

École doctorale n° 77 : IAEM

THÈSE

pour obtenir le grade de docteur délivré par

I'Université de Lorraine Spécialité doctorale "Informatique"

Calcul neuromorphique pour l'exploration et la catégorisation robuste d'environnement visuel et multimodal dans les systèmes embarqués.

présentée et soutenue publiquement par

Yann BERNARD

le 8 Vendémiaire An CCXXX

Directeur de thèse : **Bernard GIRAU**
Co-encadrant de thèse : **Nicolas HUEBER**
Co-encadrant de thèse : **Pierre RAYMOND**

Jury

Mme. Marie Cotrell,	Professeure	Rapporteure
M. Michel Verleysen,	Professeur	Rapporteur
M. Michel Paindavoine,	Professeur	Examinateur
Mme. Isabelle Debled-Renaisson,	Professeure	Examinatrice

Table des matières

Table des matières	iii
Liste des figures	v
Liste des tableaux	vii
1 Etat de l'art	1
1.1 Contexte	2
1.2 Réseaux neuronaux	12
1.3 Références	19
2 Détection de Nouveauté	23
2.1 Introduction	24
2.2 Application aux images	24
2.3 Détection de nouveauté par QVAT : quantification vectorielle et appren-tissage topologique	29
2.4 Protocole expérimental	34
2.5 Résultats expérimentaux	49
2.6 Références	60
3 Vision événementielle et attention	63
3.1 Introduction	64
3.2 Détection de mouvements	64
3.3 Mécanisme attentionnel	67
3.4 Combinaison avec la détection de nouveauté	67
3.5 Conclusion	67
3.6 Références	67
4 Calcul de Best Matching Unit accéléré avec la topologie	69
4.1 Introduction	70
4.2 Algorithme séquentiel	72
4.3 Algorithme parallèle	82
4.4 Conclusion	82
4.5 Références	82

TABLE DES MATIÈRES

Liste des figures

1.1	Illustration de l'efficacité du traitement visuel humain.	8
1.2	Phonème SOM	12
1.3	Apprentissage de SOM	15
1.4	Apprentissage de GNG	17
2.1	Lac de Nino	25
2.2	Représentation d'une image	26
2.3	Compression et décompression d'image	27
2.4	Représentation d'une image	28
2.5	Détection de nouveauté par quantification vectorielle	29
2.6	Exemples de différence avec quantification vectorielle	30
2.7	Détection de nouveauté avec topologie	31
2.8	Exemples de différence avec distance neurale.	32
2.9	Catégorie Baseline	35
2.10	Catégorie Bad Weather	35
2.11	Catégorie Camera Jitter	35
2.12	Catégorie Dynamic Background	35
2.13	Catégorie Shadow	36
2.14	Catégorie Night Videos	36
2.15	Catégorie Thermal	36
2.16	Catégorie Turbulence	37
2.17	Catégorie Intermittent Object Motion - Reduced	37
2.18	Catégorie Low Framerate - Reduced	37
2.19	Différence entre nouveauté and changement	38
2.20	Échantillonnage de l'évaluation	42
2.21	Effet de l'aléatoire sur les métriques	44
2.22	Optimisation de σ	46
2.23	Optimisation d' α	47
2.24	Nombre d'époques	48
2.25	Seuil de décision	50
2.26	Nombre de neurones et de taille des imagettes, SOM/baseline	51
2.27	Nombre de neurones et de taille des imagettes, GNG/baseline	52
2.28	Nombre de neurones et de taille des imagettes, PSNR/SOM/baseline	53
2.29	Nombre de neurones et de taille des imagettes, Global	54
2.30	Visualisation résultats normaux	58
2.31	Visualisation résultats dynamiques	58
2.32	Visualisation des effets de la normalisation	59

3.1	Détection de mouvements avec DNF sur caméra événementielle	64
3.2	Détection de mouvements multicible avec DNF sur caméra événementielle	65
4.1	Left : Example of the execution of one particle. Each cell represents one neuron. The arrow in each cell points towards the best neighbour (where a lower distance can be found). A star represents a local minimum. The green colored cells are the neurons who have been explored during the search, and the blue cells are the neurons whose distance to the input has been computed but that have not been explored by the algorithm. After having found a local minimum, new particles are created (in red) and continue the gradient descent by exploring the local neighbourhood. The cell in Yellow is the BMU, and the global minimum of the map. Right : Execution of the four particles.	75
4.2	Évaluation des gains de performance en fonction du nombre de neurones	81

Liste des tableaux

2.1	Estimations statistiques du nombre de graines requises	44
2.2	Paramètres GNG	45
2.3	Récapitulatif des paramètres SOM	45
2.4	Résultats complets sur CDNET de notre détection de nouveauté	55
2.5	Résultats complets sur CDNET de notre détection de nouveauté - Suite	56
2.6	Comparatif avec d'autres modèles de détection de changement sur CD-NET	57
3.1	Paramètres DNF 1D	66
4.1	Résultats de FastBMU sur différentes données	79

Chapitre 1

Etat de l'art

« *If I knew how I knew everything
I knew, then I would only be able
to know half has much, because
it will all be clogged up with
where I know it from.
So I cannot always cite my
sources, I'm sorry.* »

David Mitchell

Sommaire

1.1 Contexte	2
1.1.1 L'inspiration biologique	3
1.1.2 L'émergence	5
1.1.3 Particularités de la vision	7
1.1.4 Vision humaine	8
1.1.5 Vision par ordinateur : Détection de nouveauté	10
1.2 Réseaux neuronaux	12
1.2.1 Cartes auto organisatrices	12
1.2.2 Principes de fonctionnement	13
1.2.3 Gaz Neuronaux en Expansion	15
1.2.4 DNF	17
1.3 Références	19

1.1 Contexte

Depuis les premiers pas de l'informatique, une question s'est posée : est-ce que les machines peuvent penser ? **TURING [1950]** Cette question, bien qu'abstraite dans sa formulation, se réfère à l'intelligence humaine. Il n'est en effet ici pas question de savoir si ce que les ordinateurs font est de la pensée, mais bien si l'on peut reproduire l'intelligence humaine dans un cerveau électronique. Cette question a été le fondement d'un nouveau domaine scientifique qui a été nommé, lors de la conférence de Dartmouth **McCARTHY et collab. [1955]** : l'intelligence artificielle (IA).

Les années qui suivirent n'amènerent pas de résultat au niveau des attentes des chercheurs. Le domaine était encore trop limité par la puissance de calcul des ordinateurs de cette époque, et les modèles primitifs n'arrivaient qu'à résoudre des tâches simples.

La première évolution majeure est arrivée dans les années 90, lorsque les processeurs ont dépassé le million de transistors par puce et lorsque les méthodes algorithmiques étaient plus avancées, avec par exemple la recherche rapide dans une base de données. Ces changements ont permis aux modèles d'IA d'exploiter plus de données, plus rapidement et où la programmation et les heuristiques étaient les facteurs les plus importants de réussite. Les succès les plus connus de cette époque comptent la victoire de Deep Blue sur Kasparov aux échecs **CAMPBELL et collab. [2002]**, et le programme Watson d'IBM **FERRUCCI [2012]**, qui pourrait être considéré comme le dernier grand projet d'IA de cette période. Il commençait déjà à utiliser quelque chose qui allait constituer la nouvelle évolution de l'IA : l'apprentissage.

L'apprentissage automatique est une sous-discipline du domaine de l'IA qui s'est développée tout au long de son histoire et qui est passée sur le devant de la scène depuis les années 2010. C'est une combinaison de plusieurs facteurs qui l'ont amené à dépasser tous les records à ce moment-là : les réseaux de neurones multicouches étaient arrivés à maturation et permettaient l'apprentissage de données avec des représentations à haut niveau d'abstraction. Les dispositifs de calculs étaient toujours plus puissants, avec notamment les cartes graphiques dédiées devenues programmables et qui permirent d'effectuer du calcul de nombres à virgule flottante en parallèle. Et en dernier, la présence de jeux de données massifs avec lesquels on pouvait entraîner des réseaux toujours plus grands et complexes, et en produisant des résultats toujours plus impressionnantes. Les exemples de révolutions applicatives sont légion. La compétition de reconnaissance d'image ImageNet est par exemple passée de taux d'erreurs de 25% avec des approches classiques à 15% en 2012 avec le réseau apprenant AlexNet **KRIZHEVSKY et collab. [2012]**, fondé sur des travaux antérieurs de Yann Le Cun **LECUN et collab. [1989]**. Les années suivantes ont vu l'explosion de ce type de réseaux, qui grâce à leur généricité, ont été appliqués à quasiment tous les domaines de l'informatique et au-delà. En 2017, 5 ans après AlexNet, ImageNet était résolu avec la majorité des participants atteignant des taux d'erreurs inférieurs à 5%. D'autres succès notables sont la victoire au Go par AlphaGo contre Lee Sedol **SILVER et collab. [2016]**, et la série de modèles de langage GPT (Generative Pre-trained Transformer) **BROWN et collab. [2020]**.

Malgré ces nombreux succès, des nuages ont commencé à apparaître dans le ciel des réseaux de neurones. GPT-3, le modèle le plus récent d'OpenAI, utilise 175

milliards de paramètres, nécessitant 350 gigaoctets de VRAM juste pour effectuer une inférence. Son coût d'apprentissage a été estimé entre 11 et 28 millions de dollars américains¹. La consommation électrique elle, est estimée dans les environs de 190 MWh, correspondant à des émissions en gaz à effet de serre d'un aller-retour terre-lune en voiture². Le corpus de textes utilisé était 150 fois plus gros que Wikipédia, lui-même inclus dedans. Les performances étaient quant à elles toujours inférieures à un humain, qui lui n'a accès qu'à un cerveau de 25 Watts **KANDEL et collab. [2000]** et un corpus d'apprentissage extrêmement petit en comparaison. On estime qu'un enfant dans une famille aisée entend environ 11,2 millions de mots par an **HART et RISLEY [2003]**. Extrapolé pour 20 ans, cela ferait 224 millions de mots pour avoir une bonne maîtrise de la langue, soit 0,05% des 500 milliards de mots sur lesquels GPT a été entraîné.

La même observation peut être faite sur tous les succès des réseaux neuronaux. AlphaGo par exemple a eu besoin de 1920 CPUs et 280 GPUs pour battre Lee Sedol, avec une puissance nécessaire estimée à 1 MW³, soit 100 fois plus que son opposant, Lee Sedol. Les performances surhumaines des réseaux de neurones actuels ne proviennent pas tant de l'intelligence dans les modèles déployés, mais de leur capacité à mobiliser de grandes quantités de ressources pour résoudre un problème particulier. Les réseaux de neurones actuels montrant ainsi de plus en plus leurs limites, se pose la question de quelle sera la prochaine épine dorsale de l'IA, et quels seront les développements qui amèneront la prochaine évolution de la discipline.

Cette thèse s'inscrit dans un courant de pensée qui considère deux approches complémentaires comme étant les clés potentielles vers cette nouvelle évolution : l'accroissement des capacités de calculs par le neuromorphisme et l'augmentation de l'efficacité et de la puissance d'apprentissage par l'émergence et la complexité, que nous allons expliquer dans les sections suivantes.

1.1.1 L'inspiration biologique

Depuis l'avènement des semi-conducteurs, la puissance computationnelle disponible n'a cessé d'accroître exponentiellement. La célèbre prophétie auto-réalisatrice de Gordon Moore, que la densité de transistors par microprocesseur double tous les deux ans, a fait progresser l'industrie pour lui faire atteindre le million de transistors par puce à l'aube des années 1990, et le milliard en 2006. Au moment de la rédaction de ce manuscrit, la plus grosse puce est le processeur A100 à 54 milliards de transistors⁴.

Cette progression cache cependant des difficultés pour utiliser efficacement tous ces milliards de transistors. L'architecture de Von Neumann **VON NEUMANN [1945]** qui sert de modèle depuis les années 50 à quasiment tous les ordinateurs est le point bloquant de la puissance des processeurs. La séparation de la mémoire et du CPU place le bus de données entre les deux comme le centre névralgique de l'ordinateur et sa vitesse est limitée. Ce problème est compensé en partie dans les processeurs modernes avec l'utilisation de cache, mais ce n'est qu'une solution d'appoint qui ne

1. <https://bdtechtalks.com/2020/09/21/gpt-3-economy-business-model>
2. https://www.theregister.com/2020/11/04/gpt3_carbon_footprint_estimate/
3. <https://jacquesmattheij.com/another-way-of-looking-at-lee-sedol-vs-alphago/>
4. [https://en.wikipedia.org/wiki/Ampere_\(microarchitecture\)](https://en.wikipedia.org/wiki/Ampere_(microarchitecture))

tient pas la mise à l'échelle, car les tailles de cache resteront toujours très inférieures à la mémoire vive. Un second problème, plus important encore, est la limite des performances séquentielles des CPU.

L'informatique a, dès la machine de Turing, été principalement séquentielle et les algorithmes et processeurs se sont développés pour l'essentiel dans ce paradigme séquentiel. Cela n'a pas posé de problèmes tant que les fréquences augmentaient et que la finesse de gravure se réduisait, amenant toujours plus de performances. Mais un mur de fréquence a été atteint dans les années 2000, dû aux coûts énergétiques et à la chaleur engendrée par les fréquences trop élevées. Depuis les processeurs grand public ont rarement dépassé les 5 Ghz. Cette limite aux performances séquentielles des CPU combinée à la croissance exponentielle du nombre de transistors a naturellement amené vers le développement d'architectures multi-cœurs, qui ont nécessité un développement plus poussé vers le parallélisme tant au niveau matériel qu'algorithme. C'est ce parallélisme qui a été le catalyseur pour le développement des réseaux de neurones, particulièrement bien parallélisables. La limite de l'architecture de von Neumann elle, reste présente schématiquement. Lors de l'apprentissage d'un réseau de neurones avec de multiples GPU, la mémoire de ceux-ci doit être dupliquée dans chaque carte, et synchronisée régulièrement, si bien que la taille de ces réseaux de neurones est ultimement limitée par la mémoire vive du plus petit GPU. Mais l'architecture de von Neumann n'est pas une fatalité, et il existe de nombreuses idées pour pallier ses problèmes. La piste que nous avons choisie ici est de s'inspirer de la machine la plus efficace pour traiter de l'information : le cerveau.

Un cerveau est une machinerie complexe, composé d'environ 170 milliards de cellules chez l'humain, dont la moitié environ sont des neurones, le reste des cellules gliales. C'est un ordre de grandeur similaire au nombre de transistors dans un circuit imprimé récent, mais il y existe tout de même des différences importantes entre les deux. Un neurone est capable d'effectuer des opérations beaucoup plus complexes qu'un transistor. Il est également beaucoup plus grand, de l'ordre du micromètre, alors que les transistors ne font que quelques nanomètres. Ce qui explique en partie la différence de taille. Un cerveau humain occupe un volume d'environ 1200 cm^3 , avec de grandes variations entre les genres et les individus. Les puces informatiques se contentent de deux dimensions et s'étalent sur une petite surface en comparaison, soit 826mm^2 pour la plus grande. En supposant une épaisseur de 5mm, on obtient un volume de $4,13 \text{ cm}^3$. Ces comparaisons semblent indiquer que le nombre de transistors disponible n'est pas un facteur limitant pour le développement de modèles informatiques plus proches des capacités humaines de raisonnement.

La différence d'organisation entre un cerveau et un processeur est cependant majeure. Un cerveau fonctionne de façon analogique, événementiel et asynchrone alors qu'un processeur actuel fonctionne en états discrets (généralement binaires) et synchronisés sur une horloge globale, propre à chaque processeur. Les informations dans le cerveau circulent de façon locale entre neurones voisins et avec très peu de connexions "longue distance". Un processeur est plutôt une chaîne d'assemblage, où toute l'information circule dans un seul sens. La "mémoire vive" dans le cerveau est gérée localement, en modifiant la "chimie neuronale" avec des neurotransmetteurs, en modifiant les connexions synaptiques voire même avec de la neurogenèse. Pour une machine de von Neumann, la mémoire et le calcul sont complètement distincts, et un circuit *Full Adder* fera aussi bien de la cryptographie que du jeu vidéo ou la compilation

du code \LaTeX de ce manuscrit. Ces différences sont, de notre point de vue, fondamentales dans ce qui nous sépare des performances et de l'efficacité d'un cerveau humain.

Récemment, une nouvelle génération de puces appelées "neuromorphiques" a vu le jour chez les fabricants de processeurs. Notamment Loihi⁵ d'Intel, ou TrueNorth⁶ d'IBM. Ces circuits reprennent des propriétés cérébrales que nous avons évoquées, et comme attendu, la mise à l'échelle de ce genre d'architecture se fait aisément. Intel par exemple le démontre avec ses puces Loihi. Chacune d'entre elles comprend 131 072 neurones, et 10 fois plus de synapses. Une fois combinées par 32 sur une carte *Nahuku* on obtient un système neuromorphe à 4 millions de neurones. Combinez encore ces cartes dans un système plus grand et vous avez *Pohoiki Springs* et ses 100 millions de neurones distribués sur 768 puces. Une question reste cependant ouverte : comment utiliser efficacement toute cette puissance de calcul neuronal ?

1.1.2 L'émergence

La première idée pour exploiter efficacement des substrats de calculs neuromorphe serait de l'utiliser avec les outils que nous avons déjà. C'est-à-dire, combiner des algorithmes dans des programmes capables d'effectuer des tâches complexes comme jouer aux échecs, analyser une image ou piloter un avion. Cette approche fonctionne bien pour les ordinateurs classiques, car c'est relativement simple de combiner des algorithmes lorsque les opérations sont séquentielles et la mémoire un ensemble cohérent et interprétable. Il devient plus compliqué de programmer de cette façon des tâches parallèles, car il faut prendre en compte les temporalités de chaque tâche, ce qui ouvre la voie à de nombreux problèmes de blocages, d'incohérences et augmente significativement le nombre d'erreurs possibles. Le matériel neuromorphe lui est en comparaison proche d'être impossible à programmer. Ce que fait chaque neurone dépend non seulement de son état interne, mais aussi du millier de ses neurones voisins et de la temporalité des informations qu'il reçoit. Les chaînes de causalité et de rétroaction sont si complexes et avec de si nombreux éléments que la programmation d'un système neuromorphe à 100 millions de neurones pour produire un pilote automatique semble hors de portée pour l'intelligence humaine.

Le problème de la complexité n'est cependant pas unique au matériel neuromorphe. Un programmeur moderne n'a pas conscience de l'état de toutes les portes logiques dans un processeur à 50 milliards de transistors lorsqu'il écrit un programme. Mais il arrive à les coordonner pour leur faire prendre le rôle d'un joueur d'échecs ou de go. Cela est permis grâce à des langages de programmation de haut niveau d'abstraction, capables d'effectuer des tâches complexes en restant compréhensibles par un humain. Un tel langage n'existe pas encore pour les puces neuromorphiques. Copier les langages existants qui fonctionnent pour les ordinateurs classiques et les appliquer au neuromorphe copierait également leurs limites, et serait une utilisation inefficace de la puissance de calcul neuronal. Ainsi, il semble nécessaire de trouver des principes fondamentaux pouvant servir de base pour l'élaboration d'un langage de haut niveau qui permettrait d'organiser efficacement le calcul neuromorphe.

Nous pouvons de nouveau ici nous inspirer de ce que fait la biologie avec le prin-

5. <https://en.wikichip.org/wiki/intel/loihi>

6. <https://www.research.ibm.com/articles/brain-chip.shtml>

cipe d'émergence. L'émergence est un phénomène qui apparaît lorsqu'un ensemble d'éléments a des propriétés différentes que la somme de ses parties. L'émergence peut être observée à tous les niveaux dans la vie. Avec d'un côté les bancs de poissons ou nuées d'oiseaux qui sont le résultat d'un comportement simple au niveau de l'individu : rester proche des autres, aller dans la même direction, etc, et qui amène à une structure plus avancée au niveau du groupe : la nuée. Un phénomène similaire est à l'œuvre chez les fourmis, où par le dépôt de phéromones par chaque fourmi entre la fourmilière et la nourriture, les fourmis empruntent automatiquement le plus court chemin entre les deux. Les phéromones s'évaporent avec le temps, le chemin le plus court sera celui où les phéromones auront le moins de temps pour s'évaporer, et sera de fait route préférée des fourmis. L'émergence ne se limite cependant pas à l'échelle macroscopique entre individus, mais aussi dans le fonctionnement même d'un organisme.

Un être humain est composé en masse à 65% d'oxygène, 19% de carbone, 10% d'hydrogène, 3% d'azote, 2% de calcium, 1% de phosphore ainsi que d'autres éléments en plus petites quantités. Cependant, il ne suffit pas de verser tous ces éléments ensemble pour qu'un humain apparaisse. Même avec les bonnes molécules cela ne fonctionnerait pas. Pour avoir un humain, il faut que les molécules soient placées au bon endroit les unes par rapport aux autres. C'est à dire, qu'un humain est avant tout une certaine *organisation* de ces molécules. Cette description est valable pour l'ensemble du vivant, jusqu'aux plus petites bactéries et virus. L'étude de l'émergence est donc l'analyse de l'organisation de composants élémentaires pour leur faire adopter des comportements plus complexes. C'est un sujet de recherche pour de nombreux champs scientifiques autres que la biologie, comme la physique, l'économie et l'informatique avec entre autres, les systèmes complexes.

Cette idée d'émergence est cruciale dans le fonctionnement du cerveau, et par conséquent dans l'utilisation efficiente de nos processeurs neuromorphiques. Il n'existe pour l'instant pas de méthodes générales qui permettent de transformer n'importe quel ensemble de neurones artificiels en système capable de tâches de haut niveau. Nous ne savons pas non plus si une telle méthode générale pourrait exister.

Il existe des modèles en informatique qui ont pour ambition de faire le lien entre le niveau neuronal et des fonctions plus avancées avec leurs propriétés émergentes et auto-organisatrices. On pourrait considérer le premier pas comme étant l'apprentissage Hebbien. Présenté par Donald Hebb dans son livre *The organisation of behaviour* HEBB [1949] et qui introduit l'idée de l'apprentissage associatif, souvent résumé en « *Neurons that fire together, wire together* ». On peut également citer l'article de Dijkstra sur l'auto-stabilisation DIJKSTRA [1974]. Le sujet de cette publication est la tolérance aux fautes dans la programmation concurrente. Mais son concept est aussi applicable au domaine de l'émergence où l'on nomme l'auto-stabilisation l'homéostasie. Il s'agit de la capacité d'un système à se maintenir dans un état, comme par exemple la régulation de la température du corps humain. Ce principe d'auto-stabilisation est aussi très présent dans la théorie des champs neuronaux, introduite par AMARI [1977]. Les champs neuronaux essayent de définir les dynamiques d'activation dans des populations de neurones à un niveau mathématique. Selon la théorie, les champs neuronaux seraient l'élément fondamental pour produire des mécanismes cognitifs complexes SANDAMIRSKAYA [2014]. Enfin, nous avons les cartes auto-organisatrices KOHONEN [1982], qui sont un modèle de quantification vectorielle avec la particularité d'or-

ganiser automatiquement les neurones pour présenter une continuité dans ceux-ci, tels que des neurones voisins réagiront à des stimulus similaires.

Nous avons dans cette thèse utilisé principalement les cartes auto organisatrices (SOM) et les champs neuronaux dynamiques (DNF). Nous les avons choisis parce qu'ils sont très utilisés dans la littérature et appliqués à de nombreux domaines. Nous avons exploré leur utilisation dans la vision et évalué leur adéquation au rôle de couche d'abstraction pour des approches neuromorphiques. Ils seront présentés plus en détail dans les sections suivantes [1.2.1](#) et [1.2.4](#).

1.1.3 Particularités de la vision

La vision fait partie des sens les plus développés dans la nature ; presque l'intégralité des espèces vertébrées sont dotées d'yeux. Mais en plus d'être très utile, c'est également un sens très complexe. Fondamentalement, il consiste à recevoir des photons présents dans l'environnement et d'en déduire des informations sur celui-ci. Chaque photon arrive dans l'œil ou le capteur avec deux informations : sa longueur d'onde et sa position. En accumulant assez de photons sur un capteur, on peut déduire deux modalités de l'environnement. La luminosité, qui est le nombre de photons provenant d'un endroit particulier et la couleur qui est une combinaison de différentes longueurs d'ondes dans un spectre.

La vision ne se limite cependant pas à ces propriétés physiques. La lumière, qui est un ensemble de photons, provient d'une source (une ampoule, ou le soleil), et arrive ensuite soit directement, soit après une ou plusieurs réflexions dans l'œil ou le capteur. Ce sont ces réflexions, généralement la première, qui donnent le plus d'informations sur l'environnement. En effet, lorsque de la lumière reflète sur un objet, il y a une interaction entre les deux. L'objet absorbe une partie spécifique du spectre lumineux et reflète le reste. La lumière réfléchie contenant dorénavant de l'information sur la position de l'objet et de sa couleur. Par conséquent, les informations auxquelles nous avons accès grâce à la vision ne sont que partielles. L'objectif du traitement visuel est d'utiliser ces informations partielles pour créer une représentation de l'environnement cohérente.

Construire une représentation cohérente est cependant complexe. Le premier problème vient de la quantité de données à traiter. De la lumière arrive constamment à nos yeux ou capteurs photographiques, provenant de l'intégralité du champ visuel. Ce flux continu de données est multiplié par le nombre de pixels ou photorécepteurs. Ceux-ci devant être en très grand nombre pour couvrir un champ visuel assez large aussi bien horizontalement que verticalement, et pour lesquels une grande densité est nécessaire pour obtenir une précision visuelle suffisante. Par conséquent, le nombre d'informations à analyser par tout système de traitement visuel est conséquent.

Le second problème vient de la relation des différents pixels ou photorécepteurs entre eux. Un pixel pris en isolation ne donne que peu d'informations. Un pixel bleu peut par exemple provenir de la mer, du ciel, d'une voiture, d'un vêtement ou autre. Il est impossible de le savoir juste avec ce pixel, et il faut intégrer les informations contextuelles de voisinages, c'est à dire connaître les valeurs des pixels voisins, voire de l'image entière, pour pouvoir le lier à un objet. Parfois la temporalité est aussi nécessaire pour construire un environnement cohérent. Mais surtout, il faut une connais-

sance de ce que l'on peut trouver dans des images. Il faut savoir à quoi un objet, un arbre par exemple, correspond matériellement, sa forme physique 3D par exemple, et la rattacher à l'information visuelle qu'elle renvoie, son image, pour construire une représentation cohérente de l'environnement. Cette forme, ou matérialité de l'objet est inaccessible à partir d'une seule image, qui rappelons-le, ne contient que des informations partielles sur la réalité. Une mémoire est donc nécessaire pour tout traitement visuel complexe.

Tous ces facteurs rendent le traitement visuel un problème particulièrement ardu, qui mobilise un grand nombre de neurones dans le cerveau humain et de gros efforts de recherches en informatique. On oublie facilement cette difficulté car notre système visuel est assez développé pour résoudre aisément ces problèmes de façon inconsciente. La complexité cependant refait surface rapidement lorsque l'on change du format habituel dans lequel sont présentées les images. Différencier entre des chats et des chiens dans des photographies est facile pour un humain, mais faire la même tâche sur les transformées de Fourier de ces photographies est plus compliqué, comme illustré dans la figure 1.1.

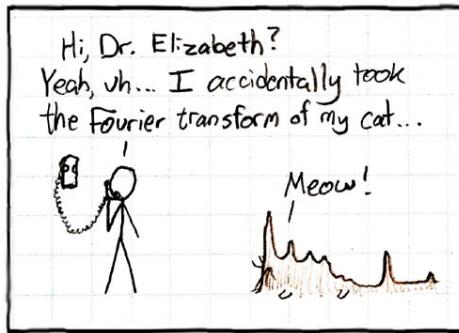


FIGURE 1.1 – Illustration de l'efficacité du traitement visuel humain. Sur la gauche, la personne et le téléphone sont aisément reconnaissables. Sur la droite cependant, sans les indices laissés par l'illustrateur, il serait impossible pour un humain de reconnaître un chat. Si le spectre affiché était effectivement la transformée de Fourier d'un chat.⁷

1.1.4 Vision humaine

Cette section a pour but de présenter brièvement les connaissances actuelles en neurosciences de la vision humaine et de faire le lien avec les modèles que nous avons choisi : les SOM et les DNF. Les informations présentées dans les paragraphes suivants proviennent majoritairement de GILBERT et DAS [2020].

Le système visuel humain commence à la rétine, point de départ du nerf optique pour traverser horizontalement le cerveau jusqu'au Corps Géniculé Latéral dans le Thalamus situé à peu près au centre du cerveau, pour finir dans le cortex visuel à l'arrière du cerveau. Les traitements visuels se font tout du long de cette chaîne, avec une gradation allant de traitements de bas niveau dans la rétine, jusqu'aux traitements complexes de haut niveau tel que la reconnaissance de visage dans le cortex visuel. Chaque traitement est fortement parallélisé et se fait dans un endroit spécifique, rendant le système visuel un amalgame de zones spécialisées. La stratégie "diviser pour

7. Source de l'image : <http://xkcd.com/26> reproduit sous licence CC-BY-NC

"conquérir" semble donc aussi utilisée dans le cerveau pour traiter des données complexes. Mais il existe aussi de nombreuses connections entre les zones, chacune influençant des zones voisines et réciproquement.

Dans les premières couches du cortex visuel (V1, V2, V3 et un peu au delà), les neurones sont organisés en carte rétinotopiques, c'est à dire en reproduisant l'arrangement spatial de la rétine. Des neurones voisins dans la carte traiteront des endroits proches dans champ visuel. Les DNF sont aussi rétinotopiques (plus de détails dans la section 1.2.4). Il y a également une organisation verticale en colonnes de neurones qui partagent des préférences proches comme la position ou l'orientation. C'est un concept qui se rapproche de l'idée derrière les cartes auto-organisatrices (section 1.2.1). Ainsi l'échange d'information dans le cortex se fait latéralement, verticalement et aussi parfois à longue distance.

