

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

2018-2019

Implémentation d'une méthode de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie (SkyEye)

Entreprise

Laboratoire Archeologie et Territoires



Tuteurs entreprise

Clement LAPLAIGE

Xavier RODIER

Étudiant

Xavier BOUCHENARD (DI5)

Tuteurs académiques

Jean-Yves RAMEL

Thierry BROUARD

Liste des intervenants

Entreprise

Laboratoire Archeologie et Territoires
40, rue James Watt
37200 Tours

citeres.univ-tours.fr/spip.php?rubrique31



Nom	Email	Qualité
Xavier BOUCHENARD	xavier.bouchenard@etu.univ-tours.fr	Étudiant DI5
Jean-Yves RAMEL	jean-yves.ramel@univ-tours.fr	Tuteur académique, Département Informatique
Thierry BROUARD	thierry.brouard@univ-tours.fr	Tuteur académique, Département Informatique
Clement LAPLAIGE	clement.laplaige@univ-tours.fr	Tuteur entreprise
Xavier RODIER	xavier.rodier@univ-tours.fr	Tuteur entreprise



Avertissement

Ce document a été rédigé par Xavier Bouchenard susnommé l'auteur.

L'entreprise Laboratoire Archeologie et Territoires est représentée par Clement Laplaige et Xavier Rodier susnommés les tuteurs entreprise.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Jean-Yves Ramel et Thierry Brouard 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

