

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](http://polytech.univ-tours.fr)

## Projet Recherche & Développement 2017-2018

# Estimation de la densité de poissons dans les rivières par analyse d'images aériennes

**POLYTECH<sup>®</sup>**  
TOURS

**Tuteurs académiques**

Donatello CONTE

Pierre GAUCHER

**Étudiant**

Mikael MOREAU (DI5)



# Liste des intervenants

Nom	Email	Qualité
Mikael MOREAU	<a href="mailto:mikael.moreau@etu.univ-tours.fr">mikael.moreau@etu.univ-tours.fr</a>	Étudiant DI5
Donatello CONTE	<a href="mailto:donatello.conte@univ-tours.fr">donatello.conte@univ-tours.fr</a>	Tuteur académique, Département Informatique
Pierre GAUCHER	<a href="mailto:pierre.gaucher@univ-tours.fr">pierre.gaucher@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Mikael Moreau susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Donatello Conte et Pierre Gaucher 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

Mikael Moreau, *Estimation de la densité de poissons dans les rivières par analyse d'images aériennes*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={Moreau, Mikael},
  title={Estimation de la densité de poissons dans les rivières par analyse d'images aé-
    riennes},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```



# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	viii
<b>I Cahier de spécification</b>	<b>1</b>
<b>Introduction</b>	<b>2</b>
<b>1 Contexte de la réalisation</b>	<b>3</b>
1 Acteurs du projet .....	3
2 Objectifs .....	4
3 Hypothèses .....	4
4 Bases méthodologiques .....	4
<b>2 Description générale</b>	<b>5</b>
1 Environnement du projet .....	5
2 Caractéristiques des utilisateurs .....	5
3 Fonctionnalités et Structure générale du système .....	5
3.1 Le système de reconnaissance d'image .....	7
3.2 Intégration de l'algorithme avec l'IHM .....	8

<b>II</b>	<b>État de l'art</b>	<b>10</b>
	<b>Introduction</b>	<b>11</b>
<b>1</b>	<b>La reconnaissance de formes et le machine learning</b>	<b>12</b>
1	Le machine learning .....	12
2	Schéma général d'un système de reconnaissance de formes .....	13
2.1	La saisie .....	13
2.2	Les prétraitements .....	13
2.3	L'extraction de caractéristiques .....	13
2.4	La décision .....	14
2.5	L'apprentissage .....	14
3	Le Perceptron .....	15
3.1	La règle d'apprentissage du Perceptron et la descente de gradient.....	15
3.2	La règle de décision du Perceptron .....	16
4	SVM (Support Vector Machines .....	17
5	La minimisation aux moindres carrés.....	17
6	Réseaux de neurones .....	17
6.1	Réseaux à un seul neurone .....	17
6.2	Réseaux de neurones multicouches.....	18
6.3	Apprentissage par rétropropagation.....	19
6.4	Option d'apprentissage d'un réseau de neurone .....	20
6.4.1	Initialisation des poids .....	20
6.4.2	Apprentissage séquentiel ou par cycle (en batch) .....	21
6.4.3	Ordre de présentation des exemples.....	21
6.4.4	Fonction d'activation .....	21
6.4.5	Nombre de noeuds cachés .....	22
6.4.6	Pas d'apprentissage.....	22
6.4.7	Noeuds de type biais .....	22
6.4.8	Elagage .....	22
6.4.9	Vecteur de sortie .....	23
6.4.10	Normalisation des données .....	23
6.4.11	La fonction de coût .....	23
6.4.12	L'algorithme de mise à jour des poids.....	23
6.4.13	Critère d'arrêt .....	23
7	L'apprentissage profond (deep learning) .....	23
7.1	Les réseaux convolutifs.....	24
7.1.1	Dropout Layer :.....	25
7.1.2	Fully connected Layer :.....	25
7.1.3	Convolution Layer : .....	25

7.1.4	Pooling Layer : .....	27
7.1.5	Batch Normalization Layer : .....	28
7.2	AlexNet .....	29
7.3	VGG (Visual Geometry Group) .....	30
7.4	ResNet .....	31
7.5	Data augmentation .....	32
7.6	Transfert learning .....	32
8	Régression .....	32
8.1	La régression linéaire .....	33
8.2	La régression non linéaire .....	34
<b>2</b>	<b>Apprendre à compter des objets dans une image</b>	<b>35</b>
1	Compter par détection .....	35
2	Compter par régression .....	36
3	Compter par segmentation .....	36
<b>3</b>	<b>Towards perspective-free object counting with deep learning</b>	<b>37</b>
<b>4</b>	<b>CrowdNet : A Deep Convolutional Network for Dense Crowd Counting</b>	<b>39</b>
1	Le réseau profond .....	39
2	Le réseau peu profond .....	40
3	L'apprentissage .....	40
<b>5</b>	<b>Using Deep Learning for Segmentation and Counting within Microscopy Data</b>	<b>42</b>
1	La segmentation des cellules .....	42
2	Estimation du nombre de cellules dans une image .....	43
<b>III</b>	<b>Analyse et conception</b>	<b>44</b>
	<b>Introduction</b>	<b>45</b>
<b>1</b>	<b>Architecture du système de reconnaissance d'image</b>	<b>46</b>
1	Data augmentation .....	46
2	Prétraitements .....	46
3	transfert learning .....	46
4	Réseau de neurone .....	47
5	Apprentissage séquentiel ou par cycle .....	47
6	Ordre de présentation des exemples .....	47
7	Fonction d'activation / couche de non linéarité .....	48
8	L'algorithme de mise à jour des poids .....	48
9	Les hyperparamètres .....	48

10	La fonction de coût .....	48
11	Elagage .....	49
12	Vecteur de sortie .....	49
13	Post-traitements .....	49
14	Prétraitement sur les images d'apprentissage .....	49
<b>2</b>	<b>Structure du code</b>	<b>50</b>
<b>IV</b>	<b>Mise en œuvre</b>	<b>51</b>
<b>1</b>	<b>Implémentation</b>	<b>52</b>
1	Les outils .....	52
2	Le système de reconnaissance de formes .....	52
2.1	Le réseau de neurones .....	52
2.2	Les prétraitements et post-traitements .....	53
2.3	Entraînement du réseau de neurones .....	54
2.4	Transfer learning .....	56
3	Diagramme de classe .....	56
3.1	Le package ui .....	56
3.2	Le package ccnn .....	57
3.3	Dépendances entre les deux packages .....	58
4	Structure des sources .....	60
<b>2</b>	<b>Le dataset</b>	<b>63</b>
<b>3</b>	<b>Résultats</b>	<b>65</b>
1	Test sur le dataset transcos .....	65
2	Entraînement sur les images du dataset .....	69
2.1	Configurations .....	69
2.2	Méthode d'évaluation des résultats .....	69
2.3	Résultats et interprétation .....	69
<b>4</b>	<b>Diagramme de Gantt du PRD2</b>	<b>79</b>
	<b>Conclusion</b>	<b>81</b>
	<b>Annexes</b>	<b>82</b>
<b>A</b>	<b>Description des interfaces externes du logiciel</b>	<b>83</b>
1	Interfaces matériel/logiciel .....	83

2	Interfaces homme/machine .....	83
3	Interfaces logiciel/logiciel .....	84
<b>B</b>	<b>Spécifications fonctionnelles</b> .....	<b>85</b>
1	Mode apprentissage.....	85
1.1	Format des entrées.....	85
1.2	Les sorties.....	85
2	Mode prédiction .....	85
2.1	Les entrées.....	85
2.2	Les sorties.....	86
3	Intégration à l'IHM.....	86
<b>C</b>	<b>Spécifications non fonctionnelles</b> .....	<b>87</b>
1	Contraintes de développement et conception .....	87
1.1	Matériel.....	87
1.2	Langage .....	87
1.3	Environnement .....	87
1.4	Bibliothèques .....	88
2	Contraintes de fonctionnement et d'exploitation.....	88
2.1	Performances.....	88
2.2	Capacités.....	88
3	Maintenance et évolution du système.....	88
<b>D</b>	<b>Plan de développement</b> .....	<b>89</b>
1	Découpage en tâches .....	89
1.1	Tâche 1 : Rédaction du cahier de spécification .....	89
1.2	Tâche 2 : Documentation générale sur la reconnaissance des formes et l'analyse d'image.....	89
1.3	Tâche 3 : Recherche détaillée d'algorithmes ou d'articles scientifique adaptés au problème étudié .....	89
1.4	Tâche 4 : Installation et prise en main des outils de développement.....	90
1.5	Tâche 5 : Tests expérimentaux d'algorithmes implémenté en Keras.....	90
1.6	Tâche 6 : Rédaction de l'état de l'art.....	90
1.7	Tâche 7 : Rédaction de l'analyse et conception .....	90
1.8	Tâche 8 : Conception de la structure du réseau de neurones .....	90
1.9	Tâche 9 : Exploration avancée de la bibliothèque Keras.....	91
1.10	Tâche 10 : Implémentation des prétraitements et du modèle du réseau de neurones.....	91
1.11	Tâche 11 : Implémentation du mode apprentissage .....	91
1.12	Tâche 12 : Implémentation du mode prédiction.....	91
1.13	Tâche 13 : Entraînement de l'algorithme .....	91

1.14	Tâche 14 : Test de performance de l'algorithme et analyse des résultats obtenu .....	91
1.15	Tâche 15 : Intégration de l'algorithme à l'IHM.....	92
1.16	Tâche 16 : Rédaction du rapport final.....	92
2	Diagramme de Gantt .....	92
<b>E</b>	<b>Manuel d'installation</b>	<b>94</b>
1	Anaconda .....	94
2	Subversion .....	94
3	Récupération des sources .....	94
4	Installation des dépendances .....	95
<b>F</b>	<b>Manuel d'utilisation</b>	<b>96</b>
1	Mode prédiction .....	96
1.1	Configuration.....	96
1.2	L'interface .....	97
2	Mode apprentissage.....	98
2.1	Configuration.....	98
2.2	Utilisation des scripts .....	99
2.2.1	Générer le fichier d'entraînement pour le réseau de neurones .....	99
2.2.2	Entraîner le réseau de neurones .....	99
2.2.3	Tester l'algorithme .....	100■
2.2.4	Automatiser la génération du fichier d'apprentissage, l'entraînement et le test .....	100■
<b>G</b>	<b>Dossier de tests</b>	<b>101</b>
1	Les tests fonctionnels .....	101■
2	Test depuis l'interface .....	103■
<b>H</b>	<b>Cahier de développeur</b>	<b>104</b>
1	Convention de nommage.....	104■
1.1	Variables.....	104■
1.2	Fonctions.....	104■
1.3	Classes.....	104■
1.4	Commentaires.....	105■
2	Structure des sources.....	105■
3	Le diagramme de classe.....	105■
4	La documentation.....	105■
	<b>Comptes rendus hebdomadaires</b>	<b>106</b>
	<b>Webographie</b>	<b>109</b>

Bibliographie
---------------

111
-----

# Table des figures

## 1 Contexte de la réalisation

1	Logo du Laboratoire Informatique de Tours.....	3
---	--	---

## 2 Description générale

1	Diagramme d'utilisation du système .....	6
2	Diagramme de classe du système de reconnaissance de formes .....	7
3	Diagramme de classe du système d'apprentissage .....	7
4	Diagramme de classe du système.....	8

## 1 La reconnaissance de formes et le machine learning

1	Schéma représentant les différents algorithmes de machine learning [WWW3] .....	12
2	Schéma général d'un système de reconnaissance de formes [10].....	13
3	Différentes entrées représentées dans l'espace des caractéristiques [10].....	14
4	Hyperplan séparant deux classes.....	15
5	Représentation connexionniste du Perceptron [WWW4].....	16
6	Les SVM introduisent la notion de marge entre l'hyperplan et les classes [WWW5]	17
7	Schéma d'un réseau multicouche avec une couche d'entrée, une couche cachée et une couche de sortie [WWW6].....	18
8	Un perceptron dont la taille du vecteur d'entrée est de 2 représenté comme un réseau de neurones [WWW7].....	19
9	La méthode de descente de gradient stochastique consiste à trouver un minimum dans un espace multidimensionnel. Ici sont représenté deux configurations différentes de l'algorithme SGD. Celle de gauche trouve directement le minimum global tandis que l'autre tombe dans plusieurs minimums locaux avant de trouver la solution optimale [WWW9] .....	20



10	Les courbes des fonctions d'activation <i>ReLU</i> , <i>Tanh</i> et <i>Sigmoid</i> [WWW11].....	21
11	Les fonctions d'activation <i>Step</i> et <i>Sign</i> [WWW12].....	22
12	Les différents niveaux de caractéristiques dans un système de reconnaissance de formes profond [8] [WWW23].....	24
13	Le Dropout consiste à désactiver des neurones durant l'apprentissage [WWW14] .	25
14	Une couche entièrement connectée, (fully connected layer) [WWW15].....	25
15	Opération de convolution sur une image [WWW15].....	26
16	Résultat d'une opération de convolution sur une image avec plusieurs filtres [WWW15].....	26
17	Résultat d'un Max Pool sur une matrice avec un filtre de taille 2x2 et de stride 2 [WWW16].....	27
18	Résultat d'un Pool sur une matrice 3D avec un filtre de taille 224x224 et de stride 2 [WWW16].....	27
19	Structure du réseau de neurones de AlexNet [WWW17].....	29
20	Un réseau avec 4 couches de convolution atteint 25% de taux d'erreur 6 fois plus vite avec des couches de non linéarité <i>ReLU</i> qu'avec des couches de non linéarité <i>Tanh</i> . [7].....	29
21	Les 6 configurations de ce réseau dans l'article original. [13].....	30
22	Les 6 configurations de ce réseau dans l'article original. [3].....	31
23	3 exemples d'opérations d'augmentation de données sur une image [WWW18].....	32
24	Droite de régression linéaire calculée avec un ensemble de points .....	33
2	<b>Apprendre à compter des objets dans une image</b>	
1	Carte de densité du nombre de voiture dans l'image [WWW19] .....	35
3	<b>Towards perspective-free object counting with deep learning</b>	
1	Architecture du réseau de neurones CCNN [12].....	37
4	<b>CrowdNet : A Deep Convolutional Network for Dense Crowd Counting</b>	
1	Architecture du réseau de neurones CrowdNet [1].....	39
2	Les images subissent des transformations pour rendre le réseau robuste au changement d'échelle [1].....	40
3	Crowd estime le nombre de personnes dans l'image en générant une carte de densité [1] .....	41
5	<b>Using Deep Learning for Segmentation and Counting within Microscopy Data</b>	
1	Feature Pyramid Network [11].....	42
2	Architecture du réseau de neurones permettant de compter les cellules dans des images de microscope [4] .....	43

**1 Architecture du système de reconnaissance d'image**

1	L'image est segmenté pour être passé dans le réseau de neurone à la fin la carte de densité finale est recomposée. [12] .....	47
2	Evolution du coût d'apprentissage de différents algorithmes de descente de gradient sur les 3 premières itérations (à gauche) et sur les 45 premières itérations (à droite). Les résultats ont été obtenus avec un réseau convolutif trois couches de convolution 5x5 intercalés avec une couche de Max Pooling et une couche de non linéarité ReLU. [6].....	48
3	Les images d'apprentissage sont transformées en carte de densité.....	49

**2 Structure du code**

1	L'attribut " <i>densityMap</i> " et la méthode " <i>saveDensityMap</i> " sont ajoutés à la classe Image .....	50
---	---	----

**1 Implémentation**

1	CCNN.....	53
2	Le découpage de l'image en patch .....	54
3	Pour chaque pixel on a un certain nombre de prédiction .....	55
4	Le nombre de paramètres par couche du réseau de neurones .....	56
5	Diagramme de classe package ui .....	57
6	Diagramme de classe package ui .....	58
7	Diagramme de classe .....	59
8	Structure des sources 1/2 .....	61
9	Structure des sources 2/2 .....	62

**2 Le dataset**

1	Exemple d'image du dataset .....	63
2	Les différentes localisations .....	64
3	Les images de test .....	64
4	Les images d'apprentissage .....	64

**3 Résultats**

1	IHM.....	66
2	Carte de densité .....	66
3	IHM.....	67
4	Carte de densité .....	67
5	IHM.....	68
6	Carte de densité .....	68
7	Configurations du CCNN.....	69
8	Images de test, epoch 0.....	70

9	Images d'entraînement, epoch 0.....	70
10	Images de test, epoch 10.....	70
11	Images d'entraînement, epoch 10.....	71
12	Images de test, epoch 20.....	71
13	Images d'entraînement, epoch 20.....	72
14	Images de test, epoch 30.....	72
15	Images d'entraînement, epoch 30.....	73
16	Images de test, epoch 40.....	73
17	Images d'entraînement, epoch 40.....	73
18	Evolution de la MAE au cours des epochs 0, 10, 20, 30, 40.....	74
19	Evolution de la MSE au cours des epochs 0, 10, 20, 30, 40 .....	74
20	Evolution de la prédiction pour l'image S1/DSC09768 au cours des epochs 0, 10, 20, 30, 40.....	75
21	L'image S1/DSC09768 originale avec les poissons annotés .....	75
22	Cartes de densité de l'image S1/DSC09768 à l'epoch 0 .....	76
23	Cartes de densité de l'image S1/DSC09768 à l'epoch 10 .....	76
24	Cartes de densité de l'image S1/DSC09768 à l'epoch 20 .....	77
25	Cartes de densité de l'image S1/DSC09768 à l'epoch 30 .....	77
26	Cartes de densité de l'image S1/DSC09768 à l'epoch 40 .....	78
<b>4</b>	<b>Diagramme de Gantt du PRD2</b>	
1	Le gantt des tâches effectuées au PRD2.....	80
<b>A</b>	<b>Description des interfaces externes du logiciel</b>	
1	Capture d'écran de l'IHM.....	84
<b>B</b>	<b>Spécifications fonctionnelles</b>	
1	Format de l'image d'apprentissage .....	86
<b>D</b>	<b>Plan de développement</b>	
1	Diagramme de Gantt .....	93
<b>E</b>	<b>Manuel d'installation</b>	
1	Logo de Anaconda .....	94
<b>F</b>	<b>Manuel d'utilisation</b>	
1	Interface de l'application.....	97

**G Dossier de tests**

1	Description des test .....	102■
---	----------------------------	------

**H Cahier de développeur**

1	Une fonction et ses commentaires.....	105■
---	---------------------------------------	------

Première partie

# Cahier de spécification



# Introduction

Ce document présente les spécifications du PRD<sup>1</sup> "Estimation de la densité de poissons dans les rivières par analyse d'images aériennes".

Le PRD s'inscrit dans un cadre théorique qui fait l'objet d'une étude bibliographique en amont, pour aboutir au développement d'un logiciel. Ce projet est proposé par le département Informatique de l'école polytechnique de l'université de Tours. Il est encadré par Donatello Conte et Pierre Gaucher, enseignants chercheurs au laboratoire informatique dans l'équipe de RFAI qui représenteront la maîtrise d'ouvrage (MOA). Le projet sera réalisé par Mikael Moreau, étudiant en 5ème année au département informatique qui représentera la maîtrise d'œuvre (MOE). Ce PRD se déroule du 20 septembre 2017 à avril 2018 à raison de deux jours complets par semaine.

---

1. Projet Recherche et Développement

# 1

## Contexte de la réalisation

Des aménageurs en écologie souhaitent étudier la biodiversité des fleuves et des rivières et en particulier la population de silures. Les silures forment un genre de poisson d'eau douce de la famille des Siluridae. Ce sont des poissons mesurant en moyenne 1,5 mètre. Pour étudier leur population, les aménageurs ont décidé de prendre des photos aériennes des fleuves et rivières à l'aide d'un drone. Pour faciliter l'estimation du nombre de silures dans une photo, les aménageurs ont fait appel au Laboratoire Informatique de Tours pour concevoir un logiciel permettant de calculer cette estimation. C'est donc dans ce contexte que la MOA propose un projet afin de trouver des solutions à ce problème.

### 1 Acteurs du projet

- La maîtrise d'œuvre (MOE) : Mikael Moreau, étudiant en 5<sup>ème</sup> année de l'Ecole d'Ingénieur Polytech'Tours au sein du département Informatique.
- La maîtrise d'ouvrage (MOA) : Donatello Conte et Pierre Gaucher de l'équipe de recherche de RFAI du Laboratoire Informatique de Tours.



**Figure 1** – Logo du Laboratoire Informatique de Tours

Selon le site du laboratoire informatique [WWW1], « Les domaines d'intérêt et de compétences de l'équipe RFAI relèvent de l'apprentissage automatique, de la fouille de données, et de l'analyse d'images. L'équipe effectue ses recherches sur les méthodes permettant de construire et d'exploiter des représentations de haut niveau sémantique à partir des données en entrée. Cela englobe des modèles et des algorithmes intervenant à différents stades : des traitements bas niveaux (filtrage, segmentation, détection de points d'intérêts, ...) aux méthodes de plus haut niveau (appariement, classification, indexation, ...). »

Ce document a été rédigé par Mikael Moreau et sera validé par les différents acteurs du projet.

## 2 Objectifs

Ce PRD a pour objectif d'aboutir à la conception d'un algorithme d'analyse d'image. La réalisation de l'IHM<sup>1</sup> constitue un autre PRD à part entière et est réalisé par Yaofutian LU, une étudiante en informatique à Polytech Tours. Dans le cadre de ce projet, la MOE doit proposer un algorithme permettant d'estimer le nombre de silures dans une image puis l'intégrer à l'IHM. De ce fait, pour pouvoir s'intégrer correctement à l'IHM, la MOE doit tenir en compte de contraintes fonctionnelles pour la conception et l'implémentation de l'algorithme.

L'algorithme de reconnaissance d'image possède deux modes d'exécution. Le premier est le mode apprentissage qui permet à l'algorithme « d'apprendre » à estimer le nombre de silures dans une image. Le second est le mode prédiction, celui-ci permet d'estimer le nombre de silures dans une image donnée.

Le format des entrées et des sorties de ces deux modes sont décrits dans la partie des spécifications fonctionnelles.

## 3 Hypothèses

Les images utilisées dans ce projet sont des photo prise par un drone. Sur les images on peut voir un fleuve ou une rivière vue du dessus. Sur certaines images, mais pas toutes, on peut voir un ou plusieurs silures nageant dans la rivière.

## 4 Bases méthodologiques

Aucune méthodologie de gestion de projet n'a été imposée par la MOA. La MOE travaille en méthodologie agile. Le projet possède plusieurs jalons avec des livrables attendus à la fin de chacun.

Nous utiliserons le dépôt SVN du Redmine [[WWW2](#)] de Polytech pour la gestion du versionning du code.

---

1. Interface Homme Machine



# 2

## Description générale

### 1 Environnement du projet

L'algorithme est implémenté en Python et fonctionne totalement indépendamment de l'IHM. C'est cette IHM qui se charge d'exécuter l'algorithme d'analyse d'image en lui fournissant une image en entrée. A la fin de l'exécution, l'IHM récupère les sorties de l'algorithme pour les afficher.

### 2 Caractéristiques des utilisateurs

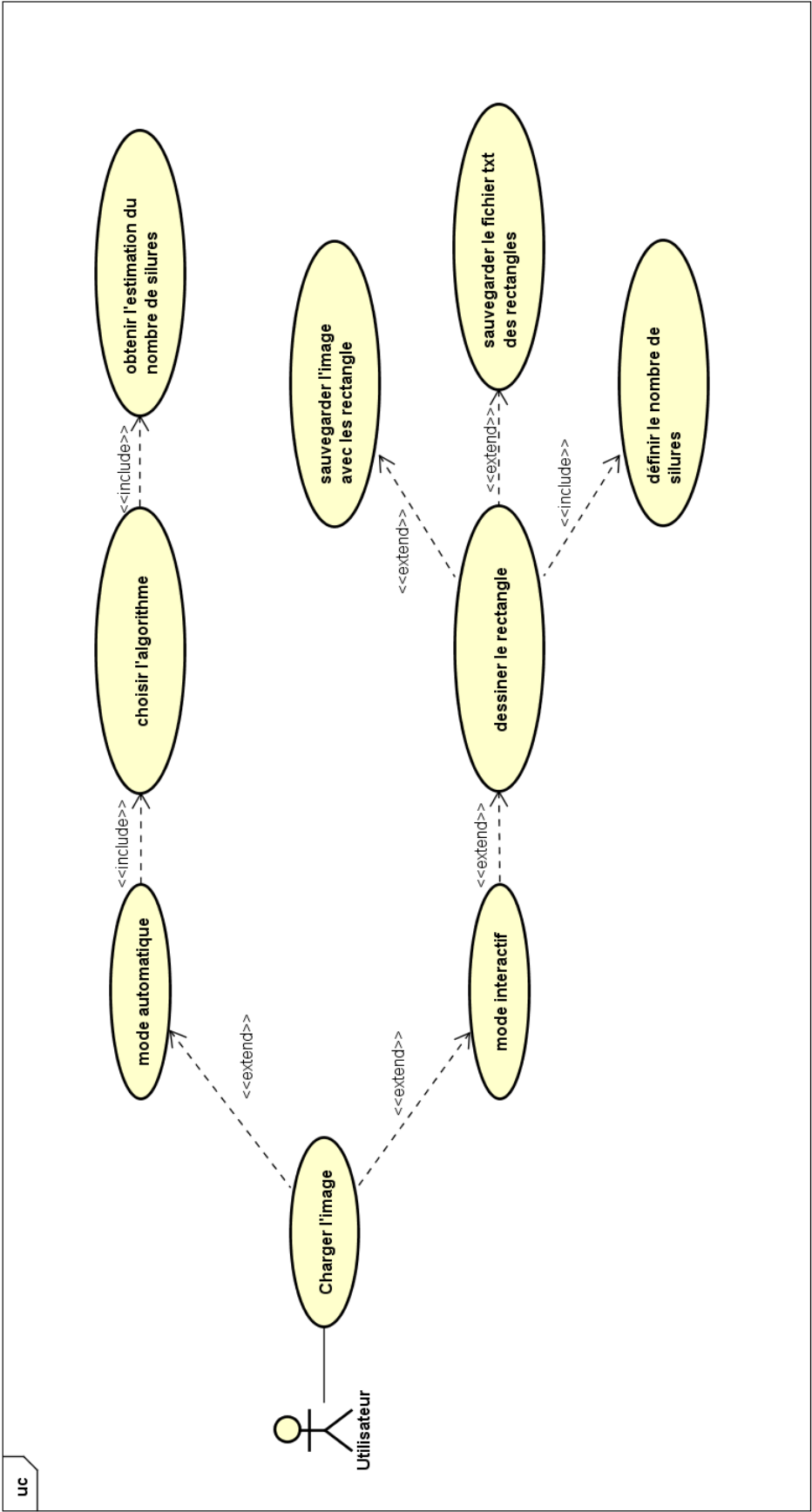
Les utilisateurs du système sont des aménageurs travaillant dans l'écologie des fleuves. L'estimation du nombre de silures dans un fleuve est utile pour évaluer la santé de l'écosystème. Les utilisateurs n'ont généralement jamais fait d'études en informatique. Leurs connaissances en informatique se limite à une utilisation occasionnelle et passive d'ordinateur.

Etant donné qu'aucun prototype du logiciel n'a été développé, les utilisateurs n'ont aucune expérience avec l'application.

L'installation du logiciel et des bibliothèques requises doit être le plus intuitif possible pour l'utilisateur.

### 3 Fonctionnalités et Structure générale du système

Le système se divise en deux modules, le mode automatique et le mode interactif. Le mode automatique est le système de reconnaissance d'image développé dans ce PRD, tandis que le mode interactif est le module développé par Yaofutian LU. Ce second module permet à l'utilisateur le nombre de silures dans l'image de manière interactive et de sauvegarder ses résultats. Pour compter, l'utilisateur dessine un rectangle dans l'image dans lequel il indique le nombre de silures qu'il a compté dedans.

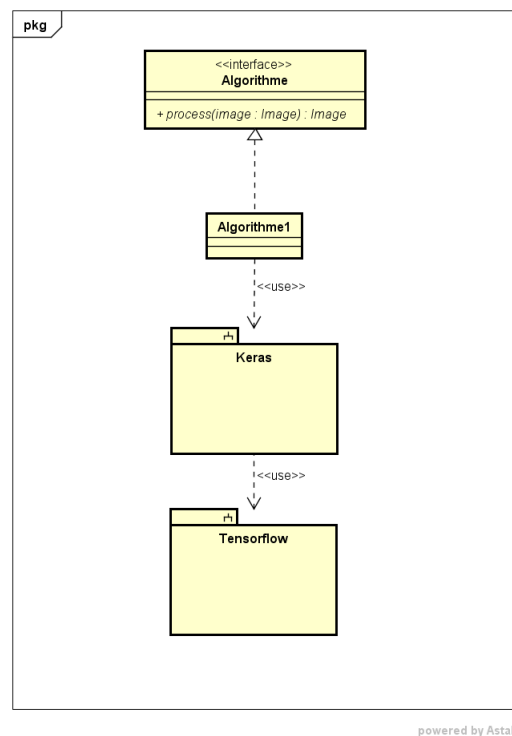


powered by Astah

Figure 1 – Diagramme d'utilisation du système

### 3.1 Le système de reconnaissance d'image

L'algorithme est une classe qui hérite d'une interface « Algorithme ». Cette interface a une méthode « process ». La méthode « process » correspond au mode prédiction de l'algorithme, c'est à dire que c'est la méthode qui permet de prédire le nombre de poissons dans une image.

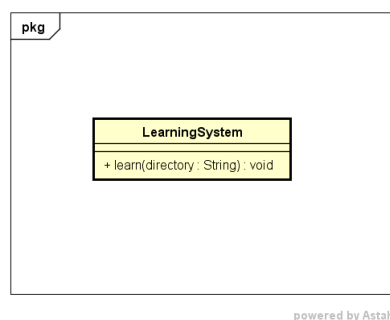


powered by Astah

Figure 2 – Diagramme de classe du système de reconnaissance de formes

Les entrées et les sorties de la méthode « process » sont décrites dans la partie « Spécification fonctionnelles ».

Le mode apprentissage doit aussi être implémenté. La MOA a spécifié qu'il serait implémenté complètement séparément du système de prédiction.