Un autre fait intéressant est que la distribution de neurones qui traite chaque modalité tel que le contraste, l'orientation ou le mouvement correspond à la distribution de ces modalités dans l'environnement. L'implication est que le système visuel est dépendant de l'environnement. Il n'est pas connu si cette dépendance est génétique ou apprise pendant la vie. Mais elle montre bien qu'une connaissance de l'environnement habituel est utile pour les tâches visuelles.

Le système visuel ne travaille pas en isolation du reste du cerveau, et il existe de nombreuses connexions entre celui-ci et des autres aires cérébrales comme le Lobe Pariétal, qui intègre toutes les informations sensorielles, ou le lobe ventral qui s'occupe de la mémoire et de la reconnaissance d'objets par exemple. Il est estimé qu'environ 10% de ces connexions entre le système visuel et les autres aires vont dans ce sens, et que les 90% restants sont dans le sens inverse. Les niveaux supérieurs du cerveau ont donc ainsi une influence importante sur comment les informations visuelles sont traitées. La raison de cette disparité est encore inconnue, mais on peut supposer, d'après nos observations sur la complexité de la vision dans la section 1.1.3, que ce soit la connaissance préalable de l'environnement nécessaire pour compléter l'information partielle de la vision pour construire la représentation cohérente de l'environnement qui est transmise.

Un dernier mécanisme essentiel de la vision humaine est l'attention. C'est un mécanisme qui mobilise plusieurs aires cérébrales de différents endroits du cerveau. Son intérêt est de concentrer les traitements complexes de la vision à une petite partie seulement du champ visuel. Cela permet d'utiliser moins de ressources pour les traitements complexes, en rendant les stimulus plus simples et en réduisant la surcharge d'informations [EVANS et collab. \[2011\]](#). C'est aussi ce mécanisme qui contrôle en partie les mouvements oculaires pour utiliser efficacement la vision foveale (une plus grande précision au centre qu'aux bords) des yeux. Nous essayerons de reproduire ce mécanisme avec des DNF dans le chapitre 3. Cependant l'attention humaine utilise une représentation complexe de l'environnement, et d'autres mécanismes cognitifs avancés que nous ne maîtrisons pas encore. Nous utiliserons donc une version simplifiée du mécanisme.

1.1.5 Vision par ordinateur : Détection de nouveauté

Il existe de nombreuses fonctions dans la vision et il serait trop ambitieux de vouloir toutes les implémenter dans cette thèse. Ainsi nous avons choisi la détection de nouveauté comme tâche à résoudre. Elle est assez complexe et haut niveau pour ne pas être triviale comme le serait la détection d'orientations, et nécessite une compréhension de l'environnement pour séparer le fond de la nouveauté.

Les domaines de la détection de nouveauté et de la détection de changements ont été l'objet de nombreuses recherches et il existe un grand nombre de modèles dans la littérature qui effectuent cette tâche. Nous ne pourrons donc pas faire une présentation exhaustive du domaine, mais nous présenterons les méthodes les plus populaires, celles qui ont les meilleurs résultats et celles qui sont intéressantes pour notre approche.

Les approches historiques classiques on généralement consisté à appliquer aux images des variables aléatoires de densité (probability density functions en anglais) pour classifier les pixels. On peut citer par exemple les *Gaussian Mixture Models* (GMM) [ZIVKOVIC \[2004\]](#), ou plus récemment *ViBe* [BARNICH et VAN DROOGENBROECK \[2009\]](#). Certaines travaillent au niveau de groupes de pixels, comme les *Kernel Density Estimators* (KDE) [ELGAMMAL et collab. \[2000\]](#), ou combinent les deux approches [TOYAMA et collab. \[1999\]](#). Il existe aussi des méthodes différentes, par *Codebook* par exemple, comme dans [KIM et collab. \[2004\]](#). Ces approches classiques ont cependant toutes été supplantées récemment par des modèles fondées sur l'apprentissage.

Les méthodes les plus performantes actuellement utilisent des réseaux de neurones profonds [TEZCAN et collab. \[2021\]](#), [BOUWMANS et collab. \[2019\]](#). L'avantage de ceux-ci est la possibilité d'adapter pour la détection de changements des modèles ayant déjà fait leur preuves sur d'autres problèmes. L'utilisation de réseaux pré-entraînés permet aussi de réduire le besoin de grosses bases d'apprentissages. Les réseaux de neurones profonds étant supervisés par nature, la généralisation requiert cependant un entraînement spécial.

Une autre option est de combiner les deux, comme par exemple dans [BRAHAM et collab. \[2017\]](#) qui utilise un modèle de segmentation sémantique neuronal pour mieux représenter les objets en enlever le bruit de l'algorithme de détection de changements. [ZENG et collab. \[2019\]](#) combine les résultats de plusieurs algorithmes de détection de changements en les faisant passer par un réseau convolutionnel.

Il est aussi intéressant de noter que les cartes auto-organisatrices ont déjà été employées pour faire de la détection de nouveauté. Le modèle le plus connu est SOBS (Self-Organized Background Subtraction) [MADDALENA et PETROSINO \[2008\]](#). Il utilise une petite SOM pour chaque pixel pour apprendre les variations de ce pixel particulier au cours du temps et ainsi décider si la variation fait partie du fond ou si c'est de la nouveauté. Ces petites SOM sont toutes reliées dans un ensemble plus grand rétino-topique représentant le fond complet de l'image et permettant un peu de prendre en compte le voisinage de chaque pixel pour la décision. D'autres versions ont été développées plus tard pour améliorer les performances et la robustesse de SOBS [GEMIGNANI et ROZZA \[2016\]](#).

Comme la plupart des approches de détection de nouveauté, nous essayerons

d'apprendre un modèle du fond à l'initialisation et de faire la détection de nouveauté en comparant le modèle de fond et l'image courante reçue par la caméra. Nous devrons donc apprendre une image avec notre SOM, et utiliser ce que la SOM a appris pour trouver les nouveautés. Nous souhaitons explorer une approche différente de celle de SOBS, en ne travaillant pas au niveau des pixels individuels, mais de groupes de pixels. L'idée est de pouvoir intégrer plus d'informations de voisinage dans nos vecteurs d'entrée pour pouvoir effectuer un traitement de plus haut niveau avec les SOM. Nous présenterons ce concept dans le chapitre 2, consacré à la détection de nouveauté. Le chapitre 3 y adjoindra un mécanisme attentionnel décrit précédemment. Le chapitre 4 évoquera une amélioration de l'algorithme de la SOM pour en réduire le coût en calculs et applicable à notre méthode.

1.2 Réseaux neuronaux

Nous présenterons dans cette partie l'histoire et le fonctionnement des modèles neuronaux que nous avons utilisés. C'est à dire les cartes auto-organisatrices 1.2.1 et 1.2.2, les gaz neuronaux en expansion 1.2.3 et les champs neuronaux dynamiques 1.2.4.

1.2.1 Cartes auto organisatrices

Les cartes auto-organisatrices regroupent un ensemble de modèles proposé initialement par Teuvo Kohonen KOHONEN [1982]. Ces modèles sont caractérisés par leur capacité à projeter des données de façon ordonnée sur un espace d'une dimension plus faible (typiquement une ou deux dimensions). Cette réduction dimensionnelle donne ainsi une "carte" représentative des données qu'on lui a fournies, car les propriétés de voisinage sont conservées. Une des premières utilisations de ces cartes fut la représentation des phonèmes du finnois comme présenté dans la figure 1.2.

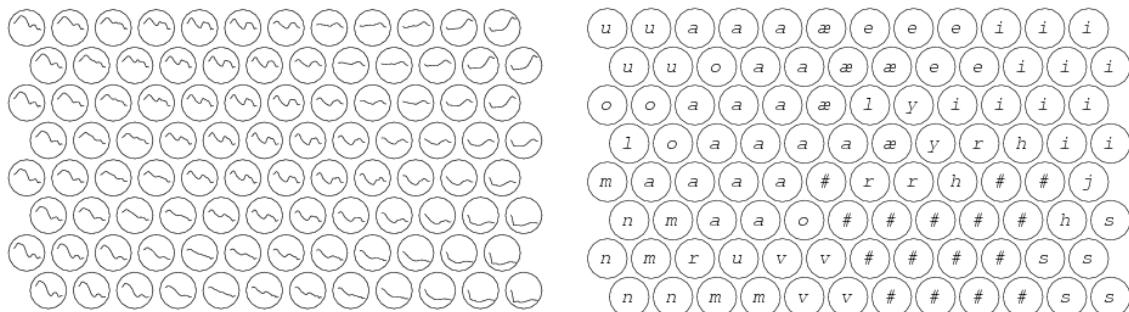


FIGURE 1.2 – Représentation des phonèmes du finnois par la première SOM. A gauche sont représentés les signaux sonores en haute dimension, et à droite leurs phonèmes correspondants. La réduction dimensionnelle provient de l'agencement de ces phonèmes sur la carte. Si ils sont proches entre eux dans leur espace d'entrée (signal), ils seront également proches dans la carte (la position des bulles) ⁸.

Le but premier de Kohonen était de présenter un modèle capable de représenter informatiquement l'organisation spatiale des informations dans le cortex humain KOHONEN [2012]. Il s'inspira pour cela d'un concept du domaine des neurosciences, les colonnes corticales, qui sont des groupes de neurones arrangés verticalement et qui réagissent tous au même stimulus.

Il y a eu de nombreuses évolutions pendant près de 40 ans d'existence des SOM. En 2002, une bibliographie recensait 5384 articles scientifiques utilisant les SOM OJA et collab. [2003]. Ils étaient estimés à plus de 10000 en 2011 PÖLLÄ et collab. [2011]. Les domaines d'applications sont très variés, allant du traitement du signal de l'analyse d'image et de la parole, jusqu'à la médecine, l'économie, la finance, l'urbanisme et d'autres encore. Pour chacun de ces domaines il y a plusieurs types d'utilisations de la SOM. Elle peut par exemple être utilisée en tant que méthode de visualisation capable de rendre humainement interprétable des données à très grande dimension en les projetant sur des dimensions plus petites. Mais aussi pour faire des traitements sur des

8. Source de l'image : http://scholarpedia.org/article/Kohonen_network licence CC-BY-NC

données, par exemple pour faire de la classification non supervisée de caractères, de chiffres ou de phonèmes, ou de la détection d'anomalies entre autres. COTTRELL et collab. [2018] présente une revue récente des différentes approches et applications des SOM.

1.2.2 Principes de fonctionnement

Préparation des données

Nous présentons dans cette section le fonctionnement de l'algorithme de la SOM que nous avons utilisé. Notre version est tout à fait classique et correspond à ce qui est communément utilisé dans la littérature.

Les données présentées à la SOM doivent être numériques et sous forme de vecteurs. La taille des vecteurs peut être aussi grande que nécessaire, mais toutes les données de la base doivent avoir la même taille de vecteur. Nous n'avons utilisé que des données normalisées, c'est à dire, dont la valeur est comprise entre 0 et 1 inclus. Par exemple pour apprendre des couleurs avec une SOM, on pourra représenter chaque couleur par un vecteur de taille 3, un élément par composante R,G et B par exemple et normalisée pour être comprise entre 0 et 1. L'ordre de présentation des vecteurs est aléatoire.

Paramètres

La forme de la SOM dépend de plusieurs paramètres. Le premier est la dimensionnalité. Les SOM peuvent aller d'une dimension de un à un nombre arbitrairement grand. Cependant, en pratique elles ne dépassent que rarement deux dimensions. La raison est que pour la visualisation de données par exemple, pouvoir afficher le résultat complet sur un écran est idéal. C'est aussi la bonne taille pour profiter de la réduction dimensionnelle sans pour autant augmenter de façon exponentielle les coûts en calculs à taille de carte égale. Une carte de 10 neurones de côté aura 100 neurones en deux dimensions et 1000 en 3 dimensions. Les coûts en calculs étant proportionnels au nombre de neurones, plus de dimensions, avec la même longueur de neurones n implique n fois plus de calculs. Nous avons ainsi utilisé exclusivement des cartes bidimensionnelles dans nos expériences. Dans notre cas, nous avons également pris en compte la contrainte matérielle qui rend toutes les dimensions supérieures à deux difficiles à implémenter efficacement en raison des coûts en connexions "longue distance" accrus, car les circuits intégrés sont naturellement en deux dimensions.

Un second paramètre important est ce que nous appelons la topologie de la SOM. Par topologie, nous entendons la forme des connexions entre les neurones qui composent la SOM. Les deux topologies les plus communes pour les SOM sont en grille et hexagonale. Dans la topologie en grille chaque neurone a quatre voisins, un à chaque direction cardinale. En hexagone, chaque neurone a 6 voisins, formant un pavage hexagonal avec les neurones au centre des hexagones (la figure 1.2 est par exemple hexagonale). Ces topologies sont bi-dimensionnelles, mais il est possible de les rendre toriques ou sphériques. Nous n'explorerons pas cette possibilité dans cette thèse, car cela apporte en général plus de contraintes topologiques, c'est plus diffi-

cile pour une sphère de bien modéliser l'espace d'entrée que pour une surface plane avec des degrés de libertés au extrémités. D'autres topologies plus exotiques existent et possèdent des propriétés intéressantes [BERNARD et collab. \[2018\]](#), cependant nous avons dû nous limiter aux topologies classiques, les différences entre les topologies des SOM n'étant pas notre objet d'étude ici.

Les autres hyper-paramètres de la SOM sont :

- La taille, communément notée n . Elle définit le nombre de neurones par côté de la SOM. Le nombre total de neurones N est obtenu à partir du carré des côtés : n^2 . Dans le cas d'une SOM non-carrée, on notera l et h respectivement sa largeur et sa hauteur.
- Le nombre d'époques, qui est un entier naturel définissant la durée de l'apprentissage.
- Le coefficient d'apprentissage α (alpha), défini dans $[0, 1]$. Il est décroissant linéairement tout au long de l'apprentissage. On notera dans cette thèse la valeur de départ et la valeur finale, toutes les valeurs intermédiaires seront extrapolées par la droite qui coupe ces deux points en fonction de l'époque courante de l'apprentissage.
- Le coefficient de voisinage σ (sigma), défini dans $[0, 1]$. Il sert à définir l'impact des neurones voisins sur les poids du neurone courant. σ est l'écart type du voisinage gaussien d'influence. Ainsi, plus il est élevé, plus la gaussienne sera étalée ce qui amènera à une contrainte topologique plus forte. Inversement, une valeur faible pour ce paramètre produit une gaussienne très centrée et réduit l'influence sur ses voisins de chaque neurone. Si placée à 0, elle enlève toute contrainte topologique et fait que la SOM se comporte comme un k-means. Il est important de noter aussi que nous utilisons une gaussienne normalisée. Cela implique que contrairement à une gaussienne standard, l'intégrale change en fonction de σ . Ainsi, plus σ sera grand, plus l'influence de chaque neurone sera plus grande sur tous les autres, et il n'y a pas d'occurrences où de réduction d'influence du fait de l'étalement de la gaussienne. Comme pour le coefficient d'apprentissage, il décroît linéairement pendant l'apprentissage et nous n'indiquerons que les valeurs de départ et de fin.

Apprentissage

Au début de l'apprentissage, tous les poids des neurones sont initialisés aléatoirement entre 0 et 1. L'apprentissage dure un certain nombre d'époques définies avant lancement. Une époque contient le nombre d'itérations requises pour que chaque élément de la base d'apprentissage soit utilisé exactement une seule fois. Lors d'une itération, on sélectionne aléatoirement un vecteur d'apprentissage parmi la base d'apprentissage, qui n'a pas déjà été utilisé lors de cette époque.

Une itération se déroule en deux étapes :

- La phase de recherche, qui consiste à trouver la *Best Matching Unit* (BMU) parmi tous les neurones. Elle correspond au neurone qui a la plus petite distance L^2 (distance euclidienne), entre ses poids et le vecteur d'apprentissage.

- La phase d’adaptation, qui modifie les poids des neurones selon l’équation suivante :

$$w_i(t+1) = w_i(t) + \alpha(t) \cdot \Theta(\sigma(t), d_{i,bmu}) \cdot (\nu - w_i(t)) \quad (1.1)$$

avec i le neurone courant, w_i les poids de ce neurone, t l’itération courante, ϵ et σ des paramètres de la SOM définis dans la section 1.2.2. $d_{i,bmu}$ est la distance L^1 (distance de manhattan) normalisée entre le neurone i et la BMU pour une SOM en grille. Θ est une fonction gaussienne centrée normalisée d’écart type σ . ν est le vecteur d’apprentissage.

On répète ces deux étapes jusqu’à ce que l’on finisse la dernière époque, et l’apprentissage sera terminé. Un exemple d’apprentissage est illustré sur la figure 1.3

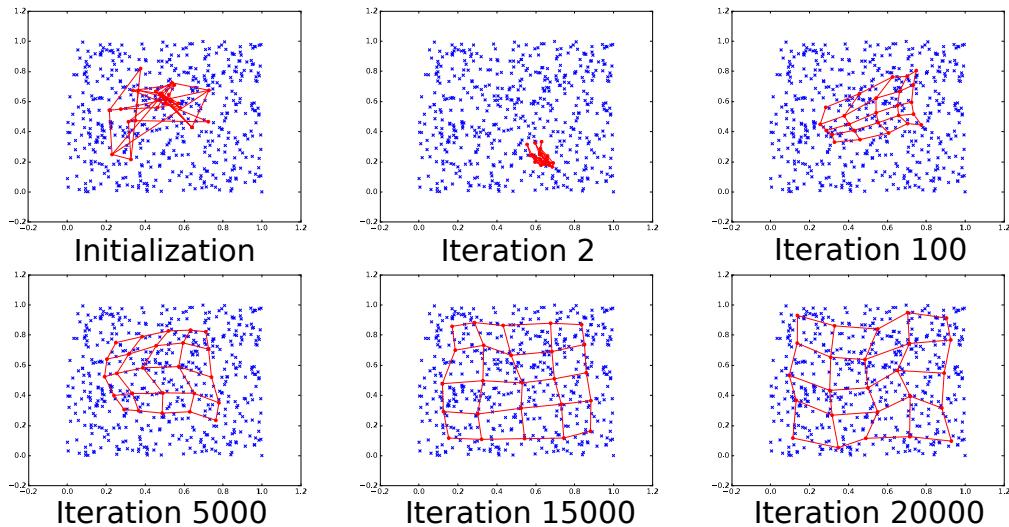


FIGURE 1.3 – Apprentissage d’une SOM. Les points bleus sont les vecteurs d’entrée en deux dimensions. Les point rouges les neurones de la SOM et les traits rouges représentent les connexions entre ceux-ci.

Reconstruction

On appelle reconstruction le fait de remplacer un ensemble de vecteurs, de longueur similaire à ceux sur laquelle la SOM a été apprise, par les vecteurs prototypes des neurones les plus proches.

Cette reconstruction est par exemple utilisée pour de la compression de données ETTAOUIL et LAZAAR [2012], car à la place de mémoriser un ensemble de vecteurs, il n’y a besoin que de mémoriser les vecteurs prototypes de la SOM et l’indice de chaque neurone le plus proche de chaque vecteur d’entrée de l’ensemble. Cette compression est avec perte, du fait que les vecteurs prototypes des neurones ne sont en général pas exactement les mêmes que les vecteurs présentés, même lorsque ceux-ci faisaient partie de la base d’apprentissage.

1.2.3 Gaz Neuronaux en Expansion

Bien que nos travaux se soient focalisés sur les cartes auto-organisatrices, notre approche se veut généraliste et transposable à d’autres modèles de quantification vec-

torielle avec topologie. Nous avons souhaité valider expérimentalement cette transposition en utilisant les Gaz neuronaux en expansion. C'est un autre modèle similaire aux SOM mais avec une approche tout à fait différente sur la topologie, qui devient dynamique et induite par les données d'apprentissage à la place de fixe et prédéterminée comme dans une SOM.

Développements

Une évolution majeure inspirée par les carte auto-organisatrices a été le développement des différents types de gaz neuronaux (Neural Gases, NG), qui a commencé avec [MARTINETZ et collab. \[1991\]](#), en tant qu'alternative aux *k-means* car ils ne disposent pas de topologie non plus. Puis ce sont les Gaz Neuronaux en Expansion (Growing Neural Gases, GNG) [FRITZKE \[1995\]](#) qui ont sensiblement amélioré l'approche en ajoutant un mécanisme de croissance qui ajoute des neurones lors de l'apprentissage et une topologie avec des connexions qui se créent et qui disparaissent entre les neurones.

De nos jours, plusieurs variantes des GNG existent qui améliorent certains aspects de cet algorithme. Notablement, *Growing when required* [MARSLAND et collab. \[2002\]](#) adapte la croissance du nombre de neurones en fonction de ce que le réseau a déjà appris, produisant une forte neurogenèse au début de l'apprentissage et une stabilisation une fois que les données ont été suffisamment apprises. Une autre variante sont les *incremental growing neural gases* [PRUDENT et ENNAJI \[2005\]](#). Ils permettent d'apprendre de nouvelles données sans oublier les anciennes en combinant plasticité et stabilité.

Pour la suite, nous avons décidé d'utiliser uniquement l'algorithme des Gaz neuronaux en expansion. C'est le plus utilisé et il est suffisant pour montrer que notre approche est généralisable.

Fonctionnement

Nous ne présenterons le fonctionnement que des gaz neuronaux en expansion. L'algorithme ne se réduisant pas aisément en quelques formules, nous prendrons une approche itérative, similaire à celle que l'on peut trouver dans l'article original, pour expliquer les différents mécanismes à l'oeuvre. Il est également illustré dans la figure 1.4.

Une itération se décompose en 9 étapes :

1. Choisir au hasard un élément de la base d'apprentissage parmi ceux qui n'ont pas déjà été tirés lors de cette époque.
2. Trouver les deux neurones les plus proches : s_1 et s_2 .
3. Incrémenter l'âge des synapses de tous les voisins directs de s_1 .
4. Ajouter la distance euclidienne au carré entre le vecteur d'apprentissage et s_1 à la variable d'erreur de s_1 .
5. Mettre à jour les poids de s_1 et de tous ces voisins directs s_j . Les formules sont :

$$w_{s_1}(t+1) = w_{s_1}(t) + \epsilon_{bmu} \times (\nu - w_{s_1}(t)) \quad (1.2)$$

$$w_{s_j}(t+1) = w_{s_j}(t) + \epsilon_j \times (\nu - w_{s_j}(t)) \quad (1.3)$$

6. Si s_1 et s_2 sont voisins directs, mettre à jour l'âge de la synapse à 0. Sinon créer une synapse.
 7. Enlever toutes les synapses avec un âge supérieur à a_{max} . Si des neurones se retrouvent sans synapses, les enlever aussi.
 8. Si l'itération courante est un multiple de λ , insérer un nouveau neurone comme suit :
 - Trouver le neurone avec l'erreur la plus élevée q .
 - Créer un nouveau neurone r à distance égale de q et de son voisin direct avec l'erreur la plus élevée f .
- $$w_r(t+1) = \frac{w_q(t) + w_f(t)}{2} \quad (1.4)$$
- Ajouter des synapses d'âge 0 entre r et q ainsi qu'entre r et f . Enlever la synapse entre q et f .
 - Réduire l'erreur de q et f en la multipliant par une constante α . Initialiser l'erreur de r avec la nouvelle valeur de l'erreur de q .
9. Réduire toutes les variables d'erreur en les multipliant par une constante d .

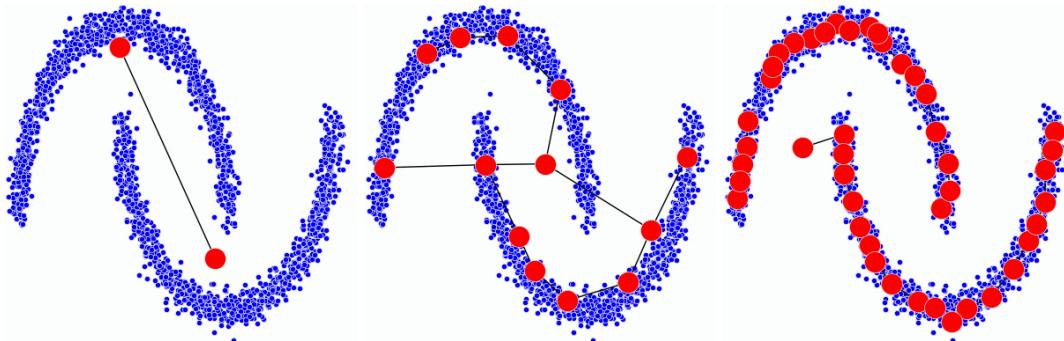


FIGURE 1.4 – Apprentissage d'un GNG de gauche à droite. Les neurones en rouge dans chaque croissant de lune sont connectés entre eux, mais il n'y a pas de connexions entre les neurones des deux croissants de lunes. La topologie du GNG représente la topologie présente dans les données.⁹

L'apprentissage s'arrête au bout d'un certain nombre prédéfini d'époques, comme pour la SOM. L'effet de chaque paramètre peut être compris aisément par le contexte dans lequel il est utilisé, ainsi nous n'irons pas dans le détail de chacun d'entre eux. Le paramètre λ , ajustant la vitesse de création de nouveaux neurones, sera fixé de telle sorte qu'à la fin de l'apprentissage il y ait le même nombre de neurones dans le GNG que dans la SOM à laquelle on souhaite se comparer. La reconstruction se passe de la même façon que pour la SOM. Les valeurs que l'on aura utilisées pour nos paramètres seront précisées dans la section expérimentale correspondante.

1.2.4 DNF

Les champs neuronaux dynamiques (DNF), introduits par AMARI [1977] représentent mathématiquement l'évolution d'une population de neurones. Une dynamique « On

9. Images provenant de <https://github.com/AdrienGuille/GrowingNeuralGas>, licence MIT

center, Off surround » a été ajoutée par ELLIAS et GROSSBERG [1975], qui ont employé des neurones inhibiteurs et excitateurs dans la forme typique de "chapeau mexicain" d'activations (un centre excitateur, un intermédiaire inhibiteur et des bords avec peu d'effet).

Les DNF ont été l'objets de nombreux travaux, notamment dans la vision avec FIX et collab. [2011] qui met en œuvre une recherche séquentielle dans un environnement (attention overt). Mais aussi pour du suivi de cibles comme par exemple dans MARTEL et SANDAMIRSKAYA [2016].

Selon leur paramétrage, les DNF peuvent prendre de nombreux rôles. Ils peuvent faire de la sélection, de la réduction de bruit, de la mémoire grâce à leur adaptabilité et leur mécanisme de *soft winner-takes-all*. C'est pourquoi ils sont étudiés en tant que modèle de base pour en faire émerger des mécanismes cognitifs complexes SANDAMIRSKAYA [2014].

Fonctionnement

Comme pour les SOM, les DNF peuvent être déclinés en plusieurs dimensions. Nous n'utiliserons que des DNF d'une ou deux dimensions dans cette thèse. Les neurones d'un DNF sont connectés de façon rétinotopique aux entrées, par conséquent, la taille et le nombre de neurones est définie par la taille de la carte d'entrée. Chaque neurone est également connecté à tous les autres neurones. Enfin, tous les neurones possèdent un potentiel, qui est représenté par un nombre réel. Ce potentiel est noté $u(x, t)$, x étant la position du neurone dans le champ et t un instant de la simulation. La valeur du potentiel est définie par l'équation différentielle suivante :

$$\tau \frac{\partial u(x, t)}{\partial t} = -u(x, t) + h + \int f(u(x', t))\omega(x - x')\delta x' + S(x, t)g - gi \quad (1.5)$$

$$\omega(d) = c_{exc} \exp\left[-\frac{d^2}{2\sigma_{exc}^2}\right] - c_{inh} \exp\left[-\frac{d^2}{2\sigma_{inh}^2}\right] \quad (1.6)$$

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1.7)$$

- τ est une constante qui règle la vitesse d'adaptation du DNF.
- $-u(x, t)$ est le terme de fuite. Son rôle est d'inhiber des neurones activés lorsqu'il n'y a plus d'activations par l'entrée ou par d'autres neurones.
- h est la valeur de repos du neurone, qui doit être négative.
- $f(u(x, t))$ est une fonction logistique. Elle représente l'activation du neurone x au temps t . Il s'agit souvent de cette valeur que l'on utilise en sortie du DNF.
- $\omega(x - x')$ est l'interaction latérale. Il représente l'effet d'autres neurones sur le potentiel de ce neurone. Nous utilisons une différence de gaussiennes. La gaussienne la plus pointue est excitatrice et la gaussienne plus allongée est inhibitrice. Cela a pour effet que les neurones proches sont excitateurs et les neurones lointains inhibiteurs. On appelle également les neurones ayant un impact

excitateur ou inhibiteurs des neurones afférents. La formule est détaillée dans l'équation 1.6, avec les quatre paramètres associés.

- $S(x, t)$ est la valeur au temps t de l'entrée à la position x .
- g est le gain, et est une constante réelle utilisée pour faire varier l'effet de l'entrée.
- gi est l'inhibition globale. C'est une constante divisée par le nombre de neurones présent dans la carte.

Par souci de simplicité et de calculabilité, nous avons implémenté une version spatialement et temporellement discrétisée de l'équation 1.5. Elle est obtenue en manipulant les potentiels d'un ensemble discret de neurones (carte neuronale au lieu de champ neuronal) et en utilisant une simple méthode d'Euler pour estimer l'état de $u(x, t + \Delta t)$ en connaissant $u(x, t)$:

$$u(x, t + \Delta t) = u(x, t) + \frac{\Delta t}{\tau} (-u(x, t) + h + \sum f(u(x', t))\omega(x - x') + S(x, t + \Delta t)g - gi) \quad (1.8)$$

Δt est le pas de temps entre deux estimations, il peut être le même pour tous les neurones (synchrone) ou différent à chaque fois (asynchrone).

Il est souvent ardu de comprendre le comportement d'un DNF à partir de sa formule uniquement, ainsi nous préciserons les paramètres et le comportement que nous avons choisi pour chaque DNF que nous avons utilisé aux endroits appropriés. Les changements dans les paramètres étant ce qui produit les différents comportements des DNF, ils sont par conséquent difficiles à paramétriser.

