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
CHEIKH TIDIANE DIABANG

AMELIORATIONS D'UN SYSTEME INTELLIGENT DE DETECTION ET DE COMPTAGE DE GRAINES DE SEMENCE DE RESINEUX.

OCTOBRE 2022

Université du Québec à Trois-Rivières

Service de la bibliothèque

Avertissement

L'auteur de ce mémoire ou de cette thèse a autorisé l'Université du Québec à Trois-Rivières à diffuser, à des fins non lucratives, une copie de son mémoire ou de sa thèse.

Cette diffusion n'entraîne pas une renonciation de la part de l'auteur à ses droits de propriété intellectuelle, incluant le droit d'auteur, sur ce mémoire ou cette thèse. Notamment, la reproduction ou la publication de la totalité ou d'une partie importante de ce mémoire ou de cette thèse requiert son autorisation.

REMERCIEMENTS

Je remercie le bon Dieu, tout puissant de m'avoir donné la force de mener à terme ce travail.

Je tiens à remercier grandement **Mr François Meunier** d'avoir accepté d'être mon **directeur de recherche**, son engagement, sa disponibilité ainsi que son soutien pour la réussite de ce projet. Et il a surtout développé en moi une capacité de recherche et d'engagement.

Je souhaiterais manifester ma reconnaissance particulièrement à **la secrétaire, Madame Guimond Chantal** pour tout le travail remarquable qu'il a eu à faire pour les étudiants durant ces deux années de formation.

Je remercie **tous les professeurs** qui ont eu à participer à ma formation depuis mon admission dans cet institut.

Je termine en remerciant tout **le personnel administratif** du département de mathématiques et informatique.

SOMMAIRE

La vision par ordinateur consiste à utiliser des systèmes performants afin de rendre compréhensibles certaines caractéristiques d'une image.
Le but de ce mémoire est d'améliorer un système déjà développé de détection et de comptage de graines de semence de résineux. Dans le procédé, nous utiliserons les réseaux de neurones convolutifs pour l'évaluation des performances d'ensemencement.
Le système implémenté, grâce à un code de couleurs, précise la présence d'une graine ou plus dans une cellule d'un plateau de même que l'absence de graine. Le choix des couleurs est défini de manière arbitraire. Ainsi une cellule ayant une couleur bleue indique la présence d'une seule graine ; une cellule colorée en vert représente qu'il y a plus d'une graine détectée et une cellule colorée en rouge réfère à l'absence de graines dans la cellule en question.
En réalité, des défauts sont repérés dans les résultats affichés. On a constaté que des cellules étaient colorées en bleu comme en vert alors qu'elles ne contenaient pas de graine. Aussi des cellules colorées en vert alors qu'elles ne contiennent qu'une seule gaine. À cet effet notre but est de faire en sorte que la détection et le comptage du nombre de graines dans une cellule soient le plus exact possible en se basant sur les réseaux de neurones convolutifs pour la prédiction des performances de notre système.

TABLE DES MATIERES

REMERCIEMENTS II

SOMMAIRE III

TABLE DES MATIERES IV

LISTE DES FIGURES VI

1 INTRODUCTION 1

1.1 Présentation du sujet 2
1.1.1 Contexte et problématique 4
1.1.2 Objectifs 7
1.2 Concepts Généraux 7

2 REVUE DE LA LITTERATURE 8

2.1 La vision par ordinateur 9
2.1.1 La segmentation 9
2.1.2 Le filtrage 11
2.1.3 Détection de contours 17
2.2 Les réseaux de neurones artificiels 22
2.2.1 Le perceptron multicouche 22
2.2.2 Les réseaux de neurones convolutifs 24
2.2.3 Les réseaux de neurones récurrents 29
2.3 Conclusion 32

3 METHODOLOGIE 33

3.1 Paramètres étudiés du système déjà développé 34
3.1.1 L'aire 34
3.1.2 Le périmètre 34
3.1.3 Le facteur de forme 34
3.1.4 La solidité 34
3.1.5 L'excentricité 35
3.2 Détection Automatique 35
3.3 Comptage de graines de semences de résineux 36
3.3.1 Méthode basée sur la hiérarchie 37
3.3.2 Méthode basée sur la hiérarchie et la surface moyenne de la graine 40
3.4 Evaluation à temps réel des performances d'ensemencement 45
3.4.1 Evaluation selon la hiérarchie 45
3.4.2 Evaluation selon la hiérarchie et la surface moyenne 49
3.5 Conclusion 50

4 PRESENTATION DES RESULTATS ET ANALYSES 51

4.1 Détection Automatique 52
4.2 Comptage de graines de semences de résineux 56
4.2.1 Approche basée sur la hiérarchie 56
4.2.2 Approche basée sur la hiérarchie et la surface moyenne de la graine 58
4.3 Evaluation à temps réel des performances d'ensemencement 60

4.3.1 Performances enregistrées avec la méthode basée sur la hiérarchie..........60
4.3.2 Performances enregistrées avec la méthode basée sur le couple hiérarchie - surface de la graine..........62
4.4 Conclusion..........63

5 CONCLUSION..........64

LISTE DES FIGURES

Figure 1.1: Planteur....2
Figure 1.2: Sous-unité du planteur....2
Figure 1.3: Plateau avec graines....3
Figure 1.4: Vue d'ensemble....3
Figure 1.5: Caméra....3
Figure 1.6: Caisson métallique....3
Figure 1.7: Plateau situé dans le caisson....3
Figure 1.8: Projecteur....3
Figure 1.9: Prototype avec système de vision....4
Figure 1.10: Processus d'assurance qualité manuelle (source [1])....4
Figure 1.11: Cycle de détection et de comptage du nombre de graine....5
Figure 1.12: Grille 4 x 12....5
Figure 1.13: Début Plateau....5
Figure 1.14: Résultats d'un plateau avec code de couleur....6
Figure 1.15: Résultats d'un plateau avec code de couleur ayant des erreurs de détection....6
Figure 2.1: Arêtes de texture....9
Figure 2.2: Arêtes d'illuminance....10
Figure 2.3: Arêtes d'illuminance basées couleur....10
Figure 2.4: Filtre Gaussien à échelles différentes....11
Figure 2.5: Filtre Moyenneur à différents masques appliqué à l'aile d'un avion....12
Figure 2.6: Filtre Médian à matrice carrée variable....13
Figure 2.7: Filtres minimum et maximum à matrice carrée variable....13
Figure 2.8: Dilatation morphologique....14
Figure 2.9: Erosion morphologique....15
Figure 2.10: Fermeture morphologique....16
Figure 2.11: Ouverture morphologique....16
Figure 2.12: Image originale couleur....17
Figure 2.13: Résultat Sy sur l'image d'origine....18
Figure 2.14: Résultat Sx sur l'image d'origine....18
Figure 2.15: Image originale dérivée seconde en x Dérivée seconde en y Laplacien....19
Figure 2.16: Résultat LoG....20
Figure 2.17: Image originale....20
Figure 2.18: Résultats DoG....21
Figure 2.19: Perceptron multicouche simplifié....22
Figure 2.20: Perceptron multicouche plus élaboré....23
Figure 2.21: Réseaux de neurones convolutifs simplifié....24
Figure 2.22: Réseaux de neurones convolutifs structurés....25
Figure 2.23: Graphe de la fonction sigmoïde....26
Figure 2.24: Graphe de la fonction tangente hyperbolique....27
Figure 2.25: Graphe de la fonction ReLu....27
Figure 2.26: Structure d'un réseau LSTM....30
Figure 2.27: Opérateurs de la structure du réseau LSTM....30
Figure 3.1: Niveaux hierarchiques de quelques contours....39
Figure 3.2: Surface moyenne graine EPN....43
Figure 3.3: Surface moyenne graine EPB....43

Figure 3.4: Surface moyenne graine PIG 44
Figure 3.5: Réseaux de neurones convolutifs utilisés pour le comptage de graines..........45
Figure 3.6: Version tensorflow 45
Figure 3.7: Version keras 45
Figure 3.8: Options usuelles classificatrices 48
Figure 4.1: Version PowerShell 52
Figure 4.2: Image originale d'un plateau d'épinette noire 52
Figure 4.3: Plateau de graines d'épinette blanche (15 cellules) 53
Figure 4.4: Image résultante de la détection automatique de graines EPN 53
Figure 4.5: Image résultante de la détection automatique de graine d'EPB 54
Figure 4.6: Détection automatique de graines d'EPB plateau (15 cellules) 55
Figure 4.7: Image résultante de la détection automatique de graines de PIG 55
Figure 4.8: Nombre de graines par cellules selon la hiérarchie sur PowerShell 56
Figure 4.9: Explorateur de fichier contenant les dossiers d'images de graines classées dans chaque classe (0 graine, 1 graine, 2 graines, etc.) 57
Figure 4.10: Cellules contenant des graines dans le bon répertoire 57
Figure 4.11: Cellules contenant des graines dont 3 mal classées 57
Figure 4.12: Nombre de graines selon la hiérarchie et le surface moyenne sur PowerShell 58
Figure 4.13: Image d'un plateau de 4 graines de PIG par cellule avec contours détectés .59
Figure 4.14: Les dossiers de l'explorateur de fichier contenant les nouvelles images.......59
Figure 4.15: Résultats méthode basée sur la hiérarchie 60
Figure 4.16: Bibliothèque additionnelle (scikit-learn) 60
Figure 4.17: Résultats matrice de confusion selon la hiérarchie 61
Figure 4.18: Résultats méthode basée sur la hiérarchie et la surface moyenne 62
Figure 4.19: Résultats matrice de confusion selon la hiérarchie et la surface moyenne ...63

CHAPITRE 1

INTRODUCTION

Dans le but de reboiser les forêts du Québec suite aux coupes de bois, la pépinière forestière de Berthierville produit annuellement plusieurs millions de petits plants de résineux, tels que, le pin blanc, le pin rouge, l'épinette blanche, l'épinette noire, le pin gris, etc. Pour ce faire, des semoirs pneumatiques et à tambours sont utilisés pour faire l'ensemencement par graines de ses plants. Pour améliorer le taux d'ensemencement et par conséquent le taux de germination des graines de résineux ensemencées, ces semoirs doivent être améliorés au niveau de leur processus d'assurance qualité. C'est dans ce contexte que s'inscrit ce présent mémoire soit de répondre à la question suivante : comment serait-il possible de maximiser le taux d'ensemencement des graines de résineux. La réponse proposée dans ce mémoire, est de développer un système de vision artificiel permettant la détection et le comptage automatique des graines de résineux pour un ensemencement optimal des plateaux d'ensemencement.

Ce mémoire comporte 5 chapitres, dont ce chapitre d'introduction qui permet de mettre au clair le sujet de notre mémoire de recherche en relatant le contexte, la problématique et les objectifs visés. Une brève présentation de quelques concepts sera également donnée dans ce chapitre. Le chapitre 2 permet de faire un état de l'art sur des terminologies qui sont abordées tout au long de ce document. Les chapitres 3 et 4 illustrent respectivement les méthodes utilisées et les résultats obtenus. Enfin, le chapitre 5 correspondant à la conclusion générale, revient en résumé sur le travail effectué.

1.1 Présentation du sujet

Le système de détection et de comptage de graines de semence de résineux est constitué principalement du système de vision artificiel. Ce système est précédé d'un système d'ensemencement. Ce dernier, présentant des irrégularités au niveau du planteur, fait que l'on peut se retrouver avec plus d'une graine ou même pas de graine dans une cellule d'un plateau.
La figure 1.1 montre le dispositif du planteur dans son ensemble. La procédure est que les graines sont injectées à travers la sous-unité du planteur (figure 1.2) en bloc de 4x12 sur 6 intervalles de temps, dans un plateau ayant 12 x 24 cellules (figure 1.3).

Figure B: Planteur

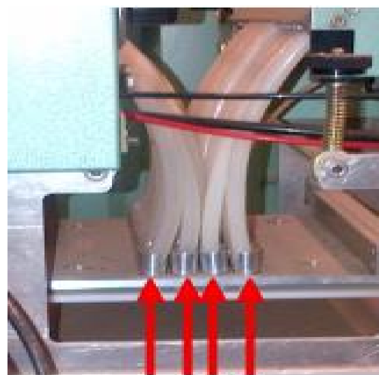


Figure A: Sous-unité du planteur

Figure D: Plateau avec graines

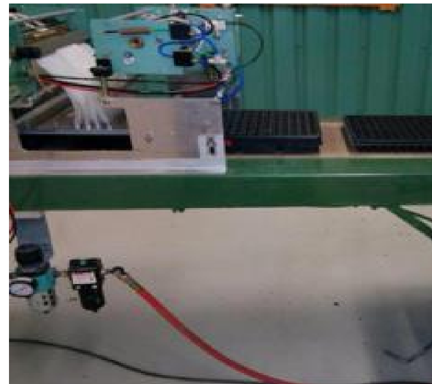


Figure C: Vue d'ensemble

Ensuite à l'aide du convoyeur (figure 1.4) qui sert de base aux plateaux, ces derniers sont acheminés vers le système de la vision artificielle.
Le caisson métallique (figure 1.6) contenant une caméra (figure 1.5), permet de recueillir l'image de l'état des cellules d'un plateau (figure 1.7). L'image obtenue est affichée via le projecteur (figure 1.8) à l'écran.

Figure E: Caisson métallique

Figure F: Caméra

Figure H: Plateau situé dans le caisson

Figure G: Projecteur

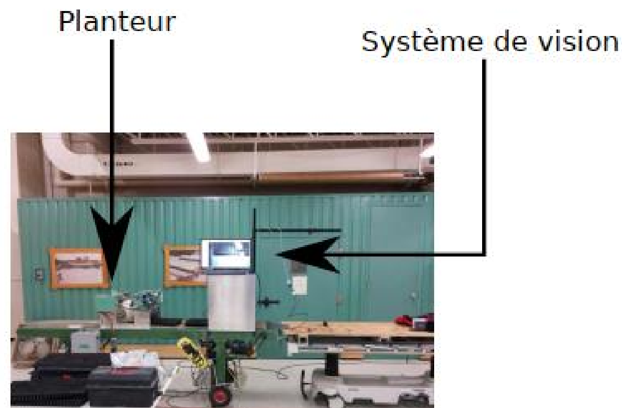


Figure I: Prototype avec système de vision

Dans le prototype avec système de vision (figure 1.9), on peut voir, hormis du caisson métallique muni d'une caméra, un écran d'ordinateur et le planteur du système d'ensemencement. En effet l'écran sert à visualiser les résultats de l'analyse du plateau dont le programme de traitement est installé dans un ordinateur.

1.1.1 Contexte et problématique

Compte tenu de ce qui précède, et avant l'avènement du système de vision artificiel, le système d'ensemencement était suivi d'agents d'assurance qualité pour apporter des corrections au défaut du planteur (figure 1.10).

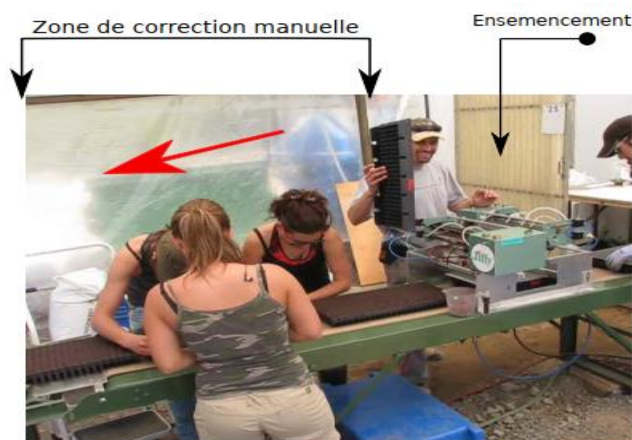


Figure J: Processus d'assurance qualité manuelle (source [1])

Cette activité de correction requiert une bonne concentration des agents. Ce qui est une chose pénible à long terme.

Ainsi le système de vision artificiel permettait de détecter les erreurs et d'orienter facilement l'opérateur dans les correctifs à apporter.
Le programme informatique déjà développé qui gère cet aspect de détection et de comptage de graines a donné une précision de 30% ([1] pages 78 et 79).
L'approche utilisée dans le cycle de détection et de comptage de graines est illustrée dans la figure (1.11) :

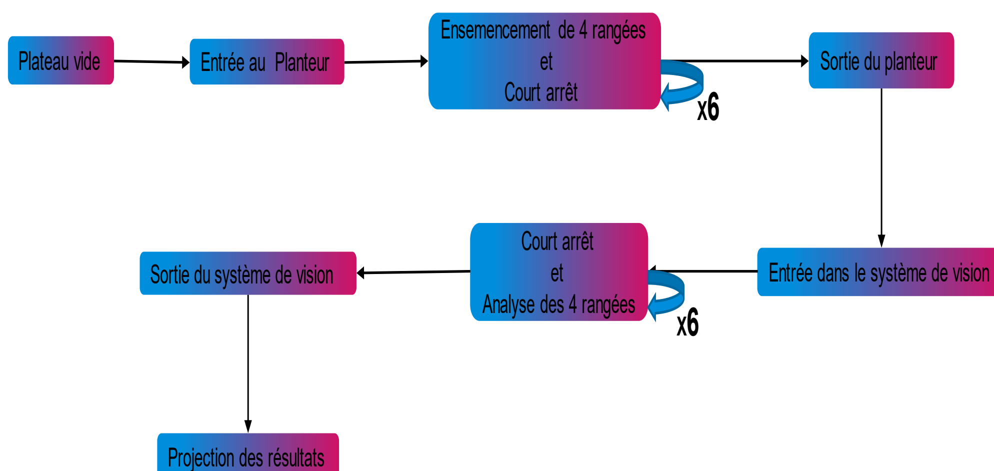


Figure K: Cycle de détection et de comptage du nombre de graine

Les figures 1.13 et 1.12 visualisent les images des séquences vidéo correspondantes à la phase de court arrêt et analyse des 4 rangées x 6 du cycle.

Figure M: Début Plateau

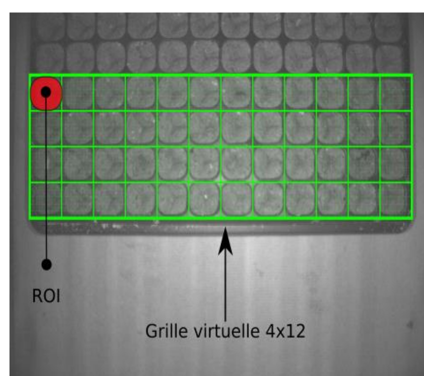


Figure L: Grille 4 x 12

ROI : Region Of Interest

Alors les critères qui sont utilisés (que l'on présentera en détails dans le chapitre 3) pour trouver le nombre exact de graines par cellule se rapportaient :

- Au périmètre
- A l'aire
- Au facteur de forme
- A la solidité
- A l'excentricité
- Au nombre de contour (0, 1 ou plus)

Après analyse des données, les résultats sont affichés par le projecteur (figure 1.14).

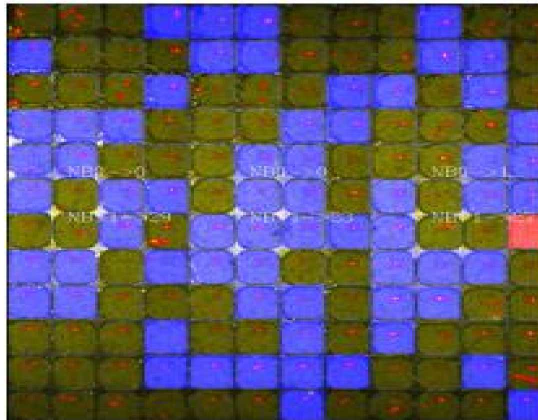


Figure N: Résultats d'un plateau avec code de couleur

Bleu : une seule graine détectée
Vert : plus d'une graine détectée
Rouge : cellule vide

On voit des cellules avec une seule graine, mais la cellule est coloriée en vert (figure 1.15). De même, on voit des cellules vides, mais elles sont colorées en bleu et en vert.

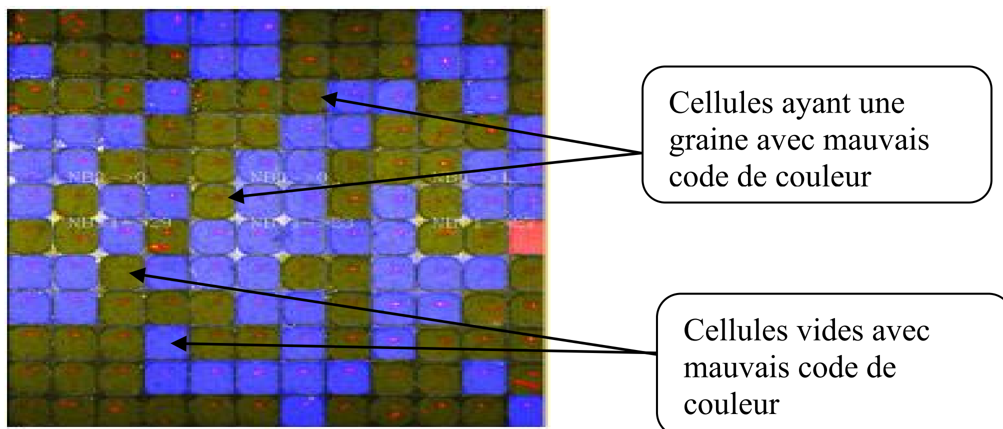


Figure O: Résultats d'un plateau avec code de couleur ayant des erreurs de détection

Tiré de [1]

Par suite des imperfections dans la détection et le comptage de graine, on nous confie la tâche d'améliorer ce système de vision artificiel.

