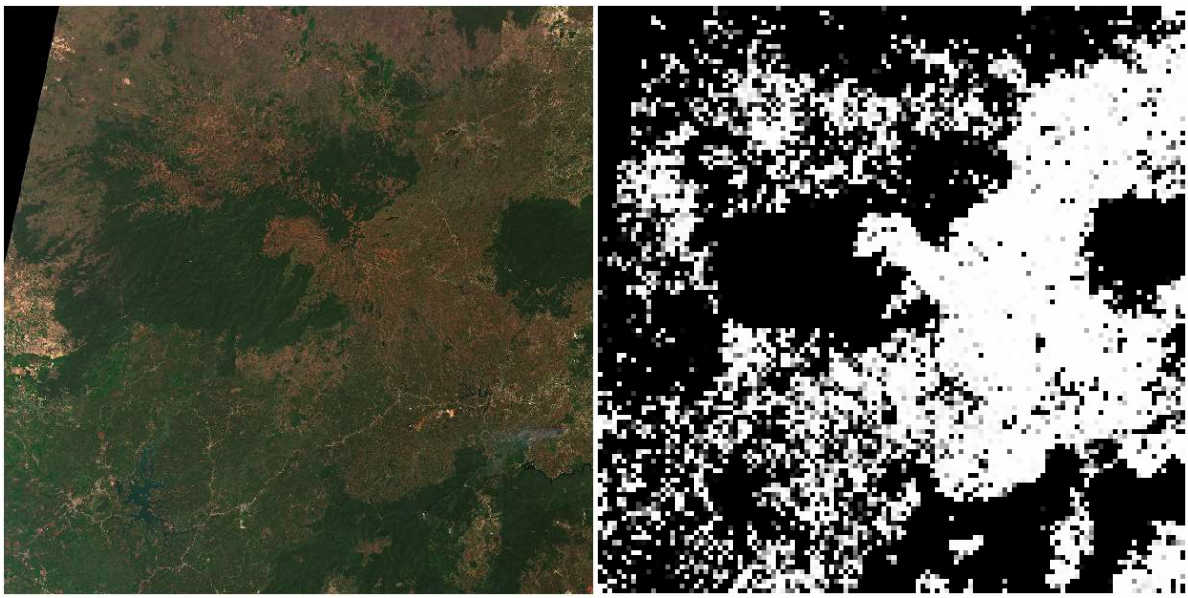


Figure 1 Image Sentinel-2 et sa prédiction

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 entrainé sur les données RG+IR, j'essaierai d'entrainer des modèles RGB dans ces prochains jours.

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). Je devrai donc bientôt être en mesure d'entrainer un réseau de neurones convolutif sur ces images-ci.

Dernièrement, j'ai rédigé ce rapport.

Description des scripts réalisés jusqu'à présent

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

my_convolutional_neural_network.ipynb

Ce script (dont le nom deviendra sans doute plus intuitif dans un futur proche) est le script permettant d'entrainer notre réseau de neurones.

Il prendra tous les fichiers se trouvant dans la liste des dossiers donnés comme ensemble d'entrainement. 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.

```
LIST_OF_DIRECTORIES = ["output/cafe_bas/rir", "output/cafe_droite/rir", "output/cafe_gauche/rir", "output/cafe_haut/rir",
                       "output/cafe_milieu/rir", "output/pas_cafe_bas/rir", "output/pas_cafe_droite/rir",
                       "output/pas_cafe_gauche/rir", "output/pas_cafe_haut/rir", "output/pas_cafe_milieu/rir"]
COUNTAIN_START = "output/cafe_"
PIXEL_LENGTH = 90
BAND_WIDTH = 3
NB_SPLIT = 5
EPOCH = 5000
```

Dans l'image ci-dessus, on voit comment sont représenter les chemins des dossiers contenant les données d'entrainement. La variable COUNTAIN_START servira à attribuer la réponse attendue, 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 3 pour RGB ou RG+IR). NB_SPLIT représente le nombre de partitions des données d'entrainement pour la validation croisée. Finalement EPOCH représente le nombre de fois que l'on doit présenter les données d'entrainement à un modèle.