1.3 Références

- AMARI, S.-I. 1977, «Dynamics of pattern formation in lateral-inhibition type neural fields», *Biological cybernetics*, vol. 27, n° 2, p. 77–87. [6](#) [17](#)
- BARNICH, O. et M. VAN DROOGENBROECK. 2009, «Vibe : a powerful random technique to estimate the background in video sequences», dans *2009 IEEE international conference on acoustics, speech and signal processing*, IEEE, p. 945–948. [10](#)
- BERNARD, Y., E. BUOY, A. FOIS et B. GIRAU. 2018, «Np-som : network programmable self-organizing maps», dans *2018 IEEE 30th international conference on tools with artificial intelligence (ICTAI)*, IEEE, p. 908–915. [14](#)
- BOUWMANS, T., S. JAVED, M. SULTANA et S. K. JUNG. 2019, «Deep neural network concepts for background subtraction : A systematic review and comparative evaluation», *Neural Networks*, vol. 117, p. 8–66. [10](#)
- BRAHAM, M., S. PIERARD et M. VAN DROOGENBROECK. 2017, «Semantic background subtraction», dans *2017 IEEE International Conference on Image Processing (ICIP)*, IEEE, p. 4552–4556. [10](#)
- BROWN, T. B., B. MANN, N. RYDER, M. SUBBIAH, J. KAPLAN, P. DHARIWAL, A. NEELAKANTAN, P. SHYAM, G. SAstry, A. ASKELL et collab.. 2020, «Language models are few-shot learners», *arXiv preprint arXiv:2005.14165*. [2](#)

- CAMPBELL, M., A. J. HOANE JR et F.-H. Hsu. 2002, «Deep blue», *Artificial intelligence*, vol. 134, n° 1-2, p. 57–83. [2](#)
- COTTRELL, M., M. OLTEANU, F. ROSSI et N. VILLA-VIALANEIX. 2018, «Self-organizing maps, theory and applications», *Revista de Investigacion Operacional*, vol. 39, n° 1, p. 1–22. [13](#)
- DIJKSTRA, E. W. 1974, «Self-stabilization in spite of distributed control», dans *Selected writings on computing : a personal perspective*, Springer, p. 41–46. [6](#)
- ELGAMMAL, A., D. HARWOOD et L. DAVIS. 2000, «Non-parametric model for background subtraction», dans *European conference on computer vision*, Springer, p. 751–767. [10](#)
- ELLIAS, S. A. et S. GROSSBERG. 1975, «Pattern formation, contrast control, and oscillations in the short term memory of shunting on-center off-surround networks», *Biological Cybernetics*, vol. 20, n° 2, p. 69–98. [18](#)
- ETTAOUIL, M. et M. LAZAR. 2012, «Improved self-organizing maps and speech compression», *International Journal of Computer Science Issues (IJCSI)*, vol. 9, n° 2, p. 197. [15](#)
- EVANS, K. K., T. S. HOROWITZ, P. HOWE, R. PEDERSINI, E. REIJNEN, Y. PINTO, Y. KUZMOVA et J. M. WOLFE. 2011, «Visual attention», *Wiley Interdisciplinary Reviews : Cognitive Science*, vol. 2, n° 5, p. 503–514. [9](#)
- FERRUCCI, D. A. 2012, «Introduction to “this is watson”», *IBM Journal of Research and Development*, vol. 56, n° 3.4, p. 1–1. [2](#)
- FIX, J., N. ROUGIER et F. ALEXANDRE. 2011, «A dynamic neural field approach to the covert and overt deployment of spatial attention», *Cognitive Computation*, vol. 3, n° 1, p. 279–293. [18](#)
- FRITZKE, B. 1995, «A growing neural gas network learns topologies», *Advances in neural information processing systems*, vol. 7, p. 625–632. [16](#)
- GEMIGNANI, G. et A. ROZZA. 2016, «A robust approach for the background subtraction based on multi-layered self-organizing maps», *IEEE transactions on image processing*, vol. 25, n° 11, p. 5239–5251. [10](#)
- GILBERT, C. et A. DAS. 2020, «The constructive nature of visual processing», *Principles of Neural Science*, p. 556–576. [8](#)
- HART, B. et T. R. RISLEY. 2003, «The early catastrophe.», *Education review*, vol. 17, n° 1. [3](#)
- HEBB, D. O. 1949, *The organisation of behaviour : a neuropsychological theory*, Science Editions New York. [6](#)
- KANDEL, E. R., J. H. SCHWARTZ, T. M. JESSELL, S. SIEGELBAUM, A. J. HUDSPETH et S. MACK. 2000, *Principles of neural science*, vol. 4, McGraw-hill New York. [3](#)

- KIM, K., T. H. CHALIDABHONGSE, D. HARWOOD et L. DAVIS. 2004, «Background modeling and subtraction by codebook construction», dans *2004 International Conference on Image Processing, 2004. ICIP'04.*, vol. 5, IEEE, p. 3061–3064. [10](#)
- KOHONEN, T. 1982, «Self-organized formation of topologically correct feature maps», *Biological cybernetics*, vol. 43, n° 1, p. 59–69. [6, 12](#)
- KOHONEN, T. 2012, *Self-organization and associative memory*, vol. 8, Springer Science & Business Media. [12](#)
- KRIZHEVSKY, A., I. SUTSKEVER et G. E. HINTON. 2012, «Imagenet classification with deep convolutional neural networks», *Advances in neural information processing systems*, vol. 25, p. 1097–1105. [2](#)
- LECUN, Y., B. BOSER, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. HUBBARD et L. D. JACKEL. 1989, «Backpropagation applied to handwritten zip code recognition», *Neural computation*, vol. 1, n° 4, p. 541–551. [2](#)
- MADDALENA, L. et A. PETROSINO. 2008, «A self-organizing approach to background subtraction for visual surveillance applications», *IEEE Transactions on Image Processing*, vol. 17, n° 7, p. 1168–1177. [10](#)
- MARSLAND, S., J. SHAPIRO et U. NEHMZOW. 2002, «A self-organising network that grows when required», *Neural networks*, vol. 15, n° 8-9, p. 1041–1058. [16](#)
- MARTEL, J. N. et Y. SANDAMIRSKAYA. 2016, «A neuromorphic approach for tracking using dynamic neural fields on a programmable vision-chip», dans *Proceedings of the 10th International Conference on Distributed Smart Camera*, p. 148–154. [18](#)
- MARTINETZ, T., K. SCHULTEN et collab.. 1991, «A "neural-gas" network learns topologies», . [16](#)
- MCCARTHY, J., M. MINSKY et N. ROCHESTER. 1955, «A proposal for the dartmouth summer research project on artificial intelligence», . [2](#)
- OJA, M., S. KASKI et T. KOHONEN. 2003, «Bibliography of self-organizing map (som) papers : 1998–2001 addendum», *Neural computing surveys*, vol. 3, n° 1, p. 1–156. [12](#)
- PRUDENT, Y. et A. ENNAJI. 2005, «An incremental growing neural gas learns topologies», dans *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, IEEE, p. 1211–1216. [16](#)
- PÖLLÄ, M., T. HONKELA et T. KOHONEN. 2011, «Bibliography of self-organizing maps», URL <http://www.cis.hut.fi/research/refs/>. [12](#)
- SANDAMIRSKAYA, Y. 2014, «Dynamic neural fields as a step toward cognitive neuromorphic architectures», *Frontiers in neuroscience*, vol. 7, p. 276. [6, 18](#)
- SILVER, D., A. HUANG, C. J. MADDISON, A. GUEZ, L. SIFRE, G. VAN DEN DRIESSCHE, J. SCHRITTWIESER, I. ANTONOGLOU, V. PANNEERSHELVAM, M. LANCTOT et collab.. 2016, «Mastering the game of go with deep neural networks and tree search», *nature*, vol. 529, n° 7587, p. 484–489. [2](#)

- TEZCAN, M. O., P. ISHWAR et J. KONRAD. 2021, «Bsuv-net 2.0 : Spatio-temporal data augmentations for video-agnostic supervised background subtraction», *IEEE Access*, vol. 9, p. 53 849–53 860. [10](#)
- TOYAMA, K., J. KRUMM, B. BRUMITT et B. MEYERS. 1999, «Wallflower : Principles and practice of background maintenance», dans *Proceedings of the seventh IEEE international conference on computer vision*, vol. 1, IEEE, p. 255–261. [10](#)
- TURING, A. M. 1950, «Computing machinery and intelligence», dans *Parsing the turing test*, Springer, p. 23–65. [2](#)
- VON NEUMANN, J. 1945, «First draft of a report on the edvac», *IEEE Annals of the History of Computing*, vol. 15, n° 4, p. 27–75. [3](#)
- ZENG, D., M. ZHU et A. KUIJPER. 2019, «Combining background subtraction algorithms with convolutional neural network», *Journal of Electronic Imaging*, vol. 28, n° 1, p. 013 011. [10](#)
- ZIVKOVIC, Z. 2004, «Improved adaptive gaussian mixture model for background subtraction», dans *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, IEEE, p. 28–31. [10](#)

Chapitre 2

Détection de Nouveauté

« *D'abord, j'observe les êtres humains car je les aime bien. J'enregistre dans ma tête tout ce que j'ai remarqué, et ensuite, avec les souvenirs de ce que j'ai vu, je dessine.* »

Hayao Miyazaki

Sommaire

2.1	Introduction	24
2.2	Application aux images	24
2.2.1	Apprentissage et reconstruction	25
2.2.2	La gestion des canaux (couleurs)	27
2.3	Détection de nouveauté par QVAT : quantification vectorielle et apprentissage topologique	29
2.3.1	Détection avec quantification vectorielle	29
2.3.2	Détection avec distance neurale	30
2.3.3	Considérations pour la combinaison	31
2.4	Protocole expérimental	34
2.4.1	Présentation de la base de données	34
2.4.2	Métriques utilisées	38
2.4.3	Préparation des données	41
2.4.4	Paramétrages des modèles	43
2.5	Résultats expérimentaux	49
2.5.1	Seuil de décision	49
2.5.2	Nombre de neurones et taille des imagettes	49
2.5.3	Résultats complets	54
2.5.4	Visualisation des résultats	57
2.5.5	Interprétations	59
2.6	Références	60

2.1 Introduction

Ce chapitre présente notre méthode pour faire de la détection de nouveauté avec les cartes auto-organisatrices (SOM). Ou plus généralement, avec des modèles de quantification vectorielle avec topologie.

Nous commencerons par montrer différentes façons d'apprendre les images avec des SOM et celle que nous avons choisie. Puis nous introduirons notre méthode de détection de nouveauté en deux parties. D'abord avec quantification vectorielle, et une seconde avec la topologie. Ensuite nous aborderons le protocole expérimental que nous avons mis en œuvre pour évaluer notre modèle. La dernière section présente nos résultats.

2.2 Application aux images

Il existe de nombreuses possibilités différentes pour apprendre des images avec la quantification vectorielle, dû au grand nombre de façons de découper une image en vecteurs d'apprentissage. Pour présenter l'impact que peut avoir ce découpage, on peut considérer deux extrêmes : apprendre l'image en entier ou apprendre chaque pixel individuellement. Le premier cas peut sembler absurde dans le cas d'une seule image (une base de données d'un seul élément), mais peut présenter un intérêt lorsque l'on considère une suite d'images par exemple. Dans cet exemple, l'environnement appris serait défini par l'entièreté de ce que voit le capteur et tous les changements, où qu'ils soient dans l'image, auraient de l'importance.

Le second extrême est l'apprentissage au niveau du pixel. Dans cette approche on prend chaque pixel individuel comme un vecteur d'entrée, ce qui rendrait l'espace d'entrée unidimensionnel pour une image en niveau de gris (ou tridimensionnel si l'image est en couleur, plus de détails dans la section [2.2.2](#)). Sur un plan conceptuel, ce choix considère qu'une image est définie uniquement par les couleurs ou luminosités présentes, peu importe leur positions dans celle-ci. C'est utilisé notamment dans SOBS [MADDALENA et PETROSINO \[2008\]](#), et tous ses descendants. Il existe un très grand nombre d'autres découpages possibles entre ces deux extrêmes, chacun représentant une certaine façon de conceptualiser une image et définissant la notion de nouveauté. Celle-ci étant un changement à un quelconque endroit de l'image dans le premier cas, et l'apparition d'une nouvelle couleur dans l'image dans le second.

Il est important donc de considérer le contexte de nos travaux pour définir la façon de représenter une image, notre application étant la détection de nouveauté. Dans ce contexte, nous considérons qu'une image est une combinaison de nombreux éléments plus petits. Par exemple, une photographie d'un lac de montagne [2.1](#) peut être décrite comme étant la combinaison d'un élément de lac (avec sa couleur, bleu sombre et sa texture uniforme), d'un élément de plaine herbeuse (verte et uniforme), d'éléments rocaillieux qui sont gris et soit uniformes (dans le premier plan) soit plus contrastés en se combinant avec la verdure de la végétation (dans les bords de l'image), et ainsi de suite. Ce découpage "sémantique" de l'image est ce qui permet la détection de nouveauté de fonctionner, car dans notre cas la nouveauté est par définition ce qui n'est pas déjà dans l'image et donc ne faisant pas partie de ces classes d'éléments. Pour regrouper les parties d'une image appartenant au même élément, il est

nécessaire d'avoir une information mise en contexte (un pixel seul ne suffit généralement pas à savoir à quel élément il appartient dans l'image), ce qui implique que les pixels doivent être pris dans leurs environnements locaux pour conserver l'information de voisinage comme la texture par exemple. Nous avons donc choisi de découper l'image en plus petites images à la façon d'une mosaïque, que nous appellerons des imagettes. Ces imagettes (d'une taille arbitraire en hauteur et largeur) conservent l'environnement local tout en étant suffisamment petites pour être précises dans l'espace. Une imagette ne représentera qu'une partie d'un élément et non pas regroupant plusieurs éléments, ce qui réduirait sa capacité de généralisation. Mais elles permettent aussi d'avoir une base d'apprentissage assez étoffée pour tirer parti des propriétés de nos modèles de quantification vectorielle et de leur topologie. La partie pratique de l'apprentissage d'une image, sa représentation et sa reconstruction sont abordés dans la section suivante 2.2.1.

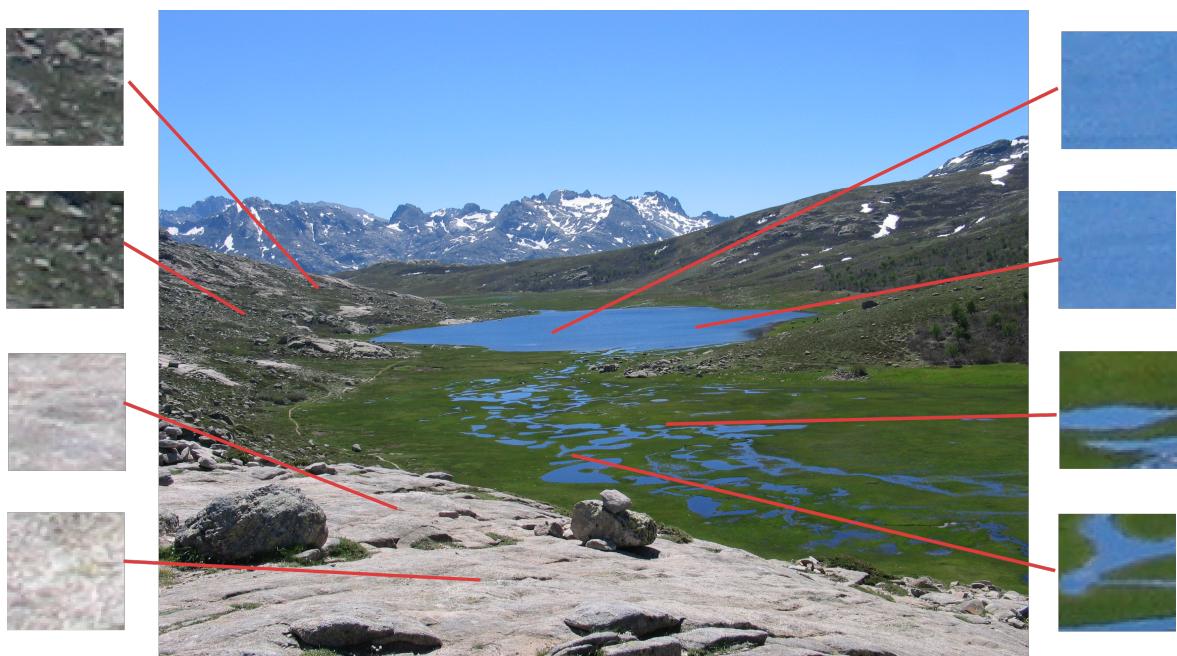


FIGURE 2.1 – Exemple d'image comportant plusieurs éléments notables tels qu'un lac (bleu sombre et uniforme), une plaine herbeuse (verte et uniforme), d'éléments rocheux qui sont gris et soit uniformes (dans le premier plan) soit plus contrastés en se combinant avec la verdure de la végétation (dans les bords de l'image), et ainsi de suite. Une image peut donc être vue comme un ensemble de zones différentes composées d'éléments proches entre eux.¹

2.2.1 Apprentissage et reconstruction

Apprentissage

La première étape nécessaire à l'apprentissage est la constitution de la base d'apprentissage à partir de l'image. L'utilisation de modèles de quantification vectorielle établissent la première contrainte pour le découpage : les imagettes doivent être d'une taille fixe. En effet il est nécessaire pour les SOM comme pour les GNG et leurs variantes que tous les vecteurs d'entrées soient de la même longueur pour que le calcul

1. Image source : Lac de Nino, https://fr.wikipedia.org/wiki/Lac_de_Nino#/media/Fichier:Lac_de_Nino.jpg, CC BY-SA

de distance avec les neurones fonctionne, pour qu'ils représentent le plus fidèlement possible les entrées qui leurs sont attachées.

Nous avons également choisi de limiter nos tailles d'imagettes à des carrés, avec la hauteur égale à la longueur, pour des raisons de simplicité. Il est possible que des imagettes plus larges que hautes ou plus hautes que larges représentent mieux les éléments de l'image que l'on apprend. Cependant ce serait une préférence spécifique à chaque image et peu généralisable, car si on effectue par exemple une rotation de 90° de l'image, la préférence s'inversera. L'inversement des tailles dans les imagettes carrées ne changeant rien, elles sont pour leur part insensibles aux rotations discrètes de l'image (par pas de 90°).

Dans la version classique AMERIJCKX et collab. [2003], le découpage de l'image se fait en mosaïque, sans superpositions entre les imagettes. C'est à dire que chaque pixel n'appartient qu'à une seule imagette. Le processus est montré sur la figure 2.2 Si les dimensions de l'images ne sont pas un multiple de la taille des imagettes, les pixels en trop sont rognés par la droite et par le bas. Les bords d'une image ne contenant en général que peu d'informations.

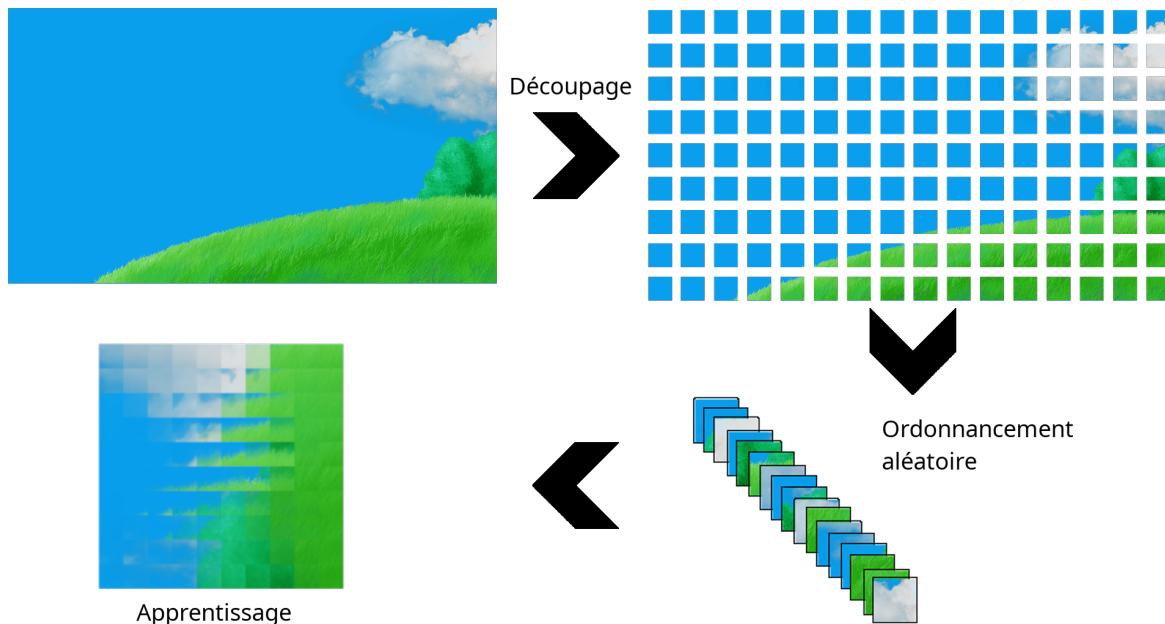


FIGURE 2.2 – Illustration du processus de représentation et d'apprentissage d'une image par une SOM.

Reconstruction

Une fois l'apprentissage terminé, on obtient deux sorties. La première est le modèle entraîné avec les différents poids des neurones codant une imagette représentante du cluster d'imagettes associé à ce neurone. La seconde est la liste, pour chaque imagette de l'image, de l'indice du neurone le plus proche de celle-ci, qui pourra être utilisé pour la reconstruction de l'image d'apprentissage. Le procédé est montré sur la figure 2.3.

Pour reconstruire une image à partir de la liste des indices de BMU, il suffit de

remplacer chaque indice par le vecteur prototype du neurone auquel il correspond. Ces vecteurs prototypes devront être réassemblés en imagettes (dans un tableau à 2 dimensions à la place d'un vecteur à une dimension), lesquels seront placés à la bonne position pour reformer l'image.

Il est aussi possible de reconstruire une image qui n'a pas été apprise. Pour cela il faut créer la liste d'indices des BMU des imagettes de la nouvelle image, et de reconstruire ensuite l'image par le même procédé que montré précédemment. Il faut noter que l'image que l'on souhaite reconstituer doit être proche de l'image apprise, c'est à dire contenir des éléments similaires, pour obtenir un résultat correct.

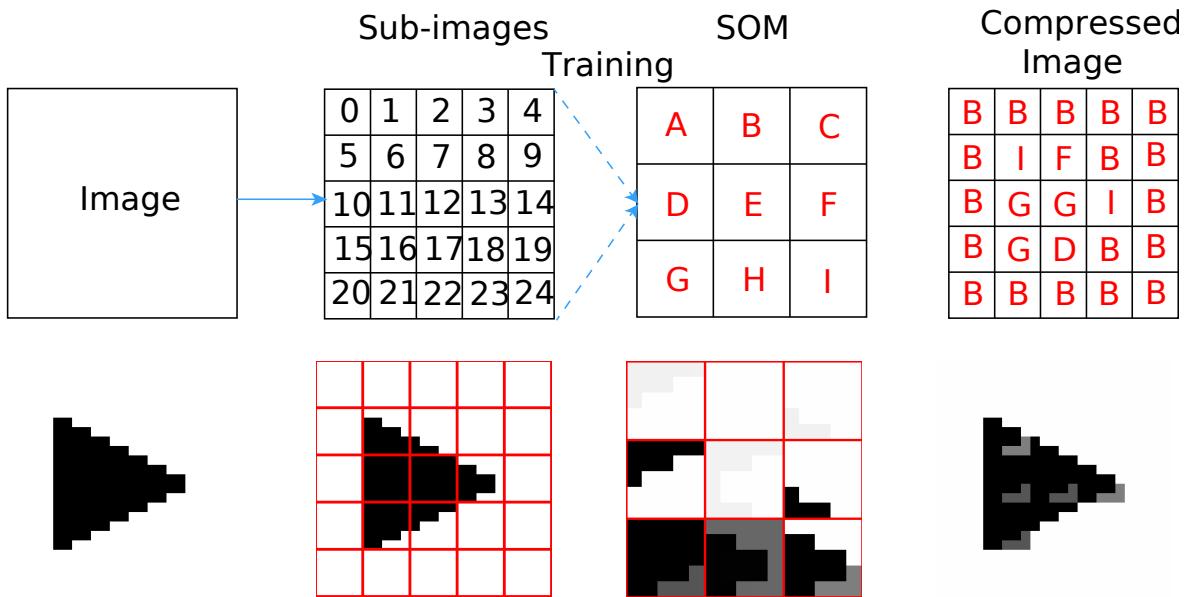


FIGURE 2.3 – Schéma simplifié du processus de compression et de reconstruction d'une image, avec ici seulement 9 neurones et 25 imagettes.

2.2.2 La gestion des canaux (couleurs)

Nous avons jusque là vu comment apprendre une image ayant une seule valeur par pixel (soit des images en nuances de gris). Cependant la majorité des dispositifs de capture actuels fournissent des images en couleurs, c'est à dire avec trois canaux. Nous allons voir dans cette section comment transposer l'apprentissage, la compression et la reconstruction à des images à un nombre arbitraire de canaux par pixels.

Une approche possible serait de séparer l'image par composante. Une image RVB par exemple donnerait trois images en niveau de gris, une R, une V et une B. Deux options s'offrent ensuite à nous. Soit utiliser une seule SOM pour apprendre toutes les images ainsi extraites en espérant que les différentes formes présentes dans chaque composante soient assez similaires entre elles. Cela augmente aussi les données que la SOM doit apprendre, car on vient de multiplier la taille de notre base d'apprentissage par le nombre de canaux. Ces données doivent être aussi cohérentes dans l'espace d'entrée pour que la réduction dimensionnelle se fasse correctement. Soit utiliser une SOM par composante pour apprendre chaque canal séparément, et de regrouper ensuite les différents canaux reconstitués en une image couleur.

Cependant ces deux approches ont un défaut majeur pour la compression d'images (ainsi que la détection de nouveauté en conséquence), c'est la création d'aberrations chromatiques dans l'image reconstituée. En effet, les canaux étant appris séparément avant d'être recombinés, la reconstruction peut donner pour certains pixels des teintes de couleurs qui n'existaient pas dans l'image de base, qui peuvent être très saillants visuellement. Par exemple un pixel blanc dans l'image d'entrée (avec une forte composante R, V et B), lorsque reconstitué par la SOM peut être bien reconstitué dans deux composantes (V et B par exemple), et mal reconstitué dans la troisième (R) avec une valeur beaucoup plus faible que dans l'image de base, car on minimise la distance entre les canaux séparément. Par conséquent ce pixel aura une couleur turquoise dans l'image reconstituée à la place de blanc. Cette erreur minimise bien la distance avec l'image de base, et ce n'est que visuellement que les changements de teintes sont apparentes et dégradent proportionnellement plus la qualité de l'image que que l'erreur mesurée.

Une meilleure approche consiste à inclure tous les canaux directement dans les imagettes. Chaque imagette devient donc une imagette en couleur, et sa taille augmente donc en conséquence. Une imagette de 10 par 10 pixels par exemple qui donnerait un vecteur prototype de taille 100, passe à 300 avec les trois couleurs. L'apprentissage et la reconstruction se déroulent de la même manière que dans la SOM classique. Il faut aussi noter que l'ordre n'a pas d'importance dans les vecteurs prototypes, on peut arranger les valeurs en RGBRGBRGB tout comme RRRGGGBBB sans que cela ne change le résultat, le calcul de distance euclidienne étant indépendant de l'ordre des coordonnées. La figure 2.4 montre une comparaison de reconstructions obtenues entre couleurs fusionnées et couleurs séparées.



FIGURE 2.4 – Comparaison entre une image avec des couleurs fusionnées et la même image avec des couleurs séparées qui présente des artefacts visuels.²

2. Image source : https://unsplash.com/photos/9A_peGrSbZc

2.3 Détection de nouveauté par QVAT : quantification vectorielle et apprentissage topologique

Nous avons à partir des différentes propriétés de nos modèles neuronaux et de l'apprentissage d'images, développé des processus de détection de nouveauté. Ces processus ne sont pas intrinsèques à nos modèles, c'est à dire que nos modèles neuronaux n'ont pas été définis dans le but d'effectuer une détection de nouveauté. Elle est une propriété émergente de ces modèles. Nous présentons ces deux méthodes dans cette section. La première est basée sur la propriété de quantification vectorielle et la seconde sur la topologie des SOM, avec une modulation par ce que nous appelons la distance neurale. La première a été introduite dans [BERNARD et collab. \[2019\]](#), et complémentée par la seconde dans [BERNARD et collab. \[2020\]](#).

2.3.1 Détection avec quantification vectorielle

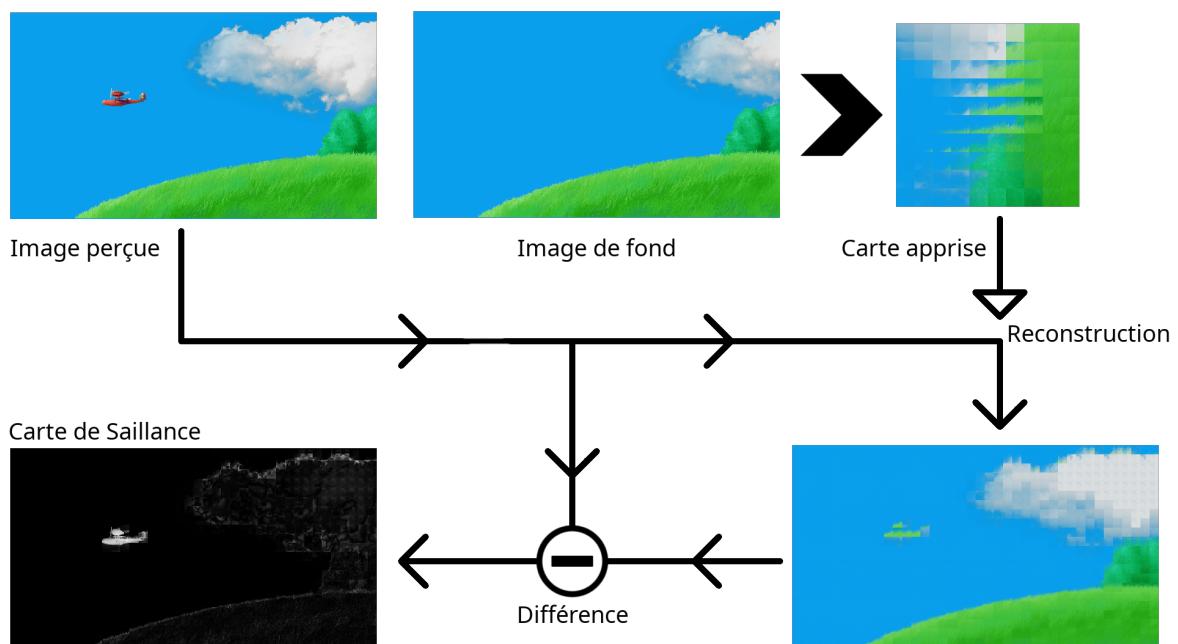


FIGURE 2.5 – On peut observer qu'il y a eu deux changements entre le fond et l'image perçue : un avion est apparu et les nuages ont bougé. Les nuages, déjà présents dans le fond sont bien reconstruits. L'avion cependant est nouveau, et n'est pas bien reconstruit. Ainsi la différence entre l'image perçue et la reconstruction rend plus saillant l'avion que les nuages. Contrairement à une simple différence entre le fond et l'image perçue, où les deux seraient saillants. Nous avons représenté le modèle appris comme étant une SOM sur cette figure, cependant il peut s'agir de n'importe quel modèle de quantification vectorielle.