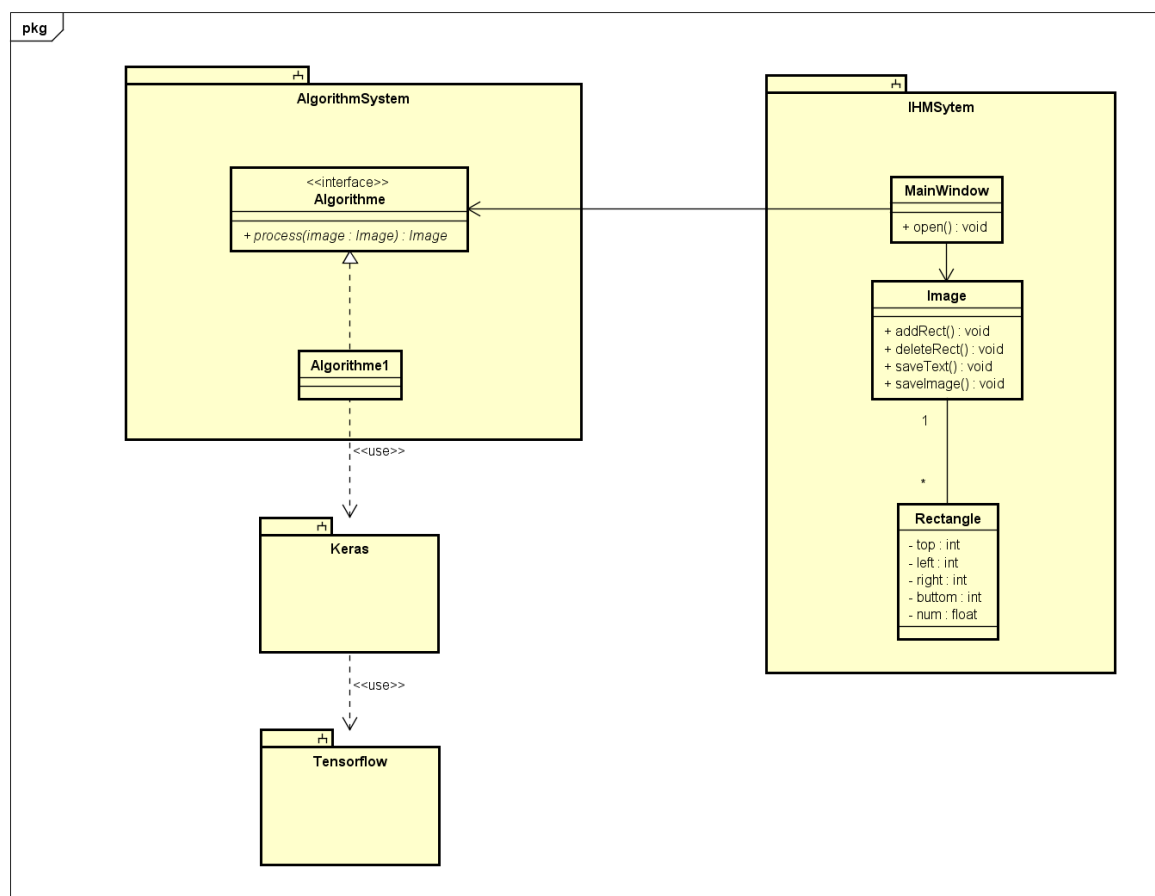
powered by Astah

Figure 3 – Diagramme de classe du système d'apprentissage

Les entrées et les sorties de la méthode « learn » sont décrites dans la partie « Spécification fonctionnelles ».

## 3.2 Intégration de l'algorithme avec l'IHM

La classe « MainWindow » implémente la méthode « open() ». Cette méthode se déclenche lorsque l'on appui sur le bouton qui permet de lancer l'analyse pour prédire le nombre de silures dans l'image.



powered by Astah

Figure 4 – Diagramme de classe du système

L'implémentation de la méthode « open » se fera selon le pseudo-code suivant :

```
1 public open()  
2 {  
3     Algorithme a ;  
4     Image img = getImage() ;  
5  
6     switch(n) {  
7     case 0:  
8         a = new Algorithme1() ;  
9         break;  
10    case 1:  
11        a = new Algorithme2() ;  
12        break;  
13    case 2:  
14        a = new Algorithme3() ;  
15        break;  
16    default :  
17        return 0 ;  
18    }  
19  
20    Image image = a.process(img) ;  
21  
22    return image ;  
23 }
```

## Deuxième partie

### État de l'art



# Introduction

La reconnaissance des formes est un domaine de recherche issue de différentes disciplines qui sont les mathématiques (probabilités et statistiques), les sciences de l'ingénieur, l'informatique et l'intelligence artificielle. C'est à partir des années 60 que la reconnaissance de formes est devenue une discipline à part entière.

L'objectif de la reconnaissance de formes est de construire des systèmes informatisés permettant de réaliser des tâches de reconnaissance et de classification. La reconnaissance de l'écrit, la reconnaissance de la parole et l'interprétation d'image sont des applications concrètes de la reconnaissance des formes.

# 1

# La reconnaissance de formes et le machine learning

## 1 Le machine learning

La reconnaissance de formes est un domaine d'application particulier du machine learning. Il existe beaucoup d'algorithmes de machine learning, la **Figure 1** permet d'avoir une vision globale des différents algorithmes du machine learning.



**Figure 1** – Schéma représentant les différents algorithmes de machine learning [WWW3]

Dans cet état de l'art nous aborderons plusieurs types d'algorithmes tels que les réseaux de neurones (Neural network), l'apprentissage profond (deep learning) et des algorithmes de régression et de classification.



## 2 Schéma général d'un système de reconnaissance de formes

Un système de reconnaissance de formes se divise généralement en 6 étapes.

- La saisie
- Les prétraitements
- L'extraction de caractéristiques
- L'apprentissage
- La décision
- Les post-traitements

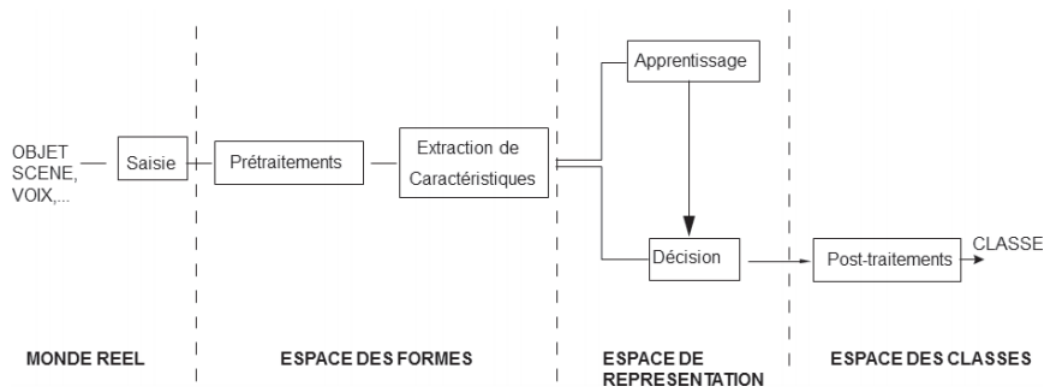


Figure 2 – Schéma général d'un système de reconnaissance de formes [10]

### 2.1 La saisie

La saisie correspond à l'étape de numérisation des données. Par exemple, il est possible de numériser le texte d'un livre ou d'enregistrer un son avec un micro. Dans cette étape l'information est transformée pour être lisible pour un ordinateur.

### 2.2 Les prétraitements

Cette étape consiste à réduire les « bruits » indésirables. Cette étape sert à préparer aux suivantes en réduisant la variabilité du signal. Par exemple pour certains algorithmes de lecture cela consiste à placer la lettre qui doit être lue au centre de l'image.

### 2.3 L'extraction de caractéristiques

Les caractéristiques correspondent aux dimensions sur lesquelles vont s'exprimer nos données d'entrée. Par exemple, nous concevons un algorithme de reconnaissance de trois types de véhicules : les voitures, les motos et les bus. Les caractéristiques peuvent être le nombre de roues, la longueur du véhicule, le nombre de fenêtre, etc. Sur chaque caractéristiques une valeur numérique sera exprimée pour chaque image.

Sur la Figure 3 on peut imaginer que chaque point correspond à une image analysée. Les véhicules du même type auront tendance à se regrouper sur le graphe. Les triangles pourraient correspondre aux voitures, les ronds aux motos et les carrés au bus. Les types de véhicules correspondent à ce que l'on appelle les « classes ».

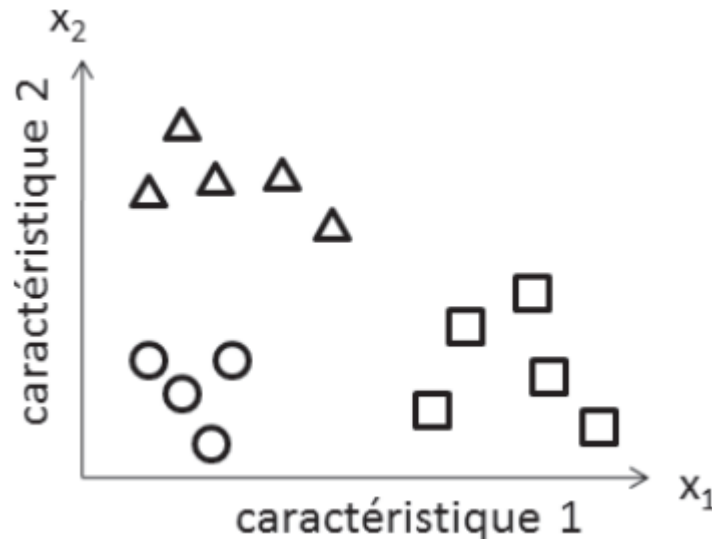


Figure 3 – Différentes entrées représentées dans l'espace des caractéristiques [10]

L'étape d'extraction des caractéristiques est l'étape qui permet de trouver ces caractéristiques. On cherchera à trouver des caractéristiques de type RST<sup>1</sup>, c'est à dire invariant à la rotation l'échelle et la translation. En reconnaissance de formes, les algorithmes de construction de caractéristiques du type SIFT<sup>2</sup> ou HOG<sup>3</sup> sont très utilisés.

Dans la réalité les caractéristiques utilisées sont beaucoup plus abstraites que celles que j'ai utilisées pour illustrer mon exemple. Bien souvent, il est très difficile d'interpréter ce à quoi elles correspondent vraiment.

## 2.4 La décision

Cette étape consiste à évaluer dans quelle classe appartient une donnée à partir du point obtenu dans l'espace des caractéristiques. Dans l'exemple précédent cela revient à évaluer si un véhicule est une moto, une voiture ou un bus en fonction de sa position dans le graphe précédent. Dans cet exemple il faut noter qu'il y a une quatrième classe, celle dans laquelle on range une image lorsque ce n'est ni une moto, ni une voiture, ni un bus.

## 2.5 L'apprentissage

Cette étape permet à l'algorithme d'effectuer l'étape de décision avec le moins d'erreur possible. Elle va conditionner le résultat de l'étape de décision.

En machine learning, il existe plusieurs types d'apprentissage :

- Supervisé
- Non supervisé

L'apprentissage non supervisé permet en plus de trouver automatiquement les différentes classes. On regarde la distribution des variables. Cela nous permet de distinguer les différentes classes.

---

1. Rotation Scale Translation  
 2. Scale Invariant Feature Transform  
 3. Histogram of Oriented Gradient

L'apprentissage supervisé est un mode d'apprentissage où les classes sont déjà définies dans l'algorithme. Pour chaque entrée on connaît la sortie attendue. Par exemple pour chaque image passé à l'algorithme on connaît à quelle classe il appartient. L'algorithme utilise l'étape de décision et compare la classe qu'il a obtenue avec celle attendue. Il met à jour le mécanisme de calcul décision de la classe.

Imaginons un algorithme de prédiction du sexe d'une personne par photo. Lors de l'apprentissage, une image avec la bonne réponse. L'algorithme va essayer de deviner le sexe de la personne puis va comparer le résultat avec la bonne réponse. Il va ensuite ajuster le système de calcul de la réponse en conséquence.

Dans le cadre du projet, nous nous intéresserons uniquement à l'apprentissage supervisé.

### 3 Le Perceptron

Le Perceptron est une méthode de discrimination paramétrique des classes. C'est-à-dire qu'il a pour but de tracer des frontières entre les classes, pour cela on utilise la séparation linéaire. La forme la plus simple de surface séparatrice est l'hyperplan. Ce classifieur linéaire intervient après la phase d'extraction des caractéristiques.

#### 3.1 La règle d'apprentissage du Perceptron et la descente de gradient

La règle d'apprentissage du Perceptron est un algorithme simple qui permet de trouver un hyperplan séparateur en un nombre fini d'itération à partir des données d'apprentissage. Cependant les classes doivent être linéairement séparables, c'est-à-dire qu'il existe un hyperplan  $H$  séparant, sans erreur de classification, les données d'apprentissage.

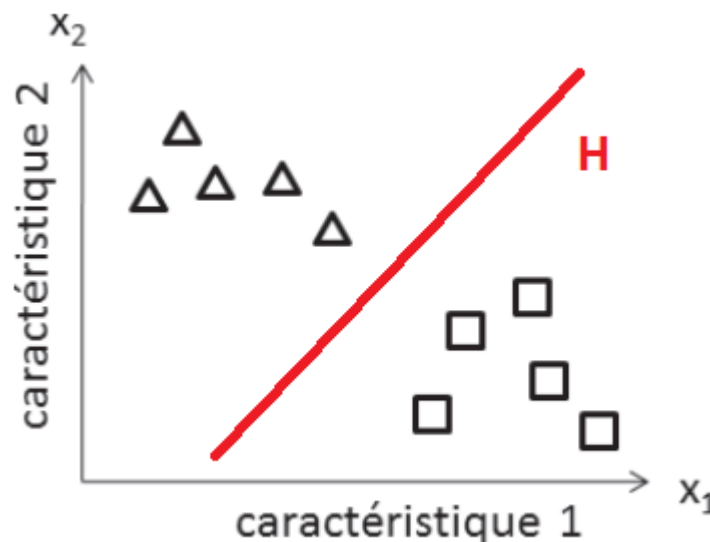


Figure 4 – Hyperplan séparant deux classes

L'hyperplan  $H$  a pour équation :

$$\vec{w} * \vec{x} + b = 0$$

$\vec{w}$  est le vecteur qui permet de donner le coefficient directeur de l'hyperplan.  $\vec{x}$  est le vecteur des valeurs pour chaque caractéristique pour une image donnée.  $b$  permet de transformer la fonction linéaire en une fonction affine.

La règle d'apprentissage du Perceptron permet de déterminer  $\vec{w}$  et  $b$ .

Cette règle cherche à minimiser une fonction de coût correspondant à la différence entre la sortie produite du système et la sortie désirée. Elle est nulle si tous les vecteurs de l'ensemble d'apprentissage sont bien classés. La règle d'apprentissage du Perceptron consiste à se déplacer dans le sens opposé au gradient de la fonction de coût. Se déplacer signifie mettre à jour les poids c'est-à-dire  $\vec{w}$ , pour se rapprocher de la sortie désirée. A chaque fois que l'on va passer un vecteur d'apprentissage, les poids seront mis à jour pour se rapprocher de la sortie désirée. On appelle cette méthode une méthode de descente de gradient stochastique.

Cet algorithme de descente de gradient utilise un paramètre important : le pas d'apprentissage. Sa valeur est positive, il définit la vitesse à laquelle l'algorithme va converger lors de la minimisation de la fonction de coût à chaque apprentissage. S'il est trop faible, la convergence est ralentie. S'il est trop fort, cela peut entraîner un dépassement de l'optimum.

Le Perceptron peut être interprété comme un réseau connexionniste à une couche. La phase d'apprentissage du réseau consiste à déterminer les valeurs des poids de chaque caractéristique.

Avec cet algorithme, l'hyperplan trouvé a tendance à passer trop près des données. De plus, l'algorithme ne converge pas si les classes ne sont pas linéairement séparables.

### 3.2 La règle de décision du Perceptron

Soit le vecteur  $\vec{a}$ , l'image en entrée représentée dans l'espace des caractéristiques. On calcule la sortie  $y$  pour cette image.

$$y = \vec{w} * \vec{a} + b$$

On passe la sortie dans une fonction d'activation  $f(y)$ . Généralement cette fonction est la fonction signe  $f = \text{sgn}$ . Celle-ci retourne 1 si  $y$  positif ou égal à zéro et  $-1$  sinon. On peut en déduire de quel côté est le point par rapport à l'hyperplan et donc de savoir si l'image appartient à la classe ou non.

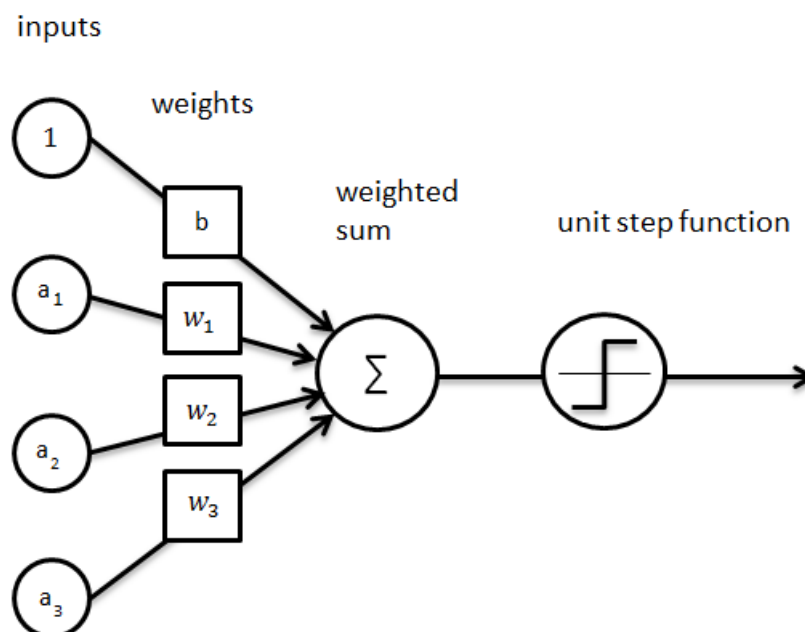


Figure 5 – Représentation connexionniste du Perceptron [WWW4]

Sur la [Figure 5](#), on peut voir qu'il y a une entrée supplémentaire de valeur 1 qui a pour poids  $b$ . Cette entrée est appelée un nœud de type biais et permet, comme dit plus haut, une liberté de mouvement supplémentaire pour l'hyperplan dans l'espace des caractéristiques.

## 4 SVM (Support Vector Machines)

Comme le Perceptron, il s'agit d'une méthode de discrimination paramétrique des classes. C'est une variante du Perceptron en introduisant une marge entre l'hyperplan et les points dans l'espace des caractéristiques.

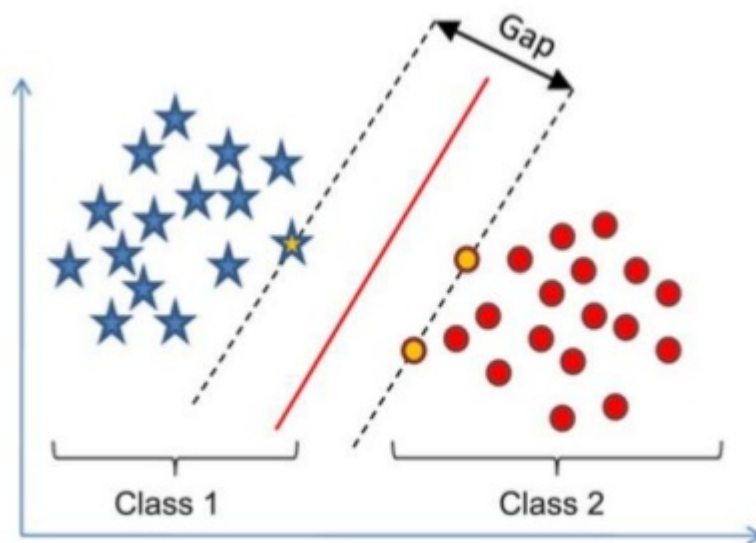


Figure 6 – Les SVM introduisent la notion de marge entre l'hyperplan et les classes [WWW5]

## 5 La minimisation aux moindres carrés

La minimisation aux moindres carrés est un algorithme qui est une variante de la règle d'apprentissage du Perceptron. Cet algorithme, dit de Widrow-Hoff, s'affranchit des problèmes d'oscillation dans le cas de classes non linéairement séparables et qui est plus robuste (moins tangent aux données). Sa caractéristique est qu'il minimise la somme des carrés des distances des points par rapport à l'hyperplan  $H$ . Cet algorithme nécessite d'avoir tous les points de l'ensemble d'apprentissage à disposition contrairement à l'algorithme de descente de gradient qui lui ne se sert que du dernier vecteur d'apprentissage et des poids actuels du Perceptron. De plus, il demande beaucoup de mémoire et beaucoup de calcul.

## 6 Réseaux de neurones

### 6.1 Réseaux à un seul neurone

Le Perceptron peut être apparenté à un neurone artificiel simple. Il existe plusieurs modèles pour représenter un réseau à un seul neurone :

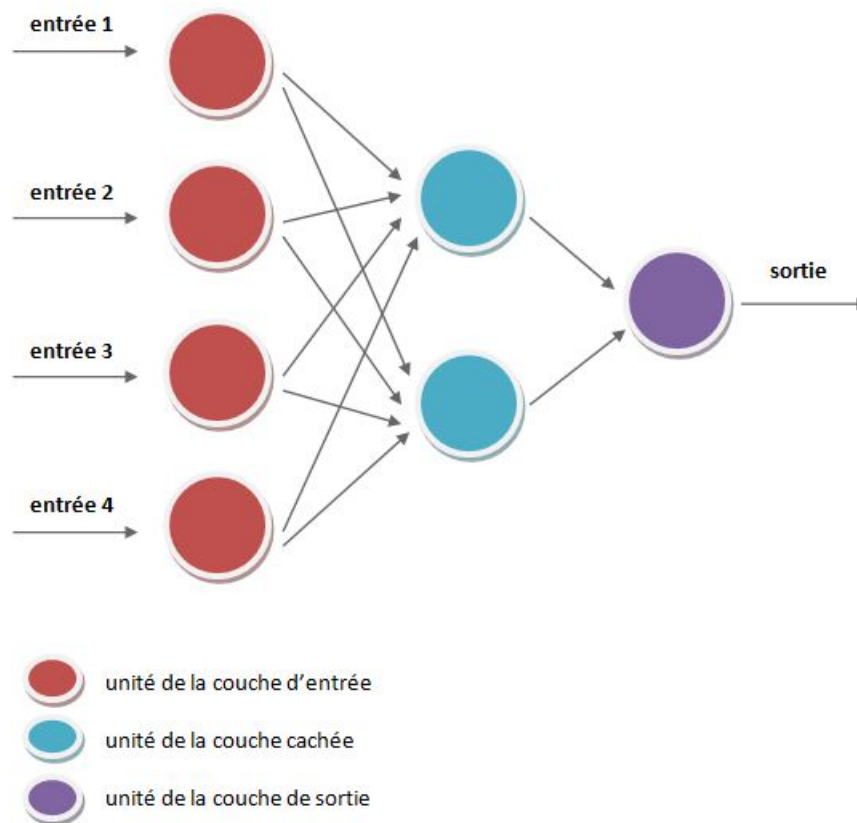
- Perceptron

- TLU = Threshold Logic Unit
- Adaline = Adaptive Linear Neurone
- ALC = Adaptive Linear Combiner

Leurs fonctionnements sont sensiblement similaires à celui du Perceptron.

## 6.2 Réseaux de neurones multicouches

Les réseaux de neurones à une seule couche sont seulement capables de résoudre des problèmes linéairement séparables. Pour remédier à ce problème, on peut combiner la sortie de plusieurs neurones. Ces réseaux de neurones peuvent construire des frontières implicites de décision.

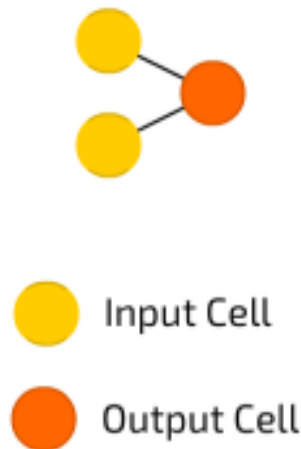


**Figure 7** – Schéma d'un réseau multicouche avec une couche d'entrée, une couche cachée et une couche de sortie [WWW6]

La **Figure 7** représente un réseau à deux couches de Perceptron avec un espace des caractéristiques à quatre dimensions. La couche d'entrée représentant les vecteurs d'entrées et non des Perceptrons.

Pour mieux comprendre la **Figure 8** représente un réseau de neurones composé d'un seul Perceptron avec un espace des caractéristiques à deux dimensions.

Perceptron (P)



**Figure 8** – Un perceptron dont la taille du vecteur d’entrée est de 2 représenté comme un réseau de neurones [WWW7]

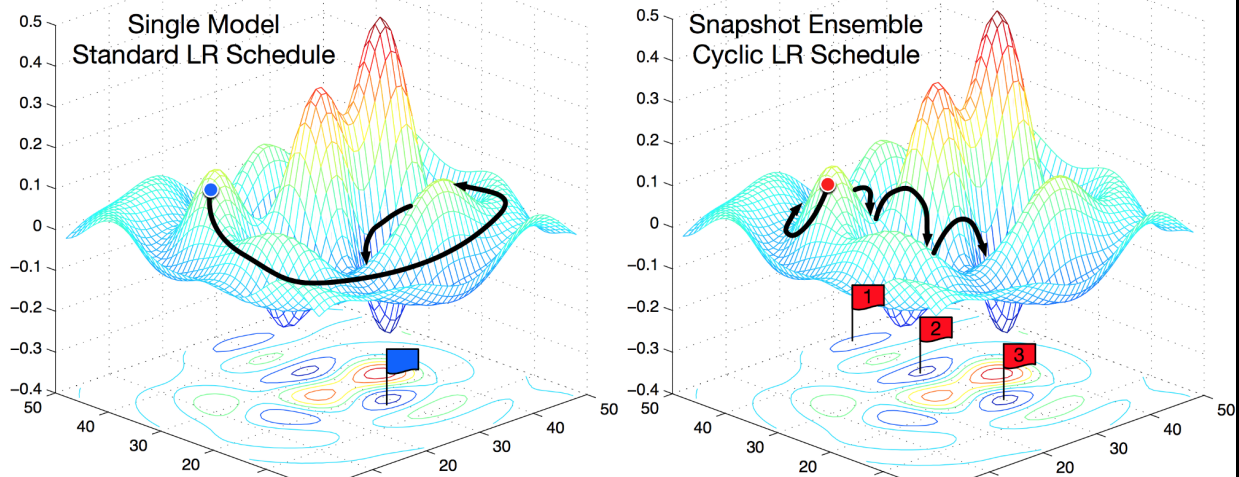
### 6.3 Apprentissage par rétropropagation

Précédemment nous avons vu qu’un Perceptron pouvait « apprendre » en trouvant un hyperplan séparateur entre deux classes. Que ce soit avec la règle de Perceptron séquentiel ou avec la descente de gradient, le Perceptron a besoin d’une valeur cible (la classe) pour pouvoir trouver l’hyperplan. Dans un réseau de neurones multicouche, les couches cachées n’ont pas de valeur cible. Il faut donc revoir la façon dont on met à jour les poids pour un réseau multicouche.

La solution est d’utiliser un apprentissage par rétropropagation. Il existe plusieurs algorithmes d’apprentissage par rétropropagation, voici ceux qui sont disponibles avec la bibliothèque de Deep Learning Keras [WWW8].

- SGD = Stochastique Gradient Descent
- RMSprop
- Adagrad
- Adadelata
- Adam
- Adamax
- Nadam
- TFOptimizer

Les algorithmes d’apprentissage par rétropropagation ont en commun que la mise à jour des poids commence à la couche de sortie et se propage jusqu’à la couche d’entrée. Ces algorithmes utilisent des méthodes de descente de gradient stochastique.



**Figure 9** – La méthode de descente de gradient stochastique consiste à trouver un minimum dans un espace multidimensionnel. Ici sont représenté deux configurations différentes de l'algorithme SGD. Celle de gauche trouve directement le minimum global tandis que l'autre tombe dans plusieurs minimums locaux avant de trouver la solution optimale [WWW9]

Comme pour la règle du Perceptron, on cherche à minimiser une fonction de coût et à se déplacer dans le sens opposé au gradient de cette fonction. On a aussi le choix concernant la fonction de coût à utiliser. Voici celle qui sont disponibles avec la bibliothèque Keras [WWW10].

- mean\_squared\_error
- mean\_absolute\_error
- mean\_absolute\_percentage\_error
- mean\_squared\_logarithmic\_error
- squared\_hinge
- hinge
- categorical\_hinge
- logcosh
- categorical\_crossentropy
- sparse\_categorical\_crossentropy
- binary\_crossentropy
- kullback\_leibler\_divergence
- poisson
- cosine\_proximity

## 6.4 Option d'apprentissage d'un réseau de neurone

Le concepteur du réseau de neurone doit faire des choix avant de pouvoir lancer l'apprentissage du réseau.

### 6.4.1 Initialisation des poids

Si, pour chaque Perceptron, on initialise les poids avec les mêmes valeurs alors la mise à jour des poids sera identique pour tous les Perceptrons d'une même couche, on perd alors l'utilité d'avoir plusieurs Perceptron. L'initialisation des poids doit être aléatoire, généralement on utilise une



loi uniforme. La seule contrainte est que la somme des poids arrivant sur un Perceptron doit être égale à 1.

#### 6.4.2 Apprentissage séquentiel ou par cycle (en batch)

Avec l'apprentissage séquentiel, on modifie les poids après chaque exemple passé à l'algorithme. Avec un apprentissage par cycle, on modifie les poids après avoir passé tous les exemples. Les poids sont modifiés avec la modification accumulée de chaque poids ou parfois la moyenne de la modification de chaque poids mais cela équivaut à prendre un pas d'apprentissage plus petit.

#### 6.4.3 Ordre de présentation des exemples

Avec un apprentissage par cycle ça ne change rien. En revanche, avec un apprentissage séquentiel cela a un impact. On peut présenter les exemples de manière aléatoire pour avoir un apprentissage équilibré, mais ce qu'il ne faut surtout pas faire c'est de présenter les exemples par classe.

#### 6.4.4 Fonction d'activation

Il existe plusieurs fonctions d'activation utilisées, en voici quelques-unes :

- La fonction *Tanh*
- La fonction *Sigmoïde*
- La fonction *ReLU* (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

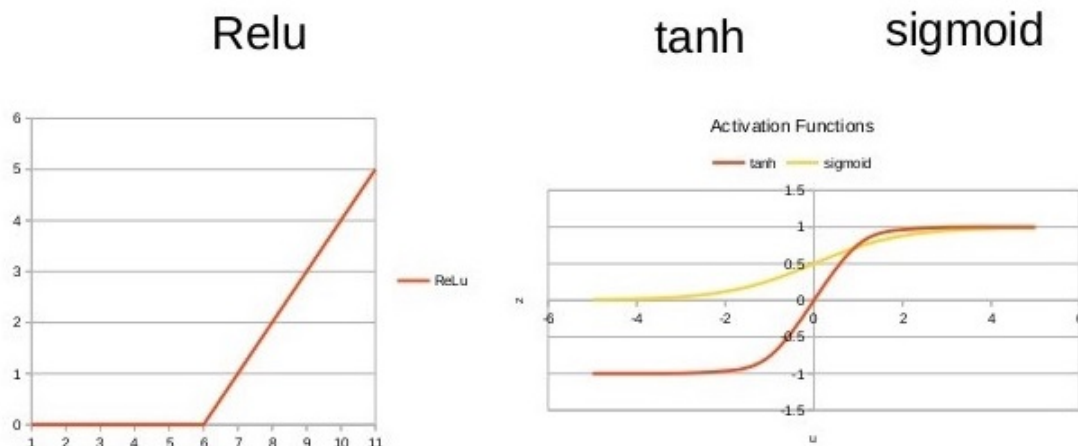


Figure 10 – Les courbes des fonctions d'activation ReLU, Tanh et Sigmoïd [WWW11]

- La fonction *Threshold* ou *Step*, celle du Perceptron
- La fonction signe *sgn*, généralement utilisé pour les réseaux Madalines (pour Multiple Adaline) et pour la classification de la couche de sortie.

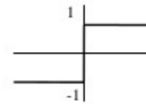
- **Step function**

$$step_t(x) = \begin{cases} 1 & x > t \\ 0 & \text{otherwise} \end{cases}$$



- **Sign function**

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & \text{altrimenti} \end{cases}$$



**Figure 11** – Les fonctions d'activation Step et Sign [WWW12]

Une chose importante est que la fonction d'activation des couches cachées ne soit pas linéaire.