Xavier Bouchenard, *Implémentation d'une méthode de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie (SkyEye)*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={Bouchenard, Xavier},
  title={Implémentation d'une méthode de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie (SkyEye) },
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2018-2019}
}
```

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 Les acteurs du projet	1
2 Contexte	2
3 Objectifs	3
3.1 Taux de reconnaissance faible	3
3.2 Interaction limitée avec le système	3
4 Bases méthodologiques	3
I Recherche	5
2 Description générale	6
1 Environnement du projet	6
2 Caractéristiques des utilisateurs	6
3 Sources fournies par la MOA	6
4 Fonctionnalités du système	7
5 Structure générale du système	8

3	État de l'art	13
1	Intelligence Artificielle.....	13
1.1	L'intelligence Artificielle faible	14
1.2	L'intelligence Artificielle forte	14
2	Reconnaissance de formes.....	14
2.1	L'apprentissage automatique.....	15
2.1.1	L'apprentissage supervisée	15
2.1.2	L'apprentissage non-supervisée	16
2.1.3	L'apprentissage par renforcement.....	16
3	La classification linéaire.....	16
3.1	Le Séparateur à Vaste Marge	18
4	Veille technologique	19
1	L'apprentissage profond ou Deep Learning.....	19
1.1	Résumé sur le Perceptron	19
1.2	Perceptrons multi-couches	21
2	Réseaux de neurones profonds.....	23
2.1	Couches d'extraction des fonctionnalités et apprentissage	24
2.1.1	Couche de Convolution (Conv).....	24
2.1.2	Couche Pooling	24
2.1.3	Couche ReLu.....	25
2.2	Couches de classification.....	26
3	Techniques pour améliorer la précision du modèle	26
3.1	L'ajout de données	27
3.2	Le traitement des données incohérentes et des oublis.....	27
3.3	La sélection des fonctionnalités à appliquer.....	27
3.4	L'utilisation d'algorithmes.....	28
3.5	Le paramétrage du classifieur	28
3.6	La validation croisée.....	28
5	Analyse et conception	30
1	Spécifications fonctionnelles.....	30
1.1	Amélioration du taux de reconnaissance	30
1.1.1	Description	30
1.2	Algorithme de recherche de la configuration optimale du réseau de neurones	32
1.2.1	Description	32
1.3	Chargement des fichiers CSV et séparation des fichiers sous forme de bases	32
1.3.1	Description	32
1.4	Interface graphique de l'application implémentant le réseau de neurones	33
1.4.1	Description	33

1.5	Conception de l'interface graphique de visualisation de l'application SkyEye	36
1.5.1	Description	36
2	Analyses approfondies	38
2.1	Bibliothèque de Deep Learning à utiliser	38
2.1.1	Facilité d'utilisation.....	40
2.1.2	Communauté et support	40
2.1.3	Outils embarqués.....	40
2.1.4	Rapidité d'entraînement du modèle créé.....	40
2.1.5	Nombre minimum de lignes de code	41
2.1.6	Modèles de références implémentés	42
2.1.7	Ne nécessite pas un environnement de fonctionnement spécifique?	42
2.1.8	Connaissances sur cette bibliothèque	42
2.1.9	Choix final de la bibliothèque à utiliser.....	42
2.2	Recherche des paramètres optimaux	43
2.2.1	Algorithme de recherche des paramètres optimaux.....	43
2.2.2	Etude de la fonction d'activation à implémenter.....	44
6	Mise en oeuvre	48
1	Bibliothèques utilisées.....	48
2	Recherche des paramètres optimaux	48
2.1	Tests.....	48
2.2	Conclusion	49
7	Bilan et conclusion	50
1	Organisation du S9	50
2	Organisation du S10.....	53
3	Conclusion du projet	54
	Annexes	55
	A Diagramme de cas d'utilisation	56
	B Cahier des spécifications	57
	C Guide utilisateur	76
	D Guide développeur	82
	Webographie	87
	Bibliographie	88

Table des figures

1 Introduction

1	Logo du groupe RFAI du Laboratoire Informatique de Tours.....	2
2	Logo du projet SOLiDAR et du laboratoire CITERES de Tours	2
3	Exemple d'image de relief prise à partir d'image LiDAR.....	2

2 Description générale

1	Exemple des images sources à fournir par la MOA	7
2	Fonctionnement global de l'application	9
3	Contenu type du fichier d'apprentissage	10
4	Contenu type du fichier à prédire.....	10
5	Diagramme de classes de l'application à développer	11

3 État de l'art

1	Etapes composant un système de reconnaissance de formes	15
2	Exemple de classification linéaire	17
3	Frontière de classification linéaire	18

4 Veille technologique

1	Structure d'un Perceptron	20
2	Exemple de Perceptron multi-couche à une seule couche intermédiaire	22
3	Architecture d'un réseau de neurones convolutif	23
4	Etapes du traitement d'une image par un CNN	23
5	Convolution entre le masque défini et l'image.....	24
6	Traitement de la couche Pooling	25

7	Traitement effectué par la couche ReLu	26
8	Exemple de données manquantes et impacts	27
9	Algorithmes présents dans la bibliothèque standard	28
10	Validation croisée	29
5	Analyse et conception	
1	Structure d'un Perceptron	31
2	Structure d'un CNN	31
3	Séparation des données du fichier d'apprentissage en bases	33
4	Diagramme des cas d'utilisation de l'application à développer	34
5	Interface graphique de la nouvelle application proposée	35
6	Diagramme des cas d'utilisation de l'interface graphique	36
7	Maquette de l'interface graphique proposée	37
8	Comparatif des bibliothèques de Deep Learning Python	39
9	Résultat de l'étude pour la rapidité d'entraînement	41
10	Algorithme de recherche de paramètres	44
11	Fonction Sigmoidale	45
12	Fonction Tangente Hyperbolique	45
13	Fonction ReLu	46
14	Probabilités pour chaque classe pour un pixel	47
6	Mise en oeuvre	
1	Tableau récapitulatif pour un taux d'apprentissage de 0.01	49
2	Tableau récapitulatif pour un taux d'apprentissage de 0.02	49
7	Bilan et conclusion	
1	Découpage de la phase Recherche du projet en sous-tâches	51
2	Diagramme de Gantt	52
3	Diagramme de Gantt du S10	53
A	Diagramme de cas d'utilisation	
1	Diagramme de cas d'utilisation	56

1

Introduction

Dans le cadre de la 5ème et dernière année d'ingénieur à Polytech'Tours, un Projet de Recherche et Développement est confié aux étudiants pour leur permettre de travailler sur un projet de grande envergure et d'appliquer les bases méthodologiques et connaissances apprises en cours. Ce projet se déroule du 19 septembre jusqu'à fin mars 2019 où par semaine, 2 jours lui sont consacrés, soit 16 heures par semaine.

A l'issu de ce projet, 2 rapports seront rédigés ainsi qu'un cahier de spécifications, dans lequel sera défini les attentes du client ainsi que les moyens pour y parvenir. Deux soutenances orales sont également prévues pour présenter le travail effectué pendant ces séances.

Le sujet de ce Projet de Recherche et Développement (PRD), vise l'amélioration d'une application de reconnaissance de reliefs dans des images archéologiques. Cette application consiste à extraire toutes les caractéristiques (zones d'intérêt) des pixels des images dont les reliefs sont identifiés, pour ensuite les analyser et déterminer à quel type de relief appartient le pixel d'images passées en paramètre de cette application.

Ce projet est l'objet d'une étude dans laquelle les besoins du client sont exprimées puis résolues grâce à l'expertise technique et les connaissances acquises de l'étudiant, ainsi que de ses encadrants techniques.

Ce document est donc le rapport de la partie **Recherche** du projet « *Implémentation du principe de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie* ».

1 Les acteurs du projet

- La maîtrise d'œuvre (MOE) : Xavier Bouchenard, étudiant en 5ème année de l'Ecole d'Ingénieur Polytech Tours au sein du département Informatique, dans le cadre du PRD ainsi que ses encadrants : Mr Jean-Yves Ramel et Mr Thierry Brouard de l'équipe de recherche RFAI – EA 6300 du Laboratoire Informatique de Tours (<http://www.rfai.li.univ-tours.fr>).

Les recherches menées au sein de cette équipe portent sur l'étude et la résolution des problèmes de reconnaissance d'images, que ce soit dans le cadre de problématiques "industrielles" ou de problèmes théoriques.

- La maîtrise d'ouvrage (MOA) : Le projet SOLiDAR du laboratoire Archéologique et Territoire CITERES de Tours, représenté par Clément LAPLAIGE et Xavier RODIER. Créé



Figure 1 – Logo du groupe RFAI du Laboratoire Informatique de Tours

en 2004, ce laboratoire basé sur Tours a pour objectif d’analyser les dynamiques spatiales et territoriales des sociétés. Dans ce contexte, leur champ de recherche se compose de quatre secteurs : la recherche urbaine, la recherche environnementale, les travaux sur le territoire et ceux sur les effets des recompositions sociales contemporaines. (<http://citeres.univ-tours.fr>)



Figure 2 – Logo du projet SOLiDAR et du laboratoire CITERES de Tours

Ce document a été rédigé par Xavier Bouchenard et sera validé par les encadrants du projet et l’ensemble des personnes constituant la MOA qui sont également les relecteurs de ce cahier.

2 Contexte

L’origine de ce projet provient du travail d’archéologues utilisant un nouveau système de détection basé sur la technologie LiDAR ou Light Detection And Ranging permettant d’acquérir des images géographiques. Ensuite, à partir des images obtenues, de nouvelles images seront générées en fonction de leurs besoins. L’une des images générées qui sera par la suite, le plus utilisée représente un relief en nuances de gris, enregistrée au format .TIF.

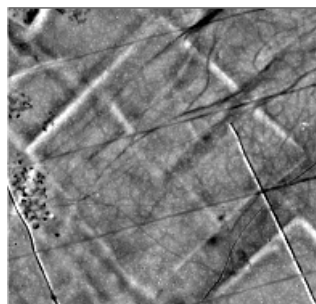


Figure 3 – Exemple d’image de relief prise à partir d’image LiDAR

Pour répondre à une demande d’automatiser le process de détection de zones archéologiques, une première application a été développée et est actuellement utilisée par les archéologues. Ce logiciel permet d’identifier la nature des pixels de chaque région de l’image (fossé, talus, chemin, ...) grâce à des calculs de probabilités. Pour se faire, une méthode de reconnaissance linéaire appelée **SVM** (Séparateur à Vaste Marge) a été implémenté.

Suite à l’utilisation de ce logiciel, quelques remarques ont été faites par le client liées au taux de reconnaissance de l’application, jugée faible, environ 60% de reconnaissance correcte dans certains cas.

3 Objectifs

La principale amélioration à apporter sur cette application est donc l'amélioration de la précision du modèle. Il s'agit d'un taux correspondant au nombre de pixels identifiés appartenant au relief prédit sur le nombre de pixels de toutes les images analysées. Cette amélioration consiste donc à trouver une méthode permettant de corriger le manque de précision du modèle pour obtenir un taux de reconnaissance plus élevé.

Ce problème a été constaté par le client, le taux de reconnaissance des pixels ne s'élève pas au-delà de 65%, dans certains cas. Il est donc important de l'améliorer.

Une problématique également abordée est la difficulté, pour l'utilisateur, de modifier des classifications effectuées par l'algorithme **SVM**, dans le cas où celui-ci se serait trompé. De ce côté-là, l'interaction entre l'application et l'utilisateur est limitée. Une fonctionnalité supplémentaire dédiée à ce problème a été pensée.

Malgré ce qui a été vu précédemment, les clients ont fait la demande pour corriger le manque de précision de l'algorithme **SVM** par l'implémentation d'un réseau de neurones. C'est pourquoi ce problème sera donc traité en priorité.

Pour éviter d'éventuels problèmes d'interactions entre les deux méthodes de résolution, une nouvelle application sera développée autour du réseau de neurones construit, ce qui permettra d'éviter quelques problèmes de compatibilité comme la version des interpréteurs.

3.1 Taux de reconnaissance faible

La reconnaissance des pixels est basée sur une méthode de résolution linéaire, ce qui fait que si le système n'a pas suffisamment d'exemples dont la nature et les caractéristiques sont différentes (taux de luminance, ...), il aura plus de chances de prendre une mauvaise décision.

Le problème lié à ça est qu'il est difficile de juger quand la machine a suffisamment d'exemples, pour pouvoir construire un modèle et pouvoir juger s'il est suffisamment précis pour se baser dessus pour les prochaines décisions à prendre.

C'est pourquoi l'une des méthodes de résolution serait soit d'implémenter une méthode de Deep Learning par la mise en place d'un réseau de neurones ou par l'amélioration de la précision du classifieur SVM. Les membres constituant la MOA de ce projet ont formulé l'envie d'implémenter une méthode de Deep Learning, une première version d'application sera développée embarquant comme méthode de reconnaissance un réseau de neurones. Le fait de dissocier, dans un premier temps, les deux méthodes de résolution, permet de comparer plus facilement les résultats obtenus.

3.2 Interaction limitée avec le système

Dans le cas où le système fait une erreur de jugement, l'utilisateur doit pouvoir être capable de modifier la décision prise. Il faut ensuite que la modification faite, soit prise en compte par la suite dans le modèle, pour pouvoir être reconstruit et éviter de faire la même erreur par la suite.

4 Bases méthodologiques

Pour ce projet, la méthode de gestion utilisée est la méthode Scrum, ce qui permet au client de suivre plus facilement le projet grâce au découpage du projet en tâches, associées à

des sprints (livraison d'une fonctionnalité après réalisation dans un temps donné). Le suivi du projet sera plus simple par le client, une fois que des fonctionnalités seront livrées, elles pourront être testées et validées par le client en personne.

En ce qui concerne le développement, il se fera en langage Python avec l'IDE PyCharm qui permet facilement de gérer les bibliothèques utilisées. Une convention de nommage a également été définie, il s'agit de la norme *PEP 8*.

Première partie

Recherche

2

Description générale

1 Environnement du projet

L'application a été développée en Python pour des machines fonctionnant sous Windows avec une version d'interpréteur supérieure ou égale à la 3.6.2 et les modules torch et torchvision pour l'implémentation et l'utilisation d'un réseau de neurones.

Un Git est utilisé afin de gérer le versionning du projet.

2 Caractéristiques des utilisateurs

L'application est prévue pour être utilisée par des archéologues, pour leur permettre de faciliter leur travail. Pour que l'utilisateur puisse utiliser correctement l'application, il faut qu'il sache comment utiliser l'application de façon instinctive et qu'elle soit le plus simple possible. Il n'a cependant pas besoin de connaître et de savoir paramétrer le réseau de neurones : un algorithme déterminera les meilleurs paramètres pour cela.

3 Sources fournies par la MOA

La MOA doit fournir deux types d'images sources : des images dites "originelles" en niveau de gris ainsi que des images dites "labellisées" binaires. Ces deux types d'images seront au format TIFF (256 nuances, 8 bits).

Les images labellisées représentent une image originelle pour une classe de reconnaissance (elles devront avoir le même nom). Un pixel noir dans cette image binaire signifie que ce pixel appartient à la classe que concerne l'image labellisée.

Dans l'exemple ci-dessous, les images sources permettent de différencier cinq classes différentes dans ce modèle. Nous pouvons remarquer ici que seuls les classes trois et quatre sont présentes dans cette image source. Le nom en-dessous de chaque image correspond au nom du dossier dans lequel elles se trouvent respectivement. En effet, si elles doivent toutes avoir le même nom, elles ne peuvent donc pas être stockées dans le même répertoire.

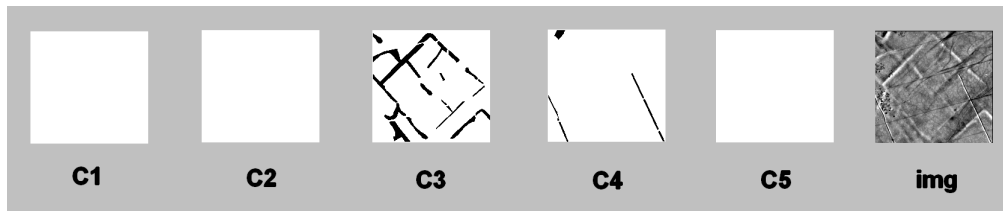


Figure 1 – Exemple des images sources à fournir par la MOA

Ces images seront par la suite traitées par l'application **SkyEye**, actuellement utilisée par les archéologues, qui va permettre d'extraire toutes les caractéristiques de l'image et d'enregistrer ces valeurs dans des fichiers CSV portant le même nom que l'image. Ce fichier contient en ligne, des pixels dont les données ont été extraites et en colonnes, il s'agit des principales caractéristiques retrouvées dans l'image. Ces données sont labellisées, c'est-à-dire que pour chaque pixel, un nom de classe connue lui est attribué.

Ce fichier peut être qualifié de "**fichier d'apprentissage**" : il contient toutes les informations de tous les pixels et, une fois appris, ces informations peuvent permettre à un modèle construit à partir d'une méthode de reconnaissance, de prédire l'appartenance d'autres pixels non classifiés en fonction des valeurs suivant les caractéristiques extraites.

Les images dont les pixels sont à labelliser subissent le même traitement, à la différence que ces pixels ne portent pas de labels, ils seront donc à déterminer. Ces informations sont également stockées dans un fichier CSV portant le même nom que l'image initiale dont on souhaitait identifier les pixels. Dans ce cadre, ce fichier peut être qualifié de "**fichier à prédire**" : les pixels doivent être étiquetés en fonction des données apprises par le modèle entraîné.

4 Fonctionnalités du système

L'application à développer se base principalement sur l'application SkyEye, qui embarque la méthode de reconnaissance SVM. Les fonctionnalités caractérisant cette nouvelle application sont les suivantes :

1. **Chargement des fichiers CSV** : l'utilisateur définit le chemin d'accès aux fichiers d'apprentissage et à prédire, puis les charge dans l'application. Cette étape correspond au chargement des données des fichiers pour pouvoir être ensuite utilisées
2. **Chargement du fichier contenant le modèle** : cette étape consiste à charger un modèle de réseau de neurones qui a été précédemment construit et que l'utilisateur a jugé bon en raison d'un taux de reconnaissance suffisamment élevé. Il choisit donc le fichier, puis le charge pour que la structure du réseau soit reconstruite et réutilisable
3. **Sauvegarde du modèle dans un fichier** : une fois qu'un modèle de réseau de neurones ait été construit et entraîné, si l'utilisateur juge que le modèle actuel possède de bonnes performances de classification, il est possible de sauvegarder sa structure et la base d'apprentissage associée dans un fichier qui sera défini par l'utilisateur
4. **Entraînement du modèle et évaluation** : dès qu'un fichier d'apprentissage a été sélectionné puis chargé, l'utilisateur peut entraîner le modèle grâce aux données du fichier d'apprentissage. Une partie des données de ce fichier sont mis de côté pour constituer une base de validation, c'est-à-dire que l'on calcule le taux de reconnaissance sur ces données en lançant une prédiction sur celles-ci. Le résultat de cette prédiction ou évaluation, est affichée à l'écran
5. **Paramétrage du classifieur** : pour obtenir un taux de reconnaissance optimal, il faut que le réseau de neurones soit le plus adapté possible. Un algorithme sera donc exécuté pour déterminer la meilleure configuration à partir des données du fichier d'apprentissage

6. **Lancement de la classification** : dès que le modèle a été construit et entraîné, si un fichier dont les pixels sont à prédire est sélectionné et chargé, il est possible de lancer une classification en passant en paramètre du modèle, les données de ce fichier. On obtiendra en sortie une matrice contenant pour chaque pixel, sa probabilité d'appartenance à une classe, ainsi que les labels associés aux pixels. Ces résultats seront stockés dans deux fichiers

Le diagramme des cas d'utilisation présent en annexe en annexe A, récapitule tous les points vus jusqu'à maintenant.

Avant de lancer l'identification, il faut que le modèle préalablement construit à partir des images de configuration et des échantillons de reliefs étiquetés, soit entraîné et sache différencier les reliefs ou classes d'appartenance.

L'application principale appelée **SkyEye** a fait l'objet de travaux de PRDs successifs visant son amélioration, le dernier travail en date a été fait par Ronan Guillaume et consistait en l'amélioration de l'implémentation des algorithmes utilisés ainsi que de l'interface graphique, on peut le retrouver ici : [1].

L'application qui sera développée est en lien avec **SkyEye** : cette application prend en entrée un fichier d'apprentissage au format CSV, pour la construction du réseau de neurones et un fichier à prédire, au format CSV, dont les pixels doivent être classifiés.

5 Structure générale du système

Le système est composé de plusieurs partis :

1. **L'interface Homme/Machine** permettant à l'utilisateur de charger les fichiers CSV d'apprentissage et à prédire, de charger un fichier contenant un modèle ou de le sauvegarder
2. **La lecture du contenu de ces fichiers et le chargement de leurs données** qui constituent des jeux de données appelés bases : base d'apprentissage, base de test, base de validation et base à prédire
3. **La classe DataHandler** : , qui va contenir toutes les informations que ce soit du modèle construit pour l'entraînement ou le lancement d'une reconnaissance ou les données des fichiers chargés, stockées sous forme d'objets
4. **L'apprentissage lancée** par l'utilisateur sur des données dont le label est à prédire, par l'appui d'un bouton, une fois que le modèle est construit et entraîné

L'architecture de l'application est résumée par le schéma ci-dessous :

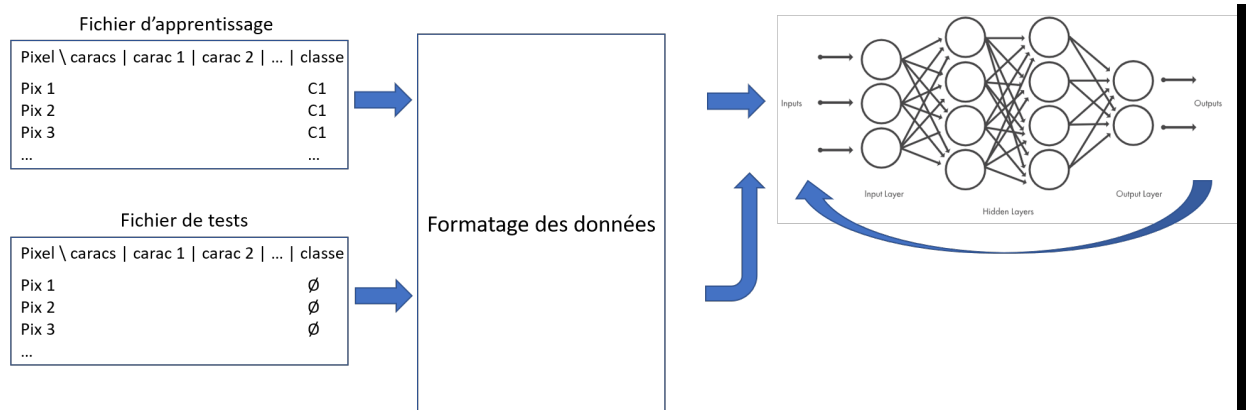


Figure 2 – Fonctionnement global de l'application

On laisse l'utilisateur passer en entrée de l'application un fichier correspondant au fichier d'apprentissage qui est un fichier CSV contenant l'ensemble des pixels d'une ou plusieurs image(s) avec pour chacun un label associé. On passe également un fichier à prédire où, cette fois, les pixels n'ont pas de labels et doivent être déterminés.

On crée par la suite plusieurs bases dans lesquelles les données des pixels sont stockées, puis utilisées sous forme de tenseurs qui est le type de données que prend le réseau de neurones en paramètre. S'il n'est pas défini, on construit un modèle à partir des données du fichier d'apprentissage et on fait apprendre celles d'entraînement à ce nouveau réseau.

On pourra, par la suite, lancer une prédiction sur les données à prédire en passant en paramètre du réseau de neurones, le tenseur contenant ces données.

Le contenu de chaque fichier CSV sélectionné et chargé est approximativement le suivant :

1. Le fichier d'apprentissage :

Class	x	y	AvgD_15x15	AvgD_19x19	AvgD_3x3
C0	0	307	44.46	99.18	25.5
C0	0	700	21.4	18.89	16.75

Figure 3 – Contenu type du fichier d'apprentissage

L'image ci-dessus n'est qu'un échantillon d'un fichier d'apprentissage, il comporte bien plus de caractéristiques et de données. Il est caractérisé par le fait que des labels sont donnés aux pixels pour faire référence au type de relief présent dans l'image initiale.

2. Le fichier à prédire :

Class	x	y	AvgD_15x15	AvgD_19x19
	0	174	30.85	21.21
	0	224	18.44	18.3
	0	440	23.54	16.2

Figure 4 – Contenu type du fichier à prédire

L'image ci-dessus n'est qu'un échantillon d'un fichier à prédire, il comporte bien plus de caractéristiques et de données. Il est caractérisé par le fait qu'aucun label n'est donné à un pixel (première colonne). Ces labels sont donc à déterminer.

Le diagramme de classes présent sur la page suivante, reprend toutes les fonctionnalités qui sont aujourd'hui implémentées.

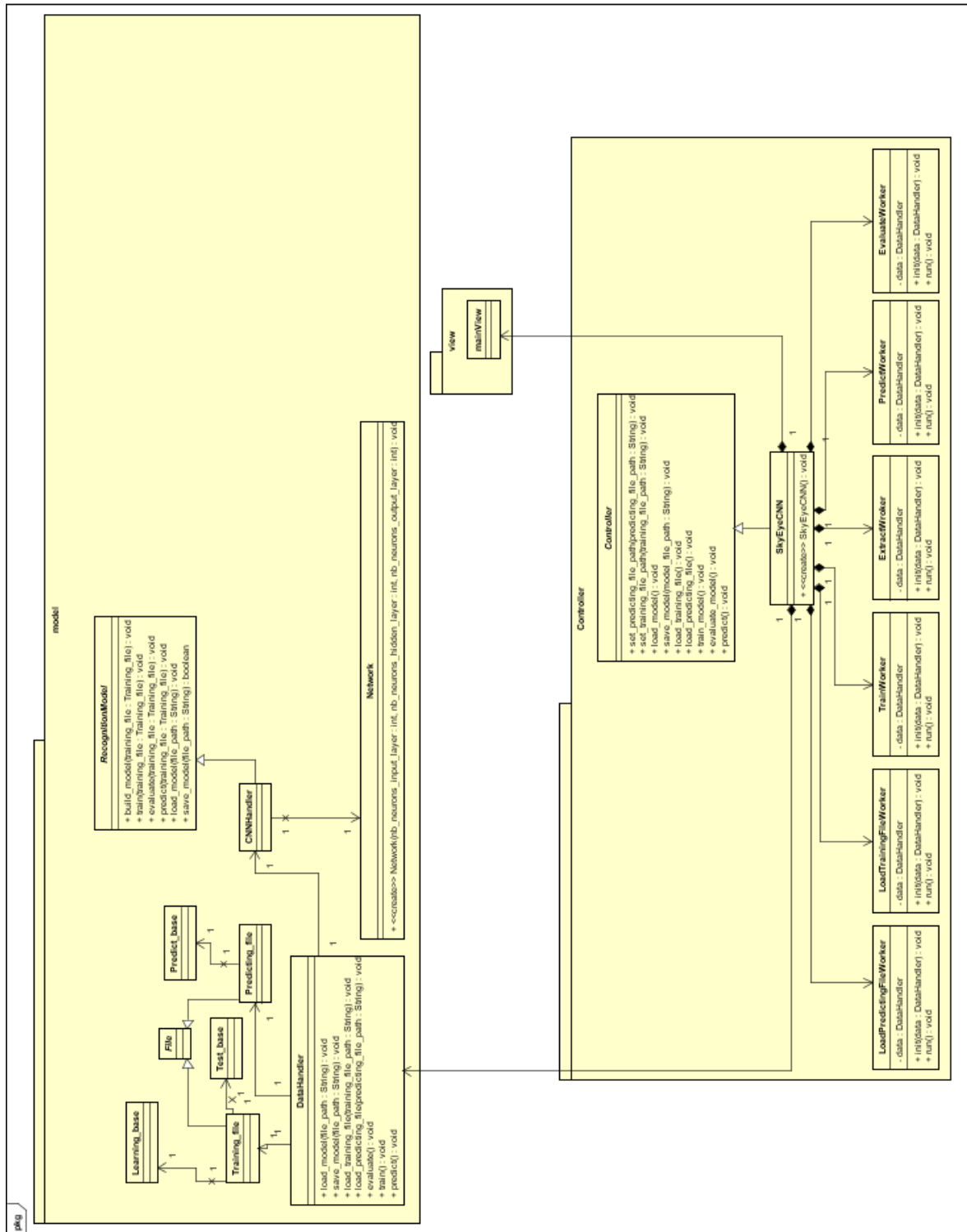


Figure 5 – Diagramme de classes de l'application à développer

Pour rendre le code plus lisible, le projet a été découpé en package de fonctionnalités. Les principaux packages sont les suivants :

1. **Model** : contient toutes les classes utilisant directement ou stockant des données de fichiers sous forme d'objets comme le modèle de réseau de neurones à construire ou les données extraites sous forme de bases des fichiers CSV qui auront été chargés
2. **View** : correspond à l'unique vue graphique de l'application, gérant les appels à des méthodes d'un contrôleur
3. **Controller** : gère les actions de l'utilisateur par des méthodes appelées lors de l'appui sur un bouton de l'interface graphique de l'application

3

État de l'art

Dans cette partie, nous verrons les plus grandes techniques implémentées, qui permettent à l'application de prendre des décisions, quant à l'appartenance des pixels à des classes ou à des reliefs déjà connus.

Nous verrons la notion d'Intelligence Artificielle, un concept qui permet à un système informatique de prendre des décisions, à la façon naturelle d'un humain. Nous parlerons du rôle des classificateurs de données, très utilisés pour la séparation et le regroupement d'un grand volume de données en fonction de certaines caractéristiques, plus particulièrement du SVM ou **Séparateurs à Vaste Marge**.

1 Intelligence Artificielle

Selon [WWW6], "l'intelligence artificielle peut être définie comme l'ensemble de théories et de techniques mises en œuvre pour réaliser des machines capables de simuler l'intelligence. Elle permet de rechercher des méthodes de résolution de problèmes logiques ou algorithmiques complexes."

L'intelligence artificielle est principalement utilisée pour reproduire ou remplacer l'humain pour certaines actions. Aujourd'hui, elle est très utilisée pour " la compréhension de langage naturel (langage parlé) mais aussi dans la perception visuelle (interprétation des images et des scènes), la perception auditive (utilisée pour la compréhension du langage parlé). " selon [WWW6].