La détection par quantification vectorielle (VQ) repose sur l'erreur de reconstruction de la nouvelle image provenant du capteur. Cette erreur est la différence entre la nouvelle image et celle reconstruite avec le modèle de VQ ayant effectué son apprentissage sur l'image de fond.

Il y a deux cas à considérer pour chaque imagette de la nouvelle image du capteur : soit la nouvelle imagette est similaire à une partie quelconque de l'arrière-plan

apris, soit quelque chose de nouveau est présent dans l'imagette. Dans le premier cas, le neurone représentatif de cette imagette sera proche de celle-ci, c'est à dire que les poids du neurones et de l'imagette seront proches. L'erreur de reconstruction de cette imagette sera faible, surtout si la caméra est statique, mais avec une petite différence toujours présente en raison des pertes de la compression et des changements naturels de l'environnement visuel.

Cependant, dans le second cas, le neurone représentatif de l'imagette sera éloigné de celle-ci, dans le sens où ses poids seront très différents de ceux de l'imagette. Car la nouveauté est par définition quelque chose qui n'était pas présent dans l'arrière-plan et donc quelque chose que le modèle n'a pas appris. Cela entraînera à une différence significative lors du calcul de la soustraction entre l'image perçue et sa reconstruction à l'endroit de la nouveauté. Ce processus est illustré dans la figure 2.5, et des exemples de résultats dans la figure 2.6.

Ce processus a l'avantage d'être précis au niveau du pixel pour la mise en évidence des changements, avec une certaine limite liée à la taille des imagettes. Si les changements sont trop importants et que la nouvelle BMU est trop différente de celle de référence, alors il y a un risque de perte de précision au niveau de l'imagette. Il est aussi théoriquement insensible au déplacement d'objets du fond pour la détection de la nouveauté. Cependant, il peut être bruité en raison de l'apprentissage imparfait de la VQ et de la variabilité naturelle de l'environnement.

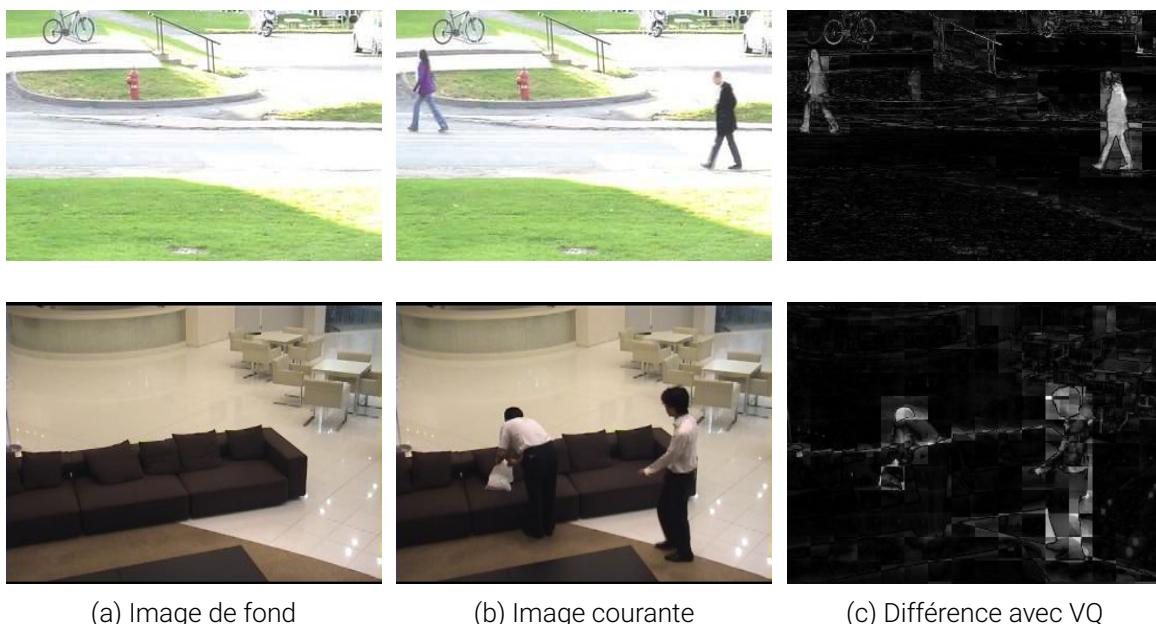


FIGURE 2.6 – Exemples de différence avec quantification vectorielle. On peut remarquer que la couleur est un facteur important pour savoir si un objet nouveau sera correctement détecté.

2.3.2 Détection avec distance neurale

La détection avec distance neurale se base sur les propriétés topologiques du modèle pour trouver la nouveauté dans une image. La topologie est ce qui relie les différents neurones de modèle par leur proximité : les neurones proches dans la topologie ont appris des poids similaires.

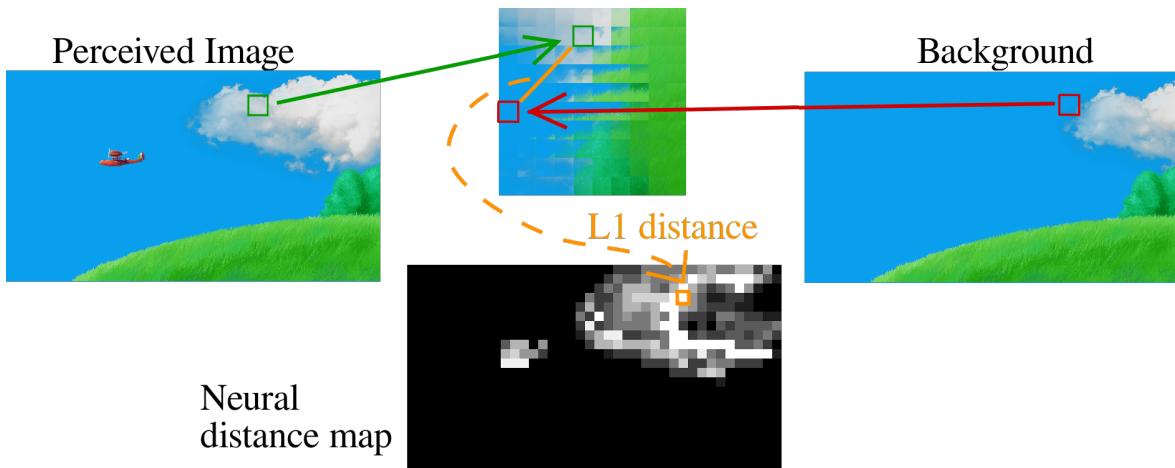


FIGURE 2.7 – Le processus présenté ici concerne une position dans l'image, et il est répété sur toute l'image pour obtenir la carte de distances neurales en bas. Nous avons représenté le modèle appris comme étant une SOM sur cette figure, cependant il peut s'agir de n'importe quel modèle avec une topologie regroupant les éléments proches.[Traduire la figure en français]

Après avoir effectué l'apprentissage du modèle, nous mémorisons la liste des positions dans la carte des BMU trouvés pour toutes les imagettes. Lorsqu'une nouvelle image est présentée au capteur, nous effectuons le processus de reconstruction comme dans la première méthode de détection de la nouveauté. Mais nous nous intéressons uniquement à la liste des positions dans la carte des BMU trouvés pour toutes les imagettes de la nouvelle image. En comparant les deux listes, nous pouvons trouver des changements dans les positions des BMU pour chaque emplacement d'imagette. Si la BMU est la même entre le fond et la nouvelle image à un endroit, alors il n'y a probablement pas de nouveauté à cet endroit. S'il y a une différence entre les deux BMU, alors il pourrait y avoir de la nouveauté à cet endroit. Pour quantifier cette différence, nous calculons la distance topologique qui les sépare sur la carte. La distance topologique est définie par le nombre de noeuds qui sont traversés par le chemin le plus court entre les deux BMU dans la topologie de la carte. Dans la SOM classique basée sur une grille, il s'agit simplement de la distance L1.

Grâce à ces distances, nous pouvons créer une carte de saillance où des distances élevées dans la topologie signifient des changements significatifs dans l'image, car la proximité dans la topologie signifie la proximité dans l'espace d'entrée. Cette méthode permet d'obtenir une carte de saillance plus robuste, avec moins de bruit que la version avec la VQ. La précision est limitée à la taille des imagettes et il n'y a pas d'inhibition pour les éléments déjà connus qui se sont déplacés dans le fond. Le processus est illustré dans la figure 2.7.

2.3.3 Considérations pour la combinaison

Une fois les deux cartes de saillance générées, il est nécessaire de les combiner pour n'avoir qu'un seul résultat qui représentera la sortie de notre système. Il existe un très grand nombre de façons de faire cette combinaison, et il existe dans la littérature des modèles qui se basent sur une bonne combinaison de différentes cartes de saillance pour obtenir de meilleurs résultats [BIANCO et collab. \[2017\]](#). Dans notre cas, nous avons préféré utiliser une combinaison simple de nos deux cartes de saillance.

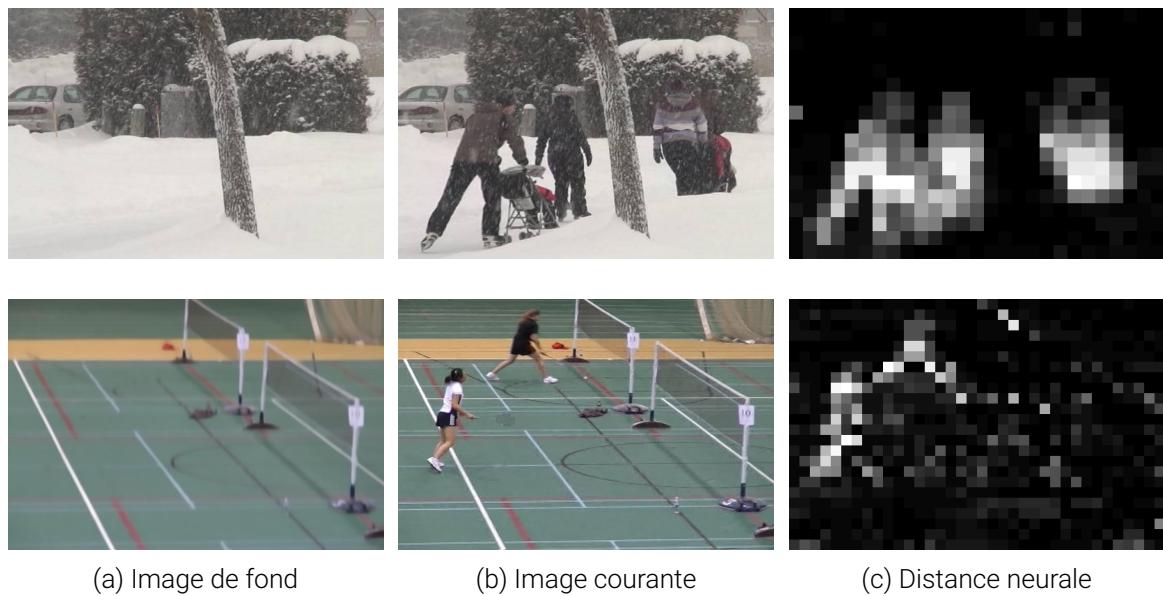


FIGURE 2.8 – Exemples de différence avec distance neurale.

C'est à dire qu'elle n'utilise pas de paramètres, pour ne pas ajouter une variable de plus à optimiser. Nous souhaitons aussi bénéficier de la complémentarité des deux cartes de saillance. La solution la plus simple est de multiplier les deux cartes ensemble. Ainsi, sera considéré comme nouveauté dans la carte de sortie, ce qui apparaît comme nouveauté en même temps dans les deux cartes de saillance, sur le principe de $\text{petit} \times \text{petit} = \text{petit}$, $\text{grand} \times \text{petit} = \text{petit}$ et seulement $\text{grand} \times \text{grand} = \text{grand}$. Le bruit présent dans la carte résultant de la quantification vectorielle et qui n'est pas présent dans la carte topologique disparaît de la carte finale. Il en va de même pour les mouvements qui ne sont pas des nouveautés qui apparaissent dans la carte topologique, mais pas dans la carte de quantification vectorielle.

Un problème qui peut apparaître avec cette méthode est la trop petite valeur du résultat et le déséquilibre d'impact de nos deux cartes, car nos cartes de saillance sont toutes les deux définies entre 0 et 1. Il est possible que des situations arrivent lors desquelles les deux cartes ont un impact disproportionnel sur le résultat. Par exemple si la saillance a une valeur de 0.2 sur une carte et 0.8 sur l'autre, alors la seconde aura plus d'impact sur les valeurs de la sortie finale. De même, en multipliant deux nombres compris entre 0 et 1, le résultat sera forcément inférieur à chacun des deux nombres. Cela a un effet réducteur sur toutes les valeurs de la carte de saillance finale. La solution que nous avons choisi à ces deux problèmes, est de re-normaliser les deux cartes de saillances avant de les multiplier. C'est à dire que l'on rééchelonne l'ensemble de la carte en mettant la valeur maximum de la carte à 1 et le minimum à 0 et d'étaler les valeurs intermédiaires entre les deux pour conserver le même espacement relatif entre elles. Cela a pour effet d'éviter une trop grande disproportion d'impact entre les deux cartes sans résoudre complètement le problème cependant. Car on re-normalise avec le maximum, et non avec la possible valeur de la nouveauté détectée. Cela permet également d'avoir un résultat avec des valeurs généralement plus hautes. Cependant, cela vient aussi avec des désavantages, comme par exemple le fait que si il n'y a pas de signal dans l'entrée, le maximum des cartes de saillance sera quand même 1. On pourrait observer des signaux positifs dans la sortie alors que l'entrée et les cartes de

saillances n'en montrent pas. En pratique, cela est peu fréquent car la valeur maximum dans un cas où il n'y a pas de signal en entrée vient du bruit, et est donc décorellée entre les deux cartes, et disparaîtra lors de la multiplication. De plus, la taille du signal d'entrée compte, et il est peu probable que du bruit seul puisse créer une zone de signal assez large pour être confondu avec une vraie nouveauté.

2.4 Protocole expérimental

Cette section regroupe l'ensemble des considérations pratiques pour la réalisation de nos expériences. Nous présenterons la base de donnée utilisée, comment ces données ont été préparées, les différentes métriques que nous avons mesurées et les paramétrages de nos modèles.

2.4.1 Présentation de la base de données

Nous avons choisi d'utiliser la base de données ChangeDetection.Net (CDnet) [WANG et collab. \[2014\]](#) pour réaliser nos expériences. C'est un jeu de données très utilisé dans le domaine de la détection de changements dans le domaine visible et infrarouge avec une caméra statique. Ce sont des données typiquement disponibles par le biais de caméras de surveillance par exemple, et l'application pour laquelle elle mesure les performances est la détection de changements, comme l'indique son nom.

Nous avons dans cette thèse avant tout parlé de détection de nouveauté, et il existe une différence conceptuelle entre la nouveauté et le changement. La détection de changement se concentre sur le mouvement pour séparer le fond des objets intéressants dans une image. La détection de nouveauté quand à elle se réfère à une représentation apprise de l'environnement. Dans les captures vidéos réelles, les deux sont généralement équivalents dû au fait que lorsqu'une nouveauté apparaît, elle le fait généralement en se déplaçant. En pratique cela veut dire que la majorité, mais pas la totalité, de CDnet peut être utilisée pour de la détection de nouveauté. Nous présenterons les catégories et vidéos que l'on a utilisées, et des exemples de vidéos qui n'ont pas été retenues avec des explications dans la suite.

CDnet regroupe 53 séquences vidéos originaires de sources variées. Elles proviennent principalement de caméras de surveillance ou de captures effectuées par des chercheurs pour leur propres besoins. Il n'y a que des captures réelles, sans images synthétiques, de scènes intérieures et extérieures. Les vidéos sont toutes en couleur, sauf pour deux catégories *thermal* et *turbulences*, et de résolution assez faible, allant de 320×240 à 720×486 pixels. Elles sont groupées en 11 catégories de 4 à 6 vidéos sensées représenter une variété de difficultés que peuvent rencontrer les modèles de détection de changement. Il existe cependant un certain biais dans CDnet, car il est fortement orienté vers de la détection de personnes et de véhicules. Ces catégories sont présentées dans les figures suivantes.

Parmi les 11 catégories de CDnet, nous avons décidé d'en utiliser 8 complètement, d'utiliser une version réduite pour 2 d'entre elles et d'en enlever une. Les deux catégories réduites sont *Intermittent Object Motion* et *Low framerate*. La réduction consiste à enlever une partie des vidéos qui ne correspondaient pas à notre application visée de détection de nouveauté et de conserver les autres. Une illustration de la différence entre la détection de changements et la détection de nouveauté qui est la source de la suppression est montré sur la figure 2.19. La catégorie que nous avons décidé de ne pas du tout utiliser car elle ne correspondait pas à notre scénario est *Pan tilt zoom*. C'est une catégorie un peu spéciale car elle change une partie fondamentale du scénario. La caméra n'est plus statique mais effectue des rotations et des zooms ce qui change significativement son environnement visuel.



FIGURE 2.9 – *Baseline* : La catégorie de base qui comprend des scénarios typiques de détection de changement (trafic, piétons) sans difficultés particulières.



FIGURE 2.10 – *Bad Weather* : Cette catégorie comprend des variations du scénario de base avec une météo dégradée. La difficulté principale vient de la neige qui tombe, et du changement de l'environnement avec les traces de pneus sur la neige par exemple.

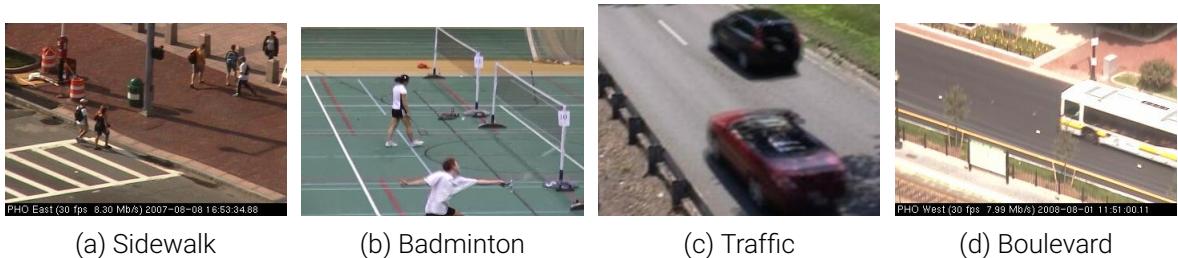


FIGURE 2.11 – *Camera Jitter* : Ces vidéos proviennent de caméras instables à cause de vent fort ou d'autres raisons. Elles ont de façon irrégulière des translations verticales et horizontales rapides et de petite amplitude.



FIGURE 2.12 – *Dynamic Background* : La difficulté se porte sur le contenu du fond qui est changeant. Il peut s'agir d'eau ou d'arbres qui bougent dans le vent.



FIGURE 2.13 – *Shadow* : Catégorie de vidéos qui présente plus d'ombres que la moyenne.



FIGURE 2.14 – *Night Videos* : Vidéos de nuit avec un contraste fort entre l'obscurité ambiante et les lumières artificielles de l'éclairage public et des phares de voitures.

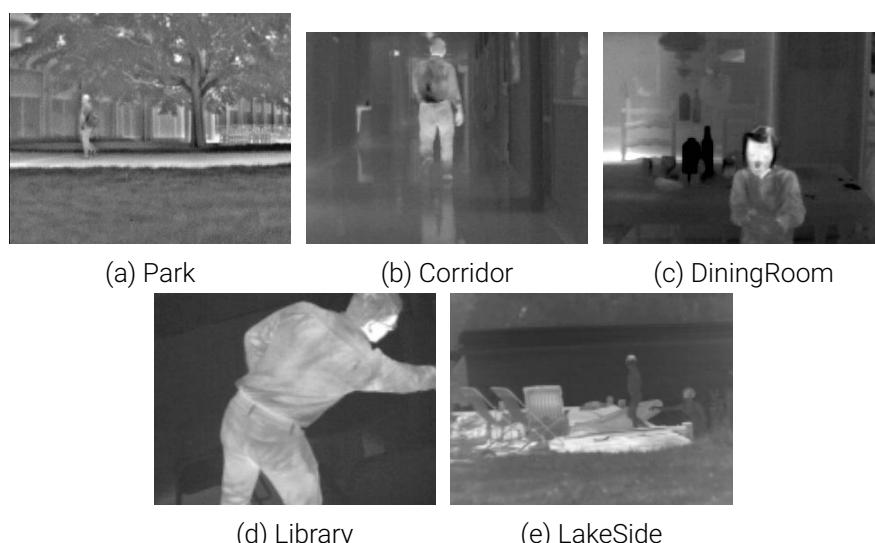


FIGURE 2.15 – *Thermal* : Ces vidéos ont été prises par une caméra infrarouge et sont en niveau de gris.

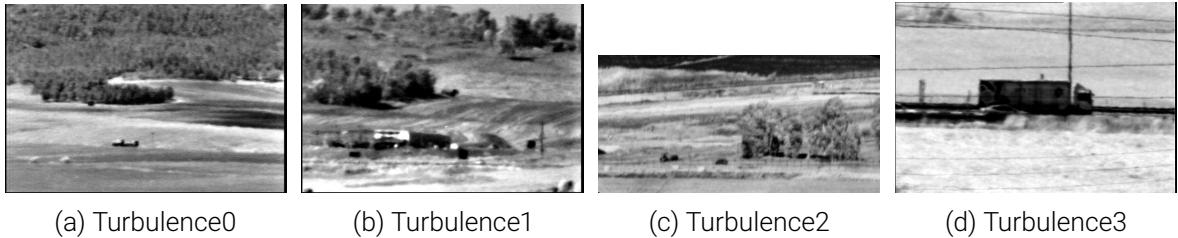


FIGURE 2.16 – *Turbulence* : Catégorie qui regroupe des vidéos provenant d'une même caméra infrarouge. Les captures ont été faites avec un objectif longue distance filmant des scènes éloignées de 5 à 15 km de l'objectif. Elle présente de nombreuses distorsions et turbulences atmosphériques dues à la chaleur et à la distance.



FIGURE 2.17 – *Intermittent Object Motion Reduced* : Cette catégorie comprend des scénarios particuliers dans lesquels le changement est intermittent (c'est à dire qu'un objet passe de mouvement à statique ou inversement). Dans cette catégorie trois vidéos sur six ont été conservées.



FIGURE 2.18 – *Low Framerate Reduced* : Cette catégorie regroupe des vidéos avec beaucoup de temps entre les images (entre 1 seconde et 6 secondes entre chaque image). Cela a pour but de pénaliser les approches à partir de flux optique, cependant notre approche n'est pas concernée. Trois vidéos sur les quatre ont été conservées. Seule une vidéo d'une marina a été retirée car la nouveauté (des bateaux) était trop similaire au fond, qui consiste en un grand nombre de bateaux similaires amarrés.

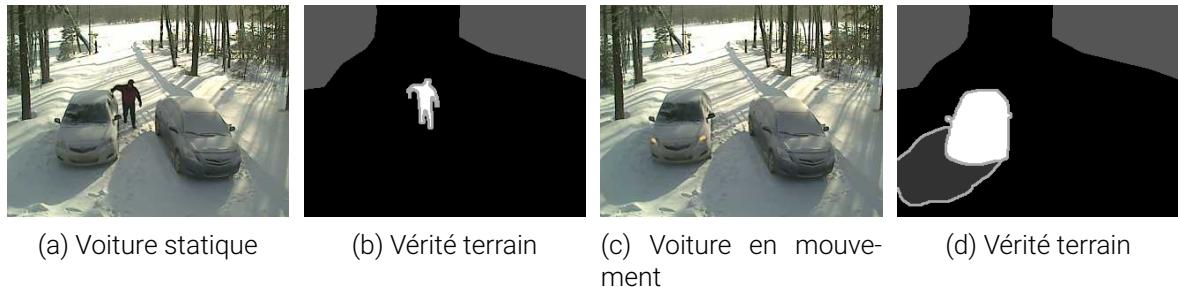


FIGURE 2.19 – Ces images sont extraites d'une vidéo de la catégorie *Intermittent Object Motion* et illustrent la différence entre détection de changement et détection de nouveauté. Pour le changement la voiture fait partie du fond pendant une partie de la vidéo car elle est statique. Elle devient objet à détecter à partir du moment où elle commence à se déplacer. Pour la nouveauté, une telle distinction n'est pas possible. Soit elle fait partie du fond, et dans ce cas, même en mouvement elle ne devrait pas être considérée comme nouveauté. Soit elle ne fait pas partie du fond, et dans ce cas elle sera tout le temps considérée comme nouveauté, même lors de la séquence statique.

2.4.2 Métriques utilisées

Nous présenterons dans cette section les différentes métriques que l'on a utilisées pour évaluer nos modèles. Elles peuvent être regroupées en deux grandes catégories, la première est proche des modèles évalués (la SOM et les GNG), et essaie de mesurer la qualité d'apprentissage de ceux-ci. La seconde est plus orientée vers la tâche de détection de nouveauté. Cette deuxième catégorie utilise des métriques définies par CDnet pour comparer les résultats avec d'autres modèles. Celles-ci étant en grand nombre, nous nous sommes limités aux trois plus pertinentes qui sont la précision, le rappel et la F-mesure.

MQE : Erreur de Quantification Moyenne

L'erreur de quantification moyenne (Mean Quantization Error) mesure la qualité de l'apprentissage ou de la reconstruction d'un algorithme de quantification vectorielle. C'est la somme des différences entre tous les vecteurs et leurs représentants, divisée par la dimension et le nombre de vecteurs pour obtenir l'erreur moyenne des composantes.

$$\text{MQE} = \frac{1}{dn} \sum_{i=0}^{n-1} |\nu_i - u_i| \quad (2.1)$$

Avec n le nombre de vecteurs d'entrée, d la dimension de ces vecteurs, ν_i le $i^{\text{ème}}$ vecteur d'entrée et u_i le prototype du neurone représentant de ce vecteur d'entrée.

Il s'agit d'une mesure simple à calculer et à comprendre. Elle présente néanmoins une mauvaise pondération des différences en pénalisant les *outliers* de la même manière que des différences diffuses. Par exemple un pixel qui aura un changement maximal (qui passe de noir à blanc par exemple) aura le même impact sur l'erreur que cent pixels qui passent de 0 à 0,01. Pour des images, le premier changement sera visible pour notre œil, mais pas le second. Nous avons tout de même choisi d'utiliser cette

mesure dans nos expériences car elle représente le mieux la différence numérique des vecteurs à leurs représentants avant notre interprétation subjective de ces valeurs en images.

PSNR : Peak Signal to Noise Ratio

Le PSNR est une mesure très présente dans le domaine de la compression d'images [HUYNH-THU et GHANBARI \[2008\]](#); [KORHONEN et You \[2012\]](#). Elle est similaire à la MQE, à la différence qu'il y a un carré à la place de la valeur absolue. D'où le nom de Mean Squared Quantization Error (MSQE) à calculer pour pouvoir obtenir le PSNR.

$$\text{MSQE} = \frac{1}{dn} \sum_{i=0}^{n-1} (v_i - u_i)^2 \quad (2.2)$$

$$\text{PSNR} = 10 \times \log_{10} \left(\frac{1}{\text{MSQE}} \right) \quad (2.3)$$

Avec n le nombre de vecteurs d'entrée, d la dimension de ces vecteurs, v_i le $i^{\text{ème}}$ vecteur d'entrée et u_i le prototype du neurone représentant de ce vecteur d'entrée.

Le PSNR est inspiré du domaine du traitement du signal, d'où la terminologie étrange pour de la compression d'image. L'idée est de trouver le ratio de bruit introduit par les pertes de la compression (Noise), avec l'intensité maximale du signal (Peak Signal), qui est la valeur maximum d'un pixel (1 dans notre cas). On peut noter que le PSNR fait passer d'un objectif de minimisation à une maximisation. L'ordre des valeurs reste inchangé par rapport au MSQE et le logarithme ajoute un effet de rendement décroissant. Si on a par exemple trois valeurs de MSQE x_1 , y_1 et z_1 avec $x_1 < y_1 < z_1$. x_1 sera la meilleure (la plus petite) et z_1 la moins bonne (la plus grande). Une fois converties en PSNR, les valeurs seront dans l'ordre suivant : $x_2 > y_2 > z_2$, avec x_2 étant toujours la meilleure et z_2 toujours la moins bonne, mais pour les raisons inverses cette fois : plus un PSNR est grand, mieux c'est. Un autre changement sera que si les intervalles étaient les même entre les trois valeurs, c'est à dire si $y_1 - x_1 = z_1 - y_1$ alors on aura $x_2 - y_2 > y_2 - z_2$, à cause de l'inversion de MSQE.