#### 6.4.5 Nombre de nœuds cachés

Le nombre de nœuds cachés reflète le montant d'information pouvant être stocké dans un réseau. Si il n'y a pas assez de nœuds alors on ne peut pas séparer les classes. Si il y a trop de nœuds on va faire du sur-apprentissage, et on ne pourra plus classer les données car le réseau considèrera des exemples d'un même classe trop différents.

#### 6.4.6 Pas d'apprentissage

Si il est trop petit, le réseau mettra trop de temps pour atteindre les poids optimaux et on risque de rester coincé dans un minimum local. Si il est trop grand, on risque de ne jamais converger et de dépasser les poids optimaux. Selon « Technosup, Reconnaissance des formes » [10], 0.1 est souvent un bon choix.

#### 6.4.7 Nœuds de type biais

On appelle le nœud de type biais le nœud supplémentaire dont la valeur d'entrée est toujours 1, voir le chapitre sur le Perceptron. Il permet de modifier une frontière linéaire de décision en une fonction affine. Il peut être ajouté sur n'importe quelle couche même les couches cachées. Un seul biais ne peut être ajouté par couche.

#### 6.4.8 Elagage

L'élagage consiste à retirer les nœuds dans les couches cachées dont les poids sont très faibles après un premier apprentissage. Cela permet d'éviter le sur-apprentissage. On peut ensuite continuer l'apprentissage sur le nouveau réseau. Le risque est que l'on peut supprimer des branches importantes.

#### 6.4.9 Vecteur de sortie

Le format de sortie le plus populaire est d'associer un nœud de sortie par classe. Ainsi, le nœud qui correspond à la classe dont est issue le vecteur d'entrée aura pour valeur cible 1. Les autres nœuds auront alors comme valeur cible 0 (ou -1). Par exemple, si les entrées du réseau peuvent être classées dans 7 classes différentes et que le vecteur d'entrée appartient à la 5ème classe alors le vecteur de sortie cible aura la valeur  $[0, 0, 0, 0, 1, 0, 0]$ . Une autre manière de classer les entrées est de n'avoir qu'un seul nœud de sortie et d'attribuer une plage de valeur de la valeur de sortie à chaque classe. Par exemple de 0 à 0,1 pour la classe 1, 0,1 à 0,2 pour la classe 2, etc. On peut aussi choisir d'avoir un seul nœud de sortie qui ne possède pas de fonction d'activation, c'est utile quand on ne souhaite pas faire de la classification mais de la prédiction de valeurs numériques.

#### 6.4.10 Normalisation des données

Les réseaux de neurones sont incapables de détecter les décalages dans les vecteurs d'entrée. Généralement les valeurs sont centrées autour de zéro et les caractéristiques sont au même ordre de grandeur.

#### 6.4.11 La fonction de coût

La fonction représentant la différence entre la sortie produite et la sortie désirée. C'est la fonction que l'on cherche à minimiser.

#### 6.4.12 L'algorithme de mise à jour des poids

L'algorithme à utiliser pour minimiser la fonction de coût.

#### 6.4.13 Critère d'arrêt

Selon les algorithmes de mise à jour des poids du réseau de neurones, il faut parfois définir un critère d'arrêt pour la fonction de coût. Dans la plupart des cas, les algorithmes de mise à jour des poids sont suffisamment intelligents, ils calculent eux-mêmes le critère d'arrêt.

## 7 L'apprentissage profond (deep learning)

La motivation principale de l'apprentissage profond est d'entraîner un système de reconnaissance d'image A à Z. Au lieu de construire le système de reconnaissance de formes avec un extracteur de caractéristiques suivi d'un classifieur entraînable, il est préférable que l'extracteur de caractéristiques soit aussi entraînable pour qu'il y ait le moins de travail possible à faire à la main.

Une image peut être vue comme une collection de petits motifs locaux, comme des contours orientés. Un assemblage de ces contours orientés peut former des motifs plus complexes qui, à leur tour peuvent être assemblés pour constituer des formes de plus en plus complexes et de plus en plus abstraites (Figure 12). En suivant ce schéma de composition de motifs de

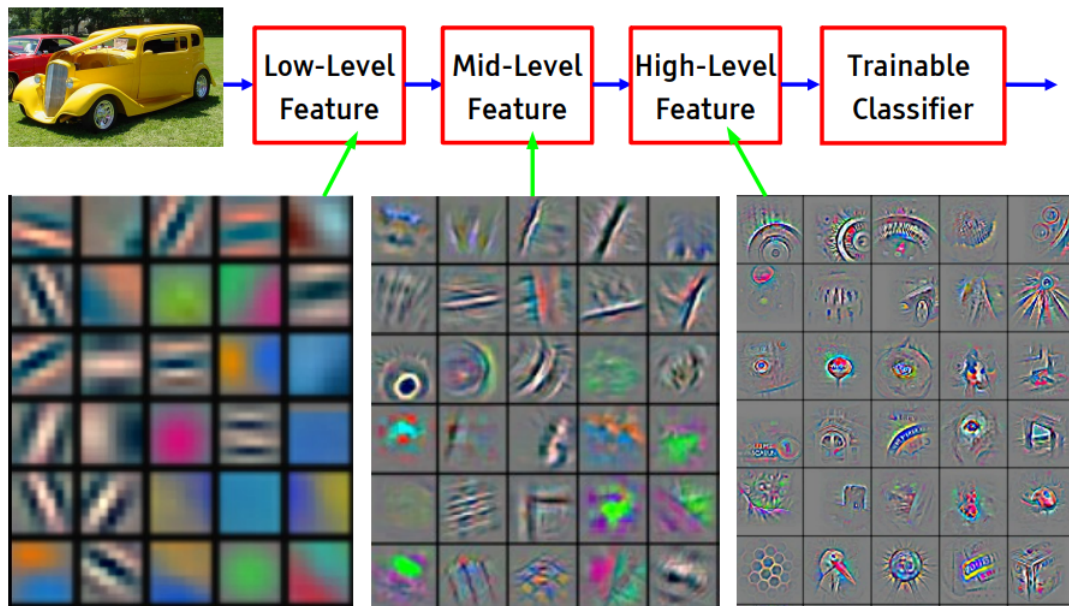


Figure 12 – Les différents niveaux de caractéristiques dans un système de reconnaissance de formes profond [8] [WWW23]

façon hiérarchique, on peut déduire les caractéristiques d'un objet. C'est sur ce paradigme que l'apprentissage profond est basé.

L'extracteur de caractéristiques est découpé en modules chacun pouvant extraire des caractéristiques de bas, de moyen ou de haut niveau. Chaque module est séparé par des frontières de décision non linéaires.

En 2012, la compétition de reconnaissance d'image organisée par Image-Net «Large Scale Visual Recognition Challenge» a été remportée par un algorithme d'apprentissage profond nommé «SuperVision» dans la catégorie classification [WWW13]. Depuis cette compétition, l'apprentissage profond est devenu très populaire dans le domaine de la reconnaissance d'images. Les types de réseaux d'apprentissage profond les plus populaires pour la reconnaissance d'images sont les réseaux convolutifs. C'est d'ailleurs sur ce type de réseau qu'était basé SuperVision.

## 7.1 Les réseaux convolutifs

Les réseaux convolutifs sont des réseaux de neurones multicouches avec plus de 2 couches cachées. Le problème est que plus un réseau est profond, plus on risque de faire un surapprentissage. En effet, de par sa trop grande capacité à stocker des informations, une structure dans une situation de surapprentissage aura de la peine à généraliser les caractéristiques des données.

Pour remédier à ce problème, les chercheurs ont introduit des nouveaux types de couches dans leurs réseaux de neurones. En voici quelques-uns :

- Convolution Layer
- Pooling Layer
- Dropout Layer
- Batch normalization layer
- Fully Connected layer

### 7.1.1 Dropout Layer :

Le Dropout consiste à désactiver des neurones durant l'apprentissage. Son rôle principal est de contrôler le sur-apprentissage.

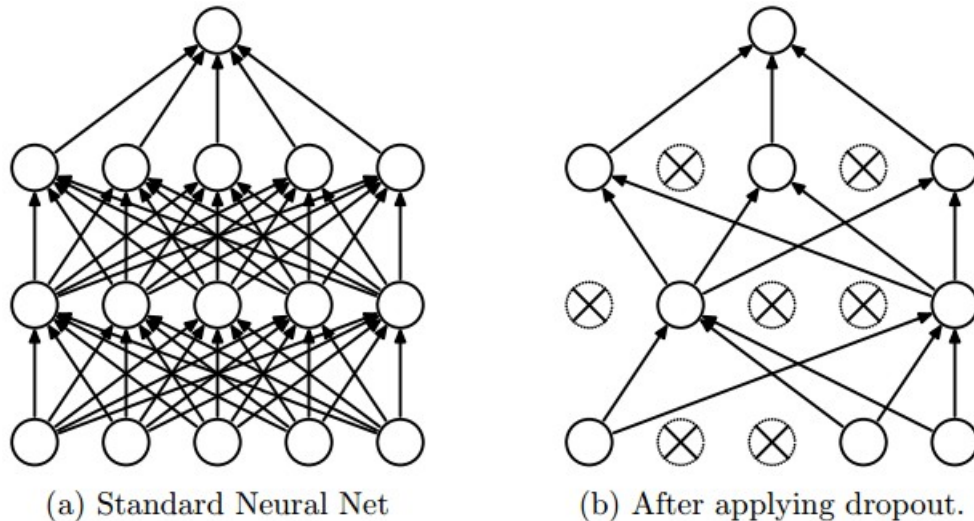


Figure 13 – Le Dropout consiste à désactiver des neurones durant l'apprentissage [WWW14]

### 7.1.2 Fully connected Layer :

Les couches entièrement connectées correspondent aux couches classiques d'un réseau de neurones où toutes les entrées sont connectées à chaque neurone de la couche (Figure 14). Dans les réseaux convolutifs, ils sont souvent utilisés à la fin pour servir de classifieur.

### 7.1.3 Convolution Layer :

Avant l'émergence des réseaux de neurones convolutifs, l'apprentissage profond ne produisait pas de résultats satisfaisants en reconnaissance d'image.

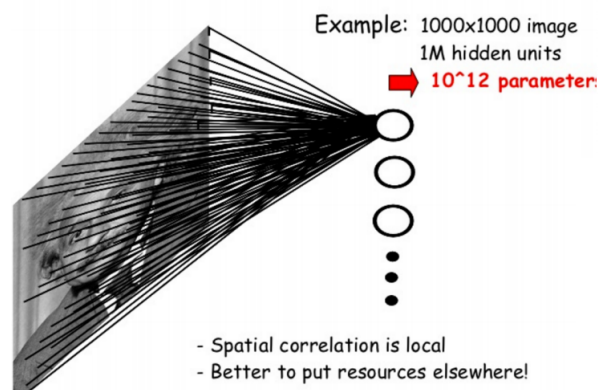


Figure 14 – Une couche entièrement connectée, (fully connected layer) [WWW15]

En effet, avec une image d'entrée de taille 1000 par 1000 pixels, on avait 1 millions de poids par neurones. De plus, pour produire des résultats satisfaisants les images devaient être toutes centrées sur l'objet à reconnaître et ce dernier devait toujours avoir à peu près la même taille.

Les réseaux convolutifs permettent de pallier ces problèmes. Pour comprendre le fonctionnement d'une convolution, prenons un exemple avec une image RVB de 32 pixel de hauteur et de largeur. Cette image est représentée dans une matrice de dimensions  $32 \times 32 \times 3$  (Figure 15). Le neurone de convolution contient un filtre. Ce filtre est une matrice où chaque élément est un poids, c'est ce que l'on cherche pendant l'apprentissage. Ce filtre est de 3 dimensions, la hauteur et la largeur sont arbitraires et ils sont à fixer par le concepteur du réseau de neurone à condition d'être inférieur à la taille de l'image. Ici la largeur et la hauteur du filtre sont de 5 pixels. La profondeur du filtre est obligatoirement la même que l'image.

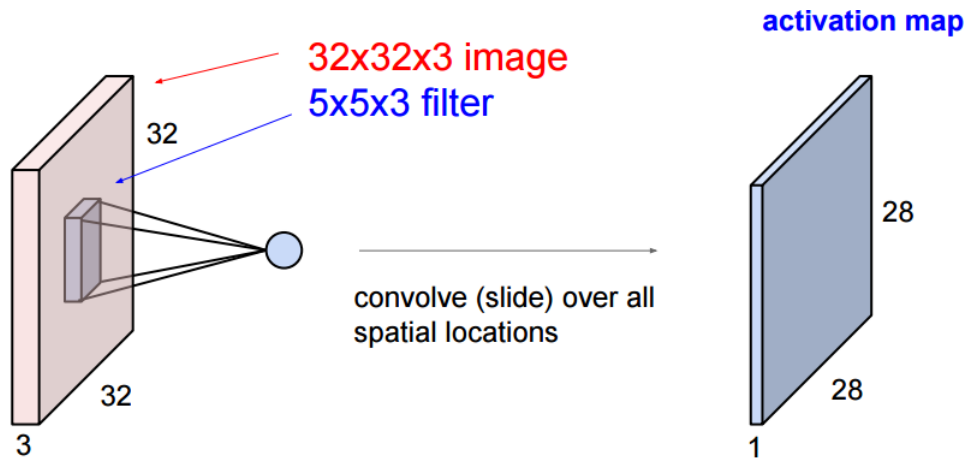


Figure 15 – Opération de convolution sur une image [WWW15]

Le filtre va littéralement scanner toute l'image. Pour chaque position où l'on peut mettre le filtre, on obtient une valeur qui est la combinaison linéaire entre les poids du filtre et le morceau de l'image où est le filtre. La sortie du neurone est donc une matrice de 2 dimensions de hauteur et de largeur :  $32 - 5 + 1 = 28$  pixels.

Si l'on veut que notre couche de neurones convolutifs recherche plusieurs formes, on peut mettre plusieurs filtres. Avec 6 filtres, la sortie de notre couche de neurones convolutifs sera une matrice de dimensions  $28 \times 28 \times 6$  (Figure 16).

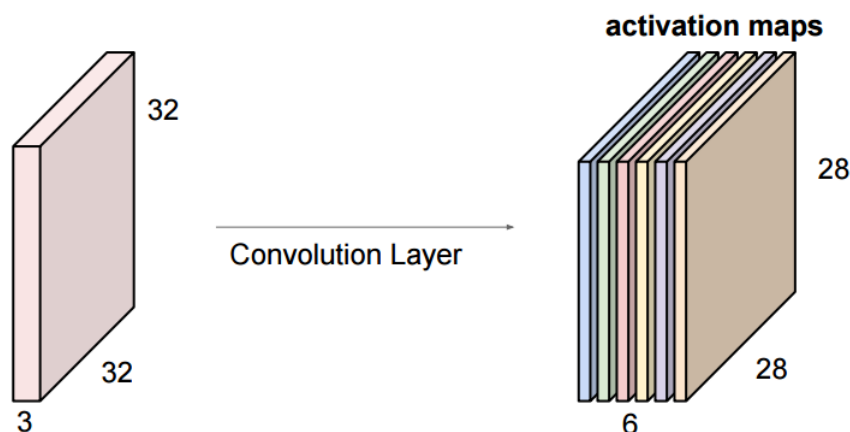


Figure 16 – Résultat d'une opération de convolution sur une image avec plusieurs filtres [WWW15]

L'utilité d'une telle technique est que les filtres nous permettent de rechercher des formes en scannant l'image.

Les paramètres à fixer pour une couche de neurones convolutifs sont :

- Kernel size(K) : Les dimensions du filtre.



- Stride(S) : Le pas de déplacement en pixel du filtre dans l'image.
- Zero Padding(pad) : Mettre des zéro autour de l'image avant l'opération de convolution, cela permet de détecter des formes qui seraient coupées par les bords de l'image et que la sortie ait les mêmes dimensions que l'image d'entrée.
- Number of filters(F) : Le nombre de filtres.

#### 7.1.4 Pooling Layer :

Il est commun d'insérer une couche de Pooling après une couche de convolution dans les réseaux convolutifs. La couche de pooling a pour but de réduire la taille de la matrice d'entrée. Cela permet de réduire le nombre de poids d'un réseau et permet de contrôler le surapprentissage. Cette couche ne contient aucun poids et n'est donc pas une couche d'apprentissage.

Tout comme pour l'opération de convolution, un filtre scanne la matrice de 2 dimensions. Pour chaque position du filtre dans la matrice 2D, on garde une seule valeur, le maximum ou la moyenne des valeurs de la matrice là où se trouve le filtre (Figure 17).

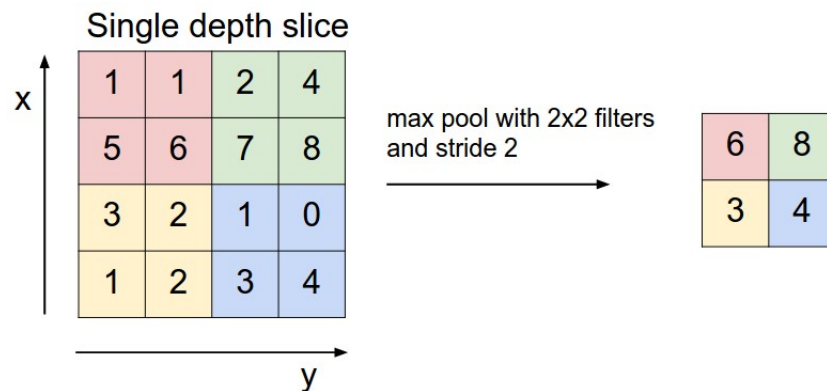


Figure 17 – Résultat d'un Max Pool sur une matrice avec un filtre de taille 2x2 et de stride 2 [WWW16]

Le Pooling diminue la hauteur et la largeur de la matrice d'entrée mais pas la profondeur (Figure 18).

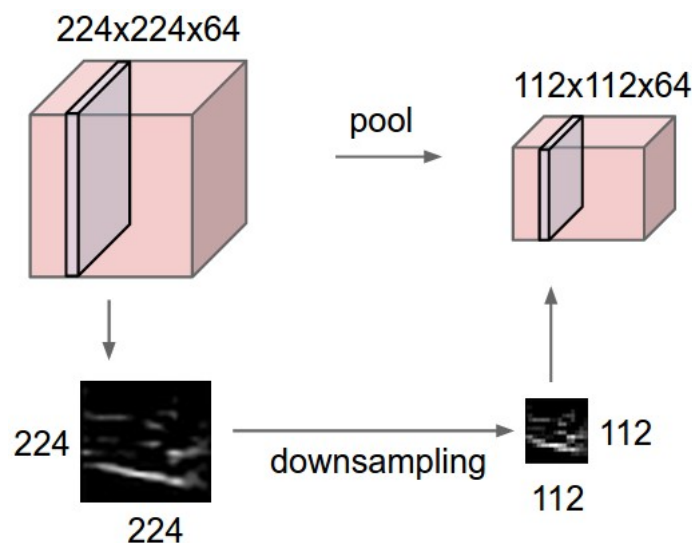


Figure 18 – Résultat d'un Pool sur une matrice 3D avec un filtre de taille 224x224 et de stride 2 [WWW16]

Les paramètres à choisir pour une couche le Pooling sont :

- Stride : Le pas de déplacement en pixel du filtre dans l'image, souvent 2 et rarement supérieur car trop destructeur.
- Les dimensions du filtre : la hauteur et la largeur du filtre, les Pooling avec des filtres plus grands que 2 ou 3 ne sont pas souvent utilisés car ils sont trop destructeurs.
- Le type de pooling : Généralement, c'est soit le maximum ou la moyenne.

#### 7.1.5 Batch Normalization Layer :

La couche de normalisation de groupe est une couche entraînable qui permet de fournir à n'importe quelle couche d'un réseau de neurones des entrées qui sont centrées autour de zéro et avec une variance unitaire. On utilise ces couches immédiatement après les couches entièrement connectées et avant les couches de non linéarité (ReLU par exemple). L'utilisation de cette couche permet d'accélérer l'apprentissage du réseau et parfois de retirer des couches de Dropout [5].



## 7.2 AlexNet

AlexNet est le réseau de neurones utilisé par SuperVision, le premier algorithme d'apprentissage profond à remporter le concours de classification organisé par ImageNet [WWW13].

Il est composé de 5 couches de convolution suivi de 3 couches entièrement connectées (Figure 19).

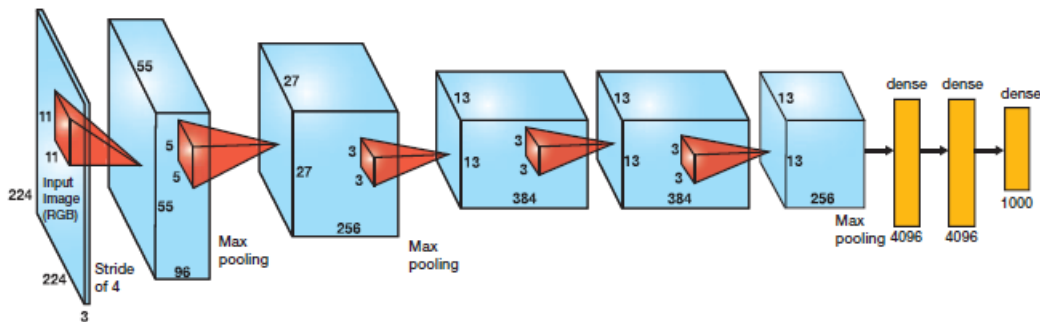


Figure 19 – Structure du réseau de neurones de AlexNet [WWW17]

AlexNet utilise ReLU comme fonction d'activation au lieu de Tanh ou Sigmoid qui étaient auparavant utilisées pour les réseaux de neurones traditionnels. Cela permet d'accélérer considérablement l'apprentissage [7] (Figure 20).

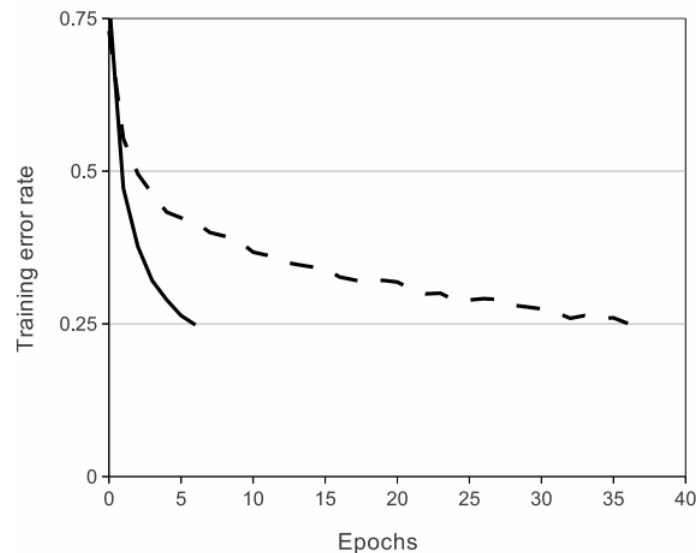


Figure 20 – Un réseau avec 4 couches de convolution atteint 25% de taux d'erreur 6 fois plus vite avec des couches de non linéarité ReLU qu'avec des couches de non linéarité Tanh.[7]

Dans le réseau, la couche de ReLU est placée après chaque couche de convolution et chaque couche entièrement connectée. Le surapprentissage a été réduit en utilisant une couche de Dropout après chaque couche entièrement connectée et en ajoutant 3 couches de Max Pooling.

### 7.3 VGG (Visual Geometry Group)

Ce réseau est une amélioration de AlexNet, il est plus profond car il contient plus de couches de convolution. La différence majeure avec AlexNet est que les couches de convolutions avec des filtres de grande taille (5x5 ou 7x7) sont remplacés par de multiples couches de convolution successives de filtres de petite taille (3x3) (Figure 21).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 21 – Les 6 configurations de ce réseau dans l'article original.[13]

Karen Simonyan et Andrew Zisserman ont montré qu'en diminuant la taille des filtres de convolution et avec un réseau plus profond, il était plus performant [13]. Ces réseaux ont permis à leur équipe d'obtenir la seconde place en classification et la première place en localisation au concours ImageNet de 2014.

## 7.4 ResNet

Les réseaux ResNet ont la particularité d'être très profonds. La Figure 22 compare 3 architectures différentes : le VGG-19, une architecture de 34 couches sans résidu et une architecture de 34 couches avec résidu.

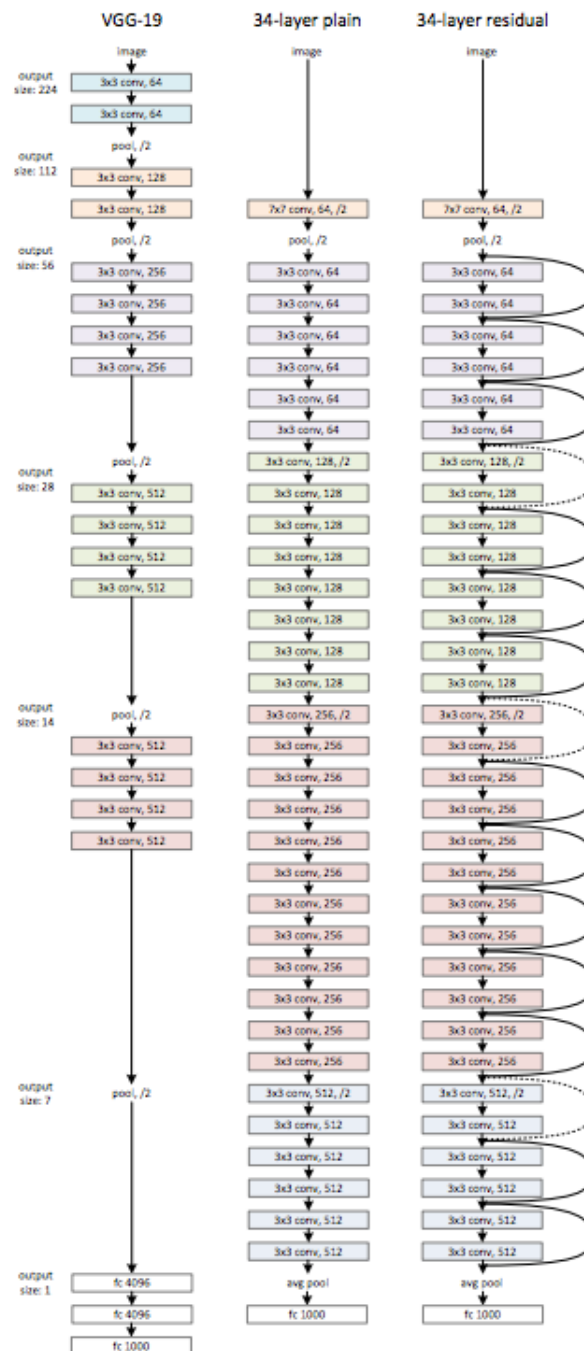


Figure 22 – Les 6 configurations de ce réseau dans l'article original.[3]

L'introduction d'un résidu permet de pallier le problème d'augmentation de l'erreur d'apprentissage au fur et à mesure que la profondeur du réseau augmente.

Les réseaux ResNet se sont révélés être très efficaces pour la détection et la classification d'objets dans des images. En 2015, les compétitions COCO et ImageNet ont été remportées par des

réseaux ResNet terminant à la première place pour les principales 5 catégories des compétitions [3].

## 7.5 Data augmentation

L'apprentissage profond demande beaucoup plus de données d'apprentissages que les autres types de machine learning. Si les échantillons d'apprentissages sont trop petits, il est courant d'avoir recours à des techniques de data augmentation. Pour les systèmes de reconnaissance d'image, on peut créer de nouvelles images en transformant les images de l'échantillon d'apprentissage.

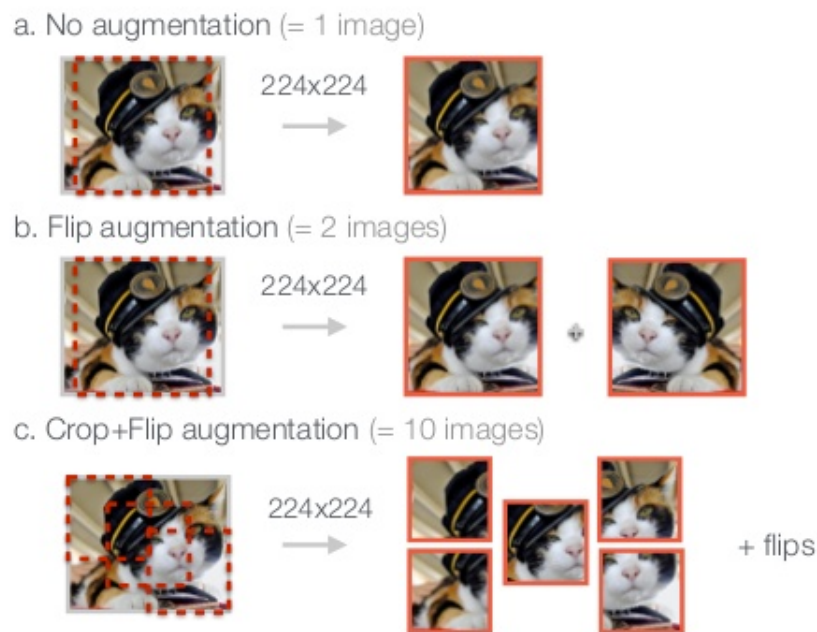


Figure 23 – 3 exemples d'opérations d'augmentation de données sur une image [WWW18]

## 7.6 Transfert learning

Une seconde manière d'entraîner un réseau de neurones profond lorsque l'on a un petit échantillon est de faire du transfert learning. Cela consiste à récupérer un réseau déjà entraîné puis de relancer une session d'apprentissage avec nos échantillons. Généralement, les couches les plus hautes du réseau sont complètement réinitialisées pour l'apprentissage. En effet les couches les plus profondes sont les moins spécialisées, comme vu précédemment, les caractéristiques de bas niveau correspondent toujours aux contours orientés. Ce sont souvent les mêmes quel que soit le problème de reconnaissance d'image. Plus on progresse dans le réseau, plus les caractéristiques sont spécialisées au problème de reconnaissance d'image. Il en va de même pour le classifieur.

## 8 Régression

En machine learning, on distingue deux types de problèmes : les problèmes de classification et les problèmes de régression. Jusqu'ici nous avons beaucoup parlé de classification. Les problèmes de classification consistent à prédire dans quelle classe appartient un vecteur d'entrée tandis que les problèmes de régression sont des problèmes où l'on cherche à prédire des valeurs

numériques en fonction du vecteur d'entrée.

Exemple de problème de classification :

On a une image d'un véhicule et on souhaite prédire si c'est une moto, une voiture ou un bus.

Exemple de problème de régression :

On cherche à prédire la valeur d'une maison en fonction de sa position géographique, sa surface, son ancienneté, etc.

Les problèmes de régression sont plutôt similaires aux problèmes de classification.

## 8.1 La régression linéaire

On cherche à établir une relation linéaire entre la variable de sortie  $y$ , variable dite expliquée, et une ou plusieurs variables d'entrée, dites explicatives, qui forme un vecteur  $\vec{x}$ . La variable de sortie peut être prédit comme une combinaison linéaire des variables d'entrée.

