UCLouvain

epl

École polytechnique de Louvain

Exploration de nouvelles techniques d'identification pour le recensement d'espèces protégées

Application pour le suivi de la population de Salamandres

Auteur: **François Duchêne**
Promoteurs: **Olivier Bonaventure, Sébastien Jodogne**
Lecteur: **Lucile Dierckx**
Année académique 2021–2022
Master [120] en sciences informatiques

Remerciements

Je voudrais tout particulièrement remercier les deux promoteurs de mon mémoire, le Prof. Olivier Bonaventure et le Prof. Sébastien Jodogne pour toute l'aide et la patience qu'ils ont pu m'accorder tout au long de la réalisation de ce travail. Je ne serais jamais arrivé à ce stade sans leurs conseils avisés.

Je tiens également à remercier ma lectrice Lucile Dierckx pour le temps qu'elle a pu me consacrer et sa patience.

Je voudrais exprimer toute ma reconnaissance la plus sincère à monsieur Jeroen Speybroeck sans qui ce mémoire n'aurait certainement pas été tel qu'il est, ainsi que pour toute la gentillesse avec laquelle il m'a apporté son aide.

Je voudrais également remercier monsieur Thibaut Thyrion pour toute la confiance et l'intérêt qu'il a pu m'accorder.

Remerciement spécial à Adrien Goffin pour tous ses encouragements et sa bonne volonté, je lui souhaite de tout mon coeur de réaliser ses projets.

Je voudrais également remercier ma famille et mes amis pour toute l'aide et les encouragements qu'ils ont pu m'apporter durant cette année. Je pense tout particulièrement à Luis et Jean-Martin qui ont bien voulu m'accompagner lors de mes virées nocturnes. Je tiens également à remercier Cosette et Pythou, qui m'ont supporté pendant mes longues soirées d'écriture.

Enfin, je tiens à remercier mon père, qui a toujours été là pour moi et qui m'a offert son temps et son aide dans les moments difficiles, et qui a su gardé patience même face à ma mauvaise humeur.

Table des matières

Chapitre 1

Introduction

La salamandre tachetée, aussi connue par son surnom de salamandre de feu, de son nom latin *Salamandra salamandra* est une espèce d'urodèles de l'ordre des amphibiens vivant sur une large portion de l'Europe de l'Ouest, centrale, méridionale et du Sud-est. Elle est bien implantée en Belgique, et jouit d'une protection de par les offices régionaux de Nature et Forêts. Bien qu'elle ne soit pas une espèce directement menacée, elle est malgré tout suivie par les scientifiques qui craignent une diminution de la population [26].

Cependant, c'est bien là le problème, jusqu'à récemment, les scientifiques ne disposaient pas d'outils efficaces de recensement de la population. La méthode couramment employée jusqu'à alors était le comptage manuel à l'aide d'un code binaire. Quand des chercheurs suivaient un tracé pré-défini et tombaient sur une salamandre sur le bord du chemin, ils donnaient un identifiant à chaque animal afin de pouvoir les identifier au prochain passage. Mais comment marquer chaque animal et être sûr que la marque sera toujours présente la prochaine fois ?

C'est très simple, ils leur coupaient les doigts ! En effet, les salamandres ont la particularité unique de pouvoir faire repousser certains de leurs membres coupés. Elles retrouvent ainsi de nouveaux doigts ou queue après quelques mois de convalescence. Cette propriété compensait en partie leurs membres tranchés. Les doigts servaient alors de "mémoire" : un doigt coupé équivalent à 1, un doigt non coupé à 0 [11].

Malgré la souffrance animale évidente, cette pratique a perduré pendant très longtemps. D'autres techniques moins intrusives ont été développées entre temps, comme des puces électroniques glissées sous la peau. Cependant, ces techniques présentaient toujours le même désavantage de devoir capturer physiquement l'animal, ce qui provoque un grand stress pour celui-ci.

C'est alors qu'il a été remarqué que les taches sur les dos de salamandres étaient vraisemblablement uniques pour chaque individu. Les salamandres naissent avec un motif unique de taches sur leur dos et pattes. En outre, ce motif ne change pas au fil du temps, à part au début de leur vie lorsqu'elles sont encore petites et que les taches s'étirent au fur et à mesure de leur croissance. Ainsi, à l'aide d'un dessin ou d'un photographie, il est possible de distinguer une nouvelle salamandre croisée sur le bord de la route d'une autre déjà répertoriée. Par conséquent, il est possible de recenser une population de salamandres avec cette méthode.

Plusieurs personnes l'ont déjà tentées, dont une étudiante en biologie de l'UCL en 2000, Violaine Fichefet, qui avait tenté de relever une population non loin de Louvain-la-neuve [11]. Cependant, la grande majorité du travail avait été manuel. Elle avait dû photographier et filmer l'intégralité de ses sorties nocturnes – car les salamandres sont des animaux nocturnes – et ensuite visionner à nouveau ses cassettes vidéos afin de compter le nombre d'individus différents présents sur ses prises. C'est un travail gargantuesque qui peut prendre des jours de travail [1].

Même s'il y a eu quelques tentatives de résolution de ce problème à l'aide de l'informatique [4], avec par exemple des applications d'aide au recensement, manuelles ou semi-automatiques, il n'existe malheureusement aucun programme permettant de le faire automatiquement pour les salamandres. En revanche, il existe des réflexions similaires pour d'autres animaux possédant des attributs visuels uniques à chaque individu d'une espèce [17]. Je peux par exemple citer le travail réalisé sur les girafes, qui possèdent des taches sur leur corps uniques à chaque individu.

Dans ce mémoire, je présenterai le travail d'expérimentation que j'ai réalisé dans le but de produire un outil automatique d'identification de salamandres. L'objectif est, qu'à partir de photographies réalisées par n'importe qui, que cela soit des professionnels ou des amateurs, une application téléphone puisse les comparer avec d'autres photos enregistrées dans une base de données. Dans ce but, un algorithme de comparaison de photographies robuste est nécessaire.

Plusieurs étapes ont été nécessaires avant de concevoir cet algorithme de comparaison. J'ai notamment dû passé un temps assez conséquent sur la préparation des données. J'ai par exemple utilisé certaines techniques de Deep-learning pour segmenter les images ainsi que de nombreuses techniques de traitement d'images.

La structure de ce travail est divisé de cette façon :

— Le chapitre 2 commencera par poser plusieurs bases de traitement d'images par ordinateur nécessaires à la bonne compréhension des chapitres suivant.

1. D'autres méthodes non-invasives ont également été mises au point en parallèle [30]

— Ensuite, le chapitre 3 décrira les ensembles de données dont j'ai disposé, et leurs particularités.
— Le chapitre 4 discutera de la normalisation des images.
— Après, le chapitre 5 parlera de l'étape cruciale de la segmentation des images, tout aussi bien par leurs contours que par leurs couleurs.
— Ensuite, l'algorithme de comparaison proposé sera abordé dans le chapitre 6. La méthode des *blobs et centroïdes* y sera décrite en détail.
— Les résultats obtenus lors des tests de performance de l'algorithme de comparaison proposé seront présentés dans l'antépénultième chapitre 7.
— Le chapitre 8 parlera des différentes pistes d'amélioration possible à la suite de ce travail. Je discuterai en outre des limites de ma solution.
— Enfin, une courte conclusion au Chapitre 9 clôtura ce mémoire.

Je mettrai en outre dans les appendices des ressources diverses qui m'ont été utiles pour la rédaction de ce document. Une partie des résultats brutes y seront présentés, ainsi que plusieurs liens utiles, notamment un lien vers le code produit.

Chapitre 2

Techniques de vision par ordinateur

Ce chapitre présente les principaux concepts et algorithmes utilisés dans ce travail pour faire du traitement d'images, en particulier les différents concepts empruntés à la littérature de la vision par ordinateur. Il traite des bases des différents espaces de couleurs, de l'algorithme de segmentation K-Moyennes et des composantes connexes.

2.1 Espaces de couleurs

Une image numérique est composée, entre autre, de plusieurs canaux de couleurs. Chaque canal prend la forme d'une matrice de deux dimensions de même résolution que l'image. Le nombre et la signification de chaque canal dépend de l'espace de couleurs utilisé pour décrire les couleurs de l'image. Je vais passer brièvement en revue les différents espaces de couleurs utilisés dans le développement de l'algorithme. Chacun a ses avantages et inconvénients, j'ai dû constamment jongler entre eux en fonction du contexte.

RGB est le système de codage de couleurs le plus proche du matériel. Il utilise la synthèse additive des trois couleurs primaires.

N&B représente une image en niveaux de gris. Au contraire des autres, il n'utilise qu'un seul canal de couleur.

HSV est un système de gestion des couleurs qui définit celles-ci comme étant une combinaison de teinte, de saturation et de luminosité.

YCbCr est un système de couleurs initialement créé pour la transmission hertzienne de vidéo en couleur. Y, Cb et Cr sont des flux composés d'information de luminance et de chrominance qui peuvent être utilisés pour recréer une image en couleur après retransmission.

CieLab Officiellement « *$L^*a^*b^*$ CIE 1976* » est un espace de couleur alternatif utilisé principalement pour la caractérisation des couleurs de surface. Les grandeurs qui caractérisent une couleur sont la luminance L^* et les paramètres a^* et b^* qui définissent des écarts de couleurs par rapport à une surface grise de même clarté. Concrètement, L^* représente une nuance de gris, a^* représente une valeur sur l'axe de couleurs vert $\rightarrow$ rouge, et b^* sur l'axe de couleurs bleu $\rightarrow$ jaune.

2.2 Segmentation par K-Means

La segmentation par **K-Means** est une technique de segmentation d'image par couleur tirée de la méthode de partitionnement de données par K-Means. La méthode a pour objectif de minimiser une fonction étant donné un nuage de points et k entiers. Elle va diviser les points en k groupes, appelés « *clusters* ». La fonction minimisée est la somme des carrés des distances d'un point à la moyenne des points de son cluster. La méthode est suffisamment généraliste pour qu'elle puisse être utilisée de différentes manières. Dans le cas de la segmentation par K-Means, la méthode consiste à grouper les pixels (points) en clusters en fonction de leur couleur, les pixels à couleurs semblables se groupant ensemble. La méthode est itérative, les clusters sont ajustés au fur et à mesure des itérations, jusqu'à ce que l'algorithme converge en une solution (fig. 2.1).

Définition mathématique Étant donné un ensemble de points $(x_1, x_2, \ldots, x_n)$, on cherche à partitionner les n points en k ensembles $S = \{S_1, S_2, \ldots, S_k\}$ $(k \leq n)$ en minimisant la distance entre les points à l'intérieur de chaque partition :

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

où μ_i est le barycentre des points dans S_i.

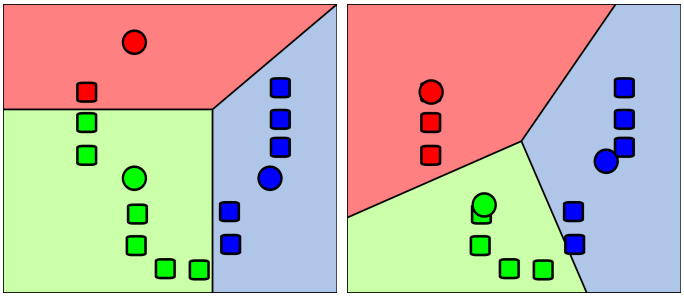


FIGURE 2.1 – Convergence de K-Moyennes – itération 0 (a) à n (b)
(Weston.pace (2007). "K Means Example Step 2.svg","K Means Example Step 4.svg". Récupéré de https://en.wikipedia.org/wiki/K-means_clustering . ©CC BY-SA 3.0)

Algorithme L'algorithme utilisé dans ce projet est celui de la bibliothèque logicielle sklearn [1], qui utilise par défaut l'algorithme d'Elkan [10]. L'algorithme d'Elkan est une optimisation de l'algorithme original rendant le même résultat mais beaucoup plus rapidement. Il utilise des propriétés d'inégalités de triangles afin d'éviter de réaliser certains calculs redondants. Cet algorithme montre son efficacité lorsque k est grand ($k > 20$), si k est plus petit, il agit comme l'algorithme de base. Il demande néanmoins plus de mémoire pour s'exécuter.

Les groupements peuvent se faire par couleur RGB, cependant en pratique, ce sont les nuances de gris qui sont utilisées. L'algorithme produit une série de k étiquettes représentant les classes.

2.3 Composantes connexes

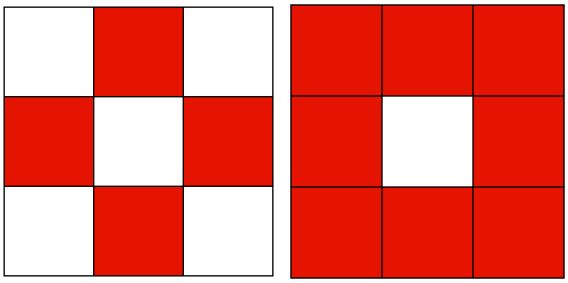
2.3.1 Aspect théorique

Les composantes connexes sont issues de la notion de connexité en topologie, où un objet est dit *connexe* s'il n'est composé que d'un seul tenant.

Une image binaire consiste en une matrice de 0 et de 1, représentant des pixels d'objets ou de fond. Il est possible, via certains algorithmes, de distinguer des structures connectées ou non à partir de cette matrice. Les algorithmes de détection des composantes connexes peuvent être séparés en deux familles distinctes en fonction de leur manière de détecter la connexité entre deux structures [24].

Ces deux familles sont la 4-connexité et la 8-connexité (fig. 2.2) :

1. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html



(a) 4-convexité (b) 8-convexité

FIGURE 2.2 – Différence entre les deux types de convexité

4-connexité : si deux pixels adjacents horizontalement ou verticalement, mais *non diagonalement*, sont de la même couleur, alors ils font partie de la même composante connexe.

8-connexité : si deux pixels adjacents horizontalement, verticalement ou diagonalement sont de la même couleur, alors ils font partie de la même composante connexe.

La 4-connexité a donc tendance à créer plus de composantes que la 8-connexité vu que la sensibilité est plus faible.

2.3.2 Aspect algorithmique

L'implémentation de l'algorithme de détection des composantes connexes nécessite l'utilisation d'étiquettes afin d'identifier les différentes composantes. L'étiquetage de composantes connexes est une procédure d'assignation d'un identifiant ("*label*") unique à chaque objet dans une image.

Chaque type de connexité utilise un algorithme différent. J'ai utilisé les algorithmes recommandés par défaut par la bibliothèque logicielle *open-cv*.

Pour la 4-convexité J'ai utilisé l'algorithme « ***S**can **A**rray **U**nion **F**ind* »(`SAUF`) [31]. Il est basé sur un algorithme d' *Union Find* utilisant des tableaux 2D et optimisé à l'aide d'arbres de décision.

Pour la 8-connexité J'ai utilisé l'algorithme « ***B**lock-**B**ased with **D**ecision **T**rees* »(`BBDT`) [13]. Il fonctionne à l'aide d'une grille de pixels 2×2 parcourant l'image à analyser et optimisé grâce au principe des arbres de décisions lui permettant de faire les choix les plus efficaces.

Chapitre 3

Jeux de données

Avant de réfléchir à la manière dont on pourrait identifier des individus, il est primordial de parler des données sur lesquelles on va pouvoir travailler. En l'occurrence, il s'agit des photographies de salamandres.

Il se trouve qu'étant une espèce protégée, il n'est pas facile d'obtenir des données sur la salamandre de feu. Il existe le site web *observation.be*[1] où les utilisateurs peuvent téléverser leurs photos. Cependant, afin de garder les sites de capture secrets, les photos sont en partie anonymisées. Les photos n'identifient en outre pas les individus à l'aide d'un code d'identification, ce qui rend leur utilisation pour un test d'algorithme d'identification et de comparaison impossible. Il a donc fallu être créatif pour récupérer des images.

J'ai réussi à réunir des photos de plusieurs sources. J'ai moi-même pu prendre des photos de salamandres non loin de Louvain-la-neuve, où je savais qu'il existait une petite colonie [11].

Après avoir contacté des professionnels du milieu, j'ai également pu avoir accès à de nouvelles photos présentant une meilleure qualité. Il s'agit de photos partagées par Jeroen Speybroeck (*Instituut natuur- en bosonderzoek*) qui a notamment développé le logiciel Mandermatcher[2]. Ce logiciel en accès libre aide notamment au recensement manuel des populations de salamandres. Je m'en suis servi pour créer mon propre ensemble de données, en identifiant manuellement plus de 300 salamandres (fig. 3.1). J'ai également pu avoir accès aux photos d'Adrien Goffin, volontaire enthousiaste de la région d'Andenne[3].

J'ai également pu rassembler quelques photos provenant des Étangs de la

1. https://observations.be/species/458/
2. https://jeroenspeybroeck.shinyapps.io/mandermatcher/
3. https://www.andenne.be/batraciens-vigilance-et-sauvetage/

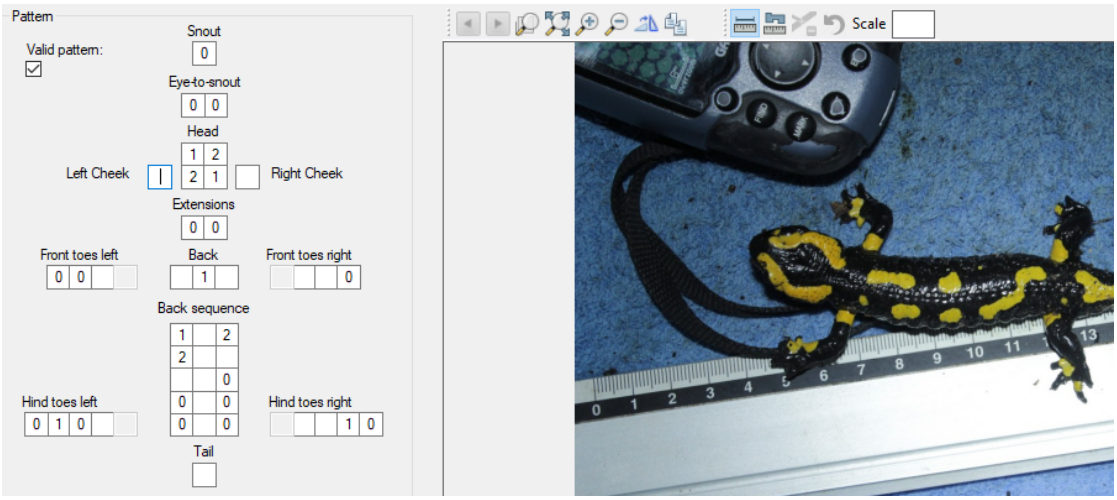


FIGURE 3.1 – Encodage manuel des taches dans Mandermatcher

Ensemble	Nb photos	Pro ?	Dimensions
LLN	313	Non	(de 1536x2049 à 4032x3024)
Mander	393	Oui	(de 680x1024 à 1280x960)
Andenne	47	Oui	(de 342x1552 à 960x2944)
Julienne	7	Non	(1536x2049)
Spy	8	Non	(560x756)
Contrôle	68	Non	(de 3000x4000 à 4032x3024)
Total	836		

TABLE 3.1 – Ensemble des photos utilisées pour la segmentation

Julienne et de Spy. Des photos neutres dites de *contrôle* ont également été collectées, mais ne contiennent aucune salamandres. Elles ont été prises par certains des appareils ayant servi à prendre en photos des salamandres. Elles seront utiles pour le chapitre concernant la segmentation d'images.