1.1.2 Objectifs

L'objectif premier de ce travail de recherche est de traiter les images à l'aide d'algorithmes tels que les réseaux de neurones convolutifs afin de segmenter avec une meilleure précision les graines de semence de résineux.
Le second objectif est d'extraire des mesures caractéristiques permettant de mieux déterminer ensuite le nombre de graines présentes dans chaque cellule d'ensemencement. Ces caractéristiques extraites, serviront ensuite à modéliser un classificateur automatique, tel que des réseaux de neurones, pour permettre après un entraînement approprié, l'estimation du nombre de graines, et ce tout en minimisant le nombre de faux positifs pouvant être causés par entre autres le chevauchement de graines.
Le dernier objectif est de procéder à l'évaluation en temps réel des performances d'ensemencement.

1.2 Concepts Généraux

Dans cette partie, nous nous intéresserons à décrire brièvement les notions de vision par ordinateur et réseaux de neurones afin d'en montrer leurs intérêts. Le chapitre 2 consacré à la revue de la littérature reviendra en détail sur chaque point.

La vision peut être vue comme un processus de reconnaissance de l'information. Et l'ordinateur est une entité pouvant faire plusieurs tâches simultanément à un temps record. Alors, un outil comme un ordinateur effectuant des traitements dans le cadre d'identifier une structure par exemple une image de nature quelconque rentre dans ce cadre de vision par ordinateur.
La vision par ordinateur fait appel à plusieurs notions incontournables en traitement d'images. Parmi ces notions, nous avons la segmentation d'images, le filtrage, la détection de contours et la classification d'images. Nous développerons chaque aspect dans le chapitre 2.
Par ailleurs, un système capable d'identifier une image est caractérisé d'intelligent. Car l'intelligence est la faculté à apprendre pour s'adapter à l'environnement ou au contraire, la faculté à modifier l'environnement pour l'adapter à ses propres besoins [2].
Dans le cadre d'apprentissage, les réseaux de neurones apprennent des séquences en vue de faire une classification de l'élément étudié avec précision. Ils sont généralement appelés réseaux de neurones artificiels, car ils reprennent le fonctionnement biologique du cerveau humain. Le qualificatif intelligence artificielle est utilisé pour englober toutes études qui, à partir d'une base de données et d'un modèle pré entrainé, soient capables de prédire en sortie d'un réseau de neurones le groupe d'appartenance des données d'entrées.

CHAPITRE 2

REVUE DE LA LITTERATURE

Ce chapitre porte sur des notions énumérées dans la section 1-2 se rapportant à la vision par ordinateur. Chaque phase dans le processus de traitement d'images sera détaillée dans cette partie.
Ensuite, nous aborderons la structure d'un réseau de neurones artificiels avec les paramètres qui interviennent dans sa conception.

2.1 La vision par ordinateur

La vision par ordinateur peut être assimilée à la dernière marche d'une échelle dans le contexte de traitement d'images. Cette dernière a pour but de mettre en exergue les structures d'une image afin d'en interpréter le sens. Pour ce faire, plusieurs techniques sont utilisées. Dans ce qui va suivre dans cette section, nous allons présenter ces techniques.

2.1.1 La segmentation

La segmentation en traitement d'images consiste à scinder l'image en zone plus ou moins homogène en se basant sur des aspects comme : la texture, la luminance, la couleur, le niveau de gris, etc.

2.1.1.1 La texture

La texture peut servir d'élément de comparaison pour identifier les différentes zones dans une image comme illustrées dans la figure 2.1.

Figure P: Arêtes de texture

Tiré de [3]

2.1.1.2 La luminance

Il peut arriver que dans des cas spécifiques la luminance soi le facteur le plus en vue dans la structure de l'image en question. Ainsi ce critère sera alors le paramètre d'étude pour le traitement de l'image.

Figure Q: Arêtes d'illuminance

Tiré de [3]

Dans la figure 2.2, par rapport à l'arrière-plan, on voit les différents niveaux de brillance des pièces de monnaie. Cela est suffisant pour représenter les zones d'intérêts dans l'image résultante.

2.1.1.3 La couleur

La couleur est l'élément qui est généralement utilisé en traitement d'images. Dans le procédé, l'image d'origine est convertie en niveau de gris. Et comme une image est constituée de plusieurs pixels, chaque pixel à une valeur comprise entre 0 et 255. Alors on a tendance à mettre les pixels ayant les mêmes valeurs (niveau de gris) ensemble dans le but de classer les zones d'intérêts (figure2.3).

Figure R: Arêtes d'illuminance basées couleur

Tiré de [3]

Selon le contexte de l'image à traiter, plusieurs paramètres peuvent servir de repère pour une bonne segmentation.

2.1.2 Le filtrage

Le filtrage est l'opération de base en traitement d'image. Il permet de mettre en évidence certaines variations de l'image. En effet, le filtrage diminue l'effet du bruit de l'image d'origine. L'image résultante du filtrage sera par la suite dérivée afin de détecter les contours. Or l'opération de dérivation a tendance à augmenter le niveau de bruits. C'est pourquoi l'opération de filtrage doit être effectuée au préalable avant celle de la dérivation.
A ce jour, il y a plusieurs sortes de filtres qui sont utilisés en traitement d'images. Nous allons passer en revue les filtres les plus usuels pour la réduction du bruit sur l'image d'origine.

2.1.2.1Les filtres Linéaires

Le principe du filtrage linéaire est qu'à partir d'une image d'origine, on obtient une nouvelle image dont les valeurs de ses pixels sont constituées à partir des pixels avoisinants.
Nous présentons ci-après deux filtres appartenant à la famille des filtres linéaires les plus usuels : le filtre gaussien et le filtre moyenneur.

2.1.2.1.1 Le filtre Gaussien

Dans sa littérature le filtre gaussien est le produit de convolution d'une fonction représentative de l'image L (x, y) avec un noyau gaussien G (x, y;t). L'opération peut être présenté comme suit :

$$L(x,y;t) = L(x,y) \otimes G(x,y;t)$$

avec

$$G(x,y;t) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}}$$

t : représente la variance = σ^2

Dans l'image de la figure 2.4 suivante, on peut voir une image de niveau de gris filtrée avec le filtre gaussien à des valeurs de variance différente.

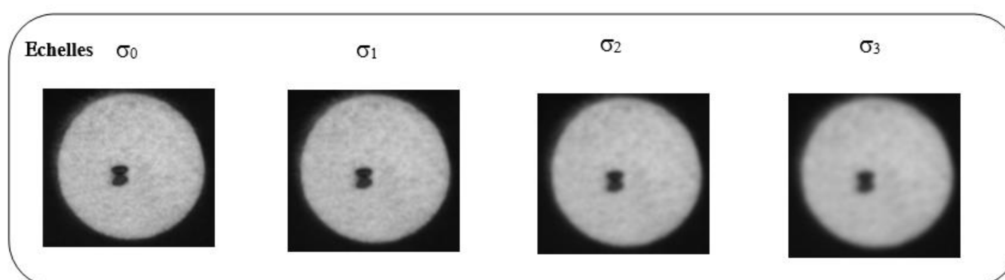


Figure S: Filtre Gaussien à échelles différentes

Le concept utilisé ici est que l'image d'origine est filtrée avec le filtre gaussien d'écart-type (échelle) t. En outre, chaque nouvelle échelle est un facteur constant de l'échelle précédente (**t_2 = k t_1**). Alors plus l'échelle augmente, plus les détails fins (plus petits que **t** = σ) s'estompent.

2.1.2.1.2 Le filtre moyenneur

Ce type de filtre calcule la moyenne pondérée des pixels situés au voisinage de chaque pixel. Ainsi l'image résultante est plus homogène. Cela permet de diminuer le bruit dans l'image, de même une visibilité réduite dans certaines zones.
La taille du masque utilisé influe fortement sur les résultats.
L'expression littérale du masque de taille n*n est de la forme :

$$\frac{1}{n^2}\begin{pmatrix} 1 & \ldots & 1 \\ . & . & . \\ . & & . \\ 1 & \ldots & 1 \end{pmatrix}$$

Dans la figure 2.5 ci-après, nous montrons les résultats du filtre moyenneur avec des masques 3*3 ; 5*5 et 7*7 appliqués sur une même image.

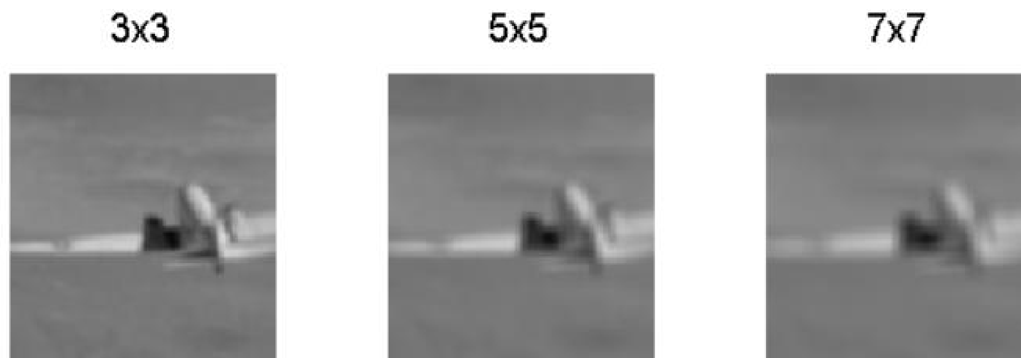


Figure T: Filtre Moyenneur à différents masques appliqué à l'aile d'un avion

Tiré de [4]

2.1.2.2Les filtres d'ordre

Ces types de filtres sont communément appelés filtres non linéaires. Ils rassemblent l'ensemble des pixels avoisinant et les classes par ordre croissant d'intensité ; ensuite le i-ème pixel de cette liste est sélectionné puis affecté aux résultats. Les filtres usuels de cette famille sont : le filtre médian, les filtres minimum et maximum.

2.1.2.2.1 Le filtre médian

Le filtre médian sélectionne la valeur des pixels du voisinage [5]. Alors dans le cas d'une matrice carrée de 7*7, 49 valeurs sont ainsi classées ; et la 25 ème valeur est sélectionnée comme résultat.
La figure 2.6 illustre ce cas de filtre médian à matrice carrée variable.

Figure U: Filtre Médian à matrice carrée variable

Tiré de [5]

2.1.2.2.2 Les filtres minimum et maximum

Ces filtres ordonnent les valeurs des pixels voisinant et choisissent soit la plus petite valeur ou la plus grande valeur de pixel. Ces types de filtres éliminent des variations très lumineuses ou très sombres, mais affectent la taille des objets.
Une illustration de chaque cas est présentée à la figure 2.7 avec des matrices carrées de taille différente.

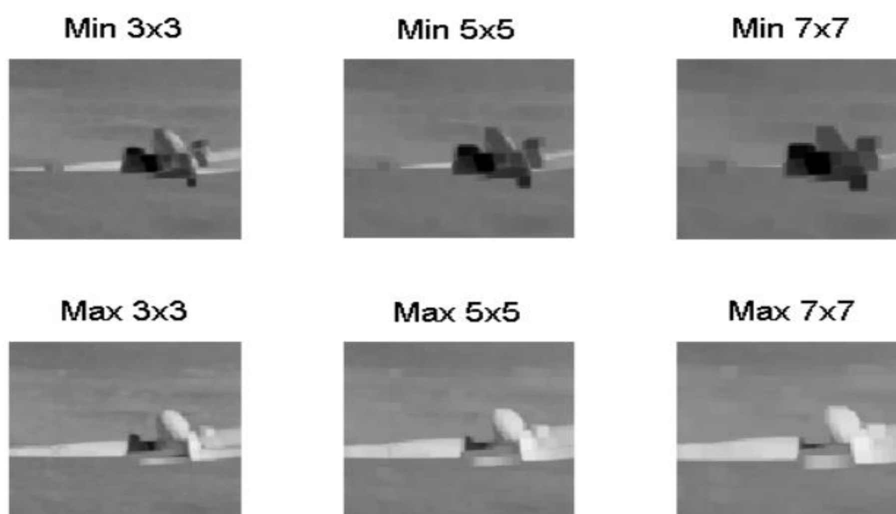


Figure V: Filtres minimum et maximum à matrice carrée variable

Tiré de [5]

En effet le filtre médian a l'avantage de mieux conserver les bords par rapport aux filtres minimum et maximum.

2.1.2.3 Les filtres morphologiques

Les filtres morphologiques, comme vous pouvez en douter, se basent sur la forme des pixels avoisinants. Ces filtres à l'origine agissaient sur des images binaires. Mais à ce jour les opérations sont étendues à des images à niveaux de gris.
Dans ce contexte, le pixel au voisinage est appelé élément structurant et peut être de taille et de forme différentes. La présentation de ces types de filtres se résume à la dilatation, l'érosion, l'ouverture et la fermeture morphologique.

2.1.2.3.1 Dilatation morphologique

Une dilatation morphologique, consiste à déplacer l'élément structurant sur chaque pixel de l'image, et à regarder si l'élément structurant « touche » (ou plus formellement intersecte) la structure d'intérêt [6]. La structure résultante est une plus grosse que la structure d'origine. Avec la taille de l'élément structurant, certaines particules peuvent se trouver connectées, et certains trous disparaître. La figure 2.8 montre cet aspect de dilatation morphologique.

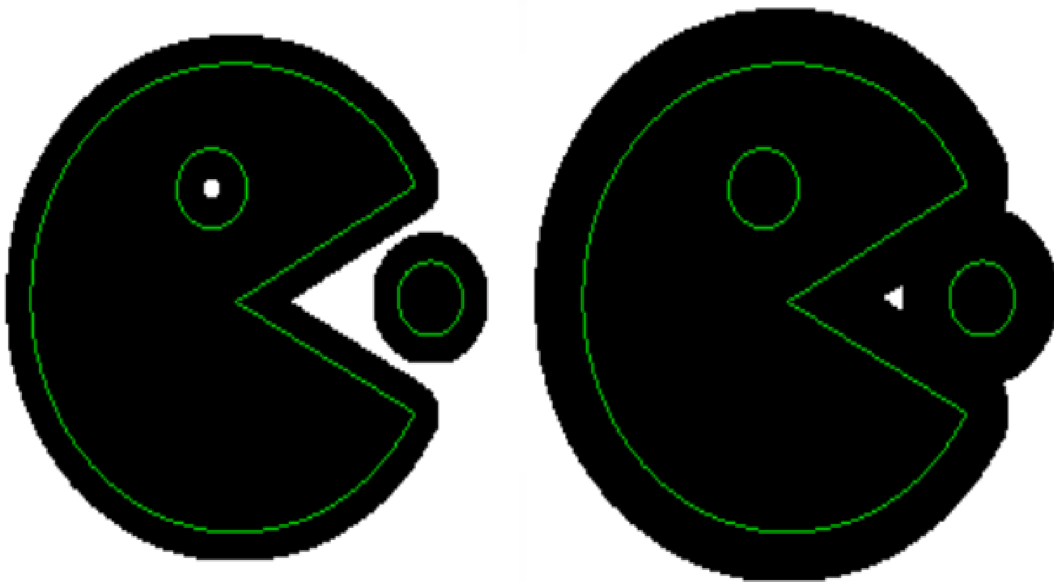


Figure W: Dilatation morphologique

Tiré de [6]

Les éléments structurants couramment utilisés sont le disque, isotrope, et le carré, qui permettent d'accélérer les calculs.

2.1.2.3.2 Erosion

L’érosion est une opération inverse de la dilatation morphologique. En effet, elle est définie comme une dilatation du complémentaire de la structure. La structure résultante est alors rognée (voir figure 2.9).

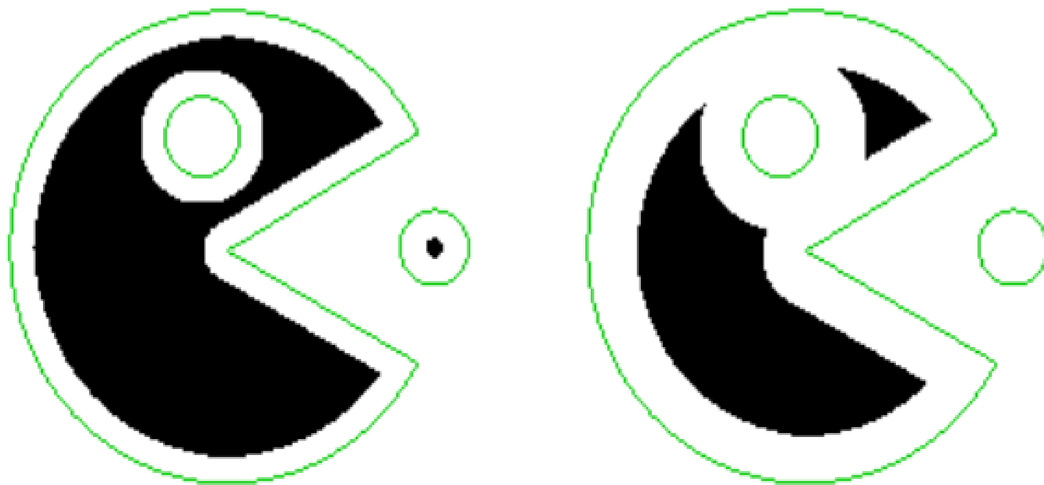


Figure X: Erosion morphologique

Tiré de [6]

On peut constater la disparition des particules plus petites que l'élément structurant utilisé, et la séparation éventuelle des grosses particules.
Sur des images à niveaux de gris, l'érosion est équivalente à l'application d'un filtre minimum, tandis que la dilatation est équivalente à l'application d'un filtre maximum [6].

2.1.2.3.3 Ouverture et fermeture morphologique

Les opérations de dilatation et d’érosion ont tendance à modifier la taille des structures de l’image. Ainsi elles sont utilisées en parallèle avec l’ouverture et la fermeture morphologique.
La fermeture morphologique est comme une dilatation suivie d'une érosion [7]. Elle a pour effets de faire disparaître les trous de petite taille dans les structures et de connecter les structures proches (figure 2.10).

Figure Y: Fermeture morphologique

Tiré de [7]

Par ailleurs l'ouverture morphologique comme une érosion suivie d'une dilatation. Elle a pour effets de faire disparaître les petites particules (dont la taille est inférieure à celle de l'élément structurant) et de séparer les grosses particules aux endroits où elles sont plus fines (figure 2.11)[7].

Figure Z: Ouverture morphologique

Tiré de [7]

L'ouverture et la fermeture morphologique ont une propriété d'idempotence : le résultat ne change pas si l'on applique plusieurs fois l'opérateur, il suffit de l'appliquer une seule fois [7].
Elles changent relativement peu la forme des grosses particules, en revanche, elles permettent de faire disparaître facilement les petites particules isolées, ou les petits trous à l'intérieur des structures. On les utilise donc souvent pour nettoyer le résultat d'une binarisation [7].

2.1.3 Détection de contours

La détection de contours facilite l'étude des éléments caractéristiques d'une image. Il n'existe pas de modèle typique optimal pour cette opération. Selon le contexte de l'image, on cherche parmi les techniques de détections de contours existantes, laquelle sera la plus adéquate.
Une présentation des opérateurs et filtres les plus utilisés est faite dans la suite de cette section.

2.1.3.1 Opérateur de Sobel

Cet opérateur donne un poids important aux 3 pixels se trouvant au voisinage du pixel central. Son masque de convolution s'écrit comme suit :

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Ces écritures permettent de mettre en évidence le gradient horizontal et le lissage vertical dans l'image résultante. Le gradient est une quantité vectorielle ayant une amplitude et une orientation. Il existe d'autres masques dans la détermination du gradient. L'opérateur de Sobel est l'un des plus utilisés. Son inconvénient est sa sensibilité au bruit. Les figures 2.13 et 2.14 montrent l'effet de l'opérateur de Sobel sur une image tirée de l'internet (figure 2.12) et traitée dans un de mes projets en classe.

Figure AA: Image originale couleur

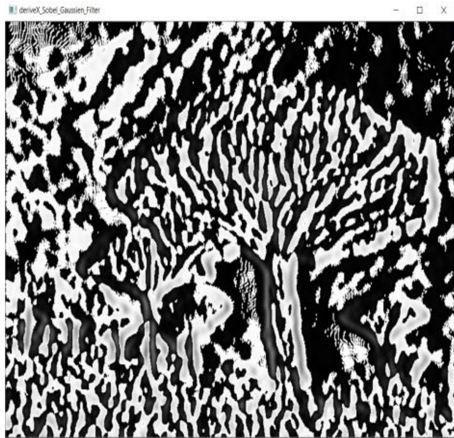


Figure CC: Résultat Sx sur l'image d'origine

Figure BB: Résultat Sy sur l'image d'origine

2.1.3.2 Opérateur de Prewitt

L'opérateur de Prewitt est également très utilisé à cause de sa simplicité. L'expression de son masque est un peu semblable à celle de l'opérateur de Sobel où il faut remplacer les valeurs des '2' par '1' et maintenir les signes '+' et '-'.

$$P_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \qquad P_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Il est aussi très sensible au bruit. Donc il est nécessaire d'appliquer un filtrage par exemple un filtre gaussien à l'image d'origine avant l'application de l'opérateur de Prewitt.

