UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

SEGMENTATION DE CAPTURES D'IMAGES AÉRIENNES APPLIQUÉE À LA NAVIGATION VISUELLE ASSISTÉE DE DRONES

Projet de mémoire présenté comme exigence partielle à la
Maîtrise en Sciences et Technologies de l'Information

PAR
Pier-Luc Proulx

Sous la direction de
Mohand Said Allili, Ph.D. (UQO)
Jean-François Lapointe, Ph.D. (CNRC)

JUILLET 2024

Table des matières

Table des figures

Liste des tableaux

Symboles

FCN	Réseaux entièrement convolutifs *(Fully Convolution Network)*
IOU	Intersection sur union *(Intersection Over Union)*
IPS	Images par seconde
MAE	Erreur moyenne absolue *(Mean Absolute Error)*
MSE	Erreur quadratique moyenne *(Mean Squared Error)*
RAG	Graphe de régions adjacentes *(Region Adjacency Graph)*
RVB	Rouge, vert, bleu
SLZ	Zones d'atterrissage sécuritaires (Safe Landing Zones)
SVM	Machine à vecteurs de support *(Support Vector Machine)*
UAV	Véhicule aérien sans humain à bord (*Unmanned Aerial Vehicle*)

Chapitre 1

Introduction & problématique

1.1 Généralités & Problématique

Les véhicules aériens sans pilote à bord (en anglais *unmanned aerial vehicles* ou UAV), familièrement appellés drones aériens ou simplement drones, sont une technologie qui a des applications diverses, autant dans les domaines militaire que civil. Dans le domaine civil, les drones peuvent être utilisés, entre autres, pour la surveillance, la livraison de matériel médical dans des zones difficiles d'accès, les premiers secours, et le divertissement [17, 31]. En général, les drones peuvent être classés en trois catégories principales en fonction de leur degré d'autonomie de pilotage. La première catégorie concerne les drones non-autonomes, qui nécessitent une intervention humaine constante pour le pilotage. La seconde catégorie englobe les drones à pilotage semi-autonome, qui peuvent effectuer certaines tâches de manière autonome, mais qui requièrent toujours une assistance humaine pour des décisions complexes. Enfin, la troisième catégorie concerne les drones à pilotage autonome complet, qui sont capables de prendre des décisions de manière autonome en se basant sur l'interprétation du milieu environnant [31].

L'automatisation du pilotage des drones représente un enjeu de recherche crucial, avec des implications significatives dans divers domaines tels que la logistique, la surveillance et les opérations de secours. Des études antérieures [25, 74, 4, 34, 43] ont suggéré l'utilisation croissante des drones pour des tâches de livraison, soulignant la nécessité d'une instrumentation précise comprenant des capteurs tels que des systèmes de positionnement par satellites GNSS (de l'anglais *Global Navigation Satellite System*), des radars et des caméras, pour permettre un vol efficace et sécuritaire. Néanmoins, l'utilisation de capteurs pose des défis majeurs, notamment en termes de poids supplémentaire supporté par les drones, qui peut limiter la capacité à transporter des charges

utiles plus lourdes. De plus, comme les capteurs utilisent des signaux, lesquels peuvent être retracés ou brouillés par diverses méthodes, il n'est pas toujours possible d'utiliser des capteurs afin d'évaluer l'environnement. Pensons, par exemple, au domaine militaire, dans lequel on doit souvent minimiser la quantité de données à transmettre.

En dépit des avancées technologiques et de la réduction du coût de production des drones, les règlements en place freinent leur utilisation dans des zones urbaines et/ou à grande densité de population. Le principal facteur expliquant des règlements est lié à la sécurité. Il s'agit à la fois de la sécurité des drones et de leurs environnements, incluant la population qui les entourent. Dans le cas des drones non-autonomes, il peut arriver que la communication entre le pilote et le drone soit interrompue abruptement en raison d'une perte de signal, de la condition de santé du pilote ou autres, laissant le drone dans une situation où il devra procéder à un atterrissage d'urgence, sans assistance. Dans de telles situations, il est primordial que le drone considère l'entièreté de son environnement, ainsi que la sécurité de celui-ci afin de prendre une bonne décision sur le meilleur endroit où il faut atterrir, en détectant les lieux d'atterrissage sûrs (en anglais *safe landing zones* ou SLZ)).

Alors que les capteurs offrent une bonne précision de mesure, permettant de potentiellement déterminer les lieux d'atterrissage sûrs, il est souvent nécessaire d'utiliser une fusion de plusieurs capteurs, tels que les Lidars, radars, gyroscopes et autres, afin d'obtenir une capture décente de l'environnement et de l'état du drone. De même, ces capteurs produisent une grande quantité de données qu'il est nécessaire de réunir et traiter dans le but d'obtenir une qualité de détails permettant de bien décrire les lieux. Comme la fusion de tous ces capteurs diminue la capacité de charge du drone, ce travail se concentre plutôt sur l'utilisation d'une caméra afin de faire la détection des lieux d'atterissage sécuritaires, comme la vision par ordinateur permet de regrouper, dans le meilleur des cas, le travail commun qu'effectue la fusion de tous ces capteurs coûteux. En effet, la vision à l'aide de caméra, combinée aux méthodes de traitement d'image et d'apprentissage machine, offre une bonne granularité de prises de vues pour des coûts financier et énergétiques inférieurs à ceux de l'utilisation de la fusion de capteurs. À partir d'une simple caméra, suivant l'utilisation de méthodes complexes, il est possible d'extraire, entre autres, les parties d'une image et leur positionnement, la distance de ceux-ci et les dangers potentiels.

C'est pourquoi, suivant cet ordre d'idées, la principale problématique traitée dans ce mémoire se trouve à être : comment peut-on détecter les lieux d'atterrissage sécuritaires (SLZ) en utilisant la vision par ordinateur, ainsi que des techniques de segmentation sémantique d'images et de l'apprentissage machine.

1.2 Organisation du mémoire

L'apprentissage, l'investigation et les résultats qui représentent l'ensemble de ce travail sont présentés à travers des sections de ce mémoire. Celui-ci présente d'abord un sommaire des travaux qui ont permis de dresser une base des termes et connaissances acquises. Par la suite, la méthodologie utilisée afin de produire une solution à la problématique est présentée, référençant les diverses contributions apportées, suivi par une liste exhaustive des résultats, supportés par une série de discussions relatives aux statistiques obtenues. Finalement, une conclusion présente les limites des expérimentations, ainsi que les potentielles ouvertures pour de futurs travaux.

Chapitre 2

État de l'art

Ce chapitre survole l'état des connaissances dans le domaine de la segmentation en général, ainsi que son apport au domaine du contrôle des véhicules. En raison du manque de jeux de données et de la nouveauté de la technologie, le domaine de la segmentation à partir d'images capturées par drones est peu représenté dans la littérature. Pour cette raison, ce chapitre traitera d'abord de l'historique de la segmentation, pour ensuite se concentrer sur les techniques de segmentation d'images aériennes plus précisément, étant plus présentes dans les recherches d'aujourd'hui.

Ce chapitre présente d'abord une mise en contexte de haut niveau quant à la segmentation, les domaines connexes, ainsi que les connaissances générales. Par la suite, un résumé des techniques et avancées utilisées dans le passé, ainsi que encore utilisées de nos jours pour certaines, est présenté. Dans un premier temps, les techniques par méthodes non-supervisées, n'impliquant pas d'annotation par un utilisateur dans la prise de décisions, sont définis. Dans un deuxième temps, les techniques semi-supervisées sont présentées, soit celles nécessitant à l'utilisateur de fournir un minimum d'information dans le but de converger plus rapidement et efficacement vers un résultat. Par exemple les contours actifs. Dans un troisième temps, les techniques supervisées nécessitant l'annotation des données sont présentées, convergent vers les pratiques courantes par l'utilisation de la segmentation à base d'apprentissage profond.

Finalement, les méthodes émergentes d'apprentissage machine appliquées à la navigation des drones sont présentées, permettant de créer un lien entre les techniques vues dans les sections précédentes et l'objectif principal de ce mémoire.

2.1 Introduction à la segmentation

Shapiro et Stockman définissent la segmentation comme étant la redéfinition d'une image en tant que les diverses parties qui la composent [79]. Ce regroupement peut être effectué de deux façons. Dans une première méthode, on tente de tracer les contours qui séparent chaque région homogène, de façon à pouvoir facilement les distinguer. Dans une seconde méthode, on regroupe plutôt les pixels homogènes afin de représenter les régions présentant des caractéristiques communes. En d'autres termes, à partir d'une image, il est nécessaire d'attribuer une valeur à chaque pixel formant ainsi, lorsqu'on les regroupe par valeurs égales, une partie. Puis, en regroupant chaque partie segmentée, on retrouve l'image originale, avec les éléments à catégories communes représentées de façon bien évidentes. Par exemple, tel que montré sur la figure 2.1 si on prend une photo d'une personne devant un paysage, la tâche de segmentation pourrait très bien consister en la scission de l'image en pixels représentant la personne, puis le restant du fond qui compose la photo.

FIGURE 2.1 – Capture montrant l'utilisaton de la segmentation afin d'isoler le sujet (chien) d'un paysage

Or, dans les applications modernes, telles que la segmentation d'images aériennes, déterminer les régions d'intérêt ne suffit pas pour comprendre l'environnement qui entoure le capteur. C'est pourquoi, les techniques modernes se sont plutôt orientées vers, certes la division de l'image en régions représentatives, mais aussi l'attribution d'étiquettes de classes à chacune de ces régions. C'est alors qu'est née la segmentation sémantique qui, comme dans l'exemple précédent, permettrait de dissocier les pixels déterminant une personne, de ceux représentant le ciel, de ceux représentant l'herbe, la route et encore plus. C'est avec ces connaissances qu'un drone peut différencier les obstacles, des espaces sécuritaires pour l'atterrissage. De plus, des recherches récentes

[90, 94, 5, 93] poussent plus loin la limite de la représentation de l'image en différenciant chaque instance d'une classe, les unes des autres, donnant ainsi une segmentation encore plus précise que la distribution de l'image en ses classes. On parle alors de la segmentation par instances. Dans l'exemple précédent, il s'agirait de séparer la personne du ciel, de la route et des autres classes importantes, mais aussi de différencier chaque nuage, chaque arbre l'un de l'autre, sachant que chaque instance représente un obstacle ayant des caractéristiques (e.g. vitesse, dimension) différentes les uns des autres. Somme toute, avec les progrès en intelligence artificielle, ces principes acquis redéfinissent la segmentation tel qu'on la connaît. Cependant, la segmentation n'est pas une notion découlant de l'intelligence artificielle, mais plutôt du domaine mathématique et statistique, tel que présenté dans la section suivante.

2.2 Segmentation par méthodes non-supervisées

L'une des premières méthodes de segmentation abordée dans l'histoire de la segmentation fût celle par regroupement des données, de façon non-supervisée.

Ne fournissant aucune information préliminaire à l'algorithme par rapport aux groupes à former et les caractéristiques définissant ceux-ci, les méthodes sans étiquettes initiales utilisent les valeurs des pixels (ex. : la segmentation par seuillage), les textures (ex. : division & fusion *[Split & Merge]*, voir section 2.2.4) et d'autres caractéristiques de l'image afin de former des régions homogènes.

2.2.1 Segmentation par seuillage [*Thresholding*]

Basé sur les méthodes mathématiques et la représentation des pixels sous un format d'histogramme, représenté à la figure 2.2, la méthode de segmentation par seuillage fut l'une des premières à être appliquée au domaine de la segmentation d'images à niveau de gris [37]. Cette technique demande la représentation de l'image sous forme d'histogramme à partir de la valeur de ses pixels. Puis, à partir de cet histogramme, il est nécessaire de déterminer, visuellement ou de façon algorithmique, une ou des valeurs pour lesquelles les valeurs des pixels seront divisées en deux ou plusieurs parties distinctes, soit les classes représentées dans l'image [79, 75].

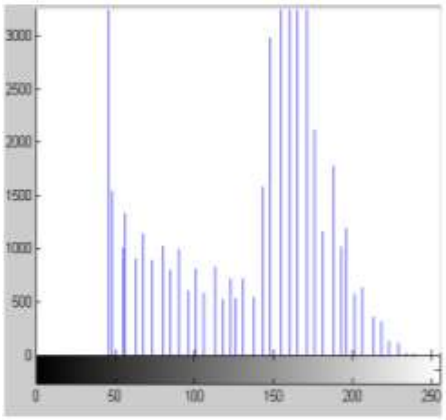


FIGURE 2.2 – Représentation d'un histogramme [44]

La technique de segmentation par seuillage peut-être décrite en deux étapes. Dans la première étape, on trouve une valeur de seuil T tel que $T \in I$, où I est l'intervalle des valeurs d'intensité des pixels, qui permet de séparer les modes en plus petits intervalles, représentant les classes. Pour la deuxième étape, on applique une fonction de seuillage s présentée à l'équation 2.1, sur chaque pixel, déterminant la classe à laquelle ce pixel appartient.

Dans le cas d'une segmentation binaire, soit dans un histogramme bimodal (c-à-d. présentant deux modes), dans laquelle on attribue souvent l'étiquette 1 aux traits faisant partie d'une première classe et 0 dans le cas contraire, on dénote la fonction de seuillage suivante appliqué à un pixel (x,y) pour lequel l'intensité est P(x,y) [65] :

$$s(x,y) = \begin{cases} 1, & si\ P(x,y) > T \\ 0, & autrement \end{cases} \tag{2.1}$$

Suivant cette logique, on peut en déduire sa forme générale, où il est possible de faire une segmentation multi-classes, on doit trouver *n-1* seuils T_{n-1}, définissants n classes distinctes, puis appliquer l'algorithme suivant [Algorithme 1] sur chaque pixel P(x,y) pour déterminer sa classe d'appartenance :

Suivant cet algorithme, les méthodes de segmentation par seuillage sont divisées en deux grandes catégories, soit le seuillage global et le seuillage local [58]. Dans le premier cas, la segmentation est faite sur l'image complète, alors que la technique est utilisée sur une partie de l'image dans la méthode locale.

Bien que les méthodes de seuillage donnent une bonne estimation de la segmentation binaire d'une image [40, 62, 11], cette méthode ne peut s'appliquer facilement dans les cas où l'histogramme ne présente pas de modes distincts [67]. De plus, cette technique

Algorithm 1: Algorithme de seuillage

```
counter ← 0;
for T ∈ T_{n-1} do
    if P(x, y) ≤ T then
        P(x, y) ← counter;
        break;
    end
    counter ← counter + 1;
end
```

ne considère que l'intensité des pixels dans la prise de décision, éliminant la possibilité que les pixels soient liés entre eux lorsqu'on divise l'image.

2.2.2 Segmentation par contours et sommets

Alors que l'approche par histogramme semblait bien fonctionner pour différencier un arrière-plan des traits importants d'avant-plan, Rosenfeld et Thurston [70], Canny [9] et plusieurs autres ont bâti, à travers leurs recherches, les fondations d'une nouvelle technique de segmentation plus adaptée à la segmentation d'images comprenant plusieurs traits importants, plutôt que la segmentation binaire, afin de remédier à la situation. Basé sur le principe de détection des changements dans l'intensité de la luminosité dans une image [79], la détection des sommets et contours a vu le jour, permettant d'isoler les sommets et contours qui définissent des objets dans une image. Ainsi, en distinguant les bordures des objets, il est plus facile visuellement de trouver les limites entre deux objets de même classe, améliorant ainsi la qualité de la segmentation produite.

2.2.3 Segmentation par ligne de partage des eaux

L'algorithme de segmentation par ligne de partage des eaux (*watershed algorithm*) basé sur le principe géographique des bassins versants qui sont formés à partir de l'eau qui s'écoule du haut des montagnes [82]. En s'écoulant, la dispersion de l'eau crée plusieurs bassins distincts, qui ont comme point commun le sommet de la montagne.

De ce principe, Digabel et Lantuéjoul [24] ont conçu un algorithme de segmentation par inondation, qui permet de retrouver la crête qui sépare deux bassins, soit la bordure qui sépare deux objets distincts dans le cadre de la segmentation d'images en imaginant l'image comme une topographie dans laquelle les pixels avec la plus haute intensité ont une altitude plus grande que ceux avec une intensité plus petite.

La première étape de l'algorithme consiste à effectuer une transformation de distance sur l'image originale transformée sous valeurs de gris, permettant d'obtenir les minimums de l'image, soit les bassins. La deuxième étape consiste à inonder les bassins, soit augmenter la superficie occupée par chaque bassin dans l'image, jusqu'à ce qu'il y ait intersection entre les bassins. Ces intersections représentent ainsi la bordure qui délimite deux objets, démontrant ainsi le contour qui sépare chaque objet distinct de l'image, tel qu'illustré à la figure 2.3.

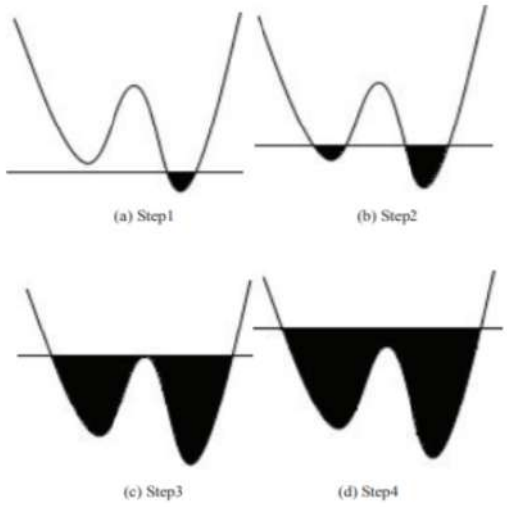


FIGURE 2.3 – Étapes de segmentation par ligne de partage des eaux [48]

Cette méthode apporte une solution au problème causé par les techniques de seuillage qui combinent plusieurs instances d'objets de même classe en un seul, représentant ceux-ci comme un seul et unique objet, alors qu'il pourrait y en avoir plusieurs à une faible distance. Par exemple, plusieurs boutons regroupés et posés sur une table ne représentent qu'un amas d'objets collés dans une segmentation par seuillage, alors que l'algorithme *watershed* retourne plutôt une segmentation dans laquelle on représente chaque bouton séparément à l'intérieur du groupe. Bien qu'elle soit plus efficace que bien d'autres méthodes, elle ne permet cependant pas d'apposer une étiquette sur chaque instance qui a été segmentée, ce qui est primordial lors de la segmentation d'un environnement par drone pour déterminer les zone d'atterissage sécuritaires.

2.2.4 Segmentation par division et fusion *[Split-and-Merge]*

La plupart des méthodes de segmentation démontrent des techniques de croissance qui commencent à partir de régions très petites, soit l'équivalent d'un seul pixel, puis

qui regroupent les pixels présentant des caractéristiques communes afin de former des régions homogènes. De plus des techniques qui cherchent les différences entre deux groupes de pixels afin d'établir les lignes et courbes directrices sépare les régions, soit les contours.

La technique de segmentation par division et fusion *[split-and-merge]* utilise plutôt la technique de la dé-construction de l'image en sous-sections afin de mieux rassembler les régions homogènes par la suite [46], soit effectuer la segmentation en deux étapes : la division et la fusion.

Basé sur la structure de données *Quadtree* [82, 92], l'idée principale de cette technique se divise en deux parties, soit la division, suivi de la fusion. Dans la première partie de l'algorithme, on définit l'image entière comme racine de l'arbre, soit une seule région. On calcule ensuite une fonction d'homogénéité, comme l'écart-type de l'intensité des pixels présents dans la région par exemple. Si le résultat de cette fonction appliquée à la région est plus grand qu'un certain seuil déterminé, le noeud, définissant la région évaluée, n'est pas décrit comme homogène et est divisé en quatre sous régions, représentant de nouveaux noeuds dans l'arbre pour lesquels la région évaluée précédemment est le parent. Suivant cette lignée, l'algorithme continue le processus, de façon récursive, jusqu'à l'atteinte d'un critère d'arrêt (Par exemple, la profondeur maximale de l'arbre, le nombre de pixels à évaluer dans chaque section, etc.) [16].

Dans la deuxième partie, lors de sa remontée dans l'arbre, l'algorithme compare la similarité des régions créées dans la partie de division entre elles afin de les fusionner. On crée alors un graphe de régions adjacentes *(Region Adjacency Graph [RAG])*, dont un exemple est esquissé dans la partie *(b)* de la figure 2.4, afin d'évaluer l'homogénéité des régions contiguës.

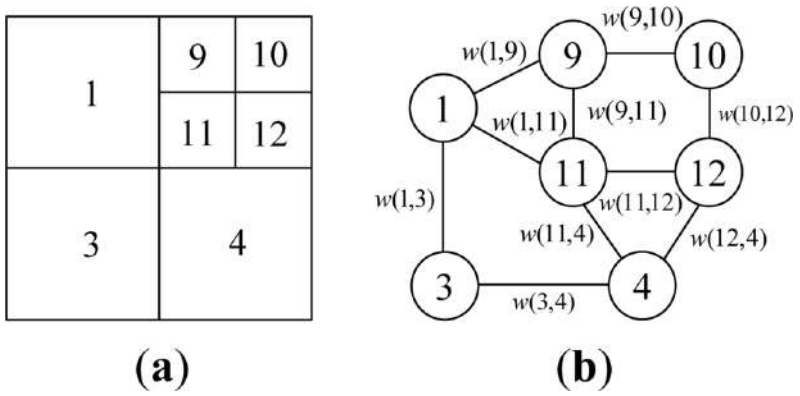


FIGURE 2.4 – Segmentation basée sur l'algorithme split-and-merge et la représentation sous forme d'un Quadtree [33] **(a)** Séparation d'une image en sous régions lors de la phase de division. **(b)** Représentation d'un graphe des régions adjacentes *RAG* présente dans la phase de fusion, tiré de l'image divisée en ***(a)***

Ayant comme noeuds chaque feuilles du Quadtree, soit les régions divisées au maximum par la condition d'arrêt ou l'homogénéité de celles-ci, le graphe présente les liens qu'il existe entre deux régions adjacentes, reliés par un segment contenant un poids w, calculé à partir d'une fonction déterminée par l'utilisateur (Par exemple, la différence de moyenne de l'intensité). Ainsi, en appliquant un seuil, il est possible de jumeler deux régions homogènes, lesquelles représentent les noeuds du *RAG*, si elles ont un poids inférieur au seuil défini.

Ainsi, en simplifiant le graphe, il est possible de rebâtir l'image originale, segmentant les régions présentant des textures similaires, lesquelles ont été trouvées durant l'étape de fusion. Cette méthode de segmentation, encore utilisée de nos jours [10], tente de démontrer des similitudes entre les pixels et régions inter-connectées se basant sur la notion de texture locale, plutôt que de traiter les valeurs d'intensité des pixels présents dans l'image. Cette technique démontre des résultats concluants [83, 28]. Cependant, comme les graphes produits grâce à cette technique sont très grands, ils nécessitent une puissance de traitement importante. De plus, bien que ceux-ci distinguent les segments d'une image, ils n'apportent pas de notion sémantique à la segmentation, ne jumelant que les régions homogènes, sans contexte sur le lien qui les relie, mise à part l'intensité des pixels.

2.2.5 L'algorithme K-moyennes

L'algorithme K-moyennes demeure l'un des plus utilisé et l'un des plus rapide du domaine de la segmentation simple. Le but de cet algorithme, à la base, est de di-

viser l'image en K groupes distincts, définis ou non par l'utilisateur de l'algorithme, présentant des caractéristiques communes. Dans le cas de l'image, on utilise la plupart du temps la valeur des pixels par couleur ou par intensité afin de créer des groupes distincts, tel qu'illustr à la figure 2.5.

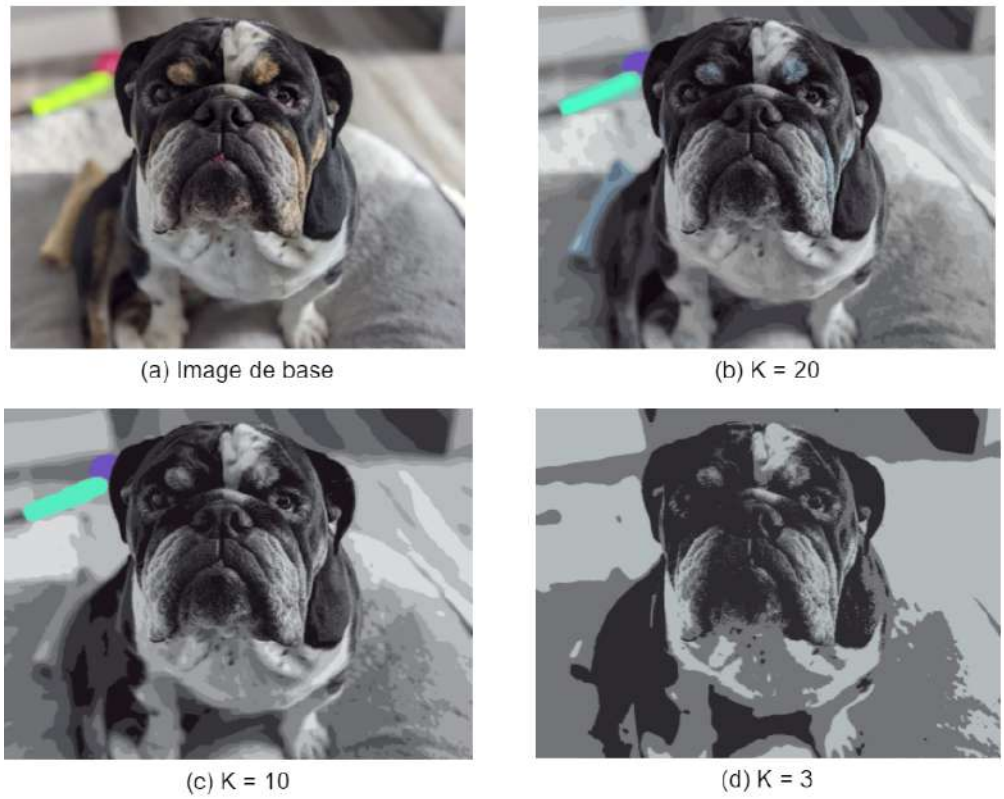


FIGURE 2.5 – **(a)** Image originale ; **(b)** Image segmentée en 20 groupes *(c-à-d. K = 20)* **(c)** Image segmentée en 10 groupes *(c-à-d k = 10)* **(d)** Image segmentée en 3 groupes *(c-à-d k = 3)*

Afin de déterminer ces groupes, cette technique défini K centroïdes aléatoires dans l'univers des caractéristiques et tente d'optimiser l'emplacement des points grâce à l'algorithme défini par la série d'instructions suivantes, dans sa forme la plus simple [79, 21] :

1. Trouver K points aléatoires définis comme les moyennes
2. Attribuer un groupe à chaque pixel selon la distance entre ses caractéristiques et celles des K points choisis initialement
3. Recalculer la valeur du point moyen de chaque groupe pour trouver les nouveaux points K
4. Recommencer la première étape jusqu'à atteindre une condition (e.g. Nombre d'itérations, précision, etc.)

Dans ses versions simples, l'algorithme nécessite l'entrée du nombre de valeurs K désirées, le catégorisant comme semi-supervisé. Or, de nouveaux algorithmes comme DBSCAN [26] et OPTICS [8] ont fait en sorte de retirer la nécessité de fournir le nombre K de groupes désiré, tentant de former d'eux-même le nombre de groupes nécessaires afin de segmenter l'image.

2.3 Segmentation par méthodes semi-supervisées

Bien que les méthodes de segmentation par regroupement des données non-supervisées présentent des résultats concluants pour l'oeil humain, elles ne distinguent pas les classes entre elles autrement que de façons binaires ou limitées. Comme elles n'ont pas d'information sur les liens sémantiques qui existent entre les pixels, les résultats produits présentent souvent des défauts et des imperfections qui lient des classes sémantiquement dissociables de façon incohérente.

Pour cette raison, l'ajout d'information connue de l'utilisateur dans l'algorithme, par exemple l'emplacement initial idéal d'un point dans l'algorithme des K-Moyennes, permet de faire converger celui-ci beaucoup plus rapidement que la recherche dans un large intervalle de solutions. Il s'agit ici-même d'une généralisation du regroupement semi-supervisé, soit l'ajout d'entrées produits par un utilisateur, sans connaître la solution au problème de regroupement, qui permet de faire converger l'algorithme plus rapidement.

2.3.1 Segmentation par contours actifs

La segmentation par contours actifs se base sur la notion de contour actif, soit une courbe dont la forme s'adapte en fonction de paramètres. Le but premier de cette technique est de segmenter l'image, de façon ascendante ou descendante, en ses grands contours, tel que qu'effectué par Rivest-Henault et al. [68] lors la segmentation de petits vaisseaux sanguins représentés à la figure 2.6.

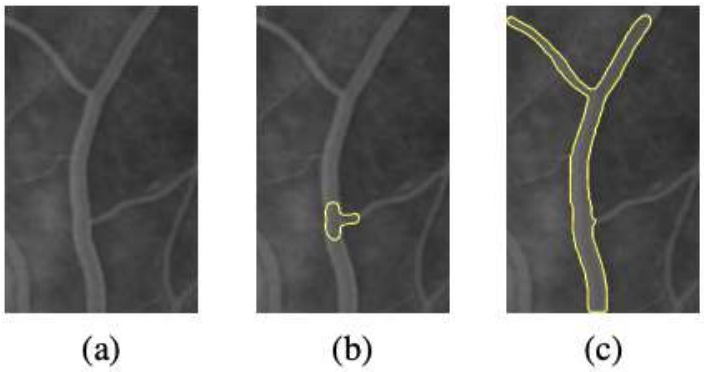


FIGURE 2.6 – **(a)** Image originale ; **(b)** Contour initialisé **(c)** Contour après plusieurs itérations

À partir d'une ou de plusieurs courbes initiales, dont la forme est basée sur une estimation ou une sélection, l'algorithme tente de minimiser une fonction d'énergie qui dicte la structure des courbes. Dans un premier temps, une fonction d'énergie interne $E_{interne}$ régit la déformation générale de la courbe, en se basant sur la souplesse et la continuité du contour. Dans un deuxième temps, une fonction d'énergie externe $E_{externe}$ s'occupe d'aligner la courbe aux frontières des régions de l'image, en se basant sur le gradient de l'intensité et d'autres caractéristiques propres à l'image. [49, 73]

Soit l'énergie totale du système définie par l'équation 2.2, basée sur les fonctions d'énergie internes et externes [49] :

$$E_{contour} = \int_0^1 E_{interne}(v(s)) + \int_0^1 E_{externe}(v(s))\, ds \tag{2.2}$$

Par méthode d'itérations, l'algorithme tente de diminuer les fonctions d'énergie et de modifier le contour, par le fait-même. On peut donc trouver le contour C, déformé par les fonctions d'énergies $\mathrm{E_{interne}}$ et $\mathrm{E_{externe}}$, lorsque les fonctions d'énergies internes et externes sont à leur plus petite valeur, tel que représenté à la troisième image **(c)** de la figure 2.6.

2.4 Segmentation par méthodes supervisées

2.4.1 Segmentation par arbre de décisions et forêts aléatoires

Les arbres de décision sont des modèles d'apprentissage qui se basent sur la structure de données des arbres de recherche [60]. Basé sur les probabilités, cette technique

consiste à bâtir un arbre pour lequel les noeuds représentent les caractéristiques et les arêtes de chaque noeud l'étendue des valeurs possibles que peuvent prendre ces caractéristiques, apportant vers un prochain noeud. À ses extrémités, cet arbre présente des feuilles qui représentent les prédictions finales, soit la sortie du système.

À partir des entrées du système, l'utilisateur peut choisir l'arête à suivre pour chaque décision en fonction de la valeur de l'entrée, pour ensuite arriver à une feuille représentant la prédiction de sortie du système en fonction des choix faits. Ainsi, en tentant d'augmenter une fonction de gain, l'algorithme bâtit un arbre, en choisissant un ordre de noeuds précis, permettant d'arriver le plus rapidement possible à une feuille, tel que représenté à la figure 2.7 tirée du livre de Breiman et Al. [6].

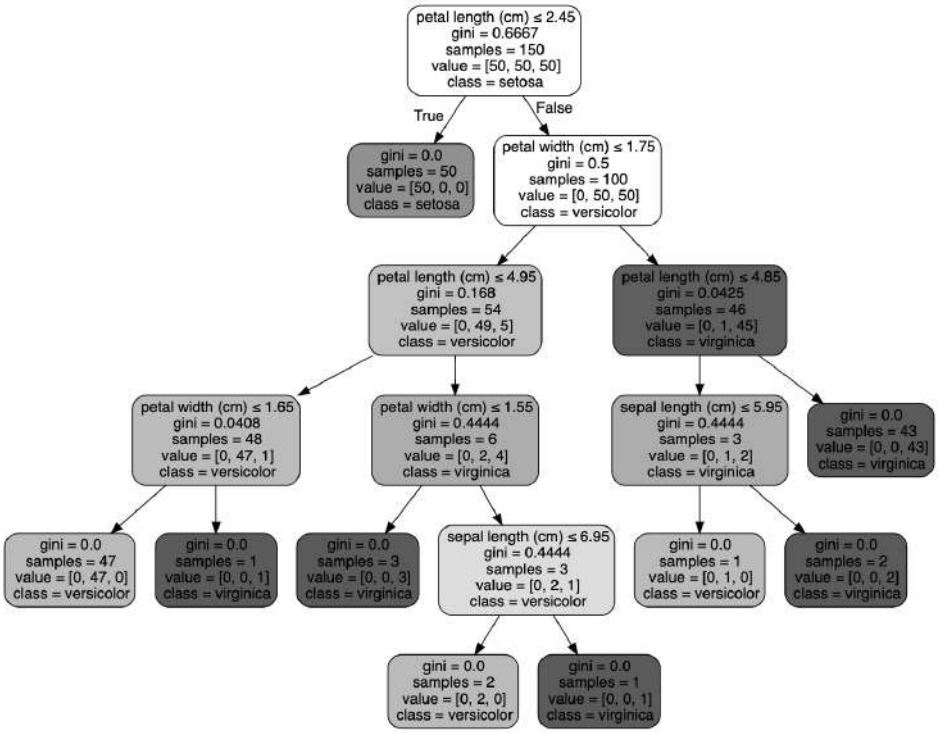


FIGURE 2.7 – Arbre de décision basé sur le jeu de données Iris (1936) [30]

Afin de bâtir l'arbre, l'algorithme calcule la valeur d'entropie, décrite à l'équation 2.3, de chaque classe S à partir d'un noeud et tente de diminuer celle-ci, de façon à réduire le désordre total dans l'arbre.

$$Entropie(S) = \sum_{n=1}^{c} -P_i log_2 p_i \tag{2.3}$$

En prenant en entrée les caractéristiques d'un pixel (ex. les valeurs des pixels, l'intensité, etc.), il est possible de bâtir un arbre de décision qui déterminera la classe

à attribuer à un pixel, produisant une image segmentée par le fait-même [36]. Tentant d'améliorer les résultats obtenus à l'aide des arbres de décisions, l'idée d'utiliser les forêts aléatoires pour la segmentation est devenue chose courante. Créant plusieurs configurations d'arbres de décisions différentes, une forêt aléatoire est une structure d'apprentissage machine qui basera son choix final sur l'apport des décisions de chaque arbre, en attribuant une portion du choix final à chaque arbre. De cette façon, plutôt que d'obtenir le résultat d'un seul arbre et s'approcher d'un sur-apprentissage, la forêt aléatoire permet d'obtenir l'opinion de plusieurs arbres et de baser sa décision sur la classe ayant été attribuée par la majorité des arbres qui composent la forêt [53, 80].

2.4.2 Machine à vecteurs de support (SVM)

Les machines à vecteurs de support (SVM) sont des modèles d'apprentissage supervisés qui, dans un espace de caractéristiques, tentent de définir des vecteurs et plans qui divisent en deux groupes distincts les valeurs des caractéristiques [22]. Dans l'exemple de la segmentation d'images, on peut distinguer une image d'une autre par ses caractéristiques, tel que les couleurs et l'intensité des pixels, la luminosité, sa texture et bien plus. Ainsi, avec les SVM, il est possible d'entraîner un modèle qui segmente l'image, de façon binaire, en prenant en entrée des vecteurs de caractéristiques représentant l'image.

L'algorithme tente de déterminer mathématiquement l'équation d'une droite, ou d'un plan dans le cas d'une entrée à plusieurs caractéristiques, qui diviserait les caractéristiques en deux groupes. De cette façon, en apportant une image inconnue, il serait possible de déterminer, en fonction de l'équation de la droite ou du plan trouvée, si un pixel fait partie d'une première classe pré-déterminée ou de la deuxième, par exemple : un arrière-plan d'un humain [29].

Cette méthode présente de bons résultats en soi, tel que démontré par McGee et al. [59] dans leur projet où ils tentent de produire une segmentation de l'image pour évitement d'obstacles dans le domaine de la conduite de véhicules aériens sans pilote (UAV). Ceux-ci se sont servis d'un modèle SVM afin d'appliquer un filtre sur chaque pixel représentant la ligne de division entre le sol et le ciel. De cette façon, tout pixel qui n'est pas décrit comme un pixel de classe ciel, mais qui se retrouve tout de même au-delà de la ligne d'horizon entre le sol et le ciel était considéré comme un obstacle à l'avion. Cette technique s'est avérée un succès, tel qu'illustré à la figure 2.8.

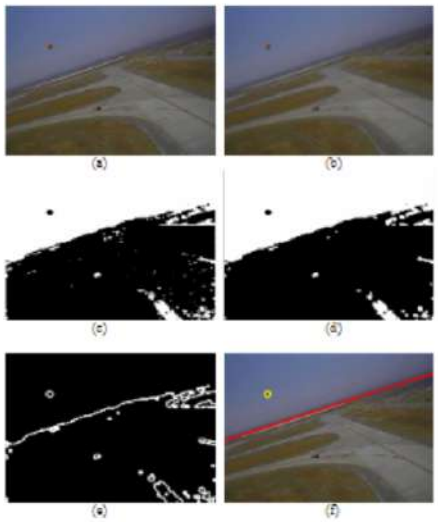


FIGURE 2.8 – **(a)** Image originale **(b)** Image embrouillée **(c)** Image binaire segmentée par SVM **(d)** Image segmentée dilatée et érodée **(e)** Séparation du sol et du ciel **(f)** Segmentation de l'horizon et des obstacles dans le ciel [59]

Cependant, comme les SVM sont robustes pour les classifications binaires, il s'avérerait difficile et coûteux en puissance de calculs d'implémenter cet algorithme dans le domaine de la livraison par drone, sachant qu'il peut y avoir un grand nombre de classes dans une vue en livraison dans un environnement urbain. Par exemple, les routes, les voitures, les humains, les bâtiments, etc.

2.5 La segmentation à base d'apprentissage profond

Avec l'arrivée de l'apprentissage profond, la segmentation d'images se jumela au contexte de celle-ci à travers l'apprentissage supervisé et non-supervisé, créant des méthodes modernes connues et performantes. Débutant par des méthodes qui permettent de répondre à des tâches de classification, tel que déterminer la présence d'un chat dans une image, ces méthodes ont évoluées, permettant de classer chacun des pixels en classes, représentant ainsi la segmentation de l'image.

Ce chapitre traite d'abord des réseaux de classification qui identifient les objets principaux dans les images ainsi que les classes de celles-ci. Dans un deuxième temps, les principales architectures des réseaux utilisés pour la segmentation sont présentées et décrites par ordre chronologique d'apparition. Finalement, celles-ci sont comparées en fonction des améliorations apportées et de leur résultat sur le jeu de données PASCAL VOC 2012 [27].

2.5.1 Réseaux de classification

Derrière chaque réseau permettant de faire de la segmentation se cache une architecture utilisée précédemment pour la tâche de classification dans sa phase d'encodage. La section suivante présente les quatre grandes architectures de classification qui sont utilisées afin de bâtir les réseaux de segmentation, soit les architectures AlexNet [54], VGG-16 [78], GoogleNet [81] et Resnet[42].

a) AlexNet

Défini par Krizhevsky et al. [54], l'architecture AlexNet, présentée à la figure 2.9 est un réseau de neurones convolutionnel qui effectue la tâche de classification.

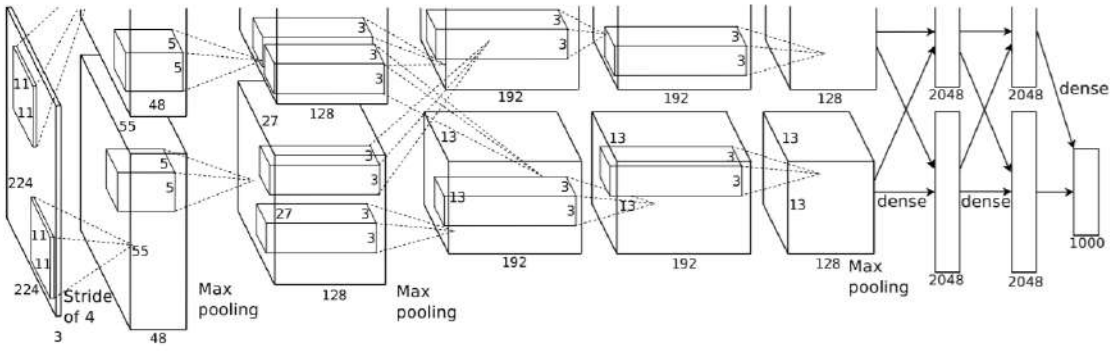


FIGURE 2.9 – Architecture du réseau AlexNet [54]

Ce réseau prend une image d'entrée de taille 224 x 224 pixels sous son format RVB à trois canaux en entrée. Au premier niveau, il effectue une convolution avec un filtre de 11 pixels x 11 pixels à pas de 4 pixels afin d'apprendre les détails de l'image, puis passe par une couche de mise en communs par maximum avec un filtre de grandeur 3 x 3 pixels à pas de 2 pixels. Cette étape permet notamment d'augmenter le champ réceptif du réseau, permettant à ce dernier de visualiser les détails de plus haut niveau par la suite.

Suivant cette étape, l'architecture ajoute une couche de convolution à filtre de 5 x 5 pixels à pas de 2 pixels, puis une couche de mise en communs par maximum avec un filtre de grandeur 3 x 3 pixels avec un pas de 2 pixels, pour une fois de plus augmenter le champ réceptif et déterminer les détails de plus haut niveau.

Pour son dernier apprentissage des détails de l'image, le réseau compte trois couches convolutionnelles avec filtres de 3 x 3 pixels avec un pas de 1 pixel, ainsi qu'une couche de mise en communs de filtre 3 x 3 pixels avec pas de 2 pixels.

Cette première partie, représentant une série de convolutions et de mises en commun, permet au réseau de visualiser les détails précis de l'image dans la première phase, puis d'augmenter le champ visuel [56] de celui-ci. Cela permet de déceler les détails de plus haut niveau vers la fin, lorsque celui-ci doit prendre sa décision sur la classification.

Dans sa deuxième section, la sortie de la première section du réseau s'aplatit sous un vecteur de 6400 pixels, puis passe à travers deux couches denses de 4096 neurones chacune pour effectuer une tâche de classification. L'un des détails important du réseau est qu'il a d'abord été conçu pour rouler sur deux processeurs graphiques, séparant chacune des étapes du réseau en deux parties, puis les combinant tout au long de l'apprentissage [54], tel que représenté à la figure 2.9

b) VGG-16

Suite au succès de l'architecture AlexNet, Simonyan et Zisserman définissaient, en 2014, deux architectures de réseaux convolutifs qu'ils appellent Visual Geometry Group 16 (VGG-16) et Visual Geometry Group 19 (VGG-19) [78] en référence au groupe de recherche de l'université Oxford et le nombre de couches qui contiennent des paramètres à entraîner, soit 16 dans le premier et 19 dans le dernier. L'architecture de base du modèle VGG-16 est illustrée à la figure 2.10.

