



ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet Recherche & Développement

2019-2020

Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR

Tuteurs académiques

Thierry BROUARD

Jean-Yves RAMEL

Étudiant

Valentin MAURICE (DI5)

12 avril 2020



Liste des intervenants

Nom	Email	Qualité
Valentin MAURICE	valentin.maurice@etu.univ-tours.fr	Étudiant DI5
Thierry BROUARD	thierry.brouard@univ-tours.fr	Tuteur académique, Département Informatique
Jean-Yves RAMEL	jean-yves.ramel@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Valentin MAURICE susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Thierry BROUARD et Jean-Yves RAMEL susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Valentin MAURICE, *Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2019-2020.

```
@mastersthesis{
  author={MAURICE, Valentin},
  title={Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2019-2020}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
1 Introduction	1
1 Contexte de la réalisation	1
2 Objectif	2
3 Hypothèses	3
2 Description générale	4
1 Environnement du projet	4
2 Caractéristiques des utilisateurs	4
3 Fonctionnalités du système	4
4 Structure générale du système	5
3 État de l’art / Veille technologique	7
1 Réseaux de neurones	7
1.1 Les neurones	7
1.2 Les réseaux de neurones	9
1.3 L’entraînement	10
2 CNN	10
2.1 Première partie du réseau.....	11

2.2	Paramètres initiaux et entraînement	12
2.3	Transfert learning	13
3	R-CNN.....	13
4	Fast R-CNN	14
5	Faster R-CNN.....	14
6	Mask R-CNN.....	15
7	YOLO	16
8	FCN	17
9	DeconvNet	19
10	U-Net	19
11	Technologies et implémentations.....	20
11.1	Keras.....	20
11.2	Pytorch	20
11.3	Récapitulatif	21
12	Conclusion de l'état de l'art.....	21
4	Mise en oeuvre	23
1	Outils utilisés.....	23
2	Déviations par rapport aux spécifications	23
3	Tests et qualimétrie	24
5	Bilan et conclusion	25
1	Bilan personnel.....	25
	Bibliographie	27
	Annexes	29
A	Cahier de spécifications	30
B	Fiche de configuration	46
C	Fiche d'activité S9	48
D	Gantt d'activité S9	49
E	Gantt prévisionnel S10	50
F	Gantt d'activité S10	51
G	Manuel développeur	52
H	Manuel utilisateur	56

Webographie

64

Table des figures

1 Introduction

1	Schéma du fonctionnement du LiDAR (C. Laplaige, 2012)	2
2	Vue aérienne / télédétection LiDAR	2
3	Structures apparaissant sur les images LiDAR	3
4	Exemple d'un résultat de classification par SkyEye	3

2 Description générale

1	Cas d'utilisation du classifieur CNN, SkyEye	5
2	Diagramme simplifié, SkyEye	6

3 État de l'art / Veille technologique

1	Schéma d'un neurone biologique	7
2	Schéma d'un neurone abstrait	8
3	Schéma d'un neurone abstrait, avec les poids synaptiques	8
4	Courbe d'une fonction sigmoïde dans l'intervalle $[-10;10]$	9
5	Schéma résumé d'un neurone abstrait	9
6	Organisation d'un réseau en couches	10
7	Fonctionnement d'une convolution	11
8	Fonctionnement d'une opération max-pool	12
9	Architecture d'un réseau CNN	12
10	Performance de différents réseaux sur le jeu de données ImageNet	13
11	Fonctionnement d'un R-CNN	14
12	Fonctionnement d'un Fast R-CNN	15
13	Fonctionnement d'un Faster R-CNN	15

14	Comparaison des performances des architectures basée R-CNN.....	16
15	Exemple d'une segmentation par Mask R-CNN	16
16	Comparaison des architectures de type R-CNN	17
17	Grandes étapes d'une analyse YOLO	17
18	Structure d'un CNN classique.....	17
19	Structure d'un FCN	18
20	Exemple d'une déconvolution ou up-convolution	18
21	Exemple de fusion (sommations) des prédictions dans un FCN	18
22	Prédictions FCN après fusions à différents niveaux, et vérité terrain	19
23	Structure d'un DeconvNet	19
24	Phase d'unpooling, avec conservation de l'information spatiale	20
25	Phases successives de déconvolution et d'unpooling	20
26	Architecture U-Net.....	21
27	Tableau récapitulatif des architectures.....	21
 C Fiche d'activité S9		
1	Fiche d'activité S9.....	48
 D Gantt d'activité S9		
1	Gantt reprenant les étapes de la phase de recherche du S9	49
 E Gantt prévisionnel S10		
1	Gantt prévisionnel de la partie développement du S10	50
 F Gantt d'activité S10		
1	Gantt d'activité de la partie développement du S10	51

1

Introduction

Ce document est le rapport du projet recherche et développement n°9 : « Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR ». Ce projet s'inscrit dans la continuité du programme SOLiDAR, programme qui consiste à utiliser la technologie LiDAR afin de cartographier des zones forestières qui sont difficilement analysables depuis des photographies aériennes classiques, notamment à cause de la végétation.

La maîtrise d'ouvrage est ici représentée par Thierry BROUARD et Jean-Yve RAMEL (enseignants chercheurs à l'Ecole Polytechnique de l'Université de Tours, département informatique), ainsi que par Clément LAPLAIGE, Xavier RODIER et Nathanaël LE VOGUER (chercheurs en archéologie UMR 7324 CITERES - LAT).

La maîtrise d'oeuvre est ici représentée par Valentin MAURICE (étudiant en 5e année à l'Ecole Polytechnique de l'Université de Tours, département informatique), auteur du présent document.

1 Contexte de la réalisation

Le LiDAR (*Light Detection And Ranging*) est une technique de détection de distance fondée sur les propriétés réfléchissantes d'un faisceau de lumière, à l'image d'un sonar pour le son (voir [Figure 1](#)).

Cette technologie présente plusieurs avantages pour la cartographie des variations microtopographiques du sol, elle permet notamment de passer outre la couverture végétale grâce à un multi-échantillonnage dense (10 points / m²). Cette caractéristique permet de faire apparaître des structures invisibles sur le terrain, et sur les images satellites et aériennes classiques.

On peut observer sur [Figure 2](#) la comparaison entre une prise de vue aérienne et une acquisition LiDAR pour le même secteur forestier, de nombreux reliefs apparaissent grâce à la télédétection en plus des voiries actuelles ([Figure 3](#)).

Ce programme nous apporte de nouvelles données sur des territoires de notre patrimoine, Chambord, Boulogne, Russy et Blois, et ouvre la porte à de nouvelles analyses archéologiques.

Le projet présente une grande quantité de données à segmenter et classifier, travail fastidieux réalisé à la main par les archéologues. L'outil de segmentation automatique SkyEye a précédemment été développé pour aider à la décision lors de l'analyse afin d'alléger la charge de travail des experts.

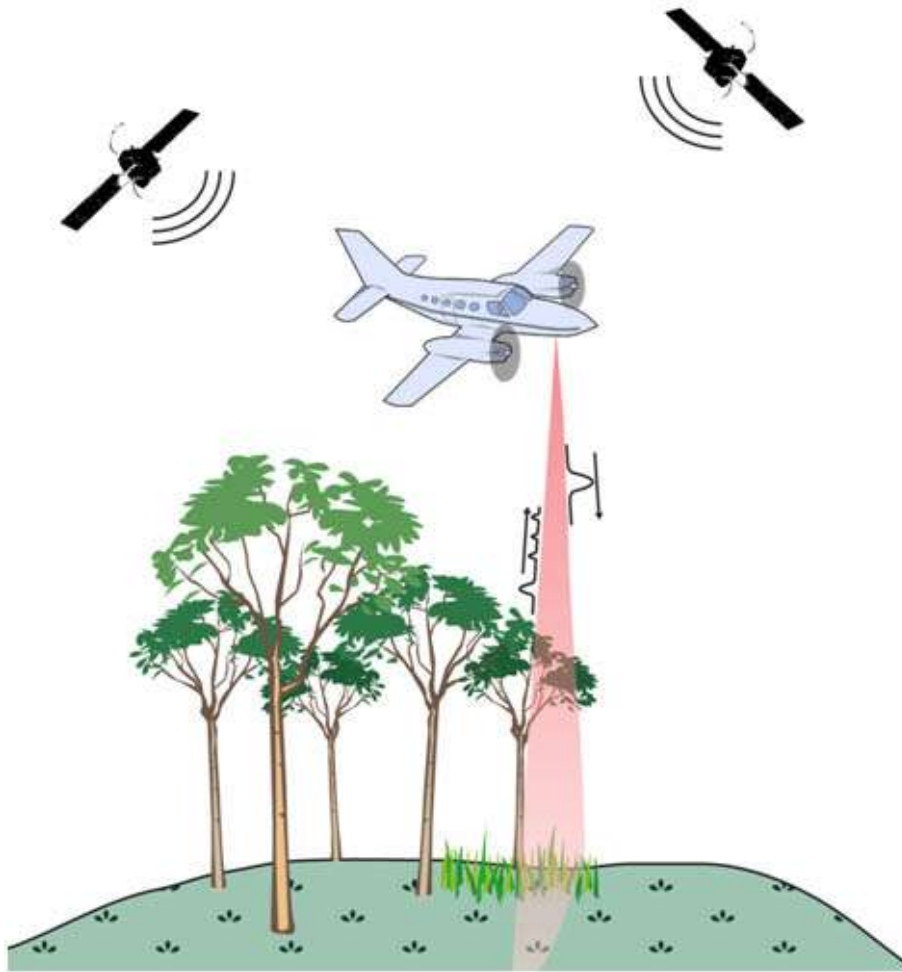


Figure 1 – Schéma du fonctionnement du LiDAR (C. Laplaige, 2012)

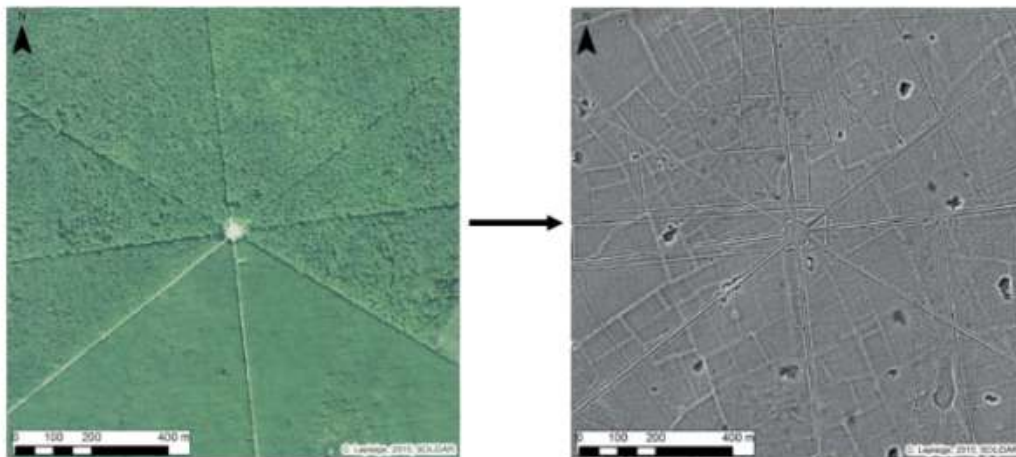


Figure 2 – Vue aérienne / télédétection LiDAR

2 Objectif

L'objectif du projet est l'amélioration de l'outil actuel SkyEye. L'outil est écrit en Python et son utilisation est mono-poste, le programme est portable, en anglais, et sert à la prise de décision dans la classification d'éléments sur une image, on cherchera donc en premier lieu la qualité et

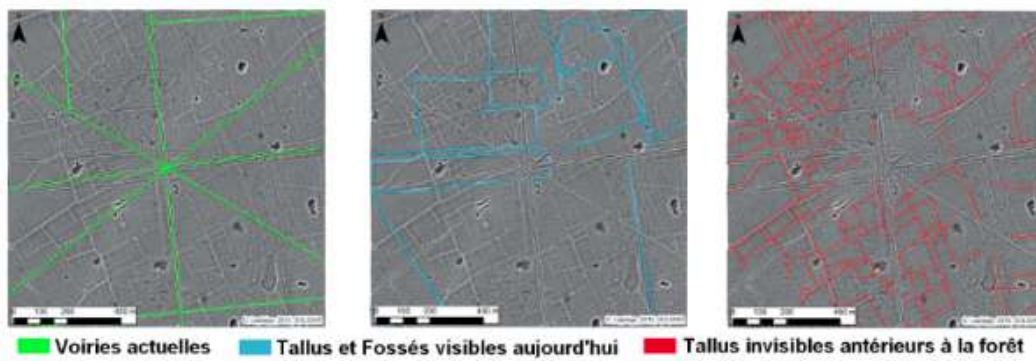


Figure 3 – Structures apparaissant sur les images LiDAR

la fiabilité plutôt que la quantité (vitesse de résolution).

Les performances actuelles de la segmentation ne sont pas si satisfaisantes, la segmentation pixel à pixel est peu lisible (Figure 4) et on dépasse difficilement les 70% de classification correcte ; en ce sens, nous souhaiterions tester une nouvelle approche orientée réseau de neurones profond, là où la classification actuelle est linéaire (Séparateur à Vaste Marge).

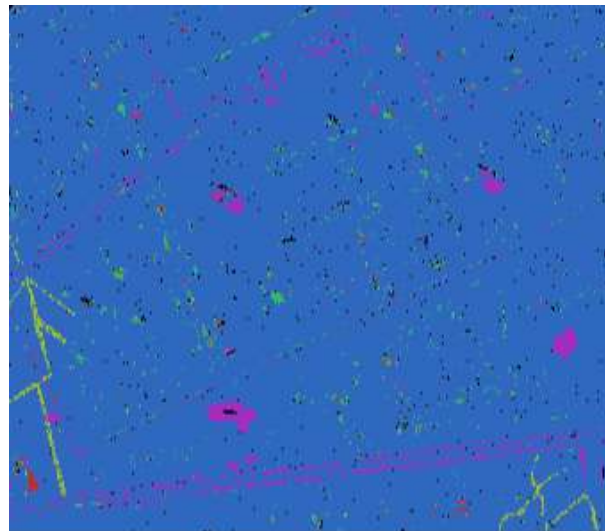


Figure 4 – Exemple d'un résultat de classification par SkyEye

L'objectif est alors de proposer une implémentation CNN adaptée aux données, et son intégration dans l'outil existant.

3 Hypothèses

La réalisation sera fonction de l'implémentation choisie lors de la phase de recherche, voir Chapitre 3. Ce faisant, les entrées et sorties des fonctions pourraient éventuellement varier, même si celles-ci restent globalement les mêmes pour la plupart des réseaux à convolution.

2

Description générale

1 Environnement du projet

L'objectif est d'intégrer la solution développée lors de ce projet dans l'outil SkyEye existant, ce faisant, on pourra d'abord développer une partie ou l'entièreté de la nouvelle implémentation et son interface séparément, puis l'intégrer à l'existant par la suite. On utilisera *PyCharm* pour le développement Python, de plus un certain nombre de bibliothèques sont nécessaires au fonctionnement de l'outil : *appdirs*, *cycler*, *matplotlib*, *numpy*, *olefile*, *opencv-python*, *packaging*, *Pillow*, *pyarsing*, *PyQt5*, *python-dateutil*, *pytz*, *scikit-learn*, *scipy*, *six*.

La partie interface est gérée par *Qt*.

Pour la partie CNN et son implémentation, plusieurs bibliothèques Python pourront être utilisées, notamment *Keras* ou *Pytorch*.

2 Caractéristiques des utilisateurs

Les utilisateurs de l'outil sont des chercheurs en archéologie, on estime qu'ils sont à priori familiers avec l'utilisation d'outils informatiques, et ont seulement quelques notions, voire pas du tout, en reconnaissance d'image et apprentissage automatique / profond.

On voudrait donc proposer une expérience utilisateur simple et intuitive qui ne nécessite pas ou peu de bagages techniques dans un domaine autre que l'archéologie, on voudrait cependant proposer aux utilisateurs ayant ces bagages, un potentiel d'action sur les aspects plus techniques (typiquement pouvoir gérer les paramètres du classifieur).

On intégrera les nouvelles fonctionnalités développées dans l'interface existante en restant cohérent avec l'interface actuelle (voir 3.2 Interfaces homme/machine).

3 Fonctionnalités du système

La classification CNN à développer servira les cas d'utilisation suivants : **Figure 1**.

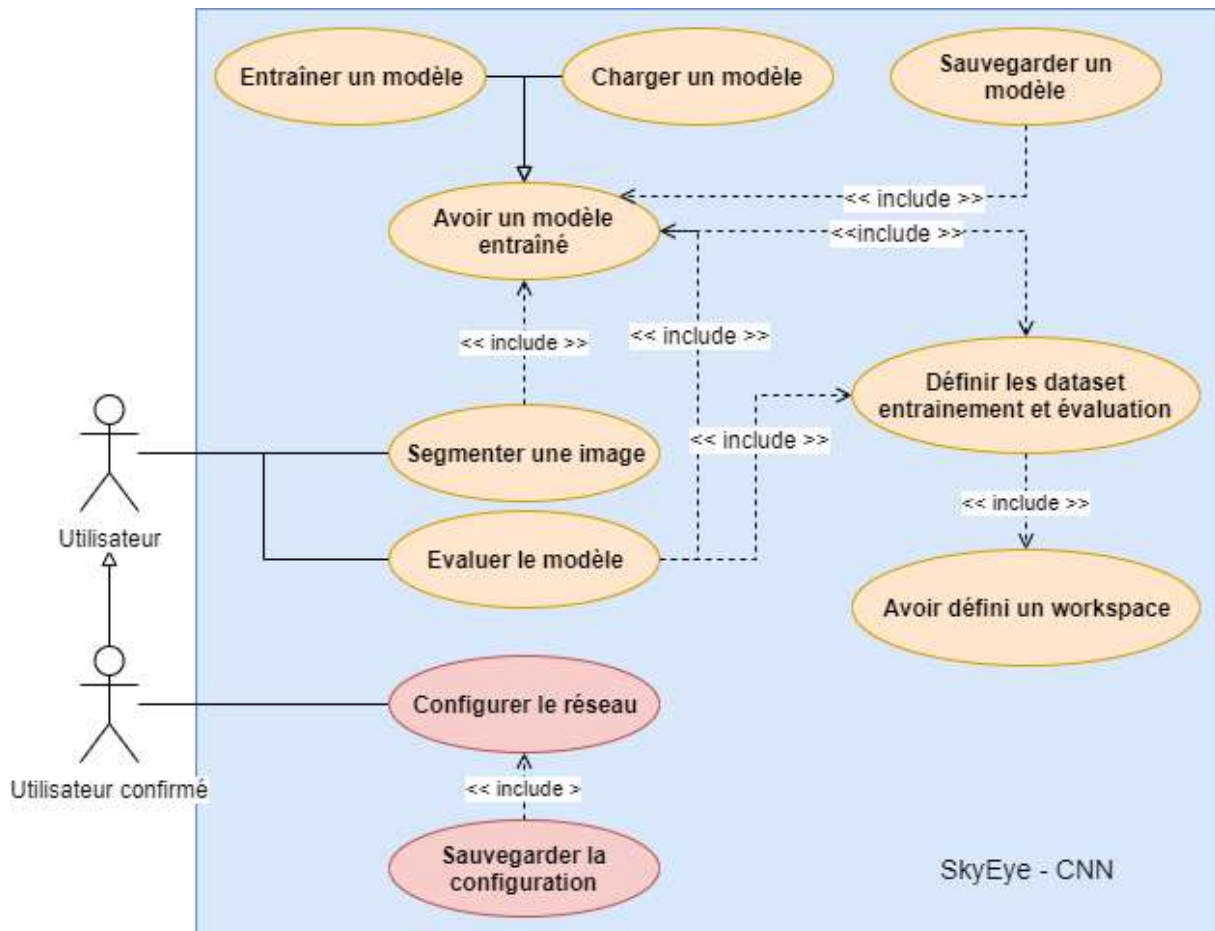


Figure 1 – Cas d'utilisation du classifieur CNN, SkyEye

4 Structure générale du système

Les composants existants de l'outil sont orientés autour de la classification linéaire (extraction des caractéristiques des images, machine learning, etc.). Le seul lien avec l'existant sera à priori le module de gestion du workspace du projet (gère la structure des dossiers d'entrée / sortie), on viendra alors greffer notre solution de la manière suivante : [Figure 2](#).