2.1.3.3 Opérateur de Roberts

Cet opérateur est peu utilisé comparé aux opérateurs de Sobel et de Prewitt, mais il est célèbre sur le plan historique.

$$R_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad R_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Ce masque proposé en 1965 permet de calculer un gradient le long des diagonales de l'image [8].

2.1.3.4 Le filtre Laplacien

Le filtre laplacien correspond à une dérivée du gradient. Or ce dernier (le gradient) est une dérivée de premier ordre, alors le filtre laplacien est souvent étiqueté de dérivée seconde.
L'expression du filtre laplacien peut s'écrire comme suit :

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

∇^2 : est l'opérateur laplacien

f : est la fonction représentative de l'image

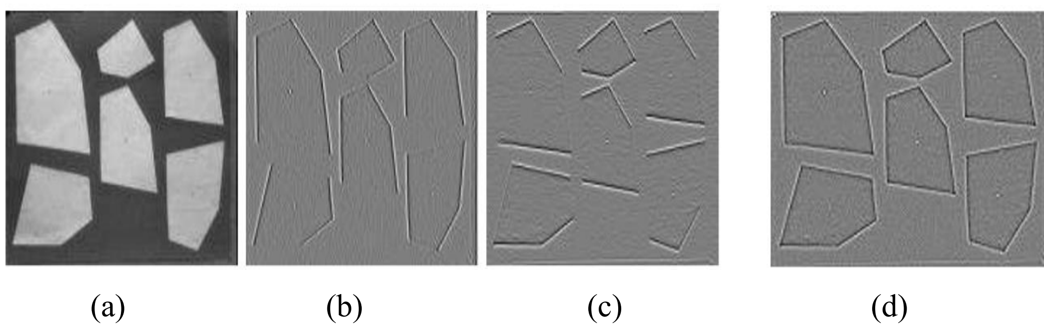
Le noyau de convolution du filtre laplacien est de la forme :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

D'autres formes de noyaux du filtre laplacien sont présentées ci-après :

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{ou} \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Des résultats obtenus sur une image avec les dérivées en **x**, en **y** et le laplacien appliqué à l'image en question sont illustrés dans la figure 2.15.



(a) (b) (c) (d)

Figure 2.15 : image originale Dérivée seconde en x Dérivée seconde en y Laplacien

Tiré de [9]

La dérivée seconde a tendance à amplifier le niveau de bruit sur une image. Ainsi le filtre laplacien est souvent combiné au filtre gaussien afin d'avoir de meilleurs résultats.

2.1.3.5 Le filtre Laplacien de Gaussienne (LoG)

Le filtre laplacien à lui seul est très sensible au bruit dans l'image. Et le filtre gaussien est assez robuste pour minimiser l'effet du bruit dans l'image résultant. La combinaison des deux filtres cités est appelée le laplacien d'une gaussienne.
L'expression littérale peut s'écrire comme suit :

$$LoG * I = \nabla^2 * G * I = \nabla^2 G * I$$

∇^2 : l'opérateur Laplacien
G : Filtre Gaussien dont la formule est donnée à la section II-1-2-1-1).
* : signe de la convolution

En se basant sur cette écriture on peut en déduire la formule **∇^2** * G(x , y) comme suit :

$$LoG(x, y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

La figure 2.16 ci-après montre le résultat du laplacien d'une gaussienne sur une image dont nous avons fait l'étude dans un projet en classe.

Figure FF: Image originale

Figure EE: Résultat LoG

2.1.3.6 Le filtre Différence d'une Gaussienne (DoG)

La différence d'une gaussienne, comme vous pouvez le deviner, est le résultat de la différence (soustraction) de deux images filtrées au préalable par un filtre gaussien dont leurs échelles ont un écart d'un facteur de 2.
L'effet de ce procédé sur des images filtrées avec le filtre gaussien d'échelles variables d'un facteur de 2 est illustré à la figure 2.18.

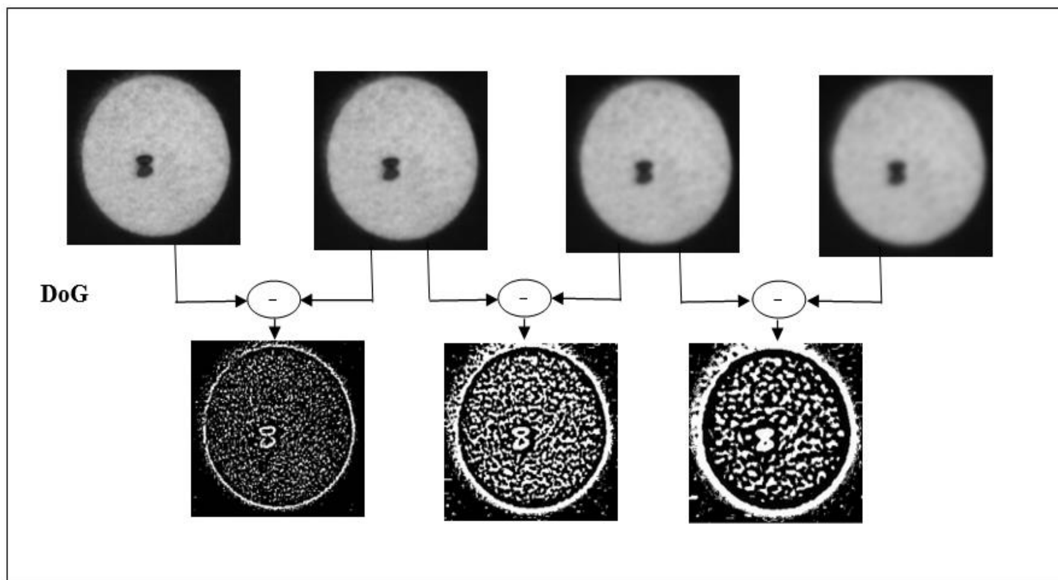


Figure GG: Résultats DoG

Les résultats obtenus sur une cascade d'images font apparaître sur chacune d'elle les contours des zones d'intérêts. On remarque l'atténuation de certaines traces au fur et à mesure que l'échelle augmente. Cela peut être un facteur remarquable dans certains cas en pratique.

2.2 Les réseaux de neurones artificiels

Les réseaux de neurones artificiels sont inspirés de la structure de fonctionnement du cerveau humain. C'est un système informatique en vogue dans plusieurs domaines à ce jour. Connus sous l'appellation d'intelligence artificielle par le commun du mortel, ils sont une partie intégrante de l'apprentissage profond ou deep learning et de l'apprentissage automatique ou machine learning.

Dans le contexte de traitement d'images, les réseaux de neurones artificiels sont utilisés afin d'identifier un ou des élément(s) parmi les structures de l'image. Cette identification se fait par le biais d'apprentissages. Il y a trois sortes d'apprentissage que sont : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage renforcé.

L'apprentissage supervisé requiert des données d'entraînement étiquetées. Des traitements sont ainsi effectués sur ces données jusqu'à ce que le programme atteigne des résultats satisfaisants.

L'apprentissage non supervisé consiste à utiliser une base de données non étiquetée. Le réseau de neurones analyse l'ensemble des données, et une fonction-coût lui indique dans quelle mesure il est éloigné du résultat souhaité [10]. Le réseau s'adapte alors pour augmenter la précision de l'algorithme.

L'apprentissage renforcé, le réseau de neurones est renforcé pour les résultats positifs et sanctionné pour les résultats négatifs [10]. C'est ce qui lui permet d'apprendre au fil du temps, de la même manière qu'un humain apprend progressivement de ses erreurs [10].

Hormis les modes d'apprentissages, les réseaux de neurones artificiels sont caractérisés par le nombre de couches se trouvant entre l'entrée et la sortie de ladite réseau. La manière dont ces couches intermédiaires sont exploitées est différente d'un modèle de réseaux de neurones à l'autre.

2.2.1 Le perceptron multicouche

Le perceptron multicouche est un algorithme d'apprentissage supervisé. Ce modèle est constitué de couches de neurones successives. L'information se propage dans un seul sens, de l'entrée vers la sortie en passant par les couches cachées. On ne peut pas avoir une connexion entre les neurones d'une même couche. Aussi tous les neurones d'une couche sont connectés à tous les neurones de la couche suivante.

La figure 2.19 illustre un exemple de réseaux de neurones du modèle perceptron multicouche.

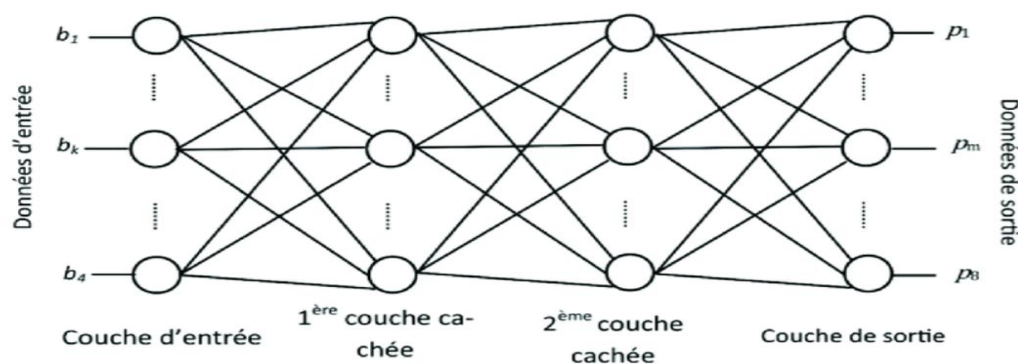


Figure HH: Perceptron multicouche simplifié

Tiré de [11]

La couche d'entrée se contente de coder les variables d'observations. La couche de sortie elle code la variable résultante.
Il est important de savoir que le nombre de neurones des couches d'entrée et sortie dépend du problème à étudier. L'article où la figure 2.18 est tirée concerne l'évaluation d'une méthode basée sur la transformation d'image dénommée IR-MAD (Iteratively Reweighted Multivariate Alteration Detection) proposée par Nielsen et Canty (2005) [11]. Les données d'entrées b_k avec k =1, 2, 3, 4 représentent le compte numérique associé à la bande numéro k du pixel à classifier ; et p_m avec m =1, , 8 données de sortie désignant la probabilité pour que le pixel traité appartienne à la classe m [11].
Dans le but de simplifier la structure d'un réseau de neurones, nous nous sommes basés sur le schéma précédent.
En effet, chaque neurone d'une couche est associé à un poids. Ce dernier est défini selon l'importance de l'entrée. Les neurones de la couche suivante effectuent une somme pondérée des entrées avec leurs poids respectifs et on ajoute une valeur seuil. Ensuite, une fonction d'activation aussi appelée fonction de transfert est appliquée au tout afin de générer une valeur en sortie. La figure 2.20 montre cet aspect plus élaboré d'un réseau de neurones.

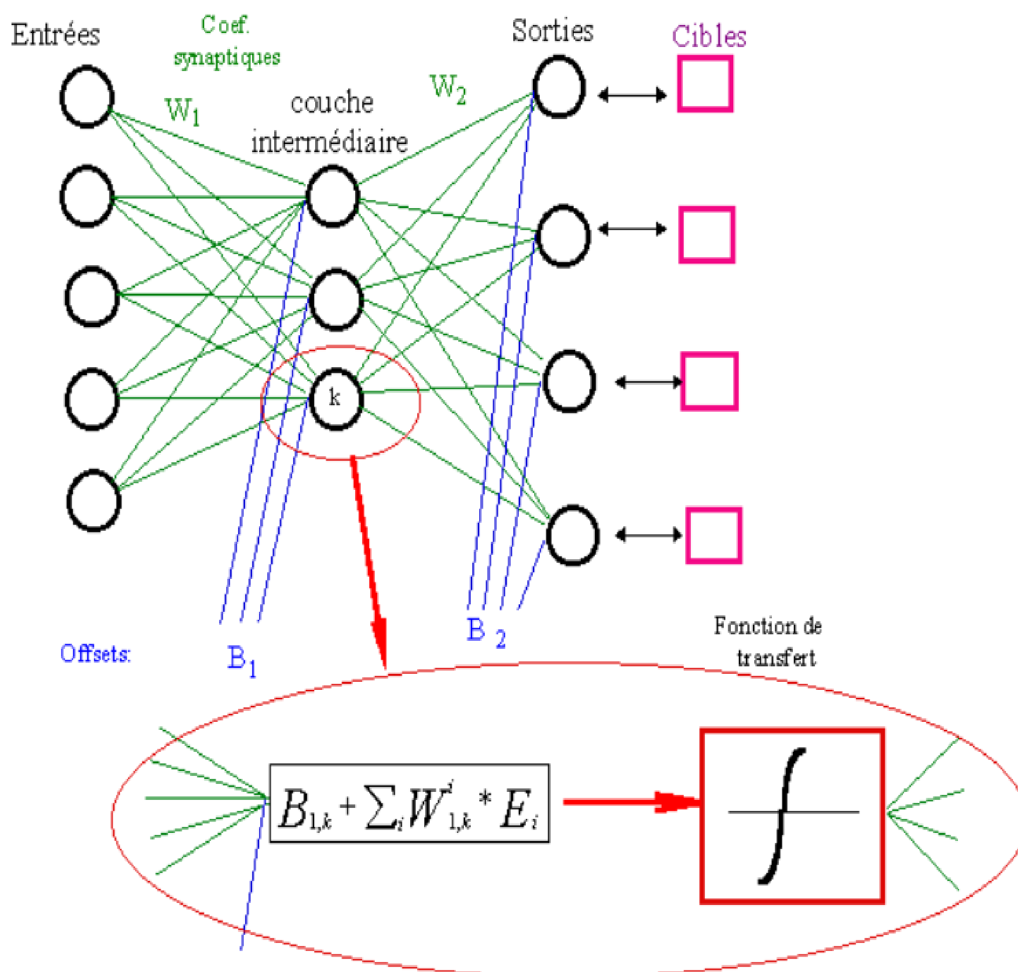


Figure II: Perceptron multicouche plus élaboré

Tiré de [12]

E_i : entrée i
$W_{i,k}$: le poids appliqué à l'entrée i de la couche intermédiaire k
$B_{i,k}$: la valeur du seuil rajoutée selon le niveau i à la couche intermédiaire k

Le symbole sous forme d'intégrale après le tout est la fonction d'activation. Son rôle est de donner une représentation significative ou non significative de la sortie. Nous reviendrons en détail sur les fonctions d'activation usuelles.
La cible représente une donnée de référence en sortie.

2.2.2 Les réseaux de neurones convolutifs

Les réseaux de neurones convolutifs sont un type de réseau de neurones multicouche. La particularité est qu'ils comptent au moins 5 couches, alors que le perceptron peut être monocouche c'est-à-dire constitué que des couches d'entrée et sortie, ou multicouche ayant une seule couche cachée ou plus.
Dans son fonctionnement, un réseau de neurones convolutifs cherche à reconnaître des motifs sur chacune de ses couches. Le résultat ainsi obtenu est transmis à la couche suivante. Nous vous montrons plus en détail ce processus à la figure 2.21.

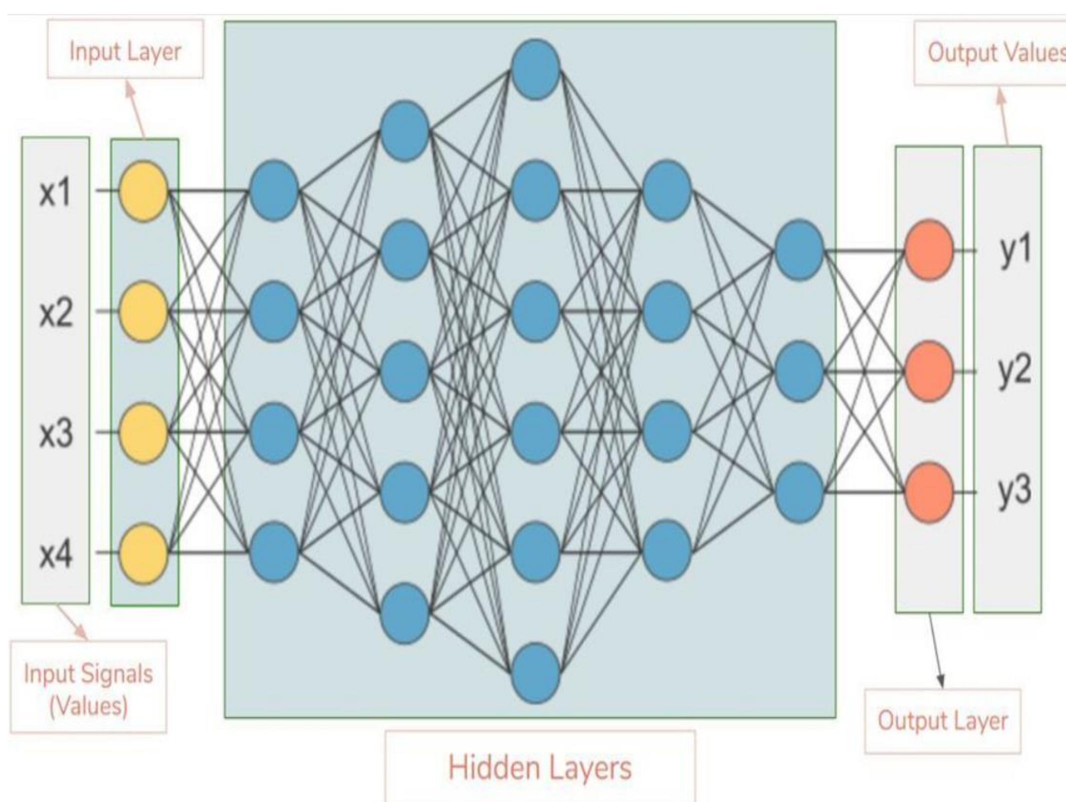


Figure JJ: Réseaux de neurones convolutifs simplifié

Tiré de [13]

Ce type de réseau de neurones est très utilisé en traitement d'image. Le choix d'au moins 5 couches cachées peut se comprendre selon la raison que les 4 premières

couches servent à l'apprentissage des caractéristiques et la 5ème couche est utilisée pour la classification des données. La figure 2.22 identifie chaque couche avec ses traitements idoines.

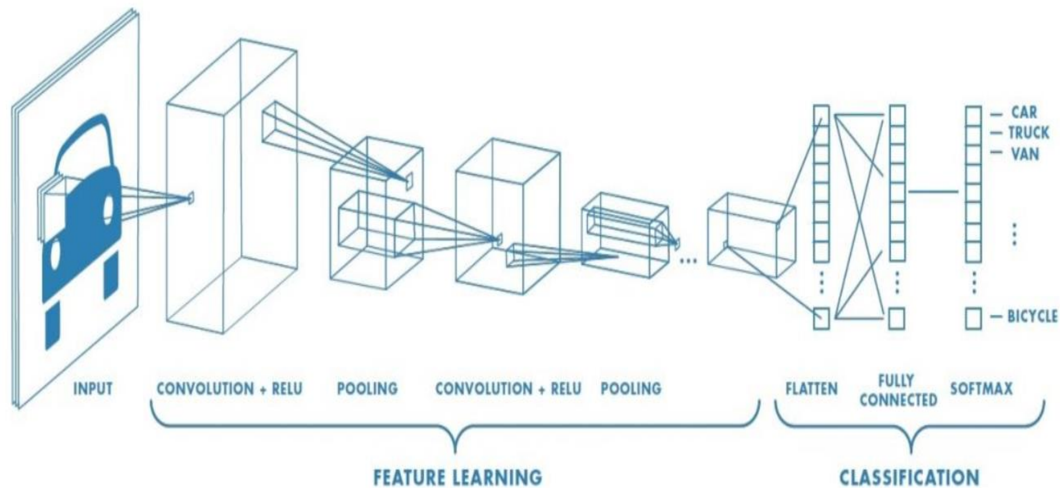


Figure KK: Réseaux de neurones convolutifs structurés

Tiré de [14]

Dans le schéma de la figure 2.22 le réseau de neurones prend en entrée l'image d'une voiture. Et à la fin du traitement, le réseau devra prédire le genre de voiture dont il s'agit. Pour ce faire, les réseaux de neurones convolutifs procèdent par différentes méthodes comme indiqué dans la figure en question. Nous vous présentons en détail les méthodes caractéristiques dans ce qui suit.

2.2.2.1 La convolution

La convolution dans ce contexte consiste à prendre la matrice représentative de l'image à qui on applique un filtre qui n'est rien d'autre qu'une matrice également qui représente quelque chose (une roue, un parebrise, un volant, un feu avant ou arrière, l'essuie-glace,…) . Ensuite, on note par rapport à notre filtre à quelle zone de l'image globale il ressemble le plus. Ainsi plusieurs convolutions différentes sont appliquées sur toute l'image.

2.2.2.2 Les fonctions d'activations usuelles

Les fonctions d'activation comme dans le système biologique humain, où l'influx nerveux qui arrive dans un bouton synaptique avec une certaine valeur excite ou non le nerf. Cette excitation est équivalente à l'activation. Et cette dernière correspond à une représentation de l'information en sortie afin de préciser la pertinence ou non de la structure étudiée. Les fonctions d'activation les plus utilisées sont élaborées dans les sous-sections de cette section.

2.2.2.2.1 Fonction sigmoïde

La fonction sigmoïde est une fonction d'activation généralement utilisée à la dernière couche du réseau de neurones. Cette fonction fournit un résultat compris entre 0 et 1.
L'expression littérale de la fonction sigmoïde s'écrit comme suit :

$$f(x) = \frac{1}{1 + e^{-x}}$$

Alors, nous pouvons donner la courbe représentative de cette fonction suivant des valeurs de x (figure 2.23).