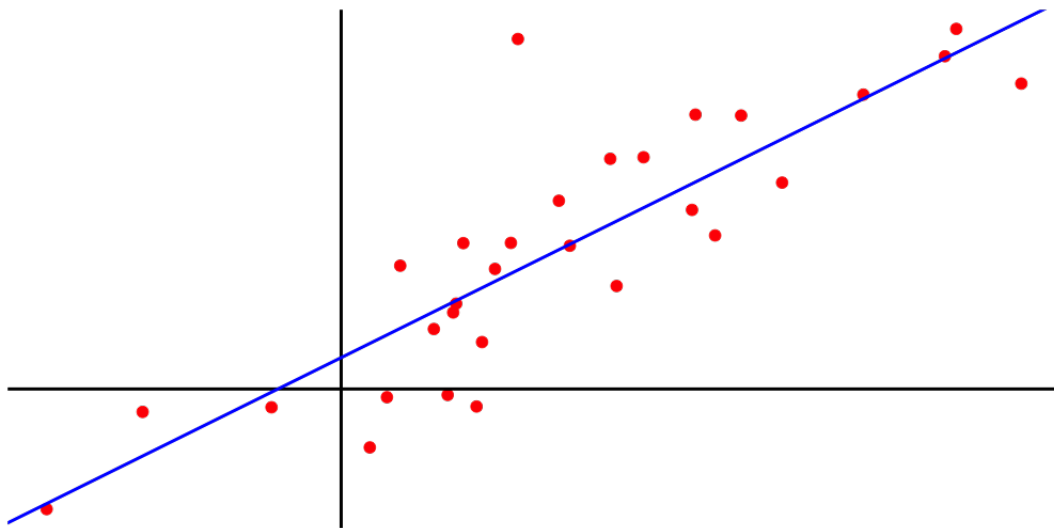


Figure 24 – Droite de régression linéaire calculée avec un ensemble de points

La droite de régression est la droite d'équation :

$$y = \vec{w} * \vec{x} + b$$

$\vec{w}$  est le vecteur qui permet de donner le coefficient directeur de la droite de régression.  $\vec{x}$  est le vecteur des valeurs pour chaque caractéristique pour une image donnée.  $b$  permet de transformer la fonction linéaire en une fonction affine.  $y$  est la sortie, la valeur que l'on cherche à prédire.

On veut estimer les poids  $\vec{w}$  pour pouvoir prédire une sortie  $y$  pour un vecteur d'entrée  $\vec{x}$  donné. Pour trouver ces poids, on cherche à minimiser la fonction de coût, qui est la moyenne des carrés des distances par rapport à la droite de régression.

Soit le vecteur  $\vec{a}$ , l'image en entrée représenté dans l'espace des caractéristiques. On calcule la sortie  $y$  pour cette image. A la différence du Perceptron, nous n'avons pas besoin de fonction d'activation car c'est la sortie numérique qui nous intéresse.

$$y = \vec{w} * \vec{a} + b$$

Comme nous pouvons le constater le problème est très similaire à celui du Perceptron. La différence est que l'on cherche une droite de régression et non un hyperplan séparateur, tout est la fonction de coût à minimiser.

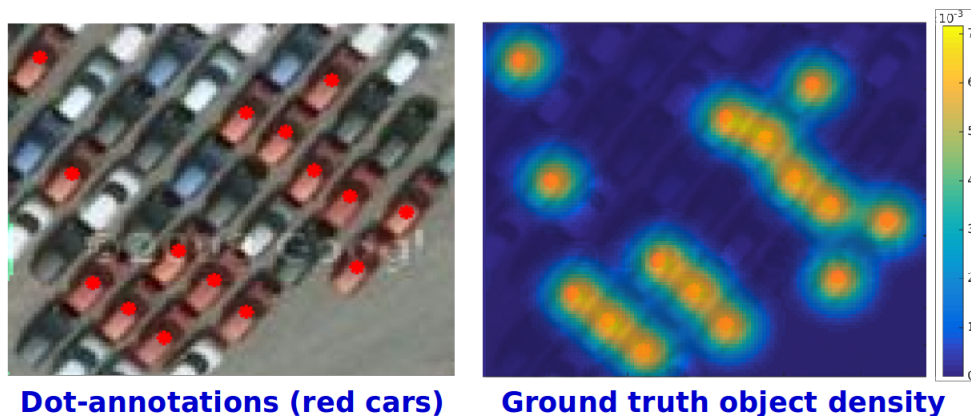
## 8.2 La régression non linéaire

De la même manière que la classification, avec des réseaux multicouches pour la classification, on peut trouver des droites de régression non linéaires. On cherche à minimiser la fonction de coût qui est la moyenne des distances au carré par rapport à la droite de régression. Comme avec les techniques de classification, on minimise la fonction à l'aide d'une méthode de descente de gradient stochastique.

## 2

## Apprendre à compter des objets dans une image

Cette partie aborde les méthodes de conception de système de reconnaissance d'image pour compter le nombre d'objets dans une image. Elle est inspirée de la publication "Learning To Count Objects in Images" de Victor Lempitsky et Andrew Zisserman [9]. Le but est d'estimer le nombre mais d'éviter la tâche difficile de détecter et de localiser l'objet. Pour cela on estime la densité d'une image où le nombre d'objets dans l'image correspond à l'intégrale de la densité de l'image.



**Figure 1** – Carte de densité du nombre de voiture dans l'image [WWW19]

Il y a plusieurs manières d'apprendre à un système à estimer le nombre d'objets dans une image. Soit en donnant uniquement le nombre d'objet dans l'image lors de l'apprentissage, soit en fournissant une liste de point dans l'image indiquant la position de chaque objet (un point par objet). Le premier cas demandera un plus grand échantillon d'apprentissage.

Il existe trois manières de compter les objets dans une image avec des méthodes d'apprentissage supervisé.

### 1 Compter par détection

Chaque objet doit être détecté et localisé dans l'image. On somme ensuite le nombre d'objets détectés dans l'image. Cela peut être accompli avec des réseaux convolutifs. Cela demande beaucoup de d'images d'apprentissage et c'est long à entraîner.

## 2 Compter par régression

Ces méthodes évitent le problème difficile qu'est la détection et la localisation des objets. On utilise directement les caractéristiques globales de l'image (l'histogramme des caractéristiques) pour estimer le nombre d'objets dans l'image.

## 3 Compter par segmentation

Compter par segmentation peut être vu comme un mélange des deux autres techniques. L'objectif est de segmenter les objets en plusieurs groupes et ensuite régresser depuis les propriétés globales de chaque groupe pour estimer le nombre d'objets dedans.



# 3

## Towards perspective-free object counting with deep learning

Cette étude [12] propose un système de reconnaissance d'image pour estimer le nombre de véhicules sur une route avec un trafic assez dense ou estimer le nombre de personnes dans une foule. Le système est composé d'un réseau convolutif et il a été renommé CCNN pour Counting Convolutional Neuronal Network. Le CCNN est un modèle de régression où le réseau apprend à projeter l'apparence de l'image dans une carte de densité.

L'échantillon d'apprentissage est un ensemble d'images annotées où chaque objet est marqué avec un point dans l'image. Les cartes de densités sont générées avec une fonction Gaussienne centrée sur chaque point.

Le réseau contient 6 couches de convolutions et aucune couche entièrement connectée (Figure 1).

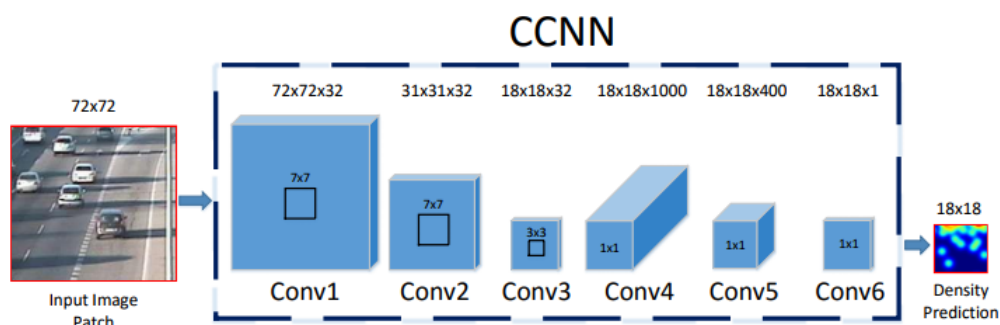


Figure 1 – Architecture du réseau de neurones CCNN [12]

Les couches Conv1 et Conv2 sont suivies d'une couche de Max Pooling de filtre de taille 2\*2. La couche Conv3 est suivie d'une couche de Max Pooling de filtre de taille 2\*2. Chaque couche de convolution est suivie par une couche de non linéarité ReLU. La couche de convolution Conv6 est en charge de retourner la carte de densité.

Le CCNN doit être entraîné pour résoudre un problème de régression. Pour faire cela, la couche Conv6 est connectée avec une fonction de coût de régression.

Contrairement à un autre article [14] au lieu d'utiliser des couches entièrement connectées les chercheurs ont utilisés 3 couches de convolutions de filtre de taille 1x1 (Conv4, Conv5, Conv6). Selon eux cela accélère l'apprentissage.

La taille de la carte de densité est d'un quart de la taille de l'image qui de 72x72 en entrée du réseau, c'est à dire 18x18 pixels. La carte est agrandie pour avoir la même taille que l'image originale. Cependant, lors de l'agrandissement, le nombre d'objets comptés est changé. La carte agrandie doit être normalisée.

# 4

## CrowdNet : A Deep Convolutional Network for Dense Crowd Counting

Cette étude [1] cherche à estimer le nombre de personne dans une image d'une dense foule. Elle utilise un réseau de neurones convolutif (CNN) pour prédire la carte de densité de l'image puis en déduire le nombre de personne.

Le réseau convolutif est utilisé pour extraire des caractéristiques de bas niveau (tâches, formes flous) et de haut niveau (visage, corps). Ces deux types de caractéristiques sont nécessaires pour compter à des échelles différentes, c'est-à-dire détecter les personnes au premier plan et ceux à l'arrière-plan.

Pour extraire ces deux différentes caractéristiques, le modèle utilise une combinaison d'un réseau de neurones convolutif peu profond et d'un autre plus profond (Figure 1).

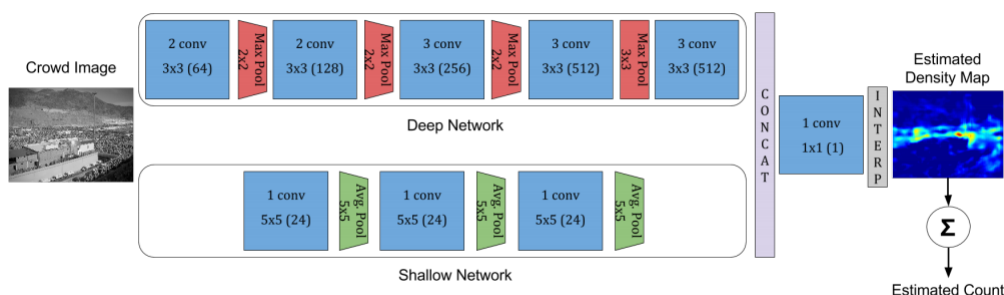


Figure 1 – Architecture du réseau de neurones CrowdNet [1]

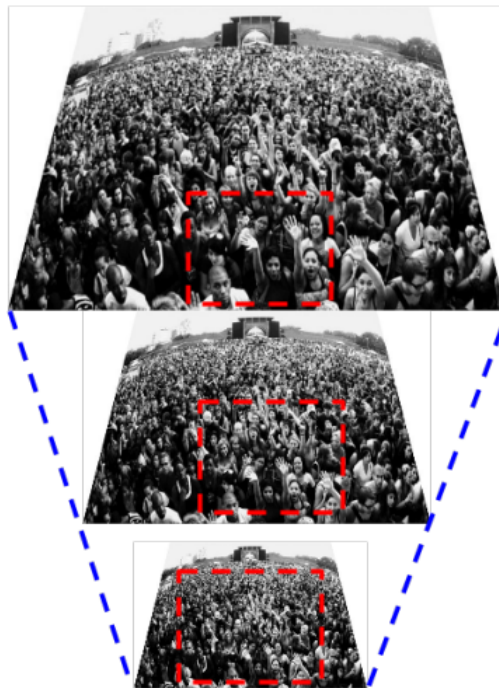
### 1 Le réseau profond

Le réseau profond est conçu pour extraire les caractéristiques de haut niveau en utilisant une architecture de réseau VGG-16. Bien que l'architecture VGG-16 a originellement été entraînée dans le but de faire de la classification d'objets, elle a plein d'application dans la reconnaissance de formes comme la prédiction et la segmentation d'objet. Cependant, il faut supprimer les 3 couches entièrement connectées à la fin car l'objectif n'est pas de faire de la classification mais de prédire une carte de densité. Le réseau possède 5 couches max-pool intercalées avec 2 couches de convolution.

## 2 Le réseau peu profond

Le réseau peu profond a seulement besoin de 3 couches de convolution avec de plus gros filtres (5x5) intercalées avec une couche de « Average Pooling » pour détecter les tâches correspondant aux têtes en arrière-plan.

Les prédictions des deux réseaux sont concaténées puis passées dans une couche de convolution avec un seul filtre de taille 1x1. La sortie de cette couche est agrandie à l'aide d'une interpolation bilinéaire pour obtenir la carte de densité. En effet lorsqu'une image est transformée (agrandissement, rotation, etc), la position des pixels dans la nouvelle image ne tombe pas toujours juste. L'interpolation linéaire est l'algorithme qui permet de déterminer la valeur de chaque pixel de l'image agrandie.

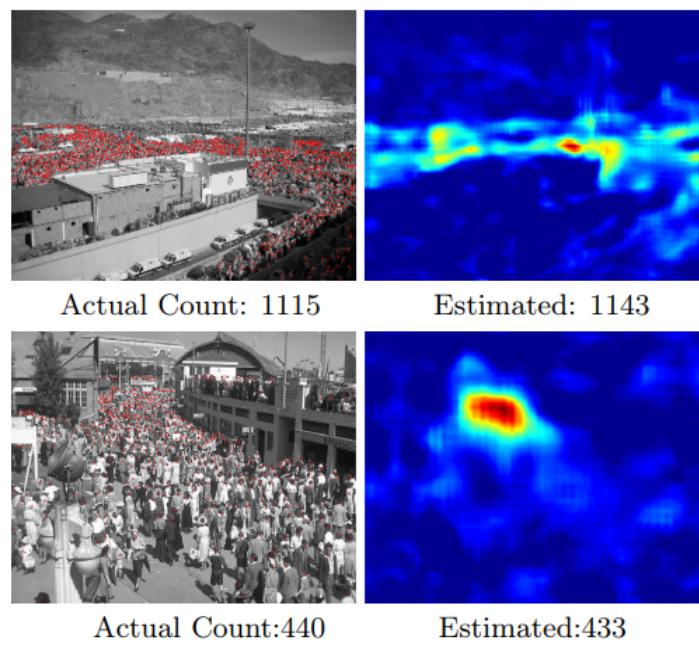


**Figure 2** – Les images subissent des transformations pour rendre le réseau robuste au changement d'échelle [1]

Comme les échantillons d'images de foules annotées sont souvent limités (moins de 100 images) et que les approches d'apprentissage profond sont basées sur une grande quantité de données, il faut augmenter le nombre de données d'apprentissage. Pour rendre le système robuste au changement d'échelle, chaque image passée en entrée est envoyée dans le réseau de neurone à l'échelle 0,5 jusqu'à 1,2 (Figure 2).

## 3 L'apprentissage

Dans les images d'apprentissage les têtes sont annotées avec un point. Comme le réseau doit prédire la carte de densité avant de prédire le nombre de personnes dans l'image, on génère la carte de densité cible à partir de l'image annotée. Chaque annotation est transformée en densité en utilisant l'estimation par noyau avec une fonction gaussienne. Pour chaque annotation, la somme de la densité est égale à 1, de cette manière la somme de la carte de densité est égale au nombre de personne dans l'image.



**Figure 3** – Crowd estime le nombre de personnes dans l'image en générant une carte de densité [1]

# 5

## Using Deep Learning for Segmentation and Counting within Microscopy Data

L'objectif de cette étude [4] est d'automatiser l'estimation du nombre de cellules dans des images de microscope. La méthodologie de l'étude se divise en deux parties :

- La segmentation des cellules
- L'estimation du nombre de cellules

### 1 La segmentation des cellules

La segmentation des cellules est achevée en générant un Mask capable d'identifier une cellule dans une image. L'algorithme Mask-RCNN conçu par Facebook AI Research (FAIR) utilise un Feature Pyramid Network (FPN) pour générer un masque des régions d'intérêts dans une image [2].

Les FPN (Figure 1) sont des réseaux conçus pour extraire des caractéristiques à différentes échelles [11].

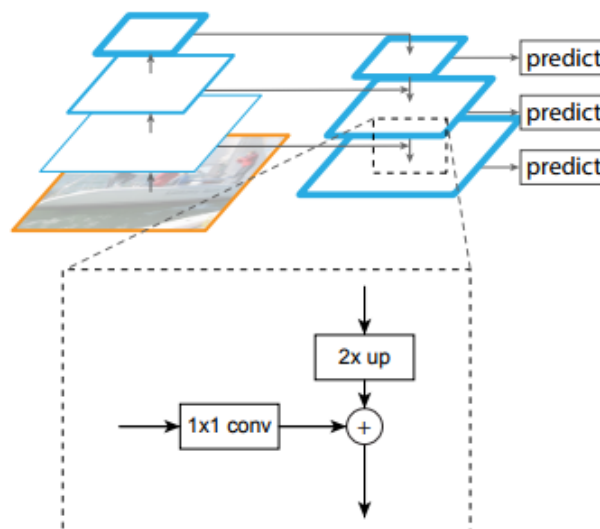
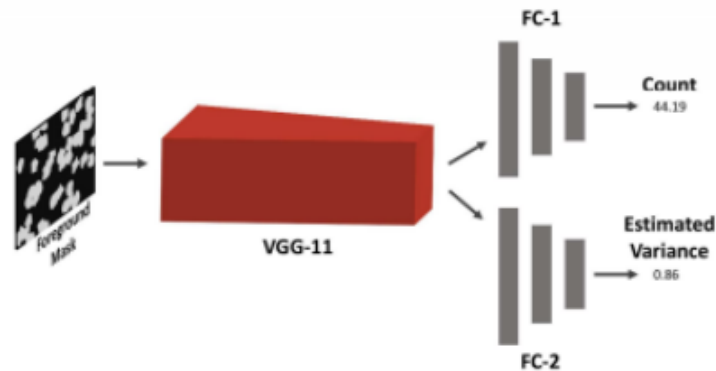


Figure 1 – Feature Pyramid Network [11]

Dans cette étude, l'image d'entrée est transformée pour obtenir 4 images sur 4 échelles différentes avec un facteur de 0.5 entre chaque échelle.

## 2 Estimation du nombre de cellules dans une image

Les chercheurs ont utilisé le réseau VGG avec 11 couches de convolution avec un filtre de 3x3 (Figure 2). Le nombre de filtres varie de 64 à 512 au fur et à mesure que le réseau progresse. Chaque couche de convolution est suivie par une couche de normalisation de groupe (batch normalization layer) pour accélérer l'apprentissage en évitant au gradient d'exploser ou d'atteindre zéro. Après cette couche on ajoute une couche de non linéarité puis une couche de Max Pooling. Ce réseau sert d'extracteur de caractéristiques dans le système d'apprentissage. Pour prédire le nombre de cellules on utilise 3 couches entièrement connectées de dimensions 1024x512x1 séparées par une couche de normalisation de groupe et une couche de non linéarité ReLU. Pour éviter d'afficher des valeurs négatives une couche de ReLU a été ajoutée après la dernière couche. On utilise la même structure pour prédire la variance estimée à la différence qu'il n'y a pas de couche ReLU après la dernière couche entièrement connectée.



**Figure 2** – Architecture du réseau de neurones permettant de compter les cellules dans des images de microscope [4]

L'échantillon d'apprentissage est une collection de 9,600 images simulées. Le FPN a été entraîné sur 600 images et le réseau VGG sur toutes.

**Troisième partie**

**Analyse et conception**





# Introduction

Cette analyse propose un système de reconnaissance d'images utilisant des techniques d'apprentissage profond pour estimer le nombre de poissons dans une image aérienne. Elle se base sur les recherches effectuées dans l'état de l'art. Même si la solution proposée peut évoluer au cours des prochains mois en fonctions des résultats et de l'avancement du projet, elle pose les bases structurelles du système.

# 1

## Architecture du système de reconnaissance d'image

Le cahier de spécification indique que le système développé doit être entraîné avec un ensemble d'images d'exemples sur lesquelles sont indiquées le nombre de poissons. C'est une méthode d'apprentissage supervisé.

### 1 Data augmentation

L'apprentissage profond demande beaucoup plus de données d'apprentissages que les autres types de machine learning.

L'augmentation du nombre de données d'apprentissage a deux avantages ; premièrement, il permet dans un premier temps d'augmenter la taille de l'échantillon lorsqu'il est trop petit pour le réseau de neurones. Deuxièmement, il peut permettre au réseau d'être robuste au changement d'échelle.

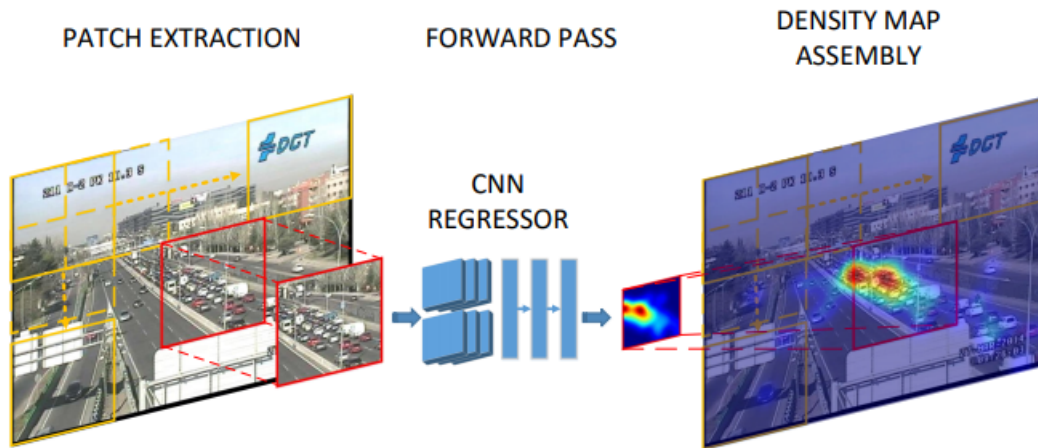
Le MOA a spécifié qu'il y avait environ 20 000 images, mais très peu contiennent des poissons dedans. De nouvelles images seront créées à partir de celles qui contiennent des poissons. Cela peut être fait à l'aide d'opération simple comme des zooms, des rotations, etc.

### 2 Prétraitements

Le nombre d'entrées dans un réseau de neurones est toujours fixe tandis que la taille de l'image que l'on veut analyser est variable. Pour remédier à ce problème, l'entrée du réseau de neurone est un carré de petite taille, 72x72 pixels dans notre cas. L'image d'entrée est segmentée en carré de 72x72 pixel, puis chaque morceau est envoyé dans le réseau un par un (**Figure 1**).

### 3 transfert learning

Si l'augmentation des données d'apprentissage ne suffit pas pour entraîner le réseau de neurones, la MOE intégrera les poids des couches de bas niveau d'un réseau déjà entraîné sur ImageNet. Avec cette technique, notre réseau aura déjà appris l'extraction des caractéristiques de bas niveau (contours orientés) avant même de commencer l'apprentissage.



**Figure 1** – L'image est segmentée pour être passée dans le réseau de neurone à la fin la carte de densité finale est recomposée. [12]

## 4 Réseau de neurone

Compter par détection n'est pas adapté pour ce projet. Le problème avec ce type de technique est que cela demande un apprentissage avec des milliers d'échantillons pour pouvoir extraire des caractéristiques de très haut niveau, ce dont nous ne disposons pas. De plus, les images aériennes fournissent des plans larges où les poissons apparaissent comme des taches au milieu de l'eau, nous n'avons pas besoins d'extraire des caractéristiques de très haut niveau pour pouvoir estimer le nombre de poissons dans l'image.

L'approche retenue dans l'état de l'art est la technique de comptage par segmentation car la régression ne permet pas à elle seule de compter le nombre d'objets dans une image.

Tout d'abord les premières couches du réseau permettent d'extraire les caractéristiques de bas et de moyen niveau. Ensuite, les couches qui suivent vont permettre au système d'apprendre avec la régression non linéaire à créer une carte de densité de poissons à partir des caractéristiques de moyen niveau trouvées dans l'image. Pour ce faire, la MOE a choisi de mettre en place une architecture proche du CCNN. Cependant, pour se rapprocher du modèle VGG, le réseau contiendra de multiples couches de convolution avec des filtres de convolution plus petits (3x3) plutôt qu'une seule couche avec un grand filtre (7x7) comme dans le CCNN.

## 5 Apprentissage séquentiel ou par cycle

Le réseau sera entraîné avec un mode d'apprentissage séquentiel, c'est-à-dire que les poids du réseau seront mis à jour à jour à chaque exemple appris.

## 6 Ordre de présentation des exemples

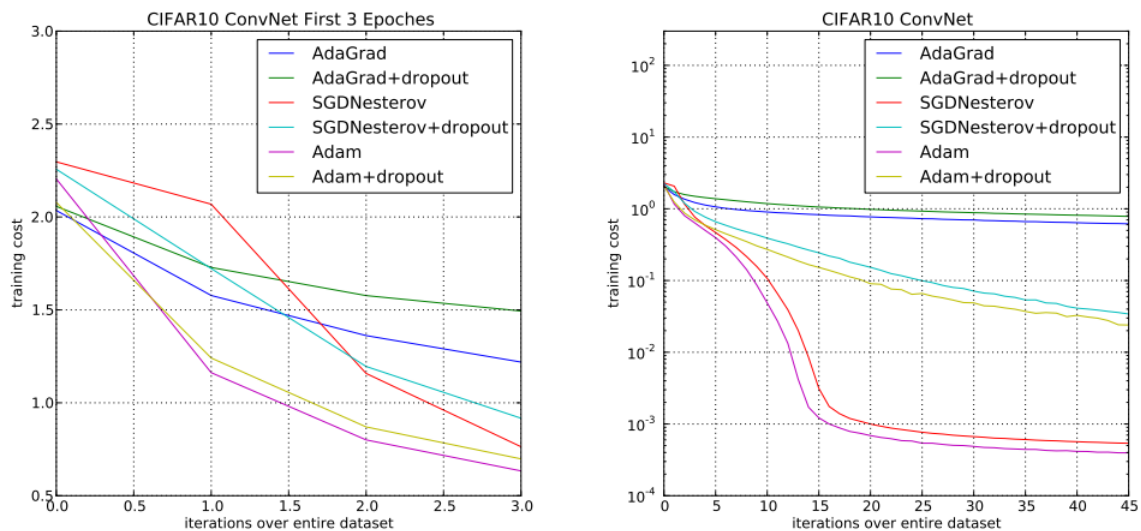
On présente les exemples de manière aléatoire pour avoir un apprentissage équilibré, il ne s'agit pas d'un problème de classification mais les images qui se suivent dans l'échantillon d'apprentissage se ressemblent beaucoup.

## 7 Fonction d'activation / couche de non linéarité

La couche ReLU (Rectified Linear Unit) est sans aucun doute la couche de non linéarité à privilégier pour les réseaux convolutifs.

## 8 L'algorithme de mise à jour des poids

L'algorithme à utiliser pour minimiser la fonction de coût. Nous allons utiliser Adam dont le coût d'apprentissage est inférieur à l'algorithme de descente de gradient classique SGD pour les réseaux convolutifs [6] (Figure 2).



**Figure 2** – Evolution du coût d'apprentissage de différents algorithmes de descente de gradient sur les 3 premières itérations (à gauche) et sur les 45 premières itérations (à droite). Les résultats ont été obtenus avec un réseau convolutif trois couches de convolution 5x5 intercalés avec une couche de Max Pooling et une couche de non linéarité ReLU. [6]

## 9 Les hyperparamètres

Les hyperparamètres sont les paramètres à définir pour l'algorithme de descente de gradient par exemple le pas d'apprentissage. Pour l'algorithme Adam, nous choisirons les hyperparamètres par défaut, c'est-à-dire ceux conseillés dans l'article original [6].

$\alpha = 0.001$  (le pas d'apprentissage)  $\beta_1 = 0.9$   $\beta_2 = 0.999$   $\epsilon = 10^{-8}$

## 10 La fonction de coût

La fonction représentant la différence entre la sortie produite et la sortie désirée. C'est la fonction que l'on cherche à minimiser avec l'algorithme Adam. Comme notre problème est un problème de régression, la moyenne des carrés des distances par rapport à la courbe de régression est une fonction de coût adaptée [WWW20].

## 11 Elagage

En cas de surapprentissage du réseau de neurones, la MOE devra peut-être ajouter des couches de Dropout pour supprimer des nœuds au cours de l'apprentissage.

## 12 Vecteur de sortie

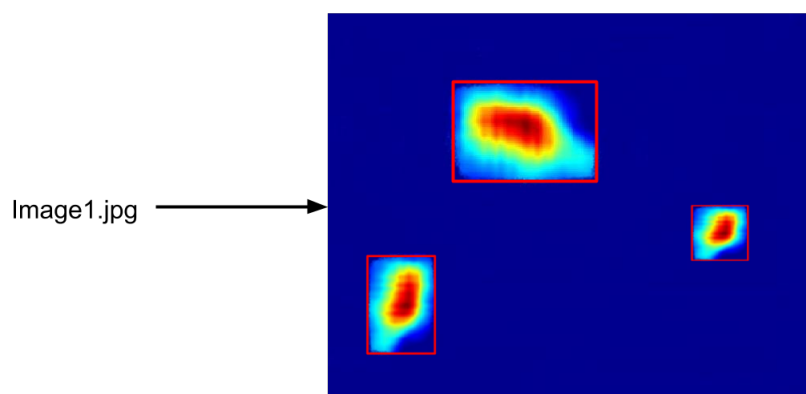
Comme cela a été précédemment introduit, le système prédit une carte de densité des poissons dans l'image d'entrée de 72x72 pixels. Cette carte de densité mesure aussi 72x72 pixels.

## 13 Post-traitements

Quand tous les morceaux de l'image sont passés dans le réseau, les carte de densité pour chaque morceau sont réunies pour former l'estimation finale. La carte de densité de l'image complète est recomposée et le nombre de poissons dans l'image est obtenu en faisant la somme des valeurs de densité sur chaque pixel.

## 14 Prétraitement sur les images d'apprentissage

Le réseau est construit pour prédire une carte de densité. Pour être entraîné il faut donc, pour chaque image d'apprentissage, la carte de densité qui lui correspond. La transformation de l'image d'entrée en carte de densité est plutôt simple. Comme on l'a vu dans le cahier de spécification, les images sont annotées avec des rectangles qui contiennent un certain nombre de poissons. Pour générer la carte de densité d'une image, pour chaque rectangle, on va générer une densité à l'aide d'une fonction gaussienne de façon à ce que la somme des valeurs des densités pour chaque pixel soit égale au nombre de poissons dans le rectangle. Les pixels qui ne sont pas dans les rectangles ont une valeur de densité nulle (Figure 3).