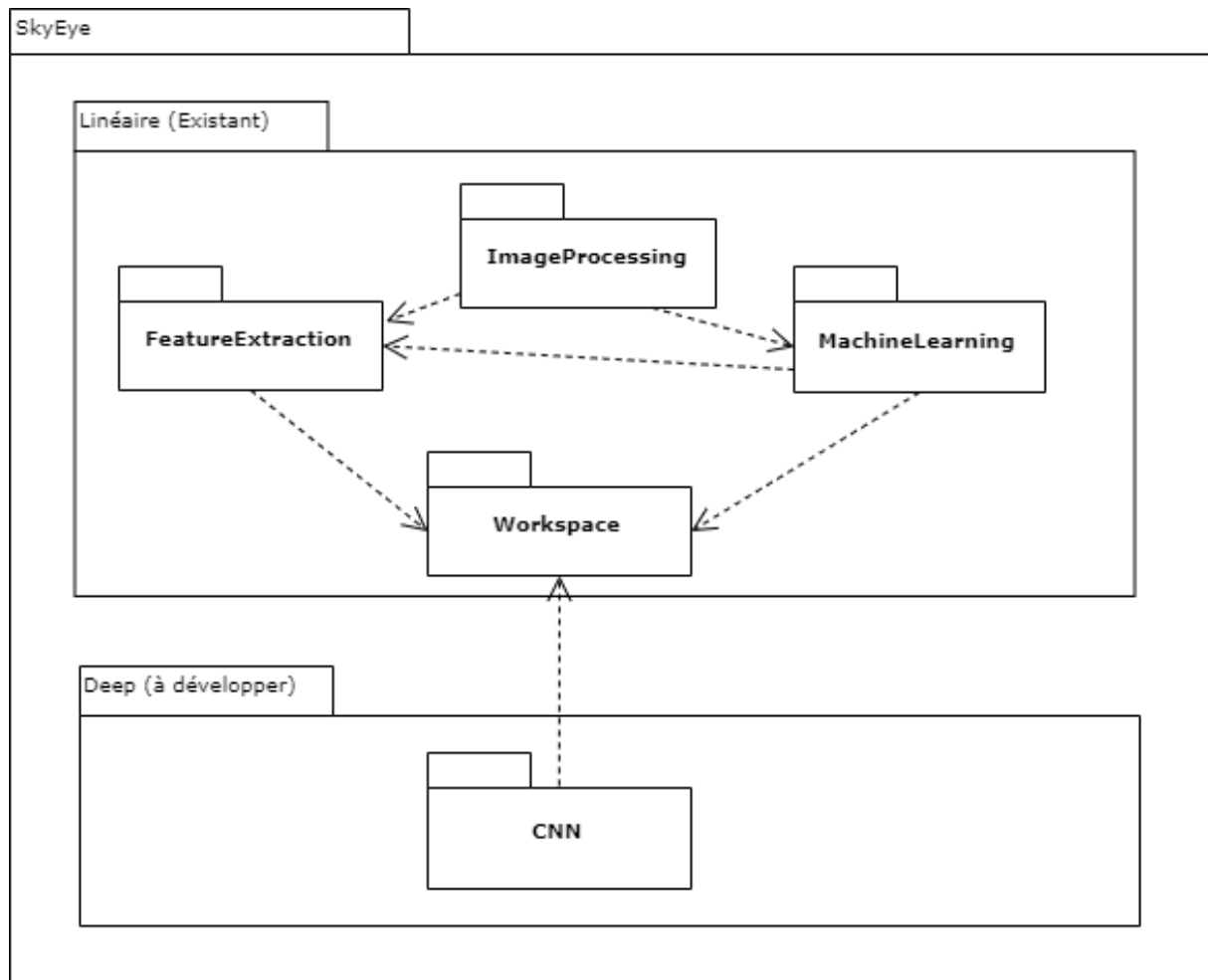


Figure 2 – Diagramme simplifié, SkyEye

3

État de l'art / Veille technologique

1 Réseaux de neurones

Webographie[WWW20][WWW18][WWW24][WWW23][WWW14][WWW11]

Dans le domaine de l'intelligence artificielle, on cherche à mettre en place des systèmes qui permettent de prendre des décisions automatiquement, les réseaux de neurones sont aujourd'hui des structures qui permettent de répondre à ce défi de manière efficace.

1.1 Les neurones

Un réseau de neurones est composé d'un ensemble de neurones, agencés en couches et communiquant les uns avec les autres.

Un neurone artificiel est inspiré d'un neurone biologique (Figure 1), c'est une abstraction de son fonctionnement basique.

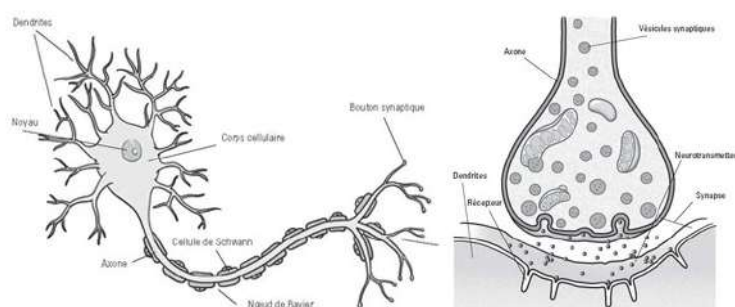


Figure 1 – Schéma d'un neurone biologique

Source: <https://icm-institute.org/fr/actualite/comprendre-le-cerveau-et-son-fonctionnement/>

Un neurone possède plusieurs **entrées** (les dendrites) ainsi qu'une **sortie** (l'axone), chaque neurone est relié au reste du réseau via ces deux éléments, l'axone d'un neurone étant relié à une dendrite de plusieurs autre neurones etc.

On peut donc schématiser un neurone de la manière suivante : Figure 2

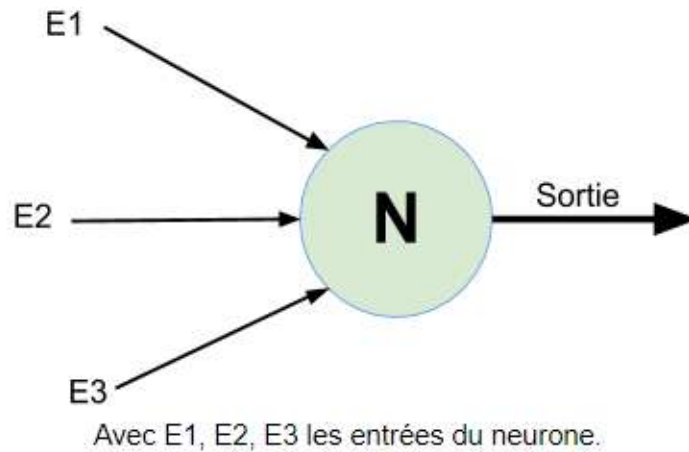


Figure 2 – Schéma d'un neurone abstrait

Les neurones communiquent entre eux via des signaux, on dit qu'un neurone s'active quand il envoie son signal de sortie. Le neurone va s'activer en fonction des signaux qu'il reçoit, et à partir de quelle dendrite il le reçoit, en effet chaque entrée possède son propre coefficient appelé poids qui pourra avoir un effet inhibiteur ou valorisant sur le signal qu'elle transmet.

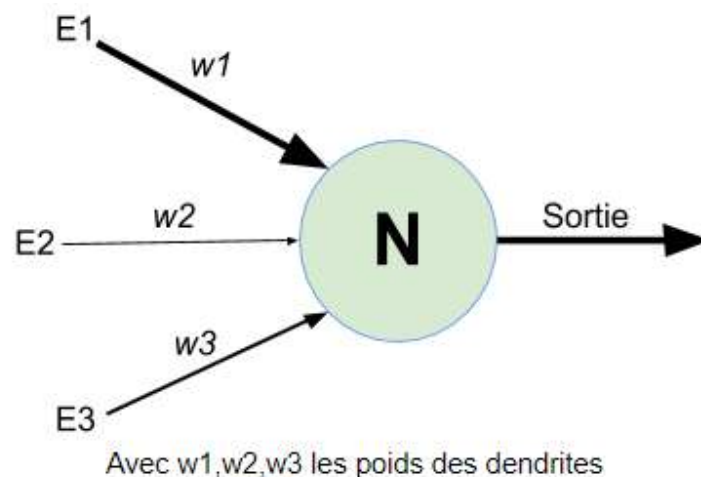


Figure 3 – Schéma d'un neurone abstrait, avec les poids synaptiques

Ainsi on peut ajuster l'importance de chaque entrée dans le processus d'activation du neurone. Ce processus s'opère en deux étapes, il va d'abord agréger les signaux qu'il reçoit en prenant en compte leurs poids associés dans une valeur appelée **potentiel**, puis va comparer le résultats obtenu avec le **seuil** d'activation du neurone, si le potentiel est supérieur au seuil, le neurone s'active et envoie son potentiel aux neurones suivants. Cette deuxième étape peut aussi être une **fonction d'activation** qui va gérer le comportement de transmission du signal, on intègre d'ailleurs la non linéarité via ce genre de fonction dans le système.

Une fonction d'activation connue est la fonction sigmoïde, une fonction qui renvoie une valeur entre 0 et 1, voir Figure 4.

Un neurone peut aussi avoir un **biais**, une sorte de constante que le neurone prend en compte lors de l'agrégation des signaux d'entrée, plus ou moins importante qui permet au système d'accorder plus ou moins d'importance à tel ou tel neurone.

Le neurone est alors capable d'intégrer un ensemble de signaux, de les évaluer puis de transmettre son propre signal.

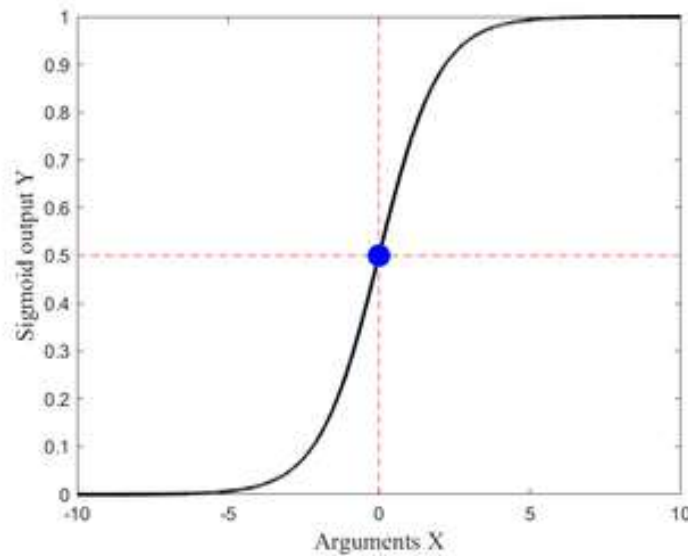


Figure 4 – Courbe d'une fonction sigmoïde dans l'intervalle $[-10;10]$

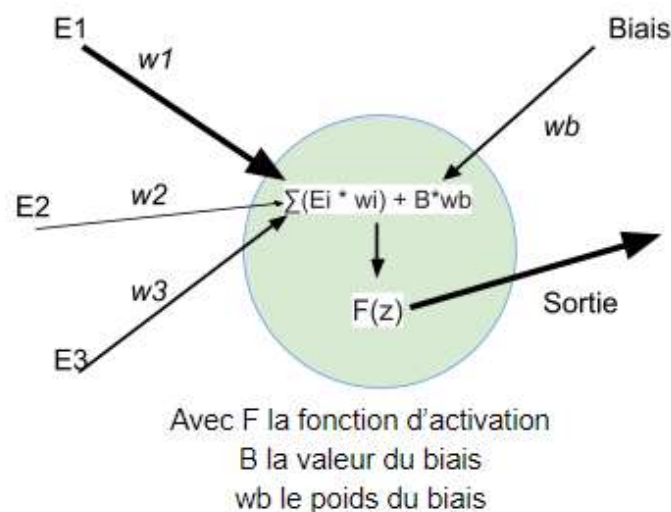


Figure 5 – Schéma résumé d'un neurone abstrait

1.2 Les réseaux de neurones

Un réseau de neurones est organisé en “couches”, et chaque neurone de chaque couche communique avec des neurones de la couche suivante.

On distingue trois types de couches dans un réseau de neurones :

- La couche d'**entrée** : les neurones qui transmettent les données en entrée du réseau
- Les couches **cachées** : l'ensemble des couches intermédiaires,
- La couche de **sortie** : les neurones qui vont présenter les données résultantes du passage dans le réseau des données initiales.

On peut donner du sens à la couche d'entrée qui peut représenter des données concrètes comme des niveaux de gris par exemple, la couche de sortie qui peut être une probabilité d'appartenance à une classe, mais les couches cachées font office de “boîtes noires”. C'est le nombre de couches du réseau qui permet d'augmenter la complexité de celui-ci et le niveau d'abstraction vers lequel

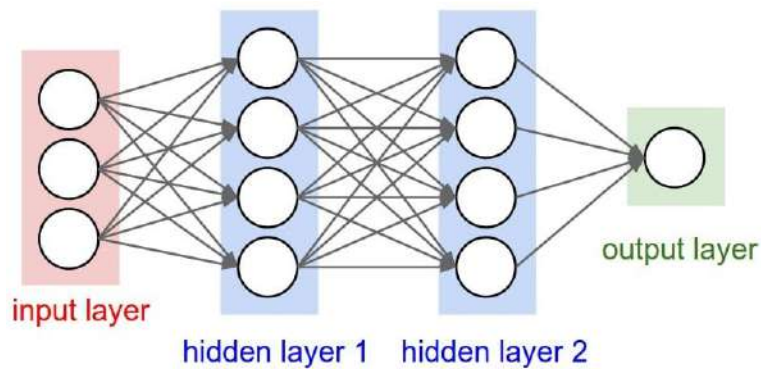


Figure 6 – Organisation d'un réseau en couches

Source:

<https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition>

il peut se diriger.

Les réseaux dans lesquels les signaux ne se déplacent que dans une direction sont qualifiés de **feedforward** à l'inverse des **réseaux récurrents (RNN)** où les signaux peuvent boucler.

1.3 L'entraînement

L'enjeu de la création d'un réseau de neurone est de trouver le bon enchaînement de couches afin de répondre au mieux au problème. Une fois la structure établie, les poids synaptiques sont affectés aléatoirement, et l'enjeu de l'entraînement va être de jouer sur ces coefficients afin d'obtenir les résultats souhaités. Le système est donc composé de la structure du réseau, et de la matrice des poids associée, qui correspond à la "connaissance" du réseau.

Pour ce faire, on entraîne le réseau avec un jeu d'entraînement, un jeu de données d'entrée pour lequel on a le jeu de données de sortie associé. Le but est alors de tester les données d'entrée, et de modifier la matrice de poids en fonction du résultat.

Il existe plusieurs méthodes de correction des poids, notamment une technique appelée **back-propagation** (ou *Rétropropagation du gradient*) qui consiste à analyser la quantité d'erreur et ajuster les coefficients en conséquence.

La décision gagne en qualité au fur et à mesure des entraînements, et de la précision des poids synaptiques. Il faut tout de même faire attention au surentraînement, comme avec le cerveau humain, si on reconnaît "par coeur" un élément dans un contexte précis, on perd notre capacité à reconnaître cet élément dans un contexte quelconque. Pour contrecarrer ce phénomène, il suffit de séparer le jeu d'entraînement en deux parties, la première pour l'entraînement en lui-même, la seconde pour l'évaluation de la qualité de l'entraînement.

2 CNN

Webographie : [WWW17][WWW10][WWW22][WWW13][WWW7]

Les réseaux neuronaux convolutifs (CNN : Convolutional Neural Networks) sont des réseaux inspirés du fonctionnement du cortex visuel des animaux, où des caractéristiques de la moins abstraite à la plus abstraite sont reconnues successivement au travers de régions¹.

1. Voir https://fr.wikipedia.org/wiki/Cortex_visuel

Les CNN sont des réseaux “feed-forward” dont les couches forment deux parties, la première partie, commençant par la couche observable, est composée d'opérations d'acquisition, de transformation des caractéristiques et de compression des données, de manière successive. La seconde partie est un réseau de neurone comme vu précédemment, plus classique qui va être chargé de l'abstraction, de la décision, à partir des données extraites par la première partie.

2.1 Première partie du réseau

La première partie du réseau est chargée de préparer une grande quantité de données (pour une image, des millions de pixels sur les trois couches RVB) en pré-analysant les caractéristiques significatives locales du jeu de donnée.

Pour réaliser cette extraction, on utilise des opérations de convolution, on applique une opération de somme pondérée pour chaque pixel de l'image et son voisinage (**champs récepteur**) à l'aide d'une matrice de poids appelée noyau de convolution.

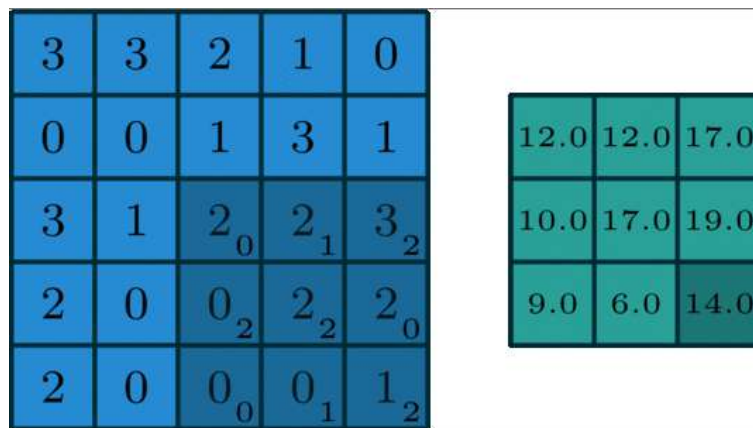


Figure 7 – Fonctionnement d'une convolution

Source: <https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>

En appliquant différents noyaux de convolution pour chaque champs récepteur, on obtient différentes **features map** qui vont décrire l'image d'origine en fonction des noyaux de convolution utilisés.

Afin d'introduire la non-linéarité dans le réseau, on applique une fonction d'activation comme celles vues dans la première partie 1. Réseaux de neurones de type **ReLU** (Rectified Linear Unit) où on conserve les valeurs positives et on remplace les valeurs négatives par zéro.

Les features map peuvent être réduites via une opération de **pooling** où on va diminuer la dimension des données en minimisant la perte d'information, cette opération peut être assimilée à une sorte de convolution non linéaire, voir Figure 8.

L'enchaînement de ces couches forme la partie **feature extractor** du réseau. Pour pouvoir raccorder ce réseau à la seconde partie, on passe par une étape d'aplatissement (**flattening**) où on passe d'un ensemble de feature map (matrices) à un unique vecteur qui sera l'entrée des couches complètement connectées (**fully connected**), où chaque neurone est connecté à toutes les sorties de la couche précédente.

La seconde partie est un réseau plus classique chargé de la classification.

On a ici (Figure 9) l'enchaînement de deux **modules convolutifs** constitués d'une couche de convolution, une couche ReLu, et d'une couche de Pooling.

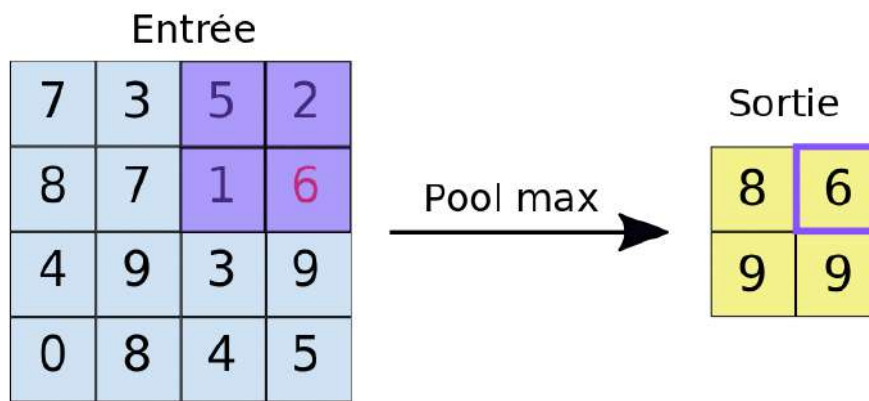


Figure 8 – Fonctionnement d'une opération max-pool

Source: <https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks>

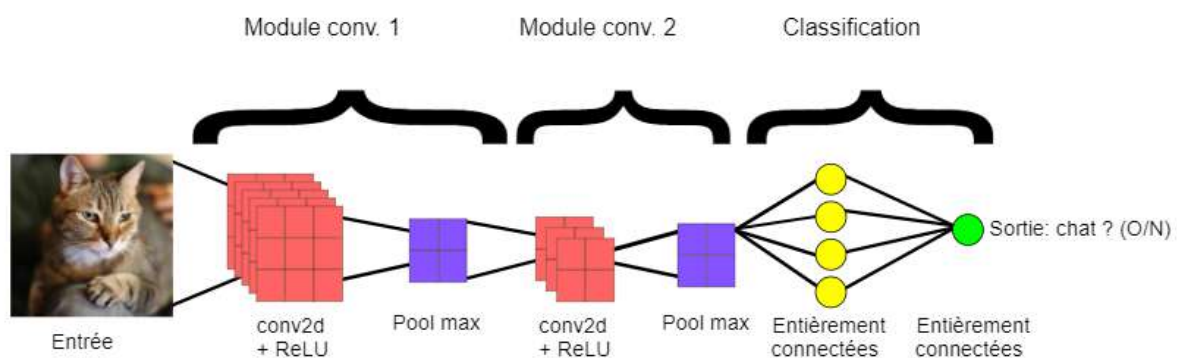


Figure 9 – Architecture d'un réseau CNN

Source: <https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks>

L'enjeu de la création de réseau type CNN est de trouver une configuration efficace de ces couches afin de coller au mieux au modèle de données avec lequel on travaille. Il existe déjà différentes architectures qui sont plus où moins efficaces selon les données, par exemple ici une comparaison entre différents modèles pour le dataset ImageNet² :

2.2 Paramètres initiaux et entraînement

Les paramètres initiaux (hyperparamètres) sont les caractéristiques du réseau qu'il faut définir lors de sa création, pour les couches de convolutions ils y a :

- Le nombre de filtres
- La taille des filtres
- Le pas à utiliser quand on translate le noyau lors de la convolution