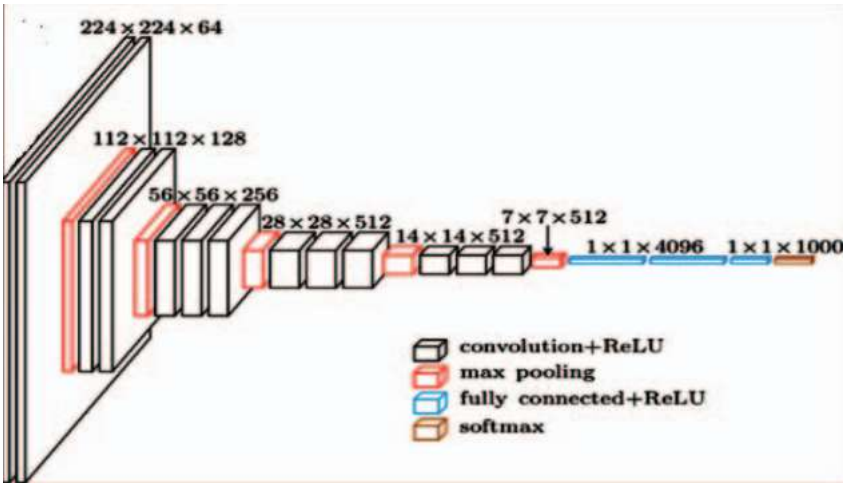


FIGURE 2.10 – Architecture de base du VGG-16 [50]

Dans sa forme la plus simple, l'architecture VGG-16 prend en entrée une image de 224 pixels de hauteur et 224 pixels de largeur, sous la forme RVB ou niveaux de gris, puis fait passer celle-ci à travers une série de couches profondes connectées B sous forme d'entonnoir.

Dans chacun des blocs B, l'image est d'abord analysée par deux couches de convolution à filtres de grandeur 3 x 3 pixels se déplaçant d'un pas de un pixel. Cette technique permet d'apprendre, à tous les niveaux de l'entonnoir, des caractéristiques importantes de l'image d'entrée qui est transmise à travers le réseau convolutionnel.

Afin de permettre au réseau d'apprendre non-seulement les détails précis de l'image d'entrée, mais aussi les formes plus spécifiques et générales, chaque bloc B se voit terminer par une couche de sous-échantillonage *(Pooling)*, étant dans la plupart des cas choisie comme une mise en commun par sélection du maximum *(MaxPooling)*.

De cette façon, en émettant 5 blocs, comprenant respectivement 64, 128, 256 et 512 filtres, connectés les uns à la suite des autres, le réseau apprend d'abord à reconnaître les traits spécifiques d'une image avec un champ réceptif assez petit au départ, mais se spécialise dans les détails plus complexes et de haut niveau de l'image vers la fin, présentant alors un champ réceptif plus grand qu'au départ [56].

Une fois les convolutions terminées, tout comme pour le réseau AlexNet, il s'aplatit pour se raccorder à deux couches denses de 4096 neurones chacune, entièrement connectées. Puis, à la sortie, les couches denses passent à travers une couche d'activation à fonction *softmax*, émettant une tâche de classification simple, qui active les neurones présentant les caractéristiques désirées d'une image, par exemple la présence d'un chat dans une image [71, 78].

Ce réseau, bien qu'il ressemble à l'architecture du Alexnet, utilise des filtres de taille 3 x 3 pixels à pas de 1 pixel, démontrant un champ réceptif beaucoup plus petit dans les premières couches du réseau, contrairement au AlexNet qui débute avec un filtre de 11 x 11 pixels à pas de 4 pixels, suivi d'un filtre de 5 x 5 pixels à pas de 2 pixels. Ce détail fait en sorte que le réseau VGG apprend des détails plus fins, de façon moins rapide que le réseau AlexNet, augmentant son exactitude au final. Cependant, celui-ci présente près du double de paramètres à entraîner, demandant une puissance ou un temps de traitement supérieur à AlexNet.

c) GoogleNet

Suite à l'étude des architectures existantes, un problème majeur a été décelé. Les réseaux, enchaînant plusieurs couches convolutionnelles les unes à la suite des autres, font exploser les demandes en calcul de façon exponentielle, ne donnant pas des résultats optimaux. Afin de régler la situation, la couche d'inception fut inventée, combinant plusieurs convolutions à l'intérieur d'un même bloc, lesquels sont entraînés en parallèles, mais combiné à la fin tel que présenté sur la figure 2.11. Basé sur la réduction du

nombre de paramètres à entraîner et la notion d'inception, Szegedy et al. [81] ont créé l'architecture GoogleNet.

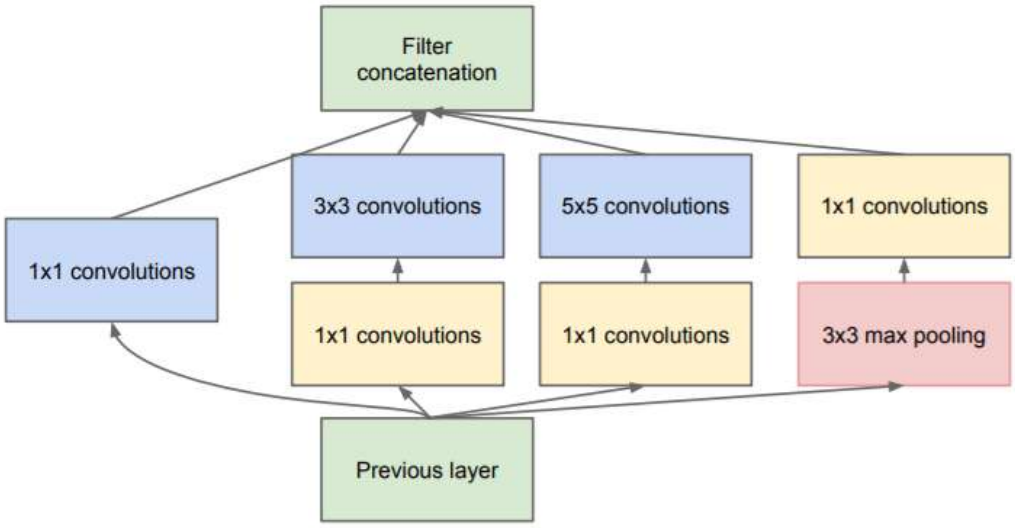


FIGURE 2.11 – Une couche d'inférence du GoogleNet [81]

Les groupes de pixels sémantiquement reliés étant quelques fois dispersés, alors que d'autres fois très près, cette couche se veut innovatrice, car elle permet de découvrir les groupes relativement près avec les filtres de petites tailles, tel que 1x1 et 3x3, alors qu'elle permet de trouver les groupes plus dispersés avec ses filtres 3x3 et 5x5. De plus, les réseaux actuels supportant positivement l'apport des couches de mises en commun pour augmenter le champ réceptif à la suite de plusieurs couches, l'ajout d'une couche de mise en commun avec filtre de 3x3 permet de rapporter ce point à la prise de décision de la couche d'inception.

De plus, comme les opérations avec filtres de 3x3 et 5x5 demandent énormément de puissance de calcul, l'insertion d'une couche de convolution à filtre de 1x1 permet de réduire la dimensionalité de la couche précédente avant de l'apporter à la convolution à filtre large, permettant de diminuer le nombre de paramètres entraînables et donc le nombre de calculs nécessaires à l'entraînement.

Afin de bâtir le réseau, ils ont superposé plusieurs couches convolutionnelles traditionnelles pour des raisons de performances, suivi de couches d'inception qui voient le nombre de filtres à chaque couche monter de 256 à 1024 à la fin, tel que démontré sur la figure 2.12. Par ailleurs, ceux-ci ont utilisé trois sorties, calculant ainsi trois fonctions de perte, de façon à obtenir baser leur décision sur les premières caractéristiques obtenues par le réseau, celles intermédiaires et les caractéristiques finales.

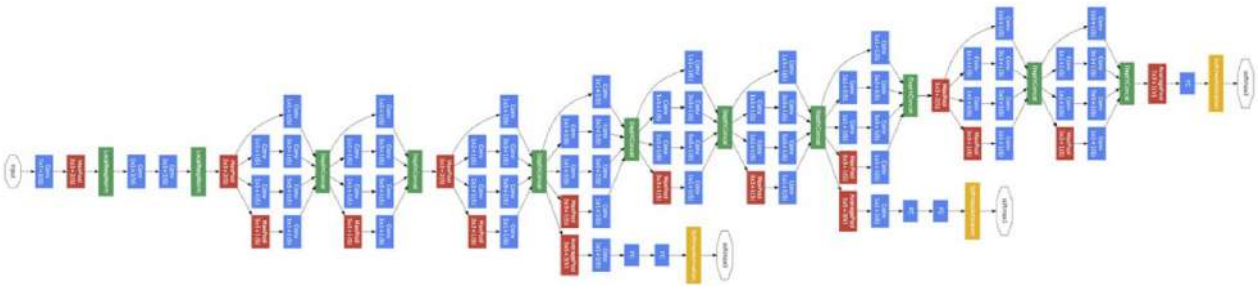


FIGURE 2.12 – Architecture du réseau GoogleNet [81]

Somme toute, le réseau fut plus performant que ses compétiteurs à la compétition ILSVRC14 [81], comprenant seulement un total d'environ 7 millions de paramètres à entraîner.

d) ResNet

Le gradient fuyant (*vanishing gradient*) est l'un des problèmes majeurs des réseaux convolutionnels profonds qui touche les architectures existantes. En effet, l'ajout de plusieurs niveaux comprenant des fonctions d'activations fait en sorte de diminuer la valeur de la dérivée lors de la propagation arrière du réseau. De ce fait, les poids et les biais du réseau ne sont pas optimisés à chaque entraînement, faisant en sorte que l'apprentissage est ralenti, ou le réseau tombe dans un état de plateau au niveau de son apprentissage [45].

Afin de contrer le problème, He et al. [42] ont bâti une nouvelle architecture basée sur les connexions sautantes (*skip connection*) nommée ResNet (*Residual Network*), construit à partir de blocs résiduels. Le pilier de l'architecture, présenté à la figure 2.13, est nommé bloc résiduel. Soit une entrée x, à laquelle on fait subir une transformation $F(x)$, basée sur des couches convolutionnelles. La sortie du bloc est l'addition de l'entrée, sautant la transformation, et de l'entrée transformée, soit $F(x) + x$.

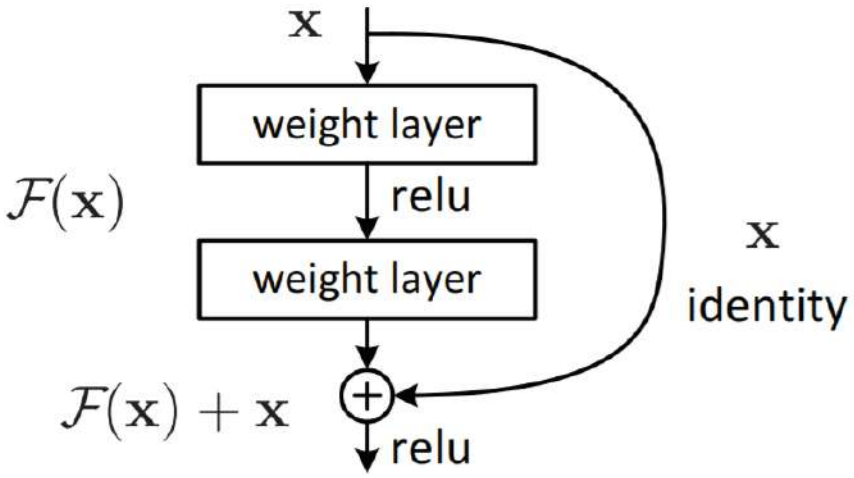


FIGURE 2.13 – Structure d'un bloc résiduel [42]

Ce bloc permet de laisser passer le gradient, sans appliquer de modifications ou de fonction d'activation à celui-ci, diminuant ainsi la répercussion de la disparition du gradient sur le réseau en entier.

En juxtaposant ainsi plusieurs blocs les uns aux autres, on obtient un réseau résiduel à N couches. Cependant, le réseau à 18 couches, présenté sur la figure 2.14, est l'un des plus utilisés, comptant seulement 12 millions de paramètres à entraîner, soit 50 millions de moins que l'architecture AlexNet, 10 fois moins que le réseau VGG-16 et seulement 5 millions de plus que le GoogleNet.

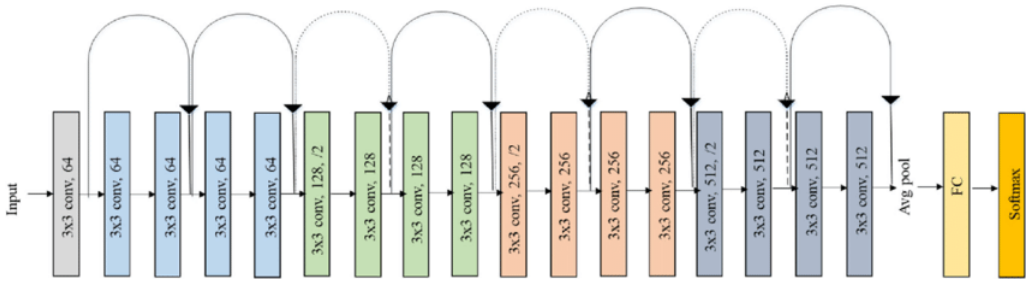


FIGURE 2.14 – Architecture du réseau ResNet-18 [42]

2.5.2 Réseaux de segmentation

Les réseaux de segmentation sont, en majorité, définis comme deux blocs connectés entre eux, soit la phase d'encodage et la phase de décodage. Basé sur les réseaux de convolution normaux, ceux-ci remplacent la sortie dense de chacun par une phase de

décodage à base de suréchantillonnage (*upsampling*) et de déconvolution. On nomme ces réseaux des réseaux entièrement convolutifs (*FCN : Fully Convolutionnal Networks*).

Dans un premier bloc, soit la phase d'encodage, le réseau prend une image en entrée, puis réduit la taille d'une image, en augmentant sa dimensionnalité pour apprendre un maximum de caractéristiques sur l'image d'origine. Par exemple, les réseaux AlexNet, VGG-16, GoogleNet et ResNet représentent des architectures propices à être utilisées dans la phase d'encodage, comme elles encodent l'image d'entrée sous un format plus simple à travers un apprentissage des caractéristiques de haut et de bas niveaux. Cette étape permet notamment d'identifier et classer les objets et caractéristiques principales de l'image.

Dans un deuxième bloc, basé sur des couches de suréchantillonnage et de déconvolution, le réseau tente de décoder l'image encodée précédemment, de façon à obtenir l'image originale, séparée en n classes, soit l'image d'origine segmentée. Cette étape permet d'identifier avec précision, soit pixel par pixel, les objets qui ont été identifiés dans la première phase d'encodage.

Cette combinaison d'encodage et de décodage représente l'architecture de base de tout réseau bâti pour répondre à la tâche de segmentation. Cette section démontre les principales architectures qui ont été utilisées jusqu'à présent, démontrant les modifications apportées entre elles.

a) FCN-X

En 2014, Long et al. [55] définissent les architectures FCN-X *Fully Convolutional Network* dans ses formes 32, 16 et 8, se basant sur l'architecture de classification déjà existante VGG-16 [78].

En se basant sur les principes des architectures déjà existantes, cette nouvelle architecture, démontrée dans ses trois formes sur la figure 2.15, propose de faire passer une image à travers plusieurs couches convolutives à filtres différents, en augmentant son champ réceptif à l'aide de la mise en commun (*pooling*). Comme dans les cas précédents, elle opte pour des couches dites de suréchantillonnage *(Upsampling)* ou de déconvolution, plutôt que des couches denses à la sortie. Cette technique permet de produire une image segmentée plutôt qu'effectuer une tâche de classification.

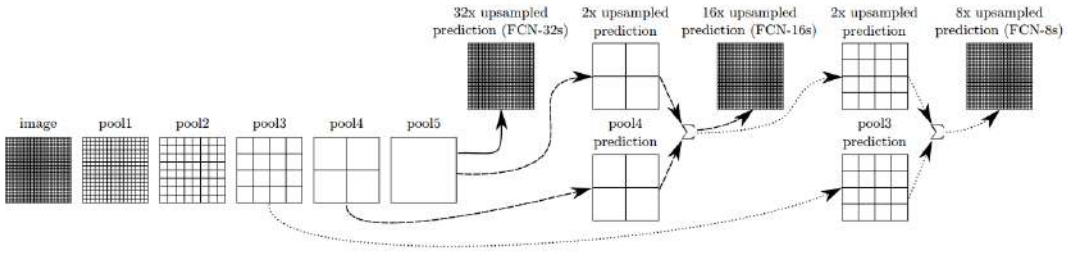


FIGURE 2.15 – Architectures FCN-X [55]

Se basant sur les techniques d'apprentissage supervisée, ces réseaux passent une image d'origine à travers un entonnoir de couches convolutionnelles et de mise en commun *(pooling)* pour ensuite reproduire l'image de base, mais sous son format segmenté en n classes. Les trois grandes architectures sont décrites ci-dessous et comparées dans un tableau.

b) FCN-32

L'architecture FCN-32, basée sur l'architecture VGG-16, est constituée de cinq blocs consécutifs, lesquels se terminent par une couche de mise en commun, diminuant ainsi la taille de l'image connectée par facteur de 2. Ainsi, après le premier bloc, l'image est à un facteur de diminution de 2, puis ensuite 4, 8, 16 et finalement 32 dans son dernier bloc, par rapport à l'image originale, tel que démontré sur la figure 2.15 où l'on représente les blocs par les couches de mise en commun *pool* 1 à 5.

Une fois le réseau entraîné sur l'image de base et celle-ci réduite à un facteur de 32, le réseau tente de produire une image segmentée à la sortie en exécutant un suréchantillonnage de facteur 32, afin de retrouver l'image originale, sous son format segmenté à n classes. Cette architecture permet, de façon rapide, de segmenter une image en ses grandes composantes. Toutefois, les résultats produits ne reflètent pas toujours l'image d'entrée par perte de locations spatiales des caractéristiques de l'image d'entrée.

c) FCN-16

L'architecture FCN-16 se base elle aussi sur l'architecture VGG-16 et les cinq blocs consécutifs que celle-ci impose. Cependant, comme l'entonnoir comporte un grand éventail de convolutions et de mise en communs, les informations importantes de l'image comme la position d'une personne sur l'image ou encore la fin d'un membre d'une personne et le début d'un vélo sont perdues ou détériorées.

Afin de remédier à ce problème, l'architecture FCN-16 propose de suréchantillonner la prédiction du bloc numéro 5, mais aussi d'aller chercher la prédiction du bloc numéro 4 et de maintenir cette information en plus du suréchantillonnement du bloc 5 afin de produire la prédiction. De cette façon, on obtient un agrandissement à facteur de 16, comprenant plus d'informations spatiales sur l'image et ses caractéristiques lors de la prédiction, ainsi qu'un facteur d'agrandissement deux fois plus petit que le modèle FCN-32.

d) FCN-8

Dans la dernière architecture proposée par l'article, les auteurs proposent d'opter pour la même technique qu'utilisée dans l'architecture FCN-16, produisant un agrandissement à facteur de 16, mais de remonter d'un bloc plus haut lors de la prise de décision pour conserver encore plus d'informations sur les caractéristiques de l'image originale.

L'architecture fait passer l'image à travers l'entonnoir, puis obtient une prédiction *P1*, agrandi à facteur deux, du bloc cinq. Celle-ci est additionnée à la prédiction *P2* du bloc quatre pour que les deux soient suréchantillonnées avec un facteur de deux. Finalement, le résultat obtenu précédemment est additionné à la prédiction *P3* du bloc trois, puis agrandi avec un facteur de 8, produisant ainsi l'image de début segmentée à n classes.

Cette technique est moins rapide que les deux dernières, mais permet de produire une segmentation sémantique en prenant compte les relations qu'il existe entre les caractéristiques de l'image, sans perdre celles-ci à travers les convolutions et les mises en commun.

Comparaison des architectures FCN

Sur la figure 2.16, on peut voir une démonstration, tirée de l'article de Long et al. [55], qui segmente une image d'origine à partir de chaque architecture présentée. Tel que déterminé, l'architecture FCN-32 présente une segmentation grossière de l'image, tandis que l'architecture FCN-8 démontre des traits plus pointilleux et similaires à ceux de l'image d'origine.

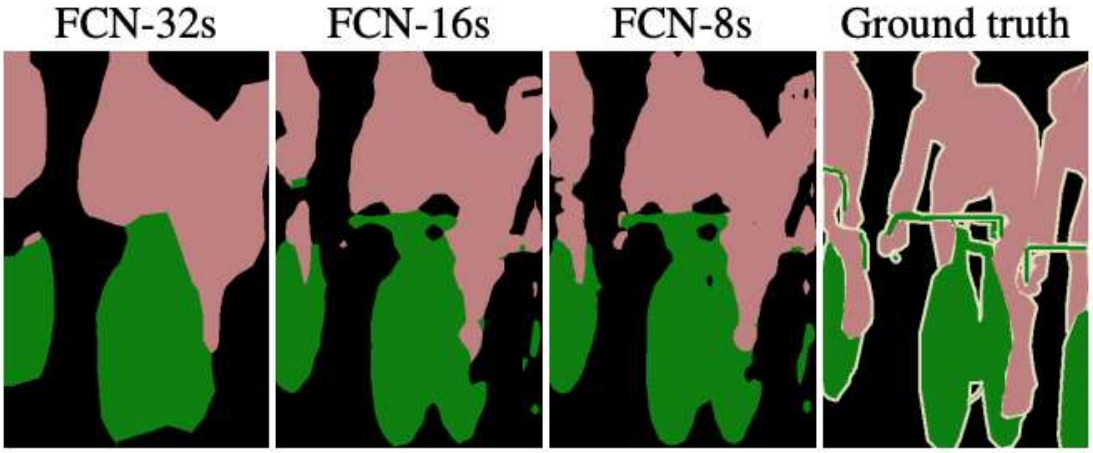


FIGURE 2.16 – Résultats d'une segmentation à partir de chaque architecture FCN-32, FCN-16, FCN-8 et de l'image originale à la droite [55]

e) U-Net

Suite aux recherches dans le domaine médical par Ronneberger et al. [69], l'architecture U-Net fit son apparition en 2015. Se basant sur les types d'architectures de réseaux encodeurs-décodeurs, le réseau U-Net prend en entrée une image de 572 x 572 pixels sous son format niveaux de gris ou RVB, d'encoder celle-ci à l'aide de couches convolutionnelles, puis de la décoder à l'aide de couches déconvolutionnelles, lui donnant sa forme de U, tel qu'illustré à la figure 2.17.

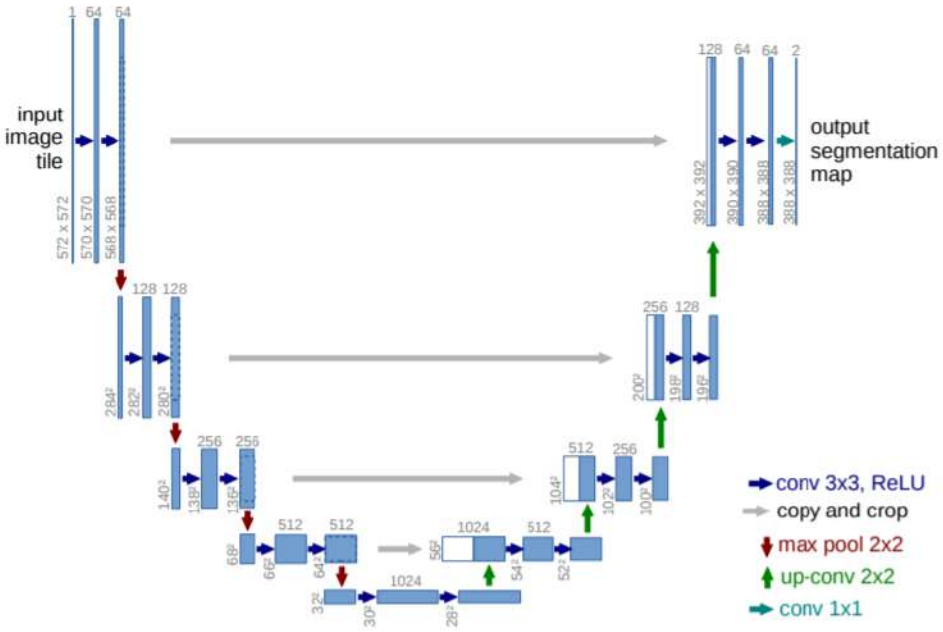


FIGURE 2.17 – Architecture du réseau U-Net [69]

Dans un premier temps, l'image de base de 572 x 572 pixels est encodée dans un format 28 x 28 pixels à 1024 filtres afin d'apprendre les caractéristiques qui la définissent. Puis, dans une deuxième phase de décodage, l'image encodée est reconstruite pour en obtenir une image de 388 x 388 pixels à n classes, tel qu'illustré à la figure 2.18, démontrant le flux simplifié des informations dans le réseau.

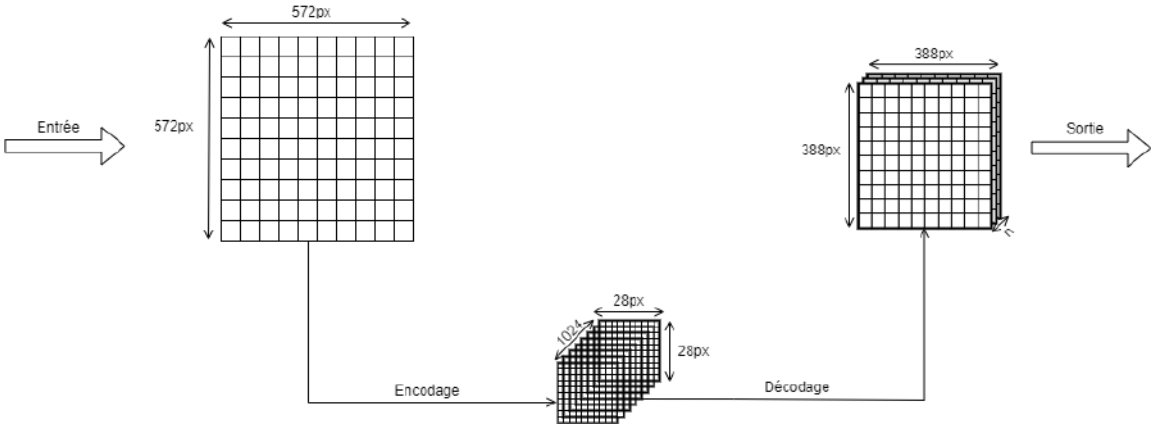


FIGURE 2.18 – Flux simplifié du U-Net

Basé sur l'architecture FCN-X de Long et al.[55], la première partie du réseau U-Net, est construite à partir de 4 blocs successifs comprenant chacun 2 couches convolutionnelles et une couche de mise en commun, définissant la phase d'encodage. Cette phase,

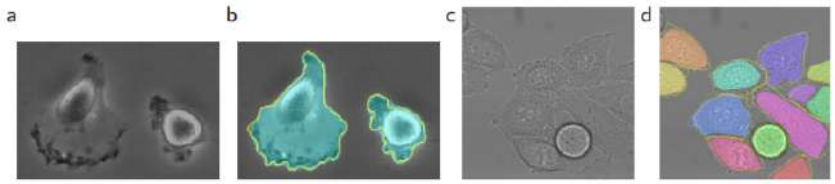


FIGURE 2.19 – **(a)** Image originale du jeu de données PhC-U373 **(b)** Segmentation produite par le réseau U-Net **(c)** Image originale du jeu de données DIC-HeLa **(d)** Segmentation produite par le réseau U-Net

tout comme dans le cas de l'architecture FCN-X, permet l'apprentissage des détails de l'image au début du réseau, puis les caractéristiques plus générales et communes dans la phase finale de son apprentissage.

Dans un deuxième temps, l'architecture propose un agrandissement de l'image encodée à partir de blocs de dé-convolution. Ayant comme but le décodage de l'image d'origine, encodée sous son format 28x28x1024 filtres, cette deuxième partie du réseau est construite à partir de 4 blocs consécutifs comprenant une opération de dé-convolution, puis deux couches de convolution.

Afin de contrer le problème de perte d'informations lors de la reconstruction de l'image encodée, tel qu'il est le cas dans le réseau FCN-32, cette architecture propose de transférer les images encodées (I_1, I_2, I_3, I_4) produites à chaque bloc de la phase d'encodage et de la transférer à chaque étape de dé-convolution de niveau égal de la phase de décodage, tel que décrit par les flèches grises sur la figure 2.17. Cette étape, par concaténation de l'agrandissement produite et de l'image encodée, permet de rapporter un maximum d'informations dans le réseau afin de produire l'image agrandie.

Finalement, l'image segmentée est reconstruite en effectuant une convolution avec filtre de grandeur 1x1 pixel à un pas de un pixel, produisant une image de 388 x 388 x n classes. Somme toute, ce réseau produit de très bons résultats, avec une moyenne d'intersection sur union (IOU) sur les jeux de données PhC-U373 et DIC-HeLa d'environ 92 % et 78% respectivement [69], tel qu'illustré à la figure 2.19.

f) DeconvNet

Défini par Noh et al. [63], l'architecture DeconvNet tente de résoudre le problème de segmentation en améliorant les résultats proposés par les architectures FCN-X.

L'un des problèmes de l'architecture FCN-X déterminé par Noh et al. lors des recherches fut la limite du champs réceptif fixe, ne permettant pas d'observer des objets qui divergent du champs réceptif, apportant des erreurs. De ce fait, les petits objets sont

souvent oubliés ou déterminés comme faisant partie de l'arrière-plan. Aussi, l'architecture présente des entrées grossières, lisses et très peu détaillées aux couches de déconvolutions, faisant en sorte de brouiller la segmentation finale, diminuant la qualité de la segmentation produite. Ainsi, ceux-ci proposent une amélioration, soit l'application des couches de déconvolution, jumelées à des couches inverses d'une mise en communs afin d'améliorer la qualité de la segmentation produite.

Tel que présenté à la figure 2.20, les mises en commun permettent, à partir d'une fenêtre de n x n pixels (2 pixels x 2 pixels sur la figure), de réduire la dimension d'une entrée, regroupant les pixels compris dans la fenêtre en choisissant le maximum *MaxPooling*, minimum *MinPooling* ou la moyenne *AveragePooling*. De l'autre côté, l'inverse d'une mise en commun permet de récupérer l'image d'entrée, en retrouvant les pixels composant la fenêtre, en prenant comme information d'entrée la valeur de la mise en commun trouvée, ainsi que l'indice de cette valeur dans la fenêtre. Les valeurs manquantes sont, la plupart du temps, remplacées par des zéros, conservant cependant les frontières des caractéristiques principales et ressortant de la mise en communs.

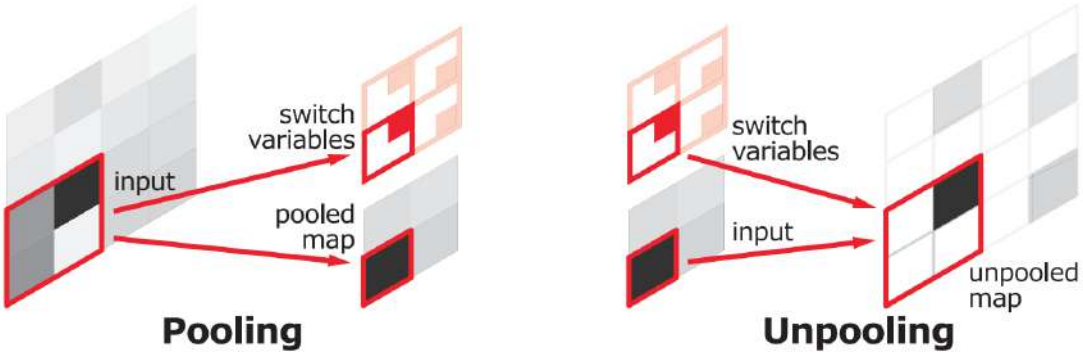


FIGURE 2.20 – Illustration démontrant une mise en commun *(Pooling)* et l'inverse d'une mise en commun *(UnPooling)* [63]

Ainsi, l'architecture Deconvnet, tout comme FCN-X, est basée sur l'architecture VGG-16 pour sa phase d'encodage, permettant d'encoder les informations importantes de l'image d'entrée avant de la décoder sous sa forme segmentée.

Au niveau de la phase de décodage, ceux-ci optent pour la mise en place d'une superposition de 5 niveaux de *UnPooling*, suivi de couches de déconvolution et de couches convolutionnelles, permettant d'augmenter les détails de première vue trouvés par les couches de *UnPooling*, tel que présenté sur la figure 2.21.

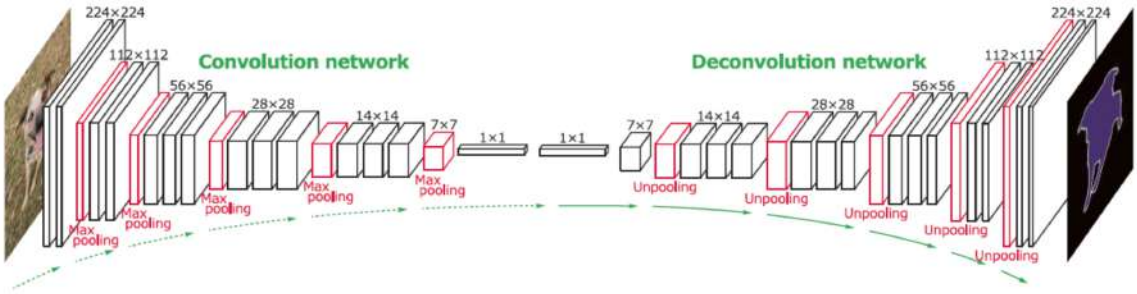


FIGURE 2.21 – Architecture de base du DeconvNet [84]

Cette technique permet notamment de conserver les frontières et détails importants qui auraient pu être perdus durant les étapes de mise en communs de la phase d'encodage. Dans son ensemble, l'architecture DeconvNet permet de surmonter des problèmes conséquents de l'architecture FCN-X, présentant des résultats d'intersection sur union de 69.6% en moyenne sur le jeu de données PASCAL VOC 2012 [27].

g) SegNet

Le réseau SegNet, présenté par Badrinarayanan et al. [3] en 2015, a principalement été conçu afin de répondre au besoin de segmentation dans le domaine routier, notamment pour les routes, les bâtiments et les véhicules.

Basé sur le réseau VGG-16 pour sa phase d'encodage, comme la plupart des autres réseaux dans le domaine de la segmentation d'images, ce réseau émet une séquence consécutive de couches de convolutions jumelé à des couches de mise en communs à sélection maximum, tel qu'illustré à la figure 2.22.

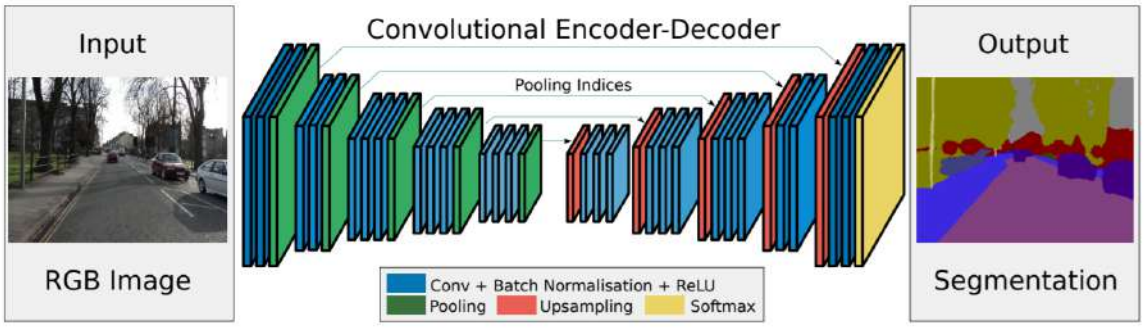


FIGURE 2.22 – Architecture du réseau SegNet [3]

Bien que l'architecture VGG-16 soit utilisée dans la plupart des réseaux de segmentation, elle présente une problématique importante au niveau de la segmentation dans le domaine de la conduite automobile, soit la notion de perte des caractéristiques spatiales de l'image. En effet, étant nécessaire de reconnaître les caractéristiques spatiales des objets perçus dans une image, la réduction de dimensions de l'image par les couches de mise en communs, jumelées aux couches de convolutions brouille les caractéristiques spatiales de l'image.

Plutôt que de transférer à la phase de décodage les filtres résultants de chaque bloc convolutionnel du réseau VGG-16, comme dans l'architecture U-Net [69], ou encore d'utiliser les techniques de *UnPooling* comme pour l'architecture DeconvNet [63] les auteurs proposent plutôt de transférer les indices des maximums qui résultent des couches de mises en communs. Cette technique permet alors de fournir aux couches de suréchantillonnage*(UpSampling)* et de déconvolution une notion de frontières des objets, augmentant ainsi l'information spatiale que le réseau a en sa possession pour produire la segmentation.

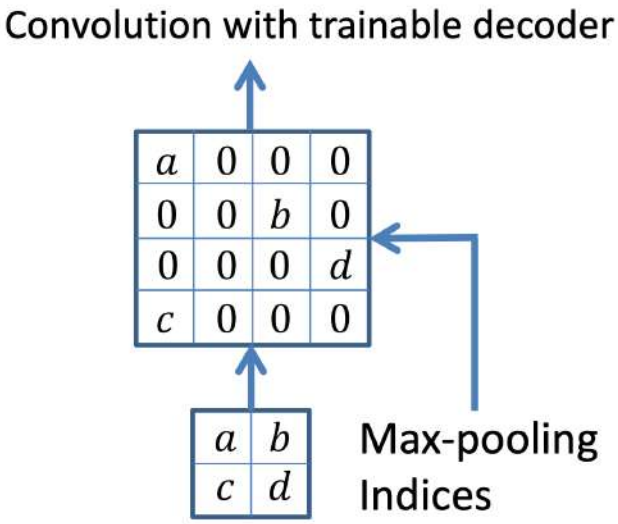


FIGURE 2.23 – Exemple d'indices de mise en communs maximum [3]

Tel que présenté sur la figure 2.23, à la suite d'une mise en commun par maximums dans la phase d'encodage, les indices des maximums sont retenus, puis transférés aux couches de décodage de niveaux égal, définissant alors les frontières des caractéristiques principales de l'image. De plus, cette technique nécessite l'utilisation d'un plus petit nombre de paramètres à entraîner, sachant qu'elle n'utilise que les indices des mises en commun, plutôt que l'information de la couche complète, accélérant ainsi le temps d'entraînement de celui-ci.

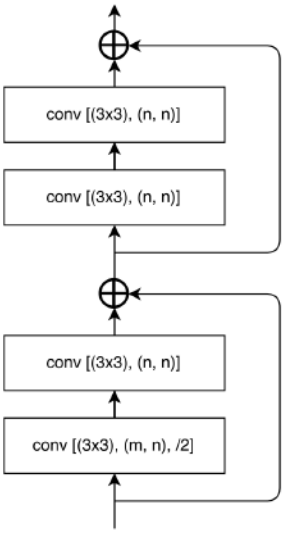


FIGURE 2.24 – Bloc de convolution du réseau LinkNet [12]

Cette architecture présente des résultats de moyenne d'intersection sur union surpassant ceux du réseau FCN et DeconvNet sur le jeu de données CamVid [7] avec une valeur test de 60.10%, comparativement à 49.83% et 59.77% respectivement, démontrant son apport dans le domaine de la conduite automatique et de la segmentation de scènes routières.

h) Linknet

Plutôt que de se baser sur un réseau d'encodage VGG-16 comme les autres architectures pour la segmentation, Chaurasia et Culurciello [12] ont opté pour l'architecture ResNet18 comme base pour leur réseau de segmentation LinkNet. Cette décision fait en sorte de ne pas simplement se servir d'un réseau de classification afin d'identifier les objets, mais de maintenir les informations spatiales de ces objets lors de l'encodage.

La phase d'encodage est constituée de quatre blocs d'encodage consécutifs. Chaque bloc de convolution, présenté à la figure 2.24, est construit à partir de deux blocs de convolution avec filtres de 3 x 3 pixels, dont le premier est suivi d'un sous-échantillonnage diminuant la grandeur de l'entrée par deux.

Chaque bloc convolutionel inclut un lien sautant (*Skip Connection*), faisant passer l'entrée vers la sortie, de façon à surmonter le problème de disparition du gradient. Cette architecture a été choisie de façon à limiter la perte des informations spatiales des images, lesquelles sont nécessaires afin de décoder l'image de façon efficace.

Au niveau de la phase de décodage, celle-ci consiste en quatre blocs de décodage,

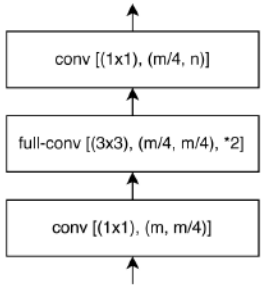


FIGURE 2.25 – Structure d'un bloc de décodage de l'architecture LinkNet [12]

augmentés par la sortie du bloc d'encodage du même niveau. Par exemple, le bloc de décodage numéro trois prend comme entrée la sortie du bloc de décodage quatre, en sommation à la sortie du bloc d'encodage trois, permettant d'ajouter des informations spatiales qui auraient été perdues avec le décodage et l'encodage des niveaux supérieurs et inférieurs.

Chaque bloc d'encodage, présenté à la figure 2.25, est construit à partir d'une série de couches comprenant une couche de convolution à fenêtre de 1 x 1 pixel à $n/4$ filtres, une couche de déconvolution avec fenêtre de 3 x 3 pixels à nombre de filtres $n/4$ et une couche de convolution à fenêtre de 1 x 1 pixel. Ces couches de convolution et de déconvolution à $n/4$, et $n/2$ filtres dans certaines alternatives, permettent de réduire le nombre de canaux produits par chaque couche, faisant en sorte de diminuer le nombre de paramètres entraînables. Cette modification fait en sorte de diminuer le nombre d'opérations à exécuter afin d'entraîner le réseau et de produire une prédiction, augmentant le nombre d'images traitées par secondes.

Finalement, le réseau se termine par une déconvolution, augmentant la grandeur de la sortie du deuxième bloc d'encodage par deux, une convolution à fenêtre de 3 x 3 pixels, puis une déconvolution finale qui esquisse alors l'image de départ segmentée en n classes. L'architecture complète est présentée sur la figure 2.26.

Cette architecture diffère des autres par son réseau d'encodage qui se base sur le réseau ResNet afin de limiter la perte d'information entre les couches d'encodage et ses couches de convolutions qui diminuent le nombre de canaux qui rendent son traitement plus rapide et le nombre de paramètres limité.

À partir de cette architecture, plusieurs variantes ont été crées, appliquées à divers domaines de la segmentation. Par exemple, D-LinkNet [95], AD-LinkNet [88] et HsgNet [89] ont tous tentés de résoudre le problème d'extraction des routes dans les images aériennes en apportant la notion de dilatation dans les couches convolutionnelles, ainsi

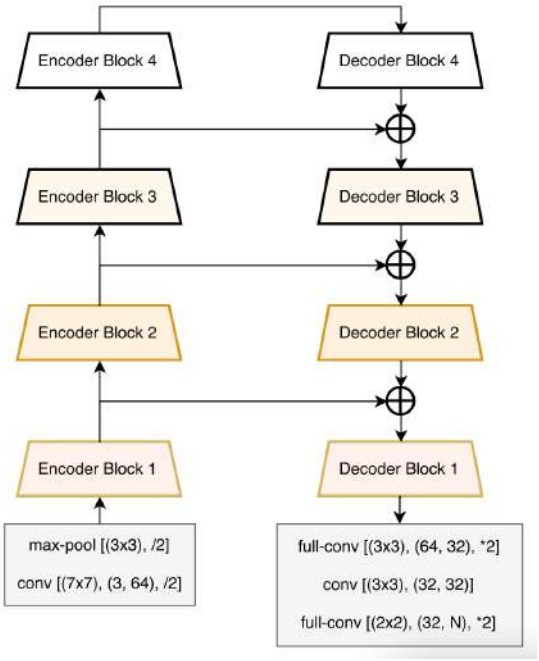


FIGURE 2.26 – Architecture LinkNet [12]

que l'ajout de blocs d'attention, amenant des résultant d'intersection sur union de plus de 64.66%, 64.79% et 71.1% respectivement sur des jeux de données d'images aériennes.

i) DeepLab

Le nombre de paramètres à entraîner et la perte d'information avec les séries de mises en communs et de convolutions sont des problèmes courants avec les réseaux FCN, U-Net et le SegNet. De plus, ces réseaux ont de la peine à trouver les objets de diverses grandeurs, ayant un champs de vision fixe à 3 x 3 pixels. L'architecture DeepLab [13] tente de résoudre le problème en apportant la notion de convolution dilatée.

