

UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
MOHAMED ISLEM RAMOUL

DÉVELOPPEMENT D'UN SYSTÈME INTELLIGENT POUR LA DÉTECTION
ET LA CLASSIFICATION DES GRAINES DE SEMENCES DE RÉSINEUX

JUILLET 2021

RÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

François Meunier, Directeur de comité de programme de cycles supérieurs.
Département de mathématiques et informatique.

Usef Faghihi, Professeur.
Département de mathématiques et informatique.

Louis Houde, Professeur.
Département de mathématiques et statistique.
Département de mathématiques et informatique.

REMERCIEMENTS

Avant toute chose, je remercie l'Université du Québec de Trois-Rivières de m'avoir accepté au sein de son établissement, ce fut un grand honneur d'avoir terminé mes études avec ses professeurs dont leur cours étaient de qualité et tout son personnel qui était présent et à l'écoute de toutes questions qui se sont posées.

Je remercie mon directeur de recherche, le professeur François Meunier, grâce à lui, j'ai pu découvrir la vision par ordinateur, un tout nouveau domaine de l'informatique pour moi, que j'ai pu mieux le maîtriser avec sa grande aide. Il était à l'écoute de toute proposition venant d'un amateur du domaine, sa clarté des objectifs attendus et malgré des difficultés rencontrées, ses commentaires étaient toujours instructifs qui donnent du courage à avancer encore plus loin. Je remercie également le directeur d'unité de recherche du laboratoire d'intelligence artificielle appliquée, monsieur Ismail Biskri, le chef de la section informatique Amar Bensaber Boucif et ainsi la professeure madame Ghazali Nadia pour leur contribution dans mon projet de maîtrise, qui ont pris le temps de répondre à toutes mes questions qui ont un rapport avec mes recherches de maîtrise de fin d'études. Je ne peux ignorer l'aide de Ahmed Belatik, Younes Mesfioui, Youssef Ammar et ainsi tous mes amis pour leur grande aide que ce soit financière ou leur motivation et surtout à toute ma famille dont mon père Ramoul said, ma mère Nora et mon frère Oualid qui ont été tout le temps présents pour m'aider à dépasser tout obstacle d'étude et de la vie.

DEVELOPPEMENT D'UN SYSTEME INTELLIGENT POUR LA DETECTION ET LA CLASSIFICATION DES GRAINES DE SEMENCES DE RESINEUX

MOHAMED ISLEM RAMOUL

RÉSUMÉ

L'être humain utilise sa vision pour analyser et contrôler toute information reçue de la nature. Avec la grande avancée technologique, les ordinateurs sont ainsi utilisés pour accomplir cette tâche, et cela de façon automatique et c'est alors que la vision par ordinateur a vu le jour. Ce domaine est largement utilisé dans différentes disciplines, telles que la détection des panneaux de signalisation de la route, prédiction météo, reconnaissance de visages et le suivi de la production en agriculture ou en foresterie.

La production en foresterie est le domaine de notre mémoire. L'objectif principal de cette recherche est de pouvoir déterminer le nombre de graines dans un plateau d'ensemencement à partir d'une image proche infrarouge dans le but d'améliorer l'ensemencement en détectant les erreurs (cellules manquantes de graines) de façon automatique.

Deux méthodes furent implémentées, la première est la reconnaissance statistique de formes qui se base sur des données statistiques expérimentales des caractéristiques des semences et la deuxième est la reconnaissance de formes basées sur les réseaux de neurones convolutifs où on applique un apprentissage machine profond, qui extrait automatiquement les caractéristiques de formes des graines avec une précision de comptage obtenue d'approximativement 93% et 97% respectivement.

Mots-clés : Vision par ordinateur, détection de semences des résineux, reconnaissance statistique de formes, reconnaissance de formes basées sur les réseaux de neurones convolutifs (CNN).

ABSTRACT

Human beings use their vision to analyze and control any information received from nature. With the great technological advance, computers are thus used to accomplish this task automatically. This field is widely used in various disciplines, such as road sign detection, weather prediction, face recognition and agricultural or forestry production monitoring.

Forestry production is the domain of this master thesis. The main objective of this research is to be able to detect the number of seeds in a tray from an infrared image in order to improve seeding by detecting errors (missing seed cells) automatically.

Two methods have been implemented, the first is the static shape recognizer which is based on experimental static data of the characteristics of the seeds and the second is the pattern recognition based on convulsive neural networks where we apply a deep machine learning which automatically extracts the dominant characteristics of the seeds, an accuracy obtained of approximately 93% and 97% respectively.

Keywords: Computer vision, softwood seed detection, static pattern recognition, pattern recognition based on convulsive neural networks (CNN).

TABLE DES MATIERES

CHAPITRE 1 INTRODUCTION	13
1.1 L'industrie bioalimentaire et forestière.....	13
1.1.1 Processus de la production.....	14
1.1.1.2 Déficiences du processus d'ensemencement.....	16
1.2 Correctifs proposés au système antérieur.....	16
CHAPITRE 2 REVUE DE LITTERATURE	21
2.1 Introduction.....	21
2.2 Reconnaissance de formes statistiques	22
2.2.1 Fonctionnement.....	22
2.2.2 Résultats expérimentaux connexes	23
2.3 Reconnaissance de formes basées sur les neurones.....	24
2.3.1 Apprentissage automatique	24
2.3.1.1 Histoire.....	24
2.3.1.2 Apprentissage supervisé.....	25
2.3.1.3 Apprentissage profond (<i>Deep Learning</i>).....	25
i. CNN (Réseau de neurones convolutif).....	26
ii. Structure du CNN.....	26
i. Couches convolutives (<i>Convolution Layers</i>)	26
ii. Couche de mise en Commun (<i>Pooling layers</i>).....	28
iii. Couche d'entrée pour le réseau de neurones artificiel (<i>Flattening</i>).....	29
iv. Couche entièrement connectée (<i>Fully connected layer</i>).....	30
2.3.2 Résultats expérimentaux connexes	31
2.4 Traitement d'image.....	31
2.4.1 Réduction du bruit	32
2.4.2 Seuillage binaire adaptatif.....	33
2.4.3 Morphologie mathématique	34
2.4.4 Détection de contours	36
2.4.4.1 Analyse de forme géométrique.....	37

i.	La taille	38
ii.	La moyenne d'illumination	38
iii.	La convexité	39
iv.	Forme Convexe	39
v.	Forme Concave.....	40
2.5	Bibliothèque graphique.....	41
2.6	Langage de programmation	42
2.6.1	C#.....	42
2.6.2	Python	42
2.6.2.1	Keras	43
2.6.2.2	Tensorflow	43
2.6.2.3	Numpy.....	43
2.6.2.4	Matplotlib.....	44
2.7	Conclusion	44

CHAPITRE 3 MATÉRIEL ET MÉTHODOLOGIE..... 45

3.1	Introduction.....	45
3.2	Matériaux expérimentaux	46
3.2.1	Epinette noire	46
3.2.2	Epinette rouge.....	46
3.2.3	Pin rouge	47
3.2.4	Pin blanc.....	47
3.2.5	Processus d'ensemencement	48
3.2.6	Caméra spectroscopique dans le proche infrarouge.....	50
3.2.7	Ordinateur	50
3.2.8	Logiciels.....	51
3.2.8.1	Visual studio 2017	51
3.2.8.2	Pycharm.....	51
3.3	Méthodologie	51
3.3.1	Reconnaissance statistique de formes	51
3.3.1.1	Segmentation	52
3.3.1.2	Filtre de surface	54
3.3.1.3	Filtre de luminosité.....	56
3.3.1.4	Filtre de forme	59

3.3.1.5	Classification.....	62
3.3.2	Reconnaissance de formes basées sur les réseaux de neurones convolutifs (Apprentissage profond)...	67
3.3.2.1	Segmentation.....	68
3.3.2.2	Architecture du réseau de neurones convolutif	68
3.3.2.3	Entraînement du réseau de neurones convolutif.....	71
i.	Préparation des données.....	71
ii.	Entraînement	73
iii.	Optimisation.....	74
3.4	Structure globale et classification.....	76
3.5	Conclusion.....	78

CHAPITRE 4 TESTS ET ANALYSES.....80

4.1	Introduction	80
4.2	Test de performance.....	80
4.3	Comparatif des résultats.....	82
4.4	Discussion des résultats	83
4.5	Conclusion.....	83

CONCLUSION85

BIBLIOGRAPHIES.....87

LISTE DES TABLEAUX

Tableaux 3.1	Caractéristique des données d’entraînement.....	73
Tableaux 3.2	Les différents résultats de prédiction du modèle en phase d’entraînement	74
Tableaux 3.3	La structure et la performance du système de reconnaissance basé sur les neurones	76
Tableaux 4.1	Définition des métriques	81
Tableaux 4.2	Les résultats de performance des deux systèmes	82

TABLE DES FIGURES

Figure 1.1	Pépinière forestière de Berthierville.....	14
Figure 1.2	Le processus de l'assurance qualité	15
Figure 1.3	Illustration du projet.....	18
Figure 1.4	Plateau d'ensemencement sous illumination NIR	18
Figure 1.5	Une Graine sous NIR	19
Figure 1.6	Un résidu sous NIR	19
Figure 2.1	Comment l'image est-elle interprétée par un ordinateur.....	27
Figure 2.2	La nouvelle matrice après l'application du filtre 3*3*1.....	27
Figure 2.3	Les résultats des deux types du polling (2*2*1).....	28
Figure 2.4	La couche d'entrée pour le réseau de neurones artificiel	29
Figure 2.5	Un réseau de neurones artificiel pour la classification.....	30
Figure 2.6	Un bruit de chrominance sur un plateau d'ensemencement.	32
Figure 2.7	Un bruit de luminance sur un plateau d'ensemencement	32
Figure 2.8	Effet du filtre gaussien dans le processus de segmentation.....	33
Figure 2.9	Seuillage binaire adaptative du plateau d'ensemencement	34
Figure 2.10	Résultat des quatre d'opérateurs appliqués sur une même image.....	36
Figure 2.11	Différence entre les deux méthodes d'approximation des contours	37
Figure 2.12	Les trois façons possibles de positionnement d'une graine.....	38
Figure 2.13	La différence de forme entre une graine et un résidu végétal.....	40
Figure 2.14	Le visuel des deux formes géométriques basées sur la convexité	41
Figure 3.1	Visuel de graines d'épinette noire	46
Figure 3.2	Visuel de graines d'épinette rouge.....	47
Figure 3.3	Visuel de graines de pin rouge.....	47
Figure 3.4	Visuel de graines de pin blanc	48
Figure 3.5	Les différents états d'une cellule.....	49
Figure 3.6	Le processus de détection des graines	52
Figure 3.7	Résultat de la segmentation de l'image d'entrée	53
Figure 3.8	Résultat du filtre de surface	56
Figure 3.9	La procédure d'extraction de la luminosité	57
Figure 3.10	Résultat du filtre de luminosité.....	58
Figure 3.11	Illustration sur le calcul de la distance du creux (concavité).....	59

Figure 3.12	Détection de forme à partir du programme de reconnaissance statistique .	61
Figure 3.13	Illustration sur le fonctionnement d'un perceptron	62
Figure 3.14	Plan cartésien sur la séparation de deux classes (classe A et B)	63
Figure 3.15	Déroulement de la classification sous le programme de reconnaissance statistique	64
Figure 3.16	Interface du système de reconnaissance statistique	67
Figure 3.17	Architecture du réseau de neurones convolutifs.....	69
Figure 3.18	Une graine après un agrandissement de 6 pixels.....	72
Figure 3.19	Une donnée nuisible.....	76
Figure 3.20	L'arrêt précoce.....	76
Figure 3.21	Interface du système de reconnaissance basée sur les neurones	78

LISTE DES ALGORITHMES

Algorithme 3.1	Segmentation de l'image d'entrée	53
Algorithme 3.2	Fonction d'extraction de la taille du contour d'un objet	54
Algorithme 3.3	Fonction d'extraction de la luminosité d'un objet	57
Algorithme 3.4	Calcul de la distance du creux (concavité)	60
Algorithme 3.5	Un perpectron écrit en C#	65
Algorithme 3.6	La procédure de segmentation	68
Algorithme 3.7	Conception du modèle d'apprentissage en python	70
Algorithme 3.8	Un zoom de 6 pixels sur une image	72
Algorithme 3.9	La fonction de prédiction	77

CHAPITRE 1

INTRODUCTION

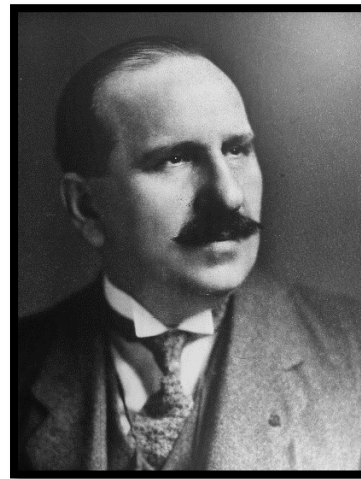
1.1 L'industrie bioalimentaire et forestière

Le Québec consacre chaque année une superficie de terre de 933 000 hectares pour produire environ 5.1 millions de tonnes de graines issues du maïs, soya, blé et le canola dont 90% de la production est destinée à l'alimentation des animaux d'élevage.

D'autre part, le Québec doit aussi produire des graines de semences d'arbres pour la régénération des forêts. Pour ce faire, la province gère quatre pépinières forestières, dont une, à Berthierville.

La pépinière de Berthierville (voir fig 1.1) fut la première pépinière forestière de la province fondée en 1908 par Gustave Clodomir Piché, qui se situe sur la route 138, à quatre kilomètres au sud-ouest de la ville de Berthierville. Une superficie de 155 hectares est consacrée à la production de plantes résineuses en récipients spécialement les épinettes blanches, rouges, noires et des épinettes de Norvège ainsi que du pin blanc et rouge, des feuillus en récipients comme du bouleau jaune, bouleau à papier, chêne rouge, érable rouge, érable à sucre, frêne de Pennsylvanie et frêne d'Amérique, et enfin des feuillus à racines nues (dénuée de terre au pied) telles que le caryer cordiforme, chêne rouge, chêne à gros fruits, cerisier tardif, érable à sucre, noyer noir et peuplier.

Avec l'appui de plus de 200 employés, la pépinière produit chaque année en moyenne 4,5 millions de plants de fortes dimensions, dont 3 millions en récipients et 1,5 million à racines nues.



a) Vue satellitaire de la pépinière de Berthierville b) Son fondateur Gustave Picher

Figure 1.1 : Pépinière forestière de Berthierville

1.1.1 Processus de la production

Les processus de la pépinière sont nombreux et variés. Dans ce projet de recherche, nous sommes intéressés par l'amélioration des performances d'un semoir pneumatique et plus spécifiquement par l'optimisation du taux d'ensemencement de chaque cavité de plateau constitué de 12 rangées et 24 colonnes (288 cavités). L'objectif principal est alors de développer un système capable de détecter les cavités d'un plateau sortant du semoir pneumatique, n'ayant aucune ou plus d'une graine et d'ensuite assister le processus d'assurance qualité pour produire au bout de la chaîne d'ensemencement des plateaux ayant qu'une seule graine par cavité.

Le processus d'ensemencement idéal est schématisé à la figure 1.2. Les plateaux vides (voir fig 1.2 (a)) passent dans un convoyeur où un planteur pneumatique (voir fig 1.2 (b))

injecte idéalement une graine par cellule, ce processus est effectué par bloc de 4 lignes et 12 colonnes (48 cavités) qui nécessite son application à six reprises pour pouvoir emplir toutes les cavités d'un plateau complet. La règle principale est que chaque cavité du plateau doit contenir au moins une graine.

1.1.1.1 Le processus de l'assurance qualité

L'assurance qualité a pour rôle d'éviter le scénario envisagé précédemment en corrigeant les erreurs lors du processus d'ensemencement, et cela après que les plateaux aient été ensemencés par le semoir pneumatique. La correction se fait par des préposés à l'assurance qualité (voir fig 1.2 (c)) en inspectant chaque plateau qui défile sur le convoyeur. Chaque erreur détectée est corrigée manuellement.

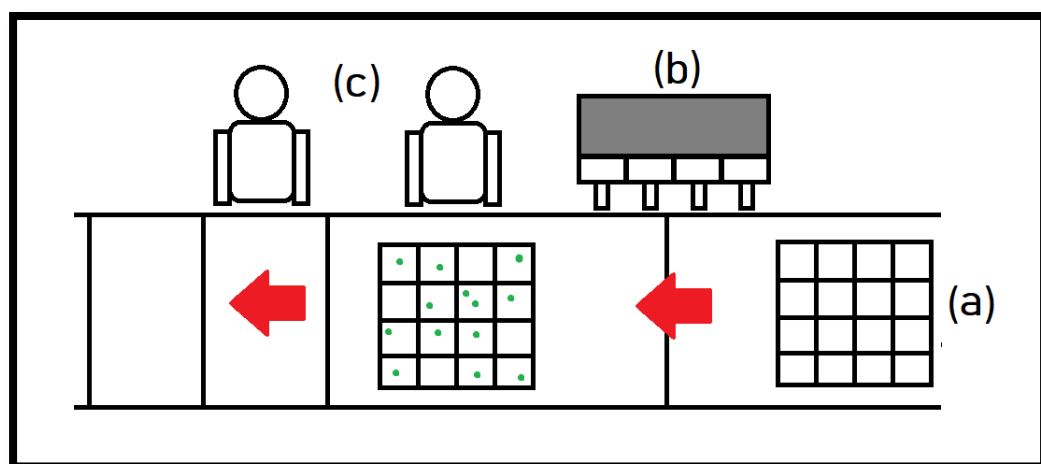


Figure 1.2 : Le processus de l'assurance qualité

Le plateau d'ensemencement (a) passe sous le planteur pneumatique (b). Ensuite, les préposés(e)s (c) vérifient l'état de chaque cellule

1.1.1.2 Déficiences du processus d'ensemencement

Le processus d'ensemencement nécessite plusieurs préposé(e)s, surtout lors de l'opération d'assurance qualité à la sortie du semoir pneumatique où le nombre de graines par cavité de chaque plateau est vérifié. Les préposé(e)s à l'assurance qualité doivent alors maintenir une posture adéquate, une grande concentration et observer les cavités en permanence, ce qui devient très fatigant pour le système visuel, sachant que chacun peut avoir à analyser visuellement près de 200 000 cavités en une seule journée.

Détecter les cavités vides n'est pas la seule problématique, des résidus végétaux incorporés au médium d'ensemencement (mousse de sphaigne) peuvent causer des problèmes de détection lors de la phase d'assurance qualité. Le système développé pour automatiser la phase d'assurance qualité doit alors pouvoir classifier tout objet présent dans les cavités et permettre de distinguer les graines des résidus végétaux.

Sachant que les graines et les résidus végétaux peuvent partager certaines caractéristiques comme la réflectance, la forme ou la taille. Le système automatique développé doit avoir une performance de classification très élevée si on souhaite améliorer la qualité du taux de germination du processus d'ensemencement de la pépinière.

1.2 Correctifs proposés au système antérieur

L'approche proposée dans des projets antérieurs [9] consiste à faire passer les plateaux de cavités (cellules) (voir fig 1.3 (1)) dans un caisson métallique, où une caméra proche-infrarouge (NIR) est intégrée (voir fig 1.3 (3)). Cette caméra NIR capture les images des plateaux de cavités préalablement ensemencés (voir fig 1.4). Ces images sont ensuite analysées par un système de vision artificielle (voir fig 1.3 (4)) doté d'un modèle de classification probabiliste implémenté par un ensemble de caractéristiques (l'aire, le périmètre, facteur de forme, la solidité et l'excentricité) favorisant l'extraction des formes

associées aux graines. Le nombre de graines segmentées dans chaque cavité d'ensemencement peut ensuite être compté et projeté à l'écran du système de vision.