Cette science a vu le jour grâce à un article nommé « Computing Machinery and Intelligence » rédigé par Alain Turing en octobre 1950; dans lequel il décrit une expérience qui consiste à faire communiquer 1 humain qui joue le rôle de juge impartial avec, dans un premier temps, une personne et ensuite une machine par l'intermédiaire d'un terminal. Le but étant de laisser décider le juge sur la nature des deux interlocuteurs avec lesquels il a interagit. Au final, ce célèbre test nommé « Test de Turing » permet de déterminer si une machine peut être qualifiée comme étant « consciente » ou non.

Ce test a permis de mettre en lumière la qualité des décisions prises par la machine et d'améliorer les méthodes pour faire en sorte qu'elle puisse apprendre de ses erreurs.

De ce test sont ressortis deux types d'intelligence artificielle avec leurs applications spécifiques :

1. L'intelligence artificielle faible
2. L'intelligence artificielle forte

1.1 L'intelligence Artificielle faible

Le but de ce type est la construction de systèmes alliant autonomie dans la prise de décisions avec le plus petit taux de supervision possible. Ce concept permet à ce qu'une machine simule une « intelligence » qui est propre aux actions qu'elle effectue et se base sur le principe d'auto-apprentissage (dans le cas d'une erreur, avec ou sans aide, la machine est capable de corriger une action ou une mauvaise décision prise).

Ce concept s'applique pour des systèmes utilisés en reconnaissance d'images, traduction automatique, reconnaissance vocale et bien d'autres domaines où l'utilisateur a une influence de décision sur le système (aide à l'ajustement du système après un retour positif ou négatif du service rendu par la machine, i.e « étape de calibrage »). Le **Machine Learning** et le **Deep Learning** sont des méthodes plus poussées pour répondre à ces problèmes, en se basant sur ce concept.

1.2 L'intelligence Artificielle forte

Ce concept s'applique principalement pour des machines comme les robots dernière génération dont l'objectif est « de produire un comportement « intelligent » mais aussi d'éprouver une impression réelle de conscience de soi, de « vrais » sentiments et une compréhension de ses propres raisonnements. » selon [WWW6]. Il s'agit de tenter de lui donner une « conscience » et ressentir des émotions, ce qui semble malheureusement impossible aujourd'hui en raison de notre technologie limitée.

Après cela, nous allons définir la notion de reconnaissance de formes.

2 Reconnaissance de formes

Ce thème est un domaine d'application dans lequel l'intelligence artificielle est, aujourd'hui, très utilisée. Il s'agit de " l'ensemble de techniques et méthodes visant à identifier des motifs " selon [WWW9], venant de données comme des images ou photos afin de prendre une décision quant à l'appartenance du motif à une catégorie ou classe. Plusieurs algorithmes d'apprentissage automatique existent pour traiter ce problème, nous en verrons quelques-unes par la suite.

Le diagramme ci-dessous récapitule les phases par lesquels un système passe pour reconnaître une image (processus très simplifié).

Les étapes détaillées sont les suivantes :

1. **L'acquisition** : récupération de l'information dans sa forme originelle (images, données, ...)
2. **Le pré-traitement** : étape qui touche à la mise en forme et à la conversion des données initiales pour les mettre dans un format acceptable pour le traitement (filtrage, élimination du bruit, redondance, ...)
3. **L'analyse** : exécution d'une ou plusieurs méthodes consistant à déterminer l'appartenance à une classe d'une donnée en fonction de représentations et caractéristiques spécifiques (calcul de caractéristiques, détermination d'indices, ...)

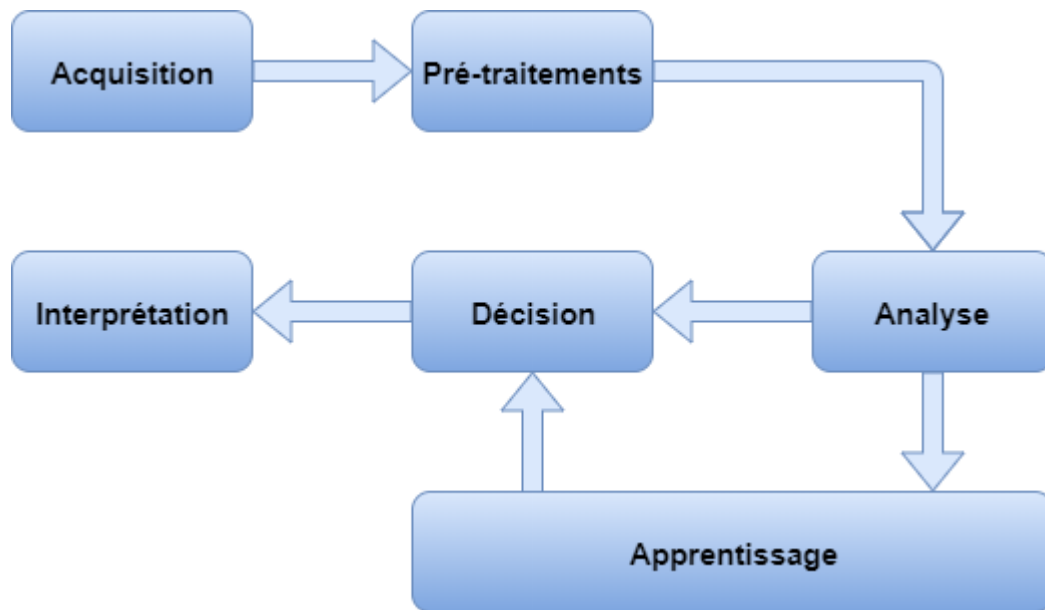


Figure 1 – Etapes composant un système de reconnaissance de formes

4. **L'apprentissage** : étape où le système exploite la connaissance et en extrait les prototypes de classe ou les modèles associés pour ensuite prendre une décision sur la nature de la donnée
5. **La décision** : étape qui consiste à classer la donnée sur la base d'une sûreté d'appartenance : définitif, ambigu avec score ou rejet (modification de la décision prise du système par l'expert si le système est en phase d'apprentissage : supervisé ou non)

Nous allons maintenant voir les différents types d'apprentissage.

2.1 L'apprentissage automatique

Comme nous l'avons vu précédemment, l'apprentissage automatique regroupe toutes les techniques permettant à un système d'apprendre pour pouvoir ensuite faire des prédictions. Ces méthodes changent en fonction du mode d'apprentissage utilisé.

Nous allons voir les trois types spécifiques qui existent.

2.1.1 L'apprentissage supervisée

L'apprentissage supervisée consiste à lancer une « fonction de prédiction » en passant en paramètre à la machine, des images étiquetées appartenant à une classe précise. Alors, le système va « apprendre » à partir des étiquetages effectués et des classes préalablement définies, à calculer une probabilité d'appartenance à une classe d'une image. Le système va alors construire une *base d'apprentissage* qu'il va utiliser pour lui permettre plus facilement de comparer les éléments qu'il connaît constituant cette table, aux éléments que l'utilisateur souhaite identifier (son, image : le plus souvent,...)

Parmi ces techniques, on retrouve les plus connus comme le **Séparateur à Vaste Marge (SVM)**, le **réseau de neurones**, la **classification bayésienne** ou encore l'**Analyse discriminante de Fisher**.

Parmi ces techniques, la seule que nous verrons plus en détail est le *Séparateur à Vaste Marge*.

2.1.2 L'apprentissage non-supervisée

L'apprentissage non-supervisée consiste à tenter de trouver des points communs entre des données dont on a à priori des connaissances et des données non étiquetées (on ignore tout des classes d'appartenance). Le système ou la machine va donc tenter d'effectuer des correspondances, en se basant sur les éléments connus, pour déterminer la nature des éléments à identifier, en effectuant des comparaisons en fonction de critères qu'il définit lui-même. Ces critères sont amenés à changer de manière automatique, en fonction des éléments que l'utilisateur souhaite faire reconnaître à la machine.

Cette méthode est plus difficile à mettre en place car il faut que le système soit capable d'apprendre sans superviseur.

2.1.3 L'apprentissage par renforcement

Il s'agit d'un mode d'apprentissage dans lequel une « entité » comme un programme, qui exécute des tâches de manière répétitives, à apprendre des décisions faites, pour optimiser une récompense qu'elle reçoit à chaque bonne décision prise. Cette manière de procéder encourage constamment une "entité" à faire de bonnes décisions ou à corriger et apprendre de ses erreurs pour toujours être récompensée.

Après avoir introduit toutes ces notions, nous allons aller plus en profondeur pour parler des techniques, qui sont le noyau du *Machine Learning*, ce sont les **classifieurs de données**. Il s'agit d'un type d'algorithme permettant de traiter un gros volume de façon rapide et efficace pour créer une démarcation entre chaque type présent dans ce volume de données. Ce procédé permet de faciliter l'identification de chaque type de donnée présente.

Nous allons parler d'un type plus spécifiquement, qui est actuellement utilisé dans l'application sur laquelle des modifications doivent être apportées, il s'agit de la gamme des classifieurs linéaires.

3 La classification linéaire

Il s'agit d'un type d'algorithmes de catégorisation automatique qui permet de classer des objets qui possèdent des propriétés similaires et de les regrouper.

L'illustration ci-dessous montre les effets de l'utilisation d'un classifieur de données linéaire sur un nuage de données, ainsi que les frontières entre chaque classe.

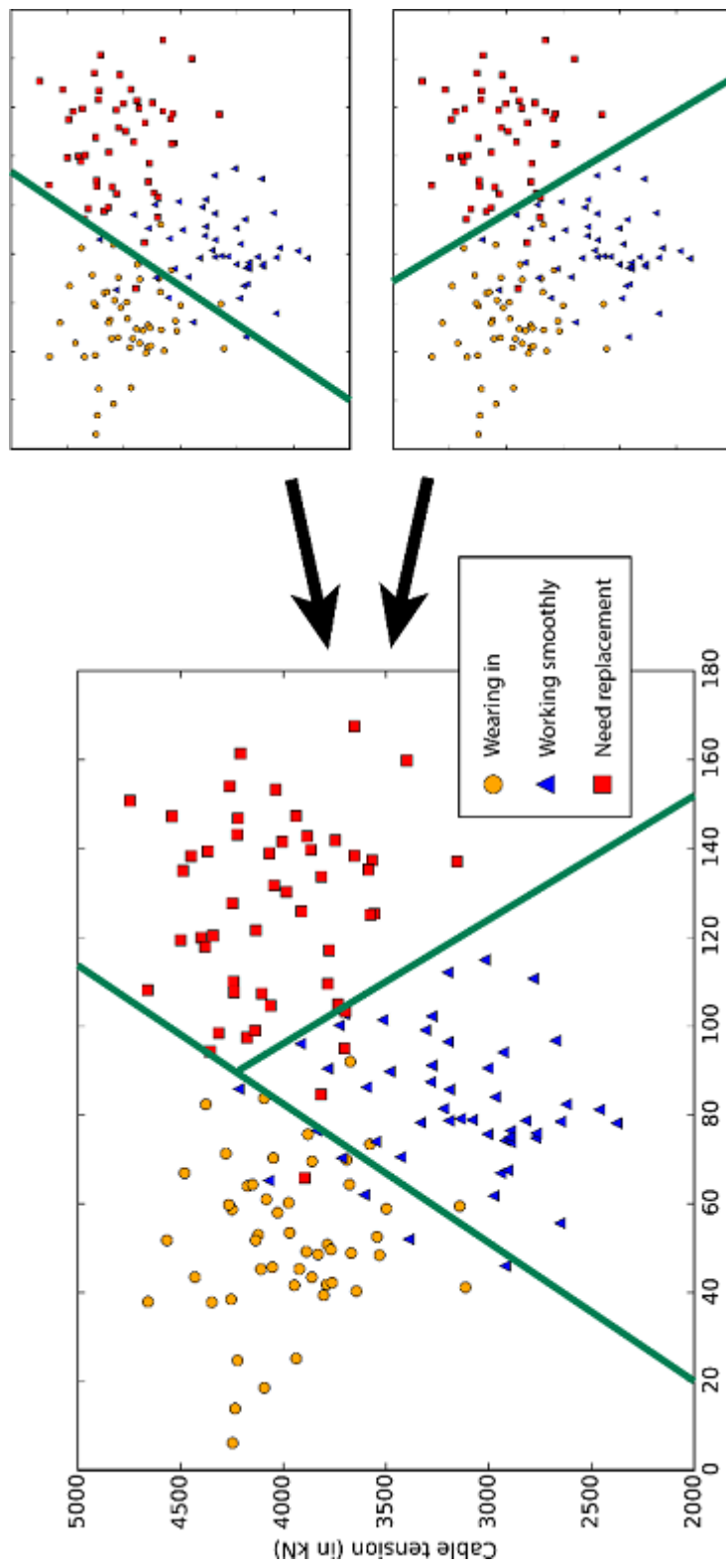


Figure 2 – Exemple de classification linéaire

Cette catégorie de classifieur comporte deux modèles différents :

1. **Le modèle génératif** : consiste à modéliser grâce à des probabilités conditionnelles dont l'algorithme le plus connu est l'*Inférence bayésienne*
2. **Le modèle discriminant** : " cherche d'abord à maximiser la qualité de la classification sur un jeu de test. Dans un second temps, une fonction de coût va réaliser l'adaptation du modèle de classification final (en minimisant les erreurs) " selon [WWW4], dont les algorithmes les plus connus sont le *Perceptron* et le *Séparateur à Vaste Marge* (SVM)

Dans notre cas, nous n'étudierons que le *Séparateur à Vaste Marge*.

3.1 Le Séparateur à Vaste Marge

Il s'agit d'un type de classifieur très utilisé pour la résolution de problèmes de discrimination (prédire l'appartenance de données à un groupe ou classe) et de régression (analyse de la relation de variables entre elles, du moment que les critères soient exprimées de façon numérique). Développée dans les années 1990, cette technique s'appuie grandement sur la théorie de Vapnik-Chervonenkis; il s'agit d'une théorie statistique où quatre grands points sont pris en compte :

1. **L'uniformité des apprentissages** : on se pose la question si pour chaque technique, l'apprentissage est fait, de base, pour minimiser le risque d'erreur
2. **Leur taux de convergence** : on essaie de déterminer la vitesse de convergence pour trouver la solution à un problème donné pour un apprentissage
3. **La capacité de contrôle de l'apprentissage** : à quel point la capacité de généralisation de l'apprentissage est efficace pour la classification et l'identification de données à partir de classes existantes
4. **Leur construction** : comment faire pour que des algorithmes puissent répondre aux critères précédents, tout en restant efficace et en étant simple d'utilisation ?

Toutes ces informations peuvent être retrouvées ici, [WWW7].

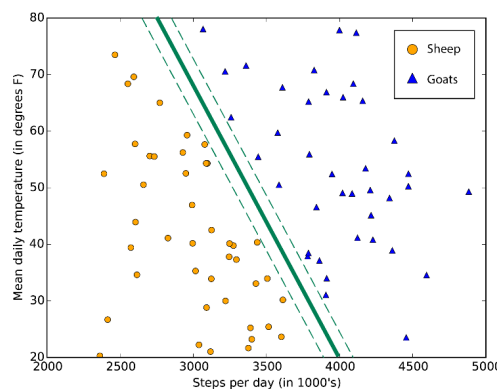


Figure 3 – Frontière de classification linéaire

Sur l'image ci-dessus, on peut voir que le classifieur a su séparer des individus appartenant à deux classes différentes et tracé une frontière entre elles, de façon distincte. En soit, ce type de classifieur a rapidement été adopté pour le traitement de gros volumes de données de grandes dimensions, en raison de leur faible nombre de paramètres à prendre en compte et des bons résultats observés en pratique après de nombreux tests effectués.

Selon [WWW7], " selon les données, la performance des machines à vecteurs de support est de même ordre, ou même supérieure, à celle d'un réseau de neurones ou d'un modèle de mélanges gaussiens " (peu voire aucun article trouvé disant qu'un SVM serait aussi voire plus efficace qu'un réseau de neurones).

4

Veille technologique

Cette partie a pour but de faire l'inventaire de toutes les techniques qui permettraient, dans le cas où elles sont implémentées, de pouvoir améliorer de manière efficace le taux de reconnaissance, pour que l'application se trompe moins, lors de la décision d'appartenance d'un pixel à une classe.

Pour cela, une introduction aux réseaux de neurones convolutifs sera fait et nous verrons certains exemples qui pourraient aider à résoudre ce problème. L'amélioration ne se faisant pas uniquement par l'implémentation d'un CNN ou encore *réseau de neurones convolutif*, dans le cas où l'utilisateur décide d'utiliser une méthode de résolution linéaire comme le **Séparateur à Vaste Marge** (SVM), il faut faire en sorte que le modèle construit puisse être le plus précis possible.

C'est pourquoi, dans ce cadre, nous verrons également des méthodes simples permettant d'améliorer au maximum la précision du modèle construit.

1 L'apprentissage profond ou Deep Learning

Selon cette source [WWW10], il s'agit d'un " ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires ". De façon simple, la machine est capable d'apprendre toute seule et de parcourir un ensemble de données toute seule pour en extraire les caractéristiques.

Dans ce cadre, des algorithmes performants ont vu le jour, comme la représentation que nous allons étudier, le réseau de neurones convolutif.

1.1 Résumé sur le Perceptron

Le Perceptron est le tout premier réseau de neurones ayant vu le jour, il est composé, dans un premier temps, que d'une seule couche.

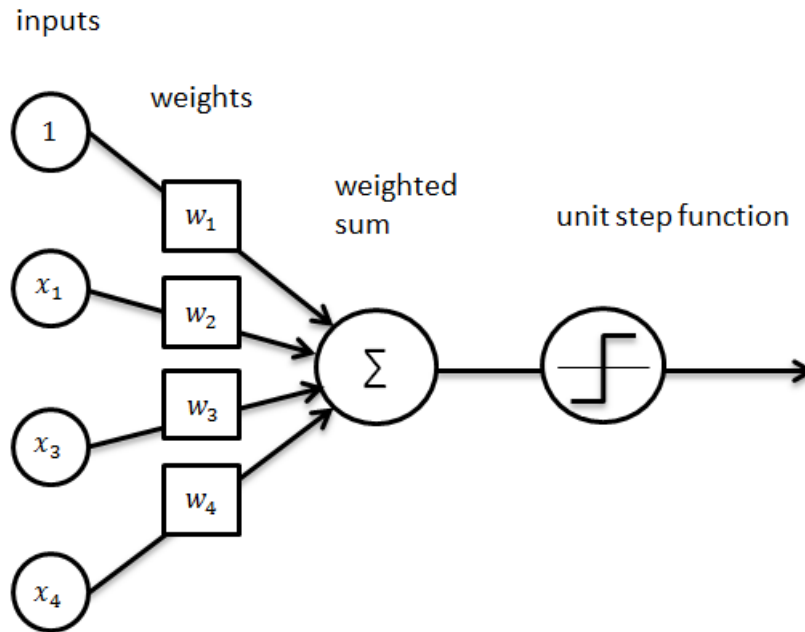


Figure 1 – Structure d'un Perceptron

Un Perceptron est composé de plusieurs points d'entrée que l'on appellera " neurones ". Pour chaque neurone, un *biais* ou un *poids*, qui est une constante fixée au début, est appliqué pour chaque " neurone ", pour effectuer une pondération sur le résultat (constamment activée, à 1).

Une couche de sortie, que l'on appellera " unité de sortie ", est reliée à la sortie de chaque neurone. Cette couche va recevoir la somme qui sera pondérée par le poids de chaque " neurone ", symbolisant son importance. Cette couche appliquera une **fonction de décision** ou appelée **fonction d'activation**, qui va permettre de prendre une décision quant à l'appartenance des données à une classe.

La couche de sortie recevra un résultat de la forme :

$$s = \omega_0 + \sum_{j=1}^p (\omega_j * x_j)$$

Dans cette relation ω_0 représente la constante de *biais*, p , le nombre de " neurones " constituant le Perceptron, ω_j , le poids associé à chaque " neurone " pour la pondération et x_j , le paramètre en entrée du " neurone ".

La fonction d'activation dépend du problème auquel nous sommes confrontés, dans le cas d'un problème de régression linéaire (les données sont exprimées numériquement), il s'agira de la fonction identité (elle retourne uniquement ce qu'elle reçoit); dans le cas d'un problème de **classification binaire** (une seule classe à traiter), on appliquera une **fonction de seuil** sur le résultat s vu précédemment, qui va permettre de prendre une décision quant à l'appartenance de l'objet à identifier à la classe.

Une fonction d'activation, pouvant être exécutée dans tous les cas, existe. Elle permet à ce que " le moindre petit changement de poids ou de son seuil peut causer des résultats complètement différents, ce qui est un problème pour l'apprentissage " [1], il s'agit de " la fonction sigmoïde " [WWW3], qui permet, de façon simple, " prédire la probabilité d'appartenir à la classe positive " selon [WWW3].

La fonction est la suivante :

$$\sigma(\omega_0 + \sum_{j=1}^p (\omega_j * x_j)) = \frac{1}{1 + e^{-(\omega_0 + \sum_{j=1}^p (\omega_j * x_j))}}$$

Cependant, ce modèle ne fonctionne pas dans le cas d'une **classification multi-classe** (plusieurs classes connues et identifiées). L'architecture du Perceptron va changer, c'est-à-dire que vu qu'il y a **n** classes connues, il y aura donc **n** couches de sortie, connectées à tous les " neurones ", où chaque couche de sortie sera spécialisée pour reconnaître une classe précise.

Dans ce cas, " on peut alors utiliser comme fonction d'activation la fonction softmax. Il s'agit d'une généralisation de la sigmoïde " selon [WWW3]. Cette fonction s'écrit dans cette forme :

$$\sigma_k(u_k) = \frac{e_k^u}{\sum_{l=1}^K e_l^u}$$

On sait aussi que " si la sortie pour la classe k est suffisamment plus grande que celles des autres classes, son activation sera proche de 1 tandis que l'activation des autres sera proche de 0 " selon [WWW3]. Ce qui signifie que l'élément analysé a une plus grande probabilité d'appartenir à cette classe qu'aux autres.

Il faut savoir que le traitement des données par le Perceptron se fait de manière itérative. " Après chaque observation, nous allons ajuster les poids de connexion de sorte à réduire l'erreur de prédiction faite par le perceptron dans son état actuel " selon [WWW3]. En effectuant cet ajustement, cela va permettre au réseau de ne pas se spécialiser pour la reconnaissance d'une classe, mais de pouvoir rester polyvalent en ne modifiant qu'un seul paramètre déterminant.

L'algorithme utilisé est celui du gradient, en l'utilisant, on cherche le minimum de cette fonction en fonction du sens de variation de la " fonction d'erreur " [WWW3], qui va nous permettre de savoir dans quel sens la parcourir.

Cette " fonction d'erreur " va être différente en fonction du cas dans lequel on se trouve :

1. Pour une régression : on utilise l'**erreur quadratique**
2. Pour une classification : on utilise le plus souvent une **entropie croisée**

Ces différentes fonctions, en fonction du cas, ont donc pour but de mettre à jour ces poids pour éviter la spécialisation, c'est-à-dire, empêcher le réseau d'être " expert " dans la reconnaissance d'un seul type d'élément ou classe d'appartenance.

Pour un traitement encore plus rapide et efficace, il a été pensé d'assembler plusieurs réseaux de neurones à une couche **Perceptrons** pour créer une architecture multi-couches, pouvant être utilisée pour la même fonctionnalité qu'un seul Perceptron.

1.2 Perceptrons multi-couches

La définition de cette architecture peut être résumé de la façon suivante : " Nous allons créer des couches intermédiaires (ou couches cachées, hidden layers en anglais) entre la couche d'entrée et celle de sortie. Chaque neurone d'une couche est connecté à tous les neurones de la couche au-dessus de lui. Et voilà ce qu'on appelle un perceptron multi-couche. " selon [WWW2].

Chaque couche intermédiaire est constituée d'au moins un " neurone " et chaque sortie de ce " neurone " sera soumis à une **fonction d'activation** qui, au final, " une combinaison linéaire des entrées " [WWW2]. Le résultat sera ensuite traité par les couches en-dessous comme ci-dessous.

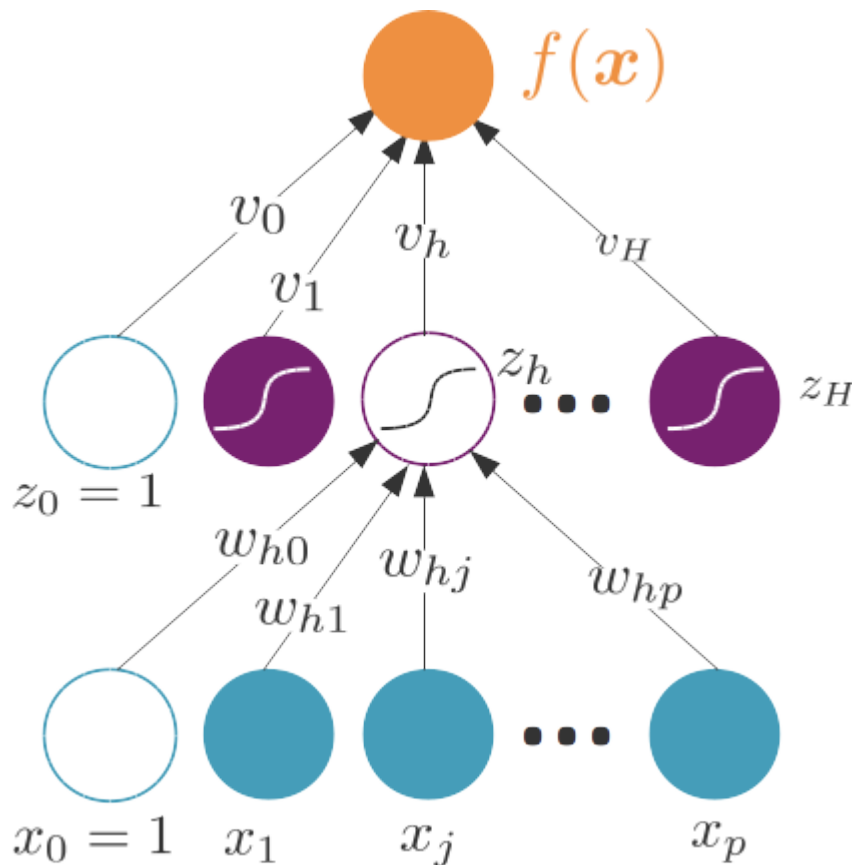


Figure 2 – Exemple de Perceptron multi-couche à une seule couche intermédiaire

En ce qui concerne la **fonction d'activation** à utiliser pour les couches intermédiaires, un autre algorithme sera utilisé, il s'agit de la " tangente hyperbolique qui permet de transformer la combinaison linéaire des signaux d'entrée en un nombre entre -1 et 1 plutôt qu'entre 0 et 1 " [WWW2], ce qui permet plus facilement aux autres couches, d'interpréter le résultat.

Cette architecture de réseau est caractérisé par le fait le fait que les couches intermédiaires sont directement reliées sur les points d'entrée et les couches les plus profondes sur les couches intermédiaires, ce qui fait que l'information ne peut aller que dans un unique sens. On qualifie donc ce genre d'architecture feed-forward car, à partir de la sortie, on ne peut pas accéder et pondérer le résultat donné par les couches intermédiaires.

Dans ce cas, la **fonction d'erreur** sera également différente ainsi que la manière dont elle sera gérée pour ajuster les **poids de connexion** entre les couches intermédiaires et les couches de sortie. On utilisera le concept de " rétropropagation de l'erreur " [WWW2], qui consiste de manière " simple ", de décomposer l'erreur finale obtenue par calcul des dérivées partielles.

Une fois que les erreurs sont calculées, il sera possible de prendre en compte ces erreurs pour ensuite faire la mise à jour des **poids** de tous les " neurones " du réseau.

Pour cela, deux phases sont identifiables :

1. **Phase forward** : consiste à mettre à jour " les sorties des couches intermédiaires " [WWW2]
2. **Phase backward** : " le gradient de l'erreur par rapport aux poids d'une couche peut être calculé à partir du gradient de l'erreur par rapport aux poids d'une couche supérieure " [WWW2]

Après avoir vu toutes ces informations, nous allons maintenant nous concentrer sur les réseaux de neurones profonds.

Toutes les images et informations citées ou utilisées précédemment sont retrouvables sur

les liens suivants : [WWW3] et [WWW2].

2 Réseaux de neurones profonds

Après avoir détaillé le fonctionnement du Perceptron, de façon très " simplifiée ", on peut dire qu'un réseau de neurones convolutif est l'assemblage de plusieurs Perceptrons multi-couche, à la différence que, comme nous allons le voir, que certaines couches du CNN (**Convolutional Neural Network**) sont utilisées pour effectuer des convolutions sur les images passées en paramètre.

L'illustration ci-dessous résume l'ensemble des éléments présentés lors de l'explication sur les perceptrons.

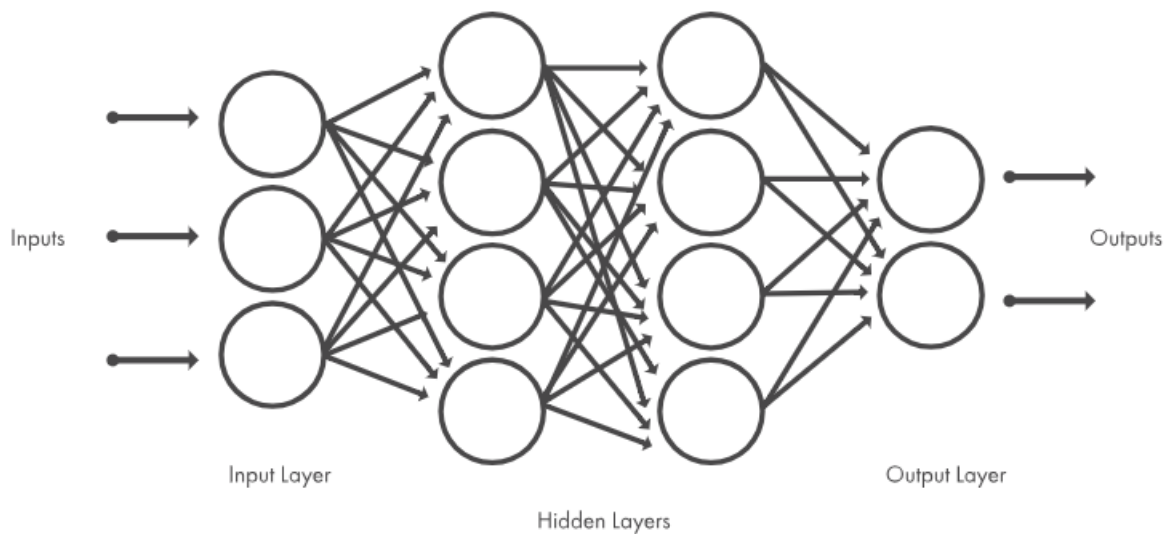


Figure 3 – Architecture d'un réseau de neurones convolutif

Nous allons plus spécifiquement des couches composant le CNN. Comme le nom l'indique, il y a bien, dans certaines couches, des convolutions qui sont faites. Elles permettent, d'une certaine façon, d'extraire les informations de chaque zones d'intérêts présentes dans l'image.

Voici une image permettant de mieux comprendre le système et nous parlerons plus en détail de ces fameuses couches.

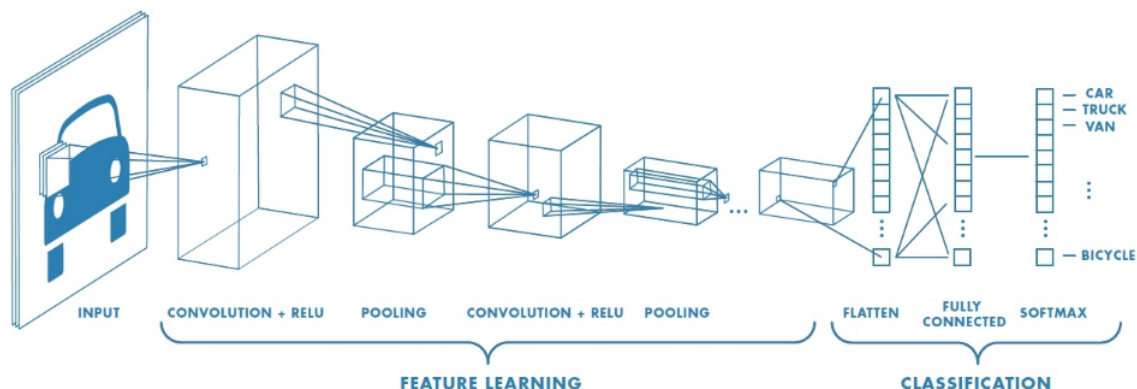


Figure 4 – Etapes du traitement d'une image par un CNN

Comme on peut le voir sur l'image ci-dessus, l'ensemble des couches présentes dans un CNN peuvent être regroupées en deux parties :

1. L'extraction des fonctionnalités et apprentissage : que ce soit pour l'apprentissage ou pour l'identification sur des images
2. La classification : prend une décision quant à l'appartenance d'un élément à une classe en fonction des probabilités calculées

Nous traiterons chaque groupe de couches séparément.

2.1 Couches d'extraction des fonctionnalités et apprentissage

Ce groupe contient les couches **Conv** (Convolution), **ReLu** (Rectified Linear Unit) et la couche **Pooling**.

2.1.1 Couche de Convolution (Conv)

A partir d'un masque de convolution au préalable défini, d'une taille précise (matrice carrée), des convolutions successives seront faites directement sur l'image pour tenter de trouver des caractéristiques dans l'image entrante par " filtrage " grâce à ce système de convolution.

En bref, il s'agit de faire une simple multiplication de matrice entre le masque et la zone en cours d'étude, comme le montre l'image ci-dessous.

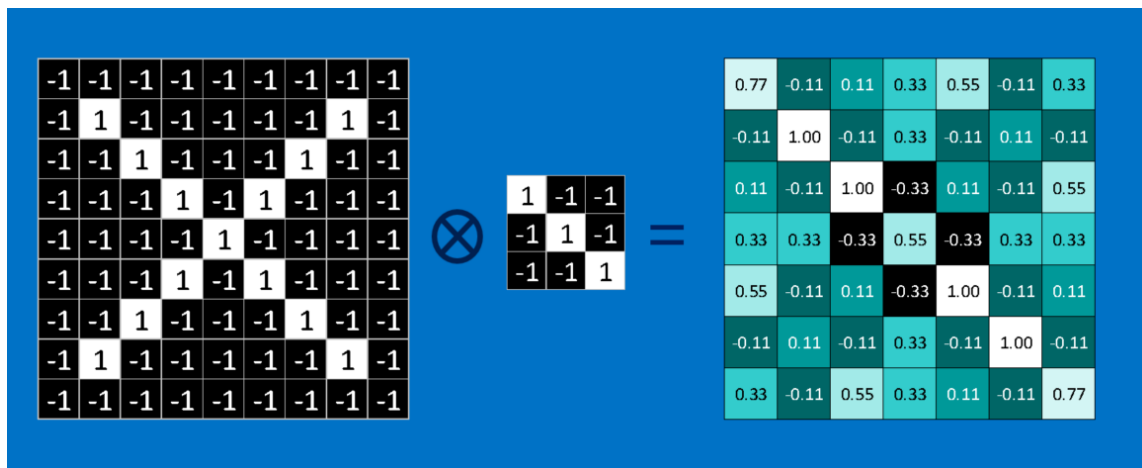


Figure 5 – Convolution entre le masque défini et l'image

Le principe est de multiplier les pixels du masque avec les pixels de l'image. Ensuite, on additionne tous les résultats obtenus et on divise la somme par le nombre de pixels du masque. Si 2 pixels ont la même couleur, le résultat de leur multiplication sera alors 1, sinon, le résultat sera -1.

2.1.2 Couche Pooling

Cette couche est utile dans le cas où la matrice résultante en sortie de la couche de Convolution n'est pas adaptée pour poursuivre les traitements dessus. C'est une opération qui est faite automatiquement après le passage par la couche de Convolution.

Le travail de cette couche consiste à " redimensionner " la matrice résultante de la convolution (en général, sa taille est trop grande et on cherche à la réduire) et en même temps, de pouvoir conserver au maximum les informations en en perdant le moins possible.

Cela est rendu possible en faisant " glisser " une sorte de fenêtre, ayant une taille précis, sur l'image, et qui pour chaque zone de l'image dans laquelle la fenêtre est, va récupérer la valeur la plus grande du pixel de la zone et va le stocker dans une nouvelle matrice temporaire. L'image ci-dessous illustre ce traitement.

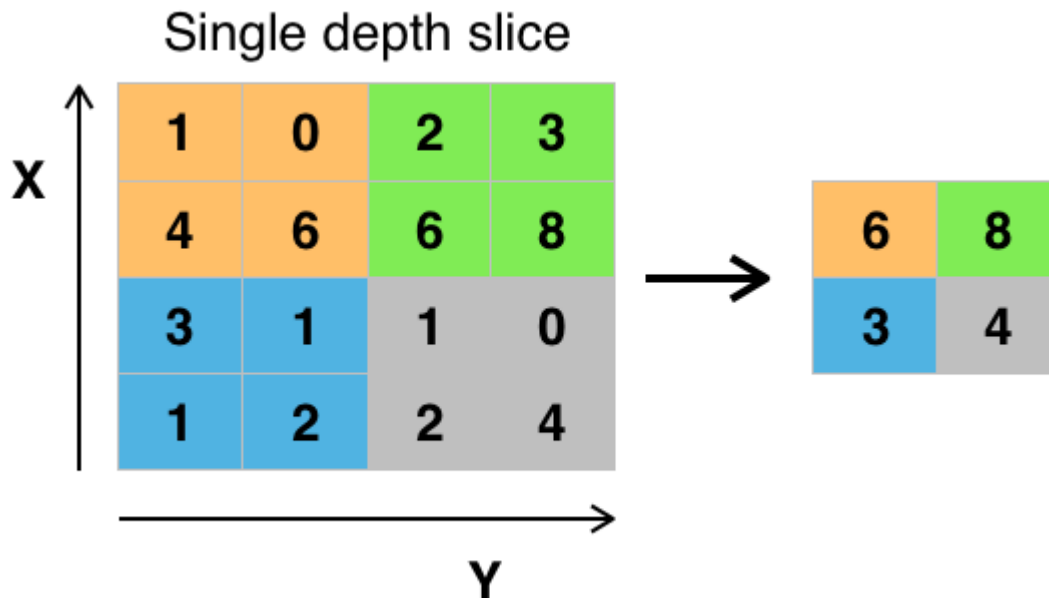


Figure 6 – *Traitement de la couche Pooling*

Selon la source [WWW5], " après avoir procédé au pooling, l'image n'a plus qu'un quart du nombre de ses pixels de départ ". Vu que les nouvelles matrices générées ont une taille réduite par rapport à l'image de départ, on peut également supposer que cette étape permet de diminuer le nombre et la complexité des calculs à faire à la suite, si ce traitement n'était pas fait.

2.1.3 Couche ReLu

Comme dit précédemment, cette couche va servir à corriger les incohérences survenues lors du traitement des deux précédentes. Elle a pour tâche d'éliminer toutes les valeurs négatives qui auraient un mauvais impact sur l'apprentissage ou sur la déduction de la classe d'appartenance. Seules les valeurs négatives sont remplacées par 0, pour éviter que " les valeurs apprises de rester coincer autour de 0 ou d'exploser vers l'infinie " [WWW5].

L'image ci-dessous récapitule le traitement.

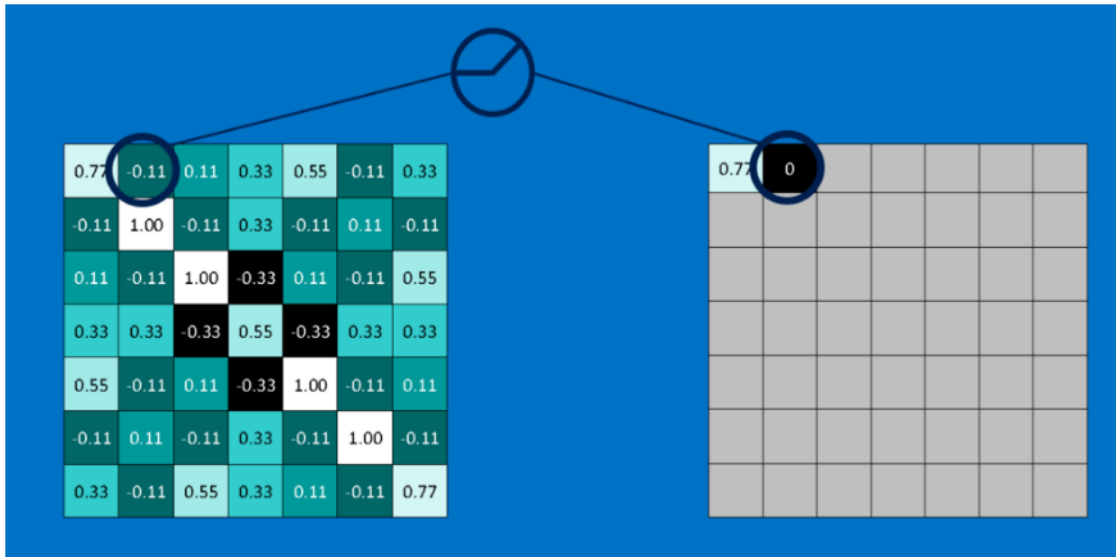


Figure 7 – Traitement effectué par la couche ReLu

2.2 Couches de classification

Ces couches ont été en partie présentées lors de l'introduction aux Perceptrons. Une fois que l'extraction des fonctionnalités et l'apprentissage soient finies, à partir des résultats, ces couches vont donc prendre une décision quant à l'appartenance d'un élément ou d'une zones d'intérêts à une classe, en se basant sur des probabilités calculées auparavant.

Comme son nom l'indique, l'une des couches est entièrement connectée (Fully Connected) où " sort un vecteur de K dimensions, où K représente le nombre de classes que le réseau sera capable de prédire " [WWW8].

Le vecteur obtenu en sortie, contient pour chaque région présente dans l'image, des probabilités pour chaque classe connue, permettant de définir à quelle classe appartient la région.