Permettant d'augmenter le champs réceptif d'une convolution, sans augmenter le nombre de paramètres à entraîner, la convolution dilatée, apportée par Yu et Koltun [91] fut pionnière des architectures de segmentation modernes. Soit une convolution à fenêtre de grandeur m x n pixels, on définit une convolution dilatée à taux r une convolution où on insérera un espace de *r-1* zéros entre deux valeurs de pixels consécutifs dans la convolution, tel que présenté sur la figure 2.27, où $r = 2$. De ce fait, avec une fenêtre de taille 3 x 3 pixels et un taux de dilatation $r = 2$, on obtient un champ réceptif de 5 x 5 pixels avec le même nombre de paramètres à entraîner.

Ainsi, afin de contrer la perte d'information amenée par la juxtaposition consécutive de plusieurs couches de mise en commun et de convolutions, la méthode *Atrous*

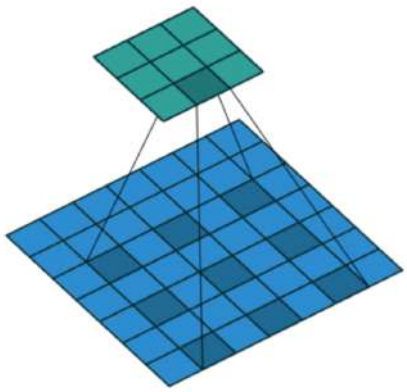


FIGURE 2.27 – Convolution Dilatée [85]

Spatial Pyramid Pooling (ASPP) fût créée. Émettant f couches de convolutions à plusieurs taux de dilatation r en parallèle, cette technique de convolution permet de créer f cartes encodant l'image d'entrée avec différents champs réceptifs. Cette technique innovatrice, permet d'augmenter le champ réceptif, sans avoir à augmenter le nombre de paramètres à augmenter, faisant en sorte de diminuer le coût de traitement relatif à ce genre d'opération. Par exemple, à la figure 2.28, on peut voir une concaténation de différentes convolutions à taux de dilatations $r = 6, 12, 18, 24$ effectuées en parallèle sur un pixel central orange.

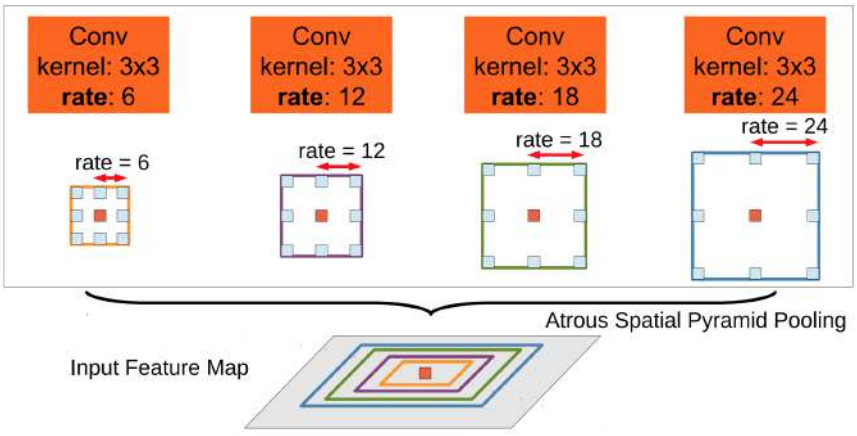


FIGURE 2.28 – *Atrous Spatial Pyramid Pooling (ASPP)* [13]

Cette idée a initié la conception des architectures DeepLab, passant d'une première version, à la version DeepLabV3+ actuelle, présentée à la figure 2.29. Dans un premier temps, l'image d'origine passe à travers un réseau d'encodage de base Xception modifié [14], basé sur le réseau Xception de base [18]. Cependant, dans certains modèles,

l'architecture d'encodage Resnet101 est utilisée pour des raisons de performance.

La sortie du réseau d'encodage, identifiant et classifiant l'image à prime abord, est séparée en deux parties. Dans une première partie, celle-ci est dirigée dans le décodeur pour passer à travers une couche de convolution de fenêtre 1 x 1 pixel afin de diminuer son nombre de canaux, puis dans un ASPP, où les couches sont concaténées, puis transférées à une couche de convolution de fenêtre 1 x 1 pixel pour réduire le nombre de canaux. La sortie réduite du ASPP est ensuite sur-échantillonnée quatre fois, pour être concaténée à la sortie réduite du réseau d'encodage. Finalement, cette sortie est passée à travers une couche de convolution à fenêtre 3 x 3 pour augmenter la précision et la justesse de la segmentation. Finalement, celle-ci est suréchantillonnée quatre fois, permettant d'obtenir l'image segmentée.

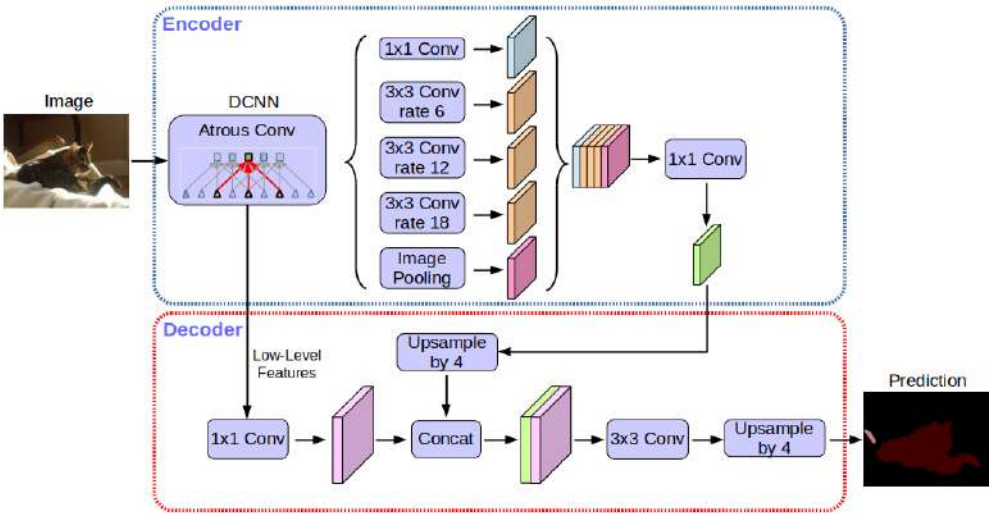


FIGURE 2.29 – Architecture du réseau DeepLabV3+ [14]

Cette architecture présente des résultats au niveau de l'intersection sur union de 89% sur le jeu de données PASCAL VOC 2012 [27], soit plus de 10% que l'architecture LinkNet et ses variantes, faisant d'elle une architecture robuste pour la segmentation.

j) FastFCN

Le réseau FastFCN, défini par Wu et al. [87], tente de résoudre le problème de complexité dans le réseau FCN Dilaté (Dilated-FCN) de Gong et al. [35]. Ce réseau, défini comme une modification du réseau FCN où les couches convolutionnelles sont présentes sous forme dilatées, obtient des résultats concluants, mais exige un temps de calcul élevé.

Basé sur l'architecture FCN-X pour sa phase d'encodage, ce réseau, présenté à la figure 2.30 prend les sorties de chaque bloc 3, 4 et 5 respectivement, pour les intégrer à une couche développée par l'équipe nommée *Joint Pyramid UpSampling (JPU)*, permettant d'extraire les caractéristiques principales d'une image et de suréchantillonner celle-ci, jusqu'à atteindre une image d'entrée segmentée, une fois passé par une couche d'encodage faisant segmentation.

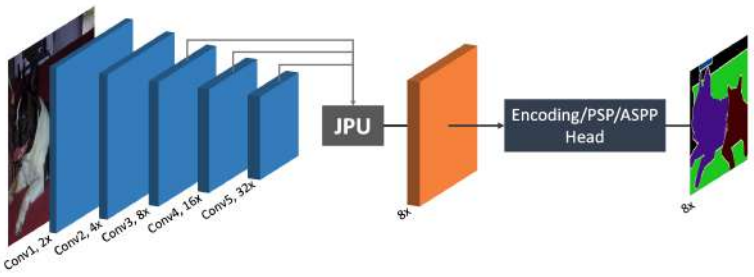


FIGURE 2.30 – Architecture du réseau Fast-RCN [87]

La couche JPU prend en entrée la sortie des blocs 3,4 et 5 du réseau FCN, soit les blocs encodés à travers des couches convolutionnelles et réduites par des couches de mise en commun. Ces sorties se voient tout d'abord imposées une couche de convolution chacune, ainsi qu'une couche de suréchantillonnage, les apportant toutes vers un nombre de canaux commun et une grandeur commune. Cette étape permet d'obtenir une concaténation de toutes les sorties des blocs 3, 4, 5 du réseau FCN, où plusieurs informations spatiales importantes sur l'image ont été recueillies.

Ces informations sont par la suite passées par une série de couches convolutionnelles avec dilatation à taux de 1, 2, 4 et 8, avec fenêtre de 3 x 3 pixels. Cette étape permet d'obtenir une combinaison de plusieurs champs réceptifs différents, sans pour autant requérir un calcul intensif par la concaténation de couches de mise en communs. Elle donne donc le maximum d'information sur les objets de toutes dimensions qui se trouvent dans les images. Finalement, cette concaténation est filtrée à travers une couche de convolution, diminuant le nombre de canaux et ainsi la complexité de la sortie de la couche JPU.

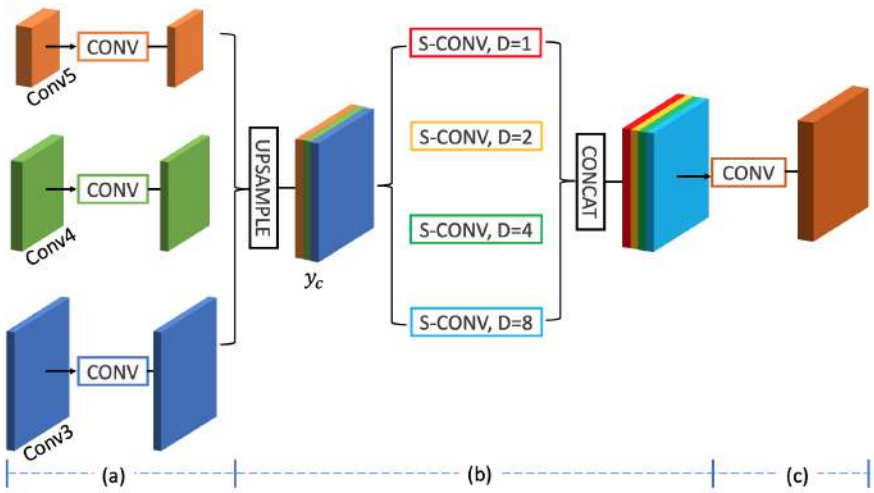


FIGURE 2.31 – Composition d'une couche Joint Pyramid UpSampling (JPU) [87]

Ce type de réseau, bien qu'il soit bâti à partir de l'architecture FCN de base, peut être remodelé en remplaçant la phase d'encodage FCN par un autre type de réseau, tel qu'une architecture ResNet50 [87] en utilisant la couche de JPU comme phase de décodage.

k) Comparaison des réseaux de segmentation

Somme toute, chaque réseau présente des caractéristiques permettant d'améliorer les résultats dans les compétitions, tel que la compétition PASCAL VOC 2012. Les résultats préalables de chaque réseau sont illustrés au tableau 2.1 ci-dessous.

Architecture	Caractéristique(s)	Performance PASCAL VOC (mIoU)
DeepLabV3+	Atrous Spatial Pyramid Pooling (ASPP)	89.0 % [14]
DeconvNet	Couches d'Unpooling	69.6 % [63]
FCN-8s	FCN-X avec agrandissement 8x	62.7 % [55]
FCN-16s	FCN-X avec agrandissement 16x	62.4 % [55]
FCN-32s	FCN-X avec agrandissement 32x	59.4 % [55]
Fast-FCN	Joint Pyramid UpSampling (JPU)	53.1 % [87]
SegNet	Transfert d'indices de mises en communs	45.1 % [23]
U-Net	Encodage-Décodage multi-niveaux	43.9 % [23]

TABLE 2.1 – Comparaison des architectures de segmentation sur le jeu de données PASCAL VOC 2012 [27]

2.6 Segmentation pour la navigation des drones

Les sections précédentes ont exploré le domaine de la segmentation sémantique de façon globale, en mettant en lumière les avancées réalisées grâce à l'utilisation de l'apprentissage automatique et des techniques de traitement de l'image. Cependant, le présent mémoire aborde un problème plus spécifique, soit la détection de lieux d'atterrissage sûrs à l'aide de la vision et prises de vues d'un drone. Ce domaine d'application se situe au croisement des techniques étudiées précédemment, mais requiert une approche spécifique et détaillée.

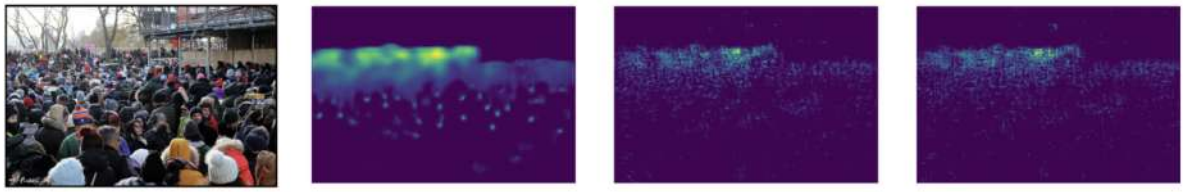
Afin de mieux appréhender les défis et les solutions pertinents dans ce contexte, il est impératif de dresser un état de l'art des techniques et des innovations spécifiquement dédiées à la détection des lieux d'atterrissage sûrs. Cette revue de littérature permettra de mettre en contexte les travaux présentés dans ce mémoire et d'identifier les avancées les plus pertinentes pour notre objectif de recherche. Cette section présente en premier temps les méthodes reliées à la mise en évidence des foules et des obstacles, soit les classes potentiellement dangereuses. Dans un second temps, elle abordera des techniques qui mettent en évidence la détection de zones d'atterrissage prédéfinies comme sécuritaires, plutôt que supposer l'environnement en entier comme sécuritaire et procéder à l'élimination des zones dangereuses. Finalement, la troisième section présente

des méthodes plus récentes, se basant sur l'apprentissage machine et plus précisément l'apprentissage profond dans le but de segmenter les zones d'atterrissage au sens large, sécuritaires ou non.

2.6.1 Détection de foules et d'obstacles

L'un des domaines les plus mentionnés dans le domaine de la recherche sur la détection des zones sécuritaires d'atterrissage (ZSA) se trouve à être la détection de foules. Cette tendance est dûe au désir de réduire le danger qu'émet un drone lorsqu'il circule dans des lieux urbains. En effet, un drone peut chuter à cause d'un bris ou une erreur de pilotage, ou même de la perte de signal, ce qui peut porter atteinte directement à la population qui s'y trouve. Le dénombrement, tant qu'à lui, peut être très utile dans des situations d'urgences dans lesquelles nous effectuons une mission de sauvetage et que nous désirons vérifier s'il y a des survivants et/ou des blessés qui ont été oubliés. De plus, cette tâche peut être très utile afin de suivre l'évolution de population des espèces en voie de disparation par exemple. Au niveau de l'atterrissage du drone, cette notion est importante, car elle permet d'évaluer les lieux plus populeux, soit ceux qui émettent un danger plus accru que les zones dans lesquelles le ratio de population est plus petit.

Les auteurs de [38] ont utilisé l'apprentissage machine afin de détecter des foules dans un environnement urbain à Venise. Plus précisément, ils attaquent le problème de dénombrement de la foule, tentant de découvrir la plus grande zone d'atterissage potentielle à partir des foules qui peuvent l'entourer. Afin d'y arriver, ils ont fait utilisation de modèles convolutifs compactes (CCCN), comme le MobileNetV2, lesquels ont été modifiés en retirant la couche de classification finale, puis ajoutant une couche de régression. De cette façon, il est possible de non-seulement montrer les pixels de l'image offrant une foule, mais aussi de dénombrer celles-ci, dotant le drone d'une meilleure prise de décisions, tel que démontré sur la figure 2.32.



(a) GT Count: 350 (b) Inferred: 335.65 (c) Inferred: 317.23 (d) Inferred: 354.86

FIGURE 2.32 – Capture, tirée de [38], montrant la répartition prédite de la foule, dénombrant un total de 350 participants (a), à partir des modèles BL MobileNetV2 (b), BL CCNN (c) et Pruned CCNN (d)

En plus d'opter pour des modèles comptant très peu de paramètres, dans le but d'augmenter la vitesse de prédiction, les auteurs mentionnent l'utilisation d'une fonction de perte "Bayes Loss", plutôt que d'opter pour une fonction de perte à base de distance euclidienne. Cette fonction de perte offre plusieurs points positifs, telle qu'une meilleure résistance aux données aberrantes, lesquelles peuvent être présentent dans les foules compactes. Elle se base aussi sur une densité (distribution) afin de calculer l'erreur, plutôt que de se baser sur un simple point, soit la tête, laquelle n'est pas toujours représentative du point central d'une foule.

Bien que la plupart des articles du domaine font état d'une utilisation d'une simple caméra attachée à l'avant du drone afin de produire les prédictions, les auteurs dans [39] montrent qu'il peut être difficile d'estimer les mouvements d'une foule à partir d'une simple caméra. Ils ont proposé plutôt d'en utiliser deux, impliquant ainsi trois différents plans dans la prise de décisions. Le but étant, qu'à partir de deux caméras, soit l'une placée à l'avant du drone, puis une autre directement sous celui-ci, en plus d'une combinaison de plusieurs capteurs, il soit possible de calculer un plan de référence global offrant un maximum d'informations sur la position réelle des foules et leur mouvement sur ce même plan.

Dans un premier temps, ils utilisent certains modèles présentés dans [38] afin de détecter des foules. Puis, ils appliquent une dilatation sur le masque créé par le modèle, dans le but d'extrapoler les zones à ne pas compter comme sécuritaires, vu le danger d'atterrir près de la foule. Par la suite, ils utilisent la projection des points trouvés dans le plan créé par les deux caméras, afin d'obtenir les références sur le repère global F_W, tel que montré à la figure 2.33.

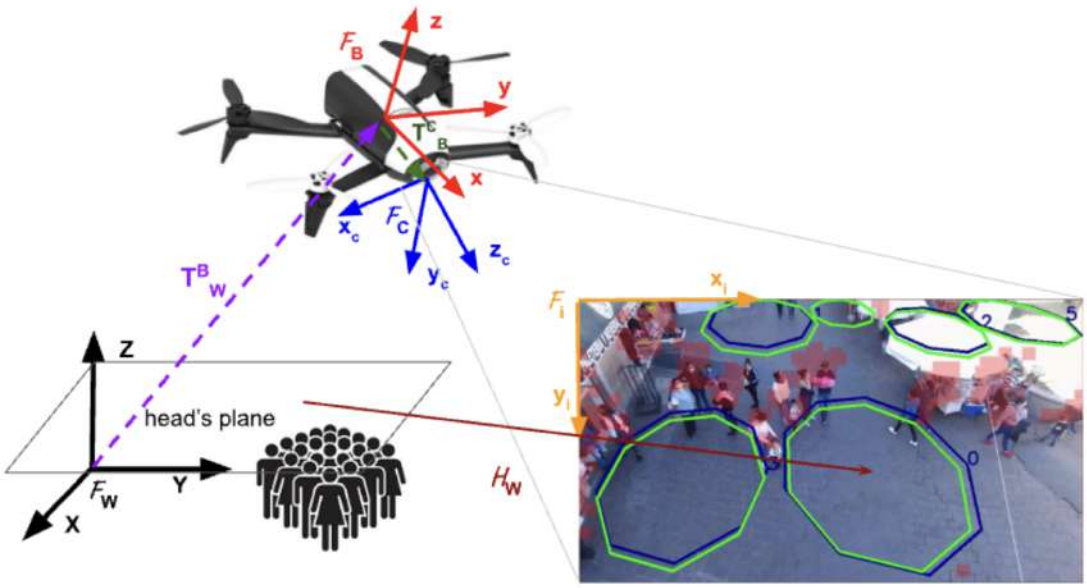


FIGURE 2.33 – Capture, tirée de [39], montrant les trois plans définis dans l'article, soit le plan F_B, obtenu à partir d'une caméra montée à l'avant de l'appareil, le plan F_C, obtenu à partir d'une caméra montée sous l'appareil, ainsi que le plan F_W, calculé à partir des deux plans précédents.

De cette façon, il est possible de bâtir une carte sur laquelle se trouve une série de points, dénotant la position actuelle de la foule ou des masses de population. En plus des positions des populations, cette carte permet d'établir la distance du drone avec la foule, augmentant la précision des zones d'atterrissage.

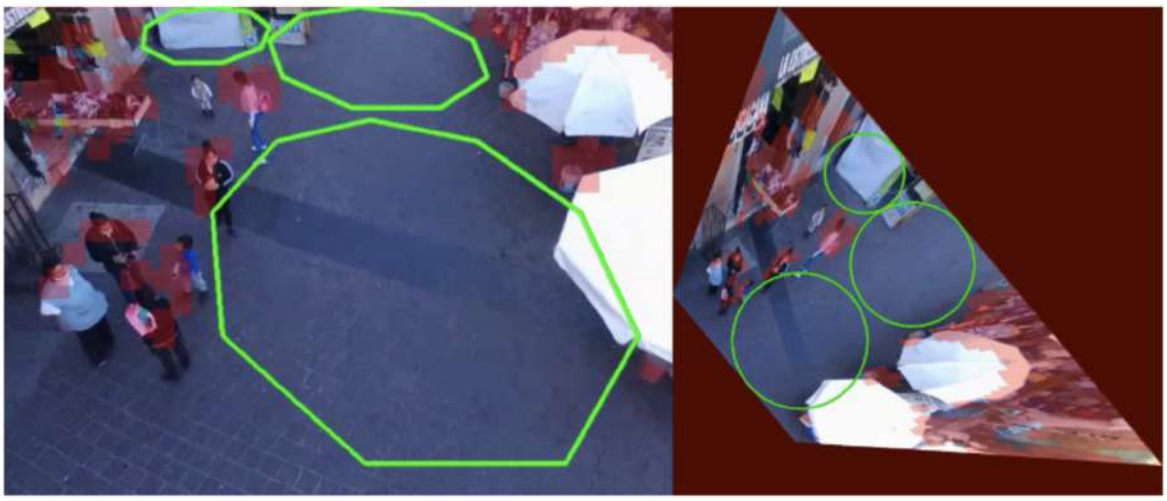


FIGURE 2.34 – Capture, tirée de [39], montrant la projection d'une foule détectée à l'aide d'un modèle convolutif (gauche) sur le plan global de référence F_W (droite).

Finalement, à travers la suite successive d'images, l'article propose le suivi des mesures des capteurs et des zones sur le plan afin d'estimer la position actuelle, ainsi que les positions futures des foules. Cela permet de déduire les zones d'atterrissages qui pourraient se voir obstruées par le mouvement des foules, des zones proposées à l'utilisateur.

2.6.2 Détection de zones d'atterrissage

La sous-section précédente a présenté les méthodes de détection des zones d'atterrissages dans la perspective d'évitement des foules. Il est cependant possible d'aborder le problème de manière plus générale, soit de restreindre le drone à atterrir dans toute zone jugée très sécuritaire.

Dans [41] et [2], les auteurs proposent une technique de détection de zones sécuritaires d'atterrissage, en ayant des motifs au sol prévus à cet effet. La technique repose sur un simple masque, basé sur les taux de coloration des pixels afin de reconnaître la zone sécuritaire. Puis, à partir de cette mesure, le drone calcule la différence entre sa position actuelle et le centre de la zone d'atterrissage, tentant de s'aligner avec cette zone. Cette technique, à première vue, semble idéale. Cependant, plus l'altitude est élevée, plus il est difficile de trouver une si petite cible. À l'opposé de cela, lorsqu'on s'approche trop de la zone, le motif occuperait presque l'entièreté de la prise de vue, ce qui empêcherait de le détecter avec précision.

Les auteurs de [2] ont utilisé un principe similaire à celui présenté précédemment, mais dans un contexte un peu plus particulier, soit l'atterrissage sur un lieu dont la position n'est pas statique. Par exemple, une plate-forme sur un bateau qui se voit bouger avec le mouvement de l'eau et des vagues. Ce problème ne pourrait pas simplement être réglé en utilisant la méthode précédente, car centrer le drone avec la cible pourrait être une tâche assez ardue, comme le centre de la cible est en mouvement continu, suivant celui de la plate-forme. Afin d'y remédier, les auteurs ont plutôt utilisé des codes AprilTag [64], soit des codes à deux dimensions, présentant des informations sur une dimension, qui sont semblables aux codes QR. En effet, en connaissant les propriétés de la caméra qui est utilisée sur le drone (distance focale, résolution, etc.), ainsi que les propriétés du code physique qui est utilisé, il est possible de calculer l'angle qui se trouve entre le drone et la cible. Avec cet angle, ainsi que les informations de position encodées sur le AprilTag, il est possible d'utiliser la projection dans le but de connaître la véritable position du drone par rapport à la cible. Le drone peut alors se situer par rapport à la plate-forme, permettant ainsi de modifier la trajectoire afin de se centrer avec la cible.

Contrairement aux méthodes de la sous-section précédente, celles-ci demandent d'assurer la sûreté de la plate-forme d'atterrissage d'avance, dans le but d'y faire un atterrissage sans problèmes par la suite. Or, ce cas n'est pas toujours une représentation de la vie réelle, où il peut arriver de devoir faire un atterrissage de dernière minute et dans lequel la cible n'est pas dans le champ visuel du drone.

2.6.3 Segmentation de zones d'atterrissage

Les sections précédentes couvrent deux méthodes populaires dans le but de bâtir une carte de lieux d'atterrissage sûrs, soit considérer tout le terrain comme sécuritaire et segmenter seulement les obstacles et zones dangereuses telles que les foules, ainsi que détecter des zones connues comme sécuritaires. Or, avec l'arrivée de l'apprentissage machine et l'avancée des techniques de segmentation sémantique de l'image, une troisième méthode s'est vue augmenter en popularité dans le domaine de la détection des zones d'atterrissage sécuritaires. Celle-ci se base sur la classification des zones dangereuses et non-dangereuses par l'utilisation de la segmentation sémantique. Cette méthode permet un étiquetage dense de la scène, offrant ainsi au pilote (ou au système de pilotage automatique) l'information des classes sécuritaire et non sécuritaire de la scène

En 2014, Patterson & al. [66] ont utilisé des méthodes de traitement de l'images dans le but de segmenter une prise de vues d'un drone à hautes altitude, puis assignant un score à chaque zone, indiquant la sûreté de du lieu. La méthode, illustrée sur la figure 2.35, suit les étapes suivantes afin de détecter les zones d'atterrissage sûres :

1. **Détection des sommets** : La première étape transforme l'image vers une représentation par niveaux de gris, puis utilise l'algorithme de détection de contours Canny afin de détecter les sommets présents dans l'image. Les sommets, représentant une différence importante dans les niveaux de gris de l'image, permettent d'identifier de principaux obstacles potentiels, tel que les arbustes, les arbres, les poteaux électriques et autres objets dangereux. Par exemple, il est facile de différencier un bâtiment du gazon, comme le bâtiment aurait des teintes plus claires par effet du soleil sur le toit, contrairement au gazon.
2. **Dilatation des zones** : Comme dans les papiers précédents, il est important d'allouer une certaine marge de sécurité sur les zones détectées permettant, du même coup, de retirer les zones trop petites pour les dimensions du drone. Le résultat est présenté dans la partie *(b)* de la figure 2.35.
3. **Clarification des zones** : Cette troisième étape demande l'ajout des détails topographiques (en anglais *Ordnance Survey* ou OS), comme la séparation des

terrains, les positions des principaux services municipaux et autres détails importants, à la segmentation effectuée à l'aide des sommets afin de raffiner celle-ci. Cet ajout de détails permet d'augmenter, ou diviser les terrains trouvés précédemment, dans le but de pouvoir les peaufiner à l'aide d'une classification, selon leur similarité, dans l'étape suivante (Voir la partie *(c)* de la figure 2.35).

4. **Classification des zones** : La quatrième étape consiste à classer les zones, divisées par les sommets, par utilisation d'un classificateur par maximum de vraisemblance (en anglais *Maximum Likelihood Classifier* ou MLC), appliqué sur chaque pixel. Cet algorithme calcule la probabilité d'un pixel d'appartenir à une classe à partir de sa ressemblance aux caractéristiques de ceux-ci en faisant utilisation de la matrice de covariance de chaque classe. Un peu comme le fait une classification par inférence Bayesienne, le classificateur assigne une probabilité à priori à chaque classe, puis successivement calcule la probabilité à posteriori d'un pixel d'appartenir à une classe à partir de ses caractéristiques de couleur et de texture. Cet algorithme est relativement fiable dans ce contexte, car en haute altitude, les classes dominantes comme le gazon, l'eau et la terre présentent des caractéristiques et textures précises. Une fois combinées aux zones délimitées par les sommets, ces prédictions représentent les zones caractéristiques de l'image, soit des zones potentiellement dangereuses ou sécuritaires. Le résultat est présenté dans la partie *(d)* de la figure 2.35.

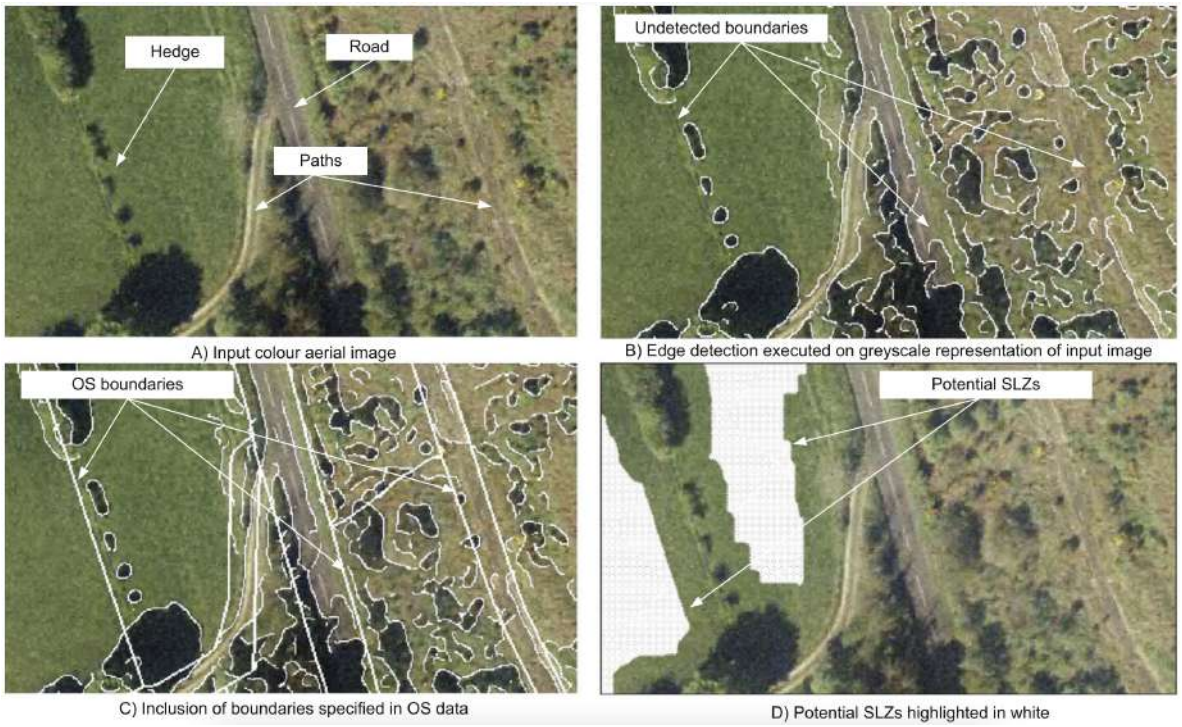


FIGURE 2.35 – Capture, tirée de [66], présentant une prise de vue (a), la détection des sommets (b), puis son raffinement par application de la classification (c), ainsi que la carte segmentée (d)

Kinahan & Smeaton [52] ont opté pour une méthode un peu plus aux goûts du jour, soit l'utilisation de la segmentation sémantique à l'aide de l'apprentissage profond afin de classer une série de pixels dans deux classes distinctes, soit sécuritaire ou non sécuritaire. À travers leur travail, les auteurs ont démontré les avantages d'utiliser des images, plutôt qu'un flux vidéo, réduisant l'énergie nécessaire aux fins de traitement, ainsi que l'augmentation de la rapidité des calculs. Se basant sur le modèle UNet [69] présenté dans la section précédente, ils ont d'abord apposé une étiquette (sécuritaire vs non-sécuritaire) sur chaque classe proposée dans le jeu de données (gazon, pavé, etc.), puis entraîné le modèle, à la fois sur des simples captures, ainsi que sur des séquences vidéos lors d'atterrissages où l'altitude varie considérablement. Leurs conclusions mettent l'accent sur la stabilité des prédictions dans les séquences vidéos lorsque le drone était à haute altitude, mais une forte variance des prédictions lorsque le drone s'approchait à très basses altitudes. Cependant, ceux-ci précisent que l'entraînement et l'évaluation se sont fait à la fois sur des données réelles, ainsi que des données simulées à partir d'un jeu de vidéo, indiquant clairement les limites de leurs résultats.

À partir des méthodes précédentes présentées à la conférence Computer Robot Vision (CRV) de l'année 2022, nous avons montré la possibilité d'utiliser un modèle lé-

ger comme le MobileNetV2, originalement conçu pour effectuer une segmentation par classification des pixels, afin d'offrir beaucoup plus d'informations au pilote lors d'un atterrissage d'urgence à l'aide d'une sortie admettant une régression, plutôt qu'une classification [1]. Cette amélioration a été proposée suite à l'hypothèse que la classification binaire (sécuritaire vs non-sécuritaire) peut porter danger au drone et l'environnement qui l'entoure dans le cas où une zone est classée comme sécuritaire avec 50.01% de confiance de la part du modèle, frôlant la classification non-sécuritaire.

Dans cette technique, le jeu de données MidAir [32] a été apprêté de façon à attribuer une classe sécuritaire (1.0), semi sécuritaire (0.5) et dangereux (0.0) à chaque pixel en fonction de leur classe sémantique respectivement. Puis, comme présenté dans [66, 39], une marge de sécurité a été dressée autour des obstacles notés comme dangereux et semi-dangereux. Cependant, plutôt que d'utiliser la dilatation, admettant une bande de sécurité supplémentaire en cas de classification, une technique de lissage grâce à un filtre *Gaussian Blur* a été utilisée, allouant au drone la possibilité d'atterrir autour des zones, mais réduisant la sûreté de l'atterrissage s'il s'en approche trop. Une fois les étiquettes assignées, les données ont été normalisées, allouant des valeurs dans l'intervalle [0.0, 1.0]. La figure 2.36 montre les étapes du traitement des images d'entrée.

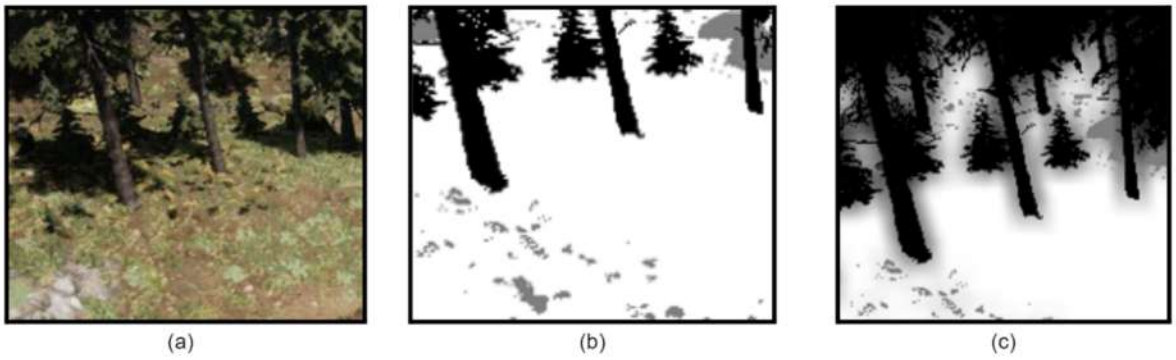


FIGURE 2.36 – Capture, tirée de [1], présentant une prise de vue du drone *(a)*, la vérité terrain admettant des classes de pixels dangereux (noir), semi-sécuritaire (gris), sécuritaire (blanc) en *(b)*, puis cette même vérité terrain avec la marge de sécurité ajoutée à l'aide d'une technique de flouage Gaussien en *(c)*

Par la suite, un modèle basée sur l'architecture UNet et un backbone MobileNetV2 a été proposé en remplaçant la convolution finale avec fonction d'activation *softmax* par une fonction *sigmoïde* permettant d'allouer une sortie décimale dans l'intervalle [0.0,1.0]. Afin d'en obtenir sa représentation sur l'échelle des niveaux de gris, il ne suffisait que de multiplier par 255, soit l'intervalle des valeurs possibles entre 0 et 255. Des exemples de résultats sont présentés sur la figure 2.37

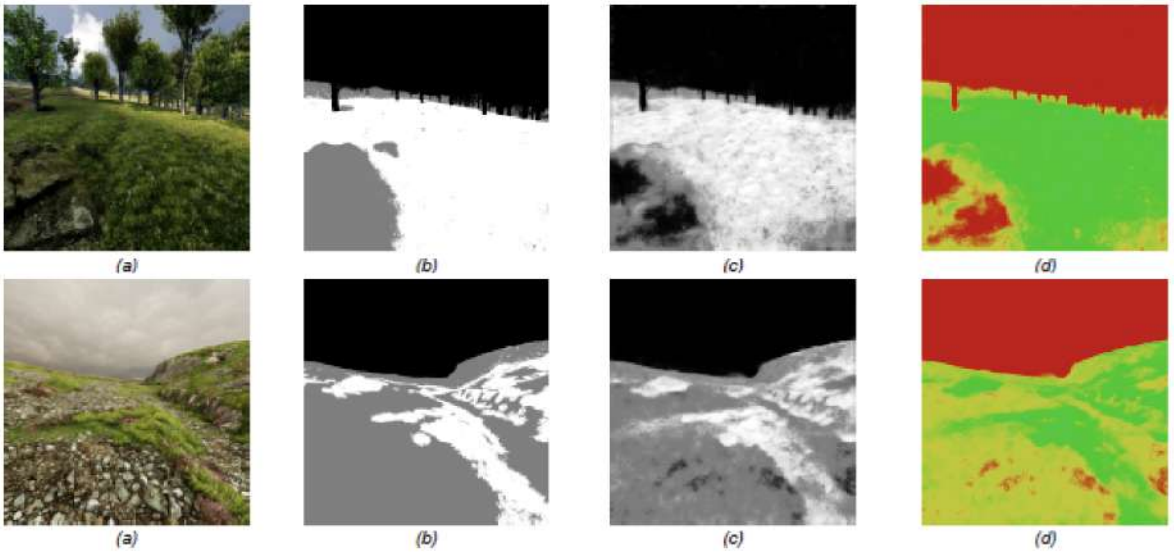


FIGURE 2.37 – Capture, tirée de [1], présentant une prise de vue du drone *(a)*, la vérité terrain de la régression *(b)*, la prédiction du modèle représentée en niveaux de gris allant de dangereux (noir) à sécuritaire (blanc) *(c)*, ainsi que cette même prédiction sur une échelle HSV colorée allant de dangereux (rouge) à sécuritaire (vert) en *(d)*

2.7 Sommaire

Somme toute, ce chapitre a présenté l'historique des méthodes et techniques utilisées dans le domaine de la vision par ordinateur et du traitement de l'image, portant une importance plus particulière aux innovations dans le domaine de la détection des lieux d'atterrissage sûrs vers la fin du chapitre. Bien que le progrès soutienne l'utilisation courante des drones dans nos vies et nos villes, la sûreté de l'environnement demeure un enjeu important comme mentionné précédemment, car la distinction entre une zone sécuritaire et une zone dangereuse peut quelques fois être ambiguë pour le drone, alors qu'elle ne l'est pas pour l'humain. C'est sur cette affirmation particulière que les sections suivantes du mémoire reposent, dans le but de proposer d'y apporter des solutions novatrices.

Chapitre 3

Méthodologie

L'objectif principal de ce mémoire est d'utiliser la segmentation afin d'extraire les zones principales définissant une capture de vue d'un drone, dans le but de bâtir une carte de zones d'atterrissage sécuritaires. Ceci permettra d'assister un pilote lors d'un vol pour assurer une navigation sécuritaire, et pour prendre des décision optimales dans les cas urgent. Autant pour la livraison de colis, que pour la vérification d'infrastructures, ou l'exécution d'atterrissages d'urgence, ces cartes sont nécessaires afin d'augmenter la sécurité d'un drone et de l'environnement dans lequel il se situe, incluant les organismes vivants.

D'une vue globale, ce problème peut être généralisé comme étant un problème de vision par ordinateur, nécessitant les approches modernes d'analyse d'images. Dans ce travail, j'opte pour l'utilisation de diverses architectures de réseaux de neurones afin de concevoir ce système d'assistance au pilotage. Ainsi, ce chapitre sera divisé en plusieurs sections, soit la collection des données, la liste des caractéristiques et architectures importantes à inclure dans le système final, ainsi que les lignes directrices à suivre afin d'évaluer la solution finale.

3.1 Acquisition de données

La première étape de cette recherche fut la collection et création des jeux de données, lesquels sont utilisés par la suite pour l'entraînement des modèles. Cette étape peut être divisée en sous-étapes :

1. **Extraction et analyse des jeux de données** : Dans cette étape, on recherche les jeux de données potentiels et sélectionnent ceux qui représentent bien le problème.

2. **Transformation et prétraitement des jeux de données** : Dans cette étape, on prépare et transforme les jeux de données avec des opérations de redimensionnement, normalisation, etc.
3. **Préparation des étiquettes de sûreté** : Comme les jeux de données aux fins de segmentation présentent des étiquettes à catégories, plutôt que des étiquettes définissant la sûreté d'atterrissage d'une zone, il est nécessaire de préparer les jeux de données en reliant les étiquettes de catégories avec une étiquette de sûreté, de façon à pouvoir entraîner les modèles par la suite.

3.1.1 Extraction et analyse des jeux de données

Comme première étape de cette méthode, j'ai évalué une série de jeux de données, afin de déterminer ceux qui corrèlent avec le problème. J'ai évalué chacun d'entre eux selon les critères suivants :

1. **Prise de vue** : Comme l'atterrissage nécessite la vue du dessous du drone afin de déterminer s'il y a des obstacles ou une inclinaison importante, un jeu de données devrait offrir une prise de vue orientée nadir, i.e pointée vers la zone d'atterrissage potentielle.
2. **Taille** : La segmentation sémantique étant un domaine pointilleux, il est attendu que le jeu de données présente une quantité d'images d'entrées substantiel.
3. **Environnement** : Les drones pouvant faire face à divers environnements, ce jeu de données devrait contenir des images provenant d'une multitude d'environnements (ex. : zones urbaines, forêts, etc.).
4. **Nombre de classes** : Directement relié avec le critère précédent, ce dernier demande que le jeu de données soit diversifié en termes de classes pour lesquelles chaque pixel peut appartenir. Comme les zones sécuritaires sont plus difficiles à déterminer que les zones non-sécuritaires, le maximum d'information que l'on peut fournir au UAV dans sa prise de décisions peut améliorer les cartes produites par le modèle.