Une cellule d'ensemencement peut avoir deux types d'états possibles, soit avec au moins une graine ou soit vide. Une cellule comportant au moins une graine peut alors comporter une ou plusieurs graines (voir fig 1.5). Cependant, dans certaines situations des résidus végétaux incorporés au médium d'ensemencement peuvent être confondus avec des graines étant donné leurs caractéristiques de forme similaires à celles des graines (voir fig 1.6).

Dans le contexte du projet de recherche présenté dans ce mémoire nous utilisons des caractéristiques extraites des images proche-infrarouge (NIR) capturées similaires à celles obtenues dans les versions antérieures du système de comptage de graine. Ces caractéristiques ont dans une première phase de cette recherche été utilisées pour modéliser un classificateur automatique telles que des réseaux de neurones pour permettre après un entraînement approprié, l'estimation du nombre de graines dans chaque cellule d'ensemencement et ce tout en minimisant le nombre de faux positifs pouvant être causés par la présence de matières végétales dans le milieu d'ensemencement comme observé dans les études antérieures [9] une sensibilité au rayonnement NIR. Dans une seconde phase de ce projet de recherche nous avons développé un système de comptage de graines de résineux basé sur un réseau de neurones convolutionnels qui permet au travers de couches convolutionnelles l'extraction automatique des caractéristiques de formes permettant la segmentation des graines.

Les résultats de la détection seront ensuite projetés sur un écran (voir fig. 1.3 (d)) où les personnes affectées à l'assurance qualité (voir fig. 1.3 (e)) corrigent les cellules défectueuses (cellules avec plus d'une graine et celles sans graine).

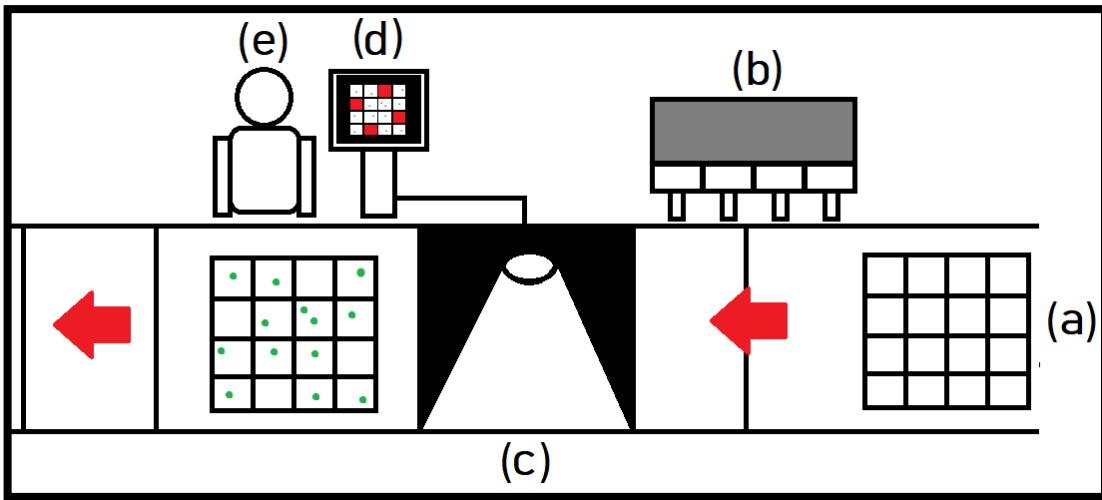


Figure 1.3 : Illustration du projet

Le plateau d'ensemencement (a) passe sous le planteur pneumatique (b), ou une caméra proche infrarouge (c) capte une image pour le système de vision intelligente (d). Enfin, les proposés(e)s (e) vérifient l'état de chaque cellule

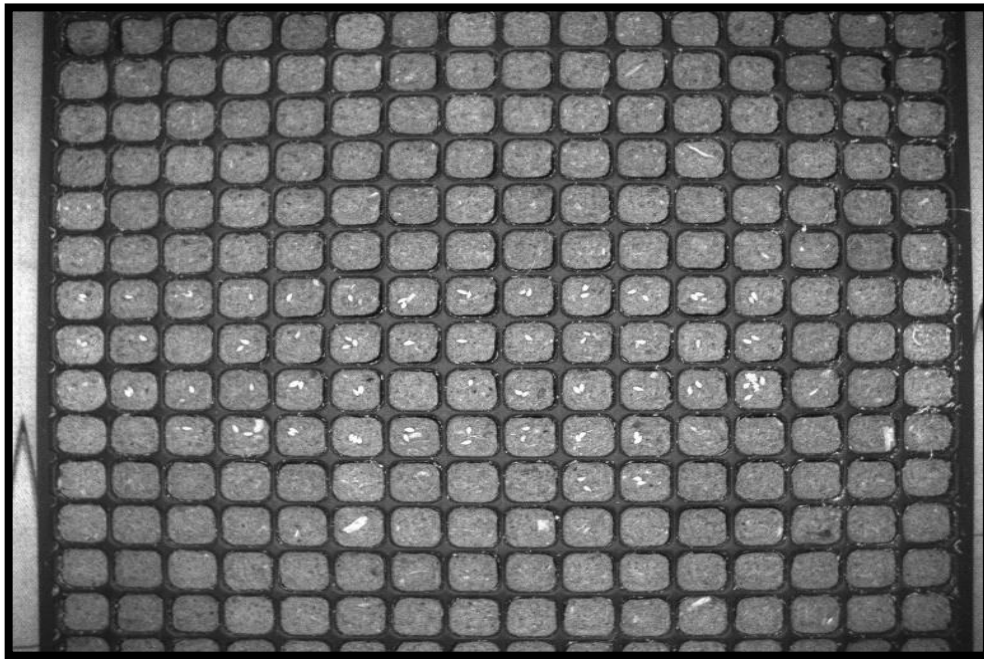


Figure 1.4 : Plateau d'ensemencement sous illumination NIR

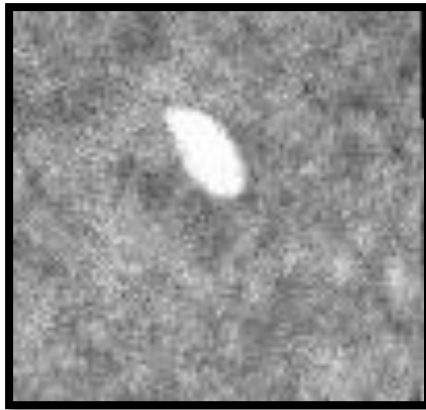


Figure 1.5 : Une Graine sous NIR



Figure 1.6 : Un résidu sous NIR

1.3 Sommaire des chapitres de ce mémoire

Nous allons détailler dans les chapitres de ce mémoire, toutes les procédures et démarches utilisées dans la conception de ce projet, passant d'abord par la revue de la littérature qui a inspiré notre travail de recherche, nous aborderons aussi quelques notions de base des systèmes de vision artificielle. Cette revue de la littérature sera exposée au chapitre 2.

Le chapitre 3 détaillera quant à lui la méthodologie utilisée pour le développement des procédures de notre système. Dans un premier temps, le processus de détection des objets présents dans les cellules sera abordé. Nous verrons alors les notions de segmentation permettant d'extraire les objets d'intérêt de l'arrière-plan. Des caractéristiques descriptives des objets d'intérêt seront utilisées ensuite comme input aux fonctions de filtrage qui classe les objets de façon à sélectionner ceux ayant une forte chance d'être des graines. La classification est la deuxième et dernière phase du système développé, basé sur deux méthodes :

1. Reconnaissance de formes statistiques : cette méthode utilise des algorithmes mathématiques afin d'extraire les caractéristiques (la taille, la moyenne de couleur, l'enveloppe convexe et la distance séparant cette dernière des points

concaves (défauts) et l'enveloppe convexe de chaque objet d'intérêt détecté dans la première phase. Ensuite, nous utilisons un simple perceptron afin de classier ces objets d'intérêt comme étant une graine ou un résidu végétal.

2. Reconnaissance de formes basées sur les réseaux de neurones convolutionnels (Deep learning) : la deuxième méthode consiste à utiliser un réseau de neurones convolutifs (CNN) qui prendra les images des objets détectés (et non l'image complète de la cellule) comme entrée. Avec le modèle parfaitement entraîné, le système nous affichera le résultat de la classification.

Nous passerons ensuite au chapitre 4, où nous allons discuter et analyser les résultats obtenus des deux approches de classification implémentées et les comparer pour déterminer celle qui est la plus performante pour notre projet.

Finalement, on conclut ce mémoire au chapitre 5, en présentant nos perspectives de notre travail et les idées de développements futurs dans l'optique de généraliser l'utilisation de notre système dans divers processus de production de la pépinière.

CHAPITRE 2

REVUE DE LITTERATURE

2.1 Introduction

Plusieurs chercheurs ont mené des expériences utilisant la vision par ordinateur en agriculture et en foresterie pour développer des techniques d'estimation de la qualité des graines de semences telles que, le riz [2] [5], le canola [11], les céréales [10], le soja [7] et la tomate [1].

Ce chapitre présente quelques-uns de ces travaux effectués ces dernières années qui ont inspiré la méthodologie de la présente recherche. Nous introduirons aussi la définition de certaines notions de base du traitement numérique des images appliquées dans ce présent projet de recherche. Nous présentons aussi les outils, les logiciels, ainsi que les langages de programmation utilisés.

2.2 Reconnaissance de formes statistiques

Cette méthode utilise les statistiques pour l'apprentissage d'un modèle de classification, ces données statistiques peuvent faire référence aux observations ou à des études expérimentales [29]. Cette approche fut très utilisée dans plusieurs domaines, tels que la reconnaissance de caractère, diagnostic médical, prédiction météorologique. Cette méthode a récemment été appliquée dans l'exploration de données « Data mining ». L'exemple le plus simple est la filtration ou la classification des courriers électroniques (email) qui est capable de classer un courriel comme étant une publicité ou une page frauduleuse (spam) en analysant son contenu (caractéristiques) :

- nom de l'objet : Gagner une voiture, Rabais hivernal.
- termes de ventes : Carte de crédit, gratuit, promotion,
- nom de l'expéditeur : Amazon, freeMoney.

2.2.1 Fonctionnement

. Cette approche est appuyée par un ou de plusieurs agents (ou filtres), où chaque agent utilise des algorithmes de mesure pour extraire les caractéristiques (ex : Forme géométrique, Couleur, Age, ...), qui serviront comme donnée d'entraînement aux algorithmes d'apprentissage (ex : support vector machine. KNN, Naive Bayes, etc.), afin de prédire l'estimation ou de prendre une décision de la classe d'appartenance de la nouvelle instance.

2.2.2 Résultats expérimentaux connexes

Cette approche a été appliquée à la classification de 4 types de graines de céréales par Patil NK et Yadahalli [5] qui ont utilisé une approche de classification basée sur la notion de distance minimale telle que la méthode des k plus proches voisins (KNN). Les performances de classification rapportées dans cette recherche sont approximativement de 84% en se basant uniquement sur les caractéristiques de couleur et de la texture, cela sans la nécessité de quelconques prétraitements ou d'une segmentation des images.

Lurstwut et Pornpanomchai [20] ont présenté une étude sur la germination de six variations de grain de riz thaïlandais (Cp111, RD41, Chiang Phatthalung, Sang Yod Phattalung, Phitsanulok 2 et Chai Nat 1). Contrairement à la précédente expérience citée, un processus de prétraitement d'image fut utilisé afin d'extraire quatre caractéristiques principales qui sont la couleur, la taille, la forme et la texture. Le réseau de neurones artificiel utilisé pour la classification a obtenu une précision équivalente à 93,06% en utilisant ces caractéristiques comme données d'entrée (Input).

Uros Skrubej, Crtomir Rozman et Denis Stajnko [1] ont classifié l'état de la germination des graines de tomate (*Solanum lycopersicum*) avec un niveau de performance de 95,44% et ce en déployant un réseau de neurones (ANN) utilisant 11 paramètres de mesures (le périmètre, la kurtose, la valeur maximum de niveau de gris, l'asymétrie et l'écart type des valeurs de gris, la longueur de l'axe principal de l'ellipse englobante, la taille de l'objet, la valeur maximale de niveau de gris de l'histogramme, la médiane, la longueur de l'axe secondaire de l'ellipse englobante et la moyenne de niveau de gris).

On mentionne une dernière étude très particulière menée par Tzu-Ching Wu, Samuel A. Belteton, Jessica Pack, Daniel B. Szymanski et David M. Umulis [15], qui ont développé un algorithme basé sur l'enveloppe convexe nommée « LobeFinder » qui a la capacité de mesurer la distance entre le périmètre de la cellule et son enveloppe convexe dans le but d'identifier l'initialisation de nouveaux lobes de cellule pour générer un modèle quantitatif qui sera appliqué aux mêmes cellules après un laps de temps afin d'analyser les phénotypes mutants, apercevoir des ruptures de symétrie et quantifier la corrélation entre le changement de forme cellulaire et les facteurs intracellulaires qui peuvent jouer un rôle dans le processus

de morphogenèse. Cet algorithme a été validé sur des images des cellules de chaussée (la couche protectrice des plantes qui réduit leur perte d'eau et maintient leur température) de différentes tailles et de formes. La performance du logiciel est mesurée en comparant les points du lobe prédits contre ceux générés manuellement par des chercheurs expérimentés dans l'analyse de la forme des cellules de chaussée de deux laboratoires selon une tolérance prédéfinie. La sensibilité optimale enregistrée est équivalente à 95% qui a démontré une baisse du taux d'erreur de détection de plus de 5 fois par rapport à l'ancienne méthode de détection des lobes nommée « Skeletonize method » [27].

2.3 Reconnaissance de formes basées sur les neurones

Inspirée du système neuronal chez les êtres vivants, dont la façon de manipuler les informations. Cette méthode [32] utilise des programmes prédéfinis dits « Boite noire », avec comme rôle de pouvoir créer des modèles en fournissant suffisamment de données en entrée du réseau de neurones pour que chaque connexion entre les nœuds du réseau puisse être adéquatement déterminée lors du processus d'apprentissage.

Cette approche décrit plusieurs méthodes d'apprentissage automatiques, qu'elle soit supervisée, non supervisée ou semi supervisée. Dans le cadre de notre étude, nous ferons uniquement une description plus détaillée d'une de ces approches ainsi que le réseau de neurones utilisé.

2.3.1 Apprentissage automatique

2.3.1.1 Histoire

Alan Turing fut un des pionniers des ordinateurs d'aujourd'hui, et développa la théorie du système intelligent avec le « Test de Turing » dans son article « L'ordinateur et l'intelligence » en 1950.

Warren McCulloch et Walter Pitts [30] ont présenté pour la première fois le fonctionnement de neurones à l'aide de circuits électriques, qui par la suite donnèrent l'illustration fondamentale des réseaux de neurones.

Le « *Machine Learning* » (une expression citée par Arthur Samuel) a connu une avancée majeure dans le développement des systèmes intelligents, tels que « Deep Blue » un ordinateur développé par IBM qui a battu le champion du monde au jeu d'échecs Garry Kasparov en 1997 ou le réseau de neurones qui est capable de reconnaître le visage des humaines et celui des chats dans des vidéos YouTube par Google en 2012.

2.3.1.2 Apprentissage supervisé

Apprentissage supervisé est un type d'apprentissage machine, qui consiste à fournir des données d'entraînement étiquetées, c'est-à-dire leurs classes d'appartenances.

Cette approche de classification se compose alors de deux phases : d'abord une phase d'apprentissage qui consiste à sélectionner une grande partie des données libellées (généralement 80% des données étiquetées totales) pour créer un modèle qui sera ensuite testé sur les données étiquetées restantes (soit 20%) [26]. La seconde phase dite « Test » dans laquelle le modèle précédemment entraîné sera mis à l'épreuve par la classification de nouvelle instance dans le but d'évaluer sa performance.

2.3.1.3 Apprentissage profond (*Deep Learning*)

Inspiré par le cortex visuel et sonore du cerveau humain, cet apprentissage [3] apprend grâce à une large quantité de données, qui effectue un entraînement à plusieurs reprises et il s'ajuste à chaque itération d'apprentissage pour améliorer le résultat de classification (de la même manière qu'un être humain apprend de l'expérience). Il se compose de plusieurs architectures (couches) de différents traitements non linéaires pour

l'extraction et la transformation des caractéristiques issues des données qui sont articulées ressemblant aux réseaux de neurones biologiques (Voir fig. 2.5).

i. CNN (Réseau de neurones convolutif)

Le réseau de neurones convolutif est un réseau de neurones artificiel de type acyclique (dont les nœuds ne forment pas un cycle, de telle sorte qu'une sortie d'une couche est l'entrée de la suivante) qui imite le comportement du cortex visuel. Hubel et Wiesel [31] ont découvert que chaque cellule neuronale de cette région du cerveau répond seulement à la présence d'orientation de forme (vertical, horizontal, diagonal, cercle...) et que tous ces neurones sont organisés dans un ensemble de colonnes de sorte qu'ils puissent donner une perception visuelle.

ii. Structure du CNN

i. Couches convolutives (*Convolution Layers*)

Contrairement à l'être humain, l'ordinateur voit une image de couleur RGB sous forme d'une matrice à N lignes * M colonnes multipliées par 3 qui représente les valeurs RGB (Bande de couleur rouge, vert et bleu) (voir fig. 2.1). Chaque cellule de cette matrice est un nombre entre 0 et 255 qui représente l'intensité de chaque composante de couleur associée à chaque pixel.

La couche convolutive consiste à extraire les caractéristiques de l'objet dans l'image, on peut avoir plusieurs couches convolutives où chacune a une fonctionnalité de capture de haut ou de bas niveau tel que les contours, couleur, l'orientation du dégradé, etc. Cet ensemble de couches donne un réseau global sur la compréhension approfondie de l'image.

Afin de bien comprendre comment ces couches fonctionnent, commençons par vous présenter un exemple (voir fig. 2.2) : prenons une image de dimensions 5*5*1 entrant dans

une couche convolutive qui utilise une matrice de dimensions 3*3*1 (cette dimension est généralement la plus utilisée, mais cela dépend de la taille et du nombre d'objets contenus dans l'image) qui effectue un processus de convolution sur chaque pixel de l'image en entrée. Cette dernière matrice est aussi appelée filtre ou *Kernel* et contient des nombres qui représentent à chaque position du filtre la pondération appliquée à la position correspondante dans le voisinage d'un pixel de l'image traitée.

Ce filtre est donc passé au travers de la matrice image en commençant par l'extrémité du haut gauche, le processus de convolution effectue une somme pondérée, en multipliant les pondérations du filtre avec les valeurs des pixels de l'image, tout en additionnant ces valeurs afin d'en tirer une seule valeur qui sera alors disponible pour la prochaine couche convolutive. Ce processus de convolution est répété sur tous les pixels fournis en entrée de la couche convolutive.