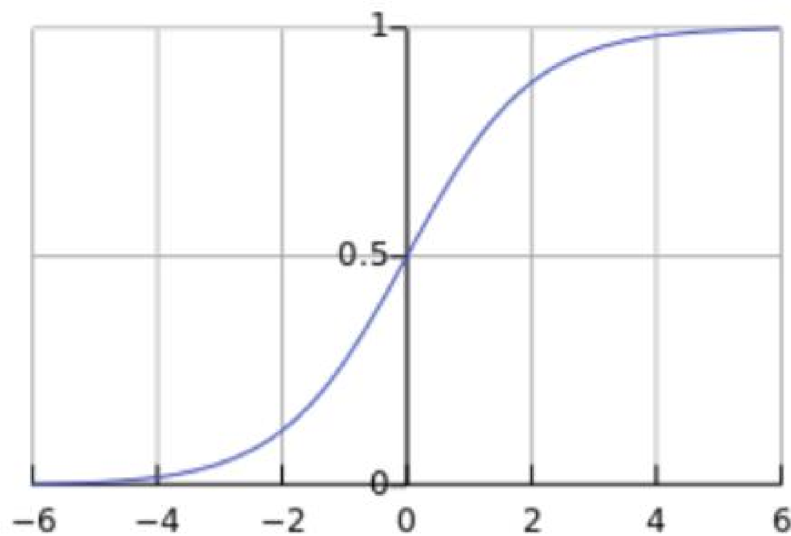


Figure LL: Graphe de la fonction sigmoïde

2.2.2.2.2 Fonction tangente hyperbolique

La fonction tangente hyperbolique ressemble à la fonction sigmoïde. La différence est que la fonction tanh produit un résultat compris entre -1 et 1.
Elle est généralement utilisée comme fonction d'activation à l'intérieur du réseau au niveau des couches cachées par ce qu'elle admet des valeurs positives et négatives, ce qui augmente la capacité d'apprentissage du réseau [15]. En revanche, on l'évite à la dernière couche si le résultat est le fruit d'une classification (valeur de 0 ou 1).
L'expression littérale de cette fonction est la suivante :

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

La courbe représentative de la tangente hyperbolique est représentée dans le graphe de la figure 2.24.

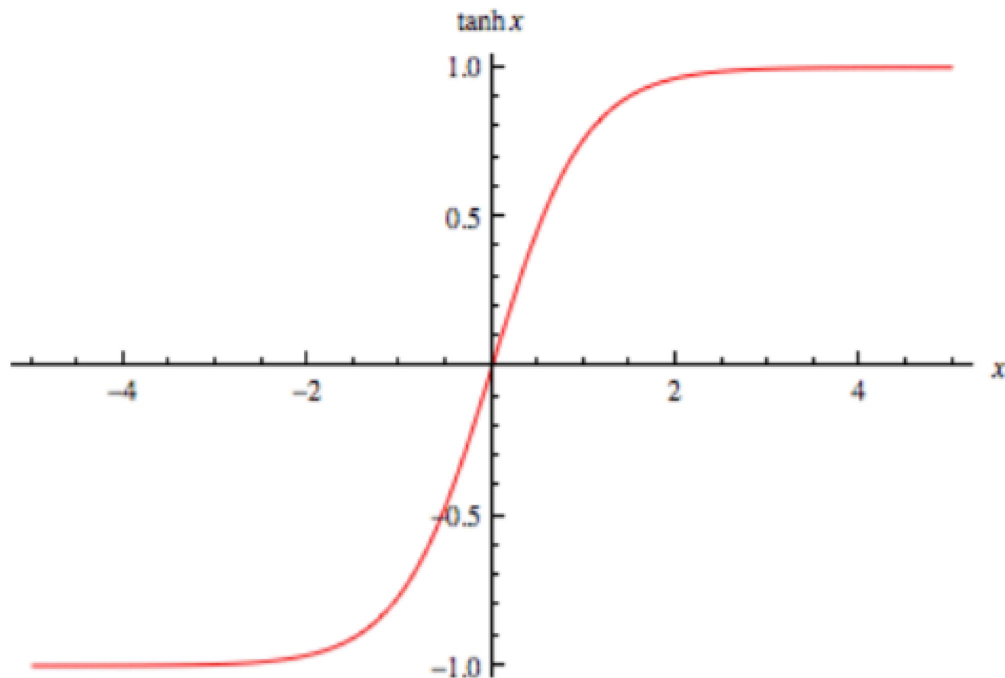


Figure MM: Graphe de la fonction tangente hyperbolique

2.2.2.2.3 Fonction ReLu

ReLu est l'acronyme d'un terme anglicisme Rectified Linear Unit ou en français Unité Linéaire Rectifiée. Elle désigne une fonction mathématique définie par f(x) = max(0, x) ∀ x. On peut également l'écrire comme ci-dessous :

$$\text{ReLU}(z) = \begin{Bmatrix} 0 & \text{si} & z < 0 \\ z & \text{si} & z \geq 0 \end{Bmatrix}$$

Le graphe de la figure 2.25 représentatif de cette fonction est simple à dessiner.

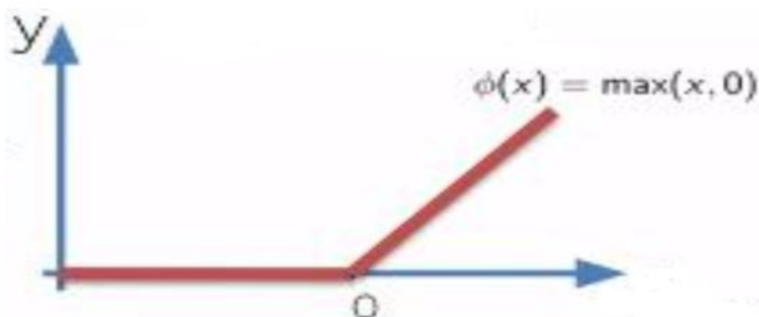


Figure NN: Graphe de la fonction ReLu

Selon l'expérience d'un certain Simon Vézina, dans sa note de cours intitulé 'Le réseau de neurones du chapitre 6.2' évoque le fait que l'usage de la fonction Relu nécessite la méthode softmax (que nous allons voir juste après), à la dernière couche du réseau. Car avançant que les agrégations générées seront trop élevées et impossibles à ajuster avec une fonction d'activation de classification comme SIGMOÏDE, car sa pente est trop faible à haute valeur d'agrégation.
Aussi dans sa logique établis que la fonction ReLU est réputée pour bien propager l'erreur en raison de sa pente de 1. Et que, si son activation est souvent négative, peu importe le vecteur d'entrée, le neurone sera alors « désactivé » ou « mort ». Ainsi, il y aura moins de neurones pour mettre à jour le réseau. Alors il serait souhaitable d'utiliser un grand nombre de neurones dans une couche avec la fonction d'activation ReLU.

2.2.2.2.4 Softmax

Comme vous pouvez vous en rendre compte à la figure 2.22, la fonction softmax est très souvent utilisée à la dernière couche du réseau afin d'effectuer une classification.
Car les résultats en sortie qu'elle fournit sont représentés sous forme de probabilité dont la somme est égale à 1. Ainsi on a tendance à voir des prédictions qui indiquent que tel objet d'une image ressemble à telle chose à un pourcentage donné dans le cadre d'un traitement d'image.
L'expression littérale de sa fonction s'écrit comme suit :

$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^{K} \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

Avec $j \in [1,....,K]$

2.2.2.3 Pooling

A l'issue de la convolution de la matrice représentative de l'image globale avec la matrice caractéristique d'un élément de l'objet à identifier, on obtient une nouvelle matrice. Le Pooling consiste alors à retirer les informations ne contenant pas de caractéristiques dans la matrice résultante. On observe ainsi une diminution des pixels. Cela a pour effet la réduction du risque de surapprentissage.

2.2.2.4 Flatten

Le Flattenning est une méthode qui consiste à aplatir la matrice obtenue après le Pooling, dans le but de placer les données à l'entrée du futur réseau pour la classification comme montrée à la figure 2.22 précédente.

2.2.2.5 Couche complètement connectée(fully connected)

Cette couche permet de combiner les caractéristiques dans la couche d'entrée, obtenues par flattening pour l'amélioration de la prédiction.

2.2.3 Les réseaux de neurones récurrents

La mémoire humaine essaie de comprendre son environnement en fonction des historiques de la vision. Les réseaux de neurones récurrents sont basés sur le même principe. Ce sont des réseaux avec des boucles, permettant aux informations de persister. Dans leurs procédés, les réseaux de neurones récurrents gardent en mémoire (au niveau des neurones) une information qui sera réinjectée à l'entrée. Ainsi lors de la prochaine itération, la donnée nouvellement chargée et la donnée sauvegardée permettront de fournir en sortie un résultat exact ou assez proche à prédire.
Mais ce modèle est confronté à un problème de dissipation du gradient. C'est-à-dire pour « apprendre », un RNN utilise la méthode de la descente du gradient afin de mettre à jour les poids entre ses neurones [16]. Ce principe est basé sur la formule ci-après :

$$w := w - \alpha \cdot F_w$$

avec :

- w : un poids du réseau
- α : la vitesse d'apprentissage du réseau
- Fw : le gradient du réseau par rapport au poids w

En réalité, la mise à jour des poids se fait de droite à gauche. A mesure que l'on avance vers la gauche, le produit $\alpha \cdot Fw$ devient très petit et les poids des premières couches de neurones ne sont quasiment pas modifiés. Ainsi, ces couches n'apprennent strictement rien. Et par conséquent, le RNN peut facilement oublier des données un petit peu anciennes (ou des mots assez éloignés du mot courant dans un texte) lors de la phase d'apprentissage : sa mémoire est courte [16].
Alors Sepp Hochreiter et Jürgen Schmidhuber ont proposé en 1997 un réseau Long short-term memory (LSTM), en français réseau récurrent à mémoire court et long terme ou plus explicitement réseau de neurones récurrents à mémoire court-terme et long terme afin de palier au problème de descente de gradient [17]. Ce modèle est composé de zones de calculs qui régulent le flot d'informations en réalisant des actions spécifiques (figure 2.26).
Tout ce qui suit dans cette section est tiré de la référence [16], car jugeant assez explicite pour bien comprendre ce modèle LSTM.

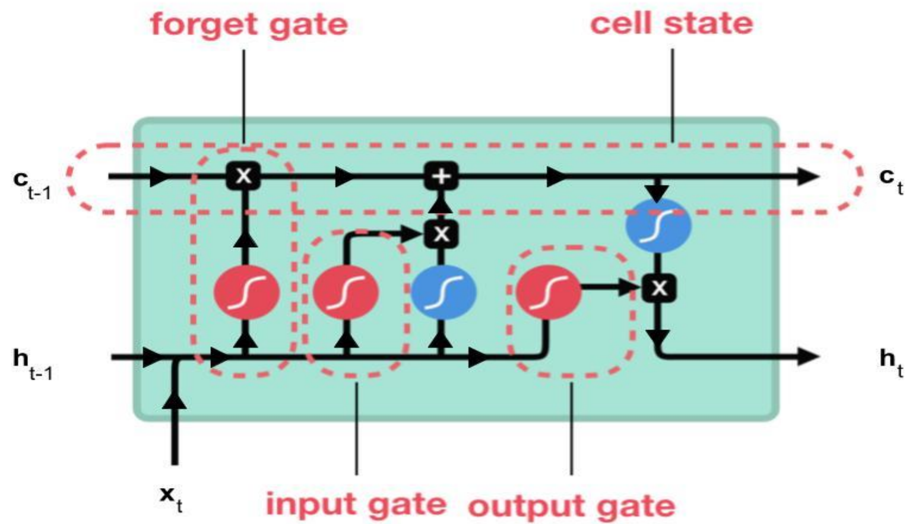


Figure OO: Structure d'un réseau LSTM

Tiré de [16]

- Forget gate (porte d'oubli)
- Input gate (porte d'entrée)
- Output gate (porte de sortie)
- Hidden state (état caché)
- Cell state (état de la cellule)

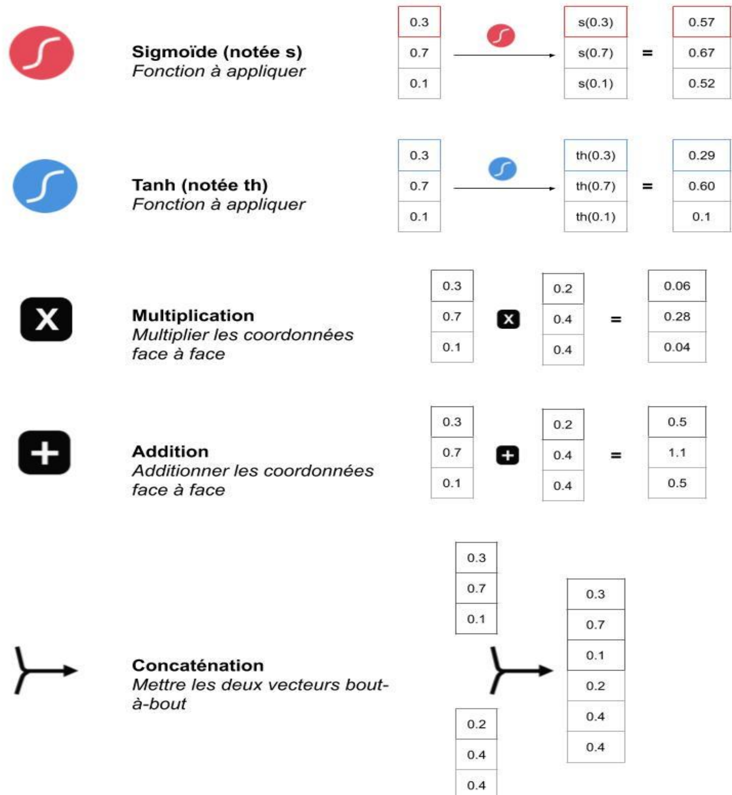


Figure PP: Opérateurs de la structure du réseau LSTM

Le LSTM effectue des opérations internes. Ces dernières (figure 2.27) permettent au LSTM de conserver ou supprimer des informations qu'il a en mémoire. Les données stockées dans la mémoire du réseau sont en fait un vecteur noté ct : l'état de la cellule. Comme cet état dépend de l'état précédent ct−1, qui lui-même dépend d'états encore précédents, le réseau peut conserver des informations qu'il a vues longtemps auparavant (contrairement au RNN classique).
Pour l'apprentissage, le réseau LSTM prend en compte certaines variables internes. Les entrées de chaque porte sont pondérées par des poids liés aux portes (figure 2.28) ainsi que par un biais. On a 4 portes :

- Wf : pondère l'entrée de la porte d'oubli (forget gate)
- Wi : pondère l'entrée de la porte d'entrée (input gate)
- WC : pondère les données qui vont se combiner à la porte d'entrée pour mettre à jour l'état de la cellule (cell state)
- Wo : pondère l'entrée de la porte de sortie (output gate)

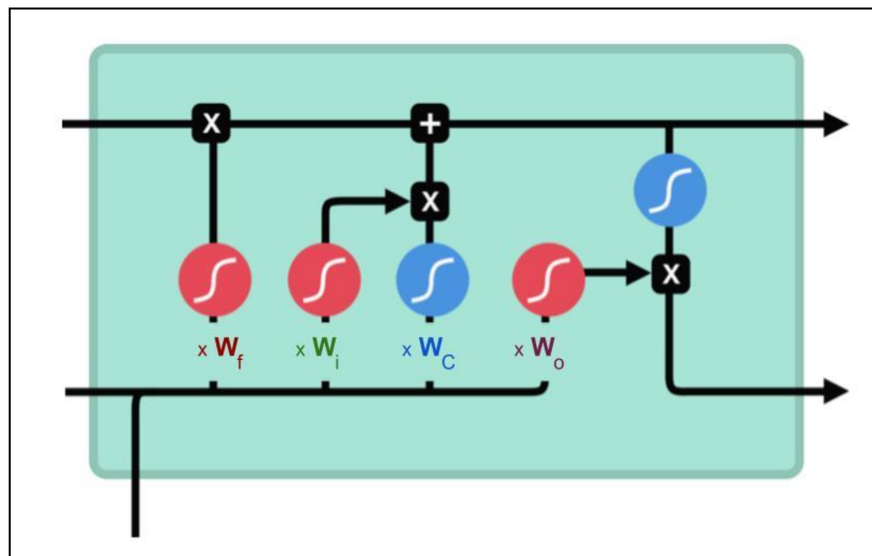


Figure 2.28 : Application des poids à un réseau LSTM

Tiré de [16]

- Porte d'oubli (forget gate)

Cette porte décide de quelle information doit être conservée ou jetée : l'information de l'état caché précédent est concaténée à la donnée en entrée (par exemple le mot « des » vectorisé) puis on y applique la fonction sigmoïde afin de normaliser les valeurs entre 0 et 1. Si la sortie de la sigmoïde est proche de 0, cela signifie que l'on doit oublier l'information et si l'on est proche de 1 alors il faut la mémoriser pour la suite.

- Porte d'entrée (input gate)

La porte d'entrée a pour rôle d'extraire l'information de la donnée courante (le mot « des » par exemple) : on va appliquer en parallèle une sigmoïde aux deux données concaténées et une tanh. Sigmoïde va renvoyer un vecteur pour lequel une coordonnée proche de 0 signifie que la coordonnée en position équivalente dans le vecteur concaténé n'est pas importante. A l'inverse, une coordonnée proche de 1 sera jugée « importante » (exemple utile pour la prédiction que cherche à faire le LSTM).

Tanh va simplement normaliser les valeurs (les écraser) entre -1 et 1 pour éviter les problèmes de surcharge de l'ordinateur en calculs. Le produit des deux permettra donc de ne garder que les informations importantes, les autres étant quasiment remplacées par 0.

• Etat de la cellule (cell state)
On parle de l'état de la cellule avant d'aborder la dernière porte (porte de sortie), car la valeur calculée ici est utilisée dedans. L'état de la cellule se calcule assez simplement à partir de la porte d'oublie et de la porte d'entrée : d'abord, on multiplie coordonnées à coordonnées la sortie de l'oublie avec l'ancien état de la cellule. Cela permet d'oublier certaines informations de l'état précédent qui ne servent pas pour la nouvelle prédiction à faire. Ensuite, on additionne le tout (coordonnées à coordonnées) avec la sortie de la porte d'entrée, ce qui permet d'enregistrer dans l'état de la cellule ce que le LSTM (parmi les entrées et l'état caché précédent) a jugé pertinent.

• Porte de sortie (output gate)
Dernière étape : la porte de sortie, doit décider de quel sera le prochain état caché, qui contient des informations sur les entrées précédentes du réseau et sert aux prédictions.
Pour ce faire, le nouvel état de la cellule calculée juste avant est normalisé entre -1 et 1 grâce à tanh. Le vecteur concaténé de l'entrée courante avec l'état caché précédent passe, pour sa part, dans une fonction sigmoïde dont le but est de décider des informations à conserver (proche de 0 signifie que l'on oublie, et proche de 1 que l'on va conserver cette coordonnée de l'état de la cellule).
Tout cela peut sembler magique en ce sens où on dirait que le réseau doit deviner ce qu'il doit retenir dans un vecteur à la volée, mais rappelons bien qu'une matrice de poids est appliquée en entrée. C'est cette matrice qui va, concrètement, stocker le fait que telle information est importante ou non à partir des milliers d'exemples qu'aura vus le réseau.

2.3 Conclusion

Dans ce chapitre, nous avons essayé d'expliquer autant que possible les expressions se rapportant à la vision par ordinateur et des réseaux de neurones artificiels. Nous retenons en particulier des étapes incontournables dans le procédé, comme la segmentation et le filtrage dans le cas de la vision par ordinateur, mais que les méthodes à utiliser diffèrent selon le cas d'application.
Des méthodes plus ou moins complexes sont également définies dans le cadre des réseaux de neurones artificiels, et feront l'objet du chapitre 3 sur notre travail à réaliser.

CHAPITRE 3

METHODOLOGIE

Ce chapitre permet d'expliquer les démarches utilisées dans notre processus de détection, de comptage et d'évaluation à temps réel du système proposé.
Nous passerons brièvement sur les paramètres étudiés comme éléments caractéristiques d'une graine du système déjà développé. Ensuite, nous indiquerons l'intérêt de la détection automatique et les méthodes illustratives. Nous présenterons également des bouts de code qui nous ont permis de faire le comptage des graines en nous basant sur la hiérarchie des éléments de l'image du plateau ayant des graines en premier lieu, et en second lieu en plus de la hiérarchie, la surface moyenne de la graine. Enfin, nous terminerons par les méthodes proposées pour l'évaluation à temps réel des performances d'ensemencement de notre système.

3.1 Paramètres étudiés du système déjà développé

Le système développé au préalable se basait sur des fonctions existantes sur python afin d'extraire des mesures caractéristiques d'un contour de graine. Toutes ces fonctions appartiennent au module cv2 qu'il faudrait installer par la commande ***pip install opencv-python*** correspondant au package des modules principaux.

3.1.1 L'aire

L'aire d'un contour d'une graine bien entendu, est une des mesures calculées. Elle est implémentée par la fonction ***contourArea()*** sous python.

3.1.2 Le périmètre

Le périmètre des contours d'intérêts a été utilisé. La fonction pour calculer le périmètre sur python est ***arcLenght()***.