Ainsi, en relation avec les critères précédents, une évaluation de divers jeux de données, disponibles et acceptant l'utilisation pour la recherche, a été produite. La table 3.1, présentée ci-dessous, répertorie les jeux de données qui ont été considérés dans le cadre de ce mémoire. De plus, un échantillon de chaque jeu de données est affiché à la figure 3.1.

Jeu de données	Largeur (px)	Hauteur (px)	Environnement	Type de vue	Nombre de classes	Simulation	Nombre d'échantillons
UAVid [57]	4096	2160	urbain	oblique	8	non	270
Virtual Aerial Image Dataset (VALID) [15]	1024	1024	urbain, nature, rural	nadir	30	oui	820
Institute of Computer Graphics (ICG) [47]	6000	4000	urbain	nadir	24	non	400

TABLE 3.1 – Sommaire des jeux de données évalués

FIGURE 3.1 – Échantillons tirés des jeux de données *(a)* VALID, *(b)* ICG, *(c)* UAVID

Bien que le faible nombre de jeu de données disponibles et répondant aux critères présente un défi de ce mémoire, ceux énumérés ci-dessus suffisent aux fins de représentation du problème.

1. **UAVid** : Ce jeu de données ne présente que 270 images d'un seul type d'environnement urbain, avec un faible nombre de classes, soit 8. Cependant, il n'est pas généré à partir de données synthétiques, pouvant démontrer la véracité de la solution lors de l'évaluation en environnement réel. De plus, son grand format d'images, soit 4096 x 2160 px, permet de conserver un maximum de détails de la prise de vue, pouvant améliorer l'information fournie aux réseaux testés.
2. **VALID** : Le jeu de données montre des environnements variés, avec prise de vue nadir, contenant des images carrées de 1024 x 1024 px. En revanche, il est basé sur des données synthétiques présentant un faible nombre d'échantillons, considérant le fait qu'il contient plus de 30 classes pouvant servir à faire la segmentation.
3. **ICG** : Le jeu de données ICG est composé d'images prises à la verticale, acquises par un vrai drone, dans un environnement urbain. Il représente donc une bonne opportunité de tester les solutions dans un cas de la vie réelle, plutôt que sur des données simulées. Cependant, il affiche un faible nombre d'échantillons (400) pour un jeu de données qui fait la segmentation à l'aide de 24 classes. Son grand format d'image (6000 x 4000 px) permet par contre d'obtenir une qualité d'images supérieure à certains autres jeux de données présentés.

D'un autre côté, le jeu de données *(dataset)* ICG dénombre peu d'échantillons pour le nombre d'étiquettes qu'il présente. Afin de régler ce problème, il aurait été possible d'effectuer un sous-échantillonnage des images de grandes tailles, afin de produire des images de plus petites tailles, augmentant ainsi le nombre d'échantillons disponibles dans le jeu de données. Cependant, afin d'obtenir des échantillons proportionnellement égaux, il aurait fallu découper chaque image selon une taille prédéfinie, apportant quelques fois une distorsion de l'information que l'on fournit au réseau. Par exemple, sur la figure 3.2, un sous-échantillon ***(b)***, de taille 1500 x 1000 pixels, est obtenu de l'image originale ***(a)***, de taille 6000 x 4000 px. Cette technique permet d'obtenir 16 sous-échantillons, à partir d'une image du jeu de données, réalisant un nouveau jeu de données comportant 6400 images, soit 16 fois plus volumineux.

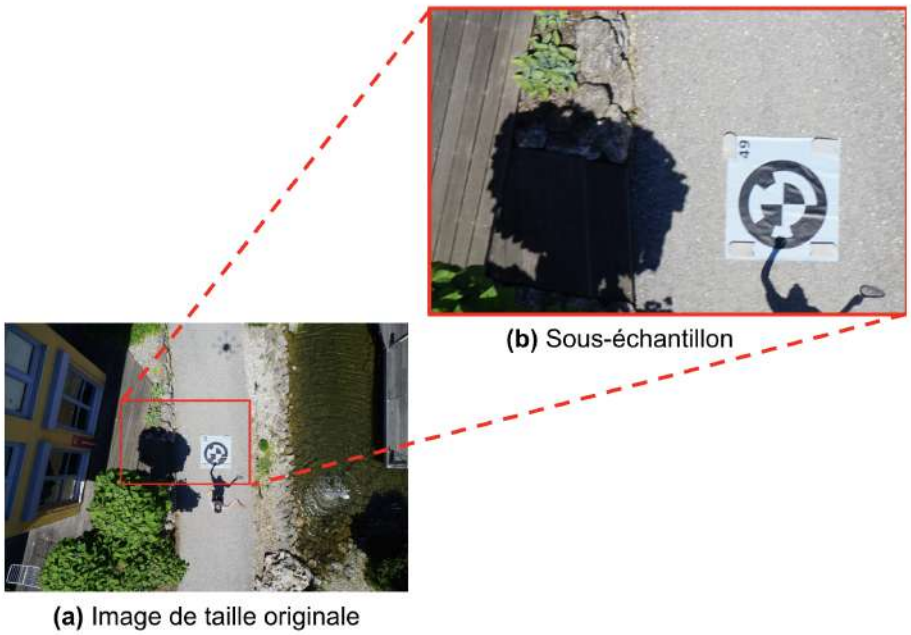


FIGURE 3.2 – Exemple de sous-échantillonnage du dataset ICG, où **(a)** présente une image originale et **(b)** un sous-échantillon de l'image originale

Cependant, tel que le démontre le sous-échantillon ***(b)*** à la figure 3.2, l'information fournie au réseau de neurones aurait été diminuée et ne représenterait pas l'information entière lui permettant de prendre une décision. Sur ce sous-échantillon, il est difficile, même à l'oeil nu, de déterminer s'il s'agit d'un humain, ou d'un ombrage dans le coin droit, pouvant amener le réseau à catégoriser la zone comme non-sécuritaire pour l'atterrissage, alors qu'elle l'est en réalité. En réalité, sur l'image originale ***(a)***, on peut apercevoir un humain, portant des couleurs, juste en dessous de cet ombrage, permettant au réseau, en cas de réussite et en prenant en compte la prise de vue entière, de le distinguer de son ombrage en comparant les couleurs des deux zones, ainsi que la

relation spatiale entre les deux.

D'un autre côté, une telle méthode aurait demandé au système de découper chaque prise de vue en 16 sous-échantillons, demander au réseau de faire la prédiction de zones sécuritaires sur chacun d'entre eux et rassembler les prédictions de chaque sous-échantillon en 1 image, laquelle serait affichée par la suite à l'utilisateur. Cette méthode augmente considérablement le nombre d'étapes à effectuer avant d'afficher une analyse au pilote, lequel a peu de temps afin d'effectuer une manoeuvre d'atterrissage dans une zone sécuritaire. Ainsi, dans un domaine où le temps est un facteur crucial, il a été préféré de conserver l'image en entier et la redimensionner, plutôt que la découper.

3.2 Prétraitement des jeux de données

Les données tirées des jeux de données ne permettent pas, à l'état brut, de segmenter une prise de vue en catégories de zones d'atterrissage sécuritaires (SLZ), à l'aide des ressources qui sont disponibles. Pour cette raison, 4 étapes de prétraitement des données ont été appliquées aux images, ainsi qu'aux masques, de chaque jeu de données, soit : Le redimensionnement, la normalisation, la préparation des étiquettes de sûreté à partir des maps de segmentation, ainsi que la transformation des cartes de segmentation sous un format ordinal.

3.2.1 Redimensionnement

La première étape des modifications s'avère être celle qui diminue directement le temps d'entraînement, en plus de réduire les ressources utilisées en diminuant la taille de l'image en entrée, ainsi que son masque.

Lors du redimensionnement, la fonction de sous-échantillonnage (*downsampling/subsampling*) doit être bien choisie. Cette fonction permet d'approximer ou de déterminer la valeur de pixels dans une image lors d'une augmentation ou réduction de sa taille [37]. Dans le cadre du redimensionnement de l'image d'entrée du réseau, cette fonction n'a pas de limite, tant qu'elle conserve le maximum de détails de l'image originale. Cependant, lorsqu'on crée un nouveau jeu de données, on associe aux masques de segmentation une couleur spécifique à chaque classe, permettant aux utilisateurs de convertir sous la forme de leur choix, permettant l'entraînement du réseau. De cette manière, lors du redimensionnement des masques, il faut utiliser une fonction de sous-échantillonnage qui conserve les valeurs des pixels présents dans les masques originaux, sans en approximer de nouveaux en utilisant la moyenne, le mode, ou d'autres fonctions statistiques par

exemple. Dans ce cas-ci, les masques ont été manipulés avec la fonction du *plus proche (Nearest)*.

Lors de l'utilisation de cette technique pour effectuer une réduction de taille d'une image de moitié, par exemple, on considère seulement que la valeur de 1 pixel sur 2 pour la hauteur, ainsi que la largeur, reliant ainsi un pixel de l'image de sortie avec le pixel le plus proche sur l'image d'entrée. Un exemple est montré à la figure 3.3, faisant passer une image de taille 4 x 4 px à une image résultante de taille 2 x 2 px.

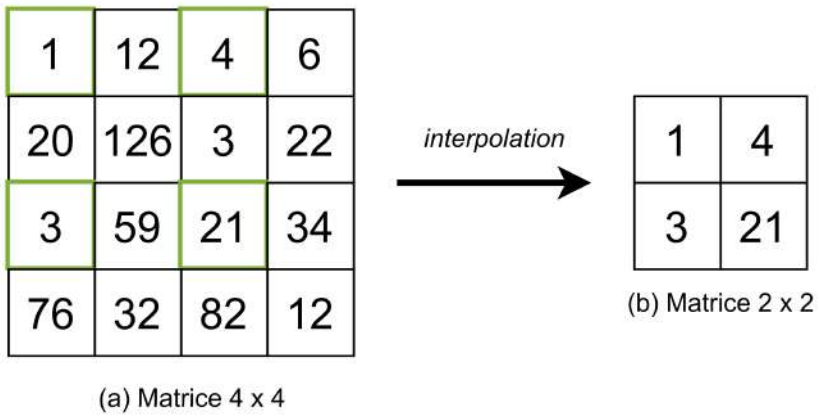


FIGURE 3.3 – Opération de sous-échantillonnage (*downsampling/subsampling*) de type *plus proche (Nearest)*

Bien que nous aurions pu utiliser les fonctions de recadrage, il était important de conserver l'entièreté d'une scène afin de pouvoir bien définir le contexte lors de la prise de décision du UAV. À titre d'exemple, supposons une prise de vue incluant un arbre. Vue de haut, lorsque nous excluons le 1/4 de cet arbre, suite à un recadrage, celui-ci pourrait être confondu par le modèle neuronal pour du gazon, allouant le droit d'atterrir dans cette zone au UAV.

3.2.2 Normalisation

Bien que tous les jeux de données mentionnés montrent, à travers l'entièreté des échantillons, des valeurs d'intensités comprises dans l'intervalle $[0, 255]$, les librairies de statistiques, d'apprentissage machine (ML) et d'apprentissage machine profond (DML) nécessitent, pour la plupart, la représentation des données sous sa forme à virgule.

Pour cette raison, durant la deuxième étape, les données sont transformées en utilisant la méthode de normalisation *min-max* [37]. En utilisant cette technique, à partir d'un étendu de valeurs de données, compris dans l'intervalle $[min, max]$, la valeur de

chaque donnée est ramenée dans l'intervalle $[0.0, 1.0]$, soit sa forme à virgule dans l'intervalle, en suivant l'équation de normalisation *f(x)* suivante :

$$f(x) = \frac{(x - min)}{(max - min)} \tag{3.1}$$

où $x \in [min, max]$ représente l'intensité d'un pixel. Dans notre cas, les pixels ont une valeur d'intensité entre 0 et 255, nécessitant l'utilisation de la valeur *min* à 0 et le *max* à 255. Une fois simplifié, on remarque qu'il ne s'agit que d'une division de la valeur du pixel par 255, comme l'autre opérateur est 0, n'apportant aucun effet sur le calcul.

3.2.3 Production des étiquettes de sûreté

Plusieurs exemples de la littérature [76, 51] débutent la résolution du problème par la classification des pixels dans des catégories thématiques, puis attribuent une étiquette de sécurité à chaque pixel par la suite, selon la classe de l'objet. Considérant plutôt la corrélation entre une catégorie d'objets et son étiquette de sécurité dans les modèles neuronaux directement, notre approche demande plutôt de prétraiter les cartes de segmentation avant l'utilisation. Cette troisième étape de prétraitement des données demande de convertir les cartes de segmentation sémantique, regroupant les pixels par classes (e.g. bâtiments, eau, etc.), en cartes montrant le niveau de sûreté d'atterrissage de chaque pixel.

Comme les jeux de données ne présentent pas de niveaux de sécurité de zones d'atterrissage, mais plutôt des classes thématiques, nous avons défini une série de règles, selon trois niveaux de sécurité (dangereux, peu sécuritaire, sécuritaire), pouvant relier les classes à un niveau de sécurité :

Définition des niveaux de sûreté

Dangereux. Ensemble des catégories thématiques qui mettent, ou pourraient mettre, en danger le UAV, son pilote, ses composantes (internes ou externes), son logiciel, ainsi que l'environnement qui l'entoure et ses habitants. Par exemple, les rivières, les arbres, etc.

Peu sécuritaire. Ensemble des catégories thématiques qui présentent un risque, moindre que la catégorie *dangereux*, qui peut être considéré comme potentiel lorsqu'il y a présence d'un cas atterrissage d'urgence. Elle ne met pas la sécurité du UAV, de son pilote, de son environnement et de ses habitants en danger, mais agit

comme dernier recours, pouvant apporter un dommage matériel au UAV, sans le détruire. Par exemple, le gravier et les terrains plats, mais rocheux.

Sécuritaire. Ensemble des catégories thématiques qui ne posent aucun(s) problème(s) au UAV, son pilote, ainsi que son environnement et ses habitants. Par exemple, la pelouse et une région pavée.

Extraction de niveaux par classe thématique

Suivant cette logique, à partir des jeux de données, la table 3.2 peut être constituée, attribuant un niveau de sûreté à chaque classe thématique présente, et ce pour chaque jeu de données.

	Niveau de sécurité		
Jeu de données	***dangereux***	***peu sécuritaire***	***sécuritaire***
UAVid	arrière-plan, voitures, humains, arbres	routes, bâtiments	végétation basse
Virtual AeriaL Image Dataset (VALID)	obstacles bas, véhicules, chaises, clôtures, poubelles, panneaux, bâteaux, avions, arbres, lignes électriques, arrêts d'autobus, lampadaires, tunnels, ports, glace, rochers, obstacles hauts, animaux, humains, eau, piscines, arrière-plan	bâtiments, ponts, terrains, toits	régions pavées
Institute of Computer Graphics (ICG)	arrière-plan, eau, rochers, piscines, végétation, toit, murs, fenêtres, portes, clôtures et poteaux, humains, chiens, voitures, vélos, espaces conflictuels, arbres, obstacles,	chemin de terre, de gravel, toits	marqueurs-ar, régions pavées, herbe

TABLE 3.2 – Distribution des classes thématiques sur 3 niveaux de sécurité pour chaque jeu de données

Bien qu'il soit simple de démontrer une carte à trois niveaux de sécurité, il y a des cas où la décision d'atterrir doit se prendre en quelques secondes, nécessitant de distinguer le niveau *peu sécuritaire* entre ce qui semble peu sécuritaire, mais tout de même sécuritaire, ainsi que ce qui est peu sécuritaire, s'approchant du dangereux. La table 3.3, se basant sur la même logique que la précédente, apporte une nouvelle dimension à cinq niveaux de sécurités aux données pouvant être utilisées pendant l'entraînement.

	Niveau de sécurité				
Jeu de données	***dangereux***	**semi-dangereux**	***ni plus ni moins sécuritaire***	**semi-sécuritaire**	***sécuritaire***
UAVid	arrière-plan, humains	voitures, arbres	bâtiments	routes	végétation basse
Virtual AeriaL Image Dataset (VALID)	clôtures, avions, lignes électriques, obstacles hauts, animaux, humains, eau, piscines, arrière-plan	véhicules, chaises, rochers, lampadaires, bateaux, signalisation, arbres, obstacles bas, arrêts d'autobus, glace, plantes autres, tunnels, ports, poubelles	bâtiments, ponts, toits	terrains	régions pavées
Institute of Computer Graphics (ICG)	arrière-plan, eau, piscines, humains, chiens, vélos, espaces conflictuels	rochers, végétation, voitures, arbres, clôture et poteau, obstacle, fenêtre, mur, portes, fenêtres, murs	toits	chemin de terre, chemin de gravel	marqueurs-ar, régions pavées, herbe

TABLE 3.3 – Distribution des classes thématiques sur 5 niveaux de sécurité pour chaque jeu de données

3.2.4 Représentation ordinale

Dans ce mémoire, comme nous utilisons des méthodes qui esquissent le sujet comme un problème attribuant une importance aux niveaux de sécurité, on considère la solution comme étant basée sur un problème de régression ordinale. Par conséquent, les cartes de segmentation produites à l'étape précédente doivent être encodées sous leur forme ordinale, soit à base de rangs.

Contrairement aux méthodes de segmentation sémantique usuelles, basées sur le principe de classification, dans lequel nous attribuons à chaque pixel une seule et unique classe la représentant, la préparation des cartes de segmentation sous la forme ordinale ne nécessite pas d'attribuer une classe à chaque pixel, mais plutôt un rang. On entend par là que si un pixel fait partie du rang n, c'est qu'il fait aussi partie des rangs $0...n-1$. À titre d'exemple, un étudiant du système d'éducation québécois doit avoir passé tous les examens de l'école primaire afin d'être admis à l'école secondaire, puis passé tous les examens de l'école secondaire afin d'être admis au cégep, et ainsi de suite. De la même façon, un pixel, ayant atteint un rang n, se doit d'avoir atteint tous les rangs inférieurs $0...n-1$ aussi.

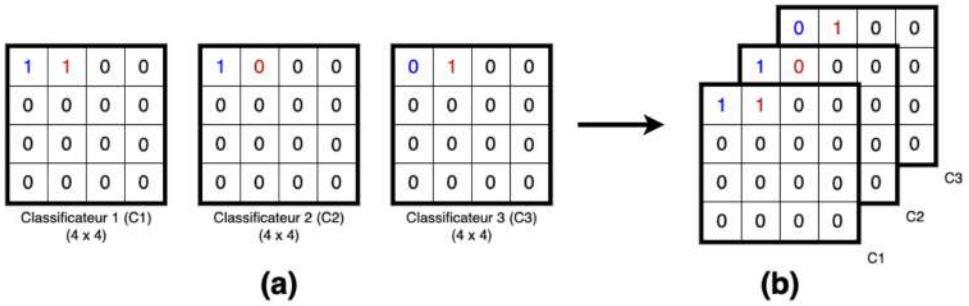


FIGURE 3.4 – Encodage d'une carte de segmentation sous sa forme ordinale, où **(a)** présente chaque classificateur composant la map résultante **(b)**. (n.b. Les pixels en bleu et rouge appartiennent aux classes 2 et 1 respectivement, comme le pixel rouge n'a pas passé le test du classificateur C2)

Afin de produire la carte permettant de résoudre un problème à n classes, on doit d'abord considérer tous les pixels comme ayant l'étiquette 0, soit n'ayant réussi à passer aucun test lui permettant de monter de rang. Puis, en considérant *n-1* tests, on indique, pour chaque pixel, on indique s'il a passé le test d'un rang r, en lui attribuant la note 1, ou non en lui attribuant la note 0, lui permettant de changer de classe, pour une classe supérieure. Un exemple de carte est présenté sur la figure 3.4. Afin d'y parvenir, on suit l'algorithme suivant.

Algorithm 2: Encodage ordinal des cartes de segmentation

```
// Créer une matrice de nbClasses - 1 tests/classificateurs
// ayant 0.0 comme valeur par défaut (Aucun test de réussi)
map ← newfloat[hauteur][largeur][nbClasses − 1];
// Pour chaque classificateur, tester tous les pixels de l'image
 for t ∈ [1..(map.depth)] do
    for Pixel p ∈ Image do
        // Changer l'indicateur du pixel p par 1.0
        // s'il réussi le test t (Classe(p) >= t)
        if classe(p) ≥ t then
            map[p] ← 1.0;
        end
    end
end
```

Prenons par exemple un pixel classifié comme *peu sécuritaire* (1), parmi une possibilité de 3 étiquettes : *dangereux* (0), *peu sécuritaire* (1) et *sécuritaire* (2).

1. On crée deux tests *n-1* (c.-à-d. une matrice avec deux canaux de classification) et on considère tous les pixels comme dangereux. (À ce moment, le pixel a une

étiquette 0 et un encodage [0, 0], soit l'échec des tests 1 et 2)

2. On fait passer le test 1 au pixel, vérifiant si sa classe *peu sécuritaire* (1) est plus grande ou égale à la classe du test 1, soit *peu sécuritaire* (1). Le cas échéant étant vrai, le pixel obtient maintenant la classe *peu sécuritaire* (1) et est encodé sous la forme [1, 0], indiquant sa réussite du test 1, mais pas du test 2.
3. On fait passer le test 2 au pixel, vérifiant si sa classe *peu sécuritaire* (1) est plus grand ou égal à la classe du test 2 *sécuritaire* (2). Comme ce n'est pas le cas, le pixel ne voit pas son encodage passer à [1,1], mais demeure à [1,0], puis conserve sa classe *peu sécuritaire* (2).

On obtient donc une carte de segmentation, formatée afin de représenter le problème de type régression ordinale, en suivant les étapes montrées ci-dessus pour chaque pixel de l'image originale.

3.3 Échantillonnage des jeux de données

Une fois les données traitées, celles-ci sont prêtes à être utilisées aux fins de traitement. Afin d'évaluer les solutions qui sont proposées durant et après l'entraînement, nous avons séparé chaque jeu de données en trois catégories : Les données d'entraînement, de test et de validation. Dépendamment du nombre d'échantillons total dans un jeu de données, le ratio dans chaque catégorie se voit modifié, permettant d'obtenir un maximum d'instances durant l'entraînement, évitant les problèmes de sur-entraînement.

		Catégorie		
Jeu de données	**Nombre d'échantillons total**	**Entraînement (%)**	**Test (%)**	**Validation (%)**
UAVid	270	72.2	22.2	5.6
Virtual AeriaL Image Dataset (VALID)	820	74.9	19.5	5.6
Institute of Computer Graphics (ICG)	400	72.0	22.0	6.0

TABLE 3.4 – Répartition du nombre d'échantillons total de chaque jeu de données à travers les catégories d'entraînement, de test et de validation

Il est à noter que nous avons tenté de maintenir des ratios qui apportent en moyenne

des valeurs frôlant un ratio entre 70% et 75% du jeu de données pour l'entraînement, 20% et 25% pour le sous-ensemble de test, ainsi qu'entre 0% et 5% pour le sous-ensemble de validation, utilisé lors de l'entraînement afin de visionner en temps réel l'allure de l'entraînement, ainsi que d'éviter toute déviation du modèle vers le sur-entraînement.

3.4 Modèles et Architectures

Tel que mentionné au chapitre 2, il existe divers réseaux de neurones, basés sur des architectures distinctes, permettant de résoudre le problème de segmentation sémantique. Dans ce travail, nous avons considéré deux techniques afin de résoudre le problème, soit la classification et la régression ordinale. Ces méthodes sont décrites dans les sous-sections suivantes.

3.4.1 Segmentation

Nous avons regroupé les réseaux évalués en trois catégories d'architectures, soit les réseaux de type *UNet*, *Linknet* et *DeepLabV3+*. Ces architectures, telles que décrites dans le chapitre 2, présentent des caractéristiques distinctes qui défini notre base dans l'évaluation de potentielles solutions.

Notre objectif est d'identifier les modèles les plus adaptés pour cette tâche complexe, en tenant compte de plusieurs critères essentiels. Parmi les modèles examinés, nous avons retenu trois architectures de réseaux profonds bien établies : InceptionV3, ResNet101 et MobilenetV2. Chacun de ces modèles présente des caractéristiques spécifiques qui justifient leur inclusion dans notre étude. La table 3.5, extraite de [19], dresse la liste des caractéristiques associées à ces modèles.

Backbone	**Nombre de paramètres (Millions)**	**Exactitude Top-1 (%)**
InceptionV3	23.9	77.9
ResNet101	44.7	76.4
MobilenetV2	3.5	71.3

TABLE 3.5 – Sommaire des modèles (*Backbones*) évalués à travers les architectures Unet, Linknet et DeeplabV3+, extrait de [19]

Nous avons choisi le modèle InceptionV3, car il est apprécié pour sa capacité à gérer le surapprentissage grâce à ses modules Inception, ce qui est un aspect important vu le jeu de données ICG, contenant peu d'échantillons. Il excelle également dans l'extraction d'informations à différentes échelles, ce qui s'avère particulièrement utile dans la détection de SLZ, sachant que les prises de vues sont effectuées à plusieurs niveaux, voyant de petites classes importantes, comme les humains, occuper très peu de pixels sur l'image entière.

Quant à lui, le modèle ResNet101 [42], fondés sur des réseaux résiduels, est renommé pour sa profondeur et sa capacité à apprendre des représentations hiérarchiques. Cette caractéristique s'avère particulièrement pertinente pour la détection des zones d'atterrissage s.curitaire, où des objets variés en taille et complexité doivent être discernés.

Enfin, le modèle MobilenetV2 se distingue par son architecture légère et son efficacité en termes de calcul, qui est une caractéristique nécessaire sachant qu'un drone se doit d'être léger, donc offrir peu de capacité de calculs.

Ces modèles ont été adapté à chacune des trois architectures, soit UNet, LinkNet et DeepLabV3+, étant utilisé dans la phase d'encodage. Dans le cadre de notre approche pour la segmentation sémantique, nous avons intégré une couche convolutionnelle à la sortie, comprenant 3 canaux dans un premier cas où l'on veut détecter 3 niveaux de sécurité et 5 canaux dans un deuxième cas, pour la classification avec 5 niveaux de sécurité. On applique par la suite une fonction d'activation Softmax à la sortie, ne permettant qu'à une neurone d'un seul canal pour chaque pixel de la carte de segmentation d'être activée.

3.4.2 SLZNet

Bien que les modèles précédents présentent de très bons résultats selon la documentation [19], ceux-ci présentent tout de même des points négatifs. Ainsi est née l'idée de concevoir une nouvelle architecture, basée sur l'architecture UNet qui performe très bien, en combinant les idées des backbones évalués. En effet, les taux de dilatations (Convolutions à trous) permettent d'augmenter le champ réceptif des convolutions. Puis, les connexions sautantes permettent de rapporter de l'information spatiale des couches précédentes dans des couches futures. De cette façon, en combinant tous ces facteurs, tout en conservant un nombre de paramètres raisonnables, nous obtenons un réseau qui contient le meilleur de tous les réseaux, soit très adapté aux conditions et différentes altitudes qui existent dans le monde des UAV.

Comme toutes les autres architectures de segmentation, le SLZNet se divise en trois

portions, soit la phase d'encodage, la phase d'entonnoir, ainsi que la phase de décodage. Chaque phase n'est qu'une superposition de plusieurs opérations de convolutions, de pooling, de normalisation et de concaténation, permettant d'extraire un maximum de caractéristiques, puis de les regrouper afin de segmenter la prise de vue en maintenant les caractéristiques spatiales initialement présentes dans l'image d'entrée.

L'architecture en soi se base sur une série de blocs convolutifs, présenté à la figure 3.5. Chaque bloc est composé de trois opérations de convolution en parallèle, caractérisés par des taux de dilatation de 1, 3 et 4. Le but de cette convolution à trous est de pouvoir extraire le maximum d'information spatiale sur l'entrée permettant, par exemple, de retenir la présence de plus petites classes comme les humains, voitures et autres dans une portion du filtre présentant majoritairement des classes plus imposantes comme des arbres, routes et autres. Le bloc présente une première sortie, soit la convolution à n filtres sans dilatation, puis une sortie *Skip Connection* utilisée dans la reconstruction de l'image lors de la phase de décodage afin d'incorporer le maximum d'information sur la couche d'entrée du même niveau de la phase d'encodage. Cette sortie n'est rien d'autre que la concaténation des trois couches convolutionnelles du bloc. Il est aussi à noter que chaque couche convolutionnelle passe à travers une couche de normalisation.

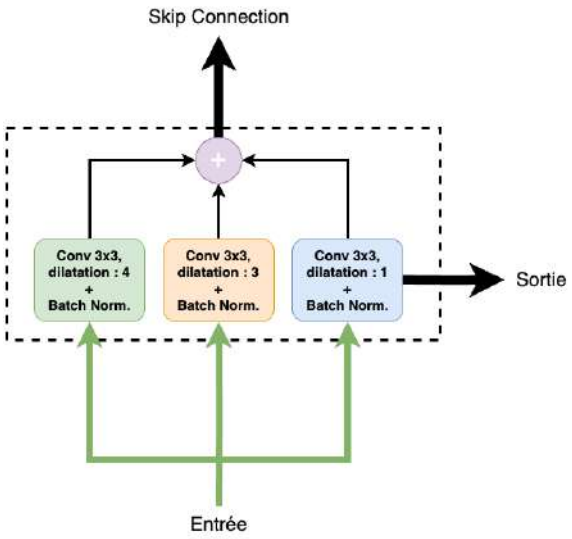


FIGURE 3.5 – Représentation d'un bloc convolutionnel du réseau SLZNet

Dans l'ensemble, tel que présenté à la figure 3.6 la phase d'encodage consiste en une série de bloc de convolutions, duquel la sortie est rétrécie à l'aide d'une opération de Max Pooling, puis passée comme entrée au prochain bloc. La série propose des connexions entre des blocs contenant 64, 128, 256 et 512 filtres, afin d'arriver à un entonnoir, soit une seule couche convolutionelle sans dilatation de 1024 filtres. Finalement, le réseau entre

dans la phase de décodage, dans lequel la sortie de la dernière opération est agrandi à l'aide d'une couche d'Upsampling, puis concaténée à la sortie à plusieurs taux de diltation (*Skip Connection*), avant d'être passée à un bloc de convolution de décodage. Chaque bloc convolutif de décodage est composé de 2 couches convolutionnelles avec taux de dilatation 1, puis une couche de normalisation.

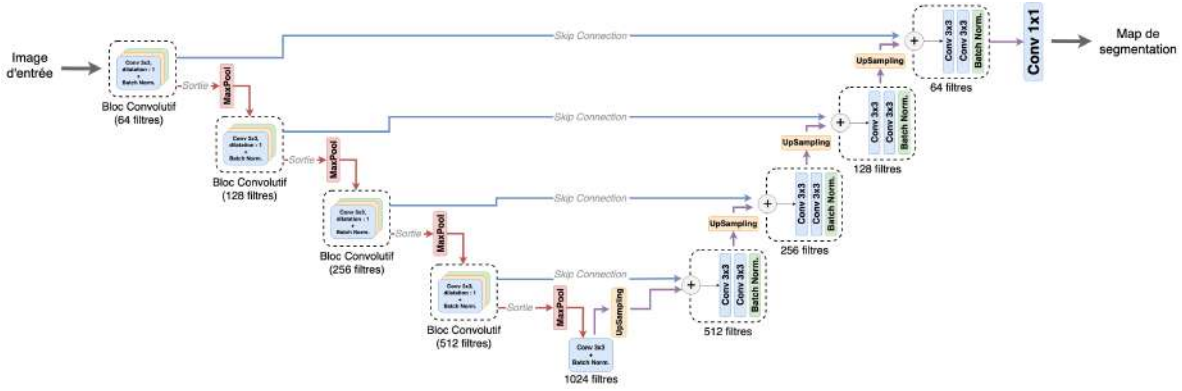


FIGURE 3.6 – Architecture du réseau SLZNet

3.4.3 Segmentation par régression ordinale

Bien que la solution basée sur la classification peut être considérée comme acceptable, elle peut émettre une erreur sévère rapide lorsqu'il y a mauvaise classification d'un pixel dangereux à sécuritaire, sautant ainsi tous les niveaux entre les classes. C'est pourquoi, nous avons fait utilisation de la régression ordinale, utilisant *n-1* classificateurs, afin de classer les zones à travers n classes. Cette modification permet de s'assurer que chaque zone a, au minimum, surpassée la classification *dangereuse*, avant d'être classée comme *semi-sécuritaire*, et ainsi de suite.

Cette nouvelle approche de la problématique n'a pas demandé de changer nos choix quant aux modèles et architectures à utiliser. En vain, il ne fallait que modifier la dernière couche convolutionnelle offrant n classificateurs pour un problème n classes avec une fonction d'activation softmax, afin qu'elle comporte maintenant *n-1* classificateurs, utilisant chacun d'entre eux une fonction d'activation sigmoid. Enfin, en sortie du réseau, lorsqu'on veut trouver la classe d'un pixel, il ne faut déterminer qu'un seuil pour chaque couche convolutionelle, déterminant si le classificateur est activé, ou non, faisant passer ce pixel d'un niveau inférieur à une classe supérieure.

3.5 Fonctions de perte

Bien que les architectures mentionnées dans la section précédente soient essentielles à la solution, l'entraînement efficace d'un modèle de segmentation sémantique repose également sur le choix judicieux d'une fonction de perte adéquate.

Combinés à ces modèles, dans ce travail, nous avons évalué des fonctions de pertes (*loss functions*), apportant des variations sur l'entraînement d'un modèle. Dans une première catégorie, les fonctions de perte symétriques telles que l'erreur absolue moyenne (*Mean Absolute Error*) et l'erreur moyenne au carré (*Mean Squared Error*) sont utilisées pour entraîner le modèle performant la segmentation par régression. Dans une deuxième catégorie, les fonctions de perte asymétriques par entropie croisée catégorielle et binaire (*categorical/binary cross-entropy*) sont utilisées dans ce travail pour faire l'entraînement des modèles complétant les tâches de classification et régression ordinale.

Finalement, une nouvelle fonction de perte asymétrique, appliquée au domaine du *SLZ*, dérivée de la fonction de perte *binary cross-entropy*, a été défini et évaluée.

Les sous-sections suivantes décrivent brièvement l'utilité et l'efficacité de chaque catégorie.

3.5.1 Fonctions de perte symétriques

La première catégorie de fonctions de perte utilisées dans ce travail est décrite comme étant symétriques. En d'autres mots, certaines fonctions de perte, comme l'erreur absolue moyenne (*MAE)* et l'erreur moyenne au carré (*MSE*), émettent une erreur proportionelle, peu importe la classe qui est prédicte, comparativement à celle qui était attendu [20].

Par exemple, prenons l'équation 3.2, montrant la fonction de perte *MAE*, dans laquelle y_t représente l'étiquette attendue et y_p celle qui a été prédite par le modèle.

$$E = |y_t - y_p| \tag{3.2}$$

Soit la prédiction exacte d'un prix d'un produit à 20$ par un modèle, ainsi qu'une mauvaise prédiction d'un même produit avec une valeur de 12$. Dans le premier cas, l'erreur sera de 0$, soit $|20\$ - 20\$|$. Dans le deuxième cas, l'erreur obtenue sera de 8$, soit |20$ - 12$|.

Bien que l'erreur calculée ait une valeur plus grande dans le deuxième cas, la fonction

de perte n'a pas pénalisé de façon supplémentaire une erreur de prédiction, contrairement au cas où il n'y avait pas d'erreur de prédiction, en ajoutant un facteur ou valeur supplémentaire. Ainsi, la fonction demeure symétrique, se basant sur des équations proportionnelles tant lorsqu'il y a une prédiction juste, que lorsqu'il y a production d'erreurs.

Dans notre cas, nous avons fait utilisation de telles fonctions (i.e. MAE, MSE) dans la solution que nous avons proposée dans [1], envisageant l'utilisation de modèles se basant sur un problème de régression afin de résoudre le problème de détection de zones d'atterrissage sécuritaires.

Dans le domaine de la segmentation, les fonctions de perte *categorical cross-entropy* et *binary cross-entropy*, dérivées de la théorie de l'information [77], sont plus appropriées, comme la segmentation demande la classification de pixels dans la bonne classe. Dans ce travail, nous les utilisons afin de s'attaquer au problème de segmentaton classique comme un problème de classification multiclasses, ainsi que pour le problème de régression ordinale.

3.5.2 Safety-Loss : Fonction de perte pour SLZ

Tandis que les fonctions de perte symétriques permettent la classification, ainsi que la régression, elles peuvent présenter certains problèmes, sachant qu'elles octroient des erreurs proportionnelles, peu importe le problème ou l'ampleur de celui-ci. En effet, dans certains cas, il peut s'avérer utile de pénaliser un type d'erreur plus qu'un autre, apportant un déséquilibre volontaire dans les erreurs qui sont calculés ; Il s'agit ici d'une fonction de perte asymétrique [96]. C'est sur cette théorie qu'a été basée la fonction de perte *Safety Loss* conçue dans ce mémoire.

Prenons par exemple un cas simple, dans lequel on tente de classer un pixel parmi trois classes de sécurité, soit [dangereux, peu dangereux, sécuritaire]. En considérant le domaine des UAVs, on peut imaginer que prédire un pixel comme réellement dangereux (i.e. [0.0 0.0]) comme *sécuritaire* (i.e. [1.0 1.0]), devrait produire une erreur plus grande que la classification d'un pixel *sécuritaire* (i.e. [1.0 1.0]) dans la classe *dangereux* (i.e. [0.0 0.0]). Or, l'erreur générée par la classification d'un pixel *sécuritaire* comme *dangereux* est plus petite que celle produite par la classification d'un pixel *dangereux* comme *sécuritaire*. Utilisée dans l'entraînement d'un modèle capable de reconnaître les SLZ, cette fonction de perte peut favoriser la réduction de la marge d'erreurs, au détriment de la sécurité du UAV.

Pour cette raison, nous avons défini un nouveau facteur de sécurité, nommé *Unsa-*

fety Factor (uf), présenté à l'équation 3.3. Pour un problème à n classes, variant de *dangereux* (0) à *sécuritaire* (n-1), soit y_p la somme du produit cumulatif des prédictions du modèle, lesquelles ont été d'abord été lissées à l'aide de la fonction sigmoïde avec seuil de 0.5, pour un pixel appartenant à une zone classée comme y_t, l'équation est la suivante :

$$uf(y_t, y_p) = \frac{|min(y_t - y_p, 0)| \cdot (n - y_t)}{n^2 - n} \tag{3.3}$$

Ce facteur normalisé octroie de grandes valeurs aux erreurs de classification apportant du danger (i.e. prédire une classe très élevée dans la hiérarchie (sécuritaire = n-1) pour un pixel réellement très bas (dangereux = 0)) et de petites erreurs dans le cas contraire.

Par exemple, revoyons le problème de tout à l'heure, dans lequel des prédictions [1.0 1.0] et [0.0 0.0] pour des étiquettes véritables [0.0 0.0] et [1.0 1.0] avaient respectivement été obtenues. Dans le premier cas, le pixel a été classé comme sécuritaire (i.e. n-1 = 2), alors qu'il est dangereux (i.e. 0). Alors que dans le deuxième cas, le pixel a été classé comme dangereux (i.e. 0), alors qu'il était sécuritaire (i.e. 2), présentant un plus petit risque pour l'UAV dans un cas réel.

Dans le premier exemple, dans lequel la somme du produit cumulatif des valeurs prédites est 2.0, et celle des valeurs véritables est 0.0, nous avons un facteur de sécurité de valeur suivante :

$$\begin{aligned} uf(y_t, y_p) &= \tfrac{|min(y_t - y_p, 0)| \cdot (n - y_t)}{n^2 - n} \\ uf(0.0, 2.0) &= \tfrac{|min(0.0 - 2.0, 0)| \cdot (3 - 0)}{3^2 - 3} \\ uf(0.0, 2.0) &= \tfrac{6.0}{6.0} \\ uf(0.0, 2.0) &= 1.0 \end{aligned}$$

Dans le deuxième cas dans lequel la somme du produit cumulatif des valeurs prédites est 0.0, et celle de la valeur véritable est de 2.0, nous avons un facteur de sécurité de valeur suivante :

$$\begin{aligned} uf(y_t, y_p) &= \tfrac{|min(y_t - y_p, 0.0)| \cdot (n - y_t)}{n^2 - n} \\ uf(2.0, 0.0) &= \tfrac{|min(2.0 - 0.0, 0.0)| \cdot (3.0 - 2.0)}{3.0^2 - 3.0} \\ uf(2.0, 0.0) &= \tfrac{0.0}{6.0} \\ uf(2.0, 0.0) &= 0.0 \end{aligned}$$

Contrairement au cas de la fonction de perte *binary cross-entropy (BCE)*, ce facteur émet une erreur plus grande pour la classification d'un pixel d'une zone dangereuse dans une zone sécuritaire, et vice-versa. Cependant, utiliser seulement ce facteur comme fonction de perte entraînerait, théoriquement, le modèle à classer toutes les zones comme dangereuses afin d'atteindre le minimum global rapidement.

Cet effet étant indésirable, nous avons bâti notre fonction de perte sur ce facteur, en le multipliant par un facteur de régularisation, puis en l'additionnant à la fonction *binary cross-entropy* afin de réprimander le modèle dans son apprentissage s'il tend à attribuer une classe *dangereuse* à tous les pixels. Notre nouvelle fonction de perte finale est présentée sur l'équation 3.4. Soit y_lt et y_lp les niveaux (*levels*) véritables et prédits de chaque pixel, obtenu à partir du produit cumulatif lissé à l'aide de la fonction sigmoïde, ainsi que y_t et y_p les vecteurs de prédictions réelles et prédites du réseau utilisés pour la cross-entropie :

$$E(y_lt, y_lp, y_t, y_p) = \frac{|min(y_lt - y_lp, 0)| \cdot (n - y_lt)}{n^2 - n}\phi + [-(y_t \log(y_p) + (1 - y_t) \log(1 - y_p))] \tag{3.4}$$

Cette fonction de perte asymétrique à deux termes permet de diminuer l'erreur globale de classification du réseau grâce à la cross-entropie d'un côté, peu importe la sécurité du réseau, puis d'équilibrer cette erreur en assurant la sécurité des prédictions avec le deuxième terme, basé sur le *Safety Factor*.

3.6 Meta Learning

Bien que le Safety Loss représente une fonction de perte qui peut apporter un réseau de neurones à améliorer la sécurité des réseaux, il peut arriver des situations dans lesquelles certains réseaux deviennent trop prudents, se voyant suivre la descente du gradient accordant plus d'importance au premier terme qu'au deuxième. Dans cette situation, un réseau pourrait décider, dans une segmentation à 3 niveaux de sécurité, de faire des prédictions de tous les pixels dans les catégories inférieures à *Sécuritaire*, soit la plus haute classe. Ainsi, aucun pixel ne serait classé dans la classe sécuritaire.

Afin de contrer cette possibilité, on a intégré dans les tests de ce mémoire une portion apprentissage à l'aide du *Meta Learning*. De base, cette méthode représente l'apprentissage de l'apprentissage, soit apprendre à apprendre [86]. Dans le cas de ce mémoire, il s'agit de la forme la plus simple, soit imposer un changement de l'hyperparamètre

phi, afin d'obliger la fonction de perte à accorder une plus grande portion de la décision sur la sécurité du drone, ou la réduction de la différence entre la classe réelle et celle prédite, hormis la sécurité du drone.

À la fin de chaque époque de l'entraînement, une matrice de confusion était calculée avec l'état actuel du modèle. Puis, en comparant cette matrice avec celle de l'époque précédente, on ajustait l'hyperparamètre de la fonction de perte selon la logique suivante :

— **Le modèle est trop prudent :** Dans le cas où la variance moyenne des scores sur la diagonale est sous un seuil de 0.05, on détermine que le modèle n'a pas été en mesure de s'améliorer, et donc qu'il est peut-être trop prudent.

— **Le modèle est moins sécuritaire :** Dans le cas où la variance moyenne des scores dans la partie inférieure droite de la matrice de confusion, soit dans lesquelles la classe prédite est d'un niveau supérieur à sa vrai classe, est supérieure à 0.05 le modèle est considéré comme imprudent et on augmente le facteur *phi* de 0.2.