Parmi toutes les photos que j'ai eu à ma disposition, je n'en ai utilisé qu'une partie, soit par manque de temps soit parce qu'elles étaient de trop mauvaises qualité. Il y a également une distinction à faire entre l'ensemble utilisé pour la segmentation (tab. 3.1) et l'ensemble utilisé pour l'identification (tab. 3.2). En effet, pour le travail de segmentation, je n'avais pas besoin d'images où les salamandres étaient identifiées. Les dimensions des photos sont restées à titre d'indication, il y avait en général une grande diversité de résolutions différentes.

Ensemble	Nb photos	Pro ?	Dimensions
LLN	313	Non	(de 1536x2049 à 4032x3024)
Mander	393	Oui	(de 680x1024 à 1280x960)
Andenne	47	Oui	(de 342x1552 à 960x2944)
Total	753		

TABLE 3.2 – Ensemble des photos utilisées pour l'identification

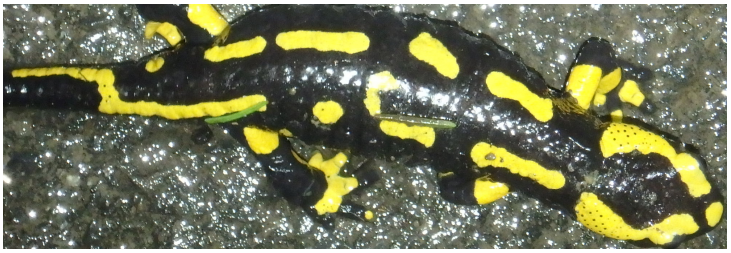
Hypothèses de base sur les données Comme dit précédemment, j'ai décidé de ne pas restreindre les sources de données aux seuls professionnels mais aussi aux amateurs, dans l'espoir d'entrainer plus efficacement le modèle de U-Net présenté dans les chapitres suivant. J'ai également émis l'hypothèse que chaque photo ne peut contenir qu'une seule et unique salamandre. Toutes les photos en comprenant plusieurs ont été écartées, et ce dans l'objectif de simplifier le travail d'identification.

Particularités du jeu de photos Les salamandres sortent la nuit, lorsque l'humidité est abondante et que la température atteint un certain niveau. Dès lors les photos doivent être prises avec une source de lumière artificielle. Compte tenu des circonstances, cela engendre des imperfections lors des prises de vue. Par exemple des photos en sur- ou sous-exposition, ou des photos floues. Parfois, les animaux se cachaient en partie dans la végétation, il y a régulièrement des cas où des parties de leur corps n'était tout simplement pas visible.

Exemples de photos Les photos présentées figure 3.2 sont représentatives des datasets utilisés et de la diversité des environnements. j'ai repris des images issues des 6 sources différentes : LLN, Mandermatcher, Andenne, Julienne, Spy et contrôle. On peut aisément remarquer la grande diversité de salamandres. La salamandre 3.2e par exemple ressort particulièrement du lot avec ses deux longues bandes de taches qui traversent son corps d'un bout à l'autre. Les photos de Mandermatcher en majorité se ressemblent fortement, toutes les salamandres ayant été précédemment déplacées dans un lieu contrôlé. Avant leur partage, les photos d'Andenne ont été normalisées au préalable. Elles ont toutes été recadrées pour centrer l'image sur leur corps, et toutes leurs têtes font face du même côté.

(a) LLN

(b) Mandermatcher



(c) Andenne

(d) Julienne

(e) Spy

(f) Contrôle

FIGURE 3.2 – Images caractéristiques de chaque source

Chapitre 4

Normalisation des données

La normalisation consiste à réduire les écarts entre les données afin d'avoir un ensemble plus homogène. Dans le cadre de ce travail, cela a consisté à :

1. réduire les écarts de couleurs ;
2. réduire le bruit.

J'ai utilisé ces techniques dans le but d'améliorer les résultats lors de l'étape d'identification de salamandres. La réduction des écarts de couleurs a été appliquée avant l'utilisation de K-Means, tandis que la réduction du bruit intervient après l'utilisation des composantes connexes.

4.1 Égalisation d'histogrammes

L'algorithme de K-Means est très influencé par l'image qui lui est donné en input, si cette image est de mauvaise qualité ou possède beaucoup de bruit, il y a un risque que le regroupement par couleur donne de mauvais résultats. C'est particulièrement vrai vu que la plupart de photos de salamandres sont prises la nuit, lorsque la luminosité et le contraste ne sont pas optimaux.

Une technique utilisée pour améliorer la luminosité et le contraste s'appelle l'**égalisation d'histogrammes**. Elle est utilisée avec des images en noir et blanc. Concrètement, cette technique ajuste l'échelle de gris de l'image de sorte que l'histogramme des niveaux de gris de l'image d'entrée soit mis en relation avec un histogramme uniformisé (cfr. 4.1) .

Pour l'implémentation de cette technique dans le projet, j'ai utilisé la fonction equalizeHist() [1] de la librairie logicielle *open-cv*[1].

1. https://docs.opencv.org/4.5.3/d6/dc7/group___imgproc___hist.html

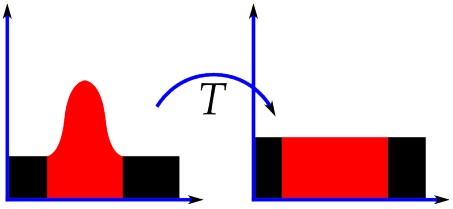


FIGURE 4.1 – Égalisation d'histogrammes

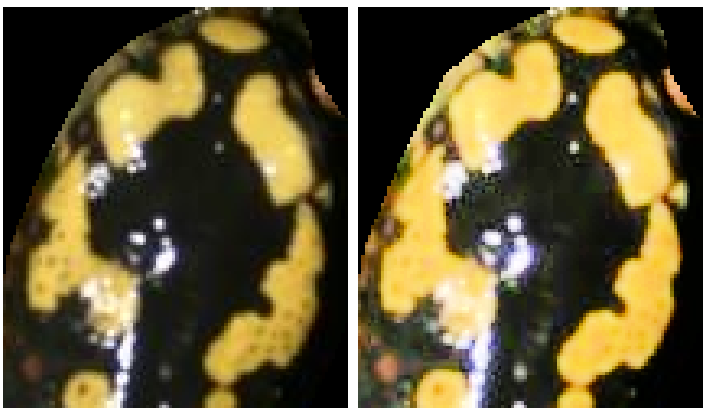
L'algorithme fonctionne de la manière suivante comme expliqué par la documentation d'*open-cv* :

```
equalizeHist(image_source) -> image_destination
```

avec `image source` qui est une image possédant un seul canal de couleur 8-bit, et `image de destination` qui est de la même taille que `image source`.

1. Calcul de l'histogramme H pour la source
2. Normalisation de l'histogramme pour que la somme des bins de l'histogramme soit de 255.
3. Calcul de l'intégrale de l'histogramme : $H'_i : \sum_{0 \leq j < i} H(j)$
4. Transformation de l'image en utilisant H' comme une table de consultation : $dst(x, y) = H'(src(x, y))$

L'utilisation de cette technique est illustrée sur la figure 4.2. Le contraste entre les couleurs est augmenté. On peut voir que des petites taches noires dans le jaune disparaissent après normalisation tandis que toutes les couleurs sont accentuées.



(a) Non normalisée (b) Normalisée

FIGURE 4.2 – Influence de la normalisation

4.2 Érosion-Dilatation

En résumé, l'érosion d'une image consiste à réduire sa taille, ce qui a pour effet de supprimer les petites structures figurant dans celle-ci. La dilatation a l'effet inverse, à savoir augmenter la taille de l'image et donc donne plus d'importance aux éléments conséquents. Ce qui est appelé « *ouverture* » est l'érosion suivie de la dilatation. L'ouverture a comme effet principal de retirer le bruit constitué par de petits points de détail et d'accentuer les parties importantes de l'image [27].

J'ai utilisé la technique de l'ouverture afin de filtrer le bruit des images causé par K-Means et ne retenir que l'essentiel. Pour l'implémentation dans le projet, j'ai utilisé la méthode morphologyEx() d'*open-cv*[2].

Érosion La fonction d'érosion est défini comme suit dans la documentation :

La fonction érode l'image source en utilisant l'élément structurant spécifié qui détermine la forme d'un voisinage de pixels sur lequel le minimum est pris :

$$\mathrm{dst}(x,y) = \min_{(x',y'):\mathrm{element}(x',y')\neq 0} \mathrm{src}(x+x', y+y') \tag{4.1}$$

Dilatation La fonction de dilatation est défini de manière similaire à l'érosion.

$$\mathrm{dst}(x,y) = \max_{(x',y'):\mathrm{element}(x',y')\neq 0} \mathrm{src}(x+x', y+y') \tag{4.2}$$

Le paramètre principal permettant de contrôler l'effet de l'érosion/dilatation est la taille de l'élément structurant, également appelé *kernel*. Les tailles standards de *kernel* (k) sont généralement comprises dans $[3, 5, 7, \ldots, 2n+1], n \in N$ (fig. 4.3).

(a) Originale (b) Ouverture $k = 3$ (c) Ouverture $k = 5$ (d) Ouverture $k = 7$

FIGURE 4.3 – Impact des différentes tailles de *kernel*

2. https://docs.opencv.org/3.4/d4/d86/group___imgproc___filter.html#ga67493776e3ad1a3df63883829375201f

Chapitre 5

Segmentation des images

Dans ce chapitre, je tâcherai de détailler les deux étapes de préparation des photos en vue de leur comparaison : la segmentation des **contours** et des **couleurs**.

La segmentation des contours permettra de simplifier l'image en ne retenant que le corps de la salamandre tandis que la segmentation des couleurs permettra de ne sélectionner que les taches, ce qui sera utile pour la suite.

En effet, si l'on souhaite pouvoir comparer efficacement deux individus et déterminer leur identité, il faut diminuer le plus possible les risques que notre algorithme se trompe, afin d'améliorer son efficacité. Dans ce but, il est nécessaire de nettoyer les images d'entrée au maximum des bruits parasites qui pourraient être confondus par la suite pour de véritables taches de salamandres, d'où l'utilisation de la segmentation des contours.

Plusieurs choix s'offrent à nous. Il y a par exemple une approche basé sur l'utilisation de réseaux de neurones convolutifs [28]. Il existe également la méthode K-Means, qui est une méthode plus simple de vision par ordinateur permettant de segmenter l'image en fonction des couleurs. C'est vers celle-ci que je me suis tournée.

Après avoir essayé plusieurs paramètres, je me suis rendu compte qu'il était quasiment impossible de séparer visuellement l'animal du fond de l'image simplement en utilisant cette technique. Bien qu'une salamandre soit intuitivement simple à distinguer pour un humain, la tâche est bien plus difficile pour un ordinateur. En fonction de la luminosité, du contraste, de l'angle de prise de vue et de la position des sources de lumières sur les photos prises, du bruit peut être généré sur l'image. Sur la figure 5.1, on peut voir un exemple d'une image segmentée avec deux grappes de couleurs, noir et blanc. Les taches sont bien définies nonobstant la grande quantité de bruit présent à cause de la végétation aux alentours. En l'état,

FIGURE 5.1 – K-Means appliqué sur une image du dataset de LLN (CIELAB, $n = 2, c = [1, 2]$)

il est presque impossible de reconnaitre automatiquement la salamandre à partir de cette image.

Plutôt que de détecter les taches directement dans l'image, il peut être intéressant de d'abord détecter le corps de la salamandre, ce qui permet de restreindre l'application de K-Means à des zones de l'image où on est sûr que les taches se trouvent. Il s'agit de l'objectif de la segmentation d'images. Il existe plusieurs modèles de deep-learning spécialisés dans la segmentation d'images. Il y a par exemple l'algorithme *YOLO* permettant de classer des éléments dans une image rapidement, que d'autres étudiants ont d'ailleurs utilisé pour segmenter des images contenant des insectes [5, 21]. Il y a également *RetinaNet*, qui a par exemple été utilisé par pour créer des *bounding boxes* contenant le corps de girafes permettant de focaliser l'algorithme d'identification sur leurs taches [17].

Pour ma part, j'ai fait le choix du réseau **U-Net** pour plusieurs raisons [23].

Premièrement, U-Net est un algorithme populaire et reconnu aujourd'hui comme étant un modèle efficace de segmentation d'images par la communauté scientifique.

Deuxièmement, la segmentation des données dans ce cas-ci est plutôt simple car j'ai fait l'hypothèse au départ que les photos ne comporteront jamais plus d'une seule salamandre à la fois. Et comme on ne recherche que le corps, il n'est pas nécessaire d'utiliser un algorithme multi-classes, bien que U-Net en soit également capable avec quelques modifications.

Troisièmement, le jeu de données que je possède étant très petit (< 1000 images), l'utilisation de modèles traditionnels est déconseillé car il y a un risque de suradaptation (*overfitting*). Le modèle U-Net est bien adapté à ce genre de petits jeu de données, les créateurs originaux utilisant eux-mêmes un très petit jeu de données pour leurs expérimentations (< 50) et obtenant de bons résultats. Un modèle U-Net présente en outre l'avantage d'être rapide à exécuter une fois entrainé. Ceci serait particulièrement bienvenu lors d'un éventuel test en situation réelle.

Structure du chapitre Tout d'abord, la section 5.1 traitera des réseaux de neurones convolutifs en général. J'expliquerai comment ils fonctionnent et comment ils peuvent être utilisé pour réaliser des segmentations d'images.

Ensuite, la section suivante 5.2 présentera une application concrète des réseaux de neurones convolutifs au détourage de salamandres avec l'utilisation du réseau *U-Net*. Il s'agit de la première étape de l'application de l'algorithme de détection de salamandres. J'expliquerai quel est son principe général, ce qu'il apporte en particulier dans ce cas-ci et quelles sont les éventuelles contraintes qu'il entraîne.

Enfin, dans la continuité des étapes de fonctionnement de l'application, la section 5.3 discutera de la segmentation par couleurs des images précédemment segmentées par U-Net. Je justifierai le choix de K-Means et décrirai les paramètres choisis pour son utilisation.

5.1 Réseaux de neurones

5.1.1 Réseaux de neurones convolutifs (CNN)

Un *Réseau de Neurones Convolutifs* ou « *Convolutional Neural Networks* »(*CNN*) est un type de réseaux de neurones artificiels (*ANN*). Les CNN ont été développés en premier lieu pour traiter des données plus conséquentes que celles traitées habituellement par les ANN, typiquement des images. Encoder des caractéristiques spécifiques d'une image dans un réseau classique requiert un grand nombre de paramètres par couche, ce qui est impossible pour des images de plus grande taille ou en couleurs. Augmenter le nombre de couches cachées du réseau n'aide pas à améliorer la situation car cela contribue à augmenter fortement le risque de **surajustement** ("*overfitting*").

Les neurones d'un CNN agissent de manière similaire à ceux de réseaux classiques. Ils peuvent recevoir une ou plusieurs entrées et produire une ou plusieurs sorties. Ils possèdent une fonction d'optimisation ainsi qu'une fonction de perte.

Ce qui diffère par rapport à un ANN classique, c'est l'organisation des couches et des neurones dans ces couches. Trois couches principales sont ajoutées pour créer une architecture de CNN. Il y a les couches **convolutives**, les couches de ***pooling layers*** et les ***fully-connected layers***. Lorsque ces couches sont superposées, une architecture de CNN est créée. Chaque couche a son but spécifique, un réseau de neurones convolutifs typique est composé de quatre couches clés [18].

- La **Couche d'entrée** du réseau, contient les pixels des images données en entrée.
- La **Couche convolutive** est une couche dont les neurones sont faiblement connectés. La couche possède un certains nombre de neurones internes "cachés" connectés aux neurones d'entrée et de sortie. Les neurones d'une colonne cachée ne sont pas forcément connectés à ceux de la colonne suivante. Le but recherché est de diminuer le nombre de paramètres utilisés tout en favorisant l'apprentissage de motifs récurrents, ce qui permet de diminuer le phénomène de surajustement.
- La **Couche de mise en commun** effectue un échantillonage de la sortie de la couche convolutive en ne gardant qu'une partie des paramètres.
- Il y a finalement une suite de **Couches complètement connectées** qui vont remplir la même fonction que pour des réseaux plus traditionnels, à savoir modifier le poids des neurones afin d'augmenter le score global du réseau.

La caractéristique principale d'un CNN est donc la diminution de paramètres stratégiques afin de retirer des motifs récurrents sur des grosses quantités de données. Le but final est de diminuer le surajustement inhérent aux grands réseaux de neurones.

Les réseaux convolutifs ont été créés initialement pour des problèmes de classification, mais d'autres problèmes pouvant être résolus de cette façon ont émergés, notamment les problèmes de segmentation d'images.

5.1.2 Modèles de CNN appliqués à la segmentation d'images

Le problème de la segmentation d'images consiste à grouper les pixels d'une image en fonction d'un critère défini. Les pixels sont alors regroupés en régions, parfois disjointes, qui partitionnent l'image. Ce procédé a été très étudié dans le domaine médical, par exemple pour identifier des tumeurs ou des anomalies sur des radiographies.

La segmentation d'une image peut être réalisée à l'aide de procédés plus classiques de vision par ordinateur. Il y a l'approche par région, avec par exemple

l'algorithme de « *region-growing* », l'approche des contours ou l'approche de la classification basé sur l'intensité des pixels.

La segmentation d'image peut également être faite automatiquement en apprentissage profond via des réseaux de neurones convolutifs. Différentes architectures ont été créées au fil des années. La plus connue étant l'architecture U-Net que j'ai par ailleurs utilisée (cfr. Chapitre 5) [23].

Architecture de U-Net

L'idée derrière le fonctionnement de U-Net est de traiter l'image à segmenter en deux temps.