Dans sa configuration actuelle, ce réseau de neurones convolutif prend des images de 90 sur 90 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):
```

Avec images_x comme input et image_y comme réponse attendue dans l'output correspondant.

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 actuelle.

```
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(Conv2D(16, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2 ,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(4, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

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.

Comme dit dans la partie « travail effectué », ce modèle a déjà été testé sur un set d'images de Sentinel-2, mais devrait aussi être compatible avec les images de Sentinel-1. Je devrai tester ceci dans les jours à venir.

L'entraînement d'un réseau de neurones pouvant prendre beaucoup de temps, ce script se trouve actuellement sur le serveur Hyperion. Si vous souhaitez l'utiliser il faut activer l'environnement Anaconda env_tb.

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

partOfImage.ipynb

Ce script permet de diviser les images de Sentinel-2 en une multitude de petites images (2 * 14'884 pour être précis) de 90 sur 90 pixels. Cela nous est utile pour la création de données d'entrainement 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) pour créer deux types d'images celles en RGB et celle en RG+IR.

```
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
```

À noter que le dossier d'output spécifié doit exister. Sinon une erreur est relevée.

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 90 et IMAGE_WIDTH de 122. Il faudra découper 3 bandes pour créer les bandes utilisées pour créer les images plus petites (RGB ou RG+IR).

```
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
```

Ce code est aussi présent dans partOfImageS1 et Predict.

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

```
chang = blue.transform * (y * IMAGE_PIXEL,x * IMAGE_PIXEL)
transform = affine.Affine(10.0, 0, chang[0], 0.0, -10.0, chang[1])
```

partOfImageS1.ipynb

Ce script est la version Sentinel-1 du script ci-dessus. En bref, ici nous prenons les bande vv et vh données par Sentinel-1 (mission GRD). Et on génère une image en RGB grâce à elle[20]. 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.

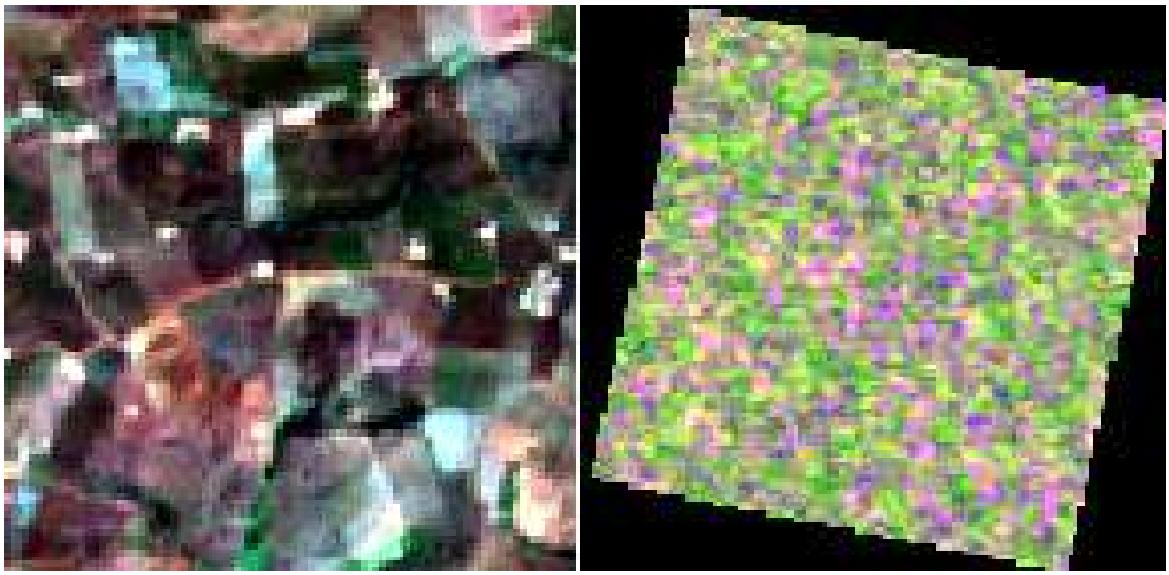
Les chemins vers les deux bandes de Sentinel-1 sont définis ainsi :

```
imagePath = '../Données/S1B_IW_GRDH_1SDV_20200414T223607_20200414T223641_021146_028202_F368.SAFE/measurement/'
vh = rasterio.open(imagePath+'s1b-iw-grd-vh-20200414t223607-20200414t223641-021146-028202-002.tiff', driver='Gtiff')
vv = rasterio.open(imagePath+'s1b-iw-grd-vv-20200414t223607-20200414t223641-021146-028202-001.tiff', driver='Gtiff')
```