Nous avons choisi d'utiliser le PSNR à la place de la MSQE car il est beaucoup plus facilement interprétable par des humains ; sa valeur typique étant comprise entre 0 et 100. L'utilisation du carré dans la MSQE entraîne une plus grande pénalisation des *outliers* en comparaison à MQE. Ce n'est malgré tout pas une mesure objective de la qualité d'une image, car elle ne prend pas en compte certains paramètres comme le voisinage par exemple, qui sont importants dans la perception visuelle [11.3](#). Puisque l'on utilise la distance quadratique dans nos algorithmes de quantification vectorielle, le PSNR est une métrique intéressante car elle est la valeur que notre algorithme d'apprentissage tente de minimiser.

Précision

La précision mesure la proportion de pixels correctement labellisés en tant que nouveauté parmi tous les pixels que le modèle a étiqueté comme nouveauté. La précision est comprise entre 0 et 1 et s'exprime souvent en pourcentage.

$$\text{Précision} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Positifs}} \quad (2.4)$$

Rappel

Le rappel mesure la proportion de pixels correctement labellisés en tant que nouveauté parmi tous les pixels avec de la nouveauté dans la vérité terrain. Le rappel est compris entre 0 et 1 et s'exprime souvent en pourcentage.

$$\text{Rappel} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Négatifs}} \quad (2.5)$$

F-mesure

La précision et le rappel sont deux mesures qui, prises séparément, peuvent être facilement maximisées. Pour avoir une très bonne précision, il suffit de n'inclure que les pixels positifs dont le modèle est sûr dans la carte de saillance, pour réduire la proportion de faux négatifs et ainsi améliorer la précision. Cela entraînera cependant un rappel faible, car le nombre total de vrais positifs est réduit. Pour maximiser le rappel, il suffit de faire l'inverse, c'est à dire de catégoriser le plus possible de pixels en positifs dans la carte de saillance et ainsi réduire le nombre de faux négatifs. Cela se fait au détriment de la précision cependant, car le nombre de faux positifs sera en augmentation. La F-mesure [HRIPCSAK et ROTHSCHILD \[2005\]](#) tente d'être une solution à ce problème en combinant la précision et le rappel en un seul nombre à maximiser. Ce n'est cependant pas une mesure sans défauts [POWERS \[2011\]](#).

$$\text{F-mesure} = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}} \quad (2.6)$$

Le formule de la F-mesure est assez simple, mais elle cache un comportement plus complexe. La précision et le rappel étant tous les deux entre 0 et 1, la F-mesure ne peut aussi prendre des valeurs que dans cet intervalle. Car lorsque les deux sont égaux à 1, la formule donne aussi 1. De par les propriétés de la multiplication entre deux nombres entre 0 et 1, la F-mesure favorise les précisions et rappels proches entre elles, et pénalise lorsque les deux valeurs sont éloignées. Ainsi, pour maximiser la F-mesure, augmenter la valeur la plus basse entre la précision et le rappel aura le plus d'effet.

Une propriété notable dans le calcul de la F-mesure est l'absence de distributivité. En pratique, cela implique que la moyenne des F-mesure n'est pas égal à la F-mesure des moyennes de précision et rappel. Cela peut poser problème lorsque l'on essaye d'agréger des valeurs sur plusieurs images par exemple. Il est donc nécessaire de calculer les F-mesures pour chaque image séparément pour ensuite en faire la moyenne. On peut également observer cette propriété dans les résultats présentés dans la section [2.5.3](#), où la valeur de la F-mesure ne suit pas la formule lorsque elle est appliquée aux moyennes des précision et rappel.

L'œil humain

Il n'existe pas de mesure objective pour déterminer la qualité de l'apprentissage d'une quantification vectorielle ou de détection de nouveauté, il n'y a que des estimations et approximations. Pour la quantification vectorielle par exemple, le calcul de l'erreur semble naturel, mais il ne capture pas toutes les spécificités des modèles. Notamment lorsque l'on travaille avec des images, le problème déjà évoqué que certaines erreurs sont visuellement plus perceptibles que d'autres, alors qu'elles ont la même valeur numérique. Mais il y a aussi des propriétés de certains modèles de VQ qui ne peuvent être simplement quantifiés, comme par exemple la gestion des *outliers*. Un algorithme peut par exemple préférer représenter le mieux possible la majorité de la base de données en laissant de côté les *outliers*, ou au contraire de faire en sorte que toutes les données soient bien représentées, mais en sacrifiant une meilleure précision sur les données les plus nombreuses.

Le même problème de subjectivité se présente pour la détection de nouveauté, par exemple dans sa différenciation avec la détection de changement déjà évoquée. Mais en plus, les mesures quantitatives que nous avons sélectionnées ne représentent pas forcément l'aspect qualitatif de la détection de nouveauté. Par exemple si on a un algorithme qui a pour résultat le contour des objets nouveaux dans une image, la tâche sera bien remplie, mais le score sur les métriques sera mauvais car il sera attendu de l'algorithme qu'il marque de façon positive tous les pixels de l'objet nouveau et non pas seulement le contour. Ce genre de problème peut parfois être résolu par l'utilisation de post processing, le remplissage à partir des contours ici par exemple. Mais ce n'est pas toujours le cas.

Il est donc nécessaire d'adoindre à ces métriques objectives, une estimation subjective du comportement des modèles évalués. Pour les images, nous avons la chance d'être naturellement dotés de très bons capteurs et d'un réseau neuronal biologique performant très entraîné sur des données visuelles. Par conséquent certaines de nos interprétations s'appuieront sur le résultat visuel de nos modèles en complément des métriques.

2.4.3 Préparation des données

Image de fond

Notre modèle a besoin pour l'apprentissage d'un fond sans cibles pour fonctionner. Il nous faut donc une image du fond pour chaque séquence vidéo. Il arrive cependant que des vidéos présentent pendant toute la séquence de la nouveauté, et qu'une image sans perturbation n'existe pas dans la séquence. Pour contrer ce problème, nous avons calculé une image médiane de la séquence pour obtenir une image de fond sans perturbations.

Pour une mise en pratique potentielle, nous supposons qu'il serait facile d'avoir une image du fond sans perturbations, ou qu'il serait aisément d'appliquer la même méthode en effectuant une capture vidéo et en calculant la médiane lors de l'initialisation.

Échantillonage de l'évaluation

Le calcul des métriques sur les séquences de CDnet se font sur de nombreuses images. Pour *baseline* par exemple, il faut en moyenne 1100 images par séquence. Chaque image nécessitant quelques secondes de temps de calculs, utiliser l'intégralité de chaque séquence serait beaucoup trop long pour pouvoir étudier en détail les nombreux paramètres de notre modèle. En partant du principe que des images proches dans le temps sont aussi généralement similaires dans leur contenu, il serait possible d'évaluer une séquence en n'en évaluant qu'un sous-échantillon afin d'obtenir des mesures représentatives du résultat complet. Les images d'une séquence étant indépendantes temporellement les unes des autres dans notre modèle, nous avons choisi de prendre un sous-échantillonnage régulier de la séquence.

Nous avons procédé à une étude pour déterminer la taille de sous-échantillon qui conviendrait le mieux, que nous présentons dans la figure 2.20. Nous avons choisi d'évaluer 105 images par séquences. Cela représente une évaluation entre 7 et 14 fois plus rapide que si on évaluait l'intégralité de la séquence. La vitesse dépendant du nombre d'images total de la séquence complète. La précision des métriques en est impactée, mais l'on reste entre 0.5% du résultat.

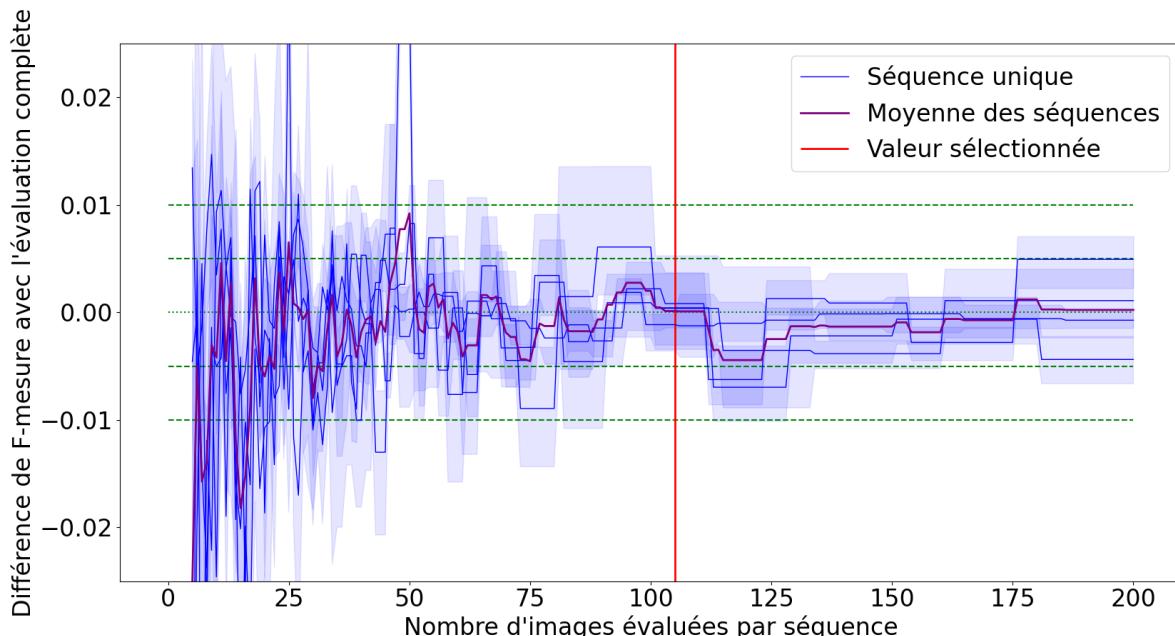


FIGURE 2.20 – Précision de l'approximation de la F-mesure en fonction de la taille de l'échantillon. 105 a été choisi car c'est le plus petit échantillon proche de la moyenne et ne dépassant quasiment pas les 0.5% de différence. Les sections droites des lignes, surtout vers des grands échantillons, sont à cause d'arrondis sur la taille du pas entier utilisé. Ainsi les échantillons ne sont pas de la taille exacte que ce que laisse indiquer l'abscisse, mais au moins de la taille indiquée, souvent plus grands. La variabilité qui augmente après 105 n'est pas nécessairement un problème, car l'on ne mesure pas un procédé stochastique, mais la représentativité des images sélectionnées. Ce qui veut dire que la variation due à l'échantillonnage devrait en théorie rester similaire même avec des paramètres différents pour la SOM.

2.4.4 Paramétrages des modèles

Notre modèle comporte de nombreux paramètres pour lesquels l'impact sur les performances n'est pas trivial. C'est à dire que changer un paramètre dans un sens pourrait amener à une augmentation des performances à une certaine valeur, et à une diminution des performances à une autre valeur. Il existe également des interactions entre les paramètres, ce qui signifie que le paramètre a optimal ne sera pas le même pour deux valeurs du paramètre b par exemple.

En général, dans ces cas de figure, on effectue une optimisation globale de tous les paramètres en même temps, pour prendre en compte l'interaction entre les paramètres. Cependant, dans notre cas, l'espace de recherche serait très grand (8 dimensions pour 8 paramètres) et nécessiterait un très grand nombre d'exécutions pour le couvrir entièrement. Chacune de nos exécutions durera en moyenne quelques minutes, il n'est pas souhaitable d'en effectuer un trop grand nombre. De plus, il est difficile avec ce genre d'approche de la paramétrisation de comprendre l'impact de chaque paramètre sur le résultat.

Nous avons ainsi choisi de faire une étude paramétrique en séparant les paramètres le plus possible. Cela nous permet d'analyser chaque paramètre, ou groupe de paramètres en détail pour mieux comprendre leur effet sur le comportement de notre modèle. Cela nous amènera également à pouvoir prédire un comportement dans des espaces paramétriques que nous n'avons pas explorés ; vers quelle valeur les performances convergent-elles si on pousse un paramètre vers l'infini par exemple. Cette étude pourra être faite avec un nombre raisonnable d'exécutions avec les moyens matériels que nous avons à disposition. Le défaut sera que l'interaction entre certains paramètres sera difficile à évaluer. Nous optimiserons nos paramètres pour maximiser la *F-mesure*, car c'est la métrique la plus proche de la tâche de détection de nouveauté. La section se conclura par un tableau récapitulatif des paramètres que nous aurons utilisés pour générer nos résultats.

Variation aléatoire

Les SOM que nous utilisons ne sont pas déterministes, et plusieurs facteurs aléatoires peuvent influencer le résultat d'un apprentissage. Ces deux facteurs sont l'initialisation des poids des neurones, que nous faisons débuter à des valeurs aléatoires entre 0 et 1 pour chaque composante. Mais aussi l'ordre de présentation de la base d'apprentissage qui est aléatoire, et donc varie avec la graine du générateur d'aléatoire paramétrée avant l'apprentissage. Cette dépendance à l'aléatoire implique que les métriques que l'on aura calculées peuvent varier d'un apprentissage à l'autre. Nous avons étudié cette variabilité pour connaître quel serait le plus petit nombre d'exécutions nécessaire pour donner une estimation fiable de la moyenne des résultats pour un ensemble de paramètres.

D'après la figure 2.21, le comportement aléatoire de notre modèle peut-être assimilé à une loi normale. Cela nous permet, en utilisant la règle empirique, de donner un intervalle de confiance lorsque l'on estimera la moyenne pour nos mesures. Nous supposerons que les variances de la catégorie *baseline* sur laquelle nous avons mesuré ces valeurs est du même ordre que sur toutes les vidéos du jeu de données CDNET. Nous supposerons aussi que toutes les distributions suivent une loi normale, comme

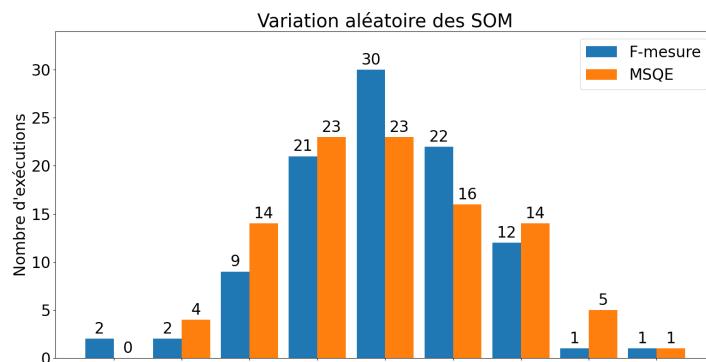


FIGURE 2.21 – Distribution des métriques pour un ensemble de paramètres donnés pour une vidéo. On a découpé l'intervalle de résultats en 9 sections égales. La section numéro 5 a la moyenne en son centre. L'épaisseur de chaque région a été ajustée pour que le maximum soit à la limite haute de la section 9 ou le minimum à la limite basse de la section 1, en choisissant celui qui donnerait les plus grandes sections. L'axe des ordonnées quant à lui donne le nombre d'exécutions incluses dans chaque catégorie, sur 100 exécutions au total.

Nous pouvons observer que les distributions suivent une loi normale. Il semblerait que la variabilité de la F-mesure est inférieure à celle de la MSQE.

celles de la *baseline*.

Video	Écart-type σ	$\delta = 1\% / 2\sigma$	$\delta = 1\% / 3\sigma$	$\delta = 0.5\% / 2\sigma$	$\delta = 0.5\% / 3\sigma$
highway	5.68×10^{-3}	1.3	2.9	5.2	11.6
office	9.9×10^{-3}	4.0	8.9	15.8	35.6
pedestrians	1.52×10^{-2}	9.2	20.8	36.9	83.1
PETS2006	1.08×10^{-2}	4.7	10.5	18.6	41.9

TABLEAU 2.1 – Nombre d'exécutions avec graines aléatoires différentes requises pour que la moyenne des F-mesure l'échantillon soit au moins d'une distance δ de la vraie moyenne, avec une probabilité de 95% pour 2σ et 99,7% pour 3σ . L'écart type à partir duquel on déduit ces valeurs, a été calculé sur un échantillon de 100 exécutions pour *highway*, et 50 échantillons pour les autres.

D'après le tableau 2.1, nous pouvons observer que l'écart type est très variable en fonction de la vidéo que l'on traite. Chaque exécution pouvant durer plusieurs minutes, il est aussi difficilement concevable de faire nos optimisations en visant un résultat à $\delta = 0.5 / 3\sigma$, car cela nécessiterait dans certains cas presque 100 exécutions pour chaque ensemble de paramètres. Nous avons ainsi choisi de se limiter à 8 exécutions avec des graines aléatoires différentes, car les ordinateurs sur lesquels nous expérimen-tions possèdent 8 coeurs, et que cela nous permet de dépasser le premier seuil de $\delta = 1\% / 2\sigma$ pour la plupart des vidéos.

Optimisation d' α et de σ

Nous avons présenté dans la section 1.2.2 ces deux paramètres et leurs effets. Ces paramètres sont très dépendants entre eux, car ils pondèrent tous les deux la formule de modification des poids de l'apprentissage. Nous allons donc les optimiser ensemble. La recherche a été faite par un *Tree-structured Parzen Estimator* BERGSTRA

et collab. [2011], sur des vidéos de la *baseline*. Nous avons fait plus de 1000 mesures par vidéo, avec pour chaque mesure, la moyenne de 8 exécutions, deux pour chaque vidéo, pour réduire le bruit statistique. La figure 2.22 explique le choix de σ que nous avons fait commencé à 0.7 et finir à 0.015. Puis nous avons relancé une optimisation pour les valeurs d'alpha uniquement pour mieux observer les tendances avec les valeurs fixées de σ . La figure 2.23 présente ces résultats. Nous avons choisi pour α un départ à 0.2 et de finir à 0.05.

Nous pouvons aussi noter l'ordre d'importance des paramètres par leur impact sur le résultat : *Sigma end > Sigma start > Alpha end > Alpha start*.

Durée de l'apprentissage

Le dernier paramètre des SOM qu'il reste à optimiser est le nombre d'époques. Une époque étant un parcours complet de la base d'apprentissage, ce paramètre doit être juste assez élevé pour que la SOM converge vers un résultat et assez bas pour que l'apprentissage ne soit pas trop long. La durée de celui-ci étant directement proportionnel au nombre d'époques. Le résultat de nos tests est montré figure 2.24. Nous avons choisi 120 époques d'apprentissage pour les SOM.

Paramètres des GNG

Nous avons choisi d'utiliser des GNG pour montrer que notre méthode est générable à d'autres modèles avec quantification vectorielle et topologie. Ainsi, nous nous sommes contentés pour la paramétrisation d'utiliser de nouveau un *Tree-structured Parzen Estimator* pour optimiser l'intégralité des paramètres en une fois. Leurs valeurs sont présentées dans le tableau 2.2.

TABLEAU 2.2 – Paramètres des GNG

ϵ_{bmu}	ϵ_j	a_{max}	α	d (erreur globale)	Époques
0.4	0.01	85	0.6	0.3	110

Récapitulatif

TABLEAU 2.3 – Récapitulatif des paramètres SOM

α -start	α -end	σ -start	σ -end	Époques	Images par séquence
0.2	0.05	0.7	0.015	120	105

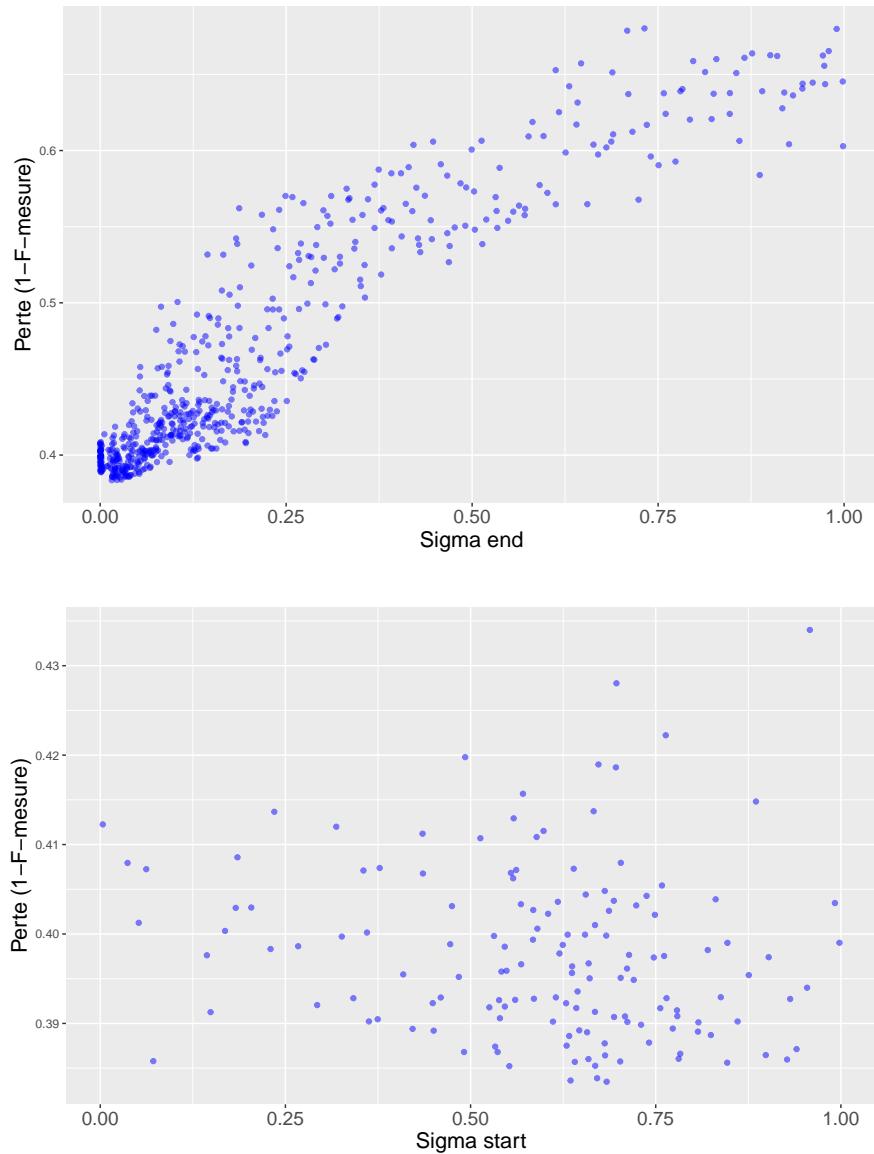


FIGURE 2.22 – Le premier paramètre qui a une valeur optimale évidente est *Sigma end*, où une petite valeur proche de zéro semble idéale. Cela s'explique facilement du fait qu'une valeur faible de *Sigma end* amène à ce que lors des dernières époques la SOM va se focaliser sur l'optimisation de la quantification vectorielle au dépend de la topologie, pour obtenir des neurones les plus proches possibles des imagettes qu'ils représentent. Nous l'avons choisi à 0.015.

Le second graphique représente un sous échantillon d'expériences sélectionnés avec une valeur de *Sigma end* inférieure à 0.05. En affichant les résultats en fonction de *Sigma start*, on observe une légère préférence pour des valeurs de *Sigma start* assez élevées, c'est à dire dans les alentours de 0.7.

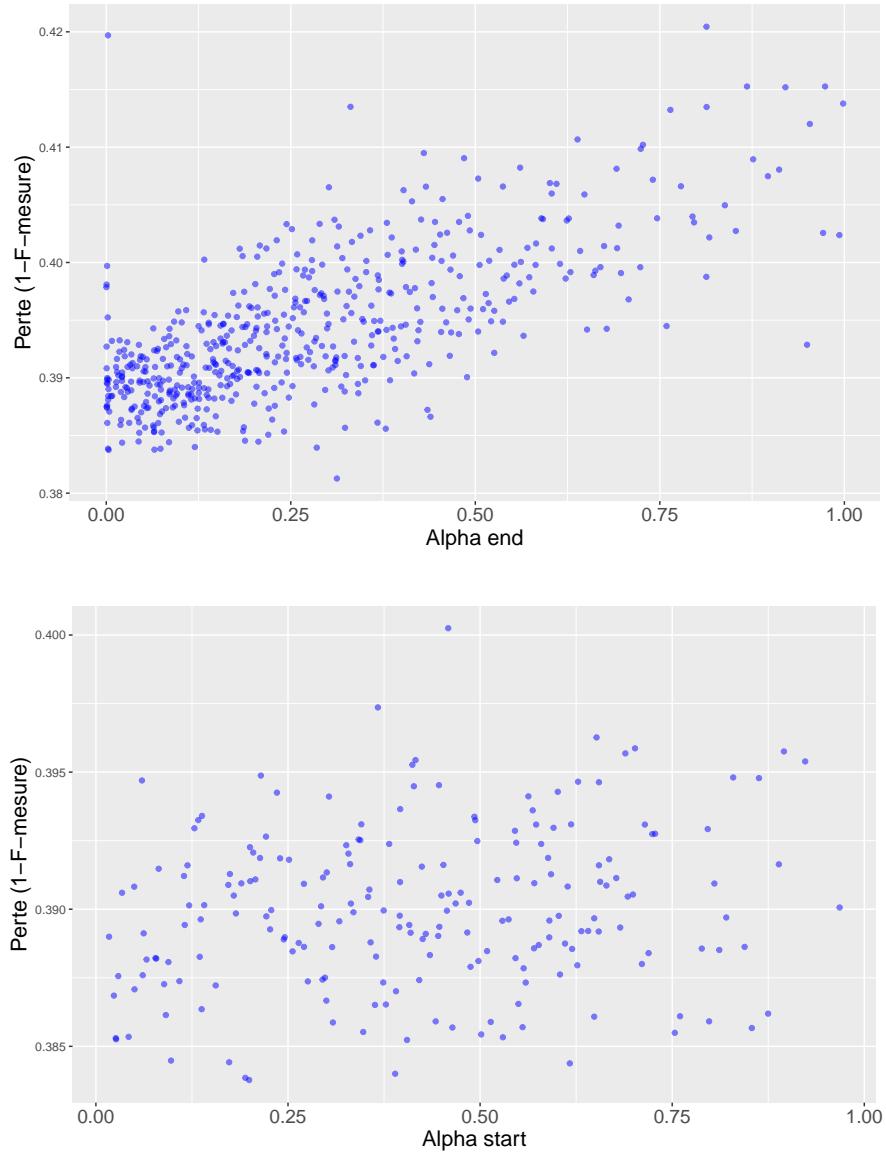


FIGURE 2.23 – *Alpha end* semble fonctionner avec une préférence similaire à *Sigma end*. Une faible valeur d'*Alpha end* non nulle semble optimale. En pratique cela se traduit par un ajustement fin des poids des neurones à la fin de l'apprentissage. Nous l'avons mis à 0.05

Pour *Alpha start* nous avons à nouveau réduit les expériences à un échantillon dont la valeur d'*Alpha end* est inférieure à 0.2. Il ne semble pas y avoir de préférence forte pour *Alpha start*, nous prendrons donc la valeur 0.2, qui semble marcher le mieux.

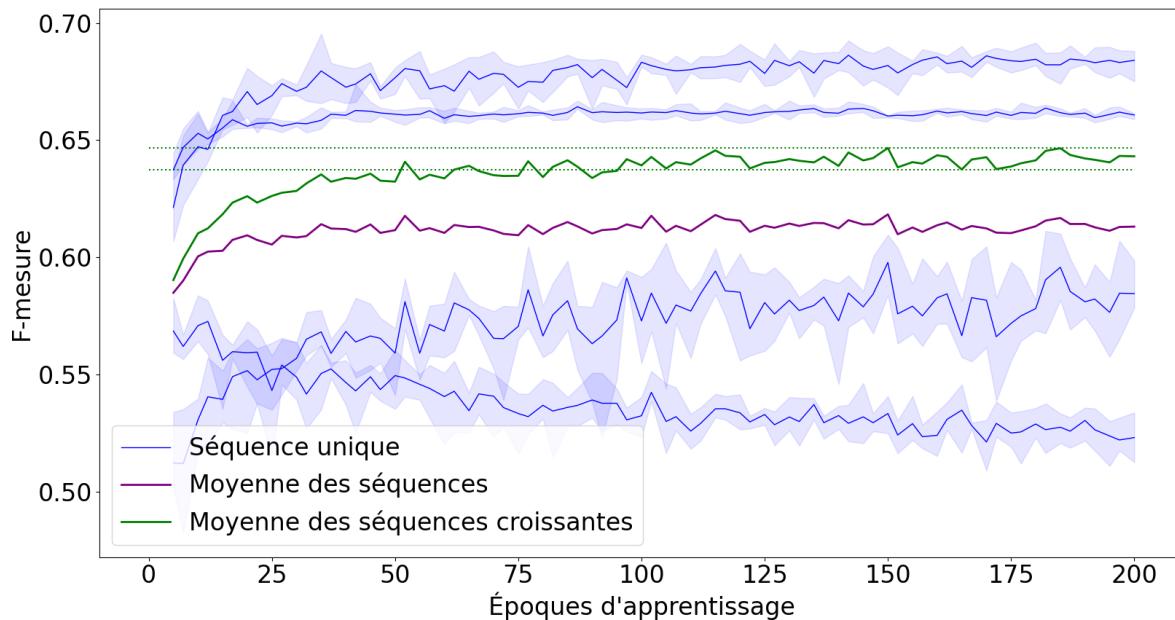


FIGURE 2.24 – F-mesure en fonction du nombre d'époques sur la *baseline*. Une chose étrange que l'on peut remarquer est une des séquences (PETS2006) a des performances décroissantes avec l'apprentissage. Cela pourrait s'expliquer par un seuil trop haut pour cette séquence, car plus l'apprentissage avance, plus les erreurs sont petites. De plus petites erreurs auront plus de chances d'être en dessous du seuil de décision, et ainsi diminuer le rappel, qui diminue ensuite aussi la F-mesure. Nous avons utilisé la moyenne des séquences croissantes uniquement pour déterminer le moment de convergence de l'apprentissage. Après environ 100 époques, les performances semblent stables et restent dans un intervalle restreint de f-mesure. Nous avons donc choisi 120 époques d'apprentissage pour que la convergence se fasse sans trop coûter en calculs.

