Système de traitement d'images temps réel dédié à la mesure de champs denses de déplacements et de déformations.

Seyfeddine Boukhtache

Université Clermont Auvergne

École Doctorale : Sciences pour l'Ingénieur

THÈSE

présentée par

Seyfeddine BOUKHTACHE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ CLERMONT AUVERGNE

Spécialité : Électronique et Architecture de Systèmes

Système de traitement d'images temps réel dédié à la mesure de champs denses de déplacements et de déformations

Soutenue le 18 décembre 2020 devant le jury :

M. Jean-Charles PASSIEUX	Professeur des universités, INSA Toulouse	Président
M. Dominique GINHAC	Professeur des universités, Université de Bourgogne	Rapporteur
Mme. Virginie FRESSE	Maître de conférences HDR, Université Jean Monnet	Rapporteure
M. Frédéric SUR	Professeur des universités, Université de Lorraine	Examinateur
M. Michel GREDIAC	Professeur des universités, Université Clermont Auvergne	Directeur
M. François BERRY	Professeur des universités, Université Clermont Auvergne	Co-directeur
M. Benoît BLAYSAT	Maître de conférences HDR, Université Clermont Auvergne	Co-directeur

Université Clermont Auvergne, CNRS, Sigma Clermont, Institut Pascal - UMR 6602

CNRS/UCA/SIGMA

Résumé

Cette thèse s'inscrit dans un cadre pluridisciplinaire. Elle traite de la problématique du temps réel et de celle des performances métrologiques en traitement d'images numériques. Elle s'intéresse plus particulièrement à la photomécanique. Il s'agit d'une discipline récente visant à développer et à utiliser au mieux des systèmes de mesure de champs entiers de petits déplacements et de petites déformations en surface de solides soumis à des sollicitations thermomécaniques. La technique utilisée dans cette thèse est la corrélation des images numériques (CIN), qui se trouve être l'une des plus employées dans cette communauté. Elle représente cependant des limitations à savoir un temps de calcul prohibitif et des performances métrologiques améliorables afin d'atteindre celles des capteurs ponctuels classiques comme les jauges de déformation.

Ce travail s'appuie sur deux axes d'étude pour relever ce défi. Le premier repose sur l'optimisation de l'interpolation d'images qui est le traitement le plus coûteux dans la CIN. Une accélération est proposée en utilisant une implémentation matérielle parallélisée sur FPGA, tout en tenant compte de la consommation des ressources matérielles et de la précision. La principale conclusion est qu'un seul FPGA (dans les limites technologiques actuelles) ne suffit pas à implémenter l'intégralité de l'algorithme CIN. Un second axe d'étude a donc été proposé. Il vise à développer et à utiliser des réseaux de neurones convolutifs pour tenter d'atteindre à la fois des performances métrologiques meilleures que la CIN et un traitement en temps réel. Cette deuxième étude a montré l'efficacité d'un tel outil pour la mesure des champs de déplacements et de déformations. Elle ouvre de nouvelles perspectives en termes de performances métrologiques et de rapidité des systèmes de mesure de champs.

Mots clés : *Déplacements sous-pixelliques, Performances métrologiques, Interpolation, Précision, Consommation des ressources matérielles, Réseaux de neurones convolutifs, FPGA, GPU.*

Abstract

This PhD thesis has been carried out in a multidisciplinary context. It deals with the challenge of real-time and metrological performance in digital image processing. This is particularly interesting in photomechanics. This is a recent field of activity, which consists in developing and using systems for measuring whole fields of small displacements and small deformations of solids subjected to thermomechanical loading. The technique targeted in this PhD thesis is Digital Images Correlation (DIC), which is the most popular measuring technique in this community. However, it has some limitations, the main one being the computing resources and the metrological performance, which should be improved to reach that of classic pointwise measuring sensors such as strain gauges.

In order to address this challenge, this work relies on two main studies. The first one consists in optimizing the interpolation process because this is the most expensive treatment in DIC. Acceleration is proposed by using a parallel hardware implementation on FPGA, and by taking into consideration the consumption of hardware resources as well as accuracy. The main conclusion of this study is that a single FPGA (current technology) is not sufficient to implement the entire DIC algorithm. Thus, a second study has been proposed. It is based on the use of convolutional neural networks (CNNs) in an attempt to achieve both better metrological performance than CIN and real-time processing. This second study shows the relevance of using CNNs for measuring displacement and deformation fields. It opens new perspectives in terms of metrological performance and speed of full-field measuring systems.

Keywords : *Subpixel displacement, Metrological performances, Interpolation, Precision, Hardware resource consumption, Convolutional neuronal networks, FPGA, GPU.*

Table des matières

Liste des tableaux

Table des figures

Introduction générale

La métrologie est la science de la mesure. Elle consiste à quantifier une variable physique pour différencier un système d'un autre ou pour analyser le même système dans différentes circonstances. On peut distinguer différents types de métrologies, par exemple la métrologie optique. L'objectif de la métrologie optique est de mesurer certains paramètres physiques à l'aide de méthodes optiques, ce qui permet de garantir et de maintenir la confiance envers les mesures qui en résultent. Elle a pour avantage la possibilité d'effectuer la mesure sans contact [1]. La métrologie optique permet par exemple de quantifier des distances, des déplacements et des déformations. Actuellement, quasiment toute mesure optique s'appuie sur un encodage réalisé par une grille discrète du signal reçu de la scène observée. Cette grille est composée de pixels. Chaque pixel représente l'information qui en est proche. Les sous-pixels sont des régions infiniment plus petites qui se situent entre ces pixels. Dans le contexte de cette thèse, la résolution sous-pixellique est cruciale pour atteindre les performances souhaitées.

La performance sous-pixellique (SP) est un facteur très important dans le choix de la technique de traitement d'images numériques dans un contexte métrologique. Les techniques offrant des performances sous-pixeliques sont souvent complexes et coûteuses sur le plan des calculs. De plus, certains domaines comme la photomécanique, traitent des images de plusieurs millions de pixels, ce qui implique que certaines traitements peuvent durer plusieurs heures. Ainsi, disposer d'algorithmes rapides représente un objectif important pour cette communauté. Des travaux de cette communauté portent sur l'extension de techniques propres à celles du calcul de structure, tels que la décomposition de domaine [2] ou l'introduction de base réduite [3]. On trouve aussi dans la littérature des implémentations sur des architectures de calcul matérielles spécialisées dans l'accélération des calculs [4].

Des dispositifs de calcul parallèle, tels que -*Graphical Processing Unit*- (GPUs) et -*Field Programmable Gate Arrays*- (FPGAs) sont devenez classiques pour accélérer et rendre possible de nombreuses applications en temps réel [5, 6, 7, 8, 9]. Afin de bénéficier de leur capacité de calcul, ces deux cibles matérielles ont été utilisées dans le cadre de cette thèse.

Dans ce travail, nous offrons une étude sur les problématiques de métrologie par vision temps réel dans le contexte de la photomécanique. Ce contexte s'est révélé complètement approprié aux challenges que pourraient relever les architectures de calcul actuelles. Pour cette

raison, nous nous sommes intéressés à la méthode de corrélation d'images numériques (CIN[1]) qui est la plus utilisée dans les mesures métrologiques de champs de déplacements et de déformations en photomécanique [10]. En effet, cette technique présente un bon compromis entre facilité d'utilisation et performances métrologiques [10]. Pour une utilisation optimale de la CIN, la surface de l'éprouvette (petit morceau de matériau à la surface duquel on cherche à mesure les déformations) doit être recouverte d'un motif moucheté pour résoudre le problème d'unicité de pixels. La texture naturelle des surfaces peut suffire dans certains cas.

L'objectif principal de la CIN est d'obtenir des champs de déplacements et de déformations entiers dans une région d'intérêt d'une éprouvette sous sollicitation mécanique. Cette méthode utilise des techniques de traitement d'images pour résoudre ce problème. Fondamentalement, la procédure de CIN consiste à enregistrer des images numériques d'une éprouvette lorsqu'elle se déforme à l'aide d'une caméra et ensuite à traiter ces images à l'aide d'un code CIN afin de fournir des mesures de champs de déplacements et de déformations associés à ces images (voir la figure 1). Pour ce faire, la CIN locale prend de petites portions de l'image de référence (image initiale non déformée), appelées imagettes, et détermine leurs emplacements et leurs déformations respectifs dans l'image déformée. Dans la plupart des cas, ces imagettes sont choisies avec un pas d'espacement pour réduire le coût de calcul [11].

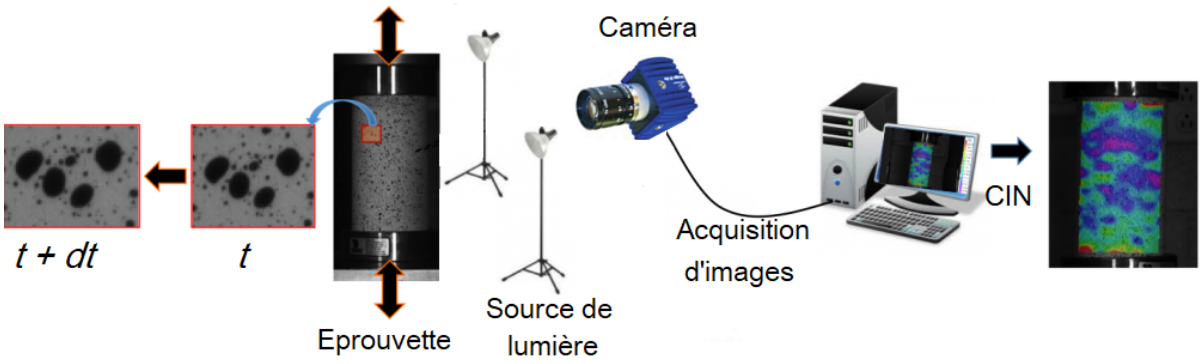


FIGURE 1 – Système de mesure par CIN [12].

Grâce à la baisse constante des coûts des systèmes de vision numérique et des architectures de calcul, la CIN devient de plus en plus accessible aux secteurs industriel et universitaire. Cependant, la CIN souffre de certaines limitations :

— La CIN est basée sur un processus de résolution itératif qui demande d'importantes ressources de calcul. Par conséquent, son utilisation est limitée lorsqu'il s'agit d'effectuer des mesures en temps réel.

— Une autre limitation réside dans le fait que la CIN agisse comme un filtre passe-bas. Dans son implémentation la plus courante, les champs de déplacements et de déformations récupérés sont des champs réels convolués par un filtre Savitzky-Golay. Cela signifie que

1. En anglais : Digital Image Correlation (DIC)

la CIN est incapable de rendre correctement les champs de déplacements ou de déformations présentant des fréquences spatiales élevées [13, 14].

Cette thèse a été conduite dans le but de limiter au mieux ces types d'inconvénients. Pour cela, ce travail a été mené en deux parties. Une première partie a cherché à accélérer la CIN en utilisant une cible FPGA. Dans cette première partie, nous avons identifié le bloc d'interpolation d'image comme étant le plus coûteux en termes d'intensité de calcul. Cette partie a traité donc des problèmes de temps réel, de précision, et de consommation des ressources matérielles dans le FPGA. Pour relever ces challenges, les premières contributions de ce travail sont orientées vers l'étude et l'optimisation de l'architecture FPGA de l'interpolation et de la précision numérique qu'elle implique. Il convient de noter que l'interpolation est un traitement clé dans plusieurs applications de traitement d'images. L'améliorer présente donc un intérêt qui dépasse la communauté de la photomécanique. Dans cette partie, nous avons conclu que malgré les améliorations apportées, il n'est pas possible d'implémenter la CIN sur un seul FPGA aussi puissant soit-il.

Pour cette raison, nous avons décidé de changer de paradigme en investiguant une nouvelle méthode de calcul de champs de déplacements et de déformations basée sur l'apprentissage profond. En effet, de nombreuses tâches de vision par ordinateur consistant à extraire des informations à partir de données brutes sont maintenant réalisées efficacement grâce à des techniques basées sur l'apprentissage profond, en particulier les réseaux de neurones convolutifs (CNNs [2]). Les CNNs ont été utilisés avec succès dans un grand nombre de problèmes de vision tels que la classification, la détection d'objets et le flot optique [15, 16, 17]. De plus, dans la plupart des applications, l'apprentissage profond surpasse les performances des algorithmes de vision conventionnels. Dans le contexte des mesures métrologiques de champs de déplacements et de déformations, à notre connaissance, les CNNs n'ont jamais été utilisés et ouvrent de nouvelles perspectives en matière de précision et de rapidité des systèmes de mesure. Afin d'optimiser l'implémentation nous avons choisi une cible de type GPU. Cela a fait l'objet de la seconde partie de ce travail. En règle générale, les CNNs consistent principalement de calculs mathématiques matriciels. Les GPUs sont adaptées pour l'apprentissage profond des réseaux de neurones car elles sont efficaces pour la multiplication des matrices et la convolution.

2. Convolutional Neural Networks

Plan du manuscrit et contributions

Ce manuscrit est structuré en six chapitres. Dans les deux premiers, nous présentons l'état de l'art concernant le traitement sous-pixellique, l'interpolation, et la problématique de précision et de consommation des ressources matérielles. Ensuite, les contributions propres à ce travail, sont détaillées dans les chapitres 3 à 5. Ces derniers ont fait l'objet de trois revues internationales. Enfin, le chapitre 6 résume les principales conclusions ainsi que les perspectives dégagées à l'issue de ce travail.

— **Le chapitre 1** a pour objectif de contextualiser le cadre dans lequel s'inscrit ce travail. D'abord, nous présentons le contexte du traitement sous-pixellique. Ensuite nous présentons l'interpolation, qui se révèle être le traitement clé pour atteindre des performances sous-pixelliques, et nous rappelons les différentes techniques pour réaliser cette interpolation. Et finalement, nous focalisons notre propos sur la CIN et ses problématiques.

— **Le chapitre 2** traite de la problématique de précision et de consommation des ressources matérielles. Il décrit les techniques les plus utilisées dans la littérature pour réduire la complexité des calculs, et en particulier la méthodologie utilisée pour optimiser et estimer l'erreur liée au calcul arithmétique en virgule fixe afin de répondre aux contraintes de précision.

— **Le chapitre 3** présente une implémentation matérielle de l'interpolation bi-cubique hétérogène. L'architecture proposée est optimisée afin de réduire le nombre de multiplieurs et d'éviter les calculs redondants. De plus, la précision arithmétique par rapport à l'utilisation des ressources matérielles est étudiée afin de minimiser l'erreur de quantification et d'obtenir le meilleur compromis entre coût de conception et précision. Ce chapitre est présenté sous la forme d'un article publié en novembre 2020 dans le "*Journal of Real-Time Image Processing*".

— **Le chapitre 4** propose un ensemble d'algorithmes qui approchent l'interpolation bi-cubique et réduit la consommation des ressources matérielles. Ces algorithmes proposés sont basés sur des combinaisons d'interpolations linéaires et cubiques. Ils sont étudiés et comparés en termes de qualité d'interpolation et de consommation des ressources matérielles. La précision arithmétique en regard de l'utilisation des ressources matérielles est aussi étudiée pour chaque algorithme. Ce chapitre fait l'objet d'un article accepté en octobre 2020 dans le Journal "*IEEE Transactions on Very Large Scale Integration (VLSI) Systems*".

— **Le chapitre 5** présente quant à lui un CNN capable de déterminer les champs de déplacements et de déformations à partir de paires d'images de référence et déformée, comme

dans le cas de la CIN. Dans ce chapitre, nous expliquons comment un CNN appelé StrainNet a été développé pour atteindre cet objectif, et comment des ensembles d'images de références et déformées ont été produits pour l'entraînement de ce CNN. StrainNet offre une alternative viable à la CIN. En effet, il fournit des résultats comparables, voire supérieurs à ceux de la CIN, en termes de performances métrologiques et de temps de calcul. Ce chapitre est présenté sous la forme d'un article publié en 2020 dans le Journal "*Optics and Lasers in Engineering*".

— **Le chapitre 6** conclut ce manuscrit et offre les principales perspectives qui en découlent.

Bibliographie

[1] Peter DE GROOT. *Optical Metrology, Basic Foundations and Practical Applications*. 2007.

[2] J.-C. Passieux, J.-N. Périé, and M. Salaün. A dual domain decomposition method for finite element digital image correlation. *International Journal for Numerical Methods in Engineering*, 102(10) :1670–1682, 2015.

[3] J.-C Passieux, R. Bouclier, and J.-N Périé. A space-time PGD-DIC algorithm. *Experimental Mechanics*, 58(7) :1195–1206, 2018.

[4] L. Zhang, T. Wang, Z. Jiang, Q. Kemao, Y. Liu, Z. Liu, L. Tang, and S. Dong. High accuracy digital image correlation powered by GPU-based parallel computing. *Optics and Lasers in Engineering*, 69 :7 – 12, 2015.

[5] P. H. W. Leong. Recent trends in FPGA architectures and applications. In *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*, pages 137–141, 2008.

[6] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar. FPGAs in industrial control applications. *IEEE Transactions on Industrial Informatics*, 7(2) :224–243, 2011.

[7] J. Xu, J. Geng, X. Yan, H. Wang, and H. Xia. Implementing real time image processing algorithm on FPGA. In *Twelfth International Conference on Digital Image Processing (ICDIP 2020)*, volume 11519, page 115191L. International Society for Optics and Photonics, 2020.

[8] S. Gandhare and B. Karthikeyan. Survey on FPGA architecture and recent applications. In *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, pages 1–4, 2019.

[9] R. Menaka, S. Janarthanan, and K. Deeba. FPGA implementation of low power and high speed image edge detection algorithm. *Microprocessors and Microsystems*, page 103053, 2020.

[10] M. Sutton, J.J. Orteu, and H. Schreier. *Image Correlation for Shape, Motion and Deformation Measurements. Basic Concepts,Theory and Applications*. Springer, 2009.

[11] B. Blaysat, J. Neggers, M. Grédiac, and F. Sur. Towards criteria characterizing the metrological performance of full-field measurement techniques. Application to the comparison between local and global versions of DIC. *Experimental Mechanics*, 60(3) :393–407, 2020.

[12] Devin harris, at the 1st international digital image correlation conference and workshop. https ://uva-moblab.com/devin-harris-presents-at-the-1st-international-digital-image-correlation-conference-workshop.

[13] H. W. Schreier, M. Sutton, and A. Michael. Systematic errors in digital image correlation due to undermatched subset shape functions. *Experimental Mechanics*, 42(3) :303–310, 2002.

[14] F. Sur, B. Blaysat, and M. Grédiac. On biases in displacement estimation for image registration, with a focus on photomechanics-extended version. 2020.

[15] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[16] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet : Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0 : Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1647–1655, 2017.

Chapitre 1

Traitement sous-pixellique et performances métrologiques

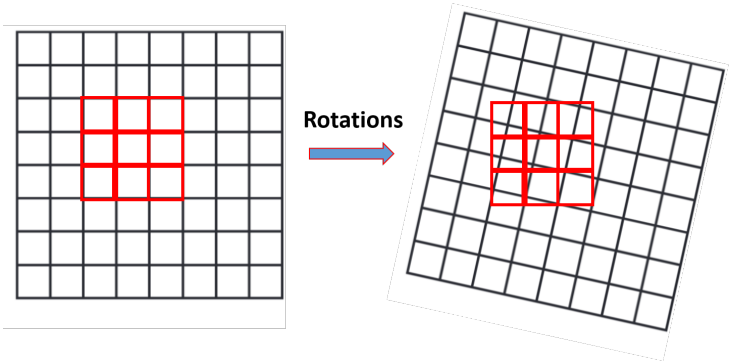
Le traitement d'images constitue un domaine de recherche important en ingénierie et en informatique. Il permet d'effectuer certaines opérations sur des images, afin d'améliorer leur qualité ou d'en extraire des informations utiles pour une application donnée. Il s'agit d'un système de traitement dans lequel l'entrée est une image et la sortie peut être une image ou des caractéristiques/fonctions associées à cette image.

En général, une mise en correspondance entre deux images de la même scène, à savoir l'image de référence et l'image courante, est réalisée afin de pouvoir comparer ou combiner leurs informations respectives. Cette mise en correspondance se fait par la recherche d'une quantité géométrique permettant de passer d'une image à une autre ou d'un pixel à un autre.

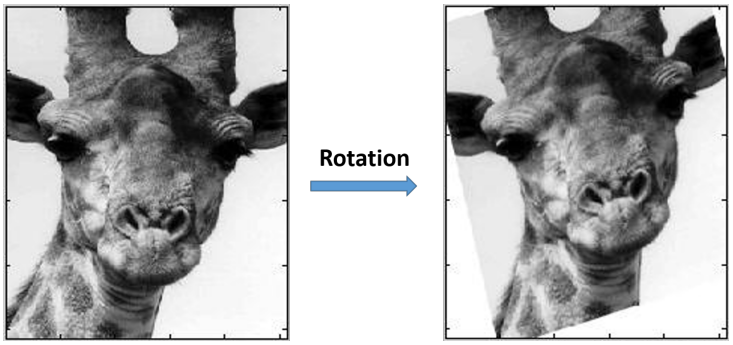
Le traitement sous-pixellique (SP) consiste à estimer la valeur de cette quantité géométrique avec une précision inférieure à 1 pixel, à partir des données échantillonnées (pixels) de l'image numérique. Le pixel est la plus petite unité physique fournit par la caméra en observant une scène continue. Chaque pixel de l'image représente uniquement l'information qui en est proche. Entre les pixels, il y a des régions infiniment plus petites appelées "sous-pixels". La précision des sous-pixels fait référence à la subdivision entre deux pixels adjacents. Cela signifie que chaque pixel sera divisé en unités plus petites. Par exemple, si un quart de pixel est choisi, chaque pixel est considéré comme quatre pixels à la fois dans les directions horizontale et verticale (voir la figure 1.2a). Les informations à une échelle sous-pixellique sont perdues lorsque les données continues de l'image sont échantillonnées ou quantifiées en pixels. Cependant, il est possible d'estimer des quantités géométriques avec une précision SP en se basant sur des algorithmes tels que l'interpolation.

La performance SP est cruciale pour de nombreuses applications de traitement d'images numériques telles que la super résolution [1, 2, 3], les transformations géométriques (voir un exemple de rotation dans la figure 1.1), l'estimation des mouvements sous-pixelliques [4], la compression vidéo [5], ou les mesures métrologiques par vision [6]. Par exemple, dans le cas de la super résolution, on cherche à récupérer une image haute résolution à partir d'une image

basse résolution afin d'améliorer la qualité et les détails de cette dernière. Pour cela des informations SP sont nécessaires (voir la figure 1.2).

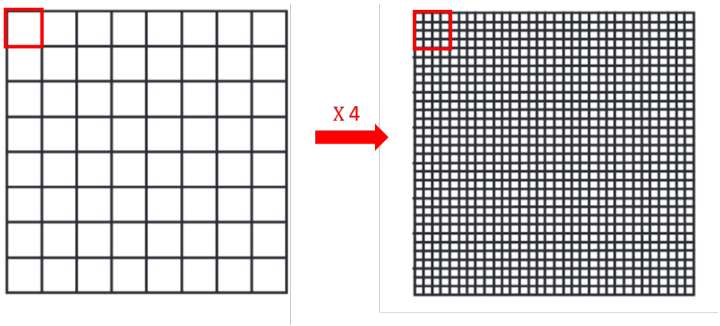


(a) Principe de la rotation d'image.

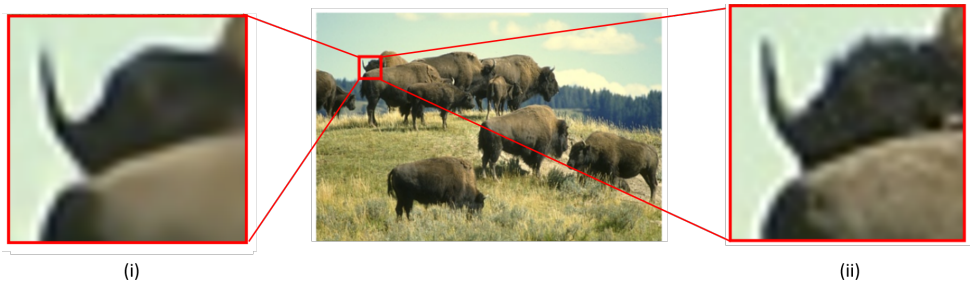


(b) Exemple de rotation.

FIGURE 1.1 – Traitement sous-pixellique : transformation géométrique.



(a) Principe de la super résolution.



(b) Exemple de la super résolution : (i) en utilisant un réseau convolutif profond (ii) la vérité terrain [7].

FIGURE 1.2 – Traitement sous-pixellique en super résolution.

L'objectif de ce chapitre est d'introduire la technique de mesure de champs de déplace-

ments sous-pixeliques appelée corrélation d'images numériques (CIN ou Digital Images Correlation, DIC, en anglais), et d'identifier le traitement critique en termes de temps de calcul et de performance de la méthode. Pour cela, nous commençons par rappeler les différentes approches utilisées en traitement d'images à performance SP. Ensuite, nous présentons une étude sur les différentes techniques d'interpolation afin de choisir l'algorithme d'interpolation offrant le meilleur compromis entre précision et complexité de calcul. Enfin, comme nous plaçons ce travail dans un contexte métrologique, ce qui implique de grands défis en termes d'exactitude de mesure, la CIN et ses enjeux sont détaillés.

1.1 Les approches à performance sous-pixelique

Dans ce paragraphe, nous rappelons les techniques les plus utilisées en traitement d'images pour obtenir une performance sous-pixelique.

1.1.1 Interpolation de la fonction de corrélation

La performance de la corrélation d'images[1] (ici, la corrélation d'images ne signifie pas la CIN telle utilisée en photomécanique) est limitée par la résolution au pixel entier. Une approche traditionnelle pour obtenir une performance SP consiste à utiliser une corrélation sur-échantillonnée. Pour cela, on calcule d'abord la fonction de corrélation entre les deux images. Elle est ensuite sur-échantillonnée dans le domaine de Fourier afin de localiser le pic de corrélation avec une performance SP [8, 9, 10, 11, 12, 13].

1.1.2 Interpolation de l'intensité de l'image

Une autre approche pour atteindre une performance SP est basée sur l'interpolation de l'intensité de l'image. Elle consiste à sur-échantillonner l'intensité de l'image, ou les niveaux de gris dans le cas d'une image monochrome, pour obtenir une résolution plus élevée [9, 14]. Ceci est réalisé en utilisant l'interpolation pour créer une grille beaucoup plus dense de l'image de référence. Ensuite, cette image sur-échantillonnée est utilisée dans le traitement souhaité. Par exemple, étant donné une image de référence de taille N$\times$N, si on souhaite obtenir une erreur de 0,1 pixel près, l'image de référence doit être interpolée pour créer une nouvelle version de dimensions (10xN)$\times$(10xN).

1.1.3 Flot optique sous-pixellique

La mesure du flot optique (FO) à partir de séquences d'images est une tâche importante dans le domaine de la vision par ordinateur et du traitement d'images. Horn et Schunck [15] ainsi que Lucas et Kanade [16] ont été les premiers à proposer une méthode différentielle pour

1. Corrélation croisée ou inter-corrélation

calculer le FO en 1981. Un grand nombre d'extensions et de modifications ont ensuite été proposées dans la littérature [17, 18, 19, 20, 21, 22, 23, 24]. Ces techniques sont classées en quatre catégories : différentielles, basées sur les régions, basées sur des caractéristiques, et fréquentielles. Les méthodes fréquentielles basées sur la phase et différentielles sont les prédominantes en estimation du FO avec une performance SP [19, 20, 25], de plus amples détails sont donnés ci-dessus.

1.1.3.1 Techniques fréquentielles

Les techniques fréquentielles calculent le FO dans le domaine de Fourier en utilisant des filtres adaptés au FO, comme les filtres de Gabor [20]. Elles peuvent être classées en deux groupes. Les méthodes basées sur l'énergie [26] et celles basées sur la phase [23]. Les méthodes basées sur la phase sont considérées comme supérieures à celles basées sur l'énergie sous deux aspects : la performance sous-pixellique et la robustesse aux changements d'éclairage, car les informations de phase sont moins sensibles aux changements de contraste [20].

Les méthodes basées sur la phase décomposent l'image originale en canaux en utilisant différents filtres passe-bande. Le flot optique est défini en termes de comportement de phase en sorties de ces filtres [23, 20].

1.1.3.2 Techniques différentielles

Également connues sous le nom "techniques basées sur le gradient" [20]. L'idée derrière cette méthodologie est de relier la différence entre deux images successives au gradient d'intensité de la première image. Ces approches peuvent être classées comme globales ou locales, en fonction de la quantité d'informations et des contraintes de régularisation introduites pour estimer le FO.

L'approche globale repose sur une contrainte de régularisation globale sur l'intégralité des déplacements de la région traitée pour fournir un champ dense. Le système d'équations obtenu dans ce cas est généralement résolu par des méthodes itératives, comme celle de Gauss–Seidel [15].

L'approche locale résout le problème du FO en considérant une fenêtre autour de chaque pixel. Elle utilise une régularisation avec contrainte locale et s'appuie sur la minimisation des moindres carrés pour estimer le déplacement convenable. Cependant, des méthodes plus complexes peuvent être appliquées pour obtenir une meilleure estimation [16, 6].

1.1.4 Apprentissage profond

Plus récemment, une nouvelle classe d'algorithmes basée sur l'apprentissage profond a connu un succès impressionnant pour la résolution d'une grande variété de problèmes de traitement d'images. Une particularité de l'apprentissage profond est qu'un réseau est entraîné pour une application et un ensemble de données spécifiques. Les algorithmes basés sur l'apprentissage profond offrant les meilleures performances présentent généralement une grande

complexité de calcul. Les réseaux de neurones convolutifs sont de plus en plus utilisés pour remplacer les algorithmes de traitement d'images classiques, y compris l'estimation du flot optique [27, 28, 29, 30, 31] et la super résolution [32, 33, 34, 35].

1.1.5 Bilan

Les deux approches basées sur le sur-échantillonnage (interpolation de la fonction de corrélation et interpolation de l'intensité de l'image) peuvent donner des résultats précis, mais elles sont les plus coûteuses en termes de coût de calcul. Les méthodes d'estimation basées sur le gradient sont plus précises que les méthodes de corrélation de phase. Par contre, elles sont plus coûteuses sur le plan des calculs. Les méthodes de corrélation de phase engendrent un biais d'estimation relativement important. Elles n'ont donc pas été largement utilisées pour les applications de métrologique à hautes performances métrologiques.

Malgré leur exigence en termes de coût de calcul, les méthodes basées sur le gradient associées avec un processus itératif restent les plus adaptées dans un contexte métrologique [16, 6]. C'est pourquoi, dans ce chapitre, nous nous concentrons sur les méthodes différentielles offrant une étude complète sur la méthode de corrélation d'images numériques, la méthode la plus utilisée pour les mesures de déplacements en photomécanique [6].

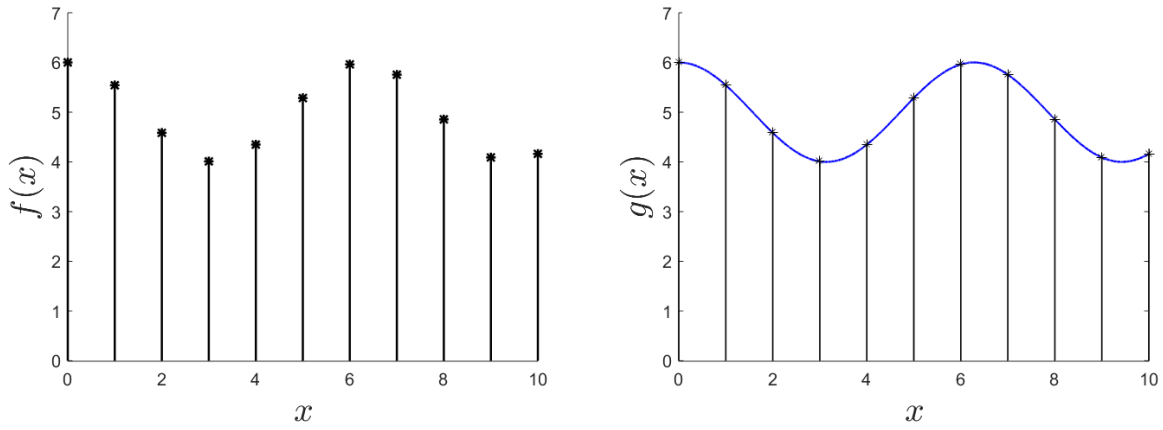
Dans beaucoup de techniques de traitement SP, y compris la CIN, l'interpolation est considérée comme un traitement très important. En effet, la complexité et la précision de la technique dépendent fortement de l'algorithme d'interpolation utilisé. Pour cela, avant de détailler la technique de CIN, nous proposons d'abord d'évaluer différentes méthodes d'interpolation utilisées dans le domaine de traitement d'image afin de choisir celle qui offre le meilleur compromis entre complexité de calcul et performances métrologiques des résultats.

1.2 Techniques d'interpolation

L'interpolation est le processus d'estimation des valeurs intermédiaires d'une fonction ou d'un signal échantillonné à des positions continues, ou la reconstitution de la fonction continue originale à partir d'un ensemble d'échantillons discrets (voir la figure 1.3) [36].

L'interpolation d'images numériques est le processus de génération d'une surface d'intensité continue à partir d'échantillons de données d'images discrètes, en leur appliquant une convolution par un noyau d'interpolation spécifique. De nos jours, de nombreux appareils électroniques utilisent des techniques d'interpolation d'images pour effectuer diverses tâches telles que la super résolution et l'estimation des déplacements. Dans ces tâches, la qualité des résultats obtenus dépend fortement de la technique d'interpolation utilisée.

La performance et le coût de calcul des algorithmes ou la consommation des ressources matérielles d'interpolation sont directement liés à leur noyau d'interpolation (différents noyaux sont définis dans les paragraphes suivants). Par conséquent, les noyaux d'interpolation font l'objet d'analyses et d'améliorations.



(a) (b)

FIGURE 1.3 – Exemple d'interpolation : (a) les données discrètes, (b) la construction d'un signal continu par interpolation.

Différents algorithmes d'interpolation (ou noyaux d'interpolation) sont proposés dans la littérature pour interpoler les images numériques [37, 38, 39]. Fondamentalement, ces algorithmes sont classés en deux catégories [40] : les algorithmes d'interpolation non adaptative et les algorithmes d'interpolation adaptative. L'interpolation non adaptative [40] traite l'ensemble de l'image pendant le processus d'interpolation sans discrimination, quel que soit son contenu. L'interpolation adaptative [41, 42, 43, 44] effectue des opérations basées sur la structure locale de l'image et ses variations d'intensité.

Chaque catégorie d'algorithmes a ses propres avantages et inconvénients. Les algorithmes d'interpolation adaptative se caractérisent par un coût de calcul plus élevé. Ils nécessitent beaucoup plus de temps de calcul pour extraire les caractéristiques locales de l'image que les algorithmes non-adaptatifs, ce qui signifie qu'ils sont très coûteux [41, 42, 43, 44]. Les algorithmes non adaptatifs sont moins complexes car ils effectuent le même processus sur toute l'image [37, 38, 39]. En outre, les techniques non adaptatives sont principalement utilisées pour des applications en temps réel en raison de leur faible complexité de calcul par rapport aux techniques adaptatives.

Dans ce travail, nous nous concentrons sur les algorithmes d'interpolation non adaptative, que nous qualifions de bon compromis entre qualité d'interpolation et complexité de calculs.

La fonction d'interpolation unidimensionnelle s'écrit comme suit :

$$g(x) = \sum_{k=0}^{3} f(x_k)\beta(x - x_k), \forall_x \in [x_1, x_2] \tag{1.1}$$

où $f(x_k)$, $k = 0...3$, représente l'ensemble des données discrètes, β est le noyau d'interpolation et g est la fonction d'interpolation.

1.2.1 Interpolations simples

L'interpolation par le plus proche voisin est la technique d'interpolation la plus simple et la plus rapide. Elle suppose que les points d'interpolation souhaités sont définis par le pixel le plus proche, comme le montrent l'équation (1.2) et la figure 1.4.

Elle est efficace sur le plan des calculs et très simple à mettre en œuvre, mais elle est la moins précise (voir la figure 1.6a).

$$\beta(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & others \end{cases} \tag{1.2}$$

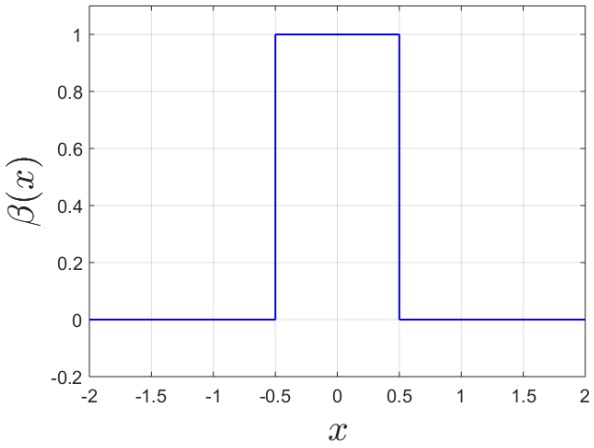


FIGURE 1.4 – Le noyau d'interpolation du plus proche voisin.

L'interpolation linéaire est également une technique simple, elle estime la valeur d'interpolation en reliant deux valeurs connues adjacentes par une ligne droite, comme décrit par d'équation (1.3) et dans la figure 1.5.

$$\beta(x) = \begin{cases} 1 - |x| & 0 \leq |x| < 1 \\ 0 & others \end{cases} \tag{1.3}$$

La figure 1.6 montre que l'interpolation linéaire offre une meilleure qualité d'interpolation par rapport au plus proche voisin. Par contre, cette technique ne préserve pas les détails du signal et elle n'est utilisée que par des applications tolérantes en performance.

Dans ce qui suit, avant d'évaluer les noyaux d'interpolation d'ordre supérieur pour obtenir une meilleure qualité d'interpolation, nous abordons d'abord le cas d'un interpolateur idéal.

1.2.2 Interpolation idéale

Selon la théorie du traitement du signal, un signal continu $g(x)$ peut être reconstruit à partir de ses échantillons $f(x)$, si ce signal a une bande de fréquence limitée et s'il est échantillonné conformément au théorème d'échantillonnage de Nyquist, en convoluant ses échantillons avec la fonction $sinc$ (figure 1.7) [36].

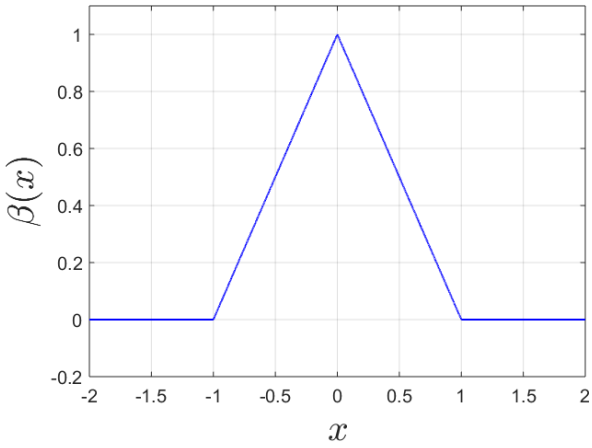
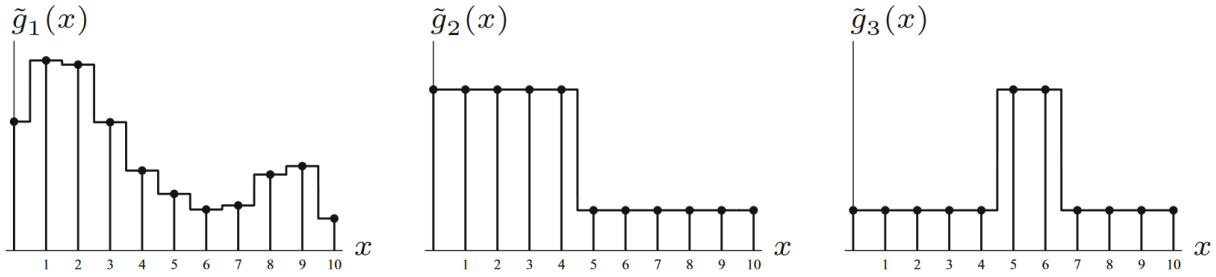
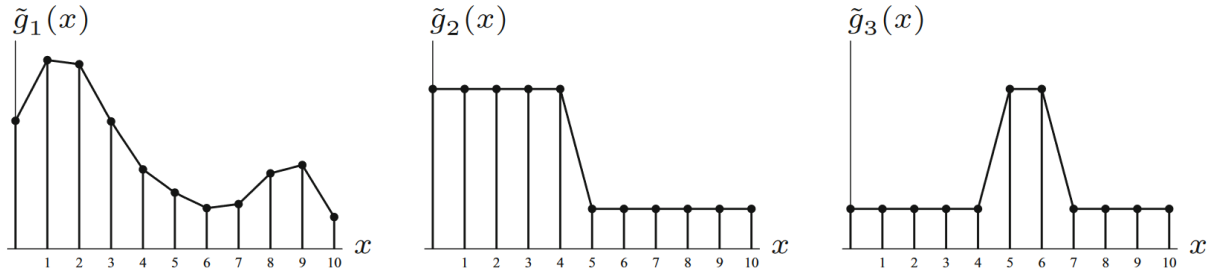


FIGURE 1.5 – Le noyau d'interpolation linéaire.



(a) Le plus proche voisin



(b) Interpolation linéaire

FIGURE 1.6 – Interpolation appliquée à trois types de signaux. g_1 signal continu à bande fréquentielle limitée, g_1 signal échelon, g_3 signal impulsion [36].

En théorie, la fonction $Sinc$ est le noyau idéal, par contre, en raison de sa nature infinie dans le domaine spatial, elle ne constitue pas un noyau d'interpolation possible dans la pratique, car il est difficile de convoluer un signal avec une fonction infinie. Par conséquent, plusieurs méthodes d'interpolation utilisent une version tronquée ou approchée de la fonction $Sinc$. Dans la suite, nous examinons deux catégories de noyaux d'interpolation. La première est basée sur une approximation de la fonction $Sinc$ par des polynômes cubiques par morceaux, et la seconde catégorie est basée sur une fonction $Sinc$ pondérée par une fenêtre.

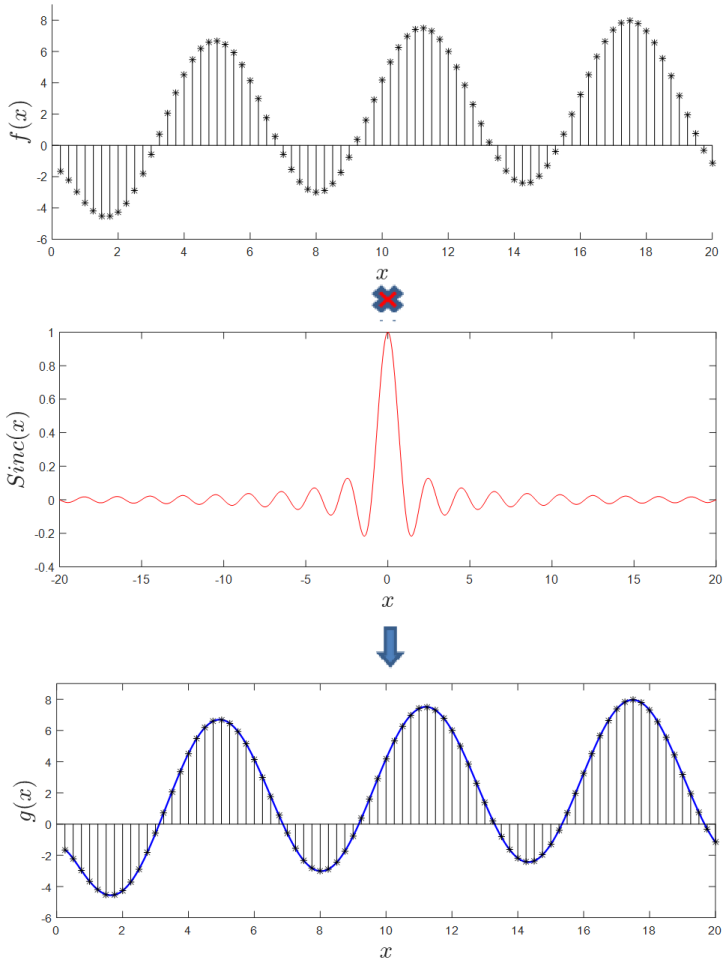


FIGURE 1.7 – Le noyau d'interpolation idéal : f(x) sont des échantillons du signal continu g(x). Ici nous représentons une fonction $Sinc$ finie mais en théorie elle a un support infini.

En général, plus le masque ou le support du noyau d'interpolation est grand, meilleure est la qualité d'interpolation, par contre, cela augmente significativement la complexité de calcul. Nous nous intéressons ici à un support de 4 pixels, le plus utilisé [45] dans la littérature. Il offre à la fois des bons résultats et moins de coût de calcul et plus particulièrement moins de consommation de ressources matérielles par rapport aux plus grands supports.

1.2.3 Interpolations basées sur des noyaux polynomiaux cubiques par morceaux

Des noyaux polynomiaux par morceaux peuvent être utilisés afin de s'approcher de la fonction $Sinc$. Plus l'ordre polynomial des morceaux est élevé, plus l'approximation du noyau $sinc$ s'améliore. D'autre part, la complexité de calcul augmente avec l'ordre polynomial. Ainsi, dans ce travail, nous nous intéressons aux polynômes du troisième ordre car ils offrent un bon compromis entre qualité d'interpolation et complexité du calcul. Une famille de noyaux polynomiaux du troisième ordre [36] est décrite dans l'équation (1.4). Cette fonction dépend de deux paramètres de contrôle (a, b).

$$\beta(x,a,b) = 1/6 \begin{cases} (-6a-9b+12)|x|^3 + (6a+12b-18)|x|^2 - 2b + 6 & 0 \leq |x| < 1 \\ (-6a-b)|x|^3 + (30a+6b)|x|^2 + (-48a-12b)|x| + 24a + 8b & 1 \leq |x| < 2 \\ 0 & others \end{cases} \tag{1.4}$$

Les paramètres de contrôle (a, b) les plus populaires sont discutés ci-dessous.