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 partOfImage.

Pour se rendre compte de comment sont arranger 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 qui est de 90 sur 90. Ce qui signifie que les surfaces qu'elles recouvrent sont de tailles équivalentes.

[20] https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/product-overview/polarimetry



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 autre), 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.tiff'

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) + \
            ", " + str(IMAGE_PIXEL) + " " + str(in_file) + " " + str(out_path) + str(output_filename) + \
            str(int(i/IMAGE_PIXEL)) + "_" + str(int(j/IMAGE_PIXEL)) + ".tiff"
        os.system(com_string)
```

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
```

rename_all_files.ipynb

Ce petit script permet de renommer un fichier. Je pensais l'utiliser pour traiter toutes les images présentes dans un dossier, puis ensuite les déplacer dans un autre, grâce à ce script. Cela peut être utile pour si on automatise les tâches de prédictions.

```
for filename in os.listdir(DIRECTORY):
    if not filename.startswith(PARTICULE):
        os.rename(DIRECTORY + "/" + filename, DIRECTORY + "/c_" + filename)
```

Predict.ipynb

Ce script permet de faire une prédiction sur la base des images de Sentinel-2. Comme pour le script d'entrainement, 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 entrainé.

Il fera la prédiction prenant des parties de 90 sur 90 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])
        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)
```

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.

Travail Restant

Il me reste à faire le point 1 du cahier des charges qui n'est actuellement pas commencé. C'est-à-dire à faire un script pour importer directement les image de Sentinel.