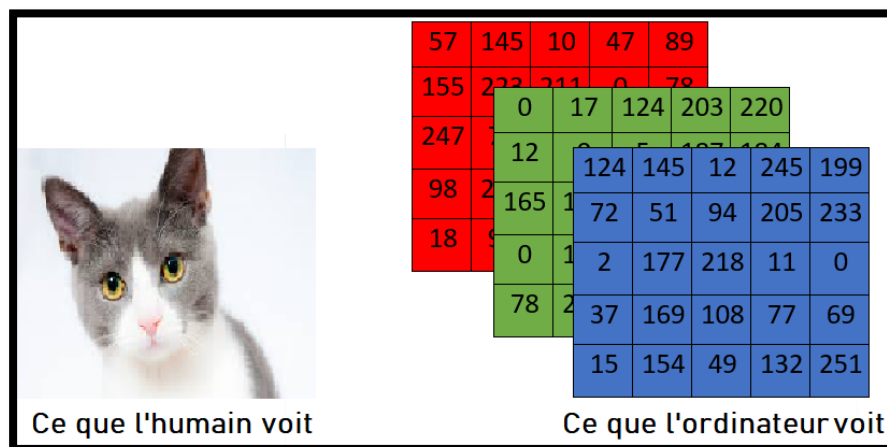


Figure 2.1 : Comment l'image est-elle interprétée par un ordinateur

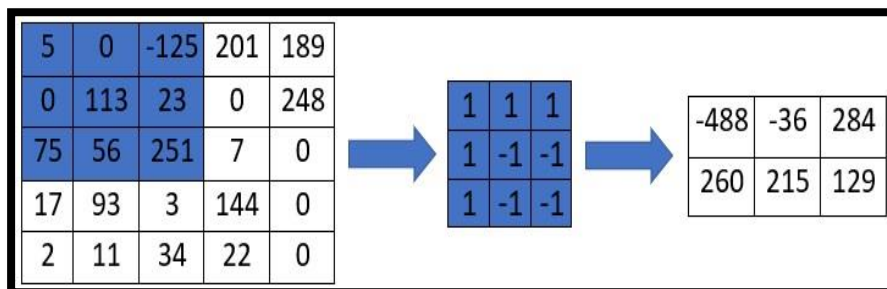


Figure 2.2 : La nouvelle matrice après l'application du filtre 3*3*1

ii. Couche de mise en Commun (*Pooling layers*)

Cette couche est responsable de la réduction spatiale des dimensions de la nouvelle couche convolutive, dans le seul but de réduire la puissance requise pour le traitement des données en récupérant seulement les caractéristiques dominantes (formes, contours, rotation, position invariante, etc.).

Cette couche est similaire à la couche convolutive, qui utilise un filtre dimensions $N \times M$, mais au lieu de faire l'opération de convolution en utilisant les pondérations de cette matrice, elle utilise un des deux calculs suivants : la mise en commun maximum (Max Pooling) ou la mise en commun moyenne (Average Pooling). Le premier type consistant à tenir la valeur maximale de la portion couverte par les dimensions du filtre, tandis que la deuxième retourne la moyenne de toutes les valeurs couvertes par ce dernier (voir fig. 2.3).

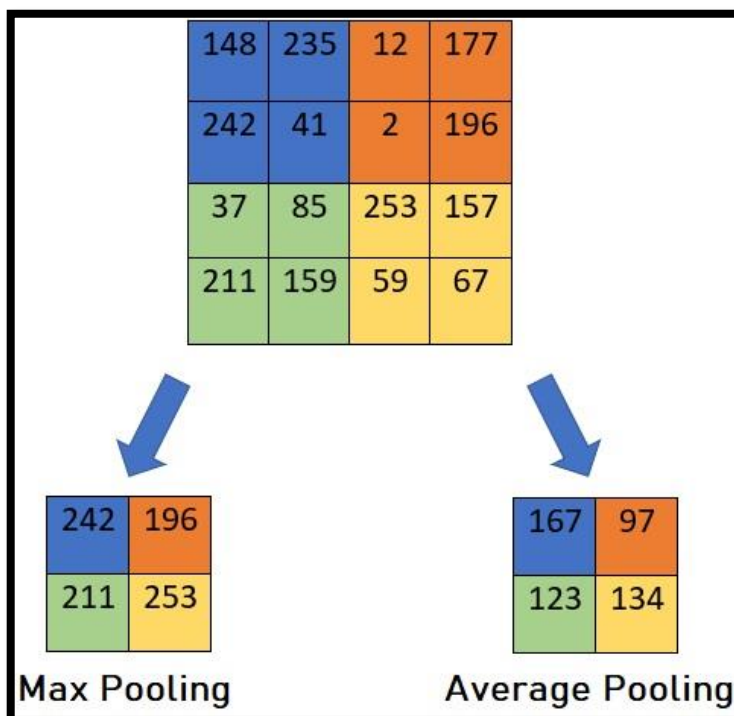


Figure 2.3 : Les résultats des deux types du polling (2*2*1)

iii. Couche d'entrée pour le réseau de neurones artificiel (*Flattening*)

Une fois les images des caractéristiques extraites par les deux couches convolutionnelles précédentes, la prochaine étape a pour rôle de convertir les sorties des couches convolutionnelles en une structure appropriée pour le réseau de neurones en les aplatissant pour former un vecteur (perceptron à multi-niveaux) de la couche d'entrée de ce réseau de neurones artificiels (voir fig. 2.4).

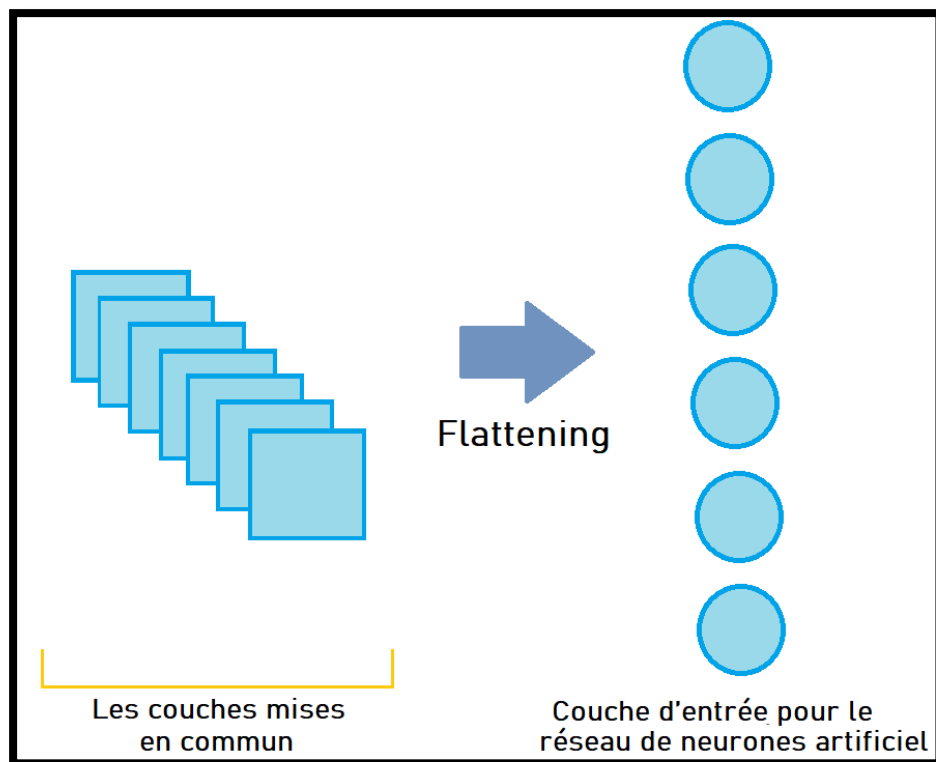


Figure 2.4 : La couche d'entrée pour le réseau de neurones artificiel

iv. Couche entièrement connectée (*Fully connected layer*)

Cette couche est un réseau de neurones artificiels, qui reprend les données (caractéristiques) du vecteur obtenu dans la couche précédente (*Flattening*) comme couche d'entrée. Ces données sont combinées dans un ensemble de variété d'attributs qui rend le réseau conventionnel plus robuste pour classifier les images reçues.

Les nœuds de la couche cachée du réseau (*Hidden layer*) sont en réalité des détecteurs de caractéristiques (Fonction d'activation) qui sont reliés par des connexions auxquelles sont associés des poids (les liens reliant les nœuds dans la figure 2.5). Lors du processus d'apprentissage, le réseau doit faire une rétropropagation à chaque itération pour permettre au modèle de bien apprendre et distinguer les caractéristiques les plus dominantes de chaque classe, ce qui favorise l'amélioration de la précision et minimiser l'erreur.

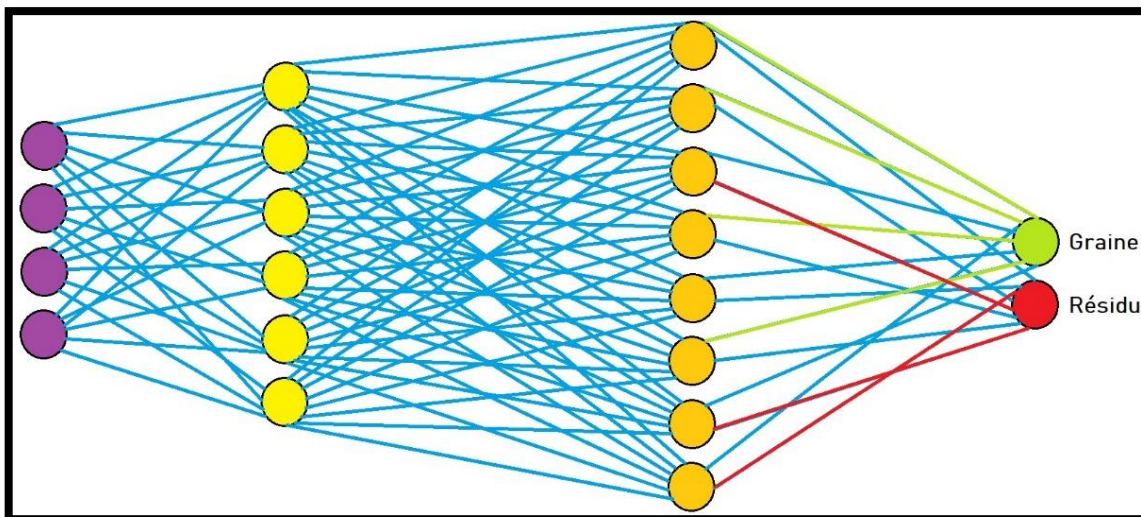


Figure 2.5 : Un réseau de neurones artificiel pour la classification

(les lignes vertes et rouge sont les poids les plus dominants pour la classe graine et résidu végétal respectivement)

2.3.2 Résultats expérimentaux connexes

Shubhra Aich et Ian Stavness [14] ont proposé des architectures du type *Segnet* pour le comptage des feuilles de rosette, en utilisant d'abord un réseau de déconvolution pour générer un modèle de segmentation permettant l'extraction des régions d'intérêt dans les images pouvant correspondre à des feuilles en utilisant quatre types d'images différentes : des images couvrant la plante en entier dans une chambre de croissance, des images couvrant la plante en entier dans une chambre de croissance avec un champ de vision plus large, des images de tabac qui englobe toute la plante et enfin des images publiques d'*Arabidopsis* aux champs de vision variés. Ensuite un réseau convolutionnel est utilisé pour le comptage. Bien que ces deux architectures soient entraînées individuellement, elles sont dépendantes l'une de l'autre de sorte que le masque généré par le modèle de segmentation est utilisé dans l'entraînement du modèle de comptage. La précision des quatre types d'images d'entraînement est équivalente à : 98 %, 94%, 80% et 96% respectivement. Des résultats forts intéressants qui ont dépassé certaines mesures de performances du vainqueur de la compétition « Counting challenge » de 2015 [27].

Jaromir Przybylo et Miroslaw Jablonski [23] ont utilisé un réseau de neurones convolutif (CNN) afin d'évaluer la viabilité des graines des chênes pédonculés (*Quercus robur* L) de façon automatique et sans quelconques prétraitements, et cela en se basant uniquement sur les caractéristiques (couleur, entropie, les bords et l'intensité de l'image des sections des graines) qui avaient apporté une meilleure performance par rapport à l'évaluation manuelle équivalente de 85%. Les deux chercheurs rajoutent aussi que la représentation et la clarté des images sont un facteur clé en apprentissage du réseau de neurones.

2.4 Traitement d'image

Le traitement des images est une phase qui examine les images numériques à l'aide d'un ensemble d'opérateurs afin d'améliorer la qualité de la perception visuelle des images ou d'en extraire des informations.

2.4.1 Réduction du bruit

Le bruit dans une image numérique est un parasite ou un effet indésirable causé principalement d'une haute valeur d'ISO (sigle désigné de l'organisation internationale de normalisation), un excès d'exposition ou une prise d'image dans un milieu sombre avec une vitesse d'obturation trop faible. On peut différencier deux types de bruit :

- bruit de chrominance : Est un ensemble de pixels colorés aléatoirement (voir fig. 2.6).
- bruit de luminance : Est un composant des pixels illuminés, surnommée effet « poivre et sel » qui définit la forme noire et blanche aléatoire des pixels (voir fig. 2.7).

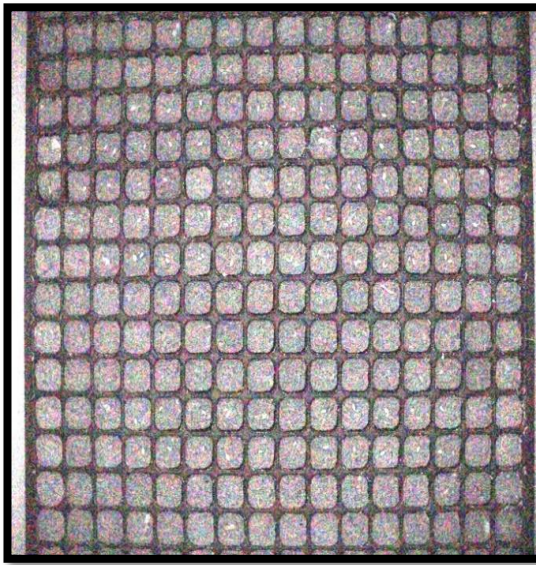


Figure 2.6 : Un bruit de chrominance sur un plateau d'ensemencement

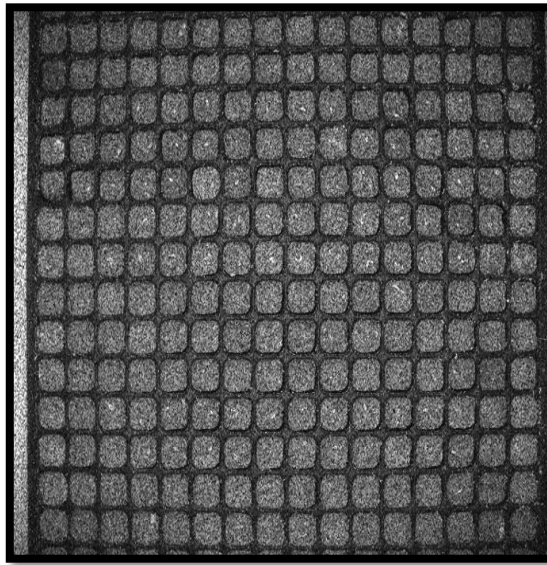


Figure 2.7 : Un bruit de luminance sur un plateau d'ensemencement

Pour y remédier, plusieurs filtres existent qui suppriment ou atténuent les effets du bruit sur ces pixels. Le filtre souvent utilisé dans des recherches répertoriées dans la littérature et aussi dans notre projet est le lissage gaussien, qui est un type de filtre de lissage qui utilise la fonction mathématique gaussienne suivante :

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

qui produit une moyenne pondérée de l'image en fonction de la distance du voisinage de chaque pixel aux positions spatiales x et y . La largeur de ce filtre (son importance) est déterminé par son écart-type σ , c'est-à-dire que si σ est plus petit qu'un pixel, le lissage n'aura pas un réel impact, cependant plus qu'on l'agrandit, plus on réduit le bruit, mais plus l'image filtrée devient floue.

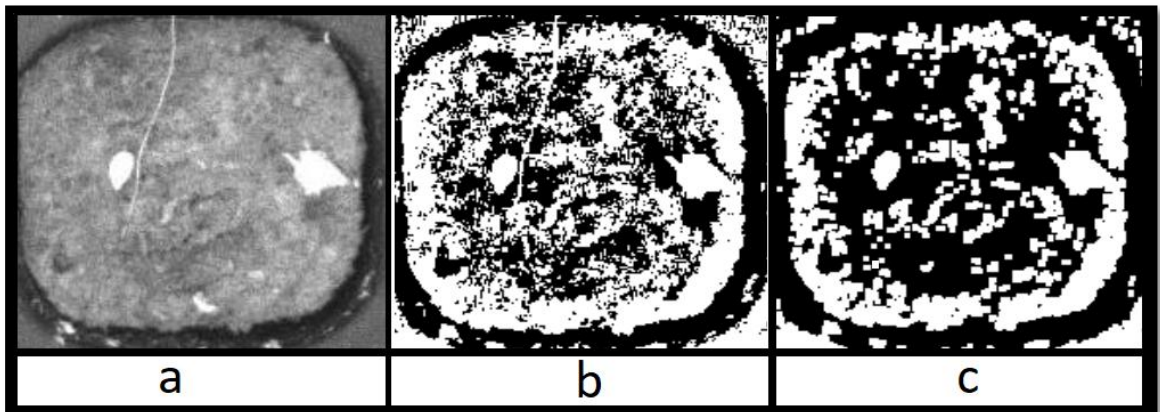


Figure 2.8 : Effet du filtre gaussien dans le processus de segmentation

- (a) Image originale d'une cellule. (b) Image binaire adaptative de l'image originale. (c) Effet du filtre lissage gaussien sur l'image binaire (b)

2.4.2 Seuillage binaire adaptatif

Le seuillage binaire est une méthode qui consiste à produire deux classes, noir et blanc à partir d'une image de niveaux du gris, afin d'isoler l'arrière-plan (*background*) des objets dont l'objectif est de les détecter.

Le seuillage adaptatif découpe l'image en bloc de $n*n$, puis calcule le seuil de chaque pixel avec une moyenne pondérée des seuils locaux des blocs voisins en fonction de leur distance, ce qui génère plusieurs seuillages pour chacune de ces différentes régions sur la même image. Très utile si une image a des conditions d'éclairage non uniforme dans différentes zones.