- Le zero-padding, un nombre de pixel à ajouter aux marges de l'image afin de l'agrandir artificiellement, on peut ainsi appliquer la convolution aux pixels de bord de l'image

Pour les couches de pooling on a :

- La taille des cellules à réduire

2. Voir : <http://www.image-net.org/>

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

Figure 10 – Performance de différents réseau sur le jeu de donnée ImageNet

Source: <https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d>

— Un pas de séparation entre les cellules à réduire

Lors de l'entraînement, les valeurs ajustées seront les filtres de caractéristique (les noyaux de convolution) ainsi que les poids synaptiques des couches complètement connectées.

2.3 Transfert learning

Le fait qu'un CNN soit séparé en une partie extraction et une partie classification permet d'intervenir l'une et l'autre selon les besoins, et notamment d'utiliser une partie extraction déjà entraînée avec une partie classification personnalisée selon l'objectif.

Cette démarche est intéressante à plusieurs niveaux, on passe une étape fastidieuse de création de cette partie du réseau, mais on évite surtout la phase d'entraînement très coûteuse. On peut ainsi partir d'une base qui s'est entraînée à extraire les caractéristiques les plus intéressantes d'un jeu d'image, et en déduire des notions abstraites pour lesquelles le réseau initial n'était pas entraîné en apportant nos propres couches complètement connectées. L'entraînement du réseau global sera alors beaucoup plus efficace et pourra fonctionner avec un jeu de donnée bien plus réduit.

Pour un transfert learning efficace, il est préférable de partir d'un modèle entraîné sur des images similaires à notre jeu de donnée personnalisé. On peut trouver des modèles entraînés sur internet, et notamment directement depuis des bibliothèques Python comme Keras, qui propose notamment des modèles basés sur ImageNet comme ResNet.

Pour les images utilisées par SkyEye, on pourrait tester les modèles ImageNet ou se diriger vers des modèles entraînés sur des images satellites³, ou des représentations topologiques.

3 R-CNN

Webographie[WWW5][WWW4]

Les réseaux du type R-CNN (Region-CNN) sont des réseaux CNN qu'on veut utiliser pour classifier des régions d'une image et non plus uniquement la labelliser. Le principe est simple,

3. A voir : <https://github.com/robmarkcole/satellite-image-deep-learning>

on découpe l'image en régions d'intérêt qu'on transmet ensuite à un CNN classique. On effectue donc globalement plusieurs classifications CNN à partir d'une seule image.

Il existe différents algorithmes de proposition de région à partir d'une image, comme des algorithmes de recherche sélective⁴.

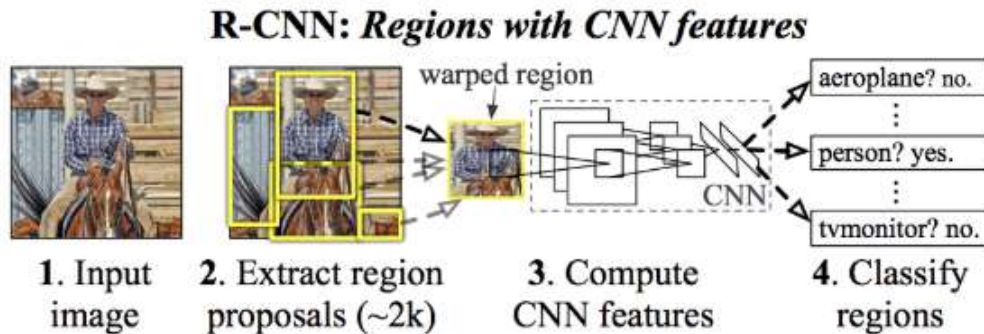


Figure 11 – Fonctionnement d'un R-CNN

Source: <https://arxiv.org/pdf/1311.2524.pdf>

Cette approche pose quelques problèmes, l'algorithme de proposition de région ne fait pas partie du réseau de neurones, et ce faisant, ne fait pas partie de l'entraînement. De plus, ces algorithmes proposent un grand nombre de régions (plusieurs centaines voir milliers), le temps de traitement est donc relativement conséquent, et ne peut être utilisé dans des situations de traitement en temps réel.

4 Fast R-CNN

Webographie[WWW5][WWW3]

Le Fast R-CNN a été pensé pour pallier aux problèmes de performance du R-CNN simple. A la place de donner 2000 images régions en entrée du réseau R-CNN, on va utiliser l'image d'origine pour créer les features maps grâce au CNN, puis on va en extraire ces 2000 régions d'intérêt et les redimensionner en une taille fixe. Cette couche de pooling est appelée ROI pooling (Region Of Interest), puis on reprend le fonctionnement classique d'un CNN avec une phase de flattening et de fully connected, pour proposer en sortie les classifications et les boîtes de détection associées.

Cette approche est bien plus rapide dans le sens où on ne manipule pas les 2000 régions d'intérêt d'un bout à l'autre du processus

5 Faster R-CNN

Le faster R-CNN vient améliorer le système du Fast R-CNN, et vient remplacer l'algorithme de sélection des régions d'intérêt qui est coûteux en ressources, par un autre réseau de neurones appelé Region Proposal Network (RPN) chargé de la sélection des régions, voir Figure 13.

En terme de performances (temps de test), chaque version est une amélioration de la précédente, et Faster R-CNN est la plus optimale pour ce genre de réseau (voir Figure 14.

4. Voir : <https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/>

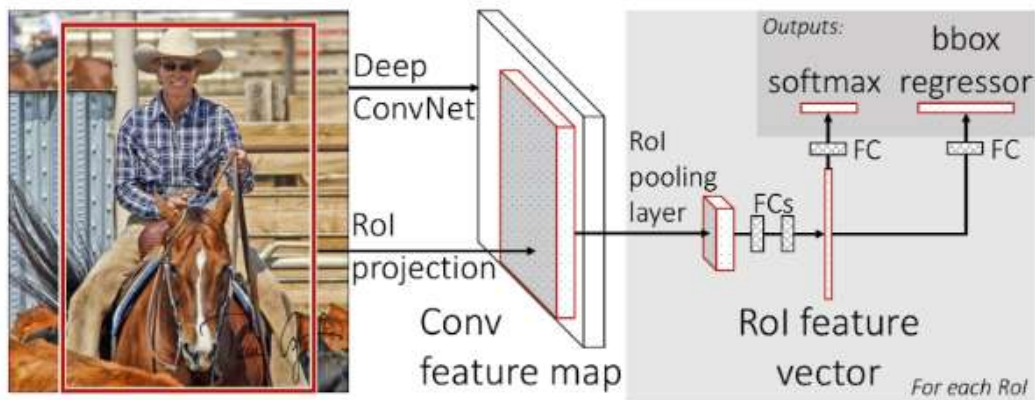


Figure 12 – Fonctionnement d'un Fast R-CNN

Source: <https://arxiv.org/pdf/1311.2524.pdf>

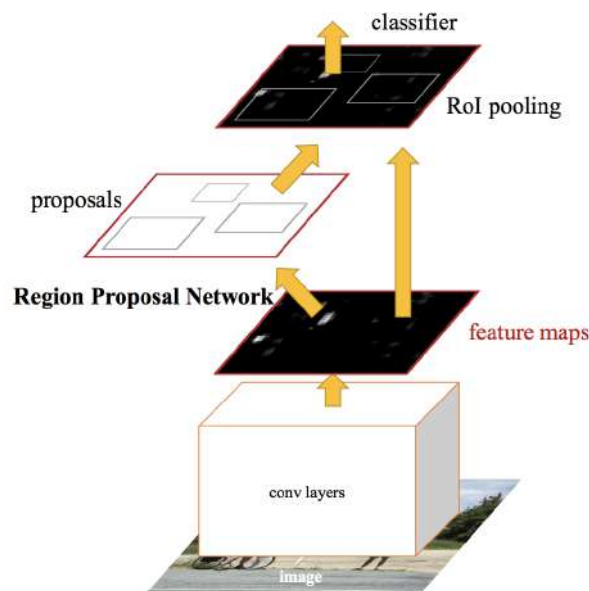


Figure 13 – Fonctionnement d'un Faster R-CNN

Source: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

6 Mask R-CNN

Webographie[WWW19][WWW1][WWW6]

Le Mask R-CNN est basé sur Faster R-CNN et vient ajouter à la détection des bounding box des instances un masque de région, exemple : Figure 15.

Pour se faire on ajoute un troisième réseau complètement connecté après l'étape de sélection des régions d'intérêt (RoI pooling) pour les "affiner".

On peut comparer les évolutions successives entre ces différentes architectures : Figure 16

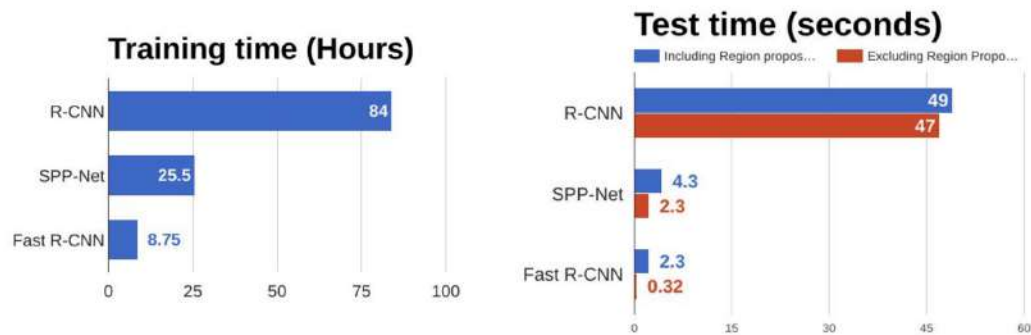


Figure 14 – Comparaison des performances des architectures basée R-CNN

Source: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>



Figure 15 – Exemple d'une segmentation par Mask R-CNN

Source: <https://towardsdatascience.com/mask-r-cnn-for-ship-detection-segmentation-a1108b5a083>

7 YOLO

Webographie[WWW2][WWW8][WWW15]

YOLO (You Only Look Once) est un algorithme de détection d'objet utilisé dans les détections temps réel pour sa rapidité. Il est architecturé en un unique CNN et l'image n'est "regardée" qu'une seule fois au cours du processus.

Le réseau va établir une grille sur l'image, et pour chaque case, va essayer de prédire un objet à l'aide de N bounding box, et va accorder un score de confiance et pour chaque classe une probabilité que l'objet en soit une instance.

Au travers du réseau, les meilleures bounding box vont être sélectionnées via un score de confiance de classe, voir Figure 17

YOLO est actuellement l'architecture la plus performante en terme de vitesse de calcul et trouve des applications dans la détection temps réel, notamment vidéo ⁵.

5. Voir https://www.youtube.com/watch?v=_kxX09i4fds

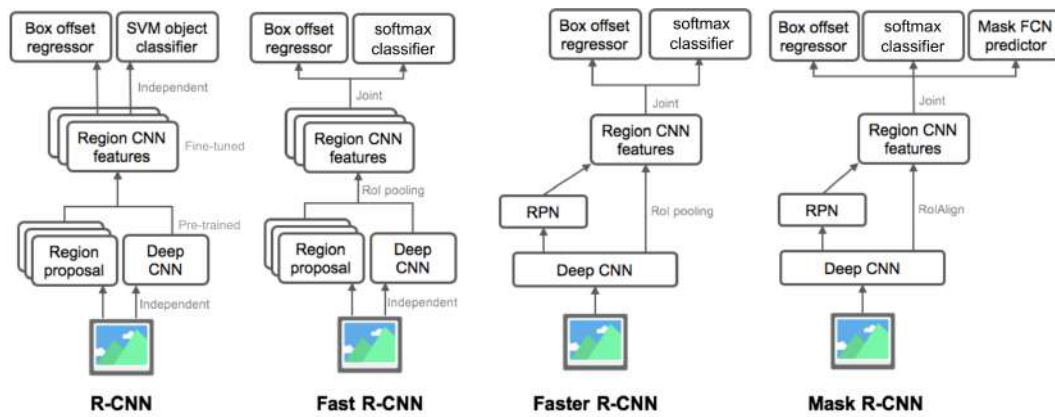


Figure 16 – Comparaison des architectures de type R-CNN

Source: <https://lilianweng.github.io/lil-log/assets/images/rcnn-family-summary.png>

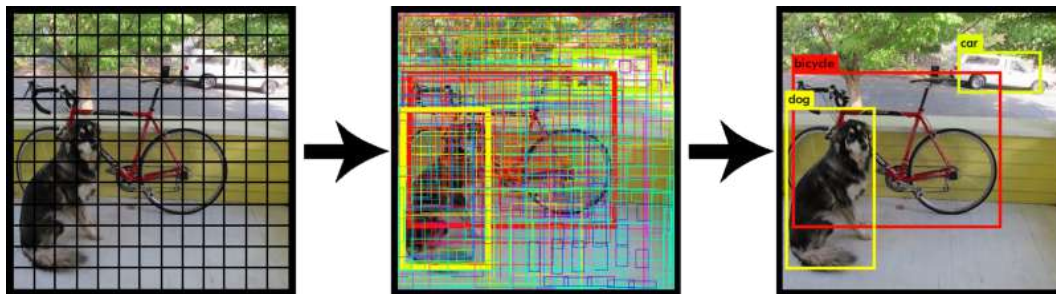


Figure 17 – Grandes étapes d'une analyse YOLO

Source: <https://www.nontee.com/en/yolo-real-time-object-detection/>

8 FCN

Webographie[WWW21][WWW9]

Fully-Convolutional Network (réseaux complètement convolutionnel) est une architecture utilisant des couches de convolution de bout en bout (Figure 19), là où un CNN classique se terminera par des couches complètement connectées (Figure 18).

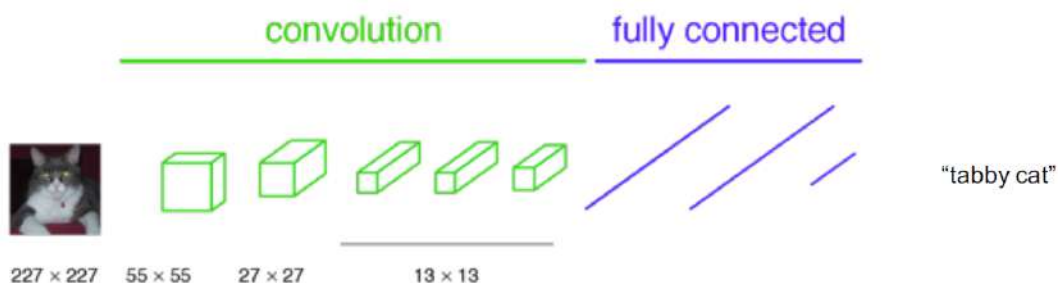


Figure 18 – Structure d'un CNN classique

Source: <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>

Cette architecture permet de classifier pixel à pixel l'image, au prix d'une résolution réduite, due aux phases successives de pooling. On effectue donc en fin de réseau des phases de "dé-

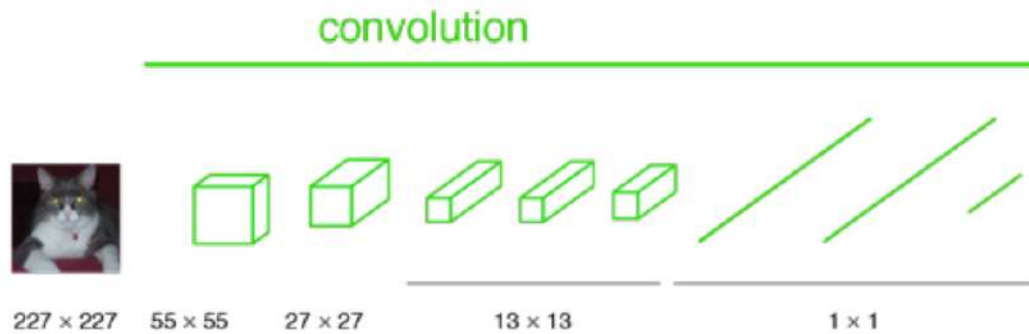


Figure 19 – Structure d'un FCN

Source: <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>

convolution" (**up convolution**, Figure 20) permettant de reconstituer l'image dans sa résolution d'origine.

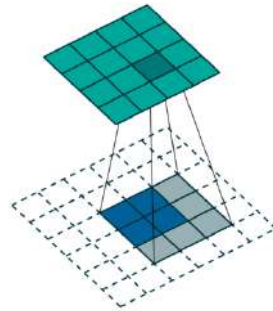


Figure 20 – Exemple d'une déconvolution ou up-convolution

Source: <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>

On perd cependant en qualité de segmentation en reconstituant, on peut alors fusionner (sommer, Figure 21) les prédictions à différents niveaux de convolution pour obtenir un résultat plus net.

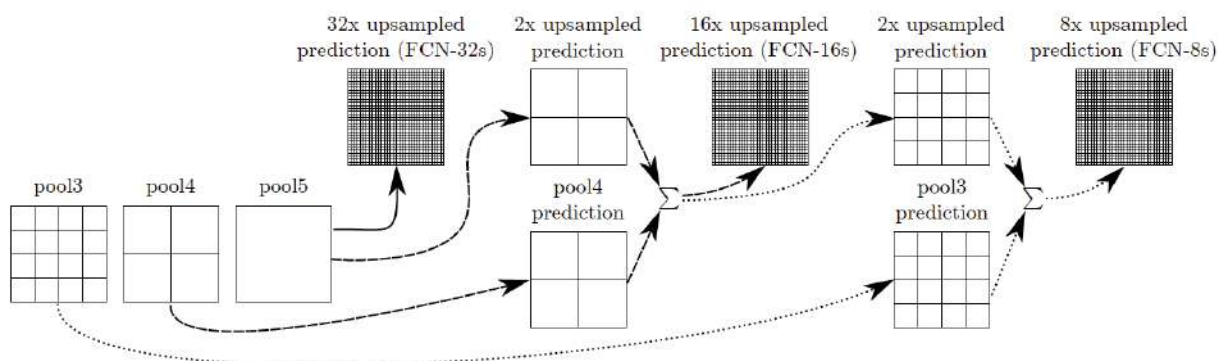


Figure 21 – Exemple de fusion (sommes) des prédictions dans un FCN

Source: <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>

On obtient ainsi un résultat plus net en additionnant les prédictions plutôt qu'en ne gardant que la reconstitution de plus bas niveau, voir Figure 22.

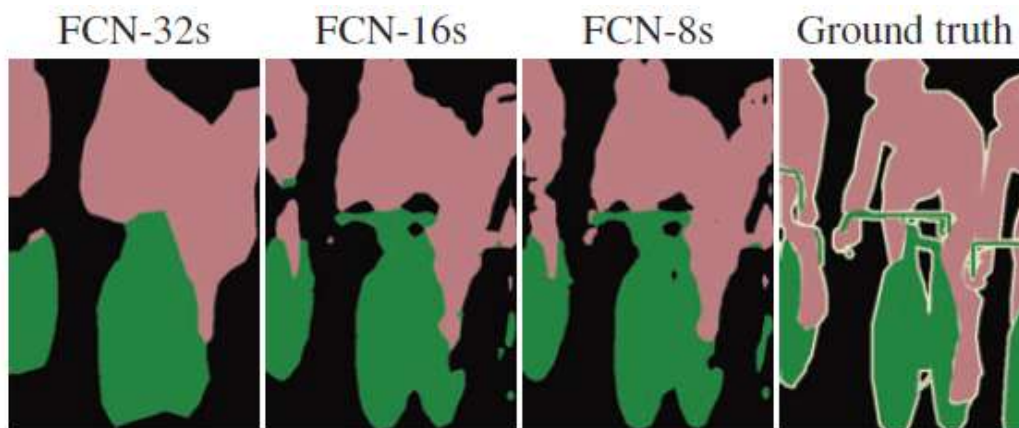


Figure 22 – Prédications FCN après fusions à différents niveaux, et vérité terrain

Source: <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>

9 DeconvNet

DeconvNet est un FCN modifié pour pallier au problème de perte d'information spatiale lors de la phase de déconvolution. Pour cela est utilisé une première partie de réseau CNN classique comme dans un FCN, puis un réseau dit de "déconvolution", enchaînant les phases d'**unpooling** (Figure 24) et utilisant les informations spatiales utilisées lors des déconvolutions (Figure 20) précédentes, voir Figure 23.

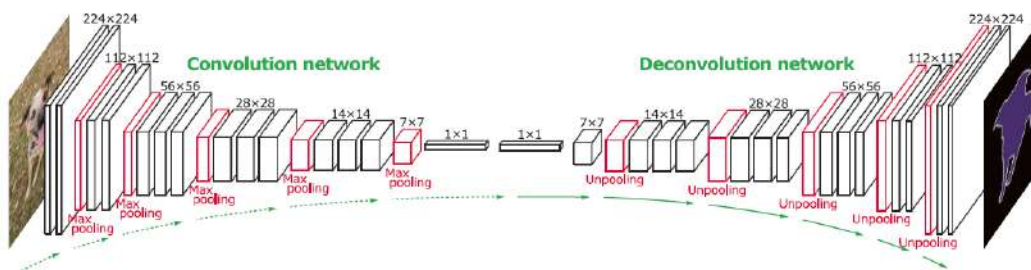


Figure 23 – Structure d'un DeconvNet

Source: <https://towardsdatascience.com/review-deconvnet-unpooling-layer-semantic-segmentation-55cf8a6e380e>

On arrive ainsi à reconstituer l'image dans sa dimension d'origine en conservant une qualité spatiale supérieure à un FCN classique.

10 U-Net

Webographie[WWW12][WWW25][WWW16]

U-Net est aussi une variation de l'architecture FCN. Le réseau a été conçu à l'université de Fribourg en Allemagne, pour la segmentation d'images biomédicales en disposant de peu de données d'entraînement. Cette architecture (Figure 26) est constituée d'une phase de compression et d'une phase de décompression.

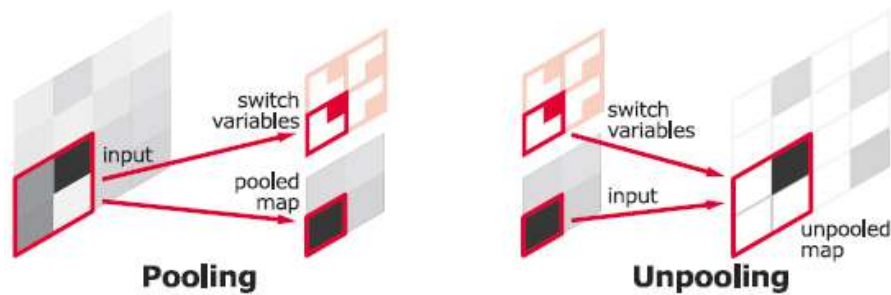


Figure 24 – Phase d'unpooling, avec conservation de l'information spatiale

Source: <https://towardsdatascience.com/review-deconvnet-unpooling-layer-semantic-segmentation-55cf8a6e380e>

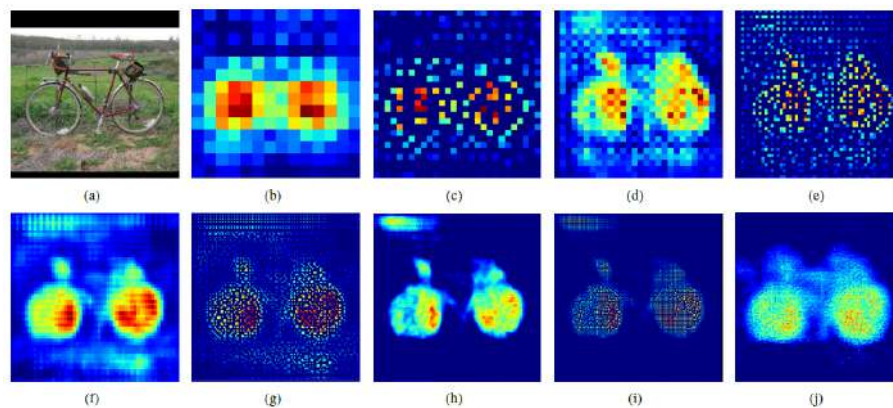


Figure 25 – Phases successives de déconvolution et d'unpooling