1.2.3.1 Interpolation cubique standard $(a, b) = (1, 0)$

L'interpolation cubique standard est un algorithme d'interpolation d'image avancé par apport à ceux déjà présentés car il prend en compte quatre pixels adjacents. Un poids supérieur est donné aux pixels proches et on applique un noyau d'ordre 3. Les paramètres de contrôle (a, b) sont fixés à $(1, 0)$. La figure 1.8 et l'équation (1.5) correspondent au noyau d'interpolation obtenu.

$$\beta(x) = \begin{cases} |x|^3 - 2|x|^2 + 1 & 0 \leq |x| < 1 \\ -|x|^3 + 5|x|^2 - 8|x| + 4 & 1 \leq |x| < 2 \\ 0 & others \end{cases} \tag{1.5}$$

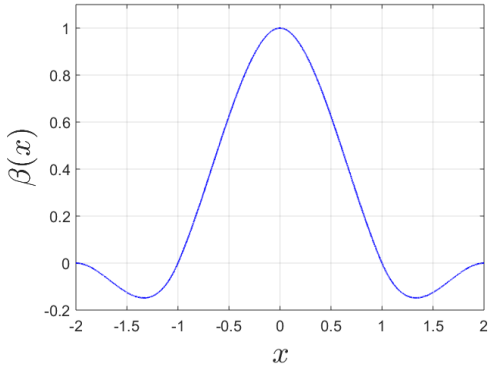


FIGURE 1.8 – Le noyau d'interpolation cubique standard.

L'interpolation cubique standard génère de meilleures images par rapport à l'interpolation par le plus proche voisin et à l'interpolation linéaire. Le temps de calcul pour cette méthode est plus important car elle nécessite un plus grand nombre de pixels pour le calcul ainsi qu'un noyau d'interpolation avec un ordre élevé. Cet algorithme produit des résultats de bonne qualité. Il est devenu un standard dans la majorité des logiciels d'édition d'images, tel Adobe Photoshop et les pilotes d'imprimante [36, 45, 37]. Par contre, comme illustré dans la figure 1.13a, il présente des dépassements importants sur les bords ainsi que de forts effets d'ondulation dans les parties continues du signal.

1.2.3.2 Interpolation de Catmull-Rom $(a, b) = (0.5, 0)$

Le noyau Catmull-Rom correspond aux paramètres $a = 0,5$ et $b = 0$, tels qu'exprimés dans l'équation (1.6). L'interpolation de Catmull-Rom donne des résultats nettement meilleurs que l'interpolation cubique standard pour une complexité de calcul un peu plus élevée [36, 45, 37]. Cette interpolation nécessite également quatre pixels adjacents. Son noyau est représenté dans la figure 1.9.

$$\beta(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}|x|^2 + 1 & 0 \leq |x| < 1 \\ -\frac{1}{2}|x|^3 + \frac{5}{2}|x|^2 - 4|x| + 2 & 1 \leq |x| < 2 \\ 0 & others \end{cases} \tag{1.6}$$

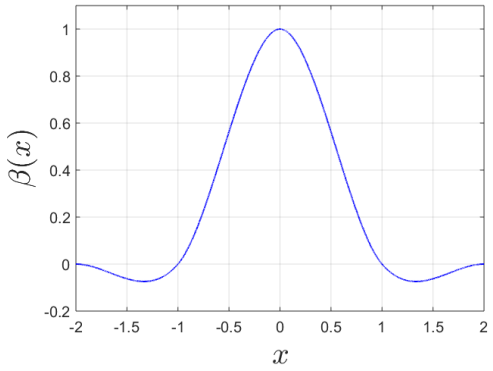


FIGURE 1.9 – Le noyau d'interpolation de Catmull-Rom.

Comme illustré dans la figure 1.13b, les résultats obtenus par l'interpolation de Catmull-Rom sont clairement supérieures à ceux obtenus par interpolation cubique-standard. Ils représentent moins de dépassements dans les zones à haut gradient et un signal beaucoup plus lisse.

1.2.3.3 B-spline $(a, b) = (0, 1)$

Le noyau B-spline est généré à partir des paramètres $a = 0$ et $b = 1$, comme décrit dans l'équation 1.7 et la figure 1.10. Il nécessite également quatre pixels adjacents. L'interpolation B-Spline présente une complexité de calcul supérieure à celles des interpolations : cubique standard et Catmull-Rom. En général, ce noyau d'interpolation permet de garder les détails de l'image et aussi de réduire le niveau de bruit. Par contre, il approxime mal la fonction $sinc$ à la position nulle '0' [36, 37].

$$\beta(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 - \frac{2}{3} & 0 \leq |x| < 1 \\ -\frac{1}{6}|x|^3 + |x|^2 - 2|x| + \frac{4}{3} & 1 \leq |x| < 2 \\ 0 & others \end{cases} \tag{1.7}$$

L'interpolation B-spline provoque un pur effet de lissage similaire à un filtre de lissage gaussien. Contrairement à toutes les méthodes d'interpolation décrites précédemment, la fonction reconstruite ne passe pas par tous les points d'échantillonnage discrets (voir la figure 1.13c).

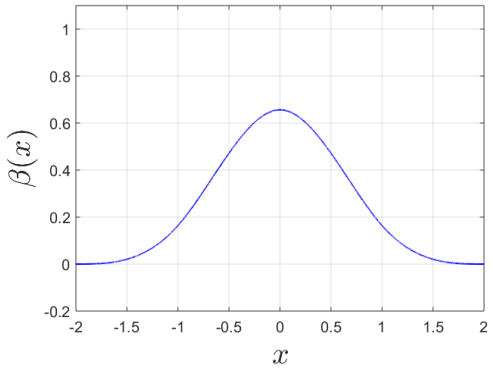


FIGURE 1.10 – Le noyau d'interpolation B-spline.

1.2.3.4 Mitchell-Netravali $(a, b) = (1/3, 1/3)$

La conception d'un noyau d'interpolation optimal est toujours un compromis entre netteté et bonne préservation des détails. L'interpolation Catmull-Rom, par exemple, met l'accent sur une grande netteté, tandis que l'interpolation cubique B-spline est floue mais ne crée pas de sur-oscillation.

Mitchell et Netravali ont proposé un noyau d'interpolation sous forme d'une somme pondérée de Catmull-Rom et de B-spline, avec les paramètres $a = 1/3$ et $b = 1/3$, comme décrit dans l'équation (1.8) et la figure 1.11 [36, 37]. Ce noyau de Mitchell et Netravali est plus complexe par rapport aux autres noyaux présentés ci-dessus.

$$\beta(x) = \begin{cases} \frac{7}{6}|x|^3 - 2|x|^2 - \frac{8}{9} & 0 \leq |x| < 1 \\ -\frac{7}{18}|x|^3 + 2|x|^2 - \frac{10}{3}|x| + \frac{16}{9} & 1 \leq |x| < 2 \\ 0 & others \end{cases} \tag{1.8}$$

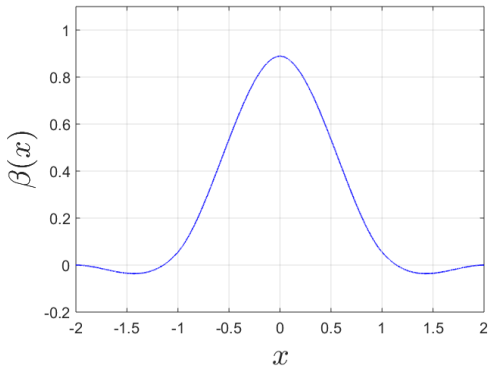


FIGURE 1.11 – Le noyau d'interpolation de Mitchell-Netravali.

Comme nous pouvons le voir dans la figure 1.13d, cette méthode d'interpolation présente un léger dépassement, une grande netteté des contours et une bonne continuité du signal dans les zones lisses. De plus, la courbe reconstruite est plus proche des points d'échantillonnage originaux que la méthode B-spline.

1.2.4 Interpolation basée sur Sinc pondérée par une fenêtre : Lanczos

Contrairement aux méthodes décrites dans le paragraphe précédent, l'interpolation de Lanczos n'utilise pas une approximation polynomiale de la fonction $Sinc$, mais elle utilise la fonction $Sinc$ elle-même pondérée par une fenêtre $\phi_n(x)$; c'est-à-dire un noyau d'interpolation de la forme :

$$\beta_n(x) = \phi_n(x) \times Sinc(x) \tag{1.9}$$

où n∈N indique l'ordre du filtre, et la fenêtre $\phi_n(x)$ est définie comme suit :

$$\phi_n(x) = \begin{cases} 1 & |x| = 0 \\ \frac{Sin(\frac{\pi x}{n})}{\pi \frac{x}{n}} & 0 < |x| < n \\ 0 & others \end{cases} \tag{1.10}$$

Les filtres de Lanczos d'ordre n=2,3 sont les plus couramment utilisés dans le traitement d'images. Dans ce travail, nous nous intéressons uniquement au noyau de 4 pixels voisins afin d'avoir une comparaison équitable avec les autres méthodes décrites ci-dessus, ce qui signifie que seul Lanczos d'ordre 2 (équation 1.11 et figure 1.12) est pris en compte. En outre, Lanczos3 nécessite 6 pixels voisins. Cela conduit à une complexité de calcul beaucoup plus élevée.

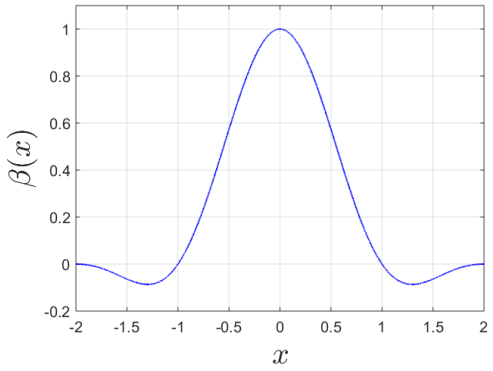


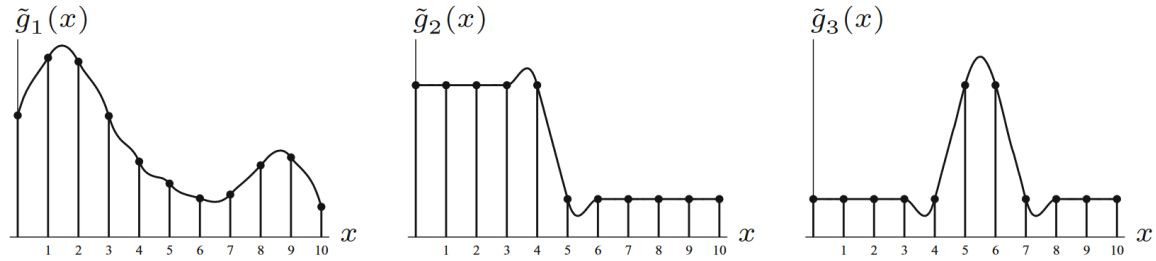
FIGURE 1.12 – Le noyau d'interpolation de Lanczos2.

$$\beta_2(x) = \begin{cases} 1 & |x| = 0 \\ 2\frac{sin(\frac{\pi x}{2})sin(\pi x)}{\pi^2 x^2} & 0 < |x| < 2 \\ 0 & others \end{cases} \tag{1.11}$$

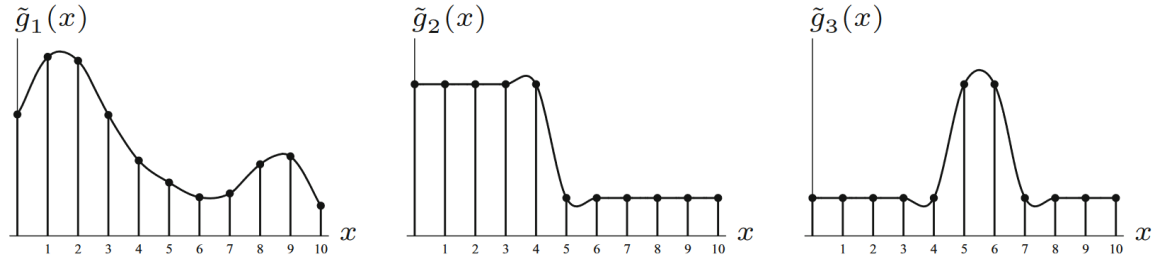
La figure 1.13e montre que les résultats de l'interpolation de Lanczos2 sont similaires à ceux de l'interpolation de Catmull-Rom.

1.2.5 Bilan

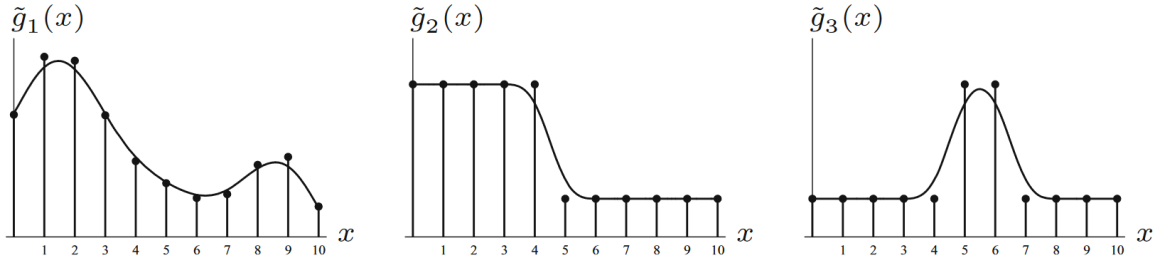
En résumé, bien que les interpolateurs de Lanczos aient connu un intérêt et une popularité ces dernières années, ils ne semblent pas offrir beaucoup d'avantages par rapport aux interpo-



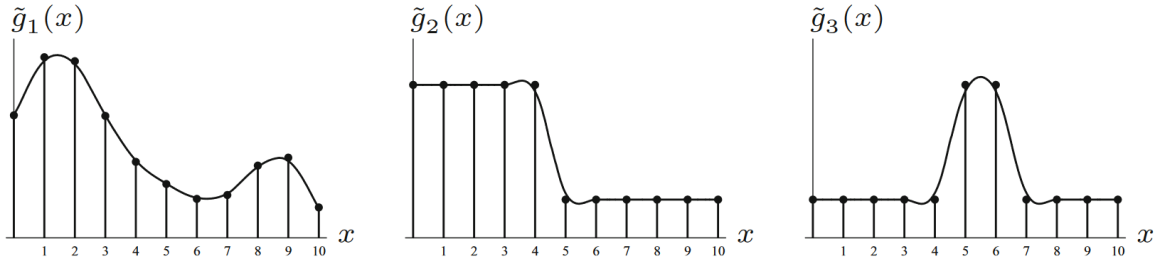
(a) Cubique standard



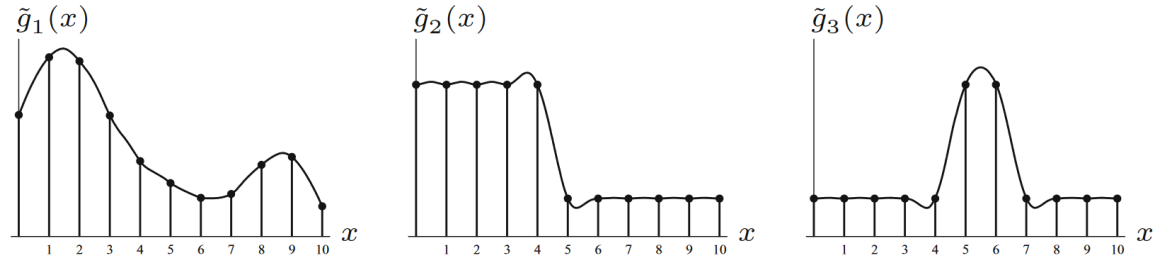
(b) Catmull-Rom



(c) B-spline



(d) Mitchell-Netraval



(e) Lanczos2

FIGURE 1.13 – Interpolation appliquée à trois types de signaux. g_1 signal continue à bande fréquentielle limitée, g_1 signal échelon, g_3 signal impulsion [36].

lations : cubique standard, Catmull-Rom, B-spline, et Mitchell-Netravali. Alors que celles-ci sont basées sur des fonctions polynomiales calculables efficacement, l'interpolation de Lanczos nécessite des fonctions trigonométriques dont le calcul est relativement coûteux, à moins qu'une tabulation ne soit utilisée.

Compte tenu de la complexité du calcul et la qualité d'interpolation, on peut considérer que l'interpolation de Catmull-Rom est la technique offrant le meilleur compromis entre ces deux facteurs. Il est important de savoir que l'interpolation de Catmull-Rom est parfois appelée interpolation cubique ou bi-cubique (dans le cas d'une interpolation 2D).

1.2.6 Interpolation bidimensionnelle

Dans les paragraphes précédents, nous avons décrit l'interpolation unidimensionnelle à partir d'échantillons discrets. Pour le traitement des images numériques, l'interpolation unidimensionnelle est transformée en bidimensionnelle en appliquant cette interpolation unidimensionnelle sur deux axes ou directions au lieu d'un(e) seul(e). La formule d'interpolation bidimensionnelle est donnée par :

$$g(x,y) = \sum_{i=0}^{3} \left(\sum_{j=0}^{3} A_{ij} \beta(y - y_j) \right) \beta(x - x_i) \tag{1.12}$$

où A_{ij}, $i, j = 0...3$, sont l'ensemble des pixels voisins, β est le noyau d'interpolation et g est la fonction d'interpolation.

L'équation précédente peut être réécrite avec $u_i = x - x_i$ et $v_i = y - y_i$:

$$\begin{aligned} g(x,y) = &(\beta(v_0)A_{00} + \beta(v_1)A_{10} + \beta(v_2)A_{20} + \beta(v_3)A_{30})\beta(u_0) \\ &+(\beta(v_0)A_{01} + \beta(v_1)A_{11} + \beta(v_2)A_{21} + \beta(v_3)A_{31})\beta(u_1) \\ &+(\beta(v_0)A_{02} + \beta(v_1)A_{12} + \beta(v_2)A_{22} + \beta(v_3)A_{32})\beta(u_2) \\ &+\beta(v_0)A_{03} + \beta(v_1)A_{13} + \beta(v_2)A_{23} + \beta(v_3)A_{33})\beta(u_3) \end{aligned} \tag{1.13}$$

L'interpolation bidimensionnelle repose sur 16 pixels voisins au lieu de 4 dans le cas unidimensionnel. Deux types d'interpolations unidimensionnelles sont effectuées dans les directions : horizontale et verticale, comme le montre la figure 1.14. On peut voir que l'interpolation verticale est effectuée quatre fois pour obtenir les quatre points intermédiaires (q_0, q_1, q_2, q_3), alors que l'interpolation horizontale basée sur ces points intermédiaires est effectuée une fois. Il convient de noter que l'ordre entre les deux directions (verticale et horizontale) peut être modifié sans incidence sur la valeur obtenue à la fin de l'interpolation.

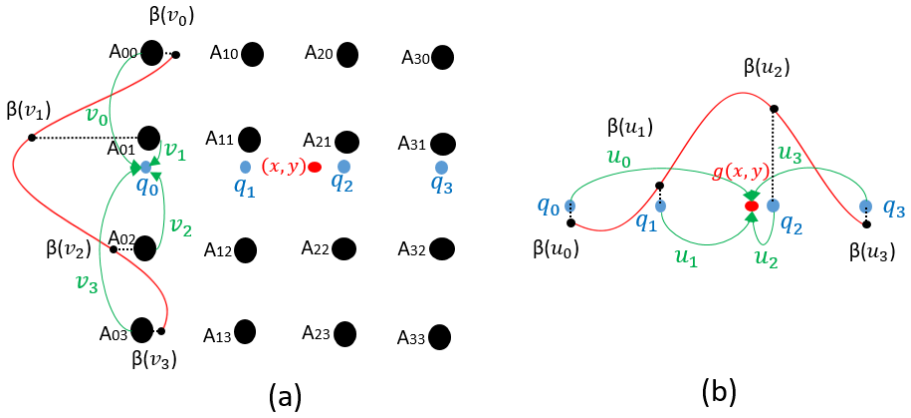
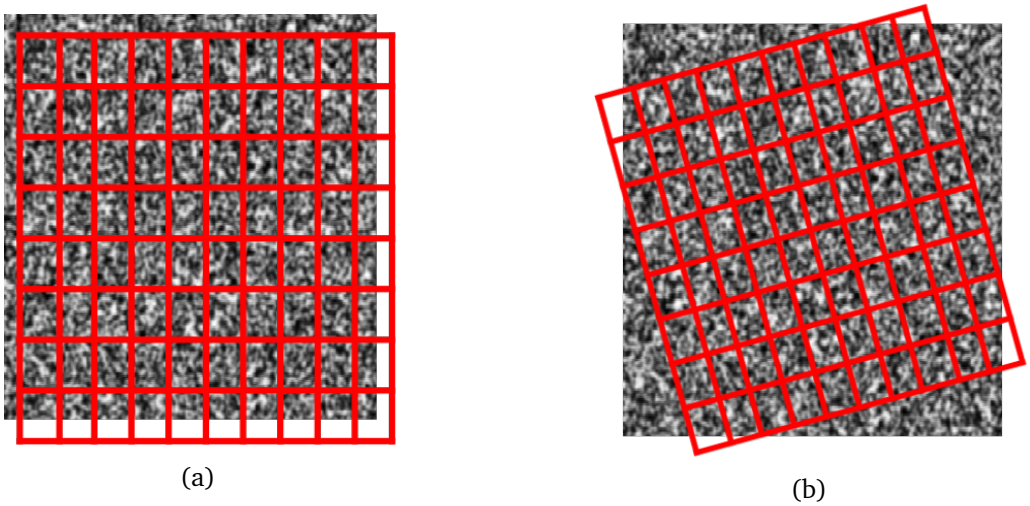


FIGURE 1.14 – Interpolation bidimensionnelle, (a) interpolation unidimensionnelle verticale et (b) interpolation unidimensionnelle horizontale.

Selon l'application, on peut définir deux catégories d'interpolations. La première catégorie est plus complexe et permet d'interpoler à différents endroits hétérogènes comme le montre la figure 1.15b. Cette dernière peut être utilisée par toutes les applications. Elle est spécialement requise par les algorithmes d'estimation de déplacements et de déformations. La seconde est un cas particulier de la première. Elle est utilisée par des applications nécessitant une interpolation à des endroits homogènes (voir la figure 1.15a) comme dans le cas de la super résolution. Dans cette catégorie appelée interpolation homogène, les coefficients d'interpolation sont calculés une fois puis ils sont réutilisés à chaque fois pour tous les pixels de l'image. Cela signifie que l'architecture peut être simplifiée. Cependant, dans le cas d'une interpolation hétérogène, les coefficients doivent être recalculés à chaque fois que l'interpolation est effectuée à chaque endroit. Dans ce travail, nous nous intéressons à l'interpolation hétérogène.



(a) (b)

FIGURE 1.15 – (a) Interpolations homogènes et (b) interpolations hétérogènes, les points rouges représentent les points d'interpolation.

Comme dans cette thèse nous nous plaçons dans un contexte métrologique, il est primordial de détailler les critères de performances métrologiques utilisés afin d'évaluer et de connaître les limites des systèmes de mesures. Cela fait l'objet du paragraphe suivant.

1.3 Métrologie par vision

La métrologie est la science de la mesure, elle embrasse à la fois les déterminations expérimentales et théoriques à tous les niveaux d'incertitude et dans tous les domaines des sciences et de la technologie (définition du Bureau international des poids et mesures). Elle s'intéresse à caractériser les incertitudes de mesure de toutes instrumentations dans tous les domaines de la science. Elle est en évolution, avec de nouvelles techniques et de nouveaux instruments afin de réduire l'incertitude de mesure des résultats. La plupart des systèmes de vision considérés comme des systèmes de métrologie se concentrent sur des mesures à haute résolution. En général, leurs performances finales dépendent du choix de la bonne combinaison entre matériel (caméra, éclairage, ...) et algorithme. Les applications et systèmes de métrologie doivent connaître les limites de leur capacité de mesure pour décider si un système ou une application peut être considéré comme suffisamment précis(e) pour la tâche à accomplir. Pour cela, sources d'erreurs et critères de performances métrologiques sont abordées ci-dessous.

1.3.1 Sources d'erreurs

On peut distinguer deux types de sources d'erreurs : intrinsèques et extrinsèques. Les sources intrinsèques sont liées principalement aux paramètres et aux hypothèses utilisées dans la technique de traitement d'images, à savoir la technique d'interpolation. Tandis que les sources extrinsèques sont liées aux conditions expérimentales et au bruit de capteur. L'effet du bruit sur l'exactitude des mesures est considéré parmi les préoccupations majeures dans le traitement d'images sous-pixellique. Pour cela, il est détaillé ci-dessous.

1.3.1.1 Généralités sur le bruit de capteur

La signification originale du terme "bruit" est "signal indésirable". Par analogie, le bruit de capteur est une variation indésirable de la luminosité ou des informations des pixels qui ajoute des informations parasites dans les images capturées. Il est toujours présent dans les images numériques lors des étapes d'acquisition, de codage, de transmission et de traitement, et entraîne une dégradation de la qualité de l'image.

Le bruit des images numériques peut provenir de différentes sources. Le processus d'acquisition, conversion des signaux optiques en signaux électriques puis en signaux numériques sont des processus dans lesquels le bruit est introduit dans les images numériques. Chaque étape du processus de conversion subit des fluctuations, causées par des phénomènes naturels, et chacune de ces étapes ajoute une valeur aléatoire à l'intensité résultante d'un pixel donné.

Différentes distributions statistiques de bruit peuvent être identifiées en fonction de l'origine du bruit. En fonction de celles-ci, les bruits les plus courants sont les suivants :

— **Bruit Gaussien :** appelé aussi bruit thermique est uniformément réparti sur le signal [46, 47, 48, 49]. Cela signifie que chaque pixel de l'image bruitée est la somme de la valeur réelle du pixel et d'une valeur aléatoire de bruit à distribution gaussienne. Un cas particulier est le bruit blanc gaussien, dans lequel le bruit est indépendant de l'intensité de la valeur du pixel à chaque point. Le bruit blanc gaussien est une bonne approximation de nombreuses situations réelles.

— **Bruit impulsionnel :** s'agit de pics d'intensité. On le trouve aussi sous le nom "bruit Poivre et Sel". Il est toujours indépendant et non corrélé aux pixels de l'image. Il est distribué de façon aléatoire sur l'image. Dans une image bruitée, seuls certains pixels sont impactés. Ces pixels bruités ne peuvent prendre que les valeurs maximales et minimales de la gamme dynamique. Pour une image de 8 bits, la valeur typique pour le bruit du poivre est 0 et pour le bruit du sel est 255 [46, 47, 48].

— **Bruit de Poisson :** également connu sous le nom de bruit de photons, a une distribution de Poisson. L'effet de ce bruit augmente avec l'amplitude moyenne du courant ou l'intensité de la lumière. Son amplitude dépend du signal et constitue la source dominante de bruit d'image, sauf dans des conditions de faible luminosité [46, 47, 48].

— **Bruit uniforme (de quantification) :** est un bruit uniforme causé par la quantification des pixels de l'image à un certain nombre de niveaux distincts [46, 47, 48]. Il a une distribution approximativement uniforme, et peut généralement être ignoré, car le rapport signal/bruit total d'un système complet est généralement dominé par les bruits les plus importants.

La figure 1.16 montre les différents types de bruit ajoutés à une image en utilisant Matlab.

1.3.1.2 Du photon à la valeur numérique

Deux technologies sont utilisées dans les capteurs des caméras : les capteurs à couplage de charge (CCD) et les semi-conducteurs à oxyde métallique complémentaire (CMOS). En bref, les CCD et les CMOS transforment les photons de la lumière entrante en valeurs de tension. Plus précisément, ces capteurs sont des circuits intégrés à base de silicium comprenant une matrice dense de photodiodes qui convertissent d'abord les photons lumineux en charges électroniques.

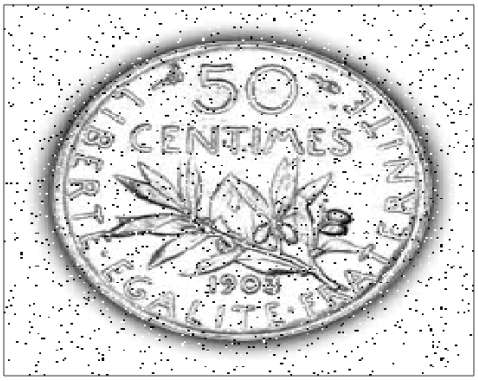
Les capteurs CCD transfèrent chaque charge jusqu'à atteindre un amplificateur de sortie où chacune des charges est convertie en une valeur de tension. Avec la technologie CMOS, les pixels sont dotés d'un circuit électronique de conversion qui effectue la conversion de la charge en tension à chaque emplacement. Ce circuit augmente le bruit et génère des sources de bruit supplémentaires par rapport aux CCD [50, 51, 52].

Le processus d'acquisition passe par trois phases de conversion : Photon-Charge, Charge-Tension, et Tension-Valeur numérique. Ces trois phases sont détaillées ci-dessous.

— **Photon-Charge :** la capacité d'un semi-conducteur à produire des électrons à partir de

(a) Image non bruitée.

(b) Bruit Gaussien, moyenne = 0 et variance = 0.01.



(c) Bruit impulsionnel, densité = 0.05.

(d) Bruit de Poisson, moyenne = 5.5.

FIGURE 1.16 – Images bruitées en utilisant les diffèrent types de bruit.

photons incidents est appelée efficacité quantique (QE). L'efficacité quantique peut être fournie par le fabricant du capteur sous forme de sensibilité spectrale en fonction de la longueur d'onde, ou mesurée empiriquement. Le processus de capture des photons présente une incertitude qui résulte de fluctuations aléatoires lorsque les photons sont collectés par la photodiode. Cette incertitude est appelée “photon shot noise". Elle est décrite par un bruit de Poisson [53, 50]. De plus, une partie des électrons accumulés ne proviennent pas de la photodiode mais de la génération thermique. Ces électrons sont présents et détectés même en l'absence de lumière. Ce bruit peut être modélisé par un bruit Gaussien. Un mauvais fonctionnement dans les pixels de la caméra peut aussi être modélisé par un bruit Poivre et Sel.

— **Charge-Tension :** la charge collectée dans chaque pixel d'un réseau de capteurs est convertie en tension. Le circuit de détection est partagé pour tous les pixels dans le cas d'un CCD. Dans le cas des capteurs CMOS, le circuit de détection est situé à l'intérieur de chaque pixel. La conversion de la charge en tension n'est pas parfaite dans un photo-détecteur. Un bruit se produit lorsque la charge est convertie en tension. Il ajoute une

incertitude au signal mesuré. Ce bruit a principalement 3 sources : 1) variations du courant d'obscurité dans la photodiode, ce qui engendre un offset différent pour chaque pixel, 2) la variation de la géométrie des pixels pendant la fabrication du capteur qui influence le gain de conversion, 3) et la variation de la taille des transistors constituant ces amplificateurs affectant l'offset du signal de sortie. La totalité de toutes ces variations conduit à ce que l'on appelle "Fixed-pattern noise".

— **Tension-valeur numérique :** un convertisseur analogique-numérique (ADC) transforme le signal analogique de tension en signal discret. La résolution de l'ADC indique le nombre de valeurs discrètes qui peuvent être produites sur la plage de valeurs analogiques. Une dernière source de bruit dans le processus d'acquisition se produit lors de cette conversion de tension analogique en valeur numérique quantifiée. Il est appelé bruit de quantification [53]. Ce bruit est généralement négligeable par rapport aux autres bruits.

1.3.2 Critères de performances métrologiques

Ce paragraphe présente les différents critères de performances métrologiques utilisées afin d'évaluer l'exactitude des résultats.

1.3.2.1 Résolution de mesure

Une préoccupation importante en métrologie est la résolution de mesure. Elle est la plus petite variation de la grandeur mesurée qui produit une variation perceptible de l'indication correspondante [54]. Elle traduit la capacité du système de mesure à détecter et à indiquer fidèlement de petits changements dans le résultat de la mesure, comme illustré dans la figure 1.17.

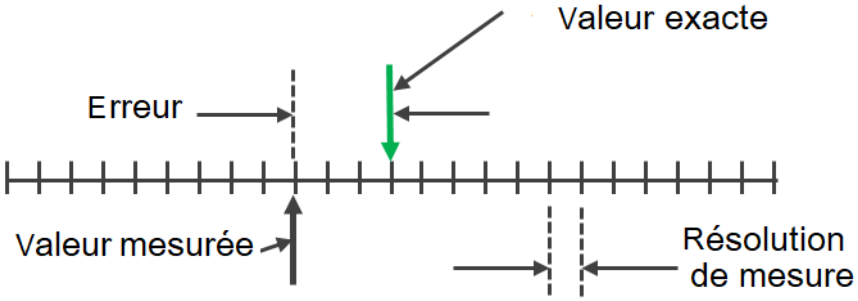


FIGURE 1.17 – Résolution de mesure.

La résolution de mesure est liée principalement à deux facteurs. Le premier est la méthode ou la technique de mesure utilisée. Le second est le niveau de bruit, plus le bruit est présent d'une façon significative plus la résolution est impactée. Le niveau de bruit quantifié par son écart-type σ_u.

1.3.2.2 Erreur de mesure

L'erreur de mesure est la différence entre la valeur réelle et la valeur expérimentale fournie par une mesure. On peut distinguer deux types d'erreurs : les erreurs aléatoires et les erreurs systématiques [55, 56].

Les erreurs aléatoires se produisent d'une manière irrégulière (voir la figure 1.18) dues aux fluctuations et aux changements imprévisibles des conditions expérimentales. Elles peuvent être positives ou négatives. Ce type d'erreur peut être réduit par effet de moyenne sur des mesures répétées [56].

L'erreur systématique est un biais constant qui s'introduit dans tous les résultats de mesure en une seule direction. Elle entraîne un décalage de la quantité mesurée par rapport à la valeur réelle, comme le montre la figure 1.18. Cette erreur est due à une conception ou un étalonnage imparfait de l'instrument ou/et une imperfection dans la technique expérimentale et la procédure utilisée. Contrairement aux erreurs aléatoires, elles ne peuvent pas être réduites par effet de moyenne [56].

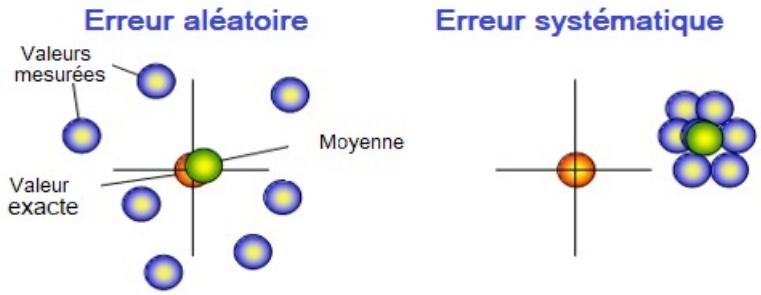


FIGURE 1.18 – Erreurs systématiques et aléatoires.

1.3.2.3 Cas particulier de la mesure de champs : résolution spatiale

Les méthodes de mesure optique ont un aspect particulier pour la métrologie, car elles fournissent un champ spatial au lieu d'un seul scalaire. La résolution spatiale d est définie par la distance la plus courte entre deux mesures fidèles et indépendantes dans l'espace [57]. Elle dépend de nombreux facteurs, notamment de tous les post-traitements numériques qui peuvent avoir lieu après l'obtention des mesures brutes, par exemple, pour réduire le bruit. En général, des images synthétiques sont utilisées pour déterminer la résolution spatiale associée à la technique utilisée. Plus de détails sont donnés dans le paragraphe 1.4.3.3.

1.4 Le cas particulier de mesure métrologique en photomécanique

Dans ce paragraphe, nous commençons par rappeler le principe de la CIN. Ensuite, nous présentons ces enjeux, à savoir les performances métrologiques et la complexité de calcul.

1.4.1 CIN

Apparue initialement dans les années 1980, la technique de CIN a connu un développement continu au cours des trois dernières décennies en raison de son côté pratique. Aujourd'hui, cette technique de traitement d'images est la plus utilisée pour les mesures dans le domaine de la photomécanique [58, 6]. Cette technique est aussi largement utilisée dans un large éventail d'applications d'ingénierie, car elle fournit un bon compromis entre facilité d'emploi et performances métrologiques [59].

La CIN est une méthode de mesure optique sans contact de champ de déplacements (figure 1.19). Pour une utilisation optimale de la CIN, la surface de l'éprouvette doit être recouverte d'un motif contrasté pour résoudre le problème d'unicité de pixels. La texture naturelle des surfaces peut suffire dans certains cas. Différents types de mouchetis sont illustrés dans la figure 1.20.

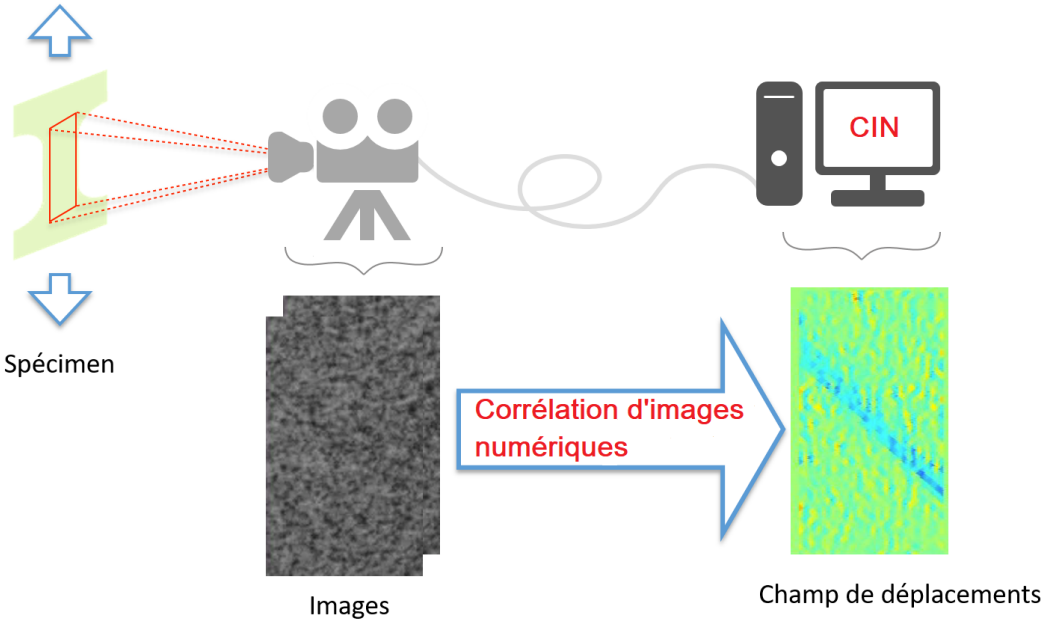
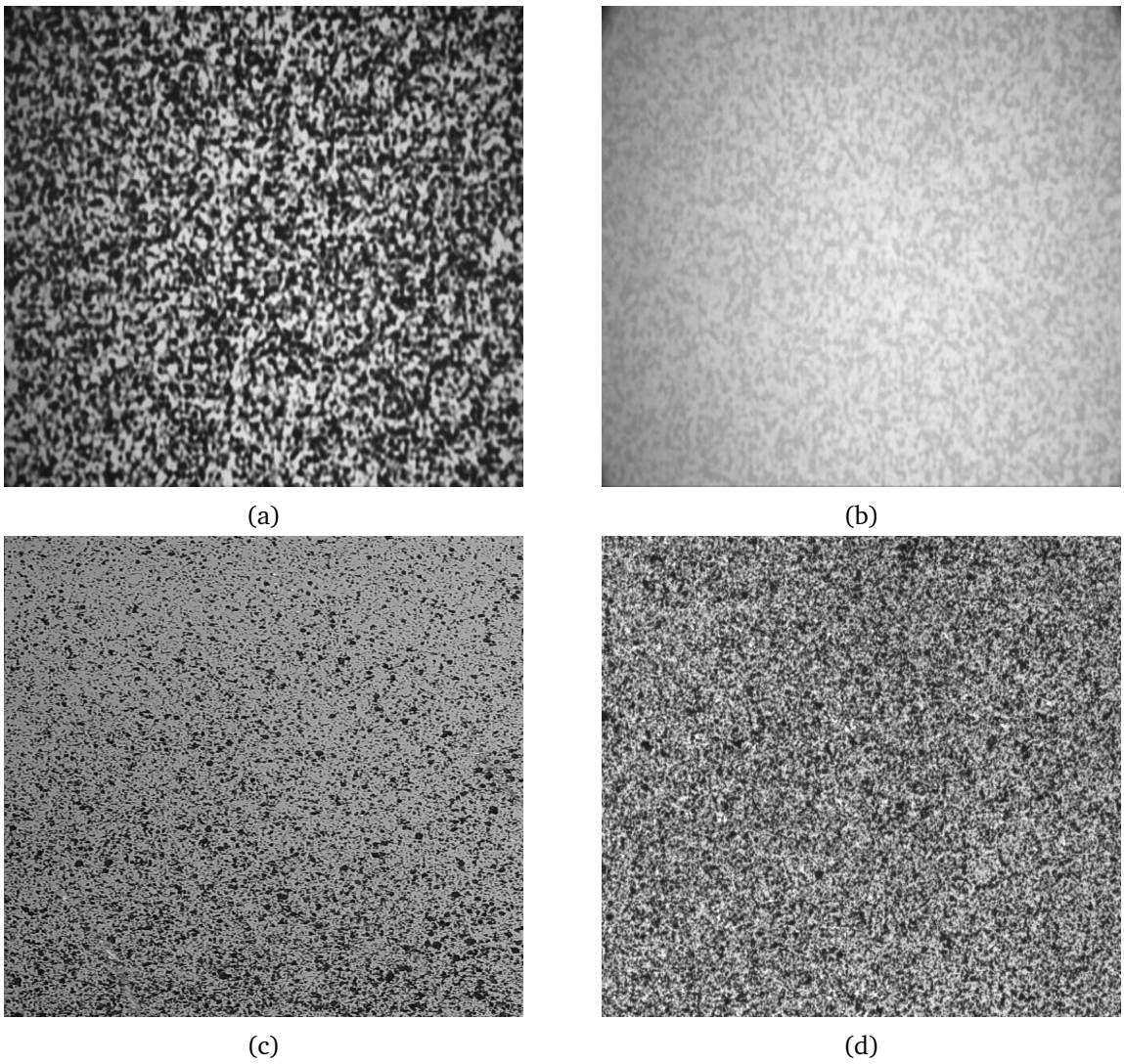


FIGURE 1.19 – Système de mesure par CIN.

La CIN permet d'estimer le champ de déplacement qui se produit entre deux images de la même surface de l'éprouvette, à savoir, l'image de référence (f) et l'image déformée (g). L'hypothèse principale de cette méthode est la conservation de l'intensité sur une région d'intérêt (RdI) pendant l'essai. En d'autres termes, les valeurs de niveau de gris sont supposées être décalées de leur emplacement initial dans l'image de référence (f) à leur emplacement actualisé dans l'image déformée (g) :



(a) (b)

(c) (d)

FIGURE 1.20 – Exemple d'une surface d'éprouvette recouverte par des mouchetis avec différents paramètres [60, 61]

$$\forall x_p \in RdI, \quad f(x_p) \approx g(x_p + U(x_p)) \tag{1.14}$$

Où U est le déplacement des points de l'éprouvette (pixels).

Le cœur de la CIN est de trouver la meilleure approximation du déplacement $U^{optimal}$. Pour cela, plusieurs critères peuvent être utilisés pour définir le résidu et évaluer le degré de similarité entre images déformée et référence (équation 1.15). Le choix de l'un de ces critères peut aider à la convergence de la CIN mais n'a aucun impact pratique sur la solution [62]. Dans ce travail, nous choisissons un critère de Somme des Carrées des Différences (SSD).

$$U(x_p)^{optimal} = \underset{U}{Argmin}\Big(\sum_{x_p \in RdI} \big(f(x_p) - g(x_p + U(x_p))\big)^2 \Big) \tag{1.15}$$

En pratique, le déplacement $U(x_p)$ est approché dans un espace cinématique sur une zone d'intérêt (ZdI). Cet espace est généralement défini comme un espace vectoriel de dimension N.

Il est construit à partir d'une base de fonctions de forme $(\phi_i)_{1\leq i\leq N}$. Tout déplacement est défini comme une combinaison $(\lambda_i)_{1\leq i\leq N}$ des N fonctions de forme :

$$\forall x_p \in ZdI, \quad U(x_p) = \sum_{i=1}^{N} \lambda_i \phi_i(x_p) \tag{1.16}$$

Où N est le nombre de degrés de liberté (ddl) et des fonctions de forme. λ_i est la valeur de ddl associée à chaque fonction de forme ϕ_i.

La figure 1.21 illustre les différents types de fonctions de forme. Les fonctions de forme d'ordre zéro sont utilisées dans le cas d'une translation pure. Celles du premier ordre sont utiles en cas d'une déformation linéaire (comme un étirement linéaire, une rotation au premier ordre et une distorsion linéaire). En cas de déformations plus complexes (comme un étirement quadratique et une distorsion non-linéaire), les fonctions de forme du second ordre sont utilisées.