Dans un premier temps on cherche à trouver des motifs récurrents de plus en plus petits en faisant passer l'image dans un réseau dit « *contractant* ». Cette partie du réseau est composée d'une succession de couches convolutives et de « *max pooling layers* » qui vont progressivement diminuer la résolution de l'image. Le nombre de pixels par canal de l'image diminue, mais le nombre de canaux augmente.

Dans un second temps l'image qui a désormais un nombre très grand de canaux mais très petit de pixels, va subir le traitement opposé. L'image va passer par un réseau *d'expansion* où le nombre de canaux va diminuer tandis que la résolution augmentera. Le réseau d'expansion est composé d'une succession de couches de suréchantillonnage, qui remplacent les couches de mise en commun, et de couches convolutives.

Au final, une nouvelle image est produite à partir du réseau. Elle possède la même résolution que l'image originale mais pas forcément le même nombre de canaux. De manière générale, il est préférable d'obtenir une image avec un seul canal de couleur, donc en noir et blanc, pour garantir des bonnes performances de calcul.

Contraintes sur l'image d'entrée Dans [23], la résolution utilisée était de 512×512, mais n'importe quelle résolution est théoriquement acceptée. Cependant dans un soucis de normalisation, il est recommandé que toutes les images possèdent la même résolution. Plus la résolution est élevée, plus le niveau de détails capturés sera grand, mais également plus le coût d'entrainement du modèle sera élevé. Un modèle dont la résolution est élevée demandera plus de données pour son entrainement afin d'éviter un phénomène de sousajustement contrairement à un modèle avec un résolution moindre.

5.2 Première Segmentation – U-Net

Lors de la première segmentation avec U-Net, j'ai d'abord repris le modèle proposé sur le site de Keras, que j'utilise notamment pour mon implémentation en Python [6]. Le modèle complet est montré dans la table B dans les appendices.

5.2.1 Paramètres du modèle utilisés

Afin de choisir les paramètres optimaux de U-Net, j'ai testé plusieurs modèles différents afin de les comparer. Voici les principaux paramètres utilisés :

Fonction de perte La fonction de perte définit l'écart entre la prédiction et la réalité. J'ai utilisé la fonction "Sparse Categorical Crossentropy" (**SCC**) de Keras. Elle est définie par :

$$Loss = -\sum_{i=1}^{\text{taille sortie}} y_i . \log \bar{y}_i \tag{5.1}$$

où $\bar{y}_i$ est la ième valeur scalaire dans le modèle de sortie, y_i est la valeur à atteindre correspondante, et `taille sortie` est le nombre de valeurs scalaires dans le modèle de sortie. Cette fonction permet de mesurer l'écart entre deux distributions de probabilité discrètes.

Optimiseur L'optimiseur minimise la fonction de perte afin de maximiser son efficacité. J'ai choisi l'optimiseur Adam, qui est une amélioration des optimisateurs RMSProp et AdaGrad, car il est populaire dans la littérature [32]. Il est possible de contrôler en partie son comportement par le choix du learning rate λ, qui par défaut est fixé à 0.001. Un learning rate plus grand permet d'apprendre plus rapidement, au risque d'un effet de sur-ajustement.

Métriques Afin d'évaluer le modèle, j'ai principalement utilisé deux métriques : la métrique du loss [1] et de l'accuracy. La métrique de l'accuracy, telle qu'elle est définie par le framework *Keras*, calcule la fréquence à laquelle les prédictions sont égales aux étiquettes. Le modèle s'optimise automatiquement par rapport au loss, j'utilise donc l'accuracy pour comparer les modèles.

1. Correspond donc à **SCC**

Taille des photos Comme dit précédemment, dans l'architecture originale de U-Net, des images de 512×512 étaient utilisées mais plusieurs tailles sont envisageables. Ainsi, j'ai également utilisé des photos de 256×256. Toutes les images en entrée utilisées par U-Net sont mises à cette échelle automatiquement.

Ensembles de validation, de test et d'entrainement L'ensemble de validation comprend 20% de l'ensemble total de photos, l'ensemble de test comprend un autre 20% tandis que l'ensemble d'entrainement comprend 60% de celles-ci.

Vitesse d'apprentissage La vitesse d'apprentissage étant limitée par l'appareil utilisé pour entrainer les modèles, deux facteurs permettraient de la contrôler : le nombre d'epochs et la batch size. Le nombre d'epochs permet de régler le temps d'apprentissage du réseau. S'il est trop grand, il y a un risque de sur-apprentissage. La batch size est le nombre d'échantillons transmis au réseau en une seule fois. Généralement, j'ai réglé le nombre d'epochs à 5 avec un nombre d'étapes par epoch à 200 et la batch size à un nombre compris entre 5 et 20, généralement 15.

5.2.2 Évaluation du modèle

Au début de l'expérimentation, j'ai réalisé des tests de segmentation à partir d'un ensemble de données restreint de 600 photos. Ensuite, après avoir ajouté de nouvelles photos, j'ai de nouveau créé des modèles avec l'ensemble de données complet, à savoir 836 photos. Avec l'augmentation de données, on obtient respectivement 2468 et 6212 photos pour le petit et le grand échantillon. Je constate une vitesse d'apprentissage fortement accrue pour le grand échantillon, dont la *loss* pour l'ensemble de validation (val loss) atteint rapidement un niveau bas après quelques epochs, contrairement au petit échantillon.

Les paramètres du modèle final sont présentés dans le tableau 5.1. Concernant l'ensemble de données utilisé, j'ai pris le grand ensemble des 836 photos. Pour rappel, il est composé de 313 photos de LLN, 47 d'Andenne, 393 de Mandermatcher, 7 des étangs Julienne, 8 de Spy et 68 photos de contrôle ne comprenant pas de salamandre.

En ce qui concerne l'augmentation de données, il est admis qu'un réseau de neurones sera d'autant plus efficace qu'il repose sur un grand nombre de données. Ici, l'augmentation de données fait que pour chaque photo (sauf les photos de contrôle), il y a les 4 orientations possibles $(0°, 90°, 180°, 270°)$, une inversion de l'axe horizontal, vertical, les deux à la fois, et enfin un zoom aux proportions aléatoires. Pour chaque photo, il y a donc 8 différentes versions.

Fct perte	optimiseur	λ	nb photos	taille image
SCC [2]	Adam	0.001	6212	256x256

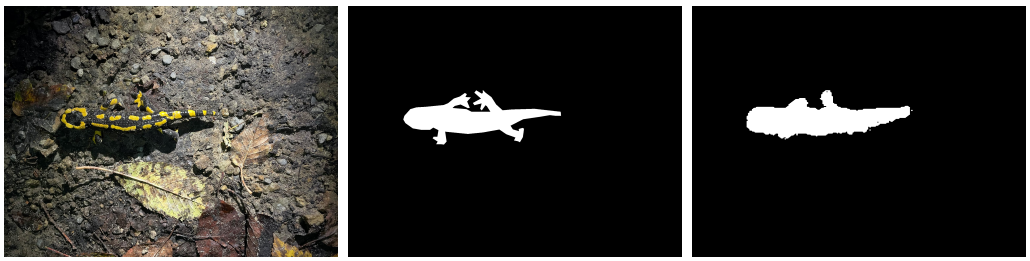
TABLE 5.1 – Modèle final

epoch	loss	accuracy	val loss	val accuracy
1	0.182	0.893	1.515	1
2	0.086	0.892	0.954	1
3	0.077	0.888	0.086	0.911
4	0.066	0.888	0.063	0.902
5	0.060	0.885	0.065	0.886

TABLE 5.2 – Résultats finaux par epoch

Un extrait des résultats du modèle figure dans la table 5.2. Le modèle a été optimisé selon le loss de l'ensemble d'entrainement, et possède au final un loss de 0.065 et une accuracy de $88,6\%$. On peut remarquer un élément étrange dans ce tableau. À la première epoch, j'ai obtenu une accuracy de 1. Cela peut être expliqué par un sous-ajustement du modèle et l'utilisation des photos de contrôle, dont le masque est forcément noir.

Des exemples de prédictions sont présentés dans les figures 5.2 et 5.3. On peut voir que le réseau arrive à reconnaitre le corps correctement, en excluant certains membres comme des pattes se trouvant un peu trop dans l'ombre (fig. 5.2a). Dans d'autres cas, le réseau réalise même un meilleur résultat que le découpage manuel (fig. 5.3c).



(a) Image de référence (b) Masque de référence (c) Masque prédit

FIGURE 5.2 – Comparaison du masque de référence et la prédiction (LLN)

Étant donné que les taches du dos des salamandres sont généralement préférées pour la comparaison, il s'agit de bons résultats. Même si le réseau a un peu plus de mal avec les photos un peu moins qualitatives (fig. 5.2a, 5.4), il permet de nettoyer les images en enlevant une majeure partie du bruit et en gardant l'essentiel du

(a) Image de référence (b) Masque de référence (c) Masque prédit

FIGURE 5.3 – Comparaison du masque de référence et la prédiction (Mandermatcher)

(a) Image de référence (b) Masque de référence (c) Masque prédit

FIGURE 5.4 – La prédiction n'est pas toujours tout à fait correcte

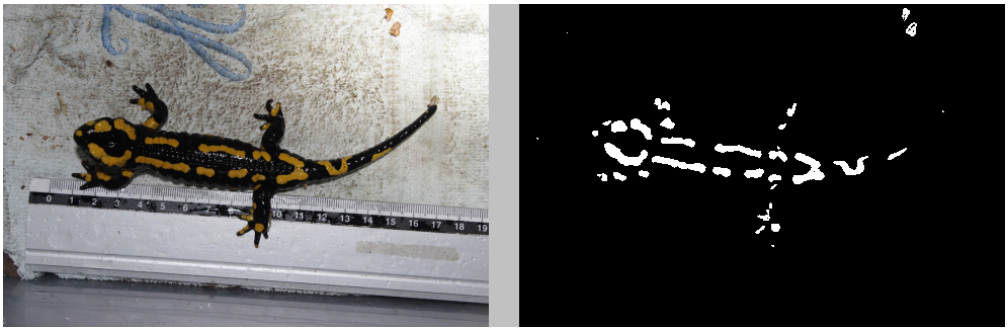
corps.

5.3 Seconde Segmentation – k-means

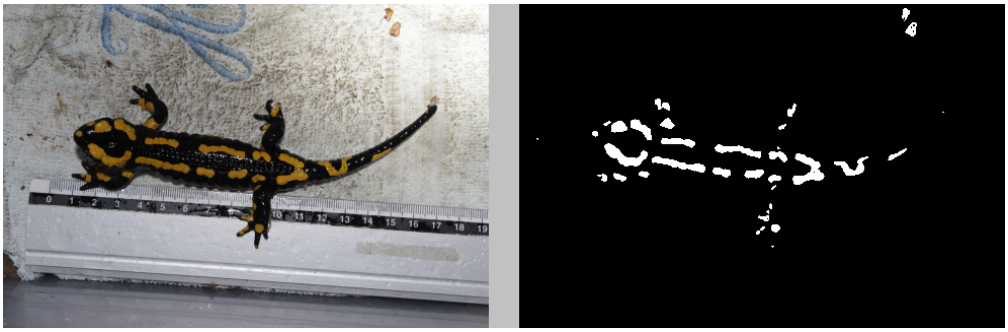
Si le but recherché par la segmentation par U-Net était de détecter le corps de la salamandre, la segmentation par K-Means a pour objectif de capturer les taches. L'intuition du choix de K-Means est que les taches sont facilement distinguables du corps des salamandres grâce à leur couleur caractéristique. Comme expliqué dans le Chapitre 2, K-Means regroupe les couleurs d'une image en clusters en fonction de leur similarité.

L'intuition est que si on utilise un faible nombre de labels à prédire, K-Means groupera les taches et le fond de l'image dans des clusters différents. Ensuite, à partir des prédictions de K-Means, on créera de nouveaux masques en noir et blanc, avec le blanc qui représentera les taches de la salamandre.

De manière pratique, j'ai utilisé l'implémentation de K-Means de Sklearn [20]. K-Means est paramétrable à l'aide du nombre de classes prédites, ainsi que deux paramètres supplémentaires : `n_init` et `max_iter`. Ceux-ci permettent



(a) CIELAB, $n = 2, c = [1, 2]$



(b) YCbCr , $n = 2, c = [1, 2]$

FIGURE 5.5 – Comparaison d'images segmentées par K-Means

respectivement d'augmenter le nombre de fois que l'algorithme sera exécuté avec différentes graines de centroïdes, et le nombre maximal d'itérations pour une seule exécution. Je les ai fixé à 40 et 500 (par défaut 10 et 300) afin d'augmenter la qualité des prédictions.

K-Means est également influencé par les images en entrée, en particulier par leur colorimétrie. Les images initialement en RGB peuvent changer d'espace de couleur (cfr section 2.1). Il est également possible de ne sélectionner que certains canaux de couleurs. Dans notre cas, on chercherait à faire ressortir le jaune par rapport au noir du corps.

J'ai cherché à sélectionner la meilleure combinaison d'espace de couleur et de canaux de couleurs permettant de ne garder que les taches. Deux candidats sont sortis du lot : CIELAB ($c = [1, 2], n = 2$) et YCbCr ($c = [1, 2], n = 2$) (fig. 5.5). Ceux-ci ne nécessitent que 2 clusters pour faire ressortir les taches. Ils présentent des résultats similaires voir quasiment identiques. Cependant, le modèle de CIELAB conserve mieux les taches de couleur jaune. C'est celui-ci que je retiendrai pour la suite.

L'utilisation d'un nombre de clusters plus grand ne semble pas mieux capturer

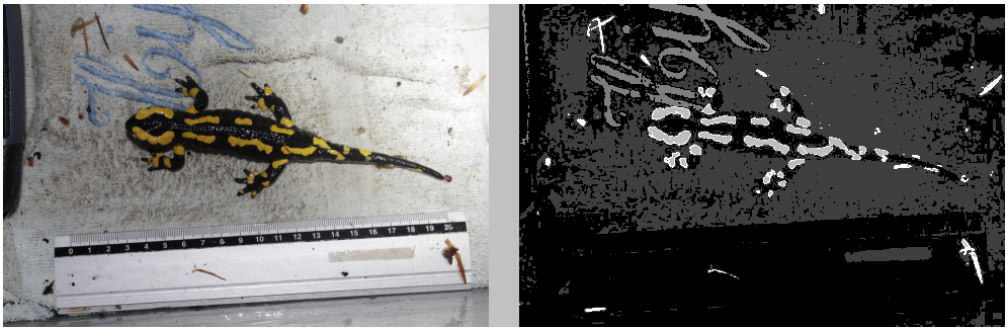


FIGURE 5.6 – L'utilisation d'un n plus grand semble aggraver les résultats (CIELAB, $n = 5, c = [1, 2]$)

les taches, au contraire cela semble aggraver la génération du masque. Sur la Figure 5.6, on peut constater qu'avec 5 clusters, on sélectionne les tâches mais également d'autres éléments de l'arrière plan, comme par exemple des tâches du sol de couleur jaune. Le nombre de clusters maximum qui conserve une bonne représentation des taches semble être de 3.

5.4 Conclusion

Dans ce chapitre, j'ai présenté les différentes techniques de segmentation et justifié leur utilité.

U-Net est un réseau de neurones convolutifs qui permet à partir d'une photo de salamandre, de ne garder que l'animal en ignorant le reste de l'image, ce qui permet de retirer une part importante du bruit. Le modèle entrainé de U-Net a une efficacité de 89%, le corps ressort bien malgré que certaines pattes ne sont parfois pas prises en compte. Le réseau arrive même à faire mieux que l'humain dans certaines situations.

La segmentation des couleurs par K-Means permet de ne garder que les taches de la salamandre. L'algorithme regroupe les couleurs par similarité. Il fonctionne bien dans ce contexte grâce à la distinction entre le jaune des taches et du noir du corps. Un grand nombre de clusters n'est pas utile pour distinguer les taches, au contraire il vaut mieux utiliser un petit nombre ($1 < n < 4$). Pour un résultat optimal, il faut convertir les images dans l'espace de couleur CIELAB en ne gardant que les canaux 1 et 2.

Chapitre 6

Identification des individus

Maintenant que la méthode de préparation et de nettoyage des données a été présentée, il est temps de détailler la méthode de comparaison en elle-même. Je commencerai par détailler la méthode d'évaluation des algorithmes de comparaison dans la Section 6.1. J'expliquerai en outre les critères de choix des algorithmes de comparaison ainsi que les différentes métriques utilisées.

Enfin dans la Section 6.2 j'expliquerai le fonctionnement de la méthode dit des « *centroïdes de blobs* ». Les avantages et inconvénients possibles de la méthode seront exposés ainsi que les différents paramètres influençant l'algorithme.

6.1 Méthode d'évaluation

Afin de pouvoir évaluer la performance de l'algorithme d'identification, il est crucial de déterminer les paramètres à surveiller. Le choix de ces paramètres dépend de notre objectif final.

Dans notre cas, on souhaite comparer deux images de salamandres, et déterminer s'il s'agit du même individu ou non. L'algorithme peut soit prédire correctement, soit se tromper. Il s'agit donc d'un problème de classification binaire.

L'algorithme parfait Définissons l'algorithme parfait dont on souhaite s'approcher. Son taux d'erreur de prédiction doit être le plus bas possible. On souhaite particulièrement que les individus identiques soient correctement identifiés. En effet, il existe un déséquilibre naturel entre les classes : il y aura naturellement beaucoup plus de comparaisons de salamandres différentes qu'identiques. Un classificateur

naïf pourrait en effet classer toutes les comparaisons comme des individus différents et obtenir un taux de réussite élevé.

Même s'il s'agit d'un point dont je me suis moins préoccupé, l'algorithme parfait devrait en outre être rapide dans l'absolu. Si on considère une mise à l'échelle d'une application utilisant l'algorithme, il est crucial que la comparaison d'images, l'opération la plus lente de l'architecture lorsque le volume de données augmente, soit la plus rapide possible.

Afin d'évaluer plusieurs algorithmes et déterminer celui le plus proche de notre définition d'*algorithme parfait*, il est nécessaire de définir des métriques qui permettront de définir des points de comparaison.

6.1.1 Métriques

Les métriques, au sens large du terme, permettent d'évaluer la qualité d'algorithmes ou de modèles mathématiques. De nombreuses métriques ont été conçues au fil des années. Les métriques que je vais présenter ici sont des métriques habituellement utilisées pour la classification multi-classes en apprentissage automatique [14]. Elles seront utiles ici puisque nous avons un problème de classification binaire.

Matrice de confusion

La matrice de confusion est un double tableau qui enregistre le nombre d'occurrences entre deux évaluations, la classification véritable et la classification prédite. Pour être consistent tout le long de ce travail et avec ce qui est généralement fait dans la littérature, la table de vérité se trouve à gauche pour les lignes et la table de prédiction se trouve en haut pour les colonnes (fig 6.1).