Pour les autres points, il faut encore que :

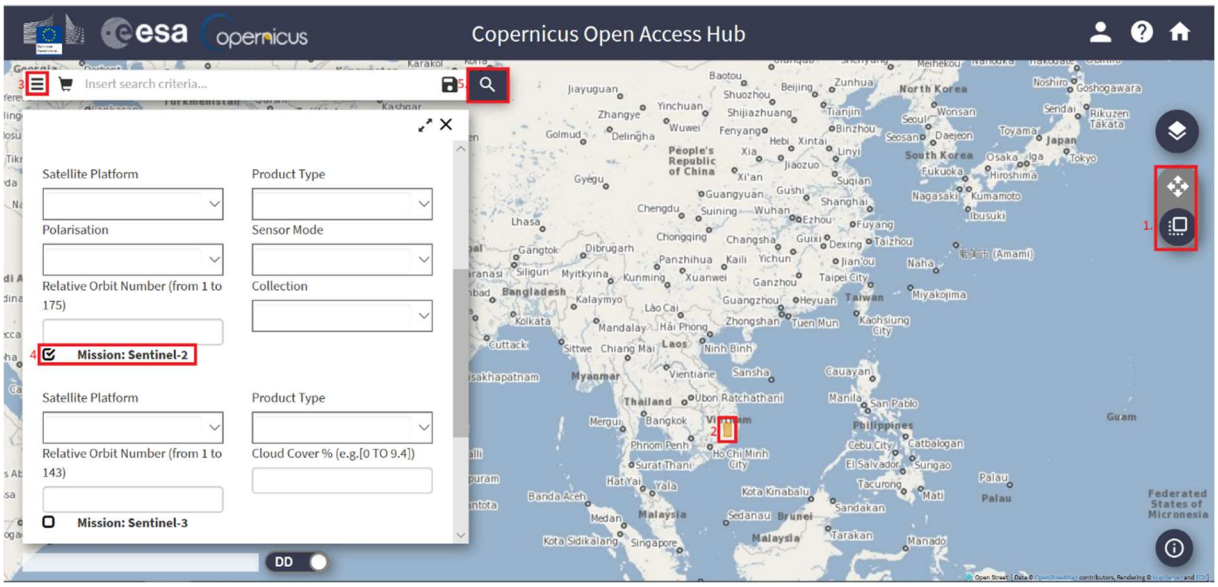
- J'essaie d'améliorer mes modèles.
- J'entraine mon réseau avec des images RGB (ne l'a seulement fait pour les images RG+IR).
- Je crée un set de données d'entrainement avec des images de Sentinel-1.
- Je fasse tourner mon réseau de neurones sur une image de Sentinel-1.
- Je crée un script de prédiction pour Sentinel-1.
- Je voie si l'image de sortie couplée à QGIS est suffisante pour l'analyse des résultats. Et le cas échéant, développe un outil plus élaboré.

Marche à suivre

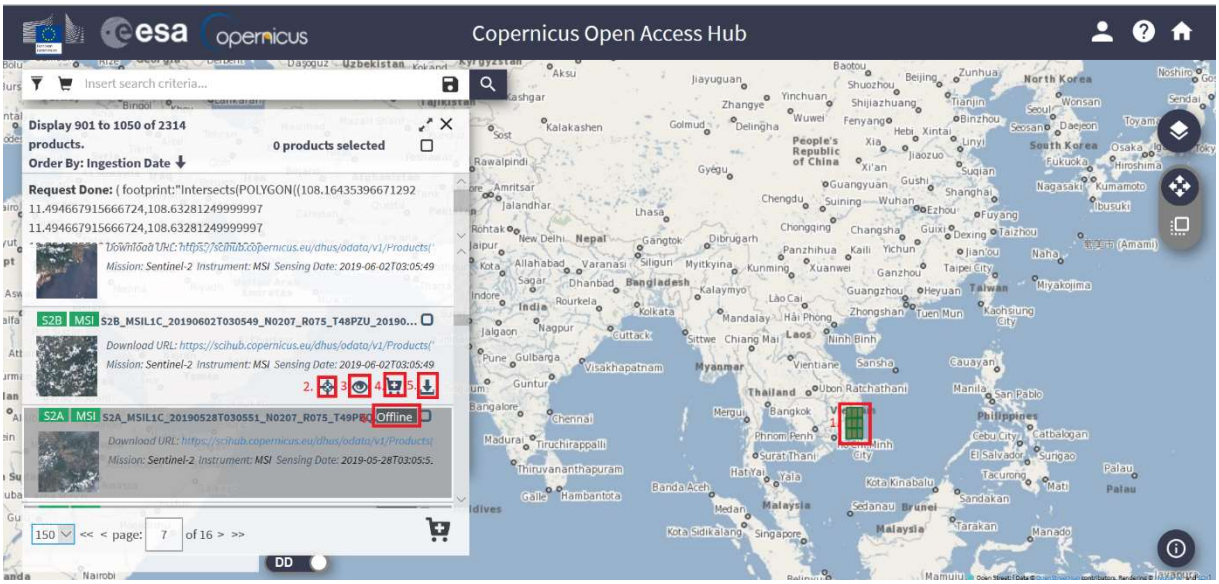
Entrainement

Télécharger des images depuis le site de l'ESA.

(À noter que cette étape sera différente une fois le point 1 du cahier des charges implémenté.)



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.
2. Exemple de rectangle obtenu via la deuxième option du point 1.
3. Sert à ouvrir le menu des options (qui est affiché en dessous).
4. 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.
5. Effectuer une recherche.



1. Zone contenant toutes les images[21] 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 d'ajout au panier (ou sur le bouton de téléchargement). Après quelque temps, l'image devrait être disponible dans notre panier. À noter que les images sont archivées si elles datent de plus d'une année environ.

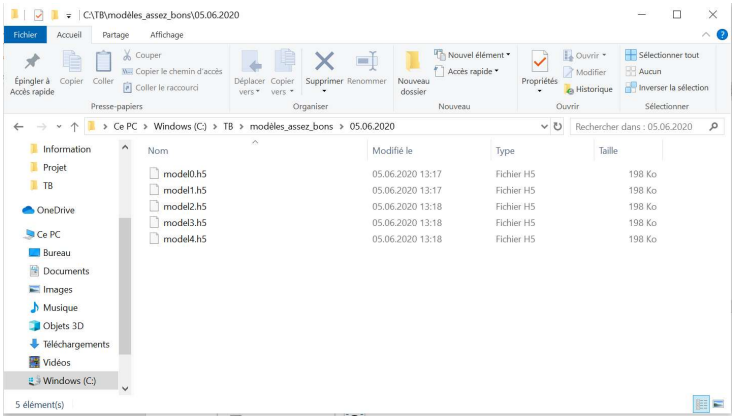
Utiliser le script partOfImage pour découper l'image téléchargée en images plus petites.

Utiliser QGIS pour sélectionner les meilleures images partitionnées pour créer les données d'entrainement. Prendre autant d'image contenant la classe voulue que d'image qui ne la contient pas.

Figure 7 : Faire correspondre des images RG+IR avec les champs de café dans QGIS.

Entrainer un réseau de neurones avec les données d'entrainements en utilisant le script my_convolutional_neural_network. Cela prendra quelques heures, suivant le nombre d'époques définies.