3.1.3 Le facteur de forme

Sous python il n'existe pas de fonction typique pour calculer le facteur de forme. L'auteur, qui a conçu le programme, s'était basé sur une formule mathématique du genre : $4 * \pi * (area/(perimeter^2))$. Où ***area*** correspond au résultat obtenu par le calcul de l'aire illustré à la section 3-1-1), et ***perimeter***, le périmètre calculé à la section 3-1-2).

3.1.4 La solidité

La solidité selon la méthode avancée par son auteur est le rapport entre l'aire d'un contour et l'aire des défauts de convexités. La convexité est obtenue par la fonction **convexHull()** d'opencv. Ensuite, on applique la fonction ***contourArea()*** au résultat de la fonction convexHull() précédente. Et enfin, le résultat de l'aire calculée à la section 3-1-1) est divisé par le résultat de l'aire des défauts de convexités.

3.1.5 L’excentricité

L’excentricité fait intervenir quelques aspects dans son calcul. Les suites d’instructions sont présentées ci-après :

> ***(x,y) = cv2.fitEllipse(contours[0])[0]***
> ***center = (int(x),int(y))***
> ***(axes) = cv2.fitEllipse(contours[0])[1]***
> ***majorAxis = max(axes)*2***
> ***minorAxis = min(axes)*2***
> ***excentricity = np.sqrt(1-(minorAxis/majorAxis)**2)***

La fonction ***fitEllipse()*** a comme effet d’adapter une ellipse à l’objet d’intérêt, ici le contour d’une graine. Le résultat renvoyé est une coordonnée. Dans cette dernière, on calcule le double de la valeur maximale et minimale. Les résultats ainsi obtenus sont ensuite utilisés dans le calcul de l’excentricité comme le montre la formule. Le paramètre ***np*** dans la formule est un alias du module numpy (***import numpy as np***).

3.2 Détection Automatique

La détection automatique est l’étape préliminaire dans toutes procédures en traitement d’image. Elle consiste à charger, l’image ou le flux d’images en me référant à des séquences vidéo, afin de mener des traitements. Il existe plusieurs méthodes compte tenu de la plateforme utilisée. Dans notre cas, nous avons travaillé sur la **version 3.8.3** de **python**. Dans cette étape nous avons également fait usage de la bibliothèque ***opencv*** (pour open computer vision) ***version 4.4.0***, qui est très utilisée en traitement d’images en temps réel.
La bibliothèque opencv comporte plusieurs méthodes qui interviennent dans chaque phase de la détection automatique. Les lignes de codes ci-après montrent l’ensemble des méthodes dont on a fait appel pour assurer la détection automatique :

```
1  import cv2
2  import numpy as np
3  import os, os.path
4  import glob
5
6  # Load the image
7
8  im = cv2.imread("CabaretEPN.jpg")       #image originale
9  im_next = cv2.imread("CabaretEPN.jpg")
10 imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)    #image à niveau de gris
11
12 #cv2.imshow('image a niveau de gris',imgray)
13 #cv2.waitKey(0)
14
15 ret, thresh = cv2.threshold(imgray, 0.0, 255.0, cv2.THRESH_OTSU) # seuillage automatique avec la méthode d'otsu
16
17 #cv2.imshow('image noire blanc avec seuil d\'otsu ', thresh) #affichage de l'image noire blanc obtenue précédemment
18 #print('seuil calculé : ' + str(ret)) #seuil obtenu automatiquement
19 #cv2.waitKey(0)
20 #cv2.imwrite('bw_'+'CabaretEPN', thresh)
21 #imbw = cv2.imread("bw_CabaretEPN.png")
22 #cv2.imshow('image noire blanc',imbw)
23 #cv2.waitKey(0)
24
25 #Détection de contours avec findcontours
26 contoures, hierarchiy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # CHAIN_APPROX_NONE ou CHAIN_AI
27                                                                                         # ou quelques points(SIMPLE) c
28 cv2.drawContours(im, contoures, -1, (0,255,0), 1)    #   0,255,0 --> 0 = blue , 255 = green , 0 = red ; c'est pourquoi le
29 cv2.imwrite('cnt_'+'CabaretEPN.png', im)   #enregistrement de l'image avec les contours détectés
30 imcnt = cv2.imread("cnt_CabaretEPN.png")   #chargement de l'image avec les contours détectés
31 cv2.imshow('draw contour on image',imcnt)   #affichage de l'image avec les contours détectés
32 cv2.waitKey(0) #permet d'observer une pause et le clique sur n'import quelle touche du clavier fait que l'on passe aux :
```

Des commentaires ont été ajoutés comme vous pouvez le constater, afin de veiller à la clarté du code.
Parmi les modules utilisés dans cette partie, nous avons parlé d'***opencv*** (cv2). Le module ***numpy*** dont on a introduit à la section 3-1-5) est une bibliothèque qui permet de faire des opérations mathématiques sur des données multidimensionnelles. Les modules ***os*** et ***glob*** permettent respectivement d'accéder au système de fichiers et de rechercher des fichiers contenant certaines extensions dans un système d'exploitation.

Cependant, nous procédons au chargement de l'image à traiter avec la méthode ***imread*** d'opencv. Ensuite, l'image est convertie en niveau de gris par la méthode cvtColor d'opencv toujours, avec le paramètre COLOR_BGR2GRAY en plus de l'objet contenant l'image originale à traiter. Le produit de cette mise en niveau de gris est ensuite affecté à la méthode ***threshold*** dans le but de convertir l'image précisée en noire et blanche avec la méthode d'Otsu. Cette dernière se base sur le seuillage global, qui consiste à partitionner l'image en deux classes grâce à un seuil optimal calculé à partir d'une mesure globale sur toute l'image. En d'autres termes, le principe de la méthode de Otsu est de trouver un seuil optimal qui maximise la différence entre deux classes en se focalisant sur quelques notions élaborées à la section2-1-1).
Par suite, on passe à la détection des contours par la méthode ***findContours*** d'opencv. Cette méthode prend en paramètre l'image binarisée, et également les éléments de contours à détecter cv2.RETR_TREE qui sert à détecter tous les contours et ensuite les classés dans une liste, ou juste les contours externes par cv2.RETR_EXTERNAL.
Le troisième paramètre de notre méthode findContours qui est cv2.CHAIN_APPROX_SIMPLE comme commenté dans le code, permet de préciser si l'on prend tous les points constituant un contour (cv2.CHAIN_APPROX_NONE) ou si l'on prendra quelques points pour représenter le contour en question.
Une dernière méthode parmi celles que l'on n'a pas encore vues dans cette partie ***drawContours***. Cette méthode sert à dessiner les contours détectés précédemment. Ses paramètres :
im : l'image originale chargée
contoures : les coordonnées des contours détectés par la méthode findContours
-1 : indique que tous les contours détectés seront dessinés
(0, 255, 0) : précise la couleur des contours à dessiner, vert dans notre cas
1 : fais référence à l'épaisseur du tracé sur les contours.

3.3 Comptage de graines de semences de résineux

Selon le contexte de notre projet, une fois que nous avons effectué la détection automatique des éléments caractéristiques de notre image, on procède au décompte des régions d'intérêts extraites correspondant aux graines. L'image de chaque cellule avec des graines est ensuite sauvegardée dans un dossier en fonction du nombre de graines trouvées. Cette section présente les techniques utilisées à cet effet.

3.3.1 Méthode basée sur la hiérarchie

Nous partons du programme développé, qui est la suite du code présenté à la section 3-2).

```
33
34  #print('hierarchiy : ' + str(hierarchiy))
35  # Iterate all blobs
36
37  #a) avant cela (Iterate all blobs) on crée un dossier third_next_with_draw_contours_plus_newf
38
39  try:
40
41      rep = os.getcwd()
42      newfolder = 'third_next_with_draw_contours_plus_newfolder'
43
44      if not os.path.exists(newfolder):
45          os.mkdir(newfolder)
46      else:
47          print("le répertoire : " + rep + "/" + newfolder + " existe déjà")
48
49  #b) dans ce dossier third_next_with_draw_contours_plus_newfolder, créer des dossiers 0_graine
50      rep_folders = rep + "\\" + newfolder
51      zero_graine = rep_folders + "/0-graine"
52      une_graine = rep_folders + "/1-graine"
53      deux_graine = rep_folders + "/2-graine"
54      trois_graine = rep_folders + "/3-graine"
55      quatre_graine = rep_folders + "/4-graine"
56      cinq_graine = rep_folders + "/5-graine"
57      six_graine = rep_folders + "/6-graine"
58      plus_de_six_graine = rep_folders + "/sup-6-graine"
59
60      if not os.path.exists(zero_graine):
61          os.mkdir(zero_graine)
62      else:
63          print("le répertoire : " + zero_graine + " existe déjà")
64
65      if not os.path.exists(une_graine):
66          os.mkdir(une_graine)
67      else:
68          print("le répertoire : " + une_graine + " existe déjà")
69
70      if not os.path.exists(deux_graine):
71          os.mkdir(deux_graine)
72      else:
73          print("le répertoire : " + deux_graine + " existe déjà")
74
75      if not os.path.exists(trois_graine):
76          os.mkdir(trois_graine)
77      else:
78          print("le répertoire : " + trois_graine + " existe déjà")
```

```
79
80      if not os.path.exists(quatre_graine):
81          os.mkdir(quatre_graine)
82      else:
83          print("le répertoire : " + quatre_graine + " existe déjà")
84
85      if not os.path.exists(cinq_graine):
86          os.mkdir(cinq_graine)
87      else:
88          print("le répertoire : " + cinq_graine + " existe déjà")
89
90      if not os.path.exists(six_graine):
91          os.mkdir(six_graine)
92      else:
93          print("le répertoire : " + six_graine + " existe déjà")
94      if not os.path.exists(plus_de_six_graine):
95          os.mkdir(plus_de_six_graine)
96      else:
97          print("le répertoire : " + plus_de_six_graine + " existe déjà")
98
99  except:
100     OSError
101
102
103 #im2 = cv2.imread("CabaretEPN.jpg")
104 areaArray = [] #tableau pour contenir l'aire des cellules contenant des graines
105 globalArea = [] #tableau pour contenir l'aire de tous les contours détecter
106 iArray = [] #tableau recueillant l'indice de chaque contour de cellule contenant des graines
107 for i, c in enumerate(contoures): # c contiendra les coordonnées des contours et i l'indice couplé à la p
108     #print('i : ' + str(i))
109     #print('c : ' + str(c))
110     area = cv2.contourArea(c) # calcul de l'aire du contour c
111     globalArea.append(area) # ajout de l'aire dans le tableau
112     if area >= 25162.5: #--->faire si area est supérieur ou égale à une valeur moyenne(de l'aire d'une ce
113                         # la valeur est obtenue en calculant (avec contourArea) l'aire d'un contour exter
114         areaArray.append(area)
115         iArray.append(i)
116     else:
117         #print("l'aire est plus petit que 25162.5 et est égale à: " + str(area))
118         continue
119
120 #print("liste des aires: " + str(areaArray))
121 print("liste des indices i : " + str(iArray))
122 print("nombre d'éléments dans la liste: " + str((len(iArray))))
123 # Sort countours based on area
124 sorteddata = sorted(zip(globalArea, contoures), key=lambda x: x[0], reverse=True)#https://www.w3schools.c
125
```

```
126 # find the nth largest contour [n-1][1], in this case 2
127 for j in range(len(iArray)):
128     print('valeur indice : ' + str(j))
129     largestcontour = sorteddata[j][1]
130     # get the bounding rectangle of the contour
131     x, y, w, h = cv2.boundingRect(largestcontour)
132
133     cropped_img = im_next[y+3:y+h-3,x+3:x+w-3]
134     #cv2.imshow("cropped", cropped_img)
135     #cv2.waitKey(0)
136     cv2.imwrite(rep_folders + '/' + 'cellule_'+ str(j + 1) +'roi.png', cropped_img)
137
138 imageNames = [imagename for imagename in glob.glob(rep_folders + '/*.png')]
139 imageSorted = sorted(imageNames)
140 for imageCropped in imageSorted:
141     print('chemin d\'accès à l\'image: ' + imageCropped)
142     replaceimageCropped = imageCropped.replace("\\", "/")
143     print('nouveau chemin d\'accès à l\'image: ' + replaceimageCropped)
144     im2 = cv2.imread(replaceimageCropped)
145     #cv2.imshow('image cropped',im2)   #affichage de l'image avec les contours détectés
146     #cv2.waitKey(0)
147     imitergray = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)
148
149     retiter, threshiter = cv2.threshold(imitergray, 0.0, 255.0, cv2.THRESH_OTSU) # seuillage automatique avec la méthode d'otsu
150
151     #cv2.imshow('image noire blanc avec seuil d\'otsu ', threshiter) #affichage de l'image noire blanc obtenue précédemment
152     print('seuil calculé : ' + str(retiter)) #seuil obtenu automatiquement
153     #cv2.waitKey(0)
154     contours, hierarchy = cv2.findContours(threshiter, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
155
156     blob_idx = np.squeeze(np.where(hierarchy[0, :, 3] == -1))
157
158     for b_idx in np.nditer(blob_idx):
159
160         # Add outer contour of blob to list
161
162         #print('b_idx : ' + str(b_idx))
163         #print('contoures[b_idx]: ' + str(contoures[b_idx]))
164         blob_cnts = [contours[b_idx]]
165         #print('blob_cnts' + str(blob_cnts))
166
167         # Add inner contours of blob to list, if present
168
169         #print(hierarchiy[0, :, 3] == b_idx)
170         cnt_idx = np.squeeze(np.where(hierarchy[0, :, 3] == b_idx))    #recupèration des contours interne de chaque blob(type d
171                                                                         #des données numériques au format binaires(images, vidé
172         print('cnt_idx.size : ' + str(cnt_idx.size))
```

```
173         #print('hierarchiy.shape' + str(hierarchiy.shape))
174
175         if (cnt_idx.size > 0):
176             blob_cnts.extend([contours[c_idx] for c_idx in np.nditer(cnt_idx)])    #consiste à ajouter les contours des fils (graines)
177
178     #-------------------------------------------------------------------------------------nombre de graine---------------------------
179                 #on parcourt chaque ligne de hierarchiy
180             i = 0
181
182             for j in (hierarchy[0, :, 3] == b_idx):
183                 if j == True:
184                     i = i + 1
185                 else:
186                     i
187                 # après avoir parcouru l'ensemble des lignes dans hierarchy, le nombre de valeur(s) dont parent est différent de -1 co
188             print("le nombre de graine(s) dans la cellule est : " + str(i))
189
190     #------------------------------------------------------------------------------------- fin nombre de graine----------------------
191
192             chaine = replaceimageCropped
193             pos1 = chaine.find('cel')
194             pos2 = chaine.find('.png')
195             pos2=pos2+len('.png')
196             sousChaine = chaine[pos1:pos2]
197             print(sousChaine)
198             if i == 0 :
199
200                 cv2.imwrite(zero_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)      # ('chemin d'acces /nom_image.png', im
201
202             elif i == 1 :
203
204                 cv2.imwrite(une_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)
205             elif i == 2 :
206
207                 cv2.imwrite(deux_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)
208             elif i == 3 :
209
210                 cv2.imwrite(trois_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)
211             elif i == 4 :
212
213                 cv2.imwrite(quatre_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)
214             elif i == 5 :
215
216                 cv2.imwrite(cinq_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)
217             elif i == 6 :
218
219                 cv2.imwrite(six_graine + '/' + str(i) + 'graine_' + str(sousChaine), im2)
```

Le comptage de graines dans chaque cellule est essentiellement basé sur la hiérarchie des contours. Cette hiérarchie évalue la relation parent-enfant des différents contours détectés comme indiqué dans la figure 3.1.

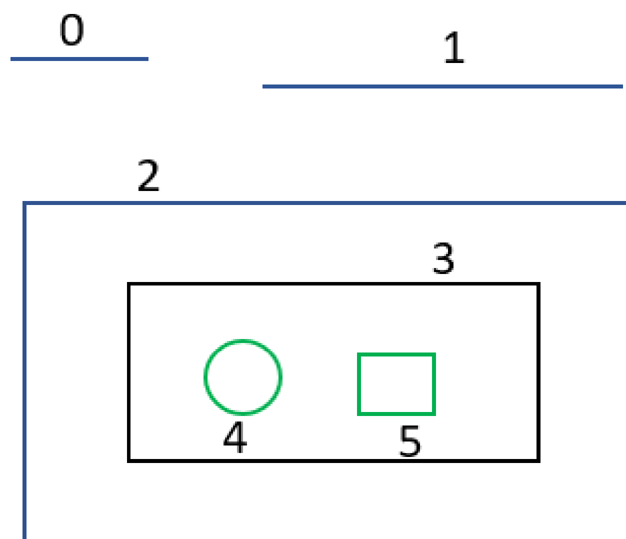


Figure QQ: Niveaux hiérarchiques de quelques contours

Nous avons numéroté quelques formes de 0 à 5. Alors les contours 0, 1 et 2 sont considérés comme externes, ainsi ils ont le même niveau d'hiérarchie. Le contour 3

est l'enfant du contour 2, ou inversement le contour 2 est le parent du contour 3 et ce dernier est seul dans ce niveau d'hiérarchie. Les contours 4 et 5 sont les enfants du contour 3 et sont dans le dernier niveau d'hiérarchie. En parallèle, le contour 4 est le premier enfant du contour 3 selon notre notation, mais cela peut être le contour 5. Nous verrons plus en détail dans le chapitre suivant les résultats sur une image.

En outre, chaque contour comporte ses propres descriptifs. Ces derniers réfèrent à son niveau d'hiérarchie, son parent et son enfant s'il en a, ainsi que le numéro de contour suivant parmi ceux détectés s'il en existe. C'est cette séquence qui est renvoyée par la variable ***hierarchiy*** à la ***ligne 26*** de notre bout de code de la section 3-2) précédente suivant un certain ordre.
L'ordre est présenté comme suit : ***[Next, Previous, First_Child, Parent]***.

Next : numéro d'hiérarchie du contour suivant
Previous : numéro d'hiérarchie du contour précédent
First_Child : numéro d'hiérarchie du premier enfant
Parent : numéro d'hiérarchie du parent du contour en question

L'absence d'un élément parmi ces quatre paramètres est matérialisée par le chiffre ***-1*** à la place.
Le décompte du nombre de graines dans chaque cellule détectée correspond exactement aux lignes de codes allant de la ligne 178 à 190 comme illustré dans cette section.

