

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

M. Alan Turing,	Professeur	Examinateur
Mme. Margaret Hamilton,	Professeur	Rapporteur
M. Emil L. Post,	Professeur	Examinateur
M. Ken Thompson,	Professeur	Examinateur

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	8
1.3 Références	14
2 Détection de Nouveauté	17
2.1 Introduction	18
2.2 Application aux images	18
2.3 Détection de nouveauté par QVAT : quantification vectorielle et appren-tissage topologique	23
2.4 Protocole expérimental	27
2.5 Résultats expérimentaux	41
2.6 Conclusion	48
2.7 Références	48
3 Vision événementielle et attention	49
3.1 Introduction	50
3.2 Détection de mouvements	50
3.3 Mécanisme attentionnel	53
3.4 Combinaison avec la détection de nouveauté	53
3.5 Conclusion	53
3.6 Références	53
4 Calcul de Best Matching Unit accéléré avec la topologie	55
4.1 Introduction	56
4.2 Observations topologiques	57
4.3 Algorithme séquentiel	58
4.4 Algorithme parallèle	67
4.5 Conclusion	67
4.6 Références	67

TABLE DES MATIÈRES

Liste des figures

1.1	Phonème SOM	7
1.2	Phonème SOM	8
2.1	Lac de Nino	19
2.2	Représentation d'une image	20
2.3	Compression et décompression d'image	21
2.4	Représentation d'une image	22
2.5	Détection de nouveauté par quantification vectorielle	23
2.6	Détection de nouveauté avec topologie	24
2.7	Catégorie Baseline	28
2.8	Catégorie Bad weather	28
2.9	categorie camera jitter	28
2.10	Categorie Dynamic Background	28
2.11	Categorie Shadow	29
2.12	Categorie Night Videos	29
2.13	Categorie thermal	29
2.14	Categorie turbulence	30
2.15	Categorie intermittent object motion - Reduced	30
2.16	Categorie low framerate - Reduced	30
2.17	Différence entre nouveauté and changement	31
2.18	Échantillonnage de l'évaluation	35
2.19	Effet de l'aléatoire sur les métriques	37
2.20	Optimisation de σ	39
2.21	Optimisation d' α	40
2.22	F-mesure en fonction du nombre de neurones et de la taille des images, SOM baseline	41
2.23	F-mesure en fonction du nombre de neurones et de la taille des images, GNG baseline	42
2.24	F-mesure en fonction du nombre de neurones et de la taille des images, global	43
3.1	Détection de mouvements avec DNF sur caméra événementielle	50
3.2	Détection de mouvements multible avec DNF sur caméra événementielle	51

- 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.

4.2 Evaluation of performance gains with the number of neurons. All results were calculated on square maps. The blue line is the standard SOM, in green is the analytical value and in red the measured value. Additionally, the purple dots are the percentage of correct BMU in an execution on the Image dataset.

Liste des tableaux

2.1	Estimations statistiques du nombre de graines requises	37
2.2	Récapitulatif des paramètres SOM	38
2.3	Résultats complets sur CDNET de notre détection de nouveauté	44
2.4	Résultats complets sur CDNET de notre détection de nouveauté - Suite	45
2.5	Comparatif avec d'autres modèles de détection de changement sur CD-NET	45
3.1	Paramètres DNF 1D	52
4.1	Résultats de FastBMU sur différentes données	64

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	7
1.1.5 Méthodes classiques informatiques	7
1.2 Réseaux neuronaux	8
1.2.1 Cartes auto organisatrices	8
1.2.2 Principes de fonctionnement	9
1.2.3 Gaz Neuronaux en Expansion	11
1.2.4 DNF	13
1.3 Références	14

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. Cet question a été le fondement d'un nouveau domaine scientifique qui a été nommé quelques années plus tard, lors de la conférence de Dartmouth **MCCARTHY et collab. [1955]** : l'intelligence artificielle.

Les années qui suivirent n'amènerent pas de résultats au niveau des espérances des chercheurs. Le domaine était encore trop limité par la puissance limitée des ordinateurs 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 transistor par puce et lorsque les méthodes algorithmiques étaient plus avancées, avec par exemple la recherche rapide dans une bases de données. Ces changements ont permis aux modèles d'IA d'exploiter plus de données, plus rapidement et avec 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 développé tout au long de son histoire et qui est passé sur le devant de la scène depuis les années 2010. C'est une combinaison de plusieurs facteurs qui l'on 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 on vu l'explosion de ce type de réseaux, qui grâce à leur généralité, 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 **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 inclut dedans. Les performances étaient quand à elles toujours inférieures à un humain, qui lui n'a accès qu'à un cerveau de 12 Watts et un corpus d'apprentissage ridiculement 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 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 quel 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 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.

1.1.1 L'inspiration biologique

Depuis l'avènement des semi-conducteurs, la puissance computationnelle disponible n'a cessé de s'accroître exponentiellement. La célèbre prophétie auto-réalisatrice de Gordon Moore, que le nombre de transistor par processeur double tous les deux ans, a fait progresser l'industrie pour lui faire atteindre le million de transistor 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 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 séquentielle et les algorithmes

-
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))

et processeurs se sont développés 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 combiné à la croissance exponentielle du nombre de transistor ont naturellement amené vers le développement d'architectures multi-coeurs, 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. 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 à ses problèmes. La piste que nous avons choisi ici est de s'inspirer de la machine la plus efficace pour traiter de l'information que nous connaissons : le cerveau.

Un cerveau est une machinerie complexe, composé d'environ de 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 de raisonnement humaines.

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 synchronisé sur une horloge globale, propre à chaque processeur. Les informations dans le cerveau circulent de façon locales entre neurones voisins et avec très peu de connexions longues 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 de compiler le 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 évoqués, 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 elle comprend 131 072 neurones, et 10 fois plus de synapses. Une fois combinées par 32 sur une carte *Nahuku* on obtient un système neuromorphique 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

Le premier réflexe lorsque l'on découvre un nouvel outil, est de voir comment il se combine avec les outils que nous avons déjà. Ainsi, la première idée pour exploiter efficacement des substrats de calculs neuromorphique serait de faire comme on a toujours fait. C'est à dire, avec des algorithmes et des programmeurs experts capables de produire un ensemble d'instructions effectuant 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 à faire 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 blocage, d'incohérences et augmente significativement le nombre de bugs possibles. Le matériel neuromorphique lui est en comparaison improgrammable. 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 qu'il est impensable de voir un jour des programmeurs produire un pilote automatique avec un système neuromorphique à 100 millions de neurones. Ni même résoudre un quelconque problème qui ne soit pas trivial.

On pourrait y objecter que les ordinateurs actuels ont un niveau de complexité similaire car même les programmeurs les plus téméraires ne se risqueraient pas à programmer en code machine nos processeurs à 50 milliards de transistors, et que malgré tout, il est possible de faire des pilotes automatiques. C'est une remarque intéressante qui montre le besoin crucial de langages plus abstraits pour effectuer des tâches complexes. Un tel language n'existe pas encore pour les puces neuromorphiques. Copier les langages existant qui fonctionnent et les appliquer au neuromorphique copierait également leurs limites, et sera une utilisation inefficace de la puissance de calcul neuronal. Ainsi il semble nécessaire de trouver des principes fondamentaux pour organiser efficacement le calcul neuromorphique.

Nous pouvons de nouveau ici nous inspirer de ce que fait la biologie avec le principe d'émergence. L'émergence est un phénomène qui apparaît lorsqu'un ensemble d'éléments à plus de propriétés 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

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

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

proche des autres individus, et qui amène à une structure plus avancée au niveau du groupe : la nuée. Un phénomène similaire est à l'oeuvre 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 ainsi sera la 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 dans une baignoire pour qu'un humain en sorte. Même si l'on avait les bonnes molécules cela ne fonctionnerait pas. Pour que cela marche, il faut que les molécules soient placées au bon endroit 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 biologiques comme la physique, l'économie et l'informatique avec 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. C'est cette problématique que nous explorerons dans cette thèse.

Il existe des modèles en informatique qui ont pour ambition de faire le lien entre le niveau neuronal et des propriétés 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, tel que « *Neurons that fire together, wire together* ». On peut également remarquer l'article de Dijkstra sur l'auto-stabilisation DIJKSTRA [1974]. Le sujet de cet article 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. Ce principe d'auto-stabilisation est très présent dans la théorie des champs neuronaux introduite par AMARI [1977]. Les champs neuronaux utilisaient également le concept de neurones inhibiteurs et la temporalité des stimulus commençait à être pris en compte. Enfin, nous avons les cartes auto-organisatrices KOHONEN [1982], qui bien qu'utilisant le clustering principalement, est un modèle qui auto-organise sa topologie.

Nous avons dans cette thèse utilisé principalement les cartes auto organisatrices et les champs neuronaux dynamiques, qui sont une implémentation de la théorie des champs neuronaux. 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'oeil 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'oeil ou le capteur. Ce sont ces réflexions, généralement la première, qui donnent le plus d'informations sur l'environnement. En effet, lorsque la lumière reflète sur un objet, il y a une interaction entre les deux. L'objet吸erce 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, nous n'avons accès qu'à des informations partielles. L'objectif de la vision est d'utiliser ces informations partielles pour créer une interprétation de l'environnement cohérente.

Construire une représentation cohérente n'est cependant pas chose aisée. Le premier problème vient de la quantité de données à traiter.

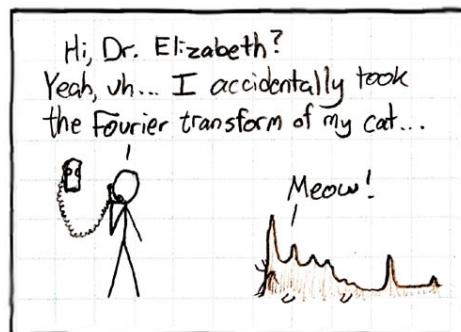


FIGURE 1.1 – [7](#)

1.1.4 Vision humaine

1.1.5 Méthodes classiques informatiques

1.2 Réseaux neuronaux

1.2.1 Cartes auto organisatrices

Les cartes auto-organisatrices (aussi appelées réseaux de Kohonen) regroupent un ensemble de modèles qui a commencé par une publication de 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 fourni, car les propriétés de voisinages sont conservées. Une des premières utilisation de ces cartes fut la représentation des phonèmes du finnois comme présenté dans la figure 1.2.