2.5 Résultats expérimentaux

Nous allons dans cette section présenter tous les résultats quantitatifs et visuels que nous avons obtenu avec notre méthode sur CDnet. Cette section est découpée en quatre parties. La première présentera l'impact du seuil, du nombre de neurones et de la taille des imagettes sur les performances et les différences entre les vidéos lorsque l'on fait varier ces paramètres. La seconde présentera les résultats complets sur CD-NET et comparera avec l'état de l'art. La troisième présentera visuellement les images que produit notre modèle. La dernière sera consacrée aux discussions et perspectives.

Le seuil, la taille des imagettes et le nombre de neurones auraient pu faire partie de la section des paramètres. Cependant compte tenu de l'importance qu'ils ont sur le résultat et des informations et perspectives qu'ils nous donnent, nous les avons inclus dans la partie résultats.

Cette section est basée sur un ensemble d'exécutions sur l'intégralité de CDnet. Les SOM ont utilisé les paramètres définis dans la section précédente [2.4.4](#), et nous avons fait varier les tailles d'imagettes et les tailles de cartes pour cette expérience. Le seuil pouvant être calculé à partir des cartes de saillance uniquement, et donc rapidement, car ne nécessitant pas réapprendre la SOM. Nous l'avons donc également inclus. Au total, cette expérience contient 2520 exécutions pour la SOM, et le même nombre pour les GNG.

2.5.1 Seuil de décision

Le seuil de décision est ce qui permet de passer de notre carte de saillance qui contient des valeurs entre 0 et 255 à la carte de détection binaire. Toutes les valeurs en dessous ou égales au seuil seront considérées comme fond, et les valeurs au dessus du seuil comme nouveauté. La figure [2.25](#) présente nos résultats pour le seuil.

2.5.2 Nombre de neurones et taille des imagettes

Baseline

La figure [2.26](#) montre une différence significative de comportement entre les séquences vidéo. La taille des imagettes semble être le facteur de variations le plus important. Pour *Pedestrians* par exemple, une taille d'imagette petite de 10×10 donnera les meilleures performances, alors que pour *PETS2006*, ce sont les grosses imagettes de 25×25 qui sont optimales. La variabilité est aussi très différente en fonction de la séquence considérée. *Highway* et *Office* sont relativement stables et peu sensibles aux changements de taille d'imagettes et de neurones, tant que ceux-ci ne sont pas trop petits. Au contraire de *PETS2006* pour qui les performances peuvent aller du simple au double en fonction de la taille choisie.

Pour le nombre de neurones, l'optimal semble toujours être du côté où les cartes sont les plus grandes. Cependant les gains apportés par une carte plus grande sont très variables. Généralement importants au début pour une carte petite, les valeurs atteignent rapidement un plateau et n'améliorent que peu la détection de nouveauté.

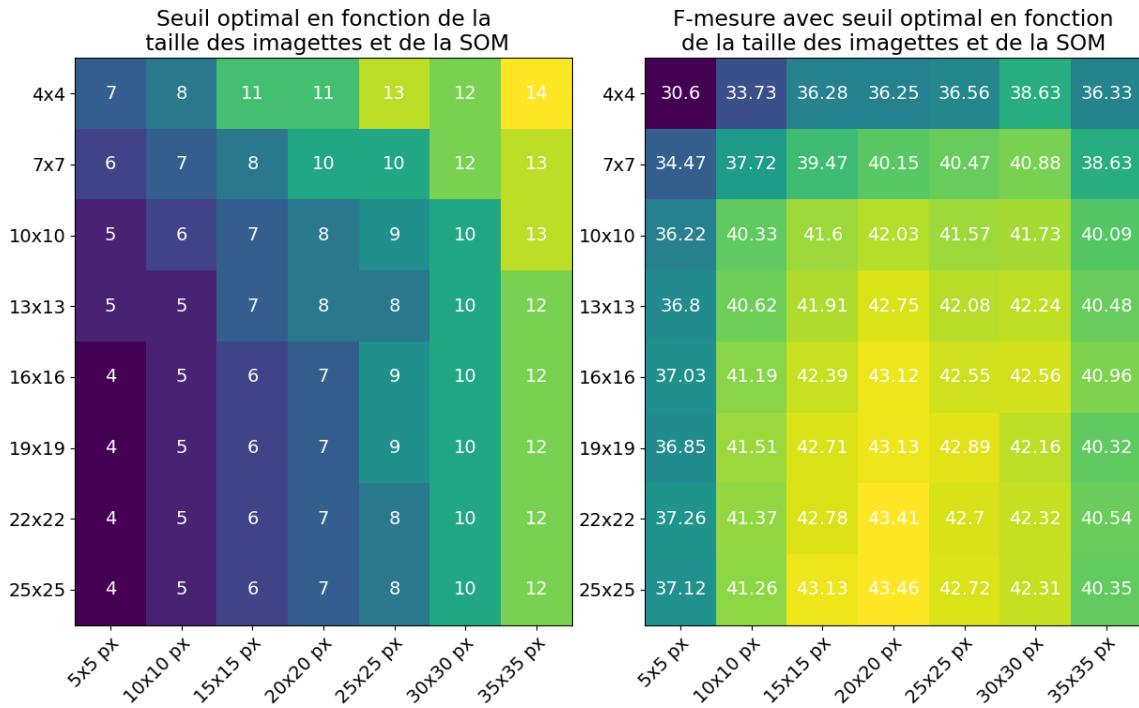


FIGURE 2.25 – La figure de gauche indique le seuil optimal pour chaque couple de taille de SOM et de taille d'imagettes que l'on a testé. On peut remarquer que le seuil optimal diminue plus la taille de la carte est grande, car il y a moins d'erreurs, et qu'il augmente avec la taille des imagettes, car cela amène plus de bruit.

La figure de droite montre la f-mesure complète (sur l'ensemble de CDnet) pour ce seuil optimal. Le maximum est obtenu pour 25×25 neurones, 20×20 pixels et un seuil de 7.

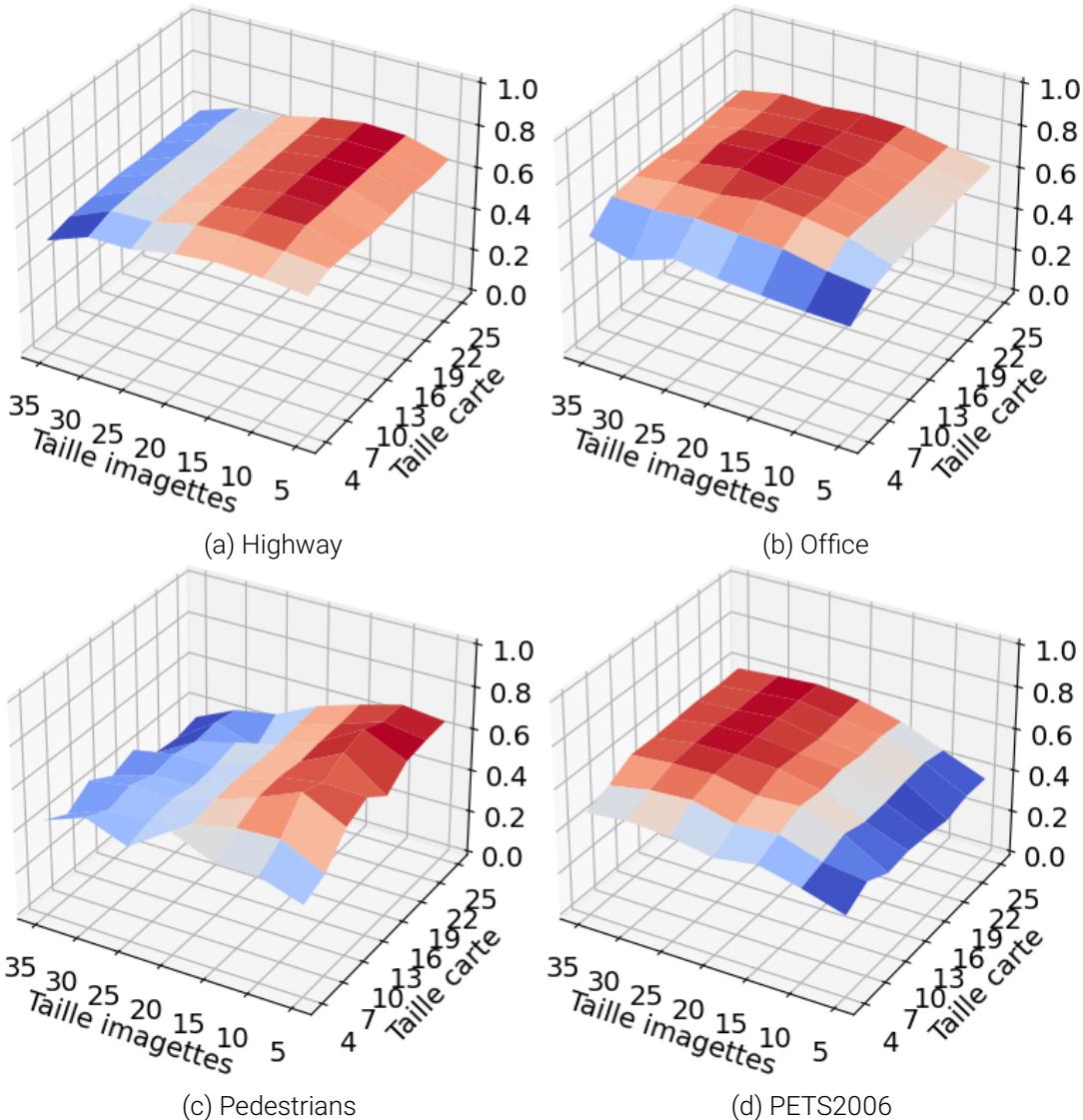


FIGURE 2.26 – F-mesure en fonction du nombre de neurones et de la taille des imagettes pour les séquences de la *baseline* avec une SOM.

C'est une information importante pour une détection rapide et efficace, car le nombre de neurones dans la carte est un facteur important du coût en calcul de la SOM.

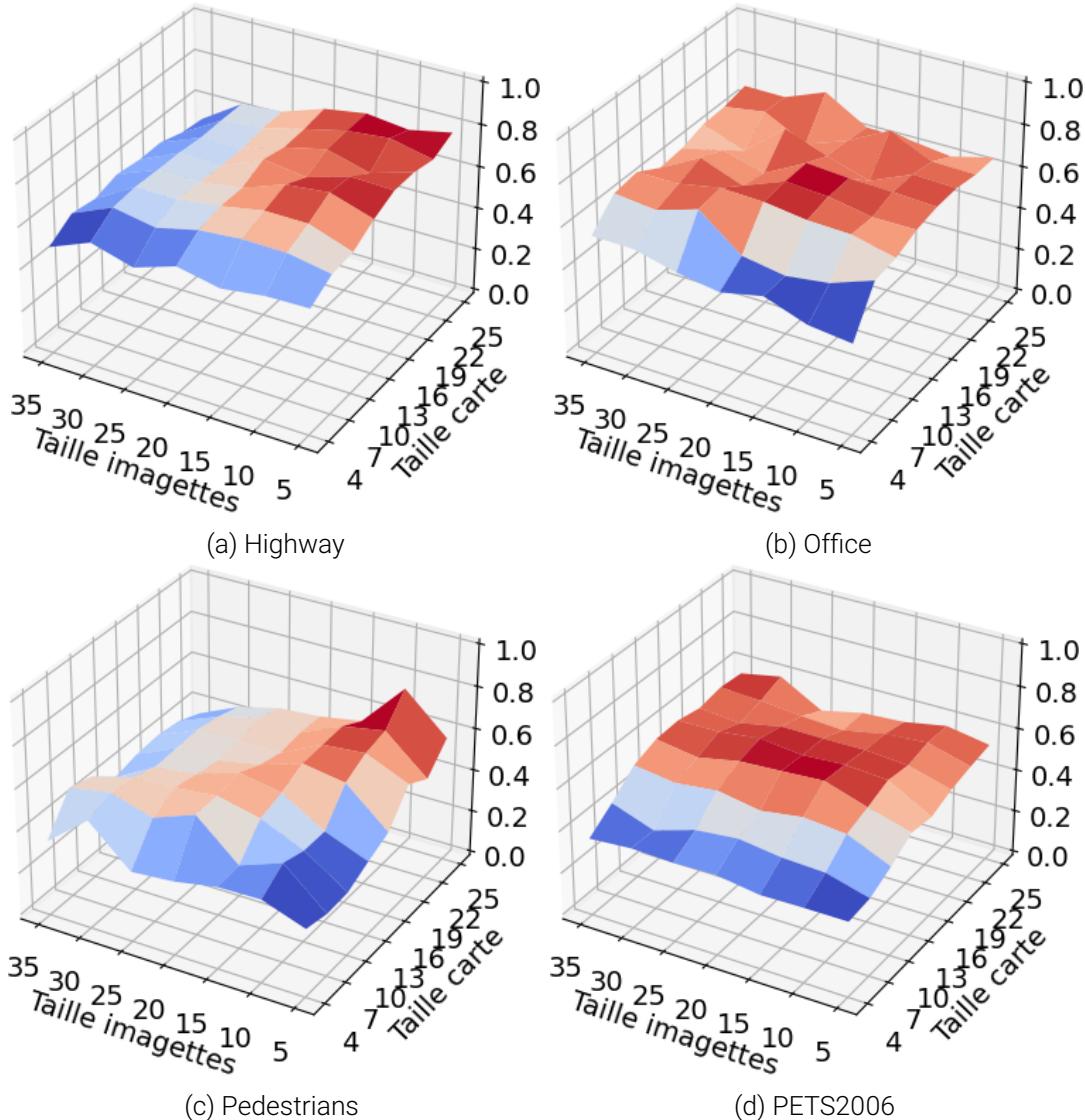


FIGURE 2.27 – F-mesure en fonction du nombre de neurones et de la taille des imagettes pour les séquences de la *baseline* avec un GNG. Les GNG n'ayant pas une topologie carrée comme les SOM, il suffit de mettre au carré la taille de la carte pour obtenir le nombre de neurones utilisés par le GNG.

Les résultats pour les GNG de la figure 2.27 montrent un comportement beaucoup plus chaotique. Les variations sont particulièrement grandes entre plusieurs exécutions, mais on peut tout de même discerner les mêmes préférences de tailles d'imagettes que pour les SOM. *Pedestrians* préfère encore des imagettes petites, et *PETS2006* des imagettes grandes. Cela semble indiquer que ces préférences ne sont pas dépendantes des modèles (SOM, GNG) mais bien seulement de la vidéo ou de la nouveauté présente dans la vidéo.

La variation du PSNR est montrée dans la figure 2.28. On peut y observer que les préférences pour la compression sont les mêmes pour toutes les vidéos, et donc différentes des préférences pour les F-mesures. La compression donne le meilleur résultat pour des petites imagettes de 5×5 et s'améliore aussi légèrement avec le

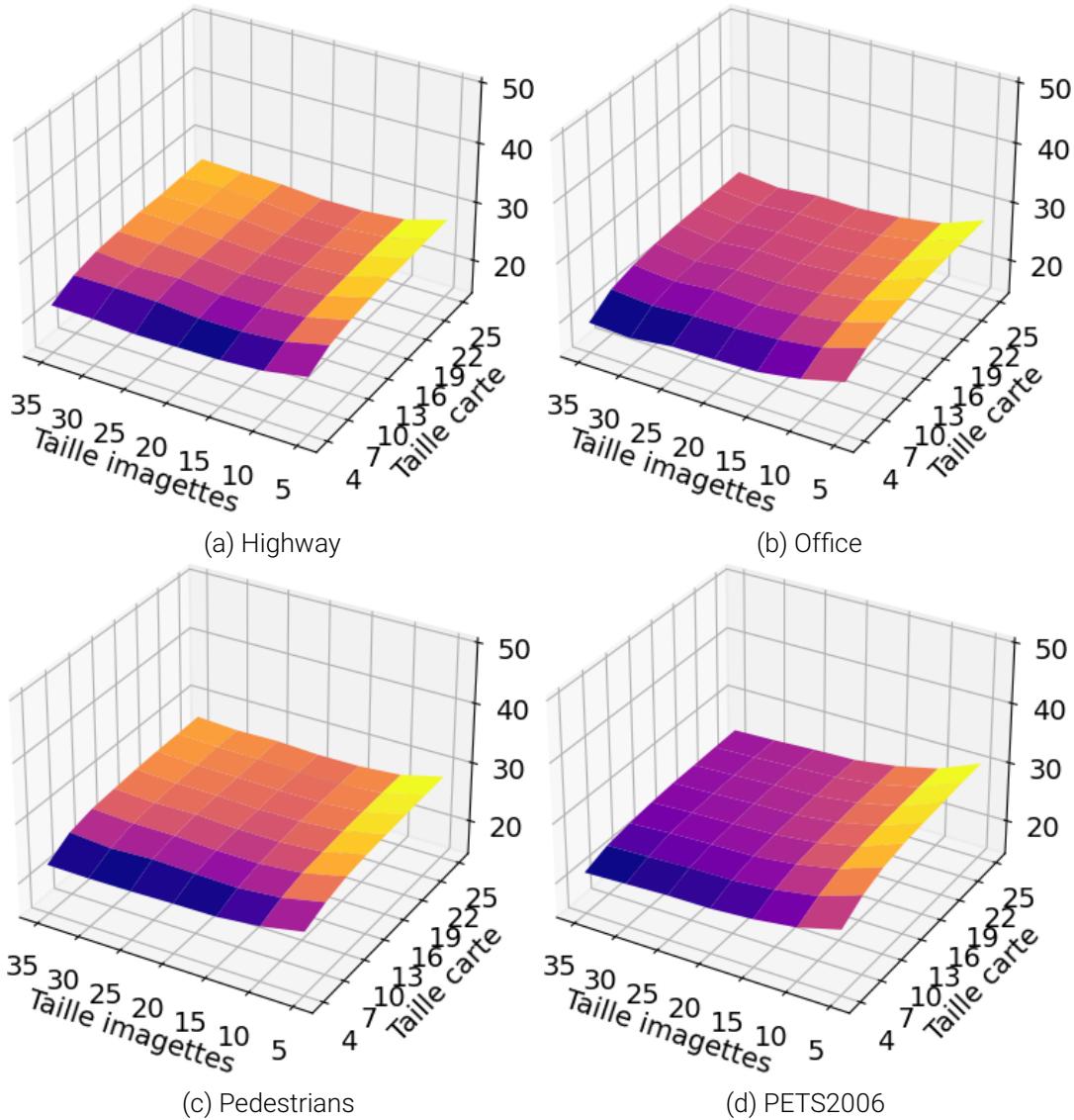


FIGURE 2.28 – PSNR en fonction du nombre de neurones et de la taille des imagettes pour les séquences de la *baseline* avec une SOM.

nombre de neurones. La conclusion est que la qualité de compression et la détection de nouveauté ne sont pas corrélées. Améliorer le résultat de la compression n'amènera ainsi pas forcément des gains au niveau de la détection de nouveauté.

Global

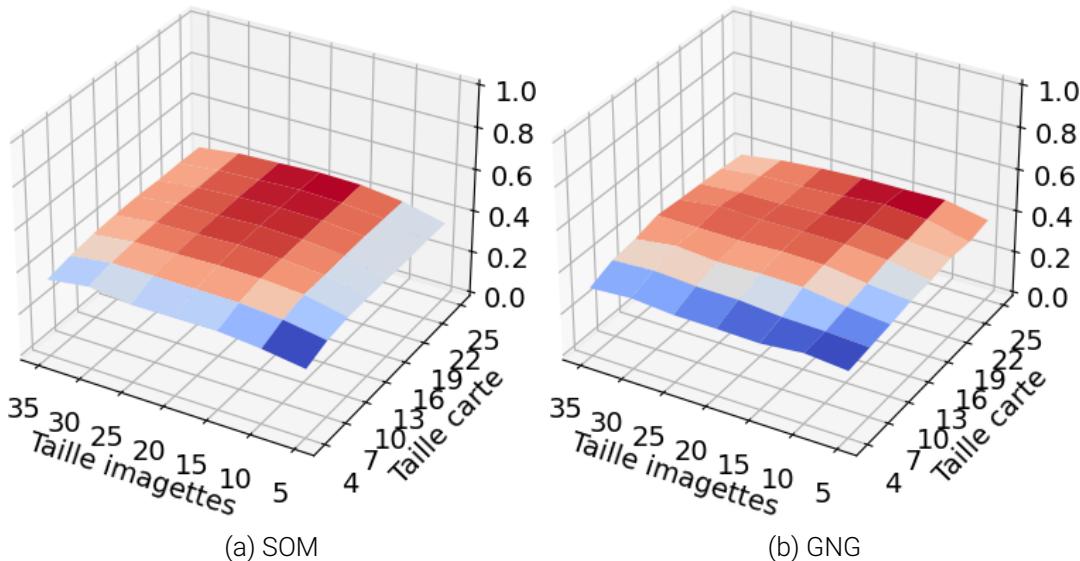


FIGURE 2.29 – Moyenne des F-mesure en fonction du nombre de neurones et de la taille des imagettes pour l'ensemble des vidéos de notre jeu de données.

Lorsque sont agrégés toutes les séquences vidéos, les graphes de la figure 2.29 montrent les tendances globales sur notre modèle. La détection de nouveauté avec la SOM fonctionne le mieux avec des tailles d'imagettes proches de 20×20 . Augmenter le nombre de neurones est quant à lui toujours bénéfique, mais amène des gains toujours plus réduits. Les résultats pour les GNG sont plus étranges, car ils sont à leur maximum pour des valeurs intermédiaires de neurones et d'imagettes, plus petites que celles de la SOM. Ce peut être cependant qu'un résultat de la forte variabilité observée dans les vidéos individuelles qui se répercute dans la moyenne globale.

2.5.3 Résultats complets

Les résultats complets sont présentés dans les tableaux 2.4 et 2.5. Les versions *Global* indiquent que l'on a choisi les valeurs de seuil, de taille de carte et d'imagettes qui étaient optimale sur l'ensemble de CDNET, c'est à dire 25×25 pour les tailles de carte et 20×20 d'imagettes et un seuil à 7 pour la SOM. Des imagettes de 15×15 pixels, 20×20 neurones et un seuil de 5 pour les GNG. Les version *Max* sont celles où on a pris la valeur optimale pour chaque vidéo.

La première chose que l'on peut remarquer, c'est que pour 8 catégories sur 10, le rappel est toujours meilleur que la précision. Pour seulement 13 vidéos sur 45, la précision est meilleure. Notre modèle semble donc montrer une préférence pour la sensibilité plutôt que la précision, bien que ce ne soit pas une règle absolue et que cela dépende des séquences évaluées. Comme par exemple avec *Tramstop* qui a une précision trois fois supérieure à son rappel.

	SOM			SOM Max	GNG	GNG Max
Categories/vidéos	Précision	Rappel	F-mesure	F-mesure	F-mesure	F-mesure
Baseline	59.2%	72.9%	63.9%	73.4%	67.0%	76.4%
Highway	63.9%	75.7%	69.3%	78.9%	75.9%	83.2%
Office	71.2%	71.0%	71.1%	74.4%	75.9%	77.8%
Pedestrians	41.4%	87.0%	56.1%	76.7%	60.0%	78.4%
PETS2006	60.1%	57.9%	59.0%	63.7%	56.2%	66.4%
Bad weather	32.6%	57.8%	38.9%	48.0%	34.8%	54.5%
Blizzard	19.4%	48.9%	27.8%	37.3%	14.4%	47.8%
Skating	72.5%	68.3%	70.3%	72.7%	74.3%	76.1%
SnowFall	17.1%	57.6%	26.4%	37.9%	24.9%	45.9%
WetSnow	21.5%	56.4%	31.1%	44.2%	25.7%	48.1%
Camera jitters	37.6%	65.2%	46.5%	51.7%	36.6%	52.3%
Badminton	49.2%	81.3%	61.3%	68.6%	62.1%	68.9%
Boulevard	47.9%	53.1%	50.4%	51.8%	34.7%	51.6%
Sidewalk	14.7%	45.4%	22.2%	27.4%	14.5%	30.9%
Traffic	38.5%	80.9%	52.1%	59.0%	35.1%	57.9%
Dynamic bkg	8.3%	67.1%	14.1%	30.1%	12.4%	40.0%
Boats	2.7%	79.8%	5.2%	34.2%	4.2%	62.8%
Canoe	16.7%	92.4%	28.2%	56.3%	26.0%	74.2%
Fall	8.8%	54.1%	15.2%	21.2%	8.5%	18.0%
Fountain01	1.1%	43.5%	2.1%	3.1%	1.9%	9.8%
Fountain02	3.1%	55.3%	5.9%	21.2%	5.0%	19.5%
Overpass	17.2%	77.8%	28.2%	44.6%	28.7%	55.7%
Night videos	27.8%	47.1%	32.1%	38.8%	32.0%	37.1%
BridgeEntry	10.8%	33.9%	16.4%	26.3%	12.9%	21.5%
BusyBoulvard	40.5%	28.9%	33.8%	39.5%	31.5%	33.1%
FluidHighway	19.1%	55.7%	28.5%	31.2%	29.6%	30.6%
StreetCornerAtNight	21.8%	59.7%	31.9%	39.7%	29.6%	35.7%
TramStation	47.9%	41.1%	44.2%	56.4%	49.5%	62.1%
WinterStreet	26.7%	63.4%	37.6%	39.8%	38.6%	39.7%
Shadow	48.6%	64.2%	52.8%	59.8%	49.7%	62.0%
Backdoor	27.1%	76.2%	40.0%	49.4%	36.6%	48.7%
Bungalows	54.9%	49.8%	52.2%	72.1%	63.0%	72.2%
BusStation	64.6%	67.3%	66.0%	66.1%	54.4%	66.3%
CopyMachine	77.7%	67.1%	72.0%	79.2%	58.9%	78.0%
Cubicle	17.0%	42.9%	24.3%	26.9%	19.5%	39.5%
PeopleInShade	50.0%	81.7%	62.1%	65.0%	65.7%	67.3%
Thermal	52.0%	66.2%	54.0%	65.5%	51.4%	66.5%
Corridor	32.4%	87.3%	47.2%	75.4%	26.4%	72.8%
DiningRoom	67.4%	55.7%	61.0%	70.8%	65.6%	72.5%
LakeSide	13.8%	53.0%	21.9%	33.4%	14.7%	30.4%
Library	85.9%	88.4%	87.1%	91.5%	87.8%	94.2%
Park	60.7%	46.4%	52.6%	56.4%	62.6%	62.8%
Global	41.3%	62.3%	43.5%	54.6%	41.9%	57.0%

TABLEAU 2.4 – Résultats complets sur CDNET de notre détection de nouveauté

	SOM Global			SOM Max	GNG Global	GNG Max
Categories/vidéos	Précision	Rappel	F-mesure	F-mesure	F-mesure	F-mesure
Turbulence	8.7%	69.6%	12.4%	35.1%	14.9%	36.9%
Turbulence0	1.5%	82.7%	3.0%	28.0%	2.9%	24.3%
Turbulence1	6.0%	68.2%	11.0%	34.1%	13.5%	34.6%
Turbulence2	1.3%	81.8%	2.6%	33.1%	0.9%	35.4%
Turbulence3	25.9%	45.6%	33.1%	45.1%	42.3%	53.1%
Inter. motion [r]	68.0%	46.7%	52.0%	69.0%	57.7%	68.5%
Sofa	61.1%	49.0%	54.4%	67.7%	58.4%	72.2%
StreetLight	64.6%	68.4%	66.4%	85.1%	77.4%	83.3%
Tramstop	78.4%	22.7%	35.3%	54.3%	37.1%	49.9%
Low framerate [r]	70.5%	66.4%	68.0%	74.3%	62.7%	75.4%
TramCrossroad	61.4%	72.9%	66.6%	68.7%	67.5%	69.3%
TunnelExit	77.3%	63.7%	69.9%	75.4%	48.1%	74.5%
Turnpike	73.0%	62.7%	67.4%	78.7%	72.5%	82.4%
Global	41.3%	62.3%	43.5%	54.6%	41.9%	57.0%

TABLEAU 2.5 – Résultats complets sur CDNET de notre détection de nouveauté - Suite

On peut également remarquer que notre modèle a beaucoup de mal dans les catégories avec un fond dynamique, c'est à dire avec des feuillages bruissants au vent ou des reflets dans l'eau, comme *Dynamic background* et *Turbulence* avec des précisions ne dépassant pas les 10%. Mais des légers mouvements de caméras ne posent pas trop de problèmes dans *Camera jitters*. Les autres catégories sont aux alentours de la moyenne globale de la SOM.

La version Max de la SOM présente des choses intéressantes. Elle gagne 10% de performance supplémentaire au global. Cela montre que des gains importants peuvent être faits en ajustant automatiquement le nombre de neurones, la taille des imagettes et le seuil par vidéo. Les gains les plus importants se trouvent dans les catégories dynamiques, avec *Dynamic background* où la F-mesure a été doublée et *Turbulence* où elle a été triplée. On suppose que le seuil est le facteur le plus influent dans ces cas là, car il peut permettre de supprimer une grande partie du bruit et ainsi d'améliorer la précision pour ces vidéos là.

Les GNG ont des performances similaires à la SOM, et réussissent et échouent dans les mêmes catégories. Au global, ils sont légèrement moins bon que les SOM avec 41,9% de F-mesure contre 43,5%. La version Max cependant arrive à 57%, soit au dessus de la version SOM Max à 54,6%. Cela provient peut-être de la plus forte variabilité observée dans les résultats des GNG, que l'on a par conséquent favorisé en choisissant le maximum pour chaque vidéo.

Comparaison avec l'état de l'art

CDNET étant un jeu de données très utilisé, il existe de nombreux algorithmes de détection de changement qui ont été testés sur celle-ci. Leur site web³ référence les résultats obtenus dans la littérature. Ceux-ci sont généralement obtenus en utilisant

3. <http://changedetection.net/>

du post processing pour maximiser les performances. Vu que nous n'en utilisons pas, nous avons choisi de se comparer à [Xu et collab. \[2016\]](#), qui est une review qui reproduit les résultats de méthodes communes de détection de changement, sans pré ni post processing. Nous avons reproduit leurs résultats dans le tableau 2.6.