La dernière couche de classification (en général, il s'agit de la couche **softmax**), qui fournit le résultat de la classification.

Les sources de toutes ces informations et illustrations présentées ci-dessus sont [WWW8] et [WWW5].

Après avoir longuement parlé des CNN, nous allons voir une autre façon qui permettrait d'améliorer la précision du modèle, dans le cas où l'utilisateur souhaite utiliser un algorithme de classification linéaire.

3 Techniques pour améliorer la précision du modèle

Dans cette partie, nous allons essentiellement étudier les méthodes qui existent pour garantir un certain niveau de précision du modèle construit, dans le cas où l'utilisateur décide d'utiliser une méthode de classification linéaire, comme un **Séparateur à Vaste Marge** (SVM) présenté précédemment.

3.1 L'ajout de données

L'ajout de données est une solution intégrante à ce problème. Cela permettrait à un modèle d'être plus représentatif grâce à la présence d'un grand volume. Dans certains cas, il n'est pas envisageable de mettre en place cette solution, c'est pourquoi les autres méthodes pourraient être utilisées pour résoudre le problème.

3.2 Le traitement des données incohérentes et des oublis

En traitant ces cas, on s'assure que les informations sur lesquelles le modèle est construit, sont pertinentes et correctes. Ces oublis peuvent être détectés si les informations ne suivent pas une sorte de continuité ou sont de format différent par rapport aux autres.

L'exemple ci-dessous illustre plutôt bien ce phénomène.

With Missing Values

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55		Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57		Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	2	1	50%
M	4	2	50%
Missing	2	2	100%

After imputation of missing values

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55	F	Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57	F	Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	4	3	75%
M	4	2	50%

Figure 8 – Exemple de données manquantes et impacts

En rectifiant les oublis, la précision du modèle serait accrue.

3.3 La sélection des fonctionnalités à appliquer

Parfois, l'utilisateur est demandé pour sélectionner les meilleurs fonctionnalités à utiliser pour extraire les caractéristiques d'un ensemble de données. Dans certains cas, les fonctionnalités sélectionnés n'ont pas suffisamment de sens pour représenter correctement les données présentes dans l'ensemble de données.

Il est possible de les estimer :

1. Graphiquement : en fonction de la forme des courbes ou de ce que l'on obtient
2. Par connaissance : des études précédentes que l'on a mené nous a permis de les sélectionner de manière naturelle
3. Par analyse statistique : méthode principalement utilisée par les algorithmes qui font cette tâche automatiquement. Cela consiste à représenter l'ensemble de données dans des espaces différents pour voir les liens qu'ils ont entre eux

Aujourd'hui, certains algorithmes permettent de décider par analyse, les meilleurs fonctionnalités à utiliser pour extraire les caractéristiques des données et faire en sorte qu'elles soient le plus représentatives possible.

3.4 L'utilisation d'algorithmes

Dans certains cas (pour le nôtre en Python), certaines bibliothèques standards des langages embarquent automatiquement des algorithmes à appliquer en fonction du type de données auquel ils sont confrontés et le résultat que l'on souhaite obtenir, par analyse.

L'ensemble de algorithmes Python sont embarqués dans la bibliothèque **scikit-learn**.

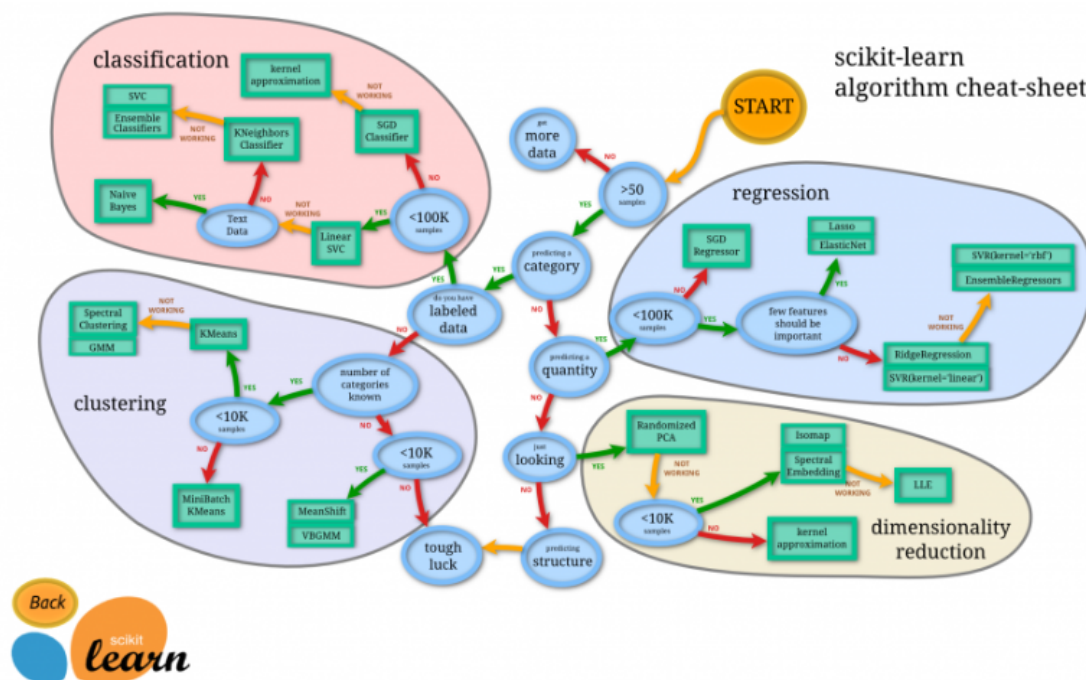


Figure 9 – Algorithmes présents dans la bibliothèque standard

Par exemple, si les données peuvent être exprimées numériquement et que l'on veut avoir une donnée numérique en sortie, il faut utiliser l'algorithme de **régression** (voir si le cas est linéaire).

3.5 Le paramétrage du classifieur

Pour que le classifieur utilisé ait un comportement qui satisfasse les exigences, il faut que les paramètres remplis par l'utilisateur soient cohérents, corrects et que l'utilisateur sache comment le classifieur fonctionne.

Dans le cas contraire, des algorithmes dits de "tuning", permettent, après avoir analysé les données à traiter, de choisir et fixer les paramètres optimaux à utiliser pour obtenir la meilleure représentation possible.

3.6 La validation croisée

Cette méthode permet de garantir à ce que le modèle ne fasse du sur-apprentissage et ne se spécialise à l'identification que d'un seul type d'élément.

Le principe est de scinder un ensemble de données en deux parties égales dont l'une va servir pour l'apprentissage et l'autre, pour la validation du modèle construit à partir des données d'apprentissage.

Une fois que le modèle est entraîné, on va séparer en n lots de données, l'ensemble des données. Chaque lot de données constituant les n , est utilisé comme lot de tests.

Ce qui fait qu'au final, chaque donnée présente dans la base d'apprentissage, est utilisée au plus une fois en tant que donnée de test pour la validation du modèle construit.

L'image ci-dessous récapitule ce cycle.

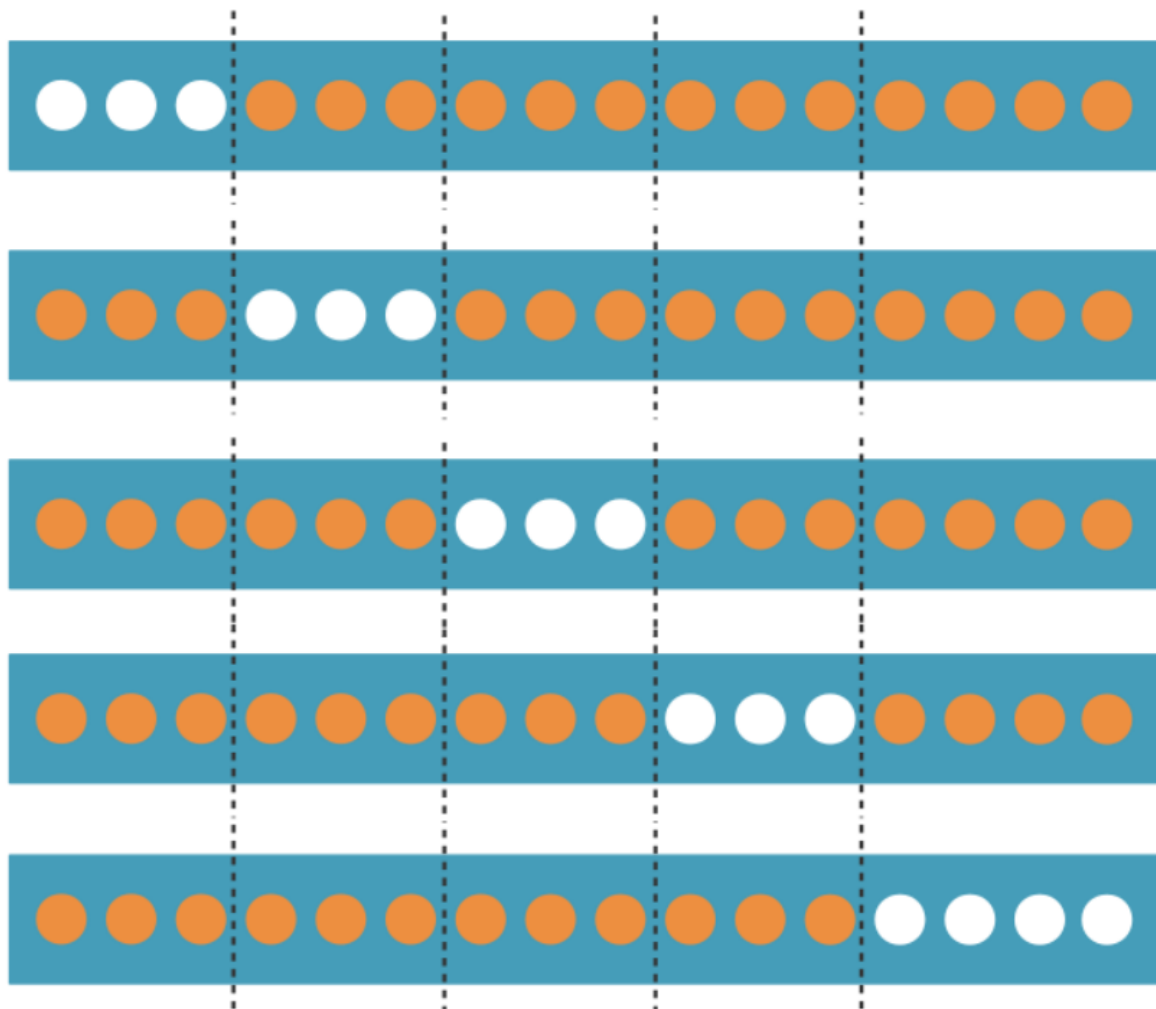


Figure 10 – Validation croisée

La source de ces informations vient de [WWW1].

5

Analyse et conception

1 Spécifications fonctionnelles

Dans cette partie, nous allons parler du travail à effectuer pour améliorer l'application ainsi que les ressources qui seront utilisées pour y arriver.

1.1 Amélioration du taux de reconnaissance

1.1.1 Description

Comme vu précédemment, cette fonctionnalité a pour but d'améliorer le taux de reconnaissance par l'implémentation d'un réseau de neurones qui sera le noyau d'une nouvelle application. Cette fonctionnalité est prioritaire sur les autres.

Dans notre cas, les fichiers passés en entrée du réseau ne sont pas des images mais directement les données de nos fichiers CSV, c'est pourquoi l'approche est un peu différente. Dans ce cadre, deux grandes architectures connues peuvent être implémentées : le réseau de neurones à convolution (CNN) ou le Perceptron.

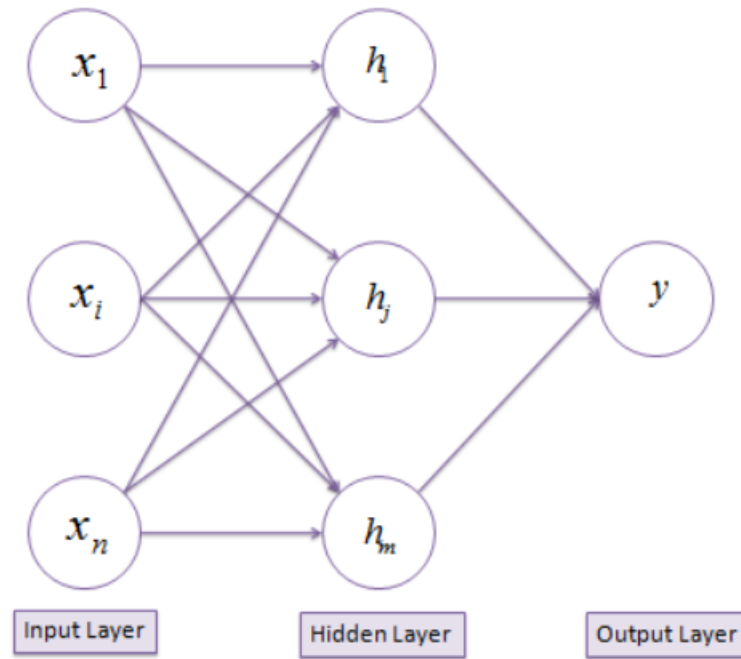


Figure 1 – Structure d'un Perceptron

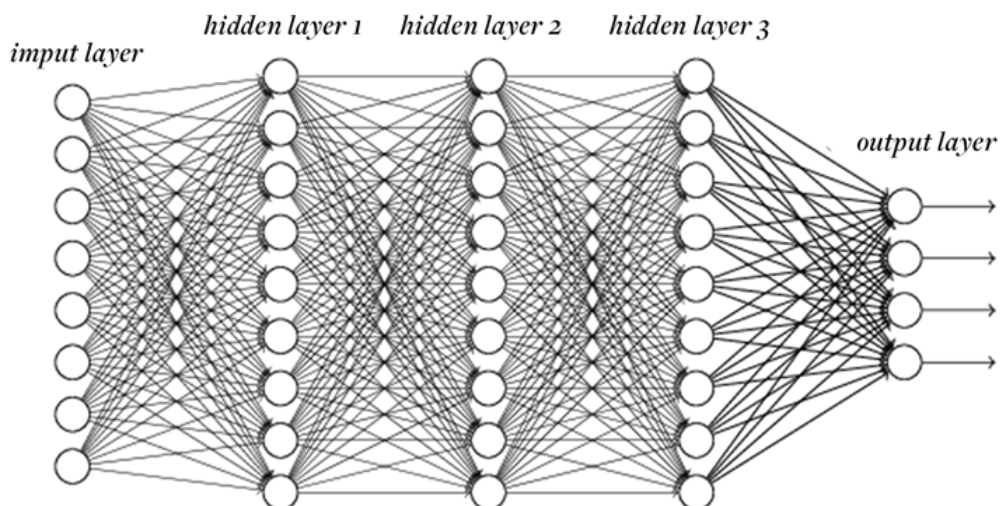


Figure 2 – Structure d'un CNN

Appliquée à notre problème de classification, il a été conseillé d'utiliser une structure simple comme le **Perceptron** pour voir les performances d'une architecture de ce type pour notre problème. Une étude de performances sera faite par la suite grâce au développement d'un algorithme pour la recherche de la configuration optimale.

Sa structure est également à définir, nous savons cependant que le nombre de neurones d'entrée correspond au nombre de caractéristiques présentes dans notre fichier d'apprentissage (variable, nb de colonnes), le nombre de neurones de sortie du **Perceptron** correspond au nombre de classes connues et apprises par le modèle. Le nombre de neurones dans la couche cachées reste tout de même à déterminer ainsi que la fonction d'activation optimale.

La couche de sortie du réseau va donc récupérer le résultat de la prédiction pour chaque pixel sous forme de probabilité d'appartenance, pour chaque classe. L'ensemble de ces probabilités seront donc récupérées et écrites manuellement dans un fichier texte pour avoir une trace des décisions prises par l'application, en fonction du fichier CSV sélectionné à prédire.

Certains détails précis manquent car ils sont liés à la bibliothèque à utiliser, comme les pa-

ramètres correspondants. En fonction de la spécificité de chaque bibliothèque, le comportement est différent et le choix est donc difficile à faire.

Des connaissances sont manquantes, c'est pourquoi une formation sur la compréhension et l'utilisation approfondie des réseaux de neurones est prévue.

En ce qui concerne l'intégration de cette architecture, la structure du code prévue de l'application permet de faire en sorte à ce que d'autres méthodes de reconnaissances puissent être implémentées par redéfinition des méthodes standards (entraînement, évaluation, prédiction, ...). La structure de la nouvelle application est profondément inspirée de l'application **SkyEye**.

1.2 Algorithme de recherche de la configuration optimale du réseau de neurones

1.2.1 Description

Pour déterminer la meilleure configuration, il faut prendre en compte plusieurs paramètres. Ces paramètres sont :

1. Le taux d'apprentissage : ne pas le fixer au-dessus de 0.05 : trop élevé, entraîne du surapprentissage, il ne pourra que reconnaître les classes connues les plus représentées dues à un apprentissage de la même donnée)
2. Le nombre de couches cachées
3. Le nombre de epoques : nb de passages des données d'apprentissage dans la boucle
4. Le type de la fonction d'activation

Ces critères ont un impact direct sur les performances de reconnaissance du futur modèle construit. Il faut également s'assurer à limiter les plages de variation de ces valeurs, d'une part, pour éviter que cette recherche ne dure trop longtemps, d'autre part, certaines valeurs sont connues pour ne pas être adaptées au réseau. Les plages de variation pour chaque critère sont donc les suivantes :

1. Taux d'apprentissage : de 0.01 à 0.02
2. Nombre de couches cachées : entre 1 et 30
3. Nombre de epoques : entre 800 et 2.000 itérations (si le taux d'apprentissage est faible, il est conseillé de fixer une valeur élevée pour ce nombre)

Pour tester, les fonctions d'activation les plus connues de la littérature seront utilisées, il s'agit respectivement de la fonction tangente hyperbolique, sigmoïde, ReLu et softmax. Un réseau de neurones avec comme fonction d'activation celles citées seront utilisées pour dresser ensuite un comparatif, ce qui nous permettra par la suite de sélectionner et valider l'efficacité de celle choisie.

1.3 Chargement des fichiers CSV et séparation des fichiers sous forme de bases

1.3.1 Description

Cette partie permet, une fois que l'utilisateur a choisi les fichiers CSV à importer, de charger ces fichiers et de constituer des ensembles des données. Quatre ensembles dits « bases » seront constituées qui sont respectivement :

1. **La base d'apprentissage** : données issues du fichier d'apprentissage permettant au réseau de neurones d'apprendre les caractéristiques de chaque classe et de les généraliser pour pouvoir reconnaître d'autres individus

2. **La base de test** : données issues du fichier d'apprentissage utilisées pour évaluer le modèle créé à partir des paramètres trouvés, ce qui permet d'extraire une matrice de confusion résultante et un taux de reconnaissance global pour visualiser la qualité de la classification
3. **La base de validation** : données issues du fichier d'apprentissage utilisées pour vérifier les performances du réseau lors de la recherche de la meilleure configuration par le calcul du taux de reconnaissance
4. **La base de prédiction** : données issues du fichier à prédire contenant l'ensemble des pixels dont le label doit être défini

En ce qui concerne la séparation des données en différentes bases pour le fichier d'apprentissage, elle est la suivante :



Figure 3 – Séparation des données du fichier d'apprentissage en bases

Il est souhaitable de garder la plus grande proportion de données pour l'apprentissage, ce qui laisse une chance d'avoir des individus appartenant à la même classe mais dont les caractéristiques sont légèrement différentes pour pouvoir repérer et apprendre ces non similarités. Cela permet au modèle de pouvoir avoir plusieurs exemples différents d'individus de la même classe. Pour les proportions, environ 60% des données de ce fichier sont utilisées pour l'apprentissage, 20% pour la validation et les autres 20% pour le test.

1.4 Interface graphique de l'application implémentant le réseau de neurones

1.4.1 Description

Le but de cette fonctionnalité est de pouvoir laisser l'utilisateur « manipuler » le réseau de neurones de l'application à développer, à travers une interface graphique l'autorisant à charger des fichiers CSV d'apprentissage, à prédire, un fichier contenant le modèle et de sauvegarder le modèle construit dans un fichier. Il peut également entraîner le modèle, une fois que celui-ci est construit, évaluer ses performances et lancer une prédiction à partir des données présentes dans le fichier à prédire.

Toutes ces actions sont résumées par le diagramme ci-dessous :

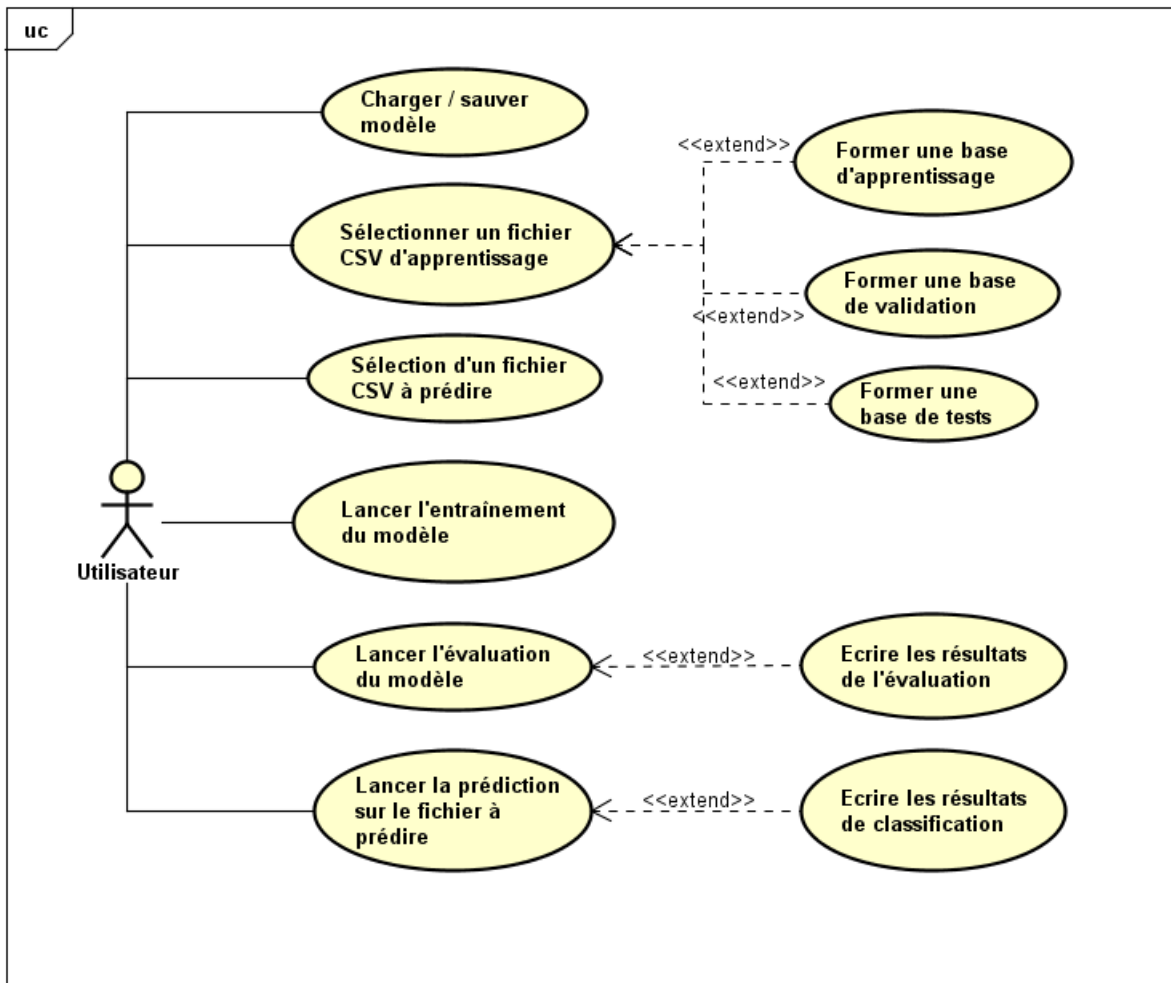


Figure 4 – Diagramme des cas d'utilisation de l'application à développer

Une ébauche de cette interface graphique a été faite et se situe sur la page suivante.

Application de reconnaissance de reliefs avec CNN	
Options	
Modèle:	<div>Charger</div> <div>Sauver</div>
Phase d'entraînement	Phase de test
Sélection d'un fichier d'entraînement:	Sélection d'un fichier de test:
<div>Parcourir</div> <div>Entraîner</div>	<div>Parcourir</div> <div>Prédire</div>
Statut: Prêt	Progression: <div></div>

Figure 5 – Interface graphique de la nouvelle application proposée

Certaines modifications pourront être faites, comme le positionnement de la section « Phase de test » en dessous de la section « Phase d'entraînement » pour faciliter la navigation de l'archéologue, ce qui permet de garder un ordre chronologique des actions à mener avant de pouvoir lancer une prédiction.

1.5 Conception de l'interface graphique de visualisation de l'application SkyEye

1.5.1 Description

Cette interface aura pour but d'afficher le résultat des décisions d'appartenance des pixels aux classes prédéfinies, ce qui permettra à la fois à l'utilisateur de pouvoir modifier manuellement la décision d'affectation d'un pixel à une classe et, même s'il a déjà été calculé précédemment, de générer un taux de performance pour la reconnaissance des reliefs dans l'image et valider définitivement le modèle construit.

Le diagramme des cas d'utilisation ci-dessous récapitule les actions que peut être amené à faire l'utilisateur.

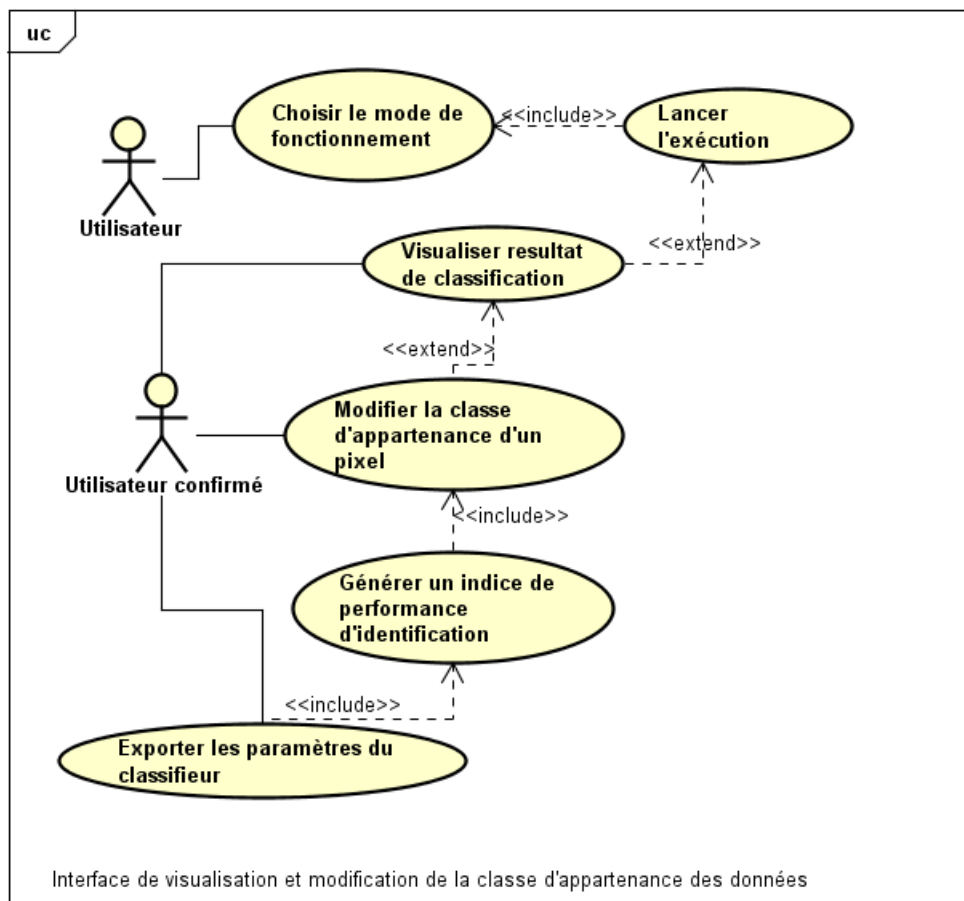


Figure 6 – Diagramme des cas d'utilisation de l'interface graphique

Il sera possible de modifier à nouveau le diagramme, pour prendre en compte les besoins supplémentaires venant de la MOA. Ce travail a été fait de façon prévisionnelle, c'est-à-dire, sous condition de validation par la MOA.

A partir de cette situation, une idée de maquette serait la suivante, sachant qu'une image résultante du traitement est générée automatiquement, l'interface pourrait ressembler à ça.

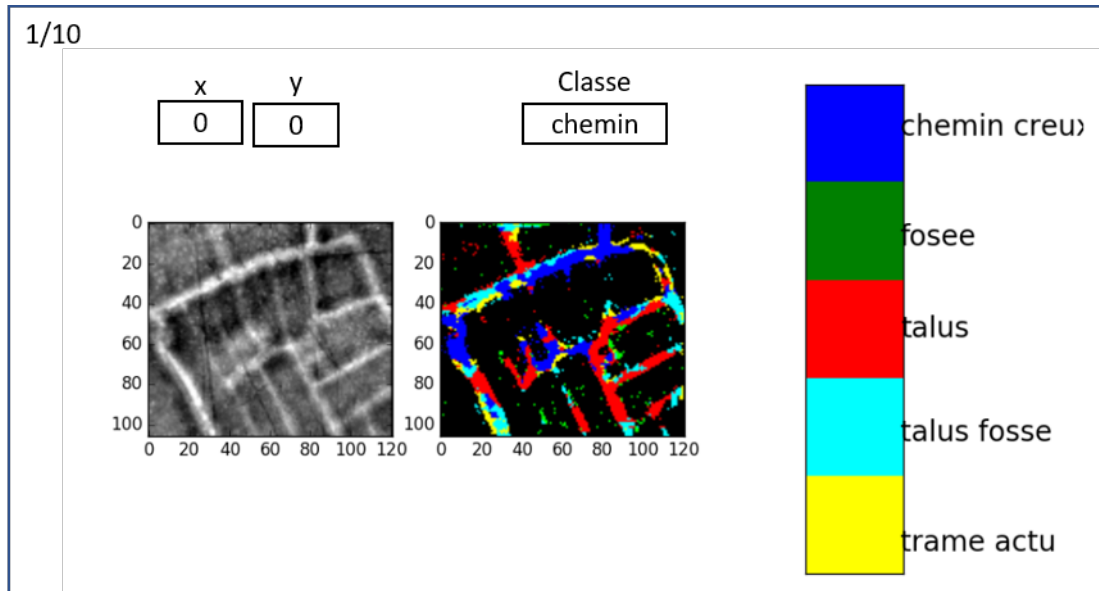


Figure 7 – Maquette de l'interface graphique proposée

Peu importe le mode de fonctionnement de l'application, que ce soit en apprentissage ou en mode prédiction, on récupère les fichiers images générés, portant le même nom que l'image passée en paramètre, au début. Le principe est d'afficher l'image originelle avec, à côté, l'image dont le système a prédit l'appartenance des pixels aux classes.

La résolution de l'image ne change pas et on a deux variables dont on peut modifier les valeurs : x et y. Ces deux variables représentent les coordonnées d'un pixel dans l'image, caractérisé par son appartenance à une classe. En modifiant les valeurs, on récupère la classe d'appartenance du pixel, par lecture dans le fichier CSV généré. Un compteur d'images traitées s'incrémente au fur et à mesure que les paires d'images sont affichées pour que l'utilisateur ait la possibilité de changer la décision prise par le système.

Cette interface n'est pas implémentée mais une base comme celle-ci pourrait être utilisée pour la modification de la classe d'appartenance des pixels des images, de manière simple. On se baserait donc sur 3 fichiers : le fichier CSV résultant de la détermination d'appartenance de tous les pixels de l'image aux classes définies précédemment, l'image initiale au format .tif en nuances de gris et l'image finale après construction grâce au premier fichier évoqué. Au final, aucun nouveau fichier n'est attendu en sortie, uniquement une mise à jour des couleurs des pixels et la reconstruction de l'image finale, après que toutes les images aient été modifiées ou non.

Encore une fois, cette partie doit être proposée et validée par la MOA, tout en prenant en compte leurs besoins en plus, pour pouvoir passer à la modélisation et l'implémentation.

Pour simplifier ce traitement, la conception et le développement d'un parseur pour les fichiers CSV générés pourrait être utile ce qui permettra de garder en mémoire son contenu, tant que l'utilisateur n'a pas terminé les modifications à apporter.

Comme vu au début de ce rapport, l'amélioration prioritaire à faire est l'amélioration du taux de reconnaissance. C'est une demande qui a été faite par les clients, que d'améliorer les performances de reconnaissance de l'application.

2 Analyses approfondies

Dans cette partie, nous allons étudier de façon détaillée l'ensemble de tous les points énoncés précédemment.

2.1 Bibliothèque de Deep Learning à utiliser

Comme vu précédemment, il faut faire un choix quant à la bibliothèque de Deep Learning à utiliser. Une étude a été faite pour tenter de les départager et de déterminer celle qui sera par la suite utilisée.

Le tableau ci-dessous résume dans les grandes lignes, les différences entre chaque bibliothèques :

Bibliothèques \ Critères	Facilité d'utilisation	Communauté et support	Outils embarqués	Rapidité d'entraînement du modèle	Nb minimum de lignes de code	Modèles de références implémentés	Ne nécessite pas un environnement spécifique	Connaissances ?
TensorFlow	x	o	o	o	x	o	o	x
Keras	o	o	x	x	o	o	o	x
Theano	x	x	x	x	x	x	o	x
Scikit-Learn	x	o	x	x	o	x	o	x
PyTorch	o	o	x	o	o	o	o	o

Figure 8 – Comparatif des bibliothèques de Deep Learning Python

2.1.1 Facilité d'utilisation

Cette caractéristique représente la facilité qu'a un utilisateur à créer son propre modèle de CNN ou de MLP et à le déployer pour ensuite pouvoir l'utiliser. Les deux principales bibliothèques sont :

1. **PyTorch** : redéfinition de la classe Net avec les différentes couches voulues dans le CNN ou le MLP. On redéfinit également la méthode **forward** qui définit le comportement et l'ordre de passage des données dans les différentes couches en associant ou non pour chaque des fonctions d'activation
2. **Keras** : appel à des méthodes prédéfinies à utiliser et à entièrement paramétrer. Elles s'appuient cependant sur une bibliothèque plus bas niveau utilisée de support comme TensorFlow, Theano, CNTK,...

Ces deux frameworks sont suffisamment haut niveau pour permettre la création d'un modèle simple en facilitant sa configuration. Pour TensorFlow, il s'agit d'une bibliothèque plus bas niveau qui, une fois que le modèle est construit, permet de manipuler des tenseurs contenant les données à traiter que l'utilisateur doit impérativement configurer, ce qui le rend plus difficile d'utilisation.

2.1.2 Communauté et support

Certaines bibliothèques sont toujours supportées et très utilisées, que ce soit par des chercheurs, des entreprises, ... Certaines ne sont plus maintenues et donc plus optimisées, comme c'est aujourd'hui le cas de *Theano*. D'autres comme *Keras*, *PyTorch* permettent de façon simple de déployer des modèles de CNN ou de MLP facilement et toujours maintenues par les éditeurs comme Facebook ou Google.

2.1.3 Outils embarqués

Certaines bibliothèques embarquent des outils entièrement dédiés pour aider au monitoring et à la visualisation de l'état du CNN après son déploiement en production. Ces bibliothèques sont majoritairement utilisées par les entreprises pour le monitoring en production.

Dans notre cas, il pourrait être pratique de pouvoir voir en temps réel l'état de chaque donnée lors des passages dans chaque neurone du modèle, ce qui nous permettrait également de voir les poids associés à chaque neurone du Perceptron à un instant t .

2.1.4 Rapidité d'entraînement du modèle créé

Cette caractéristique représente le temps mis pour entraîner un modèle construit dépend de sa complexité. Par exemple, pour les modèles préconstruits implémentés dans ces bibliothèques comme **ResNet**, **VGG**, ...

Un comparatif sur les performances d'entraînement entre Keras, TensorFlow et PyTorch a été faite en se basant sur les implémentations des algorithmes précédents (<https://wrosinski.github.io/deep-learning-frameworks/>). Pour chaque modèle construit utilisé, les données d'entraînement de chaque étude sont les mêmes.

Le graphique ci-dessous contient le résultat de l'étude effectuée :

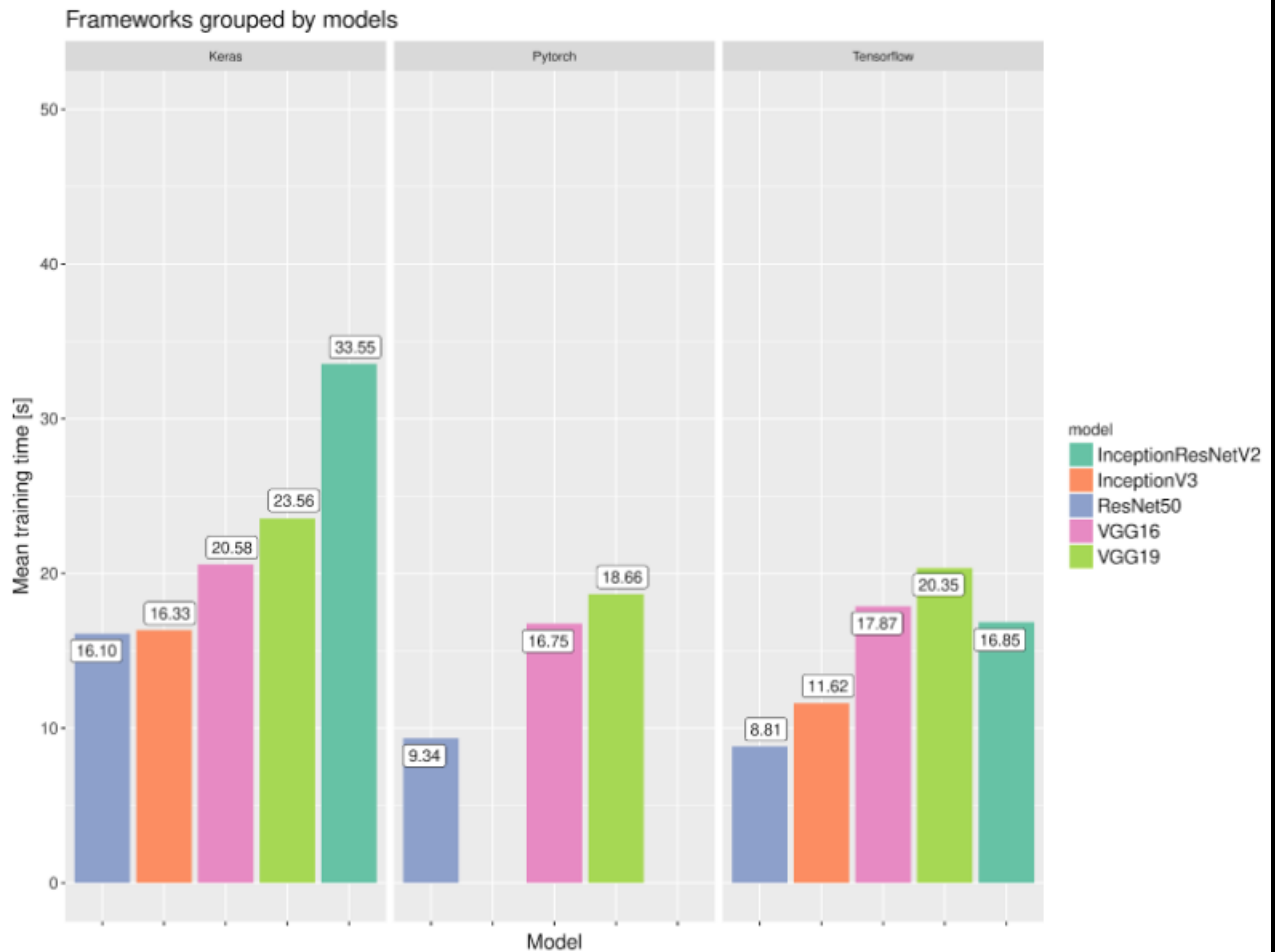


Figure 9 – Résultat de l'étude pour la rapidité d'entraînement

Sur ces graphiques, on peut voir les variations des temps d'entraînement de chaque modèle de CNN implémenté en fonction de la bibliothèque.

Dans notre cas, le modèle créé sera plus petit que les modèles cités présentés précédemment car notre modèle ne contiendra au maximum que trois couches alors que les modèles utilisés pour ce comparatif contiennent au moins 8 couches de convolution. Cela implique que la durée d'entraînement de notre modèle sera plus court.

2.1.5 Nombre minimum de lignes de code

Le facteur "facilité d'utilisation" a un fort impact sur le nombre de lignes de code à développer pour construire un modèle fonctionnel et utilisable. Pour cela, une étude documentaire liée au code à mettre en place a été faite pour répondre à ce critère, qui a donné ce résultat.

Les bibliothèques nécessitant le moins de lignes de code sont les suivantes :

1. PyTorch : via la redéfinition d'une classe (*Net*) où l'on redéfinit la méthode *forward* en indiquant la structure du réseau de neurones et on peut lancer un apprentissage par la suite. Environ 25 lignes de code sont nécessaires
2. Keras : s'agissant d'un framework haut niveau se basant sur une bibliothèque bas niveau, du moment qu'une de ces bibliothèques est utilisée (**CNTK**, **TensorFlow**, ...), cela nécessite très peu de lignes. Environ 20 lignes

3. Scikit-Learn : idem que PyTorch, il faut redéfinir une classe pour pouvoir l'utiliser. Environ 25 lignes de code sont nécessaires

TensorFlow est sujet à de prochaines mises à jour qui pourraient permettre de faciliter l'implémentation d'un modèle, même si cette bibliothèque reste compliquée à utiliser pour le moment.

2.1.6 Modèles de références implémentés

Les bibliothèques listées ci-dessous contiennent une implémentation des plus grands réseaux de neurones connus, pour la plupart à convolution. Dans le cas où le Perceptron implémenté ne serait pas suffisamment satisfaisant en terme de performances, il suffirait de changer l'implémentation de ce réseau en redéfinissant un modèle déjà implémenté et entraîné de cette bibliothèque.

Ces fameuses bibliothèques sont :

1. Keras
2. TensorFlow
3. PyTorch

2.1.7 Ne nécessite pas un environnement de fonctionnement spécifique?

Certaines bibliothèques listées pourraient être propriétaires et donc fonctionner sur un environnement propriétaire payant. Il faut que la bibliothèque que nous allons utiliser n'ait pas de contrainte de ce type, de préférence.

De toutes les bibliothèques choisies pour ce comparatif, aucune ne nécessite ce type d'environnement, ce qui est bien.

2.1.8 Connaissances sur cette bibliothèque

L'une des contraintes importantes est que le développeur ait des connaissances sur la façon d'implémenter un modèle construit avec cette bibliothèque, auquel cas, il faudrait prévoir du temps pour la formation.

Une formation portant sur le Deep Learning ayant été faite, des connaissances basiques ont été développées quant à la manipulation de réseaux de neurones avec la bibliothèque PyTorch

2.1.9 Choix final de la bibliothèque à utiliser

Suite à tous ces points, nos contraintes sont que le modèle doit pouvoir apprendre rapidement des données et classer des données dans un temps réduit. La bibliothèque ne doit pas nécessiter d'environnement spécifique d'exécution, il serait également préférable de pouvoir implémenter un modèle performant en minimisant le nombre de lignes de code à écrire pour favoriser la lecture rapide du code.

La bibliothèque remplissant à un maximum de ces critères est donc PyTorch, elle sera donc utilisée pour implémenter notre modèle.

2.2 Recherche des paramètres optimaux

Cette partie est consacrée à la recherche des paramètres optimaux pour la configuration du modèle. Nous parlerons de l'algorithme qui a été programmé et nous ferons un point sur les différentes fonctions d'activation.

2.2.1 Algorithme de recherche des paramètres optimaux

Cet algorithme a pour but de déterminer les paramètres optimaux pour lesquels un modèle, après avoir été construit, obtient le meilleur taux de reconnaissance en fonction des valeurs suivantes :

1. Taux d'apprentissage : de 0.01 à 0.02
2. Nombre de neurones de la couche cachées : de 1 à 30
3. Nombre de epoques : de 800 à 2.000

On définit donc une classe Network dont le constructeur prend en paramètre le nombre de neurones de la couche d'entrée (nombre de caractéristiques), le nombre de neurones dans la couche cachée et le nombre de neurones dans la couche de sortie (nombre de classes connues).

L'algorithme se trouve sur la page suivante.