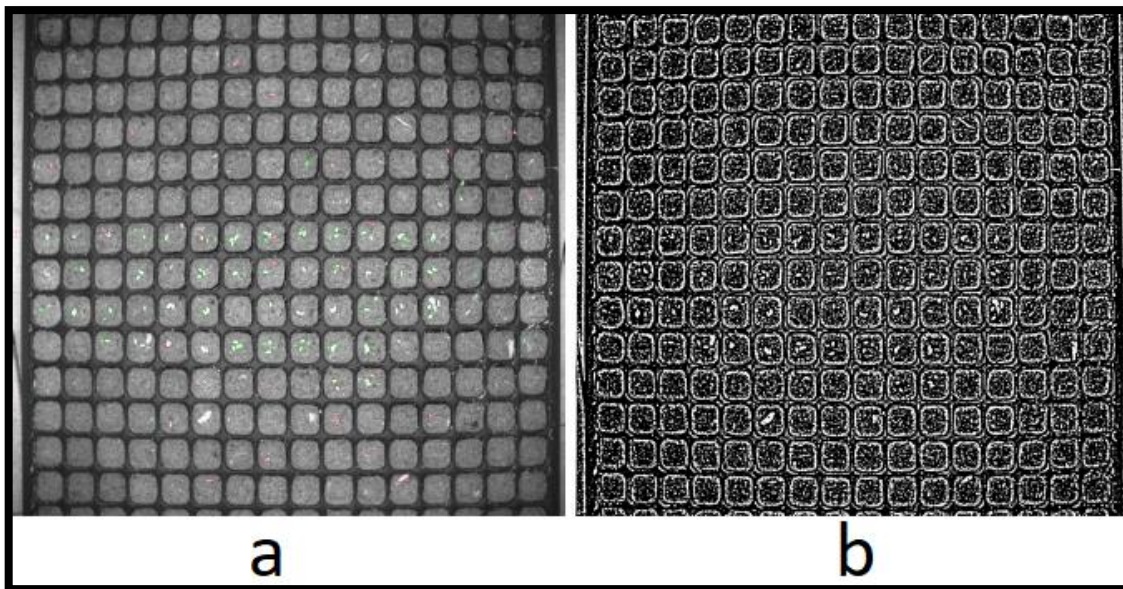


Figure 2.9 : Seuillage binaire adaptatif du plateau d'ensemencement

(a) Image originale du plateau d'ensemencement. (b) Image binaire adaptative du plateau d'ensemencement

2.4.3 Morphologie mathématique

La morphologie mathématique est un ensemble d'opérateurs mathématiques basés sur la théorie des ensembles, ils agissent comme une sorte de comparaison locale des structures dans une image avec un élément structurant (ou référence) annoté B qui possède des caractéristiques géométriques telles que la forme et sa taille (souvent symétrique, convexe et concave), ces opérateurs sont très utilisés dans le prétraitement des images [16] permettant

d'éradiquer les surfaces excédentaires pour lisser les contours. Les opérateurs les plus connus et les plus utilisés sont :

- la dilatation : son effet est d'agrandir progressivement les formes d'objets d'intérêt et ainsi diminuer le bruit de contour de ces formes par le remplissage des trous sur le pourtour de ces régions. Représenté par le signe \oplus , sa définition mathématique est : soit une forme A dans une image, le résultat de la dilatation $D^B(x)$ est composée de l'ensemble des décalages 2D x de l'élément structurant B_x donnant une intersection non vide avec la forme A :

$$D^B(x) = \{x \mid B_x \cap A \neq \emptyset\}$$

- l'érosion : cet opérateur comme son nom l'indique est d'éroder les limites de la zone des pixels associées aux formes d'objets d'intérêt dans une image et ainsi éliminer les bosses sur les pourtours des formes et aussi élargir les trous sur le pourtour de ces formes. Représenté par le signe \ominus , sa définition mathématique est : soit une forme A dans une image, le résultat de l'érosion $E^B(x)$ est composée de l'ensemble des décalages 2D x de l'élément structurant B pour lesquels B_x est complètement inclus dans la forme A :

$$E^B(x) = \{x \mid B_x \cap A = B_x\}$$

- l'ouverture : cet opérateur est dérivé des opérateurs fondamentaux d'érosion et dilatation. Son effet est identique à celui de l'érosion qui supprime les bosses sur les bords des régions, mais moins destructif, on peut aussi la définir comme une érosion suivie de dilatation :

$$A \circ B = (A \ominus B) \oplus B$$

- la fermeture : similaire à l'ouverture, mais appliquée dans l'ordre inverse. En revanche, cet opérateur tend à emplir les trous sur les pourtours des formes dans une image, mais moins destructif que la dilatation seule puisque suivie d'une érosion. On peut aussi la définir comme étant une dilatation suivie de l'érosion :

$$A \bullet B = (A \oplus B) \ominus B$$

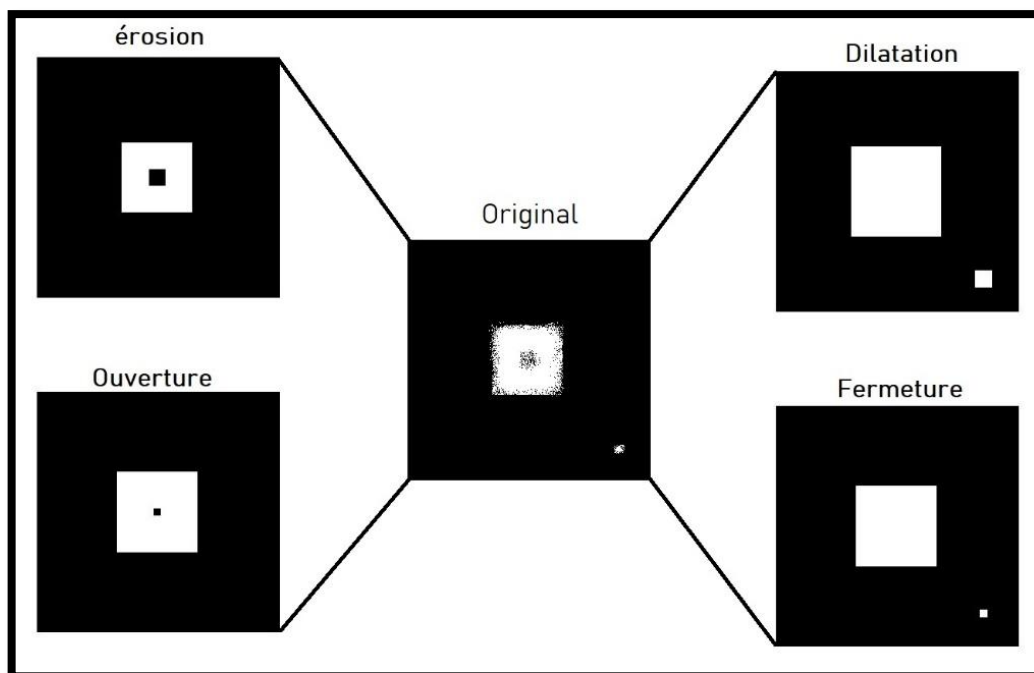


Figure 2.10 : Résultat des quatre d'opérateurs appliqués sur une même image

2.4.4 Détection de contours

Ce qui nous intéresse le plus dans une image, ce sont les objets présents. Après le traitement d'images effectué, on peut vouloir par exemple compter leur nombre ou même extraire leurs formes pour pouvoir étudier leurs caractéristiques géométriques pour la reconnaissance subséquente de ces formes. La méthode utilisée dans notre projet est extraite de la bibliothèque graphique libre « OpenCv » (que nous présentons en détail à la section « 2.5 bibliothèque graphique ») qui utilise les algorithmes introduits par Suzuki et Abe [22] qui détectent les contours dans des images binaires en prélevant les courbes joignant les points continus ayant la même couleur ou l'intensité. Il existe principalement deux types de méthode d'approximation des contours (voir fig. 2.11) : la première (CHAIN_APPROX_NONE) stocke tous les points des contours détectés et la deuxième (CHAIN_APPROX_SIMPLE) stockant uniquement les paires de points de l'extrémité des droites (le point du début et celui de la fin) d'approximation des segments de contours

rectilignes et c'est cette méthode qui est préférable pour la détection de contour, car celle-ci économise beaucoup l'espace mémoire.

Une fois les points limites non redondants de l'objet obtenus par la seconde méthode, il existe une méthode qui dessine le polygone en reliant ses points sous le nom de « *drawcontours* » disponible dans la même bibliothèque graphique utilisée et permettant ainsi la visualisation de chaque contour des formes.

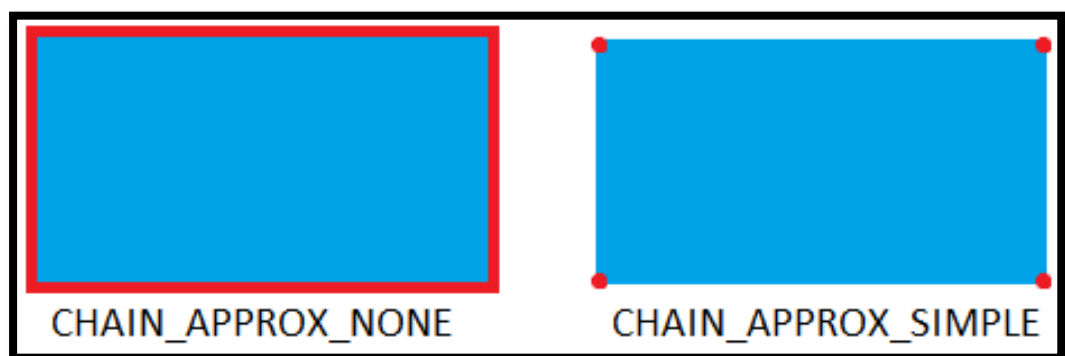


Figure 2.11 : Différence entre les deux méthodes d'approximation des contours.

2.4.4.1 Analyse de forme géométrique

Une fois les contours des objets obtenus, nous pouvons alors procéder à l'extraction des caractéristiques de chacune des formes de ces objets afin de nous en servir pour créer des filtres géométriques qui isolent les objets non désirés comme les résidus végétaux qui doivent être pris en considération dans le cadre du présent projet. Dans la procédure d'extraction, l'objet à détecter doit satisfaire les trois facteurs suivants : la taille, la moyenne d'illumination et la convexité.

i. La taille

La taille ou la zone du contour des graines ont généralement la même taille, quoi qu'on puisse avoir trois intervalles de tailles différents selon la façon avec laquelle elles sont déposées par le semoir pneumatique :

- sur le côté (voir fig. 2.12 (a)).
- sur le bout (voir fig. 2.12 (b)).
- sur la face (voir fig. 2.12 (c)).

Ayant suffisamment des données sur les tailles, on peut intégrer un filtre qui écarte tous objets d'une taille qui est inférieure ou supérieure aux intervalles de tailles valides de graines de semence et le répertorier comme étant un résidu végétal.

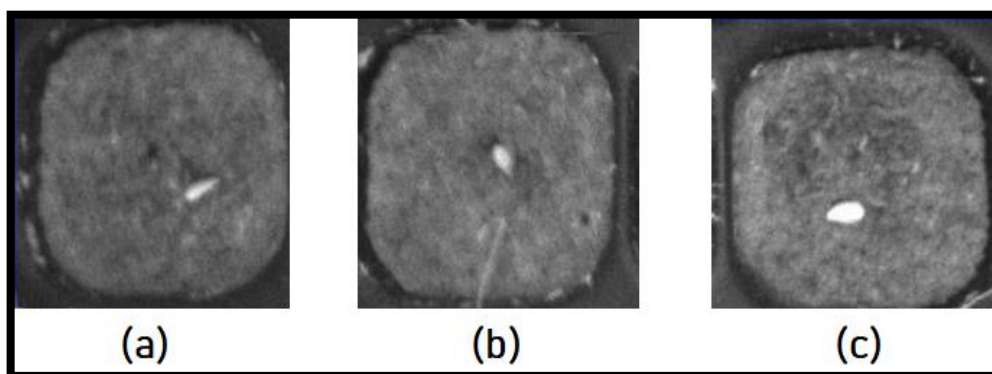


Figure 2.12 : Les trois façons possibles de positionnement d'une graine

ii. La moyenne d'illumination

Cette caractéristique définit la luminance de l'objet. En visualisant les graines dans les cellules d'ensemencement, ces dernières étant illuminées par un rayonnement infrarouge réfléchissent fortement le rayonnement infrarouge incident. Les graines ayant une forte réflectance dans le proche infrarouge leur confèrent une forte brillance dans les images

permettant de facilement les distinguer du milieu d'ensemencement (sphaigne) dans lequel elles sont semées. Cependant, une singularité est à prendre en compte, car une cellule d'ensemencement peut parfois aussi comporter des résidus de bois ayant une moyenne de luminance similaire à celle des graines rendant le processus de classification plus difficile. Le calcul de la moyenne est très simple qui est déduit par le calcul de la somme des luminances des pixels d'un objet dont la luminance est entre 200 et 255 et diviser ensuite le nombre total des pixels dans l'intervalle de 200 et 255.

$$Ml = \frac{1}{n} \sum_{k=1}^n (Px_k) \quad (\text{Condition } Px_k \in [200,255])$$

Ou Px est la luminance d'un pixel, k est sa position dans la matrice de pixels de l'objet et n est le nombre total des pixels dans cette matrice.

iii. La convexité

Comme mentionné précédemment, la présence de résidus végétaux est possible dans les cellules d'ensemencement, ayant une taille et une luminosité similaires à celles d'une graine, mais ne possède pas la même forme (voir fig. 2.13), un facteur primordial pour les discerner des graines.

iv. Forme Convexe

La forme convexe (dite aussi enveloppe convexe) est un polygone dont ses angles intérieurs sont inférieurs à 180 degrés. Pour un ensemble de points de contour d'une forme, l'enveloppe convexe est la région imitée par un élastique qu'on contracte jusqu'à ce qu'il entoure les points de contour saillants de l'extrémité de l'ensemble de points de contours.

On peut aussi la définir comme suit : l'enveloppe convexe de A est l'ensemble des combinaisons convexes d'éléments de A tel que :

$$Y = t_1 a_1 + \dots + t_p a_p$$

Où a est l'élément appartenant à A et t est un réel positif, P est le nombre total d'éléments existant dans A , alors Y est la somme de leur multiplication et qui représente l'élément de l'enveloppe convexe de A .

v. Forme Concave

La forme concave (non convexe) est un polygone ayant au moins un angle intérieur supérieur à 180 degrés.

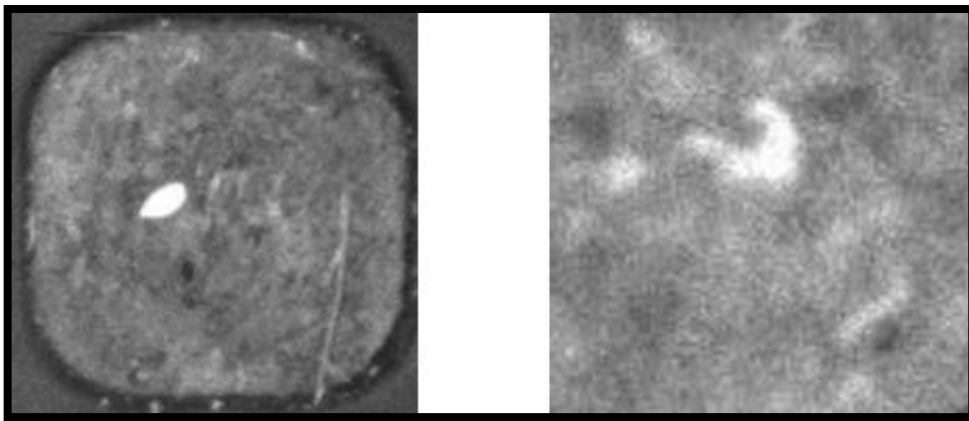


Figure 2.13 : La différence de forme entre une graine et un résidu végétal

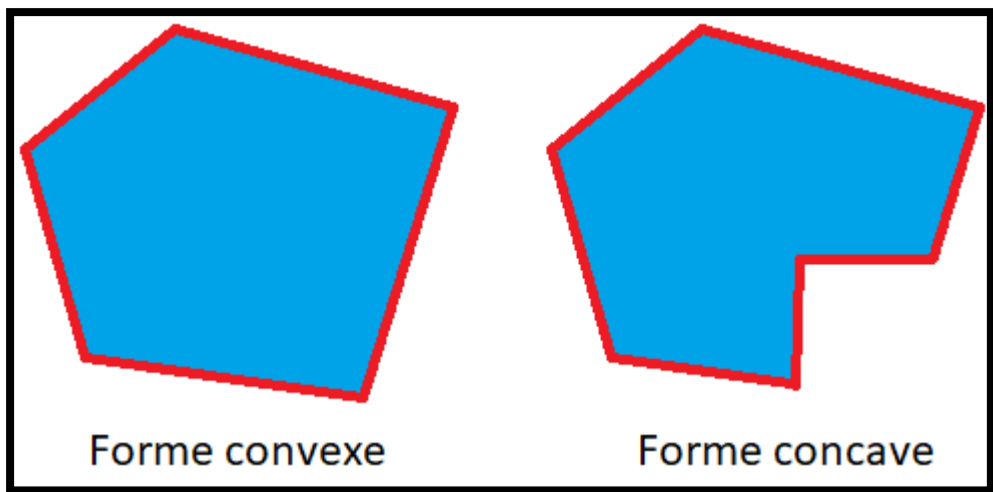


Figure 2.14 : Le visuel des deux formes géométriques basées sur la convexité

2.5 Bibliothèque graphique

La bibliothèque graphique utilisée dans ce projet de recherche est l'*OpenCV* (*Open Source Computer Library*) [17] qui est une bibliothèque libre-service développée par Intel, conçue pour le traitement d'images et vidéo en temps réel qui possèdent plus de 2000 algorithmes dans le domaine, en citant quelques-uns comme la lecture et l'écriture des images ou des vidéos, calcul d'histogrammes, lissage et filtre, seuillage binaire, segmentation, morphologie mathématique, détection de mouvement, etc. Compatible dans plusieurs systèmes d'exploitation et principalement écrit en C++, mais possède plusieurs extensions pour différents langages de programmation comme *EmguCV* (pour C#), *Python CV* et *Ruby CV*. Toutes ces capacités la rendent très utilisée dans plusieurs grandes compagnies telles que Google, Yahoo, Intel, Toyota et même la NASA.

2.6 Langage de programmation

Deux langages ont été utilisés dans ce projet, le premier est le C#, utilisé pour la première méthode d'expérimentation (Reconnaissance de formes statistiques) pour sa convivialité et son aspect visuel « interface utilisateur » pour illustrer les caractéristiques des objets détectés. Le second est le Python qui fut utilisé pour la deuxième et dernière méthode expérimentale (Reconnaissance de formes basée sur les réseaux de neurones) pour sa portabilité et son immense variété de bibliothèques de traitement de données scientifiques disponibles. Toutes les définitions qui suivent sont tirées de Wikipédia.

2.6.1 C#

Prononcée *C sharp*, est un langage de programmation orientée objet créé par Microsoft en 2001, très similaire à JAVA et dérivé du C et C++. Ce langage est destiné au développement sur la plateforme .NET des applications web, applications de bureau, services web, etc. Son point fort est la mise à disposition d'un système de construction d'interfaces visuelles de façon « saisir et déposer » plusieurs outils d'interaction (Bouton, onglet, boîte image et des zones de texte).

2.6.2 Python

À la différence du C#, ce langage de programmation conçue par Guido van Rossum est un interpréteur qui favorise la programmation impérative structurée, fonctionnelle et orientée objet (multiparadigmes). Son point fort est sa portabilité (contrairement au C#), c'est-à-dire qu'un programme python peut bien fonctionner sous divers systèmes d'exploitation tels que Linux, Windows, Mac os, IOS et Android. Une première version apparue le 20 février 1990 est à ce jour très utilisé dans les grandes entreprises

d'informatique et la communauté scientifique dans l'utilisation de l'intelligence artificielle et les études de régression statistiques.

2.6.2.1 Keras

Keras est une bibliothèque libre-service (Open source) qui permet d'interagir avec les algorithmes de réseaux de neurones profonds (deep learning) et l'apprentissage machine (Machine learning), initialement conçus par François Chollet dans le cadre du développement du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System).

2.6.2.2 Tensorflow

Tensorflow est un outil d'apprentissage automatique (Machine learning) libre-service le plus utilisé en intelligence artificielle autant dans le milieu commercial que dans la recherche. Créé par Google en 2011 (anciennement appelé DisBelief) et doté d'une interface pour Python.