Source: <https://towardsdatascience.com/review-deconvnet-unpooling-layer-semantic-segmentation-55cf8a6e380e>

Cette structure fait office de mixe entre une architecture deconvNet (Section 9) et un FCN classique (Section 8), on retrouve des couches de déconvolutions successives concaténées (au lieu d'une somme) avec les résultats des couches de convolutions associées.

Le réseau U-Net fonctionne bien avec un faible nombre d'images d'entraînement, en effet on peut appliquer des opérations de distorsions sur celles-ci pour en augmenter artificiellement le nombre, tout en gardant un entraînement efficace.

11 Technologies et implémentations

11.1 Keras

Keras est une bibliothèque python haut niveau axée sur le prototypage rapide, permet de facilement créer des réseaux personnalisés, possède un certain nombre d'architectures pré-entraînées sur ImageNet : *Xception*, *VGG16*, *VGG19*, *ResNet*, *ResNetV2*, *InceptionV3*, *InceptionResNetV2*, *MobileNet*, *MobileNetV2*, *DenseNet*, *NASNet*.

11.2 Pytorch

PyTorch est une bibliothèque plus bas niveau, permet une plus grande flexibilité mais est à priori plus compliquée à utiliser, possède des modèles pré entraînés :

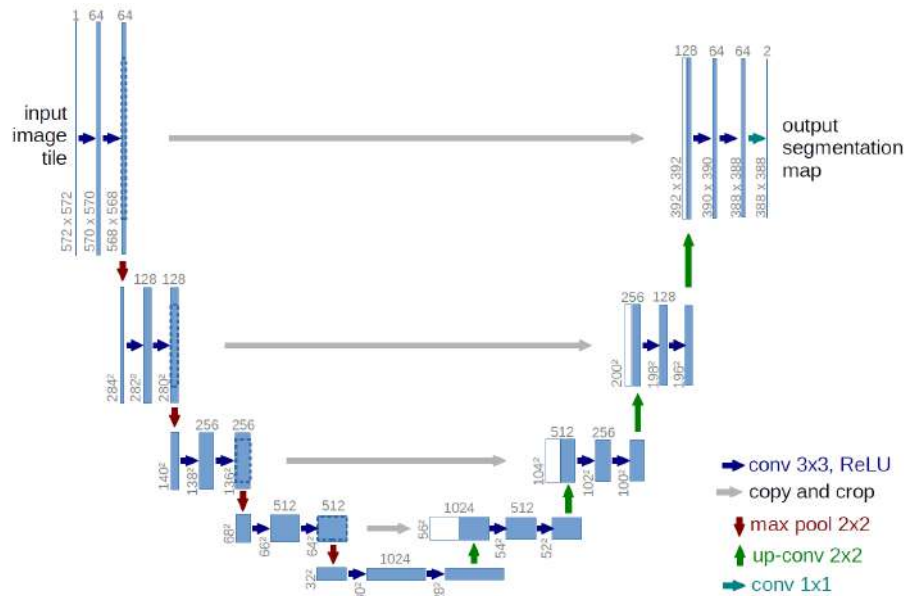


Figure 26 – Architecture U-Net

Source: <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>

- Classification : AlexNet, VGG, ResNet, SqueezeNet, DenseNet, Inception v3, GoogLeNet, ShuffleNet v2, MobileNet v2, ResNeXt, Wide ResNet, MNASNet
- Segmentation : FCN ResNet101, DeepLabV3 ResNet101
- Détection d'objets / instances : Faster R-CNN ResNet-50 FPN, Mask R-CNN ResNet-50 FPN

11.3 Récapitulatif

On peut comparer les différentes architectures de CNN en fonction de leurs objectifs et des implémentations disponibles, voir Figure 27

Algo	Labellisation	Détection d'objet	Segmentation	Techno, implémentations disponibles
CNN	O	X	X	Keras / PyTorch
Faster R-CNN	O	O	X	Keras / Pytorch
Mask R-CNN	O	O	O	Keras / Pytorch
YOLO	O	O	X	Keras / Pytorch
FCN	O	O	O	Keras / Pytorch
deconvNet	O	O	O	Keras / Pytorch
U-Net	O	O	O	MatLab / Lasagne (Theano) / Keras

Figure 27 – Tableau récapitulatif des architectures

12 Conclusion de l'état de l'art

Dans le cadre du projet SkyEye, le but est de proposer une segmentation générale de l'image, tout en ayant un dataset d'entraînement très réduit. On se dirigerait alors vers une architecture

nécessitant peu de données d'entraînement et / ou ayant un modèle pré entraîné. Dans le cas du transfert learning, il serait pertinent de trouver un modèle entraîné sur des images similaires à celles traitées par SkyEye, on peut penser à des images satellites ou médicales dans lesquelles les classes présentent une forte variabilité dans leurs formes, comme c'est le cas avec des chemins, routes, etc.

On continuerait d'utiliser Python, qui a l'avantage de proposer un grand nombre d'implémentations de réseaux de neurones via Keras, Pytorch ou autre. On veut par la suite tester quelle architecture / implémentation semble la plus pertinente et la plus efficace en terme de qualité de segmentation, en considérant tout de même la charge d'entraînement. Les réseaux de type DeconvNet ([Section 9](#)) ou U-Net([Section 10](#)) semblent être des pistes intéressantes en ce sens.

4

Mise en oeuvre

1 Outils utilisés

Les outils suivants ont été utilisés lors de ce projet :

- *PyCharm* pour le développement Python
- *Github* pour le versioning, la distribution et les manuels utilisateurs et développeurs
- *TimePerformance* pour la gestion de projet
- *Google Drive* pour le stockage et partage de documents
- *Balsamiq* pour la réalisation des mockups UI
- *Qt et Qt Designer* pour la réalisation de l'UI

Pour les bibliothèques et la version de Python utilisée, voir la fiche de configuration ([Annexe B](#)), ou le guide développeur (LIEN VERS ANNEXE).

2 Déviations par rapport aux spécifications

Lors de la rédaction du cahier de spécification, plusieurs points étaient en suspens et allaient dépendre des résultats des *Proofs Of Concept* du début de la phase de développement, notamment la séparation ou non de l'outil initial, l'interface utilisateur, le nombre exact de fonctions, etc..

Lors des tests pour trouver des implémentations de modèles CNN avec Keras, le dépôt Github¹ de *Divam Gupta* m'a semblé être une bonne base pour ce projet, en effet il implémente une vingtaine de modèles dont ceux repérés lors de la phase d'état de l'art. J'ai donc fait le choix de créer un projet forké depuis le sien, afin d'utiliser ses modèles et proposer les fonctionnalités attendues pour ce projet.

En ce qui concerne la séparation avec l'outil initial, il est finalement apparu que les deux solutions n'auraient rien en commun, pas même la gestion du workspace qui est très différente pour l'outil développé cette année. La solution ici présentée est donc totalement originale.

1. Voir <https://github.com/divamgupta/image-segmentation-keras>

3 Tests et qualimétrie

La partie *modèles* des sources présentait déjà une série de tests unitaires qui confirmait le fonctionnement des créations et utilisations des modèles implémentés. J'ai ensuite implémenté des tests de bout en bout avec l'utilitaire PyTest (voir [Annexe G](#)), en vérifiant le plus d'informations possible afin de valider le fonctionnement de chaque fonctionnalité (création des bons dossiers/fichiers, génération des images à la bonne taille, avec les bonnes valeurs de gris, etc.). Ainsi, ces tests ont petit à petit servi de tests de non régression lors de l'ajout ou la modification des ces fonctionnalités.

En ce qui concerne la qualimétrie, l'outil PyLint a été utilisé afin d'effectuer une analyse statique du code, valider les bons usages des normes et conventions Python *PEP 8* (voir [Annexe G](#)).

En terme de performances de segmentation des modèles, les résultats obtenus lors des tests ont été au mieux passables, mais attendus au vu de la faible quantité d'exemples utilisés pour entraîner les modèles (quelques dizaines d'images hétérogènes), on constate cependant moins de faux positifs qu'avec les méthodes de segmentation utilisées lors des précédents projets. Des indicateurs de performances métiers ont ici été utilisés, comme l'indice d'Intersection Over Union ² (UI) ou la matrice de confusion ³. Ces résultats restent néanmoins très encourageants, et semblent prometteurs si utilisation sur une base de données plus complète (centaines/milliers d'images homogènes). Les temps d'exécution restent raisonnables pour ces types d'opération et seront évidemment plus importants avec l'augmentation du volume de données, cependant les opérations très coûteuses comme l'entraînement des modèles (plusieurs heures/jours), peuvent être arrêtées et reprises depuis des fichiers jalons générés lors de ces traitements.

2. Voir <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>

3. Voir https://fr.wikipedia.org/wiki/Matrice_de_confusion

5

Bilan et conclusion

Lors de l'écriture de ce paragraphe, la solution a été présentée au client lors d'une réunion de démonstration, et lui est maintenant parvenue, j'attends des retours à l'utilisation afin de corriger d'éventuels bugs ou problèmes d'utilisation/d'installation.

En ce qui concerne le planning, je suis très satisfait car le projet a très peu dévié de la prévision du semestre 10 ([Annexe E](#) et [Annexe F](#)), seulement une semaine de retard lors du premier *PoC*, mais une semaine rattrapée lors des deux *PoC* suivants. Finalement toutes les fonctionnalités prévues initialement ont été implémentées.

Ce qu'il resterait à faire :

- Revoir la fonctionnalité d'augmentation afin d'effectuer un découpage plutôt qu'un redimensionnement
- Améliorer la gestion des erreurs afin d'éviter des plantages secs du programme
- Implémenter des options de qualité de vie en fonction des retours utilisateurs

1 Bilan personnel

Ce projet a été pour moi un moyen de découvrir le domaine de l'intelligence artificielle profonde qui m'était alors inconnu avant cette année, et de m'intégrer au cœur de la réalisation d'un projet de plus grande envergure que nos précédentes réalisations à Polytech. J'ai aussi beaucoup apprécié de découvrir la partie métier des clients lors des phases initiales d'échange et d'expression du besoin, ces étapes restent mes favorites lors des projets que je mène.

En ce qui concerne la gestion de projet, même si elle a pu paraître légère et informelle, elle m'a semblé être adaptée à ce type de projet qui reste relativement modeste en terme de planning et d'intervenants. J'aurais pu mettre en place une méthodologie plus lourde, des réunions hebdomadaires etc. mais cela ne m'a pas paru nécessaire ici, et le projet et sa gestion restent à ce jour un succès pour moi.

Les documentations et manuels fournis permettent à mon sens une reprise efficace du projet, que ce soit pour l'améliorer ou pour s'en inspirer.

Enfin, je remercie mon encadrant Thierry BROUARD pour sa disponibilité et son soutien technique, ainsi que les chercheurs en archéologie Clément LAPLAIGE, Nathanaël LE VOGUER et Xavier RODIER avec qui j'ai pu échanger tout au long du projet, aux intervenants SOPRA

pour la partie gestion de projet, et à Ronan BOCQUILLON et Jean-Yves RAMEL pour leurs consignes et conseils de qualité et mise en oeuvre.

Bibliographie

- [1] Ajay ARASANIPALAI. *State of the art deep learning : an introduction to Mask R-CNN*. URL : <https://medium.com/free-code-camp/mask-r-cnn-explained-7f82bec890e3>.
- [2] Manish CHABLANI. *YOLO — You only look once, real time object detection explained*. URL : <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>.
- [3] Sharif ELFOULY. *Introduction : Fast R-CNN (Object Detection)*. URL : <https://towardsdatascience.com/part-2-fast-r-cnn-object-detection-7303e1988464>
- [4] Sharif ELFOULY. *Part 1 : R-CNN (Object Detection)*. URL : <https://towardsdatascience.com/r-cnn-3a9beddfd55a>.
- [5] Rohith GANDHI. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. URL : <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.