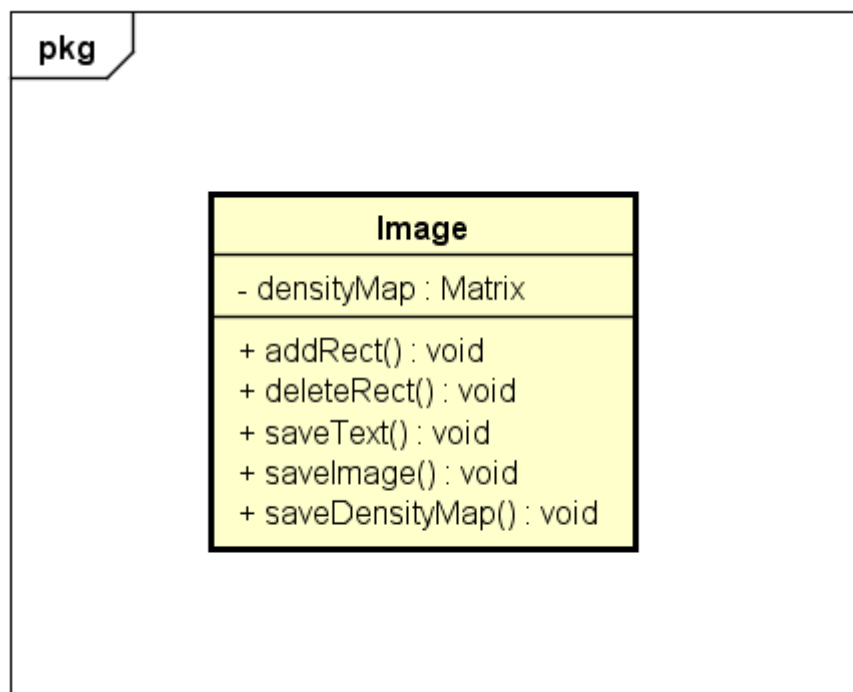
**Figure 3** – Les images d'apprentissage sont transformées en carte de densité

Tout comme les images en mode prédiction, la carte de densité est segmentée en morceau de la taille de l'entrée du réseau de neurones pour avoir une carte de densité cible. C'est exactement la technique qui a été utilisée pour le CCNN.

# 2

## Structure du code

La génération d'une carte de densité par image n'était pas prévue initialement. Cependant, il est intéressant que cette information supplémentaire puisse être utilisée. Il faut donc modifier la structure de la classe Image pour qu'elle intègre un nouvel attribut correspondant à la carte de densité. Cet attribut est une matrice de *float* de deux dimensions. La méthode *saveDensityMap()* a aussi été intégrée. Elle permet de sauvegarder dans un fichier la carte de densité (Figure 1).



powered by Astah

**Figure 1** – L'attribut "densityMap" et la méthode "saveDensityMap" sont ajoutés à la classe Image

## Quatrième partie

# Mise en œuvre

Cette partie présente la mise en œuvre du projet. Tout d'abord l'algorithme implémenté sera décrit ainsi que les outils utilisés pour son implémentation. Ensuite nous verrons quel dataset a été utilisé pour entraîner et tester le programme. Pour finir les résultats obtenus avec le système de reconnaissance de formes seront présentés.

# 1

## Implémentation

Dans cette partie nous verrons les outils utilisés pour l'implémentation, les choix de modélisation de notre système de reconnaissance de formes puis les choix de modélisation du code.

### 1 Les outils

Comme présenté dans les spécifications, le langage de programmation utilisé est Python. Concernant le système de reconnaissance de formes c'est la librairie Keras qui était imposée.

### 2 Le système de reconnaissance de formes

Pour notre système de reconnaissance de formes nous avons choisi d'implémenter le CCNN présenté dans le chapitre [Chapitre 3 - Towards perspective-free object counting with deep learning](#). Pour rappel, ce chapitre présente un article [12] qui présente un système de reconnaissance de formes utilisant l'apprentissage profond pour compter des objets dans une image en utilisant des cartes de densité.

Le système de reconnaissance de formes implémenté suit exactement le même fonctionnement que celui présenté dans l'article. Il existe tout de même quelques différences. Tout d'abord les images d'apprentissage et de test que nous utilisons sont annotées avec des rectangles contenant un certain nombre de poissons. Dans l'article, sur les trois datasets utilisés (Transcos, UCSD, UCF\_CC\_50) les trois utilisent des images annotées par des points. C'est-à-dire que dans une image chaque objet est annoté par un point situé en son centre. La génération de la carte de densité ne se fait donc pas de la même manière que dans notre cas. De plus, dans leur article, les auteurs ont implémenté des versions "Hydra" du CCNN pour que le système apprenne à appréhender le changement de perspective. Dans notre cas, par manque de temps cette fonctionnalité n'a pas été implémentée.

Les parties suivantes décrivent le fonctionnement de notre système de reconnaissance de formes.

#### 2.1 Le réseau de neurones

Le réseau de neurones utilisé est le même que celui modélisé dans l'article.

Voici les différentes couches de ce réseau de neurones :



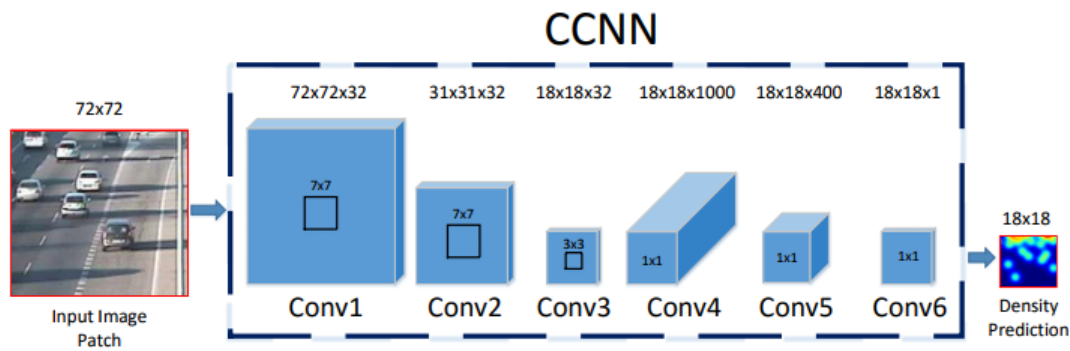


Figure 1 – CNN

- Couche d'entrée : une matrice de dimensions 72x72x3
- Couche de convolution 2D : 32 filtres de taille 7x7, stride 1x1, padding 3x3, un biais initialisé à 0, fonction d'activation ReLU.
- Couche de Max Pooling 2D de taille 2x2 : stride 2x2, pas de padding.
- Couche de convolution 2D : 32 filtres de taille 7x7, stride 1x1, padding 3x3, un biais initialisé à 0, fonction d'activation ReLU.
- Couche de Max Pooling 2D de taille 2x2 : stride 2x2, pas de padding.
- Couche de convolution 2D : 64 filtres de taille 5x5, stride 1x1, padding 2x2, un biais initialisé à 0, fonction d'activation ReLU.
- Couche de convolution 2D : 1000 filtres de taille 1x1, stride 1x1, pas de padding, un biais initialisé à 0, fonction d'activation ReLU.
- Couche de convolution 2D : 400 filtres de taille 1x1, stride 1x1, pas de padding, un biais initialisé à 0, fonction d'activation ReLU.
- Couche de convolution 2D : 1 filtres de taille 1x1, stride 1x1, pas de padding, un biais initialisé à 0, fonction d'activation ReLU.
- Sortie : une matrice de dimensions 18x18x1

Ce réseau de neurones permet de passer en entrée une image en couleur (RGB) en entrée et d'obtenir en sortie une carte de densité noir et blanc. En faisant la somme de chaque valeur sur chaque pixel on obtient le nombre estimé d'objets dans l'image.

Pour compiler ce réseau de neurones nous avons définie la Mean Squared Error comme fonction de coût comme cela est indiqué dans l'article. Pour l'algorithme de mise à jour des poids (optimizer) c'est l'algorithme Stochastique Gradient Descent (SGD) qui est utilisé dans l'article. Cependant la MOE a choisie d'utiliser un algorithme adaptatif qui converge plus vite : l'algorithme Adam.

## 2.2 Les prétraitements et post-traitements

Le réseau de neurones en lui-même n'est pas suffisant pour calculer une carte de densité d'une image car il prend en entrée une image RGB d'une certaine taille (72x72). Les prétraitements sur les images permettent de définir une stratégie pour calculer la carte de densité d'une image quelconque. Pour cette partie nous avons utilisé la même stratégie que dans l'article.

### Découpage de l'image en patch

Pour prédire la carte de densité d'une image, on va tout d'abord découper celle-ci en patch. On commence par extraire le patch dans le coin en haut à gauche et chaque nouveau patch on se

décale d'un certain nombre de pixel ("stride") à droite. Un fois la ligne de patch fini on revient au début et on se décale d'un certain nombre de pixel en bas.

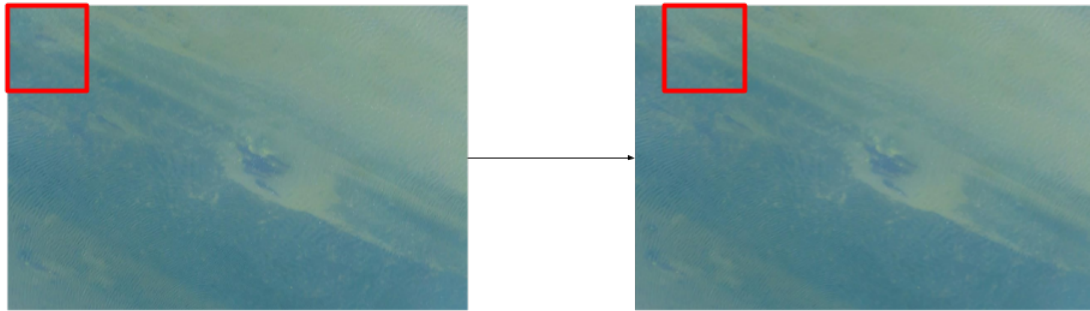


Figure 2 – Le découpage de l'image en patch

La taille des patches n'est pas nécessairement 72x72 contrairement à ce qu'on pourrait penser. Généralement il sont plus grands que ça. Les patches doivent être suffisamment grands pour contenir plusieurs objets. Ensuite on redimensionne la taille de chaque patch pour qu'elle corresponde à la taille du réseau de neurones c'est à dire 72 par 72 pixels. Durant ce processus il y a forcément une perte de qualité à chaque patch. Pour vous donner un exemple, sur le dataset Transcos, les auteurs de l'article ont choisi des patches de taille 115 par 115 pixels.

### Prédiction des patches

Une fois le redimensionnement opéré, les patches peuvent être envoyés dans le réseau de neurones. Pour chaque patch le réseau de neurones va retourner une carte de densité de taille 18 par 18 pixels. Ces cartes sont redimensionner pour mesurer la même taille que les patches de base. Par exemple pour le dataset Transcos, les auteurs les redimensionne en carte de densité de 115 par 115 pixels. Le redimensionnement est fait de manière à conserver la somme des valeurs de chaque pixel de la carte. De cette manière la prédiction du nombre d'objet dans chaque patch est conservée même avec le redimensionnement.

### Reconstituer la carte de densité

Il faut maintenant reconstituer une carte de densité totale de la même taille que l'image de base. Tout d'abord, pour chaque patch prédit il faut supprimer les valeurs négatives en les remplaçant par des zéros. Comme les patches se superposent, pour chaque pixel de la carte de densité totale, on moyenne les valeurs trouvées pour chaque patches ayant fait une prédiction sur ce pixel. Pour exemple imaginons une image de 3x3 pixels. Si on fait une prédiction avec un patch de taille 2x2 pixels, on peut calculer pour chaque pixels le nombre de prédiction qu'on va faire. (Figure 3)

Si on a cette matrice, il suffit de construire notre carte de densité en créant une matrice de zéros de la taille de l'image de base et à chaque patch de carte de densité d'ajouter les valeurs aux pixels correspondants. La carte de densité est à la fin divisée par la matrice qui contient le nombre de prédiction par pixels ce qui moyenne les prédictions par pixels.

Avec cette carte de densité on peut faire la somme des valeurs de chaque pixels pour estimer le nombre d'objets dans l'image.

## 2.3 Entraînement du réseau de neurones

Maintenant que l'on sait comment on va prédire une carte de densité pour une image donnée, il faut définir une stratégie pour entraîner notre réseau de neurones.

1	2	1
2	4	2
1	2	1

**Figure 3** – Pour chaque pixel on a un certain nombre de prédiction

### Générer les cartes de densités des images annotées

On sait que nos images d'apprentissage et de test sont annotées avec des rectangles dans lesquels il y a un certain nombre de poissons. On veut donc générer une carte de densité pour chaque image d'apprentissage en fonction des rectangles et du nombre de poissons dans chaque rectangle. Pour faire cela on va créer une matrice de zéros de la taille de l'image. Puis on va "colorier" à l'endroit des rectangles. Pour chaque pixel dans un rectangle on attribut la valeur :

$$\frac{\text{nombre de poissons}}{\text{nombre de pixels dans le rectangle}}$$

De cette manière si on fait la somme de tout les pixels on obtient le nombre de poissons dans l'image.

On obtient donc une carte de densité mais le problème est qu'elle est trop homogène dans les rectangles. On aimerait que les valeurs soit plus élevée au centre des rectangle puis diminue au fur et à mesure qu'on s'éloigne du rectangle. C'est exactement ce que fait le filtre gaussien en traitement d'image. De plus en faisant cette opération on conserve la somme des valeurs des pixels dans l'image.

### Générer les patches d'apprentissage

De la même manière que pour le mode prédiction on veut découper l'image en patches. Sauf que cette fois-ci on va tout simplement choisir la position des patches aléatoirement. Par exemple pour le dataset Transcos, les auteurs on choisis d'extraire 800 patches d'une taille de 115 par 115 pixels aléatoirement dans chaque image. Pour chaque patch on extrait aussi le patch de la carte de densité correspondant.

Avant de pouvoir l'envoyer dans le réseau de neurones il faut redimensionner les patches. Les patches de l'image sont redimensionnés en patch de 72 par 72 pixels. Les patches de la carte de densité sont redimensionnés en patch de 18 par 18 mais en conservant la somme des valeurs sur chaque pixel.

### Entraîner le réseau

On génère tout ces patches pour toutes les images et on mélange l'ordre dans lequel ils seront présentés au réseau de neurones.

On a maintenant tout les éléments pour lancer l'apprentissage du réseau de neurones.

## 2.4 Transfer learning

Comme le nombre d'image d'apprentissage du dataset des poissons est assez réduit, la MOE a choisi de se baser sur un réseau de neurones déjà pré-entraîné. Les auteurs de l'article mettent à disposition les poids des réseaux de neurones entraînés pour chaque dataset.

Seul le dataset Transcos est en couleur sur les trois, comme nos images sont aussi en couleur la MOE a utilisé les poids de ce dataset car c'était les seuls exploitables.

La MOE a choisi de ne laisser entraînable que les deux dernières couches du réseau de neurones. Cela veut dire qu'à chaque itération, seul les poids des deux dernières couches peuvent changer.

Cela peut paraître être peu mais ces deux couches composent la majorité des poids du réseau de neurones. En tout on a 400801 poids entraînaables et seulement 171208 non entraînaables. (Figure 4)

Layer (type)	Output Shape	Param #
zero_padding2d_1 (ZeroPaddin	(None, 78, 78, 3)	0
conv1 (Conv2D)	(None, 72, 72, 32)	4736
pool1 (MaxPooling2D)	(None, 36, 36, 32)	0
zero_padding2d_2 (ZeroPaddin	(None, 42, 42, 32)	0
conv2 (Conv2D)	(None, 36, 36, 32)	50208
pool2 (MaxPooling2D)	(None, 18, 18, 32)	0
zero_padding2d_3 (ZeroPaddin	(None, 22, 22, 32)	0
conv3 (Conv2D)	(None, 18, 18, 64)	51264
conv4 (Conv2D)	(None, 18, 18, 1000)	65000
conv5 (Conv2D)	(None, 18, 18, 400)	400400
conv6 (Conv2D)	(None, 18, 18, 1)	401
Total params: 572,009		
Trainable params: 400,801		
Non-trainable params: 171,208		

Figure 4 – Le nombre de paramètres par couche du réseau de neurones

## 3 Diagramme de classe

Cette partie présente les choix de modélisation du code.

Le code source du système est décomposé en deux packages : ui et ccnn.

### 3.1 Le package ui

Le package ui (Figure 5) contient le code de l'IHM du système qui a été développé lors d'un autre projet de recherche et développement réalisé par Yaofutian LU. Ce package contient aussi des classes que la MOE a développé, notamment les classes Algorithme et AlgorithmeCCNN.

Pour pouvoir intégrer facilement de nouveau algorithme dans l'interface la MOA a proposée d'utiliser le design pattern Strategy.

- L'interface *Algorithme*. Cette interface générique contient une méthode *process* qui doit être implémentée.
- La classe *AlgorithmeCCNN* implémente cette interface ainsi que la méthode *process*. Cette méthode implémente la prédiction du nombre de poissons dans une image comme expliqué dans la partie [Section 2.2 - Les prétraitements et post-traitements](#).

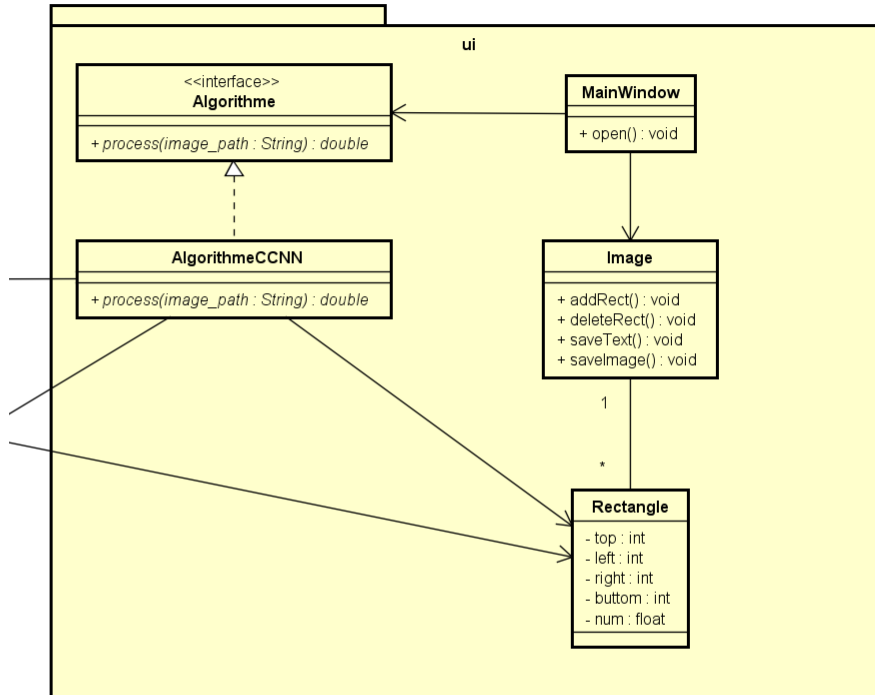


Figure 5 – Diagramme de classe package ui

### 3.2 Le package ccnn

Ce package ([Figure 6](#)) contient tout ce qu'il faut pour entraîner le réseau de neurones.

- Le script Python *gen\_features.py* permet de générer les cartes de densités et les patches comme décrits dans les paragraphes [Section 2.3 - Générer les cartes de densités des images annotées](#) et [Section 2.3 - Générer les patches d'apprentissage](#). Les patches générés sont enregistrés dans un fichier au format hdf5.
- Le script *train.py* permet de lancer un entraînement du réseau de neurones avec les patches générés avec le script *gen\_features.py*.
- Le script *test.py* permet de tester le réseau de neurones sur les images d'entraînement et de test. Pour chaque image il calcul le nombre de poissons et enregistre la carte de densité. A la fin il calcul la Mean Absolute Error et la Mean Squared Error entre les valeurs prédites et les vraies valeurs sur le dataset d'entraînement et le dataset de test. Tous les résultats sont enregistrés dans des fichiers textes.

Pour plus d'information concernant l'utilisation de ces scripts voir le manuel d'utilisation [Annexe F - Manuel d'utilisation](#).

La classe *CcnnModel* permet de construire le modèle du réseau de neurones. Le constructeur prend un fichier de poids et la méthode *build\_model()* permet de déclarer toutes les couches du réseau de neurones.

Le fichier *ccnn\_utils.py* contient toutes les fonctions de traitement des données qui sont utiles dans a peu près tous les fichiers python.

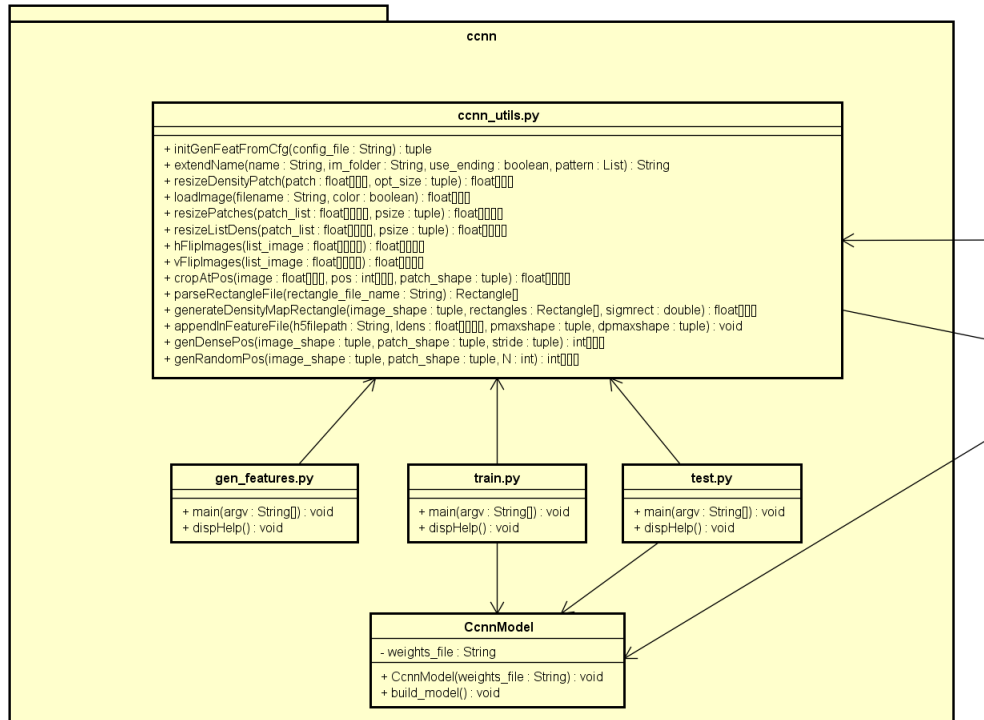


Figure 6 – Diagramme de classe package ui

Quelques-unes d'entre-elles ont pu être récupérées du Github[[WWW21](#)] mis à disposition par les auteurs de l'article.

Les fonctions récupérées :

- `resizeDensityPatch`
- `loadImage`
- `resizePatches`
- `resizeListDens`

Les fonctions modifiées :

- `initGenFeatFromCfg`
- `extendName`
- `cropAtPos`
- `genRandomPos`

Récupérer ces fonctions a permis de gagner beaucoup du temps dans le développement du système.

### 3.3 Dépendances entre les deux packages

La classe *AlgorithmeCCNN* se sert de fonctions contenus dans le fichier *ccnn\_utils.py* et du model de réseau de neurones à l'aide de la classe *CcnnModel*.

De plus, la classe *Rectangle*, initialement développée dans l'autre PRD, est utilisée dans le *ccnn\_utils.py* et par la classe *AlgorithmeCCNN*.

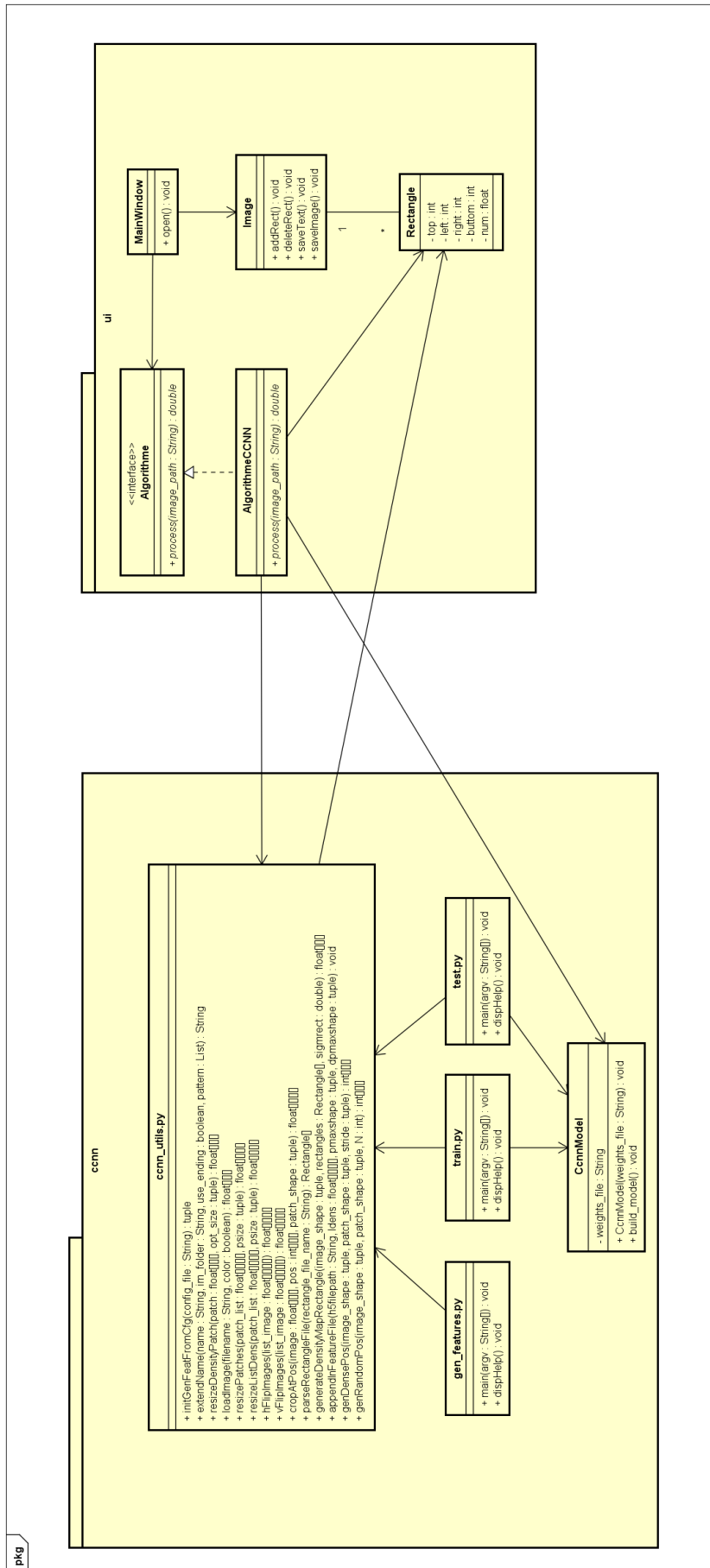


Figure 7 – Diagramme de classe

## 4 Structure des sources

Le système mis en place interagit beaucoup avec des fichiers externes, cette partie propose une façon d'organiser ces sources. Il n'est pas obligatoire de suivre cette structure car les chemins vers les fichiers externes sont indiqués dans un fichier de configuration. Pour en savoir plus référez-vous au [Manuel d'utilisation](#).

Les sources sont contenu dans un dossier nommé *analysepoisson*. Ce dossier contient plusieurs sous dossiers que je vais décrire ici :

- **data** : Contient les différentes images annotées ainsi que deux fichiers textes :
  - "train.txt" contenant la liste des images d'entraînement
  - "test.txt" contenant la liste des images de test
- **genfiles** : Contient les fichiers générés lors de l'exécution du programme :
  - "ccnn\_features.h5" : Contient les patches d'entraînement générés avec le script Python *gen\_features.py*.
  - "ccnn\_logs.log" Les logs du système (il faut rediriger les outputs vers ce fichier).
  - Le dossier "predicted\_density\_map" contient toutes les cartes de densité générées durant la phase de test ou générées avec l'interface. Les cartes de densité ont le même nom que l'image originale.
  - "train\_results.txt" Fichier des résultats des tests de prédiction sur les images d'entraînement. Il contient pour chaque image :
    - le nom de l'image
    - le nombre de d'objets prédits
    - le vrai nombre d'objetsA la fin il contient la MAE et la MSE entre les prédictions et le groundtruth sur les images d'entraînement.
  - "test\_results.txt" Fichier des résultats des tests de prédiction sur les images de test.



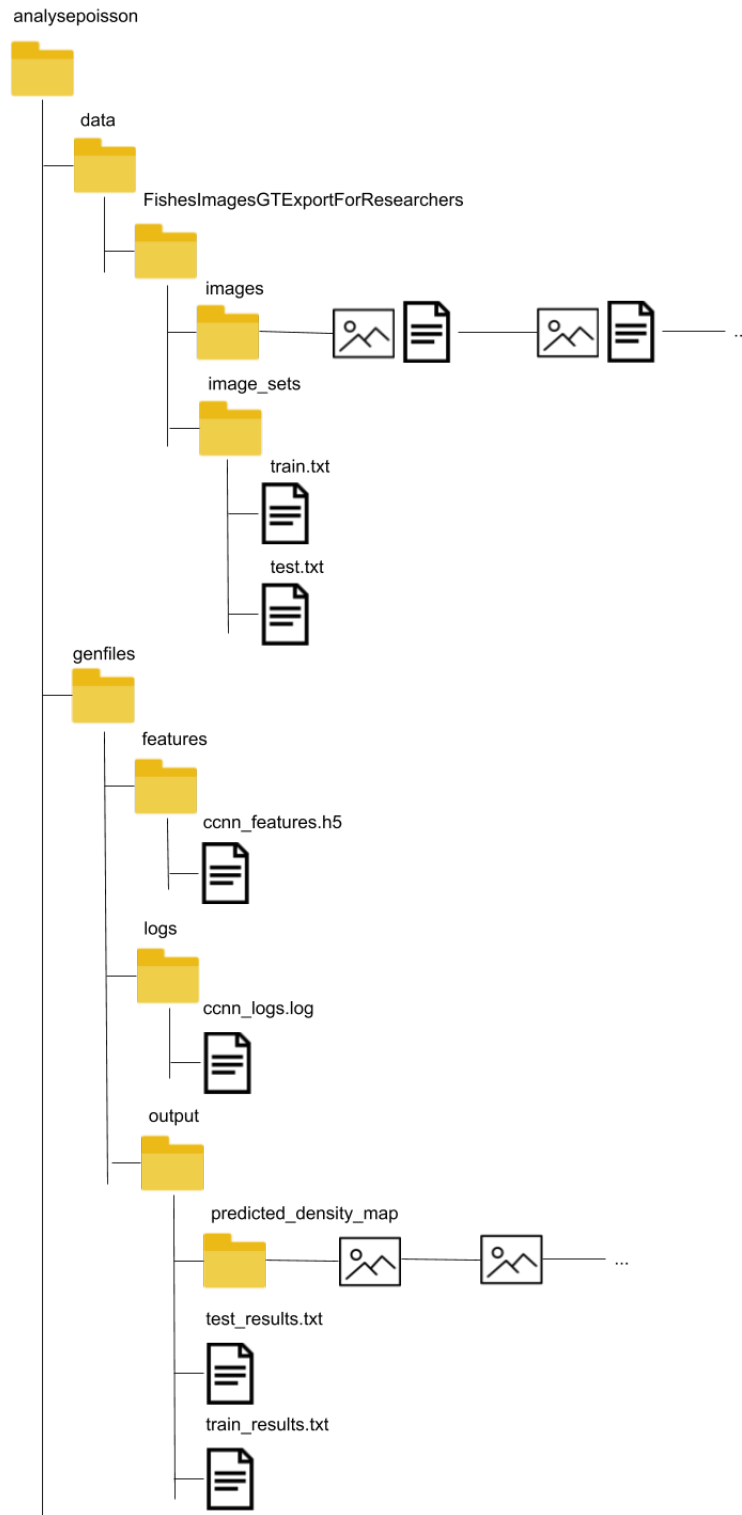


Figure 8 – Structure des sources 1/2

- **src** : Ce dossier contient les sources Python du projet. Le dossier *test* contient les tests unitaires Python qui ont été développés. Le dossier *main* contient les deux packages *ui* et *ccnn*. Il contient aussi :
  - *ccnn\_config.yml* contenant les différentes configurations pour le projet.
  - Un script *train-test.sh*, qui exécute les trois scripts Python pour entraîner et tester le réseau de neurones. De plus, il redirige les sorties vers le fichier de log.