2.6.2.3 Numpy

Numpy est la base de regroupement de bibliothèques python autour du calcul scientifique. Plus précisément, une bibliothèque logicielle libre pour manipuler les matrices et tableaux multidimensionnels à l'aide de fonctions mathématiques appropriées.

2.6.2.4 Matplotlib

Matplotlib est une bibliothèque libre-service du langage Python conçu pour interpréter les données sous forme de graphiques qu'on peut l'exporter sous divers formats matriciels (PNG, JPEG) et vectoriels (PDF).

2.7 Conclusion

Ce chapitre introduit les bases fondamentales de la composition d'un système de vision passant par le traitement d'image (morphologie mathématique, facteurs de forme) jusqu'aux classifications (réseaux de neurones), implémenter par les différents outils de programmation décrits dans l'intérêt d'initier le chapitre suivant titré méthodologie ou nous allons appliquer toutes ses notions sous forme d'algorithmes pour concrétiser notre programme de détection et classification de graines.

CHAPITRE 3

MATÉRIEL ET MÉTHODOLOGIE

3.1 Introduction

Nous allons décrire en détail notre projet de recherche en commençant d'abord par présenter les types de graines de semence de résineux [4] qui font l'objet de l'automatisation de leur classification et ultimement leur comptage. Le matériel utilisé pour implémenter le système de détection/classification des graines de résineux est aussi présenté. Ensuite, nous passons en revue étape par étape le processus d'application des deux méthodes expérimentales introduites précédemment dans le chapitre sur le survol de la littérature pour permettre la détection/classification des graines de résineux et leur comptage. Le code source essentiel du système de détection/classification est aussi exposé dans les deux langages de programmation utilisés.

3.2 Matériaux expérimentaux

3.2.1 Épinette noire

L'épinette noire possède des caractéristiques particulières, commençant par son bois léger, pale et assez fort qui résiste aux vents, sa taille peut atteindre 30 mètres de hauteur et 60 centimètres de diamètre et peut vivre 200 ans, ce qui la rend très utilisée dans le bois d'œuvre québécois et canadien. Les graines d'épinettes noires sont de couleur foncée (noire) avec une longueur moyenne dans l'intervalle 1-3 millimètre et une forme allongée.



Figure 3.1 : Visuel de graines d'épinette noire

3.2.2 Épinette rouge

Cette épinette préfère un climat froid et humide et vit plus longtemps jusqu'à 400 ans où le record de la plus vieille épinette rouge découvert est âgé de 445 ans au Nouvelle-Brunswick. Son bois est de couleur brun pâle et relativement léger qui peut atteindre une hauteur de 25 mètres et 60 centimètres de diamètre. Très exploitée pour la conception du papier et du bois maritime. Les graines d'épinettes rouges sont de couleur foncée (brune) avec une longueur moyenne dans l'intervalle 3-4 millimètres et une forme allongée.



Figure 3.2 : Visuel de graines d'épinette rouge

3.2.3 Pin rouge

Les pins rouges sont visibles à leur couleur du bois brun rougeâtre et très durs, il peut mesurer 25 mètres de long et 75 centimètres du diamètre et peut vivre plus de 200 ans. On le trouve presque dans toutes les plantations du Québec ou la majorité est transformée en poteaux électriques, car son bois imprègne facilement les produits de préservation. Les graines de pin rouge sont de couleur claire avec une longueur moyenne dans l'intervalle 4-6 millimètres et une forme ovoïde.



Figure 3.3 : Visuel de graines de pin rouge

3.2.4 Pin blanc

Le pin blanc est le plus grand et le plus répandu des arbres dans la province. Cet arbre peut atteindre une hauteur de 40 mètres et 1.5 mètre de diamètre avec une longévité moyenne de plus de 200 ans. Malgré leur sensibilité aux maladies de la rouille vésiculeuse et aux brûlures hivernales, il fut très exploité au fil des décennies dans la construction des

mâts de bateaux. Les graines de pin blanc sont de couleur claire avec une longueur moyenne dans l'intervalle 5-7 millimètres et une forme elliptique.



Figure 3.4 : Visuel de graines de pin blanc

3.2.5 Processus d'ensemencement

Le processus d'ensemencement des graines de semence par le semoir pneumatique consiste d'abord à faire déplacer des cabarets de cellules d'ensemencement sur un convoyeur. Les cabarets de cellules passent dans le semoir pneumatique qui injecte idéalement une graine dans chaque cellule d'ensemencement. Cependant, l'ensemencement est parfois imparfait avec quatre possibilités d'anomalies :

1. Une cellule sans graine (voir fig. 3.5 (A)) : Cet état doit absolument être évité, d'où l'utilité du système de détection/classification développé dans le cadre de la présente recherche.
2. Une cellule avec une graine (voir fig. 3.5 (B)) : l'état le plus désirable dans toutes les cellules d'un cabaret.
3. Une cellule avec deux graines (voir fig. 3.5 (C)) : grande probabilité qu'un cabaret possède plusieurs cellules avec cet état.
4. Une cellule avec trois graines (voir fig. 3.5 (D)) : cet état est plus rare.
5. Une cellule avec des graines superposées (voir fig. 3.5 (E et F)). : cet état est souvent rencontré où plusieurs graines se retrouvent collées et engendrent une forme non convexe qui peut tromper le système de classification.

Bien que selon les normes sur l'efficacité de l'ensemencement, les états 2, 3 et 4 soient considérés acceptables, le système de détection/classification doit quand même

pouvoir calculer le nombre de graines dans une cellule afin de corriger ce surplus pour ainsi minimiser les dépenses liées à l'utilisation d'un excédent de graines.

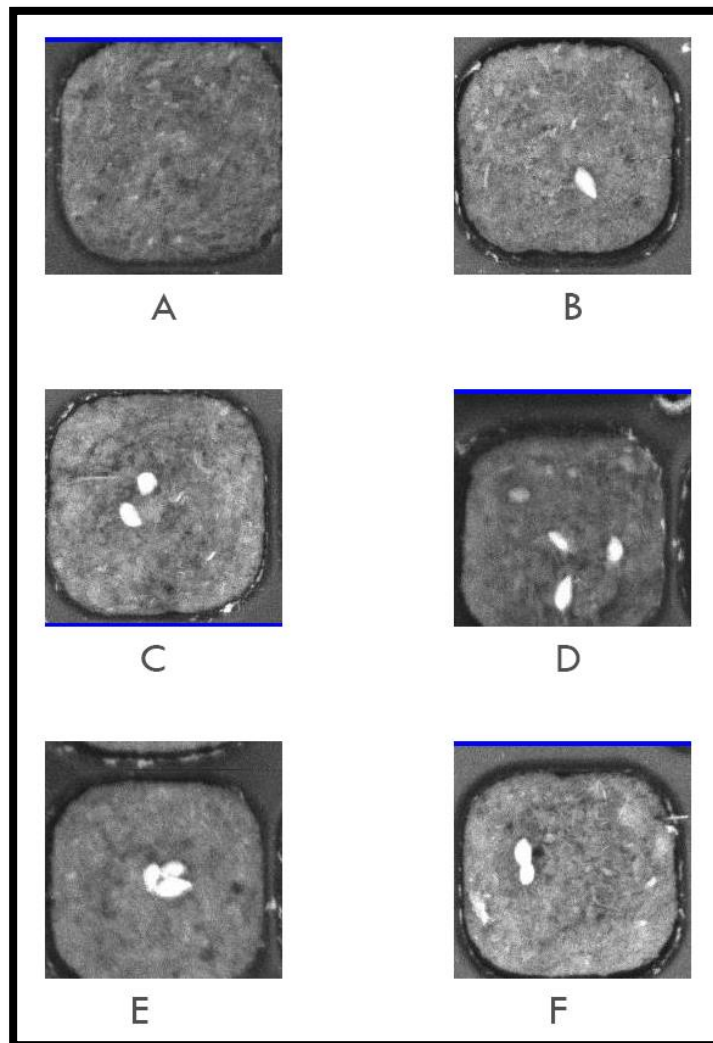


Figure 3.5 : Les différents états d'une cellule

(A) Cellule sans graine B) Cellule avec une graine C) Cellule avec 2 graines D) Cellule avec 3 graines E) Cellule avec 3 graines collées et superposées F) Cellule avec 2 graines collées et superposées)

3.2.6 Caméra spectroscopique dans le proche infrarouge

La spectroscopie dans le proche infrarouge (NIR) est une technique de mesure des spectres de réflexion de longueur d'onde infrarouges de 0,78 micromètre à 2,5 micromètres, cette technique est très utilisée dans le domaine de l'agriculture qui peut détecter les liaisons chimiques C-H, O-H et N-H, ce qui offre la possibilité de calculer le pourcentage d'humidité des semences.

Afin d'appliquer cette technique, une caméra dotée d'un capteur proche infrarouge est nécessaire, cette dernière est située à l'intérieur d'un caisson métallique permettant la capture d'images des cabarets qui se retrouvent complètement dans ce caisson métallique lors de leur déplacement sur un convoyeur. Les caractéristiques de cette caméra sont citées ci-dessous :

- Modèle : Grasshopper3.
- Fabricant : Point Grey Research.
- Capteur : Sony ICX657AL.

3.2.7 Ordinateur

L'ordinateur utilisé dans l'implémentation de notre système de détection/classification de graines de résineux se caractérise comme suit :

- processeur : Intel(R) core(TM) i7-7700 CPU @ 3.60 ghz, 4 cœurs.
- mémoire vive : DDR4 (16 go).
- système d'exploitation : Windows 10, 64 bits.
- carte graphique : Nvidia GTX 1060 (6 go).

3.2.8 Logiciels

3.2.8.1 Visual studio 2017

Ce logiciel développé par Microsoft offre un ensemble d'outils de développement (tel que Visual C# dans notre cas) permettant de concevoir des applications web, bureautique et des applications mobiles exécutées sous le framework .NET.

3.2.8.2 Pycharm

Pycharm est un environnement de développement conçu par l'entreprise tchèque JetBrains qui comporte un éditeur de texte permettant la conception orientée objet, des tests unitaires et d'intégration et ainsi qu'un débogueur graphique. L'édition professionnelle est celle employée dans notre projet qui offre des outils scientifiques variés.

3.3 Méthodologie

3.3.1 Reconnaissance statistique de formes

Nous allons subdiviser la procédure de cette méthode en étapes (voir figure 3.6) afin de bien expliquer chacune d'elle et les accompagner avec son propre code source (rappelons que cette procédure a été implémentée en langage C#).

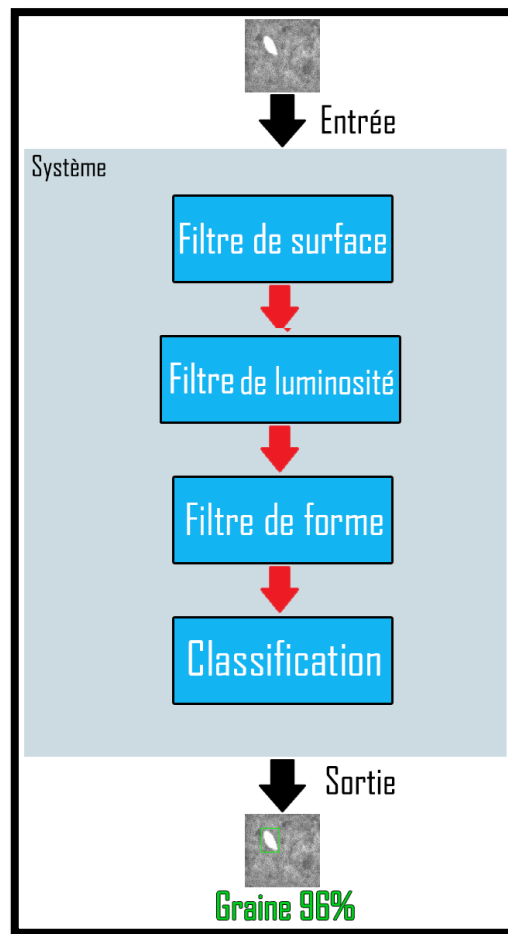


Figure 3.6 : Le processus de détection des graines

3.3.1.1 Segmentation

Une fois que l'image de la cellule est capturée au niveau du gris, nous utilisons ensuite le seuillage binaire adaptatif sur cette image au niveau du gris avec nos paramètres expérimentaux sur les graines qu'on peut apercevoir dans le code source. Finalement, on applique le lissage gaussien suivi de l'érosion et de la dilatation avec un noyau de dimension 2*2 (kernel) sur l'image récemment binarisée et on obtient le résultat exposé à la figure 3.7.

Algorithme 3.1 : Segmentation de l'image d'entrée

```
CvInvoke.AdaptiveThreshold(Mon_Image, Mon_Image_Bin, 255,  
AdaptiveThresholdType.MeanC, ThresholdType.Binary, 3, 1.0) ;  
/*  
Paramètre 1 : L'image a binariser.  
  
Paramètre 2 : le nouveau objet identique à notre objet d'image Mon_Image pour  
sauvegarder le résultat.  
  
Paramètre 3 : le seuil maximal (255 par défaut).  
  
Paramètre 4 : L'algorithme du seuillage adaptatif (on utilise la valeur de la moyenne de la  
zone du voisinage moins la constante C du paramètre 7)  
  
Paramètre 5 : Le type de seuillage (le binaire est choisie).  
  
Paramètre 6 : la taille de bloc du voisinage du pixel sélectionné (kernel).  
  
Paramètre 7 : Cette valeur permet d'affiner la valeur du seuil (la constante C soustraite de  
la moyenne).  
  
*/  
Mon_Image_Bin.SmoothGaussian(2, 2, 1, 1).Erode(2).Dilat(2)
```

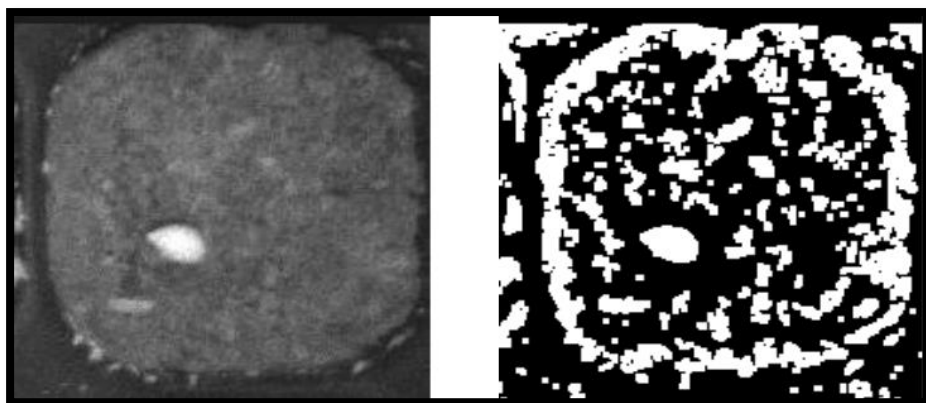


Figure 3.7 : Résultat de la segmentation de l'image d'entrée

3.3.1.2 Filtre de surface

Nous avons déjà précisé qu'une graine peut avoir différentes formes (apparences dans l'image) de la façon dont elle est déposée dans une cellule d'ensemencement par le planteur pneumatique. Avant d'en extraire l'aire, il nous faut d'abord détecter les formes pouvant être des graines. La bibliothèque graphique *Open CV* nous offre une fonction très simple qui détecte les contours des formes de façon automatique en fournissant les paramètres nécessaires. Une fois que tous les contours des objets détectés sont stockés dans un vecteur, nous créons une boucle pour calculer la surface de ces derniers :

Algorithme 3.2 : Fonction d'extraction de la taille du contour d'un objet

```
CvInvoke.FindContours(My_Image_Bin, Contour, null, RetrType.List,
ChainApproxMethod.ChainApproxSimple) ;

/*
Paramètre 1 : L'image binarisé.
Paramètre 2 : Notre vecteur de point pour sauvegarder les contours détectés.
Paramètre 3 : Mode de récupération des contours, le mode choisie RetrType.List récupère
tous les contours sans établir les relations hiérarchiques.
Paramètre 4 : Méthode d'approximation des contours, On a choisi ChainApproxSimple
tel que mentionné dans le chapitre précédent. */for (int i = 0; i < Contour.Size; i++) //
Boucle pour calculer toutes les surfaces des contours
{
double Surface = CvInvoke.ContourArea(Contour[i]) ;
};
```

Nous avons effectué plusieurs tests expérimentaux sur chaque surface des graines qui sont segmentées sur la base d'intervalles pouvant correspondre à la position avec laquelle les graines sont déposées dans une cellule d'ensemencement :

- les graines déposées sur le côté : La surface moyenne se situe dans l'intervalle de pixel [120, 200].
- les graines déposées sur le bout : La surface moyenne se situe dans l'intervalle de pixel [201, 250].
- les graines déposées sur la face : La surface moyenne se situe dans l'intervalle de pixel [251, 350].
- un ensemble deux graines superposées : La surface moyenne se situe dans l'intervalle de pixel [351, 400].
- un ensemble de trois graines superposées : La surface moyenne se situe dans l'intervalle de pixel [401, 650].

C'est ainsi qu'on obtient notre premier filtre de surface qui isole les objets (formes) dont la surface est inférieure au premier intervalle ([120, 200]) et supérieure à celui du dernier ([401, 650]) et qui nous donne approximativement l'ensemble des graines potentielles dans les cellules selon l'appartenance de leur surface dans les intervalles prédéfinis. Subséquemment, il faudra déterminer si les surfaces segmentées correspondent ou non à des amas de deux ou trois graines superposées qu'à priori, le système compte comme étant un seul objet (une graine) et non deux et trois respectivement comme il se devrait. La figure 3.8 montre le résultat de cet effet.



Figure 3.8 : Résultat du filtre de surface

3.3.1.3 Filtre de luminosité

Les objets préservés lors du processus de segmentation des formes assimilables à des graines passent ensuite dans un second filtre qui calcule la luminosité de l'objet, mais avant de procéder au calcul de l'intensité des pixels, notre image est binarisée, c'est-à-dire qu'il existe uniquement deux couleurs, le noir pour l'arrière-plan et le blanc pour les objets détectés, on ne peut calculer la luminosité avec de telles conditions, car tous ces objets sont lumineux (voir fig. 3.8 (image droite)). L'idée est de sélectionner la région d'intérêt (les coordonnées géométriques) de l'objet à partir de l'image binarisée qui est ensuite superposée sur l'image originale afin d'extraire uniquement la couleur des pixels dans cette image originale constituant la région englobante de chaque région segmentée. Avec cette liste de couleurs originales, nous appliquons deux étapes dans lesquelles nous conservons d'abord les pixels avec une intensité en niveau du gris qui se situe entre l'intervalle [200, 255]. Ensuite, on calcule leur moyenne par rapport au nombre total de pixels de notre image. Finalement, on garde uniquement les objets qui ont une moyenne supérieure ou égale à une moyenne d'illumination de 40 (après plusieurs tests expérimentaux, une graine faiblement illuminée est minimalement égale à cette valeur). Le schéma présenté à la figure 3.9 décrit cette opération ainsi que son code source suivi des commentaires pour expliquer le rôle de chaque ligne de code :

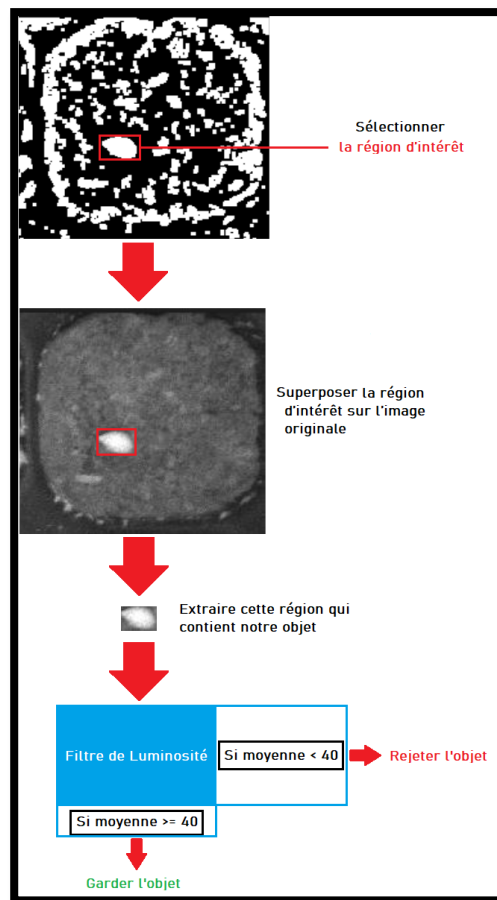


Figure 3.9 : La procédure d'extraction de la luminosité

Algorithme 3.3 : Fonction d'extraction de la luminosité d'un objet

```

double Pixel_Blanc = 0;
Region = CvInvoke.BoundingRectangle(Contour[0]); // Cette méthode extrait la région
d'intérêt du contour de notre objet.
Mon_Image_Original.ROI = Region; // On change la région d'intérêt de notre image
original avec celle de l'objet.
Image<Hsv, Byte> HSV = My_Image_Original.Convert<Hsv, Byte>(); // On transforme
notre image en teinte saturation valeur (HSV) qui un système de gestion de couleur en
informatique.
Image<Gray, Byte>[] channels = hsv.Split(); // On divise notre système de gestion HSV
pour avoir les 3 channels (teinte, saturation et valeur) pour garder uniquement la teinte
(channel[0]).