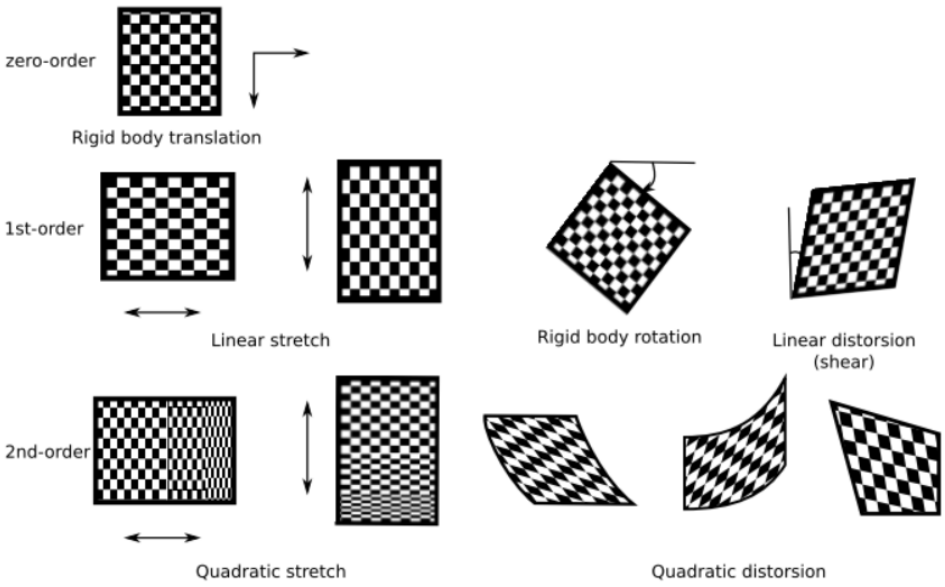


FIGURE 1.21 – Les types de déplacements à approximer en fonction de l'ordre des fonctions de forme [6].

En fonction du choix de la ZdI on peut distinguer deux versions de CIN, la locale et la globale. La version globale repose sur une cinématique avancée définie sur l'ensemble de la RdI (ZdI=RdI). La version locale de CIN repose sur une définition locale du domaine ZdI, appelée imagette, associée à une définition cinématique (ou fonctions de forme locales) de complexité relativement faible. Le calcul est effectué pour multiple imagettes afin de couvrir l'ensemble de la RdI. Souvent ces imagettes sont choisies avec un pas d'espacement pour réduire le coût de calcul [63]. De nos jours, la plupart des systèmes CIN commerciaux utilisent la CIN locale [6]. Dans ce travail nous nous concentrons aussi sur la CIN locale.

Le problème de minimisation décrit dans l'équation 1.15 peut donc être reformulé avec le vecteur $\lambda = (\lambda_i)_{1\leq i\leq N}$ comme suit :

$$\lambda^{optimal} = \underset{\lambda_i \in \mathbb{R}^N}{Argmin}\Big(\sum_{x_p \in RdI} \big(f(x_p) - g(x_p + \sum_{i=1}^{N} \lambda_i \phi_i(x_p))\big)^2 \Big) \tag{1.17}$$

Ce problème de minimisation est résolu itérativement, en mettant à jour λ :

$$\lambda^{it+1} = \lambda^{it} + M^{-1} \times L \times r(\lambda^{it}) \tag{1.18}$$

Où,

— La matrice L est la projection des fonctions de forme sur le gradient de l'image de référence,

$$L = \nabla f(x).\phi(x) \tag{1.19}$$

— La matrice M est l'opérateur tangent de CIN,

$$M = L \times L^T \tag{1.20}$$

— et le vecteur $r(\lambda^{it})$ est le résidu,

$$r(\lambda^{it}) = f(x_p) - g(x_p + \sum_{i=1}^{N} \lambda_i^{it} \phi_i(x_p)) \tag{1.21}$$

Dans ce processus itératif de Newton-Raphson, on interpole l'imagette et on recalcule le résidu jusqu'à ce que la convergence souhaitée soit atteinte [63]. On obtient ainsi des déplacements avec une performance SP. Une étude approfondie sur la CIN est fournie dans la référence [64].

Dans ce qui suit, nous présentons les enjeux de la CIN. Bruit de capteur, critères de performances métrologiques et temps de calcul sont abordées ci-dessous.

1.4.2 Bruit de capteur

En raison de leur qualité d'image très homogène avec un faible bruit, les capteurs CCD présentent des avantages pour les applications médicales et scientifiques, notamment pour celles à haute résolution [65, 52].

Les systèmes de métrologie par vision nécessitent des matériels d'acquisition d'images très fiables, qui ont souvent un haut rapport signal-bruit (SNR) important. Dans le cas de la CIN, on considère l'utilisation d'un capteur CCD à haut SNR pour réduire l'effet du bruit impactant l'image sur les mesures. Dans la plupart des études de l'effet de bruit sur les mesures par CIN estime le bruit de la caméra par un bruit appelé hétéroscédastique [66, 57, 67]. Ce type de bruit estime que la variance v du bruit du capteur de la caméra est une fonction affine de la luminosité s, donc $v = a \times s + b$, où a et b sont deux constantes caractéristiques qui dépendent du capteur.

Dans le domaine de traitement d'images numériques, des techniques de débruitage ont été proposées dans la littérature afin d'améliorer la qualité des images numériques et de réduire l'effet de bruit. Le débruitage d'image est un processus qui consiste à restaurer une image en limitant les effets du bruit. De nombreux travaux proposent des techniques de débruitage basées sur des traitements conventionnels comme le filtrage Gaussien, le filtrage moyen, le filtrage de Wiener et le filtrage médian [68, 69, 70, 71, 72, 73, 74]. Récemment, des méthodes basées sur l'apprentissage profond ont été proposées [75, 76, 77, 78, 79]. Ces méthodes ont considérablement amélioré les performances de débruitage par rapport aux méthodes de débruitage conventionnelles basées sur le filtrage. Ces techniques peuvent être utilisées dans le cas de la CIN pour améliorer la qualité des images avant de les traiter, mais cela introduira un temps de traitement supplémentaire et peut être un biais de mesure.

Il est possible de minimiser l'effet du bruit sur les résultats de la CIN en faisant attention aux : paramètres de configuration matérielle du système de test (caméra, luminosité) et au choix des paramètres de la CIN. Généralement, des sources de lumière sont utilisées afin d'avoir la bonne luminosité, et une caméra CCD avec un bon SNR est choisie pour réduire le niveau de bruit. De plus, on peut facilement modifier et utiliser les bons paramètres de la CIN. Plus la taille de l'imagette est grande moins la CIN est sensible au bruit. Par contre, plus l'ordre des fonctions de forme est élevé, plus la DIC est impactée par le bruit [80].

Le paragraphe suivant présente les différents critères de performances métrologiques utilisées afin d'évaluer l'exactitude des résultats de la CIN.

1.4.3 Performances métrologiques

Les utilisateurs de la méthode de CIN[2] ont soulevé des besoins pour évaluer les effets de divers facteurs, tels que le bruit de l'image causé par le capteur de la caméra, le biais engendré, la résolution spatiale, et la structure des motifs de moucheté sur ses performances métrologiques. De plus, le besoin de réduire le temps de calcul est une préoccupation majeure dans ce travail.

1.4.3.1 Résolution de mesure

La résolution de mesure est liée principalement à deux facteurs. Le premier est la méthode ou la technique de mesure utilisée. Le second est le niveau de bruit, plus le bruit est présent d'une façon significative plus la résolution est impactée. Le niveau de bruit quantifié par son écart-type σ_u. Parmi les avantages de la CIN qu'elle fournit une haute résolution de déplacement [59].

1.4.3.2 Biais en CIN

Divers types de biais sont introduits pour caractériser une mesure de champs de déplacements par la méthode CIN, à savoir, l'algorithme d'interpolation, les fonctions de forme, et la

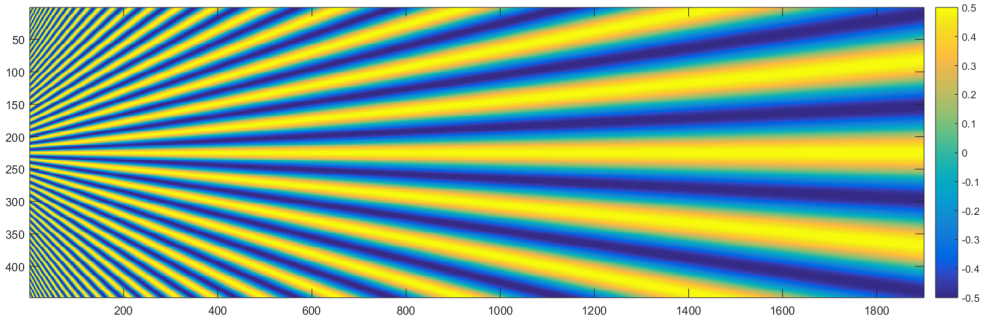
2. Il convient aussi de savoir qu'aujourd'hui la CIN n'a pas de norme.

structure des motifs de mouchetés.

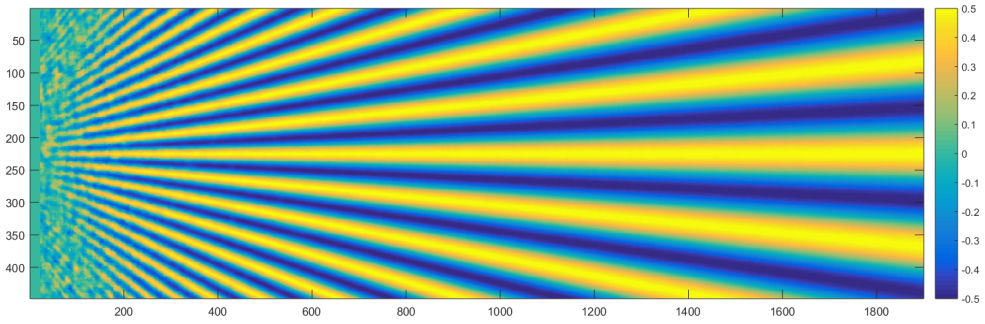
La technique d'interpolation est nécessaire pour atteindre une performance SP, et comme il n'existe pas d'interpolation idéale, un biais est généré par l'algorithme d'interpolation utilisé [67]. La qualité du motif observé est aussi un facteur important pour minimiser le biais d'estimation. Des études ont récemment montré qu'un biais lié au marquage appelé Pattern Induced Bias (PIB) en anglais, est introduit par le motif observé. Ce biais est dû principalement à la distribution du gradient d'image et à la différence entre le champ de déplacement réel et son approximation locale par des fonctions de forme [81, 82]. Des résultats sur l'effet de la PIB sont montrés dans le chapitre 5.

1.4.3.3 Résolution spatiale

Dans le cas de la mesure des déplacements avec la CIN, un champ de déplacement synthétique modélisé par une onde sinusoïdale, dont la période augmente linéairement en allant vers le côté droit de la carte (comme le montre la figure 1.22a), a été introduit récemment pour déformer des images de moucheté synthétique utilisé [57, 83, 61, 84, 60]. Dans ce cas, la résolution spatiale est estimée en considérant les déplacements le long de l'axe horizontal de symétrie. Le déplacement de référence est le même (ici, 0.5 pixel) au long de ce dernier.



(a) Champ de déplacement de référence.



(b) Champ de déplacement mesuré par la CIN, avec une imagette de taille $2M + 1 = 21$, et des fonctions de forme d'ordre 2.

FIGURE 1.22 – La résolution spatiale : on observe un biais d'estimation sous la forme d'un amortissement dans la partie gauche de (b) présentant des hautes fréquences spatiales.

Dans la figure 1.22b, on peut facilement évaluer l'effet de la fréquence spatiale sur le

champ de déplacement mesuré en observant l'atténuation de l'amplitude du déplacement mesuré lorsque l'on va vers les hautes fréquences. La résolution spatiale, est généralement définie comme l'inverse de la fréquence spatiale obtenue pour une atténuation donnée (généralement de 10%) de l'amplitude de déplacement de référence.

La résolution spatiale doit être aussi petite que possible pour refléter la capacité à distinguer des caractéristiques spatialement proches dans les cartes de déplacements et de déformations avec un léger biais. La résolution spatiale de la CIN peut être prédite théoriquement à partir de la fonction de transfert du filtre associé à cette technique (filtre de Savitzky-Golay) [81].

1.4.3.4 Bilan : indicateur de performance

Il est important de savoir aussi que, la résolution de mesure, biais, et résolution spatiale sont liés, pour une technique de mesure de champs de déplacements donnée. Plus la valeur de d est faible, plus le bruit dans la carte de déplacement est élevé [57], et vice versa. Une étude approfondie qui montre la performance métrologique de la CIN est fournie dans la référence [63].

Comme il est primordial de choisir des paramètres pertinents pour décrire les performances métrologiques des techniques de mesure, l'indicateur de performance métrologique α défini dans l'équation 1.22 est considéré comme un moyen pratique de comparer les techniques de mesure [63]. Il combine à la fois la résolution spatiale d et le niveau de bruit quantifié par son écart-type σ_u pour former un seul indicateur.

$$\alpha = \sigma_u \times d \tag{1.22}$$

Les valeurs de σ_u et d devraient idéalement être aussi petites que possible. Donc, plus la valeur α est faible, plus la technique est performante.

Après avoir présenté les différents critères de performances métrologiques, nous nous concentrons dans le paragraphe suivant sur le temps de calcul de la CIN.

1.4.4 Complexité de calcul

Afin d'accélérer le traitement d'images avec la CIN, il est primordial de commencer d'abord par l'identification du traitement le plus coûteux dans cette technique. On peut facilement voir dans l'équation 1.18 que les termes M^{-1} et $g(x_p + \sum_{i=1}^{N} \lambda_i^{it} \phi_i(x_p))$ sont les traitements les plus complexes à calculer en CIN. Le calcul du premier terme M^{-1} est simplifié en résolvant un système linéaire à la base de la méthode de Gauss-Jordan au lieu de calculer l'inversion de matrice, mais le second terme est plus compliqué à simplifier car il impacte les performances métrologiques de la CIN.

Une interpolation de l'ensemble des pixels de l'imagette est nécessaire dans le calculer du terme $g(x_p + \sum_{i=1}^{N} \lambda_i^{it} \phi_i(x_p))$ à chaque itération du processus de Newton-Raphson. De plus, ceci est répété généralement 7 fois pour atteindre la convergence et estimer le déplacement du pixel

au centre de l'imagette.

Pour obtenir le temps de calcul dédiés à chaque bloc de traitement de la CIN (montrés dans la figure 1.23), Nous avons codé la CIN en C, en utilisant une interpolation bi-cubique (de Catmull-Rom) et une imagette de taille $2M+1 = 17$. Ensuite, ce code de CIN a été exécuté dans la plateforme ARM Cortex-A9 (800 MHz dual core, 1GB SDRAM DDR3). D'après les résultats obtenus, nous pouvons facilement constater que l'interpolation est le bloc qui consomme le plus de temps de calcul, en l'occurrence 1,25ms pour chaque itération.

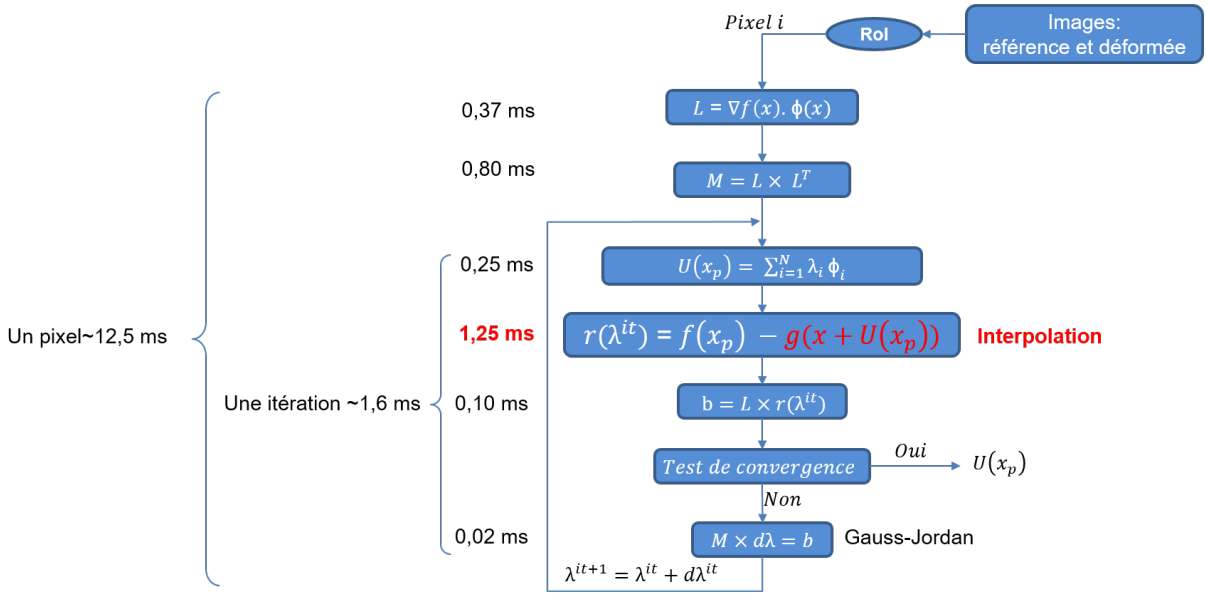


FIGURE 1.23 – Temps de calcul pour chaque bloc de l'algorithme CIN.

Comme déjà indiqué dans le paragraphe précédent, l'imperfection des méthodes d'interpolation génère un biais d'estimation. Par conséquent, l'impact du choix d'une méthode d'interpolation sur les résultats de la CIN peut être important en termes de performances métrologiques ou de temps de calcul. Plus l'interpolation est précise, plus les résultats sont précis, mais cela entraîne une augmentation du temps de calcul.

Dans le cas de la CIN, nous nous intéressons à l'interpolation hétérogène. Celle-ci est plus complexe et nécessite plus de temps de calcul. En effet, elle consiste à mettre en œuvre cinq interpolations 1D au lieu de deux interpolations 1D dans le cas homogène. Elle implique un plus grand nombre d'opérations d'addition et de multiplication par rapport au cas homogène. De plus, dans le cas de la CIN locale, pour chaque déplacement, une interpolation de l'ensemble des pixels de l'imagette est effectuée au moins 7 fois. Cela augmente encore la complexité de calcul.

Pour faire face à cette problématique et accélérer ce traitement d'interpolation, nous avons proposé d'implémenter ce bloc sur une plateforme matérielle. De plus, l'interpolation de Catmull-Rom est choisie car elle offre le meilleur compromis entre complexité de calcul (consommation des ressources matérielles) et performances des résultats (paragraphe 1.2.5).

Pour cela, dans le chapitre suivant, nous nous concentrons sur la problématique de précision et consommation des ressources matérielles, et nous rappelons les techniques les plus utilisées

dans la littérature afin de réduire la complexité de calcul et la consommation des ressources matérielles.

Dans cette thèse, nous avons apporté deux contributions afin de réduire la consommation des ressources matérielles de l'interpolation hétérogène et de maintenir la qualité de l'interpolation de Catmull-Rom (bi-cubique). Ces deux contributions ont fait l'objet de deux revues internationales. La première est détaillée dans le chapitre 3 et la seconde dans le chapitre 4.

1.5 Conclusion

Dans ce chapitre nous avons d'abord introduit les différentes approches utilisées dans le traitement d'images avec une performance sous-pixellique. Les différentes techniques d'interpolation ont été ensuite étudiées afin de choisir celle qui présente un bon compromis entre complexité de calcul et qualité des résultats. Enfin, nous avons rappelé en détail la technique CIN et ses enjeux, en particulier la complexité de calcul. Dans le chapitre suivant, nous allons détailler les problématiques et les challenges en implémentations matérielles.

Bibliographie

[1] R. R. Schultz, L. Meng, and R. L. Stevenson. Subpixel motion estimation for super-resolution image sequence enhancement. *Journal of Visual Communication and Image Representation*, 9(1) :38 – 50, 1998.

[2] J. Jeon and J. Paik. Single image super-resolution based on subpixel shifting model. *Optik*, 126(24) :4954 – 4959, 2015.

[3] J. Wang, Z. Zhou, J. Zhang, J. Cao, Q. Wu, X. Fan, Z. Gong, and B. Zhao. A new sub-pixel imaging method for image super resolution. In *2013 International Conference on Information Science and Cloud Computing*, pages 113–117, 2013.

[4] P. R. Hill, T. K. Chiew, D. R. Bull, and C. N. Canagarajah. Interpolation free subpixel accuracy motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(12) :1519–1526, 2006.

[5] Y. Chi, T. Tran, and R. Etienne-Cummings. Optical flow approximation of sub-pixel accurate block matching for video coding. In *IEEE ICASSP*, volume 1, pages 1017–1020, 01 2007.

[6] M. Sutton, J.J. Orteu, and H. Schreier. *Image Correlation for Shape, Motion and Deformation Measurements. Basic Concepts,Theory and Applications*. Springer, 2009.

[7] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks, 2016.

[8] V. N. Dvornychenko. Bounds on (deterministic) correlation functions with application to registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2) :206–213, 1983.

[9] Q. TIAN and M. N. Huuns. Algorithms for subpixel registration. In *Computer Vision Graphics and Image Processing*, 1986.

[10] M. Shimizu and M. Okutomi. An analysis of sub-pixel estimation error on area-based image matching. In *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*, volume 2, pages 1239–1242 vol.2, 2002.

[11] X. Shi, Y. Zhang, and J. Jiang. Insar image registration using modified correlation coefficient algorithm. In *2006 7th International Symposium on Antennas, Propagation EM Theory*, pages 1–4, 2006.

[12] P. Cheng and C. H. Menq. Cancelling bias induced by correlation coefficient interpolation for sub-pixel image registration. *Measurement Science and Technology*, 24(3) :035404, feb 2013.

[13] J. Almonacid-Caballer, J. Pardo-Pascual, and L. Ruiz. Evaluating fourier cross-correlation sub-pixel registration in landsat images. *Remote Sensing*, 9(10) :1051, Oct 2017.

[14] R. Szeliski and D. Scharstein. Symmetric sub-pixel stereo matching. In *Computer Vision — ECCV 2002*, pages 525–540, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[15] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1) :185 – 203, 1981.

[16] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[17] J. J. Little, H. Bulthoff, and T Poggio. Parallel optical flow using local voting. In *Proc. 2nd Intern. Conf. Comput. Vis.*, pages 454–459, 12 1988.

[18] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31 :671–679, 1983.

[19] J. L. Barron, D. J. Fleet, and S.S Beauchemin. Performance of optical flow technique. In *IJCV*, pages 43–77, 1994.

[20] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Comput. Surv.*, 27(3) :433–466, September 1995.

[21] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. In *Int J Comput Vision 2*, page 283–310, 1989.

[22] A. Singh. Optic flow computation : A unified perspective. *IEEE Computer Society Press*, 1992.

[23] D. J. Fleet and A. D. Jepson. Computation of component image velocity from local phase information. *Int J Comput Vision*, 5 :77–104, 1990.

[24] J. Wills, S. Agarwal, and S. Belongie. A feature-based approach for dense segmentation and estimation of large disparity motion. *Int J Comput Vision*, 68 :125–143, 2006.

[25] Y. Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering : Fundamentals, Algorithms, and Standards*. CRC Press, Inc., USA, 2nd edition, 2008.

[26] D. J. Heeger. Optical flow using spatiotemporal filters. *Int J Comput Vision*, 1 :279–302, 1988.

[27] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet : Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[28] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0 : Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1647–1655, 2017.

[29] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2720–2729, 2017.

[30] D. Sun, X. Yang, M. Liu, and J. Kautz. PWC-Net : CNNs for optical flow using pyramid, warping, and cost volume. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018.

[31] T. Hui, X. Tang, and C. C. Loy. Liteflownet : A lightweight convolutional neural network for optical flow estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8981–8989, 2018.

[32] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *Computer Vision – ECCV 2014*, pages 184–199, Cham, 2014. Springer International Publishing.

[33] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2) :295–307, 2016.

[34] J. Kim, J. K. Lee, and K. M. Lee. Deeply-recursive convolutional network for image super-resolution. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1637–1645, 2016.

[35] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, 2016.

[36] W. Burger and M. J. Burge. *Pixel Interpolation*, pages 539–563. Springer London, London, 2016.

[37] T. M. Lehmann, C. Gonner, and K. Spitzer. Survey : interpolation methods in medical image processing. *IEEE Transactions on Medical Imaging*, 18(11) :1049–1075, 1999.

[38] L. Roszkowiak, A. Korzynska, J. Zak, D. Pijanowska, Z. Swiderska-Chadaj, and T. Markiewicz. Survey : interpolation methods for whole slide image processing. *Journal of Microscopy*, 265(2) :148–158, 2016.

[39] M. S. Pan, X. L. Yang, and J. T. Tang. Research on interpolation methods in medical image processing. *Journal of Medical Systems*, 36(2) :777–807, Jun 2010.

[40] S. H. Mahajan and V. K. Harpale. Adaptive and non-adaptive image interpolation techniques. In *2015 International Conference on Computing Communication Control and Automation*, pages 772–775, 2015.

[41] X. Li and M. T. Orchard. New edge directed interpolation. In *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, volume 2, pages 311–314 vol.2, 2000.

[42] S. Chen. Vlsi implementation of an adaptive edge-enhanced image scalar for real-time multimedia applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(9) :1510–1522, 2013.

[43] J. W. Hwang and H. S. Lee. Adaptive image interpolation based on local gradient features. *IEEE Signal Processing Letters*, 11(3) :359–362, 2004.

[44] X. Kang, S. Li, and J. Hu. Fusing soft-decision-adaptive and bicubic methods for image interpolation. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1043–1046, 2012.

[45] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6) :1153–1160, 1981.

[46] J. Madhura and D. R. R. Babu. A survey on noise reduction techniques for lung cancer detection. In *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 637–640, 2017.

[47] G. Dougherty. *Digital image processing for medical applications*. California State University, Channel Islands, 2009.

[48] C. Boncelet. Chapter 7 - image noise models. In Al Bovik, editor, *The Essential Guide to Image Processing*, pages 143 – 167. Academic Press, Boston, 2009.

[49] H. Al-Ghaib and R. Adhami. On the digital image additive white gaussian noise estimation. In *2014 International Conference on Industrial Automation, Information and Communications Technology*, pages 90–96, 2014.

[50] K. Irie, A. E. McKinnon, K. Unsworth, and I. M. Woodhead. A model for measurement of noise in CCD digital-video cameras. *Measurement Science and Technology*, 19(4) :045207, 2008.

[51] G. E. Healey and R. Kondepudy. Radiometric CCD camera calibration and noise estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3) :267–276, 1994.

[52] hamamatsu. Image sensors.

[53] M. Konnik and J. Welsh. High-level numerical simulations of noise in CCD and CMOS photosensors : review and tutorial, 2014.

[54] JCGM Member Organization. International vocabulary of metrology–basic and general concepts and associated terms (vim 3rd edition). *BIPM*, 20, 2012.

[55] W. R. Leo. 1 - introduction to probability modeling. In John L. Stanford and Stephen B. Vardeman, editors, *Statistical Methods for Physical Science*, volume 28 of *Methods in Experimental Physics*, pages 1 – 34. Academic Press, 1994.

[56] D. J. Hand. Measurements and their uncertainties : A practical guide to modern error analysis by ifan g. hughes, thomas p. a. hase. *International Statistical Review*, 79(2) :280–280, 2011.

[57] M. Grédiac and F. Sur. Effect of sensor noise on the resolution and spatial resolution of the displacement and strain maps obtained with the grid method. *Strain*, 50(1) :1–27, 2014.

[58] W. H. Peters and W. F. Ranson. Digital Imaging Techniques In Experimental Stress Analysis. *Optical Engineering*, 21(3) :427 – 431, 1982.

[59] E. Zappa, P. Mazzoleni, and A. Matinmanesh. Uncertainty assessment of digital image correlation method in dynamic applications. *Optics and Lasers in Engineering*, 56 :140–151, May 2014.

[60] DIC challenge : http ://sem.org/dic-challenge/.

[61] M. Grédiac, B. Blaysat, and F. Sur. A robust-to-noise deconvolution algorithm to enhance displacement and strain maps obtained with local DIC and LSA. *Experimental Mechanics*, 59(2) :219–243, 2019.

[62] B. Pan, H. Xie, and Z. Wang. Equivalence of digital image correlation criteria for pattern matching. *Applied Optics*, 49(28) :5501, 2010.

[63] B. Blaysat, J. Neggers, M. Grédiac, and F. Sur. Towards criteria characterizing the metrological performance of full-field measurement techniques. Application to the comparison between local and global versions of DIC. *Experimental Mechanics*, 60(3) :393–407, 2020.

[64] J.-C Passieux and R. Bouclier. Classic and inverse compositional gauss-newton in global DIC. *International Journal for Numerical Methods in Engineering*, 119(6) :453–468, 2019.

[65] Allied vision. CCD or CMOS : can cmos sensors replace ccds in all cases ?, 2017.

[66] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian. Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 17(10) :1737–1754, 2008.

[67] Z. Gao, X. Xu, Y. Su, and Q. Zhang. Experimental analysis of image noise and interpolation bias in digital image correlation. *Optics and Lasers in Engineering*, 81 :46 – 53, 2016.

[68] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11) :1338–1351, 2003.

[69] A. Lukin. A multiresolution approach for mproveing quality of image denoising algorithms. *IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*, 2006.

[70] W. Dai and Y. Ye. Image denoising based on combination of wiener filter and wavelet shrinkage. *2007 IEEE International Conference on Integration Technology*, 2007.

[71] M. T. Yildirim, A. Basturk, and M. E. Yuksel. Impulse noise removal from digital images by a detail-preserving filter based on type-2 fuzzy logic. *IEEE Transactions on Fuzzy Systems*, 16(4) :920–928, 2008.

[72] V. Karnati, M. Uliyar, and S. Dey. Fast non-local algorithm for image denoising. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3873–3876, 2009.

[73] C. Knaus and M. Zwicker. Progressive image denoising. *IEEE Transactions on Image Processing*, 23(7) :3114–3125, 2014.

[74] A. Awad. Denoising images corrupted with impulse, gaussian, or a mixture of impulse and gaussian noise. *Engineering Science and Technology, an International Journal*, 22(3) :746 – 753, 2019.

[75] L. Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246, 2016.

[76] T. Wang, M. Sun, and K. Hu. Dilated deep residual network for image denoising. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1272–1279, 2017.

[77] Q. Xiang and X. Pang. Improved denoising auto-encoders for image denoising. In *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–9, 2018.

[78] Z. Liu, W. Q. Yan, and M. L. Yang. Image denoising based on a cnn model. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 389–393, 2018.

[79] J. H. Park, J. Hyeon Kim, and S. I. Cho. The analysis of cnn structure for image denoising. In *2018 International SoC Design Conference (ISOCC)*, pages 220–221, 2018.

[80] F. Amiot, M. Bornert, P. Doumalin, J. C. Dupré, M. Fazzini, J. J. Orteu, C. Poilâne, L. Robert, R. Rotinat, E. Toussaint, B. Wattrisse, and J. S. Wienin. Assessment of digital image correlation measurement accuracy in the ultimate error regime : Main results of a collaborative benchmark. *Strain*, 49(6) :483–496, 2013.

[81] H. W. Schreier, M. Sutton, and A. Michael. Systematic errors in digital image correlation due to undermatched subset shape functions. *Experimental Mechanics*, 42(3) :303–310, 2002.

[82] S. S. Fayad, D. T. Seidl, and P. L. Reu. Spatial DIC errors due to pattern-induced bias and grey level discretization. *Experimental Mechanics*, 60(2) :249–263, 2020.

[83] M. Grédiac, B. Blaysat, and F. Sur. A critical comparison of some metrological parameters characterizing local digital image correlation and grid method. *Experimental Mechanics*, 57(6) :871–903, 2017.

[84] M. Grédiac, B. Blaysat, and F. Sur. On the optimal pattern for displacement field measurement : random speckle and DIC, or checkerboard and LSA ? *Experimental Mechanics*, 2020. Accepted.

Chapitre 2

Implémentation matérielle : précision et ressources matérielles

Aujourd'hui, les fabricants de machines de calcul ont atteint une limite dans les performances à noyau de calcul unique. En effet, la fréquence d'horloge des processeurs ne s'est pas améliorée de manière significative durant ces dernières années en raison de l'augmentation de la dissipation thermique [1, 2, 3] et des limitations physiques. Deux solutions sont disponibles pour augmenter la puissance de calcul. La plus simple consiste à utiliser des processeurs multi-cœurs capables de calculer un plus grand nombre d'opérations par cycle d'horloge en exécutant des tâches en parallèle. La seconde est basée sur l'utilisation d'unités de calcul spécialisées, appelées accélérateurs. Cette dernière est la tendance actuelle du calcul haute performance (HPC) en permettant aux fabricants d'obtenir des machines de calcul beaucoup plus puissantes [4, 5, 6].

L'efficacité de l'utilisation de ces accélérateurs pour des applications temps réel ne dépend pas seulement du temps de calcul, mais aussi du résultat et du coût de calcul. Chaque implémentation doit satisfaire des contraintes de précision spécifique. L'implémentation matérielle peut donc être considérée comme un problème d'optimisation. On cherche soit à minimiser l'utilisation des ressources sous contraintes de performance, soit à maximiser la performance sous contraintes de ressources.

La problématique d'optimisation précision et consommation des ressources matérielles fait l'objet de ce chapitre. D'abord, nous présentons les architectures matérielles les plus utilisées pour accélérer les calculs complexes et obtenir des résultats en temps réel, à savoir -*Graphical Processing Units*- (GPU) et -*Field Programmable Gate Arrays*- (FPGA). Ensuite, nous citons les techniques les plus utilisées dans la littérature afin de réduire la complexité de calcul et la consommation des ressources matérielles en se concentrant sur la représentation arithmétique en virgule fixe. Enfin, nous présentons la méthodologie utilisée pour optimiser et estimer l'erreur liée au calcul arithmétique en virgule fixe, et pour répondre à une contrainte de précision donnée.

2.1 Plateformes matérielles

Actuellement, les deux classes d'accélérateurs matériels les plus utilisées sont les unités de calcul graphiques (GPUs) et les (FPGAs) [4, 5, 6, 7, 8]. Ces dernières ont été utilisées dans le cadre de cette thèse pour atteindre les performances souhaitées. Dans la suite, nous décrivons ces deux plateformes matérielles.

2.1.1 FPGA

Un FPGA est un circuit intégré qui peut être reconfiguré après sa fabrication, en fonction de l'application souhaitée ou des exigences fonctionnelles. Cette caractéristique distingue les FPGAs des circuits intégrés spécifiques à une application -*Application Specific Integrated Circuits*- (ASIC), qui sont fabriqués sur mesure pour des tâches données [9, 10, 11].

Les FPGAs sont composés principalement de cellules logiques reprogrammables, d'un réseau d'interconnexions qui relie ces cellules logiques entre elles pour mettre en œuvre des fonctions spécifiques, et d'autre ressources supplémentaires pour effectuer et accélérer des tâches spécifiques [11] (voir la figure 2.1). Ces derniers en plus du flot de conception sont détaillés ci-dessous.

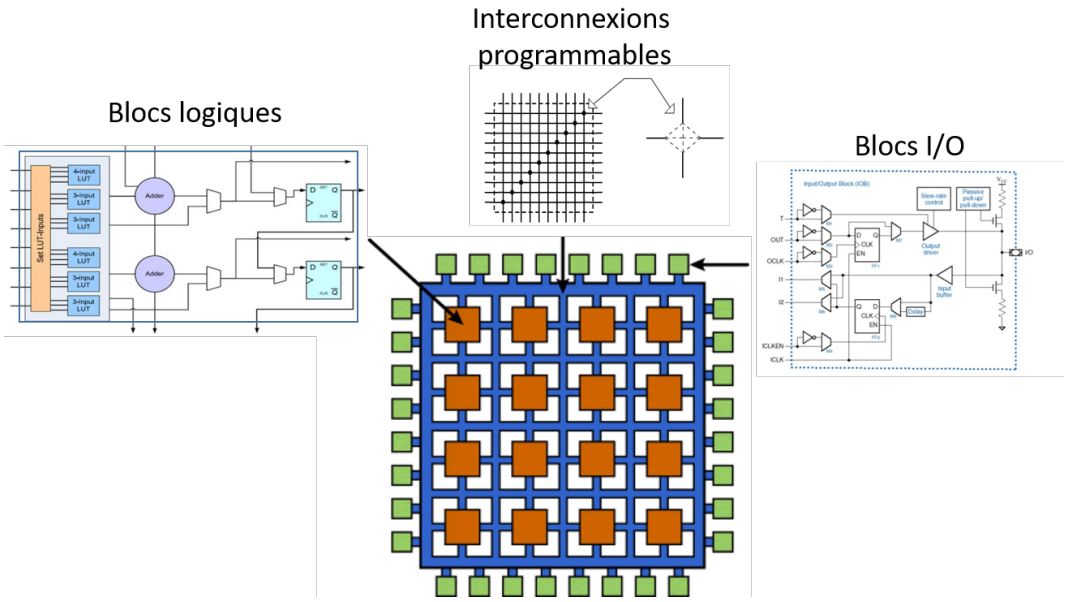


FIGURE 2.1 – Architecture de base d'un FPGA.

2.1.1.1 Cellules logiques

Les cellules logiques sont les éléments atomiques d'un FPGA et sont disposées sous forme de matrice. La fonction de base des cellules logiques est de fournir des capacités de calcul et de stockage au FPGA. Une cellule logique est généralement composée d'une lookup table (LUT) pour la mise en œuvre de la partie combinatoire et d'un élément de stockage (registre) pour la mise en œuvre des fonctions séquentielles [11]. Dans les dispositifs FPGA modernes, cette

structure a évolué en blocs complexes qui offrent une plus grande flexibilité et une meilleure optimisation de la mise en place des ressources logiques. Dans ce contexte, Intel a introduit les modules logiques adaptatifs -Adaptive Logic Module- (ALM), qui sont les nouveaux blocs élémentaires. De même, Xilinx a introduit les Slices comme blocs logiques élémentaires. Ces blocs sont relativement similaires dans leurs structures [11, 12]. La figure 2.2 montre la structure d'un ALM d'un FPGA Intel Cyclone V. Il dispose de quatre registres, d'une LUT à 8 entrées, et de deux additionneurs.

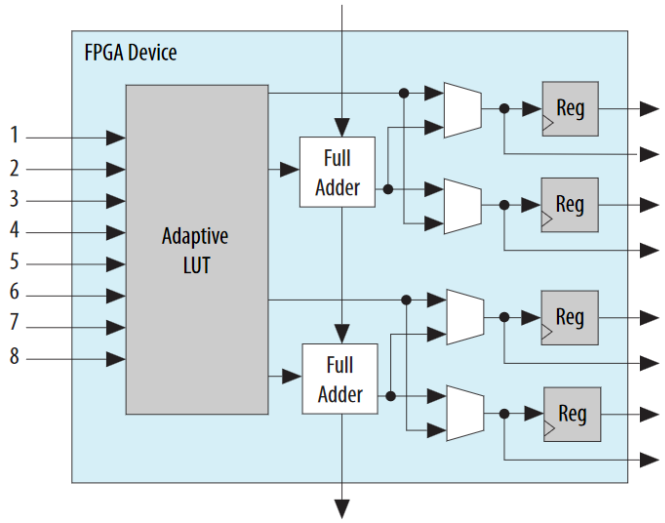


FIGURE 2.2 – Structure d'un ALM d'un FPGA Intel Cyclone V [11].

Dans les FPGAs récents, ces blocs élémentaires sont regroupés en entités plus grandes pour former des blocs logiques -Logic Array Blocks- (LAB) chez Intel et -Configurable Logic Blocks- (CLB) chez Xilinx [11, 12].

2.1.1.2 Réseaux d'interconnexions

Les blocs logiques d'un FPGA sont interconnectés les uns aux autres par un réseau programmable. Ce réseau fournit des connexions entre les ressources logiques pour mettre en œuvre un circuit déjà défini. Les interconnexions de routage sont constituées de fils et de commutateurs programmables qui forment la connexion souhaitée.

La plupart des FPGA partagent une structure de routage similaire qui implique des pistes de routage horizontales et verticales. Ces pistes sont interconnectées par des boîtiers de commutation [9, 11, 12]. La programmation de ces commutateurs permet d'interconnecter les blocs souhaités, comme le montre la figure 2.3.

Les connexions entre les blocs logiques sont un facteur clé dans la performance d'une implémentation FPGA. Plus la connexion entre deux blocs est longue, plus les retards de transmission seront élevés, ce qui réduit la performance en termes de fréquence de fonctionnement. Le placement des blocs logiques sur le FPGA est donc un élément crucial pour une meilleure performance.

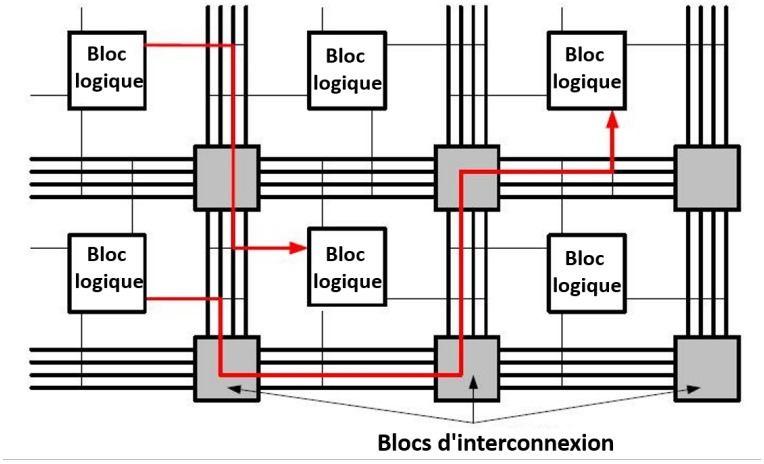


FIGURE 2.3 – Réseau d'interconnexions.

Les éléments de même bloc logique ont des communications privilégiées. Ainsi, des fonctions complexes peuvent être mises en œuvre sans passer par le réseau d'interconnexions, ce qui se traduit par de meilleures fréquences de fonctionnement.

Il convient aussi de noter la présence d'autres interconnexions utilisées pour relier un bloc logique à d'autres éléments tels que les blocs de mémoire externes. Ces éléments appelés *ressources supplémentaires* font l'objet du paragraphe suivant.

2.1.1.3 Ressources supplémentaires

Dans les architectures récentes des FPGAs, des ressources physiques supplémentaires sont câblées dans le FPGA [10, 11, 12, 13]. Ces composants physiques sont énumérés comme suit :

- **les entrées/sorties configurables (E/S) :** sont des circuits qui assurent l'interface entre le FPGA et l'environnement externe. Chaque circuit d'entrée/sortie contrôle un périphérique du FPGA et peut être configuré comme une entrée, une sortie, ou un signal bidirectionnel.
- **Blocs de mémoire SRAM (-Static Random Access Memory-) :** sont des éléments de stockage qui ont une position fixe dans le FPGA. Ils sont idéals pour stocker de grands blocs de données.
- **Blocs de mémoire distribuée :** contrairement aux blocs SRAM, les blocs de mémoire distribuée peuvent être implémentés n'importe où, et sont créés automatiquement à partir d'un ensemble de ressources logiques. Ce type de mémoire est idéal pour les petites mémorisations (registres à décalage, petits buffers).
- **Processeur -Hard Processor System- (HPS) :** est un CPU qui remplace les systèmes de co-conception basés sur le softcore (NIOS, MicroBlaze) dans un certain nombre de cartes FPGA. Actuellement, les fabricants de FPGA utilisent généralement des processeurs ARM connus pour leur faible consommation d'énergie dans les applications embarquées et leur performance de calcul [14].

— **Blocs de traitement numérique du signal (DSP) :** sont des unités arithmétiques câblées. Elles sont optimisées pour le débit de calcul et la consommation d'énergie. Les blocs DSP fonctionnent à différentes largeurs de bits. Par exemple, les blocs DSP d'Intel FPGA peuvent mettre en œuvre des multiplications de (27×27)bits, (18×18)bits, ou (9×9)bits. Grâce aux blocs DSP, les FPGAs de haut de gamme tels que Virtex Ultrascale de Xilinx ou Stratix 10 d'Intels peuvent atteindre des pics de performance importants. Il convient de savoir que certain dispositifs FPGA actuels contiennent des DSPs qui prennent en charge l'arithmétique en virgule flottante de précision simple [15, 13].

— **Blocs tenseur IA :** contiennent des réseaux denses de multiplieurs. Ils sont généralement utilisés dans les applications d'intelligence artificielle (IA). L'architecture des blocs tenseur IA est adaptée aux multiplications matrices-matrices ou vecteurs-matrices utilisées dans un large éventail de calculs AI. Ils ont la capacité de fonctionner efficacement pour les matrices de petite et de grande taille. Grâce aux blocs "tenseur IA", le FPGA Stratix 10 NX peut atteindre un débit de calcul jusqu'à 15 fois supérieur à celui de ces blocs DSP standards. La figure 2.4 montre l'architecture interne d'un tenseur IA fourni par le FPGA Stratix 10 NX.

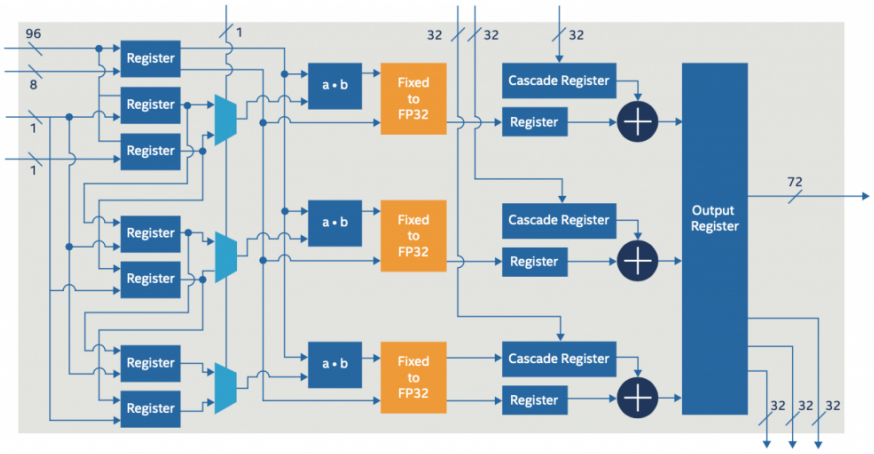


FIGURE 2.4 – Structure d'un bloc tenseur IA présent dans le FPGA Stratix 10 NX [13].