3.3.2 Méthode basée sur la hiérarchie et la surface moyenne de la graine

L'usage de la hiérarchie est au cœur de nos programmes dans le cadre du comptage du nombre de graines. Comme illustré dans la section précédente, elle nous permet d'accéder à l'élément d'intérêt (la graine).
Une fois la hiérarchie connue, on évalue la surface concernée. Le programme faisant recours à ce traitement a été développé en parallèle suite à de nouvels ajouts d'images de plateaux avec ou sans graines nous inspirant à fonctionner différemment.
Le code est identique au programme présenté jusque là (section 3-3-1) avec des modifications apportées à partir de la ligne 122 comme suit :

```
120    #print("liste des aires: " + str(areaArray))
121    print("liste des indices i : " + str(iArray))
122    print("nombre d'éléments dans la liste: " + str((len(iArray))))
123    # Sort countours based on area
124    #sorteddata = sorted(zip(globalArea, contoures), key=lambda x: x[0], reverse=True)#https://www.
125
126    # find the nth largest contour [n-1][1], in this case 2
127    r = 0
128    for j in iArray:
129        print('valeur indice : ' + str(j))
130        r = r + 1
131        largestcontour = contoures[j]
132        # get the bounding rectangle of the contour
133        x, y, w, h = cv2.boundingRect(largestcontour)
134
135        cropped_img = im_next[y+3:y+h-3,x+3:x+w-3]
136        #cv2.imshow("cropped", cropped_img)
137        #cv2.waitKey(0)
138        print('valeur de r : ' + str(r))
139        cv2.imwrite(rep_folders + '/' + 'cellule_'+ str(r) +'roi.png', cropped_img)
140
141    imageNames = [imagename for imagename in glob.glob(rep_folders + '/*.png')]
142    imageSorted = sorted(imageNames)
```

```
140
141 imageNames = [imagename for imagename in glob.glob(rep_folders + '/*.png')]
142 imageSorted = sorted(imageNames)
143 for imageCropped in imageSorted:
144     print('chemin d\'accès à l\'image: ' + imageCropped)
145     replaceimageCropped = imageCropped.replace("\\", "/")
146     print('nouveau chemin d\'accès à l\'image: ' + replaceimageCropped)
147     im2 = cv2.imread(replaceimageCropped)
148     #cv2.imshow('image cropped',im2)   #affichage de l'image avec les contours détectés
149     #cv2.waitKey(0)
150     imitergray = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)
151
152     retiter, threshiter = cv2.threshold(imitergray, 0.0, 255.0, cv2.THRESH_OTSU) # seuillage automatique avec la m
153
154     #cv2.imshow('image noire blanc avec seuil d\'otsu ', threshiter) #affichage de l'image noire blanc obtenue pré
155     print('seuil calculé : ' + str(retiter)) #seuil obtenu automatiquement
156     #cv2.waitKey(0)
157     contours, hierarchy = cv2.findContours(threshiter, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
158
159     areaParent = [] #tableau pour contenir l'aire de la cellule du parent
160     #areaGraine = [] #tableau pour contenir l'aire de la surface des graines
161     areaOther = [] #tableau pour contenir l'aire des autres surfaces
162     global_Area = [] #tableau pour contenir l'aire de tous les contours détecter
163     iArray_Parent = [] #tableau recueillant l'indice du contour parent (un seul indice normalement)
164     #iArray_graine = [] #tableau recueillant l'indice du contour de la graine
165     iArray_Other = [] #tableau recueillant l'indice des autres contours
166
167     for s, t in enumerate(contours): # t contiendra les coordonnées des contours et s l'indice couplé à la positio
168         #print('s : ' + str(s))
169         #print('t : ' + str(t))
170         area_ = cv2.contourArea(t) # calcul de l'aire du contour t
171         global_Area.append(area_) # ajout de l'aire dans le tableau
172         if area_ >= 25162.5: #--->faire si area est supérieur ou égale à une valeur moyenne(de l'aire d'une cellul
173                              # la valeur est obtenue en calculant (avec contourArea) l'aire d'un contour externe d'
174             areaParent.append(area_)
175             iArray_Parent.append(s)
176         else:
177             areaOther.append(area_)
178             iArray_Other.append(s)
179     arrayIndice_graine = np.where(hierarchy[0,:,3] != -1) and np.where(hierarchy[0,:,3] == iArray_Parent)
```

```
180     #print(arrayIndice_graine)
181     for k in arrayIndice_graine:
182         #print(k)
183         v = 0
184         for l in k:
185             cnt = contours[l]
186             area_contour = cv2.contourArea(cnt)
187             val_moy_surface = 464.5
188
189             #val_moy_surface = 72.5  #val max EPB
190             #val_moy_surface = 464.5 #val max PIG
191             #val_moy_surface = 50.5  #val max EPN
192
193             #calcul_surface_graine()
194
195                     #def calcul_nombre_graine_contour(val_moy_surface, area_contour):
196             #v = 0
197             if area_contour < val_moy_surface and float(area_contour) < 454.5:
198                 nbr_graine = 0
199             elif area_contour > 454.5 and float(area_contour) < (2 * val_moy_surface):
200                 nbr_graine = 1
201             elif float(area_contour) >= (2 * val_moy_surface) and float(area_contour) < (3 * val_moy_surface):
202                 nbr_graine = 2
203             elif float(area_contour) >= (3 * val_moy_surface) and float(area_contour) < (4 * val_moy_surface):
204                 nbr_graine = 3
205             elif float(area_contour) >= (4 * val_moy_surface) and float(area_contour) < (5 * val_moy_surface):
206                 nbr_graine = 4
207             else:
208                 print('nombre de graines plus grand que 4')
209                 nbr_graine = 5
210
211             print('le nombre de graine(s) dans ce contour est : ' + str(nbr_graine))
212             v += nbr_graine
213         print('le nombre de graine(s) dans cette cellule est : ' + str(v))
214
215
216                                 #return som_area/sum(hierarchiy[0, :, 3] == b_idx), list_area_contour
217
218
219                               #calcul_surface_graine()
220
221
222
```

```
220
221
222
223        chaine = replaceimageCropped
224        pos1 = chaine.find('cel')
225        pos2 = chaine.find('.png')
226        pos2=pos2+len('.png')
227        sousChaine = chaine[pos1:pos2]
228            #print(sousChaine)
229            #------------------------------------------------------------------------------------- fin nombre
230
231                    #c) puis on sauvegarde une copie de l'image de la cellule concerné dans le dossier corres
232                    #N.B: se trouvant dans le if, donc l'image enregistrer ci après correspond bien à une cel
233
234        if v == 0 :
235
236            cv2.imwrite(zero_graine + '/' + str(v) + 'graine_' + str(sousChaine), im2)      # ('chemin d'acce
237
238        elif v == 1 :
239
240            cv2.imwrite(une_graine + '/' + str(v) + 'graine_' + str(sousChaine), im2)
241        elif v == 2 :
242
243            cv2.imwrite(deux_graine + '/' + str(v) + 'graine_' + str(sousChaine), im2)
244        elif v == 3 :
245
246            cv2.imwrite(trois_graine + '/' + str(v) + 'graine_' + str(sousChaine), im2)
247        elif v == 4 :
248
249            cv2.imwrite(quatre_graine + '/' + str(v) + 'graine_' + str(sousChaine), im2)
250        else:
251            print('impossible de sauvegarder l\'image de cette cellule, car nombre de graine inconnu')
252
```

A l'issu des modifications apportées, vous constaterez une façon plus simple d'extraire l'image des cellules contenant des graines avec la fonction ***boundingRect*** de la bibliothèque d'***openCV***. Chaque cellule extraite est ensuite analysée pour trouver le nombre exact de graines en se référant en plus de la hiérarchie, la surface moyenne d'une graine.
Ces graines sont de tailles différentes et appartiennent à trois familles que sont l'épinette noire (EPN), l'épinette blanche (EPB) et le pin gris (PIG).
Les images ajoutées sont des plateaux de chacun des types énumérés et contiennent respectivement 0, 1, 2, 3 et 4 graines. Ainsi dans ce nouveau programme nous nous sommes limités à la création des dossiers correspondants aux nombres de graines qui existent.
Les lignes de codes 189, 190 et 191 mises en commentaire dans cette section, font allusion à la surface moyenne calculée à l'aide d'un mini programme selon le type de la graine et dont on a présenté dans les sous sections ci-après.

3.3.2.1 EPN

Sachant que la taille des graines d'épinette noire est la plus petite parmi celles étudiées, cette taille reste aussi variable. Et cette variation est partie intégrante des problèmes de classification. Ainsi, parmi les cellules extraites, nous avons pris une image dont la cellule compte une graine et assez volumineuse à l'œil nu, afin de calculer sa surface. La valeur trouvée nous servira de référence dans notre programme principal.
Notre mini programme de calcul de la surface est présenté ci-dessous.

```
import cv2
import numpy as np


im = cv2.imread("1graine_cellule_3roi.png")
im2 = cv2.imread("1graine_cellule_3roi.png")
imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

ret, thresh = cv2.threshold(imgray, 0.0, 255.0, cv2.THRESH_OTSU)


contoures, hierarchiy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(im, contoures, -1, (0,255,0), 1)
print('hierarchie :' + str(hierarchiy))

cv2.imshow('image avec contour ', im)
cv2.waitKey(0)
arrayIndice_graine = np.where(hierarchiy[0,:,3] != -1)
print(arrayIndice_graine)
for j in arrayIndice_graine:
    #print(j)
    for k in j:
        #print(k)
        cnt = contoures[k]
        cv2.drawContours(im2, cnt, -1, (0,255,0), 1)
        cv2.imshow('image2 avec contour ', im2)
        cv2.waitKey(0)
        area = cv2.contourArea(cnt) #calcule de l'aire du contour spécifié en indice
        print('area EPN  = ' + str(area))

```

Après l'exécution de ce mini programme on obtient le résultat affiché à la figure 3.2.

```
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang\manuel> python.exe .\surface_detection_comptage_imbriquer.py
hierarchie :[[[-1 -1  1 -1]
  [-1 -1 -1  0]]]
area EPN  = 50.5
```

Figure RR: Surface moyenne graine EPN

Cette valeur calculée n'est pas universelle. Nous avons procédé de la sorte afin de palier à certaines contraintes et de pouvoir continuer dans notre logique.

3.3.2.2 EPB

La graine d'épinette blanche est de taille moyenne par rapport à la graine d'épinette noire et au pin gris. Le procédé pour calculer la surface de ce type de graine reste le même que celui adopté à la section 3-3-2-1.
Dans le mini programme, nous avons évidemment changé le nom de l'image ainsi que le message à afficher dans la console. La figure 3.3 donne la valeur de la surface obtenue.

```
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang\manuel> python.exe .\surface_detection_comptage_imbriquer.py
hierarchie :[[[-1 -1  1 -1]
  [-1 -1 -1  0]]]
area EPB  = 72.5
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang\manuel>
```

Figure SS: Surface moyenne graine EPB

Lors du traitement d'un plateau contenant des graines de cette sorte, nous mettrons la valeur de la surface moyenne à 72,5. Cet ajustement se fait à la ligne 187 de notre programme principal. Les lignes de code s'y rapportant seront également ajustées en conséquence.

3.3.2.3 PIG

Le dernier de notre liste des trois sortes de graine étudiées, le pin gris. La taille de ses graines est la plus grande de tous. Le choix de l'image de la graine qui nous servira de référence respecte les mêmes critères que ceux présentés dans nos deux sous-sections précédentes. Nous reviendrons avec leurs images à l'appui dans le chapitre 4 consacré à cela.
A l'issue, de l'exécution du mini programme avec les modifications idoines, nous obtenons la valeur indiquée à la figure 3.4.

```
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang\manuel> python.exe .\surface_detection_comptage_imbriquer.py
hierarchie :[[[-1 -1  1 -1]
  [ 2 -1 -1  0]
  [-1  1 -1  0]]]
area PIG  = 464.5
```

Figure TT: Surface moyenne graine PIG

Notons à ce niveau en plus de l'aire de la graine trouvée, la hiérarchie d'un supposé autre contour fils été détecté. Et le calcule de sa surface a donné 2.5 lors de nos tests. Cela signifie qu'un élément ne correspondant pas à une graine a été détecté. Ainsi le fait d'inclure la notion de la surface dans notre processus de comptage du nombre de graines résout les zones d'ombres sur des imperfections pouvant être dues au plateau.

3.4 Evaluation à temps réel des performances d’ensemencement

Le procédé d’évaluation des performances d’ensemencement que nous présenterons dans cette partie a pour but de prédire l’efficacité du système de détection automatique et de comptage de graines proposé. Pour ce faire, on a fait recours aux réseaux de neurones convolutifs selon le schéma de la figure 3.5, afin de donner une estimation des résultats de la classification faite sur le nombre de graines par cellule.

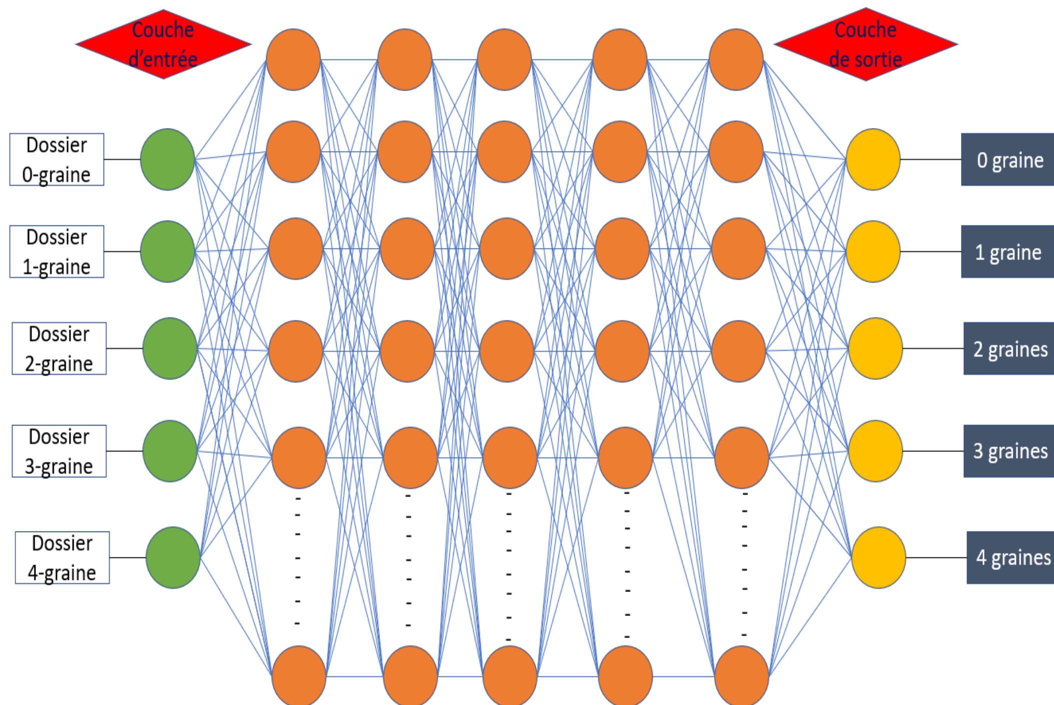


Figure UU: Réseaux de neurones convolutifs utilisés pour le comptage de graines

3.4.1 Evaluation selon la hiérarchie

Le modèle de réseaux de neurones conçu se base sur deux dossiers ***train_detection_et_comptage_de_graine*** et ***test_detection_et_comptage_de_graine*** qui contiennent chacun des dossiers ***0-graine***, ***1-graine***, ***2-graine***, ***3-graine***, ***4-graine***, ***5-graine***, ***6-graine*** et ***sup-6-graine*** pour l’entraînement et la validation. Ces dossiers, comme vous en douterez certainement, sont censés contenir des images de cellules ayant le nombre de graines équivalent au nom du dossier. La répartition des images dans les dossiers est faite via le programme présenté à la section 3-3-1).

Le programme que nous avons développé pour l’évaluation des performances de notre système de détection et de comptage de graines utilise les bibliothèques ***keras*** et ***tensorflow*** dans leur version présentée dans les figures 3.6 et 3.7.

```
>>> keras.__version__
'2.4.3'
```

Figure WW: Version

```
>>> tensorflow.__version__
'2.3.1'
```

Figure VV: Version tensorflow

Ces bibliothèques sont les plus utilisées en langage machine.
D’autres bibliothèques sont également utilisées et nous les présenterons dans la suite de cette partie.
Le programme est constitué de trois fonctions principales que sont :

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
import statistics
import cv2
import pandas as pd
import numpy as np
from numpy import mean
from numpy import std
from tensorflow.keras import regularizers

# dimensions of our images.
img_height, img_width = 200, 200

input_shape = (img_height, img_width, 3)

def evaluate_model():

    train_images_path = "C:/Users/USER/Python/projet_de_maitrise_diabang/train_detection_et_comptage_de_graine"
    test_images_path = "C:/Users/USER/Python/projet_de_maitrise_diabang/test_detection_et_comptage_de_graine"

    verbose, epochs, batch_size = 0, 25, 5

    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation="relu", input_shape = input_shape))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(64, (3, 3), activation="relu"))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
    model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())

    model.add(Dense(512, activation='relu', activity_regularizer=regularizers.l1(0.0001)))
    model.add(Dropout(0.5))
    model.add(Dense(8, activation='softmax')) #softmax #sigmoid

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    # this is the augmentation configuration we will use for training
    train_datagen = ImageDataGenerator(
        rescale=1. / 255,
        shear_range=0.2,
```

```
        zoom_range=0.2,
        horizontal_flip=True)

    # this is the augmentation configuration we will use for testing:
    # only rescaling
    test_datagen = ImageDataGenerator(rescale=1. / 255)

    train_generator = train_datagen.flow_from_directory(
        train_images_path,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='categorical')

    validation_generator = test_datagen.flow_from_directory(
        test_images_path,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical')

    model.fit(
        train_generator,
        steps_per_epoch=5,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=5)

    model.save_weights('poids_model.h5')
    _, accuracy = model.evaluate(validation_generator, steps=len(validation_generator),
    return accuracy
```

La fonction ***evaluate_model()*** où nous définissons les paramètres de notre modèle de réseaux de neurones afin d'y évaluer les performances. Elle nous retourne une valeur qui représente le pourcentage obtenu à l'issue de cette évaluation.
De la ligne 1 à la ligne 11, nous avons l'ensemble des bibliothèques et modules, dont certains ont été présentés dans les sections 2-2) et 3-2).
Notre réseau est constitué de 6 couches dont une couche d'entrée, quatre couches cachées et une couche de sortie. Les hyperparamètres utilisés sont définis ci-après afin de cerner leurs impacts sur les résultats obtenus.

- **Verbose** : permet de donner des informations sur la journalisation quand sa valeur est activée, c'est-à-dire >0.
- **Sequential** :c'est la méthode permettant d'instancier un réseau de neurones à une variable.
- **Filters** : permet d'organiser les données par lot pour la suite du traitement.
- **Kernel_size** : taille du noyau.
- **Dense** : est un paramètre caractéristique du modèle, et désigne une ou des couche(s) dense(s) entièrement connectée(s), utilisé pour interpréter les entités extraites par les couches cachées, avant qu'une couche de sortie finale ne soit utilisée pour faire des prédictions.
- **Categorical_crossentropy** : fonction de perte (calcul la différence entre deux distributions de probabilité).
- **Optimizer** : Le but de l'optimisation est de réduire l'erreur d'apprentissage
- **Epoch** : nombre de fois que le modèle voit l'ensemble des données.
- **Batch-size** : taille des sous-ensembles définis (nombre total d'exemples d'apprentissage dans l'ensemble de données disponibles).
- **dropOut** : fonction de désactivation par hasard d'une partie des neurones et de leurs connexions entrantes et sortantes pendant l'entrainement.
- **adam** : optimiseur pour réduire l'erreur d'apprentissage.
- **Loss** : permet de quantifier l'erreur en sortie sur la classification (doit être minimisé le plus possible).
- **Regularisation L1** = ensemble de techniques pour résoudre le problème de surapprentissage.

A la suite des blocs de code sur les couches du réseau de neurones viennent les fonctionnalités ***ImageDataGenerator*** et **model.fit** propre à keras. Ces fonctionnalités permettent respectivement d'augmenter la quantité de données à temps réel et à produire un modèle en utilisant de générateurs de données.
Les options utilisées dans ces fonctionnalités sont [19] :

- **rescale** : est une valeur par laquelle les données sont multipliées avant de réaliser tout autre traitement. Car les images d'origine sont constituées de coefficients RGB entre 0 et 255, mais de telles valeurs seraient trop élevées pour être traitées dans nos modèles (considérant un temps d'apprentissage typique), on cible donc des valeurs comprises entre 0 et 1 en appliquant un changement d'échelle suivant le facteur 1/255.
- **share_range** : appliquer aléatoirement des transformations de cisaillement.
- **zoom_range** : zoomer de façon aléatoire dans les images.
- **horizontal_flip** : retourne horizontalement de façon aléatoire la moitié des images. Cette option est pertinente lorsqu'il n'y a pas de présomption d'asymétrie horizontale dans le sujet à reconnaître (par exemple dans des images de purs paysages).

Par ailleurs, nous avons défini le ***class_mode*** à ***categorical*** conformément à la classification parmi les 8 dossiers (***0-graine***, ***1-graine***, ***2-graine***, ***3-graine***, ***4-graine***, ***5-graine***, ***6-graine*** et ***sup-6-graine)***. Les options disponibles pour ce paramètre sont visualisées à la figure 3.8.

```
expected one of: {'categorical', 'sparse', 'input', None, 'binary'}
```

Figure XX: Options usuelles classificatrices

Les options les plus usuelles sont définies comme suit :

- **categorical** : ce mode permet de classer les données d'entrées dans un ensemble de catégories à prédire (au moins 3).
- **sparse** : cette option fait référence à des valeurs nulles ou vides.
- **binary**: cette classification prédit l'appartenance à une classe parmi deux.

Les deux autres fonctions de notre programme que sont **summarize_results()** et **run_experiments()** permettent d'une part de calculer la moyenne et l'écart-type des résultats obtenus, et d'autre part de répéter la fonction evaluate_model() afin de bien entraîner le modèle et d'afficher un résumé de l'ensemble des résultats obtenus en guise de clarté.

```
78
79   # summarize scores
80   def summarize_results(scores):
81       #print(scores)
82       m, s = statistics.mean(scores), statistics.stdev(scores)
83       print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))
84
85
86   # run an experiment
87   def run_experiment(repeats=10):
88
89       scores = list()
90       for r in range(repeats):
91           score = evaluate_model()
92           score = score * 100.0
93           print('>#%d: %.3f' % (r+1, score))
94           scores.append(score)
95       # summarize results
96       print('l\'ensemble des scores obtenus est de: ' + str (scores))
97       summarize_results(scores)
98
99   # run the experiment
100  run_experiment()
```

3.4.2 Evaluation selon la hiérarchie et la surface moyenne

Cette partie pouvez être intégrée à la précédente (3-4-1). Mais comme précisé au deuxième paragraphe de la section 3-3-2, l'ajout de nouvelles images nous a amenés à optimiser notre code.

Concrètement, les ajustements correspondent aux lignes de code 20-21-23-41-70 et 73 de la section 3-4-1 précédente.