Modèle	Précision	Rappel	F-mesure
SACON	46.3%	38.2%	24.0%
KDE	37.6%	68.7%	40.8%
CodeBook	61.2%	38.6%	41.1%
Vibe	65.3%	46.5%	47.2%
GMM	49.9%	62.3%	47.6%
AGMM	63.4%	56.0%	53.9%
PBAS	70.9%	50.8%	55.1%
SOBS	66.7%	60.1%	56.4%
GNG Global	39.6%	66.1%	41.9%
SOM Global	41.3%	62.3%	43.5%

TABLEAU 2.6 – Comparatif avec d'autres modèles de détection de changement sur CDNET

Notre méthode est avant dernière en précision, 2^{ème} ex-æquo en rappel et 6^{ème} sur 9 en F-mesure. Les chiffres indiqués ne sont cependant pas directement comparables, car nos valeurs ont été calculées sur une partie seulement de CDNET et pas toutes les vidéos, mais l'ordre de grandeur devrait environ être le même. En comparaison des autres modèles, la précision est le plus gros facteur limitant de notre approche. Il faut aussi remarquer que les modèles utilisés dans cette review sont tous "classiques", et le meilleur algorithme actuel BSUV-Net 2 de [TEZCAN et collab. \[2021\]](#), est un réseau de neurones qui obtient 83.9% en F-mesure sur CDNET, en utilisant de la *data augmentation*.

2.5.4 Visualisation des résultats

Dans cette section nous avons regroupé les résultats de détection de notre modèle. La figure 2.30 montre les résultats qui ont obtenu un bon score de détection. On peut y voir que le découpage en bloc de la SOM est présent dans le résultat de la détection. Cela peut expliquer en partie le rappel élevé et la précision plus faible car notre modèle aura tendance à déborder au-delà de la stricte nouveauté. On remarquera aussi qu'il n'y a pas de bruit dans nos détections. Le découpage entre zone de nouveauté et zone de fond est très net, même si il a quelques erreurs, comme une zone de nouveauté qui n'apparaît pas, ou une zone de fond qui apparaît comme nouveauté.

La figure 2.31 nous présente des cas de fond dynamiques où notre modèle a eu particulièrement du mal, notamment avec une précision très faible. On peut observer que le mouvement présent dans le fond est confondu avec de la nouveauté. Le mouvement étant permanent, cela cause un nombre très important de faux positifs, ce qui réduit significativement la précision.

La figure 2.32 présente un des défauts de normaliser la carte de saillance, comme expliqué dans la section 2.3.3.

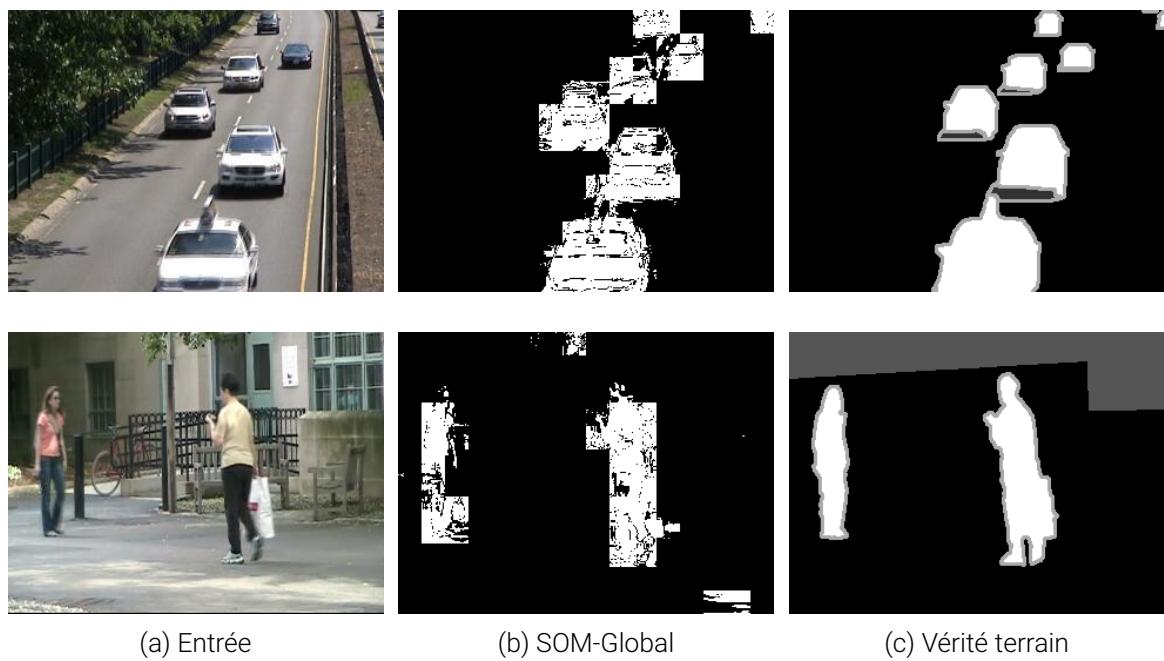


FIGURE 2.30 – Visualisations de résultats normaux de notre modèle. On peut y voir le découpage en bloc dû à l'utilisation d'imagettes et l'absence de bruit autour des nouveautés à détecter, mais avec quelques erreurs tout de même.

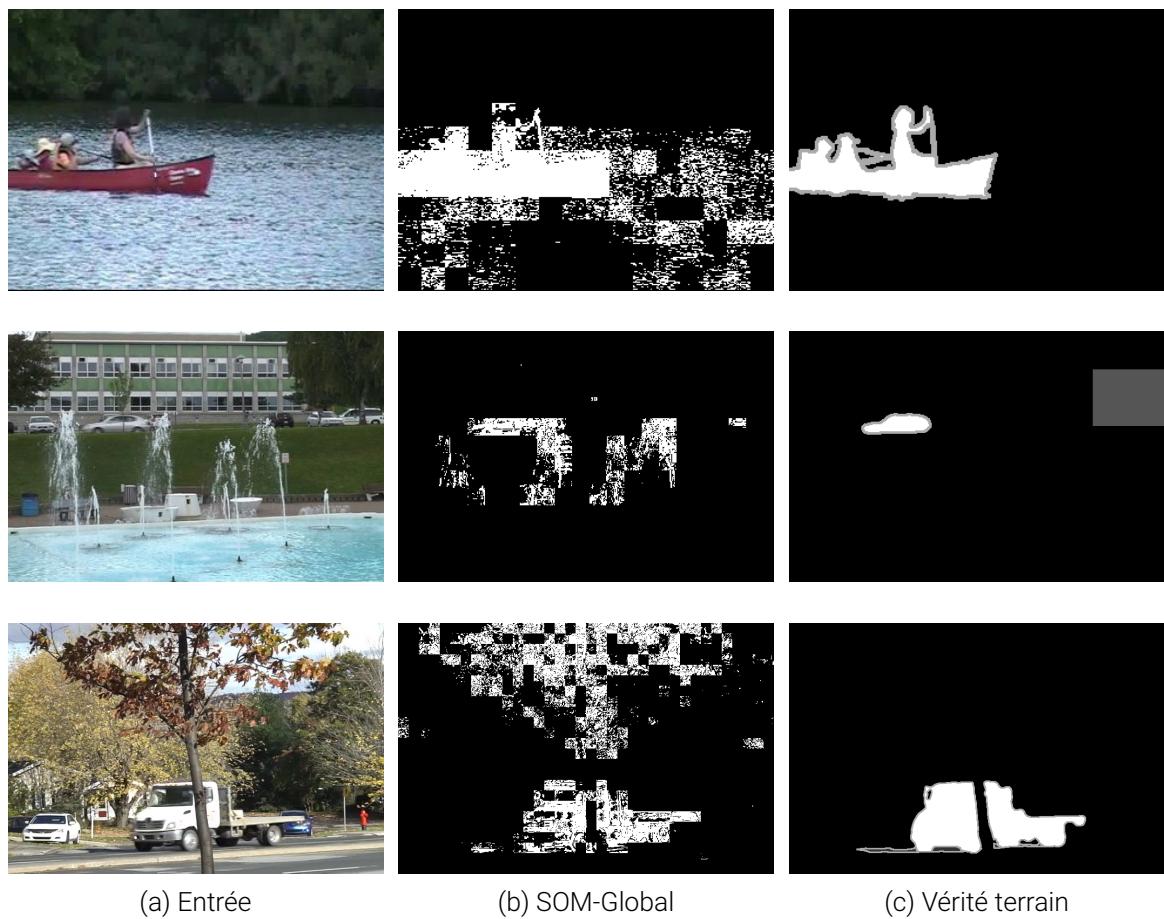


FIGURE 2.31 – Résultats sur des séquences avec un fond dynamique. Les changements trop importants du fond sont constamment considérés en nouveauté et ainsi réduisent drastiquement la précision.

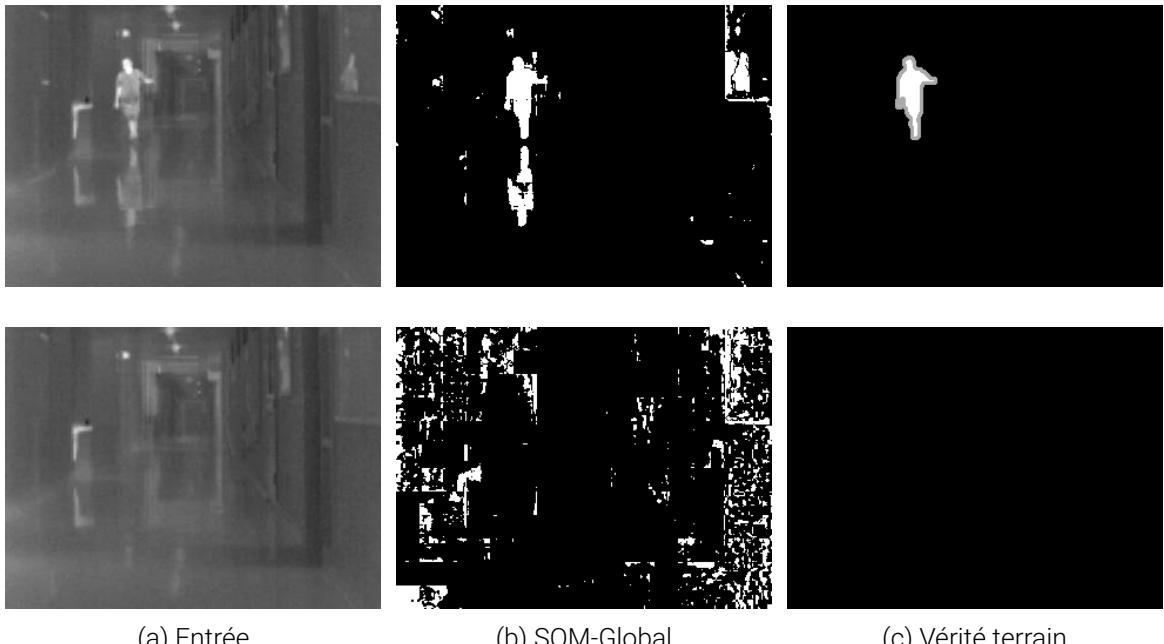


FIGURE 2.32 – Exemple d'un cas où le fait de normaliser la carte de saillance pose problème. Dans la première ligne, l'objet est correctement détecté avec les reflets absents de la vérité terrain, et avec très peu de bruit. La seconde ligne sans objet à détecter fait fortement ressortir le bruit à la place.

2.5.5 Interprétations

Les éléments que nous avons présentés montrent que la détection de nouveauté en utilisant la quantification vectorielle et la topologie d'une SOM fonctionne. Quantitativement parlant la précision de notre détection de nouveauté est loin derrière l'état de l'art, que ce soit des modèles classiques programmés ou des réseaux de neurones apprenants. Il faut cependant remarquer que la comparaison "pixel perfect" entre la vérité terrain et l'image de détection est particulièrement désavantageuse pour notre modèle, du fait d'une recherche de nouveauté au niveau des imagettes plutôt que du pixel. Si l'on ne compare que de façon binaire si un objet a été détecté ou non, nos résultats seraient probablement plus proches avec les autres méthodes.

Notre modèle étant particulièrement mauvais pour les fonds dynamiques, l'hypothèse que nous avons faite dans la section 2.2 était donc fausse. Cette hypothèse supposait que la SOM serait capable de généraliser un environnement présent dans l'image de fond, et que tout mouvement dans cet environnement ne serait pas saillant du fait qu'il ait été appris et généralisé par la SOM. En pratique, un mouvement dans une zone de fond entraîne des changements très importants sur tous les pixels d'une imagette, ce qui se traduit par une grande différence au niveau du calcul de la distance avec les neurones, même si visuellement les deux imagettes sont similaires. Par conséquent, les différences seront élevées lorsqu'un fond dynamique est présent, contrairement à ce que nous pensions.

La piste que nous considérons la plus prometteuse pour résoudre ce problème serait de modifier le calcul de distances dans la SOM pour mieux refléter les différences dans les images. Une mesure comme la Structural Similarity (SSIM) WANG et collab. [2004], qui est un calcul de différences mieux adapté pour les images, même

si elle ne suit pas les propriétés d'une distance. Une autre option plus neuronale serait d'essayer d'apprendre les types des différences qui peuvent se produire au cours du temps, comme les remous de l'eau par exemple, et ainsi ajouter une notion temporelle à notre modèle, qui n'en possède pas encore.

Nous avons aussi montré avec les GNG que notre approche est facilement généralisable à d'autres modèles de quantification vectorielle avec topologie. Des variantes de SOM ou de GNG pourraient ainsi être essayés pour cette tâche pour potentiellement amener à de meilleurs résultats que la SOM classique sur cette application.

2.6 Références

- AMERIJCKX, C., J.-D. LEGAT et M. VERLEYSEN. 2003, «Image compression using self-organizing maps», *Systems analysis modelling simulation*, vol. 43, n° 11, p. 1529–1543. [26](#)
- BERGSTRA, J., R. BARDET, Y. BENGIO et B. KÉGL. 2011, «Algorithms for hyper-parameter optimization», *Advances in neural information processing systems*, vol. 24. [44](#)
- BERNARD, Y., N. HUEBER et B. GIRAU. 2019, «Novelty detection with self-organizing maps for autonomous extraction of salient tracking features», dans *International Workshop on Self-Organizing Maps*, Springer, p. 100–109. [29](#)
- BERNARD, Y., N. HUEBER et B. GIRAU. 2020, «Novelty detection in images using vector quantization with topological learning», dans *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, IEEE, p. 1–4. [29](#)
- BIANCO, S., G. CIOCCHA et R. SCHETTINI. 2017, «Combination of video change detection algorithms by genetic programming», *IEEE Transactions on Evolutionary Computation*, vol. 21, n° 6, p. 914–928. [31](#)
- HRIPCSAK, G. et A. S. ROTHSCHILD. 2005, «Agreement, the f-measure, and reliability in information retrieval», *Journal of the American medical informatics association*, vol. 12, n° 3, p. 296–298. [40](#)
- HUYNH-THU, Q. et M. GHANBARI. 2008, «Scope of validity of psnr in image/video quality assessment», *Electronics letters*, vol. 44, n° 13, p. 800–801. [39](#)
- KORHONEN, J. et J. You. 2012, «Peak signal-to-noise ratio revisited : Is simple beautiful?», dans *2012 Fourth International Workshop on Quality of Multimedia Experience*, IEEE, p. 37–38. [39](#)
- MADDALENA, L. et A. PETROSINO. 2008, «A self-organizing approach to background subtraction for visual surveillance applications», *IEEE Transactions on Image Processing*, vol. 17, n° 7, p. 1168–1177. [24](#)
- POWERS, D. M. 2011, «Evaluation : from precision, recall and f-measure to roc, informedness, markedness and correlation», *arXiv preprint arXiv:2010.16061*. [40](#)
- TEZCAN, M. O., P. ISHWAR et J. KONRAD. 2021, «Bsuv-net 2.0 : Spatio-temporal data augmentations for video-agnostic supervised background subtraction», *IEEE Access*, vol. 9, p. 53 849–53 860. [57](#)

WANG, Y., P.-M. JODOIN, F. PORIKLI, J. KONRAD, Y. BENEZETH et P. ISHWAR. 2014, «Cdnet 2014 : An expanded change detection benchmark dataset», dans *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, p. 387–394.
[34](#)

WANG, Z., A. C. BOVIK, H. R. SHEIKH et E. P. SIMONCELLI. 2004, «Image quality assessment : from error visibility to structural similarity», *IEEE transactions on image processing*, vol. 13, n° 4, p. 600–612. [59](#)

Xu, Y., J. DONG, B. ZHANG et D. Xu. 2016, «Background modeling methods in video analysis : A review and comparative evaluation», *CAAI Transactions on Intelligence Technology*, vol. 1, n° 1, p. 43–60. [57](#)

Chapitre 3

Vision événementielle et attention

« *On ne voit que ce que l'on regarde.* »

Maurice Merleau-Ponty

Sommaire

3.1	Introduction	64
3.2	Détection de mouvements	64
3.2.1	Multi cibles	65
3.2.2	Paramètres	66
3.3	Mécanisme attentionnel	67
3.4	Combinaison avec la détection de nouveauté	67
3.5	Conclusion	67
3.6	Références	67

3.1 Introduction

3.2 Détection de mouvements

Le rôle de la première couche pour traiter les événements de la caméra sera de transformer les informations ponctuelles dans le temps et l'espace qu'envoie la caméra en zones de mouvements continues, facilitant le travail des couches supérieures. Pour ce faire, nous utiliserons des DNF, particulièrement appropriés par leur capacité d'intégration de l'information, et de leur robustesse au bruit très présent dans ce type de caméra. Habituellement, on utilise des DNF à deux dimensions pour gérer des images. Cependant cela représente 307 200 neurones avec notre caméra 640×480 , tous connectés dépendants les uns des autres, et devant être recalculés à chaque itération. Nous souhaitons que la première couche soit simple et rapide, alors nous avons choisi d'utiliser deux DNF à 1 dimension, un pour chaque axe.

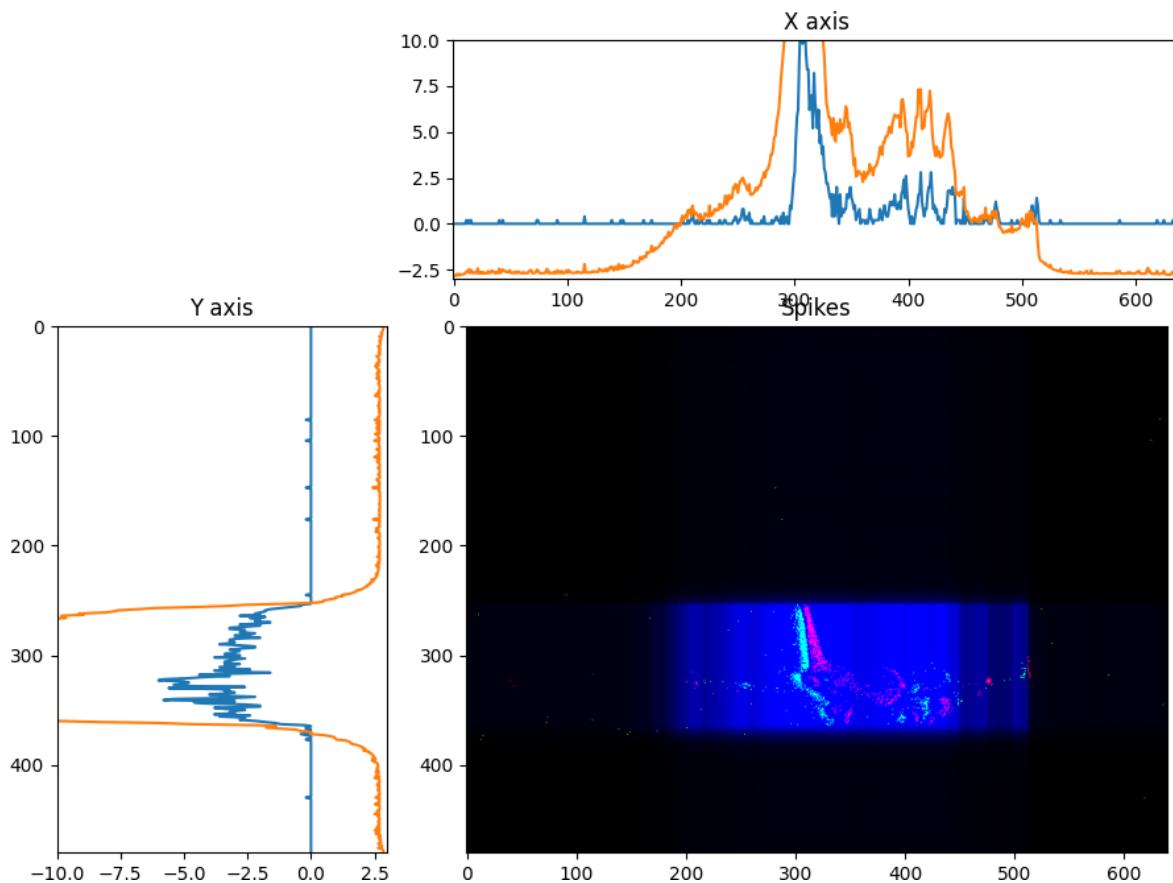


FIGURE 3.1 – Détection de mouvements avec DNF sur caméra événementielle. Les points verts et rouges dans l'image correspondent respectivement aux augmentations et réductions de luminosités détectées par la caméra. La somme des événements de chaque axe est affichée dans les courbes bleues qui sont en entrée des deux DNF. En orange sont les potentiels des DNF. On calcule le produit dyadique de la sortie des DNF, c'est à dire après la sigmoïde, pour obtenir la zone de détection de mouvement en bleu sur l'image.

Les DNF peuvent fonctionner à la fois en mode impulsionnel et en mode discrétilisé. Nous avons choisi le second mode, car il est plus rapide à calculer et plus stable.

Sa vitesse ne dépend pas du nombre d'événements à traiter, contrairement au modèle impulsionnel. Il nous a donc fallu intégrer les évènements avec un pas de temps que nous avons mis à 10 ms. Il est aussi possible de le placer à 1 ms ou en dessous, si l'on souhaite détecter des mouvements très rapides. Il y aura juste plus de mises à jour par seconde des potentiels du DNF, et donc plus de calculs. Nous n'avons pas utilisé la polarité des évènements, uniquement si un pixel détectait un changement ou pas.

Les événements que nous recevons de la caméras sont très bruités. Il existe notamment certains pixels défectueux qui envoient en permanence des événements. Il s'agit de bruit artificiel dû à la technologie et non présent dans la nature. Il n'est donc pas particulièrement intéressant pour nous de l'inclure dans notre étude. Nous avons donc désactivé logiciellement ces pixels défectueux. Le bruit présent dans la nature et plus diffus, a lui été conservé. Le résultat est présenté sur la figure 3.1.

3.2.1 Multi cibles

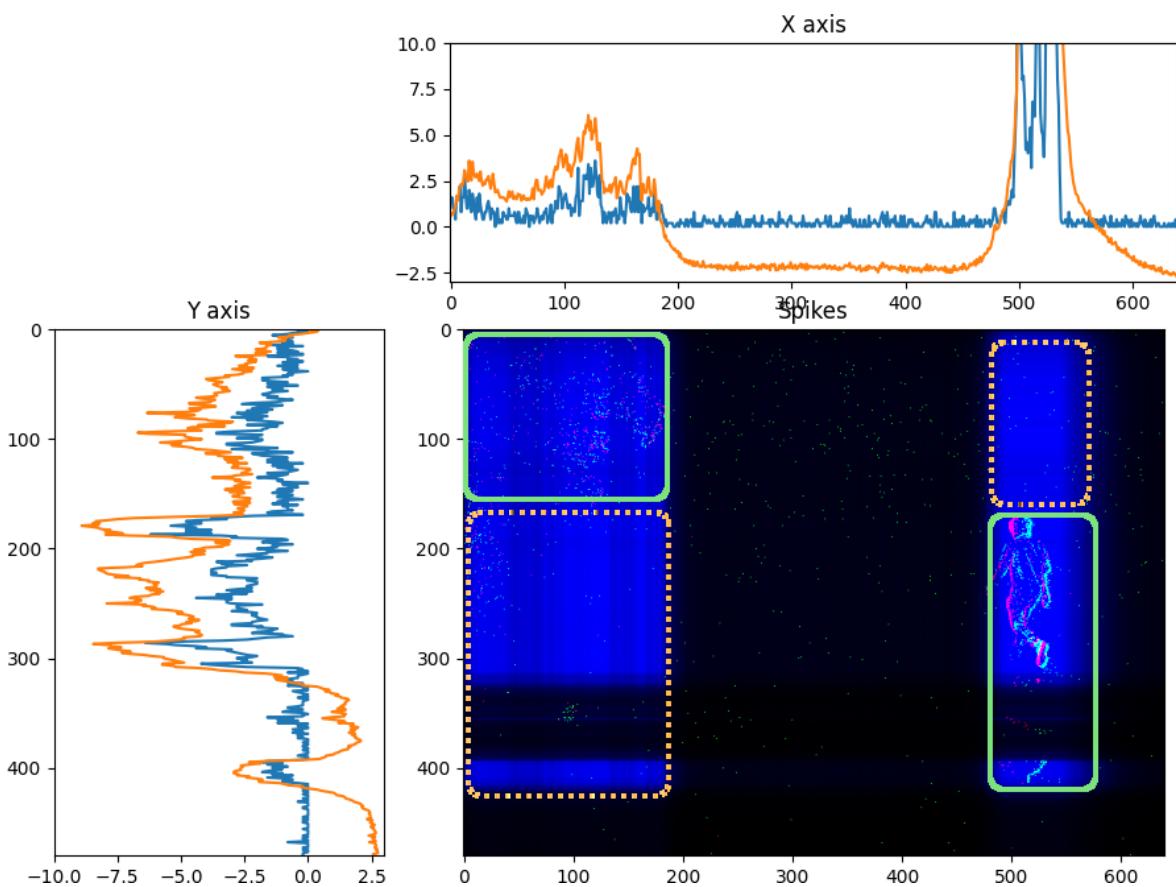


FIGURE 3.2 – Détection multicable. Il y a deux stimulus dans la scène, entourés par des lignes vertes. La personne qui se déplace en bas à droite et un mouvement dans un arbre en haut à gauche. Les activations parasites, dû à la projection de deux détections en 1D vers du 2D sont entourés par des pointillés orange.

Notre système est également capable de suivre de multiples cibles, mais au prix de détections parasites. En effet, passer de 2 dimensions à 2 fois 1 dimension pour ensuite projeter à nouveau en 2 dimensions est efficace en calculs, mais il y a une

perte d'informations dans le processus. Lorsque l'on passe en 1 dimensions, on ne connaît que les valeurs sur chaque axe des deux stimulus, sans pouvoir les associer entre elles. Un exemple est présenté sur la figure 3.2.

3.2.2 Paramètres

Le paramétrage de DNF est une tâche délicate. Nous avons ajusté à la main les valeurs de chaque paramètre à l'aide de nos connaissances sur ces modèles. Bien que nous n'ayons pas beaucoup modifiés les paramètres entre les deux exemples que nous avons présentés, il est possible qu'une séquence nouvelle ne fonctionne pas avec les paramètres que nous avons sélectionnés, et qu'il faille reparamétrer les DNF pour que cela fonctionne comme attendu. Les deux DNF de chaque axe utilisent les mêmes paramètres. Chaque événement compte pour 0.2 dans l'entrée des DNF pour une meilleure visualisation sur les figures, car mettre cette valeur à 1 et diviser le gain par 5 aurait eu le même effet sur le DNF.

TABLEAU 3.1 – Paramètres DNF 1D

Paramètre	Figure 3.1	Figure 3.2
δt	0.01	0.01
τ	0.05	0.05
Coef. exciteur	4.5	4.5
σ exciteur	0.01	0.01
Gain	4	1.5
Repos h	-3	-3

3.3 Mécanisme attentionnel

3.4 Combinaison avec la détection de nouveauté

3.5 Conclusion

3.6 Références

Chapitre 4

Calcul de Best Matching Unit accéléré avec la topologie

« Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher. »

Antoine de Saint-Exupéry

Sommaire

4.1	Introduction	70
4.2	Algorithme séquentiel	72
4.2.1	Analyse de la complexité	76
4.2.2	Protocole expérimental	77
4.2.3	Résultats	78
4.3	Algorithme parallèle	82
4.4	Conclusion	82
4.5	Références	82

4.1 Introduction

Self-Organizing Maps (SOM) are well-known unsupervised neural networks able to perform vector quantization while mapping an underlying regular neighbourhood structure onto the codebook. They are used in a wide range of applications. As with most properly trained neural networks models, increasing the number of neurons in a SOM leads to better results or new emerging properties. Therefore highly efficient algorithms for learning and evaluation are key to improve the performance of such models. In this paper, we propose a faster alternative to compute the Winner Takes All component of SOM that scales better with a large number of neurons. We present our algorithm to find the so-called best matching unit (BMU) in a SOM, and we theoretically analyze its computational complexity. Statistical results on various synthetic and real-world datasets confirm this analysis and show an even more significant improvement in computing time with a minimal degradation of performance. With our method, we explore a new approach for optimizing SOM that can be combined with other optimization methods commonly used in these models for an even faster computation in both learning and recall phases.

Self-organizing maps (SOM) are widely used algorithms that feature vector quantization with dimensionality reduction properties. An explanation of how they work can be found in ?. They are used in numerous fields like image processing, automatic text and language processing, and for visualization, analysis and classification of all kinds of highly dimensional datasets. Many applications examples are depicted in ?. However, the amount of computations required by SOMs linearly increases with the number of neurons, the number of elements in the dataset and the dimensionality of the input, in both learning and recall phases. Therefore applying SOMs on datasets with huge numbers of elements and with a high number of neurons to precisely represent the input induces a significant computational cost that may exceed some constraints such as real-time computation or low power consumption.