Precision et Recall

Les deux métriques suivantes sont utiles pour analyser certaines parties de la matrice de confusion. Même si elles sont écrites pour correspondre aux cas positifs, ces métriques peuvent être appliquées aux cas négatifs de manière similaire.

Precision La « *precision* », est le nombre d'éléments Vrais Positifs (TP) divisé par le nombre d'éléments prédits comme étant positifs. Le nombre total d'éléments prédits comme positifs est donc composé des éléments correctement prédits positifs

		Table de prédiction	
		Positif (1)	Négatif (0)
Table de vérité	Positif (1)	Vrais Positifs (TP)	Faux Négatifs (FN)
	Négatif (0)	Faux Positifs (FP)	Vrais Négatifs (TN)

FIGURE 6.1 – Matrice de confusion

(TP) et des éléments incorrectement prédits positifs (FP).

$$\frac{TP}{TP + FP} \tag{6.1}$$

La précision désigne donc la propension du modèle à correctement prédire les éléments positifs. Plus cette métrique est proche de 1, mieux le modèle prédit correctement les cas positifs, et inversement si elle est proche de 0.

Recall Le « *recall* », est la métrique sœur de la *precision*. Elle est le nombre d'éléments vrais classés correctement (TP) divisé par le nombre total d'éléments classés comme étant "vrai". Le nombre total d'éléments classés comme "vrai" est composé des éléments vrais classés correctement (TP) et des éléments vrais classés incorrectement (FN).

$$\frac{TP}{TP + FN} \tag{6.2}$$

La métrique du *recall* mesure la capacité du modèle à trouver la classe de tous les éléments positifs dans l'ensemble des données. De la même façon que la *precision*, plus le modèle est proche de 1, plus il est capable de classer les éléments positifs.

Accuracy

L' « *accuracy* », est directement calculée à partir de la matrice de confusion, c'est certainement la métrique la plus utile pour comparer plusieurs modèles.

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{6.3}$$

Pour calculer l'*accuracy*, on calcule la division entre la somme des éléments correctement prédits, et la somme de toutes les classes prédites. La métrique de la

accuracy mesure la probabilité que les prédictions du modèle soient correctes. Elle est également contenue dans l'intervalle $[0, 1]$. Plus cette métrique s'approche de 1, mieux le modèle prédit correctement la classe d'éléments.

La métrique d'accuracy possède le défaut de ne pas tenir compte du déséquilibre des classes. Elle considère que chaque unité de l'ensemble des données a le même poids, et qu'elles contribuent toutes de la même façon à la mesure de l'accuracy. Ce n'est pas toujours le cas lorsque certaines classes possèdent un grand nombre de données par rapport à d'autres. C'est pour cette raison qu'il existe une version modifiée de la métrique d'accuracy, la « *balanced accuracy* ».

Balanced Accuracy

La métrique de la « *balanced accuracy* » est similaire à l'*accuracy*, mais elle possède l'avantage de tenir compte du déséquilibre des classes.

La formule est également un peu différente :

$$\frac{\frac{TP}{TP+FN} + \frac{TN}{TN+FP}}{2} \tag{6.4}$$

En d'autres termes, cette formule constitue la moyenne de la métrique de **recall** pour chaque classe. Chaque métrique de recall répond à la question d'à quel point un individu d'une donnée est classé correctement. La moyenne des deux recall donne la précision moyenne du modèle par classe. Si les classes sont faiblement déséquilibrées, la valeur de la « *balanced accuracy* » sera très proche de la précision.

6.2 Méthode des blobs et centroïdes

Dans la suite de ce chapitre, je vais détailler la méthode de comparaison principale utilisée dans ce travail. Il s'agit de la méthode dite des « ***blobs et centroïdes*** ». L'intuition derrière cette méthode est que les taches à comparer peuvent être abstraites comme des formes, ou blobs, et que la plupart des taches possèdent des formes simples, ovales, et homogènes.

Sous ces hypothèses, on peut se permettre de comparer les blobs par un point unique, leur centre. Cependant pour servir de point de comparaison, ce point doit être suffisamment représentatif de la tache. L'hypothèse de simplicité de la forme des taches est donc primordiale pour l'efficacité de l'algorithme.

Les blobs des taches sont calculé à l'aide de la méthode des composantes connexes. La configuration de cette méthode dans ce projet spécifique sera montrée

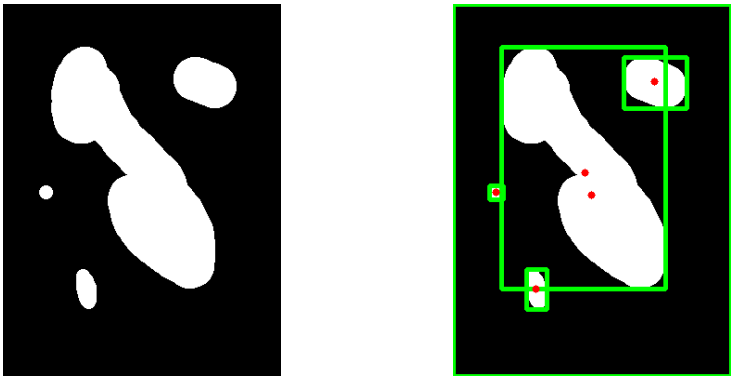


FIGURE 6.2 – Exemple de détection de composantes connexes par *open-cv*. Les rectangles verts sont les bounding boxes des CC tandis que les points rouges sont leurs centroïdes. Le fond de l'image est lui-même un CC.

dans la Sous-section 6.2.1. Une analyse des taches détectées sur les jeux de données à l'aide de cette méthode sera également présentée dans la Sous-section 6.2.2.

6.2.1 Composantes connexes

Afin d'identifier les blobs des taches de salamandres, j'ai utilisé la technique des **composantes connexes** (abbr. **CC**) présentée dans le Chapitre 2 à la Section 2.3.

De manière pratique, j'ai utilisé la bibliothèque *open-cv* et plus particulièrement la méthode `connectedComponentsWithStats` contenu dans le module d'*image processing*. Cette méthode trouve automatiquement les CC composant les blobs (fig. 6.2). Ceux-ci sont composés de blanc tandis que le fond de l'image est composé de noir. Les blobs trouvés par la méthode des CC ne reflètent pas forcément la réalité des taches sur le dos de la salamandre, il est possible que :

— des taches aient fusionné ;
— des taches soient vues comme une multitude de petites taches connexes ;
— des taches ne soient pas représentées ;
— du bruit soit pris pour des taches.

La bibliothèque d'*open-cv* permet d'utiliser plusieurs algorithmes de composantes connexes. Pour ce travail, je me suis cantonné aux algorithmes par défaut d'*open-cv* fonctionnant dans la plupart des situations.

L'algorithme de détection de CC ne donne pas beaucoup de possibilités d'altération de son comportement. Toutefois, deux choix principaux de configuration de l'algorithme peuvent être choisis pour ajuster les résultats : la sensibilité de connexité peut être fixé à 4 ou à 8 (cfr fig. 2.2).

Le choix du type de connexité est essentiellement motivé par la qualité de détection des blobs de taches. En effet, pour une comparaison de photographies la plus exacte possible, il faut qu'au préalable la détection des taches soit la plus fidèle possible. Or, la qualité de la détection est le produit des opérations précédentes, à savoir les différentes segmentations et la normalisation des données. Pour cette raison, j'ai réalisé une analyse préliminaire de l'influence des pré-traitements des images et de l'influence de la segmentation par couleur sur les taches détectées par les CC. Le but était de déterminer la meilleure connexité à adopter pour la suite en fonction des résultats précédents.

La segmentation de l'image par U-Net a été écartée de l'analyse car j'ai préféré utiliser les masques produits manuellement lors de la préparation des données. Les parties segmentées par couleur sont celles découpées à partir des masques manuels, car ces derniers sont considérés comme plus proches de la réalité. Cette éviction permet d'éviter d'inclure l'influence d'une mauvaise segmentation par U-Net, ce qui compliquerait fortement le travail d'analyse de l'influence des autres éléments.

6.2.2 Analyse des données de taches des salamandres

J'ai étudié l'influence potentielle :

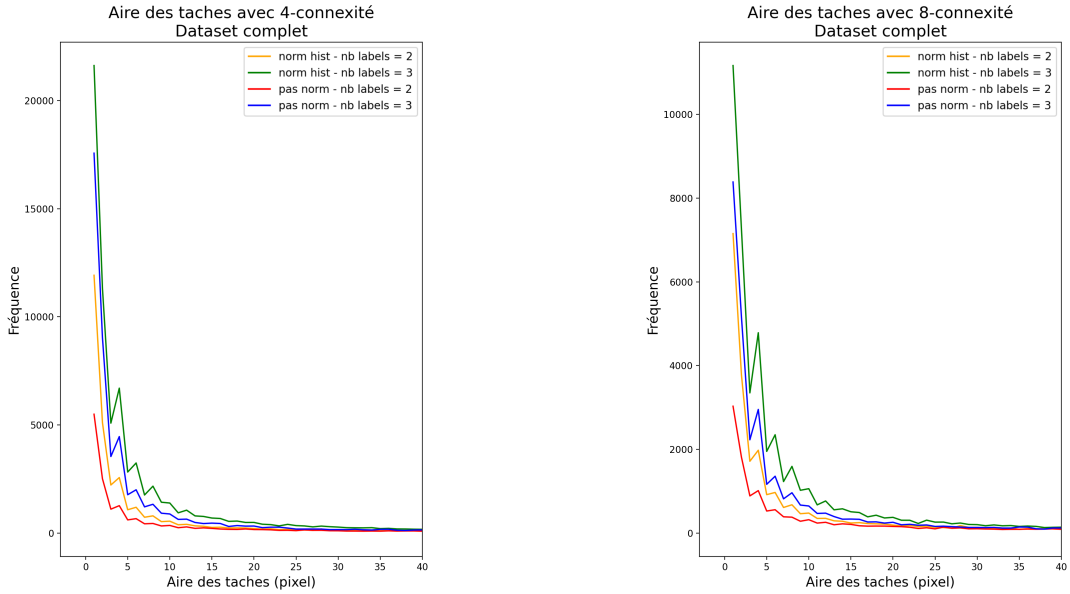
— de la normalisation des images ;
— du nombre de labels calculés par K-Means ;
— du type de connexité utilisé par les composantes connexes ;

sur **l'aire** calculée des taches de salamandres suite à l'opération K-Means.

En effet, j'ai remarqué qu'en général, la plupart des taches ont une taille sous-estimée par rapport à la réalité. La luminosité et le contraste font en sorte que certaines zones de taches jaunes ne sont pas prises en compte. La salamandre possède alors moins de surface de tache que dans la réalité.

L'aire des taches est calculée à l'aide de l'algorithme de composantes connexes. J'ai réalisé des mesures sur la distribution de la fréquence de ces aires en prenant en compte les différents moments statistiques (moyenne, écart-type, médiane, asymétrie, kurtosis).

L'analyse statistique a porté sur la normalisation par égalisation d'histogrammes en comparaison aux images non-normalisées, le nombre de labels calculés par K-Means, de 2 ou 3 labels, le paramètre de connexité de 4 ou de 8, et l'utilisation de la technique de l'érosion-dilatation. Au total, 16 tableaux de moments statistiques ont été générés. L'ensemble de données utilisé comprend les photos de ManderMatcher, celles prises à LLN et enfin celles issues d'Andenne, et ce afin d'avoir une analyse statistique plus générale en incluant même des photos de moins bonnes qualités.



(a) 4-connexité (b) 8-connexité

FIGURE 6.3 – Distribution de l'aire des taches **sans** érosion-dilatation.

Résultats préliminaires

La distribution de l'aire calculée des taches suit globalement une distribution normale avec une déviation standard et de kurtosis très élevée. Comme on pouvait s'y attendre, la 4-connexité sous-estime systématiquement la taille des taches. Je constate un très grand nombre de petites taches de moins de 7 pixels, ce qui est également le cas mais dans une moindre échelle avec la 8-connexité avec une médiane à 16 pixels. La Table 6.1 montre un extrait des mesures pour des images normalisées à l'aide de l'égalisation d'histogrammes pour 2 groupes de couleurs, avec une comparaison entre l'utilisation ou non de l'érosion-dilatation.

La distribution des aires sans érosion-dilatation est illustrée sur la figure 6.3 avec une comparaison entre l'utilisation d'une 4 ou d'une 8-connexité. Le moment de kurtosis est particulièrement intéressant ici car il semble proportionnellement corrélé au nombre de taches minuscules. Du fait d'un nombre important de petites et très grandes taches, la déviation standard et la moyenne sont fortement biaisées. Dans chaque graphique, j'observe également une séparation nette entre les courbes des différentes techniques. On peut voir qu'à la fois l'utilisation de plusieurs labels et la normalisation par égalisation d'histogrammes augmentent chacun significativement le nombre de petites taches. En outre, on peut voir une différence marquée entre

l'utilisation d'une connexité 4 (6.3a) et 8 (6.3b) sur le nombre de petites taches, leur nombre est divisé par 2 avec la 8-connexité.

J'ai également noté certains outliers possédant une aire très grande. La grande taille de certaines taches s'explique par le bruit parfois présent sur certaines photos, provenant par exemple des feuilles d'une couleur proche du jaune qui est confondue avec le jaune des taches. D'autres fois, les taches sont simplement très grandes naturellement.

L'utilisation de l'érosion-dilatation a un impact énorme sur la distribution de l'aire des taches. Pour cette raison, je différencierai son utilisation ou non dans les analyses d'impact des paramètres pour éviter une contamination de la moyenne statistique.

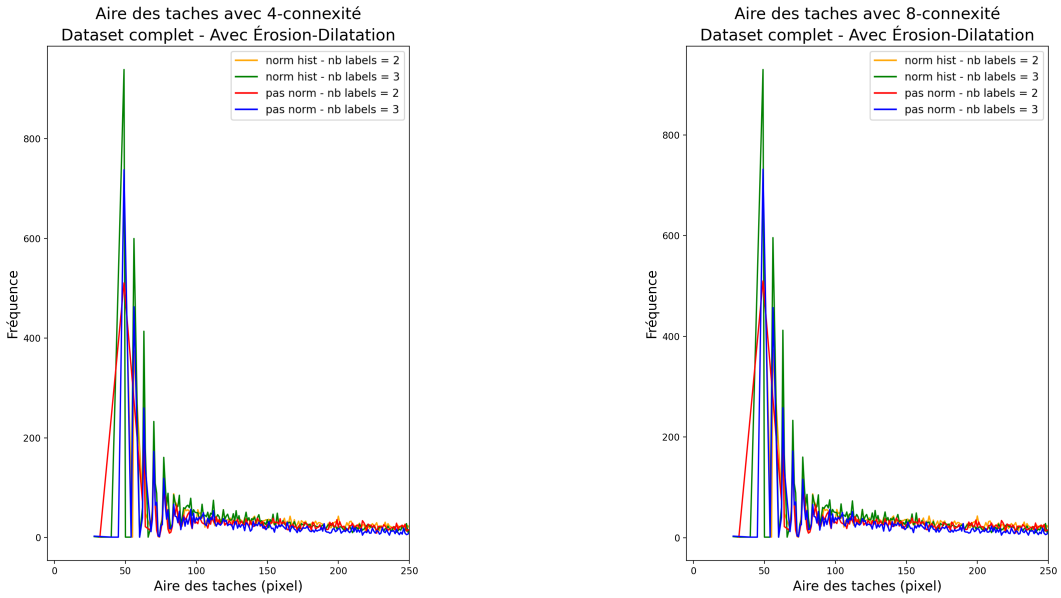
Analyse d'impact des paramètres

Impact du type de connexité J'ai constaté qu'en utilisant la connexité 8 plutôt que la connexité 4, il y a une augmentation moyenne de 30% de l'aire moyenne des taches si on exclut l'utilisation de l'érosion-dilatation. J'ai également noté une diminution de 22% du nombre de taches avec son utilisation. L'intuition serait alors **d'utiliser la 8-connexité** pour avoir des blobs plus représentatifs.

Impact de la normalisation des données J'ai constaté que l'utilisation de l'égalisation d'histogrammes sur les images avait un certain impact sur le nombre de petites taches. L'aire moyenne des taches diminue en moyenne de 19% quand l'égalisation d'histogrammes est utilisée sans érosion-dilatation. En revanche, le nombre de taches augmente de 33%, et la distribution confirme qu'il s'agit majoritairement de petites taches. L'intuition voudrait donc que dans le but de diminuer le nombre de blobs minuscules, il conviendrait d'**éviter** d'utiliser la normalisation par **égalisation d'histogrammes**.

Impact du nombre de labels de K-Means Lorsque le nombre de labels générés par K-Means augmente, l'aire moyenne des taches diminue fortement (71% de diminution en général), et le nombre de taches minuscules augmente de 72%. Afin de diminuer le nombre de petite taches, l'intuition voudrait qu'il faille utiliser un **petit nombre de labels**.

Impact de l'érosion-dilatation L'utilisation de la normalisation par érosion-dilatation ($k = 3$) a un impact élevé sur la détection de taches. Sur le tableau 6.1, on peut voir que le nombre de taches est divisé par 3 avec son utilisation,



(a) 4-connexité (b) 8-connexité

FIGURE 6.4 – Distribution de l'aire des taches **avec** érosion-dilatation.

avec une diminution de l'ordre de 71%. Le taille moyenne augmente également drastiquement avec une augmentation de 250%. Sur la figure 6.4, on peut voir que les distributions des aires n'a rien à voir avec celles de la figure 6.3. Les plus petites taches sont de l'ordre de 40 pixels, avec la médiane aux alentours de 270 pixels. Les différences entre les paramètres sont également beaucoup moins marquées. L'effet de la 8-connexité au lieu de la 4 n'est presque plus perceptible. Cela s'explique par la disparition du bruit, où les taches minuscules étaient généralement connexes.

Conséquences générales

La présence de très petites taches est problématique pour l'identification et la comparaison d'individus pour plusieurs raisons. Si ces minuscules taches sont issues de bruit, elle peuvent impacter négativement l'identification en augmentant la probabilité d'erreur FN et FP (cfr fig 6.1). Comme nous le verrons dans la section qui y est consacrée, le nombre d'histogrammes a un fort impact sur la complexité temporelle de l'algorithme de comparaison. Nous cherchons avant tout un nombre restreint de taches représentatives, qui peuvent être utilisées comme points de référence pour la comparaison.