```

donnees <- Données de la base d'apprentissage
validation <- Données de la base de validation
meilleur_precision <- 0
nbNeurones <- 0
nbEpoches <- 0
nbCarac <- Nombre de caractéristiques extraites
nbClasses <- Nombre de classes connues
meilleur_taux_reco <- 0
Pour taux_reconnaissance allant de 0.01 à 0.02 :
    precision <- 0
    Pour nb_neurones allant de 1 à 30 :
        epoches <- 800
        Tant que epoches < 2000 :
            net <- Network(nbCarac, nb_neurones, nbClasses)
            Pour epoch allant de 0 epoches :
                resultat <- net(donnees)
                Ajustement des poids des neurones en fonction de resultat
            Fin Pour
            res_prediction <- net(validation)
            precision <- CalculPrecision(res_prediction)
            Si precision > meilleur_precision :
                nbEpoches <- epoches
                nbNeurones <- nb_neurones
                meilleur_taux_reco <- taux_reconnaissance
            Fin Si
            epoches <- epoches * 1.2
        Fin Tant que
    Fin Pour
Fin Pour

```

Figure 10 – Algorithme de recherche de paramètres

2.2.2 Etude de la fonction d'activation à implémenter

Les fonctions d'activation testées sont les suivantes :

1. Sigmoid
2. Tangente Hyperbolique
3. ReLu
4. Softmax

Sigmoïde

La fonction Sigmoïde est définie par la courbe suivante :

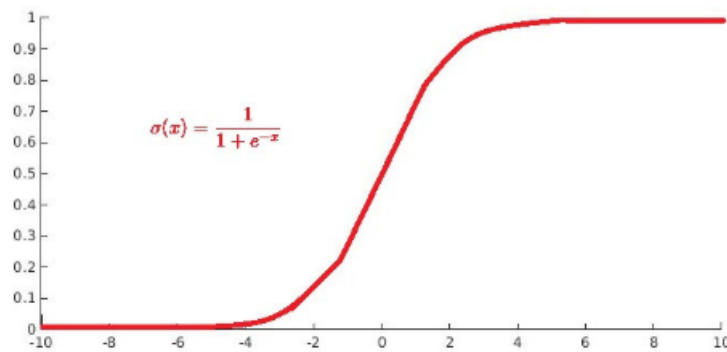


Figure 11 – Fonction Sigmoïde

Cette fonction est caractérisée par le fait qu'elle permet facilement de convertir une valeur quelconque et de la réduire sur l'intervalle $[0; 1]$, représentant donc les probabilités de classification. Ce qui fait que si une valeur négative est passée en entrée, la valeur résultante sera 0, s'il s'agit d'un grand nombre positif, le résultat sera 1.

L'un des inconvénients de cette fonction est qu'elle n'est pas centrée sur 0, ce qui fait que la majorité des valeurs susceptibles de faire varier la courbe doivent être faibles, les résultats d'entrées négatives peuvent également être positive.

Elle comprend également la fonction exponentielle, qui est caractérisée pour mettre longtemps à converger vers une valeur, ce qui fait que le temps de calcul est élevé. On peut aussi voir qu'elle partiellement plate, c'est-à-dire que le résultat qui est transmis est la plupart du temps proche de 0 ou 1, elle agit très peu sur les neurones par rapport à la correction des poids (saturation).

Tangente Hyperbolique

La courbe suivante représente la fonction tangente hyperbolique.

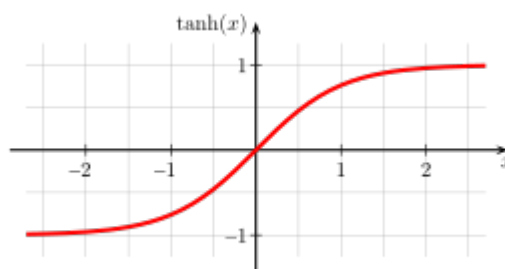


Figure 12 – Fonction Tangente Hyperbolique

Cette fonction est vue comme une alternative à la fonction Sigmoïde car elle, est centrée sur 0, ce qui élimine les chances que les valeurs négatives en entrée génèrent des sorties positives. Elle est également moins longue à traiter en termes de coût de calcul. Elle a en revanche, cette fonction possède le même défaut que la fonction Sigmoïde, qui est qu'elle plate à certains endroits, certains résultats sont proches de 0 et 1. Cela a un impact sur la mise à jour des poids des neurones, ce qui provoque une certaine saturation.

Il est préférable d'utiliser cette fonction, plutôt que la fonction Sigmoïde.

ReLu (Rectified Linear Unit)

La courbe associée à cette fonction est la suivante :

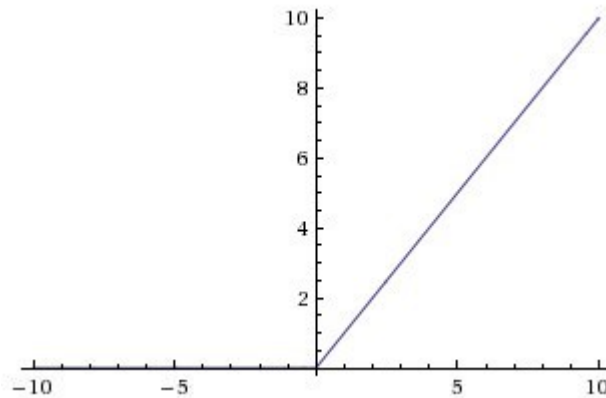


Figure 13 – Fonction ReLu

La fonction est définie par la relation suivante :

$$R(z) = \max(0, z)$$

Comme on peut le voir, cette fonction est caractérisée par le fait qu'elle est centrée sur 0 et que pour toute valeur d'entrée négative, le résultat retourné par cette fonction est 0. Toutes les valeurs positives ont toujours un résultat en sortie de cette fonction qui est positif.

Ce fonctionnement permet d'améliorer la vitesse de convergence vers le résultat et le fait qu'aucune valeur ne soit constamment proche de 0 ou 1 garantit de la non saturation des neurones.

Le principal problème de cette fonction vient du fait qu'elle soit nulle pour toute valeur négative : des valeurs négatives permettent automatiquement d'adapter son poids en fonction des valeurs reçues. Le fait qu'elles soient toutes nulles empêchent les neurones de les mettre à jour, ce qui a pour conséquence que le réseau n'apprend pas dans ce cas précis.

Des alternatives à ce problème existent pour cette fonction mais ne sont encore qu'en test (les résultats sont pour le moment non concluants).

Conclusion sur les fonctions d'activation

Pour conclure quant aux fonctions d'activation, nous avons pu voir qu'elles avaient chacun leurs avantages et inconvénients. Peu importe la fonction d'activation, elle dépend des données qui sont à traiter pour savoir si oui ou non, elle serait efficace sur celles-ci. C'est pourquoi un algorithme et plusieurs Perceptrons ont été développés pour comparer les performances pour pouvoir savoir quelle serait la fonction d'activation optimale à utiliser pour nos données.

La caractéristique de chaque Perceptron implémenté est qu'il ne contient que deux couches : une couche où toutes les données rentrent et où en sortie des neurones de cette couche, une fonction d'activation est appliquée et une dernière couche où une fonction softmax est appliquée.

Comme dit précédemment, chaque fonction d'activation génère ou est censée générer un résultat en sortie compris entre 0 et 1, correspondant à la probabilité d'appartenance d'une donnée à une de nos classes connues. Ce système fonctionne bien uniquement dans le cas où il faut classer de façon binaire des éléments (appartenance d'une donnée à une classe au choix entre deux possibles). Ce qui fait que pour de la classification multi classe, ce fonctionnement n'est pas optimal.

C'est pourquoi on utilise une couche softmax, qui a pour fonction de sommer à 1 toutes les probabilités d'appartenances aux différentes classes pour un individu. Le résultat est un tableau 1 dimension où toutes les probabilités d'appartenances aux différentes classes sont répertoriées sous cette forme :

$$\begin{bmatrix} 95\% \\ 3\% \\ \vdots \\ 2\% \end{bmatrix}$$

Figure 14 – Probabilités pour chaque classe pour un pixel

6

Mise en oeuvre

1 Bibliothèques utilisées

Les bibliothèques utilisées pour faire fonctionner le projet sont les suivantes :

1. pandas : manipulation et utilisation de fichiers comme CSV
2. Scikit-learn : manipulation et utilisation d'algorithmes pour le **Machine Learning** (principalement séparer les données du fichier d'apprentissage en plusieurs bases)
3. Numpy : manipulation d'outils scientifiques (tableaux, matrices,...)
4. Scipy : manipulation et traitement de données scientifiques plus poussées
5. PyQt5 : création d'interfaces graphiques
6. torch : manipulation et utilisation de tenseurs de données
7. torchvision : création et utilisation de structures de réseaux de neurones

2 Recherche des paramètres optimaux

Dans cette partie, on cherche la meilleure configuration pour un fichier CSV d'apprentissage contenant environ 14.000 pixels classifiés pour 41 caractéristiques extraites avec 2 classes connues.

2.1 Tests

Ces tests consistent en l'implémentation de Perceptrons où pour chaque, une fonction d'activation différente est utilisée avec une couche softmax. On recherche donc la meilleure configuration pour chaque Perceptron en fonction de la fonction d'activation et en faisant varier le taux d'apprentissage des exemples (0.01 et 0.02), le nombre de neurones dans la première couche (de 0 à 30) et le nombre de passages des exemples dans la boucle d'apprentissage (de 800 à 2000). Le fichier dont ces statistiques sont issus est appelé « 6.csv » et contient 2 classes connues. Un tableur contenant tous les résultats a été fait, les informations ci-dessous sont identiques.

Pour un taux d'apprentissage $lr = 0.01$:

Fonction d'activation	Meilleur taux de reco	Pire taux de reco	Taux de reco moyen	Temps d'exécution (en min)	Nb de neurones dans la couche	Nb de passages optimal
Sigmoïde	0,854881	0,671174	0,818498	31	19	1989
TanH	0,870712	0,670514	0,844290	21	14	1989
ReLu	0,878298	0,348944	0,729844	26	26	1989

Figure 1 – Tableau récapitulatif pour un taux d'apprentissage de 0.01

Pour un taux de reconnaissance $lr = 0.02$:

Fonction d'activation	Meilleur taux de reco	Pire taux de reco	Taux de reco moyen	Temps d'exécution (en min)	Nb de neurones dans la couche	Nb de passages optimal
Sigmoïde	0,868403	0,671174	0,836090	31	29	1989
TanH	0,882915	0,670514	0,854318	21	24	1989
ReLu	0,882255	0,348944	0,736966	26	29	1989

Figure 2 – Tableau récapitulatif pour un taux d'apprentissage de 0.02

Dans chaque relevé, on voit que la fonction Sigmoïde est plutôt efficace sur ce type de donnée : le pire taux de reconnaissance obtenu reste supérieur au taux de reconnaissance obtenu par l'utilisation de SVM dans la première application (environ 60%), tandis qu'avec ReLu, la fonction admet un taux de reconnaissance minimum de 0.34 et une moyenne de reconnaissance d'environ 72% contre respectivement 81% et 84% pour un taux d'apprentissage de 0.01. Pour les temps d'exécution, on peut également voir que l'algorithme a été le plus rapide pour déterminer la meilleure configuration pour le Perceptron implémentant la fonction TanH : 21 minutes environ contre 31 et 26 minutes.

Pour le relevé où le taux de reconnaissance est 0.02, le meilleur taux de reconnaissance avec la fonction ReLu est quasiment égal à celui de la fonction TanH. Cependant, le pire taux de reconnaissance et la moyenne la plus basse ont encore été trouvées pour la fonction ReLu malgré un bon temps d'exécution. Pour les fonctions Sigmoïde et TanH, on peut voir que les données pour la fonction TanH sont légèrement supérieures à celles de la fonction Sigmoïde sauf pour temps d'exécution (plus court pour TanH).

2.2 Conclusion

A travers ces relevés, on peut donc conclure que la fonction d'activation qui a été la plus efficace sur ces données a été majoritairement la fonction Tangente Hyperbolique, dont le meilleur taux de reconnaissance a légèrement augmenté grâce à l'augmentation du taux d'apprentissage et le temps d'exécution est pourtant resté constant alors que la moyenne des taux de reconnaissances a légèrement augmenté aussi.

Le Perceptron qui sera donc implémenté est le Perceptron dont les neurones de la couche d'entrée utilisent la fonction d'activation Tangente Hyperbolique.

7

Bilan et conclusion

1 Organisation du S9

En ce qui concerne la gestion de projet, comme dit précédemment, le projet sera mené en méthode **Scrum**, ce qui permet d'avoir toujours un retour des clients ou des encadrants techniques à chaque fin de sprint, peu importe le rendu : qu'il soit écrit ou programmé.

Pour l'organisation du projet, la Mindmap ci-dessous résume les tâches accomplies pendant la phase Recherche de ce projet.

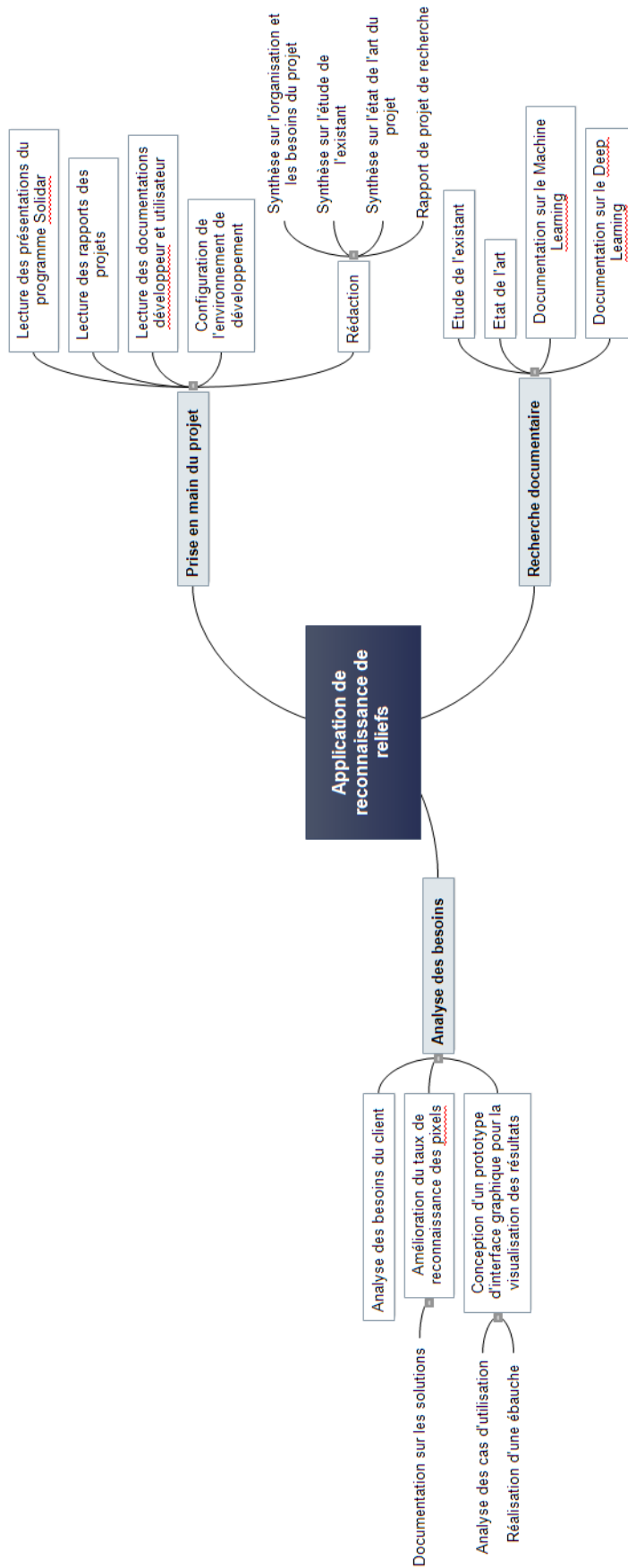


Figure 1 – Découpage de la phase Recherche du projet en sous-tâches

Comme on peut le voir, cette phase s'est découpée en trois grandes étapes :

1. **La prise en main** : consistant à prendre connaissance du sujet, comprendre ce qui est attendu, comprendre le travail effectué par les précédents étudiants ayant travaillé sur le projets ainsi que toutes les documentations, la configuration de l'environnement de travail et la rédaction de rapports
2. **La recherche documentaire** : cette partie a permis de comprendre les thématiques abordées de *Machine Learning* et *Deep Learning*, ainsi que sur le fonctionnement du classifieur linéaire implémenté, le *Séparateur à Vaste Marge*. Des recherches ont également été faites pour trouver des bibliothèques à implémenter pour la manipulation de réseaux de neurones convolutifs
3. **L'analyse des besoins** : cette partie consistait à partir des attentes des clients, de trouver un moyen permettant de corriger ces problèmes en proposant des solutions basées sur des technologies existantes comme l'implémentation d'un réseau de neurones convolutif pour améliorer le taux de reconnaissance

A partir de cette Mindmap, le Gantt ci-dessous résume le déroulement de ces tâches.

	📌	Nom de tâche	Durée	Début	Fin	Prédécesseurs	Progression
1	✓	Application de reconnaissance de reliefs	23,5 jours	19/09/2018	06/12/2018		100%
2	✓	☐ Prise en main du projet	23,5 jours	19/09/2018	06/12/2018		100%
3	✓	Lecture des présentations du programme <u>Solidar</u>	1,5 jours	19/09/2018	20/09/2018		100%
4	✓	Lecture des rapports des projets	2 jours	20/09/2018	27/09/2018	3	100%
5	✓	Lecture des documentations <u>développeur</u> et <u>utilisateur</u>	1,5 jours	27/09/2018	03/10/2018	4	100%
6	✓	Configuration de l'environnement de développement	4 jours	04/10/2018	17/10/2018	5	100%
7	✓	☐ Rédaction	14,5 jours	18/10/2018	06/12/2018		100%
8	✓	Synthèse sur l'organisation et les besoins du projet	0,5 jour	18/10/2018	18/10/2018	6	100%
9	✓	Synthèse sur l'étude de l'existant	0,5 jour	25/10/2018	25/10/2018	13	100%
10	✓	Synthèse sur l'état de l'art du projet	1 jour	01/11/2018	01/11/2018	14	100%
11	✓	Rapport de projet de recherche	2 jours	29/11/2018	06/12/2018	23	100%
12	✓	☐ Recherche documentaire	8 jours	18/10/2018	15/11/2018		100%
13	✓	Etude de l'existant	2 jours	18/10/2018	25/10/2018	8	100%
14	✓	Etat de l'art	1 jour	31/10/2018	31/10/2018	9	100%
15	✓	Documentation sur le <u>Machine Learning</u>	1,5 jours	07/11/2018	08/11/2018	10	100%
16	✓	Documentation sur le <u>Deep Learning</u>	2 jours	08/11/2018	15/11/2018	15	100%
17	✓	☐ Analyse des besoins	4 jours	15/11/2018	29/11/2018		100%
18	✓	Analyse des besoins du client	1 jour	15/11/2018	21/11/2018	16	100%
19	✓	☐ Amélioration du taux de reconnaissance des <u>pixels</u>	1 jour	21/11/2018	22/11/2018		100%
20	✓	Documentation sur les solutions	1 jour	21/11/2018	22/11/2018	18	100%
21	✓	☐ Conception d'un prototype d'interface graphique pour la visualisation des résultats	2 jours	22/11/2018	29/11/2018		100%
22	✓	Analyse des cas d'utilisation	1 jour	22/11/2018	28/11/2018	20	100%
23	✓	Réalisation d'une ébauche	1 jour	28/11/2018	29/11/2018	22	100%

Figure 2 – Diagramme de Gantt

Pour résumer, la tâche qui a pris le plus de temps est bien la configuration de l'environnement car des problèmes sur la version des paquets à installer et des interpréteurs Python ont été rencontrés.

Pour conclure sur le déroulement de la phase de recherche, malgré le fait que des problèmes ont été rencontrés comme la configuration ou la difficulté à comprendre les différentes notions, toutes les tâches prévues ont été effectuées dans les temps.

A l'avenir, il faudra élaborer des tests pour vérifier et valider le comportement des bibliothèques Python vues précédemment, pouvant être implémentées pour la construction d'un réseau de neurones convolutif. Des scénarios de tests sont à prévoir, ainsi que les cas dans lesquels ils seront validés.

2 Organisation du S10

Concernant l'organisation du semestre 10, le gantt associé avec l'avancement des tâches est le suivant :

	Nom de tâche	Durée	Début	Fin	Prédécesseurs	Progression
1	Application de reconnaissance de reliefs	24 jours	16/01/2019	04/04/2019		90%
2	☐ Documentation	1,5 jours	16/01/2019	17/01/2019		100%
3	Recherche d'informations détaillées sur les bibliothèques de <u>CNN</u>	1 jour	16/01/2019	17/01/2019	9	100%
4	Comparatif bibliothèques <u>CNN</u> Python	4 hr	17/01/2019	17/01/2019	3	100%
5	☐ Analyse	1 jour	23/01/2019	23/01/2019		100%
6	Diagrammes <u>UML</u> de l'application de classification par <u>CNN</u>	1 jour	23/01/2019	23/01/2019	4	100%
7	☐ Tâches annexes	0,5 jour	16/01/2019	16/01/2019		100%
8	Récupération de l'application utilisée par les chercheurs	2 hr	16/01/2019	16/01/2019		100%
9	Tests de l'application	2 hr	16/01/2019	16/01/2019	8	100%
10	☐ Développement	18 jours	24/01/2019	27/03/2019		89%
11	Codage de l'application	14 jours	24/01/2019	13/03/2019	6	100%
12	<u>Debuggage</u>	2 jours	14/03/2019	20/03/2019	11	100%
13	Tests de l'application	2 jours	21/03/2019	27/03/2019	12	0%
14	Génération des données au format <u>CSV</u>	2 hr	21/03/2019	21/03/2019	12	100%
15	☐ Rédaction	21 jours	24/01/2019	04/04/2019		85%
16	Documentation du code	2 hr	24/01/2019	24/01/2019	6	100%
17	Documentation de tests	4 hr	28/03/2019	28/03/2019	13	0%
18	Documentation d'installation et d'utilisation	4 hr	28/03/2019	28/03/2019	17	100%
19	Rapport du S10	2 jours	03/04/2019	04/04/2019	18	100%

Figure 3 – Diagramme de Gantt du S10

On peut voir que certaines tâches n'ont malheureusement pas pu être faites en raison d'un retard conséquent. Les livrables de certaines tâches comme la construction d'un comparatif de bibliothèques de Deep Learning Python sont non conformes aux attentes (hors sujet), ce qui fait que le retard total accumulé atteint les deux semaines.

La plupart des tâches qui ont été faites, ont été réalisées sur le temps libre pour combler le retard, à la fois à cause d'une très mauvaise organisation et une mauvaise compréhension des attentes des encadrants.

Une application a quand même été entièrement développée autour d'un Perceptron, permettant d'assurer un taux de reconnaissance correct grâce aux choix techniques faits et expliqués précédemment. Le temps a également manqué pour le développement de tests unitaires et la rédaction d'un plan de tests.

3 Conclusion du projet

Pour conclure sur le déroulement de ce projet, plusieurs points importants ont manqué comme la communication entre les encadrants et l'étudiant pour les informer ou encore une mauvaise compréhension des attentes des encadrants a provoqué un retard significatif sur le déroulement du projet.

D'autres problèmes ont été rencontrés quant à l'installation et la configuration de l'environnement de l'application **SkyEye**, manquant de documentation technique et claire. Pour le développement de l'application présentée dans ce rapport, il a fallu intégralement installer un environnement Python différent de l'application **SkyEye** en raison d'une version d'interpréteur incorrecte pour l'installation et la configuration des modules Python.

Pour finir, je souhaite remercier Monsieur Hubert Cardot pour le temps qu'il m'a accordé pour des réunions qui m'ont permis de mieux comprendre le fonctionnement des réseaux de neurones et l'idée d'implémenter l'algorithme de recherche des paramètres.

Annexes

A

Diagramme de cas d'utilisation

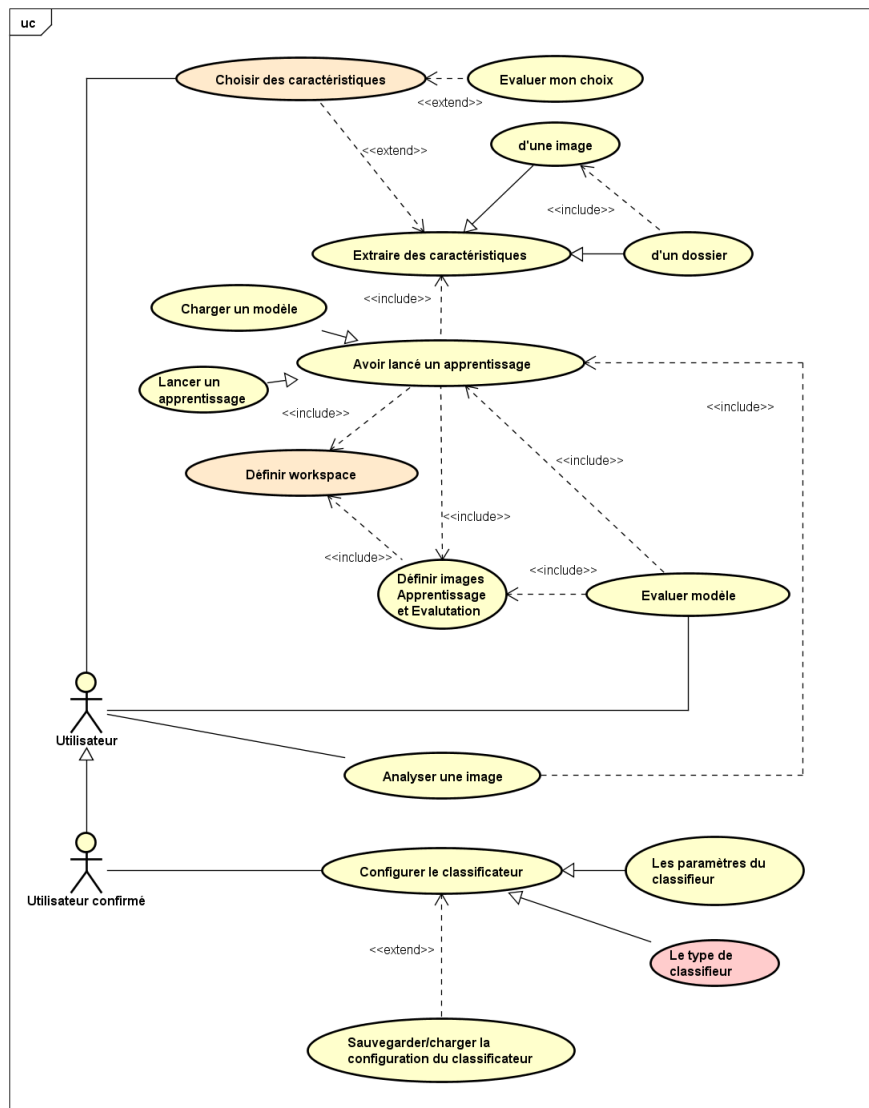


Figure 1 – Diagramme de cas d'utilisation

B

Cahier des 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 SPECIFICATION

Projet : CDS14

Implémentation du principe de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie

Emetteur :

Xavier Bouchenard

MOA : Jean-Yves Ramel
Thierry Brouard
Xavier Rodier
Clément Laplaige

Date d'émission :

25/10/2018

Validation

Nom

Date

Valide (O/N)

Commentaires

Historique des modifications

Version

Date

Description de la modification

0

25/10/2018

Création du document + Rédaction de l'introduction

0.1

01/11/2018

Rédaction de l'introduction + existant

0.2

21/11/2018

Début rédaction des spécifications fonctionnelles

1.0

28/11/2018

Fin de la rédaction des spécifications + relecture du document

1.1

06/03/2019

Correction des spécifications fonctionnelles

TABLE DES MATIERES

Cahier de spécifications	3
1. Introduction	3
1.1. Les acteurs du projet.....	3
1.2. Contexte de la réalisation	4
1.3. Description générale	5
1.4. Structure générale du système	7
2. Spécifications fonctionnelles.....	9
2.1. Amélioration du taux de reconnaissance	10
2.2. Conception de l'interface graphique de visualisation	13
3. Spécifications non fonctionnelles	15
3.1. Contraintes de développement et conception	15
3.2. Contraintes de fonctionnement et d'exploitation	15
4. Autres tâches importantes.....	15
4.1. Choix de la bibliothèque Python de Deep Learning	16
4.2. Tests de validation de la bibliothèque	16
4.3. Formation sur les réseaux de neurones	16
4.4. Amélioration du dépôt GitLab.....	16
5. Conclusion	16
Bibliographie	17
Tables des illustrations.....	18

1. Introduction

Dans le cadre de la 5^{ème} et dernière année d'ingénieur à Polytech'Tours, un Projet de Recherche et Développement est confié aux étudiants pour leur permettre de travailler sur un projet de grande envergure et d'appliquer les bases méthodologiques et connaissances apprises en cours. Ce projet se déroule du 19 septembre jusqu'à fin mars 2019 où par semaine, 2 jours lui sont consacrés, soit 16 heures par semaine.

A l'issue de ce projet, 2 rapports seront rédigés ainsi qu'un cahier de spécifications, dans lequel sera défini les attentes du client ainsi que les moyens pour y parvenir. Deux soutenances orales sont également prévues pour présenter le travail effectué pendant ces séances.

Le sujet de ce Projet de Recherche et Développement (PRD), vise l'amélioration d'une application de reconnaissance de reliefs dans des images archéologiques. Cette application consiste à extraire toutes les caractéristiques (zones d'intérêt) des pixels des images dont les reliefs sont identifiés, pour ensuite les analyser et déterminer à quel type de relief appartient le pixel d'images passées en paramètre de cette application.

Ce projet est l'objet d'une étude dans laquelle les besoins du client sont exprimés puis résolus grâce à l'expertise technique et les connaissances acquises de l'étudiant, ainsi que de ses encadrants techniques.

Ce document est donc le cahier de spécifications du projet « Implémentation du principe de **Deep Learning** dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie ».

1.1. Les acteurs du projet

- La maîtrise d'œuvre (MOE) : Xavier Bouchenard, étudiant en 5^{ème} année de l'Ecole d'Ingénieur Polytech Tours au sein du département Informatique, dans le cadre du PRD ainsi que ses encadrants : Mr Jean-Yves Ramel et Mr Thierry Brouard de l'équipe de recherche RFAI – EA 6300 du Laboratoire Informatique de Tours (<http://www.rfai.li.univ-tours.fr>).



Figure 1: Logo du groupe RFAI du Laboratoire Informatique de Tours

Les recherches menées au sein de cette équipe portent sur l'étude et la résolution des problèmes de reconnaissance d'images, que ce soit dans le cadre de problématiques « industrielles » ou de problèmes théoriques.

- La maîtrise d'ouvrage (MOA) : Le projet SOLiDAR du laboratoire Archéologique et Territoire CITERES de Tours, représenté par Clément LAPLAIGE et Xavier RODIER. Créé en 2004, ce laboratoire basé sur Tours a pour objectif d'analyser les dynamiques spatiales et territoriales des sociétés. Dans ce contexte, leur champ de recherche se compose de quatre secteurs : la recherche urbaine, la recherche environnementale, les travaux sur le territoire et ceux sur les effets des recompositions sociales contemporaines. (<http://citeres.univ-tours.fr>).



Figure 2 - Logo du projet SOLiDAR et du laboratoire CITERES de Tours

Ce document a été rédigé par Xavier Bouchenard et sera validé par les encadrants du projet et l'ensemble des personnes constituant la MOA qui sont également les relecteurs de ce cahier.

1.2. Contexte de la réalisation

1.2.1. Contexte général

L'origine de ce projet provient du travail d'archéologues utilisant un nouveau système de détection basé sur la technologie *LiDAR* ou **Light Detection And Ranging** permettant d'acquérir des images géographiques. Ensuite, à partir des images obtenues, de nouvelles images seront générées en fonction de leurs besoins. L'une des images générées qui sera par la suite, la plus utilisée représente un relief en nuances de gris, enregistrée au format **.TIF**.

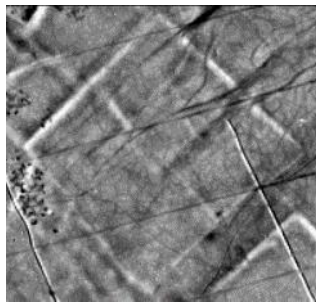


Figure 3 - Représentation d'un relief en nuances de gris

Pour répondre à une demande d'automatiser le process de détection de zones archéologiques, une première application a été développée et est actuellement utilisée par les archéologues. Ce logiciel permet d'identifier la nature des pixels de chaque région de l'image (fossé, talus, chemin, ...) grâce à des calculs de probabilités. Pour se faire, une méthode de reconnaissance linéaire appelée **SVM** (Séparateur à Vaste Marge) a été implémenté.

Suite à l'utilisation de ce logiciel, quelques remarques ont été faites par le client liées au taux de reconnaissance de l'application, jugée faible, environ 60% de reconnaissance correcte dans certains cas.

1.2.2. Objectifs

La principale amélioration à apporter sur cette application est donc l'amélioration de la précision du modèle. Il s'agit d'un taux correspondant au nombre de pixels identifiés appartenant au relief prédit sur le nombre de pixels de toutes les images analysées.

Cette amélioration consiste donc à trouver une méthode permettant de corriger le manque de précision du modèle pour obtenir un taux de reconnaissance plus élevé.

Ce problème a été constaté par le client, le taux de reconnaissance des pixels ne s'élève pas au-delà de 65%, dans certains cas. Il est donc important de l'améliorer.

Une problématique également abordée est la difficulté, pour l'utilisateur, de modifier des classifications effectuées par l'algorithme **SVM**, dans le cas où celui-ci se serait trompé. De ce côté-là, l'interaction entre l'application et l'utilisateur est limitée. Une fonctionnalité supplémentaire dédiée à ce problème a été pensée.

Malgré ce qui a été vu précédemment, les clients ont fait la demande pour corriger le manque de précision de l'algorithme **SVM** par l'implémentation d'un réseau de neurones. C'est pourquoi ce problème sera donc traité en priorité.