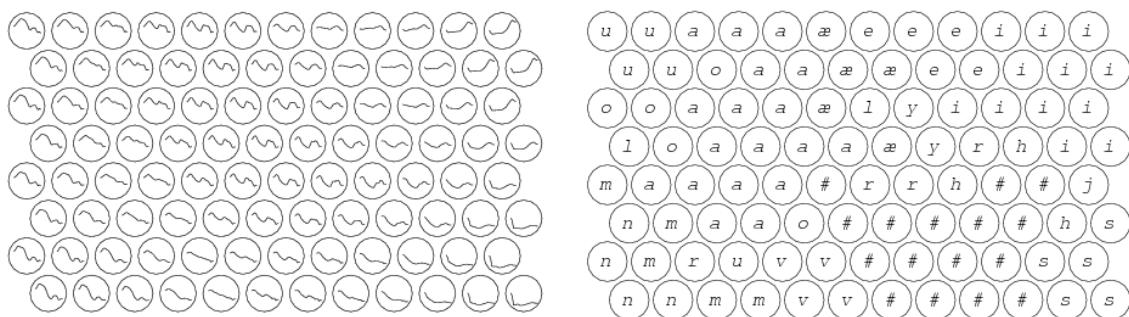


FIGURE 1.2 – Représentation des phonèmes du finnois par la première SOM. A gauche sont représentés les signaux sonores en haute dimension, et à droite leurs phonèmes correspondants. La réduction dimensionnelle provient de l'agencement de ces phonèmes sur la carte. Si ils sont proches entre eux dans leur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

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 du concept neuroscientifique de colonnes corticales, qui sont un groupe de neurones arrangés verticalement et qui réagissent tous au même stimulus.

Evolutions et utilisation contemporaine

Il y a eu de nombreuses évolutions pendant les presque 40 années 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 de l'image et la vidéo, par la parole et le traitement du signal, la médecine et la biologie, l'économie et la finance, de l'urbanisme et d'autres encore. Pour chacun de ces domaines il y a plusieurs types d'utilisations différentes 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 et en les projetant sur des dimensions plus petites. Mais aussi pour faire des traitements sur des 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. \[2018\]](#) est une revue récente évoquant les aspects les plus importants des SOM et présentant quelques applications typiques.

[Évolutions et dérivés.]

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 bases 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 vecteur 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 la visualisation est plus aisée en deux dimensions pour toutes les applications où cela en est le but premier. C'est aussi la taille idéale 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. 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 les toutes les dimensions supérieures à deux difficiles à implémenter efficacement dû à des coûts en communication accrus, car les circuits imprimés sont naturellement en deux dimensions.

Un second choix important est ce que nous appelons la topologie de la SOM. Par topologie, nous entendons la forme des connections 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. Ces topologies sont deux 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 difficile pour une sphère de bien couvrir les données 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.

Le dernier type de paramètre pour la SOM sont les paramètres numériques. Il y a parmi ceux-ci : La taille de la SOM, 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. Il y a aussi le coefficient d'apprentissage ϵ (epsilon), 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'étape courante de l'apprentissage. Le dernier paramètre est 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. Plus il est élevé, plus les voisins ont un impact et plus la contrainte topologique sera forte. Inversement, une valeur de 0 pour ce paramètre enlève toute contrainte topologique et fait que la SOM se comportera comme un k-means. 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 exactement le nombre d'itérations requises pour que chaque élément de la base d'apprentissage soit utilisé exactement une seule fois par époque. 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 :

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

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. Θ est une fonction gaussienne centrée normalisée d'écart type σ . v est le vecteur d'apprentissage.

On répète ces deux étapes jusqu'à ce que l'on finisse la dernière époque, est l'apprentissage sera terminé.

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 poids des neurones

les plus proches.

Cette reconstruction est par exemple utilisée pour de la compression de données, car à la place de mémoriser un ensemble de vecteurs, il n'y a besoin que de mémoriser les poids de la SOM et l'indice de chaque neurone le plus proche de chaque vecteur de l'ensemble. Cette compression est avec perte, du fait que les poids des neurones 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 vectorielle avec topologie. Nous avons souhaité valider expérimentalement cette transmission 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 non pas fixe.

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 à 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 les avantages qu'apportent les variants ne sont pas nécessaires dans notre cas.

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.

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 topologiques 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 topologiques directs s_n . Les formules sont :

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

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

6. Si s_1 et s_2 sont voisins topologiques 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 .
9. Réduire toutes les variables d'erreur en les multipliant par une constante d .

$$w_r(t+1) = \frac{w_q(t) + w_f(t)}{2} \quad (1.4)$$

- 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 .

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 de 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 sera précisée dans la section expérimentale correspondante.

1.2.4 DNF

Following the seminal work of ?, we choose to couple our autonomous novelty detection tool to a robust bio-inspired tracking technique based on Dynamic Neural Fields (DNF). DNF are populations of partial differential equations first mathematically analyzed by ? in a continuous framework. We use a discrete DNF built from populations of excitatory and inhibitory neurons that interact continuously, with a on-center off-surround approach modeled as a synaptic kernel computed as a difference of gaussians applied to the distance between neurons in the neural map. These DNF have been successfully applied to sequential visual exploration of an environment ? or in ?, with great robustness properties that can even improve with some adaptation like the use of simple spiking neurons ?.

Fonctionnement

Continuous Neural Fields Theory (CNFT) has lead to the development of two dimensional Dynamic Neural Fields (DNF) ?. Neural fields are models that represent the evolution of a population of neurons. In our case, we use a two dimensional DNF. The number of neurons is dependent and equal to the size of the input map, because neurons are connected in a retinotopic way to afferent inputs, and are connected in an all-to-all connection scheme between them. All neurons also have a real value attached to them that we call potential. This potential $u(x, t)$, with x being the neuron position in the field and t the time of the simulation, is ruled by the following differential equation :

$$\tau \frac{\partial u(x, t)}{\partial t} = -u(x, t) + \int u(x', t) \omega(||x - x'||) \delta y + \text{Input}(x, t)$$

With :

- τ is the time constant.
- $-u(x, t)$ is the decay term. It is meant to suppress already activated neurons when there is no input or lateral excitation.
- $\omega(||x - x'||)$ is the lateral interaction. It represents the effect of the other neurons onto this neuron's potential. We are using a difference of gaussian with the excitatory gaussian part being narrow with high intensity and the inhibitory one being wide with low intensity. This leads to close neurons having an excitatory effect onto each other and far away neurons inhibiting themselves.
- $\text{Input}(x, t)$ is the current value of the afferent input extracted from the input map for this neuron.

For the sake of simplicity and computability, we implement a spatially and temporally discretized version of the previous formula. It is obtained by handling potentials of a discrete set of neurons (neural map instead of neural manifold) and by using a simple Euler method to estimate the state of $u(x, t + \Delta t)$ knowing $u(x, t)$:

$$u(x, t + \Delta t) = u(x, t) + \frac{\Delta t (-u(x, t) + \sum u(x', t) \omega(||x - x'||) + \text{Input}(x, t + \Delta t))}{\tau}$$

Δt is the time step between two estimations, it can be the same for all neurons (synchronous) or different each time (asynchronous). It should be noted that in the original DNF formula, there are more parameters such as resting potential but since we do not use them here, we did not mention them.

It is often difficult to understand how a DNF will behave just from the formula. We have set it up with optimized parameters in order to have a winner-takes-all behaviour where the most prominent and spatially coherent features in the input map create a local bubble of activation in the neural map and suppress the ability of other such bubbles to appear elsewhere in the map.

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](#)
- BERNARD, Y., E. BROY, 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. [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. [9](#)
- DIJKSTRA, E. W. 1974, «Self-stabilization in spite of distributed control», dans *Selected writings on computing : a personal perspective*, Springer, p. 41–46. [6](#)
- FERRUCCI, D. A. 2012, «Introduction to “this is watson”», *IBM Journal of Research and Development*, vol. 56, n° 3-4, p. 1–1. [2](#)
- FRITZKE, B. 1995, «A growing neural gas network learns topologies», *Advances in neural information processing systems*, vol. 7, p. 625–632. [11](#)
- 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](#)
- KOHONEN, T. 1982, «Self-organized formation of topologically correct feature maps», *Biological cybernetics*, vol. 43, n° 1, p. 59–69. [6, 8](#)
- KOHONEN, T. 2012, *Self-organization and associative memory*, vol. 8, Springer Science & Business Media. [8](#)

- 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](#)
- 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. [11](#)
- MARTINETZ, T., K. SCHULTEN et collab.. 1991, «A "neural-gas" network learns topologies», . [11](#)
- 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. [8](#)
- 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. [11](#)
- PÖLLÄ, M., T. HONKELA et T. KOHONEN. 2011, «Bibliography of self-organizing maps», URL <http://www.cis.hut.fi/research/refs/>. [8](#)
- 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](#)
- 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](#)

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	18
2.2	Application aux images	18
2.2.1	Apprentissage et reconstruction	19
2.2.2	La gestion des canaux (couleurs)	21
2.3	Détection de nouveauté par QVAT : quantification vectorielle et apprentissage topologique	23
2.3.1	Détection avec quantification vectorielle	23
2.3.2	Détection avec distance neurale	24
2.3.3	Considérations pour la combinaison	25
2.4	Protocole expérimental	27
2.4.1	Présentation de la base de données	27
2.4.2	Métriques utilisées	31
2.4.3	Préparation des données	34
2.4.4	Paramétrages des modèles	35
2.5	Résultats expérimentaux	41
2.5.1	Nombre de neurones et taille des imagettes	41
2.5.2	Résultats complets	43
2.5.3	Visualisation des résultats	46
2.5.4	Interprétations	47
2.6	Conclusion	48
2.7	Références	48

2.1 Introduction

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 la détection de changements dans la littérature [réf]. 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 images 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 étouffée 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.[Modifier la figure]

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 de distance avec les neurones fonctionne, 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 longeur, 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 [possible de centrer et de rogner tous les côtés en même temps], car en général les bords ne contiennent pas beaucoup d'informations.

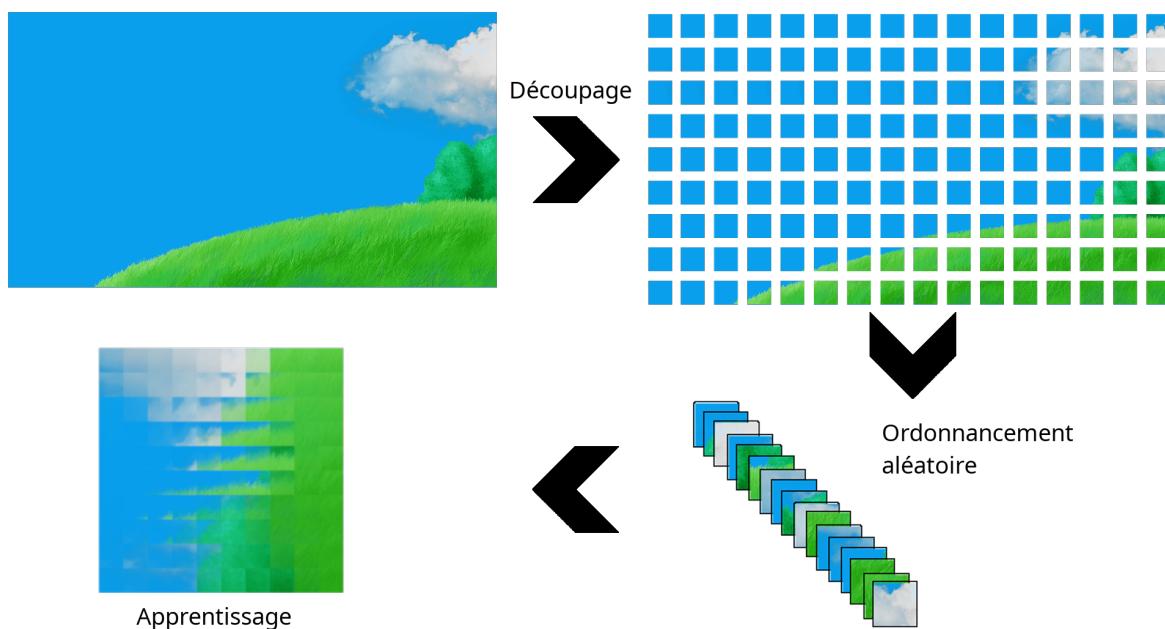


FIGURE 2.2 – Illustration du processus de représentation et d'apprentissage d'une image par une SOM.