- [6] Gabriel GARZA. *Mask R-CNN for Ship Detection Segmentation*. URL : <https://towardsdatascience.com/mask-r-cnn-for-ship-detection-segmentation-a1108b5a083>.
- [7] Adam W. HARLEY. *2D visualization of a Convolutional Neural Network*. URL : <http://scs.ryerson.ca/~aharley/vis/conv/flat.html>.
- [8] Jonathan HUI. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. URL : https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.
- [9] Trevor Darrell JONATHAN LONG Evan Shelhamer. *Fully Convolutional Networks for Semantic Segmentation*. URL : <https://arxiv.org/abs/1411.4038>.
- [10] Ludo LOUIS. *Les différents types de réseaux de neurones : réseau de convolution*. URL : <https://ludo-louis.fr/differents-types-reseaux-neurones-reseau-convolution/>.
- [11] Mehdi MOUADIL. *Introduction au Deep Learning : les réseaux de neurones*. URL : <https://meritis.fr/ia/deep-learning/>.
- [12] Thomas Brox OLAF RONNEBERGER Philipp Fischer. *U-Net : Convolutional Networks for Biomedical Image Segmentation*. URL : <https://arxiv.org/abs/1505.04597>.

- [13] Kimia Nadjahi PASCAL MONASSE. *Qu'est ce qu'un réseau de neurones convolutif (ou CNN)?* URL : <https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn>.
- [14] Gérald PETITJEAN. *Introduction aux réseaux de neurones!* URL : https://www.lrde.epita.fr/~sigoure/cours_ReseauxNeurones.pdf.
- [15] Joseph REDMON et Ali FARHADI. *YOLO : Real-Time Object Detection*. URL : <https://pjreddie.com/darknet/yolo/>.
- [16] Olaf RONNEBERGER. *U-Net : Convolutional Networks for Biomedical Image Segmentation*. URL : <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>.
- [17] SCIENCE4ALL. *Les réseaux de convolution (CNN) | Intelligence artificielle 47*. URL : https://www.youtube.com/watch?v=zG_50tgxfAg.
- [18] SCIENCE4ALL. *Les réseaux de neurones | Intelligence artificielle 41*. URL : <https://www.youtube.com/watch?v=8qL2l5Qd9L8>.
- [19] PULKIT SHARMA. *Computer Vision Tutorial : Implementing Mask R-CNN for Image Segmentation (with Python Code)*. URL : <https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/>.
- [20] SOCIALMIX. *Comprendre le DeepLearning et les Réseaux de neurones en 10 mins!* URL : <https://www.youtube.com/watch?v=gPVVsw20WdM>.
- [21] Sik-Ho TSANG. *Review : FCN — Fully Convolutional Network (Semantic Segmentation)*. URL : <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>.
- [22] WIKIPÉDIA. *Réseau neuronal convolutif*. URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.
- [23] WIKIPÉDIA. *Réseaux de neurones artificiels*. URL : https://fr.wikipedia.org/wiki/R%C3%A9seaux_de_neurones_artificiels.
- [24] WIKIPÉDIA. *Retropropagation du gradient*. URL : https://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient.
- [25] WIKIPÉDIA. *U-Net*. URL : <https://en.wikipedia.org/wiki/U-Net>.

Annexes

A

Cahier de spécifications

ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Spécialité Informatique

64 av. Jean Portalis

37200 TOURS, FRANCE

Tél +33 (0)2 47 36 14 31

www.polytech.univ-tours.fr

CAHIER DE SPÉCIFICATION			
Projet : PRD-09		Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR	
Emetteur :		Valentin MAURICE	MOA : T. BROUARD, JY. RAMEL, C. LAPLAIGE, X. RODIER, N. LE VOGUER
Date d'émission :		2019	
Validation			
Nom	Date	Valide (O/N)	Commentaires
Historique des modifications			
Version	Date	Description de la modification	
00	10/2019	Version initiale du document	
01	11/2019	Premier jet	
02	11/2019	Version corrigée selon retours MOA (C. LAPLAIGE)	
03	11/2019	Version corrigée selon retours MOA (T. BROUARD)	

TABLE DES MATIÈRES

Cahier de spécifications	4
Contexte de la réalisation	4
Objectifs	5
Hypothèses	6
Description générale	6
Environnement du projet	6
Caractéristiques des utilisateurs	6
Fonctionnalités du système	7
Structure générale du système	7
Description des interfaces externes du logiciel	8
Interfaces matériel/logiciel	8
Interfaces homme/machine	8
Interfaces logiciel/logiciel	10
Spécifications fonctionnelles	10
Définition de la fonction 1 : loadModel	10
Définition de la fonction 2 : saveModel	10
Définition de la fonction 3 : trainModel	10
Définition de la fonction 4 : evaluateModel	10
Définition de la fonction 5 : analyze	11
Spécifications non fonctionnelles	11
Contraintes de développement et conception	11
Contraintes de fonctionnement et d'exploitation	11
Performances	11
Capacités	11
Contrôlabilité	11
Sécurité	11
Glossaire	12

Ce document est le cahier des spécifications du projet recherche et développement n°9 : “Interactive Deep Learning : Application à la reconnaissance d’éléments archéologiques dans les images LiDAR”. Ce projet s’inscrit dans la continuité du programme SOLiDAR, programme qui consiste à utiliser la technologie LiDAR afin de cartographier des zones forestières qui sont difficilement analysables depuis des photographies aériennes classiques, notamment à cause de la végétation.

La maîtrise d’ouvrage est ici représentée par Thierry BROUARD et Jean-Yve RAMEL (enseignants chercheurs à l’Ecole Polytechnique de l’Université de Tours, département informatique), ainsi que par Clément LAPLAIGE, Xavier RODIER et Nathanaël LE VOGUER (chercheurs en archéologie UMR 7324 CITERES - LAT).

La maîtrise d’oeuvre est ici représentée par Valentin MAURICE (étudiant en 5e année à l’Ecole Polytechnique de l’Université de Tours, département informatique), auteur du présent document.

1. Contexte de la réalisation

Le LiDAR (Light Detection And Ranging) est une technique de détection de distance fondée sur les propriétés réfléchissantes d’un faisceau de lumière, à l’image d’un sonar pour le son.

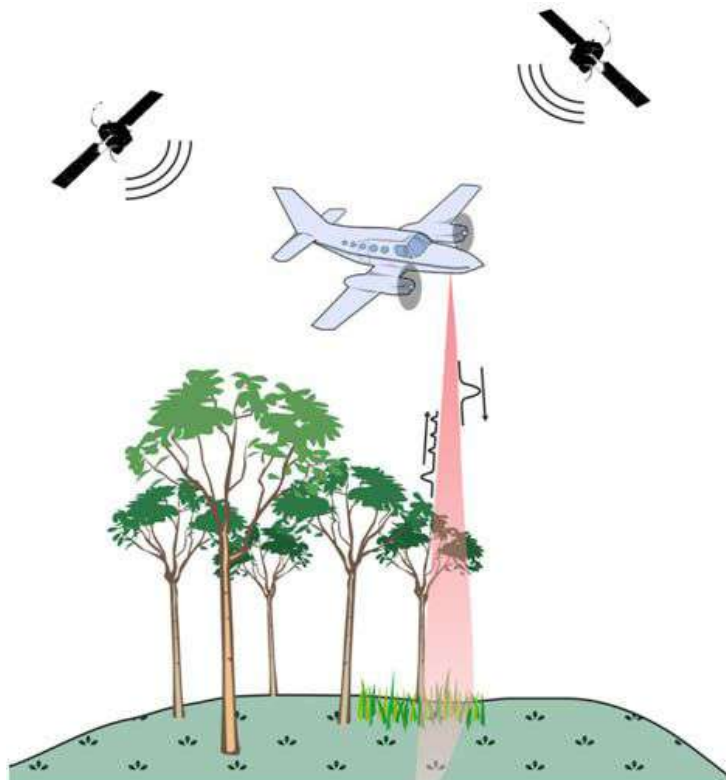
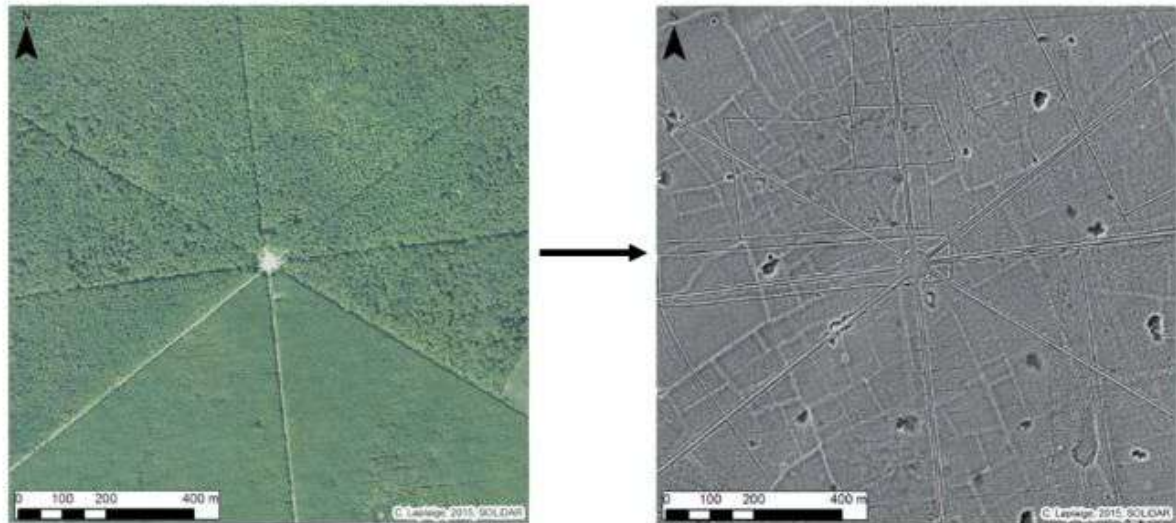


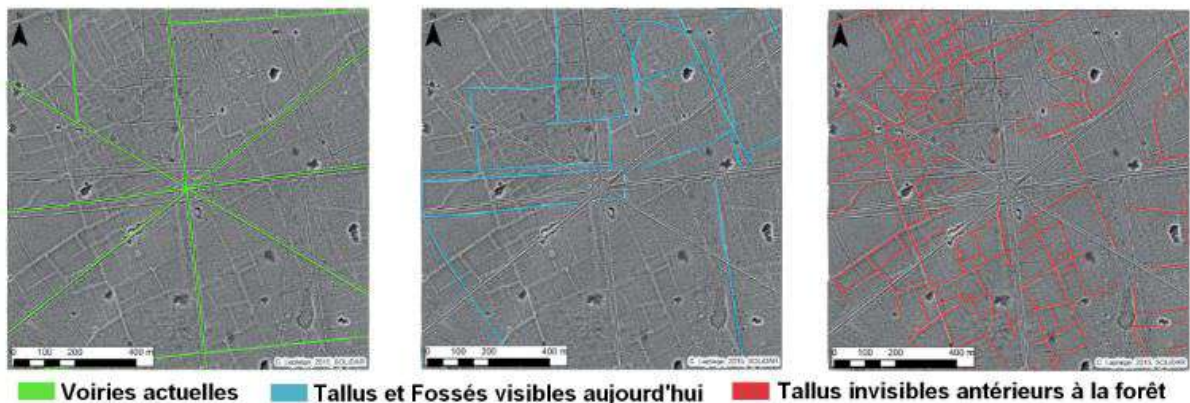
Schéma du fonctionnement du LiDAR (C. Laplaige, 2012)

source : <http://citeres.univ-tours.fr/spip.php?article2133>

Cette technologie présente plusieurs avantages pour la cartographie des variations microtopographiques du sol, elle permet notamment de passer outre la couverture végétale grâce à un multi-échantillonnage dense (10 points / m²). Cette caractéristique permet de faire apparaître des structures invisibles sur le terrain, et sur les images satellites et aériennes classiques.



Vue aérienne / télédétection LiDAR



Structures apparaissant sur les images LiDAR

On peut ici observer la comparaison entre une prise de vue aérienne et une acquisition LiDAR pour le même secteur forestier, de nombreux reliefs apparaissent grâce à la télédétection en plus des voiries actuelles.

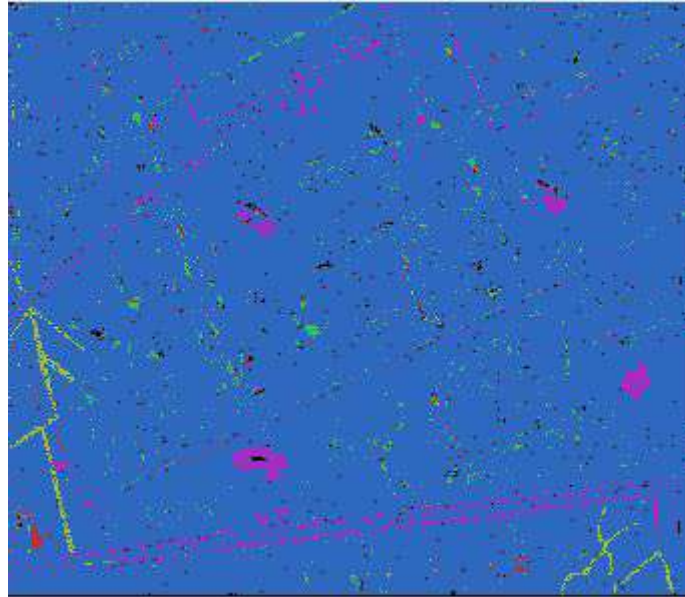
Ce programme nous apporte de nouvelles données sur des territoires de notre patrimoine, Chambord, Boulogne, Russy et Blois, et ouvre la porte à de nouvelles analyses archéologiques.

Le projet présente une grande quantité de données à segmenter et classifier, travail fastidieux réalisé à la main par les archéologues. L'outil de segmentation automatique SkyEye a précédemment été développé pour aider à la décision lors de l'analyse afin d'alléger la charge de travail des experts.

1.1. Objectifs

L'objectif du projet est l'amélioration de l'outil actuel SkyEye. L'outil est écrit en Python et son utilisation est mono-poste, le programme est portable, en anglais, et sert à la prise de décision dans la classification d'éléments sur une image, on cherchera donc en premier lieu la qualité et la fiabilité plutôt que la quantité (vitesse de résolution).

Les performances actuelles de la segmentation ne sont pas si satisfaisantes, la segmentation pixel à pixel est peu lisible (voir ci-dessous) et on dépasse difficilement les 70% de classification correcte ; en ce sens, nous souhaiterions tester une nouvelle approche orientée réseau de neurones profond, là où la classification actuelle est linéaire (Séparateur à Vaste Marge).



Exemple d'un résultat de classification par SkyEye

L'objectif est alors de proposer une implémentation CNN adaptée aux données, et son intégration dans l'outil existant.

1.2. Hypothèses

La réalisation sera fonction de l'implémentation choisie lors de la phase de recherche (voir l'état de l'art). Ce faisant, les entrées et sorties des fonctions pourraient éventuellement varier, même si celles-ci restent globalement les mêmes pour la plupart des réseaux à convolution.

2. Description générale

2.1. Environnement du projet

L'objectif est d'intégrer la solution développée lors de ce projet dans l'outil SkyEye existant, ce faisant, on pourra d'abord développer une partie ou l'entièreté de la nouvelle implémentation et son interface séparément, puis l'intégrer à l'existant par la suite.

On utilisera PyCharm pour le développement Python, de plus un certain nombre de bibliothèques python sont nécessaires au fonctionnement de l'outil actuel : *appdirs*, *cycler*, *matplotlib*, *numpy*,

olefile, opencv-python, packaging, Pillow, pyparsing, PyQt5, python-dateutil, pytz, scikit-learn, scipy, sip, six.

Voir la fiche de configuration pour les versions minimales à utiliser.

La partie interface est gérée par Qt.

Pour la partie CNN et son implémentation, plusieurs bibliothèques Python pourront être utilisées, notamment Keras ou Pytorch.

2.2. Caractéristiques des utilisateurs

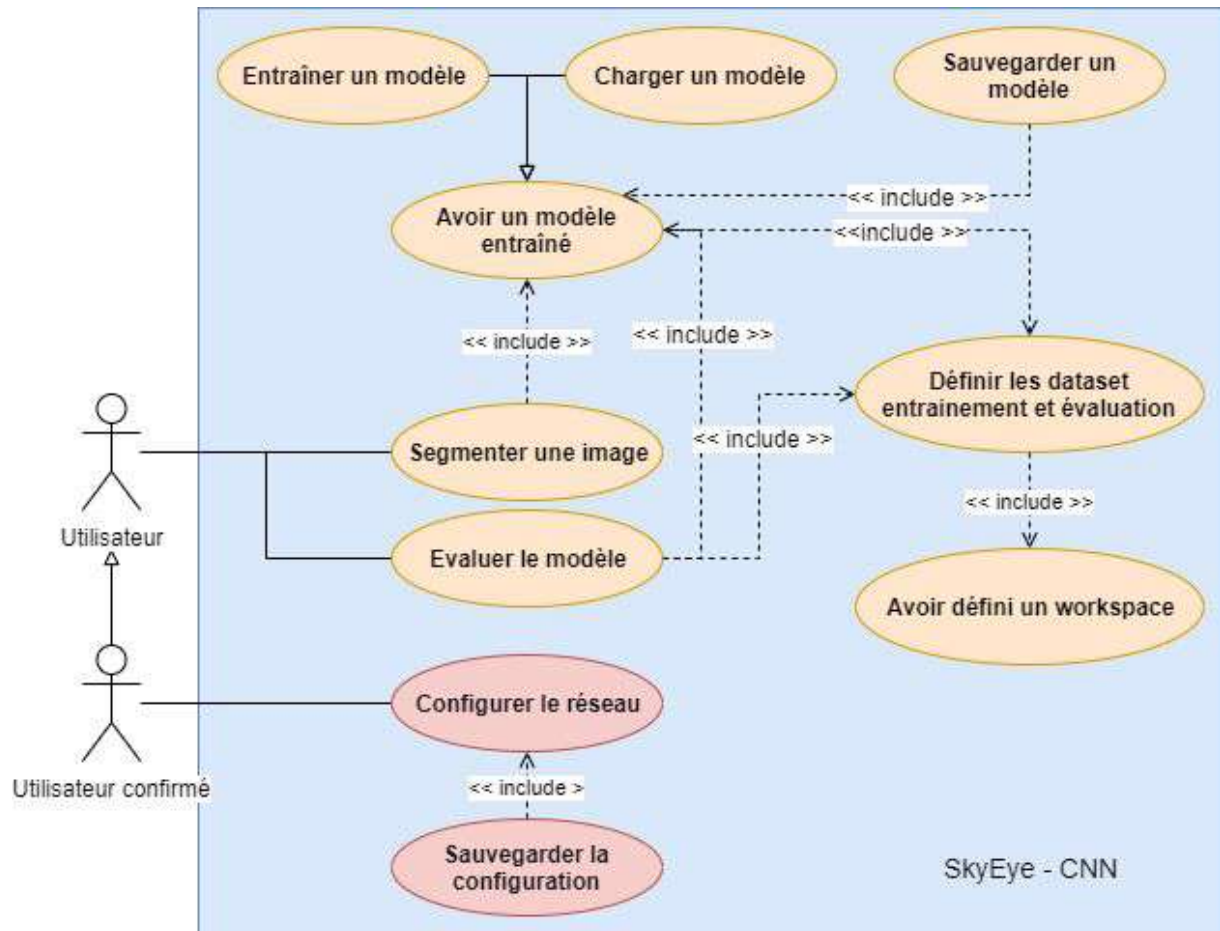
Les utilisateurs de l'outil sont des chercheurs en archéologie, on estime qu'ils sont à priori familiers avec l'utilisation d'outils informatiques, et ont seulement quelques notions, voire pas du tout, en reconnaissance d'image et apprentissage automatique / profond.

On voudrait donc proposer une expérience utilisateur simple et intuitive qui ne nécessite pas ou peu de bagages techniques dans un domaine autre que l'archéologie, on voudrait cependant proposer aux utilisateurs ayant ces bagages, un potentiel d'action sur les aspects plus techniques (typiquement pouvoir gérer les paramètres du classifieur).

On intégrera les nouvelles fonctionnalités développées dans l'interface existante en restant cohérent avec l'interface actuelle (voir **3.2 Interfaces homme/machine**).

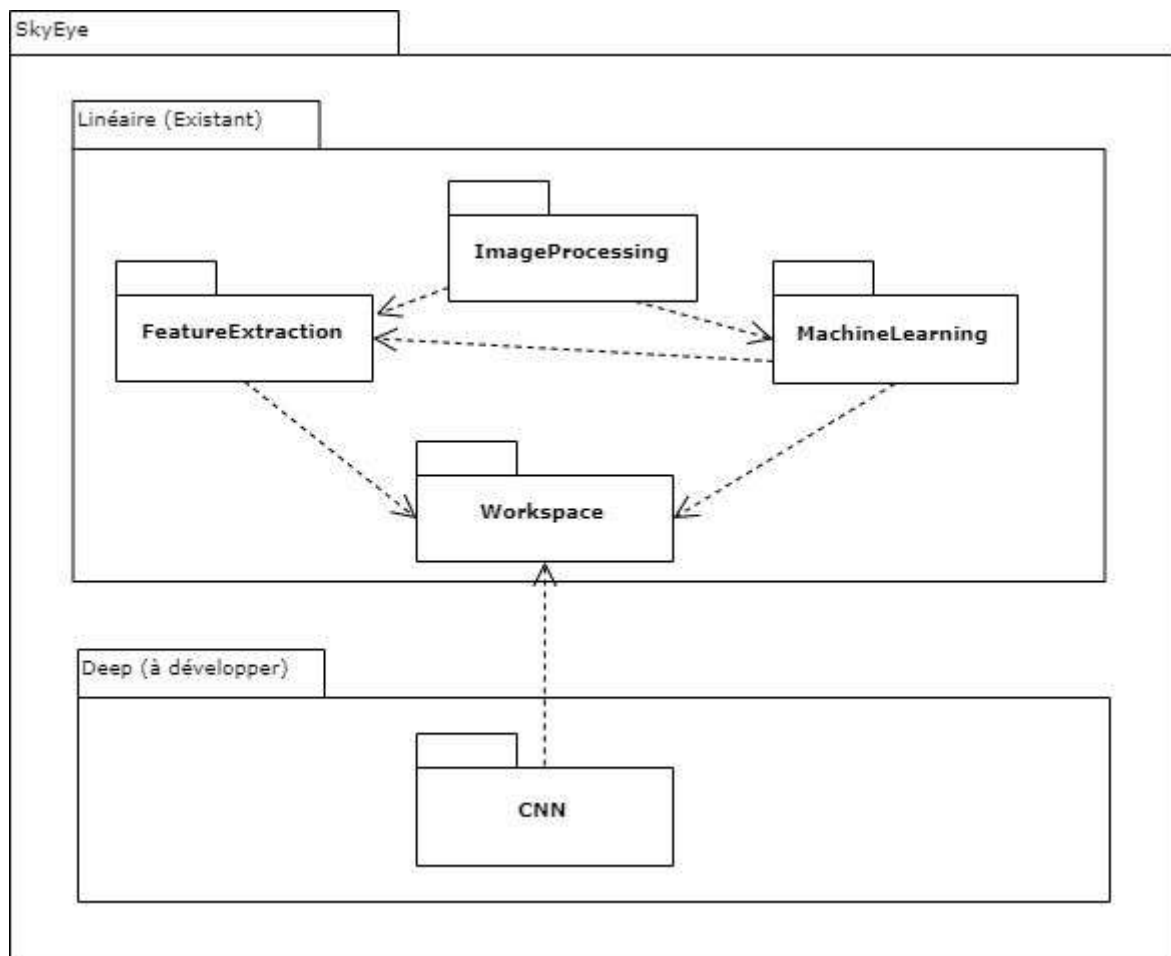