Pour éviter d'éventuels problèmes d'interactions entre les deux méthodes de résolution, une nouvelle application sera développée autour du réseau de neurones construit, ce qui permettra d'éviter quelques problèmes de compatibilité comme la version des interpréteurs.

a) Taux de reconnaissance faible

La reconnaissance des pixels est basée sur une méthode de résolution linéaire, ce qui fait que si le système n'a pas suffisamment d'exemples dont la nature et les caractéristiques sont différentes (taux de luminosité, ...), il aura plus de chances de prendre une mauvaise décision.

Le problème lié à ça est qu'il est difficile de juger quand la machine a suffisamment d'exemples, pour pouvoir construire un modèle et pouvoir juger s'il est suffisamment précis pour se baser dessus pour les prochaines décisions à prendre.

C'est pourquoi l'une des méthodes de résolution serait soit d'implémenter une méthode de **Deep Learning** par la mise en place d'un réseau de neurones ou par l'amélioration de la précision du classifieur SVM. Les membres constituant la MOA de ce projet ont formulé l'envie d'implémenter le principe de Deep Learning, une première version d'application sera développée embarquant comme méthode de reconnaissance un réseau de neurones. Le fait de dissocier, dans un premier temps, les deux méthodes de résolution, permet de comparer plus facilement les résultats obtenus.

b) Interaction limitée avec le système

Dans le cas où le système fait une erreur de jugement, l'utilisateur doit pouvoir être capable de modifier la décision prise. Il faut ensuite que la modification faite, soit prise en compte par la suite dans le modèle, pour pouvoir être reconstruit et éviter de faire la même erreur par la suite.

1.2.3. Bases méthodologiques

Pour ce projet, la méthode de gestion utilisée est la méthode **Scrum**, ce qui permet au client de suivre plus facilement le projet grâce au découpage du projet en tâches, associées à des *sprints* (livraison d'une fonctionnalité après réalisation dans un temps donné). Le suivi du projet sera plus simple par le client, une fois que des fonctionnalités seront livrées, elles pourront être testées et validées par le client en personne.

En ce qui concerne le développement, il se fera en langage Python avec l'IDE *PyCharm* qui permet facilement de gérer les bibliothèques utilisées. Une convention de nommage a également été définie, il s'agit de la norme PEP 8.

1.3. Description générale

1.3.1. Environnement du projet

L'application a été développée en Python pour des machines fonctionnant sous Windows avec une version d'interpréteur supérieure ou égale à la 3.6.2 et les modules torch et torchvision pour l'implémentation et l'utilisation d'un réseau de neurones.

1.3.2. Caractéristiques des utilisateurs

L'application est prévue pour être utilisée par des archéologues, pour leur permettre de faciliter leur travail. Pour que l'utilisateur puisse utiliser correctement l'application, il faut qu'il sache comment utiliser l'application de façon instinctive et qu'elle soit le plus simple possible. Il n'a cependant pas besoin de connaître et de savoir paramétrer le réseau de neurones : un algorithme déterminera les meilleurs paramètres pour cela.

1.3.3. Fonctionnalités du système

L'application à développer se base principalement sur l'application SkyEye, qui embarque la méthode de reconnaissance SVM. Les fonctionnalités caractérisant cette nouvelle application sont les suivantes :

- **Chargement des fichiers CSV** : l'utilisateur définit le chemin d'accès aux fichiers d'apprentissage et à prédire, puis les charge dans l'application. Cette étape correspond au chargement des données des fichiers pour pouvoir être ensuite utilisées
- **Chargement du fichier contenant le modèle** : cette étape consiste à charger un modèle de réseau de neurones qui a été précédemment construit et que l'utilisateur a jugé bon en raison d'un taux de reconnaissance suffisamment élevé. Il choisit donc le fichier, puis le charge pour que la structure du réseau soit reconstruite et réutilisable
- **Sauvegarde du modèle dans un fichier** : une fois qu'un modèle de réseau de neurones ait été construit et entraîné, si l'utilisateur juge que le modèle actuel possède de bonnes performances de classification, il est possible de sauvegarder sa structure et la base d'apprentissage associée dans un fichier qui sera défini par l'utilisateur
- **Entraînement du modèle et évaluation** : dès qu'un fichier d'apprentissage a été sélectionné puis chargé, l'utilisateur peut entraîner le modèle grâce aux données du fichier d'apprentissage. Une partie des données de ce fichier sont mis de côté pour constituer une base de validation, c'est-à-dire que l'on calcule le taux de reconnaissance sur ces données en lançant une prédiction sur celles-ci. Le résultat de cette prédiction ou évaluation, est affichée à l'écran
- **Paramétrage du classifieur** : pour obtenir un taux de reconnaissance optimal, il faut que le réseau de neurones soit le plus adapté possible. Un algorithme sera donc exécuté pour déterminer la meilleure configuration à partir des données du fichier d'apprentissage
- **Lancement de la classification** : dès que le modèle a été construit et entraîné, si un fichier dont les pixels sont à prédire est sélectionné et chargé, il est possible de lancer une classification en passant en paramètre du modèle, les données de ce fichier. On obtiendra en sortie une matrice contenant pour chaque pixel, sa probabilité d'appartenance à une classe, ainsi que les labels associés aux pixels. Ces résultats seront stockés dans deux fichiers

Le diagramme des cas d'utilisation ci-dessous récapitule tous les points vus jusqu'à maintenant.

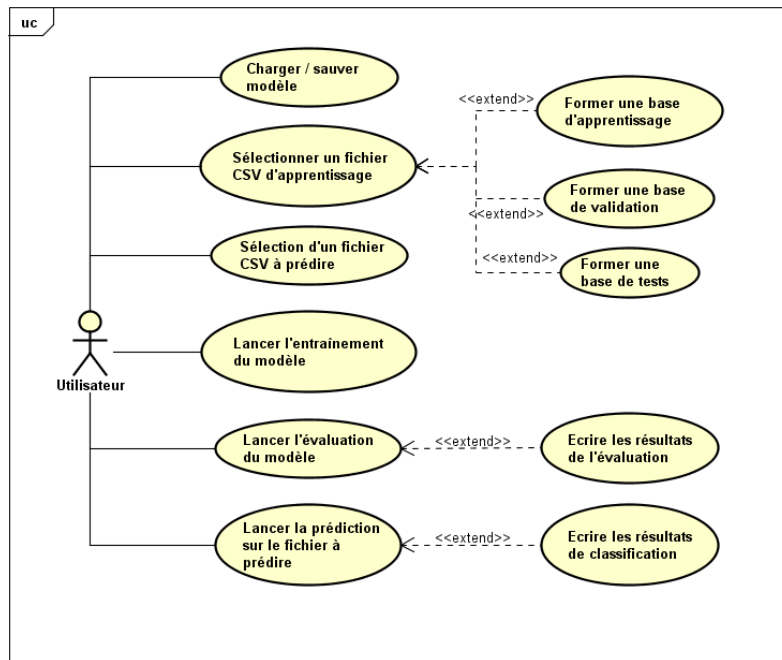


Figure 4 - Diagramme des cas d'utilisation actuelle

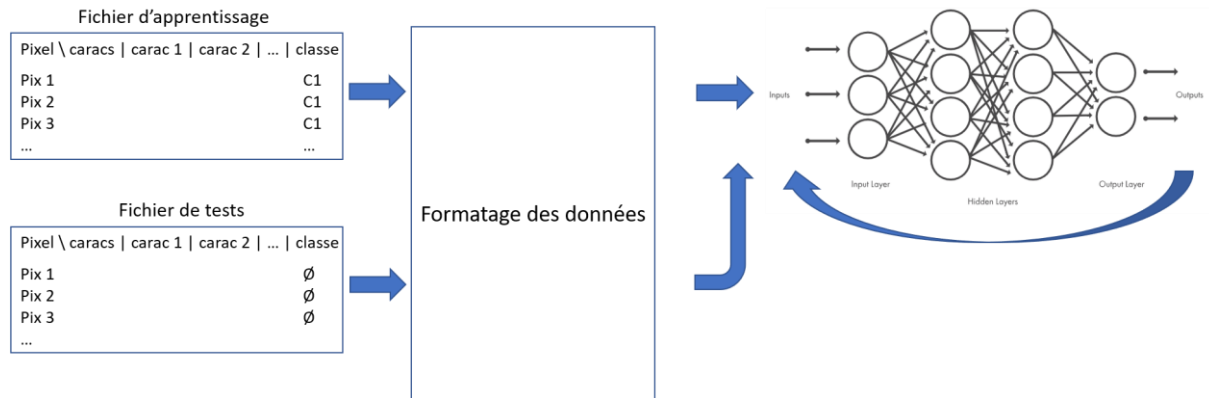
Avant de lancer l'identification, il faut que le modèle préalablement construit à partir des images de configuration et des échantillons de reliefs étiquetés, soit entraîné et sache différencier les reliefs ou classes d'appartenance.

1.4. Structure générale du système

Le système est composé de plusieurs partis :

- L'interface Homme/Machine permettant à l'utilisateur de charger les fichiers CSV d'apprentissage et à prédire, de charger un fichier contenant un modèle ou de le sauvegarder
- La lecture du contenu de ces fichiers et le chargement de leurs données qui constituent des jeux de données appelés bases : base d'apprentissage, base de test, base de validation et base à prédire
- La classe DataHandler, qui va contenir toutes les informations que ce soit du modèle construit pour l'entraînement ou le lancement d'une reconnaissance ou les données des fichiers chargés, stockées sous forme d'objets
- L'apprentissage lancée par l'utilisateur sur des données dont le label est à prédire, par l'appui d'un bouton, une fois que le modèle est construit et entraîné

L'architecture de l'application est résumée par le schéma ci-dessous :



On laisse l'utilisateur passer en entrée de l'application un fichier correspondant au fichier d'apprentissage qui est un fichier CSV contenant l'ensemble des pixels d'une ou plusieurs image(s) avec pour chacun un label associé. On passe également un fichier à prédire où, cette fois, les pixels n'ont pas de labels et doivent être déterminés.

On crée par la suite plusieurs bases dans lesquelles les données des pixels sont stockées, puis utilisées sous forme de tenseurs qui est le type de données que prend le réseau de neurones en paramètre. S'il n'est pas défini, on construit un modèle à partir des données du fichier d'apprentissage et on fait apprendre celles d'entraînement à ce nouveau réseau.

On pourra, par la suite, lancer une prédiction sur les données à prédire en passant en paramètre du réseau, le tenseur contenant ces données.

Le contenu de chaque fichier CSV sélectionné et chargé est approximativement le suivant :

- Le fichier d'apprentissage :

Class	x	y	AvgD_15x15	AvgD_19x19	AvgD_3x3
C0	0	307	44.46	99.18	25.5
C0	0	700	21.4	18.89	16.75

Figure 5: Contenu type d'un fichier d'apprentissage

L'image ci-dessus n'est qu'un échantillon d'un fichier d'apprentissage, il comporte bien plus de caractéristiques et de données. Il est caractérisé par le fait que des labels sont donnés aux pixels pour faire référence au type de relief présent dans l'image initiale.

- Le fichier à prédire :

Class	x	y	AvgD_15x15	AvgD_19x19
	0	174	30.85	21.21
	0	224	18.44	18.3
	0	440	23.54	16.2

Figure 6: Contenu type du fichier à prédire

L'image ci-dessus n'est qu'un échantillon d'un fichier à prédire, il comporte bien plus de caractéristiques et de données. Il est caractérisé par le fait qu'aucun label n'est donné à un pixel (première colonne). Ces labels sont donc à déterminer.

Le diagramme de classes ci-dessous reprend toutes les fonctionnalités qui sont aujourd'hui implémentées.

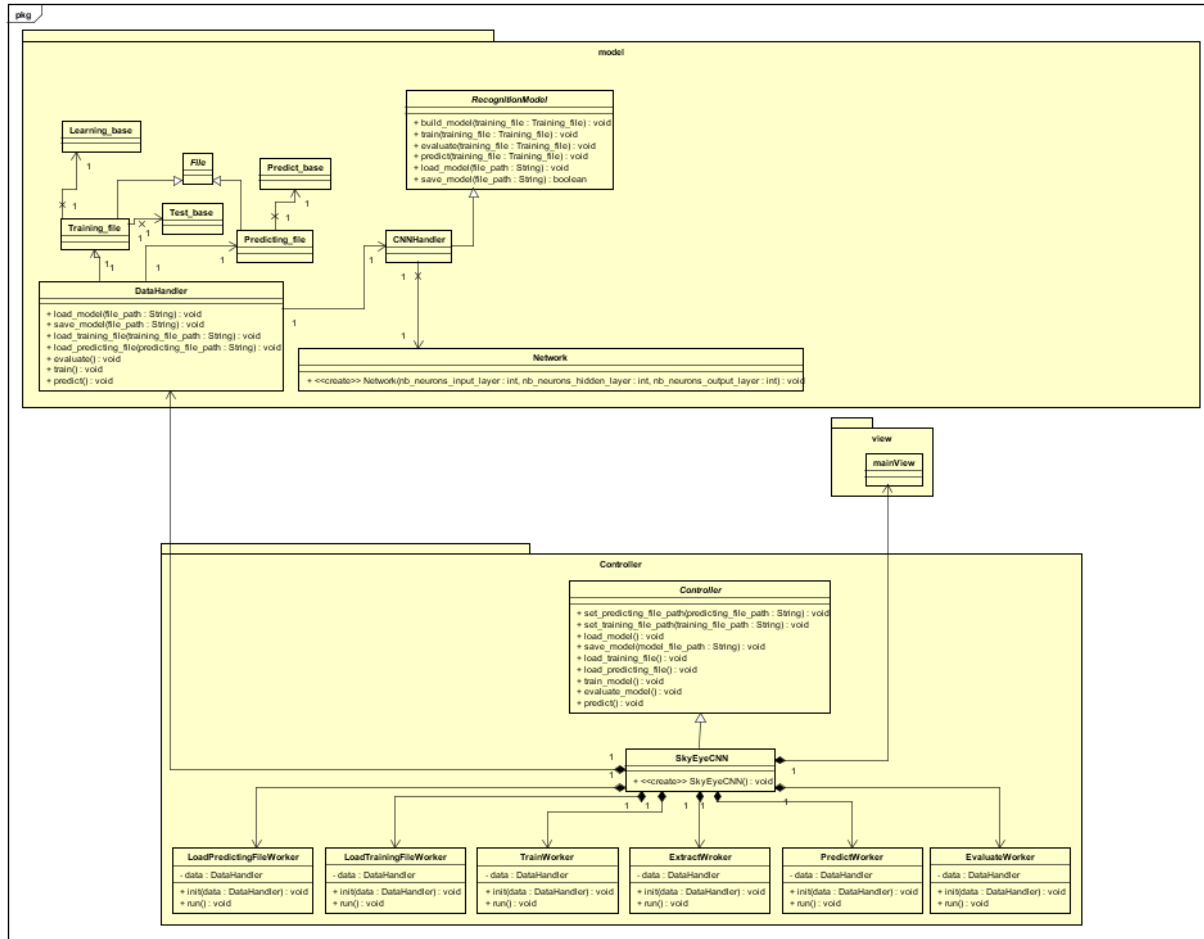


Figure 7 - Diagramme de classes de l'application

Pour rendre le code plus lisible, le projet a été découpé en package de fonctionnalités. Les principaux packages sont les suivants :

- **Model** : contient toutes les classes utilisant directement ou stockant des données de fichiers sous forme d'objets comme le modèle de réseau de neurones à construire ou les données extraites sous forme de bases des fichiers CSV qui auront été chargés
- **View** : correspond à l'unique vue graphique de l'application, gérant les appels à des méthodes d'un contrôleur
- **Controller** : gère les actions de l'utilisateur par des méthodes appelées lors de l'appui sur un bouton de l'interface graphique de l'application

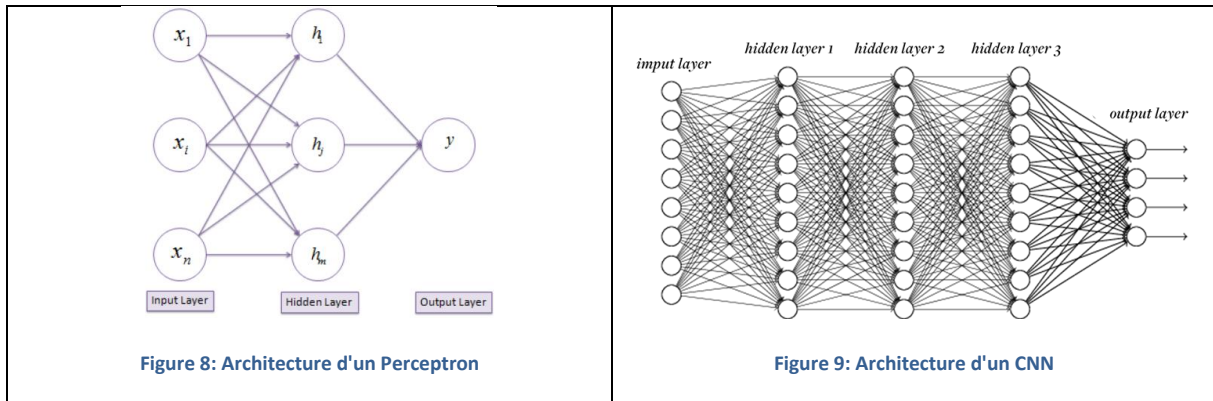
2. Spécifications fonctionnelles

2.1. Amélioration du taux de reconnaissance

2.1.1. Description

Comme vu précédemment, cette fonctionnalité a pour but d'améliorer le taux de reconnaissance par l'implémentation d'un réseau de neurones qui sera le noyau d'une nouvelle application. Cette fonctionnalité est prioritaire sur les autres.

Dans notre cas, les fichiers passés en entrée du réseau ne sont pas des images mais directement les données de nos fichiers CSV, c'est pourquoi l'approche est un peu différente. Dans ce cadre, deux grandes architectures connues peuvent être implémentées : le réseau de neurones à convolution (CNN) ou le Perceptron.



Appliquée à notre problème de classification, il a été conseillé d'utiliser une structure simple comme le Perceptron pour voir les performances d'une architecture de ce type pour notre problème. Une étude de performances sera faite par la suite grâce au développement d'un algorithme pour la recherche de la configuration optimale.

Sa structure est également à définir, nous savons cependant que le nombre de neurones d'entrée correspond au nombre de caractéristiques présentes dans notre fichier d'apprentissage (variable, nb de colonnes), le nombre de neurones de sortie du Perceptron correspond au nombre de classes connues et apprises par le modèle. Le nombre de neurones dans la couche cachées reste tout de même à déterminer ainsi que la fonction d'activation optimale.

La couche de sortie du réseau va donc récupérer le résultat de la prédiction pour chaque pixel sous forme de probabilité d'appartenance, pour chaque classe. L'ensemble de ces probabilités seront donc récupérées et écrites manuellement dans un fichier texte pour avoir une trace des décisions prises par l'application, en fonction du fichier CSV sélectionné à prédire.

Certains détails précis manquent car ils sont liés à la bibliothèque à utiliser, comme les paramètres correspondants. En fonction de la spécificité de chaque bibliothèque, le comportement est différent et le choix est donc difficile à faire.

Des connaissances sont manquantes, c'est pourquoi une formation sur la compréhension et l'utilisation approfondie des réseaux de neurones est prévue.

En ce qui concerne l'intégration de cette architecture, la structure du code prévue de l'application permet de faire en sorte à ce que d'autres méthodes de reconnaissances puissent être implémentées par redéfinition

des méthodes standards (entraînement, évaluation, prédiction, ...). La structure de la nouvelle application est profondément inspirée de l'application SkyEye.

2.2. Algorithme de recherche de la configuration optimale du réseau de neurones

2.2.1. Description

Pour déterminer la meilleure configuration, il faut prendre en compte plusieurs paramètres. Ces paramètres sont :

- Le taux d'apprentissage : ne pas le fixer au-dessus de 0.05 : trop élevé, entraîne du surapprentissage, il ne pourra que reconnaître les classes connues les plus représentées dues à un apprentissage de la même donnée)
- Le nombre de couches cachées
- Le nombre de epoques : nb de passages des données d'apprentissage dans la boucle
- Le type de fonction d'activation

Ces critères ont un impact direct sur les performances de reconnaissance du futur modèle construit. Il faut également s'assurer à limiter les plages de variation de ces valeurs, d'une part, pour éviter que cette recherche ne dure trop longtemps, d'autre part, certaines valeurs sont connues pour ne pas être adaptées au réseau. Les plages de variation pour chaque critère sont donc les suivantes :

- Taux d'apprentissage : de 0.01 à 0.02
- Nombre de couches cachées : entre 1 et 30
- Nombre de epoques : entre 800 et 2.000 itérations (si le taux d'apprentissage est faible, il est conseillé de fixer une valeur élevée pour ce nombre)

Pour tester, les fonctions d'activation les plus connues de la littérature seront utilisées, il s'agit respectivement de la fonction tangente hyperbolique, sigmoïde, ReLu et softmax. Un réseau de neurones avec comme fonction d'activation celles citées seront utilisées pour dresser ensuite un comparatif, ce qui nous permettra par la suite de sélectionner et valider l'efficacité de celle choisie.

2.3. Chargement des fichiers CSV et séparation des données sous forme de bases

2.3.1. Description

Cette partie permet, une fois que l'utilisateur a choisi les fichiers CSV à importer, de charger ces fichiers et de constituer des ensembles des données. Quatre ensembles dits « bases » seront constituées qui sont respectivement :

- **La base d'apprentissage** : données issues du fichier d'apprentissage permettant au réseau de neurones d'apprendre les caractéristiques de chaque classe et de les généraliser pour pouvoir reconnaître d'autres individus
- **La base de tests** : données issues du fichier d'apprentissage utilisées pour évaluer le modèle créé à partir des paramètres trouvés, ce qui permet d'extraire une matrice de confusion résultante et un taux de reconnaissance global pour visualiser la qualité de la classification

- **La base de validation** : données issues du fichier d'apprentissage utilisées pour vérifier les performances du réseau lors de la recherche de la meilleure configuration par le calcul du taux de reconnaissance
-
- **La base de prédiction** : données issues du fichier à prédire contenant l'ensemble des pixels dont le label doit être défini

En ce qui concerne la séparation des données en différentes bases pour le fichier d'apprentissage, elle est la suivante :



Il est souhaitable de garder la plus grande proportion de données pour l'apprentissage, ce qui laisse une chance d'avoir des individus appartenant à la même classe mais dont les caractéristiques sont légèrement différentes pour pouvoir repérer et apprendre ces non similarités. Cela permet au modèle de pouvoir avoir plusieurs exemples différents d'individus de la même classe.

Pour les proportions, environ 60% des données de ce fichier sont utilisées pour l'apprentissage, 20% pour la validation et les autres 20% pour le test.

2.4. Interface de l'application implémentant le réseau de neurones

2.4.1. Description

Le but de cette fonctionnalité est de pouvoir laisser l'utilisateur « manipuler » le réseau de neurones au travers une interface graphique l'autorisant à charger des fichiers CSV d'apprentissage, à prédire, un fichier contenant le modèle et de sauvegarder le modèle construit dans un fichier. Il peut également entraîner le modèle, une fois que celui-ci est construit, évaluer ses performances et lancer une prédiction à partir des données présentes dans le fichier à prédire.

Toutes ces actions sont résumées par le diagramme ci-dessous :

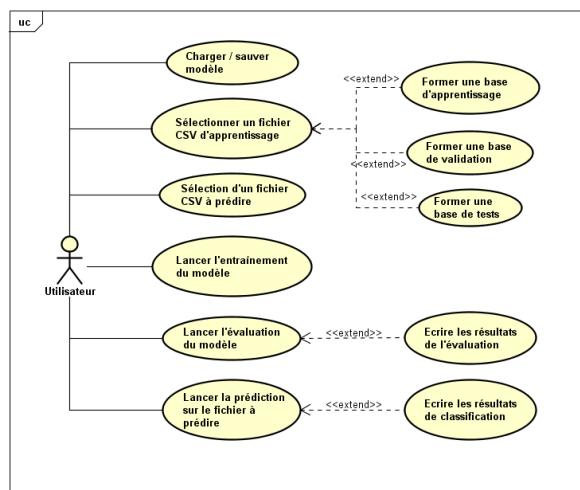


Figure 10: Diagramme des cas d'utilisation de la nouvelle application

Une ébauche de cette interface graphique a été faite, elle serait la suivante :

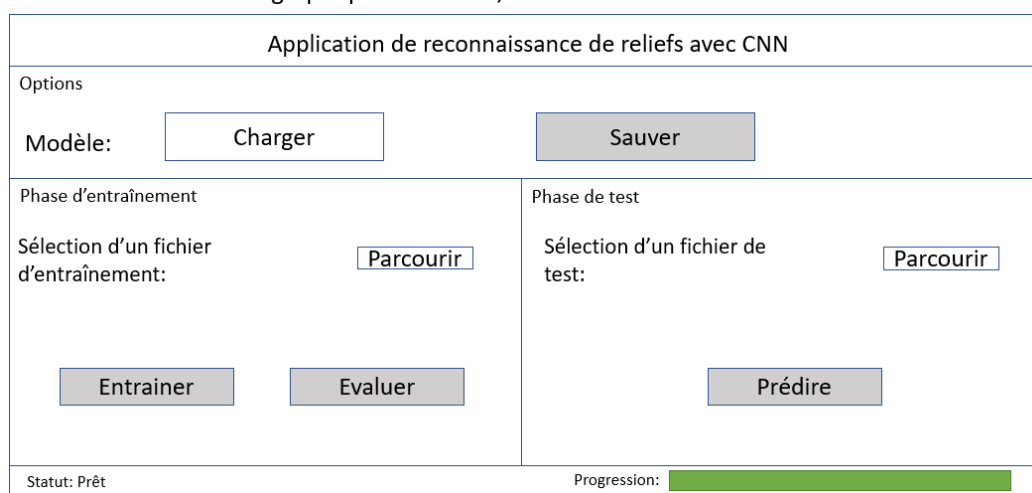


Figure 11: Interface graphique de la nouvelle application proposée

Certaines modifications pourront être faites, comme le positionnement de la section « Phase de test » en dessous de la section « Phase d'entraînement » pour faciliter la navigation de l'archéologue, ce qui permet de garder un ordre chronologique des actions à mener avant de pouvoir lancer une prédiction.

2.5. Conception de l'interface graphique de visualisation de l'application SkyEye

2.5.1. Description

Cette interface aura pour but d'afficher le résultat des décisions d'appartenance des pixels aux classes prédéfinies, ce qui permettra à la fois à l'utilisateur de pouvoir modifier manuellement la décision d'affectation d'un pixel à une classe et, même s'il a déjà été calculé précédemment, de générer un taux de performance pour la reconnaissance des reliefs dans l'image et valider définitivement le modèle construit.

Le diagramme des cas d'utilisation ci-dessous récapitule les actions que peut être amené à faire l'utilisateur.

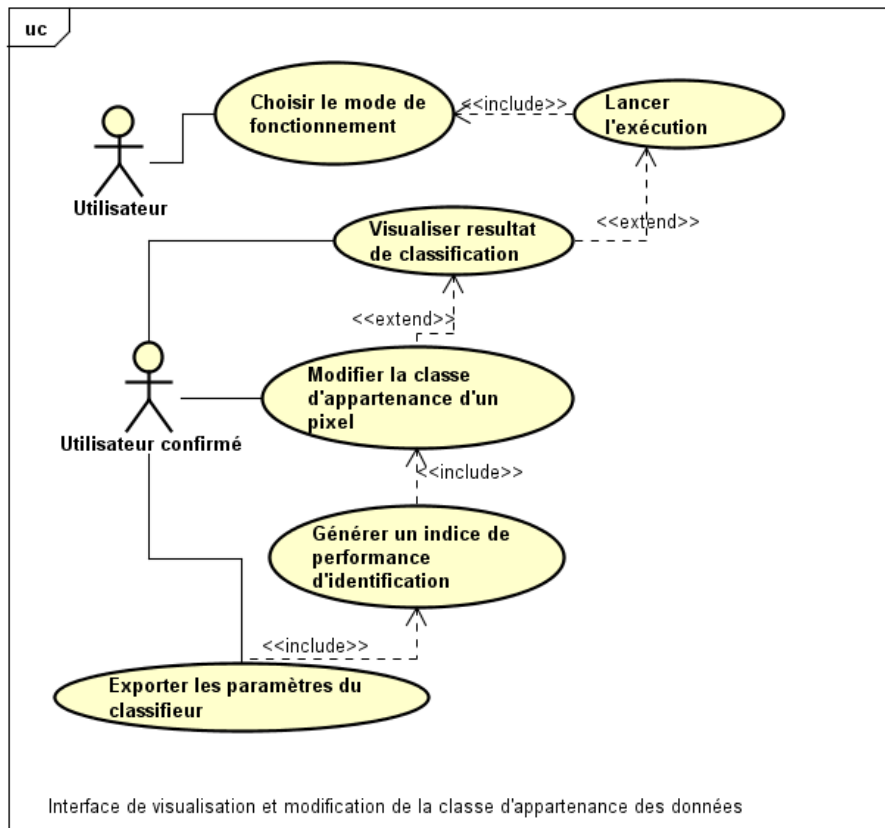


Figure 12 - Diagramme des cas d'utilisation de l'interface graphique

Il sera possible de modifier à nouveau le diagramme, pour prendre en compte les besoins supplémentaires venant de la MOA. Ce travail a été fait de façon prévisionnelle, c'est-à-dire, sous condition de validation par la MOA.

A partir de cette situation, une idée de maquette serait la suivante, sachant qu'une image résultante du traitement est générée automatiquement, l'interface pourrait ressembler à ça.

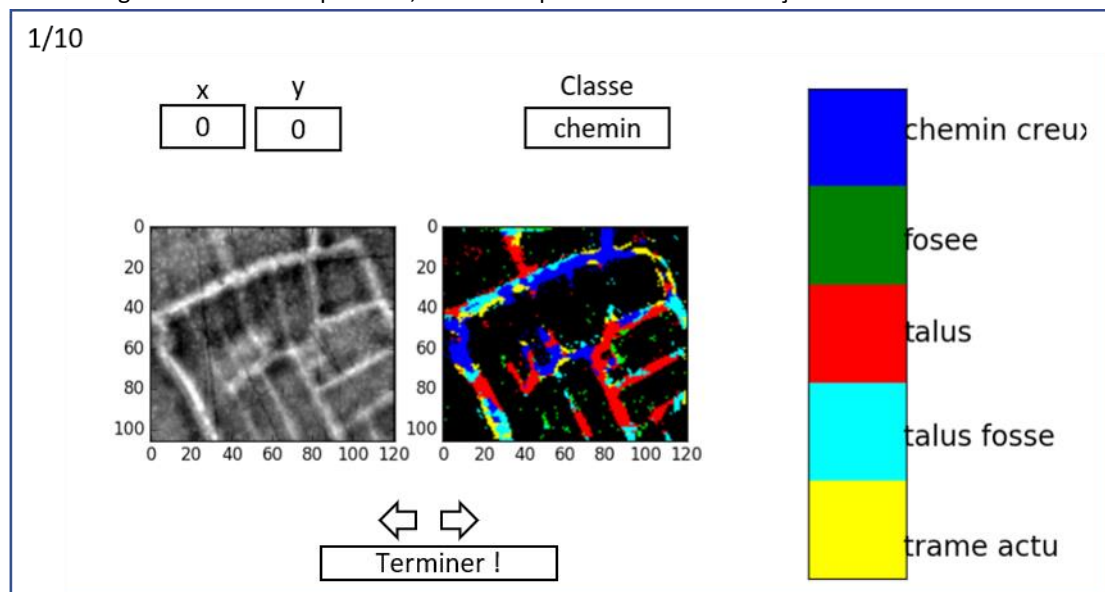


Figure 13 - Maquette d'interface proposée

Peu importe le mode de fonctionnement de l'application, que ce soit en apprentissage ou en mode prédiction, on récupère les fichiers images générés, portant le même nom que l'image passée en paramètre, au début. Le principe est d'afficher l'image originelle avec, à côté, l'image dont le système a prédit l'appartenance des pixels aux classes.

La résolution de l'image ne change pas et on a deux variables dont on peut modifier les valeurs : x et y. Ces deux variables représentent les coordonnées d'un pixel dans l'image, caractérisé par son appartenance à une classe. En modifiant les valeurs, on récupère la classe d'appartenance du pixel, par lecture dans le fichier CSV généré. Un compteur d'images traitées s'incrémente au fur et à mesure que les paires d'images sont affichées pour que l'utilisateur ait la possibilité de changer la décision prise par le système.

2.5.2. Comparaison avec l'existant

Cette interface graphique n'est pas implémentée mais une base comme celle-ci pourrait être utilisée pour la modification de la classe d'appartenance des pixels des images, de manière simple. On se baserait donc sur 3 fichiers : le fichier CSV résultant de la détermination d'appartenance de tous les pixels de l'image aux classes définies précédemment, l'image initiale au format **.tif** en nuances de gris et l'image finale après construction grâce au premier fichier évoqué.

Au final, aucun nouveau fichier n'est attendu en sortie, uniquement une mise à jour des couleurs des pixels et la reconstruction de l'image finale, après que toutes les images aient été modifiées ou non.

Encore une fois, cette partie doit être proposée et validée à la MOA, tout en prenant en compte leurs besoins en plus, pour pouvoir passer à la modélisation et l'implémentation. Il ne s'agit que d'une proposition de maquette.

2.5.3. Fonctionnalités liées

Pour simplifier ce traitement, la conception et le développement d'un parseur pour les fichiers CSV générés pourrait être utile ce qui permettra de garder en mémoire son contenu, tant que l'utilisateur n'a pas terminé les modifications à apporter.

3. Spécifications non fonctionnelles

3.1. Contraintes de développement et conception

Ce projet devra être codé en Python à partir de l'existant. Toutes les bibliothèques utilisées jusqu'à maintenant doivent impérativement être utilisées. Le choix de la bibliothèque de **Deep Learning** à implémenter, dépendra en grande partie des besoins supplémentaires exprimés par la MOA.

3.2. Contraintes de fonctionnement et d'exploitation

3.2.1. Performances

Le temps d'exécution n'est pas un enjeu important dans ce projet pour l'instant. Il devra être possible à l'avenir de l'utiliser sur des images pouvant aller jusqu'au Giga octet. Pour ce type d'image, le temps d'exécution ne devra pas dépasser la durée d'une nuit.

4. Autres tâches importantes

4.1. Choix de la bibliothèque Python de Deep Learning

Comme vu précédemment, il faut que la bibliothèque à utiliser soit suffisamment souple pour garantir une certaine transparence (l'utilisateur ne doit pas pouvoir déterminer à quel moment le système va faire appel à cette bibliothèque, pas de ralentissement).

4.2. Tests de validation de la bibliothèque

Il faut également que des tests soient effectuées sur cette bibliothèque, pour à la fois valider le comportement attendu et confirmer qu'elle est équivalente, voire supérieure au classifieur actuellement utilisé (SVM). Il s'agit d'une tâche à part entière. Les tests à poser dépendront grandement de la bibliothèque à tester, mais les principales idées sont les suivantes :

- **Vitesse d'exécution** : ne doit pas prendre plus de temps à décider sur l'appartenance des pixels aux classes que le classifieur actuel
- **Transparent** : l'utilisateur ne doit pas se rendre compte que l'algorithme est en cours d'exécution, en partie, due à une latence lors de son travail sur les images
- **Performances de traitement** : après avoir déterminé l'appartenance des pixels aux classes, il faut générer un nouvel indice de performance pour vérifier l'efficacité de cette méthode. Il ne faut donc pas que l'ancien indicateur soit supérieur au nouveau généré

4.3. Formation sur les réseaux de neurones

Au cours de ce projet, des connaissances sont requises pour l'utilisation et la mise en place du réseau de neurones qui sera utilisé pour améliorer les performances de l'application en termes de performances. C'est pourquoi des tâches de documentation et de mise en pratique sont prévues. Il s'agit d'une tâche dont le poids est de 10 jours / homme, qui se termine sur la rédaction d'un document de tests et de scénarios effectués pour valider le fonctionnement des bibliothèques étudiées.

4.4. Amélioration du dépôt GitLab

L'amélioration du dépôt GitLab est prévue pour faciliter l'accès à toute personne voulant mettre en place l'application. Il s'agit de créer une nouvelle branche qui contiendra uniquement toute la documentation technique du projet. Aujourd'hui, la documentation liée est générée à partir d'une ligne de commande à faire dans le dossier contenant le code généré par la bibliothèque **Sphinx** (génération de documentation) au format html.

5. Conclusion

Voir le rapport.

BIBLIOGRAPHIE

Guillaume, R. (2017). *Cahier de spécifications*. Tours.

MathWorks. (s.d.). *Neurones à convolution*. Récupéré sur MathWorks - Deep Learning:
<https://fr.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>

TABLES DES ILLUSTRATIONS

Figure 1: Logo du groupe RFAI du Laboratoire Informatique de Tours	3
Figure 2 - Logo du projet SOLiDAR et du laboratoire CITERES de Tours	3
Figure 3 - Représentation d'un relief en nuances de gris.....	4
Figure 4 - Diagramme des cas d'utilisation actuelle.....	7
Figure 5 - Images binaires représentant les classes et l'image source.....	Erreur ! Signet non défini.
Figure 6 - Diagramme de classes de l'application	9
Figure 7 - Fonctionnement d'un réseau de neurones convolutif.....	Erreur ! Signet non défini.
Figure 8 - Diagramme des cas d'utilisation de l'interface graphique	14
Figure 9 - Maquette d'interface proposée	14

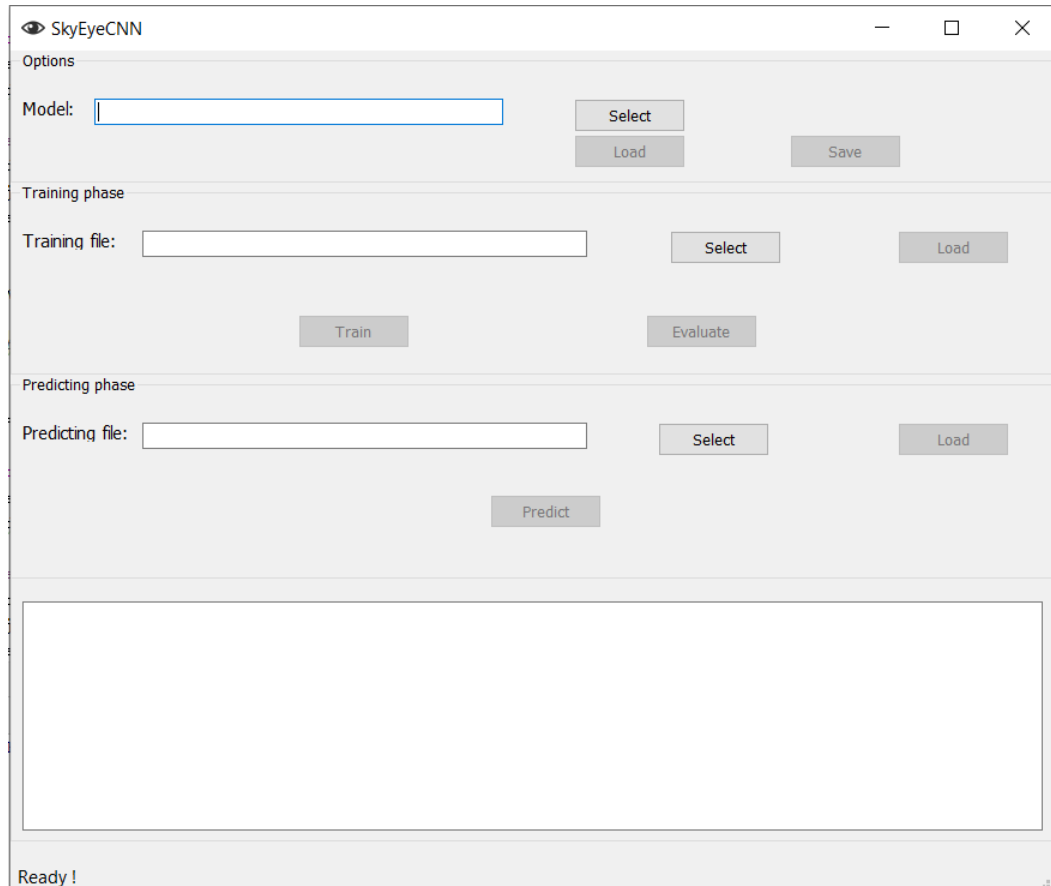
C

Guide utilisateur

Guide utilisateur

Lancement de l'application

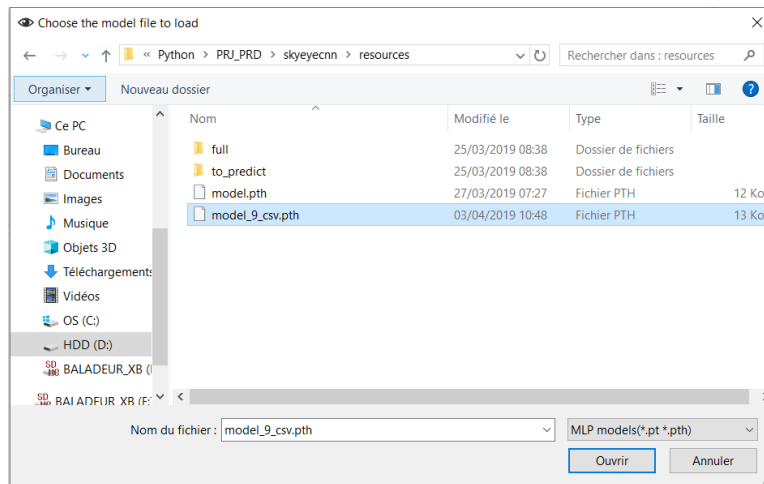
Pour lancer l'application, sélectionner et lancer le fichier « ClickMeToRun.bat ». L'application s'ouvre.



Chargement des fichiers

Chargement du modèle

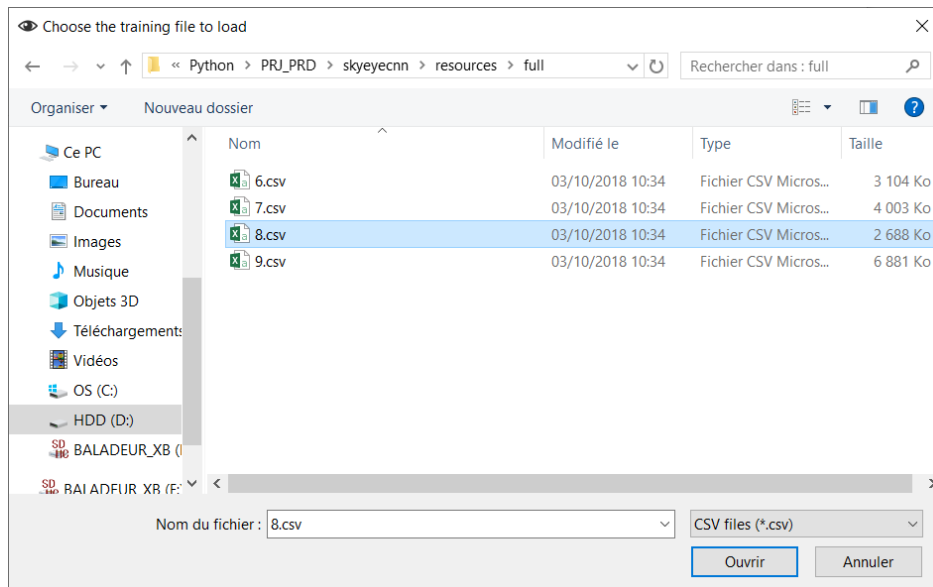
Cliquer sur le bouton « Select » dans la section Options permet de sélectionner le fichier modèle à charger, dont l'extension est « .pth ».



Cliquer sur le bouton « Load » permet de charger le fichier modèle sélectionné, un message s’affiche dans la barre de statut pour notifier du chargement.

Chargement du fichier d’apprentissage

Cliquer sur le bouton « Select » dans la section « Training phase » permet de sélectionner un fichier d’apprentissage. Le bouton à côté « Load » permet de charger le contenu de ce fichier et d’afficher son contenu dans la vue principale. Il s’agit de fichiers CSV contenant les caractéristiques des pixels et un label associé à chaque pixel de l’image.



A la fin du chargement, un message de ce type s’affiche pour récapituler les informations du fichier d’apprentissage chargé.