```

```

CvInvoke.InRange(Mon_Image_Original, new ScalarArray(new MCvScalar(200, 200,
200)),
new ScalarArray(new MCvScalar(255, 255, 255)), channels[0]); // Cette méthode garde
uniquement les pixels qui ont la teinte qui se situe dans l'intervalle [200,255] et les
sauvegarder dans notre objet channel[0].
for (int i = 0; i < channels[0].ROI.Height; i++) // notre boucle pour calculer le nombre
de pixel illuminer dans notre matrice.
{
for (int j = 0; j < channels[0].ROI.Width; j++)
{
if (channels[0].Data[i, j, 0] == 255 ||
channels[0].Data[i, j, 0] == 254)
Pixel_Blanc++; } }
return White_Px / (channels[0].ROI.Height * channels[0].ROI.Width); // retourner
la moyenne.

```

Après l'application de ce filtre, nous limitons de façon significative le nombre d'objets à classer et nous nous retrouvons avec un sous-ensemble d'objets qui ont plus de chance d'être des graines (voir fig. 3.10 (Image droite))



Figure 3.10 : Résultat du filtre de luminosité

3.3.1.4 Filtre de forme

L'avant-dernier filtre de notre système calcule la distance de tous les points sur le contour d'extrémités convexes et concaves des objets présents dans le vecteur de contours dont l'intérêt est de détecter les creux (concavités) de toutes formes.

Bien que la méthode de détection de défauts (concavités) soit fournie dans la bibliothèque Open CV, celle-ci est une boîte noire qui marque chaque déviation sur le contour original d'un objet par rapport à son enveloppe convexe, c'est-à-dire que cette méthode permet de dessiner une ligne joignant le point du début d'une concavité (voir fig. 3.11 (point vert)) sur la partie convexe de la forme et le point de fin de cette concavité (voir fig. 3.11 (point bleu)) aussi sur le contour de l'objet et ultimement marque le point sur le contour de la forme le plus éloigné qui constitue le point le plus creux de la concavité (voir fig. 3.11 (point rouge)), ensuite, nous prenons les coordonnées x et y de ces points pour calculer la distance entre l'enveloppe convexe et ce creux (voir fig. 3.11), en utilisant la formule mathématique sur le plan cartésien ci-dessous. Toutes les distances seront enregistrées dans un fichier CSV qui sera utilisé plus tard lors de la classification.

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

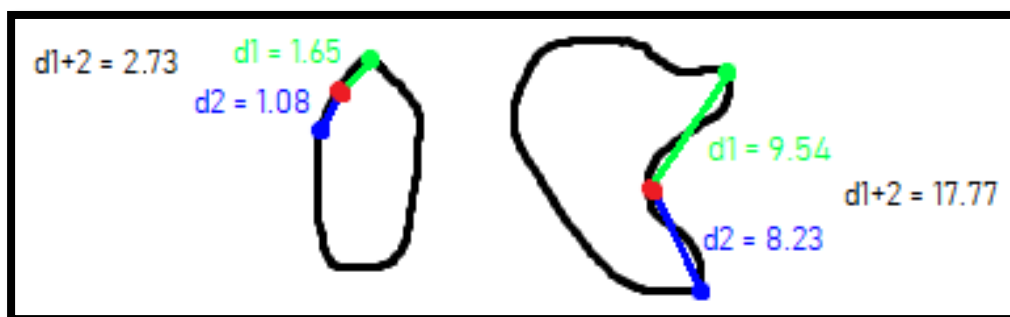


Figure 3.11 : Illustration sur le calcul de la distance du creux (concavité)

Algorithme 3.4 : Calcul de la distance du creux (concavité)

```
List<string> pointsdifference = new List<string>();
Point ConvexPointAvant = new Point();
Point ConvexPointAprès = new Point();
Point ConcavePoint = new Point();
double xDiff = 0;
double yDiff = 0;
double Distance1 = 0;
double Distance2 = 0;
double DistanceTotale = 0;
CvInvoke.ConvexHull(OneContour, hull, false, false); // créer l'enveloppe convexe.
CvInvoke.ConvexityDefects(OneContour, hull, defects); // Détecter les points convexes
et concaves.
Matrix<int> m = new
Matrix<int>(defects.Rows,defects.Cols,defects.NumberOfChannels);
defects.CopyTo(m); // Extraire les points dans une matrice.
channels = m.Split(); // divise les points en 3 catégories : point convexe avant, point
convexe après et point concave. Point[] ConvexVecAvant = new Point[defects.Rows];
Point[] ConvexVecAprès = new Point[defects.Rows];
Point[] ConcaveVect = new Point[defects.Rows];
    for (int conter = 1; conter < defects.Rows; ++conter) // stocker les points dans
chaque vecteur approprié.
{ ConvexPointAvant = OneContour[channels[0][conter, 0]];
  ConcavePoint    = OneContour[channels[2][conter, 0]];
  ConvexPointAprès = OneContour[channels[1][conter, 0]];
  ConvexVecAvant[conter] = ConvexPointBleu; // stocker les points convexes avant
dans un vecteur.
  ConcaveVect[conter] = ConcavePoint; // stocker les points concaves dans un vecteur.
  ConvexVecAprès[conter] = ConvexPointVert; // stocker les points convexes après dans
un vecteur. }
for (int conter = 1; conter < defects.Rows; ++conter) // Calculer la distance totale de tous
les point concaves {
```

```

xDiff = ConvexVecBleu[conter].X - ConcaveVect[conter].X;
yDiff = ConvexVecBleu[conter].Y - ConcaveVect[conter].Y;
Distance1 = Math.Sqrt((xDiff * xDiff) + (yDiff * yDiff));
xDiff = ConvexVecVert[conter].X - ConcaveVect[conter].X;
yDiff = ConvexVecVert[conter].Y - ConcaveVect[conter].Y;
Distance2 = Math.Sqrt((xDiff * xDiff) + (yDiff * yDiff));
DistanceTotale = (Distance1 + Distance2);
pointsdifference.Add((DistanceTotale).ToString()); // ajouter toutes les distances totales
dans une liste pour l'enregistrer plus tard dans un fichier CSV. }

```

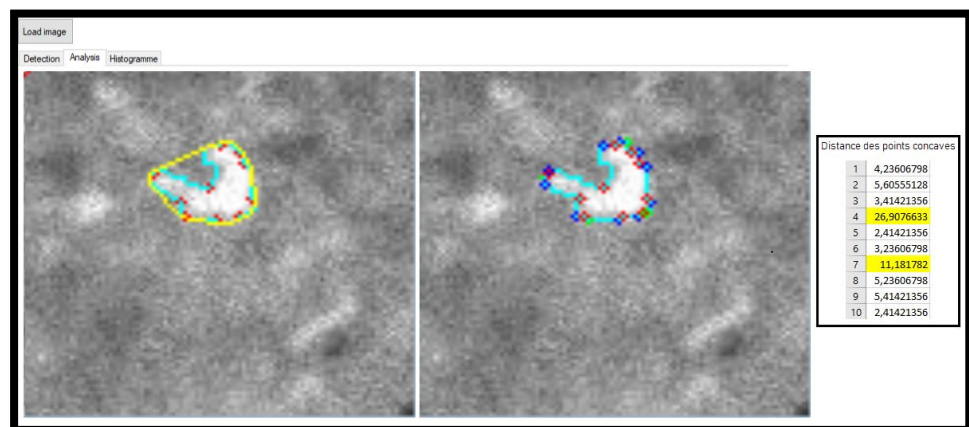
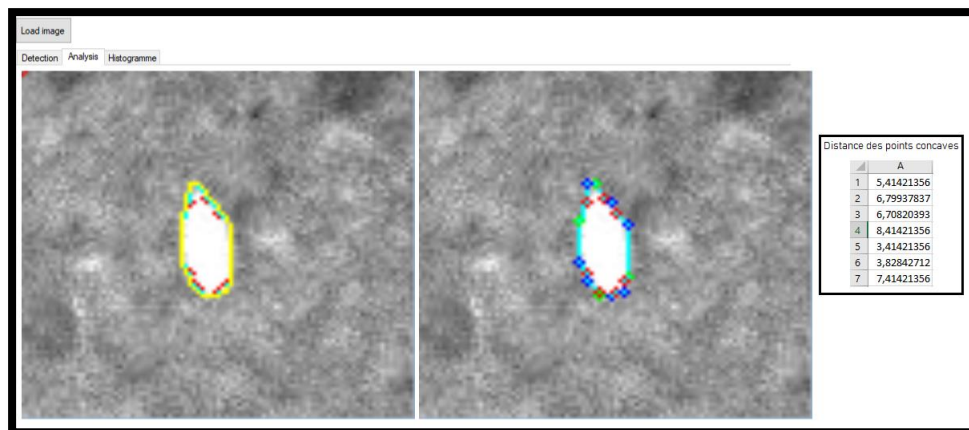


Figure 3.12 : Détection de forme à partir du programme de reconnaissance statistique (Couleur jaune : Forme convexe, Couleur bleu ciel : contour de l'objet, Couleur bleu : point du début de la concavité, Couleur verte : point de fin de la concavité, Couleur rouge, point le plus éloigné d'un creux (concavité) par rapport aux deux points convexes (départ et fin))

3.3.1.5 Classification

Tous les objets qui restent après l'application de tous les filtres ont de grandes chances d'être des graines, la classification est la dernière étape de notre méthodologie qui consiste à prédire la classe d'appartenance (graine ou non-graine) de ces objets, à savoir, si ces objets appartiennent aux résidus végétaux ou aux graines.

Le perceptron

Cette étape est plus ou moins un filtre qui utilise un simple perceptron d'apprentissage [24] qui est illustré dans la figure 3.13.

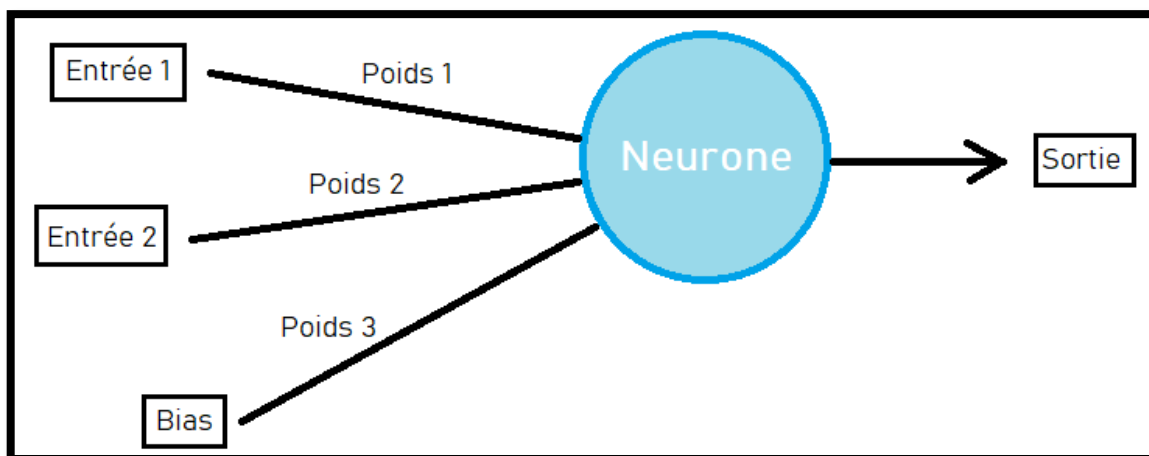


Figure 3.13 : Illustration sur le fonctionnement d'un perceptron

(l'entrée 1 et l'entrée 2 sont deux valeurs d'entrées (inputs), les poids sont a priori des valeurs aléatoires qu'on ajuste pour minimiser l'erreur, le bias est une autre entrée équivalente toujours à 1 et la sortie qui est le résultat ou la classification basée sur un entraînement approprié (output)).

Le fonctionnement du neurone est de générer des sorties linéaires séparables en fonction de polynômes (poids multipliés par des entrées) tel qu'on peut voir sur la figure 3.14.

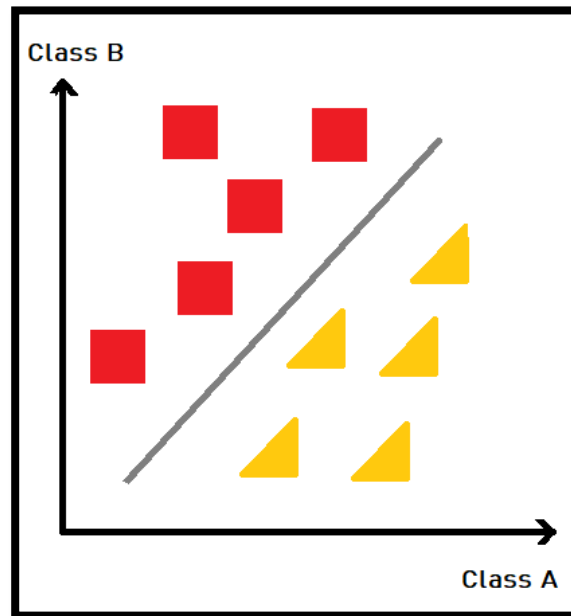


Figure 3.14 : Plan cartésien sur la séparation de deux classes (classe A et B)

Chaque forme représente une classe de sortie, ces sorties sont séparées par une droite (la ligne grise à la fig. 3.14) qui va être ajustée en rotation et en déplacement grâce au poids et la valeur du *bias*, une fois que l'erreur est minimisée le neurone génère un modèle d'entraînement qu'on va par la suite pouvoir utiliser pour prédire les nouvelles entrées telles que si la sortie appartient aux objets étiquetés par des triangles le perceptron retournera 0 ou 1 s'il appartient à la classe d'objets étiquetés par un carré.

Dans la phase d'entraînement, on fournit seulement les distances de six objets dont trois sont des graines et trois sont des résidus végétaux (le choix de ce nombre réduit de données d'entraînement est simplement dû au fait que les distances qui séparent le contour original d'une graine de sa forme convexe sont presque identiques et inférieures ou égales à une distance de point ou profondeur égale à 9 après plusieurs tests expérimentaux), chacune de ces distances est accompagnée d'une représentation en attribuant un 1 si la distance est

inférieure à 9 pour la classifier comme n'étant pas un creux (concavité non significative) et 0 si elle est supérieure à 9 pour en être considérée comme un creux (concavité significative). Dans la phase de prédiction, on fournit uniquement les distances extraites des contours d'un objet à prédire et au perceptron d'attribuer leurs classes d'appartenance, si la performance de la prédiction (voir fig. 3.15) est équivalente ou supérieure à 80%, alors l'objet est considéré comme graine ou un résidu végétal si elle est inférieure à cette valeur. Une valeur supérieure à 80% signifie qu'au moins 80% des concavités détectées sur le contour d'une forme de graine candidate sont de profondeur supérieure à 9. La figure 3.16 permet de visualiser l'interface utilisateur du système de reconnaissance statistique de formes.

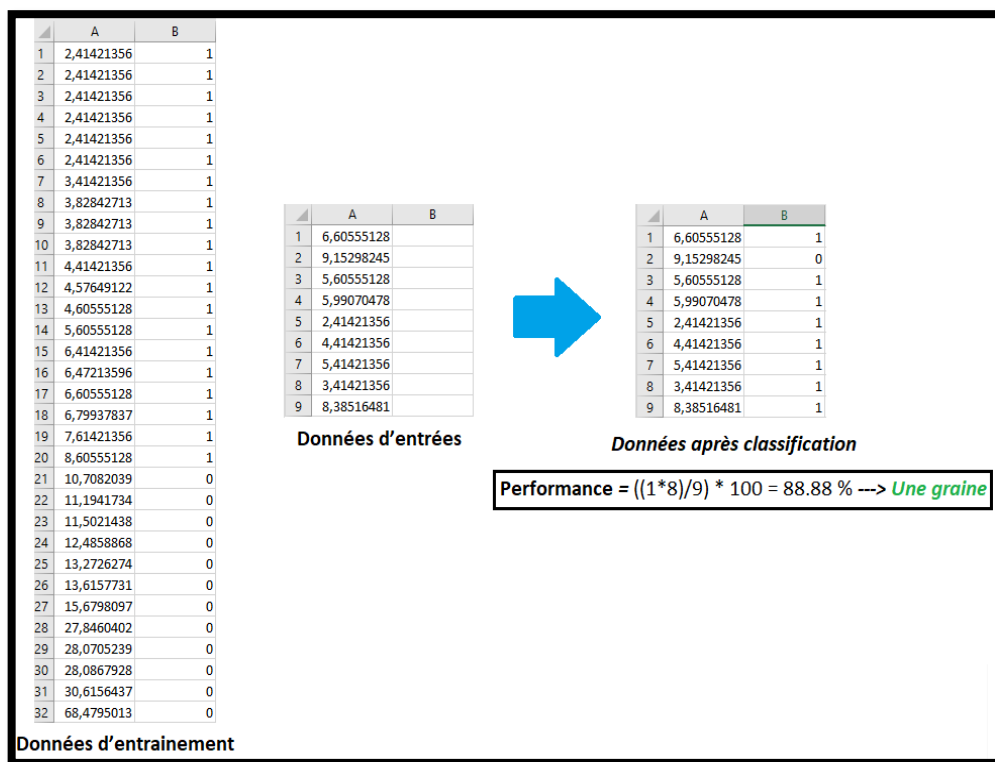


Figure 3.15 : Déroulement de la classification sous le programme de reconnaissance statistique

Voici les formules utilisées dans l'implémentation de notre neurone d'apprentissage et le code source permettant d'implémenter le perceptron :

- **Sortie (Output)** = $entrée[0]*poids[0]+entrée[1]*poids[1]+bias*poids[2]$
- **Poids[i] (des entrées)** = $poids[i] + \text{taux d'apprentissage} * \text{erreur local} * \text{entrée}[i]$.
- **Poids[i] (du bias)** = $poids[i] + \text{taux d'apprentissage} * \text{erreur local} * \text{bias}$.
- **Erreur local** = $\text{sortie désirée} - \text{sortie calculée}$.
- **Erreur total** = $\text{erreur total} + |\text{erreur local}|$.
- **Performance de la prédiction** = $(\text{nombre total de 1} / \text{nombre totale des distances}) * 100$.