— **Le modèle est trop strict :** Dans le cas où on retrouve une rangée complète de scores de valeur 0 dans des classes de niveaux supérieurs dans la matrice de confusion, on remarque que le modèle est beaucoup trop prudent, n'émettant aucune prédiction dans la dite classe, malgré le fait qu'il y a réellement des pixels dans la classe. Pour cette raison, on doit immédiatement rectifier le modèle en augmentant la valeur de son facteur *phi* de 0.2.

Cette technique permet de faire varier l'apprentissage au fûr et à mesure que le modèle avance dans son entraînement, soit modifier l'apprentissage de celui-ci durant l'apprentissage.

3.7 Entraînement

Lors du lancement de l'entraînement, il est nécessaire de déterminer des valeurs optimales de paramètres d'entraînement afin de diriger le modèle vers une valeur d'erreur minimale dans un délai raisonnable, sans emmener celui-ci vers le surapprentissage. De cette façon, le contrôle de la taille des lots (*batch size*), du nombre d'époques, ainsi que le taux d'apprentissage (*learning rate*) a dû être maintenu en tenant compte du nombre de données disponible et la taille de chaque échantillon

Abordons d'abord la question du *batch size*. Cette valeur détermine le nombre d'images que nos réseaux examinent simultanément lors de chaque itération d'entraî-

nement. Dans notre cas, nous avons choisi une taille de lot de 1, ce qui signifie qu'une seule image est traitée à la fois. Cette stratégie garantit une mise à jour des paramètres du modèle avec une granularité extrême, offrant une fine résolution de l'optimisation du réseau. Cette décision a été prise dû à la taille de celles-ci, en partie, sachant que certains jeux de données comportaient des images qui présentaient de grandes quantités de données, ainsi du fait que certaines modèles utilisent un nombre de paramètres qui pourrait excéder l'espace mémoire qui était disponible sur les GPU.

Le nombre d'époques a été fixé à 100 pour chaque entraînement. Cette décision vise à permettre une convergence suffisante du modèle tout en limitant le risque de surapprentissage. L'objectif est de parvenir à une représentation généralisable des classes d'objets au sein des images. De plus, avec les tests préliminaires effectués lors du projet de mémoire, on a pu observer un phénomène de surapprentissage lorsque les modèles dépassaient les 100 epochs, sachant que les jeux de données comme ICG ne comportent pas beaucoup d'échantillons, portant les modèles à apprendre les classifications par coeur, plutôt que d'inférer une logique.

Enfin, le taux d'apprentissage (*learning rate*) a été déterminé à 1×10^{-4}. Ce paramètre gouverne l'ampleur des ajustements apportés aux poids du modèle à chaque itération. Une valeur soigneusement calibrée est cruciale pour équilibrer la convergence rapide avec la stabilité de l'apprentissage. Cette valeur a été choisi suite aux tests préliminaires effectués lors du projet de mémoire. Bien que la valeur de 1×10^{-3} est préférée dans la documentation Keras [19], comme elle permet de converger plus rapidement vers des prédictions, les modèles semblaient stagner dans l'entraînement après quelques époques lors des tests, faisant en sorte qu'ils n'atteignaient pas leur plein potentiel, soit en corrélation avec les résultats que nous avons obtenus dans ce mémoire.

3.8 Métriques & Évaluation

Afin de déterminer l'amélioration d'un modèle ou d'une stratégie d'entraînement par rapport à une autre, il est nécessaire de se fier aux bonnes mesures. Dans notre cas, comme la classification d'un pixel dans une zone sécuritaire alors qu'il ne l'est pas pourrait engendrer des répercussions graves sur un UAV, il est nécessaire de vérifier où sont placés les pixels lors de la classification, comparativement à là où ils sont censés se retrouver. Pour cette raison, l'un des métriques majeures dont nous faisons utilisation est la matrice de confusion.

Cette dernière affiche, graphiquement, la proportion des pixels qui ont été assignés à une classe, suite à une prédiction, en comparaison à la classe réelle à laquelle ce pixel

appartient. Il est donc facile de déterminer si un modèle a tendance à bien classer les éléments, comme la diagonale de cette matrice de confusion montre les pixels qui ont été classés, lors de la prédiction par un modèle, dans la classe attendue en réalité.

3.8.1 Erreur moyenne absolue *(MAE)*

Cependant, comme la matrice de confusion peut être coûteuse à produire lors de l'entraînement afin de visualiser rapidement l'avancée d'un modèle dans son apprentissage, il était nécessaire de se fier à des métriques existants, pouvant être calculés en très peu de temps. Pour cette raison, nous avons choisi deux métriques supplémentaires pouvant être obtenus de façon plus rapide, soit l'erreur moyenne absolue (Mean Absolute Error (MAE)), ainsi que l'intersection sur union moyenne (Mean Intersection over Union).

Dans un premier temps, le MAE permet d'obtenir une valeur rapide de la distance moyenne entre les prédictions et les classes réelles attendues pour chaque pixel dans l'intervale [0.0, 1.0]. Bien que cette métrique ne soit pas la meilleure, elle permet de déterminer hâtivement si un modèle progresse ou régresse lors de son entraînement.

3.8.2 Intersection sur Union moyenne *(mean IoU)*

Dans un deuxième temps, l'intersection sur l'union moyenne (Mean IoU) permet d'obtenir une vue globale de la performance d'un modèle par rapport au nombre de pixels qu'il classifie dans la bonne classe, nonobstant la proportion des pixels que cette classe appartient dans une image au total. Cette métrique demande de calculer l'intersection, soit le nombre de pixels classifiés comme une classe A qui appartenaient réellement à la classe A, puis diviser ce nombre par l'union des prédictions et des pixels qui se trouvaient dans la classe A en réalité.

— **Tous les pixels ont bien été classés (1.0) :** Dans un tel cas, l'intersection et l'union auront la même valeur, comme chaque pixel de la classe A ont été prédits comme faisant partie de la classe A. Pour cette raison, la valeur du mean IoU serait de 1.0.

— **Aucun pixel a bien été classé (0.0) :** Dans ce second cas, il n'y aura aucun pixel prédit comme classe A qui l'était réellement, produisant une intersection de 0.0. Ainsi, en appliquant la division, le mean IoU sera donc de 0.0, admettant une division de 0 par un autre terme.

— **Certains pixels ont été bien classifiés (]0.0, 1.0[) :** Dans ce dernier cas, la

prédiction émet certains pixels faisant réellement partie de la classe A, ainsi que certains autres ne faisant pas partie de la classe A en réalité. Ainsi, l'intersection se fera toujours plus grande que l'union des deux, comme au minimum l'intersection compte tous les pixels qui appartenaient réellement à la classe A, dont moins de 100% ont été prédits par le modèle. Pour cette raison, le reséultat sera toujours une valeur pour ce métrique d'un nombre décimal compris dans l'intervalle]0.0, 1.0[.

Cette métrique est importante, car elle est représentative de l'entraînement, peu importe le débalancement des classes. À titre d'exemple dans le domaine des UAV, comme cette métrique se calcul sur chaque classe de façon distincte, l'intervalle de valeurs de la métrique pour une classe humain, très peu visible dans une prise de vue à haute altitude, est proportionnel à l'intervalle de valeurs pour une classe plus dominante comme les bâtiments ou les pavés.

3.8.3 Facteur de sûreté *(Safety Score)*

Bien que les deux métriques présentées précédemment permettent de visualiser, à travers des nombres, la progression d'un modèle dans son entraînement, elles présentent un problème, soit le fait qu'elles ne montrent pas le degré de sûreté d'une prédiction d'un modèle dans le contexte d'un UAV. Prenons un exemple simple, soit un modèle qui permet de classer un pixel à travers 3 niveaux de sécurité, soit dangereux (0), peu dangereux (1) et sécuritaire (2). En théorie, un modèle qui performe bien ne mettra pas la vie du UAV ou d'un pilote en jeu lors de ses prédictions, pour cette raison, un métrique d'une classification *sécuritaire (2)* d'un pixel faisant réellement partie de la classe *dangereux (0)* devrait donner une valeur avec une performance moins élevée que la classification contraire, soit classer un pixel comme *dangereux (0)*, alors que celui-ci est réellement *sécuritaire (2)*.

Dans le cas du MAE, soit la différence absolue moyenne, la distance serait la même dans les deux, soit $|2-0|$ dans le premier cas, puis $|0-2|$ dans le deuxième cas, donnant une métrique MAE égale de valeur *2* dans les deux cas. Du côté du mean IoU, dans les deux cas l'intersection serait de 0 pour ce pixel, sachant que la classe prédite n'est pas la même que la classe réelle de ce pixel, donnant une valeur égale de 0.0 dans les deux cas pour cette métrique.

Pour cette raison, basé sur les mêmes principes que notre *Safety Loss*, nous avons apporté une nouvelle métrique, permettant de déterminer la sûreté d'un modèle, soit sa capacité à classer les pixels dans la bonne classe, n'émettant pas de surclassification,

soit une classification d'un niveau supérieur à celui dont le pixel appartient réellement, lorsque l'on traite le problème comme un problème de régression ordinale.

On peut calculer le score du *Safety Score* en suivant les étapes suivantes :

1. **Trouver la classe d'un pixel** : Comme expliqué dans la section 3.4, il est important de d'abord transformer la représentation de forme régression ordinale par une forme numérique, soit déterminer, niveau par niveau, si un pixel a passé un seuil pour chaque classificateur, lui assignant un niveau supérieur. Prenons par exemple, une classification à trois niveaux (2 classificateurs), avec un seuil de passage de 0.5, qui a donné la prédiction [0.82, 0.23]. Dans ce cas, seul le premier classificateur surpasse le seuil de 0.5, faisant passer le pixel le premier test (classificateur), lui attribuant alors la valeur numérique de 1. Dans le cas où aucun classificateur n'aurait vu sa prédiction surpasser le seuil de 0.5, la classe numérique de 0 aurait été attribuée au pixel, et ainsi de suite.
2. **Calculer le Safety Score** : À partir des classes numériques trouvées pour la prédiction y_p et la classe réelle d'un pixel y_t, d'un problème à n niveaux possibles, utilisant $n-1$ classificateurs, on obtient le *Safety Score* en utilisant l'équation 3.5.

$$f(y_t, y_p) = 1.0 - \frac{|min(y_t - y_p, 0)|}{max(n - 1 - y_t, 1)} \tag{3.5}$$

La méthode de pensée derrière cette équation se décrit mieux lorsqu'on la sépare en deux portions, soit le numérateur et le dénominateur.

En termes mathématiques, le numérateur détermine la distance entre une prédiction et sa valeur réelle, en ne considérant que les classifications où la prédiction était d'un niveau supérieur à la classe réelle. Le minimum agit donc comme un filte, permettant d'émettre une valeur de *Safety Factor* seulement pour les pixels qui ne sont pas sécuritaires, donc qui émettraient une différence négative entre la valeur réelle et celle prédite. Par exemple, la valeur du numérateur d'un pixel que l'on a classé comme *sécuritaire* ($y_p = 2$) alors qu'il était *dangereux* ($y_t = 0$) serait de 2, comme la différence 0-2 serait négative, donc $|-2|$. Dans le cas contraire, un pixel prédit comme *dangereux* ($y_t = 0$) alors qu'il est *sécuritaire* ($y_p = 2$) émet un numérateur de 0, car nous choisissons le minimum entre 0 et 2, soit 0.

De l'autre côté, le dénominateur permet de normaliser la différence. On se base sur la distance maximale la moins sécuritaire qu'une classification pourrait effectuer. En ce

sens, réfléchissons à un problème où on fait la classification avec 5 niveaux de sécurité, soit des valeurs numériques de classification dans l'intervalle [0, 4], avec 4 classificateurs. Ainsi, pour 5 niveaux ($n = 5$), la valeur maximale que peut atteindre une classification est de 4 ($n - 1$), en surpassant les seuils de chaque classificateur. Dans ce même cas, prenons par exemple un pixel qui a été classé comme *sécuritaire* ($y_p = 3$, alors qu'il était *dangereux* ($y_t = 1$). Dans le cas où la prédiction se trouve en dessous de la valeur de la classe *dangereux* ($y_p = 1$), la prédiction ne présente aucun danger pour l'UAV, étant donc négligeable. Pour cette raison, l'erreur de sûreté est donc de valeur 0.0 lorsque la prédiction est exactement la valeur réelle attendue, puis maximale lorsque la prédiction est au niveau le plus sécuritaire possible.

Suivant cette logique, le dénominateur $n - 1 - y_t$ montre l'intervalle possible de valeurs qui auraient causées une mauvaise classification, dans l'intervalle $[y_t,\ n - 1]$, soit [0, maximum]. Puis, comme une classification parfaite produit une division par zéro, ceci même si le numérateur a la valeur de 0, l'ajout d'une opération de sélection du maximum entre ce dénominateur et la valeur 1 a été effectuée, n'affectant pas la valeur finale du score, comme le numérateur demeure 0. Finalement, on retire de la valeur maximale 1.0, ce résultat normalisé, montrant le niveau d'insécurité, permettant d'obtenir le facteur de sécurité, soit la distance du maximum (1.0) de la sûreté du modèle.

Bien que cette métrique permettre d'observer rapidement la progression d'un modèle en fonction de la sûreté des prédictions qu'il produit, cette métrique n'est pas parfaite. En effet, la normalisation permet d'établir sur un même niveau deux classifications qui, en pratique, ne représentant pas le même danger. Prenons deux exemples distincts dans une segmentation à 5 niveaux, soit un modèle qui prédit un pixel comme *Très sécuritaire* (4), alors qu'il est *Très dangereux* (0) dans un premier cas, ainsi qu'un pixel classé comme *Très sécuritaire* (4), alors qu'il est *sécuritaire* (3).

Le premier cas émet une valeur de 4 au numérateur, puis 4 au dénominateur, donnant un *Safety Factor* de 0. Quant à lui, le deuxième cas émet une valeur de 1 au numérateur, puis 1 au dénominateur, donnant un *Safety Factor* de 0 lui aussi. Pourtant, classer un pixel qui représente un danger comme sécuritaire pose un plus grand problème que classer un pixel sécuritaire comme très sécuritaire. C'est en autre la raison pour laquelle le *Safety Factor* présenté dans notre fonction de perte *Safety Loss* présente une version modifiée de cette idée originale, émettant elle un facteur beaucoup plus grand pour une erreur de plus grande taille.

Cependant lors de l'entraînement d'un modèle, de façon globale, une amélioration de ce facteur permet de montrer que le modèle progresse bien en réduisant le danger

qu'apportent ses prédictions, peu importe la taille de ce danger, représentant une métrique de plus afin de guider les entraînements. De plus, comme elle est normalisée, donc bornée dans l'intervalle [0.0, 1.0], cette métrique est facilement lisible pour un humain en temps réel, d'où les raisons pour lesquelles nous avons choisi de la conserver à travers les entraînements sous problématique de régression ordinale.

Chapitre 4

Résultats et discussion

À partir des pistes de solutions proposées qui sont décrites au chapitre précédent, nous avons utilisé un serveur, comprenant deux cartes graphiques NVIDIA RTX 3060 avec 12 Go de mémoire chacune, afin d'entraîner des modèles, permettant de déterminer la véracité des hypothèses posées précédemment.

Les tests ont été divisés en trois volets, lesquels ont permis de nous diriger dans ce vaste éventail de possibilités. Dans un premier temps, une base *(benchmark)* a été établie afin de déterminer, parmi la liste des modèles énoncée précédemment et conçus dans la librairie Keras [19], lesquels permettent d'obtenir les meilleurs résultats.

Une fois les modèles évalués séparément, nous avons sélectionné les meilleurs et nous sommes dirigé plus près d'une approche par régression ordinale, en abordant d'abord le problème sous sa forme à régression, laquelle est décrite dans un papier que nous avons publié [1].

Puis, dans une troisième approche, nous avons abordé le problème sous une approche par régression ordinale, apportant une nouvelle fonction de perte, ainsi qu'une technique d'ajustement des hyper-paramètres durant l'entraînement visant à réduire le risque d'une mauvaise classification d'une zone sur un UAV et son entourage.

Les résultats des tests, ainsi que la discussion qui s'en suit sont présentés ci-après.

4.1 Détection des SLZ par Segmentation

Le problème a tout d'abord été solutionné comme un problème de segmentation. À partir des architectures Unet, LinkNet et DeepLabV3+, nous avons fait sélection de backbones MobileNetV2, pour sa vitesse, InceptionV3 pour ses résultats, ainsi que

ResNet101, pour son approche résiduelle basique, nous avons fait l'entraînement sur 100 epochs, sans sur-entraînement, sur les jeux de données ICG, UAVID et VALID. Les résultats sont présentés dans la section suivante.

4.1.1 Prises de vues

UNet fût la première architecture testée. Après 100 époques, tant à 3 niveaux de sûreté qu'à 5 niveaux de sûreté, les classifications résultantes semblent constantes et émettent des pixels qui se trouvent dans les trois et cinq niveaux, respectivement. Dans l'ensemble, les formes des objets (e.g. maisons, piscines, clôtures, etc.) sont respectées, émettant des images qui permettent de visuellement reconnaître les grandes classes thématiques d'une prise de vues. Des résultats de classifications sont présentés sur la figure 4.1, montrant les résultats de trois backbones, appliqués à l'architecture UNet, à trois et cinq niveaux de sûreté.

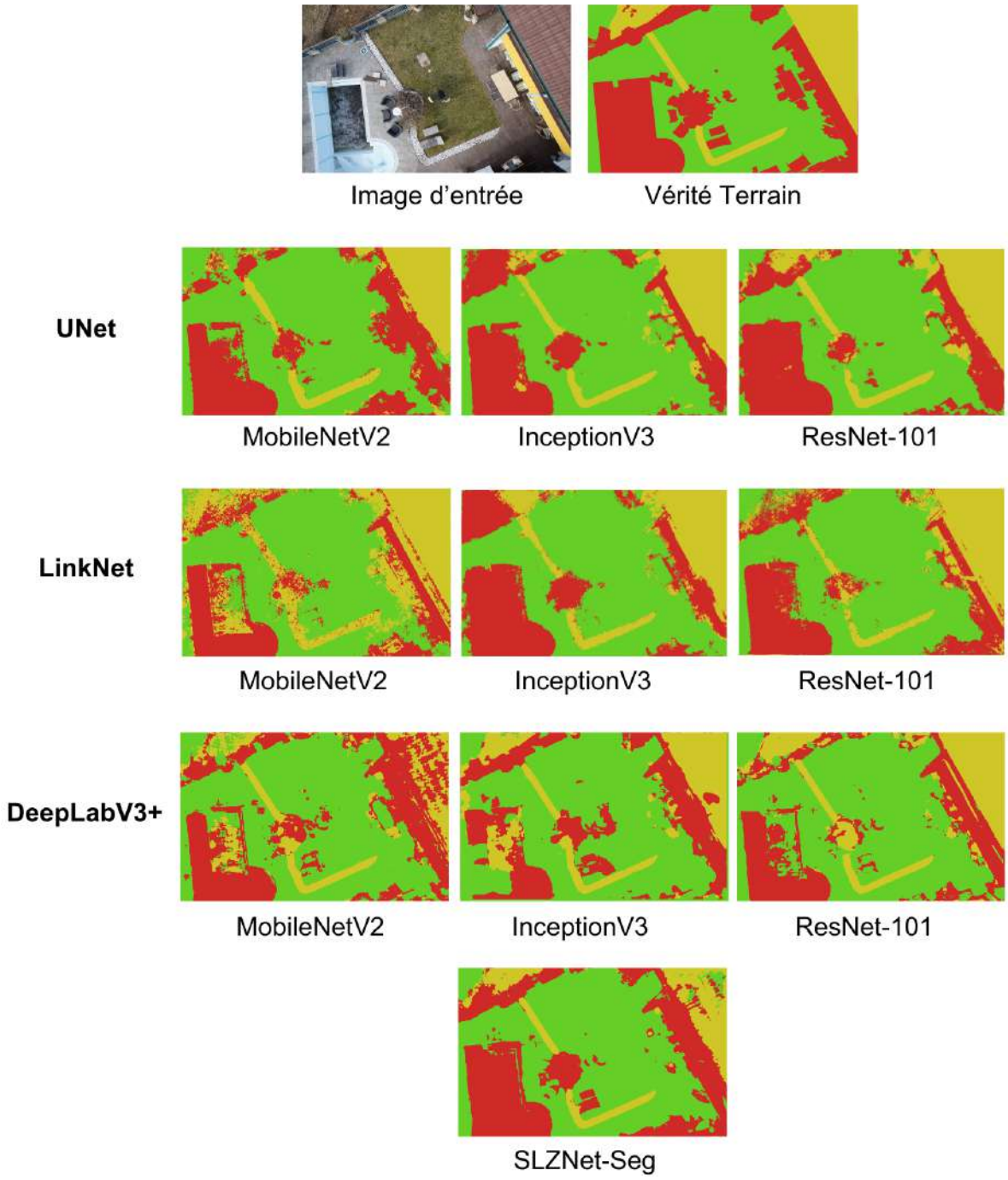


FIGURE 4.1 – Résultats de segmentation d'une image, tirée du dataset ICG, sur les architectures UNet, LinkNet et DeepLabV3+, à partir de différents backbones. Une image d'entrée est présentée avec les résultats sous une classification à 3 niveaux de sûreté (dangereux, peu sécuritaire, sécuritaire).

La deuxième architecture testée fut LinkNet, offrant un nombre plus petit de paramètres dans l'ensemble et des complexités de calculs plus minces [12] que le UNet. Afin de permettre la comparaison avec le réseaux UNet présenté précédemment, la figure

4.1 présente les résultats de classification des mêmes backbones, basés cette fois-ci sur l'architecture LinkNet, à partir d'une même image d'entrée. Ainsi, dans un terrain d'entraînement semblable, avec une image d'entrée identique, il est possible de comparer les résultats obtenus dans les deux réseaux.

On peut remarquer que les classifications obtenus grâce à l'utilisation de l'architecture LinkNet sont plus pixelisées et démontrent moins de cohérence que la classification précédente. Et ce, surtout dans sa variante avec *backbone* MobileNetV2. De plus, lorsqu'on compare avec la version à 3 niveaux de sécurité, la version à 5 niveaux de sécurité présentent peu, ou pas de zones marquées comme dangereuses, alors qu'elles sont dénombrées en grande quantité sur la vérité terrain (*Ground Truth*).

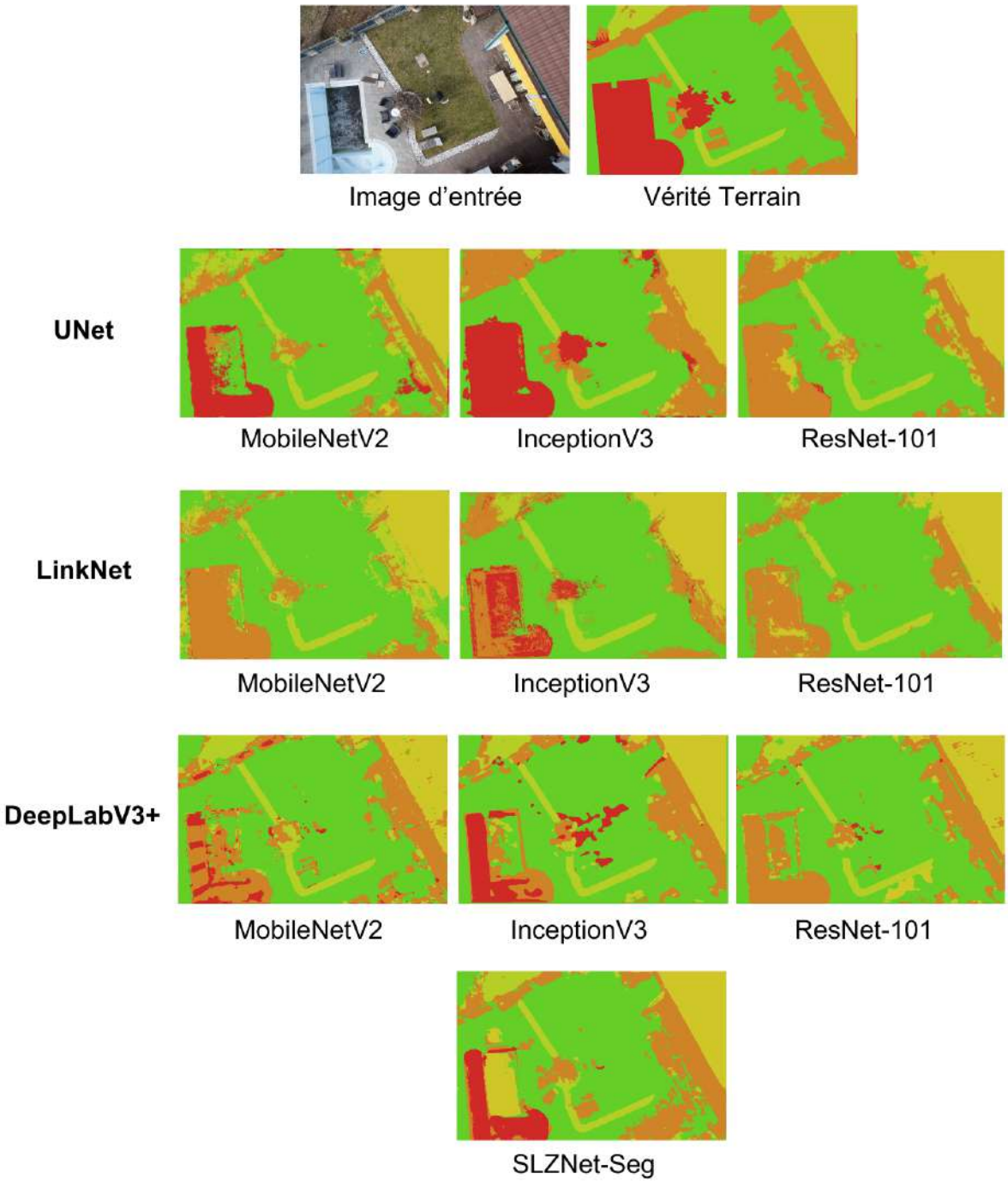


FIGURE 4.2 – Résultats de segmentation d'une image, tirée du dataset ICG, sur les architectures UNet, LinkNet et DeepLabV3+, à partir de différents backbones. Une image d'entrée est présentée avec les résultats sous une classification à 5 niveaux (très dangereux, dangereux, moyennement dangereux, peu sécuritaire, sécuritaire).

Finalement, la technique a été répétée avec la même image d'entrée, ainsi que les mêmes *backbones*, avec l'architecture DeepLabV3+, permettant de comparer avec les architectures précédentes. Dans son ensemble, les résultats semblent démontrer des re-

groupements de pixels (i.e. "*blob*"), présentant des objets, plutôt que des classifications pixelisées sans contexte regroupant les zones. Cependant, tout comme pour le cas du LinkNet, les résultats de classification à 5 niveaux présentent beaucoup moins de régions dangereuses que ce dont il en est question dans la vérité terrain (*Ground Truth*).

Bien qu'il soit facile de représenter, visuellement, les différences des *backbones* sur une seule figure, tel qu'il en est le cas sur la figure 4.2, il est important de démontrer visuellement les différences des trois architectures, telles que présentée dans le chapitre sur l'état de l'art 2.

4.1.2 Matrices de confusion

Alors que les figures précédentes permettent de visuellement distinguer une architecture d'une autre, les différences sont minces quant à la qualité de la classification produite, telle qu'elle se dénote en pixels, et non pas en zones. Pour cette raison, la figure 4.3 dresse la liste des matrices de confusion de chaque architecture, en conjonction avec les *backbones*, sur le jeu de données UAVID, à 3 niveaux de sécurité. Cette décision est dû à présenter visuellement les résultats qui sont présentés sur la figure précédente.

À partir de cette matrice de confusion, on s'aperçoit que les modèles basés sur l'architecture UNet ont tendance à produire des résultats qui présentent une diagonale plus imposante que dans les autres cas, avec des valeurs de 68%, 58% et 89% pour le *backbone* MobileNetV2, 61%, 79% et 93% pour le *backbone* InceptionV3, ainsi que 71%, 62% et 84% pour le ResNet 101, dans ses niveaux *dangereux*, *peu sécuritaire* et *sécuritaire*. De son côté, le réseau LinkNet produit des résultats qui sont plus stables, apportant sa diagonale à 62%, 63% et 67% pour le *backbone* MobileNetV2, 68%, 76% et 86% pour le *backbone* InceptionV3, ainsi que 65%, 60% et 87% pour le ResNet 101. Pour sa part, l'architecture DeepLabV3+ présente des diagonales de 70%, 50% et 81% pour le *backbone* MobileNetV2, 64%, 68% et 79% pour le *backbone* InceptionV3, ainsi que 57%, 53% et 85% pour le ResNet 101. Finalement, le réseau conçu à l'intérieur de ce projet de mémoire, SLZNet-Seg, voit sa diagonale s'aligner avec celle des meilleurs modèles, soit 78%, 55% et 91%, tirant le meilleur des modèles. Il est à remarquer qu'il produit la plus haute valeur de zones dangereuses classées comme dangereuses, et non pas comme *peu sécuritaire* ou *sécuritaire*.

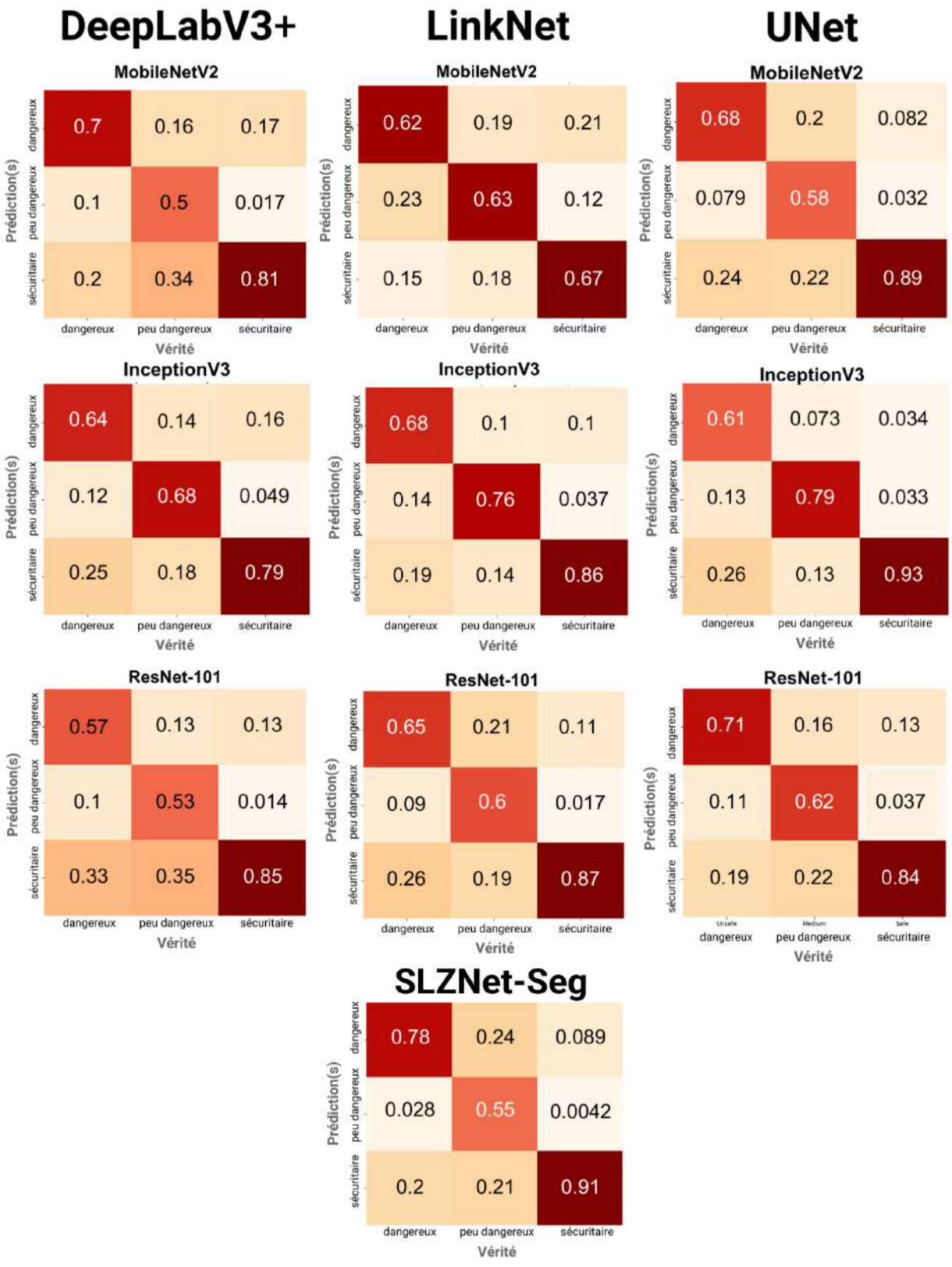


FIGURE 4.3 – Matrices de confusion, tirées d'un sous-échantillon d'images test du dataset UAVID, sur les architectures UNet, LinkNet, DeepLabV3+ et SLZNet-Seg.

La figure 4.4 montre les matrices de confusion, tirées d'un échantillon test du jeu de données UAVID, des modèles DeepLabV3+ et LinkNet, alors que la figure 4.5 montre les matrices de confusion pour les modèles basés sur les architectures UNet et SLZNet-Seg

Selon les matrices, les modèles basés sur l'architecture DeepLabV3+ émettent de bonnes performances sur la classe *peu dangereux*, classant entre 86% et 89% des pixels parfaitement. Cependant, les classes *très dangereux* et *dangereux* se voient classées entre 22% et 26% dans la classe peu dangereux, ainsi que 15% et 23% dans la classe sécuritaire respectivement. De l'autre côté, l'architecture LinkNet présente des résultats similaires, à l'exception du MobileNetV2 qui a vu sa classification des pixels réellement *très dangereux* prédite *peu dangereux* augmenter à 33%, soit nettement supérieur à ce qui est présenté dans l'architecture DeepLabV3+. De cette présentation, on comprend que les modèles basés sur l'architecture Linknet ont tendance à présenter des résultats de classifications présentant des dangers pour les UAV, soit sous la diagonale, à plus de 25%.

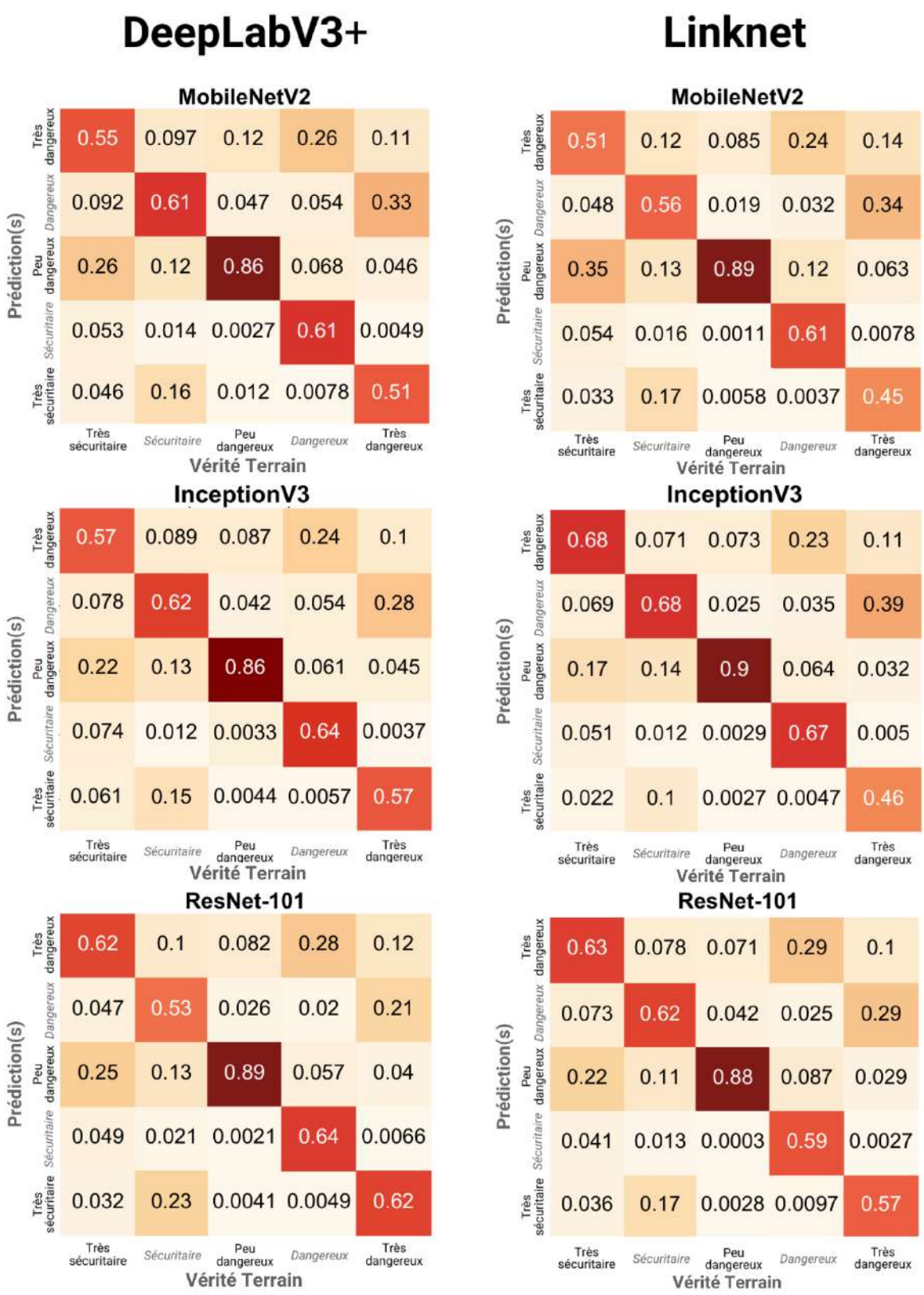


FIGURE 4.4 – Matrices de confusion à 5 niveaux de sûreté, tirées d'un sous-échantillon d'images test du dataset UAVID, sur les architectures LinkNet, DeepLabV3+ et SLZNet-Seg.

Quant à eux, les modèles basés sur l'architecture UNet ont de meilleurs résultats au niveau de la diagonale, soit même une classification des pixels peu dangereux bien classés entre 89% et 92%, ainsi que les pixels très dangereux bien classés à 52%, 71% et 62%, soit des valeurs nettement supérieures aux deux architectures présentées précédemment. Cependant, cette architecture semble très prudente dans les classifications du niveau très sécuritaire, n'ayant classé correctement qu'entre 34% et 51% les pixels appartenant à cette classe. Le restant des prédictions appartiennent majoritairement aux classes *dangereux* (entre 33% et 48%) et *très dangereux* (entre 7.8% et 12%).

Finalement, le modèle SLZNet-Seg montre, dans la classification à 5 niveaux de sûreté, une diagonale qui émet des valeurs surpassant les 50% dans toutes les catégories, réduisant le facteur de prudence montré dans les architectures UNet et LinkNet avec une valeur de 55% de bonnes classifications pour la classe *très dangereux*, ainsi que 58% pour la classe *dangereux*. Cette architecture est la seule à présenter une valeur de 71% des pixels bien classés dans la classe *dangereux*, surpassant toutes les autres architectures. Ce point est important, comme il montre que cette architecture présente un faible nombre de pixels mal classés dans les classes qui apportent un danger au UAV s'il y a mauvaise classification.s

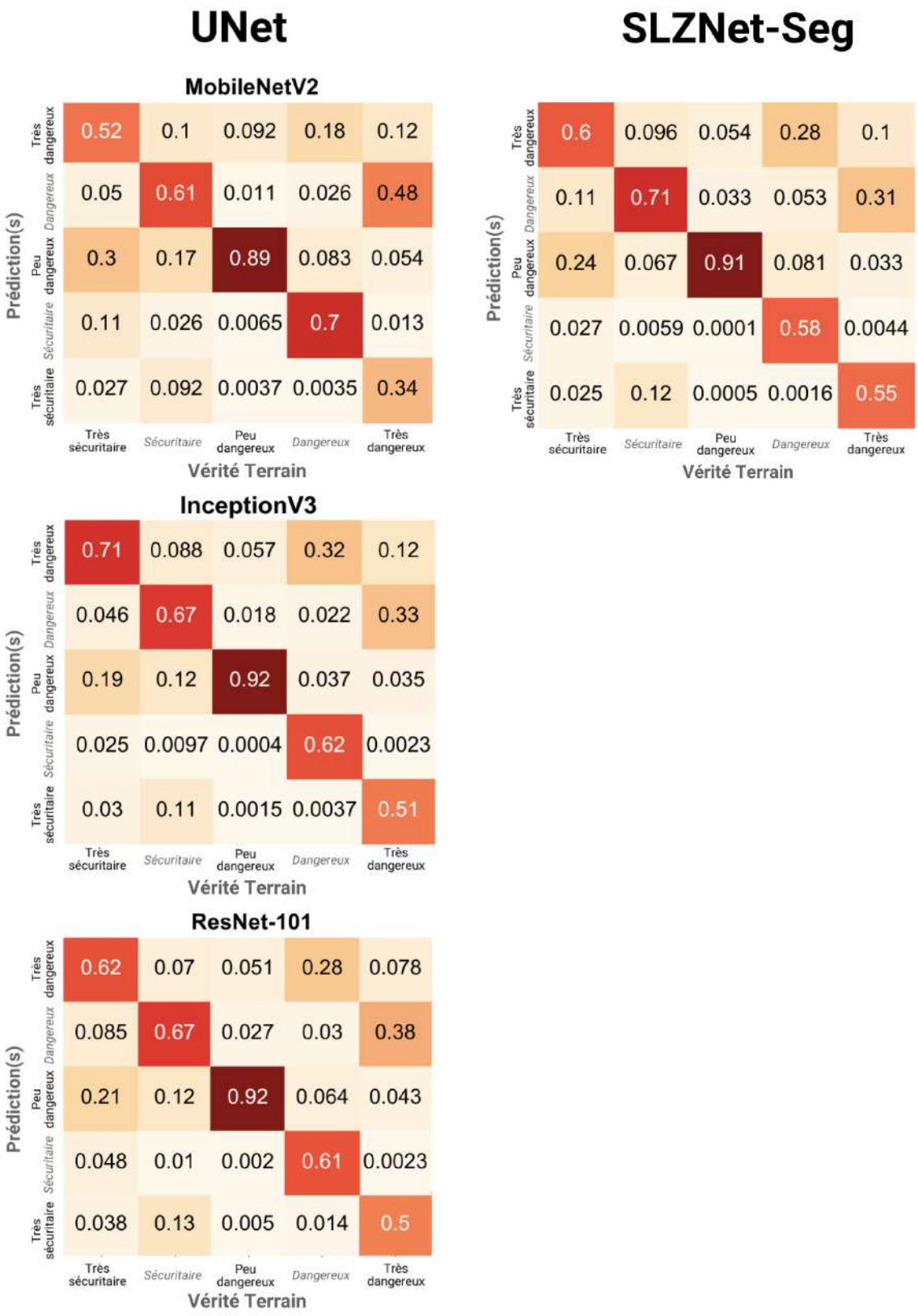


FIGURE 4.5 – Matrices de confusion à 5 niveaux de sûreté, tirées d'un sous-échantillon d'images test du dataset UAVID, sur les architectures UNet et SLZNet-Seg.

4.1.3 Métriques

Somme toute, bien que les matrices de confusion et les images permettent de visuellement différencier les architectures, bâties à partir de trois *backbones*, elles ne permettent pas de trancher rapidement sur les architectures qui produisent les meilleurs résultats en un coup d'oeil. La métrique d'intersection sur union moyenne (*Mean Intersection over Union (mean IoU)*) permet de déterminer le nombre de pixels qui se sont bien classé, en fonction du nombre de pixels total de cette catégorie. Par conséquent, si une prise de vue voit sa zone *dangereux* s'étaler sur 90% des pixels, compter le nombre de pixels sur le total de l'image ne serait pas valable, car les pixels *peu sécuritaires* et *sécuritaires* ne verraient que 10% de l'image aux 2, faisant augmenter le pourcentage d'erreurs possibles. Ce problème est rectifié dans l'intersection sur union moyenne. Les métriques de chaque modèle sont présentés dans la série de tables suivante.