```
Training file load in progress ...
The 8.csv training file contains 41 features (columns) for 13135 rows
Training file loaded
```

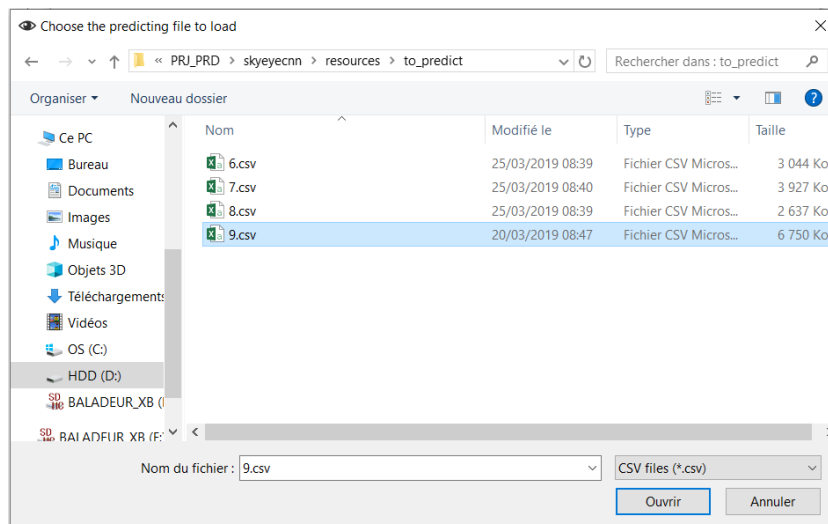
Chargement du fichier à prédire

Cliquer sur le bouton « Select » dans la section « Predicting phase » permet de sélectionner un fichier à prédire. Le bouton à côté « Load » permet de charger le contenu de ce fichier et d'afficher son contenu dans la vue principale.

Il s'agit de fichiers CSV contenant les caractéristiques des pixels et aucun label n'est associé à chaque pixel. Exemple de contenu du fichier à prédire :

Class	x	y	AvgD_15x15	AvgD_19x19	AvgD_3x3
		0	174	30.85	21.21
		0	224	18.44	18.3
		0	440	23.54	16.2
		0	487	38.92	52.97

Pour sélectionner le fichier à prédire, une fenêtre de ce type s'ouvre :



Une fois que le fichier CSV à prédire a été chargé, un message de ce type s'affiche :

Predicting file load in progress ...

The 9.csv predicting file contains 41 features (columns) for 33430 rows

Predicting file loaded

Attention : Le nombre de colonnes du fichier d'apprentissage et du fichier à prédire doivent être identiques (même nombre de caractéristiques).

Construction du réseau de neurones

Si un fichier modèle n'est pas sélectionné et chargé

Un réseau de neurones sera construit à partir du fichier d'apprentissage sélectionné : un algorithme va chercher la meilleure configuration de réseau de neurones pour avoir un taux de reconnaissance optimal. Ce traitement est fait par l'appui sur le bouton « Train » dans la section « Training phase ».

Au lancement de la recherche, le message suivant s'affiche :