```
1  from keras.models import Sequential
2  from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
3  from keras.preprocessing.image import ImageDataGenerator
4  from keras.utils import to_categorical
5  import statistics
6  import cv2
7  import pandas as pd
8  import numpy as np
9  from numpy import mean
10 from numpy import std
11 from tensorflow.keras import regularizers
12
13 # dimensions of our images.
14 img_height, img_width = 200, 200
15
16 input_shape = (img_height, img_width, 3)
17
18 def evaluate_model():
19
20     train_images_path = "C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/data/manuel_train_detection_et_comptage_de_graine"
21     test_images_path = "C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/data/manuel_test_detection_et_comptage_de_graine"
22
23     verbose, epochs, batch_size = 0, 20, 25
24
25     model = Sequential()
26     model.add(Conv2D(32, (3, 3), activation="relu", input_shape = input_shape))
27     model.add(MaxPooling2D((2, 2)))
28
29     model.add(Conv2D(64, (3, 3), activation="relu"))
30     model.add(MaxPooling2D((2, 2)))
31
32     model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
33     model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
34     model.add(Dropout(0.5))
35     model.add(MaxPooling2D((2, 2)))
36
37     model.add(Flatten())
38
39     model.add(Dense(512, activation='relu', activity_regularizer=regularizers.l1(0.0001)))
40     model.add(Dropout(0.5))
41     model.add(Dense(5, activation='softmax')) #softmax #sigmoid
```

```
68      model.fit(
69          train_generator,
70          steps_per_epoch=len(train_generator),
71          epochs=epochs,
72          validation_data=validation_generator,
73          validation_steps=len(validation_generator),
74          )
75
76      model.save_weights('poids_model.h5')
77      _, accuracy = model.evaluate(validation_generator, steps=len(validation_generator), batch_size=batch_size, verbose=0)
78      return accuracy
```

Vous vous rendrez compte que c'est principalement des adaptations concernant les dossiers pour l'entrainement et la validation d'une part, et des paramétrages rendant notre CNN plus autonome dans certains choix (lignes 70 et 73) d'autre part.

3.5 Conclusion

Nous sommes parties des paramètres étudiés du système à améliorer, qui se basent tous sur quelques fonctions du module d'opencv. A l'issue de cela, nous avons présenté chaque étape impliquée dans la procédure d'amélioration du système de base. Dans cette procédure, des lignes de code sont présentées pour indiquer d'emblée les solutions proposées.

CHAPITRE 4

PRESENTATION DES RESULTATS ET ANALYSES

Ce chapitre est consacré à la présentation des résultats obtenus avec l'implémentation des programmes élaborés au chapitre 3.
Les utilitaires pour l'exécution des différents programmes sont également présentés dans les sections du chapitre 3.
La plateforme utilisée pour l'édition des programmes est **Notepad++ version 7.9.1**.
Nous avons également utilisé **PowerShell** comme interpréteur de commande, dont la version et d'autres caractéristiques sont présentées dans la figure 4.1.

```
(base) PS C:\Users\USER> $psversiontable

Name                           Value
----                           -----
PSVersion                      5.1.19041.610
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.19041.610
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Figure YY: Version PowerShell

4.1 Détection Automatique

Nous avons à notre disposition des plateaux comportant 45 cellules (figure 4.2) et 15 cellules (figure 4.3).

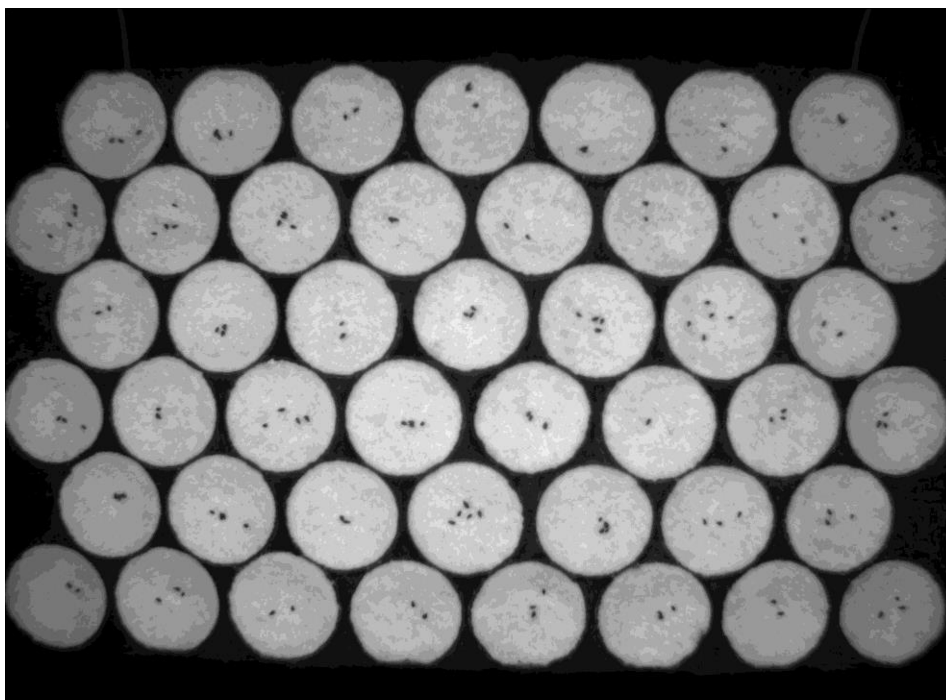


Figure ZZ: Image originale d'un plateau d'épinette noire

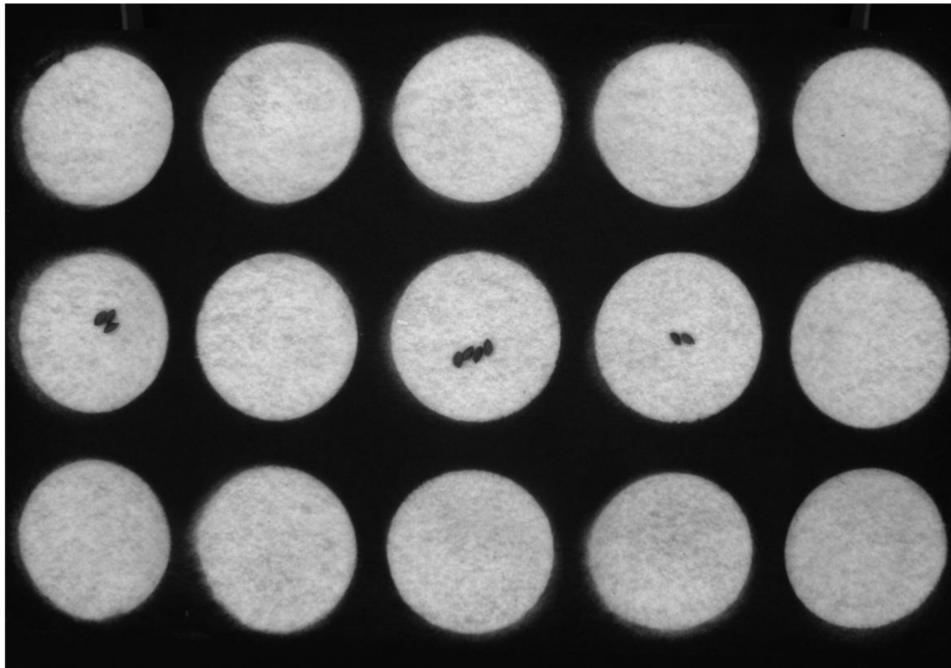


Figure AAA: Plateau de graines d'épinette blanche (15 cellules)

La détection automatique consiste ici à mettre en exergue les éléments constitutifs du plateau entre autres les cellules et les graines dans les cellules.
Avec le programme présenté à la section 3-2) nous obtenons les résultats de la figure 4.4.

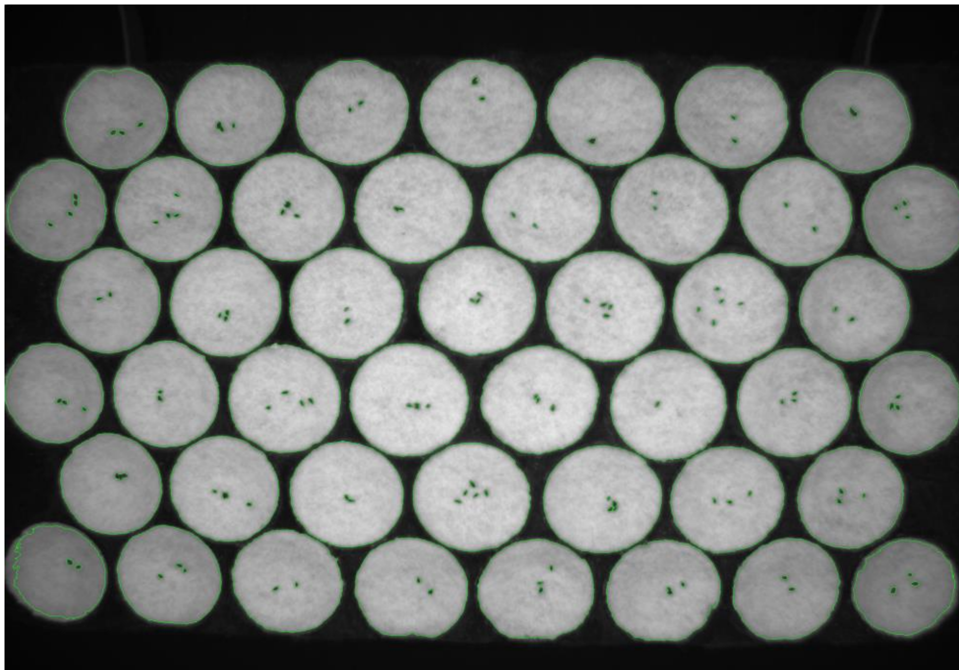


Figure BBB: Image résultante de la détection automatique de graines EPN

Nous constatons que l'ensemble des contours des cellules ont bien été dessinés. L'épaisseur des contours en vert peut être augmentée avec le dernier paramètre de la ligne 28 dans le code à la section 3-2) que nous avons mise à **1**.
La plupart des contours des graines sont bien visibles. Mais quelques rares graines collées sont considérées comme étant une seule graine. D'où l'intérêt de faire l'inclusion de la notion de surface moyenne afin de gérer ce problème.
Les images des autres types de graines (figures 4.5, 4.6 et 4.7) sont détectées de la même façon.

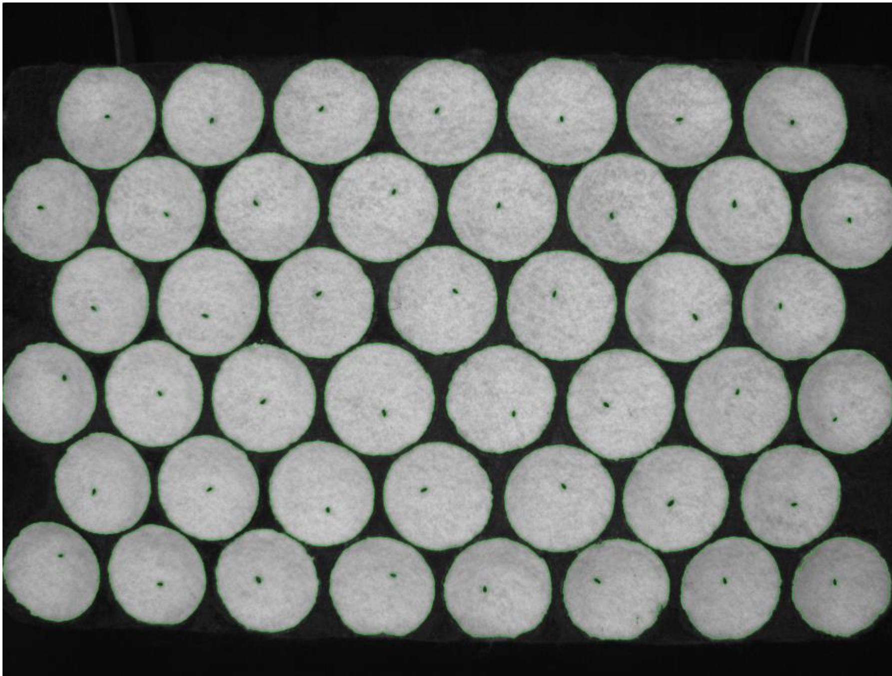


Figure CCC: Image résultante de la détection automatique de graine d'EPP

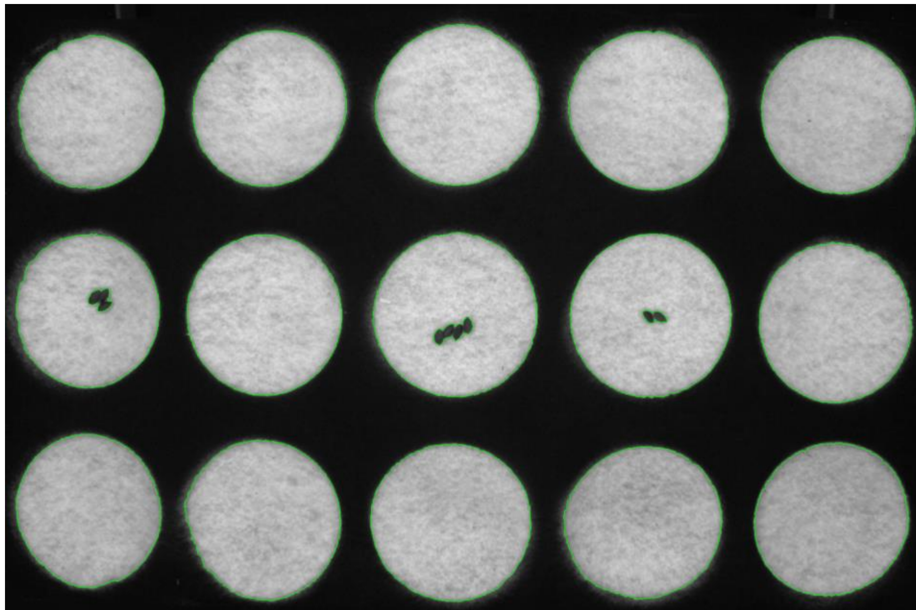


Figure DDD: Détection automatique de graines d'EPB plateau de 15 cellules

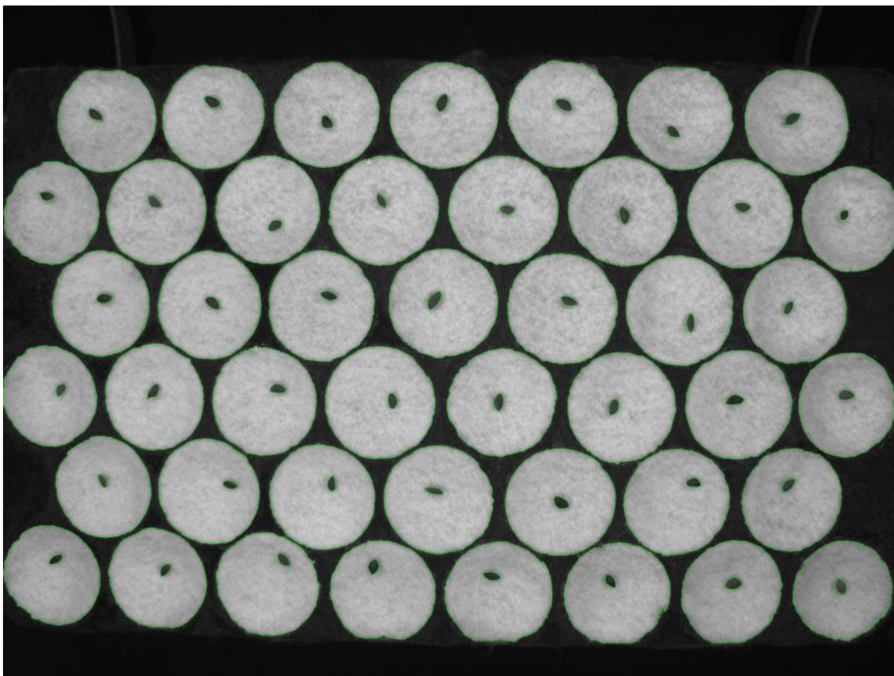


Figure EEE: Image résultante de la détection automatique de graines de PIG

4.2 Comptage de graines de semences de résineux

Le comptage est le processus par lequel on trouve le nombre de graines contenu dans chaque cellule. Un ensemble de lignes de code est présenté à la section 3-3). A cet effet, nous nous sommes basés sur la hiérarchie des contours en premier lieu comme relatée plus haut et ensuite nous avons couplé la hiérarchie avec la surface moyenne d'une graine selon le type.

4.2.1 Approche basée sur la hiérarchie

En vue d'avoir une traçabilité du nombre de graines trouvées, nous affichons les résultats dans le PowerShell (figure 4.8), qui sont matérialisés par la ligne 188 du programme à la section 3-3-1).

```
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\detection_et_comptage_de_graine
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/detection_et_comptage_c
seuil calculé : 124.0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 5
le nombre de graine(s) dans la cellule est : 5
cellule_10roi.png
cnt_idx.size : 0
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\detection_et_comptage_de_graine
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/detection_et_comptage_c
seuil calculé : 133.0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 2
le nombre de graine(s) dans la cellule est : 2
cellule_11roi.png
cnt_idx.size : 0
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\detection_et_comptage_de_graine
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/detection_et_comptage_c
seuil calculé : 103.0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 1
le nombre de graine(s) dans la cellule est : 1
cellule_12roi.png
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\detection_et_comptage_de_graine
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/detection_et_comptage_c
seuil calculé : 133.0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 2
le nombre de graine(s) dans la cellule est : 2
cellule_13roi.png
cnt_idx.size : 0
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\detection_et_comptage_de_graine
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/detection_et_comptage_c
seuil calculé : 106.0
cnt_idx.size : 0
cnt_idx.size : 0
cnt_idx.size : 2
le nombre de graine(s) dans la cellule est : 2
cellule_14roi.png
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\detection_et_comptage_de_graine
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/detection_et_comptage_c
```

Figure FFF: Nombre de graines par cellules selon la hiérarchie sur PowerShell

Une fois que le nombre de graines est calculé, la cellule concernée est enregistrée dans un dossier portant un nom faisant référence au nombre de graines (figure 4.9). Nous montrons dans l'explorateur de fichiers ce dont il s'agit :

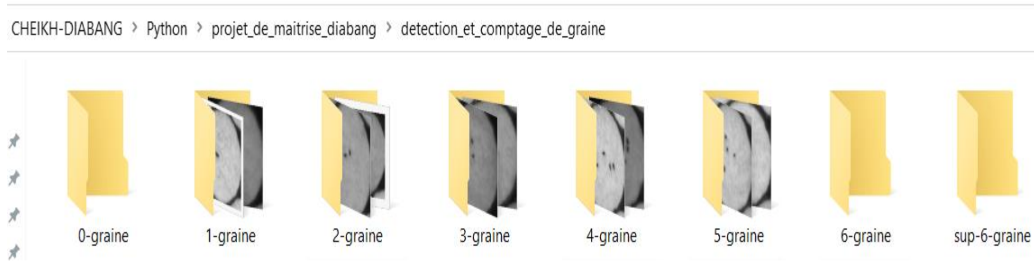


Figure GGG: Explorateur de fichier contenant les dossiers d'images de graines classées dans chaque classe (0-graine, 1 graine, 2 graines, etc.)

Par exemple, si l'on ouvre le dossier 5-graine (figure 4.10), on peut voir les images des cellules et les graines dans chacune des cellules.

Figure HHH: Cellules contenant des graines dans le bon répertoire

A ce niveau, le nombre de graines par cellule est parfaitement décompté en se basant unique sur la hiérarchie. Car les graines sont espacées les unes des autres.
Mais dans le cas où des graines sont collées, le mécanisme permettant de connaitre le nombre exact de graines dans une cellule à l'aide de la hiérarchie uniquement comporte des limites. Nous pouvons voir cela dans les cellules 1, 18 et 27 du dossier devant contenir que des images de cellules avec 3 graines (figure 4.11).

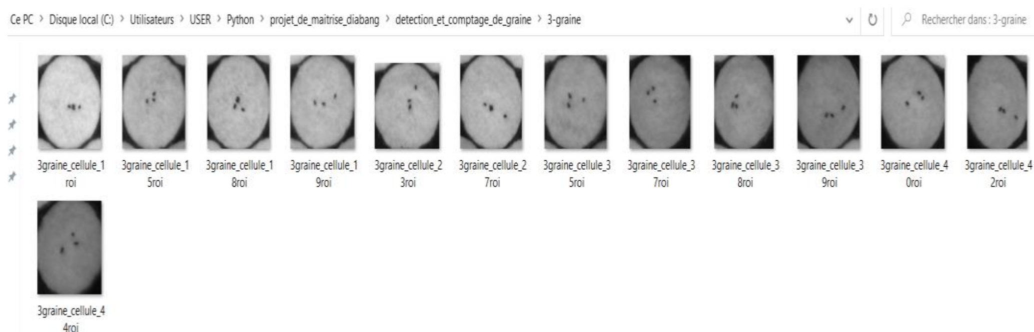


Figure III: Cellules contenant des graines dont 3 mal classées

4.2.2 Approche basée sur la hiérarchie et la surface moyenne de la graine

Nous partons des résultats affichés sur la console PowerShell en guise d'illustration de cette partie (figure 4.12). Le code présenté à la section 3-3-2 nous a permis son obtention.

```
seuil calculé : 121.0
le nombre de graine(s) dans ce contour est : 4
le nombre de graine(s) dans cette cellule est : 4
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\manuel\PIG4GR_airefix\cellule_14roi.png
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/PIG4GR_airefix/cellule_14roi.png
seuil calculé : 117.0
le nombre de graine(s) dans ce contour est : 4
le nombre de graine(s) dans cette cellule est : 4
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\manuel\PIG4GR_airefix\cellule_15roi.png
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/PIG4GR_airefix/cellule_15roi.png
seuil calculé : 107.0
le nombre de graine(s) dans ce contour est : 4
le nombre de graine(s) dans cette cellule est : 4
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\manuel\PIG4GR_airefix\cellule_16roi.png
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/PIG4GR_airefix/cellule_16roi.png
seuil calculé : 96.0
le nombre de graine(s) dans ce contour est : 4
le nombre de graine(s) dans ce contour est : 0
le nombre de graine(s) dans cette cellule est : 4
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\manuel\PIG4GR_airefix\cellule_17roi.png
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/PIG4GR_airefix/cellule_17roi.png
seuil calculé : 119.0
nombre de graines plus grand que 4
le nombre de graine(s) dans ce contour est : 5
le nombre de graine(s) dans ce contour est : 0
le nombre de graine(s) dans cette cellule est : 5
impossible de sauvegarder l'image de cette cellule, car nombre de graine inconnu
chemin d'accès à l'image: C:\Users\USER\Python\projet_de_maitrise_diabang\manuel\PIG4GR_airefix\cellule_18roi.png
nouveau chemin d'accès à l'image: C:/Users/USER/Python/projet_de_maitrise_diabang/manuel/PIG4GR_airefix/cellule_18roi.png
```

Figure JJJ: Nombre de graines selon la hiérarchie et le surface moyenne sur PowerShell

En ayant en tête que la finalité est de trouver le nombre de graines par cellule, on passe indéniablement par les contours fils détectés au sein de la cellule. Alors vous remarquerez ici que dans la plupart des cellules, un seul contour fils a été détecté, mais le nombre de graines (4) correspond au bon comptage. Ce dernier est rendu possible grâce à la notion de surface moyenne appliquée à la hiérarchie du contour.
Car si l'on se focalisait uniquement sur la hiérarchie, là où on a 4 comme nombre de graines dans le contour détecté, serai 1 en réalité. L'image de la figure 4.13, montre ce dont on parle.