2.1.1.4 Flot de conception

La transcription d'un code source décrit en HDL en une architecture matérielle suit le flot de conception décrit dans la figure 2.5. Les étapes de ce flot de conception sont détaillées ci-après.

— **Description du hardware :**consiste à décrire la fonctionnalité souhaitée en utilisant une ou plusieurs techniques. En général, l'algorithme à implémenter sur un FPGA est décrit à l'aide d'un langage de description de matériel -Hardware Description Language- (HDL). Les HDLs tels que le -Very High Speed Integrated Circuit Hardware Description Language- (VHDL) ou Verilog sont des langages spécialisés utilisés pour décrire la structure et le comportement des circuits logiques numériques, et plus particulièrement ceux

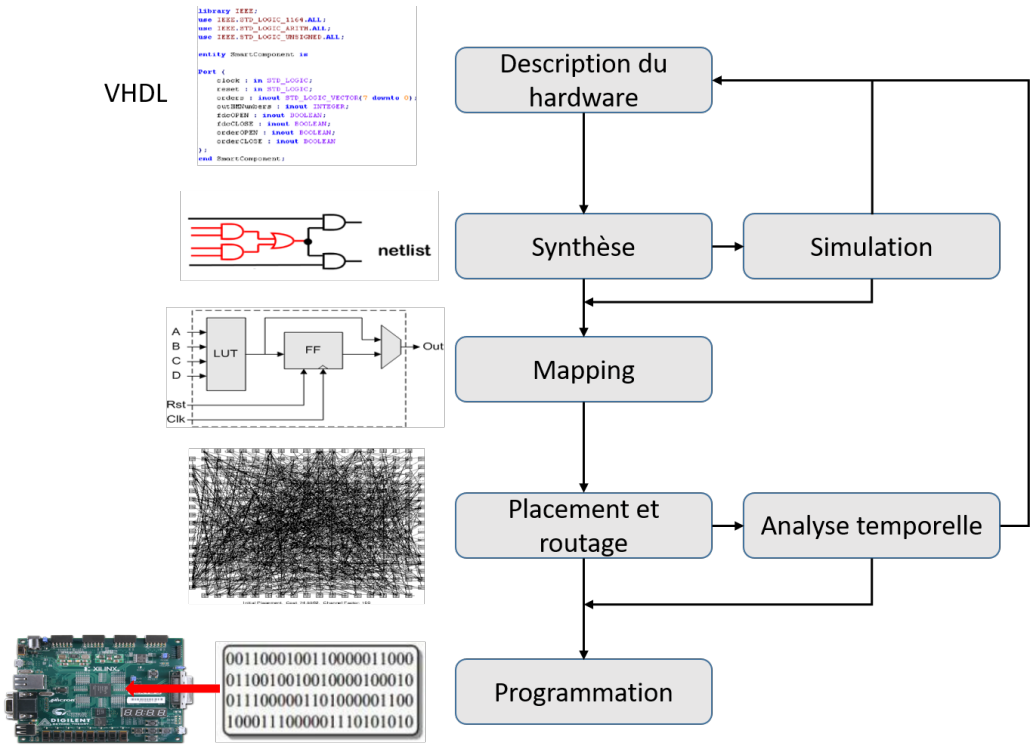


FIGURE 2.5 – Flot de conception en FPGA.

mis en œuvre sur les ASICs et les FPGAs. Une autre solution consiste à utiliser des outils de synthèse de haut niveau (HLS) pour convertir le type de code specific (généralement, C ou C++) en HDL [9]. OpenCL est un cadre open-source pour la programmation parallèle. Il élimine les complexités du HDL, permettant aux programmeurs d'écrire des fonctions du noyau accélérées par le matériel dans un code impératif de haut niveau. Bien qu'un circuit fonctionnel soit obtenu beaucoup plus rapidement que pour les autres approches, il a généralement des performances moindres [16, 17].

— **Synthèse :** c'est l'étape après la description du circuit. Elle consiste à traduire la description du circuit en une netlist. La netlist obtenue représente une description au niveau "portes" ou RTL (Register Transfer Level) du circuit. Il s'agit essentiellement de spécifier les composants du système, l'interconnexion entre eux, et les broches d'entrée/sortie.

— **Simulation :** est généralement utilisée pour vérifier la fonctionnalité du circuit avant de passer aux phases supérieures du processus. Cette phase de vérification est fonctionnelle et ne tient pas compte les retards des signaux. Elle consiste essentiellement à alimenter le simulateur par des vecteurs de test (chaque vecteur de test est un ensemble de stimuli injectés à l'entrée du circuit) et à vérifier les résultats par rapport aux spécifications. Si des erreurs sont détectées à cette phase, la conception doit être ajustée.

— **Mapping :** se fait généralement en une série d'opérations qui traitent la netlist. Il consiste à transformer la netlist en un réseau équivalent fonctionnel qui peut être mis en œuvre par le dispositif FPGA. Parmi les opérations courantes en mapping l'adaptation aux ressources physiques du FPGA, l'optimisation, et la vérification des règles de conception.

— **Placement et routage :** la phase de placement consiste à sélectionner les modules obtenus lors de la phase de mapping et à les affecter aux emplacements spécifiques du FPGA. Le routage consiste à interconnecter ces blocs en utilisant les ressources de routage disponibles dans le FPGA.
— **Analyse temporelle :** le fait d'avoir une conception placée et routée donne de nouvelles informations sur les retards des signaux (retards d'interconnexions) et des composants de la conception. Ces informations sont utilisées dans une simulation précise du temps.
— **Programmation :** à la fin du placement et du routage, un fichier est généré qui contient toutes les informations nécessaires pour configurer le dispositif FPGA, configuration des blocs logiques et des interconnexions. Ces informations sont stockées sous forme de flot binaire, où chaque bit indique l'état d'ouverture ou de fermeture d'un commutateur sur le dispositif.

2.1.2 GPU

Les GPUs ont été initialement conçus pour traiter des tâches de calcul intensif pour des applications graphiques telles que le lancer de rayons (Raytracing) afin d'accélérer le rendu des graphiques. Par la suite, les GPUs ont commencé à être utilisés comme dispositif de calcul parallèle à usage plus général (General Purpose-GPU), induisant de nouveaux paradigmes de programmation GPU. Actuellement, tous les ordinateurs modernes sont équipés d'un ou plusieurs GPUs. Il convient de préciser qu'une architecture peut être dédiée à plusieurs domaines. Ainsi, si les GPUs sont efficaces pour la gestion des flux graphiques, ils ont aussi démontré leur optimalité pour d'autres domaines, tels que l'apprentissage et l'inférence des réseaux de neurones ou bien encore les calculs parallèles [18].

2.1.2.1 Architecture GPU moderne

Les GPUs sont des processeurs multi-cœurs spécialisés dans le calcul parallèle. La figure 2.6 montre l'architecture de base d'une GPU. Les architectures GPU actuelles, telles que l'architecture Nvidia Volta décrite dans la figure 2.7, utilisent des milliers d'éléments de traitement. Ces derniers sont regroupés en streaming multiprocesseurs (SMs) [19].

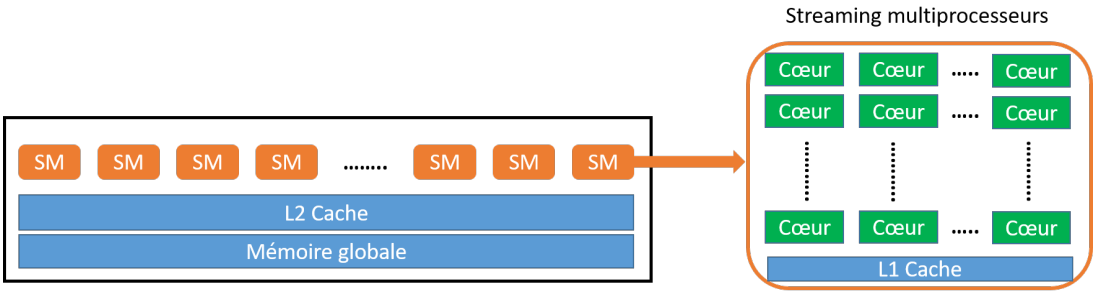


FIGURE 2.6 – Architecture de base d'une GPU.

Le SM d'un Volta GV100 est partitionné en quatre blocs de traitement. Chacun avec 16

cœurs FP32, 8 cœurs FP64, 16 cœurs INT32, et deux cœurs tenseur de précision mixte conçus spécifiquement pour l'arithmétique matricielle de l'IA et l'apprentissage profond, comme décrit dans la figure 2.9. Avec 80 SMs, une GPU Volta GV100 a un total de 5120 cœurs FP32, 5120 cœurs INT32, 2560 cœurs FP64, et 640 cœurs tenseur.

FIGURE 2.7 – Architecture de la GPU Volta GV100 [19].

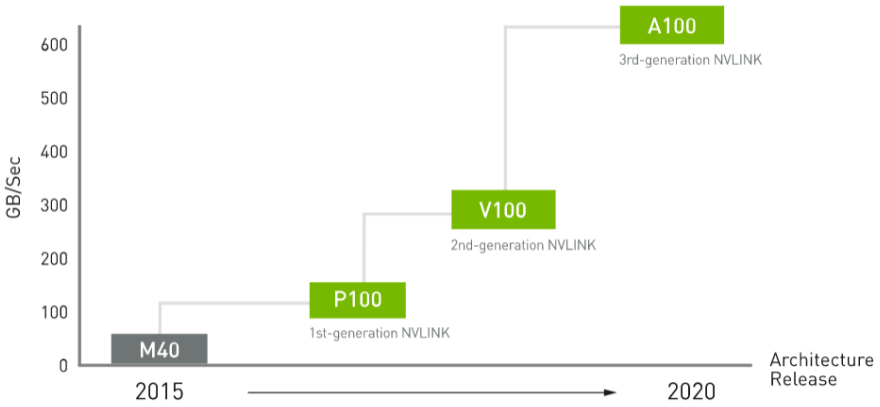


FIGURE 2.8 – Performance des différentes générations NVLink [20].

Afin d'améliorer les performances de calcul, les GPUs modernes telles que la GV100 ont ajouté de nombreuses nouvelles fonctionnalités par rapport aux générations précédentes de GPU, et ont intégré du matériel spécifique tels que les NVLink de Nvidia et des cœurs tenseur de précision mixte.

Ces dernières sont décrites ci-après.

— **NVLink :** est la technologie d'interconnexions à haut débit de NVIDIA, introduite pour la première fois en 2016 avec la GPU Pascal GP100. NVLink offre des performances nette-

ment supérieures pour les configurations GPU-GPU et GPU-CPU par rapport à l'utilisation des interconnexions PCIe, près de 10 fois par rapport au PCIe 4ème génération [20]. La figure 2.8 montre la performance des différentes générations NVLink. La GV100 intègre la deuxième génération de NVLink [19].

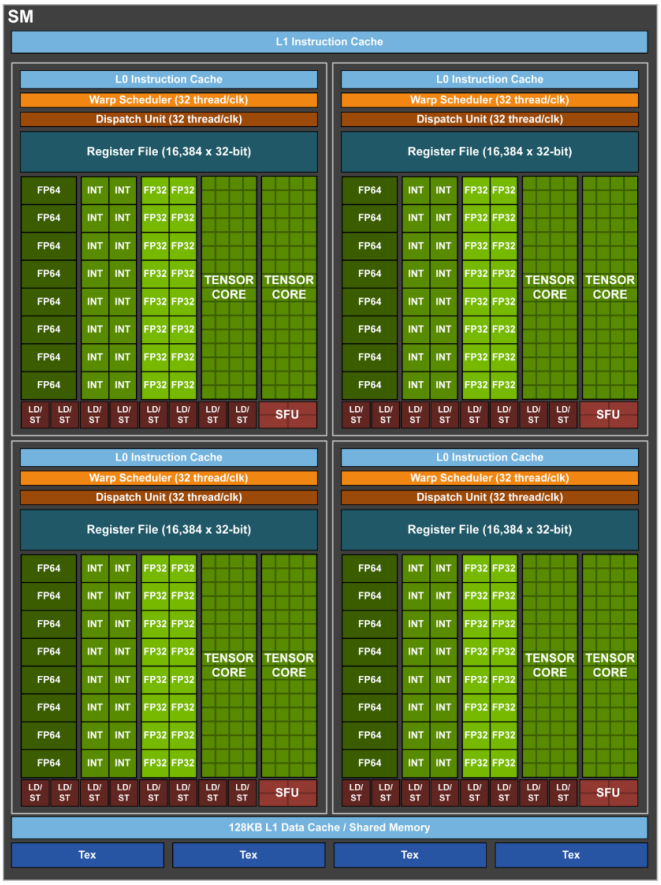


FIGURE 2.9 – Architecture d'un SM de Nvidia Volta GV100 [19].

— **Cœurs tenseur :** ont une capacité clé permettant à l'architecture des GPUs modernes d'accélérer l'arithmétique matricielle, en l'occurrence les processus d'apprentissage et d'inférence en IA. Ces GPUs offrent des performances nettement supérieures pour l'entraînement des réseaux de neurones par rapport aux architectures de générations précédentes.
Les cœurs tenseur et leurs chemins de données associés sont conçus sur mesure pour augmenter considérablement le débit de calcul en virgule flottante avec un rendement énergétique élevé. Chaque cœur fonctionne en matrices 4×4 et effectue l'opération suivante :

$$D = A \times B + C$$

Où A, B, C et D sont des matrices 4×4.
En fonction de la taille des données des matrices A et B, on distingue trois versions de cœurs tenseur [21]. Dans le cas d'une GV100, les matrices A et B sont des matrices

FP16, tandis que les matrices d'accumulation C et D peuvent être des matrices FP16 ou FP32 (voir la figure 2.10).

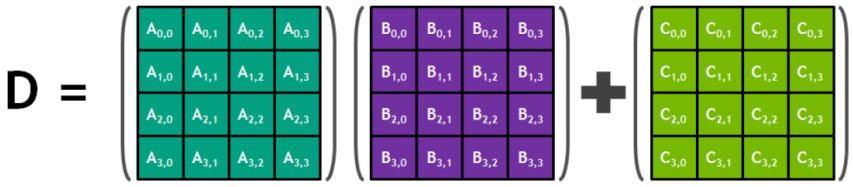


FIGURE 2.10 – Opération de base d'un cœur Tenseur GV100, multiplication et accumulation [19].

Dans la GPU Volta GV100, chaque Cœur Tenseur effectue 64 opérations -Fused Multiply-Add- (FMA) en virgule flottante par cycle d'horloge. En outre, huit cœurs tenseur dans un SM effectuent un total de 512 opérations FMA (ou 1024 opérations individuelles en virgule flottante) par cycle d'horloge. Les cœurs tenseur fournissent jusqu'à 125 -Tera Floating Point Operations Per Second- (TFLOPS) pour les applications d'apprentissage et d'inférence [19].

— **Précision mixte :** consiste à l'utilisation de différents types de données dans un modèle pour consommer moins de mémoire et rendre le modèle plus rapide. Pour cela, la précisons de certaines parties du modèle est maintenue pour garder la précision spécifique à la tâche, et elle est réduite pour le reste du modèle. Cela permet de satisfaire les deux facteurs, vitesse et précision.
Les cœurs tenseur de la GPU Volta GV100 fonctionnent avec des données d'entrée FP16 pour la multiplication, ensuite le résultat est accumulé en utilisant une addition FP32 avec les autres produits intermédiaires de multiplication de matrices [19] (voir la figure 2.11).

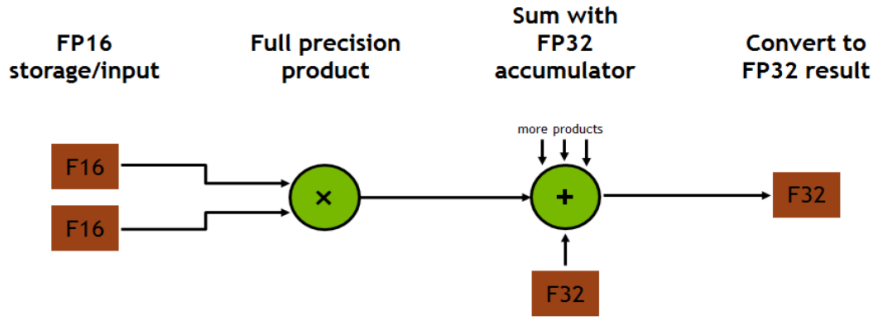


FIGURE 2.11 – Principe de la précision mixte du Tenseur GV100 [19].

Les Tenseurs à précision mixte de la Volta GV100 délivrent jusqu'à 12X plus de TFLOPS par rapport à la GPU Pascal GP100 [21].

2.1.2.2 Programmation du GPU

Afin de programmer les GPUs, de nombreuses primitives de programmation ont été proposées. Parmi elles, -Compute Unified Device Architecture- (CUDA), lancée par Nvidia en 2006, est devenue le choix du grand public en fournissant une interface de programmation C/C++ facile à utiliser [22]. NVIDIA a aussi développé le NVIDIA -CUDA Deep Neural Network library- (cuDNN) permettant aux programmeurs utilisant des frameworks de *deep learning* comme Tensorflow et Pytorch d'utiliser les GPUs [23].

2.1.3 Bilan

Le choix de la bonne plateforme matérielle dépend principalement de l'application et de ses contraintes. Pour une meilleure implémentation matérielle, les facteurs tels que la facilité de programmation, les performances et le coût doivent être pris en considération. Les tableaux 2.2 et 2.1 donnent des caractéristiques de quelques FPGAs d'Intel et GPUs de Nvidia.

TABLE 2.1 – Caractéristiques de quatre FPGAs d'Intel [11, 24, 25, 26].

FPGA	Cyclone V GT	Arria 10 SX	Stratix 10 SX	Agilex F
Technologie	28 nm	20 nm	14 nm	10 nm
ALMs	113560	250540	933120	912800
Registres	454240	1002160	3732480	3651200
Multiplieurs 18x18	684	3376	11520	17056
Mémoire (Mb)	14	48	244	287
Performance en virgule fixe (TMACs)	0.15	3.7	23.0	/
Performance en précision simple (TFLOPS)	0.1	1.5	9.2	12.8
Puissance	jusqu'à 5 W	jusqu'à 75 W	jusqu'à 225 W	> 135 W

TABLE 2.2 – Caractéristiques de cinq GPUs de Nvidia [19, 27, 28].

GPU NVIDIA	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)	GA100 (Ampère)
Technologie	28 nm	28 nm	16 nm	12 nm	7 nm
Cuda Cœurs FP32	2880	3072	3584	5120	6912
Cœurs tenseur	/	/	/	640	432
FP16 TFLOPS	/	/	21.2	30.4	78
FP32 TFLOPS	5	6.8	10.6	15.7	19.5
FP64 TFLOPS	1.7	0.21	5.3	7.8	9.7
Tenseur TFLOPS	/	/	/	125	312
Puissance	235 W	250 W	300 W	300 W	400 W
Génération NVLink	/	/	1ère	2ieme	3ieme

Au risque de répéter des évidences, les FPGAs sont conçus pour effectuer des opérations parallèles à grain fin avec une approche de programmation proche du matériel. Les GPUs sont optimisées pour le traitement parallèle (à plus gros grain) des opérations arithmétiques en virgule flottante, en utilisant des milliers de petits cœurs. La figure 2.12 montre une comparaison entre FPGA et GPU. Les GPUs présentent un avantage lorsque l'on considère la puissance de traitement totale en virgule flottante, l'effort de développement et le coût du dispositif. Les FPGAs offrent d'énormes capacités de traitement avec une grande efficacité énergétique, réduisant

la gestion thermique et l'espace requis. Cette caractéristique permet l'intégration de matériel d'accélération dans de faibles volumes, des équipements embarqués ou des environnements à température extrême [29, 30, 31, 32].

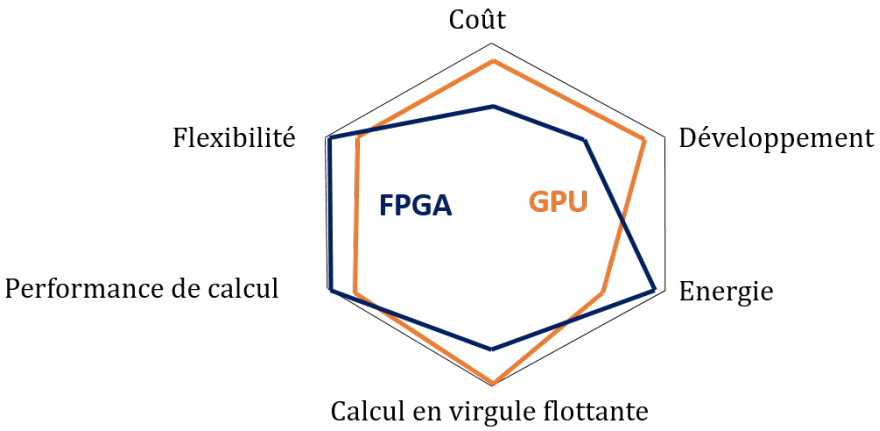


FIGURE 2.12 – Comparaison entre FPGA et GPU.

En général, les FPGAs fournissent une flexibilité plus importante que les GPUs pour jouer avec différents types de données tels que les données binaires, et même les données personnalisées, grâce à leur reconfigurabilité. De plus, les FPGAs se caractérisent par une capacité de calcul nettement supérieure grâce à la mémoire interne (on-chip memory) qui réduit le goulot d'étranglement entre les calculs et la mémoire externe. Cela permet aussi de réduire la latence qui est particulièrement importante pour les applications de cryptage, de codage, de synchronisation ou de contrôle devant gérer de faibles latences [30].

Compte tenu du coût et de la facilité d'utilisation, les GPUs sont rentables tant au niveau du développement que de l'installation du matériel. CUDA est plus accessible pour les "développeurs logiciels", qui n'ont pas besoin d'une connaissance approfondie sur le matériel. Cela est l'une des principales raisons pour lesquelles les GPUs sont largement utilisés de nos jours. Par opposition, les FPGAs nécessitent des ingénieurs concepteurs spécialisés ayant des connaissances sur un plus large spectre (électronique, HDL, algorithmes, communications, etc.) [29, 32].

Dans le cadre de cette thèse, nous avons utilisé les deux plateformes citées ci-dessus. Tout d'abord, le FPGA a été utilisé afin d'accélérer le processus d'interpolation détaillé dans le chapitre 1. Ensuite, le réseau de neurone appelé StaintNet, que nous avons développé dans le chapitre 5 a été implémenté sur GPU.

Dans la section suivante, nous nous concentrons sur les différentes techniques d'optimisation permettant d'atteindre le meilleur compromis entre précision et utilisation des ressources matérielles dans une implémentation matérielle.

2.2 Réduire la complexité de calcul : techniques d'approximation

L'implémentation matérielle peut être considérée comme un problème d'optimisation. On cherche soit à minimiser l'utilisation des ressources sous contraintes de performance, soit à maximiser la performance sous contraintes de ressources. Les techniques les plus utilisées pour accélérer le calcul et réduire l'utilisation des ressources matérielles [33, 34] sont citées ci-dessous.

2.2.1 Mémorisation

Traditionnellement, la mémorisation consiste à sauvegarder des résultats de fonctions afin de les réutiliser ultérieurement en tant que données d'entrée. Étant donné que de ces données d'entrée sont fréquemment réutilisées, leurs sorties calculées peuvent être stockées et utilisées sans qu'il soit nécessaire de ré-exécuter la fonction.

La mémorisation peut également être utilisée pour approximer des fonctions. En effet, si des entrées similaires fournissent des sorties similaires pour une fonction donnée, cela signifie que la sortie de la fonction déjà calculée peut approximativement couvrir cette gamme d'entrées.

2.2.2 Perforation en boucle

Son principe consiste à sauter certaines itérations d'une boucle pour réduire le coût de calcul. Elle n'est applicable qu'aux algorithmes à base de boucle mais peut fortement réduire le temps d'exécution. Elle peut être mise en œuvre tant au niveau logiciel que matériel dans le cadre de cible reconfigurable. Dans ce dernier cas, une boucle peut être déroulée par autant de circuits s'exécutant en parallèle (un pour chaque itération de la boucle) ou bien ré-exécuté séquentiellement. Par conséquent, l'apport de la perforation de la boucle sur les implémentations logicielles et matérielles peut être fondamentalement différent. Au niveau logiciel, il aura principalement un impact sur le temps d'exécution de l'application, tandis que dans une implémentation FPGA, il pourrait également avoir un effet sur la consommation d'énergie et de ressources matérielles.

2.2.3 Approximation fonctionnelle

Certains systèmes sont constitués de composants qui n'ont pas besoin d'être aussi précis que d'autres. L'idée est tirée du fait que même à l'intérieur d'un algorithme, certains composants affectent moins la précision finale que d'autres. Ces composants peuvent être approximés pour réduire la consommation d'énergie et de ressources, et améliorer le temps d'exécution de l'algorithme. Au niveau architectural ou matériel, l'approximation fonctionnelle peut apparaître comme une implémentation alternative aux opérateurs arithmétiques. L'exemple le plus classique étant de remplacer les multiplications et les divisions par des décalages. Cette technique est très efficace en FPGA et permet d'économiser l'utilisation des ressources matérielles. Au niveau logiciel, le calcul approximatif consiste à modifier l'algorithme de manière à ce qu'il

s'exécute approximativement, ce qui permet d'obtenir un résultat final plus rapidement. Une méthode classique d'approximation des fonctions est l'approximation polynomiale, qui utilise des polynômes représentant la fonction sur différentes plages. Il y a d'autres méthodes qui peuvent aussi être utilisées, comme l'utilisation des LUTs pour approximer des fonctions complexes (tels les fonctions trigonométriques) et l'approximation des nombres utilisés dans l'algorithme par des nombres de la forme $\sum 2^{n_i}$. Dans le cas des LUTs, la précision des résultats dépendra de la taille de la LUT utilisée et donc du pas de discrétisation.

2.2.4 Réduction de la précision des données

La réduction de la précision des données est une technique qui peut être mises en œuvre tant au niveau logiciel qu'architectural. La réduction de la précision des données d'une application, c'est-à-dire le nombre de bits utilisés pour représenter les données, est une technique permettant de réduire la consommation de la mémoire et des ressources matérielles. Cela permet également de réduire la consommation d'énergie (moins de données à transférer depuis/vers la mémoire) [35, 36].

La façon dont la réduction de la précision des données peut être utilisée pour approcher les applications est évidente. Au niveau logiciel, la précision des unités à virgule flottante peut être facilement modifiée grâce à l'utilisation de bibliothèques dédiées, ou même en changeant simplement le type et/ou la taille des variables. La même chose peut être faite dans les implémentations en VHDL/Verilog, en les adaptant pour traiter des vecteurs de données plus petits.

La réduction de la précision des données peut apporter de bonnes améliorations en termes de consommation de ressources matérielles et d'énergie pour les implémentations matérielles, mais elle ne présente souvent pas de réduction de coût de calcul au niveau logiciel [37].

2.2.5 Utilisation des réseaux de neurones

Les réseaux de neurones sont des modèles inspirés du comportement des réseaux neuronaux biologiques. Ils sont composés de neurones artificiels interconnectés comme le montre la figure 2.13. Ils sont utilisés pour résoudre des problèmes d'application informatique dans divers domaines tels que la classification des images et l'estimation des déplacements [38, 39].

Un réseau de neurones peut apprendre comment une implémentation d'une fonction standard se comporte par rapport aux différentes entrées via la phase d'apprentissage. Dans un système complexe, un réseau de neurones peut être utilisé pour mettre en œuvre des fonctions approchées par le biais d'une co-conception logicielle-matérielle (réseaux de neurones et GPU). Les algorithmes et des fonctions traditionnels peuvent être transformés en réseaux de neurones équivalents qui présentent une précision de sortie généralement inférieure mais un avec meilleur temps d'exécution. Un des exemples les plus probants est la génération d'images 4k pour les jeux vidéo. Nvidia a montré qu'il était plus intéressant de générer des images de plus basse résolution puis d'utiliser un réseau profond pour passer en 4k plutôt que de générer directement l'image 4k.

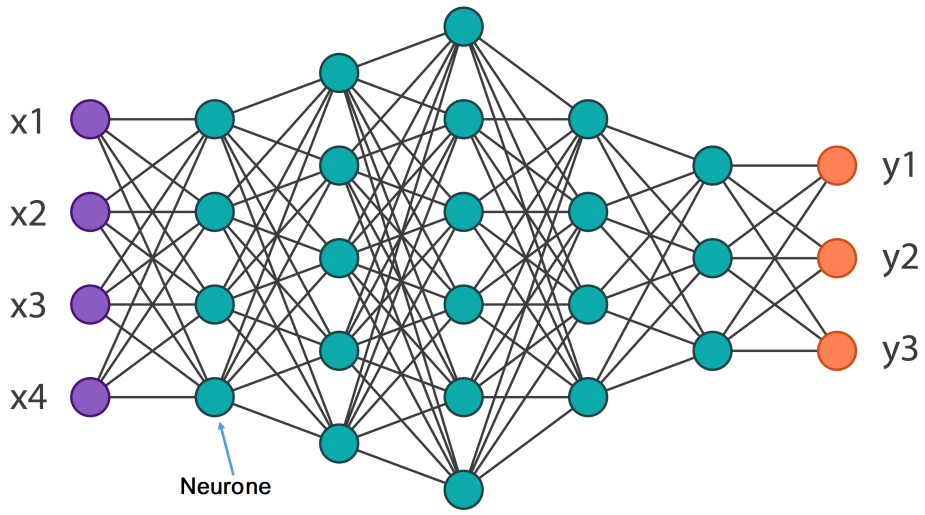


FIGURE 2.13 – Exemple d'un simple réseau de neurones, x1, x2, x3, et x4 sont des entrées et y1, y2, et y3 sont des sorties du réseau.

Dans le cadre de cette thèse, nous nous intéressons principalement aux deux dernières techniques citées ci-dessus, la réduction de la précision des données et l'utilisation des réseaux de neurones. Cette dernière est utilisée et détaillée dans la deuxième partie de ce travail (chapitre 5) pour estimer les champs de déplacements et les déformations sous-pixelliques. Quant à la réduction de la précision arithmétique, celle-ci est utilisée dans les chapitres 3 et 4 pour optimiser les résultats d'interpolation. La section suivante fait d'ailleurs un préambule à ces chapitres.

2.3 Précision arithmétique

Presque tous les systèmes numériques utilisent la représentation binaire des nombres et spécifient les règles correspondantes pour effectuer des opérations arithmétiques (addition, multiplication, etc.). La représentation exacte de la plupart des nombres réels, comme $\sqrt{2}$ par exemple, nécessite une précision infinie. Cependant, seul un nombre limité de bits peut être utilisé dans la pratique. Ainsi, dans la plupart des cas, les calculs scientifiques utilisent une approximation des valeurs exactes.

Dans l'histoire des ordinateurs, d'énormes améliorations ont été apportées à la précision des calculs. Tout d'abord, la précision des calculs a été progressivement améliorée avec l'augmentation du nombre de bits alloués à la représentation des nombres. Ceci a été rendu possible grâce à l'évolution technologique et à la miniaturisation, permettant une croissance spectaculaire du nombre d'éléments logiques de base intégrés dans les circuits de calcul. Ensuite afin de pallier aux fortes dynamiques utilisées dans certains domaines scientifiques (en Astrophysique, les calculs peuvent couvrirent 60 décades), il a été nécessaire d'introduire de nouveau paradigme telle la virgule flottante permettant de garder de grande précision.

Aujourd'hui, les processeurs polyvalents à base de silicium intègrent principalement des unités de calcul en virgule flottante sur 64 bits, ce qui devenu une norme en matière de calcul de haute précision.

Toutefois, dans la plupart des applications, des approximations arithmétiques de précision limitée sont suffisantes. Dans ce qui suit, nous rappelons le principe des deux approximations les plus utilisées, à savoir les représentations en virgule flottante et en virgule fixe.

2.3.1 Virgule flottante

L'arithmétique en virgule flottante est largement utilisée lorsqu'une grande précision est requise. Elle représente un nombre réel par un signe (S), un exposant (E), et une mantisse (M) dont la valeur associée à un nombre réel x dans la représentation en virgule flottante est donnée par l'expression suivante :

$$x = (-1)^S \times M \times 2^E.$$

La mantisse détermine la précision du nombre représenté et l'exposant définit la puissance de 2 donc le décalage.

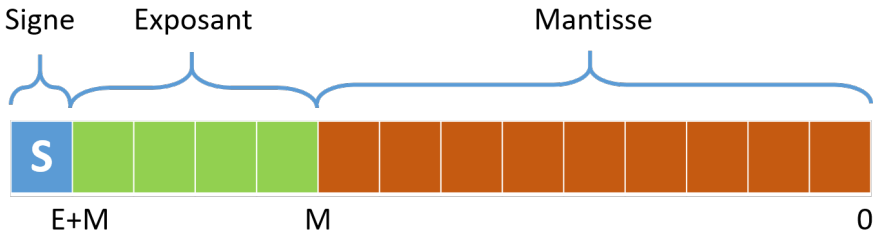


FIGURE 2.14 – Représentation en virgule flottante.

Étant donné que plusieurs couples de mantisse et d'exposant peuvent être utilisés pour représenter une même valeur, un format standard en virgule flottante a été introduit pour lever ces ambiguïtés. Aujourd'hui, la norme IEEE pour l'arithmétique binaire en virgule flottante (IEEE 745) est utilisée dans la plupart des processeurs pour assurer la portabilité des logiciels de calcul.

Cette norme définit le type de données en simple précision sur 32 bits et le type de données en double précision sur 64 bits. Plus récemment, le format en demi-précision sur 16 bits a été aussi proposé. Le tableau 2.3 montre la répartition des bits des différents types représentés par la norme IEEE 754. Les termes M et E représentent respectivement la largeur de la mantisse et l'exposant.

TABLE 2.3 – Allocation de bits dans le codage binaire en virgule flottante IEEE 754. Les chiffres entre crochets représentent la position des bits.

Représentation	Signe	E	M	Plage de E	Plus grand nombre
Demi-précision	1 [15]	5 [14-10]	10 [9-0]	[-14, 15]	$6.5 \cdots \times 10^4$
Simple précision	1 [31]	8 [30-23]	23 [22-0]	[-126, 127]	$3.4 \cdots \times 10^{38}$
Double précision	1 [63]	11 [62-52]	52 [51-0]	[-1022, 1023]	$1.8 \cdots \times 10^{308}$

2.3.2 Virgule fixe

2.3.2.1 Représentation en virgule fixe

La représentation en virgule fixe est un moyen de coder des nombres réels avec une virgule binaire virtuelle située entre deux emplacements de bits, comme le montre la figure 2.15. Un nombre représenté en virgule fixe est constitué d'une partie entière (à gauche de la virgule binaire) et d'une partie fractionnaire (à droite de la virgule binaire). La représentation en virgule fixe code les nombres avec un nombre de bits b donné, en utilisant 1 bit de signe (via le complément à deux), m bits pour la partie entière, et n bits pour la partie fractionnaire (b=1+m+n). Dans sa version la plus basique, tous les nombres sont codés avec le même nombre de bits, qu'il s'agisse de nombres entiers ou fractionnaires. Cela signifie que la position de la virgule sera identique pour tous les nombres représentés.

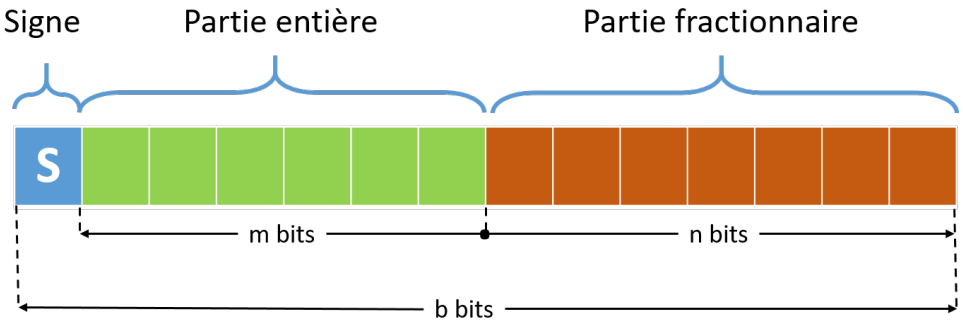


FIGURE 2.15 – Représentation en virgule fixe.

Dans cette représentation, chaque bit des parties entière et fractionnaire correspond à une puissance de 2. Intuitivement, les bits de la partie entière fournissent la représentation du nombre avec des puissances positives et les bits de la partie fractionnaire fournissent la représentation du nombre avec des puissances négatives de 2. Le bit de signe S égale 1 si la valeur représentée est une valeur négative et 0 si elle est positive.

Les valeurs maximales et minimales qui peuvent être représentées dépendent de l'emplacement de la virgule binaire et du nombre de bits utilisés dans la représentation en virgule fixe. La conversion décimale correspondant à une représentation *virgule fixe* est donné par l'équation (2.1).

$$x = -2^m S + \sum_{i=-n}^{m-1} b_i.2^i \tag{2.1}$$

Ainsi, la largeur des mots dans le cas d'une représentation en virgule fixe doit correspondre à un compromis entre dynamique et précision. En effet d'une part, l'augmentation de la largeur des mots entiers entraînera l'extension de la dynamique $D = [-2^m, 2^m - 2^{-n}]$ et d'autre part, "l'augmentation" de la partie fractionnaire du mot diminuera l'erreur de quantification $q = 2^{-n}$ et améliorera donc la précision.

2.3.2.2 Règles de l'arithmétique en virgule fixe

— **Addition/Soustraction :** L'arithmétique en virgule fixe est entièrement basée sur l'arithmétique des nombres entiers. Dans ce paragraphe, nous allons considérer l'addition (respectivement la soustraction) de deux nombres signés x et y avec un nombre de bits total respectivement de b_x et b_y, des largeurs de partie fractionnaire respectivement de n_x et n_y, et des largeurs de partie entière respectivement de $m_x = b_x - n_x$ et $m_y = b_y - n_y$.
Pour éviter le débordement, la sortie z de l'addition/soustraction de x et y doit respecter l'équation suivante :

$$m_z = max(m_x, m_y) + 1 \tag{2.2}$$

De plus, une addition/soustraction précise doit également respecter :

$$n_z = max(n_x, n_y) \tag{2.3}$$

— **Multiplication :** De manière similaire, la multiplication est aussi basée sur l'arithmétique des nombres entiers. En considérant la multiplication de deux nombres x et y. Le résultat z doit respecter l'équation (2.4) pour éviter le débordement de sa partie entière.

$$m_z = m_x + m_y \tag{2.4}$$

De plus,afin d'éviter toute perte de précision, la partie fractionnaire doit également respecter :

$$d_z = d_x + d_y \tag{2.5}$$

Ainsi, dans la multiplication de deux nombres codés en b-bits, un résultat de $2b$-bits sera renvoyé. Par conséquent, par rapport à l'addition où seul un bit supplémentaire est nécessaire, la multiplication est une source potentielle de besoins élevés en ressources de calcul. Cela justifie sans aucun doute la nécessité de quantifier les nombres tout au long des calculs afin de limiter cette contrainte.
La multiplication par une puissance de 2 est un cas particulier car elle peut être transformée en un décalage consistant simplement à translater les bits vers la gauche ce qui n'implique aucune opération physique pendant l'exécution. Il en sera de même pour la division par une puissance de 2 avec un décalage vers la droite.

2.3.3 Bilan

La mise en œuvre de calculs numériques implique le choix d'une approximation explicite des nombres réels. Les deux options les plus populaires, l'arithmétique en virgule fixe et l'arithmétique en virgule flottante, présentent des caractéristiques généralement opposées en termes de facilité d'utilisation, de coût matériel (consommation des ressources matérielles) et de précision numérique.

L'arithmétique en virgule flottante est plus polyvalente que l'arithmétique en virgule fixe et offre une meilleure précision numérique, avec un large éventail de valeurs représentables. L'arithmétique en virgule flottante est souvent associée à des calculs de haute précision avec un temps et une énergie importants par rapport au calcul en virgule fixe. En effet, la virgule flottante est aujourd'hui la représentation des nombres réels la plus utilisée dans les CPUs et les GPUs, en raison de sa grande dynamique et de sa grande précision à toute échelle d'amplitude.

Au niveau matériel, la représentation en virgule flottante est nettement plus coûteuse que celle en virgule fixe. Plus d'énergie, de temps de calcul, et de ressources matériels sont nécessaires. Cependant, la virgule fixe est toujours choisie lorsque l'on vise une faible consommation de ressources et d'énergie.

Dans cette thèse, l'arithmétique en virgule fixe est utilisée dans les implémentations FPGA afin d'atteindre des meilleurs compromis entre consommation des ressources et précision des résultats. Dans la deuxième partie de ce manuscrit concernant les réseaux de neurones, nous utilisons l'arithmétique en virgule flottante offerte par les GPUs.

L'efficacité de la mise en œuvre d'une architecture FPGA repose sur la minimisation des ressources matérielles et le maintien d'une grande précision. Les architectures en virgule fixe sont plus efficaces que leur équivalentes en virgule flottante car elles consomment moins de ressources, mais une perte de précision est attendue. Par conséquent, il est primordial d'optimiser correctement la largeur des données afin de répondre aux spécifications requises (précision et ressources matérielles). Dans ce qui suit, nous détaillons les techniques les plus utilisées pour cette fin.

2.4 Processus d'optimisation de la représentation en virgule fixe

La programmation de l'arithmétique en virgule fixe est une tâche difficile, car les débordements et les erreurs numériques peuvent dégrader considérablement la précision. C'est pourquoi ces problèmes sont généralement résolus à un stade ultérieur de la conception. L'application est spécifiée et validée en virgule flottante, avant d'être convertie en arithmétique en virgule fixe. Dans le cadre d'une conception matérielle, la représentation en virgule fixe prend la forme d'une optimisation de la largeur des mots, où l'on essaie de réduire au minimum la consommation des ressources matérielles sous réserve d'une certaine contrainte de précision.

Grâce à un processus d'optimisation minutieux de la largeur des données en virgule fixe, le concepteur doit trouver un compromis entre coût et précision du calcul. En effet, un format en virgule fixe est déterminé par la taille de ses parties entières et fractionnaires. L'amélioration de la qualité de la représentation en virgule fixe s'articulent autour de deux étapes : analyse de la dynamique des données et analyse de précision [40, 41].

Dans ce qui suit, nous décrivons en détail ces deux principales étapes du processus d'optimisation en virgule fixe. La première étape "analyse de la dynamique", correspond à la réduction de la largeur des mots entiers et vise à déterminer le nombre minimal de bits pour la partie

entière afin d'éviter tout débordement. L'objectif est de minimiser le nombre de bits tout en se protégeant contre les débordements qui annihilent la justesse des résultats. Cette étape est basée sur l'analyse de la dynamique de chaque donnée pour évaluer les limites d'amplitude. Les différentes techniques disponibles dans la littérature pour estimer la dynamique des données sont présentés dans le paragraphe 2.4.1. La deuxième étape, "analyse de précision", correspond à la sélection de la largeur des mots fractionnaires et vise à déterminer le nombre minimal de bits de la partie fractionnaire afin d'assurer une précision de calcul suffisante. Cette étape est formulée comme un processus d'optimisation basé sur les règles de propagation d'erreur en virgule fixe. La consommation des ressources matérielles est minimisée de sorte que la précision de calcul soit supérieure à une contrainte minimale.

2.4.1 Analyse de la dynamique des données

L'analyse de la dynamique en virgule fixe repose sur l'idée de partir d'un format des variables d'entrée et d'utiliser les règles de propagation de largeur de résultats pour déterminer le format des variables intermédiaires et de sortie. Avec cette technique, les résultats sont obtenus beaucoup plus rapidement, mais ils souffrent de la croissance continue d'intervalles (surdimensionnement) dans les séquences d'opération.

Afin d'éviter le surdimensionnement, il est possible d'utiliser deux grandes familles de techniques à savoir : les approches basées sur la simulation, et les approches analytiques basées sur l'arithmétique d'intervalles.

Les approches basées sur la simulation permettent d'estimer la dynamique d'une variable à partir de ses caractéristiques déterminées par des simulations en virgule flottante. L'estimation la plus simple de la dynamique consiste à déterminer la valeur absolue maximale des échantillons obtenus lors de la simulation. Cependant, la qualité de l'estimation dépend du choix des entrées et du nombre total d'échantillons. Un nombre élevé de simulations est nécessaire pour couvrir un ensemble significatif de valeurs possibles d'entrées. De plus, cela implique des temps de simulation très longs. Les approches basées sur la simulation déterminent les limites à partir d'un ensemble de résultats de simulation pouvant réduire le surdimensionnement, mais ne pouvant pas garantir l'absence de débordement [42, 43, 40, 41].

De manière corollaire, l'objectif des approches analytiques basées sur les intervalles est de déterminer une expression analytique de la métrique d'erreur due à la représentation en virgule fixe. La mesure d'erreur dépend de la largeur des mots des différentes données à l'intérieur de l'algorithme. Pour cela l'intervalle de la sortie est déterminé en propageant l'intervalle d'entrée vers la sortie. L'intervalle de chaque valeur est déterminé avec une estimation du pire cas afin d'allouer suffisamment de bits [44, 45, 46, 40, 41]. Le principal avantage de cette approche est la garantie de l'absence de débordement. Le temps nécessaire pour générer cette fonction analytique peut être plus ou moins important, mais ce processus n'est réalisé qu'une seule fois. Le principal inconvénient de ces approches analytiques est qu'elles ne prennent pas en charge tous les types de systèmes, à savoir les systèmes récursifs.