Reconstruction

Une fois l'apprentissage terminé, il y a deux résultats. Le premier est le modèle entraîné avec les différents poids des neurones codant une imagette représentante du cluster d'imagettes associé à ce neurone. Le second 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 rassemblés 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.

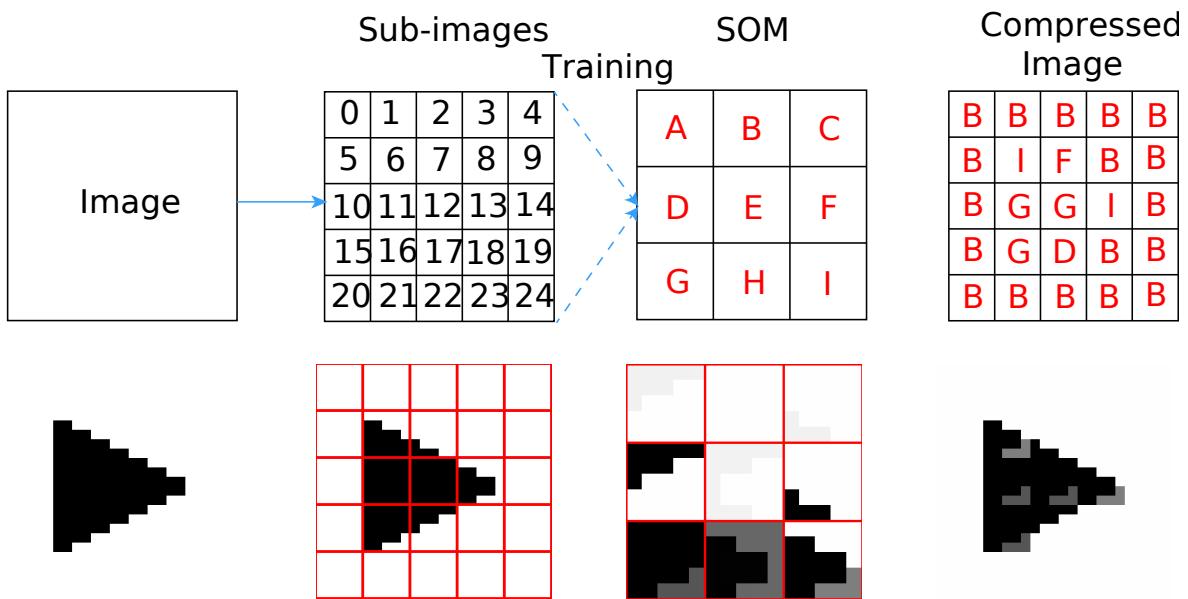


FIGURE 2.3 – Schéma simplifié du processus de compression et de reconstruction d'une image, avec ici seulement 9 neurones et 25 imagettes.

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 couleur, c'est à dire 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 par conséquent), 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, et 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 apparents 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.



(a) Couleurs fusionnées

(b) 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.[Faire un exemple plus visuel]

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

Nous avons à partir des différentes propriétés de nos modèles neuronaux 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

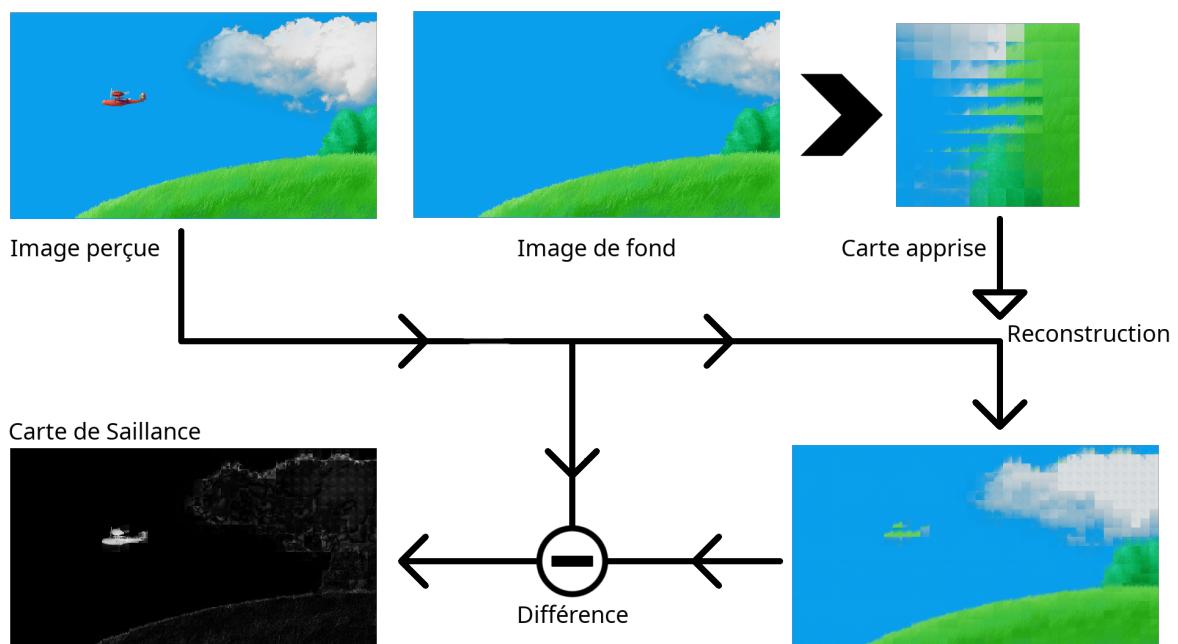


FIGURE 2.5 – On peut observer qu'il y a eu deux changements entre le fond et l'image perçue : un avion est apparu et les nuages ont bougé. Les nuages, déjà présents dans le fond sont bien reconstruits. L'avion cependant est nouveau, et n'est pas bien reconstruit. Ainsi la différence entre l'image perçue 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 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.

Ce processus a l'avantage d'être précis au niveau du pixel pour la mise en évidence des changements [Préciser que la taille des imagettes limites quand même la précision]. 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.

2.3.2 Détection avec distance neurale

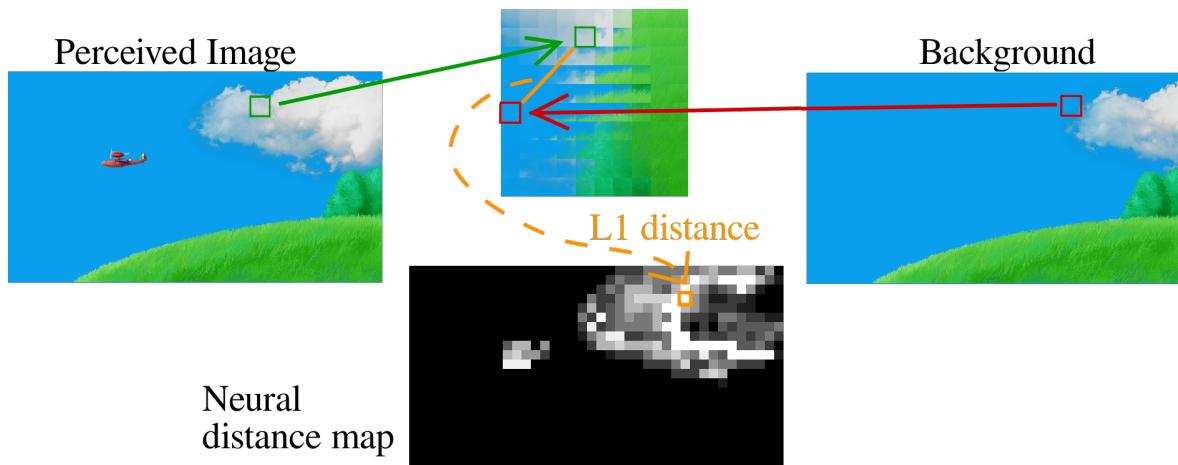


FIGURE 2.6 – Le processus présenté ici concerne une position dans l'image, et il est répété sur toute l'image pour obtenir la carte de distances neurales en bas. Nous avons représenté le modèle appris comme étant une SOM sur cette figure, cependant il peut s'agir de n'importe quel modèle avec une topologie regroupant les éléments proches.[Traduire la figure en français]

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.

Après avoir effectué l'apprentissage du modèle, nous mémorisons la liste des positions dans la carte des neurones représentant 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. Nous nous intéressons uniquement à la liste des positions dans la carte des neurones représentant

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 la fond. Le processus est illustré dans la figure 2.6.

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 [citation]. Dans notre cas, nous avons préféré utiliser une combinaison simple de nos deux cartes de saillance. 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 résultante 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 découlée 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

Il n'existe pas à notre connaissance de base de données de détection de nouveauté respectant nos hypothèses de caméra statique, de [...]. Une alternative se trouve dans la base de donnée CDnet [WANG et collab. \[2014\]](#). Elle a pour objectif d'uniformiser les résultats dans un domaine proche de la détection de nouveauté; la détection de changement. Les deux domaines peuvent sembler similaires au premier abord car les deux approches visent la même application réelle. Cependant cela cache une différence conceptuelle. 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 (discuté plus en détail dans la section [...]). 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 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 originaire 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 de 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 tâche de ces catégories 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 2.17. 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 2.7 – *Baseline* : La catégorie de base qui comprend des scénarios typiques de détection de changement (traffic, piétons) sans difficultés particulières.



FIGURE 2.8 – *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.

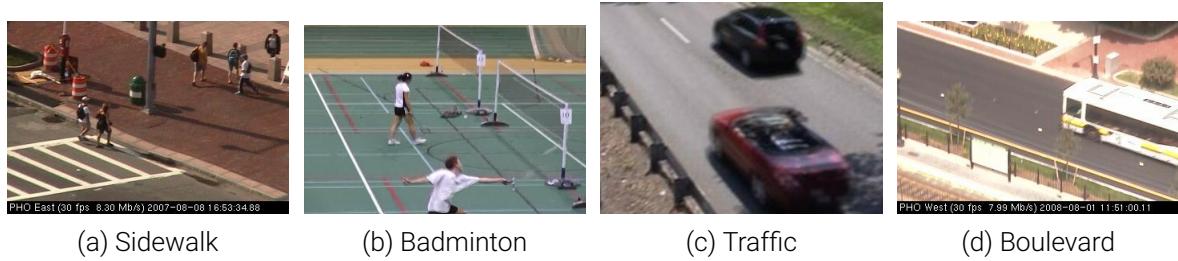


FIGURE 2.9 – *Camera Jitter* : Ces vidéos proviennent de caméras instables à cause de vent fort ou d'autres raisons. Elles ont de façon irrégulière des translations verticales et horizontales rapides et de petite amplitude.