		Entraînement		**Validation**		**Test**	
Architecture	**Backbone**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**
DeepLabV3+	InceptionV3	0.2125	0.6985	0.2023	0.7127	0.2143	0.6854
DeepLabV3+	MobileNetV2	0.2523	0.6723	0.2387	0.6624	0.2484	0.6294
DeepLabV3+	ResNet101	0.2676	0.6937	0.2534	0.6509	0.2734	0.6310
LinkNet	InceptionV3	0.1558	0.7905	0.1541	0.7919	0.1661	0.7779
LinkNet	MobileNetV2	0.1637	0.7889	0.1580	0.7974	0.1739	0.7776
LinkNet	ResNet101	0.1648	0.7815	0.1537	0.7942	0.1718	0.7728
Unet	InceptionV3	0.1680	0.7838	0.1736	0.7788	0.1798	0.7709
Unet	MobileNetV2	0.1779	0.7795	0.1695	0.7928	0.1838	0.7741
Unet	ResNet101	0.1685	0.7765	0.1593	0.7868	0.1804	0.7624
SLZNet-Seg	-	0.1261	0.8260	0.1227	0.8291	0.1380	0.8114

TABLE 4.1 – Métriques des modèles évalués sur le jeu de données VALID [15] lorsque entraînés sur 3 niveaux de sécurité, allant de peu dangereux à sécuritaire.

		Entraînement		Validation		Test	
Architecture	**Backbone**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**
DeepLabV3+	InceptionV3	0.6583	0.5182	0.6270	0.5390	0.6869	0.5083
DeepLabV3+	MobileNetV2	0.6510	0.5203	0.6181	0.5426	0.6812	0.5095
DeepLabV3+	ResNet101	0.5047	0.6955	0.4851	0.7043	0.5116	0.6958
LinkNet	InceptionV3	0.3106	0.7996	0.3050	0.8062	0.3278	0.7938
LinkNet	MobileNetV2	0.3035	0.7968	0.3043	0.7950	0.3301	0.7856
LinkNet	ResNet101	0.3116	0.8023	0.3041	0.8070	0.3252	0.7983
Unet	InceptionV3	0.2868	0.8093	0.2875	0.8059	0.3194	0.7953
Unet	MobileNetV2	0.3114	0.8031	0.3083	0.8041	0.3440	0.7885
Unet	ResNet101	0.3203	0.7915	0.3024	0.7991	0.3373	0.7850
SLZNet-Seg	-	0.2297	0.8447	0.2283	0.8455	0.2476	0.8366

TABLE 4.2 – Métriques des modèles évalués sur le jeu de données VALID [15], lorsqu'entraînés sur 5 niveaux de sécurité, allant de dangereux à sécuritaire.

Les résultats de l'évaluation des modèles sur le jeu de données VALID, caractérisés par des niveaux de sécurité variables (3 niveaux dans la Table 4.1 et 5 niveaux dans la Table 4.2), révèlent des nuances significatives dans les performances en fonction des architectures et des backbones employés.

Au niveau des trois niveaux de sécurité, les performances des modèles varient considérablement, indiquant une sensibilité à la complexité des situations de sécurité. Sur les données de test, avec l'architecture DeepLabV3+, le backbone InceptionV3 présente des valeurs d'erreur moyenne de 0.2143, soit beaucoup plus petite que ses deux comparables de la même architecture à 0.2484 et 0.2734. De plus ce backbone présente une valeur d'intersection sur union touchant le 0.6854, comparativement à 0.6294 et 0.6310 pour les deux autres backbones. Cette observation ne continue cependant pas lorsque l'architecture est confronté à 5 niveaux, voyant le backbone ResNet101 mieux performer avec un MAE de 0.5116, comparativement aux deux autres présentant 0.6869 et 0.6812. De plus, le backbone ResNet101 présente une moyenne d'IoU de 0.6958, soit nettement supérieure aux autres backbone, ne présentant que 0.5083 et 0.5095. Ces résultats montrent donc que ce backbone a tendance à classer une plus grande quantités de pixels dans leur classe respective, étant plus près des vrais valeurs que les deux autres backbones.

Pour l'architecture LinkNet, le phénomène précédent revient de nouveau pour le backbone InceptionV3, présentant une valeur de MAE de 0.1661, comparativement à 0.1739 et 0.1718, et une valeur de mean IoU de 0.7779, comparativement à 0.7778 et 0.7728 pour les réseaux avec backbone MobileNetV2 et ResNet101. Pour ce que qui est du 5 niveaux, tout comme pour l'architecture DeepLabV3+, l'architecture LinkNet

obtient de meilleurs résultats avec le backbone ResNet101, présentant un MAE de 0.3252 et mean IoU de 0.7983, comparativement au MobileNetV2 (0.3301 et 0.7856) et InceptionV3 (0.3278 et 0.7938).

L'architecture UNet présente presque la même variante que les deux autres architectures, à une exception près. À trois niveaux de sécurité, le backbone InceptionV3 obtient bel et bien le plus bas score de MAE avec 0.1798, mais le backbone MobileNetV2 obtient un plus haut score de mean IoU avec une valeur de 0.7741. De l'autre côté, le réseau avec un backbone de ResNet101 obtient un score MAE de 0.1804 et un score mean IoU de 0.7624. Quant à l'architecture présentant 5 niveaux de sécurité, le backbone InceptionV3 démontre une MAE plus faible de 0.3194 que ses comparables, soit 0.3440 pour MobileNetV2 et 0.3373 pour ResNet101. Pour ce qui est du score moyen IoU, il s'élève à 0.7953 pour le backbone InceptionV3, alors qu'à seulement 0.7885 et 0.7850 pour le backbone MobileNetV2 et ResNet101 respectivement.

Finalement, le réseau SLZNet-Seg conçu, a obtenu un score MAE de 0.1380, soit le plus beau de tous les réseaux, et un score mean IoU de 0.8114 pour son architecture à 3 niveaux de sécurité. Pour son architecture à 5 niveaux, il a obtenu un score MAE de 0.2297 et un score mean IoU de 0.8366, soit encore une fois des valeurs de métriques beaucoup plus élevées que les comparables des autres réseaux, démontrant sa capacité à tirer le meilleur des architectures précédentes, sans pour autant conserver leurs points faibles.

		Entraînement		**Validation**		**Test**	
Architecture	**Backbone**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**
DeepLabV3+	InceptionV3	0.3222	0.7183	0.4296	0.6680	0.2480	0.7504
DeepLabV3+	MobileNetV2	0.3676	0.6861	0.4239	0.6685	0.2582	0.7326
DeepLabV3+	ResNet101	0.3790	0.6803	0.4484	0.6587	0.3397	0.6712
LinkNet	InceptionV3	0.2552	0.7637	0.3205	0.7324	0.2276	0.7632
LinkNet	MobileNetV2	0.3975	0.6490	0.5100	0.5904	0.2722	0.7224
LinkNet	ResNet101	0.3133	0.7260	0.3711	0.7050	0.2828	0.7257
Unet	InceptionV3	0.2445	0.7704	0.2739	0.7638	0.2582	0.7626
Unet	MobileNetV2	0.3058	0.7293	0.3339	0.7203	0.2681	0.7292
Unet	ResNet101	0.2663	0.7614	0.3585	0.7056	0.2223	0.7735
SLZNet-Seg	-	0.2551	0.7647	0.2923	0.7574	0.2044	0.7923

TABLE 4.3 – Métriques des modèles évalués sur le jeu de données ICG, lorsque entraînés sur 3 niveaux de sécurité, allant de peu dangereux à sécuritaire.

		Entraînement		Validation		Test	
Architecture	**Backbone**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**
DeepLabV3+	InceptionV3	0.6952	0.7030	0.8494	0.6691	0.5428	0.7265
DeepLabV3+	MobileNetV2	0.5782	0.7373	0.7208	0.7024	0.3635	0.7902
DeepLabV3+	ResNet101	0.6602	0.7036	0.8168	0.6703	0.5502	0.7120
LinkNet	InceptionV3	0.4629	0.7800	0.5969	0.7305	0.5403	0.7319
LinkNet	MobileNetV2	0.5895	0.7090	0.6939	0.6847	0.5323	0.7011
LinkNet	ResNet101	0.5403	0.7498	0.6492	0.7137	0.4874	0.7486
Unet	InceptionV3	0.3706	0.8123	0.4567	0.7896	0.3249	0.8172
Unet	MobileNetV2	0.4997	0.7600	0.5873	0.7368	0.4591	0.7537
Unet	ResNet101	0.5103	0.7574	0.6322	0.7188	0.3863	0.7854
SLZNet-Seg	-	0.4800	0.7682	0.6315	0.7393	0.2974	0.8322

TABLE 4.4 – Métriques des modèles évalués sur le jeu de données ICG, lorsque entraînés sur 5 niveaux de sécurité, allant de peu dangereux à sécuritaire.

Comme le jeu de données ICG est un jeu de données qui présente des images en moins grande nombre, les modèles avaient tendances à se diriger rapidement vers le surapprentissage. Or, le point positif de ce jeu de données est qu'il présente des captures de la vie réelle, non pas une simulation, à une haute définition. Contrairement au jeu de données VALID, le jeu de données ICG a été capturé à une altitude moyenne ce qui permet aux modèles de pouvoir bien définir les humains, les voitures, les vélos et les autres détails qui ne représentent que quelques pixels sur une capture du jeu de données VALID.

Cette remarque s'exprime à travers les métriques qui ont été obtenus, voyant l'intervalle des Mean IoU des données tests passer de [0.6294, 0.8114] à [0.6712, 0.7923] lors de la segmentation à 3 niveaux de sécurité, ainsi que de [0.5095, 0.8366] à [0.7011, 0.8322] lors de la segmentation à 5 niveaux de sécurité. L'augmentation du seuil minimal de l'intervalle des valeurs de Mean IoU permet d'indiquer que les classes, autant petites quelles soient, semblent mieux représentées par les modèles dans le jeu de données ICG que dans le jeu de données VALID.

Pour ce qui est des modèles, il y a cependant une variation des meilleurs backbone par architecture, contrairement au jeu de données VALID. Dans le cas de la segmentation à 3 classes, les meilleurs modèles (selon le score le plus haut de Mean IoU) semblent comporter le backbone InceptionV3 dans les architectures DeepLabV3+ et Linknet. Cependant, le UNet conserve tout de même le backbone ResNet101 comme modèle avec le meilleur score Mean IoU. Parmi tout le lot d'architectures, le SLZNet-Seg demeure cependant le modèle qui performe le mieux avec un Mean IoU de 0.7923, soit supérieur au deuxième meilleur modèle le Unet avec backbone ResNet101 qui a un Mean IoU de

0.7735.

Quant à la segmentation à 5 niveaux de sûreté, le modèle SLZNet-Seg demeure le meilleur au niveau du Mean IoU, offrant une valeur de 0.8322, comparativement au deuxième meilleur modèle, basé sur l'architecture UNet avec le backbone InceptionV3 offrant un Mean IoU de 0.8172. Dans cette configuration, le backbone MobileNetV2 présente une meilleure valeur Mean IoU que les deux autres backbones pour l'architecture DeepLabV3+, puis le backbone ResNet101 pour l'architecture Linknet.

		Entraînement		**Validation**		**Test**	
Architecture	**Backbone**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**
DeepLabV3+	InceptionV3	0.3139	0.7240	0.3885	0.6614	0.3922	0.6583
DeepLabV3+	MobileNetV2	0.4400	0.6002	0.4329	0.6054	0.4829	0.5645
DeepLabV3+	ResNet101	0.3775	0.6657	0.3872	0.6554	0.4004	0.6403
LinkNet	InceptionV3	0.2659	0.7611	0.3261	0.7106	0.3364	0.7053
LinkNet	MobileNetV2	0.3291	0.7075	0.3777	0.6617	0.4119	0.6312
LinkNet	ResNet101	0.2762	0.7560	0.3377	0.7005	0.2980	0.7285
Unet	InceptionV3	0.2911	0.7490	0.3403	0.7060	0.3182	0.7200
Unet	MobileNetV2	0.3634	0.6801	0.3975	0.6495	0.4283	0.6245
Unet	ResNet101	0.3101	0.7342	0.3658	0.6843	0.3467	0.6970
SLZNet-Seg	-	0.3365	0.7047	0.3502	0.6882	0.3660	0.6781

TABLE 4.5 – Métriques des modèles évalués sur le jeu de données UAVID [57], lorsque entraînés sur 3 niveaux de sécurité, allant de peu dangereux à sécuritaire.

		Entraînement		**Validation**		**Test**	
Architecture	**Backbone**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**	**MAE**	**Mean IoU**
DeepLabV3+	InceptionV3	0.5725	0.7386	0.6900	0.6913	0.6910	0.6748
DeepLabV3+	MobileNetV2	0.7103	0.6911	0.7539	0.6667	0.7391	0.6687
DeepLabV3+	ResNet101	0.6003	0.7341	0.7025	0.6921	0.6636	0.6954
LinkNet	InceptionV3	0.5338	0.7569	0.6234	0.7188	0.6249	0.7096
LinkNet	MobileNetV2	0.7312	0.6842	0.7718	0.6641	0.7287	0.6684
LinkNet	ResNet101	0.6001	0.7369	0.6686	0.7031	0.6242	0.7084
Unet	InceptionV3	0.5037	0.7730	0.6072	0.7307	0.5720	0.7285
Unet	MobileNetV2	0.7236	0.6835	0.7496	0.6697	0.7646	0.6532
Unet	ResNet101	0.5606	0.7487	0.6457	0.7122	0.6651	0.6948
SLZNet-Seg	-	0.5718	0.7426	0.6078	0.7225	0.6497	0.6956

TABLE 4.6 – Métriques des modèles évalués sur le jeu de données UAVID [57], lorsque entraînés sur 5 niveaux de sécurité, allant de peu dangereux à sécuritaire.

En observant les métriques précédentes, on s'aperçoit que le jeu de données UAVID

présente, pour tous les modèles, une tendance très similaire à celle présentée dans le jeu de données VALID, même si ce dernier est une simulation, non pas des captures de la vie réelle. Cependant, le nombre de classes, la résolution des prises de vues, ainsi que l'altitude simulée pour ces captures représentent des facteurs qui sont très différents du jeu de données ICG, permettant une forte ressemblance avec le jeu de données VALID.

Suivant cette tendance, lorsqu'on observe la segmentation sur 3 niveaux de sûreté, on remarque que, pour l'architecture DeepLabV3+, le backbone InceptionV3 prend la première place, présentant un score MAE de 0.3922 et un mean IoU de 0.6583 pour ses données de tests, comparativement aux deux autres backbones se situant en haut du 0.4 pour le MAE, ainsi qu'en dessous du 0.65 pour le mean IoU. Cette tendance se reporte de même pour le UNet, présentant sa version supérieure avec le backbone InceptionV3, avec des scores de MAE de 0.3182 et un mean IoU de 0.72 sur ses données de test, soit des scores nettement supérieurs à ceux obtenus avec l'architecture DeepLabV3+. Quant à l'architecture LinkNet, celle-ci montre un meilleur score avec son backbone ResNet101, offrant un MAE de 0.2980 et un mean IoU de 0.7285, soit les meilleurs pour tous les modèles entraînés pour la segmentation à 3 niveaux de sécurité sur le jeu de données UAVID. Pour le SLZNet-Seg, celui-ci semble montrer plus de difficulté, n'affichant un score de MAE que de 0.3660 et un mean IoU de 0.6781, soit dans la moyenne, mais pas au maximum des métriques.

Pour ce qui est de la segmentation à 5 niveaux de sécurité, InceptionV3 demeure le backbone avec le plus haut score pour les architectures LinkNet et UNet, alors que le backbone ResNet101 prévaut pour l'architecture DeepLabV3+ dans les données de test.

4.1.4 Discussion

D'après les résultats présentés dans les tables, on réalise assez vite que le modèle SLZNet-Seg conçu fait partie des modèles qui performent le mieux, émettant des valeurs pour le MAE qui sont faibles, par rapport à des valeurs élevées au niveau de l'intersection sur union moyenne. comparativement aux autres modèles, sauf dans le cas du jeu de données UAVID, lequel demeure un défi en soi, vu le nombre de classes proportionnellement au nombre de pixels dans une image. De plus, ce jeu de données comprend des prises de vues d'une haute altitude, représentant l'un des jeux de données les plus difficiles à segmenter.

Cette différence peut s'expliquer par le fait que le modèle SLZNet-Seg a initialement été conçu en se basant sur les points forts, ainsi que les points faibles de chacun des modèles précédents. Dans un premier temps, sur des jeux de données présentant de plus

grandes captures en un plus faible nombres d'échantillons, soit le jeu de données ICG, le SLZNet-Seg se démarque des comparables avec une différence de près de 0.02.

Bien qu'en observant les valeurs des métriques sur les trois jeux de données la performance des modèles semble déjà très bonne, les matrices de confusion nous laissent perplexes. La diagonale présente des résultats qui sont relativement très bons, mais le nombre de pixels qui sont classés dans les catégories sécuritaire et peu dangereux, alors que ceux-ci se trouvent dans les catégories peu dangereux et dangereux présentent un réel danger pour le UAV dans le cas où une manoeuvre doit être effectuée rapidement. C'est cette raison qui nous a poussé à concevoir et mettre en oeuvre notre nouvelle fonction de perte, en plus de faire utilisation de la régression ordinale. De plus, il était nécessaire de se baser rapidement sur une métrique qui ne nécessite pas le calcul de la matrice de confusion au complet afin d'ajuster l'apprentissage des modèles rapidement, d'où la raison de la conception de notre métrique *Safety Factor*. La section suivante présente la sommation des résultats obtenus lors d'un passage d'une méthode de segmentation en utilisant la classification et la crossentropie, vers une méthode basée sur la régression ordinale et notre fonction de perte le *Safety Loss*.

4.2 Segmentation par régression ordinale

Comme nous l'avons vu dans la section précédente, les résultats qui sont obtenus en considérant le problème comme un simple problème de segmentation en utilisant la classification émettent de fausses classifications, pouvant mettre le UAV dans une position de danger. Afin d'y remédier, nous avons plutôt abordé le problème comme un problème de segmentation ordinale, faisant utilisation d'une nouvelle fonction de perte que nous avons défini, comme étant le *Safety Loss*. Les sous section ci-dessous montrent les résultats obtenus après l'entraînement, sous les mêmes contraintes et conditions que les modèles par classification ont été entraînés.

4.2.1 Prises de vues

La figure 4.6 montre une prise de vues obtenu à partir du jeu de données ICG sur les architectures UNet, LinkNet, DeepLabV3+ basées sur un backbone InceptionV3, ainsi que l'architecture SLZNet-Ord, soit l'architecture SLZNet-Seg pour laquelle nous avons modifié la sortie pour une couche de convolution à $n-1$ classifieurs pour une classification à n classes. En plus de ces modifications, les modèles ont été entraînés en utilisant la fonction de perte *Safety Loss* défini précédemment.

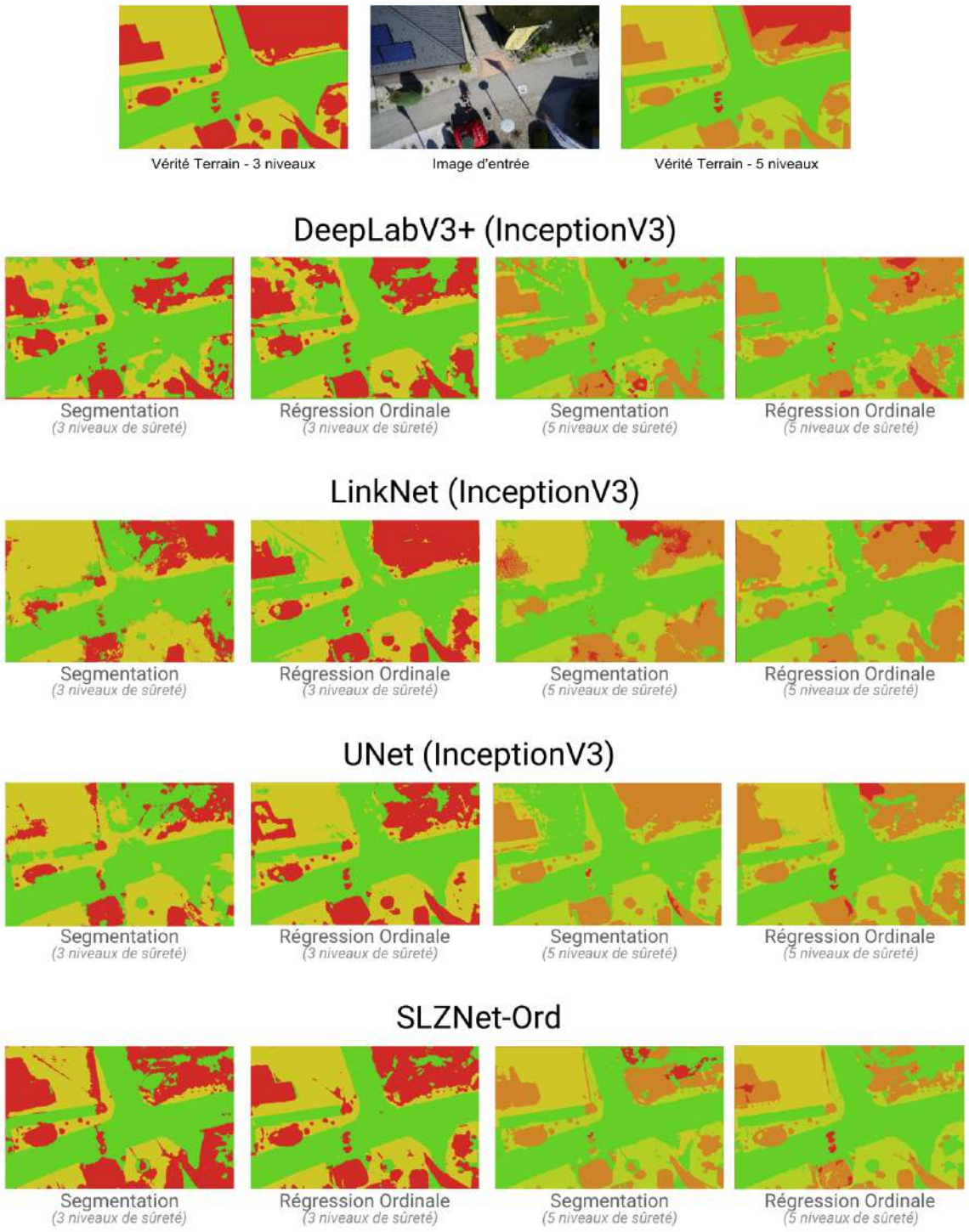


FIGURE 4.6 – Résultats de segmentation d'une capture du UAV, tirée du dataset ICG [57], sur les architectures DeepLabV3+, UNet et LinkNet à partir du backbone InceptionV3, en plus du modèle SLZNet-Ord, lequel est bâtit à partir de l'architecture SLZNet-Seg, avec la couche de sortie modifiée afin de supporter la segmentation par régression ordinale.

L'un des problème principal traité par cette nouvelle fonction de perte, ainsi que cette méthode d'entraînement par régression ordinale était la classification des pixels classés comme sécuritaires, alors que ceux-ci présentaient un danger pour le UAV. Dans la figure précédente, on peut s'apercevoir que dans la plupart des cas, les modèles basés sur la segmentation ont accordés plusieurs pixels dans les classes sécuritaires (vert et lime), alors que ceux-ci étaient réellement classés comme dangereux ou semi-dangereux. Pour ces mêmes prédictions, les modèles qui émettent une sortie régression ordinale ont émis des prédictions qui étaient plus près de la réalité, soit une classification de pixels qui étaient dangereux ou semi-dangereux dans des catégories très près de dangereux et semi-dangereux. Par exemple, dans la figure 4.6, les prédictions émises par l'architecture DeepLabV3+ dans les classifications à 5 niveaux, on peut remarquer le coin droit supérieur de l'image présentant des régions sécuritaires en majorité (vert) dans la classification en utilisant la segmentation, alors que la classification par régression ordinale émet des pixels faisant plutôt partie des classes dangereux (rouge) et semi-dangereux (orange).

La même constatation peut être faite avec la même architecture dans la classification à 3 niveaux de sûreté. En observant la même région que précédemment, la classification par segmentation montre beaucoup plus de régions sécuritaires (vert) que de région dangereuses (rouge), alors que l'entièreté de la zone en haut à droite de la vérité terrain est entièrement considéré comme dangereuse (rouge). La classification par sortie à régression ordinale présente quant à elle une majorité de pixels considérés comme dangereux (rouge) dans cette même zone, étant plus près de la vérité terrain.

En observant d'une façon globale, ce constat s'applique pour tous les modèles présents dans la figure, soit que les prédictions émises par les modèles ayant été entraînés avec la fonction de perte Safety Loss, ainsi que modifiés pour supporter la régression ordinale, émettent des prédictions qui sont beaucoup plus sûres que lorsqu'on utilise les modèles de segmentation classique. Par exemple, lors de la segmentation à 5 niveaux de sûreté à l'aide de l'architecture UNet, le toit de la maison, qui était précédemment considéré comme majoritairement sécuritaire (vert) lors des prédictions, se voit maintenant considéré comme ni-sécuritaire, ni-dangereux, soit la bonne classe.

Bien que tout semble être entièrement positif pour cette nouvelle fonction de perte, il est à noter qu'elle comporte cependant quelques limites et contraintes. En effet, cette fonction semble visuellement produire des zones beaucoup plus prudentes que lors de l'utilisation de la fonction de perte crossentropie. Par exemple, la segmentation à 5 niveaux de sûreté produite à partir de l'architecture SLZNet-Ord présente des zones inscrites comme dangereuses (rouge) dans la zone de panneaux solaires, ainsi que dans le bas de l'image vers le centre, alors que la classe réelle sur la vérité terrain est peu

dangereux (orange). Ce même phénomène se retrouve sur la classification du UNet à 5 niveaux de sûreté sur le toit de la maison, voyant une classification peu dangereux (orange) autour des panneaux solaires, alors que cette zone est majoritairement ni-dangereuse, ni-sécuritaire (jaune) sur la vérité terrain. Le modèle entraîné avec la fonction de perte *Safety Loss* a donc tendance à classer les pixels dans des classes de niveaux plus bas (dangereux / peu dangereux), alors que ces pixels sont dans des classes plus sécuritaires. Ce phénomène peut s'expliquer par l'apport du facteur de sécurité (*Safety Factor*) dans la fonction de perte, augmentant le coût d'une classification d'un pixel dangereux comme non-dangereux, et non le contraire, dans la fonction de perte.

Il est donc évident que le facteur de sécurité représente un paramètre important à ajuster lors de l'entraînement, dépendamment du modèle, demandant d'être plus sécuritaire, donc plus réticent à classer dans des classes sécuritaires, pour certain, ou plus permissif pour d'autres.

4.2.2 Matrices de confusion

Les figures précédentes ont montrées visuellement les ressemblances et différences qui font en sorte de démarquer les modèles entraînés avec le *Safety Loss*, des modèles qui ont été entraînés avec la crossentropie selon la problématique de classification simple. Plutôt que de se concentrer sur une image particulière afin de tirer des conclusions, les matrices de confusion suivantes, tirées du sous-ensemble test du jeu de données ICG, montrent la distribution des prédictions pour chaque architecture (SLZNet-Ord, DeepLabV3+, UNet et LinkNet) avec le backbone *InceptionV3*, si applicable, selon les deux méthodes d'entraînement (classification, régression ordinale avec *Safety Loss*).

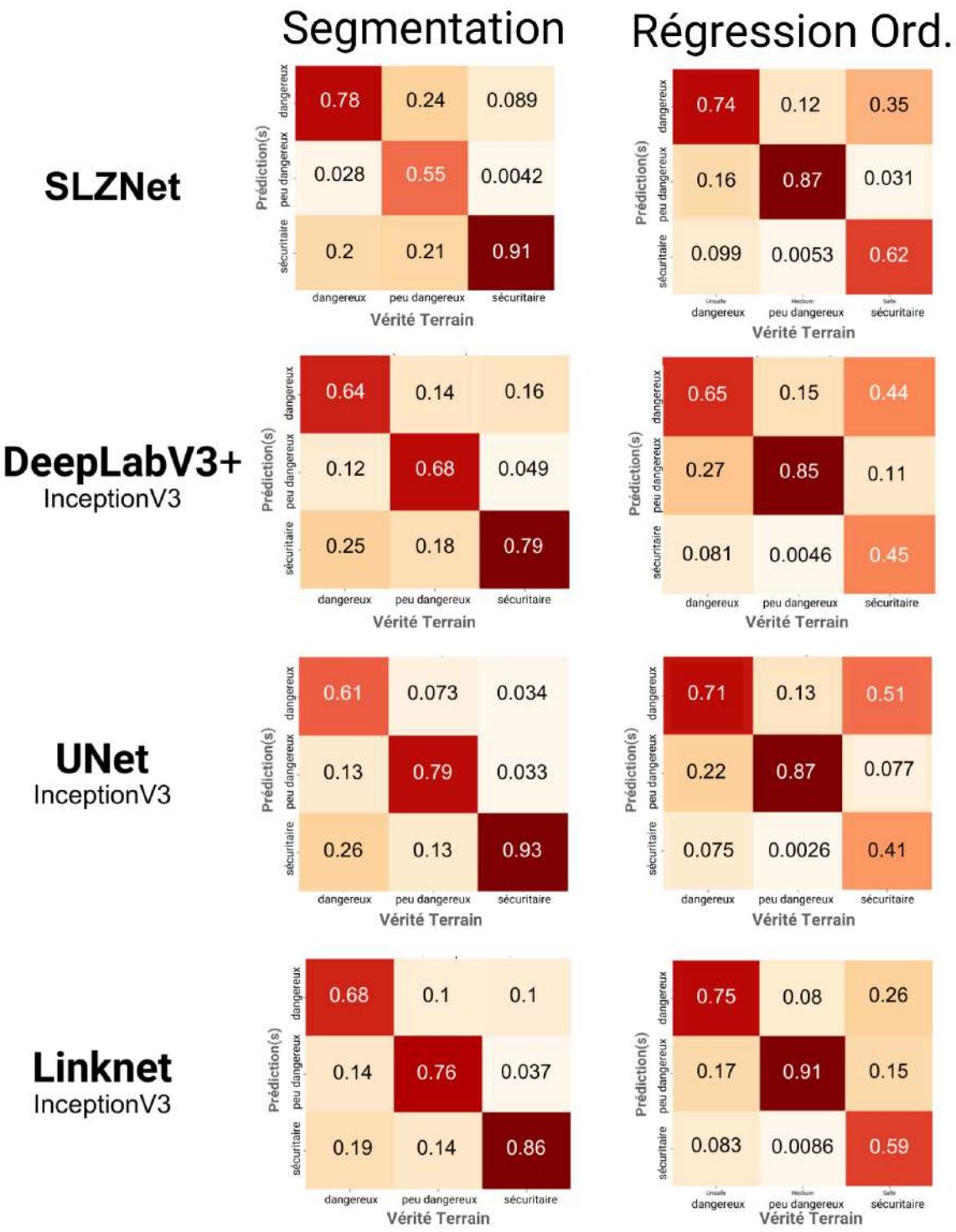


FIGURE 4.7 – Comparaison des matrices de confusion à 3 niveaux, tirées d'un sous-échantillon test du jeu de données UAVID [57], entre les modèles entraînés selon la problématique de classification et leur comparable entraîné à l'aide de la régression ordinale et du *Safety Loss*

.

La figure 4.7 montre les matrices de confusion de chaque modèle. Dans notre cas, on s'intéresse particulièrement aux zones en bas à gauche de la matrice, soit les zones dans lesquelles il se devait d'être prédit, pour chaque pixel, une classe dangereuse, ou peu dangereux, mais que le modèle a prédit une classe peu dangereux, ou sécuritaire, soit la section sous la diagonale de la matrice. Dans les quatres cas présentés dans la figure, le pourcentage de pixels qui sont mal classés, apportant aussi une augmentation du danger sur le UAV, s'est vu diminué. Cette diminution se voit dans le SLZNet, voyant sa variation passer de 20% des pixels à 9.9%, dans l'architecture DeepLabV3+ de 25% à 8.1%, dans UNet de 26% à 7.5% et dans le Linknet de 19% à 8.3%. Dans tous les cas, tous les pixels dangereux qui avaient été classés comme sécuritaires ont vu leur nombre diminuer, promettant une map de segmentation beaucoup plus sécuritaire que lorsqu'on utilise la classification comme problématique.

Dans l'ensemble, la nouvelle fonction de perte apporte de nombreux points positifs faisant diminuer la portion des classifications qui pourraient porter danger au UAV. Cependant, dans un niveau de compréhension moins théorique et plus pratique, on peut s'apercevoir qu'elle le fait au détriment des classifications des pixels sécuritaires qui étaient bien classés. En effet, en ajoutant le facteur de sécurité (*Safety Factor*) à la fonction de perte, les modèles ont tendance à devenir trop stricts, diminuant la proportion de pixels classés comme sécuritaires. Les modèles semblent avoir plus peur de classer les pixels comme sécuritaires ou peu dangereux, faisant augmenter les proportions de pixels dans les catégories peu dangereux et dangereux. On peut notamment le distinguer, voyant la dernière cellule de la diagonale des modèles SLZNet, DeepLabV3+, UNet et Linknet passer de 91% à 62%, 79% à 45%, 93% à 41% et 86% à 59% respectivement.

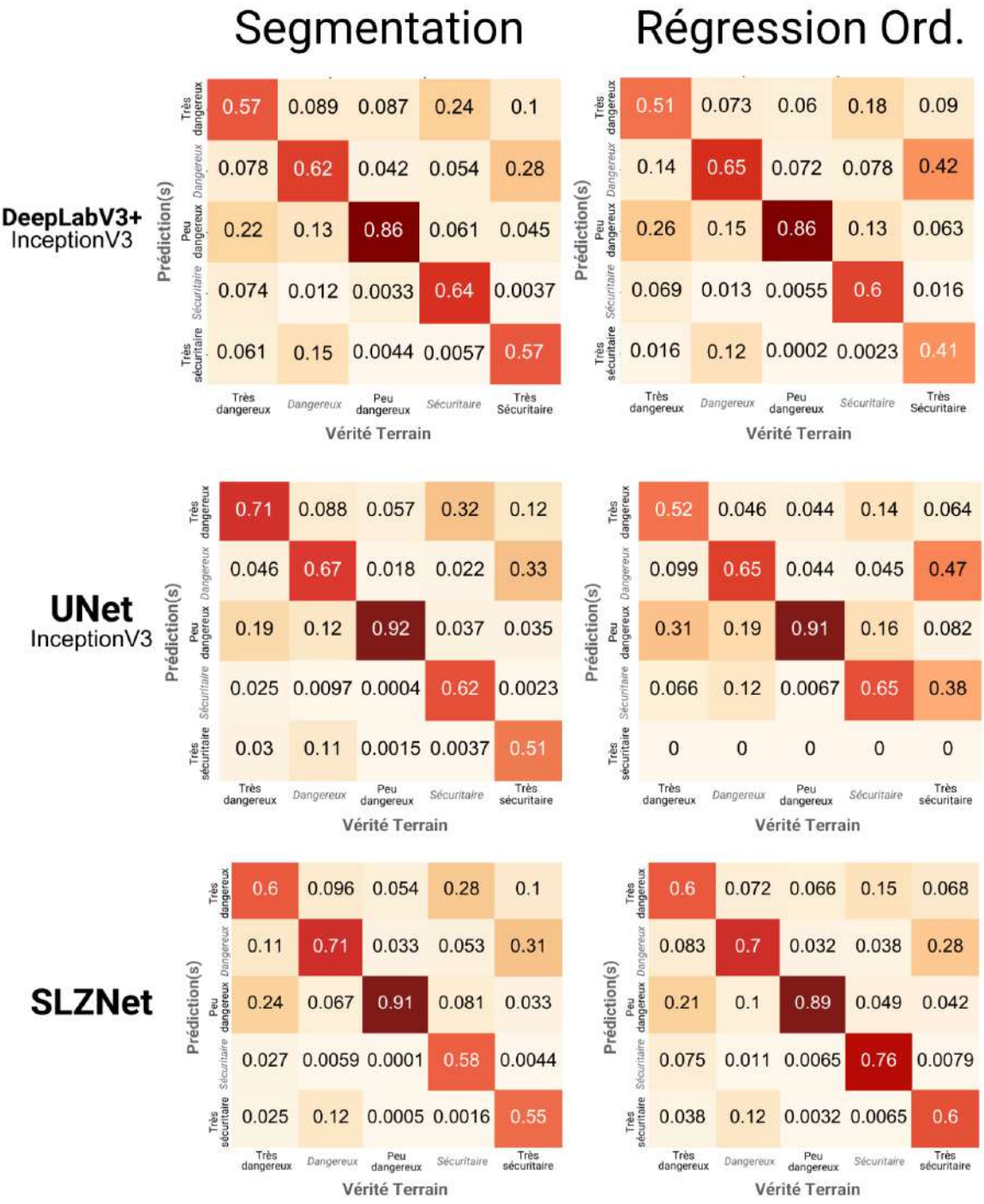


FIGURE 4.8 – Comparaison des matrices de confusion à 5 niveaux, tirées d'un sous-échantillon test du jeu de données UAVID [57], entre les modèles entraînés selon la problématique de classification et leur comparable entraîné à l'aide de la régression ordinale et du *Safety Loss*

.

De l'autre côté, au niveau de la segmentation des zones de sécurité à 5 niveaux, la comparaison semble porter les mêmes conclusions que la solution à 3 niveaux. Dans tous les cas, il y a une diminution du nombre de pixels classés dans des classes supérieures (i.e. peu dangereux, sécuritaire et très sécuritaire), alors qu'ils ne le sont pas, augmentant ainsi la sûreté des prédictions des modèles. Bien qu'il y ait une plus petite quantité de pixels mal classés qui présentent des risques pour le UAV, les modèles donnent des prédictions qui sont beaucoup plus prudentes. Il est possible de remarquer ce fait en observant la matrice de confusion de l'architecture UNet. Celle-ci n'a donné aucune prédiction dans la classe *très sécuritaire*, ce qui montre la réticence du modèle à émettre des prédictions de niveaux supérieurs, de peur de faire augmenter la portion *Safety Factor* de la fonction de perte.

4.2.3 Métriques

Les tables 4.9, 4.10, 4.7, 4.8, 4.11, 4.12 montrent les résultats obtenus sur les jeux de données ICG, VALID et UAVID en utilisant les métriques de l'erreur absolue moyenne (MAE) et de l'intersection sur union moyenne (Mean IoU) lors de l'entraînement, la validation et le test des modèles. Chaque table comporte les architectures testées précédemment avec leurs scores initiaux reportées des tables précédentes, puis leur comparatif par rapport au même modèle entraîné à l'aide du *Safety Loss*.

Le jeu de données UAVID présente des images de la vie réelle dans un milieu urbain, étant l'un des plus difficiles à utiliser, comme les caractéristiques sont moins prononcées que le jeu de données VALID, par exemple, dans lequel la définition de l'image est beaucoup plus claire, selon son point de vue d'une simulation. Pour cette raison, les résultats du jeu de données UAVID, présentés aux tables 4.7 et 4.8, semble moins chaleureux que ceux qui sont présentés dans les tables suivantes.

		Entraînement						Validation						Test					
		Classification			Safety Loss			Classification			Safety Loss			Classification			Safety Loss		
Architecture	Backbone	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score
DeepLabV3+	InceptionV3	0.314	0.724	0.917	0.304	0.802	0.917	0.389	0.661	0.908	0.364	0.667	0.909	0.392	0.658	0.914	0.370	0.697	0.915
DeepLabV3+	MobileNetV2	0.440	0.600	0.872	0.358	0.622	1.000	0.433	0.605	0.886	0.356	0.663	1.000	0.483	0.565	0.873	0.467	0.600	1.000
DeepLabV3+	ResNet101	0.378	0.666	0.912	0.322	0.694	1.000	0.387	0.655	0.917	0.292	0.727	1.000	0.400	0.640	0.918	0.338	0.645	1.000
LinkNet	InceptionV3	0.266	0.761	0.959	0.234	0.832	0.997	0.326	0.711	0.947	0.253	0.778	0.997	0.336	0.705	0.953	0.283	0.777	0.997
LinkNet	MobileNetV2	0.329	0.708	0.904	0.299	0.764	0.978	0.378	0.662	0.900	0.364	0.709	0.979	0.412	0.631	0.912	0.336	0.695	0.974
LinkNet	ResNet101	0.276	0.756	0.936	0.181	0.816	1.000	0.338	0.701	0.933	0.337	0.743	1.000	0.298	0.729	0.955	0.217	0.731	1.000
UNet	InceptionV3	0.291	0.749	0.931	0.206	0.751	0.931	0.340	0.706	0.930	0.244	0.780	0.924	0.318	0.720	0.943	0.241	0.773	0.935
UNet	MobileNetV2	0.363	0.680	0.887	0.329	0.690	0.948	0.397	0.650	0.889	0.370	0.685	0.935	0.428	0.625	0.879	0.388	0.640	0.929
UNet	ResNet101	0.310	0.734	0.921	0.222	0.745	0.937	0.366	0.684	0.919	0.307	0.696	0.916	0.347	0.697	0.936	0.302	0.719	0.926
SLZNet-Ord	-	0.336	0.705	0.911	0.246	0.709	0.948	0.350	0.688	0.915	0.347	0.767	0.924	0.366	0.678	0.921	0.294	0.683	0.932

TABLE 4.7 – Métriques des modèles entraînés sous la problématique de classification et avec la fonction de perte *Safety Loss* avec le jeu de données UAVID à 3 niveaux de sécurité.

		Entrainement						Validation						Test					
		Classification			Safety Loss			Classification			Safety Loss			Classification			Safety Loss		
Architecture	**Backbone**	**MAE**	**Mean IoU**	**Mean Safety Score**	**MAE**	**Mean IoU**	**Mean Safety Score**	**MAE**	**Mean IoU**	**Mean Safety Score**	**MAE**	**Mean IoU**	**Mean Safety Score**	**MAE**	**Mean IoU**	**Mean Safety Score**	**MAE**	**Mean IoU**	**Mean Safety Score**
DeepLabV3+	InceptionV3	0.572	0.739	0.916	0.532	0.828	0.923	0.690	0.691	0.907	0.701	0.758	0.913	0.691	0.675	0.915	0.662	0.749	0.919
DeepLabV3+	MobileNetV2	0.710	0.691	0.881	0.632	0.704	0.910	0.754	0.667	0.895	0.699	0.723	0.918	0.739	0.669	0.890	0.642	0.691	0.905
DeepLabV3+	ResNet101	0.600	0.734	0.904	0.521	0.833	0.910	0.703	0.692	0.905	0.659	0.733	0.918	0.664	0.695	0.911	0.609	0.756	0.905
LinkNet	InceptionV3	0.534	0.757	0.945	0.473	0.835	0.999	0.623	0.719	0.940	0.589	0.765	0.998	0.625	0.710	0.948	0.556	0.749	0.999
LinkNet	MobileNetV2	0.731	0.684	0.896	0.724	0.735	0.866	0.772	0.664	0.900	0.735	0.674	0.874	0.729	0.668	0.915	0.707	0.705	0.862
LinkNet	ResNet101	0.600	0.737	0.920	0.591	0.829	0.978	0.669	0.703	0.922	0.661	0.790	0.980	0.624	0.708	0.946	0.595	0.738	0.978
UNet	InceptionV3	0.504	0.773	0.934	0.491	0.863	0.934	0.607	0.731	0.933	0.604	0.823	0.925	0.572	0.729	0.944	0.569	0.743	0.935
UNet	MobileNetV2	0.724	0.683	0.895	0.665	0.684	0.923	0.750	0.670	0.901	0.744	0.750	0.912	0.765	0.653	0.894	0.763	0.753	0.904
UNet	ResNet101	0.561	0.749	0.917	0.521	0.829	0.923	0.646	0.712	0.919	0.613	0.776	0.896	0.665	0.695	0.933	0.664	0.755	0.896
SLZNet-Ord	-	0.572	0.743	0.919	0.326	0.828	0.947	0.608	0.722	0.924	0.565	0.735	0.923	0.650	0.696	0.933	0.570	0.637	0.932

TABLE 4.8 – Métriques des modèles entraînés sous la problématique de classification et avec la fonction de perte *Safety Loss* avec le jeu de données UAVID à 5 niveaux de sécurité.