Sans Erosion-Dilatation				Avec Erosion-Dilatation			
Connexité 4		Connexité 8		Connexité 4		Connexité 8	
μ	998	μ	1211	μ	2990	μ	2999
σ	9894	σ	10899	σ	17007	σ	17032
$\tilde{X}$	7	$\tilde{X}$	16	$\tilde{X}$	273	$\tilde{X}$	274
$Skew[X]$	18	$Skew[X]$	16	$Skew[X]$	10	$Skew[X]$	10
$Kurt[X]$	407	$Kurt[X]$	334	$Kurt[X]$	132	$Kurt[X]$	131
n	49560	n	40874	n	15824	n	15778

TABLE 6.1 – Images normalisées (equalHist), 2 groupes de couleurs

6.2.3 Conclusion

La méthode de découverte de composantes connexes est une méthode efficace afin de détecter les taches, bien qu'elle soit parfois imprécise. Le paramètre important pour régler la configuration de l'algorithme est le choix de la 4 ou de la 8-convexité.

Dans l'analyse des données, il a aussi été montré que les différents paramètres des nombreux algorithmes employés **avant** la comparaison en tant que telle, avaient une forte influence sur les données d'input. Un paramètre ressort plus que les autres : l'**érosion-dilatation**. D'autres intuitions ont également été tirées de cette analyse, comme l'utilisation d'un nombre moindre de labels pour *K-Means* ou de la 8-connexité. Une intuition qui va un peu à contre-courant du consensus de la littérature est également ressortie en l'objet de la normalisation des images. Il semblerait que cette normalisation serait contre-productive par rapport à nos objectifs et qu'il serait préférable de s'en passer.

6.3 Histogrammes polaires

Une fois que les blobs ont été détectés ainsi que les centroïdes de taches, la méthode utilise les histogrammes polaires pour créer des points de comparaison valides. La méthode des **histogrammes polaires**, ou histogrammes log-polaires, consiste à évaluer les distributions de distance et d'angles entre un point et ses points voisins. Elle peut être utilisée pour créer des descripteurs locaux.

Dans le cadre de la comparaison d'images digitales, cette méthode encode la configuration spatiale de nuages de points. Elle repose sur le principe général de contraintes semi-locales, qui est présentée- dans la Sous-section 6.3.1 [25].

Le principe général des histogrammes polaires est présenté dans la Sous-section 6.3.2, tandis que leur utilisation pour la comparaison de taches est expliquée dans

la Sous-section 6.3.3.

Une fois que les histogrammes de comparaison ont été créé, il reste à définir précisément comment les utiliser pour la comparaison d'images. Cela sera le sujet de la Sous-section 6.3.4 où le calcul de comparaison sera détaillé.

Enfin, une brève présentation d'autres méthodes utilisant des histogrammes polaires sera proposée à la fin du chapitre dans la Sous-section 6.3.5, suivie d'une brève conclusion.

6.3.1 Principe général des contraintes semi-locales

Les contraintes semi-locales sont utilisées pour comparer des images à l'aide de vecteurs de points. Ces vecteurs de points sont générés à partir de l'image. Au contraire d'autres algorithmes de comparaison, l'originalité des contraintes semi-locales vient du fait qu'elles utilisent des motifs locaux comme points de comparaison au lieu de motifs globaux. Le principe prend en compte la position des points relatifs par rapport aux autres.

Concrètement Pour chaque élément d'un nuage de points, les p éléments les plus proches sont sélectionnés. Pour améliorer le taux de reconnaissance, on tient également compte d'une contrainte géométrique, l'angle entre les voisins et l'élément que l'on cherche à caractériser. Pour établir la correspondance entre deux points, une partie significative des points voisins et de leurs angles associés doivent correspondre. Comme on suppose que les nuages de points peuvent subir des distorsions via des transformations d'images, on doit tenir compte d'une certaine marge de tolérance lorsque l'on fait correspondre le nombre de voisins et leur angle. On peut voir un exemple de comparaison de points sur la figure 6.5, où deux points et leurs 5 voisins sont comparés. Les angles α_1 et α_2 doivent être égaux sous une certaine marge d'erreur.

L'utilisation de configurations locales au lieu de configurations globales permet de diminuer l'impact d'éléments étrangers et de visibilité partielle de l'image. Par exemple, si on utilise des caractéristiques globales lors de la comparaison de deux images, dont l'une est un fragment de l'autre, l'algorithme aura tendance à considérer qu'il s'agit d'objets différents. La prise en compte de configurations locales permet d'augmenter la probabilité que l'algorithme considère qu'il s'agit d'objets similaires.

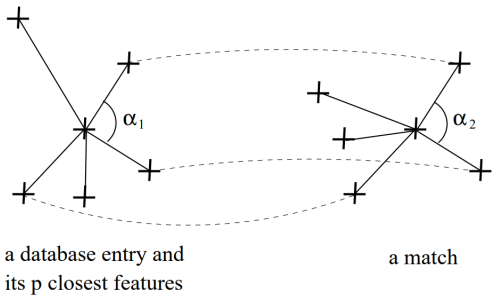


FIGURE 6.5 – Contraintes semi-locales – Les voisins d'un point et les angles doivent correspondre. Note : tous les voisins ne sont pas forcés de correspondre. ([25])

6.3.2 Principe général des histogrammes polaires

Les histogrammes polaires sont une application concrète des contraintes semi-locales.

Un histogramme polaire tient compte des positions relatives de points en rapport à un point donné, souvent recentré en $(0; 0)$ (fig. 6.6a). Le voisinage pour lequel l'histogramme est calculé est déterminé par le rayon du cercle centré en ce point. Le rayon peut être soit fixé, soit calculé dynamiquement en fonction du contexte global [16].

Le cercle est divisé verticalement et horizontalement en plusieurs régions. Ces régions, aussi appelées « boites » ou "*bins*", servent à compter le nombre de points voisins se trouvant à l'intérieur de celles-ci. Le nombre de divisions horizontales du cercle permettent d'évaluer plus ou moins exactement la distance par rapport à un voisin, tandis que les divisions verticales évaluent leur direction relative en fonction de leur angle (fig 6.6b). De cette façon, les histogrammes tiennent compte d'une géographie locale abstraite et sont donc moins influencés par des distorsions d'image. Il est à noter que certains algorithmes utilisent des histogrammes sous forme rectangulaire et non pas circulaire.

6.3.3 Histogrammes de comparaison

Pour la comparaison de photographies de salamandres, j'ai utilisé une approche similaire à celle des empreintes digitales. A partir des composantes connexes calculées des images en noir et blanc des taches, on recueille leur centre. On calcule ensuite les histogrammes polaires pour tous les centres de taches.

Afin d'avoir plus de contrôle sur la procédure de calcul des histogrammes, j'ai moi-même ré-implémenté l'algorithme de création d'histogrammes. J'ai utilisé la

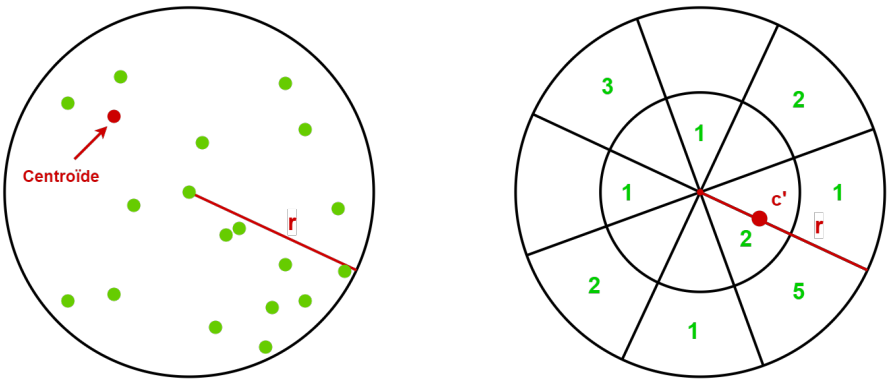


FIGURE 6.6 – Exemple d'utilisation d'histogrammes

description théorique de l'outil de correspondance locale proposée par Junjian et al. et par Minjae et al. [7, 16].

Pour la création d'un histogramme, il y a deux phases principales : la **préparation des histogrammes**, et le **remplissage des boites**.

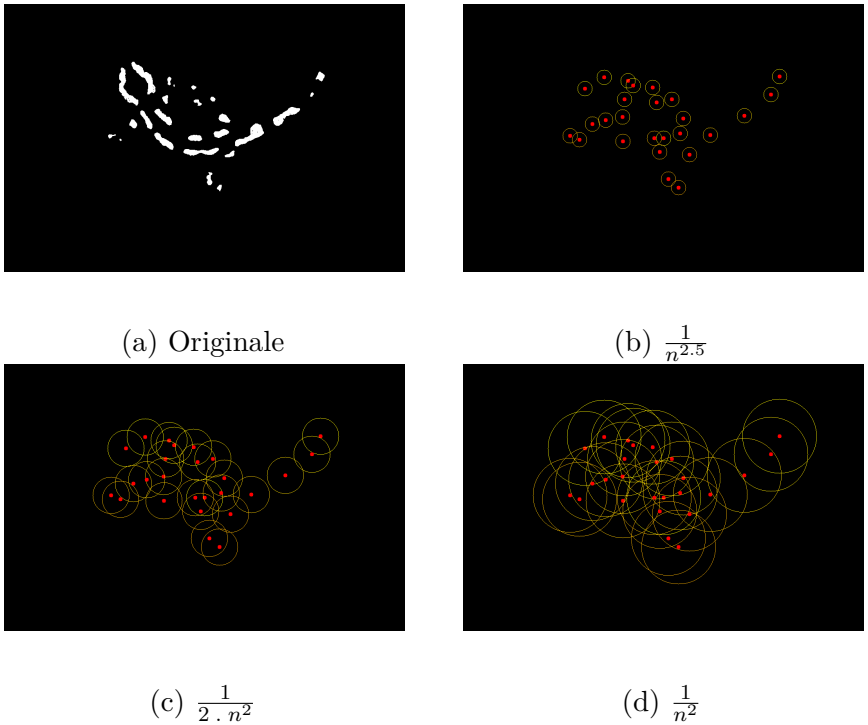
Préparation des histogrammes

La première chose à déterminer pour créer des histogrammes est leur **rayon**.

Deux photos de la même salamandre peuvent être fortement différentes en fonction de la prise de vue. Les mêmes taches peuvent être translatées, tournées, agrandies ou rétrécies. Afin de prendre en compte ces distorsions, il convient de standardiser au maximum les histogrammes créés. Le rayon doit tout d'abord être le même pour tous les histogrammes d'une même image, et ensuite il ne peut pas être fixé arbitrairement. L'équation de son calcul doit prendre en compte les distances entre les points. Pour cela, on calcule la moyenne des distances Euclidiennes entre chaque point.

Soient le rayon r de l'histogramme, la distance Euclidienne $dist$, l'ensemble des centroïdes R, les centroïdes c et le nombre total de centroïdes n, pour calculer le rayon d'un histogramme nous avons :

$$r = \frac{1}{n^2} \sum_{c_p \in R} \sum_{c_q \in R} \frac{1}{2} dist(c_p, c_q) \tag{6.5}$$



(a) Originale (b) $\frac{1}{n^{2.5}}$

(c) $\frac{1}{2 \cdot n^2}$ (d) $\frac{1}{n^2}$

FIGURE 6.7 – Influence de la constante de contrôle sur le rayon

De cette façon, le rayon du cercle devient indépendant de la résolution de l'image. Cette formule issue de [16], n'est pas tout à fait expliquée dans cet article. Le facteur $\frac{1}{n^2}$ agit comme une constante de contrôle sur le rayon. S'il est très **petit**, le rayon des histogrammes devient très **grand** et inversement (fig. 6.7).

Remplissage des boites

Après que le rayon global aie été calculé, vient l'étape de la création des histogrammes individuels.

Angle 0° Pour chaque histogramme, il faut déterminer l'« abscisse » du cercle qui déterminera l'angle à 0° de l'histogramme. Pour ce faire, on construit un nouveau centroïde c' qui servira de première base à l'abscisse. On utilise le centre de référence de l'histogramme comme seconde base (fig. 6.6b). Les coordonnées de ce nouveau point c' sont déterminées par la moyenne des coordonnées des autres points.

Avec c les centroïdes, c_i le centroïde de référence de l'histogramme et $N(c_i)$ le nombre de voisins de c_i, le nouveau point c' est calculé par :

$$c' = \frac{1}{|N(c_i)|} \sum_{c_k \in N(c_i)} c_k \tag{6.6}$$

Un point est considéré comme voisin du centroïde de référence s'il est situé à l'intérieur de l'histogramme, et donc si sa distance Euclidienne est plus petite que le rayon.

$$N(c_i) = \{c_k \mid dist(c_i, c_k) \leq r, c_k \in R, k \neq i\} \tag{6.7}$$

De cette façon, l'histogramme résultant sera invariant aux rotations et aux translations des centroïdes, et donc des taches. Seules les positions relatives des points par rapport aux autres est nécessaire pour générer l'histogramme polaire. Ainsi l'histogramme est indépendant des autres informations que l'image pourrait contenir.

Comptage des boites Une fois que l'abscisse a été déterminée, il faut remplir les boites de l'histogramme. Comme expliqué dans la sous-section 6.3.2, l'histogramme est découpé horizontalement et verticalement. Le découpage vertical détermine l'angle tandis que le découpage horizontal détermine la distance. L'algorithme de remplissage passe de boite en boite et compte le nombre de points contenus dans une boite. Le nombre de boites est déterminé à l'avance.

Dans ce projet, j'ai choisi 8 boites verticales, autrement dit chaque boite possède un angle de $\sphericalangle 45°$, et 2 boites horizontales découpées équitablement à $\frac{r}{2}$, avec r le rayon de l'histogramme polaire. J'ai remarqué que dans la littérature, plusieurs combinaisons ont été essayées. Par exemple [7] utilise des histogrammes avec 12 régions verticales et 10 régions horizontales tandis que [16] utilise 8 régions verticales et 3 régions horizontales. J'ai réutilisé le modèle de Minjae en n'utilisant que 2 boites horizontales car le contexte des pores et des taches est différent. D'une part, il y a un nombre très important de pores sur la peau en comparaison au nombre de taches d'une salamandre, ce qui justifie un très grand nombre de boites. D'autre part, les taches de salamandre ont tendance à être beaucoup plus espacées les unes des autres. Un découpage moins important diminue la précision, mais il permet également un calcul de comparaison plus rapide.

6.3.4 Calcul de similarité

Une fois que les histogrammes polaires ont été créés, l'étape suivante consiste à trouver un moyen de les comparer. L'étape finale sera de définir comment comparer deux images comprenant plusieurs histogrammes et d'en tirer un score de similarité.

Comparaison d'histogrammes

J'ai posé l'hypothèse que tous les histogrammes allaient avoir le même nombre d'éléments, ce qui facilitera la comparaison. L'algorithme de comparaison doit pouvoir donner un score de similarité pour les deux histogrammes comparés ; plus ils sont similaires, plus le score sera élevé.

La distance chi-carrée utilisée par Minjae répond à ces attentes. Elle compare boite à boite la différence de points, en commençant par l'angle 0°. Le résultat est un score donné dans l'intervalle $[0, 1]$.

Soient deux histogrammes h_1, h_2, la distance chi-carrée est calculée par :

$$dist(h_1, h_2) = 1 - \chi^2(h_1, h_2) = 1 - \frac{1}{2}\sum_k \frac{\{h_1(k) - h_2(k)\}^2}{h_1(k) + h_2(k) + \gamma} \tag{6.8}$$

où γ est une constante de régularisation pour éviter une division par zero, et où $\gamma = 1\ .\ 10^{-7}$.

Matrice de similarité

Maintenant que nous avons défini comment comparer deux histogrammes isolés, il faut déterminer le moyen de comparaison de deux images, et donc de plusieurs histogrammes. La méthode décrite par [7, 16] définit une matrice de similarité dont les lignes correspondent aux histogrammes d'une première image A et les colonnes aux histogrammes d'une seconde image B. Il n'est donc pas garanti d'obtenir une matrice carrée, le nombre d'histogrammes peut être différent d'une matrice à l'autre. La matrice est ensuite remplie par les scores chi-carré de chaque paire d'histogrammes.

Pour deux histogrammes $h_i \in A$ et $h_j \in B$ appartenant respectivement à la i-ème ligne et j-ème colonne, l'entrée s_{ij} de la matrice de similarité S est calculée par :

$$s_{ij} = dist(h_i, h_j) \tag{6.9}$$

Si s_{ij} possède la valeur maximum *à la fois* sur toute la ligne i et la colonne j, alors (h_i, h_j) est considéré comme ayant une correspondance locale.

L'algorithme de comparaison consiste alors à parcourir cette matrice S pour compter les correspondances locales. Un score global est calculé en faisant une moyenne du nombre de correspondances locales sur le nombre d'histogrammes

total, en ne prenant en considération que le nombre minimum entre les deux images. Ainsi si une image est comparée à sa copie parfaite, le score sera de 1, tandis que le score pour deux images différentes tendra vers 0.

Avec C le nombre de correspondances locales et n, m les nombres d'histogrammes pour deux images comparées, le score est calculé par :

$$Score = \frac{C}{\min\{n, m\}} \tag{6.10}$$

6.3.5 Autres méthodes de comparaison d'histogrammes

Empreintes digitales Dans ce travail, je me base notamment sur des travaux sur les empreintes digitales où une approche à base d'histogrammes a été appliquée. Les histogrammes sont utilisés afin de comparer des images d'empreintes, les pores de la peau servent de centroïde pour leur création [7, 16]. J'ai repris en grande partie leur approche en l'adaptant au contexte de mon problème.

Local Binary Patterns Les LBP sont un autre type de descripteur visuel utilisant les histogrammes polaires. Ils ont été initialement conçu pour classifier des textures [19]. Le principe général des LBP consiste à diviser l'image en motifs réguliers puis dans ces zones, à comparer le niveau de luminance de pixels par rapport à leurs voisins. Ces zones sont converties en binaire en fonction des résultats de comparaisons de pixels. Les valeurs binaires (parfois converties en décimales) des zones de deux images peuvent ensuite être comparées directement.

Algorithmes perdus dans l'espace Le problème de la relation géométrique a également été étudié sous le nom de « *Lost in space algorithms* »en astronomie, où des photographies de ciels étoilés étaient comparées. Les étoiles formant le nuage de points, des descriptions locales des étoiles étaient comparées afin de déterminer la similitude entre les images [22, 29].

6.4 Conclusion

Dans ce chapitre, j'ai tout d'abord exposé la méthode d'évaluation de l'algorithme de comparaison en parlant des différentes métriques utilisées.