Pour les systèmes critiques, la largeur de mot de la partie entière doit couvrir toute la gamme des valeurs possibles. Dans ce cas, les limites doivent être déterminées par des techniques qui garantissent l'absence de débordement et permettent de certifier la dynamique des données. Pour cela, les différentes méthodes analytiques basées sur les intervalles permettant de fournir des limites certifiées de la dynamique des signaux sont détaillées ci-après.

2.4.1.1 Analyse de la dynamique par l'arithmétique d'intervalles

L'arithmétique d'intervalles est une méthode de limitation des valeurs de calculs basée sur les intervalles des entrées. Elle est l'une des techniques fournissant des limites certifiées aux incertitudes des systèmes complexes [47, 48]. En arithmétique d'intervalle, une variable x est remplacée par son intervalle $[a, b]$. En définissant correctement ces intervalles, on peut calculer efficacement la dynamique pour la somme, le produit et d'autres opérations sur les intervalles. Par exemple, les règles pour les opérations de base sont données par :

$$x \in [a,b] \Longrightarrow (-x) \in -[a,b] = [-b,-a]$$
$$x \in [a,b] \;\; et \;\; y \in [c,d] \Longrightarrow (x+y) \in [a,b] + [c,d] = [a+c, b+d]$$
$$x \in [a,b] \;\; et \;\; y \in [c,d] \Longrightarrow (x-y) \in [a,b] - [c,d] = [a-d, b-c]$$
$$x \in [a,b] \;\; et \;\; y \in [c,d] \Longrightarrow (x \times y) \in [a,b] \times [c,d] = [min(ac,bc,ad,bd), max(ac,bc,ad,bd)]$$

L'arithmétique d'intervalles garantit l'absence de débordement dans l'algorithme si les valeurs des données d'entrée se trouvent dans le domaine de définition utilisé. Cependant, cette méthode ne tient pas compte des corrélations entre les opérandes d'entrée. Elle considère que tous les signaux sont indépendants et peuvent prendre n'importe quelle valeur dans leur intervalle. Mais toutes les valeurs de l'intervalle ne sont pas vraiment possibles s'il existe une corrélation entre les opérandes. De plus, l'estimation est basée sur une analyse qui tient compte du scénario le plus défavorable, ce qui pourrait entraîner une surestimation des limites [40, 41].

2.4.1.2 Analyse de dynamique par l'arithmétique multi-intervalles

L'arithmétique multi-intervalles a été proposée comme une amélioration de la méthode basée sur l'arithmétique d'intervalles. Cette méthode consiste à diviser les intervalles d'entrée en un ensemble de sous-intervalles plus petits couvrant toute la plage de valeurs. Les opérations sont effectuées sur des intervalles plus petits et la dynamique totale est déterminée en fusionnant tous les intervalles intermédiaires. Ainsi, les dimensions des résultats finaux sont réduites par rapport à la méthode basée sur l'arithmétique d'intervalles traditionnelle [49, 50]. Comme l'arithmétique d'intervalles, l'arithmétique multi-intervalles garantit l'absence de débordement et ne résout pas le problème de corrélation des données.

2.4.1.3 Analyse de la dynamique par l'arithmétique affine

L'arithmétique affine est proposée pour résoudre les problèmes dus à la dépendance des données inhérente aux méthodes précédentes. Pour résoudre le problème de dépendance de données, une variable x est représentée comme une expression affine de valeurs numériques x_i et de valeurs symboliques d'erreur e_i comme suit :

$$x = x_0 + x_1.e_1 + x_2.e_2 + ... + x_n.e_n \tag{2.6}$$

Chaque symbole de d'erreur e_i $(1 \leq i \leq n)$ est une variable réelle dont la valeur est inconnue, comprise entre [-1, +1] et indépendante des autres symboles de d'erreur. Le coefficient x0 = (a+b)/2 est appelé la valeur centrale de la forme affine. Les coefficients x_i $(1 \leq i \leq n)$ sont appelés les déviations partielles associées aux symboles d'erreur e_i $(1 \leq i \leq n)$ en x. Le nombre n de symboles de bruit dépend de la forme affine. Différentes formes affines avec un nombre différent de symboles de bruit peuvent être utilisés. Cette méthode présente une légère amélioration par rapport à l'arithmétique d'intervalles [51, 52].

La propriété la plus importante de la méthode basée sur l'arithmétique affine est qu'un coefficient d'erreur peut être partagé entre les variables, ce qui permet de suivre les corrélations ou les dépendances entre elles. Toutefois, la méthode risque de conduire à un trop grand nombre de termes. L'arithmétique affine peut modéliser avec précision la propagation de la dynamique dans un système linéaire non récursif, mais la non-linéarité ou la présence de boucles donne lieu à des approximations. Il en résulte une perte d'informations et des limites surdimensionnées.

Dans la première partie de ce travail, nous nous sommes intéressés à la mise en œuvre d'une architecture efficace pour effectuer une interpolation 2D hétérogène. Sachant que l'algorithme d'interpolation n'est pas récursif et que ses entrées ne sont pas corrélées, l'arithmétique d'intervalles a été adoptée pour garantir à la fois l'absence de tout débordement et la simplicité d'utilisation. Dans ce qui suit, nous montrons comment utiliser cette technique afin de définir la largeur de la partie fractionnaire offrant le meilleur compromis entre précision et coût matériel.

2.4.2 Analyse de précision

L'évaluation de la précision analytique consiste à prédire l'influence des erreurs de quantification sur le rendement du système. Il est donc nécessaire de vérifier que le comportement de l'algorithme utilisant l'arithmétique en virgule fixe soit modifié dans une limite raisonnable. Ainsi, la partie fractionnaire est déterminée sur la base d'un compromis entre précision nécessaire et coût matériel afin de limiter la croissance des largeurs de bus.

Les quantifications peuvent être exprimées explicitement sous forme d'opérations supplémentaires, comme le montre la figure 2.16. Il convient de noter que si les règles de propagation en virgule fixe ne sont pas respectées, une erreur E_{xy} sera rajoutée. Dans l'évaluation de précision, chaque quantification est remplacée par l'addition du signal original avec son erreur de

quantification associée.

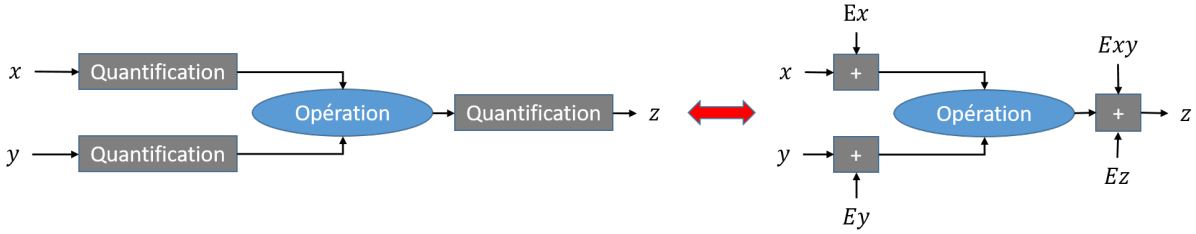


FIGURE 2.16 – Quantification en virgule fixe.

Dans ce cadre, nous analysons les erreurs de quantification des opérateurs arithmétiques de base tels que ceux de l'addition, la soustraction, et la multiplication. Pour chaque opérateur, nous expliquons comment l'erreur de quantification se propage en utilisant l'arithmétique d'intervalles décrite dans la section précédente. A cette fin, nous analysons deux quantités $F(z)$ et E_z pour chaque variable z de la conception, en considérant que les règles de propagation sont respectées ($E_{xy} = 0$).

$$z = F(z) + E_z \tag{2.7}$$

Où $F(z)$ est la dynamique de la variable z en virgule fixe, et E_z est la plage d'erreur de quantification associée au calcul de $F(z)$.

2.4.2.1 Addition/soustraction

En arithmétique virgule fixe, les deux opérandes doivent être alignées dans le même format, c'est-à-dire qu'elles doivent partager les mêmes parties entières et fractionnaires. Si ce n'est pas le cas, alors les opérations d'alignement doivent être appliquées aux opérandes avant le nœud d'addition. Ces alignements sont effectués par des simples opérateurs de décalage.

La dynamique de sortie d'une addition ou une soustraction est définie par :

$$F(z) = F(x) \;\; op \;\; F(y), \quad op \in \{+, -\}$$

Quant à l'erreur :

$$\begin{aligned} E_{addition/soustraction} &= z - F(z) \\ &= (x \;\; op \;\; y) - (F(x) \;\; op \;\; F(y)) \\ &= (x - F(x)) \;\; op \;\; (y - F(y)) \\ &= E_x \;\; op \;\; E_y \end{aligned}$$

C'est-à-dire l'erreur de quantification d'une opération d'addition ou de soustraction est respectivement la somme ou la différence des erreurs de quantification des opérandes. Si on consi-

dère la quantification après l'opération d'addition ou de soustraction, une erreur de quantification E_{xy} à ce stade doit être rajoutée, comme indiqué dans la figure 2.16.

2.4.2.2 Multiplication

Contrairement à l'addition et à la soustraction, il n'y a pas de contraintes sur les formats des opérandes. En outre, en règle générale, le produit de x et y donne une variable en virgule fixe avec une partie entière et une partie fractionnaire constituées respectivement de la somme des deux parties entière et fractionnaire des opérandes. Pour la dynamique de sortie d'une multiplication, il vient :

$$F(z) = F(x).F(y)$$

Ce qui permet d'exprimer le terme d'erreur tel que :

$$\begin{aligned} E_z &= z - F(z) \\ &= (x.y) - F(z) \\ &= (F(x) + E_x).(F(y) + E_y) - F(z) \\ &= F(x).F(y) - F(z) + F(x).E_y + F(y).E_x + E_x.E_y \\ &= F(x).E_y + F(y).E_x + E_x.E_y \end{aligned}$$

Dans le cas de la multiplication, l'erreur de quantification E_z ne dépend pas seulement des erreurs des opérandes, E_x et E_y, mais aussi de leur dynamique $F(x)$ et $F(y)$. Cela montre que la limitation précise de ces dynamiques conduit à de faibles plages d'erreur. En général, le pire cas est considéré pour déterminer l'erreur maximale.

Comme dans le cas d'addition/soustraction, une erreur E_{xy} est rajoutée au cas où une autre étape de quantification est nécessaire afin d'adapter le résultat de multiplication au format de la sortie souhaitée.

La détermination de $F(z)$ et E_z pour chaque nœud d'une représentation arborescente d'une expression arithmétique ne nécessite rien de plus que leur propagation en une traversée ascendante, comme le montre la figure 2.17.

Cette technique a été utilisée dans les deux chapitres suivants dans le cadre d'implémentations FPGA d'une interpolation hétérogène, afin d'atteindre un bon compromis entre précision et consommation des ressources matérielles.

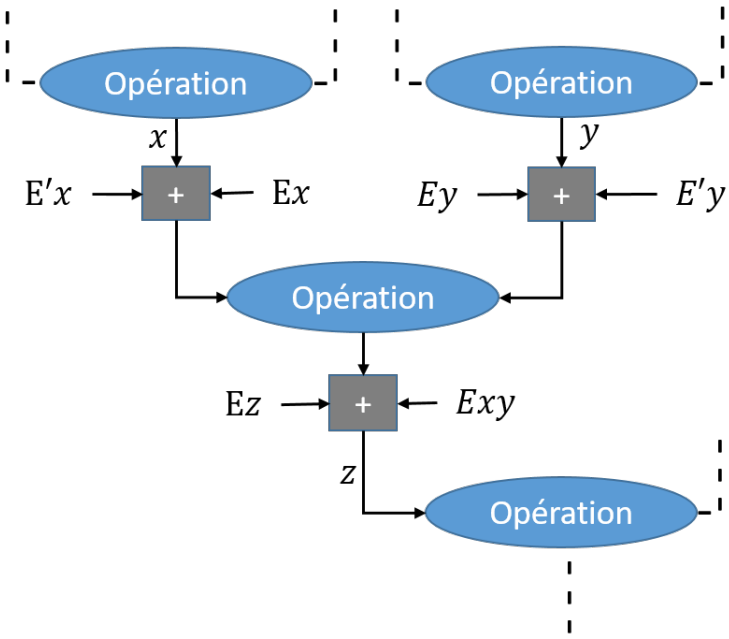


FIGURE 2.17 – Propagation de l'erreur de quantification.

2.5 Conclusion

L'arithmétique en virgule fixe est un paradigme classique lorsqu'il s'agit de systèmes à faible consommation de ressources matérielles et d'énergie. Dans ce chapitre, nous avons d'abord cité les techniques les plus utilisées dans la littérature pour réduire la complexité de calcul et la consommation des ressources matérielles, dont la représentation arithmétique en virgule fixe. Ensuite, un processus d'optimisation de la largeur des données en virgule fixe a été détaillé. Ce processus est basé sur deux étapes, l'analyse de la dynamique des données et l'analyse de précision. L'étape d'analyse de la dynamique des données permet de définir la partie entière. La seconde étape consiste à déterminer la largeur de la partie fractionnaire permettant de respecter la contrainte de précision. Les approches analytiques basées sur les intervalles ont été utilisées dans ces deux étapes car elles certifient l'absence de débordement.

Bibliographie

[1] L. B. Kish. End of moore's law : thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305(3-4) :144–149, 2002.

[2] T. N. Jackson. Beyond moore's law. *Nature materials*, 4(8) :581–582, 2005.

[3] L. B. Kish. Moore's law and the energy requirement of computing versus performance. *IEE Proceedings-Circuits, Devices and Systems*, 151(2) :190–194, 2004.

[4] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute-intensive applications with GPUs and FPGAs. In *2008 Symposium on Application Specific Processors*, pages 101–107. IEEE, 2008.

[5] S. Sarkar, T. Majumder, A. Kalyanaraman, and P. P. Pande. Hardware accelerators for biocomputing : A survey. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 3789–3792. IEEE, 2010.

[6] A. HajiRassouliha, A. J. Taberner, M. P. Nash, and P. M. F. Nielsen. Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms. *Signal Processing : Image Communication*, 68 :101–119, 2018.

[7] R. Spurzem, P. Berczik, G. Marcus, A. Kugel, G. Lienhart, I. Berentzen, R. Männer, R. Klessen, and R. Banerjee. Accelerating astrophysical particle simulations with programmable hardware (FPGA and GPU). *Computer Science-Research and Development*, 23(3-4) :231–239, 2009.

[8] C. Gonzalez, S. Sánchez, A. Paz, J. Resano, D. Mozos, and A. Plaza. Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing. *Integration*, 46(2) :89–103, 2013.

[9] Introduction to FPGA design with vivado high-level synthesis. `https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf`.

[10] xilinx, what is an FPGA ? `https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html`.

[11] Intel, cyclone v device overview. `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf`.

[12] Xilinx documentation, 7 series FPGAs configurable logic block. `https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf`.

[13] Intel stratix 10 nx FPGA. `https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/stratix-10-nx-technology-brief.pdf`.

[14] Xilinx, bringing the benefits of cortex-m processors to FPGA. https://www.xilinx.com/publications/events/developer-forum/2018-frankfurt/bringing-the-benefits-of-cortex-m-processors-to-fpga.pdf.

[15] Xilinx, ultrascale architecture and product data sheet : Overview. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf.

[16] D. R. Kaeli. *Heterogeneous computing with OpenCL 2.0*. Elsevier, 2015.

[17] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. S. Seo, and Y. Cao. Throughput-optimized opencl-based FPGA accelerator for large-scale convolutional neural networks. FPGA '16, page 16–25, New York, NY, USA, 2016. Association for Computing Machinery.

[18] Nvidia ai platform for developers. https://developer.nvidia.com/deep-learning.

[19] Nvidia tesla v100 GPU architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[20] Nvidia, nvlink and nvswitch. https://www.nvidia.com/en-us/data-center/nvlink/.

[21] Nvidia, tensor cores. https://www.nvidia.com/en-us/data-center/tensor-cores/.

[22] Nvidia developer, about cuda. https://developer.nvidia.com/about-cuda.

[23] Nvidia developer, nvidia cudnn. https://developer.nvidia.com/cudnn.

[24] Intel agilex FPGAs and socs, advanced information brief. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/agilex/ag-overview.pdf.

[25] Intel FPGA product catalog, version 20.3. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/sg/product-catalog.pdf.

[26] Intel programmable acceleration card (pac) with intel stratix 10 sx FPGA. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/po/product-brief-intel-pac-with-intel-stratix-10-sx.pdf.

[27] Nvidia a100 tensor core GPU architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf.

[28] Nvidia tesla p100. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[29] Xilinx all programmable devices : A superior platform for compute-intensive systems. https://www.xilinx.com/support/documentation/white_papers/wp492-compute-intensive-sys.pdf.

[30] BERTEN Digital Signal Processing White Paper. GPU vs FPGA performance comparison. 2016.

[31] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. Performance comparison of FPGA, GPU and cpu in image processing. In *2009 international conference on field programmable logic and applications*, pages 126–131. IEEE, 2009.

[32] Wim Vanderbauwhede and Khaled Benkrid. *High-performance computing using FPGAs*, volume 3. Springer, 2013.

[33] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), March 2016.

[34] G. Rodrigues, F. Lima Kastensmidt, and A. Bosio. Survey on approximate computing and its intrinsic fault tolerance. *Electronics*, 9(4), 2020.

[35] C. C. Hsiao, S. L. Chu, and C. Y. Chen. Energy-aware hybrid precision selection framework for mobile GPUs. *Computers & Graphics*, 37(5) :431–444, 2013.

[36] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough. Precimonious : Tuning assistant for floating-point precision. In *SC'13 : Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2013.

[37] J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan. An FPGA-based fixed-point architecture for binary logarithmic computation. In *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, pages 383–388. IEEE, 2013.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[39] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet : Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[40] D. Menard, G. Caffarena, L. A. Lopez, D. Novo, and O. Sentieys. Fixed-point refinement of digital signal processing systems, 2019.

[41] D. Ménard, G. Caffarena, J. A. Lopez, D. Novo, and O. Sentieys. Analysis of finite word-length effects in fixed-point systems. In *Handbook of Signal Processing Systems*, pages 1063–1101. Springer, 2019.

[42] S. Kim and W. Sung. A floating-point to fixed-point assembly program translator for the tms 320c25. *IEEE Transactions on Circuits and Systems II : Analog and Digital Signal Processing*, 41(11) :730–739, 1994.

[43] L. De Coster. *Bit-true simulation of digital signal processing applications*. PhD thesis, PhD thesis, KU Leuven, 1999.

[44] M. L. Chang and S. Hauck. Precis : a usercentric word-length optimization tool. *IEEE Design & Test of Computers*, 22(4) :349–361, 2005.

[45] N. Doi, T. Horiyama, M. Nakanishi, and S. Kimura. Minimization of fractional wordlength on fixed-point conversion for high-level synthesis. In *ASP-DAC 2004 : Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)*, pages 80–85. IEEE, 2004.

[46] P. Banerjee, M. Haldar, A. Nayak, V. Kim, V. Saxena, S. Parkes, D. Bagchi, S. Pal, N. Tripathi, D. Zaretsky, et al. Overview of a compiler for synthesizing matlab programs onto FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3) :312–324, 2004.

[47] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to interval analysis*. SIAM, 2009.

[48] R. E. Moore. *Methods and applications of interval analysis*. SIAM, 1979.

[49] A. Benedetti and P. Perona. Bit-width optimization for configurable DSP's by multi-interval analysis. In *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers (Cat. No. 00CH37154)*, volume 1, pages 355–359. IEEE, 2000.

[50] J. A. López, C. Carreras, G. Caffarena, and O. Nieto-Taladriz. Fast characterization of the noise bounds derived from coefficient and signal quantization. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.*, volume 4, pages IV–IV. IEEE, 2003.

[51] D. Figueiredo, L. Henrique, and J. Stolfi. Affine arithmetic : concepts and applications. *Numerical Algorithms*, 37(1-4) :147–158, 2004.

[52] N. S. Nedialkov, V. Kreinovich, and S. A. Starks. Interval arithmetic, affine arithmetic, taylor series methods : why, what next ? *Numerical Algorithms*, 37(1-4) :325–336, 2004.

Chapitre 3

FPGA-based architecture for bi-cubic interpolation : the best trade-off between precision and hardware resource consumption

Préambule

Comme décrit dans le chapitre 1, l'interpolation bi-cubique présente un bon compromis entre qualité d'interpolation et consommation des ressources matérielles. L'objectif principal de ce chapitre est de fournir une implémentation matérielle de l'interpolation bi-cubique hétérogène. Par rapport à la littérature, l'architecture proposée est optimisée afin de réduire le nombre de multiplieurs et d'éviter les calculs redondants. De plus, la précision arithmétique par rapport à l'utilisation des ressources matérielles est étudiée afin de minimiser l'erreur de quantification et d'obtenir le meilleur compromis entre coût de conception et précision.

Ce chapitre a fait l'objet d'un article qui a été publié en novembre 2020 dans le "*Journal of Real-Time Image Processing*".

FPGA-based architecture for bi-cubic interpolation : the best trade-off between precision and hardware resource consumption

S. Boukhtache[1], B. Blaysat[1], M. Grédiac[1], F. Berry[1]

[1]*Institut Pascal, UMR 6602, Université Clermont-Auvergne, CNRS, SIGMA Clermont, Clermont-Ferrand, France*

Abstract

An accurate and low-cost design of the image interpolation unit is a crucial part for many real-time image processing systems. To reach this goal, bi-cubic interpolation is generally selected because it provides the best trade-off between computational complexity and interpolation quality. The aim of this paper is to study the optimal hardware implementation of heterogeneous bi-cubic interpolation. Bi-cubic algorithm is reformulated and improved for FPGA implementation. This improved algorithm avoids twelve redundant calculations and reduces the number of multipliers by 25%. Hardware precision versus resource utilization is studied in order to minimize the quantization error of hardware realization, and to obtain the best trade-off between design cost and accuracy. A compromise that reduces $33,33\%$ of bit-width utilization with a precision higher than 99.922% is reached. Besides, the proposed architecture is fully-pipelined in order to reach high operating frequency. Instantiation on Xilinx and Intel targets shows the benefit of our approach, especially in terms of hardware resource consumption.

Keywords : *Bi-cubic interpolation, FPGA, Hardware resource consumption, Precision*

3.1 Introduction

Image interpolation is one of the fundamental operations used in digital image processing. It is widely used and required by many real-time applications such as super resolution, image resizing, geometric transformation, or in multiple domains such as digital image correlation for metrological measurement purposes [1, 2]. The interpolation principle is based on the determination of an infinite resolution kernel from a set of known discrete data. According to the Nyquist reconstruction theorem, the ideal interpolation kernel of a band-limited function is the

$sinc$ function [3], but its infinite extent makes it unsuitable. Hence many other types of model approximations have been proposed in the literature with their corresponding algorithms [4]. Choosing the one which offers the best trade-off between computational complexity and interpolation quality is crucial to fulfill real-time requirements, and reach hardware efficiency for applications based on interpolation.

Among the existing interpolation algorithms, the following ones are the most commonly used. With the nearest neighbor (NN) algorithm, the interpolation kernel is a zero-order polynomial. The nearest neighbor is the simplest and fastest algorithm. However, its main drawback is to induce the highest error. Bilinear interpolation is rapid. It relies on a first-order polynomial kernel which is simple to implement. It is more accurate than the nearest neighbor but interpolation quality can still be improved. B-spline interpolation algorithm involves high computational complexity and a lot of processing time in order to achieve higher quality of interpolation. Thus, hardware implementation is difficult. Bi-cubic algorithm is performed by using a third-order polynomial function. It is of better quality than bilinear interpolation and is less complex to implement than b-spline interpolation [4, 5]. The higher the order of the interpolation kernel (the computational complexity), the better the interpolation quality. The speed is also inversely proportional to the function order. Lanczos interpolation consists in using a Look-Up-Table of a windowed-$sinc$ function [6]. Its interpolation quality depends on the discretization step (memory utilization), which is not suitable for many applications. These algorithms and their corresponding kernels type are listed in Table 3.1.

TABLE 3.1 – Kernel type of the most commonly used interpolations.

Method	NN	Bilinear	Bi-cubic	B-Spline	Lanczos
Kernel type	0-order	1st-order function	3rd-order function	nth-order function	Look-Up Table

Among all these algorithms, the bi-cubic one is considered to be the best for interpolation quality, hardware resource and real-time requirements [5]. It is also successfully used for various applications such as image scaling [7], image quality enhancement [8] and super-resolution [1].

The aim of this work is to efficiently implement bi-cubic interpolation on FPGA. This paper is organized as follows. Section 2 briefly recalls the basics of the bi-cubic interpolation algorithm. Section 3 surveys the most important hardware implementations available in the literature. The proposed architecture as well as the different improvements compared to existing architectures are detailed in Section 4. Section 5 discusses hardware resource utilization according to a requested precision. Section 5 presents the results obtained with the proposed implementation and compares them with those found in related studies.

3.2 Bi-cubic interpolation

Bi-cubic interpolation is the natural extension of the one-dimensional cubic interpolation used to estimate unknown data in two-dimensional space. The one-dimensional interpolation function can be written as follows (see also Fig. 3.1)

$$g(x) = \sum_{k=0}^{3} A_k \beta(x - x_k) = \sum_{k=0}^{3} A_k \beta(u_k) \tag{3.1}$$

where $g(x)$ is the corresponding interpolation function. A_k, $k = 0...3$, are the values to be interpolated. β is the interpolation kernel.

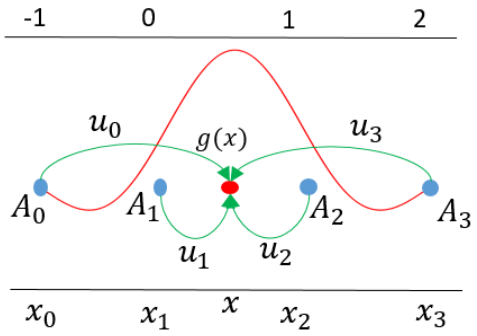


FIGURE 3.1 – Cubic interpolation

Cubic interpolation kernel requires 4 neighbor pixels. It is based on a third-degree polynomial [5]. This leads to the following expression for $\beta(u)$:

$$\beta(u) = \begin{cases} \frac{3}{2}|u|^3 - \frac{5}{2}|u|^2 + 1 & 0 \leq |u| < 1 \\ -\frac{1}{2}|u|^3 + \frac{5}{2}|u|^2 - 4|u| + 2 & 1 \leq |u| < 2 \\ 0 & elsewhere \end{cases} \tag{3.2}$$

Bi-cubic interpolation relies on 16 neighboring pixels. Two types of one-dimensional interpolations are performed in turn along the horizontal and the vertical directions, as shown in Fig. 3.2. It can be seen that the vertical interpolation is performed four times to obtain the four intermediate points (q_0, q_1, q_2, q_3), whereas the horizontal interpolation based on these intermediate points is performed once.

It is worth noting that the order between the two directions (vertical and horizontal) can be switched without any impact on the value obtained at the end of the procedure.

The bi-cubic interpolation formula is given by :

$$g(x, y) = \sum_{i=0}^{3} \left(\sum_{j=0}^{3} A_{ij} \beta(y - y_j) \right) \beta(x - x_i) \tag{3.3}$$

The previous equation can be rewritten as follows with $u_i = x - x_i$ and $v_i = y - y_i$:

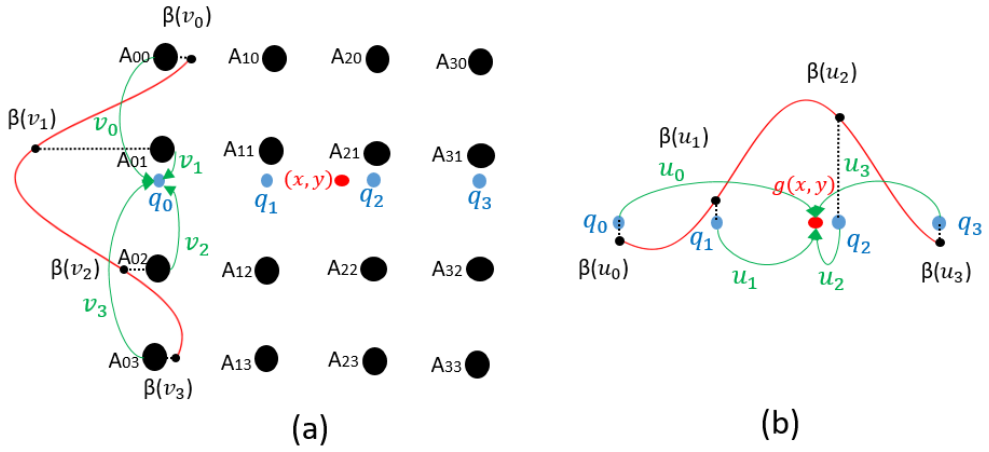


FIGURE 3.2 – Bi-cubic interpolation, (a) vertical and (b) horizontal one-dimensional interpolations.

$$\begin{aligned} g(x,y) = &\big(\beta(v_0)A_{00} + \beta(v_1)A_{10} + \beta(v_2)A_{20} + \beta(v_3)A_{30}\big)\beta(u_0) \\ &+\big(\beta(v_0)A_{01} + \beta(v_1)A_{11} + \beta(v_2)A_{21} + \beta(v_3)A_{31}\big)\beta(u_1) \\ &+\big(\beta(v_0)A_{02} + \beta(v_1)A_{12} + \beta(v_2)A_{22} + \beta(v_3)A_{32}\big)\beta(u_2) \\ &+\big(\beta(v_0)A_{03} + \beta(v_1)A_{13} + \beta(v_2)A_{23} + \beta(v_3)A_{33}\big)\beta(u_3) \end{aligned} \tag{3.4}$$

where $\beta(u_k)$ and $\beta(v_k)$ are the coefficients used for the horizontal and vertical interpolation, respectively. They are calculated by using Eq. (3.2).

3.3 Previous studies

This section is a brief overview of the most important hardware implementations of the bi-cubic interpolation algorithm. These hardware implementations can be split into two categories :

— Homogeneous architecture is suited when interpolation is performed at the same location for all windows of the image, such as in image scaling.

— Heterogeneous architecture interpolates at different locations, as depicted in Fig. 3.3. It is suitable for applications such as image rotations and displacement field measurement.

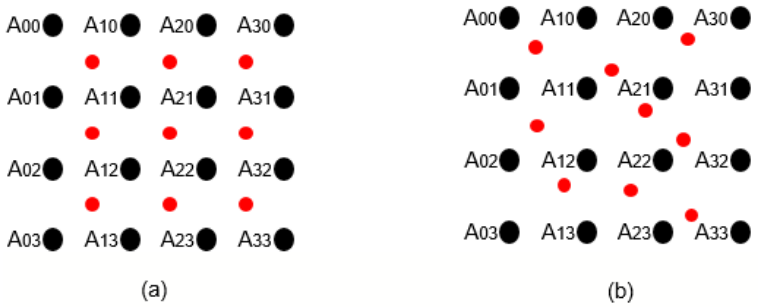


FIGURE 3.3 – (a) Homogeneous and (b) heterogeneous interpolation

Heterogeneous architecture is more complex and needs more hardware resources. Indeed,

it consists in implementing five 1D-cubic interpolations instead of two 1D-cubic interpolations in the homogeneous case. In the homogeneous architecture, the intermediate points (q_0, q_1, q_2) are obtained directly from the results of the previous window since the interpolation is always at the same location. Consequently, only point (q_3) is recalculated. In the heterogeneous case, all the intermediate points (q_0, q_1, q_2, q_3) are recalculated for each window of the image. The most important bi-cubic architectures are classified in Table 3.2.

TABLE 3.2 – Classification of the most important bi-cubic architectures available in the literature.

Work	Architecture
Homogeneous	
Lin *et al.* [11]	VLSI design of bi-cubic interpolation. FPGA requires 10 multipliers and 437 LBs. Maximum operating frequency (Fmax) is not mentioned. ASIC (0.13 μm) requires 30k gates and operates at 279 MHz.
Pan *et al.* [12]	Applies Laplacian filter to bi-cubic interpolation in order to enhance the quality. Bi-cubic architecture similar as in [11].
Lin *et al.* [13]	The third-order cubic kernel is approximated by a piecewise linear kernel. Virtex-II FPGA : requires 8 multipliers and 376 LBs, it operates at Fmax of 104.3 MHz. ASIC (0.13 μm CMOS) requires 26k gates and operates at 267 MHz.
Wang *et al.* [14]	Employed the Farrow structure. FPGA : requires 6 multipliers and operates at Fmax of 130 MHz. ASIC (0.13 μm) requires 16k gates and operates at 312 MHz.
Gour, *et al.* [15]	Based on a four piecewise linear approximation of the bi-cubic kernel. The architecture requires 8 multipliers. Hardware implementation not realized.
Heterogeneous	
Nuno *et al.* [9]	The bi-cubic formula (3.4) is implemented in 4 parallel sub-blocks. Virtex-II pro FPGA : requires 32 multipliers and 890 LBs, it operates at Fmax of 100 MHz.
Zhang *et al.* [10]	Divides the interval between two neighboring pixels into eight subintervals. The coefficients are computed and then stored to be used each time the interpolation is performed. The architecture requires 20 multipliers. The authors did not realize a hardware implementation.

The heterogeneous concept is the general one. It can be used in any application. On the contrary, the homogeneous one is limited to a special application. The heterogeneous case is also more challenging than the homogeneous one. Its architecture is more complex and requires more hardware resources. The heterogeneous case is considered in this study for all these reasons.

Nuno *et al.* [9] decomposed the bi-cubic algorithm into three main blocks. The first one generates the interpolation coefficients. The second one performs bi-cubic interpolation and the third one is the control unit. The second block corresponds thus to the core of the algorithm. The bi-cubic formula (3.4) is implemented in 4 parallel sub-blocks, each sub-block representing one of the four lines of this equation. The design was implemented on a Virtex-II Pro FPGA. The maximum operating frequency is observed to be equal to 100 MHz. 32 multipliers and 890 LBs are necessary in this case.

Zhang *et al.* [10] divided the interval between two neighboring pixels into eight subinter-

vals. The coefficients for each of these subintervals are computed offline and then stored to be used each time the interpolation is performed. The accuracy of this method depends on the number of subintervals. A drawback is therefore that this architecture is counterbalanced by a poor interpolation quality and high memory utilization. Unfortunately, the cost of the hardware resource of the proposed design is not given by the authors.

In this paper, we aim to provide an efficient real-time implementation of heterogeneous bi-cubic interpolation, which presents better performance compared to previous studies on this problem [9, 10].

3.4 Proposed architecture

This work aims at providing a better balanced (resources vs precision) implementation of heterogeneous bi-cubic interpolation dedicated to a wider range of applications. To tackle this problem, we propose a reformulation of Eq. (3.4) in order to reduce the number of multipliers. By applying a factorization, Eq. (3.4) can be then rewritten as follows :

$$\begin{aligned} g(x,y) =& t_{00} + (t_{01} + (t_{02} + t_{03} \times y) \times y) \times y) \\ &+ \Big[(t_{10} + (t_{11} + (t_{12} + t_{13} \times y) \times y) \times y) \\ &+ \big[((t_{20} + (t_{21} + (t_{22} + t_{23} \times y) \times y) \times y) \\ &+ ((t_{30} + (t_{31} + (t_{32} + t_{33} \times y) \times y) \times y)) \times x) \times x \big] \Big] \times x \end{aligned} \tag{3.5}$$

where $t_{ij} = f(A_{00}, A_{01}, ..., A_{33})$ is given in Table 3.3.

TABLE 3.3 – Interpolation coefficients.

Coefficient	Expression
t_{00}	A_{11}
t_{01}	$-0.5A_{10} + 0.5A_{12}$
t_{02}	$A_{10} - 2.5A_{11} + 2A_{12} - 0.5A_{13}$
t_{03}	$-0.5A_{10} + 1.5A_{11} - 1.5A_{12} + 0.5A_{13}$
t_{10}	$-0.5A_{01} + 0.5A_{21}$
t_{11}	$0.25A_{00} - 0.25A_{02} - 0.25A_{20} + 0.25A_{22}$
t_{12}	$-0.5A_{00} + 1.25A_{01} - A_{02} + 0.25A_{03} + 0.5A_{20} - 1.25A_{21} + A_{22} - 0.25A_{23}$
t_{13}	$0.25A_{00} - 0.75A_{01} + 0.75A_{02} - 0.25A_{03} - 0.25A_{20} + 0.75A_{21} - 0.75A_{22} + 0.25A_{23}$
t_{20}	$A_{01} - 2.5A_{11} + 2A_{21} - 0.5A_{31}$
t_{21}	$-0.5A_{00} + 0.5A_{02} + 1.25A_{10} - 1.25A_{12} - A_{20} + A_{22} + 0.25A_{30} - 0.25A_{32}$
t_{22}	$A_{00} - 2.5A_{01} + 2A_{02} - 0.5A_{03} - 2.5A_{10} + 6.25A_{11}$ $-5A_{12} + 1.25A_{13} + 2A_{20} - 5A_{21} + 4A_{22} - A_{23} - 0.5A_{30} + 1.25A_{31} - A_{32} + 0.25A_{33}$
t_{23}	$-0.5A_{00} + 1.5A_{01} - 1.5A_{02} + 0.5A_{03} + 1.25A_{10} - 3.75A_{11} + 3.75A_{12}$ $-1.25A_{13} - A_{20} + 3A_{21} - 3A_{22} + A_{23} + 0.25A_{30} - 0.75A_{31} + 0.75A_{32} - 0.25A_{33}$
t_{30}	$-0.5A_{01} + 1.5A_{11} - 1.5A_{21} + 0.5A_{31}$
t_{31}	$0.25A_{00} - 0.25A_{02} - 0.75A_{10} + 0.75A_{12} + 0.75A_{20} - 0.75A_{22} - 0.25A_{30} + 0.25A_{32}$
t_{32}	$-0.5A_{00} + 1.25A_{01} - A_{02} + 0.25A_{03} + 1.5A_{10} - 3.75A_{11} + 3A_{12} - 0.75A_{13}$ $-1.5A_{20} + 3.75A_{21} - 3A_{22} + 0.75A_{23} + 0.5A_{30} - 1.25A_{31} + A_{32} - 0.25A_{33}$
t_{33}	$0.25A_{00} - 0.75A_{01} + 0.75A_{02} - 0.25A_{03} - 0.75A_{10} + 2.25A_{11} - 2.25A_{12} + 0.75A_{13}$ $+0.75A_{20} - 2.25A_{21} + 2.25A_{22} - 0.75A_{23} - 0.25A_{30} + 0.75A_{31} - 0.75A_{32} + 0.25A_{33}$

With this reformulation, the number of multipliers drops from 20 to 15 and the calculation of each t_{ij} is only a combination of additions and shiftings (power of 2). By contrast, the $\beta(u_i)$ and $\beta(v_j)$ coefficients involved in Eq. (3.4) are calculated with the third-order polynomial function (3.2).

The architecture performing the bi-cubic interpolation based on Eq. (3.5) is split into 3 blocks, see Fig. 3.4. First, the 16 neighbor pixels $[A_{00}, \ldots, A_{33}]$ required by the bi-cubic interpolation are provided by the "4 $\times$ 4 window extractor" block. Then, these latter are used by the "Coefficient generator" block to compute all the t_{ij} coefficients given in Table 3.3. Finally, the block named "Interpolation block" corresponds to Eq. (3.5). Its output is the result of the interpolation. These blocks are detailed in the following.

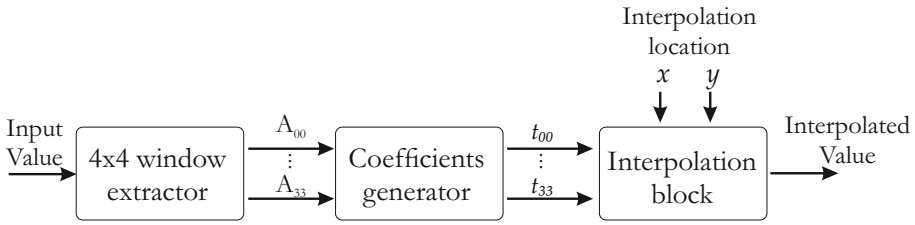


FIGURE 3.4 – Block diagram of bi-cubic interpolation.

The "4×4 window extractor" block is a pipelined operator that receives serial pixels stream from the camera and extracts the 16 pixels required for the interpolation. It is a classic block made by a shift buffer storing three lines and four pixels. This component must be instantiated on on-chip memories to benefit from 16 simultaneous and rapid accesses to the required neighbors. Due to the pipelined structure, a new neighborhood $[A_{00}, \ldots, A_{33}]$ is extracted on the fly at each clock pixel cycle. It is worth to know that using off-chip memories impairs the performance of the architecture [16, 17].

The "Coefficient generator" block determines all the t_{ij} coefficients involved in Eq. (3.5). It is based on the 16 neighbor pixels $[A_{00}, \ldots, A_{33}]$ extracted by the first "4 $\times$ 4 window extractor" block. All the t_{ij} coefficients are calculated by using combinations of additions and shifts (power of 2). For instance, $0.75 \times A_{ij}$ is calculated as shift(A_{ij},1) + shift(A_{ij},2). Furthermore, the architecture of this block is optimized by considering recurrent calculations. For instance, $A_{10} - 2.5A_{11} + 2A_{12} - 0.5A_{13}$ is used to calculate t_{02}, t_{22}, and t_{32}, see Table 3.3. Eleven other polynomials are found in order to simplify the architecture.

Finally, the "Interpolation block" is made of 5 sub-blocks named SUB_COMP, see Fig. 3.5). Each one is fully-pipelined with a latency of 6 cycles, as illustrated by its internal structure shown in Fig. 3.6.

First, 4 parallelized SUB_COMPs calculating $(t_{i0} + (t_{i1} + (t_{i2} + t_{i3} \times y) \times y) \times y)$ are used to perform the one-dimensional interpolations in the first direction. Then, one SUB_COMP calculating $(out_1 + (out_2 + (out_3 + out_4 \times x) \times x) \times x)$ is used to perform the interpolation in the second direction, where, out_1, out_2, out_3, and out_4 are the results obtained by the 4 parallelized SUB_COMPs.

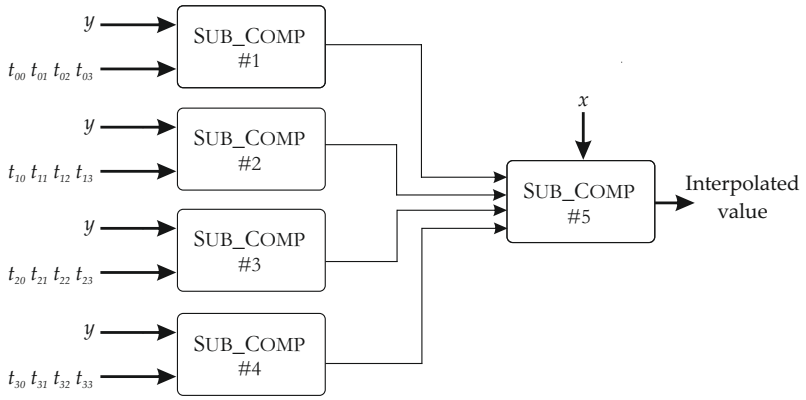


FIGURE 3.5 – Parallelization of interpolation blocks.

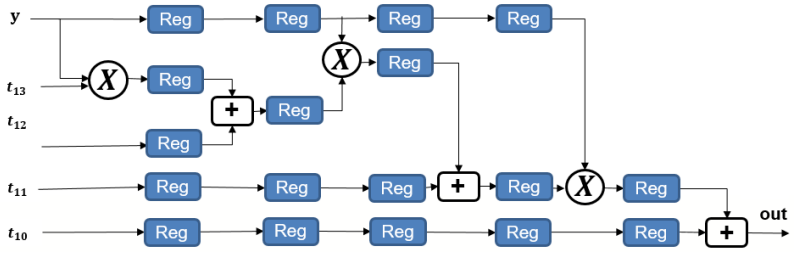


FIGURE 3.6 – SUB_COMP architecture.

3.5 Resource utilization and precision analysis

This section discusses hardware resource utilization according to a given or a requested precision. The interpolation architecture is built on data multiplication/addition stages. The hardware precision depends on the basic fixed-point artifact of bit-width growth of each stage. Compounding the issue, input and constant data come from varying sources (grey-levels and coefficients) and must be combined. Here we look at the bottom-line issue of maintaining precision and overall data fidelity while using a reasonable amount of hardware resources, i.e. achieving the optimal fixed-point implementation.

As can be seen in Eq. (3.5), interpolation depends on the coordinates (x, y) (x and y are values ranging between 0 and 1) , and on 16 coefficients $(t_{00}, ..., t_{33})$. A fixed-point number is composed of three parts : the first part determines the sign (positive or negative). The second and third parts are the integer and decimal parts, respectively. In this section, we define three parameters :

— m : is the pixel data length $(A_{00}, A_{01}, ...A_{33})$.

— n : is the length of the coordinates (x, y).

— k : defines the length of the multipliers inputs.

The precision analysis is performed by using Binary Tree Arithmetic (BTA). For instance, the BTA of sub-block SUB_COMP #1 is depicted in Fig. 3.7, where block T truncates the output of the adder in order to control the size of the multiplier. This analysis is based on considering the worst case for evaluating the bit-width of each coefficient and the output of each arithmetic operation.