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

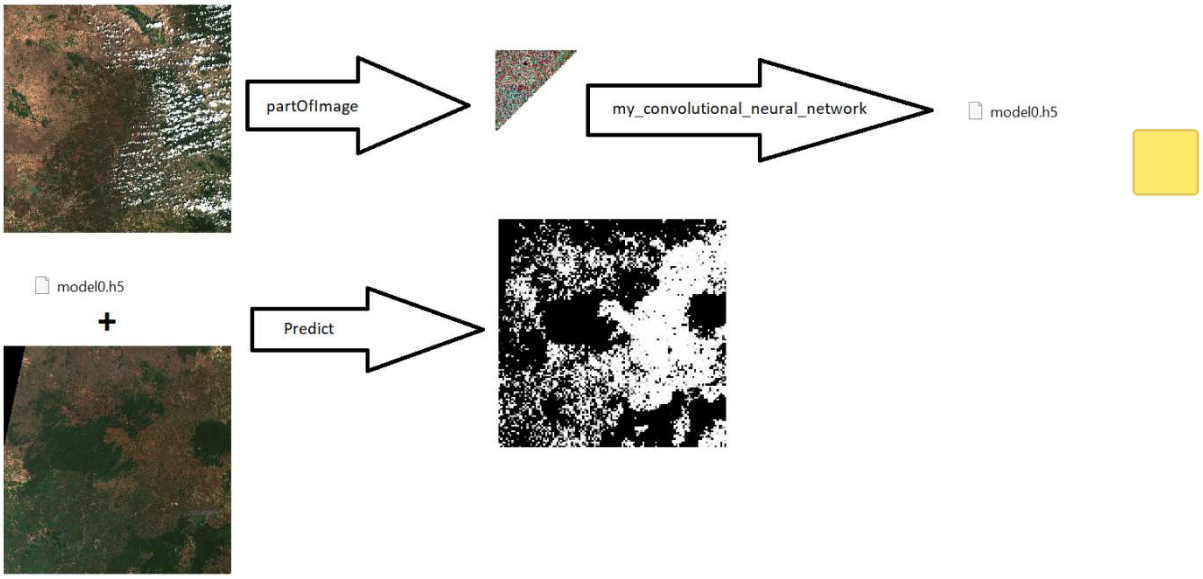


[21] Dans ce paragraphe, le mot image fait référence à la totalité des mesures effectuées. Il s'agit souvent de plusieurs bandes et d'autres informations.

Prédiction

Utiliser le script Predict en lui spécifiant une image Sentinel-2 et un modèle créé grâce à la marche à suivre ci-dessus.

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



Remarques et conclusion

On peut le voir dans l'exemple du chapitre « travail effectué », pour l'instant le réseau de neurones arrive à faire la différence entre la forêt et les champs. En revanche, il serait bien de savoir plus précisément si le champ contient bel et bien une plantation de café. Cela m'étonnerait que du café se trouve dans toutes les zones catégorisées comme fortement probable par le système.

Sentinel-1 me paraît plus adapté pour la détection des champs de café, notamment grâce à sa capacité à reconnaitre les différentes plantations. Je pense qu'il pourrait nous apporter un point de vue différent pour améliorer l'efficacité de nos réseaux de neurones.

Cependant, je vois mal comment on pourrait faire pour combiner les 2 types d'images. Leurs tailles, leurs orientations et les mécanismes permettant de définir leurs positions me paraissent trop différents pour réussir à savoir quel pixel de Sentinel-1 correspond à un pixel de Sentinel-2 donné. La solution la plus réaliste consiste simplement à superposer les images sorties des prédictions.

Une autre solution pour essayer d'améliorer les prédictions serait d'avoir des données plus pertinentes. Pour l'instant, pour l'entrainement, j'utilise des données dans lesquelles je suis absolument sûr qu'elles ne contiennent pas de champs de café, comme par exemple des images de forêt, de ville ou de mer, voir même certaines images dans lesquelles d'autres