— *weights* : Contient les différents fichiers de poids du réseau de neurones.

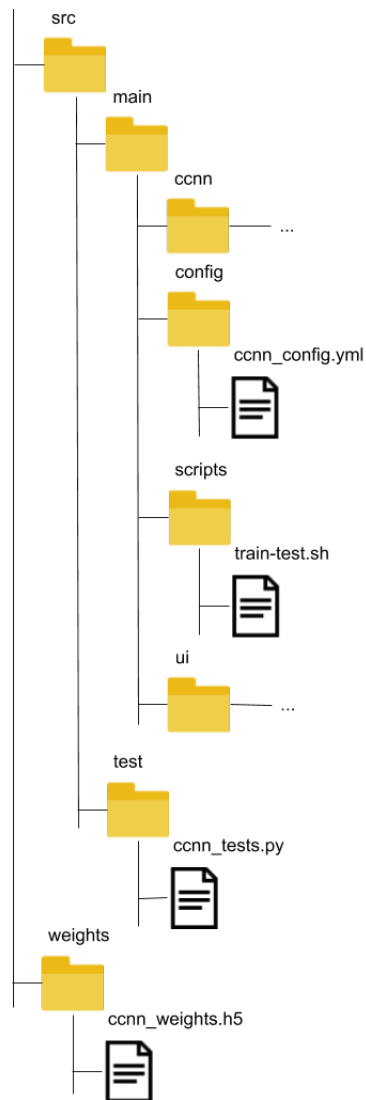
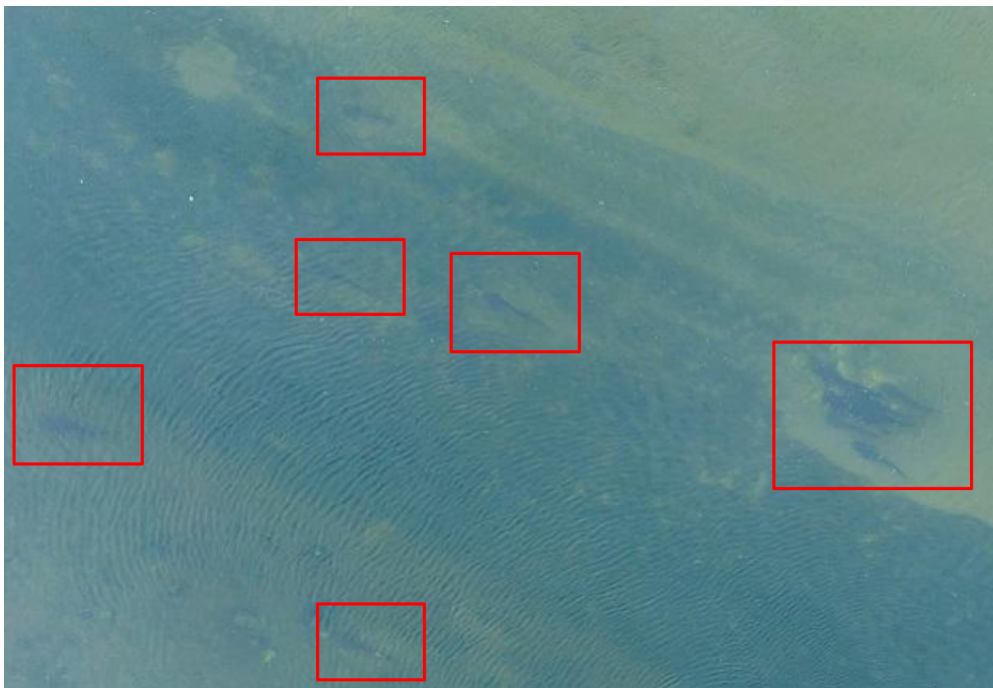


Figure 9 – Structure des sources 2/2

# 2

## Le dataset

Le dataset fourni par la MOA contient 93 images mais contient seulement 12 images positives pour un total de 21 poissons.



**Figure 1** – *Exemple d'image du dataset*

Pour entraîner notre réseau de neurones la MOE a choisie d'utiliser que les images positives. Introduire trop d'images négatives aurait pour conséquence de faire converger les poids du réseau de neurones vers zéros.

Les images du dataset peuvent être très différentes en fonction de l'endroit où elles ont été prises. Ainsi, les images sont divisées en sous dossier de S1 à S9 en fonction de leur localisation.

### **Découpage en images d'apprentissage et images de test**

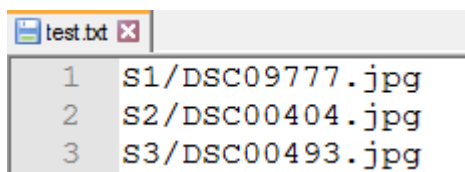
Les images positives sont réparties dans les dossiers S1, S2 et S3. La MOE a donc décidé de découper les images en deux groupes (d'entraînement et de test) de manière à ce qu'il y ait au



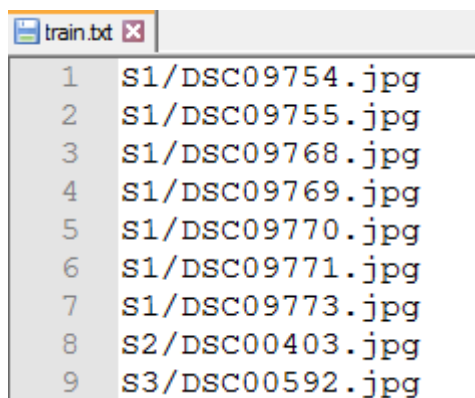
**Figure 2** – *Les différentes localisations*

moins une image de test de chaque localisation.

La MOE a effectuée le découpage indiqué à la (Figure 4) et (Figure 3).



**Figure 3** – *Les images de test*



**Figure 4** – *Les images d'apprentissage*

Ce qui donne 75% des images en images d'entraînement et 25% en images de test.

# 3

## Résultats

### 1 Test sur le dataset transcos

Pour vérifier le bon fonctionnement du système, la MOE a intégré à l'algorithme les poids du réseau de neurones entraînés sur le dataset Transcos qui étaient fournis par les auteurs de l'article. Ensuite, à l'aide de l'IHM la MOE a testé l'algorithme avec des images du dataset Transcos. Si l'algorithme fonctionne correctement on devrait obtenir des cartes de densité cohérentes et des résultats proches des vraies valeurs.

#### Configurations

Pour obtenir les mêmes résultats que ceux qui vont être présentés ci-dessous, il faut configurer le système avec une taille de patch de 115 par 115 pixels et un décalage (stride) de 10 pixels. Cela correspond aux paramètres de configurations indiqués dans l'article.

#### Les résultats

La **Figure 1** présente le résultat obtenu avec l'image "*image-1-000004*" du dataset Transcos. On obtient une prédiction d'environ 101 véhicules dans l'image. D'après les annotations il y a 29 véhicules dans l'image. La différence entre les deux résultats est explicable. En effet normalement l'évaluation des résultats ne se fait pas sur toute l'image mais juste là où il y a de la route dans l'image. Notre algorithme ne prend pas en compte ce paramètre et fait la somme sur toute l'image au lieu de ne prendre que les pixels où il y a de la route. Ce qui explique pourquoi notre algorithme sur-estime le nombre de véhicule. La **Figure 2** présente la carte de densité obtenue pour cette image, on voit qu'elle est bien cohérente avec la position des véhicules dans l'image. On voit aussi que la carte s'est aussi un peu activée en dehors des routes ce qui explique le résultat.

La **Figure 3** présente le résultat obtenu avec l'image "*image-1-0000027*" du dataset Transcos. On obtient environ 66 véhicules prédits pour 35 véhicules selon les annotations. La **Figure 4** présente la carte de densité prédite pour cette image.

La **Figure 5** présente le résultat obtenu avec l'image "*image-2-0000083*" du dataset Transcos. On obtient environ 101 véhicules prédits pour 74 véhicules selon les annotations. La **Figure 6** présente la carte de densité prédite pour cette image.

Les résultats obtenus sont assez satisfaisants pour conclure que notre algorithme donne des résultats similaires à celui de l'article.



Figure 1 – IHM

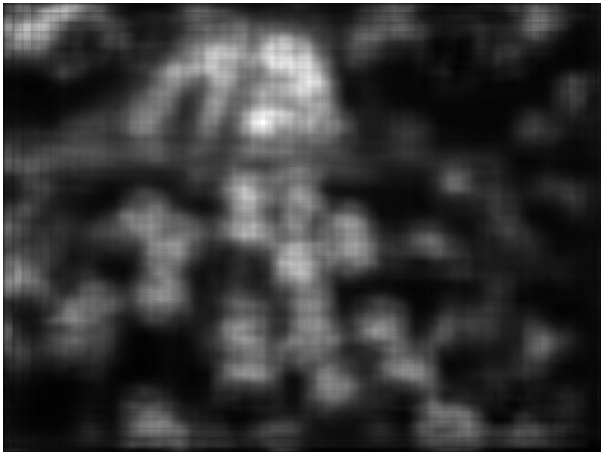


Figure 2 – Carte de densité



Figure 3 – IHM

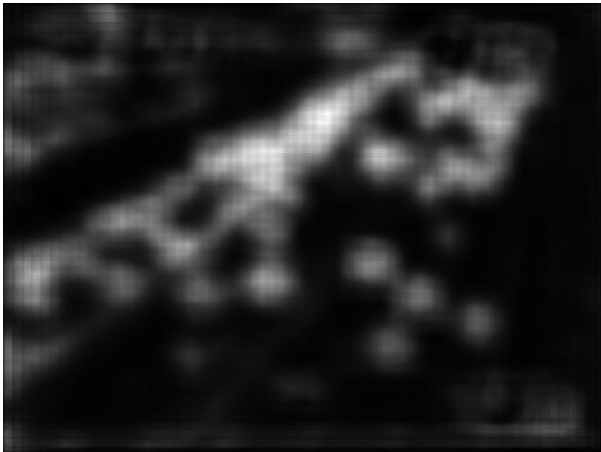


Figure 4 – Carte de densité

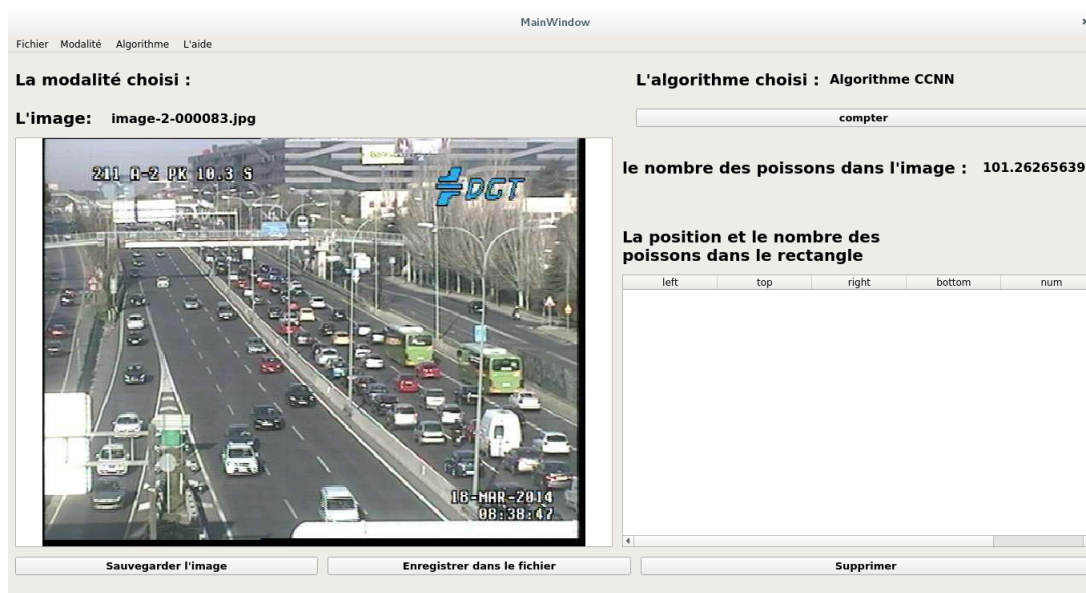


Figure 5 – IHM



Figure 6 – Carte de densité



## 2 Entrainement sur les images du dataset

### 2.1 Configurations

La MOE a choisi de fixer la taille des patches à 250 pixels de côté car il faut au moins des patches de cette taille là pour contenir plusieurs poissons dans un patch. Pour l'évaluation, le décalage (stride) entre chaque patch a été fixé à 25 pixels en largeur et en hauteur. De plus, dans les images d'entraînement on fait le choix de prendre 800 patches aléatoirement. Une "data augmentation" est faite en retournant chaque patch horizontalement et verticalement.

La **Figure 7** présente les différentes configurations utilisées.

```

1  DATASET: 'FishesImagesGTEExportForResearchers'
2  PROJECT_FOLDER: '/media/mika/Bibliothèque/Documents/Cours/Cours5A/S9/PRD/svn/analysepoisson'
3  PATCH_SIZE: 250 # patch are squared
4  STRIDE: 25 # stride for the patches, only for testing
5  NUMBER_OF_PATCH: 800 # number of patches per image, only for training
6  SIGMA: 30.0
7  FLIP_H: True
8  FLIP_V: True
9  IM_FOLDER: 'data/FishesImagesGTEExportForResearchers/images'
10 TRAIN_LIST: 'data/FishesImagesGTEExportForResearchers/image_sets/train.txt'
11 TEST_LIST: 'data/FishesImagesGTEExportForResearchers/image_sets/test.txt'
12 RECTANGLE_FILE_ENDING: '.txt'
13 FEATURES_FILE: 'genfiles/features/ccnn_features.h5'
14 #WEIGHTS_FILE: 'weights/ccnn_weights.h5'
15 WEIGHTS_FILE: 'weights/transcos_ccnn.h5'
16 OUTPUT_RESULTS_FOLDER: 'genfiles/output'
17 OUTPUT_DENS_MAP_FOLDER: 'genfiles/output/predicted_density_map'
18
19 # CNN model params
20 CNN_PW_IN: 72 # CNN patch width in
21 CNN_PW_OUT: 18 # CNN patch width out
22 BATCH_SIZE: 128

```

**Figure 7** – Configurations du CCNN

### 2.2 Méthode d'évaluation des résultats

La MOE a décidé qu'il serait pertinent d'évaluer le système toutes les 10 epochs. Une epoch correspond à un entraînement sur tous les patches des images d'apprentissage. A chaque évaluation les carte de densité prédites ont été sauvegardées ainsi que les différentes mesures de performances (MAE et MSE). La MOE a fait le choix d'arrêter à la quarantième epoch. Dans les résultats qui vont être présentés dans la partie suivante, l'epoch 0 correspond à l'algorithme pré-entraîné sur le dataset Transcos sans aucun entraînement sur les poissons.

### 2.3 Résultats et interprétation

Cette partie présente les résultats obtenus aux epochs 0, 10, 20, 30 et 40. Pour chaque image on peut voir le nombre de poissons prédits.

**Figure 8** et **Figure 9** : Les résultats obtenus par l'algorithme sont bien supérieurs aux vraies valeurs. Ce n'est pas très étonnant comme l'algorithme n'a jamais été entraîné sur les poissons.

**Figure 10** et **Figure 11** : Au bout de 10 epochs seulement les valeurs prédites ont brutalement diminuées. Pour 7 images sur 12, les prédictions sont en dessous de la vraie valeur. On remarque que pour l'image *DSC09769* l'algorithme sort une valeur aberrante comparé aux autres images.

image	prediction	groundtruth
S1/DSC09777.jpg	85.6299128058	2
S2/DSC00404.jpg	128.742753211	2
S2/DSC00493.jpg	91.5147563946	1
mean absolute error :	100.29580747	
mean squared error :	10416.8696442	

Figure 8 – Images de test, epoch 0

image	prediction	groundtruth
S1/DSC09754.jpg	101.975560585	2
S1/DSC09755.jpg	99.032961503	2
S1/DSC09768.jpg	104.26793553	13
S1/DSC09769.jpg	90.0337235247	14
S1/DSC09770.jpg	96.4517757289	1
S1/DSC09771.jpg	96.3334043368	1
S1/DSC09773.jpg	88.6073052314	1
S2/DSC00403.jpg	219.674142937	2
S3/DSC00592.jpg	102.898115779	1
mean absolute error :	106.919436128	
mean squared error :	13017.9188229	

Figure 9 – Images d'entraînement, epoch 0

image	prediction	groundtruth
S1/DSC09777.jpg	4.41062813817	2
S2/DSC00404.jpg	0.719417992305	2
S3/DSC00493.jpg	0.532850421483	1
mean absolute error :	1.38611990813	
mean squared error :	2.55641567589	

Figure 10 – Images de test, epoch 10

Figure 12 et Figure 13 : Les valeurs prédites sont de nouveaux au dessus des valeurs réelles pour la plus part des images. Les valeurs prédites semblent être plus homogènes et beaucoup de valeurs sont autour de 11, 12 et 13.

Figure 14 et Figure 15 : Les valeurs prédites sont toutes comprises entre 4 et 8.

Figure 16 et Figure 17 : Pour toutes les images les valeurs prédites frôle 0. Les poids semblent converger vers 0. Pour la première fois les mesures d'erreur se remettent à augmenter.

Sur les Figure 18 et Figure 19 on peut voir que globalement les deux mesures de l'erreur semblent diminuer. Sur les images de tests la MAE et la MSE augmente à l'époch 20. Cela est dû

image	prediction	groundtruth
S1/DSC09754.jpg	1.08293469394	2
S1/DSC09755.jpg	0.347450452733	2
S1/DSC09768.jpg	4.83434486631	13
S1/DSC09769.jpg	154.326757194	14
S1/DSC09770.jpg	3.00007109438	1
S1/DSC09771.jpg	3.25169139317	1
S1/DSC09773.jpg	5.04687873822	1
S2/DSC00403.jpg	1.26005271664	2
S3/DSC00592.jpg	0.585072279933	1
mean absolute error :	17.8350603789	
mean squared error :	2198.6684389	

Figure 11 – Images d’entraînement, epoch 10

image	prediction	groundtruth
S1/DSC09777.jpg	13.9845936502	2
S2/DSC00404.jpg	28.4241741612	2
S3/DSC00493.jpg	59.8755249587	1
mean absolute error :	32.4280975901	
mean squared error :	1436.06496808	

Figure 12 – Images de test, epoch 20

au fait que les images de tests ne contiennent que 1 ou 2 poissons et donc quand les prédictions ont augmentées les erreurs ont augmentées aussi.

La Figure 20 montre comment la prédiction a évolué par rapport à la valeur réelle pour l’image *DSC09768*.

Les Figure 22, Figure 23, Figure 24, Figure 25 et Figure 26 permettent de visualiser l’évolution des cartes de densité prédites pour l’image *DSC09768* dont le groundtruth est présenté sur la Figure 21. Les cartes de densité présentées sont une version visualisable des vraies cartes de densité. Pour générer ces images on change la valeur du pixel qui a la plus grande valeur pour la mettre à 255, de cette manière il apparaît blanc. Ces images ne rendent donc pas compte de la valeur présente sur chaque pixel.

### Interprétation des résultats

Si on observe les cartes de densité, on peut voir que le système a du mal à reconnaître les poissons. Il est très rare que la carte s’active à l’endroit des poissons et pas ailleurs, et ceci sur toutes les epochs observées. Même si des fois on a une prédiction proche du nombre réel de poisson cela semble plus être du hasard. Par exemple à l’epoch 20, pour l’image *DSC09768* on a un résultat très proche de la vraie valeur : 13 poissons. Cependant si on regarde les autres résultats, on observe que pour toutes les images le système prédit à peu près 13 poissons. De plus si on regarde la carte de densité à l’epoch 20 de l’image *DSC09768* (Figure 24), le système

image	prediction	groundtruth
S1/DSC09754.jpg	13.898534201	2
S1/DSC09755.jpg	12.4674580586	2
S1/DSC09768.jpg	13.5056491004	13
S1/DSC09769.jpg	12.143080999	14
S1/DSC09770.jpg	11.8898140709	1
S1/DSC09771.jpg	12.3172691699	1
S1/DSC09773.jpg	12.2607237112	1
S2/DSC00403.jpg	33.689092721	2
S3/DSC00592.jpg	18.0253147069	1
mean absolute error :	11.8789749712	
mean squared error :	213.597676927	

Figure 13 – Images d’entraînement, epoch 20

image	prediction	groundtruth
S1/DSC09777.jpg	5.07199777336	2
S2/DSC00404.jpg	5.53754172203	2
S3/DSC00493.jpg	8.19422964193	1
mean absolute error :	4.60125637911	
mean squared error :	24.5694372985	

Figure 14 – Images de test, epoch 30

ne semble pas s’activer spécialement au niveau des poissons.

Si on se tient aux résultats obtenus à la quarantième epoch, les poids semblent converger vers 0, faisant augmenter les mesures d’erreur. Même si la MOE n’a pris que les images positives, si on prend 800 patchs dans une image avec 1 seul poisson, la majorité des patchs à prédire ne contiendront aucun poisson. Cela a pour conséquence que le réseau de neurones doit retourner des carte de densité composé entièrement de zéros pour un grand nombre de patchs. Cela explique en partie que les poids du réseau de neurones semblent converger vers zéro.

image	prediction	groundtruth
S1/DSC09754.jpg	6.06626847739	2
S1/DSC09755.jpg	4.78962164901	2
S1/DSC09768.jpg	5.81400977877	13
S1/DSC09769.jpg	5.34754481204	14
S1/DSC09770.jpg	5.40931740165	1
S1/DSC09771.jpg	5.25928692375	1
S1/DSC09773.jpg	5.36347620963	1
S2/DSC00403.jpg	7.05239519471	2
S3/DSC00592.jpg	4.83192060503	1
mean absolute error :	4.9567479856	
mean squared error :	27.5170896578	

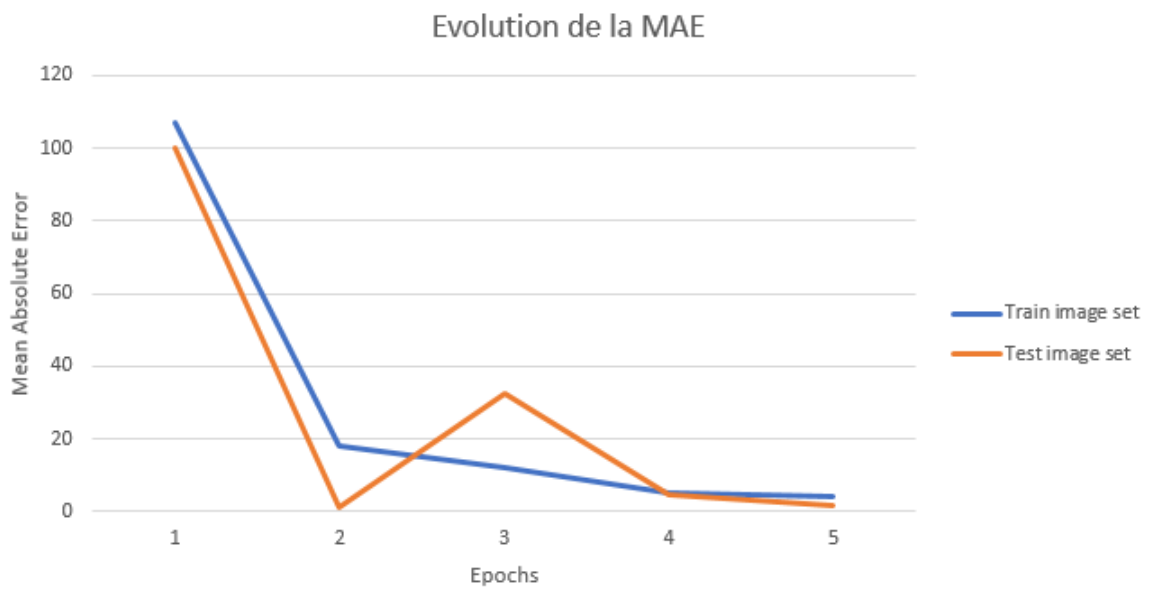
Figure 15 – Images d'entraînement, epoch 30

image	prediction	groundtruth
S1/DSC09777.jpg	2.4274579324e-06	2
S2/DSC00404.jpg	0.00215225584572	2
S3/DSC00493.jpg	0.0327957522106	1
mean absolute error :	1.6550165215	
mean squared error :	2.97562331865	

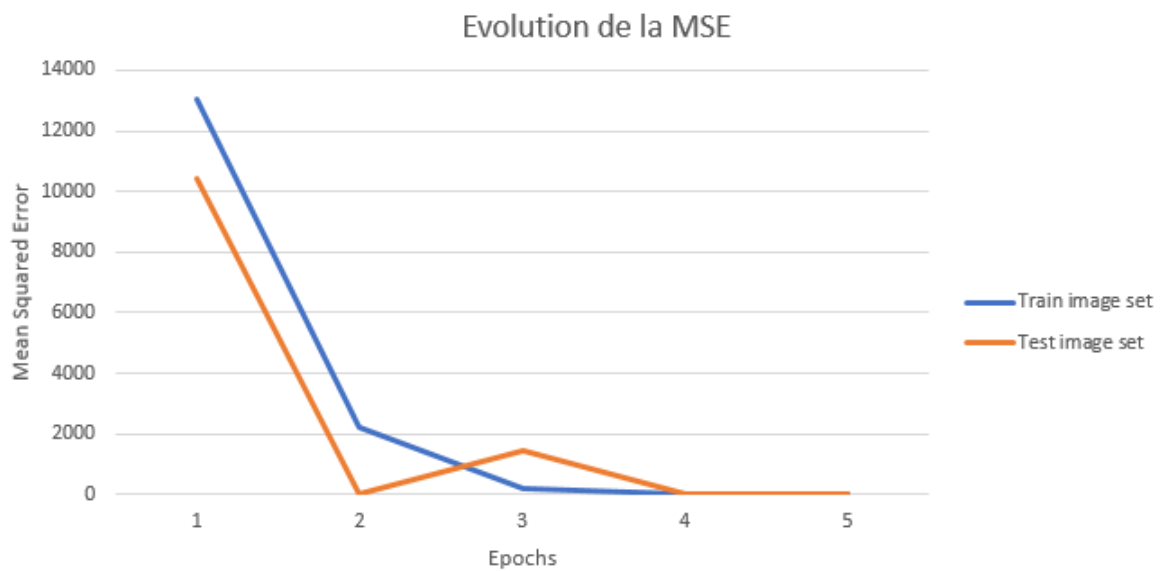
Figure 16 – Images de test, epoch 40

image	prediction	groundtruth
S1/DSC09754.jpg	0.00259123895359	2
S1/DSC09755.jpg	0.000412589735076	2
S1/DSC09768.jpg	3.18865479844e-07	13
S1/DSC09769.jpg	0.000215460587073	14
S1/DSC09770.jpg	4.38656350859e-07	1
S1/DSC09771.jpg	3.35892417726e-05	1
S1/DSC09773.jpg	7.31366092989e-07	1
S2/DSC00403.jpg	0.00917252320566	2
S3/DSC00592.jpg	7.2558064316e-07	1
mean absolute error :	4.10973026487	
mean squared error :	42.3272526115	

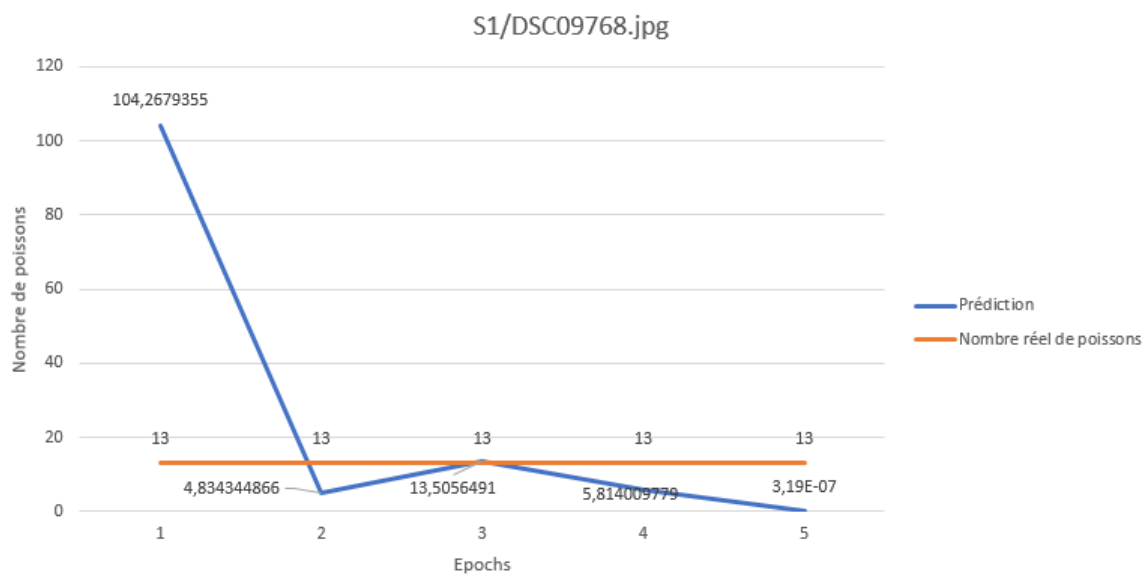
Figure 17 – Images d'entraînement, epoch 40



**Figure 18** – Evolution de la MAE au cours des epochs 0, 10, 20, 30, 40



**Figure 19** – Evolution de la MSE au cours des epochs 0, 10, 20, 30, 40



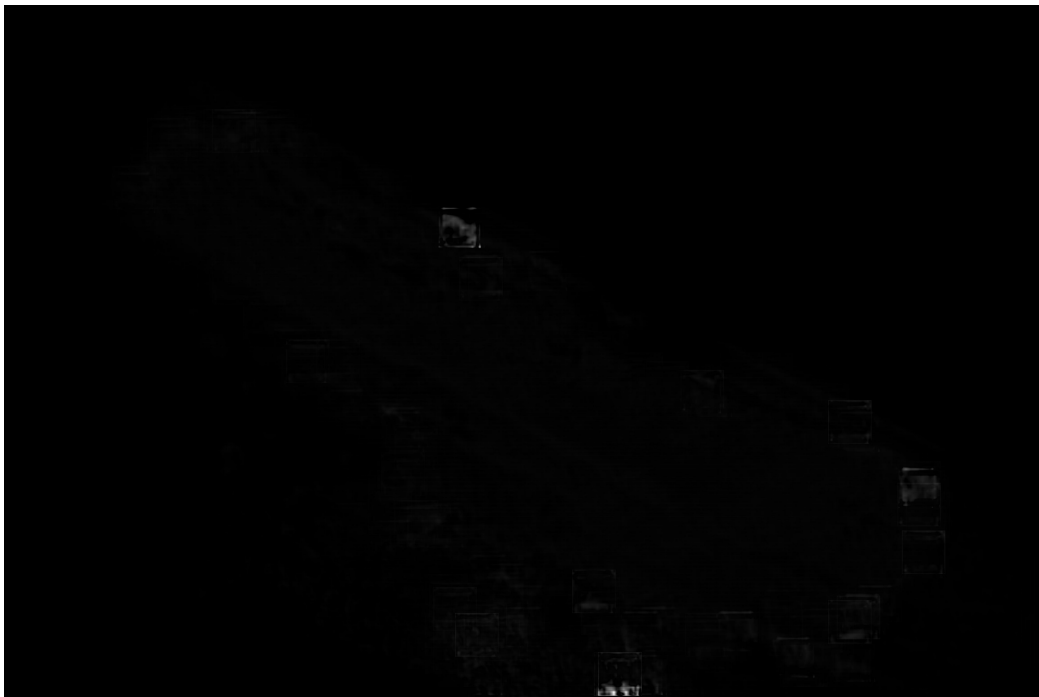
**Figure 20** – Evolution de la prédiction pour l'image S1/DSC09768 au cours des epochs 0, 10, 20, 30, 40