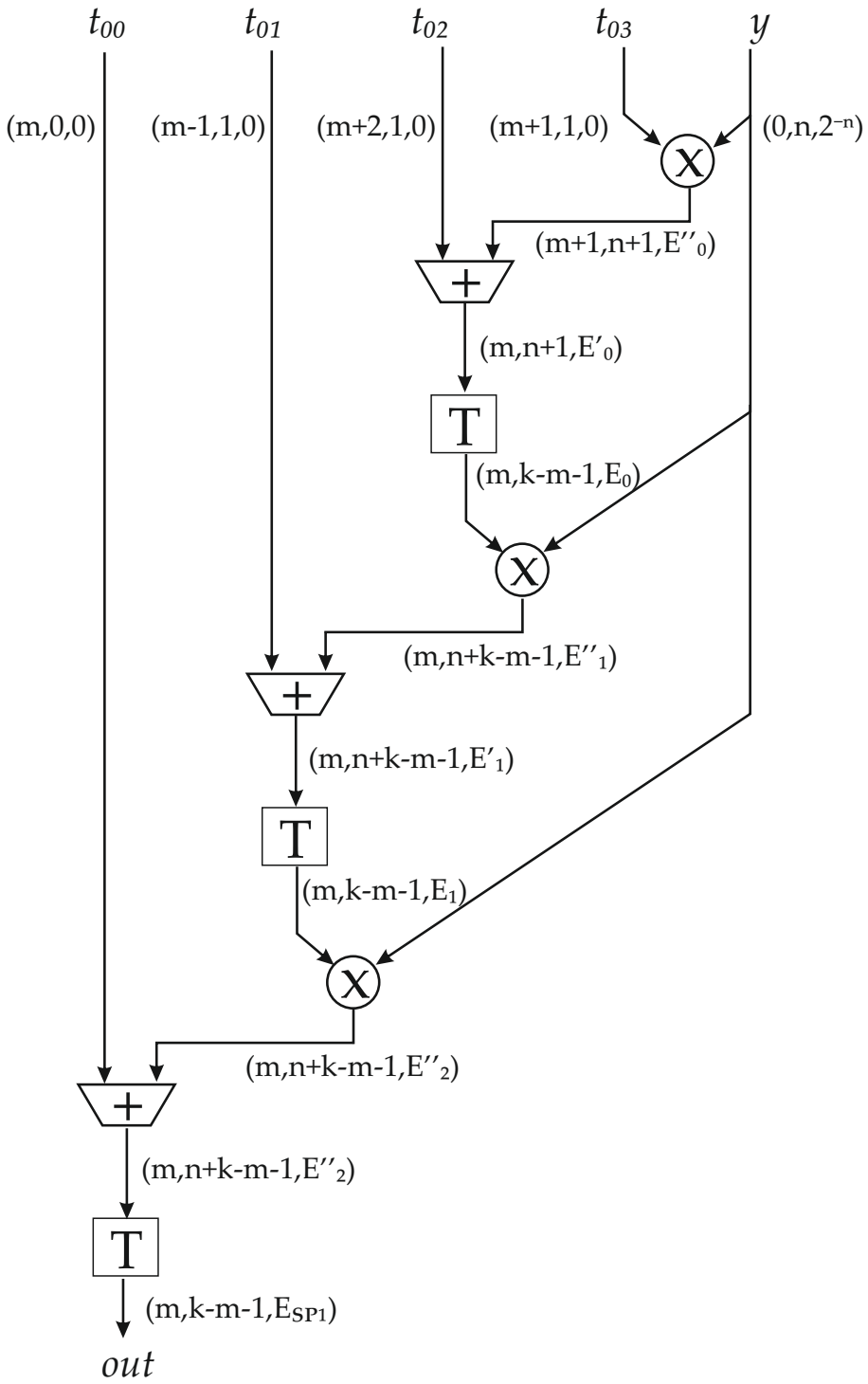


FIGURE 3.7 – Binary Tree Arithmetic of sub-block SUB_COMP #1 with multipliers, adders and truncating operators. For each quantity, the labels in the brackets refer to (length of the integer part, length of the fractional part, Error).

In Fig. 3.7, E'_0 and E''_0 represent different stages of the error propagation up to E_0. The same remark holds for E'_1 and E''_1, as well as for E'_2 and E''_2. The final error E_{SP1} due to the sub-block SUB_COMP #1 is given by :

$$\begin{aligned} & E_{SP1} = 2^{-n+m} + E_1(1-2^{-n}) + 2^{-k+m+1} - 2^{-n-k+m+2} \\ \text{with} \quad & E_1 = 2^{-n+m} + E_0(1-2^{-n}) + 2^{-k+m+1} - 2^{-n-k+m+2} \\ \text{and} \quad & E_0 = 2^{-n+m+1} + 2^{-k+m+1} - 2^{-n} \end{aligned} \tag{3.6}$$

In order to optimize the proposed architecture, the full width of the multiplier operands must be exploited. In our architecture, the multipliers are used to multiply by the coordinates x and y, as depicted in Fig. 3.6. Thus, the best coding corresponds to n=(k-1). In this case the error is given by Eq. (3.7).

$$\begin{aligned} & E_{SP1} = 2^{-k+m+2} + E_1(1-2^{-k+1}) + 2^{m+k-1} - 2^{-2k+m+3} \\ \text{with} \quad & E_1 = 2^{-k+m+1} + E_0(1-2^{-k+1}) + 2^{m+k-1} - 2^{-2k+m+3} \\ \text{and} \quad & E_0 = 2^{-k+m+2} + 2^{m+k-1} - 2^{-k+1} \end{aligned} \tag{3.7}$$

Moreover, in most cases, $k > 9$, and $7 > m > 12$. Then $1 >> 2^{-k+1}$ and $2^{-k+m+1} >> 2^{-2k+m+3}$. 2^{-k+1} is negligible. Consequently, the upper bound of the error is $7 \times 2^{m+1-k}$. Thus,

$$E_{SP1} \leq 7 \times 2^{m+1-k}$$

The maximum error for the other sub-blocks is obtained with a similar approach. These maximum errors are given in Table 3.4.

TABLE 3.4 – Interpolation error as a function of multiplier size (k) and pixel data length (m).

Sub-block	Maximum error
SP1	$7 \times 2^{m+1-k}$
SP2	$7 \times 2^{m+1-k}$
SP3	$52 \times 2^{m+1-k}$
SP4	$26 \times 2^{m+1-k}$
SP5	$\mathbf{99 \times 2^{m+1-k}}$

These results illustrate that the larger the fractional part ($k-m-1$), the higher the accuracy, which is logical. The global interpolation error E_{interp} for this architecture is equal to the error of the last sub-block, hence, $E_{interp} = E_{SP5} = 99 \times 2^{m+1-k}$. The optimal balance between precision and resources utilization depends on the hardware platform and the application requirements. Most of the FPGAs (Intel or Xilinx) include DSP blocks based on 9×9, 18×18 or 27×27 multipliers. Considering that pixel data are coded on 8 bits ($m = 8$), Table 3.5 compares the different cases of multiplier size.

Using 18 fractional bits (corresponding to 27×27 multipliers) provides the lowest error. Its order of magnitude is 10^{-4}. In this case, the price to pay is to double the fractional bit-width

TABLE 3.5 – Hardware error for the three different multiplier width.

Multiplier width	9×9	18×18	27×27
Interpolation error (in gray level)	99	0.2	3.7×10^{-4}
Error (%)	38.82	0.078	1.45×10^{-4}

compared to the case of 9 fractional bits (corresponding to 18×18 multipliers). Coding the fractional part on 9 bits instead of 18 bits significantly decreases the hardware resources. Indeed, the size of the multipliers, adders and registers used to store the intermediate results is reduced by 33.33%. This case also leads to a small error (order of magnitude : 10^{-1}). Using 9×9 multipliers is the best solution in terms of hardware resource utilization, but it provides the worst error, as illustrated Table 3.5.

In order to validate these results with a ground truth, we down-scaled a reference image shown in Fig. 3.8 with a factor of 3. Then, we up-scaled a thumbnail (red rectangle in Fig. 3.8) of 128x128 pixels with a factor of 3 in order to retrieve the reference thumbnail. The thumbnails obtained with 9×9, 18×18 and 27×27 multipliers are depicted in Fig. 3.9. The Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) are calculated to compare the interpolation quality of the obtained thumbnails with the reference one.

The MES of a N $\times$ M image is calculated by using the following equation, where p and g are the reference and the interpolant images, respectively.

$$MSE = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} [p(i,j) - g(i,j)]^2 \tag{3.8}$$

The corresponding PSNR is defined in Eq. (3.9), where MAX represents the maximum pixels value. The value of MAX is 255 because the pixels are coded in 8 bits in this study.

$$PSNR = 10 \times log_{10} \frac{MAX^2}{MSE} \tag{3.9}$$

The results obtained in each case are given in Table 3.6.

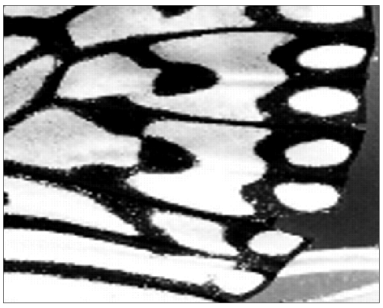
TABLE 3.6 – PSNR and MSE corresponding to 9×9, 18×18 and 27×27 multipliers.

Width of multiplier	9×9	18×18	27×27
MSE	177.3026	168.7880	168.7879
PSNR	25.643	25.8574	25.8574

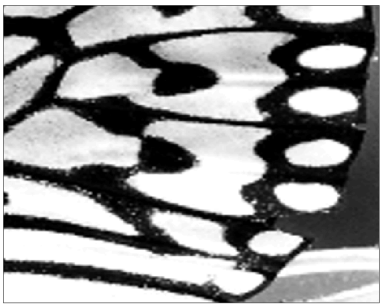
There is no clear visual difference between the three cases shown in Fig. 3.9. However, by comparing the MSE and the PSNR obtained in the three cases, we notice that lower interpolation quality is obtained with 9×9 multipliers than with 18×18 and 27×27 multipliers. The same

PSNR is obtained in the last two cases.

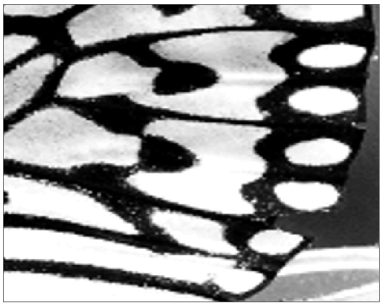
FIGURE 3.8 – Down-scale of the reference image. The red rectangular represents the chosen thumbnail.



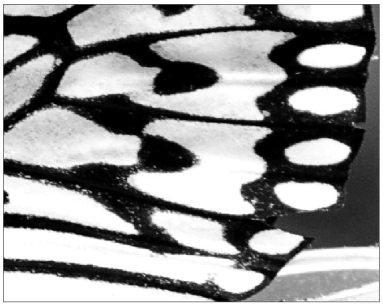
(a) 9×9 multipliers



(b) 18×18 multipliers



(c) 27×27 multipliers



(d) Reference thumbnail

FIGURE 3.9 – Reconstructed thumbnails compared to the reference.

The hardware precision of the interpolation algorithm depends on the coordinates (x, y), where interpolation is performed, and on values of the neighbor pixels. The worst case of the hardware precision is obtained at $(x, y) = (max(x), max(y))$, where $0 < x < 1$ and $0 < y < 1$. In this case, the up-scaled image is obtained by performing interpolations for $x = \{1/3, 2/3\}$ and $y = \{1/3, 2/3\}$, which means that the error between the three cases of multipliers size is not significant. Consequently, no visual difference can be seen between the different cases. Furthermore, even with a higher scaling factor, the error may be higher only at some locations, which are here the nearest to the location $(1, 1)$. Of course, the error also depends on the neighbor pixels values at these locations.

In order to better evaluate the hardware precision of the bi-cubic interpolation algorithm as a function of the multipliers size, we applied a displacement of 0.99 pixel in the two directions. This means that each pixel of the reference thumbnail is interpolated at (0.99, 0.99). The worst case of hardware precision is obtained in this case. The error and the interpolated image for each multiplier size are given in Table 3.8. Here, the obtained image based on 64 floating-point multipliers is considered as the reference for error calculation.

It is clear that the use of 9×9 multipliers impairs the image quality and produces higher error ($\sim 10^1$). Furthermore, the obtained image is darker than the reference.

The results of Table 3.8 for 18×18 and 27×27 multipliers show almost the same quality. These obtained images are almost similar to the reference, so the error in these two cases is very low, namely $\sim 10^{-2}$ and $\sim 10^{-4}$, respectively. Higher quality can be obtained by using 27×27 multipliers, but more resources are required in this case. Using 18×18 multipliers requires less hardware resources and provides good precision, as depicted in Table 3.8.

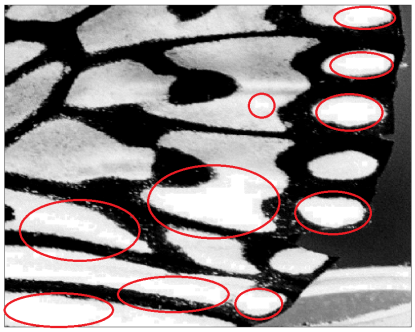
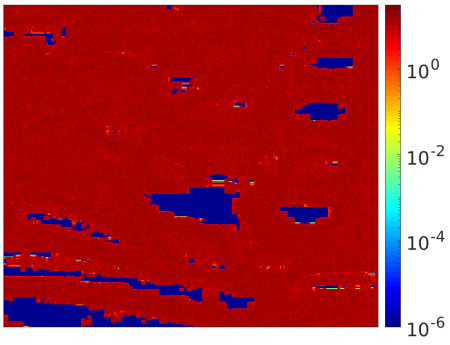
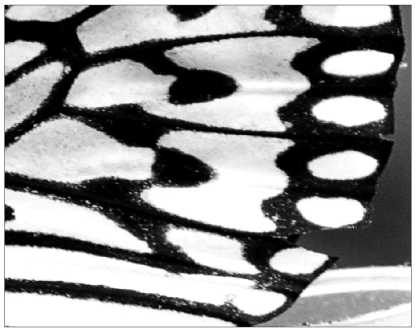
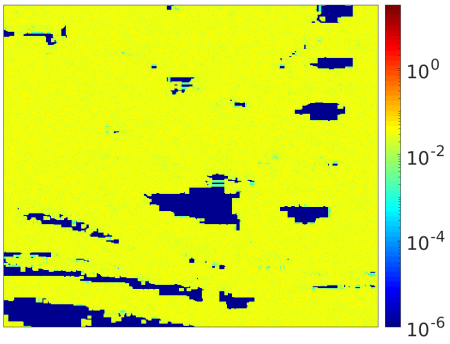
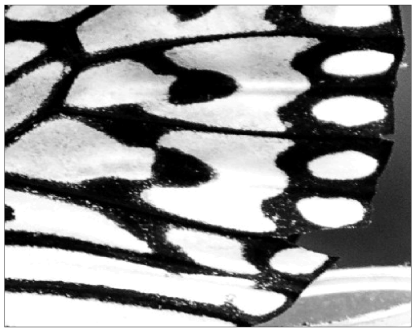
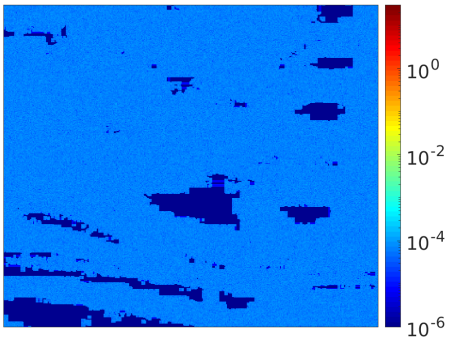
The maximum errors are also computed in each case. They are given in Table 3.7. We notice that the obtained results are consistent with those given in Table 3.5.

TABLE 3.7 – Maximum hardware error of the displaced image versus 9×9, 18×18 and 27×27 multipliers.

Multiplier width	9×9	18×18	27×27
Maximum error (in gray level)	33,64	0,0628	$1,24.10^{-4}$
Error (%)	13,19	0,0246	$4,88.10^{-5}$

In conclusion, a 9-bit fractional part or the case of 18×18 multiplier can be considered to be a good compromise to reach hardware efficiency and precision requirement based on FPGA platform providing 18×18 and 27×27 multipliers. This leads to a precision $\geq 99.922\%$, and reduces by 33.33% the hardware resource consumption compared to the case of 27×27 multipliers.

TABLE 3.8 – Hardware error due to the multipliers size : displaced images "interpolation at (0.99, 0,99)" and the error in each case.

Multiplier size	Displaced image	Error (grey level) "log scale"
9x9		
18x18		
27x27		

3.6 Results

The proposed architecture has been coded in VHDL and implemented on Intel and Xilinx targets (Cyclone II, Virtex-II Pro, and Cyclone V). The required resources are provided by Quartus 13.1 and 17.1 for Cyclone II and V, respectively, and by Xilinx ISE 9.1 for Virtex-II Pro.

In order to evaluate the proposed architecture, it is compared to the results given in [11, 13, 14, 15, 9, 10] in terms of number of required DSP, logic utilization, and maximum operating frequency. The results are given in Table 3.9.

TABLE 3.9 – Comparison between hardware implementations.

Architecture	Platform	DSP	Logic	Max frequency
[11]	FPGA	10	437 LBs	-
[13]	Virtex-II FPGA	8	376 LBs	104.3 MHz
[14]	FPGA	6	-	130 MHz
[15]	-	8	-	-
[9]	Virtex-II Pro FPGA	32	890 LBs	100 MHz
[10]	-	20	-	-
Proposed	Virtex-II Pro	15	277 LBs	145 MHz
	Intel Cyclone II	15	271 LBs	133 MHz
	Intel Cyclone V	15	253 LBs	205 MHz

The procedures described in [11, 13, 14, 15] are those suited to homogeneous interpolation. They need a smaller number of multipliers than our architecture because they are only based on two one-dimensional interpolations. Gour, *et al.* [15] did not realize a hardware implementation. Furthermore, the architecture used in [13] features lower interpolation quality because of the piecewise linear approximation of the cubic kernel.

The present architecture describes and implements the exact cubic kernel without any approximation or impairment, which means that the interpolation quality of the bi-cubic algorithm is preserved. It also reduces the logic consumption and the number of multipliers compared to the heterogeneous architectures available in the literature, which are based on five one-dimensional interpolations [9, 10]. The number of multipliers in our architecture is only 15 instead of 32 in [9] and 20 in [10].

The heterogeneous architecture implemented in our case is fully-pipelined. It relies on a stream serial architecture. In other words, a bi-cubic interpolation is performed at each rising-edge of the clock. It reaches a higher operating frequency (133 MHz and 145 MHz in the Cyclone II and Virtex-II Pro target, respectively) than the heterogeneous architecture proposed in [9], which operates at 100 MHz on the Virtex-II Pro target. Hence, it is well suited to real-time applications.

The implementation of the proposed architecture on Intel and Xilinx targets (Cyclone II and Virtex-II Pro target, respectively) requires 15 DSPs in both cases. It can also be noticed that the logic utilization and the operating frequency remain in the same range.

The analysis of the hardware precision is also given. This point is not discussed in previous

studies on this topic available in the literature. It optimizes the architecture and reduces the hardware resource consumption, in this case the size of multipliers and adders.

In conclusion of this section, the architecture proposed here provides high-quality results and estimates the sub-pixel value at very accurate locations. Compared to other architectures proposed in the literature, less hardware resources (or low design cost) are needed to reach high hardware precision.

3.7 Conclusion

This paper presents a heterogeneous architecture of bi-cubic interpolation. The proposed solution is based on data stream. It relies on a mathematical reformulation of the bi-cubic interpolation in order to reduce computational complexity. The proposed architecture reduces the number of multipliers by 25% and avoids twelve redundant calculations. It is also fully-pipelined to reach high operating frequency. A fractional bit-width of 9 is selected in order to reach hardware efficiency and result precision (precision $\geq 99.922\%$ with 33.33% of less bit-width). The architecture was implemented using VHDL on Xilinx and Intel targets. It requires less hardware resources compared to previous heterogeneous architectures, and operates at higher frequency.

Acknowledgment

This work has been sponsored by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01).

Bibliographie

[1] Ruangsang, W., and Aramvith, S. : Efficient super-resolution algorithm using overlapping bicubic interpolation. In : IEEE 6th Global Conference on Consumer Electronics, pp. 1-2 (2017)

[2] Sutton, M.A., Orteu, J.J, Schreier, H. : Image Correlation for Shape, Motion and Deformation Measurements : Basic Concepts, Theory and Applications. In : Springer Verlag-US, (2009)

[3] Schaum, A. : Theory and design of local interpolators. In : CVGIP : Graphical Models and Image Processing **55**(6), pp. 464-481 (1993)

[4] Lehmann, T.M., Gonner, C., and Spitzer, K. : Survey : interpolation methods in medical image processing. IEEE Trans. Med. In : Imaging **18**(11), pp. 1049-1075 (1999)

[5] Keys, R.G. : Cubic convolution interpolation for digital image processing. In : IEEE Transactions on Acoustics, Speech, and Signal Processing **29**(6), pp. 1153-1160 (1981)

[6] Duchon, C.E. : Lanczos filtering in one and two dimensions. In : Journal of Applied Meteorology **18**(8), pp. 1016-1022 (1979)

[7] Xiang, Z., Zou, X., and Liu, Z. : An high quality image scaling engine for large-scale LCD. In : Int. Conf. Signal Process. Proceedings, (2006)

[8] Hung, N.V., Hien, N.T.T., Vinh, P.T., Thao, N.T., and Dzung, N.T. : An utilization of edge detection in a modified Bicubic interpolation used for frame enhancement in a camera-based traffic monitoring. In : Proc. KICS-IEEE Int. Conf. Inf. Commun. with Samsung LTE 5G Spec. Work. ICIC, (2017)

[9] Nuno-Maganda, M.A., and Arias-Estrada, M.O. : Real-time FPGA-based architecture for bicubic interpolation : An application for digital image scaling. In : International Conference on Reconfigurable Computing and FPGAs, (2005)

[10] Zhang, Y., Li, Y., Zhen, J., Li, J., and Xie, R. : The hardware realization of the bicubic interpolation enlargement algorithm based on FPGA. In : Proceedings - 3rd International Symposium on Information Processing, pp. 277-281 (2010)

[11] Lin, C.C., Sheu, M.H., Chiang, H.K., Liaw, C., Wu, Z.C., and Chen, C.H. : The Efficient VLSI Design of BI-CUBIC Convolution Interpolation for digital image Processing. In : IEEE International Symposium on Circuits and Systems, pp. 480-483 (2008)

[12] Pang, Z.Y., Tan, H.Z., and Chen, D.H. : An improved low-cost adaptive bicubic interpolation arithmetic and VLSI implementation. In : Acta Automatica Sinica **39**(4), pp. 407-417 (2013)

[13] Lin, C.C., Sheu, M.H., Chiang, H.K., Liaw, C., Wu, Z.C., and Tsai, W.K. : An efficient architecture of extended linear interpolation for image processing. In : J. Inf. Sci. Eng. **26**(2), pp. 631-648 (2010)

[14] Wang, X., Ding, Y., Liu, M.Y., and Yan, X.L. : Efficient implementation of a cubic convolution based image scaling engine. In : Journal of Zhejiang University SCIENCE C **12**(9), pp. 743-753 (2011)

[15] Gour, P.N., Narumanchi, S., Saurav, S., and Singh, S. : Hardware accelerator for real-time image resizing. In : 18th Int. Symp. VLSI Des. Test, (2014)

[16] Horowitz, M. : Computing's energy problem (and what we can do about it). In : IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, pp. 10-14 (2014)

[17] Kumar, A. K., Patnaik, S., and Jeevaratnam, N. : On-chip memory for image processing applications based on FPGA. In : International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), Paralakhemundi, 2016, pp. 1598-1602 (2016)

Chapitre 4

Alternatives to bi-cubic interpolation considering FPGA hardware resource consumption

Préambule

L'interpolation bi-cubique est généralement choisie parce qu'elle offre un bon compromis entre complexité de calcul et qualité d'interpolation. Dans le chapitre précédent, nous avons présenté une architecture hétérogène optimisée de l'interpolation bi-cubique où la consommation des ressources matérielles a été réduite par rapport à la littérature. L'objectif principal de ce chapitre est de réduire davantage la consommation de ressources matérielles. Pour cela, un ensemble d'algorithmes qui approchent au mieux l'interpolation bi-cubique et qui réduisent la complexité de calcul est proposé. Ces algorithmes sont basés sur des combinaisons d'interpolations unidimensionnelles linéaire et cubique. Ils seront implémentés sur un FPGA Intel Cyclone V et comparés en termes de qualité d'interpolation, de consommation des ressources matérielles, et de fréquence de fonctionnement maximale.

En conclusion, ce chapitre présente plusieurs choix d'algorithmes d'interpolation pour remplacer la bi-cubique et répondre aux différentes exigences, plus particulièrement la précision et la consommation des ressources matérielles.

Ce chapitre 4 a fait l'objet d'un article qui a été accepté en octobre 2020 pour être publié dans le Journal "*IEEE Transactions on Very Large Scale Integration (VLSI) Systems*".

Alternatives to bi-cubic interpolation considering FPGA hardware resource consumption

S. Boukhtache[1], B. Blaysat[1], M. Grédiac[1], F. Berry[1]

[1]*Institut Pascal, UMR 6602, Université Clermont-Auvergne, CNRS, SIGMA Clermont, Clermont-Ferrand, France*

Abstract

Bi-cubic interpolation is widely used in real-time image processing systems because of its quality. The real-time implementation of bi-cubic interpolation requires a lot of hardware resources, especially the number of multipliers because it represents high computational complexity. In this paper, a set of algorithms that approximate the bi-cubic interpolation and reduce the hardware resource consumption are proposed. The proposed algorithms are based on combining linear and cubic interpolations. These algorithms are surveyed and compared in terms of interpolation quality, number of adders, number of multipliers, Adaptive Logic Modules, LUTs, registers, and maximum operating frequency. These algorithms are implemented and tested on an Intel Cyclone V target. This paper provides various choices of interpolation algorithms to cater to different application requirements including accuracy, hardware resource consumption, and throughput performance. The implementation codes are available at :

`github.com/DreamIP/Interpolation`

Keywords : *Bi-cubic interpolation, FPGA, hardware resource consumption, interpolation quality, hardware precision.*

4.1 Introduction

Nowadays a continuous increase use of digital image processing can be observed in both commercial and industrial applications. Interpolation plays a crucial role in digital image processing. It defines the accuracy and may affect the processing time of the target application. It is widely used and required by many real-time applications such as resolution enhancement, image resizing, geometric transformation, or in various domains such as displacement estimation with

digital image correlation. Interpolation retrieves an infinite resolution data from a set of discrete inputs. Image interpolation aims at computing image value at any (sub-pixel) location, by relying on the neighbor pixels intensities and location.

The accuracy of the obtained results depends on the interpolation quality. The highest performance of interpolation is primordial in order to reach the desired requirements of the target application, especially accuracy. The higher the interpolation quality, the more the hardware resources required for its implementation and thus the higher the design cost, as this cost heavily relies on the use of FPGA hardware resources.

An optimal architecture represents the best trade-off between accuracy and hardware cost. Consequently, interpolation must be carefully studied in terms of output quality, hardware resource consumption, and throughput performance for practical applications.

The quality of the results provided by interpolation may significantly change according to the used algorithm. Many interpolation algorithms are available in the literature and choosing the one offering the best trade-off between computational complexity, processing time, and interpolation quality is crucial to fulfill real-time requirements, and to reach hardware efficiency for applications based on interpolation.

Among the existing interpolation algorithms, the following ones are the most commonly used. With the Nearest Neighbor (NN) method, the interpolation model is a zero-order polynomial. It is the simplest and fastest algorithm. However, its main drawback is to induce the highest error. Bilinear interpolation is rapid. It relies on a first-order polynomial model which is simple to implement. It is more accurate than NN but interpolation quality can still be improved [1, 2, 3]. Winscale interpolation [4, 5] is a linear scaling algorithm. It delivers better image quality than the Bilinear algorithm. Its hardware cost is low but the resulting images involve undesirable effects. Cubic spline interpolation involves high computational complexity (i.e., resolving matrix problems) in order to achieve higher quality. Thus, its hardware implementation is not simple and requires a lot of processing time. Bi-cubic interpolation is performed by using a third-order polynomial function. It features better quality than Winscale interpolation, and it is less complex to implement than cubic spline interpolation [2, 1, 6]. The higher the order of the kernel function, the better the interpolation quality. Also, the throughput performance is inversely proportional to the kernel order. Lanczos interpolation consists in using a Look-Up-Table of a windowed-$sinc$ function [7]. Of course, its accuracy depends on the discretization step, in other words, on the memory utilization. Lanczos 2 and 3 require 16 and 36 neighbor pixels, respectively. Lanczos 2 offers almost similar quality than bi-cubic interpolation. Lanczos 3 offers higher interpolation quality. It requires 10 more multipliers and adders to perform a two-dimensional interpolation compared to Lanczos 2. The kernels of the interpolation methods discussed above are listed in Table 4.1.

Furthermore, various interpolation algorithms based on convolutional neural network (CNN) are used for super-resolution purpose. In the works of Dong *et al.* [8, 9], the input image is enlarged in advance to the desired output size by using the bi-cubic interpolation. It is then given to the CNN. Kim *et al.* [10] proposed Deeply-Recursive CNNs that outperform the work of Dong *et*

TABLE 4.1 – Kernel of the interpolation techniques.

Method	NN	Bilinear	Bi-cubic	spline	Lanczos
Kernel type	0-order	1st-order function	3rd-order function	nth-order function	Look-Up Table

al. [8, 9] in terms of quality. However, deeply-recursive architecture is not suitable for real-time implementation on FPGAs. Shi *et al.* [11] reconstructed the output image by using multiple input images but this increases the computational complexity. Manabe *et al.* [12] implemented a CNN-based architecture on an FPGA to provide super-resolution images in real-time. CNN-based algorithms offer high interpolation quality at the price of very high computational complexity, which makes them not suitable for FPGA implementation. This is the reason why bi-cubic interpolation is considered as the technique leading to the best trade-off between interpolation quality and computational complexity or hardware resource consumption compared to the other algorithms [1, 6]. Bi-cubic interpolation is also successfully used for various applications such as image scaling [13], image quality enhancement [14] and super-resolution [15].

In this work, four algorithms are proposed in order to reduce the consumption of hardware resources while keeping the desired bi-cubic interpolation quality. These algorithms reduce the use of multipliers by more than 65%, require 30% less of ALMs and registers compared to the bi-cubic algorithm, and approximate at best the bi-cubic interpolation.

This paper is organized as follows. Section 4.2 briefly recalls the basics of bi-cubic interpolation. Section 4.3 surveys previous hardware implementations of the bi-cubic algorithm, which are available in the literature. The approximation of the third-order cubic kernel by n-piecewise linear functions is detailed in Section 4.4. Section 4.5 introduces combinations based on linear and cubic interpolations. Section 4.6 presents and compares the interpolation quality and the hardware implementation results of these algorithms.

4.2 Bi-cubic interpolation

Bi-cubic interpolation is the natural extension of one-dimensional cubic interpolation to construct new data on a two-dimensional space. The one-dimensional interpolation function can be written as follows :

$$g(x) = \sum_{k=0}^{3} A_k \beta(x - x_k) = \sum_{k=0}^{3} A_k \beta(u_k) \tag{4.1}$$

where A_k, $k = 0...3$, are the set of discrete data, β is the interpolation kernel and g is the output, i.e the interpolated function.

Keys [6] used the cubic interpolation kernel given in (4.2). It requires 4 neighbor pixels.

$$\beta(u) = \begin{cases} (a+2)|u|^3 - (a+3)|u|^2 + 1 & 0 \leq |u| < 1 \\ a|u|^3 - 5a|u|^2 + 8a|u| - 4a & 1 \leq |u| < 2 \\ 0 & others \end{cases} \tag{4.2}$$

Parameter a is set to -0.5 in order to reach and to offer the best interpolation quality [6]. This leads to the following expression of $\beta(u)$:

$$\beta(u) = \begin{cases} \frac{3}{2}|u|^3 - \frac{5}{2}|u|^2 + 1 & 0 \leq |u| < 1 \\ -\frac{1}{2}|u|^3 + \frac{5}{2}|u|^2 - 4|u| + 2 & 1 \leq |u| < 2 \\ 0 & others \end{cases} \tag{4.3}$$

Bi-cubic interpolation relies on 16 neighboring pixels. Two types of one-dimensional interpolations are performed in turn along the horizontal and the vertical directions, as shown in Fig. 4.1.

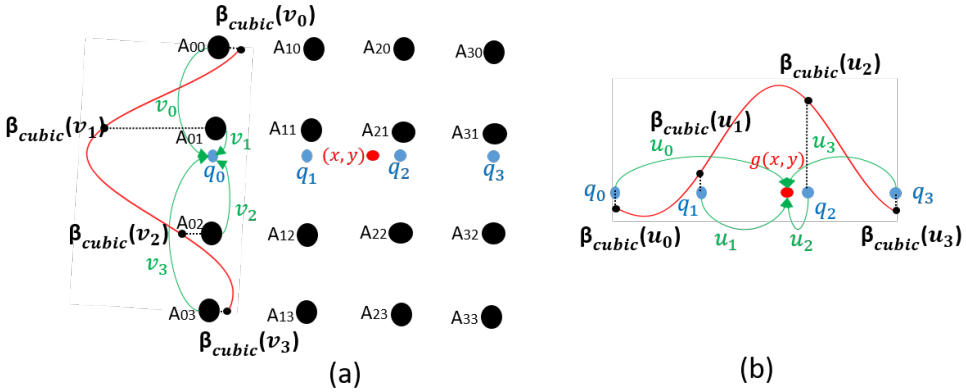


FIGURE 4.1 – Bi-cubic interpolation, (a) vertical and (b) horizontal one-dimensional interpolations.

It can be seen that the vertical interpolation is performed four times to obtain four intermediate points (q_0, q_1, q_2, q_3), whereas the horizontal one, which is based on these intermediate points, is performed once. It is worth noting that the order between the two directions (vertical and horizontal) can be switched without any impact on the value obtained at the end of the procedure.

The bi-cubic formula is given by :

$$g(x,y) = \sum_{i=0}^{3} \Big(\sum_{j=0}^{3} A_{ij} \beta(y - y_j) \Big) \beta(x - x_i) \tag{4.4}$$

The preceding equation can be rewritten with $u_i = x - x_i$ and $v_i = y - y_i$. Thus :

$$\begin{aligned} g(x,y) = &(\beta(v_0)A_{00} + \beta(v_1)A_{10} + \beta(v_2)A_{20} + \beta(v_3)A_{30})\beta(u_0) \\ &+(\beta(v_0)A_{01} + \beta(v_1)A_{11} + \beta(v_2)A_{21} + \beta(v_3)A_{31})\beta(u_1) \\ &+(\beta(v_0)A_{02} + \beta(v_1)A_{12} + \beta(v_2)A_{22} + \beta(v_3)A_{32})\beta(u_2) \\ &+\beta(v_0)A_{03} + \beta(v_1)A_{13} + \beta(v_2)A_{23} + \beta(v_3)A_{33})\beta(u_3) \end{aligned} \tag{4.5}$$

where $\beta(u_k)$ and $\beta(v_k)$ are the coefficients used for the horizontal and vertical interpolation, respectively. They are calculated by using (4.3).

4.3 Previous works

This section is a brief overview of previous hardware implementations of the bi-cubic algorithm presented above. These previous hardware implementations can be classified into two categories :

— **Heterogeneous architecture** is the standard one. It can interpolate at different locations, as depicted in Fig. 4.2.(a). Indeed, different coefficients are recalculated for each location. This architecture can be used by any application such as image rotations and deformation measurements.

— **Homogeneous architecture** is a special case of the heterogeneous one (see Fig. 4.2.(b)). It is suited only when interpolation is performed at the same location for all windows of the image. This means that the coefficients are calculated only one time, because the same ones will be used for all the pixels, such as in image scaling. The existing CNN-based interpolations are based on homogeneous architecture.

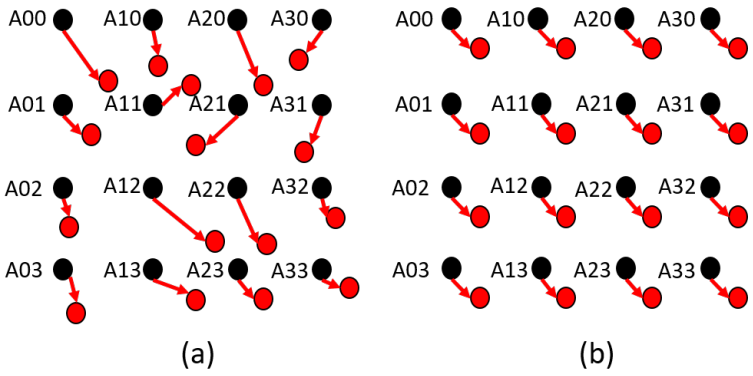


FIGURE 4.2 – (a) Heterogeneous and (b) homogeneous interpolations, the red points represent the locations of interpolation.

Bi-cubic implementations available in the literature are reported in Table 4.2.

The heterogeneous architecture is more complex and calls for more hardware resources. Indeed, it consists in implementing five 1D-cubic interpolations instead of the two 1D-cubic interpolations required in the homogeneous case. This paper considers the heterogeneous case.

Nuno *et al.* [21] decomposed the bi-cubic algorithm into two main blocks. The first one

TABLE 4.2 – Classification of bi-cubic implementations available in the literature.

Work	Architectural features
Homogeneous	
Lin *et al.* [16]	10 multipliers and 437 Logic Blocks. ASIC : 30k gates.
Lin *et al.* [17]	8 multipliers and 376 Logic Blocks. ASIC : 26k gates.
Lin *et al.* [18]	8 multipliers and 414 Logic Blocks. ASIC : 28k gates.
Wang *et al.* [19]	6 multipliers. ASIC : 16k gates.
Gour, *et al.* [20]	8 multipliers.
Heterogeneous	
Nuno *et al.* [21]	32 multipliers and 890 Logic Blocks.
Liu *et al.* [22]	36 multipliers.
Zhang *et al.* [23]	20 multipliers.

generates the interpolation coefficients, and the second one performs the bi-cubic interpolation (4.5). The bi-cubic formula given in (4.5) is implemented in 4 parallel sub-blocks, each sub-block representing one of the four lines of this equation. The design was implemented on a Virtex-II Pro FPGA and operated at a maximum operating frequency of 100 MHz. 32 multipliers and 890 Logic Blocks are necessary in this case.

Liu *et al.* [22] also implemented a heterogeneous bi-cubic interpolation. They simplified the calculation of the interpolation coefficients according to the relationship between them. Their architecture required 36 multipliers to perform interpolation, making it more complex than [21].

In order to reduce the computational complexity, Zhang *et al.* [23] divided the interval between two neighboring pixels into eight subintervals. Then, the coefficients for each of these subintervals were computed and stored to be used each time interpolation was performed. The accuracy of this method depends on the number of subintervals. A drawback is that this architecture is counterbalanced by a poor interpolation quality and high memory utilization. The hardware resource consumption of the proposed design is not reported by the authors.

A homogeneous architecture of the bi-cubic interpolation is implemented in [17, 20, 16, 19]. The architecture of [16, 19] is based on the exact cubic kernel. While an approximated kernel is used in [17, 20, 18] to reduce the hardware resource consumption. The cubic kernel is approximated in [17, 20] by using 2 and 4-piecewise linear functions, respectively. The authors applied the piecewise linear functions in order to approximate a simpler cubic kernel. They use a kernel with '$a = -1$' instead of '$a = -0.5$', which leads to reduce not only the complexity of the cubic kernel but also the interpolation quality. Gour *et al.* [20] applied 4-piecewise linear functions but they did not perform a hardware implementation. Lin *et al.* [18] presented an approximation of the ideal $sinc$-function over the [-2, 2] interval with 4-piecewise linear functions.

In this work, we propose a set of algorithms that reduce the hardware resource consumption of the heterogeneous architecture and accurately approximate bi-cubic interpolation. First, we

extend the idea of approximating the interpolation kernel with n-piecewise linear functions used in [17, 20, 18] to the heterogeneous architecture. The third-order cubic kernel ($a = -0.5$) is approximated with 2, 4, and 6-piecewise linear functions. Then, we combine one-dimensional cubic and one-dimensional linear interpolations to obtain two-dimensional interpolation with less hardware resource consumption.

4.4 Approximation of the cubic kernel with n-piecewise linear functions

A piecewise function is a function in which more than one formula is used to define the output over different pieces or segments of the domain. Each formula is valid over its domain, and the domain of the function is the union of all these small domains. The references [17, 20, 18] approximate the interpolation kernel using 2 and 4-piecewise linear functions and address a homogeneous architecture, as described in the previous section.

In this section, the third-order cubic kernel ($a = -0.5$) is approximated with 2, 4, and 6-piecewise linear functions. Furthermore, the coefficients of each case are adapted such that the lowest hardware resource consumption is obtained. First, coefficients that can be accurately represented in fixed-point are provided. These new coefficients maintain almost the same quality of the approximation. Then, the use of multipliers when multiplying by a constant value is avoided by using constants equal to a power of 2. The adapted hardware kernels are given in (4.6),(4.7),(4.8) and Fig. 4.3.

$$\beta_2(x) = \begin{cases} -|x| + 1 & 0 \leq |x| < 1 \\ -0.03125|x| + 0.03125 & 1 \leq |x| < 2 \\ 0 & others \end{cases} \tag{4.6}$$

$$\beta_4(x) = \begin{cases} -0.25|x| - 0.125|x| + 1 & 0 \leq |x| < 0.17 \\ -0.125|x| - |x| + 1.125 & 0.17 \leq |x| < 1 \\ -0.25|x| + 0.25 & 1 \leq |x| < 1.33 \\ 0.125|x| - 0.25 & 1.33 \leq |x| < 2 \\ 0 & others \end{cases} \tag{4.7}$$

$$\beta_6(x) = \begin{cases} -0.5|x| + 1 & 0 \leq |x| < 0.25 \\ -0.25|x| - |x| + 1.1875 & 0.25 \leq |x| < 0.875 \\ -0.5|x| - 0.25|x| + 0.75 & 0.875 \leq |x| < 1 \\ -0.25|x| - 0.0625|x| + 0.3125 & 1 \leq |x| < 1.218 \\ -0.0625|x| + 0.0156 & 1.218 \leq |x| < 1.375 \\ 0.125|x| - 0.25 & 1.375 \leq |x| < 2 \\ 0 & others \end{cases} \tag{4.8}$$

The Mean Absolute Error (MAE) is determined in order to evaluate and compare the ap-

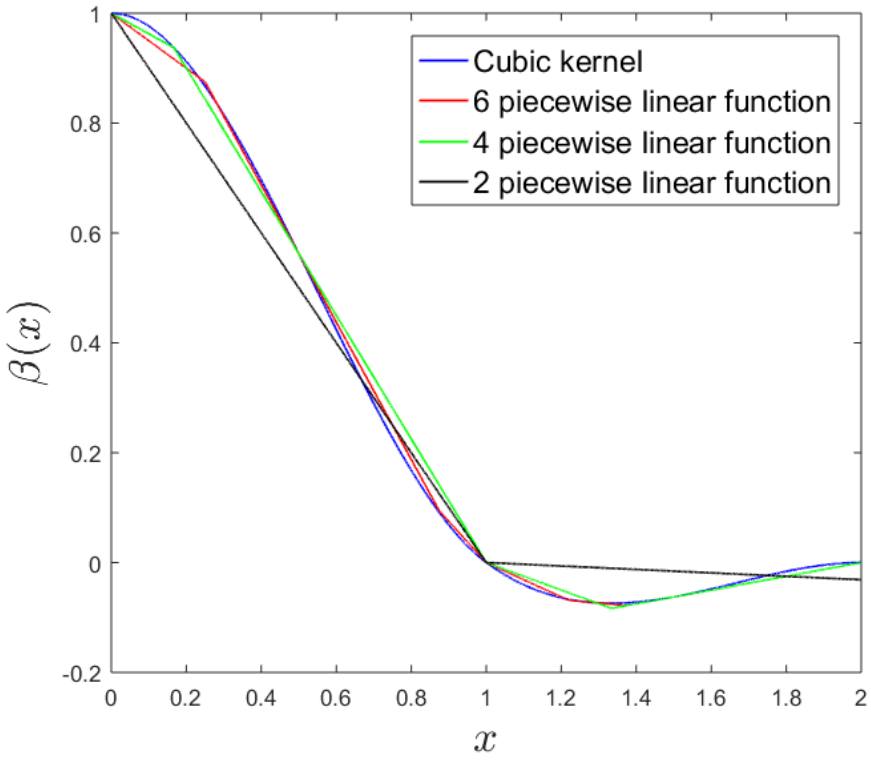


FIGURE 4.3 – Approximation of the cubic kernel based on n-piecewise linear functions.

proximations based on different numbers of piecewise linear functions. It is based on the following equation :

$$MAE_j = \frac{1}{N}\sum_{i=1}^{N} |\beta(x_i) - \beta_j(x_i)| \tag{4.9}$$

Implementing a heterogeneous architecture is based on instantiating the one-dimensional architecture five times, as described in Section 4.2. Four one-dimensional interpolations are used in order to perform the one-dimensional interpolation in the first direction. The outputs of these four previous one-dimensional interpolations provide the inputs of the interpolation in the second direction. The characterizations of each algorithm are summarized in Table 4.3.

TABLE 4.3 – Comparison between piecewise linear algorithms.