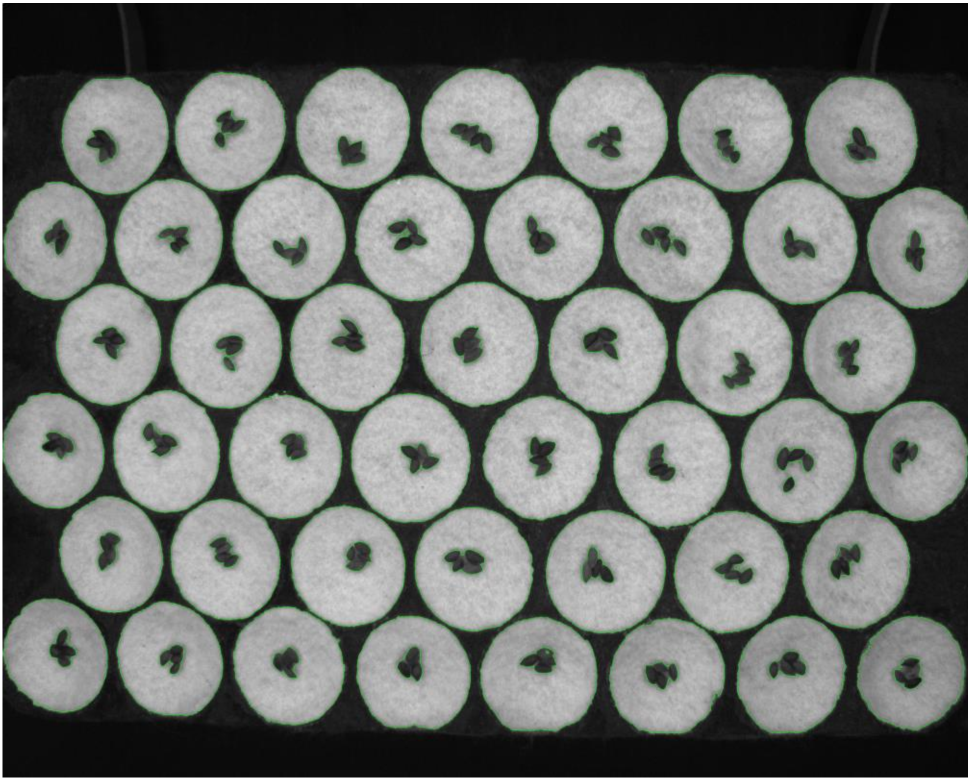


Figure KKK: Image d'un plateau de 4 graines de PIG par cellule avec contours détectés

Néanmoins, nous notons par moment des limites par rapport à cette approche du fait de la taille variable des graines. Comme dans la figure 4.14, le dossier 3-graine compte une cellule ayant 4 graines, mais mal classé.

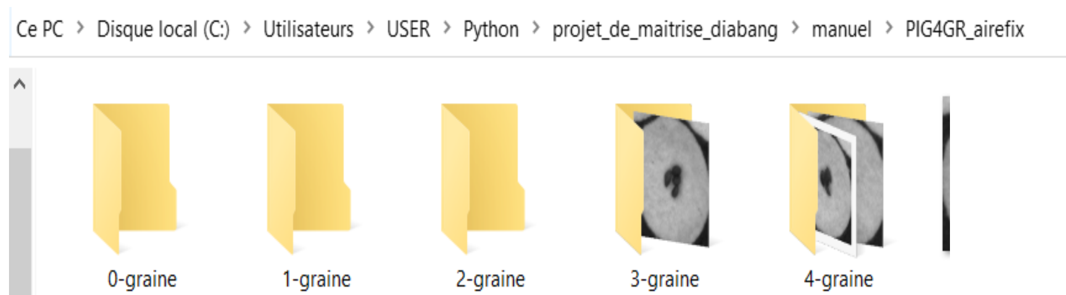


Figure LLL: Les dossiers de l'explorateur de fichier contenant les nouvelles images

Ce problème pourrait être résolu, si le commerce demandait des graines respectant certaines normes à ses fournisseurs pour chacun des trois types de graines.

4.3 Evaluation à temps réel des performances d’ensemencement

Après la phase de comptage de graines, nous procédons à l’estimation des performances des programmes proposés à cet effet.
Pour se faire, les réseaux de neurones convolutifs sont utilisés. Une analyse détaillée des procédures est présentée dans les sous-sections de la section 3-4) du chapitre 3.

4.3.1 Performances enregistrées avec la méthode basée sur la hiérarchie

L’exécution du programme en 3-4-1) fournit les résultats de la figure 4.15.

```
>#10: 48.980
l'ensemble des scores obtenus est de: [48.97959232330322, 48.97959232330322, 48.97959232330322, 48.97959232330322, 48.97959232330322, 48.97959232330322, 48.9795
Accuracy: 48.980% (+/-0.000)
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang>
```

Figure MMM: Résultats méthode basée sur la hiérarchie

Nous pouvons voir le score obtenu à chaque répétition (10 au total).
Le résultat de l’évaluation donne une moyenne de 48,98% et un écart type de 0.

En comparant ce résultat avec le système développé au préalable, qui avait une précision de 30%, on note une nette amélioration. Mais ce score dans l’ensemble sera jugé insuffisant d’autant plus qu’il n’arrive pas à 50%.
Dans le but de mieux visualiser l’efficacité de la classification par classe, nous avons ajouté des bouts de code pour inclure la matrice de confusion. Pour ce faire, nous avons installé une bibliothèque additionnelle (scikit-learn) (figure 4.16).

```
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang> pip install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-1.1.1-cp38-cp38-win_amd64.whl (7.3 MB)
     |████████████████████████████████| 7.3 MB 58 kB/s
Collecting joblib>=1.0.0
  Downloading joblib-1.1.0-py2.py3-none-any.whl (306 kB)
     |████████████████████████████████| 306 kB 6.8 MB/s
Requirement already satisfied: numpy>=1.17.3 in c:\users\user\miniconda3\lib\site-packages (from scikit-learn) (1.18.5)
Requirement already satisfied: scipy>=1.3.2 in c:\users\user\miniconda3\lib\site-packages (from scikit-learn) (1.5.0)
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
Installing collected packages: joblib, threadpoolctl, scikit-learn
Successfully installed joblib-1.1.0 scikit-learn-1.1.1 threadpoolctl-3.1.0
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang>
```

Figure NNN: Bibliothèque additionnelle (scikit-learn)

Ce module intègre des fonctions spécifiques pour l’apprentissage automatique et la modélisation statistique.
Les lignes de code faisant référence à cette fonctionnalité (matrice de confusion) sont données ci-après :

```python
import sklearn.metrics as metrics
from sklearn.metrics import confusion_matrix, classification_report
```

```
#----------------------------------------VALIDATION REPORT--------------------------------------------
test_steps_per_epoch = np.math.ceil(validation_generator.samples / validation_generator.batch_size)

predictions = model.predict_generator(validation_generator, steps=test_steps_per_epoch)
# Get most likely class
predicted_classes = np.argmax(predictions, axis=1)

true_classes = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

report = metrics.classification_report(true_classes, predicted_classes, target_names=class_labels)
print('--------------------------------------VALIDATION REPORT--------------------------------------------')
print(report)
print('Confusion Matrix Validation')
print(confusion_matrix(true_classes, predicted_classes))
#----------------------------------------TRAIN REPORT--------------------------------------------
train_steps_per_epoch = np.math.ceil(train_generator.samples / train_generator.batch_size)

train_predictions = model.predict_generator(train_generator, steps=train_steps_per_epoch)
# Get most likely class
train_predicted_classes = np.argmax(train_predictions, axis=1)

train_true_classes = train_generator.classes
train_class_labels = list(train_generator.class_indices.keys())

train_report = metrics.classification_report(train_true_classes, train_predicted_classes, target_names=train_class_labels)
print('--------------------------------------TRAIN REPORT--------------------------------------------')
print(train_report)
print('Confusion Matrix Train')
print(confusion_matrix(train_true_classes, train_predicted_classes))
```

Les résultats de la matrice de confusion sont présentés à la figure 4.17.

```
--------------------------------------VALIDATION REPORT--------------------------------------------
              precision    recall  f1-score   support

    0-graine       0.55      1.00      0.71        12
    1-graine       0.00      0.00      0.00         2
    2-graine       0.00      0.00      0.00         4
    3-graine       0.00      0.00      0.00         3
    4-graine       0.00      0.00      0.00         1

    accuracy                           0.55        22
   macro avg       0.11      0.20      0.14        22
weighted avg       0.30      0.55      0.39        22

Confusion Matrix Validation
[[12  0  0  0  0]
 [ 2  0  0  0  0]
 [ 4  0  0  0  0]
 [ 3  0  0  0  0]
 [ 1  0  0  0  0]]
C:\Users\USER\Miniconda3\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning: Precision and F-score are ill-def
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\USER\Miniconda3\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning: Precision and F-score are ill-def
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\USER\Miniconda3\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning: Precision and F-score are ill-def
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
--------------------------------------TRAIN REPORT--------------------------------------------
              precision    recall  f1-score   support

    0-graine       0.59      1.00      0.74        48
    1-graine       0.00      0.00      0.00         4
    2-graine       0.00      0.00      0.00        16
    3-graine       0.00      0.00      0.00        10
    4-graine       0.00      0.00      0.00         3

    accuracy                           0.59        81
   macro avg       0.12      0.20      0.15        81
weighted avg       0.35      0.59      0.44        81

Confusion Matrix Train
[[48  0  0  0  0]
 [ 4  0  0  0  0]
 [16  0  0  0  0]
 [10  0  0  0  0]
 [ 3  0  0  0  0]]
>#1: 54.545
```

Figure OOO: Résultats matrice de confusion selon la hiérarchie

Comme vous avez dû le constater, nous avons essayé d'avoir en plus de la matrice de confusion, un rapport détaillé du taux de prédiction positif (**precision**), du taux de

positif correctement prédit (**recall**), de la capacité du modèle à bien prédire les images positif (**f1-score**) et le nombre d'images par sous-dossier (**support**) pour chaque dossier contenant des images pour l'entrainement et la validation.

En effet, la métrique **precision** permet de calculer le nombre de faux positifs, c'est-à-dire des images détectées par erreurs. Alors, minimiser cette valeur, cele revient à limiter ces dernières.

Sur l'ensemble des résultats obtenus, nous pouvons voir que le dossier contenant des images de 0 graine est plus en mesure d'être bien classifié avec un taux de 71% de la f1-score pour les données de validation et 74% pour les données d'entrainement.

En même temps, lors de la première itération de l'exécution du programme, nous constatons que la valeur du occuracy est de 54,545, qui est différent des 48,979 obtenues précédemment. Cela est dû à un ajustement sur le nombre de dossiers qui est passé de 7 au préalable à 5.

4.3.2 Performances enregistrées avec la méthode basée sur le couple hiérarchie - surface de la graine

L'optimisation du programme permettant d'obtenir le score précédent est liée non seulement à l'ajout de nouvelles images, mais également à la nature du résultat.

Alors le concept de la surface comme autre paramètre à prendre en compte est censé donner plus de précision par rapport à la classification.

Ainsi, la figure 4.18 permet d'élucider les résultats avec cette approche.

```
>#10: 81.818
l'ensemble des scores obtenus est de: [81.81818127632141, 81.81818127632141, 81.81818127632141, 81.81818127632141, 81.8181812
Accuracy: 81.818% (+/-0.000)
(base) PS C:\Users\USER\Python\projet_de_maitrise_diabang\manuel>
```

Figure PPP: Résultats méthode basée sur la hiérarchie et la surface moyenne

Nous enregistrons un score assez élevé de 81,82%. Ce résultat est en effet un matching entre les différents dossiers 0-graine, 1-graine, 2-graine, 3-graine et 4-graine placés dans les dossiers de test et de validation.

Par suite, nous avons également intégré la matrice de confusion pour avoir un aperçu sur l'efficacité de la classification par classe. Les lignes de code ajoutées restent inchangées par rapport à celles de la section précédente. Les résultats ainsi obtenus sont affichés à la figure 4.19.

```
---------------------------------------VALIDATION REPORT--------------------------------------------
              precision    recall  f1-score   support

    0-graine       0.38      0.85      0.52        27
    1-graine       0.33      0.04      0.07        24
    2-graine       0.10      0.06      0.08        16
    3-graine       0.00      0.00      0.00         9
    4-graine       0.00      0.00      0.00         6

    accuracy                           0.30        82
   macro avg       0.16      0.19      0.13        82
weighted avg       0.24      0.30      0.21        82

Confusion Matrix Validation
[[23  1  2  0  1]
 [14  1  5  0  4]
 [11  1  1  0  3]
 [ 8  0  1  0  0]
 [ 5  0  1  0  0]]
C:\Users\USER\Miniconda3\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning: Precision
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\USER\Miniconda3\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning: Precision
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\USER\Miniconda3\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning: Precision
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
---------------------------------------TRAIN REPORT--------------------------------------------
              precision    recall  f1-score   support

    0-graine       0.35      0.78      0.48       108
    1-graine       0.27      0.03      0.06        98
    2-graine       0.15      0.12      0.13        66
    3-graine       0.00      0.00      0.00        31
    4-graine       0.05      0.05      0.05        22

    accuracy                           0.30       325
   macro avg       0.16      0.20      0.14       325
weighted avg       0.23      0.30      0.21       325

Confusion Matrix Train
[[84  5 13  0  6]
 [67  3 19  0  9]
 [55  0  8  0  3]
 [19  2  9  0  1]
 [15  1  5  0  1]]
>#3: 45.122
```

Figure QQQ: Résultats matrice de confusion selon la hiérarchie et la

Les statistiques montrent des résultats significatifs de la métrique **recall** (vraie positive) des images correspondantes au dossier 0-graine, dont 85% enregistré pour les données de validation et 78% pour les données d'entrainement. Cela vient confirmer les 81,82% d'**accuracy** obtenus au début de cette section. Car cette valeur a été enregistrée avec les dossiers 0-graine placés dans les dossiers d'entrainement et de validation.
Par contre, les autres métriques dans l'ensemble donnent des résultats peu concluants.

En effet, il n'y a pas de méthode universelle pour définir les bons paramètres et qu'il faut en réalité ''tâtonner'' en s'appuyant sur des heuristiques plus ou moins heureuses [20].

4.4 Conclusion

La détection automatique affiche bien les zones d'intérêts, c'est-à-dire les contours des graines. Mais, quelques soit la taille de la graine, si les graines se trouvent coller les unes aux autres, on constate une mauvaise délimitation des contours. Ainsi le comptage du nombre de graines par contour donne des résultats aberrants si l'on se focalise uniquement à la hiérarchie du contour. L'ajout de la surface moyenne de la graine comme nouvelle métrique, permet de résoudre en partie ce problème. De ce fait, on voit plus ou moins l'impact de cet ajustement dans les performances enregistrées.

CHAPITRE 5

CONCLUSION

Le but de ce travail était d'améliorer un système intelligent de détection et de comptage de graines de semence de résineux.
Pour ce faire, nous avons présenté dans l'introduction la structure du système à améliorer. De même, nous avons mis à nu les problématiques existantes qui ont d'ailleurs motivé le thème présent. Par exemple des codes de couleurs inappropriés par rapport aux nombres de graines dans les cellules du plateau. Cela avait pour effet le faible taux de précision de 30% du système que l'on a cherché à améliorer.
Nous avons ainsi dans le chapitre 2, passé en revue les termes se rapportant à la littérature des concepts afin de mieux cerner leurs importances dans nos travaux. En général, les concepts peuvent être regroupés en deux ensembles : la vision par ordinateur d'une part et les réseaux de neurones artificiels d'autre part.

Afin d'illustrer nos démarches sur les solutions proposées, le chapitre 3 relate exclusivement la méthodologie adoptée. Après que le plateau a été détecté via des fonctions intégrées d'openCV, nous nous sommes basés premièrement sur la hiérarchie des contours pour trouver le nombre de graines dans une cellule. Deuxièmement, nous avons intégré la nation de la surface moyenne de la graine après qu'on nous ait ajouté de nouvelles images. Ensuite, des algorithmes ont été implémentés afin d'enregistrer l'image de la cellule ayant des graines dans des dossiers qui font allusion au nombre de graines trouvées dans chaque cellule. Et ces dossiers sont par la suite utilisés par un modèle de réseaux de neurones convolutifs avec 6 couches au total dans notre cas pour prédire les performances du système de détection et de comptage de graines proposé selon les deux cas.
Dans le chapitre 4, on a montré les résultats obtenus. L'estimation des performances est en moyenne de l'ordre de 48.98% dans le cas où la hiérarchie est utilisée comme seul paramètre ; et de 81,82% (plus grande valeur enregistrée durant le matching des dossiers contenant les graines) avec un second paramètre d'inclus faisant référence à la surface moyenne de la graine via notre modèle de réseau de neurones. Ces résultats, bien qu'ils soient à améliorer, restent supérieurs à 30% du système étudié.

Des pistes pour l'amélioration de nos résultats sont à pourvoir. Il s'agit d'optimiser le programme de détection et de comptage de graines. Pour ce faire, il y a lieu d'utiliser des paramètres additionnels qui satisferont l'ensemble des conditions pouvant indiquer avec exactitude la structure d'une graine. Aussi voir des possibilités du côté du fournisseur comment avoir des graines de même taille selon son type d'appartenance. Par ailleurs, il faut également songer à avoir un dataset assez conséquent comme jeu de données à notre modèle de réseau de neurones convolutifs.

BIBLIOGRAPHIE

[1] EVANS GIRARD. (2016, juin). DETECTION AUTOMATIQUE DE SEMENCES DE RESINEUX POUR L'EVALUATION EN TEMPS REEL DE L'EFFICACITE D'UN SEMOIR.

[2] Wikipedia contributors. (2020, 6 décembre). Intelligence.

[3] Patrick Hébert, Samy Metari & Denis Laurendeau. (2017, juin). Traitement des images (Partie 2 : segmentation de bas niveau).

[4] Plateforme Logicielle Cépia - Filtres moyenneurs. (2004-2020)

[5] Plateforme Logicielle Cépia - Filtres d'ordre : min, max et médian. (2004-2020).

[6] I.N.R.A. (2004-2020). Plateforme Logicielle Cépia - Erosion et Dilatation.

[7] I.N.R.A. (2004-2020). Plateforme Logicielle Cépia - Ouverture et fermeture.

[8] optique pour l'ingénieur, O. P. I. (2009). Détection des contours - Quelques opérateurs gradient.

[9] Max Mignotte. (s. d.). TRAITEMENT D'IMAGES FILTRAGE SPATIAL.

[10] L, B. (2019, 5 avril). Réseau de neurones artificiels : qu'est-ce que c'est et à quoi ça sert ?

[11] Andrianjafinony Rosa Fidelys JOHARY, Solofo RAKOTONDRAOMPIANA, Hibrahim Rijasoa RAVON JIMALALA, Solofoarisoa RAKOTONIAINA. (2018, juin). DES CHANGEMENTS D'OCCUPATION DU SOL AVEC LA MÉTHODE IR-MAD. APPLICATION À LA FORÊT SÈCHE DES MIKEA (S.-O. DE MADAGASCAR).

[12] Réseaux Neuronaux. (2020, octobre) https://www.emse.fr/~pbreuil/amv/nn/index.html

[13] Elhani Djenaihi. Deep learning. 2 Plan réseaux de neurones artificiels Définition Fonction d'activation Fonction de cout Propagation et rétropropagation Algorithme d'optimisation.

[14] R., L. (2020, 11 février). Focus : MobileNet, une reconnaissance d'images temps réel et embarquée surpuissante. Pensée Artificielle.

[15] Simon Vézina. (s. d.). Chapitre 6.2 - Le réseau de neurones.

[16] Lambert R. (2019, 9 octobre). Comprendre le fonctionnement d'un LSTM et d'un GRU en schémas. Pensée Artificielle.

[17] Wikipedia contributors. (2020, 11 mars). Réseau de neurones récurrents.

[18] OpenCV : Contours Hierarchy. (s. d.). https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html

[19] (2017, 7 novembre). Construire de puissants modèles de classification d'images. SuperMaker.space.

[20] Eric. (2014, 19 octobre). Tanagra Optimal Neurons Perceptron.