2.3. Fonctionnalités du système

La classification CNN à développer servira les cas d'utilisation suivants :



2.4. Structure générale du système

Les composants existants de l'outil sont orientés autour de la classification linéaire (extraction des caractéristiques des images, machine learning, etc.). Le seul lien avec l'existant sera à priori le module de gestion du workspace du projet (gère la structure des dossiers d'entrée / sortie), on viendra alors greffer notre solution de la manière suivante :



3. Description des interfaces externes du logiciel

3.1. Interfaces matériel/logiciel

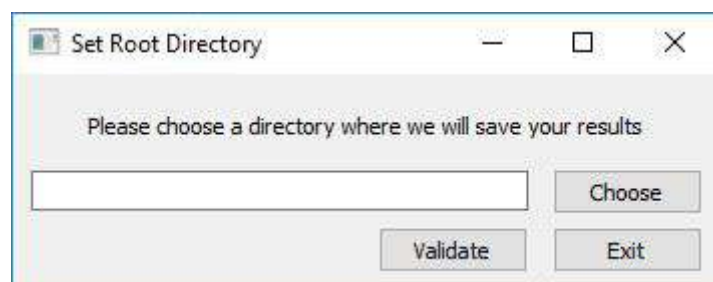
Le logiciel est standalone, il ne nécessite pas d'interfaces matérielles supplémentaires.

3.2. Interfaces homme/machine

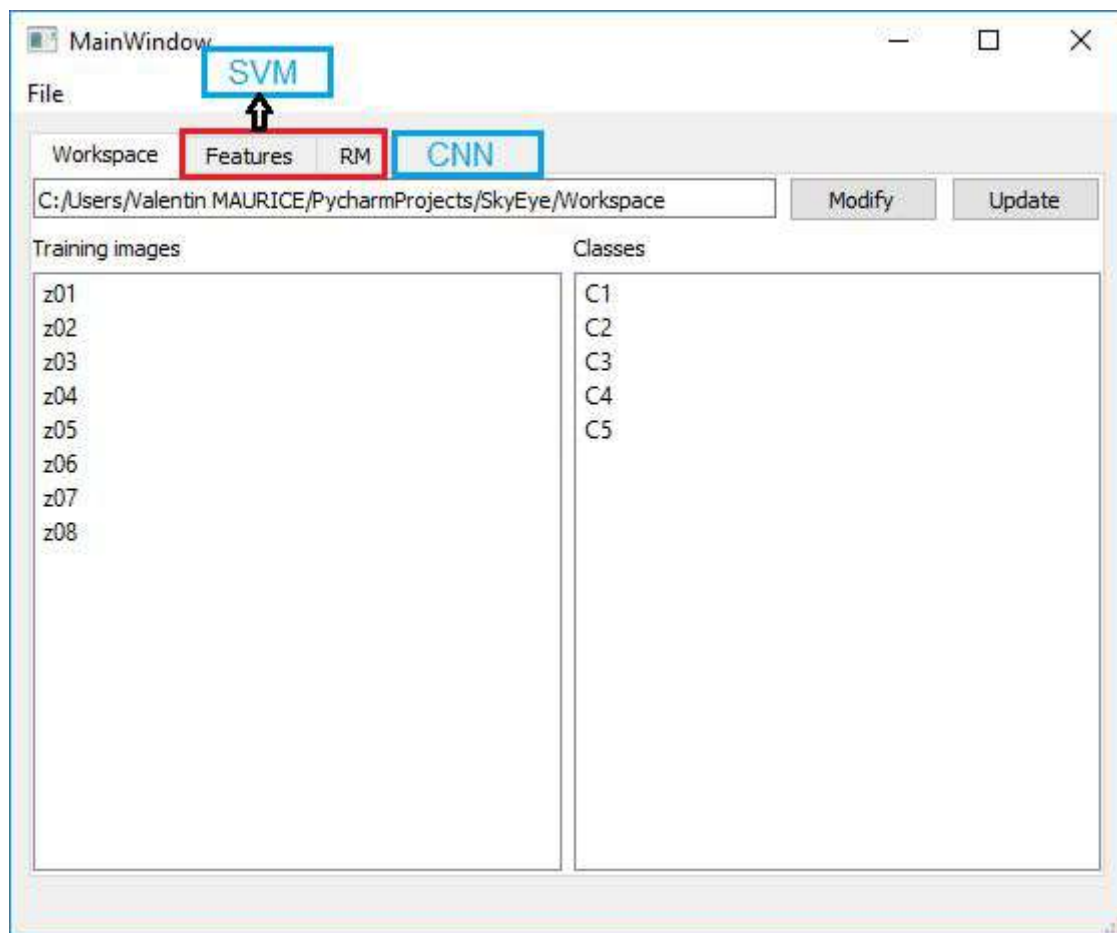
Dans le cadre d'une intégration dans l'existant, on souhaite garder le même type d'interface.

Propositions d'évolution de l'interface :

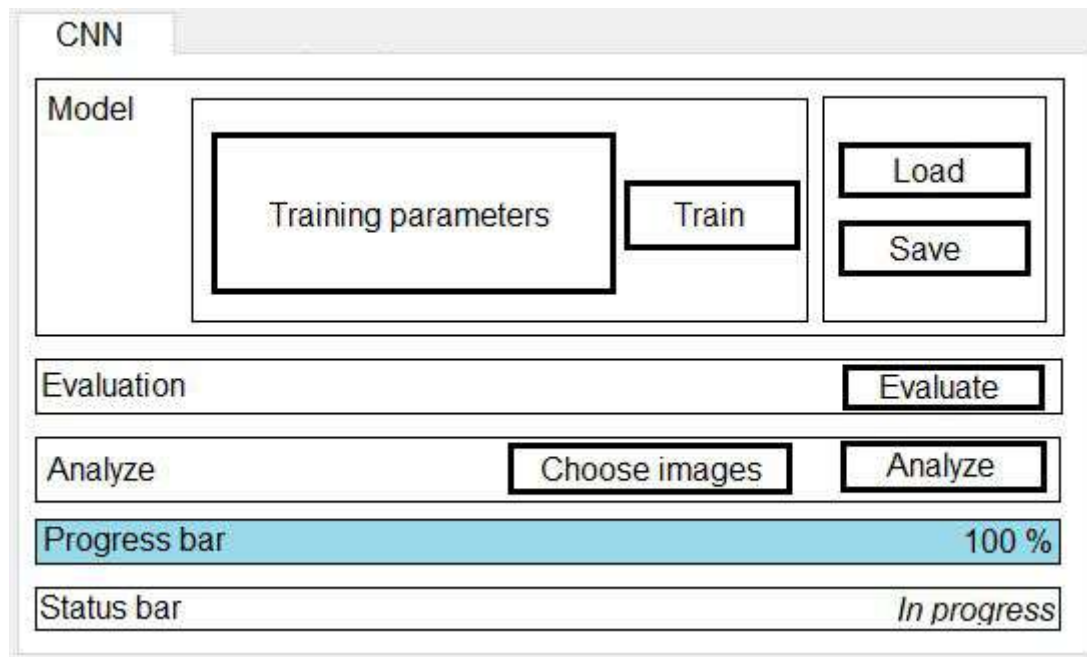
On conserverait l'écran de sélection du workspace actuel.



On fusionnerait ici les onglets *Features* et *RM* en un seul onglet *SVM*, et on ajouterait un onglet *CNN*.



L'onglet CNN serait organisé de la manière suivante :



3.3. Interfaces logiciel/logiciel

Le logiciel est standalone, il ne nécessite pas d'interface logicielle supplémentaires.

4. Spécifications fonctionnelles

4.1. Définition de la fonction 1 : loadModel

Cas d'utilisation : "Charger un modèle"

Identification de la fonction 1

Cette fonction permet de charger un modèle existant afin d'être utilisé par les fonctions d'entraînement et d'analyse.

Description de la fonction 1

Cette fonction prend en entrée un fichier à utiliser pour dé-sérialiser l'objet modèle à charger, et retourne cet objet, configuration incluse.

4.2. Définition de la fonction 2 : saveModel

Cas d'utilisation : "Sauvegarder un modèle"

Identification de la fonction 2

Cette fonction permet de sauvegarder le modèle courant sur le disque afin d'être chargé à nouveau dans le logiciel.

Description de la fonction 2

Cette fonction prend en entrée le modèle, configuration incluse, et un emplacement à utiliser pour sérialiser l'objet.

4.3. Définition de la fonction 3 : trainModel

Cas d'utilisation : "Entraîner le modèle"

Identification de la fonction 3

Cette fonction permet d'entraîner le modèle chargé avec le jeu d'entraînement.

Description de la fonction 3

La fonction prend en entrée le jeu de données présent dans le workspace, ainsi que les paramètres d'entraînement (**Utilisateur confirmé**), le modèle courant est alors entraîné et mis à jour.

4.4. Définition de la fonction 4 : evaluateModel

Cas d'utilisation : "Evaluer le modèle"

Identification de la fonction 4

Cette fonction permet d'évaluer la qualité de l'entraînement du modèle, en analysant un jeu d'images pour lesquelles on a la vérité terrain, et de mesurer l'écart entre la prédiction et cette vérité. On utilisera au minimum une matrice de confusion comme indicateur.

Description de la fonction 4

La fonction prend en entrée un jeu de données d'évaluation présent dans le workspace, et retourne une matrice de confusion ainsi qu'un coefficient de reconnaissance : nb pixels reconnus / nb pixels totaux.

4.5. Définition de la fonction 5 : analyze

Cas d'utilisation : "Segmenter une image"

Identification de la fonction 5

Cette fonction permet de segmenter un jeu d'images en utilisant le modèle courant.

Description de la fonction 5

La fonction prend en entrée une ou plusieurs images, et va l'analyser en produisant une image segmentée en fonction des classes reconnues, ainsi qu'une image binaire représentative par classe. On pourra éventuellement régler certains paramètres ici pour les utilisateurs avancés.

5. Spécifications non fonctionnelles

5.1. Contraintes de développement et conception

Pour le développement, on utilisera le langage Python pour rester dans la continuité du projet, et pour les outils de deep learning disponibles au travers de bibliothèques comme Keras ou Pytorch.

5.2. Contraintes de fonctionnement et d'exploitation

5.2.1. Performances

On souhaite en terme de temps d'entraînement et de prédiction, que l'opération ne dépasse pas le temps d'une nuit (~12h). On souhaite aussi que le programme ne "bloque" pas pendant le traitement.

5.2.2. Capacités

On estime que les images à traiter ne dépassent pas une certaine taille à définir, de l'ordre de la dizaine de Mo.

5.2.3. Contrôlabilité

On souhaite éventuellement proposer un suivi de la progression du traitement via une barre de progression, une estimation du temps restant, et au minimum un fichier log reprenant les étapes du traitement.

5.2.4. Sécurité

Aucune mesure de sécurité particulière n'est prévue, les utilisateurs utilisent librement l'outil.

CITERES : UMR Cités, TERRitoires, Environnement et Sociétés

CNN : Convolutional Neural Network, Réseau de Neurones à Convolution

LAT : Laboratoire Archéologie et Territoires

LiDAR : Light Detection And Ranging

SVM : Machine à Vecteurs de Support / Séparateurs à Vaste Marge

B

Fiche de configuration

Fiche de configuration

PRD-09 : Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR SkyEye

Configuration requise pour la version initiale de SkyEye :

- Python 3.5.1
- Librairies Python :
 - OpenCV 3.1.0-dev
 - Matplotlib 1.5.3
 - Scikit-learn 0.18.1
 - Scikit-image 0.12.3
 - Numpy 1.12.0
 - Scipy 0.19.0
 - Sphinx 1.5.1
 - Treelib 1.5.1

Configuration requise pour la version CNN de SkyEye :

- Python 3.6.5
- Libraries Python :
 - tensorflow==1.9.0
 - keras==2.2.5
 - protobuf==3.6.0
 - six (1.14.0)
 - pillow (7.0.0)
 - matplotlib (3.1.3)
 - scikit-image (0.16.2)
 - numpy (1.18.1)
 - h5py (2.10.0)
 - tqdm (4.43.0)
 - pyqt5 (5.14.1)
 - opencv-python (4.2.0.32)
 - imgaug (0.4.0)
 - sklearn (0.0)

Etudiant : Valentin MAURICE

Encadrants : Thierry BROUARD, Jean-Yves RAMEL

C

Fiche d'activité S9

Semaine 38			
18/09/2019	Découverte du sujet	Rdv encadrant	Installation / Prise en main de l'outil
19/09/2019	Analyse de l'outil	Tests (performances...)	
Semaine 39			
25/09/2019	Tests (paramètres optimaux...)	Rdv encadrant	
26/09/2019	Tests (paramètres optimaux...)	Rédaction document d'état des lieux	
Semaine 40			
02/10/2019	Indisponible		
03/10/2019	Test du Random Forest	Rédaction document d'état des lieux	Conférence IA Orange
Semaine 41			
09/10/2019	Rédaction document d'état des lieux	Lectures CNN	
10/10/2019	Lectures CNN	Rdv encadrant / archeos	Récupération SkyEye actuel
Semaine 42			
16/10/2019	Lectures biblio	Analyse de l'outil	Rédaction du document d'état des lieux V2
17/10/2019	Conférence DevUp	Rédaction doc d'état des lieux V2	
Semaine 43			
23/10/2019	RDV specif N. RAGOT	Recherches : réseaux de neurones	
24/10/2019	Rédaction état de l'art (réseaux de neurones)		
Semaine 44			
30/10/2019	Pause pédagogique		
31/10/2019			
Semaine 45			
06/11/2019	Recherches : CNN / R-CNN & co / YOLO	Rédaction état de l'art	
07/11/2019		Conférence Framasoft	Conférence Sopra
Semaine 46			
13/11/2019	Thèse Maxime MARTINEAU	Recherches : DeconvNet / U-Net	Rédaction état de l'art
14/11/2019	Rédaction état de l'art	Rédaction CDS	
Semaine 47			
20/11/2019	Rédaction CDS		
21/11/2019	Rédaction CDS	Forum stage-emplois	
Semaine 48			
27/11/2019	Correction CDS	Rédaction Rapport	
28/11/2019	Correction CDS	RDV Gestion/Suivi Sopra	Rédaction Rapport
Semaine 49			
04/12/2019	Rédaction rapport		
05/12/2019	Préparation soutenance		Nuit de l'info

Figure 1 – Fiche d'activité S9

D

Gantt d'activité S9

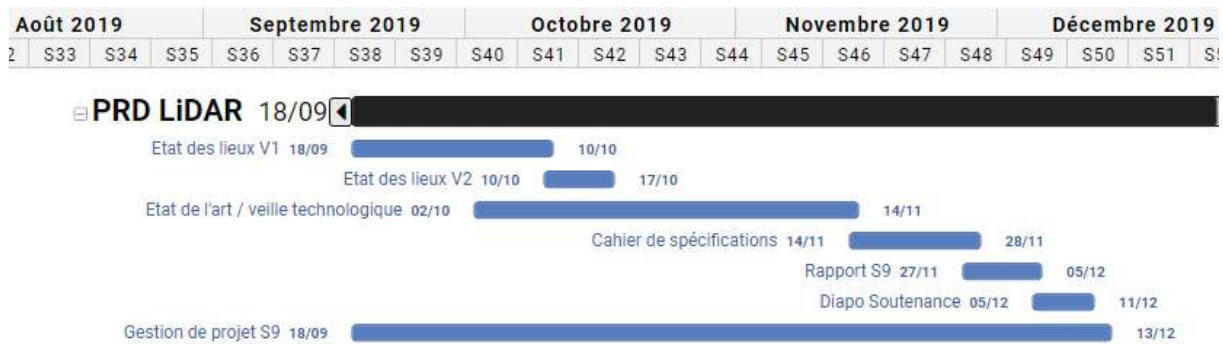


Figure 1 – Gantt reprenant les étapes de la phase de recherche du S9

E

Gantt prévisionnel S10



Figure 1 – Gantt prévisionnel de la partie développement du S10

F

Gantt d'activité S10

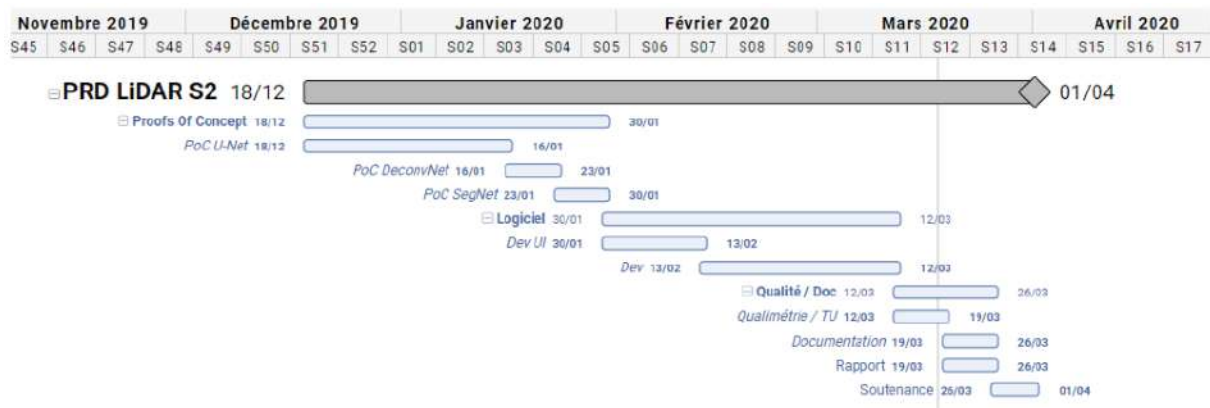


Figure 1 – Gantt d'activité de la partie développement du S10



G

Manuel développeur

Outil de segmentation d'image en Deep Learning : SkyEye

Ce projet est un *fork* du projet <https://github.com/divamgupta/image-segmentation-keras> :
Implémentation de Segnet, FCN, UNet, PSPNet et d'autres modèles avec Keras.

Le projet a été réalisé par Valentin MAURICE dans le cadre du **Projet Recherche et Développement (Projet de Fin d'Étude)** de la troisième année du cycle ingénieur au département informatique de l'école Polytech Tours, lors de l'année 2019-20.

L'idée est d'utiliser l'implémentation des modèles et classes utilitaires du projet forké afin de proposer une interface répondant aux besoins de l'appel d'offre initial : *Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR*. L'outil proposé peut néanmoins s'apparenter à un outil de segmentation d'image de toutes sortes, en utilisant des techniques de deep learning basée sur des réseaux CNN.

Prérequis

Le projet a été développé avec une distribution Python 3.6.0 (64bit) sur windows, et nécessite les dépendances suivantes (à retrouver dans requirements.txt) :

- **tensorflow==1.9.0**
- **keras==2.2.5**
- **protobuf==3.6.0**
- six (1.14.0)
- pillow (7.0.0)
- matplotlib (3.1.3)
- scikit-image (0.16.2)
- numpy (1.18.1)
- h5py (2.10.0)
- tqdm (4.43.0)
- pyqt5 (5.14.1)
- opencv-python (4.2.0.32)
- imgaug (0.4.0)
- sklearn (0.0)

Lancement

Il suffit d'exécuter `skyeye_segmentation/entrypoint.py` avec Python, l'application devrait s'ouvrir.

Structure du projet

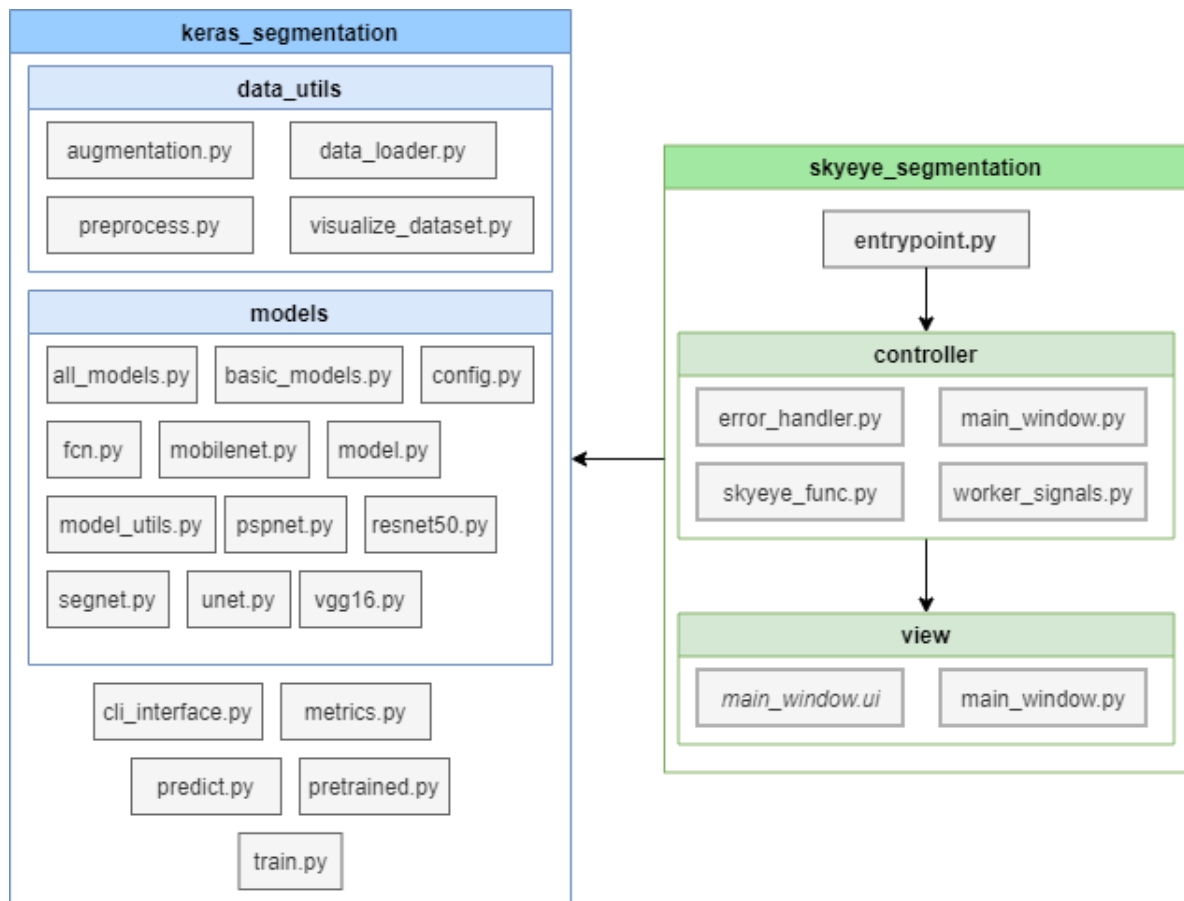
Le projet est structuré de la manière suivante :

- **html/** : contient la documentation générée avec [pdoc](#)

- **keras_segmentation/** : sources du projet forké
- **log/** : fichiers logs
- **README/** : ressources de ce document
- **skyeeye_segmentation/** : source du projet
- **test_keras_segmentation/** : tests unitaires du projet forké
- **test_syeye_segmentation/** : tests unitaires du projet

Modules

Les modules pythons s'organisent de la manière suivante :



Le module **view** contient un fichier .ui généré avec [Qt Designer](#), qui a été automatiquement traduit en .py grâce au convertisseur pyuic5 (contenu dans PyQt5) :