Algorithm	Multipliers	Adders	MAE
2-piecewise	5	25	0.0463
4-piecewise	20	39	0.0163
6-piecewise	20	39	0.0081

The 2-piecewise linear functions algorithm requires only 5 multipliers and 25 adders. It reduces the hardware consumption compared to the 2-piecewise linear functions proposed in [17]. However, it features a higher MAE than the 4 and 6-piecewise linear functions. The 6-piecewise linear functions algorithm provides a better quality of interpolation for the same hardware price

as the 4-piecewise linear functions. 20 multipliers and 39 adders are required in both cases.

Combinations of linear and cubic interpolations are proposed in the following section in order to reach better performances.

4.5 Combining cubic and linear interpolations

We propose to combine the one-dimensional linear and cubic interpolations in order to obtain a two-dimensional interpolation, which features lower hardware cost and keeps an interpolation quality of the same order as the bi-cubic one. Consequently, this section first presents the simplified architecture of linear and cubic interpolations. Then, all their possible combinations are detailed and described in turn.

Linear interpolation is simple and easy to implement. It is based on a first-order interpolation kernel. It requires only two pixels to perform one-dimensional interpolation. Furthermore, the linear kernel as shown in (4.10) does not require any intensive calculations. However, it is lower in terms of interpolation quality compared to the cubic one.

$$\beta_{linear}(x) = \begin{cases} 1 - |x| & 0 \leq |x| < 1 \\ 0 & others \end{cases} \tag{4.10}$$

By substituting (4.10) in (4.1), we obtain the one-dimensional linear interpolation given by (4.11).

$$g_{linear}(x) = A_0 \times (1 - x) + A_1 \times x \tag{4.11}$$

(4.11) is simplified by a simple factorization in order to reduce the number of multipliers from 2 to 1, as illustrated in Equation (4.12) and in Fig. 4.4.

$$g_{linear}(x) = (A_1 - A_0) \times x + A_0 \tag{4.12}$$

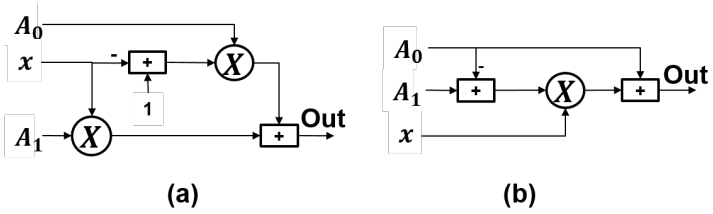


FIGURE 4.4 – Block diagram of (a) linear interpolation and (b) simplified linear interpolation.

Cubic interpolation is implemented by using the same principle of factorization in order to reduce the number of multipliers. The obtained formula of cubic interpolation is given in (4.13), where coefficients (t_0, t_1, t_2, t_3) are obtained by using zero-order function that depends only on the neighbor pixels (A_0, A_1, A_2, A_3) (see Table.4.4). Thus :

$$g_{cubic}(x) = t_0 + \big(t_1 + (t_2 + t_3 \times x) \times x\big) \times x \tag{4.13}$$

TABLE 4.4 – Coefficients of the simplified cubic interpolation.

Coefficient	Expression
t_0	A_1
t_1	$0.5A_2 - 0.5A_0$
t_2	$A_0 - 2.5A_1 + 2A_2 - 0.5A_3$
t_3	$-0.5A_0 + 1.5A_1 - 1.5A_2 + 0.5A_3$

The architecture of the simplified cubic interpolation is depicted in Fig. 4.5.

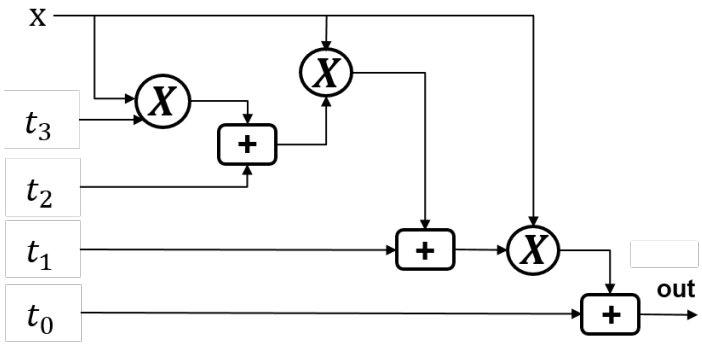


FIGURE 4.5 – Block diagram of the simplified cubic interpolation.

Combining linear and cubic interpolations leads to three possibilities : applying one cubic interpolation and four linear interpolations, two cubic interpolations and one linear interpolation, three cubic interpolations and two linear interpolations. Another algorithm based on three cubic interpolations and a modified linear interpolation is also presented.

Combining two different interpolation kernels may lead to obtain an anisotropic algorithm. In other words, the order of the one-dimensional interpolations (horizontal then vertical, or vertical then horizontal) is an important factor. Consequently, it is of prime importance to study the isotropy or the anisotropy of these algorithms.

The isotropy of the proposed algorithms based on linear and cubic combinations is studied analytically. This is done by comparing the analytic interpolation expressions of a reference interpolant with its transpose. The error due to the anisotropy is assessed with a quantity named "$E_{Isotropy}$" defined in (4.14). An algorithm called isotropic if "$E_{Isotropy}$" is null.

$$E_{Isotropy} = |g(x, y, A_{ij}) - g(y, x, A_{ji})| \tag{4.14}$$

The following namings are used in order to study the error due to the anisotropy of the proposed algorithms. β_{linear} and β_{cubic} are the linear and cubic kernels given by (4.10) and (4.3), respectively. (x_0, x_1, x_2, x_3) and (y_0, y_1, y_2, y_3) replace the expressions $(1 + x, x, 1 - x, 2 - x)$ and $(1 + y, y, 1 - y, 2 - y)$, respectively.

The hardware resource consumption according to the requested hardware precision is also

studied based on interval arithmetic analysis, like in [24]. In this analysis, the worst case is used for evaluating the error due to the fixed-point representation (bit-width) at the output of each arithmetic operation. It is important to notice that this error is cumulative with the one due to the approximation of the interpolation kernel. In this analysis, the following parameters are used.

— 'm' is the pixel data length $(A_{00}, A_{01}, ...A_{33})$

— 'k' defines the length of the inputs of the multipliers

— '$k-m-1$' defines bit-width of the fractional part

The proposed algorithms are described and detailed below.

4.5.1 One cubic and four linear interpolation

In this algorithm, we propose to apply a linear interpolation in the first direction, and a cubic one in the second direction. Thus, the two-dimensional interpolation is performed by applying first four linear interpolations in the first direction in order to obtain the four intermediate points (S_0, S_1, S_2, S_3). These intermediate points are then used to perform a cubic interpolation in the second direction, as depicted in Fig. 4.6.

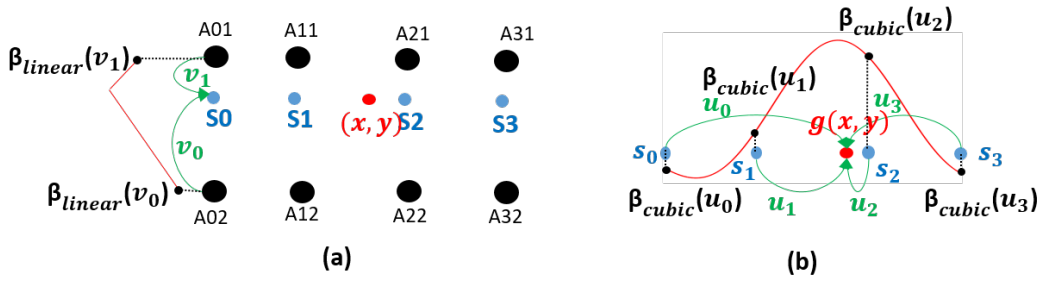


FIGURE 4.6 – Combination of linear and cubic interpolations : (a) four linear interpolations and (b) one cubic interpolation.

Linear interpolation requires only two pixels to interpolate intermediate locations by linearly connecting them. Consequently, two-dimensional interpolation requires only 8 pixels to perform the four linear interpolations instead of 16 pixels in the case of bi-cubic interpolation. This leads to less memory requirement as only one image line and four pixels values are required compared to standard bi-cubic interpolation with three lines and four pixels. Furthermore, the architecture is based on a simplified linear and cubic interpolation, as depicted in Fig. 4.7. This leads to an optimal architecture in terms of hardware resource, especially the number of multipliers.

The architectures of "*Linear*" and "*Cubic*" blocks are illustrated in Fig. 4.4.b and Fig. 4.5, respectively. The hardware architecture of four linear and one cubic interpolation requires 8 neighbor pixels, 7 multipliers, and 21 adders.

The isotropy of this algorithm is studied below. The two-dimensional interpolation writes as follows : by (4.15).

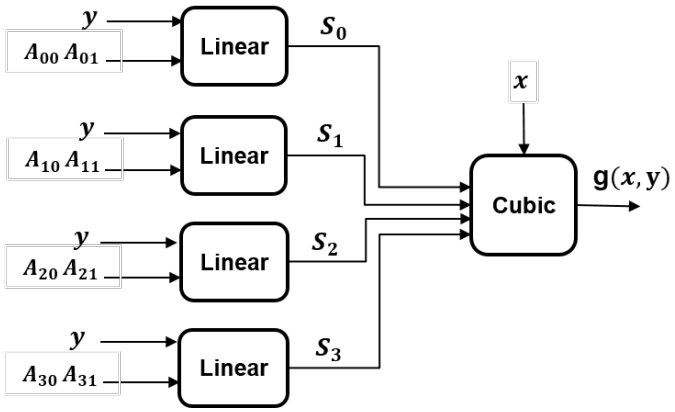


FIGURE 4.7 – Block diagram of four linear interpolations and one cubic interpolation.

$$\begin{aligned} g(x,y,A_{ij}) = &[\beta_{linear}(y_1)A_{01} + \beta_{linear}(y_2)A_{02}]\beta_{cubic}(x_0) \\ &+[\beta_{linear}(y_1)A_{11} + \beta_{linear}(y_2)A_{12}]\beta_{cubic}(x_1) \\ &+[\beta_{linear}(y_1)A_{21} + \beta_{linear}(y_2)A_{22}]\beta_{cubic}(x_2) \\ &+[\beta_{linear}(y_1)A_{31} + \beta_{linear}(y_2)A_{32}]\beta_{cubic}(x_3) \end{aligned} \tag{4.15}$$

Note that : $\beta_{linear}(y_1) + \beta_{linear}(y_2) = 1$. The A_{ij} parameters are maximized when its coefficient is positive and minimized when it is negative in order to reach the worst case. These leads to (4.16).

$$\begin{aligned} &Isotropy_{error} = |g(x,y,A_{ij}) - g(y,x,A_{ji})| \leq \\ &max(A_{ij})[\beta_{cubic}(x_1) + \beta_{cubic}(x_2) - \beta_{cubic}(y_1) - \beta_{cubic}(y_2)] \\ &+min(A_{ij})[\beta_{cubic}(x_0) + \beta_{cubic}(x_3)] \\ &+max(A_{ij})[-\beta_{cubic}(y_0) - \beta_{cubic}(y_3)] \end{aligned} \tag{4.16}$$

Developing (4.16) leads to obtain the maximum error, where 'm' is the pixel data length in bits.

$$Isotropy_{error} \leq 0.125 \times 2^m \tag{4.17}$$

The error due to the hardware representation (bit-width) is studied based on the procedure described above. Table 4.5 gives the maximum hardware error versus hardware resource consumption for each block of this architecture.

TABLE 4.5 – Maximum hardware error *versus* multiplier size (k) and pixel data length (m).

Block	Maximum error
Linear	$2 \times 2^{m+1-k}$
Cubic	$\mathbf{15 \times 2^{m+1-k}}$

The global interpolation error for this architecture consists of the error of the last block, which is $E_{inter} = E_{Cubic} = 15 \times 2^{m+1-k}$.

4.5.2 Two cubic and one linear interpolation

In this algorithm, the cubic interpolation is applied in the first direction and the linear interpolation in the other direction. Hence, two cubic interpolations are performed first by using eight pixels to obtain two intermediate points (S_0, S_1). These two intermediate points are required to perform the linear interpolation in the second direction, as depicted in Fig. 4.8, respectively.

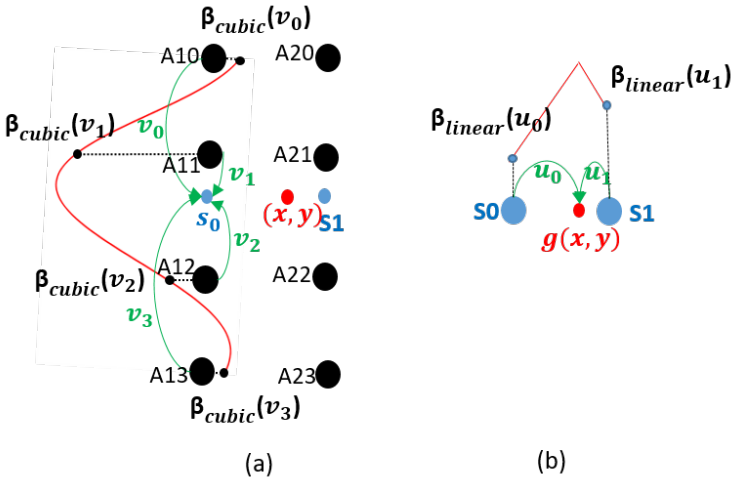


FIGURE 4.8 – Combination of linear and cubic interpolations : (a) two cubic and (b) one linear interpolation.

The architecture of this algorithm is presented in Fig. 4.9. It is based on the simplified linear and cubic interpolations described in Fig. 4.4 and Fig. 4.5.

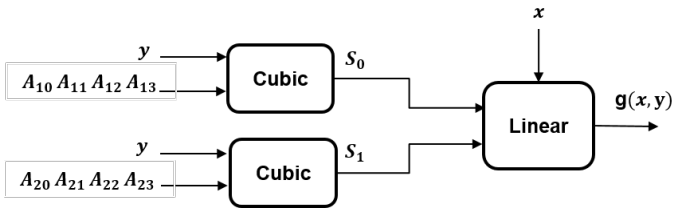


FIGURE 4.9 – Block diagram of two cubic and linear interpolation.

The hardware implementation of the presented algorithm requires 7 multipliers and 28 adders. Furthermore, like the previous algorithm, only 8 neighbor pixels are required. This algorithm consumes 7 more adders than the algorithm combining one cubic and four linear interpolations.

A similar method is also applied for this algorithm in order to evaluate its isotropy or anisotropy. The same maximum error is obtained, because in this algorithm only the order of the one-dimensional interpolations is reversed.

The maximum hardware error versus resource consumption for each block is given in

Table 4.6.

TABLE 4.6 – Maximum hardware error *versus* multiplier size (k) and pixel data length (m).

Block	Maximum error
Cubic	$7 \times 2^{m+1-k}$
Linear	$\mathbf{16 \times 2^{m+1-k}}$

The global interpolation error for this architecture consists of the error of the last block, which is $E_{inter} = E_{Linear} = 16 \times 2^{m+1-k}$.

4.5.3 Three cubic and two linear interpolation

In the present case, the four intermediate points (S_0, S_1, S_2, S_3) are obtained by performing two cubic and two linear interpolations, as depicted in Fig. 4.10. Since the interesting points are the two points, which are the nearest to the interpolation location (S_1, S_2), two cubic interpolations are applied to obtain them and to keep the desired quality of the interpolation. Points S_0 and S_3 are obtained by two linear interpolations to reduce the hardware resource utilization. Then, a one-dimensional cubic interpolation is applied in the other direction to perform the two-dimensional interpolation without impairing the interpolation quality.

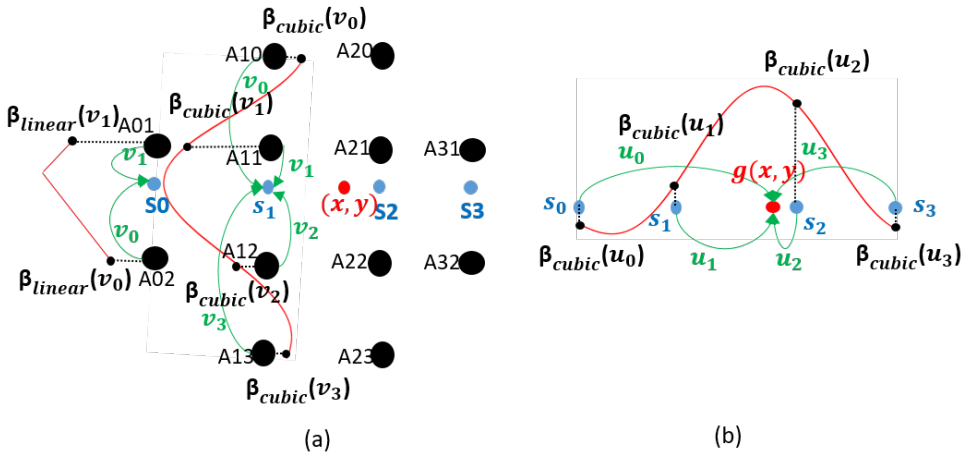


FIGURE 4.10 – Combination of linear and cubic interpolations : (a) two cubic and two linear (b) cubic interpolation.

The hardware architecture of this algorithm is depicted in Fig. 4.11. It requires 12 neighbor pixels. 8 pixels to perform the two cubic interpolations, and 4 pixels to perform the two linear interpolations in the first direction. It consumes 11 multipliers and 43 adders to perform the two-dimensional interpolation.

The isotropy of this algorithm is addressed in the following. Two-dimensional interpolation based on three cubic and two linear one-dimensional interpolations is described by (4.18).

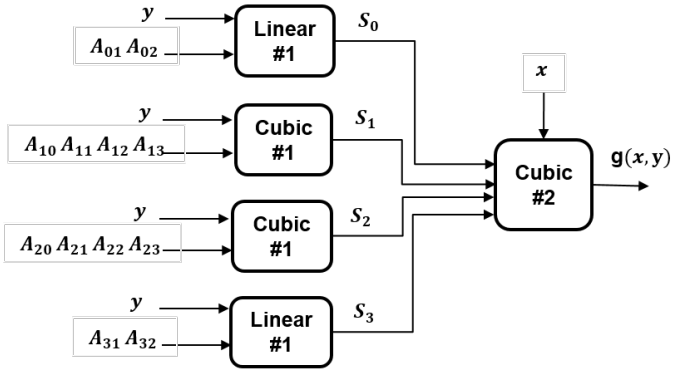


FIGURE 4.11 – Block diagram of three cubic and two linear interpolation.

$$\begin{aligned} g(x,y,A_{ij}) = & [\beta_{linear}(y_1)A_{01} + \beta_{linear}(y_2)A_{02}]\beta_{cubic}(x_0) \\ & + [\beta_{cubic}(y_0)A_{10} + \beta_{cubic}(y_1)A_{11} + \beta_{cubic}(y_2)A_{12} \\ & + \beta_{cubic}(y_3)A_{13}]\beta_{cubic}(x_1) \\ & + [\beta_{cubic}(y_0)A_{20} + \beta_{cubic}(y_1)A_{21} + \beta_{cubic}(y_2)A_{22} \\ & + \beta_{cubic}(y_3)A_{23}]\beta_{cubic}(x_2) \\ & + [\beta_{linear}(y_1)A_{31} + \beta_{linear}(y_2)A_{32}]\beta_{cubic}(x_3) \end{aligned} \tag{4.18}$$

The error due to anisotropy is given by :

$$\begin{aligned} E_{Isotropy} = & |g(x,y,A_{ij}) - g(y,x,A_{ji})| \leq \\ & min(A_{ij})\beta_{cubic}(y_0)[\beta_{cubic}(x_1) + \beta_{cubic}(x_2) - 1] \\ & + min(A_{ij})\beta_{cubic}(y_3)[\beta_{cubic}(x_1) + \beta_{cubic}(x_2) - 1] \\ & + max(A_{ij})\beta_{cubic}(x_0)[1 - \beta_{cubic}(y_1) - \beta_{cubic}(y_2)] \\ & + max(A_{ij})\beta_{cubic}(x_3)[1 - \beta_{cubic}(y_1) - \beta_{cubic}(y_2)] \end{aligned} \tag{4.19}$$

Developing the previous equation leads to :

$$E_{Isotropy} \leq 0.0156 \times 2^m \tag{4.20}$$

The maximum hardware error versus the resource consumption for each block is given in Table 4.7.

TABLE 4.7 – Maximum hardware error *versus* multiplier size (k) and pixel data length (m).

Block	Maximum error
Linear#1	$2 \times 2^{m+1-k}$
Cubic#1	$7 \times 2^{m+1-k}$
Cubic#2	$\mathbf{25 \times 2^{m+1-k}}$

The global interpolation error for this architecture consists of the error of the last block,

which is $E_{inter} = E_{Cubic\#2} = 25 \times 2^{m+1-k}$.

4.5.4 Three cubic and two modified-linear interpolation

Using three cubic interpolations better maintains the interpolation quality compared to one and two cubic interpolations. Consequently, we propose in this algorithm to fix the number of one-dimensional cubic interpolations to three. This is obtained by applying two one-dimensional cubic interpolations in the first direction in order to obtain the two nearest intermediate points (S_1, S_2), and a cubic interpolation in the second direction. The two cubic interpolations applied in the first direction provide only two intermediate points, but four intermediate points are required in order to perform a cubic interpolation in the second direction. Thus, we propose an algorithm that provides better quality than the nearest neighbor and less hardware resource consumption than the linear one. In order to enhance the quality of the interpolation, we proposed to split interval [0, 1] into three subintervals [0, 0.25], [0.25, 0.75] and [0.75, 1]. Concerning to subintervals [0, 0.25] and [0.75, 1], the nearest neighbor algorithm is applied and for subinterval [0.25, 0.75], a mean of the two neighbors is applied, as illustrated in Fig. 4.12.

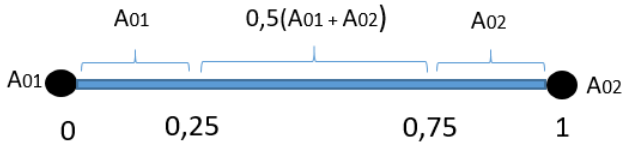


FIGURE 4.12 – Modified linear interpolation strategy.

The architecture of this algorithm is illustrated in Fig. 4.13. The values S_0 and S_3 are obtained based on the idea proposed above. Compared to the previous algorithm, the hardware resource consumption of this algorithm is reduced (gain of 2 multipliers and 2 adders).

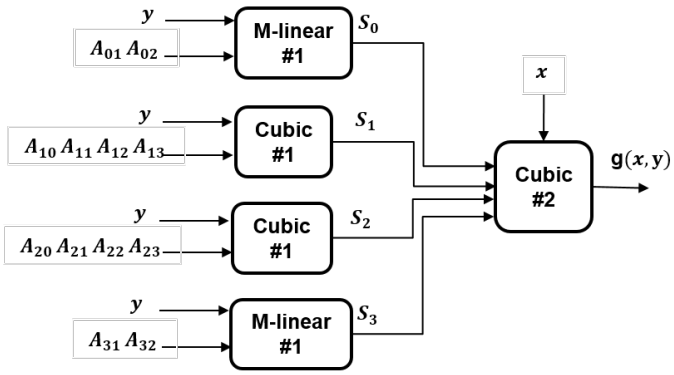


FIGURE 4.13 – Block diagram of three cubic and two modified linear interpolation.

The same maximum isotropy error as the previous algorithm is obtained because "$\beta_{mlinear}(y_1) + \beta_{mlinear}(y_2) = 1$" is also verified.

The error due to the hardware representation (bit-width) is given in Table 4.8.

TABLE 4.8 – Interpolation error *versus* multiplier size (k) and pixel data length (m).

Block	Maximum error
M_linear#1	0
Cubic#1	$7 \times 2^{m+1-k}$
Cubic#2	$\mathbf{21 \times 2^{m+1-k}}$

The global error for this architecture is equal to the error of the last block, which is $E_{inter} = E_{Cubic\#2} = 21 \times 2^{m+1-k}$.

The requirements of the proposed algorithms above are summarized in Table 4.9.

4.6 Results

This section provides and details the implementation results of all the two-dimensional algorithms described above. A study of their interpolation quality is also given.

For each algorithm, hardware resource consumption and maximum operating frequency are given for an Intel Cyclove V target. A classic block made with a shift buffer that receives serial pixels stream from the camera and extracts the neighbor pixels required for the interpolation is used.

In addition to the proposed algorithms, simplified architectures proposed in [21, 23] are also implemented on the same hardware target to provide a fair comparison. The architecture proposed in [22] was not implemented because of its higher hardware complexity compared to [21, 23]. All these algorithms are coded in VHDL and implemented on an Intel Cyclove V target.

The effect of the bit-width on image quality and hardware resource consumption has been evaluated in order to select this parameter for the hardware implementation. It was then possible to define the best trade-off between two main factors.

First, image quality was addressed by representing the maximum rounding error according to the variation of fractional bit-width, as depicted in Fig. 4.14. This is based on the study of the precision (or rounding error) versus the bit-width reported in Table 4.9 by considering that the pixel data length m is equal to 8.

TABLE 4.9 – The requirements of the proposed algorithms. In memory buffer column, the number in brackets represents the number of lines required when applying horizontal then vertical interpolations.

Algorithm	Memory access	Memory Buffer	Adders	Multip	Maximum $E_{Isotropy}$	Hardware precision
1cubic_4linear	8 reads	1 (3) lines	21	7	0.125×2^m	$15 \times 2^{m+1-k}$
2cubic_1linear	8 reads	3 (1) lines	28	7	0.125×2^m	$16 \times 2^{m+1-k}$
3cubic_2linear	12 reads	3 (3) lines	43	11	0.015×2^m	$25 \times 2^{m+1-k}$
3cubic_2mlinear	12 reads	3 (3) lines	41	9	0.015×2^m	$21 \times 2^{m+1-k}$

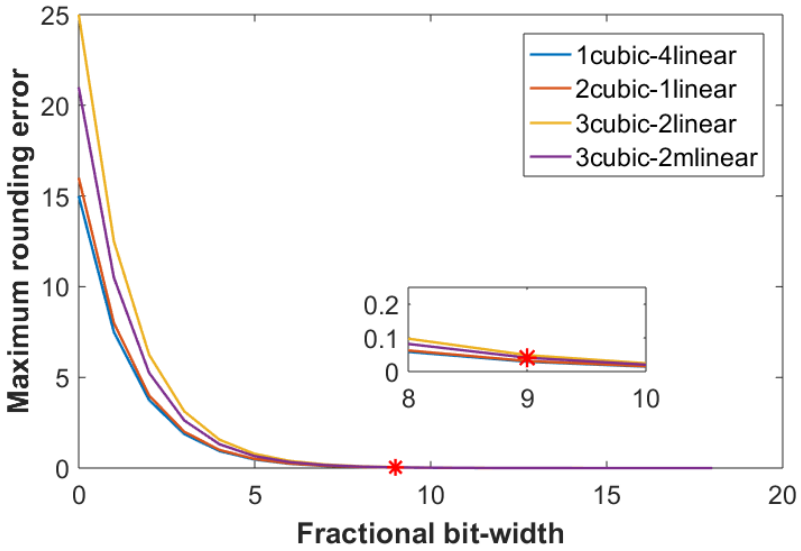


FIGURE 4.14 – Maximum rounding error in grey level versus fractional bit-width.

The hardware resource consumption was then examined. As in FPGAs, multipliers of size equal to 9×9, 18×18, and 27×27 are generally provided, two versions of one of the proposed algorithms, namely the 3cubic-2modified linear algorithm, were implemented. The first version is based on 18 bits and the second one on 27 bits. The results obtained in these cases and their maximum rounding errors are given in Table 4.10.

TABLE 4.10 – Implementations of the 3cubic-2modified linear algorithm, using 9 and 18 fractional bit-width.

Fractional bit-width	Multipliers (size)	LUTs	Registers	ALMs	Frequency	Maximum rounding error
9	9 (18×18)	2112	1671	1057	212 MHz	4.10E-2
18	9 (27×27)	2832	2301	1417	197 MHz	8.01E-5

It can be seen in Fig. 4.14 and Table 4.10 that the best choice is 9 fractional bits, which corresponds to 18×18 multipliers. This leads to nearly the same results as those obtained in the case of floating-point, and reduces the hardware cost by more than 25% compared to the implementation with 18 fractional-bits. Consequently, 9 fractional-bits were considered in this study.

The results of implementations (post place & route) obtained from Quartus for each algorithm are shown in Table 4.11. It is worth noting that the present results refer to a single interpolation core in a streaming application, which means that the operating frequency in MHz implies the throughput in Mpx/s.

In order to compare the approximation quality of each method or the error generated by each algorithm compared to the bi-cubic one, the Mean Square Error (MSE) is provided below. The MSE of an image N $\times$ M is calculated as defined in (4.21) where $P(i,j)$ and $P'(i,j)$ are the reference and the approximated pixel values, at the location (i,j), respectively. The reference

TABLE 4.11 – Hardware implementations based on an Intel Cyclone V target.

Architecture	Multipliers	Adders	Neighbor pixels	LUTs	Registers	ALMs	Frequency	MSE
[21]	32	31	16	3078	2233	1543	158 MHz	Reference
[22]	36	40	16	na	na	na	na	Reference
[23]	20	15	16	870	574	514	209 MHz	427
2-piecewise	5	25	16	902	688	452	213 MHz	24.73
4-piecewise	20	39	16	2512	1879	1277	175 MHz	12.01
6-piecewise	20	39	16	2730	1959	1399	154 MHz	1.66
1cubic-4linear	7	21	8	1800	1586	901	210 MHz	3.99
2cubic-1linear	7	28	8	1357	1131	679	205 MHz	3.99
3cubic-2linear	11	43	12	2470	2046	1236	212 MHz	0.03
3cubic-2mlinear	9	41	12	2112	1671	1057	212 MHz	0.08

TABLE 4.12 – PSNRs of the proposed algorithms and the bi-cubic one with a scaling-factor of 3 (PSNR / normalized PSNR to bi-cubic). The values are rounded to three decimal digits.

Images	Lena	Bridge	Baboon	Boat	Lake	Peppers	Fingerprint
Bi-cubic	30.112 / 1.000	22.684 / 1.000	21.828 / 1.000	25.090 / 1.000	27.642 / 1.000	29.497 / 1.000	26.155 / 1.000
2-piecewise	26.482 / 0.879	22.228 / 0.980	21.378 / 0.979	23.912 / 0.953	25.242 / 0.913	26.022 / 0.882	23.307 / 0.891
4-piecewise	29.464 / 0.978	22.589 / 0.996	21.677 / 0.993	24.885 / 0.992	27.204 / 0.984	28.820 / 0.977	25.901 / 0.990
6-piecewise	30.096 / 0.999	22.672 / 0.999	21.824 / 1.000	25.066 / 0.999	27.641 / 1.000	29.506 / 1.000	26.197 / 1.002
1cubic-4linear	30.053 / 0.998	22.785 / 1.004	22.106 / 1.013	25.099 / 1.000	27.703 / 1.002	29.669 / 1.006	25.121 / 0.960
2cubic-1linear	29.958 / 0.995	22.838 / 1.007	22.004 / 1.008	25.189 / 1.004	27.714 / 1.003	29.608 / 1.004	25.477 / 0.974
3cubic-2linear	30.109 / 1.000	22.685 / 1.000	21.826 / 1.000	25.092 / 1.000	27.642 / 1.000	29.501 / 1.000	26.159 / 1.000
3cubic-2mlinear	30.108 / 1.000	22.682 / 1.000	21.825 / 1.000	25.090 / 1.000	27.640 / 1.000	29.501 / 1.000	26.143 / 1.000

TABLE 4.13 – PSNRs of the proposed algorithms and the bi-cubic one with a scaling-factor of 9/4 (PSNR / normalized PSNR to bi-cubic). The values are rounded to three decimal digits.

Images	Lena	Bridge	Baboon	Boat	Lake	Peppers	Fingerprint
Bi-cubic	32.984 / 1.000	25.019 / 1.000	23.770 / 1.000	27.561 / 1.000	29.898 / 1.000	31.293 / 1.000	30.015 / 1.000
2-piecewise	22.907 / 0.694	21.926 / 0.876	20.447 / 0.860	22.392 / 0.812	22.231 / 0.744	22.175 / 0.709	21.562 / 0.718
4-piecewise	26.168 / 0.793	23.525 / 0.940	21.979 / 0.925	24.748 / 0.898	25.067 / 0.838	25.212 / 0.806	25.241 / 0.841
6-piecewise	29.995 / 0.909	24.564 / 0.982	23.214 / 0.977	26.645 / 0.967	28.049 / 0.938	28.752 / 0.919	28.295 / 0.943
1cubic-4linear	32.753 / 0.993	24.975 / 0.998	23.990 / 1.009	27.440 / 0.996	29.935 / 1.001	31.501 / 1.007	28.271 / 0.942
2cubic-1linear	32.540 / 0.987	25.081 / 1.002	23.913 / 1.006	27.466 / 0.997	29.932 / 1.001	31.397 / 1.003	28.846 / 0.961
3cubic-2linear	33.056 / 1.002	25.032 / 1.001	23.766 / 1.000	27.595 / 1.001	29.931 / 1.001	31.379 / 1.003	30.062 / 1.002
3cubic-2mlinear	33.055 / 1.002	25.023 / 1.000	23.762 / 1.000	27.591 / 1.001	29.920 / 1.001	31.378 / 1.003	30.060 / 1.001

value is the value obtained by using the bi-cubic interpolation and the approximated value is obtained by the proposed algorithms.

$$MSE = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}[P(i,j) - P'(i,j)]^2 \tag{4.21}$$

The approximation error of each algorithm changes from one location to another. Thus we should cover almost all possible locations. The interpolation quality is therfore evaluated by performing an image interpolation at all locations (x, y), where x and y take the values from 0 to 1 with a step of "0.1". Seven test images (Bridge, Baboon, Boat, Lake, Lena, Peppers, and Fingerprint) of size 512×512 were used. The average MSE value of each algorithm is given in Table 4.11.

In addition, the Peak Signal to Noise Ratio (PSNR) is used to measure the quality of the interpolation of each algorithm. The seven images used in the MSE calculation are down-scaled and then up-scaled by a factor 3 and 9/4. The same interpolation method as the one used in up-scaling was considered for the interpolation during down-scaling. The PSNRs obtained in

each case are given in Tables 4.12 and 4.13.

The maximum rounding error (in the worst case) that can occur by using 9 fractional-bits hardware implementation is 0.04 grey level (see Table 4.9 and Fig. 4.14), which is very small. Furthermore, the results obtained when down-scaling and then up-scaling the seven benchmark images by using 9 fractional-bits feature the same PSNR as that obtained by using floating-point. Thus, it can be concluded that using 9 fractional bits do not impact the interpolation quality.

The 2-piecewise linear functions technique is the simplest one with only 5 multipliers, 25 adders, and 452 ALMs. It is also more accurate than [23], which requires 20 multipliers and 514 ALMs, and features an MSE value equal to 427. However, it significantly reduces the interpolation quality of the bi-cubic algorithm and features the lowest interpolation quality compared to the other proposed algorithms (see PSNRs in Tables 4.12 and 4.13).

The cubic-4linear and 2cubic-linear algorithms provide a similar MSE value equal to 3.99, but 2cubic-linear is advantageous in terms of hardware resource cost. Both require also the same number of multipliers and neighbor pixels (8 pixels) to perform two-dimensional interpolation. These two algorithms require less hardware resources compared to [21, 23, 22], and provide a good approximation of the bi-cubic interpolation.

The 4-piecewise linear functions approximation enhances the interpolation quality (decreases the MSE to 12.01) compared to 2-piecewise linear functions. However, it represents a lower quality (higher MSE and lower PSNRs) and higher hardware resource consumption compared to the cubic-4linear and 2cubic-linear algorithms.

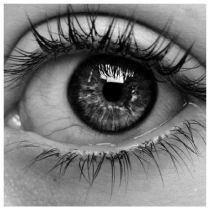
The 6-piecewise linear functions algorithm reduces the MSE to 1.66, which means a better quality of approximation compared to cubic-4linear and 2cubic-linear algorithms. It requires less hardware resources compared to [21, 22], almost the same hardware resource consumption compared to 4-piecewise linear functions, but 50% more than the cubic-4linear and 2cubic-linear algorithms.

The 3cubic-2linear algorithm is the best one in terms of approximation quality with an MSE value equal to 0.03. It requires lower hardware resources compared to 6-piecewise linear functions and [21, 22]. Furthermore, it requires only 11 multipliers instead of 20, 32, and 36 as in [23, 21, 22].

The 3cubic-2modified linear algorithm keeps almost the same accuracy (MSE = 0.08) as the 3cubic-2linear algorithm with lower design cost. It reduces the hardware cost by more than 20% compared to the 3cubic-2linear algorithm. As shown in Tables 4.12 and 4.13, it is clear that the 3cubic-2modified linear and 3cubic-2linear algorithms provide similar PSNRs as the bi-cubic ones.

Fig. 4.15 illustrates the results obtained by down-scaling and then up-scaling a reference image of size 377×377 by a factor of 3.25. The proposed algorithms and the bi-cubic one are used. The absolute errors of the obtained images compared to the reference one are given in order to better visualize and compare the interpolation quality. We can see that the proposed algorithms based on combining linear and cubic interpolations present similar quality (or error) as the bi-cubic one. n-piecewise linear functions algorithms present higher error, especially in

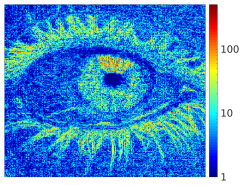
the cases n=2 and n=4.



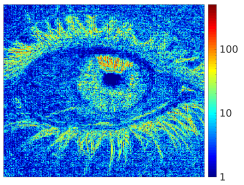
(a) Reference image

(b) Reference image after down-scaling

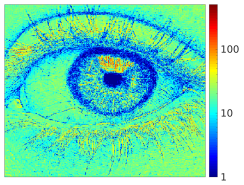
(c) After bi-cubic up-scaling



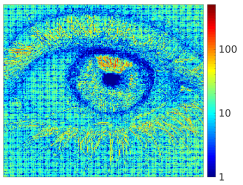
(d) Bi-cubic, PSNR = 24.057



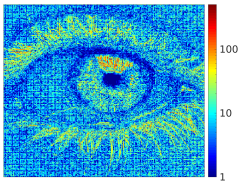
(e) 3cubic-2modified-linear, PSNR = 24.055



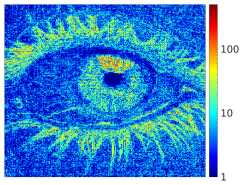
(f) 2-piecewise, PSNR = 21.426



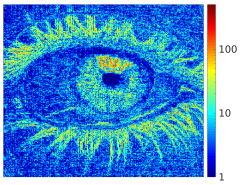
(g) 4-piecewise, PSNR = 22.501



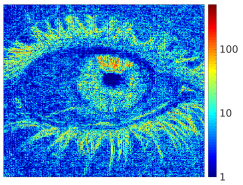
(h) 6-piecewise, PSNR = 23.533



(i) Cubic-4linear, PSNR = 24.284



(j) 2cubic-linear, PSNR = 24.295



(k) 3cubic-2linear, PSNR = 24.055

FIGURE 4.15 – The obtained results of down-scaling then up-scaling a reference image by a factor of 3.25 using the proposed algorithms. (d)-(k) represent the absolute errors compared to the reference image (a) in log scale.

In FPGAs, multipliers are precious and reducing their utilization means decreasing the hardware cost. As illustrated in Table 4.11, our proposed algorithms based on combining linear and cubic interpolations significantly reduce the multipliers use. Compared to [21, 22], the number of multipliers is reduced by more than 65%. Furthermore, these proposed algorithms require less ALMs and registers, operate at a higher frequency compared to [21], and approximate at best the bi-cubic interpolation.

The choice of one of these algorithms relies on the target application requirements. For instance, in the case of metrological application [25], the 3cubic-2modified linear algorithm is an efficient alternative to the standard bi-cubic interpolation.

4.7 Conclusion

In this work, we studied a set of algorithms proposed as alternatives to bi-cubic interpolation. These algorithms are based on combining cubic and linear interpolations. The architecture of each algorithm is implemented on an Intel Cyclove V target. Then the consumption of the hardware resource is studied. The quality of the proposed algorithms is evaluated by comparing their MSE. The main advantage of the proposed algorithms is the reduction of hardware resource consumption.

Acknowledgment

This work has been sponsored by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01).

Bibliographie

[1] T. M. Lehmann, C. Gonner, and K. Spitzer. Survey : interpolation methods in medical image processing. *IEEE Transactions on Medical Imaging*, 18(11) :1049–1075, 1999.

[2] L. Roszkowiak, A. Korzynska, J. Zak, D. Pijanowska, Z. Swiderska-Chadaj, and T. Markiewicz. Survey : interpolation methods for whole slide image processing. *Journal of Microscopy*, 265(2) :148–158, 2016.

[3] M. S. Pan, X. L. Yang, and J. T. Tang. Research on interpolation methods in medical image processing. *Journal of Medical Systems*, 36(2) :777–807, Jun 2010.

[4] C. H. Kim, S. M. Seong, J. A. Lee, and L. S. Kim. Winscale : an image-scaling algorithm using an area pixel model. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(6) :549–553, 2003.

[5] E. Aho, J. Vanne, K. Kuusilinna, and T. d. Hamalainen. Comments on "winscale : an image-scaling algorithm using an area pixel model". *IEEE Transactions on Circuits and Systems for Video Technology*, 15(3) :454–455, 2005.

[6] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6) :1153–1160, 1981.

[7] C. E. Duchon. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology*, 18(8) :1016–1022, 1979.

[8] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *Computer Vision – ECCV 2014*, pages 184–199, Cham, 2014. Springer International Publishing.

[9] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2) :295–307, 2016.

[10] J. Kim, J. K. Lee, and K. M. Lee. Deeply-recursive convolutional network for image super-resolution. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] T. Manabe, Y. Shibata, and K. Oguri. FPGA implementation of a real-time super-resolution system using flips and an rns-based cnn. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E101.A(12) :2280–2289, Jan 2018.

[13] Z. Xiang, X. Zou, and Z. Liu. An high quality image scaling engine for large-scale lcd. *2006 8th international Conference on Signal Processing*, 2006.

[14] N. V. Hung, N. T. T. Hien, P. T. Vinh, N. T. Thao, and N. T. Dzung. An utilization of edge detection in a modified bicubic interpolation used for frame enhancement in a camera-based traffic monitoring. *2017 International Conference on Information and Communications (ICIC)*, 2017.

[15] W. Ruangsang and S. Aramvith. Efficient super-resolution algorithm using overlapping bicubic interpolation. *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, 2017.

[16] C. C. Lin, M. H. Sheu, H. K. Chiang, C. Liaw, and Z. C. Wu. The efficient vlsi design of bi-cubic convolution interpolation for digital image processing. *2008 IEEE International Symposium on Circuits and Systems*, 2008.

[17] C. C. Lin, M. H. Sheu, H. K. Chiang, C. Liaw, Z. C. Wu, and W. K. Tsai. An efficient architecture of extended linear interpolation for image processing. *J. Inf. Sci. Eng.*, 26 :631–648, 2010.

[18] C. C. Lin, M. H. Sheu, C. Liaw, and H. K. Chiang. Fast first-order polynomials convolution interpolation for real-time digital image reconstruction. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(9) :1260–1264, 2010.

[19] X. Wang, Y. Ding, M. Y. Liu, and X. L. Yan. Efficient implementation of a cubic-convolution based image scaling engine. *Journal of Zhejiang University SCIENCE C*, 12(9) :743–753, 2011.

[20] P. N. Gour, S. Narumanchi, S. Saurav, and S. Singh. Hardware accelerator for real-time image resizing. *18th International Symposium on VLSI Design and Test*, 2014.

[21] M. A. Nuno-Maganda and M. O. Arias-Estrada. Real-time FPGA-based architecture for bi-cubic interpolation : an application for digital image scaling. *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig05)*, 2005.

[22] D. Q. Liu, G. Q. Zhou, X. Zhou, C. Y. Li, and F. Wang. FPGA-based on-board cubic convolution interpolation for spaceborne georeferencing. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-3/W10 :349–356, Jul 2020.

[23] Y. Zhang, Y. Li, J. Zhen, J. Li, and R. Xie. The hardware realization of the bicubic interpolation enlargement algorithm based on FPGA. *2010 Third International Symposium on Information Processing*, 2010.

[24] S. Boukhtache, B. Blaysat, M. Grédiac, and F. Berry. FPGA-based architecture for bi-cubic interpolation : The best trade-off between precision and hardware resource consumption. *Journal of Real-Time Image Processing*, 2020. In press.

[25] M. A. Sutton, J. J. Orteu, and H. W. Schreier. *Image correlation for shape, motion and deformation measurements : basic concepts, theory and applications*. Springer US, 2009.

Bilan

Comme décrit dans le chapitre 1, l'algorithme de la CIN est basée sur un processus itératif. En effet, pour calculer le déplacement correspondant à chaque pixel, une imagette de taille $M \times M$ est généralement interpolée 7 fois successivement.

Dans ce cas, il existe deux façons pour implémenter l'interpolation itérative d'imagettes, soit en séquentielle, soit en parallèle. En séquentielle, la vitesse de calcul d'une itération est divisée par le nombre de pixels de l'imagette M^2 (sans considérer la latence de l'architecture). Par exemple dans le cas de $M = 21$, la fréquence de fonctionnent maximale sera divisée par 441. De plus, dans le processus itératif, le passage à l'itération suivante nécessite les résultats de l'itération présente. Par conséquent, effectuer ce processus itératif en séquentiel entraîne une dégradation de la vitesse de calcul par 7 fois le nombre d'étages de pipeline [1], ce qui va à l'encontre d'une rapidité d'exécution optimale. Par conséquent, l'implémentation parallèle est considérée comme le bon choix.

Afin d'effectuer une implémentation parallèle, M x M x 7 blocs d'interpolations sont nécessaires. Le Tableau 4.14 montre les résultats de l'implémentation de ces blocs d'interpolations pour $M = 17$, en utilisant les algorithmes d'interpolations proposés et un algorithme de l'état de l'art.