**Figure 21** – L'image S1/DSC09768 originale avec les poissons annotés

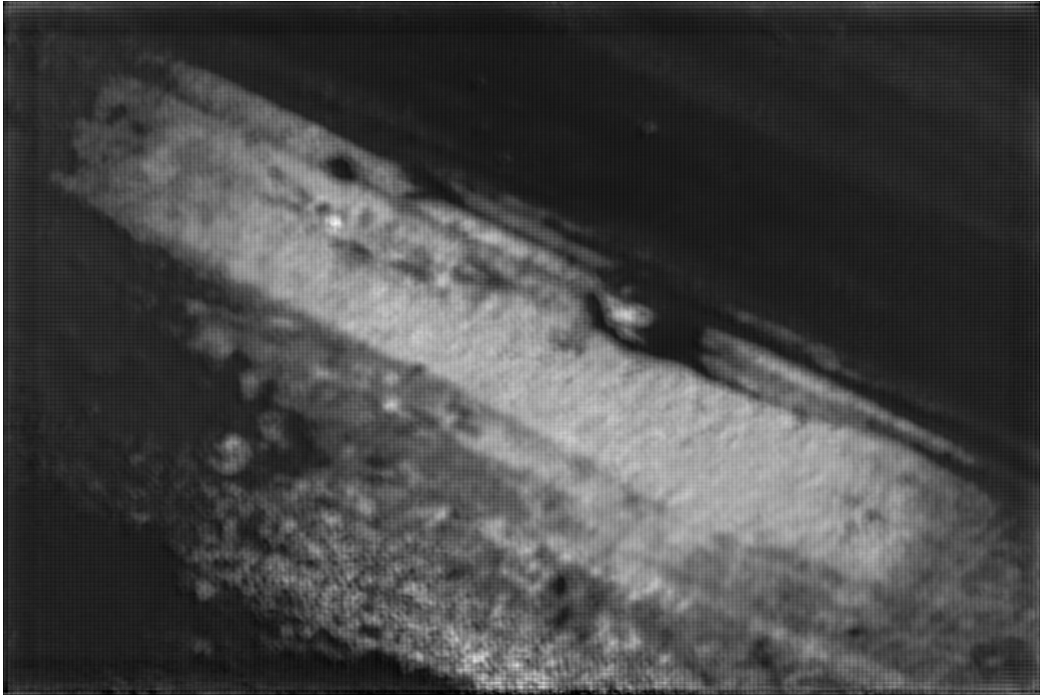


**Figure 22** – *Cartes de densité de l'image S1/DSC09768 à l'epoch 0*



**Figure 23** – *Cartes de densité de l'image S1/DSC09768 à l'epoch 10*

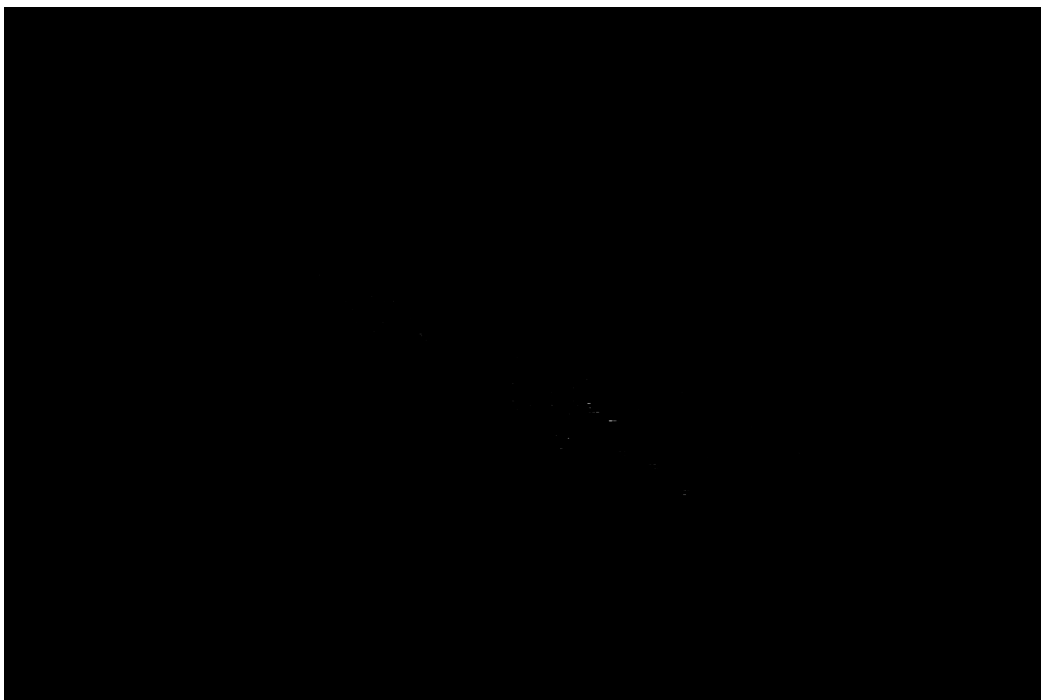




**Figure 24** – *Cartes de densité de l'image S1/DSC09768 à l'epoch 20*



**Figure 25** – *Cartes de densité de l'image S1/DSC09768 à l'epoch 30*

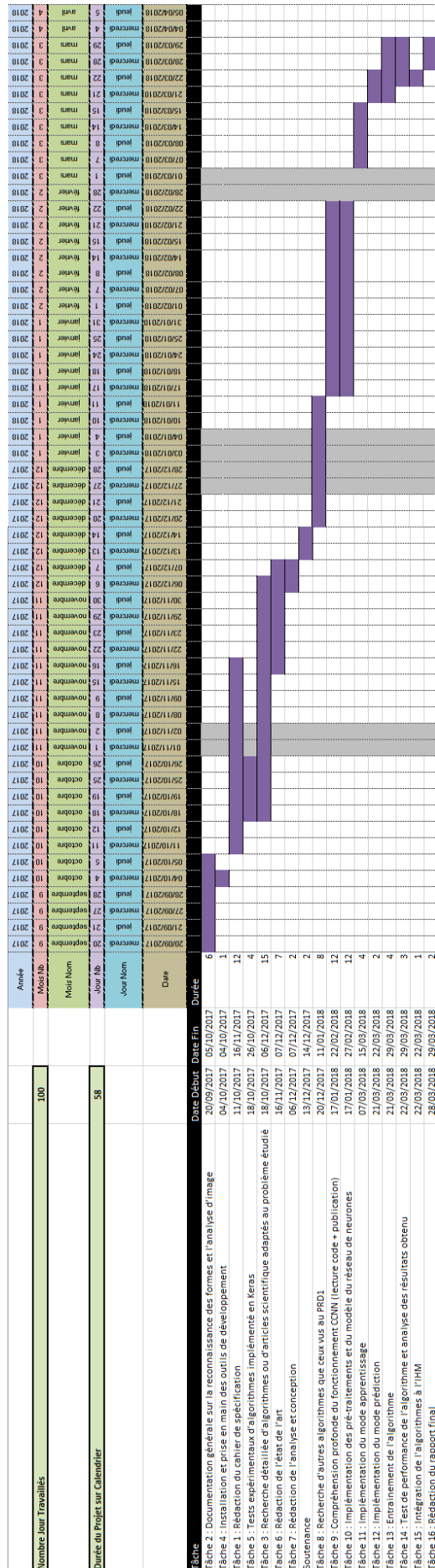


**Figure 26** – *Cartes de densité de l'image S1/DSC09768 à l'epoch 40*

# 4

## Diagramme de Gantt du PRD2

Ce chapitre présente le diagramme de gantt des tâches effectuées depuis le début de l'année. Il a un peu évolué par rapport à celui produit à la fin du premier semestre ([Section 2](#) (Annexe D)).



**Figure 1** – Le gantt des tâches effectuées au PRD2

# Conclusion

Dans cette partie, nous allons essayer d'avoir un regard critique sur ce qui a été produit et voir les pistes à explorer pour améliorer le système.

Dans la partie mise en oeuvre (**Partie IV**) la MOE a fait le choix de baser l'implémentation du système de reconnaissance de formes sur un le travail d'autres chercheurs en analyse d'images [12]. Les résultats obtenus dans cet article sont très bons, pour le dataset Transcos, c'est d'ailleurs toujours ce système qui possède les meilleurs performances à l'heure où ces lignes sont écrites [WWW22].

La raison pour laquelle le système développé n'a pas pu converger vers des résultats aussi satisfaisant avec les poissons est lié au dataset que la MOE disposait. La première raison est que le dataset était déjà trop petit : seulement 12 images positives. Pour contourner ce problème la MOE a proposer de faire une "*data augmentation*" en retournant les patches d'entraînement verticalement et horizontalement. Cela a permis de tripler le nombre de patches d'apprentissage. Cependant cela n'a pas été suffisant.

La seconde raison est que le nombre d'objets par image est loin d'être aussi grand que dans le dataset Transcos. Ce qui a eu pour conséquence de construire un grand nombre de patches négatifs. Pour remédier à ce problème, la MOE aurait pu proposer une autre manière d'extraire les patches d'apprentissage dans les images annotées. Plutôt que de les tirer aléatoirement partout dans l'image, il aurait peut-être fallut augmenter les chances de tirer un patch à l'endroit des poissons.

Concernant le développement du système, la MOE a essayer de développer un système qui est le plus évolutif possible. Avec une documentation générée, un manuel d'installation, un manuel d'utilisation et un manuel du développeur, la MOE espère que le projet sera facilement repris.

## Annexes

# A

## Description des interfaces externes du logiciel

### 1 Interfaces matériel/logiciel

Nous utilisons la bibliothèque Tensorflow qui interagit directement avec les CPU<sup>1</sup> ou les GPU<sup>2</sup>.

### 2 Interfaces homme/machine

L'interface Homme-Machine est développée par Yaofutian LU, une étudiante de l'école dans le cadre d'un PRD. Ce projet est indépendant de celui présenté dans ce cahier de spécification. Néanmoins, pour pouvoir intégrer l'algorithme à l'IHM, il doit tenir compte de plusieurs contraintes fonctionnelles. Celles-ci sont décrites dans la partie des spécifications fonctionnelles.

---

1. Central Processing Unit  
2. Graphics Processing Unit

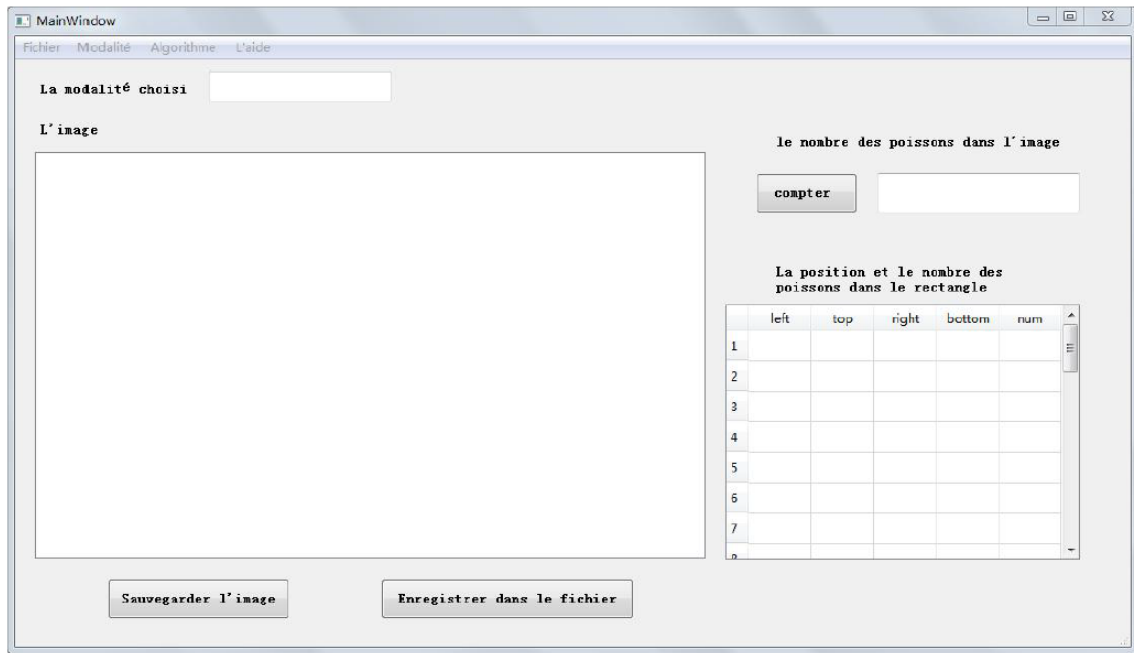


Figure 1 – Capture d'écran de l'IHM

### 3 Interfaces logiciel/logiciel

L'application utilise la bibliothèque de Deep Learning Keras qui s'exécute par-dessus TensorFlow une autre bibliothèque de Deep Learning de plus bas niveau développée par Google.



# B

## Spécifications fonctionnelles

### 1 Mode apprentissage

#### 1.1 Format des entrées

En mode apprentissage les entrées sont une liste d'images avec pour chaque image l'estimation du nombre de silures dans l'image. Concrètement, l'algorithme prend en entrée le chemin du dossier contenant les images. Dans ce dossier, pour chaque image se trouve un fichier texte du même nom que l'image avec les données sur le nombre de silures. Le fichier texte est de la forme suivante :

**Image1.txt**

*Top Left Bottom Right Num*

*Left*, *Top*, *Right*, et *Bottom* représentent une position en pixel dans l'image. Ces quatre chiffres suffisent pour représenter un rectangle dans l'image. *Num* indique quant à lui le nombre de silures estimés dans le rectangle.

Chaque valeur est séparée par un espace. Si l'image contient plusieurs rectangles, alors on revient à la ligne pour chaque rectangle.

#### 1.2 Les sorties

Lors de l'apprentissage l'algorithme met à jour les poids du réseau de neurone. Ces poids sont enregistrés dans un fichier hdf5. Si le réseau de neurone n'a jamais été entraîné, ce fichier est créé lors du premier apprentissage puis modifié les fois suivantes.

### 2 Mode prédiction

#### 2.1 Les entrées

En mode prédiction l'algorithme prend une seule image en entrée avec une taille variable. Comme le nombre de silures est à déterminer par l'algorithme, l'image est le seul paramètre

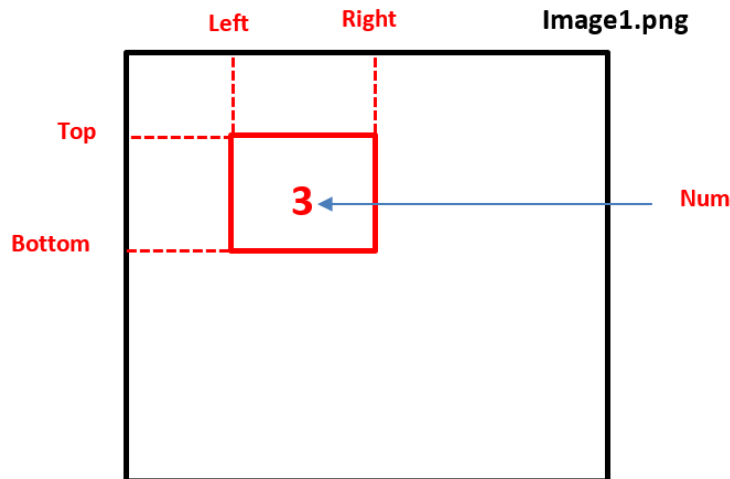


Figure 1 – Format de l'image d'apprentissage

dont a besoin l'algorithme dans ce mode.

## 2.2 Les sorties

En mode prédiction l'algorithme retourne son estimation du nombre de silures dans l'image passée en entrée, pour faire cela il retourne un objet de la classe « Image ». Cette image contient un seul rectangle qui entoure de l'image et dont l'attribut « num » contient l'estimation du nombre de silures.

## 3 Intégration à l'IHM

Cette partie comprend la reprise du code réalisé par l'autre élève de Polytech. L'intégration à l'IHM se fera comme définie dans la partie « structure générale du système ». Il est possible que la MOE soit amenée à ajouter plus de code dans l'IHM.

# C

# Spécifications non fonctionnelles

## 1 Contraintes de développement et conception

### 1.1 Matériel

Si l'algorithme de reconnaissance est exécuté sur le CPU de la machine, aucune contrainte matérielle particulière n'est nécessaire pour que l'application fonctionne. Cependant, on peut diminuer le temps d'exécution de l'application lorsque l'on analyse une image. Pour ce faire, on peut choisir d'installer l'application pour qu'elle exécute l'algorithme sur le GPU de l'ordinateur. Un GPU NVIDIA est alors fortement recommandé.

A contrario, lors de l'apprentissage de l'algorithme, une plus grande puissance de calcul est nécessaire. Pour réduire le temps d'apprentissage nous n'aurons pas d'autres choix que d'utiliser le GPU. Pour ce faire nous pouvons utiliser les machines de l'école qui possèdent des GPU NVIDIA avec CUDA.

Le système informatique sur lequel est entraîné l'algorithme doit fournir de grandes capacités de stockage avec au moins 200 Go de mémoire pour pouvoir supporter l'échantillon d'images d'apprentissage.

### 1.2 Langage

L'IHM et l'algorithme sont développés en Python. La raison pour laquelle la MOA a choisi ce langage est qu'il existe de nombreuses bibliothèques de machine learning dans ce langage.

### 1.3 Environnement

Comme l'application est développée en Python, aucune contrainte n'est imposée pour le système d'exploitation. Pour effectuer les tests et l'apprentissage de l'algorithme, nous utiliserons Anaconda comme environnement de travail.

## 1.4 Bibliothèques

L'algorithme sera implémenté avec la bibliothèque de Deep Learning Keras. C'est une bibliothèque de haut niveau qui peut s'exécuter par-dessus TensorFlow, la bibliothèque de Deep Learning de Google. Keras est la bibliothèque privilégiée car c'est celle qui est utilisée par les chercheurs de l'équipe de RFAI.

## 2 Contraintes de fonctionnement et d'exploitation

### 2.1 Performances

En mode prédiction, il n'y a pas vraiment de contraintes concernant le temps de réponse. Cependant, si le temps de réponse est trop important en faisant fonctionner l'algorithme sur le CPU, il sera possible de le faire fonctionner sur le GPU pour gagner du temps. Pour le mode apprentissage, le temps de réponse peut être long car il n'est pas destiné à être utilisé souvent.

### 2.2 Capacités

L'algorithme de machine learning fonctionne avec une phase d'apprentissage avant de pouvoir faire de la prédiction. Ce qui implique que la prédiction dépend des exemples passés durant la phase d'apprentissage. De ce fait, la prédiction sera plus effective si l'image passée en mode prédiction ressemble aux images passées lors de la phase d'apprentissage. Par exemple, si les images d'apprentissage ne sont que des images aériennes, il est peu probable que l'algorithme fournisse des résultats satisfaisants si on lui demande de faire une prédiction à partir d'une image prise depuis le sol.

## 3 Maintenance et évolution du système

La maintenance n'est pas prévue pour ce projet. Cependant, le système doit être suffisamment évolutif pour intégrer facilement de nouveaux algorithmes de reconnaissance d'image. Cela implique qu'il doit y avoir un faible couplage entre l'IHM et les algorithmes. C'est pour cette raison que la MOA et la MOE ont choisi d'implémenter une interface « Algorithme » générique à tous les algorithmes d'analyse d'image.

# D

# Plan de développement

## 1 Découpage en tâches

### 1.1 Tâche 1 : Rédaction du cahier de spécification

**Description de la tâche :**

La rédaction du cahier de spécification permet de définir le contexte, les objectifs, le planning, les spécifications fonctionnelles et non fonctionnelles.

**Livrables :**

Le cahier de spécification est à rendre le 07/12/2017.

**Estimation de charge :**

Cette tâche est estimée à 6 jour/homme.

### 1.2 Tâche 2 : Documentation générale sur la reconnaissance des formes et l'analyse d'image

**Description de la tâche :**

Identifier les enjeux de la reconnaissance d'image et du machine learning. Faire un tour d'horizon de la discipline et s'approprier les termes techniques.

**Estimation de charge :**

Cette tâche est estimée à 6 jour/homme.

### 1.3 Tâche 3 : Recherche détaillée d'algorithmes ou d'articles scientifique adaptés au problème étudié

**Description de la tâche :**

Chercher s'il existe des solutions pour des problèmes similaires au notre. Identifier les algorithmes utilisés, leur fonctionnement et leurs limites.

**Estimation de charge :**

Cette tâche est estimée à 6 jour/homme.

**1.4 Tâche 4 : Installation et prise en main des outils de développement****Description de la tâche :**

Cette tâche consiste à installer Anaconda, prendre en main l'environnement, installer les bibliothèques Keras et Tensorflow puis tester un algorithme de machine learning dans cet environnement.

**Estimation de charge :**

Cette tâche est estimée à 1 jour/homme.

**1.5 Tâche 5 : Tests expérimentaux d'algorithmes implémenté en Keras****Description de la tâche :**

Pour chaque algorithme identifié trouver une implémentation sur Keras et l'exécuter dans notre environnement.

**Estimation de charge :**

Cette tâche est estimée à 1 jour/homme par algorithme.

**1.6 Tâche 6 : Rédaction de l'état de l'art****Description de la tâche :**

Rédaction de l'état de l'art. Faire un résumé des enjeux et des applications de la reconnaissance d'image et du machine learning. Définir les principaux axes de recherche. Résumer l'ensemble des algorithmes et les articles étudiés et leurs résultats expérimentaux.

**Livrables :**

L'état de l'art est à rendre le 07/12/2017 avec le cahier de spécification.

**Estimation de charge :**

Cette tâche est estimée à 6,5 jour/homme.

**1.7 Tâche 7 : Rédaction de l'analyse et conception****Description de la tâche :**

Rédiger l'analyse de l'état de l'art et en déduire une architecture pour notre système de reconnaissance d'image.

**Livrables :**

L'analyse est à rendre le 07/12/2017 avec le cahier de spécification.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.8 Tâche 8 : Conception de la structure du réseau de neurones****Description de la tâche :**

Définir la structure du modèle du réseau de neurones c'est à dire le nombre de couche, les entrées et sorties de chaque couche, l'algorithme de mise à jour des poids. Détailler l'architecture décrite dans l'analyse et conception.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.9 Tâche 9 : Exploration avancée de la bibliothèque Keras****Description de la tâche :**

Prise en main des principales fonctions de Keras, comment fonctionne la création des modèles, les bonnes pratiques, etc.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.10 Tâche 10 : Implémentation des prétraitements et du modèle du réseau de neurones****Description de la tâche :**

Implémenter les prétraitements et le modèle du réseau de neurones en Keras.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.11 Tâche 11 : Implémentation du mode apprentissage****Description de la tâche :**

Implémenter l'algorithme décrit dans l'analyse et conception avec la bibliothèque Keras en tenant compte des spécifications décrites dans le cahier de spécification.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.12 Tâche 12 : Implémentation du mode prédiction****Description de la tâche :**

Implémenter l'algorithme décrit dans l'analyse et conception avec la bibliothèque Keras en tenant compte des spécifications décrites dans le cahier de spécification.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.13 Tâche 13 : Entraînement de l'algorithme****Description de la tâche :**

L'algorithme est entraîné avec les images aériennes des silures.

**Estimation de charge :**

Cette tâche est estimée à 1 jour/homme.

**1.14 Tâche 14 : Test de performance de l'algorithme et analyse des résultats obtenu****Description de la tâche :**

Test des performances de l'algorithme avec des images aériennes de silures. Les résultats obtenus peuvent amener à changer des configurations du système de reconnaissance d'image.

**Estimation de charge :**

Cette tâche est estimée à 9 jour/homme par algorithme.

**1.15 Tâche 15 : Intégration de l'algorithme à l'IHM****Description de la tâche :**

L'algorithme est intégré à l'IHM.

**Livrables :**

Le code est à rendre en avril 2018.

**Estimation de charge :**

Cette tâche est estimée à 2 jour/homme.

**1.16 Tâche 16 : Rédaction du rapport final****Description de la tâche :**

Cette tâche consiste à la rédaction du rapport final de ce PRD, afin de rendre compte du projet. Celui-ci pourra être réalisé tout au long du projet.

**Livrables :**

Document à rendre en avril 2018.

**Estimation de charge :**

Cette tâche est estimée à 6 jour/homme.

**2 Diagramme de Gantt**



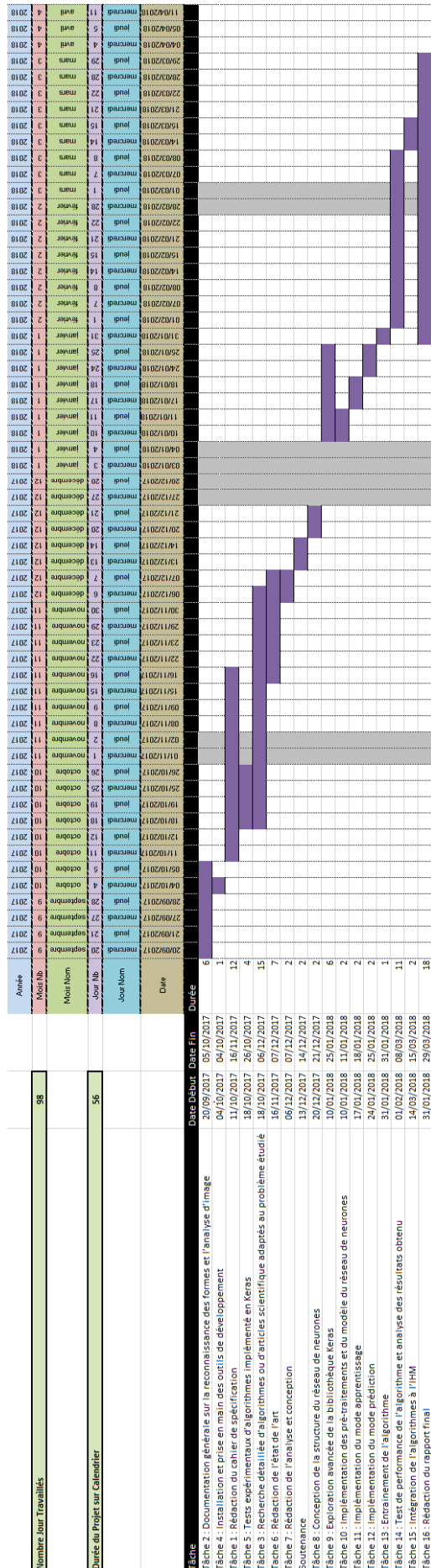


Figure 1 – Diagramme de Gantt

# E

# Manuel d'installation

## 1 Anaconda

Anaconda est une distribution Python. Sa principale fonctionnalité est qu'il permet de gérer de multiples environnements virtuels Python possédant leur propre librairie et version de Python. Il possède aussi son propre outil de gestion de package appelé Conda.



Figure 1 – Logo de Anaconda

Anaconda n'est pas obligatoire pour que les sources du projet puissent compiler, cependant il facilite largement l'installation des dépendances Python nécessaires pour que le projet fonctionne.

Il est possible de télécharger Anaconda depuis leur site officiel :

<https://www.anaconda.com/>

Pour la suite, nous considérerons que Anaconda est installé.

## 2 Subversion

Si vous ne disposez pas des sources, elles sont disponibles sur le redmine de Polytech' Tours dans le projet "analysepoisson" [[WWW2](#)].

Les sources sont gérées dans le dépôt avec le gestionnaire de version Subversion (SVN). Pour récupérer les sources il faut installer un client Subversion. Sur Windows, vous pouvez télécharger [Tortoise SVN](#) ou [Win32svn](#) pour avoir la CLI. Sur Linux vous pouvez installer Subversion en utilisant aptitude, pour cela ouvrez un terminal et entrez la commande suivante :

***apt-get install subversion***

## 3 Récupération des sources

Pour récupérer les sources ouvrez un Terminal ou un invite de commande et entrez la commande suivante avec myusername votre nom d'utilisateur :

```
svn checkout http://redmine.polytech.univ-tours.fr/svn/analysepoisson --username myuser-name
```

Entrez votre mot de passe et validez. Les sources sont alors téléchargées dans le répertoire où vous êtes placé.

## 4 Installation des dépendances

Pour installer les dépendances ouvrez un Terminal et placez vous dans le projet des sources. Entrez la commande suivante pour installer les dépendances dans un nouvel environnement virtuel en utilisant Anaconda :

***conda env create***

Cette commande fonctionne à l'aide du fichier *environment.yml*. Celui-ci contient le nom de l'environnement à créer ("ccnn"), la version de Python à utiliser (3.5) et toutes les dépendances et leurs versions.

Si vous ne souhaitez pas installer les dépendances à l'aide d'Anaconda voici la liste des dépendances et leurs versions. Vous pourrez les installer à l'aide de la commande suivante :

***pip install <packagename>==<version>***

- numpy 1.12.0
- matplotlib 2.0.0
- scikit-learn 0.18.1
- scikit-image 0.13.1
- scipy 0.19.0
- pillow 4.0.0
- seaborn 0.7.1
- h5py 2.7.0
- nose 1.3.7
- tensorflow 1.1.0
- keras 2.0.4
- opencv-python 3.4.0.12

# F

# Manuel d'utilisation

Ce document présente le manuel d'utilisation du logiciel de reconnaissance de formes permettant d'estimer la densité de poisson à partir d'image aérienne. Ce manuel se divise en deux parties, la première décrivant les étapes pour utiliser le logiciel pour prédire le nombre de poisson dans une image et la seconde décrivant les étapes pour entraîner le logiciel à compter les poissons.

## 1 Mode prédiction

Dans cette section, nous allons décrire comment utiliser le logiciel pour prédire le nombre de poissons dans une image. Tout d'abord, nous allons voir comment correctement configurer le système, ensuite nous verrons comment utiliser l'interface pour lancer le système d'analyse d'image.

### 1.1 Configuration

Les sources contiennent un fichier de configuration *ccnn\_config.yml* regroupant les informations que le système a besoin pour fonctionner. Ce fichier est situé dans le dossier *src/main/config/* des sources.

Dans ce fichier il faut éditer :

- Le nom du jeu de données.  
*DATASET* : *'FishesImagesGTExportForResearchers'*
- Le chemin du dossier des sources du projet.  
*PROJECT\_FOLDER* : *'/home/mika/analysepoisson'*
- La taille des patches en pixel, cette taille doit être ajustée pour que dans un patch on puisse voir plusieurs poissons.  
*PATCH\_SIZE* : *250*
- Le décalage entre chaque patch en pixel, environ un dixième de la taille des patches.  
*STRIDE* : *25*
- Le chemin relatif vers les poids du réseau de neurones.  
*WEIGHTS\_FILE* : *'weights/transcos\_ccnn.h5'*

- Le chemin relatif vers le dossier dans lequel la carte de densité sera enregistrée.