Algorithme 3.5 : Un perceptron écrit en C# (tiré de Dan Sporic dans Coding.Vision. Permission de publication accordée le 25 Mars 2021)

```
public class Perceptron
{
    private static readonly Random _random = new Random();
    private readonly double[] _weights;
    private readonly double[] _inputs;
    private double _biasWeight;
    private const double _bias = 1;
    private const double learningRate = 1.0d;
    public double TotalError { get; set; }
    public Perceptron(int size)
    {
        _weights = new double[size];
        _inputs = new double[size];
        _biasWeight = _random.NextDouble();
        for (var i = 0; i < size; i++)
        {
            _weights[i] = _random.NextDouble();
        }
    }
    public double Pulse(double[] input)
```

```

{
    for (var i = 0; i < input.Length; i++)    {
        _inputs[i] = input[i];
    }
    var tmp = _inputs.Select((t, i) => ((IReadOnlyList<double>)_weights)[i] *
t).Sum();
    tmp += _bias * _biasWeight;
    return tmp >= 0 ? 1 : 0 ;
}
public void Train(double[] input, double desiredOutput)
{
    var output = Pulse(input);
    var localError = desiredOutput - output;
    TotalError += Math.Abs(localError);
    for (var i = 0; i < input.Length; i++)
    {
        _weights[i] += learningRate * localError * _inputs[i];
    }
    _biasWeight += learningRate * localError * _bias;
}
public void ResetErrorTracking()
{ TotalError = 0; } }

```

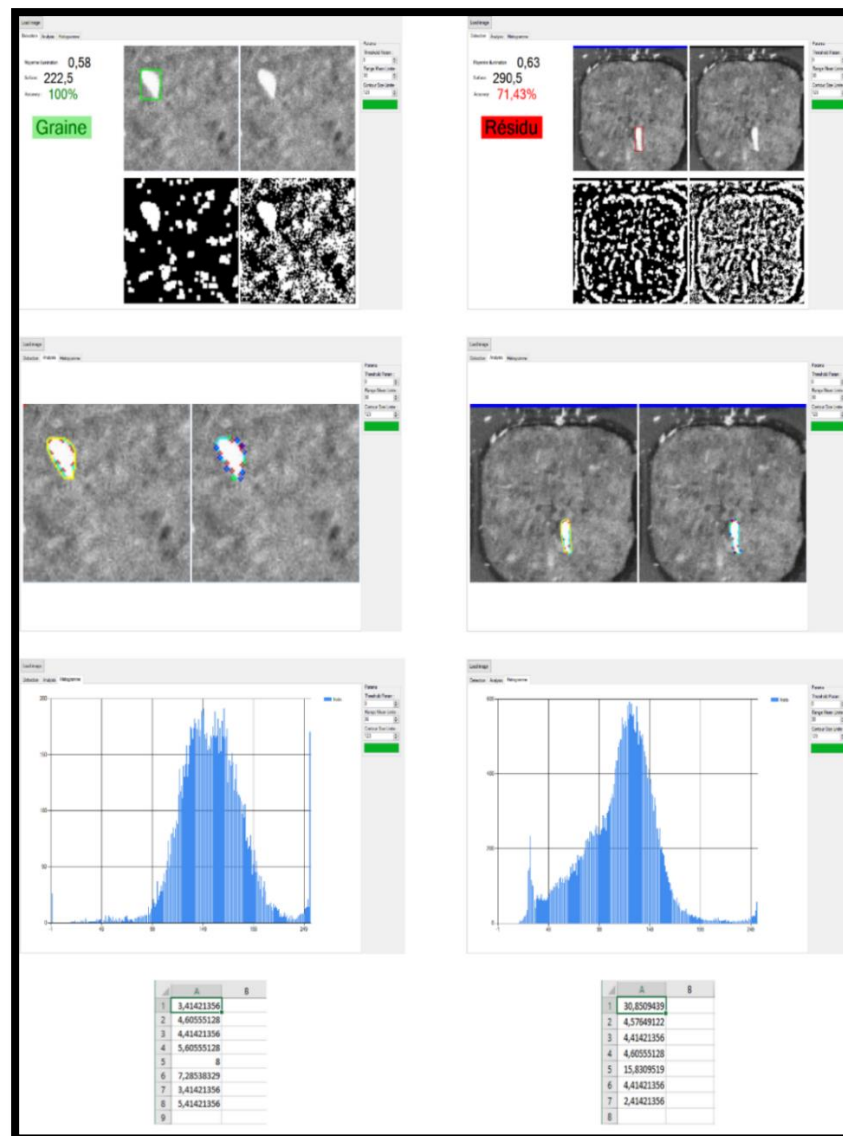


Figure 3.16 : Interface du système de reconnaissance statistique

3.3.2 Reconnaissance de formes basées sur les réseaux de neurones convolutifs (Apprentissage profond)

Nous passons à la dernière méthode implémentée de ce projet de recherche dans laquelle nous appliquons un apprentissage machine profond administré par des données étiquetées (supervisé). On détaillera chacune des étapes permettant de faire l'entraînement et de tester

un réseau de neurones convolutifs profond, accompagné aussi de leur code source écrit en langage python.

3.3.2.1 Segmentation

La phase de segmentation de cette méthode est très simple. On applique un seuillage binaire dans un intervalle de valeurs du niveau du gris proche du blanc (on rappelle que les graines sont blanches suite au rayonnement proche infrarouge) de valeur [180,255], suivies d'une opération morphologique d'ouverture (d'une dimension 3*3 qui est souvent utilisée dans le traitement d'image) en trois itérations pour bien réduire les bruits et lisser les contours de nos objets, le code suivant permet l'implémentation de la phase de segmentation.

Algorithme 3.6 : La procédure de segmentation

```
os.chdir("Cellules") # Cellules // GrainesTest
img_name = "G{0}.png".format(number)
img = cv.imread(img_name)
img2 = cv.imread(img_name)
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
_,thresh = cv.threshold(gray,180,255,cv.THRESH_BINARY)
kernel = np.ones((3,3),np.uint8)
opening = cv.morphologyEx(thresh,cv.MORH_OPEN,kernel, iterations = 3)
```

3.3.2.2 Architecture du réseau de neurones convolutif

Notre architecture fut inspirée du groupe de géométrie visuelle (*VGGNet*) [25] qui fut le premier finaliste d'*imageNet challenge* en 2014, cette architecture à la fois simple et très

efficace pour les images dont nous disposons, elle se caractérise par un empilement de couches convolutives avec des tailles de filtres croissantes.

La première couche convolutive se compose de deux filtres d'une taille 32×32 avec un noyau de dimension 3×3 , on a choisi la fonction d'activation linéaire rectifiée (annotée Relu) qui est la fonction la plus utilisée dans l'apprentissage profond, car très rapide et simple à calculer qui laisse passer les valeurs positives dans les couches suivantes du réseau de neurones. La deuxième et dernière couche convolutive est identique à la première sauf qu'on augmente la taille des deux filtres à 64×64 . Les deux couches convolutives sont suivies par une mise en commun maximum (*Max Pooling*) d'une dimension 2×2 .

Finalement, après avoir appliqué la méthode d'aplatissement (*Flatteninig*), on crée notre réseau de neurones qui se compose de deux couches cachées : la première avec 64 nœuds et 32 nœuds dans la deuxième couche et enfin une couche de sortie avec trois nœuds qui représentent les trois classes de classification : résidu végétal, une seule graine et deux graines ou plus (si elles sont collées). La figure 3.17 permet de schématiser l'architecture du réseau de neurones convolutifs utilisé dans le cadre de cette recherche, avec le code python permettant d'en faire l'implémentation.

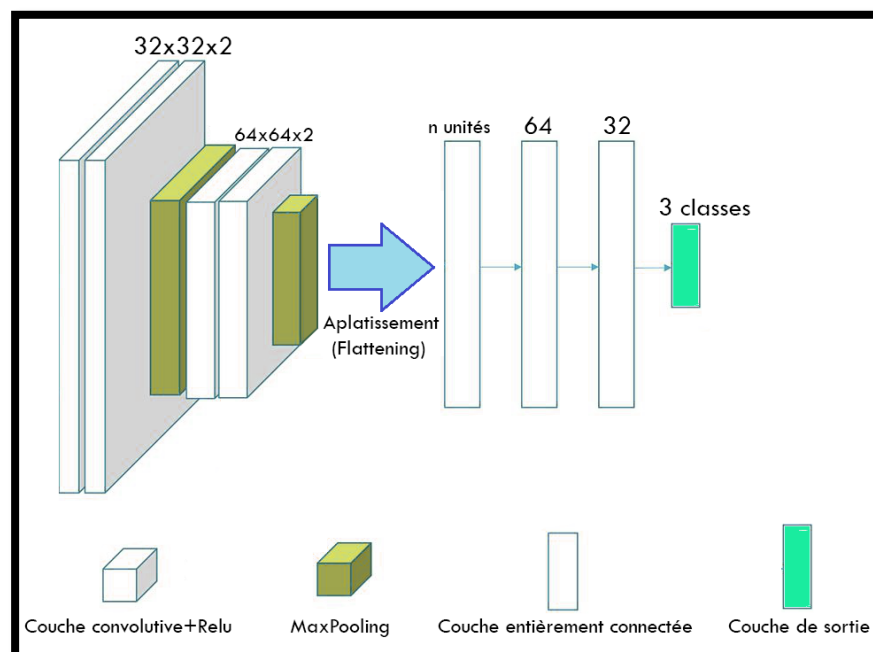


Figure 3.17 : Architecture du réseau de neurones convolutifs

Algorithme 3.7 : Conception du modèle d'apprentissage en python

```
def Creat_Model():
    model = k.Sequential()

    model.add(k.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(32, 32, 3)))
    model.add(k.layers.Conv2D(32, (3, 3), activation="relu"))

    model.add(k.layers.MaxPooling2D((2, 2)))

    model.add(k.layers.Conv2D(64, (3, 3), activation="relu"))
    model.add(k.layers.Conv2D(64, (3, 3), activation="relu"))

    model.add(k.layers.MaxPooling2D((2, 2)))
    model.add(k.layers.Flatten())

    model.add(k.layers.Dense(64, activation="relu"))
    model.add(k.layers.Dropout(0.2))

    model.add(k.layers.Dense(32, activation="relu"))
    model.add(k.layers.Dropout(0.2))

    model.add(k.layers.Dense(3, activation="softmax"))

    model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
    callbacks_list = [
        # Sauvegarde du modèle
        tf.keras.callbacks.ModelCheckpoint(
            f"MY_MODEL.h5",
            verbose=1,
```

```

        save_best_only=True,
        monitor="val_accuracy",
    ),
    # Arrêt automatique si aucune amélioration
    tf.keras.callbacks.EarlyStopping(
        patience=10, restore_best_weights=True, verbose=1, monitor="val_loss"
    )
]
history = model.fit(train_images, train_labels, epochs=100,
                    validation_data=(test_images, test_labels), callbacks = callbacks_list)
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
model.save('C:/Users/Sammy/PycharmProjects/OpenCv/MY_MODEL.h5')
return model

```

3.3.2.3 Entraînement du réseau de neurones convolutif

i. Préparation des données

Avant tout entraînement, nous devons disposer de données étiquetées, car il s'agit d'un apprentissage supervisé. Au cours de ce projet, nous avons utilisé les images de huit plateaux (cabaret) après leur passage dans le planteur pneumatique, six plateaux seront destinés à l'entraînement et les deux restants seront destinés à la phase de test de notre modèle de classification récemment entraîné.

Les données seront les images de tous objets détectés dans les cellules (rappelons qu'on a 288 cellules dans un seul plateau, ce qui donne un total de 2304 cellules, pour l'entraînement), pour extraire ces images, nous devons d'abord détecter les contours après avoir appliqué la segmentation sur l'image originale, ensuite nous sélectionnons la région d'intérêt de ces contours et nous augmentons ses coordonnées géométriques de 6 pixels pour

avoir une vue globale de l'objet sur l'arrière-plan pour que les contours et la forme soient bien distingués par notre réseau de neurones convolutif (voir fig. 3.18).

Algorithme 3.8 : Un zoom de 6 pixels sur une image

```
cntr,h = cv.findContours(opening,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)
echelle = 6

for i,cnr in enumerate(cntr):
    imgOriginal = cv.imread(img_name)
    x, y, w, h = cv.boundingRect(cntr[i])
    imCrop = imgOriginal[y-echelle:y + h+echelle, x-echelle:x + w+echelle]

cv.imwrite("C:/Users/Sammy/Desktop/imgtest/Objet{0}.jpg".format(i),imCrop)
```

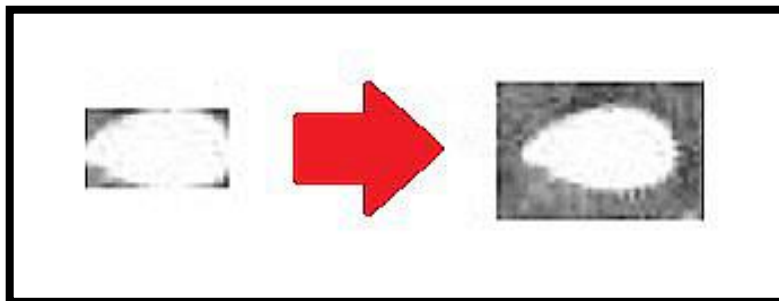


Figure 3.18 : Une graine après un agrandissement de 6 pixels

Une fois que toutes les images des objets détectés sont sauvegardées, nous passons au triage où nous allons les étiqueter au format suivant :

- s'il s'agit d'une graine : G.numéro d'image.png.
- s'il s'agit d'un résidu végétal : V.numéro d'image.png.
- s'il s'agit de deux graines ou plus : 2G.numéro d'image.png.

Après le triage, nous aurons en main les statistiques de nos données suivantes :

- nombre d'images d'une seule graine : 1334.
- nombre d'images d'un résidu végétal : 139.
- nombre d'images de deux graines ou plus : 108.

Environ 80% des données de chaque catégorie sont destinées à l'entraînement et ceux qui restante sont destinées aux tests de validation du réseau de neurones convolutif (voir Tableau 3.1).

Tableau 3.1 : Caractéristiques des données d'entraînement

Donnée	Nombres donnés d'entraînement	Nombres de données des tests de validation
Une seule graine	1123	211
Un résidu végétal	101	20
Deux graines ou plus	88	38
Total	1312	269

ii. Entraînement

L'entraînement du réseau de neurones convolutif est nécessaire pour générer un modèle de classification grâce aux données à notre disposition. Cette étape fonctionne de sorte que le réseau s'alimente de toutes les données fournies et génère un modèle de classification qui est aussitôt testé sur des données jamais vues par ce dernier dans l'unique intérêt d'analyser sa performance selon deux variables suivantes :

- la précision : le taux de bonnes classifications du réseau de neurones sur les données de validation.
- taux d'erreur : le taux de mauvaises classifications du réseau de neurones sur les données de validation.

Tableau 3.2 : les différents résultats de prédiction du modèle en phase d'entraînement (un résultat correct est bénéfique au taux de précision, tandis que l'erreur accroît le taux d'erreur)

La classe de la nouvelle instance	La Classe Prédit par le modèle	
	0	1
0	Correct	Erreur
1	Erreur	Correct

iii. Optimisation

Ces variables ne sont presque jamais parfaites après la première exécution, il existe un moyen d'optimiser notre modèle en appliquant plusieurs cycles d'apprentissage (epoch) dans lequel l'apprentissage automatique a eu une présentation complète de l'ensemble de données d'entraînement. Avoir cet ensemble de cycles d'apprentissage permet de mettre à jour les poids des neurones qui minimisent l'erreur de classification de façon à avoir une très bonne précision de classification. Il se peut bien qu'on obtienne une excellente performance de classification et que le taux d'erreur sur les données utilisées soit faible après plusieurs epochs, mais une dégradation significative des résultats sur de nouvelles données peut ensuite survenir, ce phénomène s'appelle le surajustement.

iv. Surajustement

Le surajustement (*Overfitting*) est un problème commun dans l'apprentissage automatique et la science des données. Cette anomalie est observée quand le modèle ne se généralise pas correctement sur les données jamais observées. Cette situation peut être causée par la présence de « bruit » qui peut faire référence à des données non pertinentes ou aléatoires présentes dans l'ensemble des données d'entraînement que le réseau de

neurones a mémorisées comme étant alors des données pertinentes, ce qui génère des prédictions basées sur ces dernières et performe alors mieux sur les données d'entraînement que sur les nouvelles entrées. Pour y remédier, plusieurs techniques existent pour minimiser les effets du surajustement, mais nous citons seulement celles qu'on a utilisées :

1. Validation croisée (*Cross-validation*) : cette technique permet d'évaluer les performances d'un modèle en séparant un échantillon des données globales destinées à l'entraînement qui seront par la suite prédites par le modèle récemment généré en phase d'apprentissage. Si par exemple la précision sur les données entraînées est 98% et celle sur la partie d'échantillons est 91%, alors une telle grande différence n'est que le synonyme d'un surajustement.
2. Ajustement des données : une technique simple, mais très minutieuse qui consiste soit à fournir plus de données d'entraînement pour détecter le signal (caractéristiques dominantes) de la classification ou soit écarter les données non pertinentes (bruit) qui sont jugées très complexe à déterminer et trempe le modèle de classification (voir fig. 3.19), mais rien n'empêche d'utiliser les deux si cela est envisageable.
3. L'arrêt précoce (*Early stopping*) : Lorsqu'on entraîne notre modèle de manière itérative (epochs), il arrive qu'après un certain nombre d'itérations, le modèle ne s'améliore plus et que sa capacité à généraliser devienne trop faible, parce qu'il commence à surcharger les données d'entraînement. L'arrêt précoce fonctionne de sorte qu'il arrête le processus de formation avant qu'il dépasse cette limite (voir fig. 3.20).

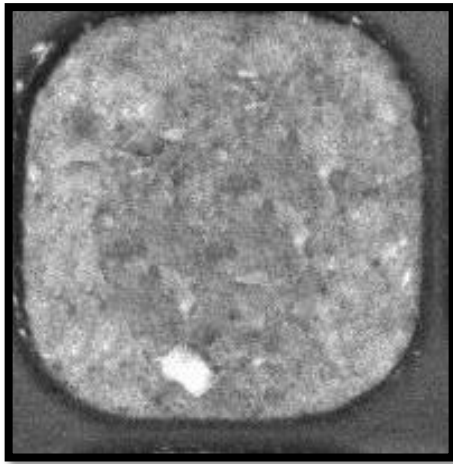


Figure 3.19 : Une donnée nuisible

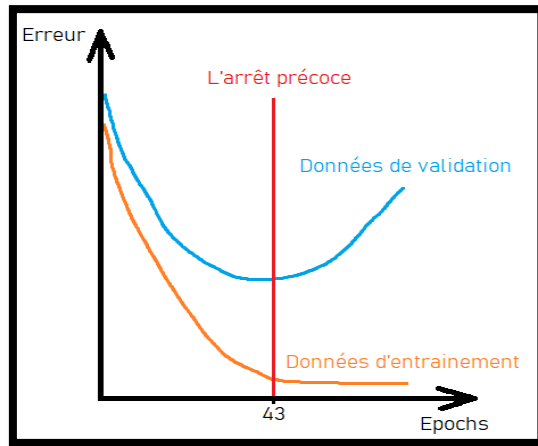


Figure 3.20 : L'arrêt précoce

3.4 Structure globale et classification

Le tableau ci-dessous (voir tableau 3.3) représente toutes les données contribuant à la production de notre modèle de classification accompagné de sa performance.