```
>>> pyuic5 main_window.ui > main_window.py
```

Documentation

La documentation disponible dans *html/* a été générée avec [pdoc](#) :

```
>>> pdoc skyeeye_segmentation
```

Tests unitaires

Lancer les tests unitaires du module *keras_segmentation* (**se placer dans `test_keras_segmentation/`**) :

```
>>> pytest
```

Lancer les tests unitaires du module *skyeye_segmentation* (**se placer dans `test_skyeye_segmentation/`**) :

```
>>> pytest
```

Qualimétrie

Pour lancer une analyse statique du code avec [PyLint](#) (fichier de configuration : *.pylintrc*) :

```
>>> pylint --rcfile=.pylintrc skyeye_segmentation > pylint.txt
```

Le rapport PyLint est alors disponible et donne des indications sur la qualité du code analysé, ainsi qu'une note générale :

```
-----  
Your code has been rated at 9.25/10 (previous run: 9.29/10, -0.04)
```

Manuel d'utilisation

Voir le [manuel d'utilisation](#)

Téléchargement

Télécharger l'archive de la dernière version [ici](#).

A logo consisting of a blue square containing a white capital letter 'H'.

H

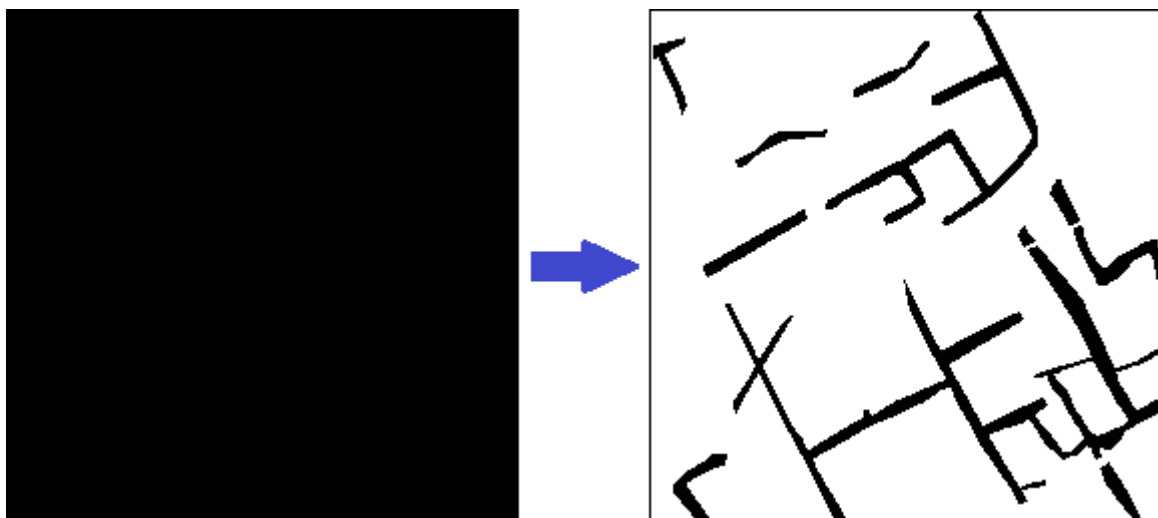
Manuel utilisateur

Manuel d'utilisation SkyEye

L'outil SkyEye permet d'entraîner un modèle de segmentation d'images, d'évaluer ses performances, effectuer des prédictions sur de nouvelles images, et préparer les données en amont de ces opérations.

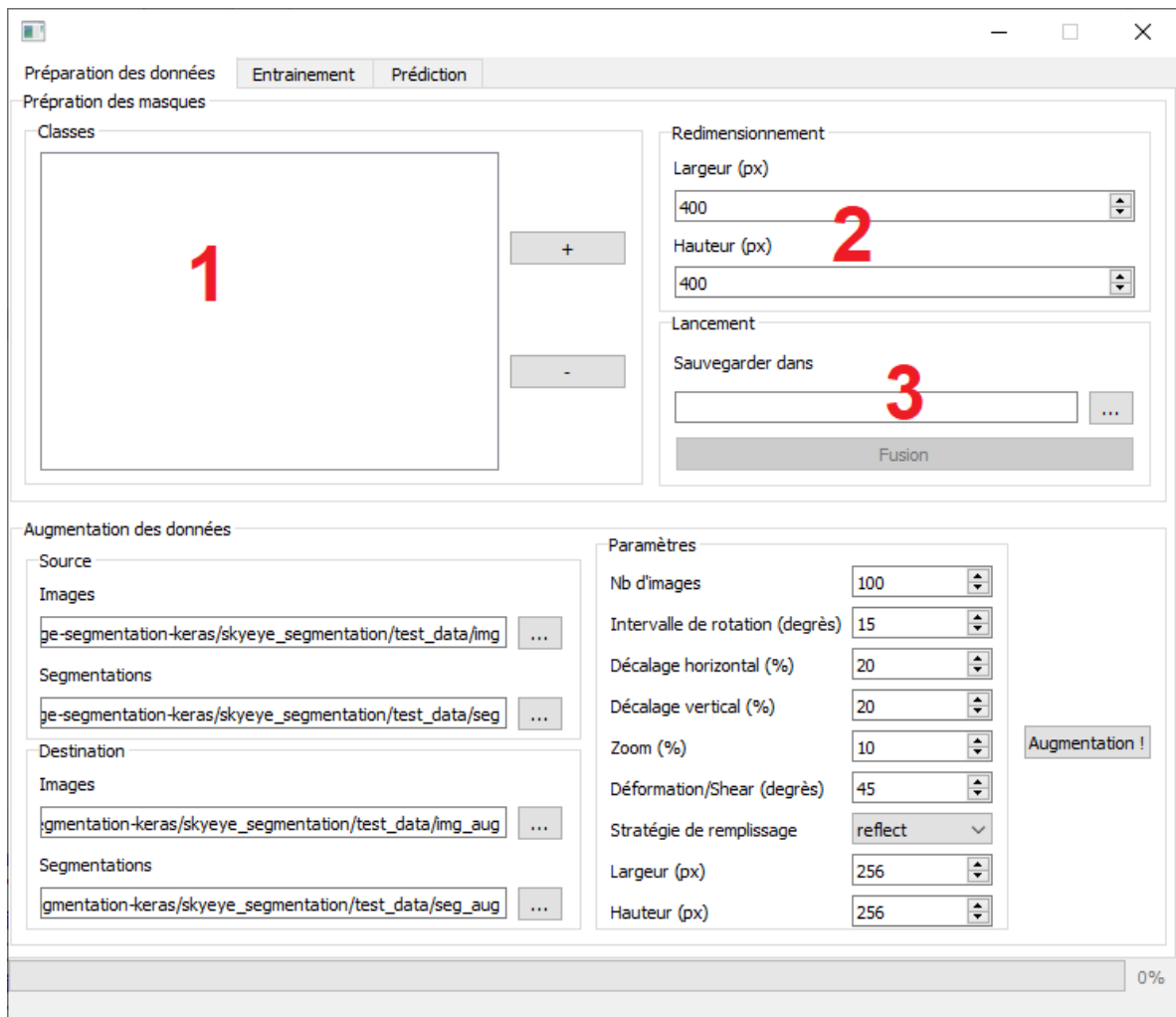
Création des masques

Les modèles utilisés travaillent tous avec le même type de données : des images (.jpg, .png, .tif), et leur masque (**doit avoir le même nom**) contenant les segmentations des différentes classes identifiées sur l'image. Les masques sont uniquement constitués de niveaux de gris, et se faisant, peuvent supporter au maximum 255 classes différentes (256 valeurs de gris - la couleur de fond). Par exemple, pour une image contenant des éléments de talus et de charbonnière, les pixels talus auront la valeur de gris 1 et les pixels de charbonnière auront la valeur de gris 2, le fond ayant toujours la valeur 0. De ce fait, ce format de masque est impossible à vérifier à œil nu :



La première fonctionnalité de l'outil, permet de créer ces masques en fusionnant plusieurs segmentations au format .tif en noir et blanc, en spécifiant :

1. Les différents dossiers contenant les segmentation N&B, ex: *talus/ charb/* **à noter** les masques seront créés en prenant l'ordre de la liste des classes, (0: fond, 1: talus, 2: charb) cet ordre est à noter quelque part car le nom des classes n'est pas conservé dans les autres processus
2. La taille souhaitée des images créées
3. Le dossier où seront créés les masques



Augmentation des données

Entraîner des modèles de deep learning nécessite un jeu très conséquent de données (plusieurs centaines, voir milliers d'exemples), dans le cas où la base d'exemple que l'on possède n'est pas aussi fournie, on peut l'*augmenter*, en appliquant des transformations aux images d'origines, on en crée de nouvelles. SkyEye propose cela, il faudra spécifier :

1. Les dossiers images et masques que l'on veut augmenter (notre base d'apprentissage)
2. Les dossiers des futurs images et masques générés
3. Les paramètres de l'augmentation :
 1. Nb d'images : nombre voulu d'images à générer, (+1000 est un bon début)
 2. Intervalle de rotation : pour 15° les images subiront une rotation entre -15° et 15°
 3. Décalage horizontal/vertical : pour 20% les images subiront une translation entre -20% et +20% de leur taille sur l'axe indiqué
 4. Zoom : pour 10% les images subiront un zoom entre -10% et +10%
 5. Déformation (shear) : [déformation en cisaillement](#) de + ou - l'angle indiqué
 6. Stratégie de remplissage : les différentes opérations de transformation ont certainement laissé des "trous" dans l'image (décalage, rotation, etc.), ce paramètre spécifie comment combler ces trous :
 1. 'constant': kkkkkkkk | abcd | kkkkkkkk (k=0 ici)
 2. 'nearest': aaaaaaaa | abcd | dddddddd

3. 'reflect': abcd dcba | abcd | dcba abcd
4. 'wrap': abcd abcd | abcd | abcd abcd
7. Taille des images générées (redimensionnement)

Préparation des données

Entraînement

Prédiction

Préparation des masques

Classes

Redimensionnement

Largeur (px)

400

Hauteur (px)

400

Lancement

Sauvegarder dans

Fusion

Augmentation des données

Source

Images

ge-segmentation-keras/skyeye_segmentation/test_data/img

Segmentations

ge-segmentation-keras/skyeye_segmentation/test_data/seg

Destination

Images

gmentation-keras/skyeye_segmentation/test_data/img_aug

Segmentations

gmentation-keras/skyeye_segmentation/test_data/seg_aug

Paramètres

Nb d'images

100

Intervalle de rotation (degrès)

15

Décalage horizontal (%)

20

Décalage vertical (%)

20

Zoom (%)

10

Déformation/Shear (degrès)

45

Stratégie de remplissage

reflect

Largeur (px)

256

Hauteur (px)

256

Augmentation !

0%

Entraînement du modèle

Une fois qu'on dispose d'un jeu d'entraînement suffisamment fourni, on peut commencer à entraîner un modèle, il faudra alors spécifier :

1. Le dossier contenant les images et le dossier contenant les segmentations de notre jeu d'entraînement
2. **Si on veut créer un nouveau modèle** : Sélectionner le modèle souhaité, et définir les tailles des images de notre jeu d'entraînement
3. **Si on veut utiliser un modèle qu'on a déjà créé** : sélectionner le fichier du modèle (laisser ce champs vide si on veut en créer un nouveau), **Attention**: les images du jeu d'entraînement devront coller aux dimensions qu'on a défini lorsque le modèle à utiliser a été créé.
4. Le nombre de classes du modèle (ie. le nombre de niveaux de gris de nos masques), par exemple pour un masque contenant les classes TALUS et CHARB, ce nombre vaudra **3** (2 + 1, la classe "fond"). Si cette valeur ne colle pas avec les masques du jeu d'entraînement, ou le nombre de classes pour lesquelles le modèle existant qu'on voudrait utiliser a été créé, une erreur surviendra.

5. Les paramètres de l'entraînement :

1. Taille du batch : combien d'images vont être montées en mémoire par itération, une plus grande valeur pourra produire un entraînement de meilleure qualité, mais sera plus couteux en ressources (RAM et CPU)
2. Nb d'étapes par époque : nombre d'itération par entraînement, on essaye en général d'utiliser toute notre base d'entraînement, donc si je dispose 1000 images, et que 5 images sont utilisées à chaque itération, je spécifierais $1000/5 = 200$ itérations
3. Nb d'époques : nombre d'entraînements que je souhaite effectuer
4. Sauvegarder dans : endroit où sauvegarder les modèles entraînés + **un nom de modèle**, un fichier sera généré à chaque époque, par exemple si je spécifie *modele/test*, pour 3 époques j'aurais :
 - *modele/test.0* : modèle à l'issue de la première époque
 - *modele/test.1* : deuxième
 - *modele/test.2* : troisième
 - *modele/test_config.json* : contient des informations pour le logiciel

The screenshot shows a software window with three tabs: "Préparation des données", "Entraînement", and "Prédiction". The "Entraînement" tab is active. The "Paramètres" section contains several input fields and buttons. Red numbers 1 through 5 are overlaid on the image to highlight specific parameters:

- 1: Points to the "Images" field under "Jeu d'entraînement", which contains the path `_segmentation/test_data/img_aug`.
- 2: Points to the "Largeur des images d'entrée" field, which is set to 256.
- 3: Points to the "Modèle existant (ou le pour un nouveau modèle)" field, which is empty.
- 4: Points to the "Nombre de classes" field, which is set to 2.
- 5: Points to the "Nb d'étapes par époque" field, which is set to 10.

Other visible parameters include:

- "Nouveau modèle": `fcn_8`
- "Hauteur des images d'entrée": 256
- "Taille du batch": 2
- "Nb d'époques": 1
- "Sauvegarder dans [chemin]/[nom]": `_segmentation/test_data/modèles`
- "Jeu d'évaluation" fields are empty.
- Buttons for "Entraînement" and "Evaluation" are at the bottom right.
- A "Logs" section is at the bottom, currently empty.
- A progress bar at the very bottom shows 0%.

Modèles disponibles

Liste des modèles disponibles et leurs contraintes :

- `fcn_8` : réseau complètement convolutionnel

- fcn_32 : même structure, moins précis
- fcn_8_vgg : fcn_8 avec les poids du modèle [vgg](#) (entraîné sur le jeu [ImageNet](#)), **dim multiples de 32**
- fcn_32_vgg : fcn_32 avec les poids du modèle vgg, **dim multiples de 32**
- fcn_8_resnet50 : fcn_8 avec les poids du modèle [resnet50](#) (entraîné sur les jeux ImageNet et [COCO](#)), **dim multiples de 32**
- fcn_32_resnet50 : fcn_32 avec les poids du modèle resnet50, **dim multiples de 32**
- pspnet : [réseau d'analyse pyramidale de scène](#), **dim multiples de 192**
- vgg_pspnet : pspnet avec les poids du modèle vgg, **dim multiples de 192**
- resnet50_pspnet : pspnet avec les poids du modèle resnet50, **dim multiples de 32**
- pspnet_50 : mix entre pspnet et resnet50, **dim (473, 473) ou (713, 713)**
- pspnet_101 : mix entre pspnet et resnet101, **dim (473, 473) ou (713, 713)**
- unet_mini : version allégée de unet
- unet : [réseau de neurones à convolution pour la segmentation d'images biomédicales](#)
- vgg_unet : unet avec les poids du modèle vgg
- resnet50_unet : unet avec les poids du modèle resnet50, **dim multiples de 32**
- mobilenet_unet : mix entre [mobilenet](#) et unet, **dim (224, 244)**
- segnet : [Réseau encodeur-décodeur](#)
- vgg_segnet : mix entre vgg et segnet, **dim multiples de 32**
- resnet50_segnet : mix entre resnet50 et segnet, **dim multiples de 32**
- mobilenet_segnet : mix entre mobilenet et segnet, **dim (224, 244)**

Evaluation du modèle