FIGURE 2.10 – *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 2.11 – *Shadow* : Catégorie de vidéos qui présente plus d'ombres que la moyenne.

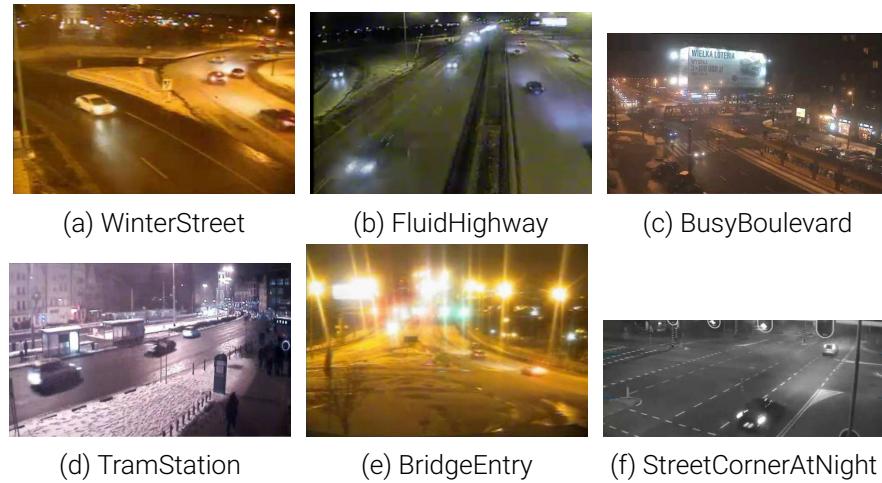


FIGURE 2.12 – *Night Videos* : Vidéos de nuit avec un contraste fort entre l'obscurité ambiante et les lumières artificielles de l'éclairage public et des phares de voitures.

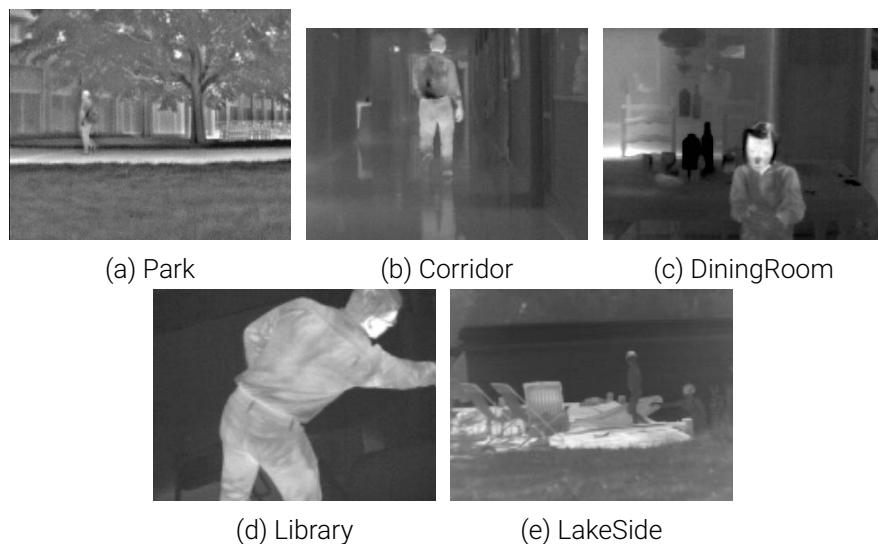
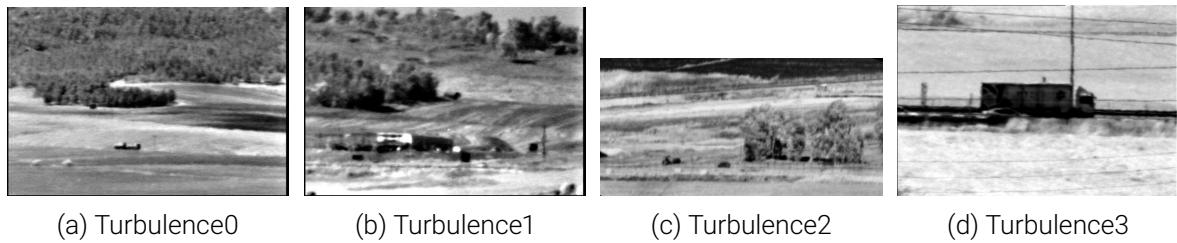


FIGURE 2.13 – *Thermal* : Ces vidéos ont été prises par une caméra infrarouge et sont en niveau de gris.



(a) Turbulence0 (b) Turbulence1 (c) Turbulence2 (d) Turbulence3

FIGURE 2.14 – *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 atmosphérique dûes à la chaleur et à la distance.



(a) StreetLight (b) Sofa (c) Tramstop

FIGURE 2.15 – *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 on été conservées.



(a) Turnpike (b) TramCrossroad (c) Tunnel Exit

FIGURE 2.16 – *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 flow 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 amarrés.

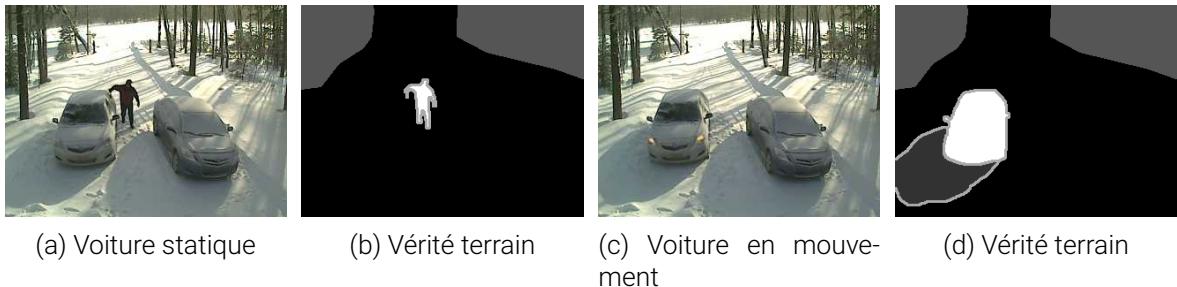


FIGURE 2.17 – Ces images sont extraites d'une vidéo de la catégorie *Intermittent Object Motion* et illustrent la différence entre détection de changement et détection de nouveauté. Pour le changement la voiture fait partie du fond pendant une partie de la vidéo car elle est statique. Elle devient objet 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.

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

[glossaire pour la terminologie mathématique employée]

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 quand 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.

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

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

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 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 $y_2 - x_2 < z_2 - y_2$, car le logarithme met plus d'espace deux hautes valeurs de MSQE qu'entre deux petites.

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 quand même 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 notre perception humaine [réf biblio]. 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 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-mesure pour chaque image séparément pour ensuite en faire la moyenne. On peut également observer cette propriété dans les résultats présentés dans la section [réf section résultat], où la valeur de la F-mesure ne suit pas la formule lorsque l'on applique aux moyennes des précision et rappel.

L'oeil 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 quantifiées, 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, que nous avons déjà évoqué dans les sections [ref]. 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 attendra une version positive pour tous les pixels de l'objet 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 se baseront 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. Une idée simple serait de sélectionner une image sans cible dans la séquence et de l'utiliser comme fond. Cependant il arrive 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, une technique fréquemment utilisée [réfs] pour enlever des objets d'une image est de générer une image médiane à partir d'une séquence. [détails de l'implémentation à voir plus tard].

Un problème qui peut survenir avec la médiane, est l'adoucissement des images enlevant les valeurs extrêmes qui peuvent apparaître dans certaines images. Par exemple pour les images de la catégorie *Bad Weather*, les flocons de neiges qui sont constamment devant la caméra disparaissent dans l'image médiane. Mais cela ne pose pas de problème particulier en pratique [réf tableau].

[tableau comparatif]

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 2.18. 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.

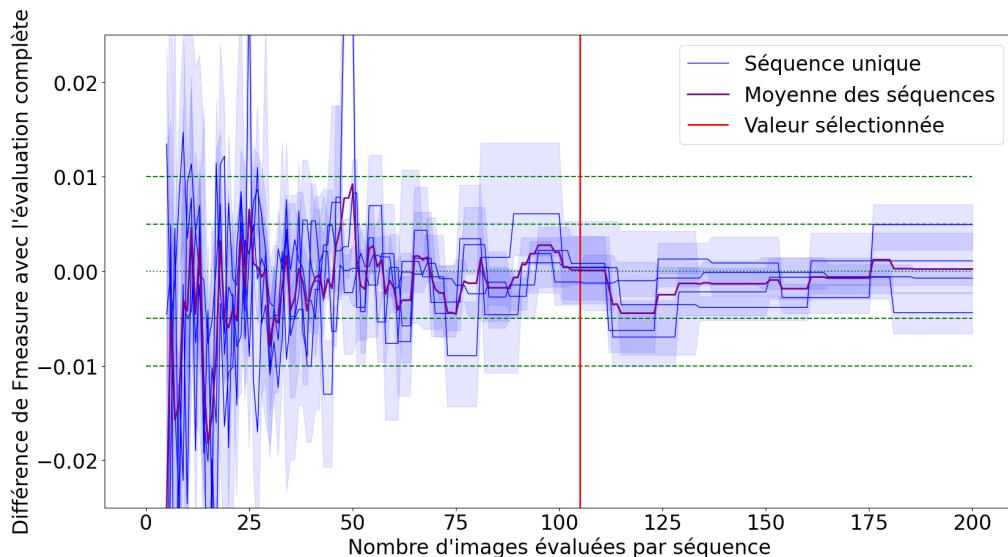


FIGURE 2.18 – Précision de l'approximation de la F-mesure en fonction de la taille de l'échantillon. 105 a été choisi car c'est le plus petit échantillon proche de la moyenne et ne dépassant quasiment pas les 0.5% de différence. Les sections droites des lignes, surtout vers des grands échantillons, sont 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 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 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 y a 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 durant en moyenne quelques minutes, il n'est pas souhaitable d'en effectuer un trop grand nombre.

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 2.19, 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 celles de la *baseline*.

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

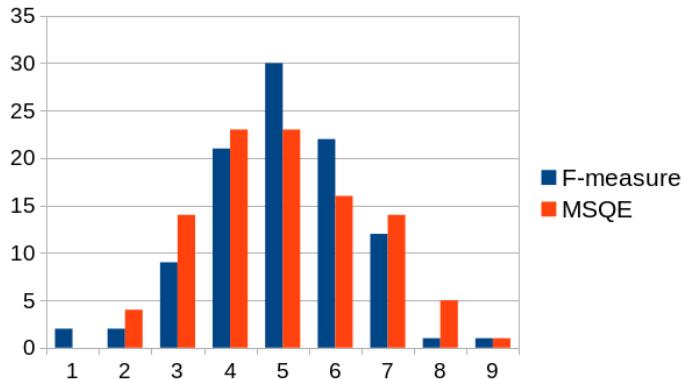


FIGURE 2.19 – Distribution des métriques pour un ensemble de paramètres donnés pour une vidéo. On a découpé l'intervalle de résultats en 9 sections égales. La section numéro 5 a la moyenne en son centre. L'épaisseur de chaque région a été ajustée pour que le maximum soit à la limite haute de la section 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.

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.