`OUTPUT_DENS_MAP_FOLDER` : 'genfiles/output/predicted\_density\_map'

Les autres paramètres n'ont pas besoin d'être édités, ils sont utiles pour le mode apprentissage.

## 1.2 L'interface

Une fois l'application correctement configurée, on peut se servir de l'interface pour lancer l'analyse sur une image. Pour lancer l'interface on peut utiliser python en ligne de commande.

Pour cela ouvrez un terminal ou un invite de commande et placez vous dans le répertoire des sources. Tout d'abord, vous devez activer l'environnement "ccnn" avec Anaconda si ce n'est pas déjà fait, pour cela exécutez la commande suivante :

**source activate ccnn**

Ensuite exécutez la commande suivante pour lancer l'application :

**python src/main/ui/Application.py**

Si tout se passe bien une nouvelle fenêtre devrait s'ouvrir.

Depuis l'interface sélectionnez une nouvelle image en cliquant sur "ouvrir" dans l'onglet "fichier".

Ensuite sélectionnez l'algorithme ccnn en cliquant sur "AlgorithmeCCNN" dans l'onglet "Algorithme".



Figure 1 – Interface de l'application

Pour lancer l'analyse de l'image cliquez sur "Compter", le résultat s'affichera à côté de la phrase "Le nombre de poisson dans l'image". L'analyse de l'image peut prendre plusieurs minutes selon la taille de l'image. Pour une image de 6000x4000 pixels il faut compter environ 10 minutes.

De plus, la carte de densité correspondant à l'image sera enregistrée dans le dossier indiqué dans la variable `OUTPUT_DENS_MAP_FOLDER` dans le fichier de configuration.

## 2 Mode apprentissage

Il est possible d'utiliser les sources du projet pour entrainer le réseau de neurones. Il est possible d'entrainer le réseau de neurones depuis zéro ou d'utiliser des poids déjà existants. Pour cette partie il est nécessaire de posséder un jeux de données d'entraînement comprenant des images annotées. Il faut ensuite indiquer dans le fichier de configurations où elles sont situées.

### 2.1 Configuration

Le jeux de données doit les images d'entraînements annotées. Pour chaque image du jeux de données, il doit y avoir un fichier texte du même nom que l'image comportant la localisation des rectangles et le nombre d'objets qu'il y a dans chaque rectangle. Pour chaque rectangle, le fichier texte doit suivre la nomenclature suivante décrite dans le cahier de spécifications.

#### Image1.txt

*Top Left Bottom Right Num*

Les fichiers textes doivent être placés dans le même dossier que les images.

Les sources contiennent un fichier de configuration `ccnn_config.yml` regroupant les informations que le système a besoin pour fonctionner. Ce fichier est situé dans le dossier `src/main/config/` des sources.

Dans ce fichier il faut éditer :

- Le nom du jeux de données.  
`DATASET` : `'FishesImagesGTExportForResearchers'`
- Le chemin du dossier des sources du projet.  
`PROJECT_FOLDER` : `'/home/mika/analysepoisson'`
- La taille des patches en pixel, cette taille doit être ajustée pour que dans un patch on puisse voir plusieurs poissons.  
`PATCH_SIZE` : `250`
- Le nombre de patches à extraire dans chaque image d'apprentissage.  
`NUMBER_OF_PATCH` : `800`
- La déviation standard du noyau Gaussien pour la génération des cartes de densité.  
`SIGMA` : `30.0`
- On retourne horizontalement les images d'entraînements pour augmenter le nombre d'images d'apprentissages.  
`FLIP_H` : `True`
- On retourne verticalement les images d'entraînements pour augmenter le nombre d'images d'apprentissages.  
`FLIP_V` : `True`
- Le chemin relatif au dossier des sources du projet vers le dossier contenant toutes les images annotées.  
`IM_FOLDER` : `'data/FishesImagesGTExportForResearchers/images'`
- Le fichier texte contenant la liste des images d'apprentissages.  
`TRAIN_LIST` : `'data/FishesImagesGTExportForResearchers/image_sets/train.txt'`
- Le fichier texte contenant la liste des images de tests.  
`TEST_LIST` : `'data/FishesImagesGTExportForResearchers/image_sets/test.txt'`
- Le pattern permettant de reconnaître les fichiers contenant la position des rectangle.  
`RECTANGLE_FILE_ENDING` : `'.txt'`
- Le nom du fichier dans lequel les données transformées d'entraînements seront enregistrées.

- `FEATURES_FILE` : `'genfiles/features/ccnn_features.h5'`
- Le chemin relatif vers les poids du réseau de neurones.
- `WEIGHTS_FILE` : `'weights/transcos_ccnn.h5'`
- Le dossier dans lequel on les fichiers indiquant les résultats seront enregistrés (train\_results.txt et test\_results.txt).
- `OUTPUT_RESULTS_FOLDER` : `'genfiles/output'`
- Le chemin relatif vers le dossier dans lequel la carte de densité sera enregistrée.
- `OUTPUT_DENS_MAP_FOLDER` : `'genfiles/output/predicted_density_map'`
- La taille des entrées du réseau de neurones (ne pas changer).
- `CNN_PW_IN` : `72`
- La taille des sorties du réseau de neurones (ne pas changer).
- `CNN_PW_OUT` : `18`

Les autres paramètres n'ont pas besoin d'être édités, ils sont utiles pour le mode prédiction.

## 2.2 Utilisation des scripts

### 2.2.1 Générer le fichier d'entraînement pour le réseau de neurones

Dans cette partie nous allons voir comment on va générer les données qui vont être utilisé pour entrainer notre réseau de neurones. Comme expliqué dans la partie mise en œuvre, le réseau de neurone ne prend pas directement une image en entrée mais seulement des bouts d'une certaine taille (72x72 pixels). De plus, pour chaque bout d'image il faut générer la carte de densité correspondante de même taille que la sortie du réseau de neurone (18x18 pixels). De plus, il peut y avoir de la "data augmentation". Toutes ces transformations sont faites lors de l'exécution d'un script python et sont enregistrées dans un fichier au format hdf5.

N'oubliez pas de renseigner dans le fichier de configuration la listes des images sur lequel l'apprentissage va être fait dans la variable `TRAIN_LIST`.

Nous allons voir comment utiliser ce script.

Tout d'abord, ouvrez un terminal ou un invite de commande et placez vous dans le répertoire des sources. Vous devez activer l'environnement "ccnn" avec Anaconda si ce n'est pas déjà fait, pour cela exécutez la commande suivante :

***source activate ccnn***

Le script prend en paramètre le fichier de configuration. Pour générer le fichier d'entraînement pour le réseau de neurones entrez la commande suivante :

***python src/main/ccnn/gen\_features.py -cfg src/main/config/ccnn\_config.yml .***

Le fichier sera enregistré dans le fichier indiqué dans le fichier de configuration pour la variable `FEATURES_FILE`.

### 2.2.2 Entraîner le réseau de neurones

Cette partie décrit comment utiliser un script python pour entrainer le réseau de neurones. Pour cette partie il est nécessaire qu'un fichier d'entraînement ait été généré.

Tout d'abord, ouvrez un terminal ou un invite de commande et placez vous dans le répertoire des sources. Vous devez activer l'environnement "ccnn" avec Anaconda si ce n'est pas déjà fait, pour cela exécutez la commande suivante :

***source activate ccnn***

Le script prend en paramètre le fichier de configuration. Pour lancer l'entraînement du réseau de neurones exécutez la commande suivante :

```
python src/main/ccnn/train.py -cfg src/main/config/ccnn_config.yml
```

### 2.2.3 Tester l'algorithme

Une fois le réseau de neurone entraîné, il faut tester l'algorithme complet sur des images complètes car cela n'a pas vraiment de sens de le faire sur seulement des bouts d'images.

Vérifiez que les fichiers contenant les listes d'images d'entraînement [TRAIN\\_LIST](#) et de test [TEST\\_LIST](#) aient bien été renseignés dans le fichier de configuration.

Tout d'abord, ouvrez un terminal ou un invite de commande et placez vous dans le répertoire des sources. Vous devez activer l'environnement "ccnn" avec Anaconda si ce n'est pas déjà fait, pour cela exécutez la commande suivante :

```
source activate ccnn
```

Le script prend en paramètre le fichier de configuration. Pour tester l'algorithme sur les images utilisées pour l'apprentissage puis sur les images de test, lancez la commande suivante :

```
python src/main/ccnn/test.py -cfg src/main/config/ccnn_config.yml
```

Dans le dossier renseigné dans la variable du fichier de configuration, il sera créé deux nouveau fichiers. L'un se nommant `train_results.txt` et l'autre se nommant `test_results.txt`. Tout deux comporteront 3 colonnes :

- Le nom de l'image testée.
- La prédiction du nombre de poisson par l'algorithme.
- La vraie valeur du nombre de poissons.

A la fin des fichiers, l'erreur absolue moyenne et l'erreur quadratique moyenne entre les résultats trouvés et les résultats attendus sont données.

### 2.2.4 Automatiser la génération du fichier d'apprentissage, l'entraînement et le test

Il est possible d'automatiser les trois actions précédemment décrite en utilisant le script shell nommé `train-test.sh` mis à disposition dans le dossier `src/main/scripts/` dans les sources.

Ce script exécute les trois commandes python présentées précédemment et redirige les sorties standards et les sorties d'erreur vers un fichier de log nommé `ccnn_logs.log` situé dans le dossier `genfiles/logs/`.

Pour exécuter ce script LINUX il faut se placer dans le dossier des sources depuis un terminal et exécuter la commande suivante :

```
bash src/main/scripts/train-test.sh .
```



# G

## Dossier de tests

### 1 Les tests fonctionnels

Les tests unitaires ont été réalisés avec la bibliothèque *unittest*.

Seule la fonction *genDensePos()* a été testée avec des tests unitaire. La raison est qu'elle était plus complexe a développer.

La manière dont j'ai développé cette fonction a été avec ce que l'on appelle le "*Test-driven development*". Cela consiste à écrire les tests avant de commencer à développer et l'objectif et de passer les tests unitaires.

Il y a 7 tests unitaires, chacun d'entre eux compare les positions générées par la fonction avec les position attendues.

Numéro Test		Nom Fonction Test	Type Test	Fonctionnalités testées	Scénarios de test		Vérifications
T01		testgenDensePos1	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 5x7 pixels, des patches d'une taille de 2x2 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
					On a une image d'une taille de 5x6 pixels, des patches d'une taille de 2x2 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
T02		testgenDensePos2	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 2x2 pixels, des patches d'une taille de 2x2 pixels avec un décalage de 1 pixels en hauteur et 1 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
T03		testgenDensePos3	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 4x4 pixels, des patches d'une taille de 2x2 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
T04		testgenDensePos4	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 5x7 pixels, des patches d'une taille de 3x3 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
T05		testgenDensePos5	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 5x7 pixels, des patches d'une taille de 3x3 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.

T06		testgenDensePos6	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 5x8 pixels, des patches d'une taille de 3x3 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
					On a une image d'une taille de 6x8 pixels, des patches d'une taille de 3x3 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.
T07		testgenDensePos7	Fonctionnel	La génération des positions des patches dans une image	On a une image d'une taille de 5x8 pixels, des patches d'une taille de 3x3 pixels avec un décalage de 2 pixels en hauteur et 2 pixels en largeur à chaque nouveaux patches. On vérifie que les positions retournées correspondent bien en prenant compte de la dernière colonne de patches si il y a un reste de pixels non couverts en fin de ligne. De même pour la dernière ligne et le coin en pas à droite.		Les positions correspondent.

Figure 1 – Description des test

## 2 Test depuis l'interface

Avant de lancer l'entraînement du réseau de neurones j'ai intégré les poids pré-entraînés du CCNN sur le dataset Transcos. Je parle des résultats que j'ai obtenu en testant avec des images de ce dataset dans la partie [Section 1](#) (Chapitre 3) - [Test sur le dataset transcos](#).

## 1 Convention de nommage

Tout d'abord tout le code (variables, fonctions, classes ...) et les commentaires sont écrits en anglais.

### 1.1 Variables

Les variables suivent les conventions de nommage suivantes :

- Toutes les lettres sont en minuscule.
- Les mots sont séparés par un underscore "\_".

Exemple : `im_train_list_file`

### 1.2 Fonctions

Les fonctions suivent les conventions de nommage suivantes :

- La première lettre de chaque mot est une majuscule, le reste est en minuscule.
- Les fonctions commencent par une minuscule, c'est à dire que la première lettre du premier mot est une minuscule.

Exemple : `resizeDensityPatch`

### 1.3 Classes

Les classes suivent les conventions de nommage suivantes :

- La première lettre de chaque mot est une majuscule, le reste est en minuscule.
- Le nom de la classe commence toujours par une majuscule.

Exemple : `CcnnModel`

## 1.4 Commentaires

Les commentaires suivent aussi une convention. Chaque fonction doit être commentée, et doit se servir les mots clés suivants :

- **@brief** Donne une brève description de ce que fait la fonction
- **@param** Pour chaque paramètre on donne le nom du paramètre et sa description.
- **@return** Ce que la fonction retourne.

Exemple :

```
def generateDensityMapRectangle(image_shape, rectangles, sigma_rect):
    """
    @brief: Generate a density map of an image using the coords of the rectangles
    @param image_shape: The image shape.
    @param rectangles: The list of Rectangles in the image.
    @return: a density map.
    """
```

Figure 1 – Une fonction et ses commentaires

## 2 Structure des sources

La structure de mes sources est expliquée dans la partie [Section 4](#) (Chapitre 1) - [Structure des sources](#).

## 3 Le diagramme de classe

Le diagramme de classe de mes sources est décrit dans la partie [Section 3](#) (Chapitre 1) - [Diagramme de classe](#).

## 4 La documentation

La documentation Python des sources a été générée avec PyDoc au format HTML. Celle-ci se trouve dans le dossier *doc/* des sources.



## Comptes rendus hebdomadaires

### Compte rendu n°1 du 21/09/2017

Cette semaine j'ai débuté le projet en lisant le livre "Technosup, Reconnaissance des formes". J'ai lu les chapitres 1, 4 et 7 et je suis actuellement en train de lire le chapitre 8. Cela est plus que je ne pensais à lire car il faut souvent lire plusieurs fois avant de bien comprendre et des fois chercher les définitions sur internet.

Ces deux premiers jours m'ont permis d'avoir une vision globale de ce qu'est la reconnaissance de formes que ce soit l'extraction des caractéristiques, l'apprentissage supervisé ou la décision.

La semaine prochaine je compte finir le chapitre 8 et me renseigner sur le transfert learning (ce qui va nous être utile si on utilise des réseaux profonds pour notre projet).

### Compte rendu n°2 du 28/09/2017

Jeudi j'ai terminé de lire et de bien comprendre les chapitres 1, 4, 7 et 8 du livre "Technosup, Reconnaissance des formes". J'ai bien avancé dans la documentation des réseaux convolutifs (deep learning) et ses applications pour la reconnaissance de formes notamment avec des articles et des conférences de Yann Lecun. Cependant il reste quelques zones floues car c'est un sujet assez technique. J'ai commencé à aborder aussi le transfert learning.

### Compte rendu n°3 du 05/10/2017

Cette semaine j'ai installé l'environnement Anaconda sur mon ordinateur dans le but de tester des algorithmes de machine learning. J'ai créé un environnement "tensorflow" où j'ai installé Tensorflow et Keras. J'ai pu tester plusieurs algorithmes de deep learning entraînés sur ImageNet. J'ai aussi regardé une conférence de Yann Lecun.

### Compte rendu n°4 du 12/10/2017

Cette semaine je me suis consacré à la rédaction d'une première version de mon cahier de spécification et de mon état de l'art. J'ai vu M. Donatello pour clarifier certains points sur le projet à propos du cahier de spécification.

### Compte rendu n°5 du 19/10/2017

Jeudi j'ai travaillé sur les méthodes de régression linéaire en machine learning. J'ai tout d'abord fait une synthèse de leur fonctionnement pour mon état de l'art et j'ai ensuite testé un

petit réseaux utilisant la régression linéaire pour prédire des sorties numériques. J'ai ensuite fait beaucoup de recherche pour trouver une utilisation des algorithmes de régression (pas forcément linéaire) pour compter des objets dans une image. Vendredi je me suis intéressé à l'estimation du nombre d'objet dans une images en générale, sans utiliser la régression. Il existe des réseaux qui permettent de faire de la détection d'objets dans une image. Ce qui consiste à tracer un rectangle autour des différents objets reconnus dans l'image. J'ai récupéré un projet en Keras qui permet de détecter plusieurs objets dans une images utilisant des réseaux de type Faster-RCNN.

<https://github.com/yhenon/keras-frcnn>

Cependant, pour arriver à ce résultat, le réseaux a été entraîné avec le type d'objet (ex : voiture) et la position de cette objet dans l'image (2 points pour tracer le rectangle). Nos images ne possède quant à elles que le nombre de silure dans l'image mais pas leurs positions, n'est ce pas ? Je ne sais pas si cette solution sera alors exploitable ou si il y a un moyen de contourner le problème.

#### **Compte rendu n°6 du 26/10/2017**

Jeudi matin j'ai étudié les méthodes d'estimation du nombre d'objets dans une image en utilisant la régression au lieu de la détection des objets. La détection des objets dans une image est un problème complexe qui demande beaucoup d'images d'apprentissage. Les méthodes de régression utilise directement les caractéristiques globales de l'image pour estimer le nombre d'objets dans l'image. Cela demande beaucoup moins d'images d'apprentissage et encore moins si l'on indique la position des objets dans l'image. Ces méthodes sont très utilisés pour estimer le nombre de personnes dans une foule.

Le jeudi après midi j'ai assisté aux réunions concernant le cahier de spécification et l'état de l'art.

Le vendredi matin j'avais rendez-vous avec M. Donatello pour lui posé quelques questions. Nous avons notamment parler du format des images d'apprentissage et de la façon dont l'algorithme pourrait fonctionner.

Le reste de la journée a été consacré au cahier de spécification.

#### **Compte rendu n°7 du 09/11/2017**

Cette semaine j'ai consacré mes deux journée à la rédaction de mon PRD. J'ai globalement bien avancé il me reste à créer un diagramme de Gantt (le découpage du projet en tâche étant déjà fait) avant de faire vérifier le cahier auprès de Monsieur Soukhal et auprès de vous.

#### **Compte rendu n°8 du 16/11/2017**

Cette semaine j'ai consacré le mercredi à la rédaction de mon cahier de spécification. Il me manquait plus de chose que ce que je ne pensais (Méthodologie + structure générale avec diagramme de classe et de usecase + diagramme de Gantt). Le jeudi j'ai commencé à avancer dans mon état de l'art et j'ai été au Forum des entreprises.

#### **Compte rendu n°9 du 23/11/2017**

Cette semaine j'ai consacré le mercredi à la rédaction de mon état de l'art. Le jeudi j'ai vu M. Soukhal qui m'a fait quelques remarques sur mon cahier de spécification. J'ai aussi eu un entretien avec M. Donatello pour discuter des points évoqués par M. Soukhal sur mon cahier de spécification, notamment le nombre d'algorithmes que je dois implémenter. Le jeudi après midi j'ai continué mon Etat de l'art, j'espère l'avoir fini la semaine prochaine.

**Compte rendu n°10 du 30/11/2017**

Jeudi matin j'ai assisté à la réunion de présentation des options de S10 de 8h à 10h. Nous avons ensuite une réunion bilan de projet collectif avec M. Venturini de 11h à 12h30. Le jeudi après midi j'ai avancé dans mon Etat de l'art. Le vendredi matin j'ai vu M. Soukhal pour mon cahier de spécification, j'ai donc travaillé dessus le reste de la matinée. L'après midi fut consacré à la rédaction de mon état de l'art.

**Compte rendu n°11 du 07/12/2017**

Jeudi et Vendredi ont été consacré à la rédaction de mon état de l'art et de mon analyse. Vendredi j'ai vu M. Conte, nous avons fait le point sur le cahier de spécification, l'état de l'art et l'analyse. Nous avons revu : - Comment l'algorithme sera intégré dans l'IHM avec le diagramme de classe de Yaofutian LU. - Le nombre d'images dans l'échantillon d'apprentissage. - La validation de mon analyse c'est à dire du système proposé, avec les prétraitements sur les images, le modèle du réseau de neurones, les sorties et les entrées, etc. - Le découpage en tâche et le diagramme de Gantt. - Ce qu'il faut dire à la présentation.



# Webographie

- [WWW1] URL : [www.li.univ-tours.fr](http://www.li.univ-tours.fr).
- [WWW2] URL : <http://redmine.polytech.univ-tours.fr/projects/analysepoisson>.
- [WWW3] URL : <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>.
- [WWW4] URL : <http://ataspinar.com/2016/12/22/the-perceptron/>.
- [WWW5] URL : <https://www.slideshare.net/ankitksharma/svm-37753690>.
- [WWW6] URL : [https://fr.wikiversity.org/wiki/R%C3%A9seaux\\_de\\_neurones/Les\\_neurones\\_en\\_r%C3%A9seaux](https://fr.wikiversity.org/wiki/R%C3%A9seaux_de_neurones/Les_neurones_en_r%C3%A9seaux).
- [WWW7] URL : [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/neural\\_networks.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/neural_networks.html).
- [WWW8] URL : <https://keras.io/optimizers/>.
- [WWW9] URL : <https://www.kdnuggets.com/2017/08/train-deep-learning-faster-snapshot-ensembling.html>.
- [WWW10] URL : <https://keras.io/losses/>.
- [WWW11] URL : <https://www.slideshare.net/oeuia/neural-network-as-a-function>.
- [WWW12] URL : <https://www.slideshare.net/fcollova/introduction-to-neural-network>.
- [WWW13] URL : <http://image-net.org/challenges/LSVRC/2012/results.html>.
- [WWW14] URL : [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/dropout\\_layer.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/dropout_layer.html).
- [WWW15] URL : [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/convolutional\\_neural\\_networks.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/convolutional_neural_networks.html).
- [WWW16] URL : <http://cs231n.github.io/convolutional-networks/#pool>.
- [WWW17] URL : <http://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>.
- [WWW18] URL : [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/deep\\_learning.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/deep_learning.html).
- [WWW19] URL : <http://www.robots.ox.ac.uk/~vgg/research/counting/>.

- [WWW20] URL : [https://keras.io/losses/#mean\\_squared\\_error](https://keras.io/losses/#mean_squared_error).
- [WWW21] URL : <https://github.com/gramuah/ccnn>.
- [WWW22] URL : <http://agamenon.tsc.uah.es/Personales/rlopez/data/trancos/>.
- [WWW23] Yann LeCUN. URL : <http://www.di.ens.fr/willow/events/cvml2013/materials/slides/tuesday/lecun-20130723-vrml-01.pdf>.

# Bibliographie

- [1] Lokesh BOOMINATHAN, Srinivas S S KRUTHIVENTI et R. Venkatesh BABU. « CrowdNet : A Deep Convolutional Network for Dense Crowd Counting ». In : (August 2016). URL : <https://arxiv.org/pdf/1608.06197.pdf>.
- [2] Kaiming HE, Georgia Gkioxari and Piotr DOLLAR et ROSS GIRSHICK. *Mask R-CNN*. Rapp. scient. Facebook AI Research (FAIR), 5 avr. 2017. URL : <https://arxiv.org/pdf/1703.06870.pdf>.
- [3] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN. *Deep Residual Learning for Image Recognition*. Rapp. scient. Microsoft Research, 2015. URL : <https://arxiv.org/pdf/1512.03385.pdf>.
- [4] Carlos Xavier HERNANDEZ et Mohammad M. SULTAN. *Using Deep Learning for Segmentation and Counting within Microscopy Data*. Stanford University. URL : <http://cs231n.stanford.edu/reports/2017/pdfs/533.pdf>.
- [5] Sergey IOFFE et Christian SZEGEDY. « Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift ». In : (2015). URL : <https://arxiv.org/pdf/1502.03167v3.pdf>.
- [6] Diederik P. KINGMA et Jimmy Lei BA. *ADAM : A METHOD FOR STOCHASTIC OPTIMIZATION*. Rapp. scient. University of Amsterdam, 23 juil. 2015. URL : <https://arxiv.org/pdf/1412.6980v8.pdf>.
- [7] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E. HINTON. « ImageNet Classification with Deep Convolutional Neural Networks ». In : (2012). URL : <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [8] Yann LECUN. « Pourquoi l'apprentissage profond ? » In : Collège de France, 2017. URL : <https://www.youtube.com/watch?v=5F5Iz1p06Xw>.
- [9] Victor LEMPITSKY et Andrew ZISSERMAN. « Learning To Count Objects in Images ». In : ().
- [10] Laurence LIKFORMAN-SULEM et Elisa BARNEY SMITH. *Reconnaissance des formes. Théorie et pratique sous Matlab*. Sous la dir. d'ELLIPSES. 2013.
- [11] Tsung-Yi LIN, Piotr DOLLAR, ROSS GIRSHICK, Kaiming HE, Bharath HARIHARAN et Serge BELONGIE. *Feature Pyramid Networks for Object Detection*. Facebook AI Research (FAIR), 19 avr. 2017. URL : <https://arxiv.org/pdf/1612.03144.pdf>.

- [12] Daniel ONORO-RUBIO et Roberto J. LOPEZ-SASTRE. *Towards perspective-free object counting with deep learning*. Rapp. scient. GRAM, University of Alcala, Alcala de Henares, Spain, 2016. URL : <http://agamenon.tsc.uah.es/Investigacion/gram/publications/eccv2016-onoro.pdf>.
- [13] Karen SIMONYAN et Andrew ZISSERMAN. « VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION ». In : *ICLR* (10 avr. 2015). URL : <https://arxiv.org/pdf/1409.1556.pdf>.
- [14] C. ZHANG, H. LI, X. WANG et X YANG. « Cross-scene crowd counting via deep convolutional neural networks ». In : *CVPR* (June 2015).

# Estimation de la densité de poissons dans les rivières par analyse d'images aériennes

Mikael Moreau

Encadrement : Donatello Conte et Pierre Gaucher

## Objectifs

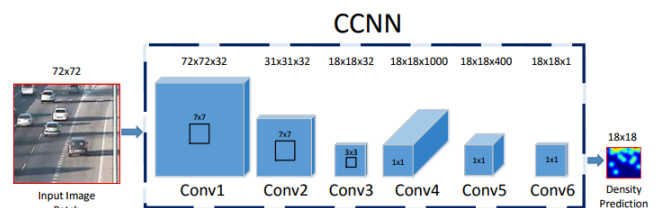
- Concevoir un système de reconnaissance de formes pour estimer la densité de silures à partir d'une image aérienne.
- Implémenter le système avec la bibliothèque de deep learning Keras.
- Intégrer le système à une IHM.



Logo du Laboratoire Informatique de Tours

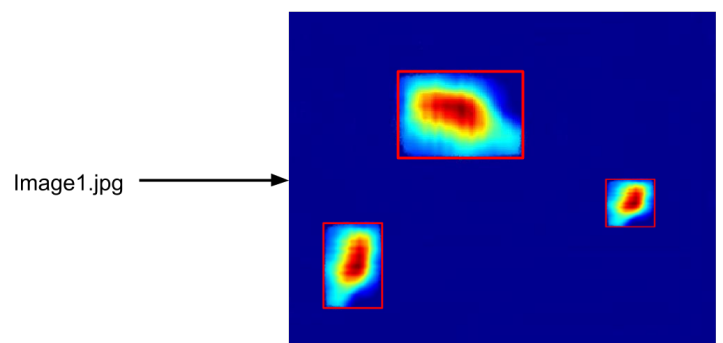
## Mise en œuvre

Le système de reconnaissance de formes utilise des réseaux convolutifs minimisant une fonction de régression avec un algorithme de descente de gradient. Ce réseau est entraîné avec un échantillon d'apprentissage où est annotée la position des poissons dans l'image. Ce système permet de prédire une image de densité à partir de l'image d'entrée.



## Résultats attendus

- Écrire le cahier de spécifications
- Faire un état de l'art
- Proposer l'architecture du système et l'implémenter
- Entraîner le système avec des images annotées
- Tester les performances du système
- Intégrer le système à l'IHM



# Estimation de la densité de poissons dans les rivières par analyse d'images aériennes

Mikael Moreau

Encadrement : Donatello Conte et Pierre Gaucher

## Objectifs

- Concevoir un système de reconnaissance de formes pour estimer la densité de silures à partir d'une image aérienne.
- Implémenter le système avec la bibliothèque de deep learning Keras.
- Intégrer le système à une IHM.

## Mise en œuvre

Le système de reconnaissance de formes utilise des réseaux convolutifs minimisant une fonction de régression avec un algorithme de descente de gradient. Ce réseau est entraîné avec un échantillon d'apprentissage où est annoté la position des poissons dans l'image. Ce système permet de prédire une image de densité à partir de l'image d'entrée.

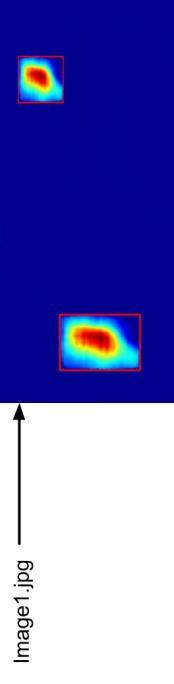
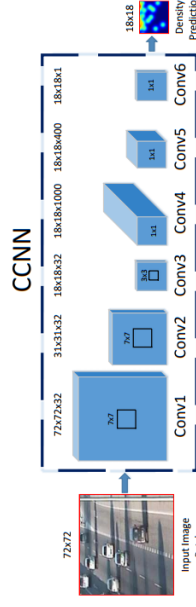
## Résultats attendus

- Écrire le cahier de spécifications
- Faire un état de l'art
- Proposer l'architecture du système et l'implémenter
- Entraîner le système avec des images annotées
- Tester les performances du système
- Intégrer le système à l'IHM

# LI

Laboratoire d'Informatique  
EA 6300

Logo du Laboratoire Informatique de Tours



# Estimation de la densité de poissons dans les rivières par analyse d'images aériennes

## Résumé

Des aménageurs en écologie souhaitent étudier la biodiversité des fleuves et des rivières et en particulier la population de silures. Les silures forment un genre de poisson d'eau douce de la famille des Siluridae. Pour étudier leur population, les aménageurs ont décidés de prendre des photos aériennes des fleuves et rivières à l'aide d'un drone. Pour faciliter l'estimation du nombre de silures dans une photo, les aménageurs ont fait appel au Laboratoire Informatique de Tours pour concevoir un logiciel permettant de calculer cette estimation. Ce document présente un système de reconnaissance de formes pour estimer la densité de silures dans une image en utilisant une méthode d'apprentissage profond.

## Mots-clés

apprentissage profond, silure, densité, estimation, image

## Abstract

Ecological planners want to study the biodiversity of rivers and rivers and in particular the population of silurid fish. The silurid fish form a kind of freshwater fish of the Siluridae family. To study their population, ecological planners decided to take aerial photos of rivers and streams using a drone. To make it easier to estimate the number of silurid fish in a photo, they called on the Tours Computer Laboratory to design software to calculate this estimation. This document presents an image recognition system for estimating the density of silures in an image using a deep learning method.

## Keywords

deep learning, silurid fish, density, estimation, image

**Tuteurs académiques**

Donatello CONTE

Pierre GAUCHER

**Étudiant**

Mikael MOREAU (DI5)