J'ai enchaîné en parlant de la méthode des blobs et centroïdes, méthode de comparaison utilisant les composantes connexes. Cette méthode des composantes

connexes permet, entre autre, à partir d'une photo en noir et blanc du dos d'une salamandre, de déceler ses taches. L'algorithme trouve égalementle centre de ces taches, centres qui seront utilisés pour la méthode de comparaison.

Comme la bonne détection des taches est primordiale pour la performance de l'algorithmes de comparaison, j'ai également mené une analyse des différents facteurs pouvant influencer la taille et le nombre de taches trouvées sur les photos. Cette analyse a permis d'obtenir un certain nombre d'intuitions sur les paramètres optimaux à utiliser afin d'obtenir une bonne performance de l'algorithme de comparaison.

Enfin, j'ai attaqué la partie de l'algorithme de comparaison lui-même, en expliquant le principe des histogrammes polaires, et leur utilisation pour la comparaison de photographies de salamandres.

Chapitre 7

Résultats

Dans ce chapitre, je vais analyser les performances de l'algorithme de comparaison. Ce chapitre est divisé en deux parties principales :

1. La première section traite des performances de l'algorithme de comparaison sur le dataset de base, celui de ManderMatcher (cfr. 7.1) ;
2. La seconde section traite des performances de l'algorithme de comparaison sur le dataset complet, à savoir la conjonction de celui de ManderMatcher, de LLN et d'Andenne (cfr. 7.2).

Les tests sont standardisés et comportent deux parties. La première partie présente les paramètres choisis pour le test, par exemple la 8-connexité, ainsi que des photos segmentées par K-Means avec CIELAB et 2 labels. La seconde partie présente les résultats observés. J'ai repris les métriques présentées dans la section 6.1.1 ainsi que 3 nouvelles données brutes : la moyenne des probabilités totales, la moyenne des probabilités pour `TP` (*True Positives* cfr. 6.1) et la moyenne des probabilités pour `TN` (*True Negatives*). Ces 3 données brutes sont utiles afin de trouver le *threshold* qui sera utilisé afin de déterminer à partir de quelle probabilité deux images sont considérées comme étant similaires.

Pour réaliser les différents tests, j'ai créé plusieurs datasets à partir des données destinées à l'identification (cfr. Tableau 3.2). Ils sont repris dans la Table 7.1.

Remarque importante Les tests de comparaison ont été réalisés avec l'aide de l'ensemble des masques créés manuellement, et non pas ceux générés par U-Net. Les résultats sont donc certainement sur-évalués.

Dataset	Nb photos	Individus distincts	Source(s)
DummySet	60	30	Mandermatcher
ManderGoodSet	88	23	Mandermatcher
ManderSet	393	38	Mandermatcher
TotalSet	753	103	Mander, Andenne, LLN

TABLE 7.1 – Résumé des datasets des tests de l'algorithme d'identification

Test naïf	
Threshold	0.99
Formule rayon	$r = \frac{1}{n^2}$
Normalisation d'images	HistEgal
K-Means	Cielab ($ch = 12, n = 2$)
Nombre de photos	60 (individus : 30)
Dataset	DummySet
Érosion-dilatation	Oui (kernel : 5)
Connexité	8

TABLE 7.2 – Paramètres du test naïf

7.1 Résultats sur les données de ManderMatcher

7.1.1 Test naïf

Avant d'aller plus loin dans les tests, j'ai d'abord effectué un test naïf de performance en donnant les mêmes photographies deux fois à l'algorithme de comparaison. Les salamandres sur ces photographies ne sont pas présentes plusieurs fois. Ce qui fait que sur les 60 images de salamandres, il y a exactement 30 individus distincts. Pour être certain que la méthode des blobs et centroïdes soit efficace, il faut que la métrique s'approche de 1. Les paramètres complets du test sont présentés dans la table 7.2. Il s'agit des paramètres dont j'avais eu l'intuition qu'ils donneraient de bons résultats. Au vu de la probabilité moyenne du taux de similarité du groupe TP qui est de 1.0, j'ai décidé de fixer le threshold à 0.99. C'est à dire, seuls les individus dont on est prsque sûr qu'ils sont identiques sont considérés pour le test, tous les autres sont rejetés.

Les résultats du test naïf sont montrés sur la table 7.3. On peut remarquer que sur l'entièreté des 30 paires de photos identiques, l'algorithme a toujours pu les classer correctement comme étant "*similaires*". Le taux d'erreur de classification pour les individus non similaires classés comme étant similaires est très faible. La *balanced accuracy* est pratiquement parfaite (99%), ce qui confirme l'exactitude de

Matrice de confusion	
30	0
1	434
Precision	0.97
Recall	1.0
Bal. Accuracy	0.99

TABLE 7.3 – Résultats du test naïf

(a) Photo originale (b) Photo érodée (c) Photo "*ouverte*"

FIGURE 7.1 – Effet très négatif de l'érosion-dilatation

l'algorithme pour les individus similaires.

Par soucis de comparaison pour les tests à venir, j'ai également fait les tests en fixant le threshold à 0. Les résultats sont beaucoup moins encourageant mais restent élevés : la balanced accuracy est fixé à 79% et le recall est toujours fixé à 1.0.

Pourquoi l'algorithme s'est-il malgré tout trompé dans 1 cas sur 435 ? En effet, il est curieux que l'algorithme se soit quand même trompé lors de la comparaison d'individus différents. La raison ? L'opération *érosion-dilatation* qui a réduit trop fortement la taille des taches. On peut voir l'effet de l'opération d'"*ouverture*" avec les figures 7.1a, 7.1b et 7.1c. Suite à cette opération, l'algorithme compte seulement 2 taches, ce qui fausse complètement les résultats de comparaison. Cela pourrait peut-être s'expliquer par la résolution trop faible de l'image.

7.1.2 Test sur un ensemble limité de photos

Pour ce test, j'ai réalisé une sélection de photographies basée sur la qualité (subjective) des images en noir et blanc comprenant les taches. Il y a au total 88 photos avec 23 individus distincts. J'ai écarté les images où le **bruit** est trop présent, où les taches sont **fusionnées** et où des taches sont **manquantes**. La Table 7.4 montre les paramètres communs aux différents tests. J'ai comparé les

Test photos sel. Mandermatcher	
Formule rayon	$r = \frac{1}{n^2}$
K-Means	Cielab ($ch = 12, n = 2$)
Nombre de photos	88 (individus : 23)
Dataset	ManderGoodSet
Connexité	8

TABLE 7.4 – Paramètres du test photos sélectionnées de Mandermatcher

Bal. Accuracy	Taille noyau		
Normalisation	3	5	7
∅	**0.58**	0.54	0.55
Égal-hist	0.52	**0.57**	0.47

TABLE 7.5 – Résultats des tests photos sélectionnées de Mandermatcher ($th = 0.0$)

résultats en modifiant les paramètres de normalisation, de taille de noyaux pour érosion-dilatation, et le threshold.

Sur la Table 7.5, nous pouvons remarquer que l'intuition selon laquelle l'utilisation de normalisation par égalisation d'histogrammes donnerait de moins bons résultats est vérifiée. On peut aussi constater que la taille du noyau a une certaine importance sur la **balanced accuracy**. Le threshold a été conservé à 0 afin d'être comparable aux autres résultats.

Les résultats sont beaucoup plus bas que ceux observés lors du test naïf ($0.57 < 0.79$), tout paramètre étant égal par ailleurs. Comme j'ai sélectionné les photos au préalable, d'autres facteurs doivent expliquer la différence de résultats. On peut par exemple citer le fait que je n'ai pas sélectionné les photos en fonction de la position de la salamandre. Parfois, les salamandres ne sont pas tout à fait droites, ou leurs pattes ne sont pas toujours visibles. Or, bien que l'algorithme ait une certaine résistance à la variance des points, la position relative des taches par rapport aux autres sur ces dernières images peut induire des erreurs d'identification.

Cela signifie qu'afin d'obtenir de meilleurs résultats, une normalisation des photos est indispensable.

7.1.3 Test sur l'ensemble des données de Mandermatcher

Pour ce troisième test, j'ai utilisé le dataset complet issu de Mandermatcher afin de vérifier la robustesse de mon algorithme. J'ai utilisé différents paramètres comme l'utilisation ou non de l'érosion-dilatation, la taille du noyau de ce dernier,

Test photos Mandermatcher	
Formule rayon	$r = \frac{1}{n^2}$
K-Means	Cielab ($ch = 12, n = 2$)
Nombre de photos	393 (individus : 38)
Dataset	ManderSet
Connexité	8

TABLE 7.6 – Paramètres du test photos de Mandermatcher

Bal. Accuracy	Taille noyau			
Normalisation	∅	3	5	7
∅	0.49	0.49	0.49	**0.52**
Égal-hist	0.5	0.49	**0.51**	0.49

TABLE 7.7 – Résultats des tests photos sélectionnées de Mandermatcher ($th = 0.0$)

et la normalisation des images. Les paramètres communs utilisés pour calculer les différents résultats sont présentés dans la Table 7.6. Par facilité, je me suis également concentré sur la **balanced accuracy** pour évaluer les meilleurs paramètres à utiliser.

Dans la Table 7.7, j'ai retranscris les résultats des tests qui portaient sur l'influence de la normalisation et de la taille du noyau utilisé par l'érosion-dilatation. On peut constater que la **balanced accuracy** est beaucoup plus basse que la précédente valeur calculée sur un ensemble limité ($\max 0.52 < \max 0.58$). Il y a également une forte homogénéité dans les résultats, l'écart-type de l'ensemble des **balanced accuracy** est de 0.011.

On peut en conclure que le changement de ces paramètres n'a pas beaucoup d'influence sur le résultat global. Le plus faible score peut s'expliquer par l'ajout d'images de moins bonne qualité.

7.2 Résultats sur l'ensemble des données

Après avoir évalué ma méthode avec l'ensemble de données de Mandermatcher, je l'ai également testée sur l'ensemble des photos dont je dispose, à savoir, les ensembles de Mandermatcher, d'Andenne et de LLN. J'ai repris les meilleurs paramètres trouvés jusqu'à présent (cfr. 7.1.3). Les paramètres communs figurent dans la Table 7.8.

Les résultats finaux sont présentés dans la table 7.9. La meilleure configuration n'utilise pas de normalisation par égalisation d'histogrammes et un noyau de 7

Test global	
Formule rayon	$r = \frac{1}{n^2}$
K-Means	Cielab ($ch = 12, n = 2$)
Nombre de photos	753 (individus : 103)
Dataset	TotalSet
Connexité	8

TABLE 7.8 – Paramètres du test global

Bal. Accuracy	Taille noyau	
Normalisation	5	7
∅	0.57	**0.59**
Égal-hist	0.55	0.54

TABLE 7.9 – Résultats des tests photos sélectionnées de Mandermatcher ($th = 0.0$)

pour l'érosion-dilatation. Le score de 59% est largement supérieur à ceux donnés pour l'ensemble des données de Mandermatcher uniquement ($0.59 > \max 0.52$). La matrice de confusion pour cette configuration figure sur la table 7.10. L'algorithme se trompe rarement quand il s'agit de déterminer que deux salamandres sont différentes, le nombre de faux négatifs (FN) est faible. En revanche, le nombre de faux positifs (FP) est très élevé.

Le score plus élevé peut s'expliquer par un biais statistique, le jeu de données final ayant été fortement augmenté et diversifié. Cependant, ce résultat reste décevant, l'algorithme se trompe assez souvent. Dans la prochaine section, je vais proposer des pistes de réflexion sur les limites de la méthode.

Matrice de confusion	
1543	626
147774	133185
Precision	0.01
Recall	0.71
Bal. Accuracy	0.59

TABLE 7.10 – Résultats finaux sur l'ensemble des données

7.3 Limites de la méthode

Dans cette section, je vais mettre en évidence les limites de la méthode en comparant son implémentation à celle de référence. Le contexte étant différent, les résultats le sont également.

7.3.1 Forme des taches

Les centres des taches sont calculés à partir des blobs de taches données lors de la seconde segmentation, par K-Means (Section 5.3). Ils sont calculés de manière automatique par les fonctions d'*open-cv*, qui fait une somme des points constituant un blob, et déduit le point permettant de calculer la moyenne de ces coordonnées. C'est une méthode plutôt simple à utiliser, mais elle est complètement dépendante de la forme des blobs.

En outre, si les blobs ont des formes plutôt convexes, les centres peuvent ne pas se trouver à l'intérieur des taches. De plus, si les taches ont des formes très allongées ou même très grande, le centre ne tiendra pas vraiment compte de l'importance des taches. Ainsi, avec la méthode des histogrammes, une salamandre possédant beaucoup de petites taches sera mieux caractérisée qu'une salamandre possédant peu de grosses taches.

L'homogénéité de la forme et de la taille des taches n'est donc pas assurée dans notre cas. Or, dans l'article scientifique sur lequel je me suis basé pour implémenter cette méthode, les pores de la peau ont des formes et tailles similaires.

7.3.2 Nombre des taches

En outre, la comparaison par histogrammes polaires se base avant tout sur le comptage de taches. Or, si ce nombre est exagéré, la comparaison sera faussée. Toutefois, comme je l'ai indiqué dans la section 6.2.2, le nombre de petites taches est très élevé. L'utilisation de la technique d'érosion-dilatation permet de parer en partie cet effet. Cependant, cela ne suffit manifestement pas pour obtenir un meilleur score de comparaison.

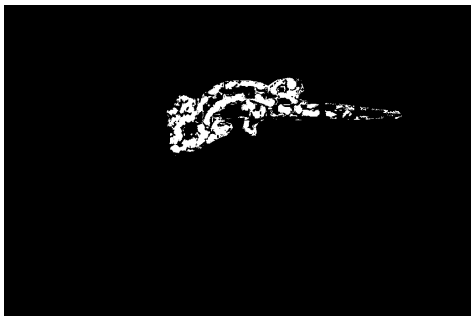
7.3.3 Effets liés à la prise de vue

Les salamandres sont difficiles à capturer dû à leur mode de vie nocturne. Cela rend la prise de photographies une tâche ardue. J'ai compté plusieurs facteurs pouvant dégrader la qualité des photos, en voici une liste non-exhaustive :

Luminosité

La luminosité affecte énormément la détection des taches. Si la salamandre est mal éclairée, il y a un risque de sous-exposition et de sur-exposition. Un phénomène récurrent était la surexposition d'un coté latéral du dos et la sous-exposition de l'autre. K-Means aura tendance à morceler les taches sur-exposées, et à ignorer les taches sous-exposées (fig 7.2).

(a) Original



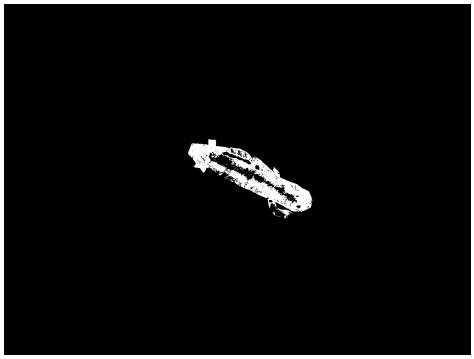
(b) Segmentée par K-Means

FIGURE 7.2

Couleurs

Ce facteur est en partie lié à la luminosité et dû au fonctionnement de K-Means. Les salamandres sont parfois trop éclairées, le noir prendra alors une teinte plus grise et le jaune une teinte claire. Dans ce cas, K-Means regroupera les couleurs ensemble (fig 7.3).

(a) Original



(b) Segmentée par K-Means

FIGURE 7.3

Végétation

Même si la salamandre est correctement segmentée, dans certains cas les prises de vues sont faites alors que des brins d'herbes ou des feuilles se trouvent au-dessus de leur corps. La segmentation ne retire pas toujours cette végétation, ce qui provoque un biais de K-Means qui tiendra compte de la clarté des taches et les associera généralement au jaune des taches (fig 7.4).

(a) Original (b) Segmentée par K-Means

FIGURE 7.4

Effets de bord

Cette effet est également lié à la segmentation des salamandres par rapport au fond. Si la segmentation n'est pas bien effectuée, ce qui arrive même manuellement, certains pixels du fond peuvent se retrouver malgré tout dans le masque de comparaison (fig 7.5).

Or, lorsque le fond est très clair, par exemple un tapis de feuilles mortes brunes, ces quelques pixels ont une forte incidence sur la détection de taches. Même si l'utilisation de l'érosion-dilatation diminue cet effet, lorsqu'il y a un trop grand nombre de ces pixels contigus, ils faussent complètement le résultat de la comparaison.

(a) Original (b) Segmentée par K-Means

FIGURE 7.5

7.4 Conclusion

Dans ce chapitre, j'ai testé plusieurs configurations afin de trouver celle qui donnerait le meilleur score. J'ai utilisé des ensembles de données différents afin d'observer leurs effets.

Au final, je constate que l'algorithme n'obtient pas un score de comparaison élevé, ce qui semble dû notamment à la qualité des photos d'une part, et d'autre part à la méthode en elle-même.

Certains facteurs, comme la normalisation par égalisation d'histogrammes ou la taille du Kernel de l'érosion-dilatation, ont parfois des conséquences négatives. L'utilisation de l'érosion-dilatation a parfois des conséquences imprévues.

Chapitre 8

Travaux futurs

Dans ce chapitre, je vais aborder diverses pistes d'amélioration pour l'algorithme d'identification de salamandres. Tout le long de l'écriture de ce travail, j'ai dû faire des choix d'implémentation. J'ai dû choisir certaines pistes au détriment d'autres, soit par manque de temps, soit parce que je ne les aie pas jugées suffisamment intéressantes. Je vais revenir ici sur certaines pistes non explorées qui mériteraient, selon moi, que l'on s'y intéresse plus profondément.

8.1 À propos des données

Je suis parti de rien, j'ai dû, avec l'aide de mes promoteurs, trouver un grand nombre de photographies. Avant d'en recevoir de la part de professionnels du secteur, j'ai dû moi-même créé un dataset de départ. Comme je suis parti de l'hypothèse que les photos pouvaient provenir tout aussi bien d'amateurs que de professionnels, j'ai gardé un nombre assez important de photos de mauvaise qualité. Bien que cela ait eu un impact sur l'entrainement du réseau U-Net, cela a également pu avoir un impact significatif sur les résultats de l'algorithme de comparaison.

D'autre part, ma sélection de "bonnes photos" s'est basé exclusivement sur des critères subjectifs. Je suggérerais une meilleure sélection basé sur des critères objectifs, lorsque cela est possible. J'ai par exemple découvert, un peu tard pour l'implémenter dans ce travail, qu'il était possible de quantifier le flou dans une image. On pourrait imaginer un premier filtrage des images basé sur un threshold de flou qui retirerait un grand nombre de photos de piètre qualité.