Pour évaluer la qualité de l'entraînement de notre modèle, on utilise un jeu d'évaluation, quelques images et leur masque ayant le même format que les images du jeu d'entraînement mais qui n'y figurent pas, on aura ainsi une évaluation pertinente. Pour cela, on spécifie :

1. Le dossier des images et le dossier des segmentations du jeu d'évaluation
2. Le fichier du modèle à évaluer
3. Le nombre de classes pour lesquelles le modèle a été entraîné

The screenshot shows a software window with three tabs: 'Préparation des données', 'Entrainement', and 'Prédiction'. The 'Entrainement' tab is selected. The interface is divided into several sections:

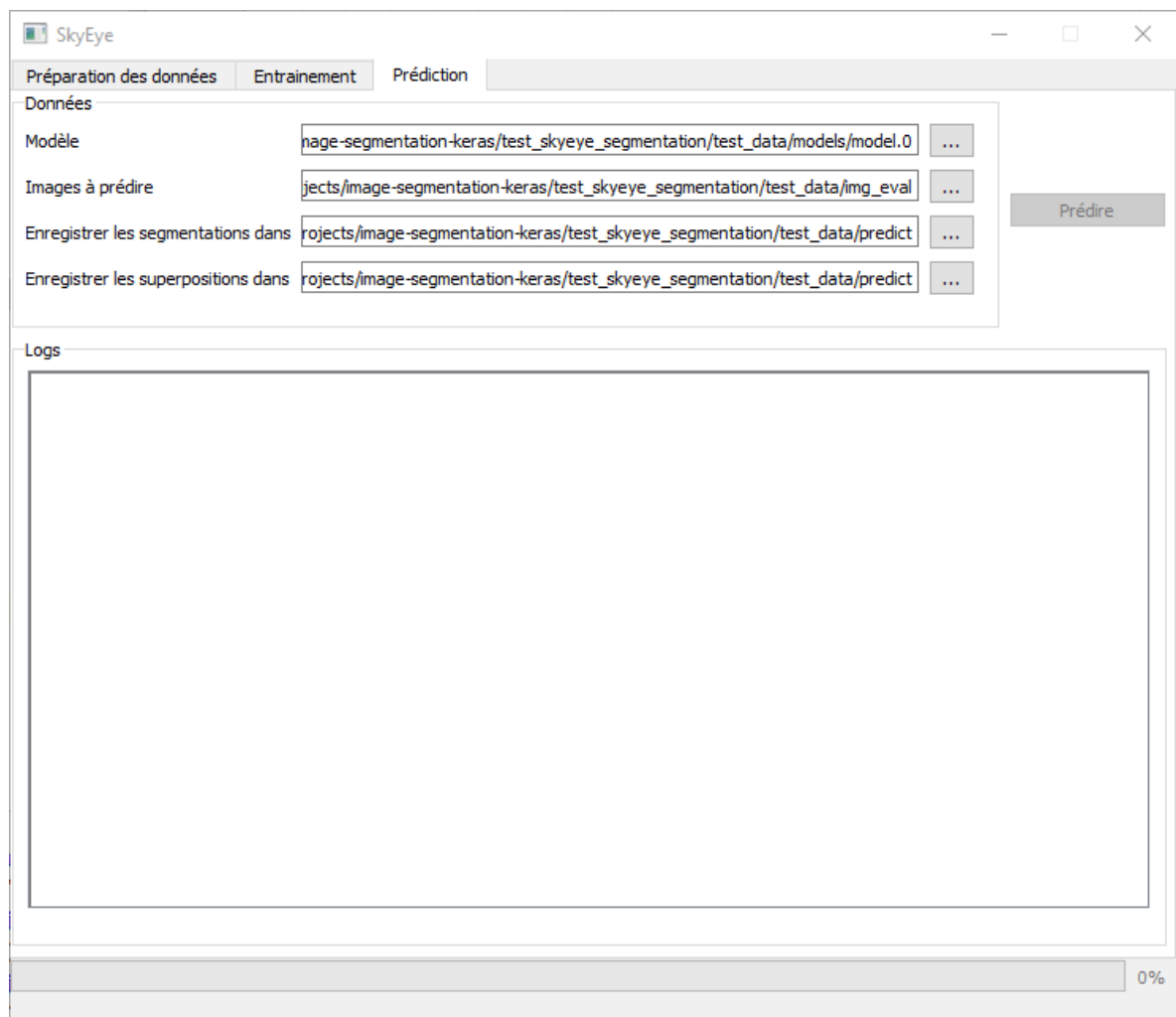
- Paramètres**:
 - Jeu d'entraînement**:
 - Images: ...
 - Segmentations: ...
 - Jeu d'évaluation**:
 - Images: ... (labeled with a red '1')
 - Segmentations: ...
 - Nouveau modèle**:
 - Nouveau modèle: ...
 - Largeur des images d'entrée: ...
 - Hauteur des images d'entrée: ...
 - Modèle existant (vide pour un nouveau modèle)**: ... (labeled with a red '2')
 - Nombre de classes**: ... (labeled with a red '3')
- Entrainement**:
 - Taille du batch: ...
 - Nb d'étapes par époque: ...
 - Nb d'époques: ...
 - Sauvegarder dans [chemin]/[nom]: ...
- Buttons**: 'Entrainement' and 'Evaluation' buttons.
- Logs**: A large empty text area for logs, with a progress bar at the bottom showing 0%.

L'évaluation donnera la matrice de confusion résultante (pixels bons/mauvais par classe), ainsi qu'une mesure [IU](#) (Intersection over Union, bon à partir de 0.5).

Prédictions

La dernière fonctionnalité offerte par l'outil permet d'utiliser un modèle entraîné afin de prédire des segmentations sur des images inconnues du modèle, pour cela on spécifie :

- Le fichier du modèle entraîné à utiliser
- Le dossier contenant les images à prédire
- Le dossier qui contiendra les segmentations générées (et la carte des probabilités au format CSV)
- Le dossier qui contiendra les superpositions entre les images et segmentations (peut être le même que le dossier précédent)



Webographie

- [WWW1] Ajay ARASANIPALAI. *State of the art deep learning : an introduction to Mask R-CNN*. URL : <https://medium.com/free-code-camp/mask-r-cnn-explained-7f82bec890e3>.
- [WWW2] Manish CHABLANI. *YOLO — You only look once, real time object detection explained*. URL : <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>.
- [WWW3] Sharif ELFOULY. *Introduction : Fast R-CNN (Object Detection)*. URL : <https://towardsdatascience.com/part-2-fast-r-cnn-object-detection-7303e1988464>
- [WWW4] Sharif ELFOULY. *Part 1 : R-CNN (Object Detection)*. URL : <https://towardsdatascience.com/r-cnn-3a9beddfd55a>.
- [WWW5] Rohith GANDHI. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. URL : <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.
- [WWW6] Gabriel GARZA. *Mask R-CNN for Ship Detection Segmentation*. URL : <https://towardsdatascience.com/mask-r-cnn-for-ship-detection-segmentation-a1108b5a083>.
- [WWW7] Adam W. HARLEY. *2D visualization of a Convolutional Neural Network*. URL : <http://scs.ryerson.ca/~aharley/vis/conv/flat.html>.
- [WWW8] Jonathan HUI. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*. URL : https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.
- [WWW9] Trevor Darrell JONATHAN LONG Evan Shelhamer. *Fully Convolutional Networks for Semantic Segmentation*. URL : <https://arxiv.org/abs/1411.4038>.
- [WWW10] Ludo LOUIS. *Les différents types de réseaux de neurones : réseau de convolution*. URL : <https://ludo-louis.fr/differents-types-reseaux-neurones-reseau-convolution/>.
- [WWW11] Mehdi MOUADIL. *Introduction au Deep Learning : les réseaux de neurones*. URL : <https://meritis.fr/ia/deep-learning/>.
- [WWW12] Thomas Brox OLAF RONNEBERGER Philipp Fischer. *U-Net : Convolutional Networks for Biomedical Image Segmentation*. URL : <https://arxiv.org/abs/1505.04597>.

- [WWW13] Kimia Nadjahi PASCAL MONASSE. *Qu'est ce qu'un réseau de neurones convolutif (ou CNN)?* URL : <https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn>.
- [WWW14] Gérald PETITJEAN. *Introduction aux réseaux de neurones!* URL : https://www.lrde.epita.fr/~sigoure/cours_ReseauxNeurones.pdf.
- [WWW15] Joseph REDMON et Ali FARHADI. *YOLO : Real-Time Object Detection*. URL : <https://pjreddie.com/darknet/yolo/>.
- [WWW16] Olaf RONNEBERGER. *U-Net : Convolutional Networks for Biomedical Image Segmentation*. URL : <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>.
- [WWW17] SCIENCE4ALL. *Les réseaux de convolution (CNN) | Intelligence artificielle 47*. URL : https://www.youtube.com/watch?v=zG_50tgxfAg.
- [WWW18] SCIENCE4ALL. *Les réseaux de neurones | Intelligence artificielle 41*. URL : <https://www.youtube.com/watch?v=8qL2lS0d9L8>.
- [WWW19] PULKIT SHARMA. *Computer Vision Tutorial : Implementing Mask R-CNN for Image Segmentation (with Python Code)*. URL : <https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/>.
- [WWW20] SOCIALMIX. *Comprendre le DeepLearning et les Réseaux de neurones en 10 mins!* URL : <https://www.youtube.com/watch?v=gPVVsw20WdM>.
- [WWW21] Sik-Ho TSANG. *Review : FCN — Fully Convolutional Network (Semantic Segmentation)*. URL : <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>.
- [WWW22] WIKIPÉDIA. *Réseau neuronal convolutif*. URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.
- [WWW23] WIKIPÉDIA. *Réseaux de neurones artificiels*. URL : https://fr.wikipedia.org/wiki/R%C3%A9seaux_de_neurones_artificiels.
- [WWW24] WIKIPÉDIA. *Retropropagation du gradient*. URL : https://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient.
- [WWW25] WIKIPÉDIA. *U-Net*. URL : <https://en.wikipedia.org/wiki/U-Net>.

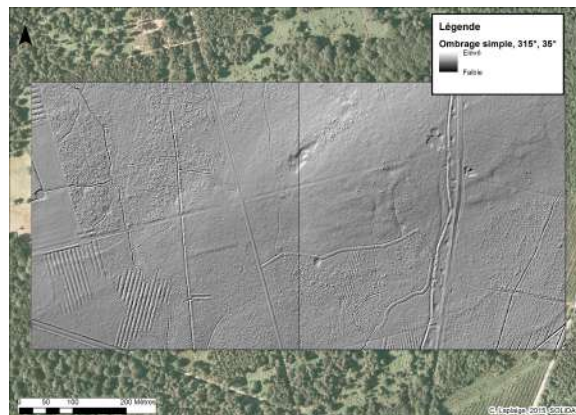
Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR

Valentin MAURICE

Encadrement : Thierry BROUARD et Jean-Yves RAMEL

Images LiDAR

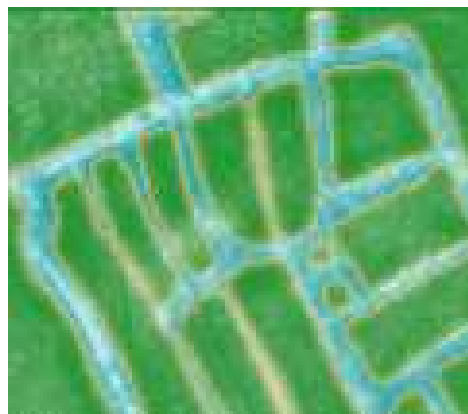
LiDAR est une technologie de télédétection permettant entre autres de représenter les micros reliefs d'un terrain tout en éliminant la couverture végétale. Ces images font apparaître des artefacts invisibles sur les prises de vue classiques ou sur le terrain.



Superposition d'une prise de vue aérienne et image LiDAR d'une zone boisée.

Objectifs

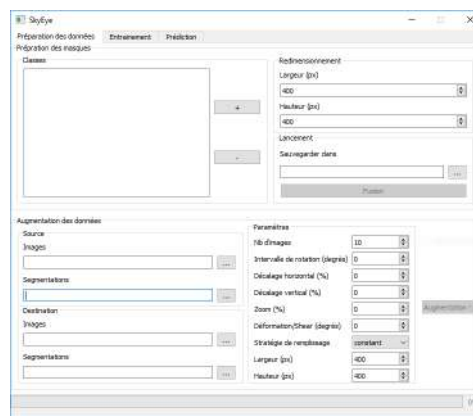
Ces images sont une source d'information nouvelle pour les archéologues et ils ont aujourd'hui accès à une grande quantité de données de ce type. Jusqu'à présent on identifiait manuellement les artefacts sur les images, l'objectif du projet est d'explorer les possibilités du Deep Learning afin de proposer une segmentation automatique.



Superposition d'une image LiDAR et d'un masque de segmentation de talus.

Réalisation

L'outil alors proposé permet de bout en bout de préparer les données, entraîner un modèle choisi parmi une sélection de réseaux de neurones, l'évaluer et enfin segmenter automatiquement de nouvelles images. Cet outil peut alors être utilisé pour tout type d'images.



Outil SkyEye version Deep Learning.

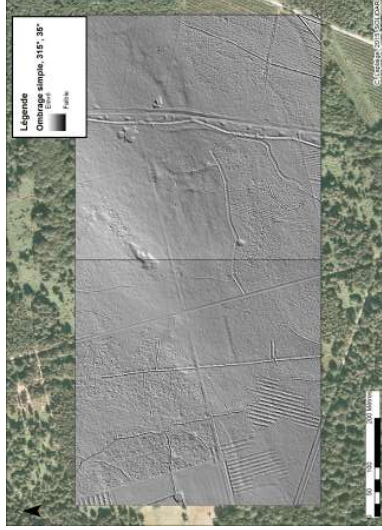
Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR

Valentin MAURICE

Encadrement : Thierry BROUARD et Jean-Yves RAMEL

Images LiDAR

LiDAR est une technologie de télédétection permettant entre autres de représenter les micro-reliefs d'un terrain tout en éliminant la couverture végétale. Ces images font apparaître des artefacts invisibles sur les prises de vue classiques ou sur le terrain.



Superposition d'une prise de vue aérienne et image LiDAR d'une zone boisée.



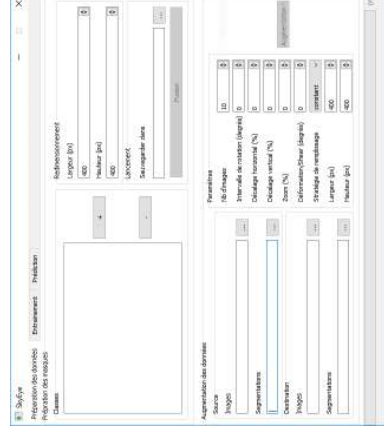
Superposition d'une image LiDAR et d'un masque de segmentation de talus.

Objectifs

Ces images sont une source d'information nouvelle pour les archéologues et ils ont aujourd'hui accès à une grande quantité de données de ce type. Jusqu'à présent on identifiait manuellement les artefacts sur les images, l'objectif du projet est d'explorer les possibilités du Deep Learning afin de proposer une segmentation automatique.

Réalisation

L'outil alors proposé permet de bout en bout de préparer les données, entraîner un modèle choisi parmi une sélection de réseaux de neurones, l'évaluer et enfin segmenter automatiquement de nouvelles images. Cet outil peut alors être utilisé pour tout type d'images.



Outil SkyEye version Deep Learning.

Interactive Deep Learning : Application à la reconnaissance d'éléments archéologiques dans les images LiDAR

Résumé

Ce document est mon rapport de projet recherche et développement du département informatique de Polytech Tours, réalisé lors de la dernière année du cycle ingénieur. L'objectif était d'explorer les possibilités offertes par les techniques d'apprentissage profond actuelles afin de répondre aux besoins des archéologues : segmenter automatiquement certains artefacts qui apparaissent sur des images LiDAR. Je propose à l'issue de ce projet, un outil de segmentation automatique proposant différents réseaux de neurones et pouvant être utilisé avec tout type d'images.

Mots-clés

LiDAR, Python, CNN, Apprentissage Profond, Segmentation, Topographie

Abstract

This document is my research and development project report from the IT department of Polytech Tours, achieved through the last year of the engineering course. The main goal was to look into the current state of the art of Deep Learning technics to fulfill the needs of the archaeologists : to automatically segment artifacts from LiDAR images. At the end of this project, I am able to propose a segmentation tool with different neural networks that can be used with a wide range of images.

Keywords

LiDAR, Python, CNN, Deep Learning, Segmentation, Topography

Tuteurs académiques

Thierry BROUARD

Jean-Yves RAMEL

Étudiant

Valentin MAURICE (DI5)