Tableau 3.3 : La structure et la performance du système de reconnaissance basé sur les neurones

Caractéristique	Valeur approprié
Nombre de données initiales	1543
Nombre de données d'entraînement	1312
Nombre de données de test	269
Ajustement des données (réapprovisionnement)	54
Ajustement des données (Suppression des données nuisibles)	16
Nombre de cycle (Epoch)	100
Précision (Validation croisée)	98%
Erreur (Validation croisée)	4%

Une très bonne précision sur les données d'entraînement est certes très intéressante, mais avant de conclure à l'excellence, nous devons aussi tester la performance de notre modèle sur des données nouvelles jamais observées (test de performance). Le code python de la méthode *Prediction()* , ci-après, permet de tester les performances du réseau de neurones convolutionnel entraîné.

La figure 3.21 permet de visualiser l'interface du module de reconnaissance de graines basée sur un réseau de neurones convolutionnels. Cette figure permet aussi d'observer les performances de classification du système de reconnaissance des objets détectés dans l'image d'une cellule d'ensemencement. Au départ, trois objets sont qualifiés de graines potentielles, mais seulement un d'entre eux est qualifié de graine.

Algorithme 3.9 : La fonction de prédiction

```
def Prediction(img):
    img = cv.resize(img, (32, 32))
    img = img.reshape(1, 32, 32, 3)
    img = img.astype('float32')
    img = img / 255.0
    model = Load_Model()
    predict = model.predict(img)
    return np.array(predict[0]).tolist()

def plot_value_array(predictions_array):
    predictions_array = predictions_array
    plt.grid(False)
    plt.xticks(range(3),["Vegetal ({}%)".format("%.2f" %
(100*predictions_array[0]),"Graine ({}%)".format("%.2f" %
(100*predictions_array[1]),
                "2 Graines ({}%)".format("%.2f" % (100*predictions_array[2]))])
    plt.yticks([])
    thisplot = plt.bar(range(3), predictions_array, color="blue")
    plt.ylim([0, 1]
```

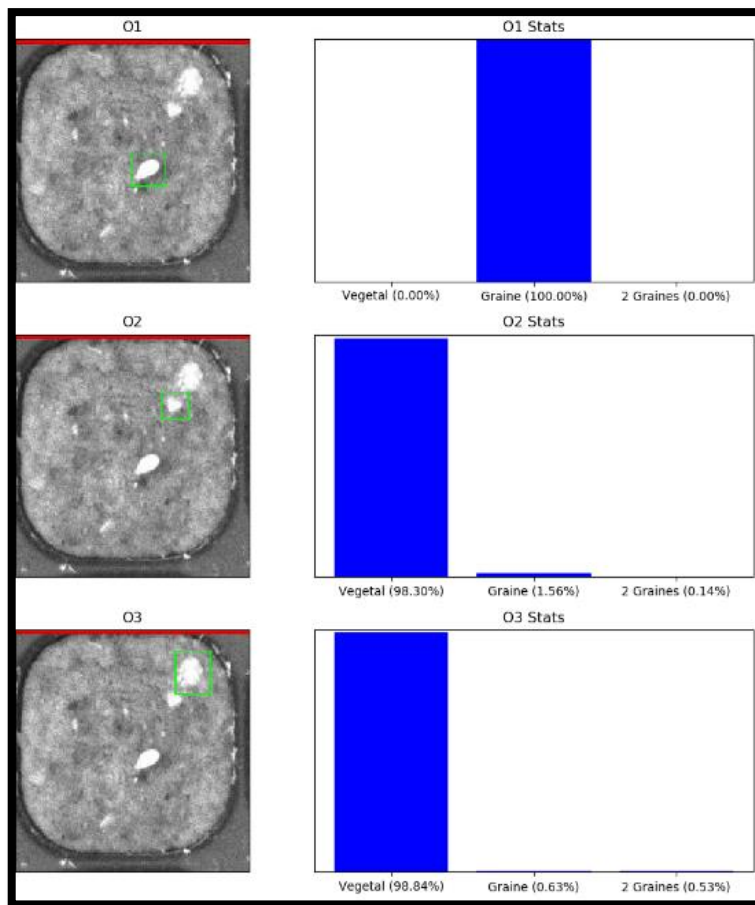


Figure 3.21 : Interface du système de reconnaissance basée sur les neurones

3.5 Conclusion

Dans ce présent projet, deux méthodes différentes de reconnaissance sont implémentées, la première utilise des couches de filtrage afin d'isoler les objets non conformes aux caractéristiques d'une graine à l'aide des données expérimentales statistiques alors que la deuxième utilise un réseau de neurones convolutionnel où l'importance est portée sur son architecture et plus spécifiquement ces couches convolutionnelles qui permettent d'extraire automatiquement à partir des données d'entraînement (images) les caractéristiques dominantes des graines de semence de résineux.

Le prochain chapitre est dédié aux tests dans l'intérêt d'évaluer laquelle de ces deux approches performe le mieux pour notre projet de recherche.

Chapitre 4

TESTS ET ANALYSES

4.1 Introduction

Dans ce chapitre, nous présentons le comparatif des performances des deux méthodes de reconnaissance implémentées dans ce projet de mémoire en se basant sur des formules d'évaluation des performances calculées sur un ensemble de données identiques.

4.2 Test de performance

Les deux programmes sont testés sur deux des huit (2/8) plateaux de cellules d'ensemencement, dans lesquels nous disposons de 576 cellules (2 * 288 cellules/plateau). La première étape de ces tests consiste à catégoriser les objets détectés selon le résultat prédit par les deux systèmes testés (système de classification basée sur une approche statistique et système de classification basée sur les neurones) et sa classification dans l'image d'origine, le tableau 4.1 suivant les définit :

Tableau 4.1 : Définition des métriques (TP : vrai positif, TN : vrai négatif, FP : faux positif, FN : faux négatifs)

Réalité	Classification	
	Graine	Végétal
Graine	TP	FN
Végétal	FP	TN

La seconde et dernière étape est de calculer les mesures d'évaluation de performance des classificateurs. Les critères de performance utilisés sont :

- A. **La précision de la classification (P-C)** : qui fait référence au pourcentage de la bonne prédiction, c'est-à-dire les moments dans laquelle le système a correctement prédit les objets détectés par rapport à tout l'ensemble des données du test. L'équation de calcul est la suivante (tirée de Witten et Frank, 2005) :

$$\mathbf{P-C} = (TP + TN) / (TP + TN + FP + FN) * 100\%.$$

- B. **La précision (P)** : est la proportion des instances correctement prédites par le système sur l'ensemble des données jugées positives. L'équation de calcul est la suivante (tirée de Witten et Frank, 2005) :

$$\mathbf{P} = TP / (TP + FP).$$

- C. **Le rappel (R)** : est le rapport de toutes les données qui sont correctement classifiées sur l'ensemble des données pertinentes. Appelé aussi « la sensibilité », l'équation de calcul est (tirée de Witten et Frank 2005) :

$$\mathbf{R} = TP / (TP + FN).$$

La mesure-F (M-F) : est la moyenne harmonique de la précision et du rappel, on ne peut maximiser la précision et le rappel simultanément, on peut maximiser cette métrique pour avoir un parfait équilibre des deux et ainsi améliorer le modèle de prédiction. L'équation de calcul est comme suit (tirée de Baeza et Riberio. 1999) :

$$\mathbf{M-F} = (2 * Précision * Rappel) / (Précision + Rappel).$$

4.3 Comparatif des résultats

576 cellules sont passées dans nos deux systèmes et voici le tableau comparatif (voir tableau 4.2) des mesures d'évaluation de leurs performances.

Tableau 4.2 : Les résultats de performance des deux systèmes

Métrique	Système de reconnaissance de formes statistique	Système de reconnaissance de formes basée sur les neurones
TP (Vrai positif)	549	563
TN (Vrai négatif)	13	58
FP (Faux positif)	41	20
FN (Faux négatif)	21	11
Cellule vide/Graine non détectable (sous terre)	70	64
Cellule Valide	507	513
Précision de la classification	90.06%	95,24%
Précision	93.05%	96,56%
Rappel	96.31%	98,08%
Mesure-F	94.44%	97,31%

4.4 Discussion des résultats

Le système de reconnaissance de formes statiques obtient un taux de classification de 90.06%. Cette approche a réalisé une précision des vrais positifs équivalente à 93.05% avec un rappel de 96.31% et leur moyenne harmonique (la mesure-F) produit un pourcentage égal à 94.44%. Ce qui concerne le système de reconnaissance de forme basée sur les neurones, sa classification est de 95.24% avec une précision des vrais positifs de 96.56% et un pourcentage de 98.08% pour la sensibilité, ces deux mesures engendrent une moyenne harmonique (la mesure-F) égale à 97.31%.

Ces statistiques indiquent que le système basé sur les neurones performe mieux que le système basé sur les données expérimentales statistiques, le système de reconnaissance de formes statistique performe moins dû au faible taux de précision qui n'arrive pas à déceler correctement les objets aux caractéristiques trompeuses de sorte qu'une graine à moitié sous terre devient non admissible au filtre de luminosité ou un résidu végétal parfaitement lumineux et partiellement circulaire qui traverse quant à lui tous les filtres et qui devient un intrus non désirable. Ces diverses situations expliquent la présence de presque le double de ces fausses prédictions (faux positifs) dans le système de reconnaissance statistique contre celles enregistrées par le système basé sur les neurones. Rappelons que cette catégorie représente des données nuisibles et fut écartée pour ne pas tromper notre apprentissage et améliorer la précision, dont la méthodologie basée sur les neurones en tire avantage.

4.5 Conclusion

La reconnaissance de formes basée sur les neurones est la meilleure méthodologie pour notre projet, très facile à implémenter où le travail se concentre plus sur le réseau de neurones artificiel et la manipulation de données qui offre des conditions d'entraînement parfaites de façon dynamique et automatique contrairement à la méthodologie basée sur la reconnaissance de formes statistique dans laquelle l'entraînement est quasi inexistant et se fie aux données expérimentales statistiques des caractéristiques de nos objets, mais sachant

que l'environnement naturel est constamment en changement et que les conditions d'ensemencement sont très variantes qu'il est alors impossible de toutes les considérer avec cette méthodologie.

CONCLUSION

Notre projet consistait à améliorer les performances de classification de la détection de graines de résineux d'un système antérieurement développé pour faciliter le processus d'assurance qualité d'un semoir pneumatique d'encensement de graines de résineux. Les améliorations apportées au système déjà implémenté sont basées sur deux méthodologies différentes dont chacune utilise sa propre méthodologie et fonctionnalité de vision artificielle. Ces améliorations sont implémentées par des fonctions et des algorithmes de détection et de classification prédéfinis et qui après entraînement ont été validés par une série de tests. Ces tests ont alors permis de calculer les métriques de mesure de performance de la classification de chacune des approches implémentées pour en tirer la meilleure et par le fait même celle-là plus appropriée à notre projet. Ce sont toutes les étapes envisagées pour ce projet de maîtrise qui ont été présentées en détail dans les chapitres précédents. Le système automatisé classification de graines de résineux est maintenant, suite à ces améliorations, doté d'une performance accrue qui améliore la précision de détection des cellules vides et la présence de résidus végétaux en un temps respectant le temps de réponse requis du système de détection (classification) et de comptage des graines de résineux qui offre un avancement majeur par rapport au projet initial. Par ces améliorations tangibles, le nouveau système de détection et de comptage de graines de résineux, permet encore plus d'appuyer le contrôle de l'assurance qualité pendant le processus d'encensement en économisant le temps et l'effort important requis pour effectuer les tâches inhérentes du processus d'assurance qualité qui étaient essentiellement manuelle avant l'implémentation des systèmes automatisés de détection et de comptage. Le nouveau système même avec ses

nouvelles améliorations reste cependant perfectible. Une amélioration envisageable serait de mieux détecter les graines sous terre que le système actuel a encore du mal à détecter. Dans les tests effectués dans ce mémoire, un peu plus de 70 cas sur 577 sont des cellules vides selon les résultats de détection du système, mais environ 60% sont en fait des cellules avec des graines cachées sous la surface du médium d'ensemencement d'une cellule. Si le système avait le pouvoir percevoir ces graines cachées, cela augmenterait grandement le taux de détection des semences et ainsi éviter une validation manuelle rendue alors inutile. Côté matériel, cette optimisation pourrait se faire en optant pour une caméra plus performante pouvant capturer ces graines même si elles sont enfouies sous la surface du médium d'ensemencement. Une telle caméra devra être munie d'un capteur suffisamment sensible pouvant aller capturer la réflexion ou le cas échéant l'émission de la réponse spectrale des composants physico-chimiques des graines, et ce même si elles sont partiellement ou même complètement cachées. Le système de détection et de classification des graines de résineux pourrait ainsi être entraîné avec ces nouvelles données pour le rendre plus robuste et donc plus sensible à la détection de ces graines cachées.

Nous n'excluons pas non plus des idées futures complémentaires sur la généralisation de l'utilisation de notre système qui pourrait servir à détecter par exemple les champignons ou faire le suivi des phases de germination de plants de résineux.

Ce système pourrait aussi éventuellement permettre de regrouper certains processus d'assurance qualité de la pépinière forestière, passant par, le contrôle de la qualité de l'ensemencement, jusqu'à la fin du processus de germination. L'ensemble des données collectées par ce système permettrait alors de mieux évaluer l'efficacité des processus de production des plants de résineux.

Ce projet de mémoire nous a de plus permis d'acquérir de nouvelles connaissances en matière de contrôle d'assurance qualité automatisé. Ce présent système reste en phase d'amélioration continue puisque d'autres fonctionnalités pourront éventuellement y être ajoutées. Ce système pourra aussi être installé dans d'autres pépinières forestières du même genre à travers la province de Québec et d'autres provinces au Canada. L'avenir et la pérennité de ce système de détection et classification de graines de résineux sont donc très prometteurs.

Bibliographies

1. Škrubej U, Rozman Č, Stajnko D. the Accuracy of the germination rate of seeds based on image processing and artificial neural networks. *Agricultura*. 2015.
2. Lurstwut B, Pornpanomchai C. Application of image processing and computer vision on rice seed germination analysis. *International Journal of Applied Engineering Research*. 2016.
3. Schmidhuber J, “Deep learning in neural networks: An overview.” *Neural networks : the official journal of the International Neural Network Society* 61 (2015): 85-117 .
4. arboquebecium. Arbres indigènes 2021 [Available from: <https://www.arboquebecium.com/fr/arbres-du-quebec/arbres-indigenes/><https://fr.wikipedia.org/wiki/NumPy>.
5. Patil NK, Yadahalli, R.M. Classification of food grains using HSI color model by combining color and texture information without performing pre-processing and segmentation. *World Journal of Science and Technology* 2012. 2012.
6. Silva C, Sonnadara U. Classification of Rice Grains Using Neural Networks.2013.

7. Patrício D, Rieder R. Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review. *Computers and Electronics in Agriculture*. 2018.
8. Mochida K, Koda S, Inoue K, Hirayama T, Tanaka S, Nishii R, et al. Computer vision-based phenotyping for improvement of plant productivity: A machine learning perspective. *GigaScience*. 2018.
9. Girard E. Détection automatique de semences de résineux pour l'évaluation en temps réel de l'efficacité d'un semoir. UQTR: UNIVERSITE DU QUEBEC. 2016.
10. Komyshev E, M.A G, Afonnikov D. Evaluation of the SeedCounter, A Mobile Application for Grain Phenotyping. *Frontiers in Plant Science*. 2017.
11. Muhammad Hasham ul Hassan MT, Muhammad Nadeem. Identification of Canola Seeds through Computer Vision Image Processing. *Journal of Information Engineering and Applications*. Ghazi University Dera Ghazi Khan, Dera Ghazi Khan, Pakistan. 2017.
12. Tanwar H, Sharma S, Mor V, Yadav J, Bhuker A. Image Analysis: A Modern Approach to Seed Quality Testing. *Current Journal of Applied Science and Technology*. 2018;27.
13. Forets FePQ. La pépinière de Berthier 2014. [Available from: <https://mffp.gouv.qc.ca/forets/semences/semences-pepinieres-berthierville.jsp>].
14. Aich S, Stavness I. Leaf Counting with Deep Convolutional and Deconvolutional Networks. 2017.
15. Wu T-C, Belteton S, Pack J, Szymanski D, Umulis D. LobeFinder: A Convex Hull-Based Method for Quantitative Boundary Analyses of Lobed Plant Cells. *Plant Physiology*. 2016.

16. Lefèvre, S.. “Approches multivaluées et supervisées en morphologie mathématique et applications en analyse d’image. (Multivalued and Supervised Approaches within Mathematical Morphology, and Applications in Image Analysis).” (2009).
17. OpenCV. Open Source Computer Vision Library 2021 [Available from: <https://opencv.org/>].
18. Laganiere R. OpenCV2 Computer Vision Application Programming Cookbook.2011.
19. Sporici D. Perceptron (C# Algorithm). 2012. Available from: <https://codingvision.net/c-perceptron-tutorial>.
20. Lurstwut B, Pornpanomchai C. Plant Seed Image Recognition System (PSIRS). International Journal of Engineering and Technology. 2011.
21. Jain A, Duijn R, Mao J. Statistical Pattern Recognition: A Review. IEEE Trans Pattern Anal Mach Intell. 2000.
22. Suzuki S, be K. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing. 1985.
23. Przybyło J, Jabłoński M. Using Deep Convolutional Neural Network for oak acorn viability recognition based on color images of their sections. Computers and Electronics in Agriculture. 2019.
24. Awty-Carroll D, Clifton-Brown J, Robson P. Using k-NN to analyse images of diverse germination phenotypes and detect single seed germination in *Miscanthus sinensis*. Plant Methods. 2018.

25. Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 14091556. 2014..

26. Douglass M, “Book Review: Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2nd edition by Aurélien Géron.” *Physical and Engineering Sciences in Medicine* 43 (2020): 1135-1136.

27. Arganda-Carreras, I. et al. “3D reconstruction of histological sections : Application to mammary gland tissue.” *Microscopy Research and Technique* (2010).

28. M. V. Giuffrida, M. Minervini, and S. Tsafaris. Learning to count leaves in rosette plants. In S. A. Tsafaris, H. Scharr, and T. Pridmore, editors, Proceedings of the Computer Vision Problems in Plant Phenotyping (CVPPP), September 2015.

29. Jain, Anil K. et al. “Statistical Pattern Recognition: A Review. 2000.

30. McCulloch, W. and W. Pitts. “A logical calculus of the ideas immanent in nervous activity.” *Bulletin of Mathematical Biology* 52 (1990): 99-115.

31. Hubel, D. and T. Wiesel. “Binocular interaction in striate cortex of kittens reared with artificial squint.” *Journal of neurophysiology* 28 6 (1965): 1041-59 .

32. Voulodimos, A. et al. “Deep Learning for Computer Vision: A Brief Review.” *Computational Intelligence and Neuroscience* 2018 (2018): n. pag.