La position des salamandres peut également avoir un impact très négatif sur les résultats finaux. Je pensais pourtant avoir pris des dispositions suffisantes en utilisant l'algorithme des empreintes digitales, qui est dit "invariable aux rotations,

translations et mise à l'échelle des images". Basé sur cette hypothèse, je n'ai pas standardisé les images en rapport à la position prises par les salamandres. Par exemple, les placer systématiquement tête vers le haut, ou éliminer les photos où la queue est courbée, parfois dans des positions étranges (!). Pour la première hypothèse, il serait par exemple possible d'utiliser la méthode des squelettes topologiques, qui permettrait d'identifier la position de la salamandre dans l'image [15, 12].

8.2 Amélioration de la segmentation

Je pense avoir obtenu des résultats satisfaisant avec mon modèle de U-Net. Cependant, je pense qu'il serait possible de faire mieux. J'ai utilisé certains paramètres "par défaut" du modèle de U-Net pour lesquels j'aurais pu tester des variants, et qui auraient pu être testé. Je suis d'avis par exemple que la fonction de perte "sparse categorical crossentropy" n'est pas vraiment adaptée à la segmentation d'images. D'autres fonctions de perte existent, je pense par exemple à l'index de Jaccard et au score de Dice qui sont utilisés notamment dans le domaine médical [3], et que je n'ai malheureusement pas eu l'occasion d'implémenter.

8.3 Algorithme de comparaison

L'algorithme des blobs et centroïdes, bien qu'efficace pour les empreintes digitales, souffre de plusieurs limites structurelles qui l'empêche de donner de bons résultats. Cependant, il y a tout de même certaines parties de l'algorithme qui pourraient être améliorées. Par exemple, les histogrammes pourraient gagner en précision si le nombre de régions angulaires et spatiales était augmenté. Ici, j'ai utilisé 8 régions angulaires et 2 spatiales car je pensais que ce nombre était suffisant vu le contexte, mais il est possible qu'une meilleure précision des bins rendrait l'algorithme plus robuste.

D'autre part, compte tenu de la nature des données que je possédais, il est possible qu'une sélection plus stricte de photos de bonnes qualité améliore significativement le test de probabilité de similitude.

8.4 Autres algorithmes d'identification

Enfin, d'autres algorithmes d'identification existent ! J'ai choisi de focaliser mon travail sur la méthode des blobs et centroïdes, mais d'autres méthodes plus adaptées

aux formes particulières des taches existent également. Je citerai notamment la méthode des shape context et la méthode des local binary patterns (cfr. 6.3.5) qui sont deux candidats intéressants pour une future étude.

Un shape context est un descripteur de forme utilisant un ensemble discret de points échantillonnés sur les contours de cette forme à intervalles réguliers. Chaque point enregistre via un histogramme log-polaire les positions relatives de tous les autres points du shape context. Chaque point possède alors une représentation grossière de l'ensemble de la forme. L'intuition étant que des formes similaires auront des points avec shape contexts similaires. Un descripteur riche comme celui-ci prédit potentiellement de manière plus correcte qu'une simple approche à base de centroïdes ne tenant pas compte de la forme des taches [2].

8.5 Conclusion

Une sélection plus stricte de l'ensemble de données devrait améliorer significativement l'exactitude des résultats, au risque de perdre l'hypothèse de mixité de départ : ne prendre uniquement que les photos sans défauts, faites par des professionnels.

La segmentation des images par U-Net présente de bons résultats et une mise en production future pourrait même être envisagée. Cependant, il existe toujours des pistes d'améliorations, notamment l'utilisation d'autres fonctions de perte utilisées pour la segmentation d'images médicales.

Bien qu'elle aie été en grande partie explorée, la méthode des blobs et centroïdes pourrait encore bénéficié d'une amélioration de la précision des histogrammes polaires, en augmentant le nombre de régions angulaires et spatiales.

Enfin, le plus important pour moi serait de s'intéresser aux autres méthodes d'identification, comme la méthode des **shape context** qui présente un gros potentiel. Des méthodes tenant mieux compte de la réalité des formes de taches devraient mieux s'en sortir que celle des blobs et centroïdes.

Chapitre 9

Conclusion

Dans le cadre de ce mémoire, j'ai cherché à trouver des solutions informatiques à l'identification des salamandres, avec en perspective le dénombrement des populations de salamandres en Belgique. L'objectif final de ce travail était la création d'un outil de reconnaissance automatique de salamandres, vu le besoin ressenti par les professionnels du milieu.

Je pensais pouvoir résoudre ce problème de manière assez "simple" grâce au deep-learning, par l'identification des salamandres via leurs taches caractéristiques.

C'est là que j'ai découvert tout un monde nouveau et toutes les spécificités liées à cette étude :

- découverte des caractéristiques spécifiques de cet amphibien (sorties nocturnes, variétés du nombre des taches, de leur nature, ... qui a entraîné des biais de sur- ou sous-exposition) ;
- découverte du réseau de neurones U-Net et la nécessité de disposer d'un nombre important d'images au préalablement traitées. J'ai en effet dû détourer prés de 800 photos à l'aide du logiciel VIA [8, 9], ce qui a consommé beaucoup de temps puisque cette segmentation a dû être réalisé manuellement ;
- découverte du traitement informatique des images, avec toutes ses spécificités, ses réglages, ... ;
- pour vérifier que l'algorithme d'identification des salamandres fonctionnait correctement, j'ai dû mettre au point une méthode afin de repérer les salamandres apparaissant plusieurs fois sur les photos dont je disposais. J'ai notamment utilisé le logiciel *Mandermatcher* pour reconnaitre près de 300 photos manuellement.

Toutes ces difficultés et spécificités que j'ai pu découvrir tout au long de l'année

expliquent les résultats finaux mi-figue mi-raisin. Je suis en revanche assez satisfait d'avoir pu atteindre 89% d'accuracy avec le réseau U-Net, même s'il est encore possible d'améliorer ce résultat. Les résultats du test de performance de la méthode des blobs et centroïdes montrent qu'il ne s'agit probablement pas de la méthode à utiliser pour résoudre ce problème. Une exigence plus stricte sur la qualité des photos utilisées pour tester l'algorithme devrait également être établie. Le public visé par cette application, à savoir aussi bien monsieur tout-le-monde que les professionnels du secteur, gagnerait peut-être à être réduit à ces derniers.

L'ensemble de données que j'ai collectées sera en partie en accès libre – selon les souhaits des détenteurs légaux – et pourra être utilisé dans de futures études. Le code, ainsi que l'ensemble des productions, telles que les "photos transformées" sera également en libre accès. J'espère que dans le cadre d'un autre mémoire, d'autres techniques pourront être testées et qu'elles permettront d'améliorer l'identification des salamandres et par la suite, le dénombrement de celles-ci. Les données collectées pourront servir de base à ce nouveau mémoire, où le travail de collecte et de transformation de données sera allégé.

Je tiens également à mettre en lumière que tout aussi bien des étudiants en informatique qu'en biologie pourraient être intéressés par ces données. Bien que je ne me suis pas vraiment attardé sur le sujet car il était hors de mes compétences, je crois que ces données pourraient également servir de base à un nouveau travail de recensement de la population de salamandres non loin de Louvain-la-neuve, dans la continuité du travail de Violaine Fichefet [11].

Finalement, j'espère que celui ou celle qui reprendra le flambeau prendra autant de plaisir que moi sur ce mémoire et qu'il découvrira ces nouveaux univers.

Bibliographie

[1] Tinku Acharya et Ajoy K Ray. *Image processing : principles and applications*. John Wiley & Sons, 2005. isbn : 0-471-71998-6.

[2] Serge Belongie, Jitendra Malik et Jan Puzicha. « Shape Context : A New Descriptor for Shape Matching and Object Recognition ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de T. Leen, T. Dietterich et V. Tresp. T. 13. MIT Press, 2000. url : https://proceedings.neurips.cc/paper/2000/file/c44799b04a1c72e3c8593a53e8000c78-Paper.pdf.

[3] Jeroen Bertels et al. « Optimizing the Dice Score and Jaccard Index for Medical Image Segmentation : Theory and Practice ». In : *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Sous la dir. de Dinggang Shen et al. Cham : Springer International Publishing, 2019, p. 92-100. isbn : 978-3-030-32245-8.

[4] Marco Carafa et Maurizio Biondi. « Application of a method for individual photographic identification during a study on Salamandra salamandra gigliolii in central Italy ». In : *Italian Journal of Zoology* 71.sup2 (2004), p. 181-184. doi : 10.1080/11250000409356631. eprint : https://doi.org/10.1080/11250000409356631. url : https://doi.org/10.1080/11250000409356631.

[5] Gilles Charlier et Simon Hick. « Automated nocturnal insect monitoring in real-time using YOLOv4 ». Anglais. Mém. de mast. UCL - Ecole polytechnique de Louvain, 2021. url : http://hdl.handle.net/2078.1/thesis:30560.

[6] François Chollet. *Image segmentation with a U-Net-like architecture*. 2019. url : https://keras.io/examples/vision/oxford_pets_image_segmentation/.

[7] Junjian Cui, Moon-Soo Ra et Whoi-Yul Kim. « Fingerprint pore matching method using polar histogram ». In : *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*. 2014, p. 1-2. doi : 10.1109/ISCE.2014.6884381.

[8] A. Dutta, A. Gupta et A. Zissermann. *VGG Image Annotator (VIA)*. http ://www.robots.ox.ac.uk/ vgg/software/via/. Version : 3.0.11, Accessed : 12/11/2021. 2016.

[9] Abhishek Dutta et Andrew Zisserman. « The VIA Annotation Software for Images, Audio and Video ». In : *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France : ACM, 2019. isbn : 978-1-4503-6889-6/19/10. doi : 10.1145/3343031.3350535. url : https://doi.org/10.1145/3343031.3350535.

[10] Charles Elkan. « Using the triangle inequality to accelerate k-means ». In : *Proceedings of the 20th international conference on Machine Learning (ICML-03)*. 2003, p. 147-153.

[11] Violaine Fichefet. « Distribution de la salamandre terrestre (Salamandra salamandra terrestris) en région wallonne et étude d'une population au bois de Lauzelle ». Français. Mém. de mast. UCLouvain - Faculté des bioingénieurs, 2000. url : http://hdl.handle.net/2078.1/thesis:10814.

[12] P. Golland, W. Eric et L. Grimson. « Fixed topology skeletons ». In : *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*. T. 1. 2000, 10-17 vol.1. doi : 10.1109/CVPR.2000.855792.

[13] Costantino Grana, Daniele Borghesani et Rita Cucchiara. « Optimized Block-Based Connected Components Labeling With Decision Trees ». In : *IEEE Transactions on Image Processing* 19.6 (2010), p. 1596-1609. doi : 10.1109/TIP.2010.2044963.

[14] Margherita Grandini, Enrico Bagli et Giorgio Visani. *Metrics for Multi-Class Classification : an Overview*. 2020. doi : 10.48550/ARXIV.2008.05756. url : https://arxiv.org/abs/2008.05756.

[15] Ramesh Jain, Rangachar Kasturi, Brian G Schunck et al. *Machine vision*. T. 5. McGraw-hill New York, 1995, p. 55. isbn : 0-07-032018-7.

[16] Min-jae Kim, Whoi-Yul Kim et Joonki Paik. « Optimum Geometric Transformation and Bipartite Graph-Based Approach to Sweat Pore Matching for Biometric Identification ». In : *Symmetry* 10.5 (2018). issn : 2073-8994. doi : 10.3390/sym10050175. url : https://www.mdpi.com/2073-8994/10/5/175.

[17] Vincent Miele et al. « Revisiting animal photo-identification using deep metric learning and network analysis ». In : *Methods in Ecology and Evolution* 12.5 (2021), p. 863-873. doi : https://doi.org/10.1111/2041-210X.13577. eprint : https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.

1111/2041-210X.13577. URL : https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13577.

[18] Keiron O'Shea et Ryan Nash. « An Introduction to Convolutional Neural Networks ». In : *arXiv e-prints*, arXiv :1511.08458 (nov. 2015), arXiv :1511.08458. arXiv : 1511.08458 [cs.NE].

[19] T. Ojala, M. Pietikainen et D. Harwood. « Performance evaluation of texture measures with classification based on Kullback discrimination of distributions ». In : *Proceedings of 12th International Conference on Pattern Recognition*. T. 1. 1994, 582-585 vol.1. DOI : 10.1109/ICPR.1994.576366.

[20] F. Pedregosa et al. « Scikit-learn : Machine Learning in Python ». In : *Journal of Machine Learning Research* 12 (2011), p. 2825-2830.

[21] Joseph Redmon et al. *You Only Look Once : Unified, Real-Time Object Detection*. 2016. arXiv : 1506.02640 [cs.CV].

[22] David Rijlaarsdam et al. « A Survey of Lost-in-Space Star Identification Algorithms Since 2009 ». In : *Sensors* 20.9 (2020). ISSN : 1424-8220. DOI : 10.3390/s20092579. URL : https://www.mdpi.com/1424-8220/20/9/2579.

[23] Olaf Ronneberger, Philipp Fischer et Thomas Brox. « U-Net : Convolutional Networks for Biomedical Image Segmentation ». In : *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Sous la dir. de Nassir Navab et al. Cham : Springer International Publishing, 2015, p. 234-241. ISBN : 978-3-319-24574-4.

[24] Azriel Rosenfeld. « Connectivity in digital pictures ». In : *Journal of the ACM (JACM)* 17.1 (1970), p. 146-160.

[25] C. Schmid et R. Mohr. « Local grayvalue invariants for image retrieval ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.5 (mai 1997), p. 530-535. ISSN : 1939-3539. DOI : 10.1109/34.589215.

[26] Annemarieke Spitzen-van der Sluijs et al. « Rapid enigmatic decline drives the fire salamander (Salamandra salamandra) to the edge of extinction in the Netherlands ». In : *Amphibia-Reptilia* 34.2 (2013), p. 233-239. DOI : https://doi.org/10.1163/15685381-00002891. URL : https://brill.com/view/journals/amre/34/2/article-p233_8.xml.

[27] Pierre Soille. « Opening and Closing ». In : *Morphological Image Analysis : Principles and Applications*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 105-137. ISBN : 978-3-662-05088-0. DOI : 10.1007/978-3-662-05088-0_4. URL : https://doi.org/10.1007/978-3-662-05088-0_4.

[28] Yu Song et al. « Deep learning-based automated image segmentation for concrete petrographic analysis ». In : *Cement and Concrete Research* 135 (2020), p. 106118. ISSN : 0008-8846. DOI : https://doi.org/10.1016/j.cemconres.2020.106118. URL : https://www.sciencedirect.com/science/article/pii/S0008884620301873.

[29] Benjamin B. Spratling et Daniele Mortari. « A Survey on Star Identification Algorithms ». In : *Algorithms* 2.1 (2009), p. 93-107. ISSN : 1999-4893. DOI : 10.3390/a2010093. URL : https://www.mdpi.com/1999-4893/2/1/93.

[30] Goran Šukalo et al. « Novel, non-invasive method for distinguishing the individuals of the fire salamander (Salamandra salamandra) in capture-mark-recapture studies ». In : *Acta Herpetologica* 8.1 (2013), p. 41-45.

[31] Kesheng Wu, Ekow Otoo et Kenji Suzuki. « Optimizing Two-Pass Connected-Component Labeling Algorithms ». In : *Pattern Anal. Appl.* 12.2 (fév. 2009), p. 117-135. ISSN : 1433-7541. DOI : 10.1007/s10044-008-0109-y. URL : https://doi.org/10.1007/s10044-008-0109-y.

[32] Zijun Zhang. « Improved Adam Optimizer for Deep Neural Networks ». In : *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018, p. 1-2. DOI : 10.1109/IWQoS.2018.8624183.

Annexe A

Données et liens

L'intégralité du code utilisé dans le cadre de ce mémoire se trouve sur le site *github* sous license libre à l'adresse suivante :
https://github.com/FrancoisDuchene/salamandres-identification.

Une partie des données utilisées dont je suis l'auteur ou dont on m'aura expressément donné l'autorisation, sera stockée sur les serveurs de l'UCLouvain et pourra être soumise à certaines restrictions d'accessibilité. Les données qui m'ont été partagées (cfr. Mandermatcher, Andenne) ne feront pas parties des données gérées par l'université, et resteront donc privées sauf autorisation expresse de leur(s) auteur(s).

Je me permets cependant de noter qu'une partie des données de Mandermatcher que j'ai utilisé est en libre accès sur le site officiel pour la version DEMO. Elles sont disponibles à l'adresse suivante à la date dont j'écris ces lignes (06/06/2022) :
https://jeroenspeybroeck.shinyapps.io/mandermatcher/.