```
Model training in progress ...  
Searching the best configuration parameters for the CNN
```

A la fin de la recherche, on construit le réseau de neurones à partir des paramètres déterminés et il est entraîné par la suite. Le temps d'exécution de cet algorithme dépend de la taille du fichier d'apprentissage : environ 20 minutes pour un fichier d'apprentissage contenant 15.156 pixels, par exemple.

```
The best precision is 0.8697229551451188 for 17 neurones in the hidden layer and 1658 epoches in  
1239.8197267055511 seconds  
Model train finished
```

Quand l'algorithme a fini, les meilleurs paramètres sont affichés avec le taux de reconnaissance lié et le modèle construit à partir de ces paramètres sont est aussi entraîné.

Si un fichier modèle est sélectionné et chargé

Le modèle est chargé à partir du fichier sélectionné, il est possible de lui faire apprendre à nouveau (le modèle chargé est déjà entraîné) en sélectionnant et chargeant un fichier d'apprentissage.

Entraînement du modèle

Il faut charger un fichier d'apprentissage puis cliquer sur le bouton « Train » pour faire apprendre le modèle construit.

Evaluation du modèle construit

Il faut qu'un fichier d'apprentissage soit chargé, ainsi, une partie des données de ce fichier seront utilisées pour évaluer le modèle. L'évaluation est lancée en cliquant sur le bouton « Evaluate », le message ci-dessous s'affiche, contenant la précision du modèle pour ces données et la matrice de confusion, répertoriant l'ensemble des classifications faites :

```
Model evaluation in progress ...  
Model precision: 0.8845360824742268  
Confusion matrix: [[ 610  154]  
[ 126 1535]]  
The model has been evaluated
```

Prédiction

Une fois qu'un fichier à prédire a été chargé, il faut appuyer sur le bouton « Predict » pour lancer la prédiction et le message suivant s'affiche :

Labels of pixels in the predicting file are beeing predicted
Prediction results file has been written
Probabilities file has been created
Pixels labels predicted

Deux fichiers CSV résultants de la prédiction ont été écrits : un correspondant au fichier CSV à prédire où chaque pixel est labellisé (prediction_results_<nom_fichier>.csv) et un autre fichier CSV contenant les coordonnées de chaque pixel avec pour chaque classe connue, la probabilité d'appartenance à cette classe (probabilities_results_<nom_fichier>.csv).

Ces fichiers sont écrits dans le dossier « results » à la racine du package.

D

Guide développeur

Guide développeur

Introduction

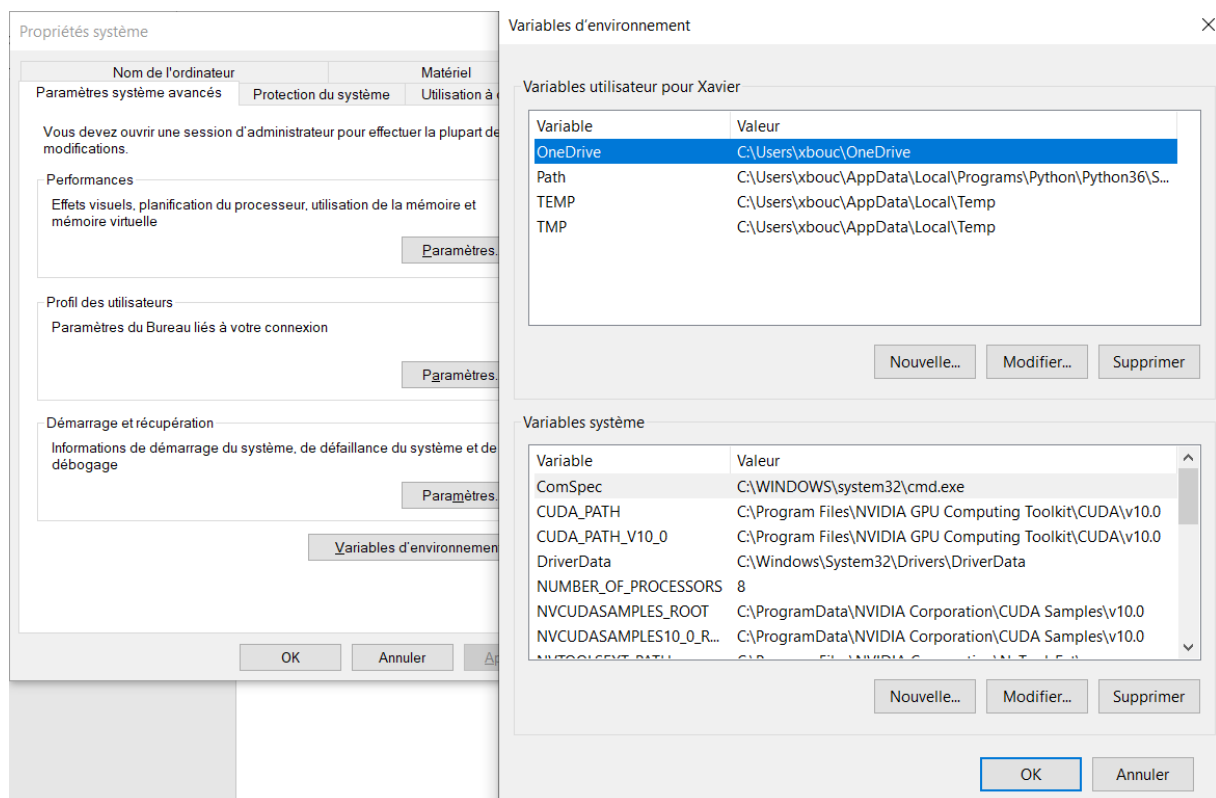
Ce document a pour but d'apporter des conseils supplémentaires à la documentation du code Doxygen générée des sources Python de l'application. Cela permet d'éviter l'installation de modules Python existants déjà dans l'environnement d'exécution, par exemple.

Développement

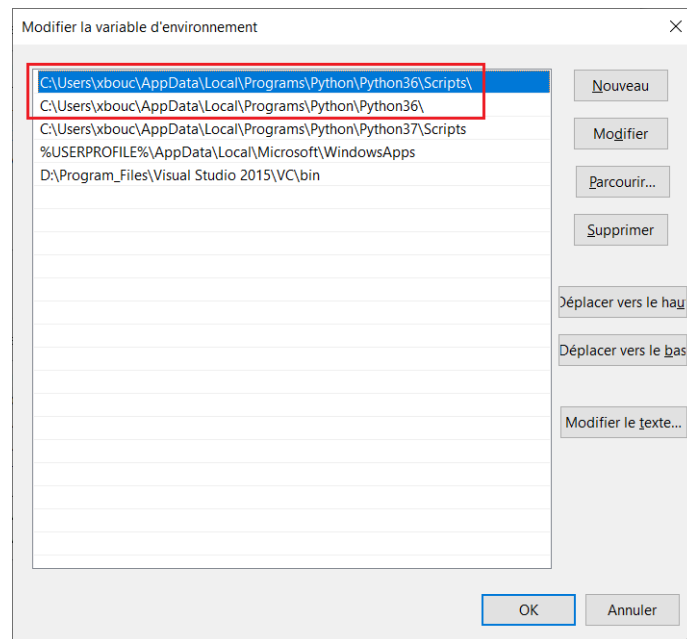
Pour le développement, pas besoin d'installer à nouveau les modules nécessaires pour faire tourner l'application. L'interpréteur Python présent dans le package « skyeyecnn_portable » contient déjà tous les modules nécessaires, il faut dans ce cas, copier ce dossier dans un répertoire et l'utiliser comme interpréteur de référence.

Pour cela, il faut aller dans les variables d'environnement, pour Python, deux répertoires sont mentionnés :

- le répertoire contenant le fichier « python.exe » (environnement Python)
- le répertoire contenant tous les scripts exécutables pouvant être appelés :
 - ex : pyuic5 est un utilitaire Python permettant de convertir un fichier graphique .ui Qt en script Python du module PyQt5



Il faut d'abord cliquer sur la variable « Path » utilisateur pour pouvoir y accéder. Dans le cas où une version 3.X d'interpréteur Python est installée, il est conseillé de remplacer ces chemins d'accès par ceux où le dossier nommé « Python » contenant l'interpréteur Python et tout l'environnement est situé.



Une fois que c'est fait, vous pouvez vérifier la version de l'interpréteur en lançant un invité de commande et en tapant la commande **python** en obtenant ce résultat :

```
Invite de commandes
Microsoft Windows [version 10.0.17763.379]
(c) 2018 Microsoft Corporation. Tous droits réservés.

C:\Users\xbouc>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\xbouc>
```

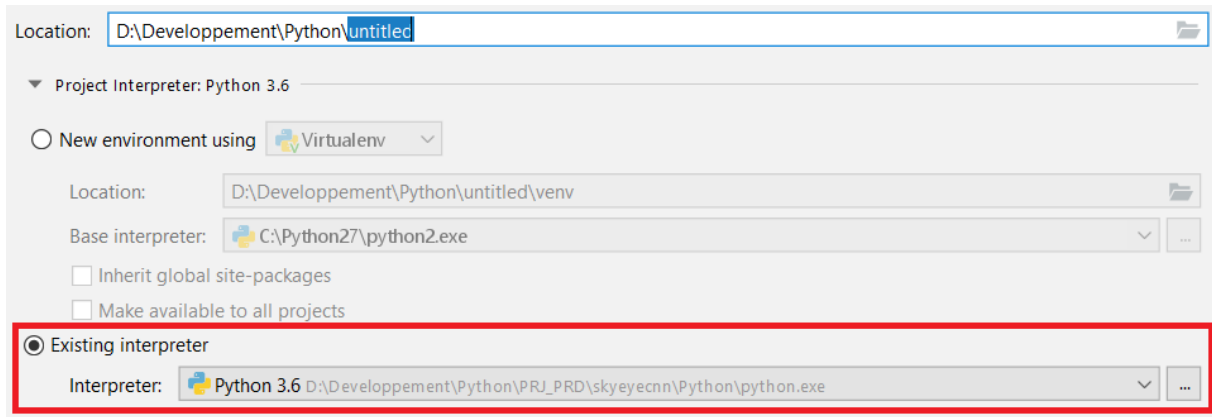
Pour le vérifier, vous pouvez également taper la commande suivante **python -m pip -V**, qui vous donnera la version du module pip utilisée par l'interpréteur dont le chemin d'accès est affiché avec sa version.

Si la version est identique à la capture précédente, il faut maintenant configurer au nouveau projet que vous allez créer dans votre IDE à partir des sources existantes, pour pouvoir prendre en compte l'interpréteur voulu.

Pour PyCharm, la démarche à suivre est la suivante :

- Créer un nouveau projet en indiquant comme répertoire de sauvegarde, le dossier contenant les sources

- Dérouler l'option notée « Project Interpreter » sur la capture d'écran ci-dessous, choisir l'option « Existing Interpreter », sélectionner le répertoire dans lequel le fichier exécutable est stocké comme sur la capture ci-dessous



L'IDE va charger tous les modules installés dans cet environnement pour ensuite pouvoir les utiliser et lancer l'application. Pour l'installation de nouveaux paquets, il est conseillé de le faire directement depuis un terminal avec la commande suivante **pip install -U <nom_package>**.

L'option -U permet de mettre à jour à la dernière version le module à installer, s'il était déjà installé ou d'installer la toute dernière version sinon.

Il est également possible de voir tous les modules installés en tapant la commande suivante :

```
C:\Users\xbouc>python -m pip freeze
numpy==1.16.2
pandas==0.24.2
Pillow==5.4.1
PyQt5==5.12
PyQt5-sip==4.19.14
python-dateutil==2.8.0
pytz==2018.9
scikit-learn==0.20.3
scipy==1.2.1
six==1.12.0
torch==1.0.1
torchvision==0.2.2.post3
C:\Users\xbouc>
```

⚠ La version de cet interpréteur est différente de celle de la première application « skyece » car il était impossible d'installer les paquets **torch** et **torchvision** pour la construction et l'utilisation d'un réseau de neurones. Seuls les interpréteurs dont la version est compatible avec les architectures 64 bits peuvent installer ces paquets (comme l'interpréteur actuel). On peut le vérifier sur la troisième capture de ce document, qui correspond à **[MSC v.1900 64 bit (AMD64)]**.

Génération de documentation

Pour la documentation des sources, le format utilisé est docstring, c'est-à-dire en utilisant des caractères # ou des guillemets. Les sources suivantes ont été utilisées pour cela :

- <http://www.doxygen.nl/manual/docblocks.html#pythonblocks>
- <https://axiomcafe.fr/tutoriel-documenter-un-code-avec-doxygen>

Doxygen a ensuite été utilisé pour générer la documentation du code en fichier HTML pour y accéder facilement. La documentation de ce code se situe dans le dossier **doxygen/html** et lancer le fichier **index.html**. Un fichier de configuration doxygen a également été sauvegardé, il est possible de le réutiliser après avoir installé doxygen et changer le répertoire de sauvegarde de la nouvelle documentation à générer.

Aspect général

L'ensemble de toutes les sources utilisées pour l'application se trouve dans le répertoire **program**. Voici un bref récapitulatif de l'ensemble du dépôt GitLab :

- **Best_parameters_CNN_algorithm** : contient les scripts Python pour déterminer les meilleurs paramètres pour la construction du réseau de neurones en fonction du nombre de couches cachées (entre 1 et 30), du taux d'apprentissage (0.01 ou 0.02) et du nb de passage des données d'apprentissage dans la boucle (epoches : de 800 à 2000). Une étude de performances a été faite basée sur ce travail (trouver la meilleure fonction d'activation)
- **Logs** : doit contenir des fichiers txt correspondant aux logs de l'application. Cette partie de l'application n'a pas pu être développée en raison d'un gros manque de temps (ajout d'une classe dans le modèle dont les méthodes sont statiques pour être appelées partout).
- **Doxygen** : contient la documentation liée au code des scripts sources de l'application
- **Resources** : contient des fichiers CSV de tests dont certains contiennent des labels et d'autres à prédire
- **Results** : contient tous les fichiers CSV résultants d'une prédiction. Chaque fichier est identifiable par le nom du fichier prédit
- **Skyeyecnn_portable** : package contenant l'interpréteur Python, les sources et un guide d'utilisation

Webographie

- [WWW1] Chloé-Agathe AZENCOTT.
- [WWW2] Chloé-Agathe AZENCOTT. *Empilez les perceptrons*. URL : <https://openclassrooms.com/fr/courses/4470406-utilisez-des-modeles-supervises-non-lineaires/4732186-empilez-les-perceptrons>.
- [WWW3] Chloé-Agathe AZENCOTT. *Entraînez un réseau de neurones simple*. URL : <https://openclassrooms.com/fr/courses/4470406-utilisez-des-modeles-supervises-non-lineaires/4730716-entraenez-un-reseau-de-neurones-simple>.
- [WWW4] *Classifieur linéaire*. URL : https://fr.wikipedia.org/wiki/Classifieur_lin%C3%A9aire.
- [WWW5] Charles CROUSPEYRE. *Comment les Réseaux de neurones à convolution fonctionnent*. 2017. URL : <https://medium.com/@CharlesCrouspeyre/comment-les-r%C3%A9seaux-de-neurones-%C3%A0-convolution-fonctionnent-b288519dbcf8>.
- [WWW6] Solene LIMOUSIN. *L'intelligence artificielle*. 2018. URL : <https://www.supinfo.com/articles/single/7084-intelligence-artificielle>.
- [WWW7] *Machine à vecteurs de support*. 2018. URL : https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support.
- [WWW8] *Neurones à Convolution - 3 choses à savoir*. URL : <https://fr.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>.
- [WWW9] *Reconnaissance de formes*. 2018. URL : https://fr.wikipedia.org/wiki/Reconnaissance_de_formes.
- [WWW10] WIKIPEDIA. *Apprentissage profond*. URL : https://fr.wikipedia.org/wiki/Apprentissage_profond.



Bibliographie

- [1] Ronan GUILLAUME. « Plateforme d'extraction et de caractérisation automatique d'éléments d'intérêt dans les images archéologiques par apprentissage interactif ». Projet Recherche & Développement. Tours, France : Ecole Polytechnique de l'Université François Rabelais de Tours, 2017-2018.

Implémentation d'une méthode de Deep Learning dans une application de reconnaissance de relief dans des images dédiée à l'archéologie (SkyEye)

Xavier Bouchenard

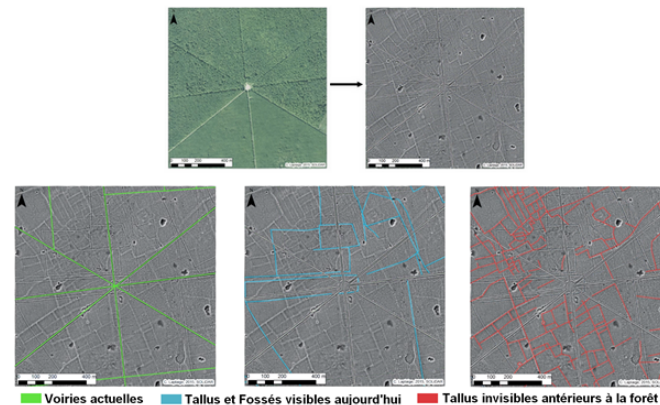
Encadrement : Jean-Yves Ramel et Thierry Brouard



En collaboration avec Laboratoire
Archeologie et Territoires

Contexte

Travail sur un projet en collaboration avec les archéologues du programme SoLiDar. Mise en place d'une méthode de reconnaissance automatique de reliefs dans des images LiDar

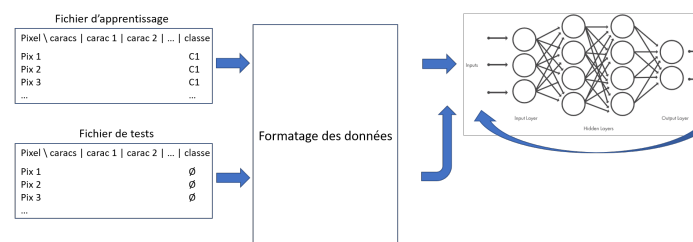


Reconnaissance des reliefs dans des images LiDar

Améliorations à apporter

Fonctionnalités prévues:

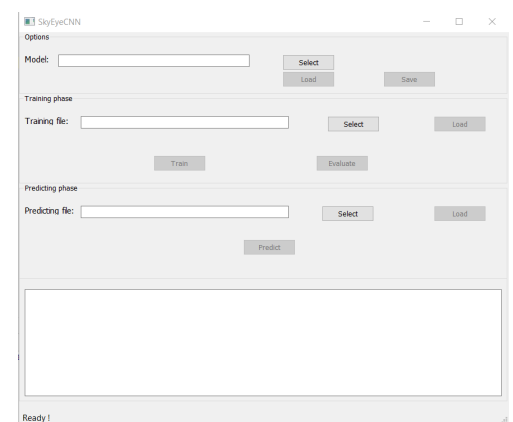
- Implémentation d'une méthode de Deep Learning
- Implémentation d'une interface graphique de modification de prédiction
- Clarification du dépôt de **SkyEye**



Structure de l'application contenant le réseau de neuro

Objectifs

- Implémenter d'un réseau de neurones dans l'application **SkyEye** pour améliorer le taux de reconnaissance
- Implémenter un algorithme de détermination de paramètres optimaux pour réseau de neurones



Interface graphique de l'application

Implémentation d'une méthode de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiées à l'archéologie (SkyEye)

Xavier Bouchenard

Encadrement : Jean-Yves Ramel et Thierry Brouard

Contexte

Travail sur un projet en collaboration avec les archéologues du programme SoLiDar. Mise en place d'une méthode de reconnaissance automatique de reliefs dans des images LiDar

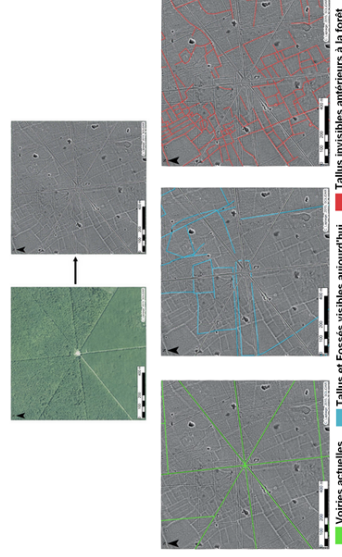
Améliorations à apporter

Fonctionnalités prévues:

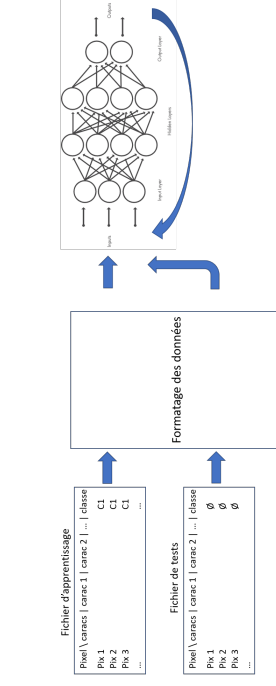
- Implémentation d'une méthode de Deep Learning
- Implémentation d'une interface graphique de modification de prédiction
- Clarification du dépôt de **SkyEye**

Objectifs

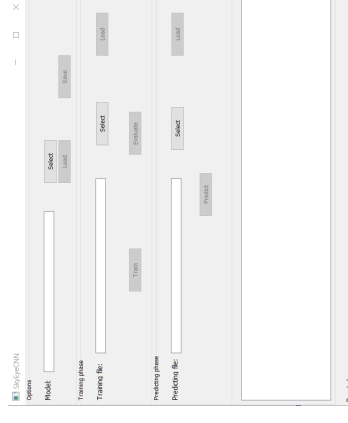
- Implémenter d'un réseau de neurones dans l'application **SkyEye** pour améliorer le taux de reconnaissance
- Implémenter un algorithme de détermination de paramètres optimaux pour réseau de neurones



Reconnaissance des reliefs dans des images LiDar



Structure de l'application contenant le réseau de neurones



Interface graphique de l'application

Implémentation d'une méthode de Deep Learning dans une application de reconnaissance de reliefs dans des images dédiée à l'archéologie (SkyEye)

Résumé

Ce rapport décrit la phase de recherche du projet recherche & développement. L'objectif du projet est de détecter les vestiges archéologiques à partir d'images dérivées de données LiDAR. En phase de recherche, ce rapport décrit une méthode permettant d'améliorer les performances de la reconnaissance de pixels dans les images. En phase de développement, il faudra améliorer l'existant en lui ajoutant une méthode de Deep Learning et une interface graphique pour la visualisation et modifications des décisions prises, pour rendre l'application plus intuitive.

Mots-clés

Reconnaissance de Formes; Deep Learning; intuitif; bibliothèque

Abstract

This report describes the research phase of the Research & Development project. The goal of the project is to detect archaeological remains from images derived from LiDAR data. In the research phase, this report describes a method to improve the performance of pixel recognition in images. In the development phase, it will improve the existing by adding a Deep Learning method and a graphical interface for viewing and changes decisions made to make the application more intuitive.

Keywords

Pattern Recognition; Deep Learning; intuitive; library; CNN

Entreprise

Laboratoire Archeologie et Territoires



Tuteurs entreprise

Clement LAPLAIGE

Xavier RODIER

Étudiant

Xavier BOUCHENARD (DI5)

Tuteurs académiques

Jean-Yves RAMEL

Thierry BROUARD