Dans l'ensemble, les modèles qui utilisent l'architecture DeepLabV3+ ont vu leur performance augmenter tant au niveau de la diminution du MAE qu'à l'augmentation du Mean IoU et du Safety Score. Le même phénomène est répété sur les architectures UNet et LinkNet, en plus de l'architecture SLZNet-Ord. On peut voir une augmentation majeure du Safety Score moyen, d'où l'apport du facteur de sécurité dans le *Safety Loss* pour la plupart des modèles. Cependant, l'intersection sur union moyenne démontre une augmentation un peu plus lente, ce qui peut être justifié par le même phénomène qu'observé dans les matrices de confusion de la section précédente. Dans le cas où il y a augmentation du Safety Score, soit la diminution du nombre de pixels qui sont classés comme sécuritaire ou peu dangereux, alors qu'ils représentent un vrai danger, on voit aussi apparaître un facteur faisait en sorte que les modèles sont un peu plus conservateurs dans leurs classifications, classant très peu de pixels comme sécuritaires, d'où la faible montée de l'intersection sur union moyenne.

Alors que le jeu de données UAVID présente des scènes urbaines avec des proportions inégales entre les classes, par exemple les bâtiments qui sont gigantesques versus les humains qui représentent moins d'une dizaine de pixels, le jeu de données ICG est aussi basé sur de réelles prises de vues, mais à des altitudes beaucoup plus faibles, présentant ainsi une plus haute définition d'images, ainsi qu'une meilleure représentation des classes dans l'ensemble pour un UAV.

		Entrainement						Validation						Test					
		Classification			Safety Loss			Classification			Safety Loss			Classification			Safety Loss		
Architecture	Backbone	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score
DeepLabV3+	InceptionV3	0.322	0.718	0.892	0.231	0.793	0.879	0.430	0.668	0.884	0.346	0.673	0.885	0.248	0.750	0.902	0.243	0.809	0.883
DeepLabV3+	MobileNetV2	0.368	0.686	0.869	0.312	0.691	0.989	0.424	0.668	0.875	0.360	0.721	0.996	0.258	0.733	0.854	0.215	0.784	0.987
DeepLabV3+	ResNet101	0.379	0.680	0.830	0.305	0.711	0.978	0.448	0.659	0.837	0.373	0.698	0.940	0.340	0.671	0.794	0.327	0.704	0.994
LinkNet	InceptionV3	0.255	0.764	0.920	0.246	0.773	0.995	0.321	0.732	0.906	0.270	0.824	0.995	0.228	0.763	0.913	0.165	0.856	0.988
LinkNet	MobileNetV2	0.397	0.649	0.910	0.367	0.657	0.998	0.510	0.590	0.897	0.453	0.646	0.998	0.272	0.722	0.936	0.210	0.796	0.994
LinkNet	ResNet101	0.313	0.726	0.886	0.308	0.735	0.986	0.371	0.705	0.882	0.312	0.712	0.985	0.283	0.726	0.898	0.219	0.774	0.984
UNet	InceptionV3	0.244	0.770	0.911	0.203	0.815	0.923	0.274	0.764	0.886	0.264	0.799	0.922	0.258	0.763	0.886	0.185	0.820	0.948
UNet	MobileNetV2	0.306	0.729	0.903	0.225	0.790	0.882	0.334	0.720	0.886	0.247	0.736	0.891	0.268	0.729	0.906	0.242	0.771	0.894
UNet	ResNet101	0.266	0.761	0.916	0.194	0.800	0.910	0.358	0.706	0.897	0.333	0.740	0.903	0.222	0.773	0.928	0.133	0.782	0.934
SLZNet-Ord	-	0.255	0.765	0.906	0.213	0.819	0.930	0.292	0.757	0.907	0.276	0.769	0.920	0.204	0.792	0.927	0.187	0.860	0.959

TABLE 4.9 – Métriques des modèles entraînés sous la problématique de classification et avec la fonction de perte *Safety Loss* avec le jeu de données ICG à 3 niveaux de sécurité.

		Entrainement						Validation						Test					
		Classification			Safety Loss			Classification			Safety Loss			Classification			Safety Loss		
Architecture	Backbone	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score
DeepLabV3+	InceptionV3	0.695	0.703	0.872	0.654	0.742	0.860	0.849	0.669	0.879	0.815	0.719	0.872	0.543	0.727	0.871	0.521	0.797	0.838
DeepLabV3+	MobileNetV2	0.578	0.737	0.869	0.502	0.817	0.993	0.721	0.702	0.869	0.664	0.707	0.921	0.363	0.790	0.880	0.285	0.835	0.956
DeepLabV3+	ResNet101	0.660	0.704	0.828	0.639	0.742	0.784	0.817	0.670	0.830	0.727	0.680	0.768	0.550	0.712	0.791	0.513	0.762	0.706
LinkNet	InceptionV3	0.463	0.780	0.912	0.433	0.856	0.990	0.597	0.731	0.892	0.591	0.820	0.986	0.540	0.732	0.900	0.459	0.758	0.990
LinkNet	MobileNetV2	0.589	0.709	0.891	0.571	0.741	0.966	0.694	0.685	0.880	0.687	0.779	0.955	0.532	0.701	0.902	0.480	0.742	0.964
LinkNet	ResNet101	0.540	0.750	0.873	0.468	0.807	0.987	0.649	0.714	0.863	0.580	0.724	0.982	0.487	0.749	0.877	0.475	0.820	0.989
UNet	InceptionV3	0.371	0.812	0.928	0.296	0.827	0.920	0.457	0.790	0.903	0.372	0.833	0.918	0.325	0.817	0.919	0.246	0.821	0.942
UNet	MobileNetV2	0.500	0.760	0.899	0.426	0.809	0.892	0.587	0.737	0.875	0.529	0.746	0.883	0.459	0.754	0.898	0.368	0.767	0.888
UNet	ResNet101	0.510	0.757	0.903	0.495	0.804	0.903	0.632	0.719	0.890	0.560	0.816	0.893	0.386	0.785	0.913	0.322	0.810	0.922
SLZNet-Ord	-	0.480	0.768	0.901	0.448	0.854	0.909	0.631	0.739	0.900	0.594	0.781	0.901	0.297	0.832	0.924	0.278	0.930	0.935

TABLE 4.10 – Métriques des modèles entraînés sous la problématique de classification et avec la fonction de perte *Safety Loss* avec le jeu de données ICG à 5 niveaux de sécurité.

Tel que dans le cas du jeu de données UAVID, les modèles présentent une augmentation de leur performance sur tous les métriques en général. Sommairement, les modèles basés sur l'architecture UNet et le modèle SLZNet-Ord présentent des scores beaucoup plus petits au niveau du MAE sur les données de test que les deux autres architectures, et ce dans les deux cas, soit à 3 et 5 niveaux de sécurité. Cependant, en comparant les échantillons à 3 et à 5 niveaux de sécurité, on remarque que l'intervalle de valeurs pour le MAE sont beaucoup plus bas lorsqu'on fait une classification à 3 niveaux, dans l'intervalle [0.133, 0.327], que dans la classification à 5 niveaux avec l'intervalle [0.246, 0.521]. En effet, l'intervalle a presque doublé entre les deux niveaux de classification, montrant ainsi l'élévation de la difficulté entre une classification à 3 niveaux versus celle à 5 niveaux, et ce peu importe le jeu de données ou le modèle évalué. Ainsi, même pour ce jeu de données, la fonction de perte *Safety Loss* apporte de nouveau une augmentation de la performance moyenne des modèles, en plus d'augmenter la sécurité du UAV en améliorant d'abord les classifications qui pourraient porter un direct danger au UAV.

		Entrainement						Validation						Test					
		Classification			Safety Loss			Classification			Safety Loss			Classification			Safety Loss		
Architecture	Backbone	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score
DeepLabV3+	InceptionV3	0.253	0.689	0.911	0.163	0.724	0.980	0.232	0.713	0.921	0.199	0.718	0.983	0.257	0.685	0.913	0.184	0.760	0.973
DeepLabV3+	MobileNetV2	0.253	0.689	0.911	0.229	0.747	0.934	0.232	0.713	0.921	0.228	0.772	0.944	0.257	0.685	0.913	0.199	0.724	0.931
DeepLabV3+	ResNet101	0.253	0.689	0.911	0.230	0.719	0.928	0.232	0.713	0.921	0.158	0.756	0.936	0.257	0.685	0.913	0.164	0.724	0.922
LinkNet	InceptionV3	0.156	0.790	0.955	0.096	0.803	0.955	0.154	0.792	0.958	0.140	0.832	0.959	0.166	0.778	0.949	0.130	0.798	0.958
LinkNet	MobileNetV2	0.164	0.789	0.965	0.071	0.820	0.917	0.158	0.797	0.970	0.111	0.811	0.968	0.174	0.778	0.961	0.126	0.835	0.972
LinkNet	ResNet101	0.165	0.781	0.957	0.148	0.821	0.961	0.154	0.794	0.965	0.056	0.884	0.938	0.172	0.773	0.952	0.160	0.837	0.968
UNet	InceptionV3	0.126	0.826	0.967	0.066	0.925	0.990	0.123	0.829	0.971	0.086	0.873	0.992	0.138	0.811	0.959	0.126	0.860	0.982
UNet	MobileNetV2	0.168	0.784	0.957	0.074	0.807	0.992	0.174	0.779	0.951	0.102	0.814	0.993	0.180	0.771	0.951	0.105	0.858	0.992
UNet	ResNet101	0.178	0.779	0.951	0.094	0.809	0.979	0.170	0.793	0.956	0.070	0.843	0.979	0.184	0.774	0.951	0.175	0.828	0.970
SLZNet-Ord	-	0.169	0.776	0.935	0.110	0.846	0.991	0.159	0.787	0.942	0.108	0.847	0.992	0.180	0.762	0.926	0.120	0.834	0.990

TABLE 4.11 – Métriques des modèles entraînés sous la problématique de classification et avec la fonction de perte *Safety Loss* avec le jeu de données VALID à 3 niveaux de sécurité.

		Entrainement						Validation						Test					
		Classification			Safety Loss			Classification			Safety Loss			Classification			Safety Loss		
Architecture	Backbone	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score	MAE	Mean IoU	Mean Safety Score
DeepLabV3+	InceptionV3	0.658	0.518	0.921	0.593	0.525	0.981	0.627	0.539	0.929	0.612	0.588	0.982	0.687	0.508	0.922	0.662	0.569	0.974
DeepLabV3+	MobileNetV2	0.651	0.520	0.924	0.638	0.609	0.938	0.618	0.543	0.933	0.564	0.640	0.967	0.681	0.509	0.925	0.650	0.561	0.947
DeepLabV3+	ResNet101	0.505	0.696	0.942	0.466	0.763	0.988	0.485	0.704	0.947	0.402	0.745	0.951	0.512	0.696	0.942	0.418	0.748	0.923
LinkNet	InceptionV3	0.311	0.800	0.950	0.244	0.851	0.967	0.305	0.806	0.954	0.211	0.846	0.970	0.328	0.794	0.944	0.254	0.847	0.982
LinkNet	MobileNetV2	0.304	0.797	0.961	0.246	0.853	0.913	0.304	0.795	0.964	0.238	0.868	0.938	0.330	0.786	0.956	0.302	0.842	0.912
LinkNet	ResNet101	0.312	0.802	0.954	0.234	0.808	0.955	0.304	0.807	0.961	0.225	0.826	0.993	0.325	0.798	0.951	0.231	0.819	0.937
UNet	InceptionV3	0.230	0.845	0.971	0.192	0.850	0.987	0.228	0.846	0.974	0.145	0.876	0.990	0.248	0.837	0.964	0.157	0.909	0.978
UNet	MobileNetV2	0.287	0.809	0.951	0.194	0.820	0.987	0.287	0.806	0.948	0.280	0.839	0.990	0.319	0.795	0.943	0.295	0.797	0.982
UNet	ResNet101	0.311	0.803	0.954	0.240	0.833	0.979	0.308	0.804	0.957	0.271	0.871	0.978	0.344	0.788	0.950	0.309	0.847	0.969
SLZNet-Ord	-	0.320	0.792	0.947	0.089	0.934	0.984	0.302	0.799	0.953	0.088	0.935	0.986	0.337	0.785	0.940	0.119	0.917	0.977

TABLE 4.12 – Métriques des modèles entraînés sous la problématique de classification et avec la fonction de perte *Safety Loss* avec le jeu de données VALID à 5 niveaux de sécurité.

Au niveau du dernier jeu de données, celui-ci est plutôt un jeu de données qui est simulé, imposant un faible nombre de classes distinctes présentes dans une même image, ainsi qu'une altitude de vol se situant plus près du sol que dans le cas des deux autres jeux de données. Ainsi, on élimine toutes les contraintes visuelles difficiles imposées à un UAV, telles que l'éclairage, les reflets et bien plus. Pour cette raison, on remarque rapidement que, autant au niveau de la classification à 3 niveaux de sécurité, qu'au niveau de la classification à 5 niveaux de sécurité, la limite inférieure de l'intervalle de score MAE tourne autour du 0.120 avec des valeurs de 0.105 pour le 3 niveaux de sécurité et 0.119 pour le 5 niveaux de sécurité.

Somme toute, les modèles ont vu leur performance suivre la tendance des deux autres jeux de données et augmenter, comparativement à lorsqu'on a entraîné les modèles en suivant une problématique de classification, plutôt que la régression ordinale. Cependant, une différence a été trouvée au niveau pratique lors de l'entraînement avec le jeu de données VALID, tant avec le problème de classification que régression ordinale. En effet, les modèles basés sur l'architecture DeepLabV3+, peu importe le backbone, se sont tous vu obtenir des scores MAE et mean IoU très semblables. Bien qu'il puisse sembler d'une erreur à première vue, de légères différences au niveau du Safety Score,

ainsi que suite à l'utilisation du *Safety Loss* ont permis d'observer ce phénomène comme une limite du modèle dans ce problème précis. De plus, les valeurs obtenues à l'aide du *Safety Loss* ne sont pas très loin de celles obtenues à l'aide de la fonction de crossentropie pour la classification, pointant vers un phénomène dans lequel le modèle aurait atteint son plein potentiel, peu importe le squelette utilisé.

4.2.4 Discussion

Après avoir testé notre nouvelle fonction de perte *Safety Loss* et analysé les résultats, il est possible de conclure que celle-ci bénéficie à l'entraînement de modèles dans le domaine de la segmentation sémantique, en particulier dans le domaine de la détection des zones d'atterrissage sécuritaires. Peu importe le modèle où la fonction de perte était placée, comparativement à la crossentropie dans le cas de la segmentation par classification classique, les performances étaient améliorées, tant au niveau des métriques standards (MAE, Mean IoU), qu'au niveau de notre nouvelle métrique le *Mean Safety Score*, tiré de la conception initiale de notre fonction de perte.

Cependant, la fonction de perte semble augmenter les performances au détriment de certaines classifications. En se basant sur les matrices de confusion, on remarque que les modèles ont tendance à être plus prudents dans leurs classifications, faisant en sorte que les pixels sécuritaires sont souvent sous-estimés et classés comme peu dangereux, voire même dangereux. C'est pourquoi, nous nous sommes tournés vers des ajustements fins lors de l'entraînement, permettant de contrôler l'entraînement et de restreindre le modèle à travers son apprentissage, tout au long de l'entraînement soit le Meta-Learning.

La section suivante montre les résultats des ajustements obtenus à l'aide du Meta-Learning, une méthode permettant au modèle d'apprendre à apprendre, i.e. chercher des valeurs des hyper-paramètres optimaux lors de l'entraînement.

4.3 Meta-Learning

Tandis que les résultats présentés précédemment montrent une amélioration de la segmentation des zones de sécurité, ainsi que la sûreté des classifications, les matrices de confusion prouvent qu'il y a présence d'une prudence dans les classifications, ayant certes fait réduire le nombre de pixels classés comme sécuritaires alors qu'ils ne l'étaient pas, mais aussi fait augmenter le nombre de pixels classés comme dangereux, alors que ceux-ci ne l'étaient pas du tout.

Afin de remédier à cette situation, nous avons fait usage d'un niveau simple de Meta-Learning, variant le facteur *phi* de notre Safety Loss, de façon à diminuer ou augmenter le facteur de pénalité *Safety Factor*, selon l'allure de la matrice de confusion à la fin de chaque époque. Comme dans les sections précédentes, les résultats sont présentés dans les sections suivantes, suivant des démonstrations au niveau de prises de vues, des matrices de confusion, ainsi que de tables justificatives.

4.3.1 Prises de vue

Parmi les résultats obtenus, les prises de vues tirées des jeux de données demeurent la méthode la plus efficace afin de comprendre les résultats visuellement. La figure 4.9, tout comme pour la section précédente, présente les prédictions des architectures DeepLabV3+, LinkNet, UNet avec l'utilisation d'un backbone InceptionV3, ainsi que l'architecture SLZNet-Ord entraînés à l'aide du Safety Loss, ainsi que par l'apport du Meta-Learning afin de contrôler le Facteur de sûreté dans la fonction de perte tout au long de l'entraînement. Les prédictions sont effectuées pour la classification à 3 niveaux de sûreté, ainsi que 5 niveaux de sûreté. Cette présentation aide donc à cibler les différences et montrer la valeur du Meta-Learning dans l'entraînement des modèles.

Pour ce qui est de la segmentation à 3 niveaux de sûreté, dans l'ensemble, les modèles semblent s'approcher de la vérité terrain avec l'ajout du Meta-Learning. Pour le modèle DeepLabV3+, la segmentation précédente émettait beaucoup de pixels sécuritaires dans des zones qui n'étaient pas considérés comme sécuritaires, la zone forestière dans le coin supérieur droit par exemple, ou encore le toit de la maison à gauche. Le Meta-Learning a fait en sorte d'ammener le modèle à être moins permissif dans la zone forestière, faisant augmenter le nombre de pixels classés comme dangereux, soit plus près de la vérité terrain. Cependant, cette retenu des zones sécuritaires a aussi fait en sorte d'augmenter le nombre de pixels classés comme dangereux dans le coin inférieur droit, alors que ceux-ci sont peu dangereux (jaune) en réalité. On peut apercevoir ce phénomène sur le SLZNet-Ord, par exemple, raffermissant sa classification de la zone forestière, mais émettant la diagonale du toit comme dangereuse. L'architecture LinkNet, quant à elle, présente des caractéristiques semblables à ce qui a été aperçu dans laquelle les humains sont maintenant considérés comme dangereux, ainsi que le coin inférieur de l'image, mais le toit est maintenant considéré comme majoritairement dangereux.

Dans l'ensemble, pour tous les modèles, on aperçoit cet effet de balance entre un modèle trop sécuritaire ou trop permissif avec l'ajout du Meta-Learning. Alors que l'entraînement à l'aide de cette méthode amène le modèle à être plus ou moins prudent d'un côté, il diminue la qualité des prédictions par rapport à la vérité terrain de l'autre.

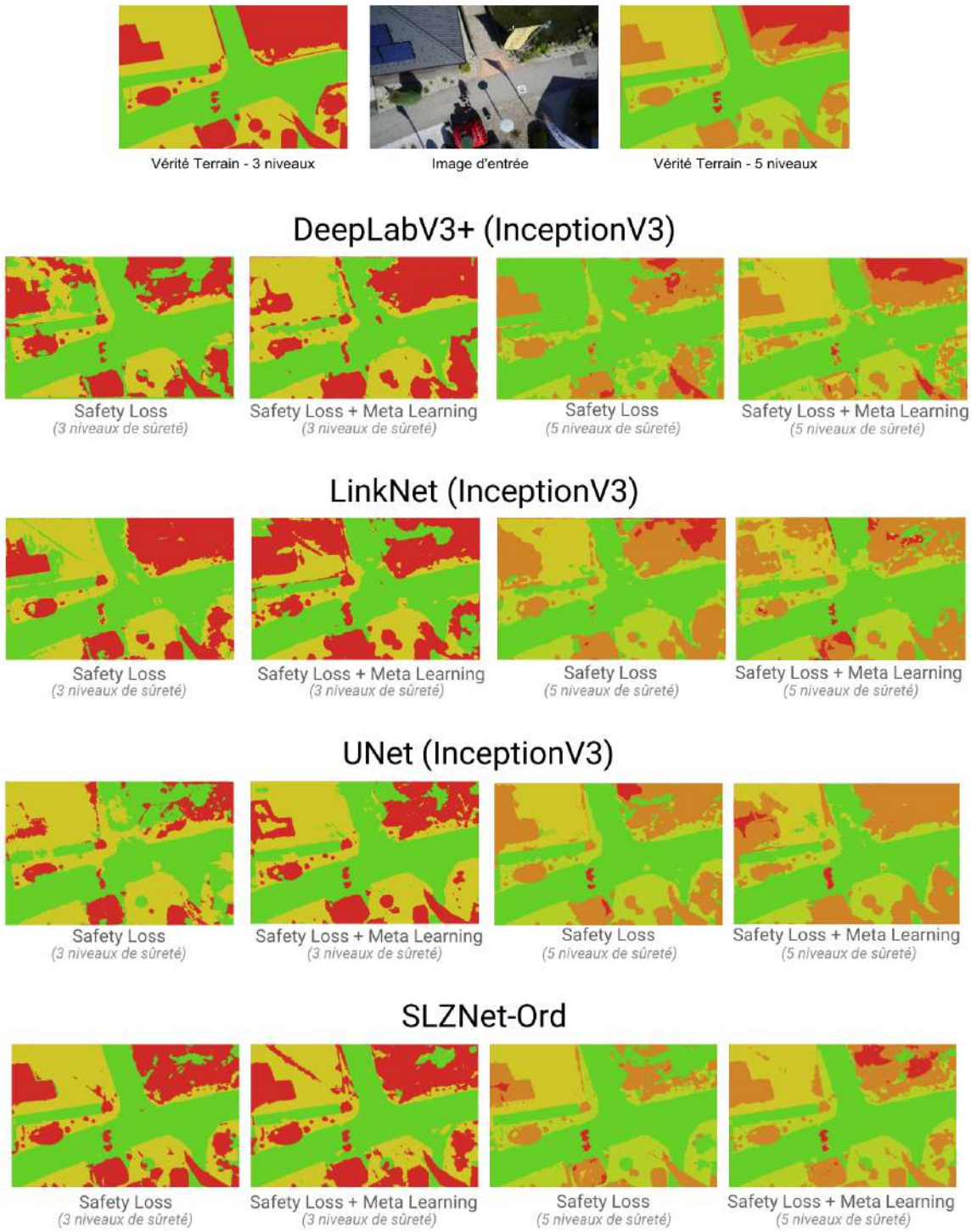


FIGURE 4.9 – Résultats de segmentation d'une capture du UAV, tirée du dataset ICG sur les architectures DeepLabV3+, LinkNet, UNet et SLZNet-Ord, entraîné avec le Safety Loss et ajout du Meta-Learning.

Quant à la segmentation à 5 niveaux de sûreté, les résultats présentés tendent à

supporter le phénomène énuméré précédemment, à savoir que la sûreté des prédictions est ajustée, au détriment de la qualité des prédictions. Au niveau du UNet, dans le coin droit inférieur, l'ajout du Meta-Learning a augmenté le nombre de pixels classés comme peu dangereux (orange), alors que la vérité terrain présente majoritairement des zones peu sécuritaires (vert pâle) et sécuritaires (vert fluorescent). Cette constatation est aussi observée pour le Linknet. Bien que les humains (une classe très importante au niveau de la sécurité) soient mieux définis comme un danger, cet ajustement a fait apparaître des classifications de pixels comme dangereux (rouge) dans le bas de l'image, alors que cette zone est majoritairement peu dangereuse (orange) ou peu sécuritaire (vert lime). Cependant, dans l'ensemble, ces ajustements permettent d'obtenir une meilleure segmentation qui s'aligne plus avec la vérité terrain que la prédiction sans apport du Meta-Learning.

En bref, on remarque que le Meta-Learning permet d'ajuster automatiquement le paramètre de sécurité, faisant tendre le modèle vers la sécurité en étant strict sur les pixels qu'il classifie comme sécuritaire et peu sécuritaire, ou le faisant plutôt tendre vers l'alignement à la vérité terrain, en mettant à risque la sécurité du drone et de son environnement.

4.3.2 Matrices de confusion

Dans un second volet de l'évaluation de cette technique, on s'intéresse aux matrices de confusion, dans le but de prouver les biens faits de l'ajout du Meta Learning par l'amélioration de la diagonale, soit le nombre de pixels qui sont bien classés, ainsi que la diminution des scores des cellules qui se trouvent sous la diagonale, présentant les pixels qui ont mal été classé, et qui présentent un certain danger pour le UAV dans le cas d'un atterrissage d'urgence.

Sur les figures suivantes, on voit la comparaison des trois méthodes, soit la segmentation des niveaux de sécuritaire par une méthode utilisant la classification, par une méthode utilisant la régression ordinale en conjugaison avec notre Safety Loss, en plus de la régression ordinale, utilisant le Safety Loss, avec l'ajout du Meta Learning. La figure 4.10 présentant le modèle SLZNet-Ord, la figure 4.11 l'architecture UNet, puis la figure 4.12 l'architecture DeepLabV3+.

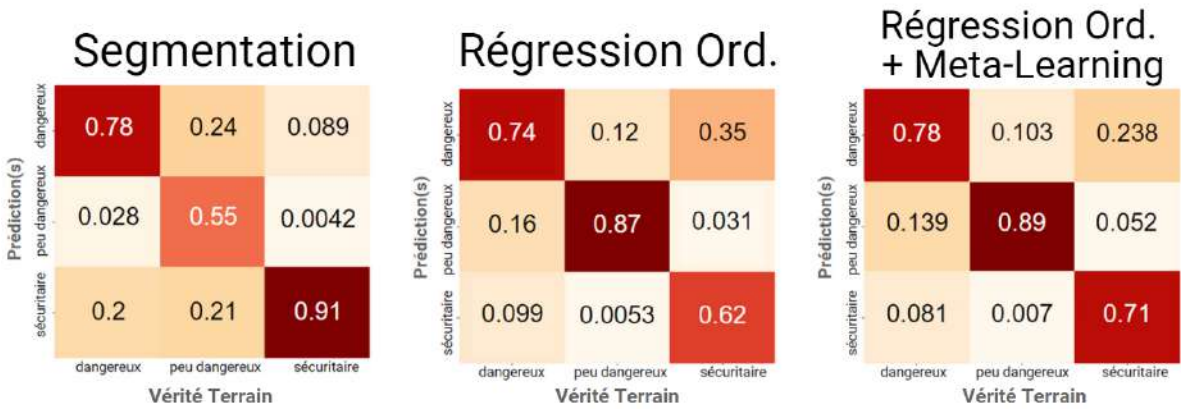


FIGURE 4.10 – Matrice de confusion de l'architecture SLZNet-Ord sur un échantillon test du jeu de données VALID [15]

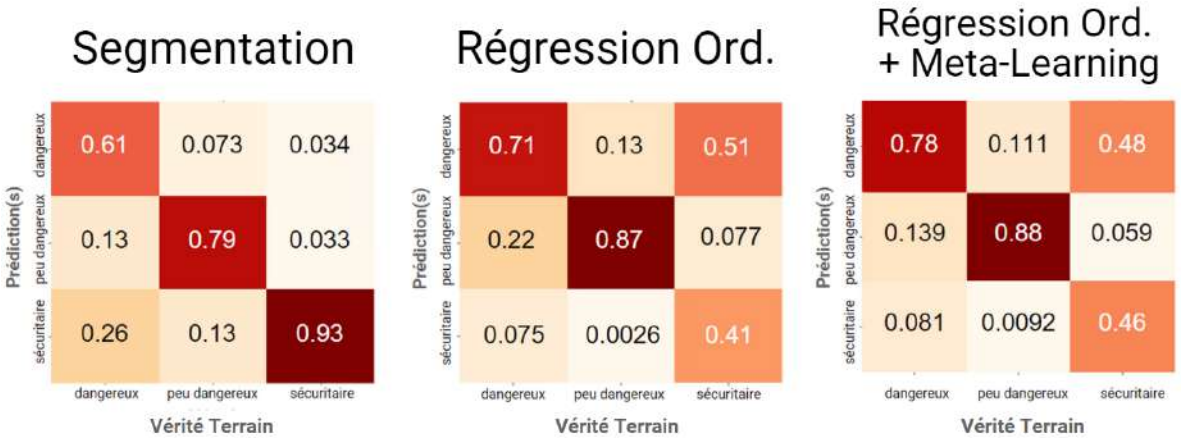


FIGURE 4.11 – Matrice de confusion de l'architecture UNet avec backbone InceptionV3 sur un échantillon test du jeu de données VALID [15]

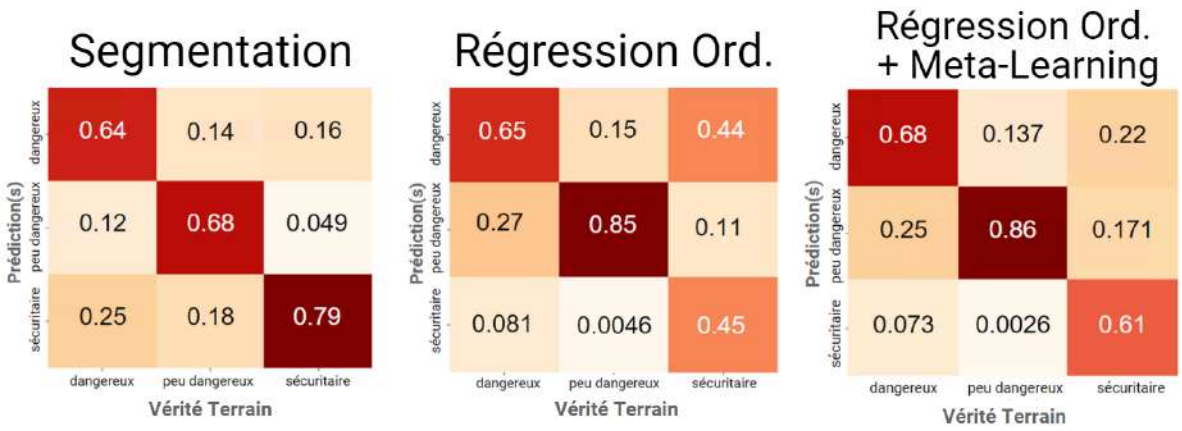


FIGURE 4.12 – Matrice de confusion de l'architecture DeepLabV3+ avec backbone InceptionV3 sur un échantillon test du jeu de données VALID [15]

Dans l'ensemble, les trois comparaisons montrent les mêmes phénomènes. La matrice de confusion de la méthode classique par classification affiche des scores sur la diagonale qui sont très forts pour la classification des pixels sécuritaires, ainsi que peu dangereux. Or, le score de la classe *dangereux* présente de moins bons résultats, voyant de très grands nombres de pixels étant classés dans les classes peu dangereux et sécuritaire, alors qu'ils ne le sont pas, de même que sécuritaire, alors que ceux-ci étaient peu dangereux en réalité.

En faisant utilisation du Safety Loss, soit la matrice de confusion centrale des figures, on voit une amélioration des scores de la diagonale pour les classes dangereux et peu dangereux, affichant de différences en moyenne de 5% pour la classe dangereux et en moyenne 19% pour la classe peu dangereux. Cependant, comme présenté dans la section précédente, la classe sécuritaire s'est vu ses classification justes diminuer, comme les modèles devenaient plus prudents dans leurs classifications, affichant des diminutions moyennes de 38% quant au nombre de pixels sécuritaires réellement classés comme sécuritaires.

Ce qui nous amène donc à la comparaison de la troisième portion de la figure, soit la matrice de confusion produite en ajoutant le Meta Learning au Safety Loss afin de diminuer la prudence du modèle, dans certains cas, ou augmenter celle-ci, dans d'autres cas. Cette méthode s'est avérée effective, car le score des cellules sur la diagonale s'est vu augmenter en moyenne de 4.6% pour la classe dangereux, 1.3% pour la classe peu dangereux et 10% pour la classe sécuritaire.

Bien que le score ne soit toujours pas parfait quant à la classe sécuritaire, les hyperparamètres pourraient toujours être ajustés en diminuant l'ampleur du facteur de sûreté (*Safety Factor*) sur la fonction de perte totale, ayant comme effet de diminuer la pru-

dence du réseau dans ses prédictions, augmentant ainsi le nombre de pixels sécuritaires, mais aussi le nombre de pixels réellement dangereux qui sont classés comme sécuritaires. Cependant, les matrices de confusion finales permettent réellement de produire, avec plus de 70% d'assurance en moyenne, des maps définissant les zones dangereuses ou potentiellement dangereuses pour un UAV, soit le problème traité dans ce mémoire.

4.3.3 Métriques

Le troisième volet à explorer afin de valider la solution proposée se voit être les métriques. Soit, les comparaisons des scores MAE, mean IoU et *Safety Score*, que nous avons défini dans le chapitre précédent, permettant d'évaluer les modèles et de les comparer entre eux, incluant les fonctions de pertes et autres paramètres d'entraînement. Cependant, contrairement aux sections des résultats précédents, vu le nombre de paramètres devant être comparés, nous affichons les résultats dans des tables qui séparent chaque score, permettant d'effectuer une meilleure comparaison des résultats, soit une table pour le MAE (Tables 4.19, 4.16, 4.13), le mean IoU (Tables 4.20, 4.17, 4.14)), ainsi que le *Safety Score* (Tables 4.21, 4.18, 4.15).

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.322	0.231	0.141	0.430	0.346	0.256	0.248	0.243	0.221
DeepLabV3+	MobileNetV2	3	0.368	0.312	0.252	0.424	0.360	0.280	0.258	0.215	0.178
DeepLabV3+	ResNet101	3	0.379	0.305	0.225	0.448	0.373	0.323	0.340	0.327	0.287
LinkNet	InceptionV3	3	0.255	0.246	0.196	0.321	0.270	0.170	0.228	0.165	0.135
LinkNet	MobileNetV2	3	0.397	0.367	0.267	0.510	0.453	0.403	0.272	0.210	0.192
LinkNet	ResNet101	3	0.313	0.308	0.228	0.371	0.312	0.222	0.283	0.219	0.205
UNet	InceptionV3	3	0.244	0.203	0.143	0.274	0.264	0.174	0.258	0.185	0.169
UNet	MobileNetV2	3	0.306	0.225	0.175	0.334	0.247	0.197	0.268	0.242	0.212
UNet	ResNet101	3	0.266	0.194	0.104	0.358	0.333	0.243	0.222	0.133	0.132
SLZNet-Ord	-	3	0.255	0.213	0.113	0.292	0.276	0.206	0.204	0.187	0.137
DeepLabV3+	InceptionV3	5	0.572	0.532	0.442	0.690	0.701	0.641	0.691	0.662	0.592
DeepLabV3+	MobileNetV2	5	0.710	0.632	0.562	0.754	0.699	0.639	0.739	0.642	0.582
DeepLabV3+	ResNet101	5	0.600	0.521	0.451	0.703	0.659	0.589	0.664	0.609	0.559
LinkNet	InceptionV3	5	0.534	0.473	0.393	0.623	0.589	0.509	0.625	0.556	0.506
LinkNet	MobileNetV2	5	0.731	0.724	0.664	0.772	0.735	0.655	0.729	0.707	0.607
LinkNet	ResNet101	5	0.600	0.591	0.511	0.669	0.661	0.581	0.624	0.595	0.525
UNet	InceptionV3	5	0.504	0.491	0.431	0.607	0.604	0.524	0.572	0.569	0.519
UNet	MobileNetV2	5	0.724	0.665	0.585	0.750	0.744	0.644	0.765	0.763	0.762
UNet	ResNet101	5	0.561	0.521	0.471	0.646	0.613	0.523	0.665	0.664	0.584
SLZNet-Ord	-	5	0.572	0.326	0.276	0.608	0.565	0.485	0.650	0.570	0.500

TABLE 4.13 – Erreur absolue moyenne des modèles sur le jeu de données UAVID avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.718	0.793	0.873	0.668	0.673	0.723	0.750	0.809	0.819
DeepLabV3+	MobileNetV2	3	0.686	0.691	0.741	0.668	0.721	0.791	0.733	0.784	0.804
DeepLabV3+	ResNet101	3	0.680	0.711	0.731	0.659	0.698	0.718	0.671	0.704	0.714
LinkNet	InceptionV3	3	0.764	0.773	0.803	0.732	0.824	0.864	0.763	0.856	0.926
LinkNet	MobileNetV2	3	0.649	0.657	0.707	0.590	0.646	0.696	0.722	0.796	0.876
LinkNet	ResNet101	3	0.726	0.735	0.785	0.705	0.712	0.762	0.726	0.774	0.844
UNet	InceptionV3	3	0.770	0.815	0.845	0.764	0.799	0.849	0.763	0.820	0.880
UNet	MobileNetV2	3	0.729	0.790	0.810	0.720	0.736	0.786	0.729	0.771	0.851
UNet	ResNet101	3	0.761	0.800	0.830	0.706	0.740	0.780	0.773	0.782	0.862
SLZNet-Ord	-	3	0.765	0.819	0.839	0.757	0.769	0.789	0.792	0.860	0.880
DeepLabV3+	InceptionV3	5	0.739	0.828	0.878	0.691	0.758	0.788	0.675	0.749	0.769
DeepLabV3+	MobileNetV2	5	0.691	0.704	0.764	0.667	0.723	0.743	0.669	0.691	0.741
DeepLabV3+	ResNet101	5	0.734	0.833	0.893	0.692	0.733	0.803	0.695	0.756	0.766
LinkNet	InceptionV3	5	0.757	0.835	0.905	0.719	0.765	0.845	0.710	0.749	0.829
LinkNet	MobileNetV2	5	0.684	0.735	0.785	0.664	0.674	0.714	0.668	0.705	0.775
LinkNet	ResNet101	5	0.737	0.829	0.899	0.703	0.790	0.860	0.708	0.738	0.778
UNet	InceptionV3	5	0.773	0.863	0.893	0.731	0.823	0.873	0.729	0.743	0.793
UNet	MobileNetV2	5	0.683	0.684	0.724	0.670	0.750	0.780	0.653	0.753	0.793
UNet	ResNet101	5	0.749	0.829	0.909	0.712	0.776	0.816	0.695	0.755	0.835
SLZNet-Ord	-	5	0.743	0.828	0.868	0.722	0.735	0.805	0.696	0.637	0.657

TABLE 4.14 – Intersection sur Union moyenne (Mean IoU) des modèles sur le jeu de données UAVID avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.892	0.879	0.899	0.884	0.885	0.905	0.902	0.883	0.903
DeepLabV3+	MobileNetV2	3	0.869	0.989	9.993	0.875	0.996	0.992	0.854	0.987	0.989
DeepLabV3+	ResNet101	3	0.830	0.978	0.988	0.837	0.940	0.960	0.794	0.994	0.988
LinkNet	InceptionV3	3	0.920	0.995	9.998	0.906	0.995	0.992	0.913	0.988	0.991
LinkNet	MobileNetV2	3	0.910	0.998	0.998	0.897	0.998	0.998	0.936	0.994	0.993
LinkNet	ResNet101	3	0.886	0.986	0.985	0.882	0.985	0.995	0.898	0.984	0.985
UNet	InceptionV3	3	0.911	0.923	0.925	0.886	0.922	0.942	0.886	0.948	0.958
UNet	MobileNetV2	3	0.903	0.882	0.916	0.886	0.891	0.911	0.906	0.894	0.914
UNet	ResNet101	3	0.916	0.910	0.920	0.897	0.903	0.913	0.928	0.934	0.944
SLZNet-Ord	-	3	0.906	0.930	0.940	0.907	0.920	0.940	0.927	0.959	0.979
DeepLabV3+	InceptionV3	5	0.916	0.923	0.924	0.907	0.913	0.923	0.915	0.919	0.929
DeepLabV3+	MobileNetV2	5	0.881	0.910	0.930	0.895	0.918	0.938	0.890	0.905	0.925
DeepLabV3+	ResNet101	5	0.904	0.910	0.920	0.905	0.918	0.928	0.911	0.905	0.915
LinkNet	InceptionV3	5	0.945	0.999	0.993	0.940	0.998	0.998	0.948	0.999	0.998
LinkNet	MobileNetV2	5	0.896	0.866	0.876	0.900	0.874	0.884	0.915	0.862	0.872
LinkNet	ResNet101	5	0.920	0.978	0.984	0.922	0.980	0.990	0.946	0.978	0.988
UNet	InceptionV3	5	0.934	0.934	0.954	0.933	0.925	0.945	0.944	0.935	0.955
UNet	MobileNetV2	5	0.895	0.923	0.937	0.901	0.912	0.922	0.894	0.904	0.914
UNet	ResNet101	5	0.917	0.923	0.933	0.919	0.896	0.906	0.933	0.896	0.906
SLZNet-Ord	-	5	0.919	0.947	0.957	0.924	0.923	0.943	0.933	0.932	0.952

TABLE 4.15 – Score de sûreté (*Safety Score*) des modèles sur le jeu de données UAVID avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.322	0.231	0.134	0.430	0.346	0.316	0.248	0.243	0.149
DeepLabV3+	MobileNetV2	3	0.368	0.312	0.263	0.424	0.360	0.282	0.258	0.215	0.138
DeepLabV3+	ResNet101	3	0.379	0.305	0.248	0.448	0.373	0.299	0.340	0.327	0.280
LinkNet	InceptionV3	3	0.255	0.246	0.239	0.321	0.270	0.244	0.228	0.165	0.130
LinkNet	MobileNetV2	3	0.397	0.367	0.364	0.510	0.453	0.385	0.272	0.210	0.192
LinkNet	ResNet101	3	0.313	0.308	0.257	0.371	0.312	0.251	0.283	0.219	0.158
UNet	InceptionV3	3	0.244	0.203	0.166	0.274	0.264	0.233	0.258	0.185	0.148
UNet	MobileNetV2	3	0.306	0.225	0.143	0.334	0.247	0.242	0.268	0.242	0.233
UNet	ResNet101	3	0.266	0.194	0.105	0.358	0.333	0.250	0.222	0.133	0.102
SLZNet-Ord	-	3	0.255	0.213	0.168	0.292	0.276	0.255	0.204	0.187	0.128
DeepLabV3+	InceptionV3	5	0.695	0.654	0.566	0.849	0.815	0.772	0.543	0.521	0.429
DeepLabV3+	MobileNetV2	5	0.578	0.502	0.431	0.721	0.664	0.575	0.363	0.285	0.200
DeepLabV3+	ResNet101	5	0.660	0.639	0.540	0.817	0.727	0.709	0.550	0.513	0.430
LinkNet	InceptionV3	5	0.463	0.433	0.415	0.597	0.591	0.563	0.540	0.459	0.409
LinkNet	MobileNetV2	5	0.589	0.571	0.521	0.694	0.687	0.685	0.532	0.480	0.405
LinkNet	ResNet101	5	0.540	0.468	0.416	0.649	0.580	0.569	0.487	0.475	0.420
UNet	InceptionV3	5	0.371	0.296	0.246	0.457	0.372	0.282	0.325	0.246	0.216
UNet	MobileNetV2	5	0.500	0.426	0.401	0.587	0.529	0.527	0.459	0.368	0.323
UNet	ResNet101	5	0.510	0.495	0.424	0.632	0.560	0.518	0.386	0.322	0.229
SLZNet-Ord	-	5	0.480	0.448	0.370	0.631	0.594	0.535	0.297	0.278	0.220