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 expérimentons 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 2.20 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' α uniquement pour mieux observer les tendances avec les valeurs fixées de σ . La figure 2.21 présente ces résultats. Nous avons choisi pour α un départ à 0.5 et de finir à 0.06.

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

Graph incoming!

Seuil de décision

Seuil de décision (transformation de carte de saillance 0-255 en binaire)

Paramètres des GNG

[epsilon, maximum age, error decrease, neurons nbr, epochs nbr]

Récapitulatif

TABLEAU 2.2 – Récapitulatif des paramètres SOM

α -start	α -end	σ -start	σ -end	Époques	Images par séquence	Seuil
0.2	0.05	0.7	0.015	100?	105	10?

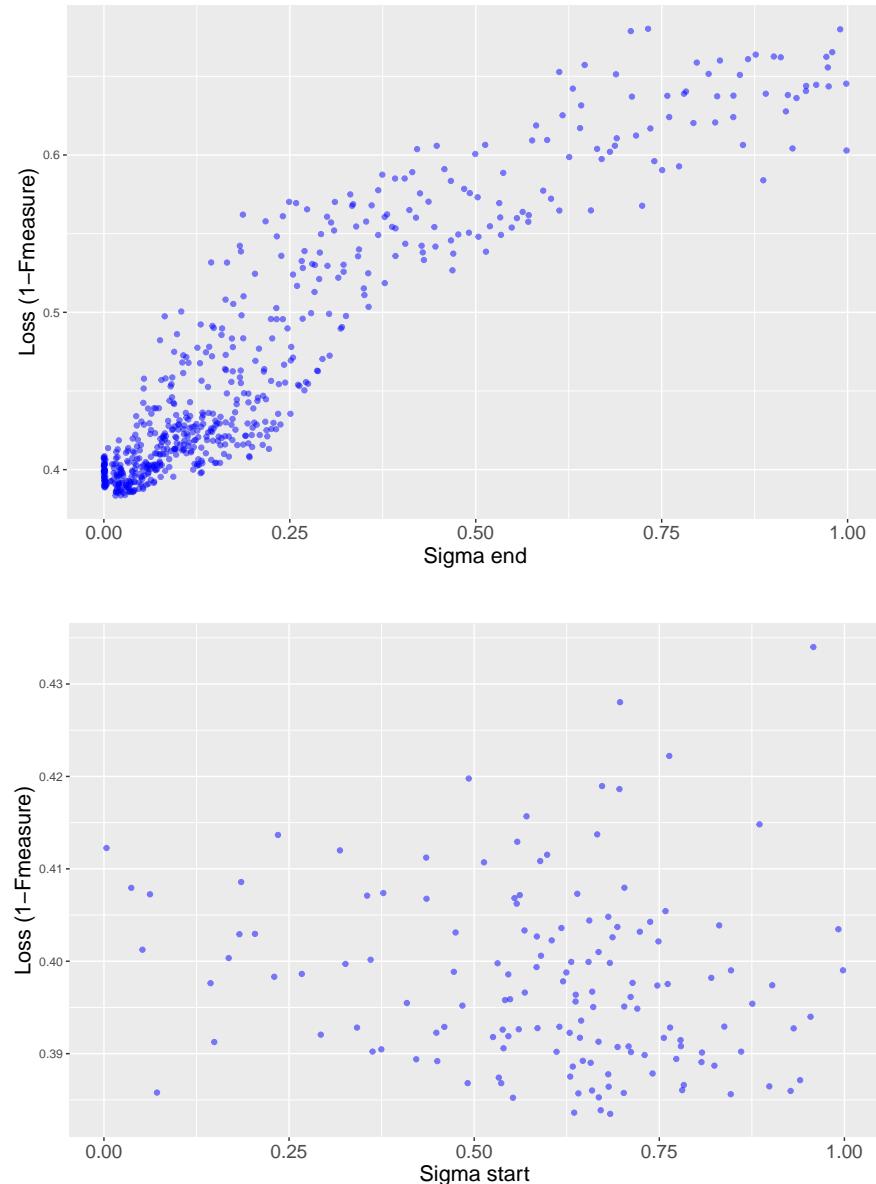


FIGURE 2.20 – Le premier paramètre qui a une valeur optimale évidente est $Sigma end$, où une petite valeur proche de zéro semble idéale. Cela s'explique facilement du fait qu'une valeur faible de $Sigma end$ induit que lors des dernières époques la SOM va se focaliser sur l'optimisation de la quantification vectorielle au dépend de la topologie, pour obtenir des neurones les plus proches possibles des imagettes qu'ils représentent.

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.

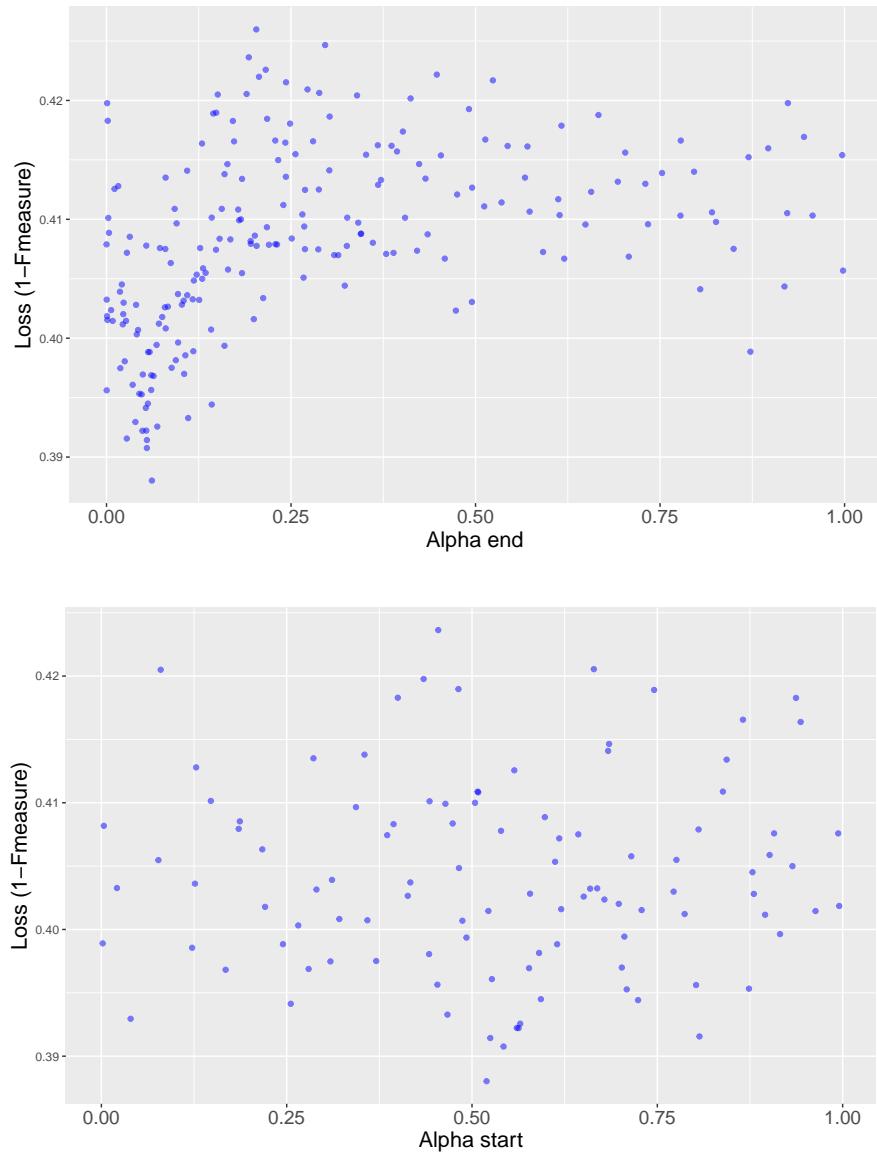


FIGURE 2.21 – Les différences sont moins prononcées pour *Alpha end* que pour *Sigma end*, mais une préférence est tout de même visible. Une faible valeur d'*Alpha end* non nulle semble optimale. En pratique cela se traduit par un ajustement fin des poids des neurones à la fin de l'apprentissage.

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 une valeur intermédiaire de 0.5, qui semble marcher le mieux.

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 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 CDNET 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.

2.5.1 Nombre de neurones et taille des imagettes

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.

Baseline

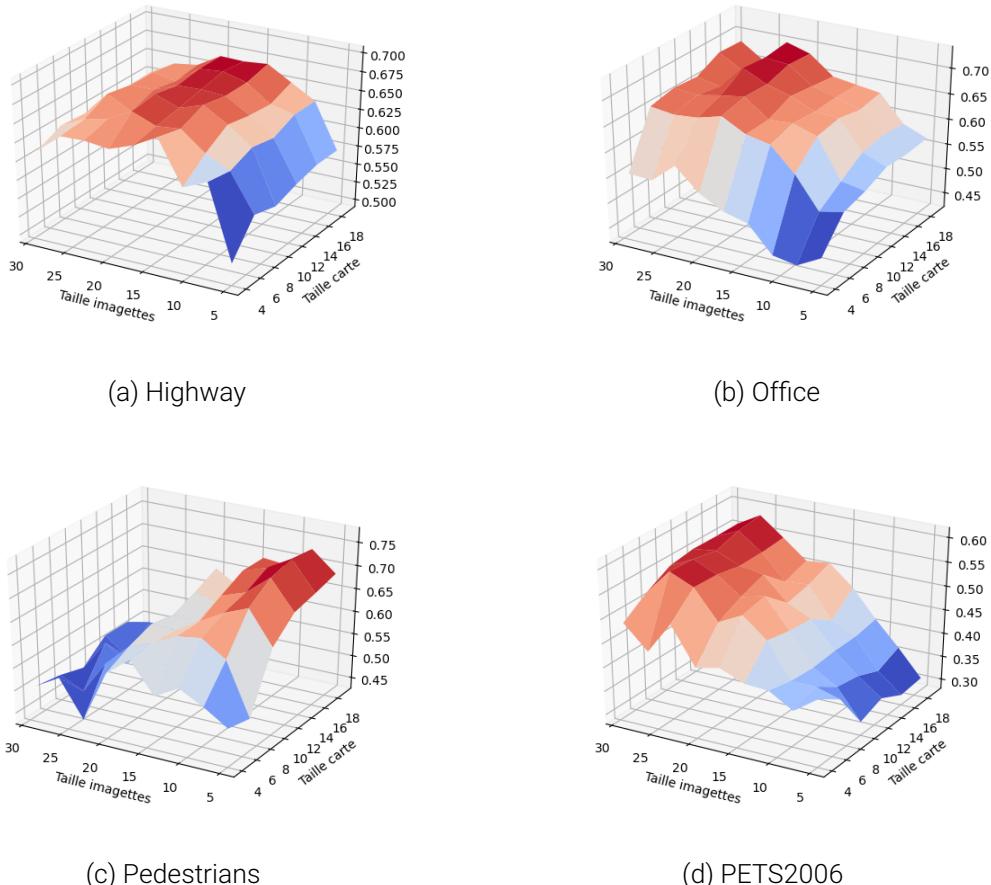


FIGURE 2.22 – F-mesure en fonction du nombre de neurones et de la taille des imagettes pour les séquences de la *baseline* avec une SOM.