types de végétations y sont plantées en utilisant la liste fournie par le CIAT. Mais pour une image donnée contenant des champs, je suis incapable de savoir si elle contient un champ de café ou non (chaque image représentant 900 mètres carrés, il se peut qu'il y ait plusieurs types de culture dans une seule image). Avoir plus d'informations sur la culture exacte se trouvant sur chaque champ ou être sûr que les champs sans étiquette ne contiennent pas de café pourrait sans doute améliorer la différenciation entre les champs de café et les autres cultures.

J'avais envoyé un mail au CIAT, fin mars, pour savoir précisément de quand datait la liste de points donnés pour pouvoir télécharger les images satellites correspondantes, mais j'attends encore leur réponse… Bonne nouvelle récente j'aurai une conférence ce vendredi 26 juin.

Les images de Sentinel-2 sont pour la plupart inutilisables une grande partie de l'année, à cause de la saison humide sévissant pendant l'été et une grande partie de l'automne. Les images de Sentinel-1 ne semblent quant à elles ne pas être affectées par la présence de nuages.

En guise de conclusion, je pense que le projet avance assez bien même si j'ai un peu peur de piétiner sur la différenciation entre les champs de café et les autres types de cultures. Pour cela il me faudrait sans doute plus de données peut-être plus récentes.

Mon prochain objectif sera d'implémenter le point 1 du cahier des charges tout en faisant tourner un entrainement sur les données de Sentinel-2 en RGB. Mon deuxième objectif sera sans doute de créer un set de données pour pouvoir entrainer mon réseau de neurones sur les données de Sentinel-1.

Table des abréviations utilisées

CIAT
: Centre de Recherche en Agriculture Tropicale, 3

ESA
: European Space Agency, 3

RG+IR
: Red Green + Infra Red (Couleurs composant l'image), 4

RGB
: Red Green Blue (couleurs composants l'image), 4