TABLE 4.16 – Erreur absolue moyenne (*MAE*) des modèles sur le jeu de données ICG avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.718	0.793	0.857	0.668	0.673	0.732	0.750	0.809	0.825
DeepLabV3+	MobileNetV2	3	0.686	0.691	0.743	0.668	0.721	0.723	0.733	0.784	0.867
DeepLabV3+	ResNet101	3	0.680	0.711	0.765	0.659	0.698	0.772	0.671	0.704	0.726
LinkNet	InceptionV3	3	0.764	0.773	0.801	0.732	0.824	0.878	0.763	0.856	0.891
LinkNet	MobileNetV2	3	0.649	0.657	0.754	0.590	0.646	0.669	0.722	0.796	0.882
LinkNet	ResNet101	3	0.726	0.735	0.780	0.705	0.712	0.728	0.726	0.774	0.867
UNet	InceptionV3	3	0.770	0.815	0.892	0.764	0.799	0.884	0.763	0.820	0.902
UNet	MobileNetV2	3	0.729	0.790	0.878	0.720	0.736	0.760	0.729	0.771	0.846
UNet	ResNet101	3	0.761	0.800	0.821	0.706	0.740	0.828	0.773	0.782	0.870
SLZNet-Ord	-	3	0.765	0.819	0.859	0.757	0.769	0.847	0.792	0.860	0.892
DeepLabV3+	InceptionV3	5	0.703	0.742	0.778	0.669	0.719	0.761	0.727	0.797	0.897
DeepLabV3+	MobileNetV2	5	0.737	0.817	0.827	0.702	0.707	0.748	0.790	0.835	0.858
DeepLabV3+	ResNet101	5	0.704	0.742	0.767	0.670	0.680	0.681	0.712	0.762	0.857
LinkNet	InceptionV3	5	0.780	0.856	0.922	0.731	0.820	0.891	0.732	0.758	0.854
LinkNet	MobileNetV2	5	0.709	0.741	0.796	0.685	0.779	0.866	0.701	0.742	0.821
LinkNet	ResNet101	5	0.750	0.807	0.889	0.714	0.724	0.739	0.749	0.820	0.889
UNet	InceptionV3	5	0.812	0.827	0.883	0.790	0.833	0.901	0.817	0.821	0.888
UNet	MobileNetV2	5	0.760	0.809	0.894	0.737	0.746	0.843	0.754	0.767	0.798
UNet	ResNet101	5	0.757	0.804	0.887	0.719	0.816	0.838	0.785	0.810	0.846
SLZNet-Ord	-	5	0.768	0.854	0.876	0.739	0.781	0.805	0.832	0.930	0.938

TABLE 4.17 – Intersection sur Union moyenne (*mean IoU*) des modèles sur le jeu de données ICG avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.892	0.879	0.879	0.884	0.885	0.888	0.902	0.883	0.892
DeepLabV3+	MobileNetV2	3	0.869	0.989	0.994	0.875	0.996	0.996	0.854	0.987	0.994
DeepLabV3+	ResNet101	3	0.830	0.978	0.981	0.837	0.940	0.945	0.794	0.994	0.997
LinkNet	InceptionV3	3	0.920	0.995	0.998	0.906	0.995	1.000	0.913	0.988	0.991
LinkNet	MobileNetV2	3	0.910	0.998	1.006	0.897	0.998	1.000	0.936	0.994	0.996
LinkNet	ResNet101	3	0.886	0.986	0.989	0.882	0.985	0.992	0.898	0.984	0.990
UNet	InceptionV3	3	0.911	0.923	0.932	0.886	0.922	0.932	0.886	0.948	0.954
UNet	MobileNetV2	3	0.903	0.882	0.889	0.886	0.891	0.899	0.906	0.894	0.903
UNet	ResNet101	3	0.916	0.910	0.911	0.897	0.903	0.908	0.928	0.934	0.939
SLZNet-Ord	-	3	0.906	0.930	0.933	0.907	0.920	0.923	0.927	0.959	0.965
DeepLabV3+	InceptionV3	5	0.872	0.860	0.862	0.879	0.872	0.882	0.871	0.838	0.844
DeepLabV3+	MobileNetV2	5	0.869	0.993	0.998	0.869	0.921	0.926	0.880	0.956	0.964
DeepLabV3+	ResNet101	5	0.828	0.784	0.793	0.830	0.768	0.777	0.791	0.706	0.716
LinkNet	InceptionV3	5	0.912	0.990	0.991	0.892	0.986	0.989	0.900	0.990	0.995
LinkNet	MobileNetV2	5	0.891	0.966	0.972	0.880	0.955	0.955	0.902	0.964	0.970
LinkNet	ResNet101	5	0.873	0.987	0.993	0.863	0.982	0.990	0.877	0.989	0.996
UNet	InceptionV3	5	0.928	0.920	0.925	0.903	0.918	0.925	0.919	0.942	0.951
UNet	MobileNetV2	5	0.899	0.892	0.897	0.875	0.883	0.893	0.898	0.888	0.896
UNet	ResNet101	5	0.903	0.903	0.910	0.890	0.893	0.893	0.913	0.922	0.928
SLZNet-Ord	-	5	0.901	0.909	0.912	0.900	0.901	0.910	0.924	0.935	0.937

TABLE 4.18 – Score de sécurité (*Safety Score*) des modèles sur le jeu de données ICG avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.253	0.163	0.149	0.232	0.199	0.177	0.257	0.184	0.165
DeepLabV3+	MobileNetV2	3	0.253	0.229	0.215	0.232	0.228	0.203	0.257	0.199	0.197
DeepLabV3+	ResNet101	3	0.253	0.230	0.223	0.232	0.158	0.149	0.257	0.164	0.169
LinkNet	InceptionV3	3	0.156	0.096	0.083	0.154	0.140	0.128	0.166	0.130	0.129
LinkNet	MobileNetV2	3	0.164	0.071	0.067	0.158	0.111	0.093	0.174	0.126	0.121
LinkNet	ResNet101	3	0.165	0.148	0.117	0.154	0.056	0.043	0.172	0.160	0.144
UNet	InceptionV3	3	0.126	0.066	0.061	0.123	0.086	0.051	0.138	0.126	0.119
UNet	MobileNetV2	3	0.168	0.074	0.072	0.174	0.102	0.094	0.180	0.105	0.105
UNet	ResNet101	3	0.178	0.094	0.088	0.170	0.070	0.052	0.184	0.175	0.175
SLZNet-Ord	-	3	0.169	0.110	0.108	0.159	0.108	0.108	0.180	0.120	0.108
DeepLabV3+	InceptionV3	5	0.658	0.593	0.544	0.627	0.612	0.603	0.687	0.662	0.659
DeepLabV3+	MobileNetV2	5	0.651	0.638	0.586	0.618	0.564	0.563	0.681	0.650	0.621
DeepLabV3+	ResNet101	5	0.505	0.466	0.464	0.485	0.402	0.402	0.512	0.418	0.388
LinkNet	InceptionV3	5	0.311	0.244	0.235	0.305	0.211	0.203	0.328	0.254	0.237
LinkNet	MobileNetV2	5	0.304	0.246	0.158	0.304	0.238	0.226	0.330	0.302	0.243
LinkNet	ResNet101	5	0.312	0.234	0.226	0.304	0.225	0.211	0.325	0.231	0.251
UNet	InceptionV3	5	0.230	0.192	0.154	0.228	0.145	0.129	0.248	0.157	0.134
UNet	MobileNetV2	5	0.287	0.194	0.183	0.287	0.280	0.259	0.319	0.295	0.283
UNet	ResNet101	5	0.311	0.240	0.251	0.308	0.271	0.264	0.344	0.309	0.274
SLZNet-Ord	-	5	0.320	0.089	0.081	0.302	0.088	0.088	0.337	0.119	0.104

TABLE 4.19 – Erreur absolue moyenne (*MAE*) des modèles sur le jeu de données VALID avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.689	0.724	0.735	0.713	0.718	0.804	0.685	0.760	0.779
DeepLabV3+	MobileNetV2	3	0.689	0.747	0.842	0.713	0.772	0.784	0.685	0.724	0.728
DeepLabV3+	ResNet101	3	0.689	0.719	0.806	0.713	0.756	0.771	0.685	0.724	0.724
LinkNet	InceptionV3	3	0.790	0.803	0.810	0.792	0.832	0.838	0.778	0.798	0.825
LinkNet	MobileNetV2	3	0.789	0.820	0.820	0.797	0.811	0.813	0.778	0.835	0.864
LinkNet	ResNet101	3	0.781	0.821	0.823	0.794	0.884	0.881	0.773	0.837	0.899
UNet	InceptionV3	3	0.826	0.925	0.931	0.829	0.873	0.882	0.811	0.860	0.902
UNet	MobileNetV2	3	0.784	0.807	0.814	0.779	0.814	0.825	0.771	0.858	0.889
UNet	ResNet101	3	0.779	0.809	0.842	0.793	0.843	0.856	0.774	0.828	0.845
SLZNet-Ord	-	3	0.776	0.846	0.877	0.787	0.847	0.862	0.762	0.834	0.851
DeepLabV3+	InceptionV3	5	0.518	0.525	0.535	0.539	0.588	0.588	0.508	0.569	0.584
DeepLabV3+	MobileNetV2	5	0.520	0.609	0.609	0.543	0.640	0.641	0.509	0.561	0.561
DeepLabV3+	ResNet101	5	0.696	0.763	0.792	0.704	0.745	0.745	0.696	0.748	0.746
LinkNet	InceptionV3	5	0.800	0.851	0.862	0.806	0.846	0.873	0.794	0.847	0.862
LinkNet	MobileNetV2	5	0.797	0.853	0.866	0.795	0.868	0.872	0.786	0.842	0.854
LinkNet	ResNet101	5	0.802	0.808	0.893	0.807	0.826	0.835	0.798	0.819	0.820
UNet	InceptionV3	5	0.845	0.850	0.862	0.846	0.876	0.881	0.837	0.909	0.893
UNet	MobileNetV2	5	0.809	0.820	0.894	0.806	0.839	0.858	0.795	0.797	0.832
UNet	ResNet101	5	0.803	0.833	0.852	0.804	0.871	0.891	0.788	0.847	0.858
SLZNet-Ord	-	5	0.792	0.934	0.937	0.799	0.935	0.942	0.785	0.917	0.923

TABLE 4.20 – Intersection sur Union moyenne (*mean IoU*) des modèles sur le jeu de données VALID avec segmentation à 3 et 5 niveaux de sûreté

			Entraînement			Validaton			Test		
Architecture	**Backbone**	**Niveau de sûreté**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**	**Classification**	**Safety Loss**	**Safety Loss + Meta Learning**
DeepLabV3+	InceptionV3	3	0.911	0.980	0.983	0.921	0.983	0.993	0.913	0.973	0.973
DeepLabV3+	MobileNetV2	3	0.911	0.934	0.943	0.921	0.944	0.954	0.913	0.931	0.938
DeepLabV3+	ResNet101	3	0.911	0.928	0.937	0.921	0.936	0.946	0.913	0.922	0.930
LinkNet	InceptionV3	3	0.955	0.955	0.961	0.958	0.959	0.961	0.949	0.958	0.962
LinkNet	MobileNetV2	3	0.965	0.917	0.921	0.970	0.968	0.973	0.961	0.972	0.976
LinkNet	ResNet101	3	0.957	0.961	0.968	0.965	0.938	0.940	0.952	0.968	0.978
UNet	InceptionV3	3	0.967	0.990	0.990	0.971	0.992	0.991	0.959	0.982	0.989
UNet	MobileNetV2	3	0.957	0.992	0.997	0.951	0.993	0.899	0.951	0.992	0.997
UNet	ResNet101	3	0.951	0.979	0.988	0.956	0.979	0.982	0.951	0.970	0.976
SLZNet	-	3	0.935	0.991	0.993	0.942	0.992	0.993	0.926	0.990	0.991
DeepLabV3+	InceptionV3	5	0.921	0.981	0.985	0.929	0.982	0.982	0.922	0.974	0.980
DeepLabV3+	MobileNetV2	5	0.924	0.938	0.942	0.933	0.967	0.965	0.925	0.947	0.949
DeepLabV3+	ResNet101	5	0.942	0.988	0.987	0.947	0.951	0.959	0.942	0.923	0.924
LinkNet	InceptionV3	5	0.950	0.967	0.975	0.954	0.970	0.976	0.944	0.982	0.979
LinkNet	MobileNetV2	5	0.961	0.913	0.917	0.964	0.938	0.947	0.956	0.912	0.917
LinkNet	ResNet101	5	0.954	0.955	0.952	0.961	0.993	0.993	0.951	0.937	0.939
UNet	InceptionV3	5	0.971	0.987	0.982	0.974	0.990	0.989	0.964	0.978	0.986
UNet	MobileNetV2	5	0.951	0.987	0.988	0.948	0.990	0.985	0.943	0.982	0.991
UNet	ResNet101	5	0.954	0.979	0.978	0.957	0.978	0.983	0.950	0.969	0.969
SLZNet-Ord	-	5	0.947	0.984	0.987	0.953	0.986	0.989	0.940	0.977	0.981

TABLE 4.21 – Score de sûreté (*Safety Score*) des modèles sur le jeu de données VALID avec segmentation à 3 et 5 niveaux de sûreté

Dans l'ensemble, peu importe l'architecture, les modèles présentent la plupart du temps les mêmes caractéristiques qui ont été mentionnés précédemment. Lors de l'utilisation de la méthode de segmentation basée sur la classification, les modèles produisent les scores les plus faibles. Ceci est justifiable par le fait que la classification se base sur une fonction de perte qui émet une erreur symétrique, poussant le modèle à donner une erreur aussi grande pour une classification d'un pixel de classe *peu dangereux* autant dans la classe *dangereux* que *sécuritaire*. Ainsi, le Safety Score ne peut être à son meilleur, car l'erreur ne prend pas en compte une classification plus dangereuse d'une autre.

En ajoutant le Safety Loss, on peut remarquer une croissance du mean IoU, ainsi qu'une diminution du score MAE. Ce changement des scores signifie que les modèles émettent des prédictions qui classifient un plus grand nombre de pixels dans leurs classes

respectives en moyenne, pour chaque classe.

La nouveauté de cette section est l'ajout du Meta Learning. Dans l'ensemble, on voit que le Meta Learning a un effet semblable à celui de l'ajout du Safety Loss seul, apportant une augmentation du mean IoU, ainsi que du Safety Score, tout en diminuant le score MAE globablement. Cependant, de légères différences se voient apparaître. En effet, le but premier de l'ajout du Meta Learning fut la réduction de la prudence des réseaux lors de l'entraînement, dans le but d'augmenter le score de la diagonale, soit le nombre de pixels classés dans la classe à laquelle ils appartiennent réellement.

En modifiant la valeur de l'hyper-paramètre *phi* dans le Safety Loss de façon dynamique, la valeur du mean IoU s'est bel et bien vu augmenter. À titre d'exemple, il s'agit d'une moyenne d'augmentation de 0.048 pour le jeu de données UAVID, 0.0608 pour le jeu de données ICG et 0.01625 pour le jeu de données VALID. L'une des raisons pour lesquelles le score est plus petite pour le jeu de données VALID est expliqué par cette différence qu'entraîne le Meta Learning dans notre cas. Comme les scores de mean IoU étaient très élevés, frôlant même les 0.9 dans plusieurs cas, quelques architecture présentaient un score de *Safety Score* plus faible que d'autres. Ainsi, le Meta Learning a ajusté le facteur *phi*, de sorte que les modèles présentant des scores *Safety Factor* plus faibles privilégient celui-ci, augmentant le facteur, le réduisant dans les cas où la diagonale n'a pas tendance à augmenter.

À titre d'exemple, bien que le mean IoU du modèle UNet avec backbone InceptionV3 est passé de 0.909 à 0.893 dans le jeu de données VALID à 5 niveaux de sécurité, son *Safety Score* s'est vu passer de 0.978 à 0.986. L'effet contraire est aussi appliqué, comme par exemple avec le backbone InceptionV3 de l'architecture LinkNet qui a vu son *Safety Score* passer de 0.982 à 0.979, mais son mean IoU en augmentation de 0.847 à 0.862. Dans le cas précédent, la valeur de l'hyperparamètre a été baissée, réduisant l'effet d'une classification plus dangereuse sur la fonction de perte totale, augmentant ainsi le poids de la crossentropie sur la fonction de perte. On peut donc observer cet effet de balance apporté avec le Meta Learning, jouant entre la prudence, soit une plus forte valeur de l'hyperparamètre, ainsi qu'une meilleure classification avec une plus faible valeur de l'hyperparamètre.

Somme toute, l'ajout du Meta Learning a contribué à l'ajustement des modèles en optimisant, premièrement, la sécurité des prédictions, visiblement par l'augmentation du *Safety Score*, puis deuxième en optimisant la véracité des prédictions, quantifiable par l'augmentation du mean IoU.

Chapitre 5

Conclusion et perspectives

L'exploration des applications des UAV et de l'automatisation de leur pilotage a révélé des perspectives fascinantes et prometteuses pour divers secteurs. Néanmoins, assurer leur sécurité, ainsi que celle de l'environnement qui les entoure est primordial dans un monde où leur utilisation ne fait que croître. Bien que plusieurs se penchent sur l'utilisation et le perfectionnement de capteurs aux fins d'amélioration du pilotage, la segmentation d'images, combinée à l'augmentation de la vue du pilote, demeure un choix léger et peu coûteux permettant d'obtenir des résultats considérables quant à la sécurité du drone, du pilote et de son environnement, en plus d'améliorer l'expérience utilisateur du pilote. Le perfectionnement des techniques de segmentation, ainsi que l'arrivée des réseaux neuronaux ont eu un effet réducteur sur les contraintes qui limitaient l'utilisation de ceux-ci de façon sécuritaire. Toutefois, les architectures de modèles neuronaux populaires utilisées pour la segmentation montrent de bons résultats, aux dépens de la sécurité des UAV et/ou de l'environnement qui les entourent.

Ce mémoire s'est penché sur cette problématique en présentant des contributions innovantes dans le domaine de la segmentation des zones d'atterrissage sécuritaires. Tandis que nous avons traité la problématique des *SLZ* directement, les contributions notées du domaine de la segmentation par régression ordinale pourraient très bien être réutilisées dans des domaines connexes nécessitant de diminuer la sur-classification dans les prédictions d'un modèle. Parmi ces contributions, on note les suivantes :

— **Métrique *Safety Score*** : Construit à des fins d'évaluation de la sécurité des prédictions des modèles de segmentation par régression ordinale, cette nouvelle métrique normalisée permet de déterminer, lors de l'entraînement, si les prédictions d'un modèle deviennent plus ou moins sécuritaires par l'évaluation pixels en surclassification, soit dans des classes de niveaux supérieurs à leur classe respective.

— **Fonction de perte *Safety Loss*** : On a conçu une nouvelle fonction de pertes à deux termes, basée sur les bien-faits des fonctions de pertes asymétriques, ainsi que la pensée derrière notre métrique *Safety Score*. D'un côté, l'un de ces termes pousse le réseau à diminuer son erreur de classification peu importe la sécurité puis de l'autre, le deuxième terme (i.e. *Safety Score*), contrôlé par un hyperparamètre/facteur *phi*, pénalise le réseau pour toutes les classifications de niveaux supérieurs que celui-ci a produit.

En plus de ces contributions, on a montré la possibilité d'améliorer les valeurs des métriques, ainsi que la véracité des prédictions en faisant utilisation du Meta Learning. Bien qu'on ait utilisé une méthode de très bas niveau, avec des conditions d'entraînement bien précises, les résultats présentés montrent un potentiel quant à l'utilisation de cette méthode d'entrainement novatrice. Ainsi, dans le but d'améliorer les résultats et d'ouvrir les possibilités offertes par les contributions mentionnées dans ce mémoire, certaines perspectives futures seraient à considérer, telles que les suivantes :

— **Auto Machine Learning (AutoML) & Tuning :** L'ajustement automatique des hyperparamètres et la recherche exhaustive de valeurs idéales d'hyperparamètres pourrait permettre de trouver des valeurs de facteur *Phi*, dans la fonction de perte *Safety Loss*, qui s'adaptent aux modèles et qui émettent les meilleures valeurs possibles pour chaque architecture.

— **Ouverture du contexte d'application du *Safety Loss* :** Bien qu'on ait montré de très bons résultats quant à l'utilisation de la fonction de perte *Safety Loss* dans le domaine du pilotage des UAV, il existe un vaste éventail de domaines qui nécessitent une précision dans les prédictions émises par un modèle, pouvant apporter un certain danger à un agent ou son environnement. À titre d'exemple, le domaine médical regorge de problématiques dans lesquelles une sur-classification d'une donnée à travers n niveaux peut entraîner de fortes répercussions négatives sur un patient.

Bien que ce travail ne montre qu'une petite partie du potentiel découlant de l'utilisation de l'apprentissage machine dans l'automatisation des drones, il permet d'ouvrir une discussion sur l'harmonisation de automatisation et la sécurité, ouvrant un monde de possibilité de futurs travaux de recherche.

En premier lieu, partir de ces cartes de segmentation, il serait possible de bâtir une portion d'un système, basé sur la réalité augmentée, montrant les informations qui ressortent du modèle à un pilote afin d'effectuer une manoeuvre d'urgence. De plus, voyant la popularité de l'automatisation totale des vols, ces cartes de segmentation peuvent être incorporées dans un modèle plus grand permettant le contrôle autonome

du drone, tout en conservant la sécurité de celui-ci et de son environnement. Finalement, bien que la nouvelle fonction de perte *Safety Loss* soit efficace aux fins de segmentation des lieux sécuritaires d'atterrissage, elle demeure une fonction de perte. Il serait donc intéressant de vérifier son efficacité sur des modèles dans des domaines différents, où la régression ordinale est possible, tel que le domaine médical par exemple.

Finalement, il est vrai que les modèles montrés ont bien performés lors des simulations sur des prises de vues. En revanche, faire rouler ces modèles directement sur le drone peut représenter un enjeu majeur, sachant que la taille et la complexité de ceux-ci augmentent à la fois la complexité du processeur nécessaire, ainsi que l'énergie requise pour effectuer les calculs. Ainsi, afin d'en diminuer la taille et la complexité, des techniques comme la quantization, soit la réduction de la précision des poids et activations du modèle à des espaces mémoires plus petits [61], ou encore le pruning, soit l'évaluation et suppression des poids qui ont le plus faible impact sur les prédictions [72], seraient des techniques à considérer afin de réduire la taille des réseaux et ainsi la puissance de calcul nécessaire. Néanmoins, il faudrait tout de même conserver une juste balance entre efficacité et véracité, sachant que la réduction de la précision et la suppression de poids modifierait directement la qualité des prédictions qui sont produites.

Bibliographie

[1] Sakineh Abdollahzadeh, Pier-Luc Proulx, Mohand Said Allili et Jean-François Lapointe. "Safe Landing Zones Detection for UAVs Using Deep Regression". In : *2022 19th Conference on Robots and Vision (CRV)*. 2022, p. 213-218. doi : 10.1109/CRV55824.2022.00035.

[2] Morteza Alijani et Anas Osman. "Autonomous Landing of UAV on Moving Platform : A Mathematical Approach". In : *2020 International Conference on Control, Automation and Diagnosis (ICCAD)*. 2020, p. 1-6. doi : 10.1109/ICCAD49821.2020.9260498.

[3] Vijay Badrinarayanan, Alex Kendall et Roberto Cipolla. *SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 2016. arXiv : 1511.00561 [cs.CV].

[4] Dane Bamburry. "Drones : Designed for Product Delivery". In : *Design Management Review* 26.1 (2015), p. 40-48.

[5] Alex Bewley, Vitor Guizilini, Fabio Ramos et Ben Upcroft. "Online self-supervised multi-instance segmentation of dynamic objects". In : *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, p. 1296-1303. doi : 10.1109/ICRA.2014.6907020.

[6] Leo Breiman, Jerome Friedman, Richard Olshen et Charles Stone. *Classification And Regression Trees*. Oct. 2017, p. 1-358. isbn : 9781315139470. doi : 10.1201/9781315139470.

[7] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur et Roberto Cipolla. "Segmentation and Recognition Using Structure from Motion Point Clouds". In : *ECCV (1)*. 2008, p. 44-57.

[8] Ricardo J.G.B. Campello, Peer Kröger, Jörg Sander et Arthur Zimek. "Density-based clustering". English. In : *Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery* 10.2 (avr. 2020). issn : 1942-4787. doi : 10.1002/widm.1343.

[9] J. Canny. "Finding Edges and Lines in Images". In : *Theory of Computing Systems Mathematical Systems Theory* (1983), p. 16.

[10] Joao Carreira et Cristian Sminchisescu. "Constrained parametric min-cuts for automatic object segmentation". In : *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, p. 3241-3248. doi : 10.1109/CVPR.2010.5540063.

[11] Ashutosh Kumar Chaubey. "Comparison of the local and global thresholding methods in image segmentation". In : *World Journal of Research and Review* 2.1 (2016), p. 1-4.

[12] Abhishek Chaurasia et Eugenio Culurciello. "LinkNet : Exploiting Encoder Representations for Efficient Semantic Segmentation". In : *CoRR* abs/1707.03718 (2017). arXiv : 1707.03718. url : http://arxiv.org/abs/1707.03718.

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy et Alan L. Yuille. *DeepLab : Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. arXiv : 1606.00915 [cs.CV].

[14] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff et Hartwig Adam. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. arXiv : 1802.02611 [cs.CV].

[15] Lyujie Chen, Feng Liu, Yan Zhao, Wufan Wang, Xiaming Yuan et Jihong Zhu. "Valid : A comprehensive virtual aerial image dataset". In : *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2020, p. 2009-2016.

[16] Patrick C Chen et Theodosios Pavlidis. "Segmentation by texture using a co-occurrence matrix and a split-and-merge algorithm". In : *Computer Graphics and Image Processing* 10.2 (1979), p. 172-182. issn : 0146-664X. doi : https://doi.org/10.1016/0146-664X(79)90049-2. url : https://www.sciencedirect.com/science/article/pii/0146664X79900492.

[17] A. Choi-Fitzpatrick, Dana Chavarria, Elizabeth Cychosz, John Paul Dingens, Michael K. Duffey, K. Koebel, Sirisack Siriphanh, M. Tulen, Heath Watanabe, Tautvydas Jukauskas, J. Holland et L. Almquist. "Up in the Air : A Global Estimate of Non-Violent Drone Use 2009-2015". In : 2016.

[18] François Chollet. *Xception : Deep Learning with Depthwise Separable Convolutions*. 2017. arXiv : 1610.02357 [cs.CV].

[19] Francois Chollet et al. *Keras*. 2015. URL : https://github.com/fchollet/keras.

[20] Gülcan Gencer et Kerem Gencer. "Comparison Of Maximum Likelihood And Bayes Estimators Under Symmetric And Asymmetric Loss Functions By Means Of Tierney Kadane's Approximation For Weibull Distribution". In : *Turkish Journal of Science and Technology* 14.2 (2019), p. 69-78. ISSN : 1308-9080.

[21] G.B. Coleman et H.C. Andrews. "Image segmentation by clustering". In : *Proceedings of the IEEE* 67.5 (1979), p. 773-785. DOI : 10.1109/PROC.1979.11327.

[22] Corinna Cortes et Vladimir Vapnik. "Support-Vector Networks". In : *Machine Learning*. 1995, p. 273-297.

[23] Jing-Yan Yang Deng-Yuan Huang et Yu-Yun Wang. "Performance Evaluation of Semantic Segmentation Using Different Encoder-Decoder Architectures". In : *Journal of Science and Engineering Technology*. T. 16. 2. 2020, p. 35-42.

[24] H. Digabel et C. Lantuéjoul. "Iterative algorithms". In : *Actes du Second Symposium Européen d'Analyse Quantitative des Microstructures en Sciences des Matériaux, Biologie et Médecine*. Sous la dir. de R Verlag. Oct. 1978, p. 85-99.

[25] Kevin Dorling, Jordan Heinrichs, Geoffrey G. Messier et Sebastian Magierowski. "Vehicle Routing Problems for Drone Delivery". In : *IEEE Transactions on Systems, Man, and Cybernetics : Systems* 47.1 (2017), p. 70-85. DOI : 10.1109/TSMC.2016.2582745.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander et Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In : KDD'96. Portland, Oregon : AAAI Press, 1996, p. 226-231.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn et A. Zisserman. "The Pascal Visual Object Classes (VOC) Challenge". In : *International Journal of Computer Vision* 88.2 (juin 2010), p. 303-338.

[28] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn et Andrew Zisserman. "The Pascal Visual Object Classes (VOC) challenge". In : *International Journal of Computer Vision* 88 (juin 2010), p. 303-338. DOI : 10.1007/s11263-009-0275-4.

[29] Xing Fan, Guoping Zhang et Xuezhi Xia. "Performance Evaluation of SVM in Image Segmentation". In : *2008 IEEE International Workshop on Semantic Computing and Applications* (2008), p. 160-165.

[30] R. A. FISHER. “THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS”. In : *Annals of Eugenics* 7.2 (1936), p. 179-188. DOI : https://doi.org/10.1111/j.1469-1809.1936.tb02137.x. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-1809.1936.tb02137.x. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x.

[31] Dario FLOREANO et Robert J. WOOD. “Science, technology and the future of small autonomous drones”. In : *Nature* 521.7553 (mai 2015), p. 460-466. ISSN : 1476-4687. DOI : 10.1038/nature14542. URL : https://doi.org/10.1038/nature14542.

[32] Michael FONDER et Marc Van DROOGENBROECK. “Mid-Air : A multi-modal dataset for extremely low altitude drone flights”. In : *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*. Juin 2019.

[33] Gang FU, Hongrui ZHAO, Cong LI et Limei SHI. “Segmentation for High-Resolution Optical Remote Sensing Imagery Using Improved Quadtree and Region Adjacency Graph Technique”. In : *Remote Sensing* 5.7 (2013), p. 3259-3279. ISSN : 2072-4292. DOI : 10.3390/rs5073259. URL : https://www.mdpi.com/2072-4292/5/7/3259.

[34] Wolfgang GARN. “Drones reveal efficiency savings in delivery services”. In : (2015).

[35] Shuyu GONG, Zhewei WANG, Tao SUN, Yuanhang ZHANG, Charles D. SMITH, Li XU et Jundong LIU. “Dilated FCN : Listening Longer to Hear Better”. In : *CoRR* abs/1907.11956 (2019). arXiv : 1907.11956. URL : http://arxiv.org/abs/1907.11956.

[36] J.P. GONZALEZ et U. OZGUNER. “Lane detection using histogram-based segmentation and decision trees”. In : *ITSC2000. 2000 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.00TH8493)*. 2000, p. 346-351. DOI : 10.1109/ITSC.2000.881084.

[37] Rafael C. GONZALEZ et Richard E. WOODS. *Digital Image Processing (3rd Edition)*. USA : Prentice-Hall, Inc., 2006. ISBN : 013168728X.

[38] Javier GONZALEZ-TREJO et Diego MERCADO-RAVELL. “Lightweight Density Map Architecture for UAVs Safe Landing in Crowded Areas”. In : *Journal of Intelligent Robotic Systems* 102 (mai 2021). DOI : 10.1007/s10846-021-01380-8.

[39] Javier GONZÁLEZ-TREJO, Diego MERCADO-RAVELL, Israel BECERRA et Rafael MURRIETA-CID. “On the Visual-Based Safe Landing of UAVs in Populated Areas : A Crucial Aspect for Urban Deployment”. In : *IEEE Robotics and Automation Letters* 6.4 (2021), p. 7901-7908. DOI : 10.1109/LRA.2021.3101861.

[40] Vairaprakash Gurusamy, Subbu Kannan et G.Nalini. "REVIEW ON IMAGE SEGMENTATION TECHNIQUES". In : oct. 2014.

[41] Joko Hariyono, Joko Saputro, Faisal Rahutomo, Sutrisno, Meiyanto Eko Sulistyo, Subuh Pramono et Muhammad Hamka. "Landing Pad Detection and Computing Direction of Motion for Autonomous Precision Landing Quadcopter". In : *E3S Web of Conferences* 465 (déc. 2023). DOI : 10.1051/e3sconf/202346502068.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv : 1512.03385 [cs.CV].

[43] Hal Hodson. *Enter the drone zones*. 2014.

[44] Zhang Hong et Fan Jiulun. "A Threshold Segmentation Method for Sparse Histogram Image". In : *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*. T. 5. 2009, p. 340-344. DOI : 10.1109/FSKD.2009.557.

[45] Yuhuang Hu, Adrian Huber, Jithendar Anumula et Shih-Chii Liu. *Overcoming the vanishing gradient problem in plain recurrent networks*. 2019. arXiv : 1801.06105 [cs.NE].

[46] Gregory Hunter et Kenneth Steiglitz. "Operations on Images Using Quad Trees". In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-1 (mai 1979), p. 145-153. DOI : 10.1109/TPAMI.1979.4766900.

[47] Institute of Computer Graphics and Vision (ICG). Accessed Jan. 31, 2021. Graz University of Technology (TU Graz). URL : http://dronedataset.icg.tugraz.at.

[48] Xiaoqiang Ji, Yang Li, Jiezhang Cheng, Yuanhua Yu et Meijiao Wang. "Cell image segmentation based on an improved watershed algorithm". In : *2015 8th International Congress on Image and Signal Processing (CISP)*. 2015, p. 433-437. DOI : 10.1109/CISP.2015.7407919.

[49] Michael Kass, Andrew Witkin et Demetri Terzopoulos. "Snakes : Active contour models". In : *INTERNATIONAL JOURNAL OF COMPUTER VISION* 1.4 (1988), p. 321-331.

[50] Taranjit Kaur et Tapan Kumar Gandhi. "Automated Brain Image Classification Based on VGG-16 and Transfer Learning". In : *2019 International Conference on Information Technology (ICIT)*. 2019, p. 94-98. DOI : 10.1109/ICIT48102.2019.00023.

[51] Joe KINAHAN et Alan F SMEATON. "Image Segmentation to Identify Safe Landing Zones for Unmanned Aerial Vehicles". In : *arXiv preprint arXiv :2111.14557* (2021).

[52] Joe KINAHAN et Alan F. SMEATON. "Image Segmentation to Identify Safe Landing Zones for Unmanned Aerial Vehicles". In : *CoRR* abs/2111.14557 (2021). arXiv : 2111.14557. URL : https://arxiv.org/abs/2111.14557.

[53] Abbas Z. KOUZANI. "Road-Sign Identification Using Ensemble Learning". In : *2007 IEEE Intelligent Vehicles Symposium*. 2007, p. 438-443. DOI : 10.1109/IVS.2007.4290154.

[54] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON. "ImageNet Classification with Deep Convolutional Neural Networks". In : *Advances in Neural Information Processing Systems*. Sous la dir. de F. PEREIRA, C. J. C. BURGES, L. BOTTOU et K. Q. WEINBERGER. T. 25. Curran Associates, Inc., 2012. URL : https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[55] Jonathan LONG, Evan SHELHAMER et Trevor DARRELL. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv : 1411.4038 [cs.CV].

[56] Wenjie LUO, Yujia LI, Raquel URTASUN et Richard ZEMEL. "Understanding the effective receptive field in deep convolutional neural networks". In : *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, p. 4905-4913.

[57] Ye LYU, George VOSSELMAN, Guisong XIA, Alper YILMAZ et Michael Ying YANG. *The UAVid Dataset for Video Semantic Segmentation*. 2018. eprint : arXiv:1810.10438.

[58] Eka MANDYARTHA, Fetty ANGGRAENY, Faisal MUTTAQIN et Fawwaz AKBAR. "Global and Adaptive Thresholding Technique for White Blood Cell Image Segmentation". In : *Journal of Physics : Conference Series* 1569 (juil. 2020), p. 022054. DOI : 10.1088/1742-6596/1569/2/022054.

[59] T.G. MCGEE, R. SENGUPTA et K. HEDRICK. "Obstacle Detection for Small Autonomous Aircraft Using Sky Segmentation". In : *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, p. 4679-4684. DOI : 10.1109/ROBOT.2005.1570842.

[60] Anthony J MYLES, Robert N FEUDALE, Yang LIU, Nathaniel A WOODY et Steven D BROWN. "An introduction to decision tree modeling". In : *Journal of Chemometrics : A Journal of the Chemometrics Society* 18.6 (2004), p. 275-285.

[61] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen et Tijmen Blankevoort. *A White Paper on Neural Network Quantization*. 2021. arXiv : 2106.08295 [cs.LG]. URL : https://arxiv.org/abs/2106.08295.

[62] HP Narkhede. "Review of image segmentation techniques". In : *International Journal of Science and Modern Engineering* 1.8 (2013), p. 54-61.

[63] Hyeonwoo Noh, Seunghoon Hong et Bohyung Han. "Learning Deconvolution Network for Semantic Segmentation". In : *CoRR* abs/1505.04366 (2015). arXiv : 1505.04366. URL : http://arxiv.org/abs/1505.04366.

[64] Edwin Olson. "AprilTag : A robust and flexible visual fiducial system". In : *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, mai 2011, p. 3400-3407.

[65] Haris Papasaika-Hanusch. "Digital image processing using matlab". In : *Institute of Geodesy and Photogrammetry, ETH Zurich* 63 (1967).

[66] Timothy Patterson, Sally McClean, Philip Morrow, Gerard Parr et Chunbo Luo. "Timely autonomous identification of UAV safe landing zones". In : *Image and Vision Computing* 32.9 (2014), p. 568-578. ISSN : 0262-8856. DOI : https://doi.org/10.1016/j.imavis.2014.06.006. URL : https://www.sciencedirect.com/science/article/pii/S026288561400105X.

[67] Judith M. S. Prewitt et Mortimer L. Mendelsohn. "The Analysis of Cell Images". In : *Annals of the New York Academy of Sciences* 128.3 (jan. 1965), p. 1035-1053. DOI : 10.1111/j.1749-6632.1965.tb11715.x.

[68] D. Rivest-Hénault, M. Cheriet, S. Deschênes et C. Lapierre. "Length Increasing Active Contour for the Segmentation of Small Blood Vessels". In : *2010 20th International Conference on Pattern Recognition*. 2010, p. 2796-2799. DOI : 10.1109/ICPR.2010.685.

[69] Olaf Ronneberger, Philipp Fischer et Thomas Brox. *U-Net : Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv : 1505.04597 [cs.CV].

[70] A. Rosenfeld et M. Thurston. "Edge and Curve Detection for Visual Scene Analysis". In : *IEEE Transactions on Computers* C-20.5 (1971), p. 562-569. DOI : 10.1109/T-C.1971.223290.

[71] Stuart J. (Stuart Jonathan) Russell. *Artificial intelligence : a modern approach*. Includes bibliographical references (pages 1063-1093) and index. Third edition. Upper Saddle River, N.J. : Prentice Hall, [2010] ©2010, [2010]. URL : https://search.library.wisc.edu/catalog/9910082172502121.

[72] Abdullah Salama, Oleksiy Ostapenko, Tassilo Klein et Moin Nabi. *Pruning at a Glance : Global Neural Pruning for Model Compression*. 2019. arXiv : 1912.00200 [cs.CV]. URL : https://arxiv.org/abs/1912.00200.

[73] R. Samadani. "Changes in connectivity in active contour models". In : *[1989] Proceedings. Workshop on Visual Motion*. 1989, p. 337-343. DOI : 10.1109/WVM.1989.47127.

[74] Judy Scott et Carlton Scott. "Drone Delivery Models for Healthcare". In : *Proceedings of the 50th Hawaii International Conference on System Sciences (2017)* (2017). DOI : 10.24251/hicss.2017.399.

[75] M. Sezgin et Bulent Sankur. "Survey over image thresholding techniques and quantitative performance evaluation". In : *Journal of Electronic Imaging* 13 (jan. 2004), p. 146-168. DOI : 10.1117/1.1631315.

[76] Md Shah Alam et Jared Oluoch. "A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs)". In : *Expert Systems with Applications* 179 (2021), p. 115091. ISSN : 0957-4174. DOI : https://doi.org/10.1016/j.eswa.2021.115091. URL : https://www.sciencedirect.com/science/article/pii/S0957417421005327.

[77] Claude E Shannon. "A mathematical theory of communication". In : *Bell System Technical Journal* 27.3 (1948), p. 379-423.

[78] Karen Simonyan et Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In : *International Conference on Learning Representations*. 2015.

[79] George Stockman et Linda G. Shapiro. *Computer Vision*. 1st. USA : Prentice Hall PTR, 2001. ISBN : 0130307963.

[80] Jörg Stückler et Sven Behnke. "Combining depth and color cues for scale- and viewpoint-invariant object segmentation and recognition using Random Forests". In : *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, p. 4566-4571. DOI : 10.1109/IROS.2010.5654338.

[81] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke et Andrew Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv : 1409.4842 [cs.CV].

[82] Martin Thoma. *A Survey of Semantic Segmentation*. 2016. arXiv : 1602.06541 [cs.CV].

[83] Yafu TIAN, Ke WANG, Ruifeng LI et Lijun ZHAO. "A fast incremental map segmentation algorithm based on spectral clustering and quadtree". In : *Advances in Mechanical Engineering* 10.2 (2018), p. 1687814018761296. DOI : 10.1177/1687814018761296.

[84] Sik-Ho TSANG. *Review : DeconvNet — Unpooling Layer (Semantic Segmentation)*. 2018. URL : https://towardsdatascience.com/review-deconvnet-unpooling-layer-semantic-segmentation-55cf8a6e380e (visité le 07/11/2021).

[85] Sik-Ho TSANG. *Review : DilatedNet — Dilated Convolution (Semantic Segmentation)*. Nov. 2018. URL : https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5.

[86] Joaquin VANSCHOREN. "Meta-Learning : A Survey". In : *CoRR* abs/1810.03548 (2018). arXiv : 1810.03548. URL : http://arxiv.org/abs/1810.03548.

[87] Huikai WU, Junge ZHANG, Kaiqi HUANG, Kongming LIANG et Yizhou YU. *FastFCN : Rethinking Dilated Convolution in the Backbone for Semantic Segmentation*. 2019. arXiv : 1903.11816 [cs.CV].

[88] Ming WU, Chuang ZHANG, Jiaming LIU, Lichen ZHOU et Xiaoqi LI. "Towards Accurate High Resolution Satellite Image Semantic Segmentation". In : *IEEE Access* 7 (2019), p. 55609-55619. DOI : 10.1109/ACCESS.2019.2913442.

[89] Yan XIE, Fang MIAO, Kai ZHOU et Jing PENG. "HsgNet : A Road Extraction Network Based on Global Perception of High-Order Spatial Information". In : *ISPRS International Journal of Geo-Information* 8.12 (2019). ISSN : 2220-9964. DOI : 10.3390/ijgi8120571. URL : https://www.mdpi.com/2220-9964/8/12/571.

[90] Jingru YI, Pengxiang WU, Daniel J. HOEPPNER et Dimitris METAXAS. "Pixel-wise neural cell instance segmentation". In : *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. 2018, p. 373-377. DOI : 10.1109/ISBI.2018.8363596.

[91] Fisher YU et Vladlen KOLTUN. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016. arXiv : 1511.07122 [cs.CV].

[92] Z. ZHANG et M. SIMAAN. "Knowledge-based texture image segmentation using iterative linked quadtree splitting". In : *International Conference on Acoustics, Speech, and Signal Processing*. 1990, 2321-2324 vol.4. DOI : 10.1109/ICASSP.1990.116046.

[93] Ziyu ZHANG, Alexander G. SCHWING, Sanja FIDLER et Raquel URTASUN. "Monocular Object Instance Segmentation and Depth Ordering with CNNs". In : *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, p. 2614-2622. DOI : 10.1109/ICCV.2015.300.

[94] Wufan ZHAO, Claudio PERSELLO et Alfred STEIN. "Building Instance Segmentation and Boundary Regularization from High-Resolution Remote Sensing Images". In : *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*. 2020, p. 3916-3919. DOI : 10.1109/IGARSS39084.2020.9324239.

[95] Lichen ZHOU, Chuang ZHANG et Ming WU. "D-LinkNet : LinkNet with Pretrained Encoder and Dilated Convolution for High Resolution Satellite Imagery Road Extraction". In : *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2018, p. 192-1924. DOI : 10.1109/CVPRW.2018.00034.

[96] Xiong ZHOU, Xianming LIU, Junjun JIANG, Xin GAO et Xiangyang JI. "Asymmetric Loss Functions for Learning with Noisy Labels". In : *Proceedings of the 38th International Conference on Machine Learning*. Sous la dir. de Marina MEILA et Tong ZHANG. T. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, p. 12846-12856. URL : https://proceedings.mlr.press/v139/zhou21f.html.