La figure 2.22 montre une différence significative de comportement entre les séquences vidéo. La taille des imagettes semble être le facteur le plus important. Pour *Pedestrians* par exemple, une taille d'imagette petite de 8×8 donnera les meilleures performances, alors que pour *PETS2006*, ce sont les grosses imagettes de 26×26 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. 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.

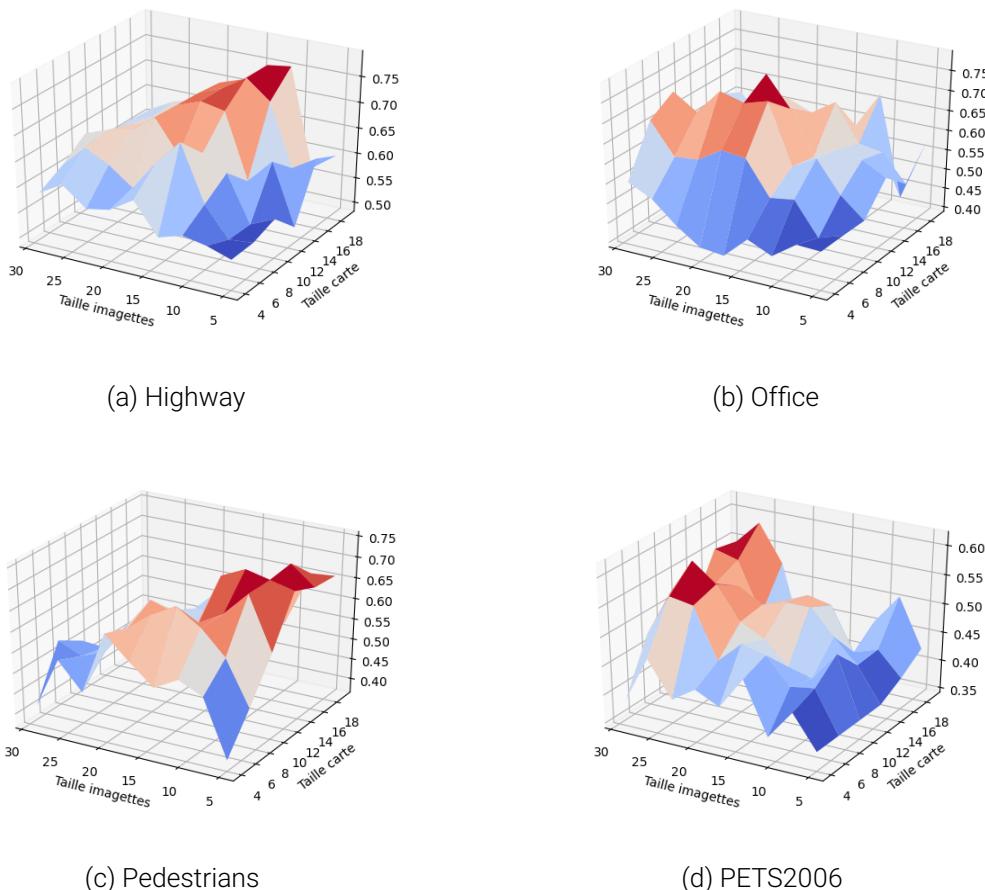


FIGURE 2.23 – F-mesure en fonction du nombre de neurones et de la taille des imagettes pour les séquences de la *baseline* avec un GNG. Les GNG n'ayant pas une topologie carrée comme les SOM, il suffit de mettre au carré la taille de la carte pour obtenir le nombre de neurones utilisés par le GNG.

Les résultats pour les GNG de la figure 2.23 montrent un comportement beaucoup plus chaotique. Les variations sont particulièrement grandes entre plusieurs exé-

cutions, 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 et/ou de la nouveauté présente dans la vidéo. La valeur des F-mesure maximales est similaire entre les GNG et les SOM.

[figures PSNR]

Global

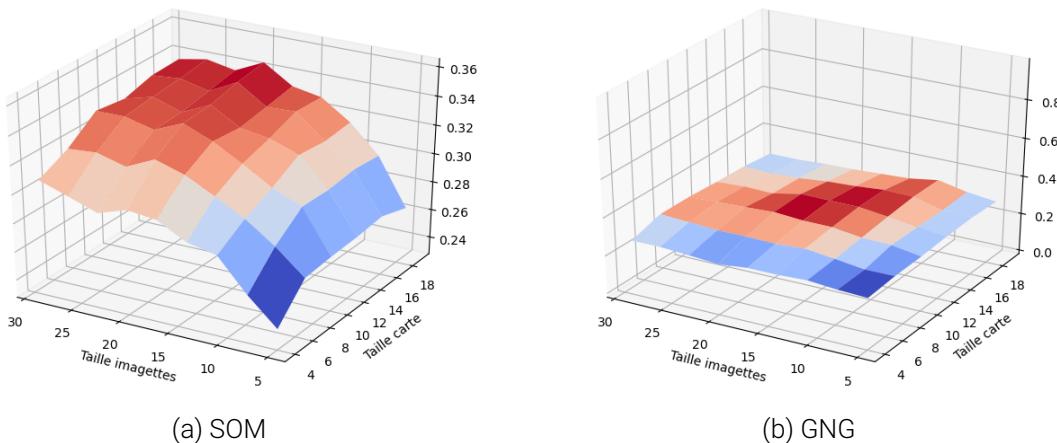


FIGURE 2.24 – Moyenne des F-mesure en fonction du nombre de neurones et de la taille des imagettes pour l'ensemble des vidéos de notre jeu de données. Le graphe des GNG a été aplati pour que l'on puisse voir l'intégralité de celui-ci.

Lorsque sont agrégés toutes les séquences vidéos, les graphes de la figure 2.24 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 au delà, les performances stagnent, voire décroissent. 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.

2.5.2 Résultats complets

Les résultats complets sont présentés dans les tableaux 2.3 et 2.4. Les versions *Global* indiquent que l'on a choisi les valeurs de taille de carte et d'imagette qui étaient optimale sur l'ensemble de CDNET, c'est à dire 20×20 pour les tailles de carte et d'imagette pour la SOM et ?×? 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 toutes les catégories, le rappel est toujours meilleur que la précision. Cependant pour 12 vidéos sur 39 (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.

	SOM Global			SOM Max	GNG Global	GNG Max
Categories/vidéos	Précision	Rappel	F-mesure	F-mesure	F-mesure	F-mesure
Baseline	62.1%	63.8%	61.5%			
Highway	63.5%	66.4%	64.9%			
Office	74.1%	61.9%	67.5%			
Pedestrians	51.0%	83.9%	63.5%			
PETS2006	59.8%	43.1%	50.1%			
Bad weather	37.1%	47.1%	40.0%			
Blizzard	33.6%	40.6%	36.7%			
Skating	71.6%	60.8%	65.7%			
Snowfall	21.9%	45.9%	29.6%			
Wet snow	21.4%	41.1%	28.0%			
Camera jitters	42.4%	55.5%	47.1%			
Badminton	49.9%	72.3%	59.0%			
Boulevard	52.7%	41.8%	46.6%			
Sidewalk	22.9%	33.1%	27.1%			
Traffic	44.2%	74.7%	55.5%			
Dynamic bkg	10.1%	57.8%	16.3%			
Boats	3.7%	70.7%	7.0%			
Canoe	20.8%	89.2%	33.7%			
Fall	9.3%	37.4%	14.8%			
Fountain1	1.4%	36.7%	2.6%			
Fountain2	4.2%	45.5%	7.7%			
Overpass	21.2%	66.9%	32.2%			
Night videos	33.4%	40.5%	34.2%			
Bridge entry	9.7%	27.4%	14.2%			
Busy boulevard	47.3%	25.0%	32.7%			
Fluid highway	21.7%	42.6%	28.7%			
Street corner	46.7%	60.2%	52.6%			
Tram station	44.8%	33.8%	38.5%			
Winter street	30.4%	53.7%	38.8%			
Shadow	51.3%	52.1%	49.7%			
Backdoor	32.4%	63.1%	42.8%			
Bungalows	56.0%	39.5%	46.3%			
Bus station	70.4%	60.4%	65.0%			
Copy machine	79.5%	52.4%	63.1%			
Cubicle	18.3%	29.1%	22.4%			
People in shade	51.5%	67.8%	58.5%			
Thermal	56.1%	60.3%	55.5%			
Corridor	43.1%	84.5%	57.0%			
Dining room	71.6%	46.8%	56.6%			
Lakeside	18.5%	38.3%	24.9%			
Library	88.6%	86.2%	87.4%			
Park	58.6%	45.8%	51.4%			
Global	37.9%	54.9%	39.8%			

TABLEAU 2.3 – Résultats complets sur CDNET de notre détection de nouveauté

	SOM Global			SOM Max	GNG Global	GNG Max
Catégories/vidéos	Précision	Rappel	F-mesure	F-mesure	F-mesure	F-mesure
Turbulence	10.6%	62.5%	13.7%			
Turbulence0	2.2%	77.1%	4.3%			
Turbulence1	8.6%	58.5%	15.0%			
Turbulence2	1.5%	78.4%	2.9%			
Turbulence3	30.2%	36.0%	32.7%			
Inter. motion [r]	?	?	?			
Street light	?	?	?			
Sofa	?	?	?			
Tramstop	?	?	?			
Low framerate [r]	?	?	?			
Turnpike	?	?	?			
Tram crossroad	?	?	?			
Tunnel exit	?	?	?			
Global	37.9%	54.9%	39.8%			

TABLEAU 2.4 – 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 comme *Dynamic background* et *Turbulence* avec des précisions ne dépassant pas les 10%.

[Ajouter remarques sur Max et GNG une fois que les valeurs seront là]

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 web¹ référence les résultats obtenus dans la littérature. Ceux-ci sont généralement obtenus en utilisant 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 2.5.

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%
SOM Global	37.9%	54.9%	39.8%

TABLEAU 2.5 – Comparatif avec d'autres modèles de détection de changement sur CDNET

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

Notre méthode est avant dernière en précision, 5^{ième} sur 9 en rappel et avant dernière en F-mesure, bien que plus proche des 6 et 7^{ièmes} que du dernier. Les chiffres indiqués ne sont pas non plus 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 être environ 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*.

2.5.3 Visualisation des résultats

Dans cette section nous avons regroupé les résultats de détection de notre modèle. La figure 2.25 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.