Annexe B

U-Net

Les structures complètes des modèles de U-Net telles qu'elles sont données par *keras* via la commande model.summary() sont fournies pour les images de résolution 256x256 et 512x512, respectivement aux Figures B et B

TABLE B.1 – U-Net raw model summary – Images de 256x256

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
conv2d (Conv2D)	(None, 128, 128, 32)	896	input_1
batch_normalization (BN)	(None, 128, 128, 32)	128	conv2d
activation (Activation)	(None, 128, 128, 32)	0	batch_normalization
activation_1 (Activation)	(None, 128, 128, 32)	0	activation
separable_conv2d (SC2D)	(None, 128, 128, 64)	2 400	activation_1
batch_normalization_1 (BN)	(None, 128, 128, 64)	256	separable_conv2d
activation_2 (Activation)	(None, 128, 128, 64)	0	batch_normalization_1
separable_conv2d_1 (SC2D)	(None, 128, 128, 64)	4 736	activation_2
batch_normalization_2 (BN)	(None, 128, 128, 64)	256	separable_conv2d_1
max_pooling2d (MP2D)	(None, 64, 64, 64)	0	batch_normalization_2
conv2d_1 (Conv2D)	(None, 64, 64, 64)	2 112	activation
add (Add)	(None, 64, 64, 64)	0	max_pooling2d','conv2d_1
activation_3 (Activation)	(None, 64, 64, 64)	0	add
separable_conv2d_2 (SC2D)	(None, 64, 64, 128)	8 896	activation_3
batch_normalization_3 (BN)	(None, 64, 64, 128)	512	separable_conv2d_2
activation_4 (Activation)	(None, 64, 64, 128)	0	batch_normalization_3
separable_conv2d_3 (SC2D)	(None, 64, 64, 128)	17 664	activation_4
batch_normalization_4 (BN)	(None, 64, 64, 128)	512	separable_conv2d_3
max_pooling2d_1 (MP2D)	(None, 32, 32, 128)	0	batch_normalization_4
conv2d_2 (Conv2D)	(None, 32, 32, 128)	8 320	add
add_1 (Add)	(None, 32, 32, 128)	0	max_pooling2d_1,conv2d_2
activation_5 (Activation)	(None, 32, 32, 128)	0	add_1
conv2d_transpose (C2DT)	(None, 32, 32, 128)	147 584	activation_5
batch_normalization_5 (BN)	(None, 32, 32, 128)	512	conv2d_transpose

TABLE B.1 – (continued)

Layer (type)	Output Shape	Param #	Connected to
activation_6 (Activation)	(None, 32, 32, 128)	0	batch_normalization_5
conv2d_transpose_1 (C2DT)	(None, 32, 32, 128)	147 584	activation_6
batch_normalization_6 (BN)	(None, 32, 32, 128)	512	conv2d_transpose_1
up_sampling2d_1 (US2D)	(None, 64, 64, 128)	0	add_1
up_sampling2d (US2D)	(None, 64, 64, 128)	0	batch_normalization_6
conv2d_3 (Conv2D)	(None, 64, 64, 128)	16 512	up_sampling2d_1
add_2 (Add)	(None, 64, 64, 128)	0	up_sampling2d,conv2d_3
activation_7 (Activation)	(None, 64, 64, 128)	0	add_2
conv2d_transpose_2 (C2DT)	(None, 64, 64, 64)	73 792	activation_7
batch_normalization_7 (BN)	(None, 64, 64, 64)	256	conv2d_transpose_2
activation_8 (Activation)	(None, 64, 64, 64)	0	batch_normalization_7
conv2d_transpose_3 (C2DT)	(None, 64, 64, 64)	36 928	activation_8
batch_normalization_8 (BN)	(None, 64, 64, 64)	256	conv2d_transpose_3
up_sampling2d_3 (US2D)	(None, 128, 128, 128)	0	add_2
up_sampling2d_2 (US2D)	(None, 128, 128, 64)	0	batch_normalization_8
conv2d_4 (Conv2D)	(None, 128, 128, 64)	8 256	up_sampling2d_3
add_3 (Add)	(None, 128, 128, 64)	0	up_sampling2d_2,conv2d_4
activation_9 (Activation)	(None, 128, 128, 64)	0	add_3
conv2d_transpose_4 (C2DT)	(None, 128, 128, 32)	18 464	activation_9
batch_normalization_9 (BN)	(None, 128, 128, 32)	128	conv2d_transpose_4
activation_10 (Activation)	(None, 128, 128, 32)	0	batch_normalization_9
conv2d_transpose_5 (C2DT)	(None, 128, 128, 32)	9 248	activation_10
batch_normalization_10 (BN)	(None, 128, 128, 32)	128	conv2d_transpose_5
up_sampling2d_5 (US2D)	(None, 256, 256, 64)	0	add_3
up_sampling2d_4 (US2D)	(None, 256, 256, 32)	0	batch_normalization_10
conv2d_5 (Conv2D)	(None, 256, 256, 32)	2 080	up_sampling2d_5
add_4 (Add)	(None, 256, 256, 32)	0	up_sampling2d_4',conv2d_5
conv2d_8 (Conv2D)	(None, 256, 256, 2)	258	add_4
Total params :	509 186		
Trainable params :	507 458		
Non-trainable params :	1 728		
Légende :	[1]		

TABLE B.2 – U-Net raw model summary – Images de 512x512

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 512, 512, 3)]	0	[]
conv2d (Conv2D)	(None, 256, 256, 32)	896	input_1
batch_normalization (BN)	(None, 256, 256, 32)	128	conv2d
activation (Activation)	(None, 256, 256, 32)	0	batch_normalization
activation_1 (Activation)	(None, 256, 256, 32)	0	activation
separable_conv2d (SC2D)	(None, 256, 256, 64)	2 400	activation_1

1. BN : BatchNormalization, SC2D : SeparableConv2D, C2DT : Conv2DTranspose, US2D : UpSampling2D, MP2D : MaxPooling2D

TABLE B.2 – (continued)

Layer (type)	Output Shape	Param #	Connected to
batch_normalization_1 (BN)	(None, 256, 256, 64)	256	separable_conv2d
activation_2 (Activation)	(None, 256, 256, 64)	0	batch_normalization_1
separable_conv2d_1 (SC2D)	(None, 256, 256, 64)	4 736	activation_2
batch_normalization_2 (BN)	(None, 256, 256, 64)	256	separable_conv2d_1
max_pooling2d (MP2D)	(None, 128, 128, 64)	0	batch_normalization_2
conv2d_1 (Conv2D)	(None, 128, 128, 64)	2 112	activation
add (Add)	(None, 128, 128, 64)	0	max_pooling2d',’conv2d_1
activation_3 (Activation)	(None, 128, 128, 64)	0	add
separable_conv2d_2 (SC2D)	(None, 128, 128, 128)	8 896	activation_3
batch_normalization_3 (BN)	(None, 128, 128, 128)	512	separable_conv2d_2
activation_4 (Activation)	(None, 128, 128, 128)	0	batch_normalization_3
separable_conv2d_3 (SC2D)	(None, 128, 128, 128)	17 664	activation_4
batch_normalization_4 (BN)	(None, 128, 128, 128)	512	separable_conv2d_3
max_pooling2d_1 (MP2D)	(None, 64, 64, 128)	0	batch_normalization_4
conv2d_2 (Conv2D)	(None, 64, 64, 128)	8 320	add
add_1 (Add)	(None, 64, 64, 128)	0	max_pooling2d_1,conv2d_2
activation_5 (Activation)	(None, 64, 64, 128)	0	add_1
separable_conv2d_4 (SC2D)	(None, 64, 64, 256)	34 176	activation_5
batch_normalization_5 (BN)	(None, 64, 64, 256)	1 024	separable_conv2d_4
activation_6 (Activation)	(None, 64, 64, 256)	0	batch_normalization_5
separable_conv2d_5 (SC2D)	(None, 64, 64, 256)	68 096	activation_6
batch_normalization_6 (BN)	(None, 64, 64, 256)	1 024	separable_conv2d_5
max_pooling2d_2 (MP2D)	(None, 32, 32, 256)	0	batch_normalization_6
conv2d_3 (Conv2D)	(None, 32, 32, 256)	33 024	add_1
add_2 (Add)	(None, 32, 32, 256)	0	max_pooling2d_2',’conv2d_3
activation_7 (Activation)	(None, 32, 32, 256)	0	add_2
conv2d_transpose (C2DT)	(None, 32, 32, 256)	590 080	activation_7
batch_normalization_7 (BN)	(None, 32, 32, 256)	1 024	conv2d_transpose
activation_8 (Activation)	(None, 32, 32, 256)	0	batch_normalization_7
conv2d_transpose_1 (C2DT)	(None, 32, 32, 256)	590 080	activation_8
batch_normalization_8 (BN)	(None, 32, 32, 256)	1 024	conv2d_transpose_1
up_sampling2d_1 (US2D)	(None, 32, 32, 256)	0	add_2
up_sampling2d (US2D)	(None, 32, 32, 256)	0	batch_normalization_8
conv2d_4 (Conv2D)	(None, 32, 32, 256)	65 792	up_sampling2d_1
add_3 (Add)	(None, 32, 32, 256)	0	up_sampling2d,conv2d_4
activation_9 (Activation)	(None, 32, 32, 256)	0	add_3
conv2d_transpose_2 (C2DT)	(None, 64, 64, 128)	295 040	activation_9
batch_normalization_9 (BN)	(None, 64, 64, 128)	512	conv2d_transpose_2
activation_10 (Activation)	(None, 64, 64, 128)	0	batch_normalization_9
conv2d_transpose_3 (C2DT)	(None, 64, 64, 128)	147 584	activation_10
batch_normalization_10 (BN)	(None, 64, 64, 128)	512	conv2d_transpose_3
up_sampling2d_3 (US2D)	(None, 128, 128, 256)	0	add_3
up_sampling2d_2 (US2D)	(None, 128, 128, 128)	0	batch_normalization_10
conv2d_5 (Conv2D)	(None, 128, 128, 128)	32 896	up_sampling2d_3
add_4 (Add)	(None, 128, 128, 128)	0	up_sampling2d_2,conv2d_5
activation_11 (Activation)	(None, 128, 128, 128)	0	add_4

TABLE B.2 – (continued)

Layer (type)	Output Shape	Param #	Connected to
conv2d_transpose_4 (C2DT)	(None, 128, 128, 64)	73 792	activation_11
batch_normalization_11 (BN)	(None, 128, 128, 64)	256	conv2d_transpose_4
activation_12 (Activation)	(None, 128, 128, 64)	0	batch_normalization_11
conv2d_transpose_5 (C2DT)	(None, 128, 128, 64)	36 928	activation_12
batch_normalization_12 (BN)	(None, 128, 128, 64)	256	conv2d_transpose_5
up_sampling2d_5 (US2D)	(None, 256, 256, 128)	0	add_4
up_sampling2d_4 (US2D)	(None, 256, 256, 64)	0	batch_normalization_12
conv2d_6 (Conv2D)	(None, 256, 256, 64)	8 256	up_sampling2d_5
add_5 (Add)	(None, 256, 256, 64)	0	up_sampling2d_4','conv2d_6
activation_13 (Activation)	(None, 256, 256, 64)	0	add_5
conv2d_transpose_6 (C2DT)	(None, 256, 256, 32)	18 464	activation_13
batch_normalization_13 (BN)	(None, 256, 256, 32)	128	conv2d_transpose_6
activation_14 (Activation)	(None, 256, 256, 32)	0	batch_normalization_13
conv2d_transpose_7 (C2DT)	(None, 256, 256, 32)	9 248	activation_14
batch_normalization_14 (BN)	(None, 256, 256, 32)	128	conv2d_transpose_7
up_sampling2d_7 (US2D)	(None, 512, 512, 64)	0	add_5
up_sampling2d_6 (US2D)	(None, 512, 512, 32)	0	batch_normalization_14
conv2d_7 (Conv2D)	(None, 512, 512, 32)	2 080	up_sampling2d_7
add_6 (Add)	(None, 512, 512, 32)	0	up_sampling2d_6,conv2d_7
conv2d_8 (Conv2D)	(None, 512, 512, 2)	578	add_6
Total params :	2 058 690		
Trainable params :	2 054 914		
Non-trainable params :	3 776		
Légende :	[2]		

2. BN : BatchNormalization, SC2D : SeparableConv2D, C2DT : Conv2DTranspose, US2D : UpSampling2D, MP2D : MaxPooling2D

Annexe C

Identification

C.1 Analyse des données de taches

Dans le chapitre sur l'identification, et plus précisément dans la Sous-section 6.2.2, j'ai mentionné qu'à la suite de mon analyse statistique sur l'aire des taches, j'avais produit 16 tableaux de moments statistiques. Ils sont présentés sous forme condensée dans les Tables C.1, C.2, C.3 et C.4. Pour l'utilisation de l'érosion-dilatation, j'ai fixé la taille du *kernel* à 3.

Nb clusters : 2		**Nb clusters : 3**	
Connexité 8		Connexité 8	
Mean	1501.1	Mean	475.6
Std	11157.2	Std	4605.3
Med	69	Med	10
Skewness	14.1	Skewness	19.8
Kurtosis	253.7	Kurtosis	485.9
Nombre	30752	Nombre	48218
Connexité 4		Connexité 4	
Mean	1312.7	Mean	331.1
Std	10439.2	Std	3816.6
Med	39	Med	5
Skewness	15.1	Skewness	23.7
Kurtosis	290.8	Kurtosis	698.6
Nombre	35164	Nombre	69255

TABLE C.1 – Pas d'Érosion dilatation – Images non normalisées

Nb clusters : 2		**Nb clusters : 3**	
Connexité 8		Connexité 8	
Mean	2978.9	Mean	1827.4
Std	15687.3	Std	7815.3
Med	330	Med	184
Skewness	9.9	Skewness	8.4
Kurtosis	125.1	Kurtosis	88.6
Nombre	14991	Nombre	10067
Connexité 4		Connexité 4	
Mean	2974.1	Mean	1809.1
Std	15674.9	Std	7744.9
Med	330	Med	183
Skewness	9.9	Skewness	8.4
Kurtosis	125.3	Kurtosis	89.9
Nombre	15015	Nombre	10169

TABLE C.2 – Érosion dilatation $k = 3$ – Images non normalisées

C.2 Blobs et centroïdes

J'ai effectué un grand nombre de tests différents afin de trouver les paramètres les plus importants. Les tableaux C.5 à C.18 illustrent une fraction de l'ensemble des tests que j'ai pu réalisé.

Le script utilisé pour lancer les comparaisons d'images s'appelle run_comparisons.py, plusieurs paramètres permettent de contrôler le test de performance. Il peut notamment être exécuter avec ou sans multi-threading.

Nb clusters : 2		**Nb clusters : 3**	
Connexité 8		Connexité 8	
Mean	1210.6	Mean	398.4
Std	10898.9	Std	4408.5
Med	16	Med	7
Skewness	16.3	Skewness	21.6
Kurtosis	334.3	Kurtosis	576.9
Nombre	40874	Nombre	63085
Connexité 4		Connexité 4	
Mean	998.4	Mean	280.8
Std	9893.9	Std	3639.7
Med	7	Med	5
Skewness	17.9	Skewness	25.6
Kurtosis	406.6	Kurtosis	812
Nombre	49560	Nombre	89516

TABLE C.3 – Pas d'Érosion dilatation – Images Normalisées égal-hist

Nb clusters : 2		**Nb clusters : 3**	
Connexité 8		Connexité 8	
Mean	2998.6	Mean	1474.6
Std	17031.5	Std	6652.8
Med	274	Med	177
Skewness	10.2	Skewness	9.6
Kurtosis	131.1	Kurtosis	123.5
Nombre	15778	Nombre	13993
Connexité 4		Connexité 4	
Mean	2989.9	Mean	1464.4
Std	17006.7	Std	6622.6
Med	273	Med	177
Skewness	10.2	Skewness	9.7
Kurtosis	131.5	Kurtosis	124.7
Nombre	15824	Nombre	14091

TABLE C.4 – Érosion dilatation $k = 3$ – Images Normalisées égal-hist

Test 1	
threshold	0
Rayon hist	$r = \frac{1}{2n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	754 (277 ind. distincts)
Dataset	Tout
Érosion-dilatation	Non
Connexité	8
Matrice confusion	
1302	876
151914	129789
Precision	0.00849
Recall	0.598
Accuracy	0.462
Balanced Accuracy	0.529

TABLE C.5 – Test 1 – Rayon modifié

Test 2	
threshold	0
Rayon hist	$r = \frac{1}{2n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	394 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	Non
Connexité	8
Matrice confusion	
686	194
64423	12118
Precision	0.0105
Recall	0.779
Accuracy	0.165
Balanced Accuracy	0.469

TABLE C.6 – Test 2 – Rayon modifié – Dataset limité

Test 3	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	Non
Connexité	8
Matrice confusion	
314	557
28193	47964
Precision	0.011
Recall	0.36
Accuracy	0.627
Balanced Accuracy	0.495

TABLE C.7 – Test 3 – Rayon original – Dataset limité

Test 4	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 3$
Connexité	8
Matrice confusion	
346	525
32061	44096
Precision	0.0107
Recall	0.397
Accuracy	0.577
Balanced Accuracy	0.489

TABLE C.8 – Test 4 – Érosion-dilatation $k = 3$ – Dataset limité

Test 5	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 5$
Connexité	8
Matrice confusion	
389	482
32979	43178
Precision	0.0117
Recall	0.447
Accuracy	0.566
Balanced Accuracy	0.507

TABLE C.9 – Test 5 – Érosion-dilatation $k = 5$ – Dataset limité – $th = 0$

Test 5	
threshold	0.05
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 5$
Connexité	8
Matrice confusion	
337	534
29840	46317
Precision	0.0112
Recall	0.387
Accuracy	0.606
Balanced Accuracy	0.498

TABLE C.10 – Test 5 – Érosion-dilatation $k = 5$ – Dataset limité – $th = 0.05$

Test 6	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	60 (30 ind. distincts)
Dataset	Dummy
Érosion-dilatation	$k = 5$
Connexité	8
Matrice confusion	
30	0
186	249
Precision	0.139
Recall	1.0
Accuracy	0.6
Balanced Accuracy	0.786

TABLE C.11 – Test 6 – Érosion-dilatation $k = 5$ – Dataset naïf

Test 7	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 7$
Connexité	8
Matrice confusion	
576	295
51659	24498
Precision	0.011
Recall	0.661
Accuracy	0.326
Balanced Accuracy	0.491

TABLE C.12 – Test 7 – Érosion-dilatation $k = 7$ – Dataset limité

Test 8	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Non
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 3$
Connexité	8
Matrice confusion	
383	488
34277	41880
Precision	0.011
Recall	0.439
Accuracy	0.549
Balanced Accuracy	0.495

TABLE C.13 – Test 8 – Érosion-dilatation $k = 3$ – Dataset limité - Sans normalisation

Test 9	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Non
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 5$
Connexité	8
Matrice confusion	
383	488
34623	41534
Precision	0.011
Recall	0.439
Accuracy	0.544
Balanced Accuracy	0.493

TABLE C.14 – Test 9 – Érosion-dilatation $k = 5$ – Dataset limité - Sans normalisation

Test 10	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Non
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 7$
Connexité	8
Matrice confusion	
454	417
37078	39079
Precision	0.0121
Recall	0.521
Accuracy	0.513
Balanced Accuracy	0.517

TABLE C.15 – Test 10 – Érosion-dilatation $k = 7$ – Dataset limité - Sans normalisation

Test 11	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Non
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	393 (38 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	Non
Connexité	8
Matrice confusion	
321	550
29461	46696
Precision	0.0108
Recall	0.369
Accuracy	0.61
Balanced Accuracy	0.491

TABLE C.16 – Test 11 – Pas d'Érosion-dilatation – Dataset limité - Sans normalisation

Test 12	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	753 (103 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 5$
Connexité	8
Matrice confusion	
1194	975
125645	155314
Precision	0.00941
Recall	0.55
Accuracy	0.553
Balanced Accuracy	0.552

TABLE C.17 – Test 12 – Érosion-dilatation $k = 5$ – Dataset Tout - Avec normalisation

Test 13	
threshold	0
Rayon hist	$r = \frac{1}{n^2}$
Normalisation	Égal-hist
KMeans	CIELAB($c = [12], n = 2$)
Nb photos	753 (103 ind. distincts)
Dataset	Mandermatcher
Érosion-dilatation	$k = 5$
Connexité	8
Matrice confusion	
1576	593
183397	97562
Precision	0.00852
Recall	0.727
Accuracy	0.35
Balanced Accuracy	0.537

TABLE C.18 – Test 13 – Érosion-dilatation $k = 7$ – Dataset Tout - Avec normalisation

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**