With the goal of reducing the required computational time of SOMs in mind, variants of the classical SOM algorithm have been developed. The most well-known SOM modification is the Batch Learning algorithm, as explained in ?. Contrary to the classical online learning, the batch learning averages the modifications over multiple training vectors before updating the neurons weights. Similar efforts have been made in ? or in ?. However, all those variants are only focusing on reducing the convergence time of the SOM training. To the best of our knowledge, no work has been carried out to reduce the time required for each iteration. This can be partially explained by the highly parallel nature of the computations inside each iterations, in so far as when a fast real world implementation is required, parallel solutions are proposed, like the use of an FPGA substrate with each neuron having its own circuitry, as in ? and in ?. However, parallel solutions should not lead to a lack of effort in optimizing the algorithms, as the majority of SOM training is performed on CPU, and parallel hardware can be costly and difficult to program. Furthermore one can parallelise multiple iterations within an epoch instead of inside the iteration itself, and therefore can benefit from our improvements on parallel hardware.

A SOM training iteration consists of two major parts, a competitive part which searches the Best Matching Unit (or winner neuron), and a cooperative part which updates all the neurons weights proportionally to their distance with the BMU. In the

classic SOM, both steps have the same algorithmic complexity (number of neurons multiplied by the dimensionality of the data) and take roughly the same time (depending on implementation details). In this paper we focus on improving the competitive part by reducing the number of neurons evaluated to find the BMU. This optimization also applies to the recall phase.

After a brief description of the standard SOM model in section ??, section ?? defines the proposed method to speed up the computation of BMU, before analyzing its computational complexity. The experimental setup is described in section ??, and the corresponding results are discussed in section ??.

We present here our new algorithm for searching the Best Matching Unit (BMU) in the SOM. It is traditionally determined by means of an exhaustive search by comparing all the distances between the input vector and the neurons weights. This method, while able to always find the correct minimum of the map, is computationally inefficient. The dimensionality reduction property of the SOM means that close neurons in the SOM topology represent close vectors in the input space. Therefore when an input vector is presented to the SOM, a pseudo-continuous gradient appears in the SOM when considering the distances between the neuron weights and the input vector whose minimum is located at the BMU. Using some kind of discretized gradient descent approach, we expect that this distance will progressively reduce until the Best Matching Unit is reached, with the minimal distance in the map.

4.2 Algorithme séquentiel

Pour bien comprendre le fonctionnement de notre algorithme Fast-BMU, nous allons le présenter partie par partie. Nous commencerons par l'idée générale de la descente du gradient de la SOM. Puis nous aborderons les cas spéciaux en expliquant pourquoi ils existent, les conséquences possibles et comment Fast-BMU les résoud.

Algorithme 1 : FastBMU

Entrées :

`vecteur` : vecteur d'entrée
`l, h` : largeur et hauteur de la SOM

Données :

`positions` : positions de départ des particules
`mémoire_dist` : historique des distances déjà évaluées
`resultats` : dictionnaire vide

Sorties :

`bmu` : index de la Best Matching Unit

début

```

positions ← [(0,0),(l-1,0),(0,h-1),(l-1,h-1)]
pour pos dans positions faire
    bmu_particule = Particule(vector, pos)
    resultats[bmu_particule] ← mémoire_dist[bmu_particule]
fin
bmu = minimum de resultats par valeurs
retourner bmu
fin
```

La partie

Dans notre algorithme, nous utilisons cette réduction progressive des distances pour effectuer une descente de gradient discrétisée afin de trouver la meilleure unité de correspondance. Le pseudo-code est présenté dans l'Algorithme ??, et illustré dans la figure 4.1 (à gauche). L'algorithme commence à une position choisie arbitrairement dans la carte. Pour cet exemple, considérons qu'il commence aux coordonnées (0,0) (coin supérieur gauche). Ce neurone a deux voisins, un à l'est (1,0) et un au sud (0,1). Nous évaluons les distances au vecteur d'entrée de ces trois neurones pour trouver la prochaine étape de la descente du gradient. Si la plus petite distance est trouvée là où nous sommes actuellement positionnés, alors nous considérons qu'il s'agit d'un minimum local. Au contraire, si l'un des neurones voisins donne la plus petite distance, alors la descente du gradient va dans la direction de ce neurone particulier, qui devient la prochaine position sélectionnée à partir de laquelle nous répétons ce processus jusqu'à ce que nous trouvions un minimum local. Dans notre exemple, la plus petite distance est mesurée pour le neurone oriental en (1,0). Le processus de recherche se déplace donc d'un pas vers l'est, et ce neurone a trois voisins, un au sud (1,1), un à l'est (2,0) et un à l'ouest dont nous venons (et que nous ignorons donc). Nous comparons à nouveau les trois distances et nous nous déplaçons vers la distance la plus faible (sud).

Algorithme 2 : Particule

Entrées :

vecteur : vecteur d'entrée

pos : position de la particule

Données :

mémoire_dist : historique des distances déjà évaluées

Sorties :

bmu : index de la Best Matching Unit potentielle

début

```
// On commence par regarder les voisins directs pour trouver si l'un
// d'entre eux est plus proche du vecteur d'entrée.
bmu_voisin ← RechercheVoisin(vecteur, pos, 1)
si mémoire_dist[bmu_voisin] < mémoire_dist[pos] alors
    retourner Particule(vecteur, bmu_voisin)
fin
// Si il n'y en a pas, alors on se trouve dans un minimum local.
// On effectue alors une nouvelle recherche dans le voisinage de 2.
bmu_voisin ← RechercheVoisin(vecteur, pos, 2)
si mémoire_dist[bmu_voisin] < mémoire_dist[pos] alors
    retourner Particule(vecteur, bmu_voisin)
fin
// Si on ne trouve toujours rien, alors on peut considérer que l'on a atteint
// la fin de la descente de gradient.
retourner pos
fin
```

Algorithme 3 : RechercheVoisin

Entrées :

vecteur : vecteur d'entrée

pos : position de la particule

x : distance topologique

Données :

resultats : dictionnaire vide

prototypes : valeurs des vecteurs prototypes de la SOM

memoire_dist : historique des distances déjà évaluées

Sorties :

bmu : index de la Best Matching Unit potentielle

début

```

pour voisin dans le voisinage à distance topologique x de pos faire
    si memoire_dist [voisin] est vide alors
        memoire_dist [voisin] ← distance(prototypes [voisin],
        vecteur)
        resultats [voisin] ← memoire_dist [voisin]
    fin
fin
    bmu_voisin ← minimum de resultats par valeurs
    retourner bmu_voisin
fin
```

Ce processus itère jusqu'à atteindre un minimum local à la position (6,5), où tous les neurones voisins ont une distance au vecteur d'entrée plus élevée que le neurone sélectionné. Afin de s'assurer que le minimum local qui a été trouvé est le meilleur dans le voisinage local, nous effectuons ensuite une recherche de l'espace local en continuant la descente du gradient à partir de tous les neurones directement voisins qui sont encore inexplorés. Si cette recherche trouve un meilleur minimum local, alors ce minimum local est considéré comme le BMU. Cette partie de l'algorithme vise à contourner les problèmes qui peuvent découler de la topologie. Dans une topologie de grille par exemple, nous ne pouvons regarder que les axes orthogonaux. Mais parfois, le gradient est orienté vers une direction diagonale, et en étendant la recherche à partir d'un minimum local, nous sommes en mesure de suivre ce gradient vers un BMU potentiel.

In our algorithm we use this progressive reduction of distances to perform a discretized gradient descent to find the Best Matching Unit. The pseudo-code is shown in Algorithm ??, and illustrated in figure 4.1 (left). The algorithm starts at an arbitrarily selected position in the map. For this example let us consider that it starts at coordinates (0,0) (top left corner). This neuron has two neighbours, one to the east (1,0) and one to the south (0,1). We evaluate the distances to the input vector of these three neurons to find the next step in the gradient descent. If the smallest distance is found where we are currently positioned, then we consider this is a local minimum. On the contrary if one of the neighbouring neurons gives the smallest distance, then the gradient descent goes in the direction of this particular neuron, that becomes the next

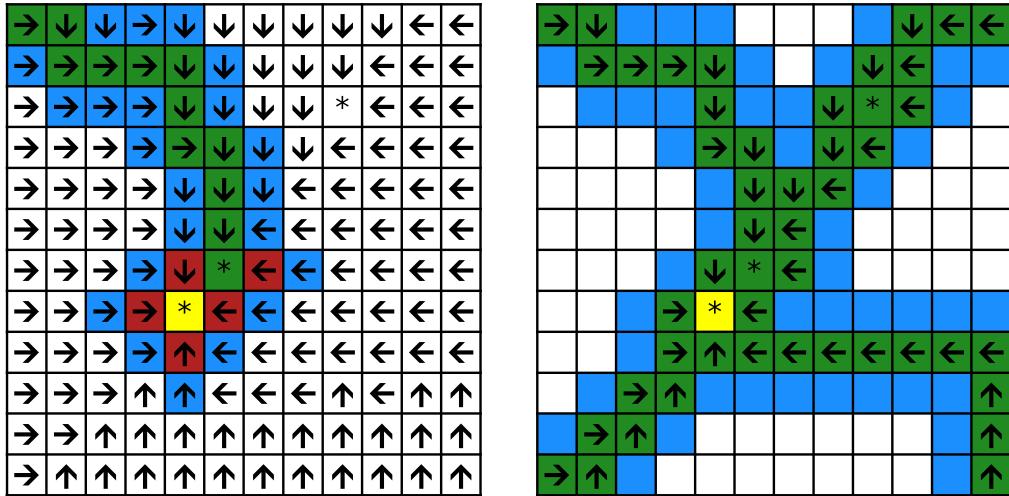


FIGURE 4.1 – Left : Example of the execution of one particle. Each cell represents one neuron. The arrow in each cell points towards the best neighbour (where a lower distance can be found). A star represents a local minimum. The green colored cells are the neurons who have been explored during the search, and the blue cells are the neurons whose distance to the input has been computed but that have not been explored by the algorithm. After having found a local minimum, new particles are created (in red) and continue the gradient descent by exploring the local neighbourhood. The cell in Yellow is the BMU, and the global minimum of the map. Right : Execution of the four particles.

selected position from which we repeat this process until we find a local minimum. In our example the smallest distance is measured for the eastern neuron in (1,0). Thus the search process moves one step to the east, and this neuron has three neighbours, one to the south (1,1), one to the east (2,0) and one to the west that we come from (and thus we ignore it). We compare the three distances again, and move towards the lowest distance (south).

This process iterates until it reaches a local minimum at position (6,5), where all neighbouring neurons have a higher distance to the input vector than the selected neuron. In order to ensure that the local minimum that has been found is the best one in the local neighbourhood, we then perform a search of the local space by continuing the gradient descent from all directly neighbouring neurons that are still unexplored. If this search finds a better local minimum, then this local minimum is considered as the BMU. This part of the algorithm aims at circumventing problems that may arise from the topology. In a grid topology for instance, we can only look at orthogonal axes. But sometimes, the gradient is oriented towards a diagonal direction, and by extending the search from a local minimum, we are able to still follow this gradient towards a potential BMU.

Another problem that can arise with the particle algorithm is edge effects. When mapping high dimensional data onto a 2D map, the map sometimes become twisted no gradient between the top left neuron and the bottom right neuron. Imagine a net that is thrown onto a ball and nearly completely enveloping it : the shortest path between one corner of the net and the opposite corner of the net does not follow the mesh of the net. If in the learning phase, the particle ends up in the wrong corner of the SOM, it will find a terrible BMU and the following weight update will locally break the neighbourhood continuity, and create a negative feedback loop that will remove

all dimensionality reduction properties from the SOM. Thankfully there is an easy way to avoid this, by simply starting 4 particles, one in each corner of the SOM, and selecting the smallest BMU found. This technique preserves the continuity of the SOM and makes the Fast-BMU algorithm more robust to local discontinuities in the gradient, especially at the beginning of the training process, when the map is not yet well unfolded. In the case where there is no clear gradient present in the SOM (typically when the SOM is randomly initialized before training), this algorithm will fail to find the correct BMU. However it is not a problem at the start of training as the neighborhood function will create a gradient by unfolding the SOM regardless of which neuron is initially selected. The result of a full Fast-BMU execution is shown in figure 4.1 (right).

4.2.1 Analyse de la complexité

Le temps d'exécution d'une recherche de BMU dans la SOM classique dépend de deux facteurs : le nombre de neurones multiplié par la durée d'une comparaison entre le vecteur d'entrée et les poids d'un neurone. L'idée de Fast-BMU étant de réduire le nombre de ces comparaisons, la durée d'une comparaison ne sera pas prise en compte dans cette analyse. C'est normalement une valeur constante qui ne dépend que de la dimensionnalité des données. Le nombre de comparaisons pour Fast-BMU est quand à lui dépendant de la taille de la carte, c'est à dire hauteur et largeur, et de la topologie, c'est à dire du nombre de voisins par neurone.

Nous ferons dans cette analyse l'hypothèse qu'il y a une certaine continuité dans la carte et qu'en partant de n'importe quel point de la carte, on atteigne la BMU en suivant le gradient. C'est une hypothèse forte, mais qui doit être vraie pour que notre algorithme de Fast-BMU fonctionne. En pratique, Fast-BMU est plus robuste que cela, et n'a pas besoin que tous les gradients de la carte mènent à la BMU, il en faut juste assez pour qu'une *particule* la trouve. De plus, les problèmes les plus communs dans le gradient de la carte sont des minimums locaux, qui réduisent le nombre de pas en arrêtant la *particule* avant la fin. Pour augmenter le nombre de pas effectués par une particule, il faudrait un gradient qui prenne une direction à l'opposé de ce qu'il avait précédemment. Qu'il fasse des méandres sur son trajet par exemple, ce qui est inhabituel dans des SOM en général. Ainsi, la complexité que nous allons calculer sera probablement une limite haute à la vraie complexité de l'algorithme Fast-BMU.

Pour estimer la complexité de notre algorithme, c'est à dire le nombre de comparaisons dans notre cas, nous devons estimer le nombre de pas de chaque particule pour trouver la BMU potentielle. En considérant que la BMU est localisée à la position x, y de la SOM, on a :

- La *particule* en haut à gauche commençant aux coordonnées $0, 0$ devra faire x pas horizontaux et y pas verticaux pour atteindre la BMU.
- De la même façon, la *particule* du haut à droite avec les coordonnées $(w, 0)$ devra faire $w - x$ pas horizontaux et y pas verticaux.
- Pour la *particule* en bas à gauche $(0, h)$, ce sera x et $h - y$ pas.
- Pour la *particule* en bas à droite (w, h) , ce sera $w - x$ et $h - y$ pas.

Pour obtenir le nombre total de pas pour une itération, nous additionnons les nombres de pas de toutes les particules ensemble comme montré dans l'équation 4.1.

$$\begin{aligned} \text{NbrPas} &= (x + y) + (w - x + y) + (x + h - y) + (w - x + h - y) \\ &= 2x - 2x + 2y - 2y + 2w + 2h \end{aligned} \quad (4.1)$$

Les x et y s'annulant, le nombre total de pas ne dépend donc que de la largeur et la hauteur de la carte, soit $(2w + 2h)$. On peut donc en déduire la Complexité Estimée \mathcal{C} avec l'équation 4.2

$$\mathcal{C}(w, h) = 2 \times (w + h) \times \text{NbrEvalParPas} \quad (4.2)$$

avec w, h la largeur et la hauteur de la SOM respectivement et NbrEvalParPas le nombre de nouvelles distances à calculer pour chaque pas d'une particule. Sa valeur dépend de la topologie. Elle est au maximum de 3 avec une grille (4 voisins moins le neurone de l'étape précédente) et également de 3 avec une topologie hexagonale (6 voisins moins le neurone de l'étape précédente et 2 neurones qui étaient voisins du neurone précédent et qui ont donc déjà été évalués).

D'un point de vue analytique, nous pouvons estimer que notre algorithme Fast-BMU ($\mathcal{C}(w, h) = 6(w + h)$ dans le pire des cas dans une configuration de grille standard) est significativement plus rapide que l'algorithme actuel de recherche exhaustive ($O(w, h) = wh$) lorsque le nombre de neurones dans la carte est important. Par exemple, il est théoriquement deux fois plus rapide avec des SOM de 24 par 24, et 10 fois plus rapide avec des SOM de 120 par 120. Une évaluation expérimentale de la différence de vitesse est présentée dans la section 4.2.3.

4.2.2 Protocole expérimental

Pour évaluer la robustesse de notre algorithme avec différents types de données, nous avons sélectionné 6 jeux de données représentatifs du type de données sur lesquelles les SOM sont habituellement entraînés. Pour le cas 2D, nous avons généré des données avec différentes propriétés (une distribution uniforme et une forme hautement non convexe), pour les données 3D nous utilisons une forme cubique uniformément distribuée et les valeurs de couleur des pixels d'une image. Pour les hautes dimensions, nous considérons l'apprentissage par imagettes d'une image déjà présenté, avec des sous-images de 10 par 10 comme vecteurs d'entraînement (100 pixels avec 3 couleurs chacun, ce qui donne 300 dimensions), ainsi que le Free Spoken Digits Dataset [JACKSON et collab. \[2018\]](#) qui utilise des ondes sonores que nous avons réduites à 1000 dimensions. Nous emploierons des SOM à 2 dimensions avec des topologies en grille et hexagonales, car ce sont les plus utilisées.

Afin d'évaluer les différences entre tous les modèles testés, nous avons utilisé deux métriques. La première est la *Mean Squared Quantization Error* (MSQE) que nous avons déjà utilisée dans le chapitre 2. MSQE mesure la qualité de quantification vectorielle de l'algorithme testé. La seconde métrique est la *Mean Squared Distance to Neurons* (MSDtN) qui calcule la distance moyenne au carré entre les prototypes des neurones et ceux de leurs voisins directs. Plus cette valeur est faible, plus les neurones

voisins sont proches dans l'espace d'entrée et plus la propriété de réduction dimensionnelle est bonne. De nombreuses métriques existent dans la littérature sur les SOM pour la qualité de la réduction dimensionnelle d'une SOM, mais MSDtN a l'avantage d'être facile à calculer, sans paramètres et de ne dépendre que des poids des neurones. Pour toutes les métriques, une valeur plus faible est meilleure.

$$\text{MSQE} = \frac{1}{dn} \sum_{i=0}^{n-1} (v_i - u_i)^2 \quad (4.3)$$

$$\text{MSDtN} = \frac{1}{dN} \sum_{i=1}^N \sum_{j=1}^N \begin{cases} (w_i - w_j)^2, & \text{Si } \text{dist}(i, j) = 1 \\ 0, & \text{Sinon} \end{cases} \quad (4.4)$$

Avec n étant le nombre de vecteurs dans le jeu de données, d leur dimensionnalité, v_i les poids du $i^{\text{ème}}$ vecteur d'entrée, et $\text{bm}(i)$ l'indice du neurone qui est la *best matching unit* de v_i . De même, N est le nombre de neurones de la SOM et w_i les poids du $i^{\text{ème}}$ neurone, tandis que $\text{dist}(i, j)$ est le nombre de connexions dans le chemin le plus court entre les neurones i et j dans la carte neuronale.

4.2.3 Résultats

Performances

In this section, we explore the differences in quality of learning and recall between the standard version and our fast version for the computation of the BMU in a SOM. We also look at the practical differences in the amount of computations required for the two versions and we compare it with the previously presented complexity analysis.

Quality tests were performed on a 32×32 SOM (1024 neurons). For each combination of dataset and model (choice of topology and BMU algorithm), we ran 50 executions with different random seeds which affect the datasets that are generated, the initialization of the neuron weights (who are all randomly initialised with no pre-existing gradient) and the order in which the training vectors are presented. Results are shown in table ??.

The algorithm column of the table shows the combination of BMU finding algorithm and topology that was used for training. The MSDtN (see section ??) is calculated on the trained neurons weights. The MSQE_S (Standard) is the quantization error in the recall phase (after learning) when using the standard BMU algorithm. Comparing the different MSQE_S values for a standard version and a fast version gives an indication of the influence of the Fast-BMU algorithm on training quality only, as it always selects the real BMU in the recall phase. The MSQE_F (Fast) metric measures the vector quantization error with the BMU selection done by the Fast-BMU algorithm. If the training was performed on the standard SOM, it gives an indication of the influence of the Fast-BMU algorithm on recall accuracy only; if it was trained on a Fast version, it represents the MSQE result of a SOM that only uses the Fast-BMU algorithm. The mismatch column gives the proportion of BMU that are selected differently by the two algorithms.

TABLEAU 4.1 – Résultats avec une SOM de 32×32 neurones, avec 50 executions par ligne. La colonne *Algorithme* précise avec quel algorithme et quelle topologie la SOM a été entraînée. MSQE_S est le MSQE calculé après apprentissage avec l'algorithme BMU standard (exhaustif) tandis que MSQE_F utilise la version Fast-BMU. Nous combinons les deux apprentissages avec les deux reconstructions possibles (Standard et Fast). Les différences de MSQE_S entre les différents algorithmes reflètent ainsi la qualité de la phase d'apprentissage. Le *Mismatch* est la proportion de BMU qui sont sélectionnés différemment par les deux algorithmes.

Données	Algorithme	MSDtN	MSQE_S	MSQE_F	Mismatch
Carré (2D)	Grille	1,94e-4	2,22e-4	2,22e-4	0%
	Fast-Grille	1,93e-4	2,23e-4	2,23e-4	0%
	Hexagonal	2,39e-4	2,12e-4	2,12e-4	0%
	Fast-Hexa	2,38e-4	2,15e-4	2,15e-4	0%
Forme (2D)	Grille	1,38e-4	1,40e-4	1,40e-4	\approx 0%
	Fast-Grille	1,38e-4	1,40e-4	1,40e-4	\approx 0%
	Hexagonal	1,65e-4	1,31e-4	1,31e-4	\approx 0%
	Fast-Hex	1,65e-4	1,31e-4	1,31e-4	\approx 0%
Cube (3D)	Grille	4,48e-4	2,21e-3	2,50e-3	4.8%
	Fast-Grille	4,61e-4	2,25e-3	3,21e-3	9.8%
	Hexagonal	5,29e-4	2,09e-3	2,34e-3	3.1%
	Fast-Hex	5,38e-4	2,11e-3	2,79e-3	7.6%
Couleurs (3D)	Grille	1,15e-4	8,64e-5	8,80e-5	4.4%
	Fast-Grille	1,19e-4	8,91e-5	9,08e-5	5.4%
	Hexagonal	1,33e-4	8,29e-5	8,30e-5	0.4%
	Fast-Hex	1,35e-4	8,26e-5	8,29e-5	0.7%
Image (300D)	Grille	1,64e-4	1,80e-3	1,83e-3	4.2%
	Fast-Grille	1,65e-4	1,82e-3	1,85e-3	4.4%
	Hexagonal	1,97e-4	1,75e-3	1,77e-3	1.2%
	Fast-Hex	1,99e-4	1,75e-3	1,76e-3	1.2%
Sons (1000D)	Grille	2,02e-4	1,42e-2	1,49e-2	31.3%
	Fast-Grille	1,93e-4	1,44e-2	1,51e-2	32.2%
	Hexagonal	2,29e-4	1,41e-2	1,45e-2	19.8%
	Fast-Hex	2,25e-4	1,42e-2	1,45e-2	13.3%

We first observe that the distance between neurons after learning (MSDtN) is lower with the grid topology than with the hexagonal one, but this difference could be attributed to the different number of neighbours between the two topologies and therefore should only be used to compare the standard and Fast-BMU algorithms, and not topologies. We also remark that the Fast algorithm does not make any significant mismatch on the Square and Shape datasets, and therefore MSQE_S and MSQE_F have similar recall results on these datasets.

The mismatch percentages vary greatly between the datasets. From 0 to 6% for the Images and Colors Datasets, 3 to 10% for the Cube and 13 to 33% for the Spoken Digits dataset. Such differences could be explained by the distribution of the data in the datasets, as Images and Colors feature data that are closely related together. In pictures for instance, there are usually a few dominant colors with a lot of color gradients that make the continuities in the distribution easier to learn for the SOM, thus improving the performance of our Fast-BMU algorithm. The Spoken Digits dataset on the other hand has high mismatch values, which seems to indicate that a strong continuity in the neurons weights is not present after learning the SOM. The hexagonal topology also performs better with the Fast-BMU algorithm than the grid topology as mismatches are significantly lower with it. Finally the dimensionality of the dataset does not seem to play a key role here, as the Image dataset (300 dimensions) has lower mismatches than the Cube dataset (3 dimensions).

For the vector quantization part, the hexagonal topology leads again to the lowest error values. What is more surprising is that the Fast-BMU version has quantization results that are very similar to the standard version. Even with high mismatches (30% with Digits using a grid-based SOM) the MSQE is only around 5% higher, and even less when only training is compared. The only significantly higher MSQE values with Fast-BMU is with the Cube dataset where the algorithm selects quite bad BMU choices in the recall phase while being able to correctly train the SOM. In most cases, a difference in the topology of the SOM has more impact on the resulting MSQE than the use of the Fast-BMU algorithm.

Gain computationnels

To evaluate the computational gains that are induced by the use of the Fast-BMU algorithm independently from implementation techniques, we compared the percentage of neurons that must be evaluated in order to find the BMU. The results are shown in figure 4.2. The standard SOM is evaluating all neurons by definition, so the percentage is always 100%. The complexity curve for Fast-BMU plots the function defined in section ???. To obtain the Fast-measured curve, we ran tests with $n \times n$ SOM, where n is every even number between 10 and 50 (21 tests in total). Each test featured all datasets and all topologies (so 12 executions per test).

With these results, we can observe significant improvements in the required computational time. Our algorithm is twice as fast with (16×16) SOMs, four times faster with 1000 neurons (32×32) . The (50×50) SOM evaluates approximately 375 neurons per iteration, which is similar to the 400 neurons a standard (20×20) SOM has to evaluate. We can also observe that the complexity curve follows a similar shape to the measured curve, while overestimating the required number of evaluated neurons by approximately 75%.

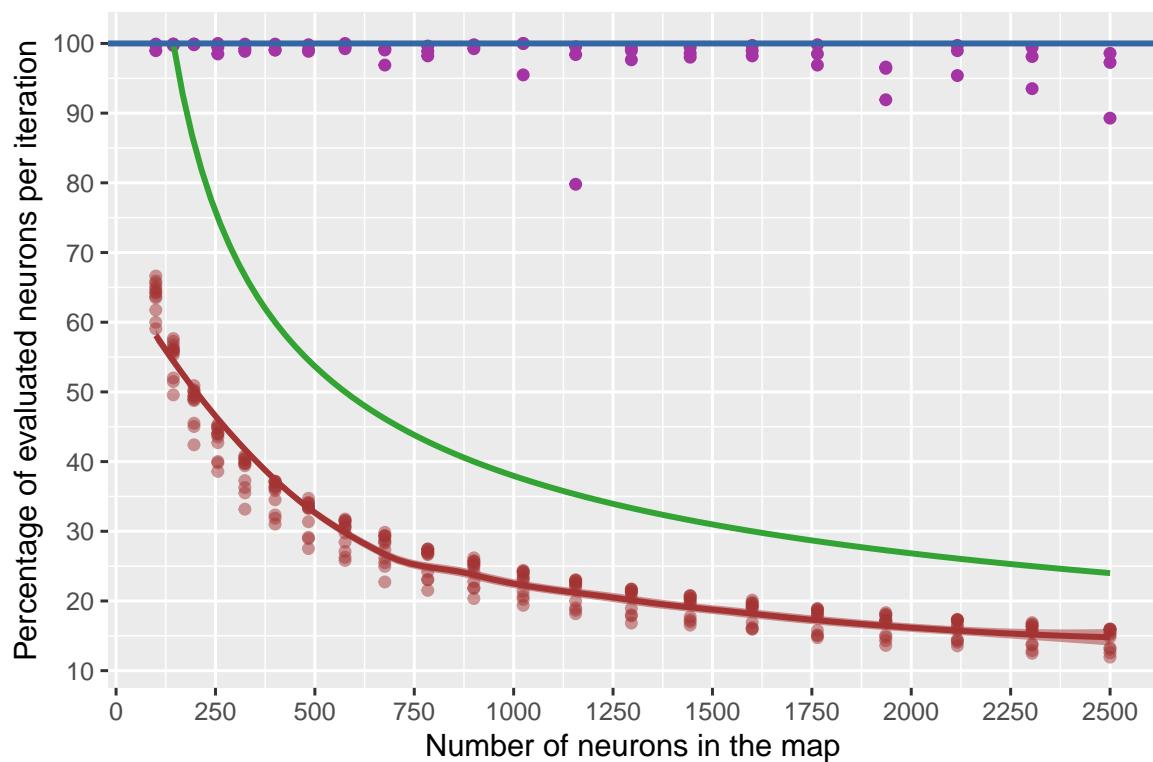


FIGURE 4.2 – Évaluation des gains de performance en fonction du nombre de neurones. Le gain en performance est exprimé en pourcentage de neurones qui ont dû être évalués pendant la recherche de BMU. Tous les résultats ont été calculés sur des cartes carrées. La ligne bleue représente la SOM standard, en vert la valeur analytique calculée et en rouge la valeur mesurée. En outre, les points violetts représentent le pourcentage de BMU correctes dans une exécution sur le jeu de données Image.

4.3 Algorithme parallèle

4.4 Conclusion

We have presented a novel method to find the Best Matching Unit in Self-Organizing Maps by taking advantage of their ability to preserve their underlying topology within the codebook. Our algorithm is significantly faster to compute while performing barely worse than the standard approach in vector quantization.

This result makes SOM with a high number of neurons a more viable solution for many applications. We expect future improvements of this algorithm, or new approaches to further reduce the computational cost of SOM by modifying the BMU searching algorithm. We also study how the Fast-BMU approach can reduce the bandwidth demands in a fully parallel implementation on a manycore substrate, where all neurons are simultaneously evaluated but the BMU selection uses the simple unicast propagation of particles instead of full broadcast-reduce schemes. Finally it must be pointed out that the computational gains offered by our Fast-BMU algorithm specifically rely on preservation of neighbourhood relations when mapping the input space onto the neural map. This property is not present when using more conventional VQ models such as k-means, so that the use of SOM could be extended to more applications where their specific mapping properties would not be useful for the application itself, but would induce potential computational gains out of reach for other models.

4.5 Références

JACKSON, Z., C. SOUZA, J. FLAKS, Y. PAN, H. NICOLAS et A. THITE. 2018, «Jakobovski/free-spoken-digit-dataset : v1.0.8», . [77](#)