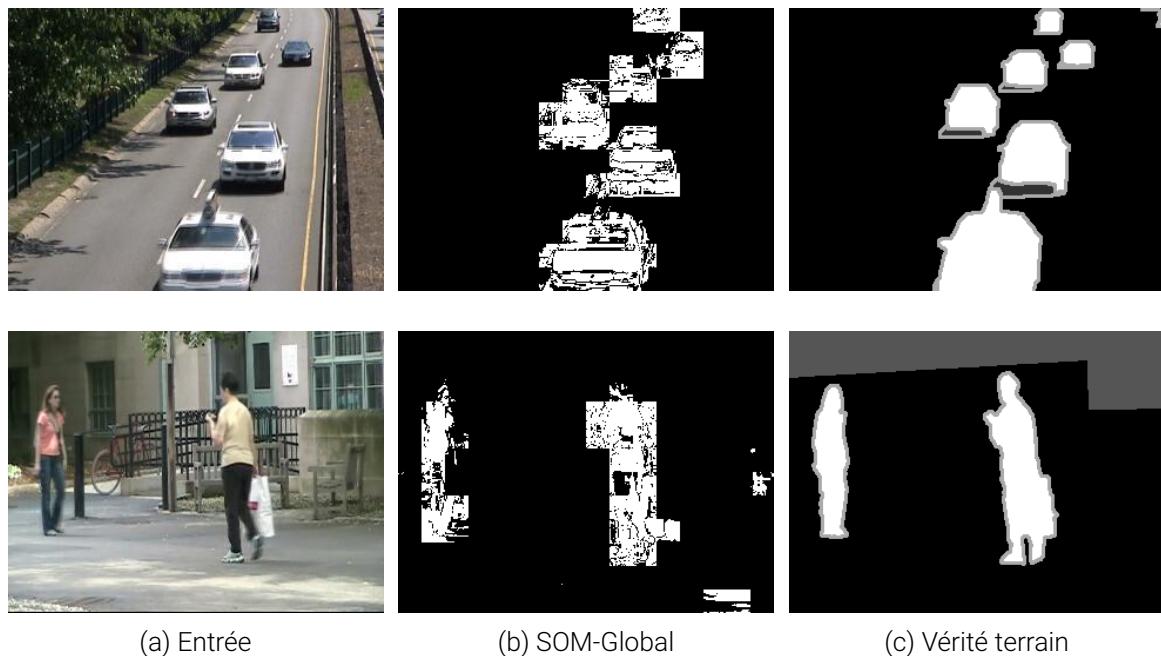


FIGURE 2.25

La figure 2.26 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.

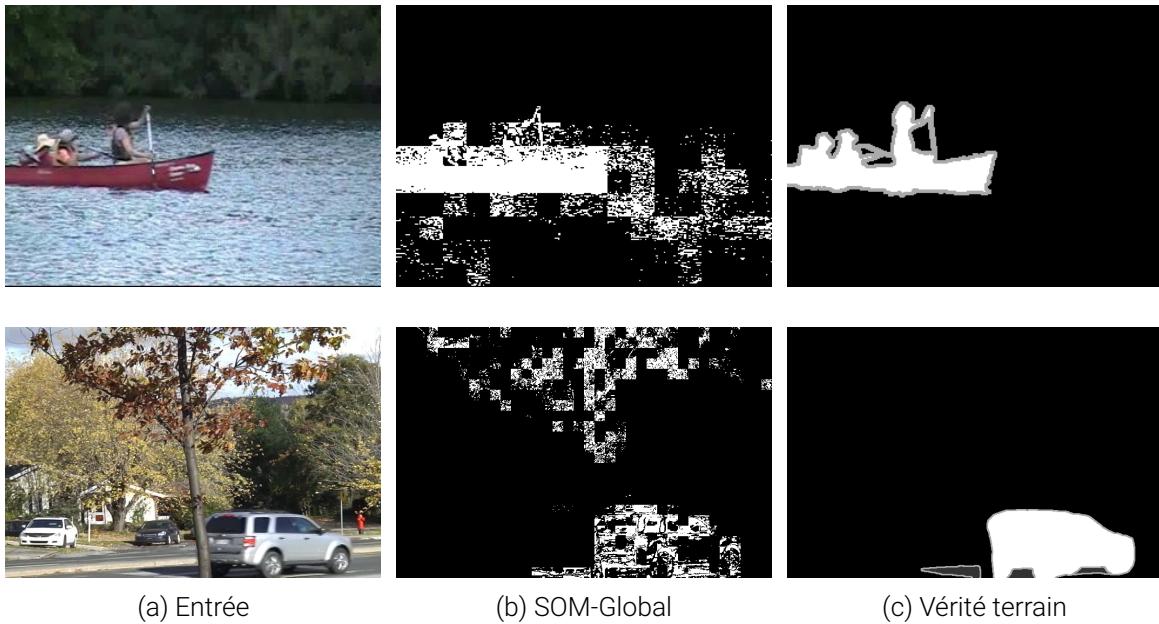


FIGURE 2.26

2.5.4 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 comparables avec les autres approches.

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.

2.6 Conclusion

2.7 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. [20](#)
- BERGSTRA, J., R. BARDET, Y. BENGIO et B. KÉGL. 2011, «Algorithms for hyper-parameter optimization», *Advances in neural information processing systems*, vol. 24. [38](#)
- 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. [23](#)
- 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. [23](#)
- 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. [33](#)
- 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. [32](#)
- 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. [32](#)
- POWERS, D. M. 2011, «Evaluation : from precision, recall and f-measure to roc, informativeness, markedness and correlation», *arXiv preprint arXiv:2010.16061*. [33](#)
- 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. [46](#)
- 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. [27](#)
- 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. [47](#)
- 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. [45](#)

Chapitre 3

Vision événementielle et attention

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

Maurice Merleau-Ponty

Sommaire

3.1	Introduction	50
3.2	Détection de mouvements	50
3.2.1	Multi cibles	51
3.2.2	Paramètres	52
3.3	Mécanisme attentionnel	53
3.4	Combinaison avec la détection de nouveauté	53
3.5	Conclusion	53
3.6	Références	53

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.

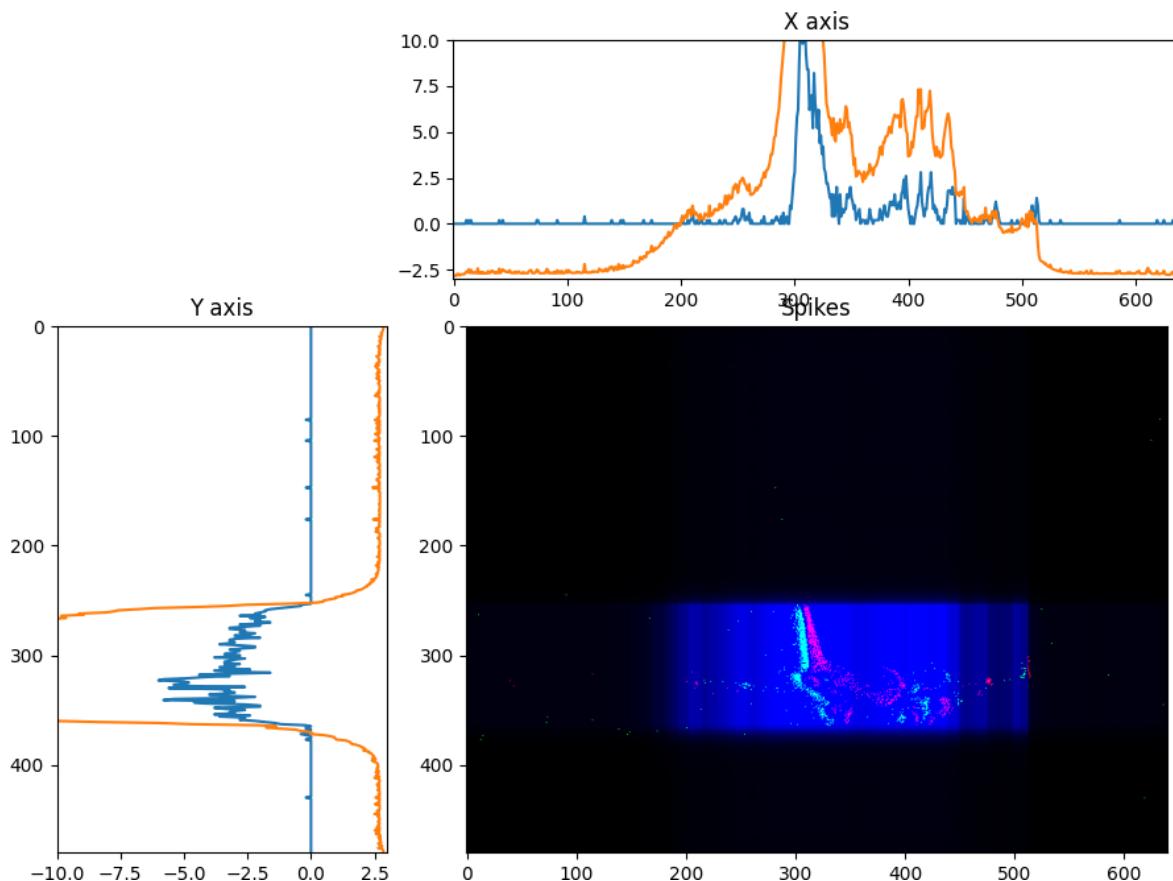


FIGURE 3.1 – Détection de mouvements avec DNF sur caméra événementielle. Les points verts et rouges dans l'image correspondent respectivement aux augmentations et réductions de luminosités détectés par la caméra. La somme des événements de chaque axe est affichée dans les courbes bleues qui sont en entrée 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

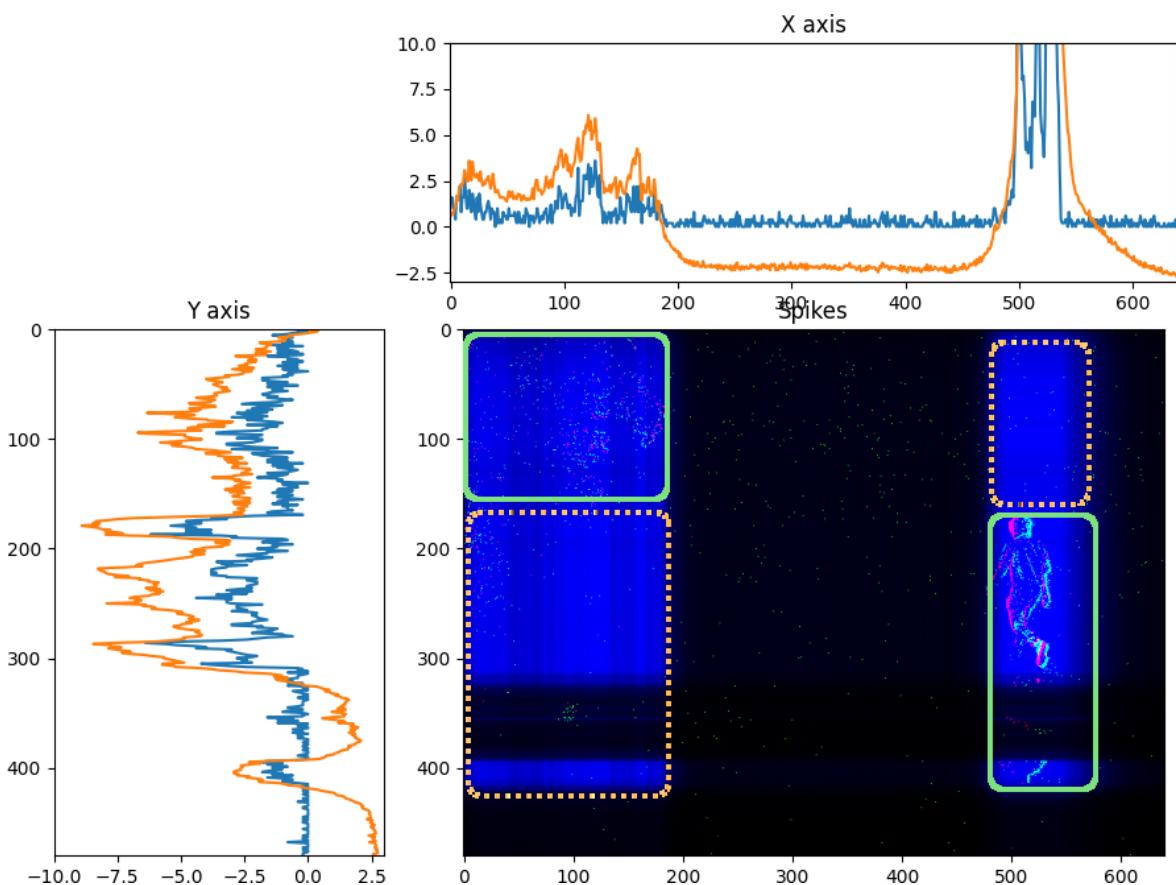


FIGURE 3.2 – Détection multicable. Il y a deux stimulus dans la scène, entourés par des lignes vertes. La personne qui se déplace en bas à droite et un mouvement dans un arbre en haut à gauche. Les activations parasites, dû à la projection de deux détections en 1D vers du 2D sont entourés par des pointillés 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	56
4.2	Observations topologiques	57
4.3	Algorithme séquentiel	58
4.3.1	Analyse de la complexité	61
4.3.2	Protocole expérimental	62
4.3.3	Résultats	63
4.4	Algorithme parallèle	67
4.5	Conclusion	67
4.6	Références	67

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 Observations topologiques

4.3 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.

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 2, 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).

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 gra-

Algorithm 1 FastBMU

Entrée :

vector : vecteur d'entrée
w, h : largeur et hauteur de la SOM

Variables :

dist_memory : historique des distances déjà évaluées

positions : positions de départ des particules

Sortie :

bmu : index de la Best Matching Unit

```

1: dist_memory ← None
2: positions ← [(0,0), (w-1, 0), (0, h-1), (w-1, h-1)]
3: bmu ← None
4:
5: if eval(pos) is True then
6:   return pos
7: end if
8: eval[pos] ← True
9: Let new_pos take the index of the closest not evaluated neighbour to the input
   vector (ignoring neighbours p such that eval(p)=True)
10: if values[new_pos] ≤ values[pos] then
11:   return Particle(new_pos)
12: end if
13: return The index of the smallest value between pos and the returned indexes by
   execution of Particle() on all direct neighbours of pos.
```

gradient descent goes in the direction of this particular neuron, that becomes the next 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

Algorithm 2 Particle

Input : pos : current position

Parameters : $values$: global table of distances, $eval$: global history of evaluations

Output : bmu : best matching unit's index

```

1: if  $eval(pos)$  is True then
2:   return  $pos$ 
3: end if
4:  $eval[pos] \leftarrow$  True
5: Let  $new\_pos$  take the index of the closest not evaluated neighbour to the input
   vector (ignoring neighbours  $p$  such that  $eval(p) =$ True)
6: if  $values[new\_pos] \leq values[pos]$  then
7:   return Particle( $new\_pos$ )
8: end if
9: return The index of the smallest value between  $pos$  and the returned indexes by
   execution of Particle() on all direct neighbours of  $pos$ .

```

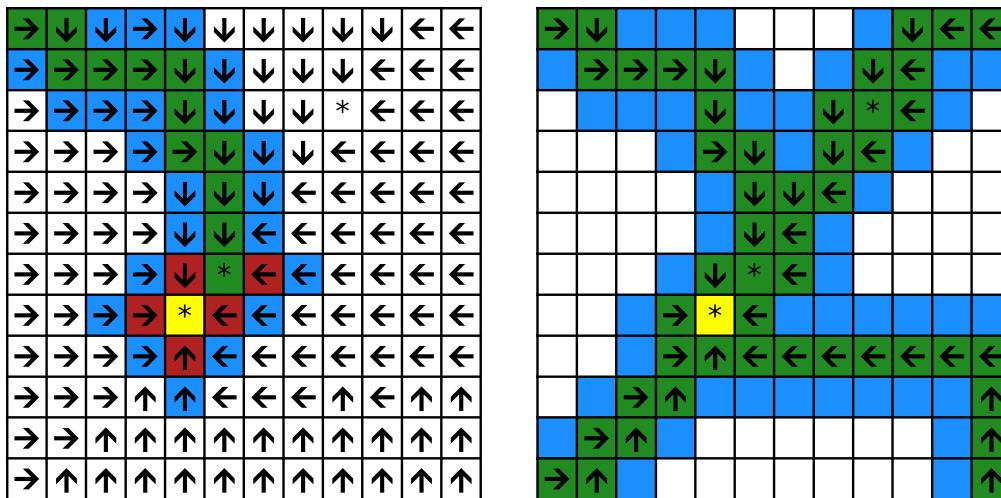


FIGURE 4.1 – Left : Example of the execution of one particle. Each cell represents one neuron. The arrow in each cell points towards the best neighbour (where a lower distance can be found). A star represents a local minimum. The green colored cells are the neurons who have been explored during the search, 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.

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).

4.3.1 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 additionons 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

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

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 4.3.3.

4.3.2 Protocole expérimental

To evaluate the robustness of our algorithm with various kinds of data, we have selected 6 datasets aimed at being representative of the kind of data SOMs are usually trained on. For the 2D case, we have generated data with different properties (uniform distribution and a highly non-convex shape), for 3D data we use a uniformly distributed cube shape and the pixel color values of an image. For the high dimensional case, we consider an image compression application ? with 10 by 10 sub-images as training vectors (100 pixels with 3 colors each, resulting in 300 dimensions), and the Free Spoken Digits Dataset ? which uses soundwaves that we have reduced to 1000 dimensions.

Our algorithm is mostly designed for a 2 dimensional SOM, although adaptations to higher dimensions can be easily defined by just adapting the neighbourhood and the number of initial particles, even leading to possibly greater gains in computational cost. We tested it with a grid and a hexagonal shaped 2D topology, which are the most commonly used in SOM applications.

In order to evaluate the differences between all tested models, we used three metrics. The first one is the standard *Mean Squared Quantization Error* (MSQE) which estimates the Vector Quantization quality of the tested algorithm. The *Mean Squared Distance to Neurons* (MSDtN) which computes the average squared codeword distance between neurons and all of their direct neighbours. The lower this value is, the more closely related neighbouring neurons are, and the better the dimensional reduction property is. Numerous metrics exist in the SOM literature to estimate the topographical mapping of the SOM, but MSDtN has the advantage of being easy to compute,

without parameters and only dependent on the neurons weights. For all metrics, lower is better.

$$\text{MSQE} = \frac{1}{V} \sum_{i=1}^V \|\nu_i - w_{\text{bmu}(i)}\|^2 \quad (4.3)$$

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

With V the number of vectors in the dataset, ν_i the weights of the i^{th} vector of the dataset, and $\text{bmu}(i)$ is the index of the neuron which is the best matching unit for the i^{th} vector of the dataset. Similarly N is the number of neurons and w_i the weights of the i^{th} neuron, while $\text{dist}(i, j)$ is the number of links in the shortest path between neurons i and j in the neural map.

4.3.3 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.

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

TABLEAU 4.1 – Results with a 32x32 neurons SOM, averaged over 50 executions. The Algorithm column specifies with which algorithm and topology the SOM was trained, FastG and FastH stand respectively for Fast Grid and Fast Hexagonal. MSQE_S is the MSQE calculated with the standard (exhaustive) BMU algorithm whereas MSQE_F uses the Fast-BMU version. Differences of MSQE_S between different algorithms reflect the quality of the training phase. The mismatch is the proportion of BMU that are differently selected by the two algorithms.

Données	Algorithmme	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%
Sons (2D)	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%
Image (2D)	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%

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%.

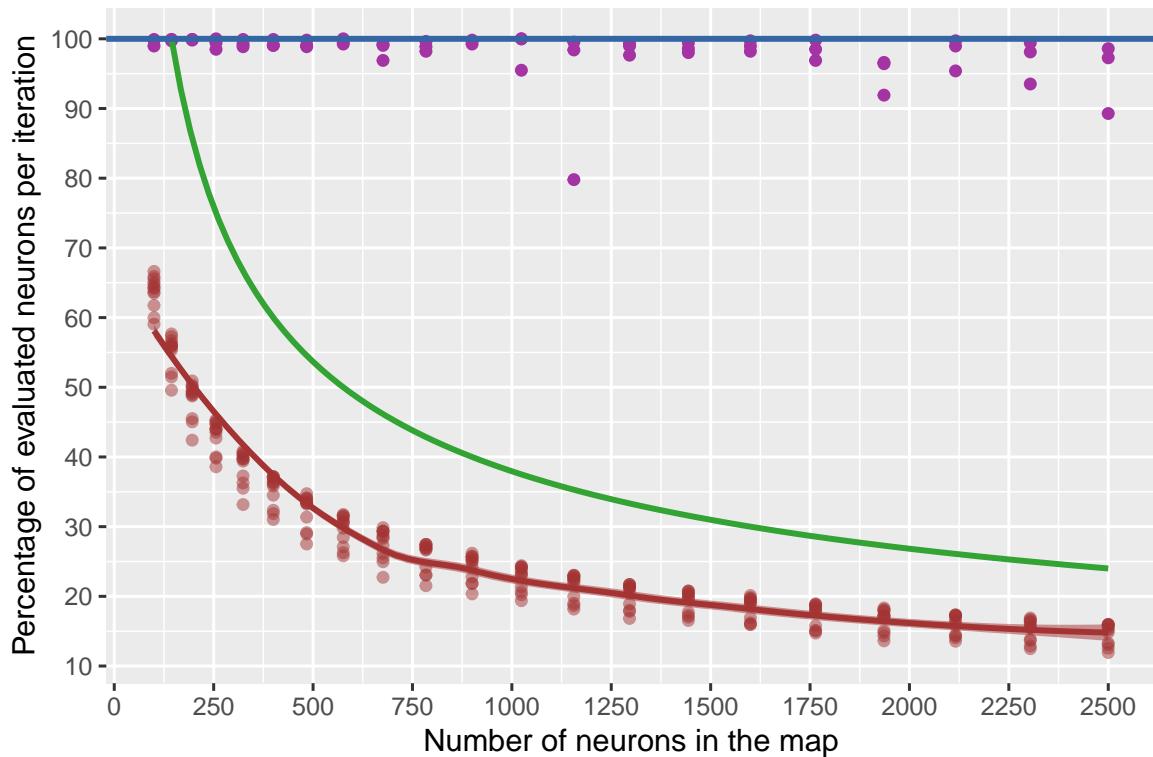


FIGURE 4.2 – Evaluation of performance gains with the number of neurons. All results were calculated on square maps. The blue line is the standard SOM, in green is the analytical value and in red the measured value. Additionally, the purple dots are the percentage of correct BMU in an execution on the Image dataset.

4.4 Algorithme parallèle

4.5 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.

4.6 Références