TABLE 4.14 – Résultats de l'implémentation matérielle sur un FPGA Stratix 10 Gx 2800.

Architecture	Multipliers 18×18	ALMs	MSE
[21] Bi-cubic	64.7 K (562%)	3.1 M (334%)	Bi-cubic
1cubic-4linear	14.1 K (123%)	1.8 M (195%)	3.99
2cubic-1linear	14.1 K (123%)	1.3 M (147%)	3.99
3cubic-2linear	22.2 K (193%)	2.5 M (267%)	0.03
3cubic-2modified linear	18.2 K (158%)	2.1 M (229%)	0.08

Nous concluons que malgré les améliorations apportées pour réduire la consommation des ressources matérielles, il n'est pas possible de mettre en œuvre l'algorithme de la CIN sur un seul FPGA. Ainsi, plusieurs FPGAs sont nécessaires, mais cette solution conduit à un système dont le coût serait rédhibitoire.

Vu la conclusion ci-dessus, nous avons cherché une alternative à l'implémentation de la CIN sur FPGA. Notre choix s'est porté sur l'usage d'un réseau de neurones convolutif. cette solution, qui à notre connaissance n'a jamais été exploitée pour ce genre d'application fait l'objet des développements présentés dans le chapitre suivant.

1. Il convient de savoir que dans le cas de la CIN le nombre d'étages est élevé car en plus de l'interpolation, il y a des multiplications matricielles, comme décrit dans le chapitre 1.

Chapitre 5

When Deep Learning Meets Digital Image Correlation

Préambule

Les réseaux de neurones convolutifs (CNNs [1]) ont été utilisés récemment pour résoudre de nombreux problèmes de vision par ordinateur, en particulier l'estimation du flot optique. La mesure des champs de déplacements et de déformations peut être considérée comme un cas particulier de ce problème. Cependant, les CNNs n'ont jamais été utilisés jusqu'à présent pour effectuer de telles mesures dans le contexte de la photomécanique.

L'objectif principal de ce chapitre est de décrire la mise en œuvre d'un CNN capable d'extraire les champs de déplacements et de déformations à partir d'une paire d'images de référence et déformée d'une surface plane mouchetées, comme le fait la CIN. Ce chapitre explique comment un CNN appelé "StrainNet" a été développé pour atteindre cet objectif, et comment des ensembles de paires d'images de référence et déformée ont été élaborés pour entraîner ce CNN.

En conclusion, les résultats de ce chapitre montrent la viabilité de StrainNet pour effectuer des mesures (sous-pixeliques) précises de champs de déplacements entiers en temps réel. Ainsi, les CNNs comme StrainNet offrent une alternative viable à la CIN, en particulier pour les applications en temps réel.

Ce chapitre 5 a été publié dans le numéro de Janvier 2021 du Journal "*Optics and Lasers in Engineering*". De plus, les principaux résultats ont été présentés sous la forme d'une communication orale au cours de la conférence virtuelle "*International Digital Image Correlation Society (iDICs) 2020*" qui s'est déroulée du 19 au 22 octobre 2020.

1. Convolutional Neural Networks

When Deep Learning Meets Digital Image Correlation

S. Boukhtache[1], K. Abdelouahab[2], F. Berry[1], B. Blaysat[1], M. Grédiac[1], F. Sur[3]

[1]*Institut Pascal, UMR 6602, Université Clermont-Auvergne, CNRS, SIGMA Clermont, Clermont-Ferrand, France*

[2]*Sma-RTy SAS, Aubière, France*

[3]*LORIA, UMR 7503, Université de Lorraine, CNRS, Inria, Nancy, France*

Abstract

Convolutional Neural Networks (CNNs) constitute a class of Deep Learning models which have been used in the recent past to resolve many problems in computer vision, in particular optical flow estimation. Measuring displacement and strain fields can be regarded as a particular case of this problem. However, it seems that CNNs have never been used so far to perform such measurements. This work is aimed at implementing a CNN able to retrieve displacement and strain fields from pairs of reference and deformed images of a flat speckled surface, as Digital Image Correlation (DIC) does. This paper explains how a CNN called 'StrainNet" can be developed to reach this goal, and how specific ground truth datasets are elaborated to train this CNN. The main result is that StrainNet successfully performs such measurements, and that it achieves competing results in terms of metrological performance and computing time. The conclusion is that CNNs like StrainNet offer a viable alternative to DIC, especially for real-time applications.

Keywords : *Convolutional Neural Network, Deep learning, GPU, Digital Image Correlation, Error Quantification, Photomechanics, Speckles*

5.1 Introduction

Digital Image Correlation (DIC) is a full-field displacement and strain measurement technique which has rapidly spread in the experimental mechanics community. The main reason is that this technique achieves a good compromise between versatility, ease of use, and metrological performance [1]. Many recent papers illustrate the use of DIC in various situations [2]. Some others discuss how to characterize or improve its metrological performance, as [3] written within the framework of the DIC Challenge [4]. Despite all its advantages, DIC suffers from

some drawbacks. For instance, DIC is by essence an iterative procedure, which automatically leads to mobilizing significant computational resources. Consequently, its use is limited when dense (i.e. pixelwise-defined) displacement or strain distributions are to be measured. Another drawback is the fact that DIC acts as a low-pass filter, which causes the retrieved displacement and strain fields to be blurred. Indeed, it is shown in [5, 6] that the displacement field rendered by a DIC system is not the actual one, but regardless of noise, the actual one convolved by a Savitzky-Golay filter. This makes DIC to be unable to correctly render displacement or strain fields featuring high spatial frequencies. Overcoming these limitations seems however difficult without introducing a paradigm shift in image processing itself. For instance, the minimization of the optical residual that DIC performs iteratively in the spatial domain can be switched to the Fourier domain [7]. The benefit is to considerably reduce the computing time and to allow the use of optimal patterns in terms of camera sensor noise propagation [8, 9, 10]. The drawback is that such patterns are periodic, and depositing them on specimens remains challenging as long as no simple transferring technique is commercially available. In the present study, we propose to use random speckle patterns as in DIC and to investigate to what extent a Convolutional Neural Network (CNN) can be used to retrieve displacement and strain fields from a pair of reference and deformed images of a speckle pattern. To the best of the authors' knowledge, this route has never been explored so far to retrieve dense subpixel displacement fields. However, CNNs have been widely used in computer vision in the recent past, but mainly in order to perform image classification or recognition [11], or to estimate rigid-body displacements of solids [12]. The problem addressed here is somewhat different because deformation occurs, and because the resulting displacement is generally much lower in amplitude than in the aforementioned cases of rigid-body movements. Consequently we mainly focus here on the estimation of subpixel displacements.

The present paper is organized as follows. The basics of CNNs and deep learning are first briefly given in Section 5.2. We examine in Section 5.3 how a problem similar to ours, namely optical flow determination, has been tackled with CNNs in the literature. CNNs must be trained, and since no dataset suited to the problem at hand is available in the literature, we explain in Section 5.4 how a first dataset containing speckle images has been generated. Then we test in Section 5.5 how four pre-existing networks are fine-tuned with this dataset, and examine whether these networks are able to determine displacement fields from different speckle images artificially deformed. The network giving the best results is then selected and improved in several ways to give displacement fields of better quality. The different improvements and the corresponding results are given in Section 5.6. Finally, we use the network resulting from all the suggested improvements, named here "StrainNet", to process some pairs of speckle images from the DIC Challenge and from a previous study.

The numerical experiments proposed in this paper can be reproduced with Matlab and Pytorch codes as well as with datasets available at the following URL :
https ://github.com/DreamIP/StrainNet

5.2 A short primer on deep learning

Data-driven approaches have revolutionized several scientific domains in the last decade. This is probably due to a combination of several factors, in particular the dramatic improvement of computing and information storage capacity. For example, it is now quite easy to have large datasets gathering many examples for a task of interest, as millions of labeled images for an image classification task. However, the most crucial point is probably the rise of new machine learning techniques built on artificial neural networks. While neural networks have been introduced in the 1950s and have been continuously improved since then, a major breakthrough occurred a few years ago with the demonstration that deep neural networks [13] give excellent results in many signal processing applications. The most iconic breakthrough is probably the famous 2012 paper [11] which demonstrates a deep learning based system outperforming the competing approaches in the ImageNet classification challenge.

The basic elements of neural networks are neurons connected together. In feedforward neural networks, neurons are organized in layers. The input layer encodes input data (an image in image classification tasks, two images in the present problem) and the output layer encodes the associated label (the posterior probability of a class in classification, a displacement field here). The intermediate layers are the hidden layers. They are made of neurons which output a signal. Each neuron of the hidden and output layers are connected to neurons from the preceding layer. The output of these neurons are a weighted sum of the connected neurons modulated by a continuous non-decreasing function called activation function, except for the output layer in a regression problem as here where no activation is involved. The most popular activation function is the so-called Rectified Linear Unit (ReLU) [13].

Deep neural networks are called “deep" because they may be made of several tens of layers. As we have seen, neural connections between layers involve weights. Note that “weights" are also commonly referred to as the “network parameters" in the literature. The term “weight" is used here to avoid confusion with the other parameters defined in the paper.

Computer vision applications typically call for a subclass of deep neural networks known as convolutional neural networks (CNNs) where the number of weights is mitigated by imposing that the weighted sums correspond to convolutions, which turn out to be the basic operators of signal processing. Neurons from a convolutional layer are represented as the output of the implemented convolutions, hence the blue parallelepipeds in Figure 5.1. Another ingredient used in CNN is down-sampling which reduces the width of the layers involved. In the present work, down-sampling by a factor (the so-called stride) of two is obtained by computing convolutions shifted by two units instead of one unit as in a standard convolution. This explains the narrower and narrower layers in the feature extraction part in Figure 5.1. Note that in this figure, up-sampling is performed through the so-called transposed convolutions when predicting the displacement field.

However, deep CNNs are still likely to require several millions of weights. As in any supervised learning task, the weights are set by training the CNN over a dataset made of observations,

that is, pairs of inputs and corresponding expected outputs. More precisely, training, called deep learning in this context, consists in minimizing with respect to the weights the cost of errors between the expected outputs and the CNN outputs obtained from the inputs and the current weights. The error cost is measured through the so-called loss function. The optimization method implementing the training process is most of the time a variation of the mini-batch stochastic gradient descent (SGD) algorithm : small sets of observations are iteratively randomly drawn from the whole dataset, giving a so-called mini-batch, and the weights of the CNN are then slightly updated in order that the sum of the losses over each mini-batch decreases. At each iteration, the outputs of the CNN computed from the mini-batch inputs are thus supposed to get closer to the expected outputs. The magnitude of the weight update is proportional to the step size of the SGD algorithm, called learning rate in machine learning applications and denoted λ in this paper. This parameter has to be set carefully. Each complete pass over the dataset is called an epoch. Many epochs are needed to perform the full training of a CNN.

However, training a deep CNN requires a large dataset and heavy computational resources, which is simply not possible in many situations. A popular alternative approach consists in using freely available pre-trained networks, that is, CNN that have been trained on standard datasets, and in adjusting their weights to the problem of interest. Adjustment can be performed by fine tuning or transfer learning. The former consists in marginally changing the weights in order to adapt them to the problem of interest and its dataset. The latter consists in changing a part of the architecture of the pre-trained network and in learning the corresponding weights from the problem of interest, the remaining weights being kept constant. Let us now examine how such networks have been used in the literature for solving a classic computer vision problem similar to ours, namely optical flow estimation.

5.3 A brief review of CNN-based methods for optical flow estimation

Optical flow is the apparent displacement field obtained from two views of a scene. It is caused by the relative motion of the observer and objects in the scene which may move or deform. Recent algorithms of optical flow estimation based on CNNs provide a promising alternative to the methods classically used to resolve this problem in computer vision after the seminal papers by Horn and Schunck [14] or by Lucas and Kanade [15]. As a typical machine learning setup, AI-based optical flow estimation algorithms can be divided into three categories : unsupervised, semi-supervised, or supervised. Unsupervised [16, 17, 18] and semi-supervised [19, 20] methods are reviewed in the literature to address the problem of limited training data in optical flow estimation. In contrast, these methods do not yet reach the accuracy of their supervised counterparts. Furthermore, supervised methods are the predominant way of learning, as described in the preceding section, and generally provide good performance. However, they require a large amount of accurate, ground-truth optical flow measurements for training. Most accurate

models use CNNs as one of the components of their system, as in DCFlow [21], MRFlow [22], Deepflow [23], and Flow Fields [24]. None of these previous approaches provide end-to-end trainable models or real-time processing performance. The most efficient algorithms recently proposed in the literature for optical flow estimation are reviewed below. In general, they share the same architecture (a schematic view is represented in Figure 5.1). The first part of the network extracts the features of the images and the optical flow is predicted through an up-sampling process in the second part of the network.

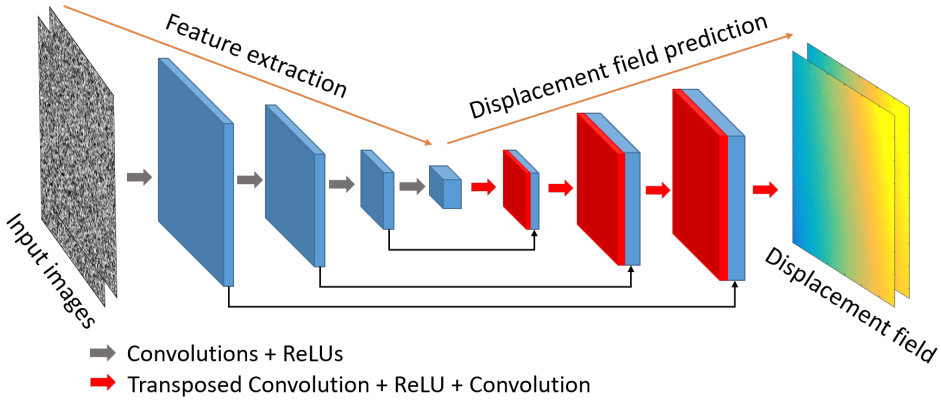


FIGURE 5.1 – Schematic view of a convolutional neural network. Anticipating the results discussed in Section 5.6.1, we present here a schematic view of StrainNet-f. The feature extraction part consists of several convolution layers followed by a ReLU (Rectified Linear Unit [13]). The last layer has a stride of two to perform down-sampling. Such a stack is called a level. The output of these levels can thus be represented by narrower and narrower blocks, in blue here. In the displacement prediction part, the levels are made of transposed convolution layers (for up-sampling) and convolution layers. The input of each level of this part is the output of the preceding level, concatenated to the output of the levels from the feature extraction part, as represented by the black arrows.

Dosovitskiy *et al.* [12] presented two CNNs called FlowNetS and FlowNetC to learn the optical flow from synthetic dataset. These two CNNs are constructed based on the U-Net architecture [25]. FlowNetS is an end-to-end CNN-based network. It concatenates the reference and the current images together and feeds them to the network in order to extract the optical flow. In contrast, FlowNetC creates two separate processing pipelines for each of these two images. Then, it combines them with a correlation layer that performs multiplicative patch comparisons between the two generated feature maps. The resolution of the output optical flow in both networks is equal to 1/4 of the image resolution. Indeed, the authors explain that adding other layers to reach full resolution is computationally expensive, and does not really improve the accuracy. A consequence is that with this network, the optical flow at full resolution is obtained by performing a bilinear interpolation with an upscale factor of 4. This point is discussed further below as we do not use exactly the same approach to solve the problem addressed in this paper.

Hu *et al.* [26] proposed a recurrent spatial pyramid network that inputs the full-resolution images and generates an initial optical flow of 1/8 full-resolution. The initial flow is then upscaled and refined recurrently based on an energy function. The initial generated optical flow is

converted to reach the full-resolution flow by performing this recurrent operation on the spatial pyramid. This network achieves comparable performance to FlowNetS with 90 times less weights. It is however twice as slow as FlowNetS because of the recurrent operations performed on the spatial pyramid. Note that when CNNs are compared, it is important to use the same image size and the same GPU for all of them. In addition, the computing time, or inference time, is generally considered as equal to the mean time value over a certain number of pairs of images.

FlowNet 2.0 [27] is the extension of FlowNet [12]. It stacks multiple FlowNetS and FlowNetC networks in order to obtain a more accurate model. FlowNet 2.0 reduces the estimation error by more than 50% compared to FlowNet. However it has three limitations. First, the model size is 4 times larger than the original FlowNet (over 160 million weights). Second, FlowNet 2.0 is 4 times slower than FlowNet. Third, the sub-networks need to be trained sequentially to reduce overfitting problem. Overfitting is a non-desirable phenomenon, which occurs in deep learning when the cardinality of the dataset is too small with respect to the number of weights to be trained. This leads to a network, which fails in correctly predicting the optical flow from images which are too different from those contained in the training dataset.

Ranjan *et al.* [28] developed a compact spatial pyramid network, called SpyNet. It combines deep learning with classical optical flow estimation principle such as image warping at each pyramid level, in order to reduce the model size. In SpyNet, the pyramid levels are trained independently. Then, each level uses the previous one as an initialization to warp the second image toward the first through the spatial pyramid network. It achieves similar performance to FlowNetC on the same benchmark but is not as accurate as FlowNetS and FlowNet2. In contrast to FlowNet [12], SpyNet model is about 26 times smaller but almost similar in terms of running speed. In addition, it is trained by level, which means that it is more difficult to train than FlowNet.

LiteFlowNet [29] is composed of two compact sub-networks specialized in pyramidal feature extraction and optical flow estimation, respectively named NetC and NetE. NetC transforms the two input images into two pyramids of multi-scale high-dimensional features. NetE consists of cascaded “flow-inference" and “regularization" modules that estimate the optical flow fields. LiteFlowNet performs better than SpyNet and FlowNet, but it is outperformed by FlowNet 2.0 in terms of accuracy. It has 30 times less weights than FlowNet 2.0, but it is 1.36 times faster.

Sun *et al.* [30] proposed PWC-Net. This network is based on simple and well-established principles such as pyramidal processing, warping, and cost volume. Similarly to FlowNet 2.0, PWC-Net shows the potential of combining deep learning and optical flow knowledge. Compared to FlowNet 2.0, the model size of PWC-Net is about 17 times smaller and the inference time is divided by two. This network is also easier to train than SpyNet [28]. PWC-Net outperforms all the previous optical flow methods on the MPI Sintel [31] and KITTI 2015 [32] benchmarks.

All these studies demonstrate the usability of CNNs for estimating the optical flow. This motivated the development of similar models in order to tackle analogous problems arising in other research fields. This is typically the case in [33], where the authors used FlowNetS to

extract dense velocity fields from particle images of fluid flows. The method introduced in this reference provides promising results, but the experiments indicate that the deep learning model does not outperform classical methods in terms of accuracy. Cai *et al.* [34] developed a network based on the LiteFlowNet [29] to extract dense velocity fields from a pair of particle images. The original LiteFlowNet network is enhanced in order to improve the performance by adding more layers. This leads to more accurate results than those given in [33] at the price of extra computing time.

In the following section, we propose to employ a deep convolutional neural network to measure full-field displacement fields that occur on the surface of deformed solids. As for 2D DIC, we assume here that the deformation is plane and that the surface is patterned with a random speckle. The main difference with the examples from the literature discussed above is twofold. First a deformation occurs and second, the resulting displacements are generally small, which means that subpixel resolution must be achieved. In addition, the idea is to use this measurement in the context of metrology, meaning that the errors affecting these measurement fields must be thoroughly estimated. A specific dataset and a dedicated network christened “StrainNet" are proposed herein to reach these goals, as described in the following sections.

5.4 Dataset

5.4.1 Existing datasets

Supervised training of deep CNNs requires large datasets. In the context of optical flow, datasets are made of pairs of images together with a ground truth optical flow, which may be achieved by rendering synthetic images. In spite that generating such large datasets is a tedious task, several datasets were generated and used in previous studies on optical flow estimation [35]. The most commonly datasets used for optical flow estimation are listed in Table 5.1.

TABLE 5.1 – Commonly used optical flow datests.

Dataset	Number of frames	Resolution	Displacements
Middlebury [36]	72	varies	Small ($\leq$ 10 pixels)
KITTI2012 [37]	194	1226×370	Large
KITTI2015 [32]	800	1242×375	Large
MPI Sintel [31]	1064	1024×436	Small and large
FlyingThings3D [38]	21818	960×540	Small and large
Flying Chairs [12]	22782	384×512	Small and large

Almost all existing datasets are limited to large rigid-body motions or quasi-rigid motions and deal with natural images. Hence, an appropriate dataset consisting of deformed speckle images had to be developed in the present study. This dataset should be made of pairs of images mimicking typical reference and deformed speckles, as well as their associated deformation fields. It has to be representative of real images and deformations so that the resulting CNN has

a chance to perform well when inferring the deformation field from a new pair of images that are not in the training dataset. Speckle images have a smaller variability than natural images processed by computer vision applications. However, we shall see that we cannot use smaller datasets as the one of Table 5.1 because we seek tiny displacements.

5.4.2 Developing a speckle dataset

In order to estimate very small subpixel displacements (of the order of 10^{-2} pixel), we propose a new dataset called Speckle dataset [2]. Two versions were developed in this study. The first one, referred to as Speckle dataset 1.0 in the following, contains 36,663 pairs of frames (this number is justified below) with their corresponding subpixel displacements fields. A second version called Speckle dataset 2.0 was also developed, as explained later in this paper. Speckle dataset 1.0 was generated as follows :

1. Generating speckle reference frames.
2. Defining the displacement fields.
3. Generating the deformed frames.

These different steps are detailed in turn in the following subsections.

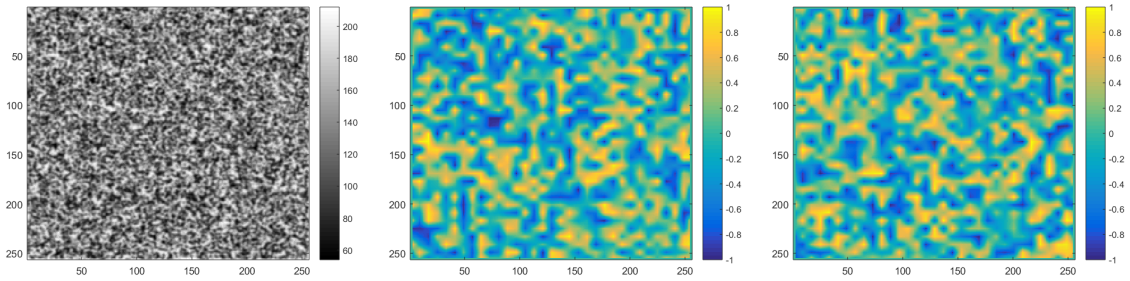
5.4.2.0.1 Generating reference speckle frames In real experiments, the first step is to prepare the specimen, often by spray-painting the surface in order to deposit black droplets on a white surface. This process was mimicked here by using the speckle generator proposed in [39]. This generator consists in randomly rendering small black disks superimposed in an image, in order to get synthetic speckled patterns that are similar to actual ones. Reference frames of size 256×256 pixels were rendered with this tool. This frame size influences the quality of the results, as discussed in Section 5.6.1. These reference frames were obtained here by mixing the different settings or parameters which are listed in Table 5.2 (see [39] for more details).

TABLE 5.2 – Parameters used to render the images for Speckle dataset 1.0 and 2.0 (second column), and the images deformed through the Star displacement of Section 5.5.3 (third column).

	Speckle datasets 1.0 and 2.0	Star images
Probability distribution function of the radius of the disks	Uniform, Exponential and Poisson	Exponential
Average radius of disks	0.45 to 0.8 pixel by step of 0.025	0.5
Average number of disks per image	36,000	556,667
Contrast of the speckle	0.5 to 1 by step of 0.05	0.6
Size of the images	256×256	501×2000
Average number of disks per pixel	0.549	0.556

2. Reference speckle frames used in speckle dataset generation and Matlab code are available at `github.com/DreamIP/StrainNet`

The quality of these reference frames was visually estimated and those featuring very large spots or poor contrast were eliminated. Only 363 frames were kept for the next step. Figure 5.2-a shows a typical speckle obtained with our generator and used to build Speckle dataset 1.0. It can be visually checked that the aspect is similar to a real speckle used for DIC in experimental mechanics, see typical examples in [1].



(a) Reference speckle image (b) Random displacements along the x axis (c) Random displacements along the y axis

FIGURE 5.2 – Typical synthetic speckle images and random displacement (in pixels) along x and y used in Speckle dataset 1.0. All dimensions are in pixels.

5.4.2.0.2 Defining the displacements The following strategy was used in order to generate a rich set of displacement fields covering the widest possible types of displacement fields. The first step consisted in splitting each reference frame into adjacent square regions of size 8×8 pixels. The pixel located at the corners of these regions were affected by a random displacement. The second step consisted in linearly interpolating the displacements between all these pixels to get the displacement at the remaining pixels of each square region.

Since we were interested here in estimating subpixel displacements, the imposed random displacements lied between -1 and +1 pixel. Furthermore, the displacements at pixels located along the boundary were set to zero to limit the error due to the missing information in these regions. Figures 5.2-b and -c show typical random displacements used to deform reference speckle images to obtain their deformed counterparts.

5.4.2.0.3 Generating the deformed frames Each of the 363 reference frames was deformed 101 times, each time by generating a new random displacement field. $363 \times 100 = 36,300$ pairs of frames were used to form the dataset on which the network was trained. 363 other pairs were used for validation purposes in order to assess the ability of the network to render faithful displacement fields after training. As in other studies dealing with optical flow estimation, this ability is quantified by calculating the so-called Average Endpoint Error (AEE). AEE is the Euclidean distance between predicted flow vector and ground truth, averaged over all the pixels. Thus

$$\text{AEE} = \frac{1}{KL} \sum_{i=1}^{K} \sum_{j=1}^{L} \sqrt{(u_e(i,j) - u_g(i,j))^2 + (v_e(i,j) - v_g(i,j))^2} \tag{5.1}$$

where (u_e, v_e) and (u_g, v_g) are the estimated optical flow and the ground truth, respectively, defined at each pixel coordinate (i, j). K and L are the dimensions of the zone over which the AEE value is calculated. This quantity is in pixel, the displacements being expressed in pixel.

The main interest of the speckle generator described in [39] is that it does not rely on interpolation when deforming the artificial speckled patterns in order to prevent additional biases caused by interpolation. However, since it relies on a Monte Carlo integration scheme, a long computing time is required. This generator is therefore only suitable for the generation of a limited number of pairs of synthetic reference and deformed images. Such an approach is not adequate here since several thousands of images had to be generated. The solution adopted here was to use bi-cubic interpolation to deform the reference images through the randomly-defined displacement fields, considering that it yielded a reasonable trade-off between computing time and quality of the results. Hence the speckle generator described in [39] was only employed to generate the reference frames mimicking real speckles used in experimental mechanics.

5.5 Fine-tuning networks of the literature

In general, CNNs are developed for specific applications and trained with datasets suited to the problem to be solved. The question is therefore to know if the studies reviewed in Section 5.3 above, which have good performance for estimating large displacements, can be fully or partially used to address the problem of the present paper, namely, resolving subpixel ($\sim$ 0.01 pixel) displacements. Transfer learning and fine tuning are possibilities to respond to this question, as explained in Section 5.2.

When classification is the ultimate goal, transfer learning is carried out by replacing only the last fully-connected layer and training the model by updating the weights of the replaced layer, while maintaining the same weights for the convolutional layers. The idea behind this is that if the dataset at hand is similar to the training dataset, the output of the convolutional layers keeps on being relevant for the problem of interest. Since our dataset is completely different from the datasets used for training the CNNs described in the studies reviewed above, we proceeded to a fine tuning by updating the weights of all the layers and not only the last one. The weights found in the cases discussed above were considered as starting points for the fine-tuning performed on our own dataset.

In the following, we discuss the fine tuning of the networks considered as the most interesting ones in terms of accuracy and computational efficiency. We did not choose to fine-tune Flownet 2.0 because of its high computational cost, as discussed in Section 5.3. Instead, we mainly relied on the FlowNet, PWC-NET, and LiteFlownet models [12, 30, 29] because they exhibit better computational efficiency during both fine tuning and inference. In practice, we used the PyTorch implementations of these models provided in [40], [41], and [42], respectively.

5.5.1 Training strategy

The following strategy was used in order to fine-tune the networks proposed in [40, 41, 42] :

1. Keeping the network architecture, loss function, and optimization method of the original model.
2. Increasing the batch size to 16 in order to speed up the training process on our hardware configuration.
3. Initiating the learning rate (the specific machine learning vocabulary is defined in Section 5.2 above) with $\lambda = 10^{-4}$, and then dividing this rate by 2 every 40 epochs. These values correspond to a good trade-off between computing time and accuracy of the results.
4. Fine-tuning each network for 300 epochs because this is a value for which the value of the loss function of the considered models stabilizes.

With this strategy, fine-tuning each of the networks on an NVIDIA TESLA V100 GPU required between three to four days, depending on the model complexity.

5.5.2 Obtained results

The AEE value defined in Equation 5.1 was used to evaluate the fine-tuning results of the different networks. It was averaged over the test dataset made of 363 displacement fields deduced from the 363 pairs of reference and deformed speckle images considered for validation purposes. Figure 5.3 gives the average AEE value for each network trained on the Speckle dataset 1.0. The main conclusion is that the average AEE value is much lower with FlowNetS, which hints that more accurate estimations are expected with this network. It has also been observed that the average AEE was about the same for both the train and the test datasets for all the networks under consideration (for this reason, Figure 5.3 only reports the AEE of the test dataset), suggesting that they do not suffer from overfitting. Finally, the main conclusion is that the AAE value is equal at best to 0.1 pixel, which is too high since a resolution of 0.01 pixel is expected to be obtained to reach a similar performance to DIC. Since fine-tuning was not sufficient, these preexisting networks had to be improved, as explained in Section 5.6.

5.5.3 Assessing the results with a reference Star displacement field

Another synthetic displacement field than the interpolated random field described above has been proposed in recent papers to assess the metrological performance of various full-field measurement methods used in experimental mechanics [43, 7, 44, 10, 4]. This displacement field, named “Star displacement" in the following to be consistent with the name given in [4], is a synthetic vertical displacement modelled by a sine wave, whose period gently and linearly increases when going toward the right-hand side of the map. This vertical reference displacement field v_g is shown in Figure 5.4b (u_g is null in this case). In this figure, the green rectangle is the

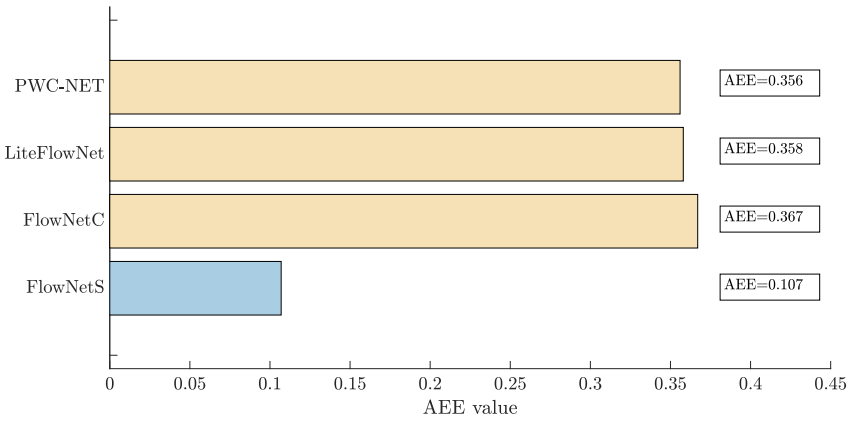
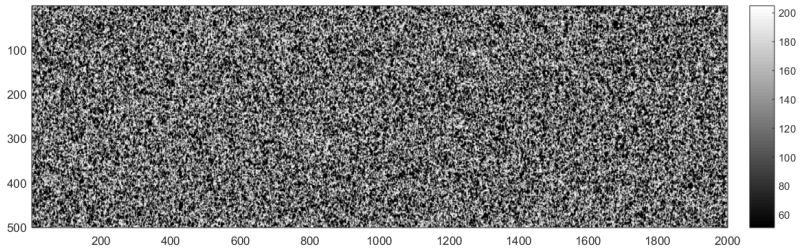
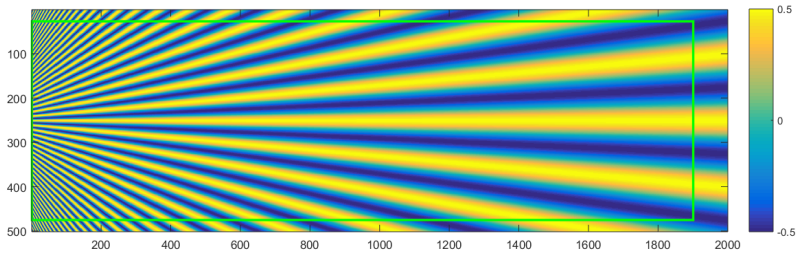


FIGURE 5.3 – Average AEE value for four networks trained on Speckle dataset 1.0.

zone over which the results are numerically evaluated. It is not centered to emphasize the influence of the high spatial frequencies and to account for some practical constraints imposed by the use of LiteFlowNet and PWC-NET. The speckle images deformed through this displacement field are named “Star images". They were obtained by using the generator described in [39]. It means that no interpolation was performed, which was not the case for the deformed images of the dataset. The parameters used to render these speckle images are given in Table 5.2, third column. It can be seen that the average number of disks per pixel is nearly the same for all types of images.



(a) Reference image



(b) Reference displacement

FIGURE 5.4 – (a) Reference image corresponding to (b) the Star displacement. The green rectangle is used in this work to evaluate the results. All dimensions are in pixels.

The benefit of using images deformed with this displacement field is that the effect of the spatial frequency on the measured displacement field can be easily assessed by observing the attenuation of the displacement amplitude when going toward the left-hand side of the map. In

other words, this reference displacement field permits the characterization of the transfer function of the estimation process. Images deformed through this displacement field were therefore considered in this study to validate the different variants of the network we developed, and more precisely to observe the attenuation of the displacement rendered for the highest spatial frequencies located on the left.

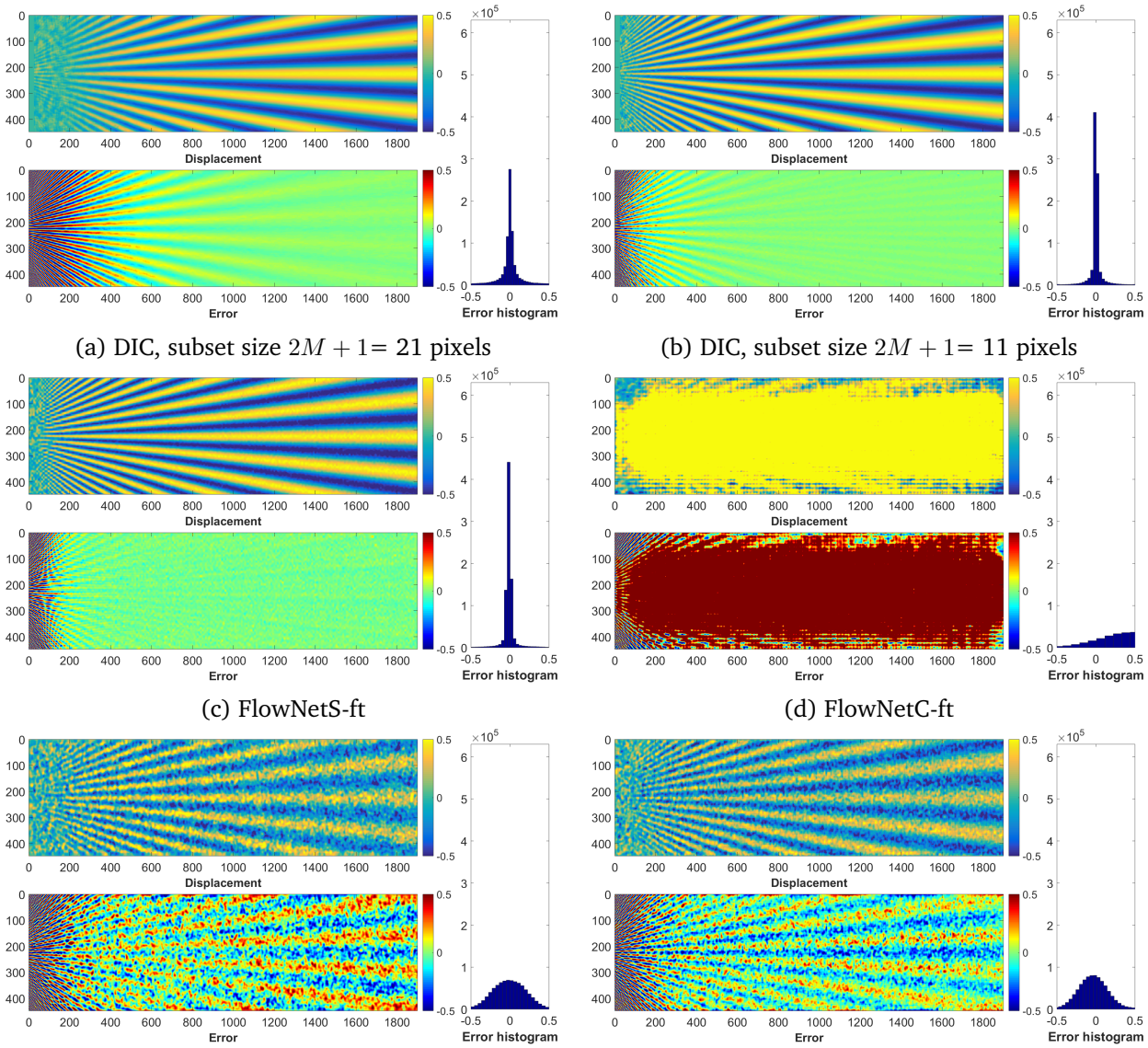
A pair of reference image and image deformed through the Star displacement field [3] corresponding to the reference displacement field was used to evaluate the results obtained with those given by the four fine-tuned CNNs selected above, namely FlowNetS, FlowNetC, LiteFlownet and PWC-NET. In the following, extension “-ft” is added to the names of the networks when they are fine-tuned. No artificial noise was added to these images. The obtained results were compared to those given by classic subset-based DIC. The latter was performed with two subset sizes, namely $2M+1=11$ and 21 pixels, where M is an integer which governs the size of the subset. Interpolation was performed by using spline functions. First-order subset shape functions, which are the most common ones in commercial codes [1], were considered here (“subset shape functions" are also called “matching functions" in the literature, but we adopt here the terminology proposed in [45]). Note that results obtained with second-order subset shape functions are also presented and discussed in Section 5.7. As in recent papers dealing with the metrological performance of full-field measurement systems [10, 46], the shift between two consecutive subsets is equal to one pixel, which puts us in a position where DIC is at its best performance level (regardless of computing time). The results obtained in these different cases are given in Figure 5.5.

The worst results are obtained with FlownetC. Those obtained with LiteFlownet and PWC-NET are better, but it is clear that FlownetS provides the best results, with a displacement map which is similar to that obtained with DIC and $2M+1=11$ pixels. The poor accuracy obtained by the first three networks (FlownetC, LiteFlownet, and PWC-Net) may be due to the fact that these networks make use of predefined blocks such as correlation or regularization, and that they were originally developed for the determination of large displacements instead of subpixel ones.

As in previous studies dealing with quality assessment of full-field measurement techniques, such as [7, 46, 4] for instance, the displacement along the horizontal axis of symmetry, denoted here by Δ, is plotted in Figure 5.6 in order to clearly see the attenuation of the signal when going to the highest spatial frequencies of the Star displacement (thus when going to the left of the displacement map), as well as the high-frequency fluctuations affecting the results. It is worth remembering that the reference value of the displacement along Δ is equal to 0.5 pixel. A blue horizontal line of ordinate 0.5 pixel is therefore superimposed to these curves to visualize this reference value. The closer the curves to this horizontal line, the better the results. Displacements obtained with DIC are also superimposed for comparison purposes. The mean absolute error obtained column-wise is also given.

It is worth noting that at high spatial frequencies (left-hand part of the graphs), FlowNetS

3. Star frames used here are available at `github.com/DreamIP/StrainNet`



(a) DIC, subset size $2M+1$= 21 pixels

(b) DIC, subset size $2M+1$= 11 pixels

(c) FlowNetS-ft

(d) FlowNetC-ft

(e) LiteFlownet-ft

(f) PWCNET-ft

FIGURE 5.5 – Star displacement obtained (a)(b) with DIC and (c)-(f) with four selected CNNs. In each case, the retrieved displacement field, the difference with the reference displacement field and the histogram of this difference are given in turn in the different sub-figures. All dimensions are in pixels.

provides results which are similar to those given by DIC with a subset of size $2M+1 = 11$ pixels. At low spatial frequencies (right-hand part of the graphs), DIC provides smoother and more accurate results than FlowNetS. Calculating in each case the mean absolute error for the vertical displacement gives an estimation of the global error over the displacement map in each case. This quantity is calculated over the green rectangle plotted in Figure 5.4b to remove boundary effects. This quantity, denoted by MAE in the following, is defined by the mean value of the absolute error throughout the field of interest. Thus

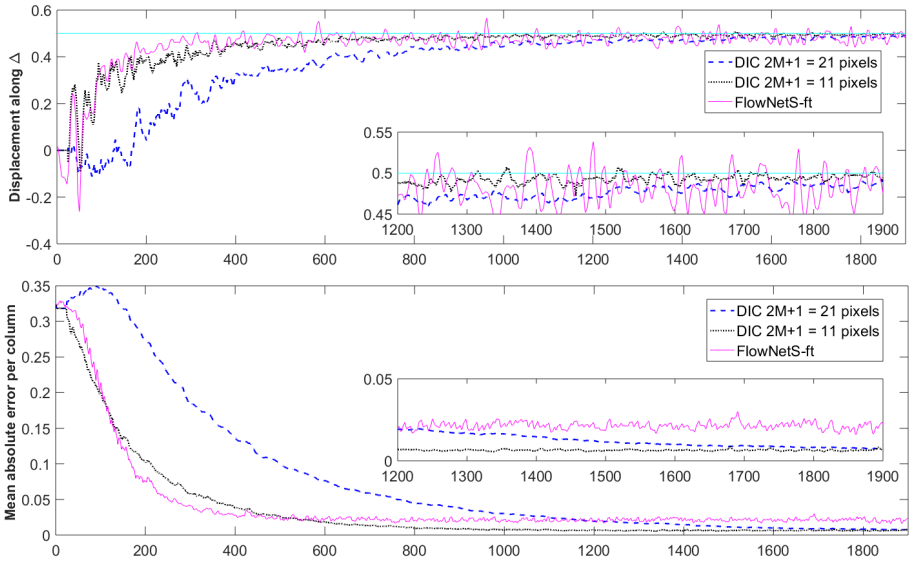


FIGURE 5.6 – Comparison between FlowNetS-ft and DIC. Top : displacement along the horizontal axis of symmetry of the Star displacement field. The horizontal blue line corresponds to the reference displacement along Δ. This reference displacement is equal to 0.5 pixels. Bottom : mean absolute error obtained column-wise. Close-up views of the rightmost part of the graphs are also inserted. All dimensions are in pixels.

$$\text{MAE} = \frac{1}{KL}\sum_{i=1}^{K}\sum_{j=1}^{L}|v_e(i,j) - v_g(i,j)| \tag{5.2}$$

where v_e and v_g are the estimated displacement and ground truth, respectively. K and L are here the dimensions of the green rectangle over which the results are calculated. MAE is equal to AEE defined in Equation 5.1, in which the horizontal displacement has been nullified. It is introduced here in order to focus on the error made along the vertical direction only, the relevant information being along this direction for the displacement maps extracted from the Star images. Table 5.3 gives this quantity calculated for DIC and FlowNetS-ft.

TABLE 5.3 – MAE (in pixels) for DIC (subset size $2M+1 = 11$ and 21 pixels) and FlowNetS-ft.

Technique used	DIC, $2M+1 = 21$ pixels	DIC, $2M+1 = 11$ pixels	FlowNetS-ft
MAE	0.0828	0.0365	0.0437

It is clear that the MAE value is the lowest for DIC with $2M+1$=11 pixels. It is followed by FlowNetS-ft. These first results are promising but they come from predefined networks which are therefore not specifically developed to resolve the problem at hand. The best candidate, namely FlowNetS-ft, was therefore selected for further improvement. This is the aim of the next section.

5.6 Tailoring FlowNetS to estimate displacement fields

Two types of modifications were proposed to enhance the accuracy of the results. The first one concerns the network, the second the dataset.

5.6.1 Improvement of the network

First, let us examine in more details the architecture of FlowNetS [12]. This network can be split into two different parts, as illustrated in Figure 5.1. The first one extracts the feature maps using 10 successive convolutional layers. The first layer uses 7×7 filters, the second and third layers use 5×5 filters, and the remaining layers use 3×3 filters. The second part predicts the optical flow at different levels and relies, in the case of FlowNetS, on 5 convolutional layers with 3×3 filters and 8 transposed convolutional layers. The transposed convolutional layers are used for the up-sampling process (see the right-hand side of the schematic view depicted in Figure 5.1). We refer the interested reader to [12] for more details on the architecture.

As mentioned in Section 5.3, the FlowNetS architecture provides an optical flow of 1/4 of the original image resolution. The full-resolution optical flow is obtained by applying an up-scaling step with a factor of 4 using a bi-linear interpolation. This not only reduces the computational complexity but also the quality of the results. In order to enhance the prediction of the optical flow, it was therefore proposed to improve the network to directly output a higher resolution optical flow, without interpolation.

In general, the deeper the network, the better the results. The dataset shall however be enlarged accordingly to avoid overfitting (see definition in Section 5.3 above)). Two approaches were examined in this study in order to enhance the accuracy of the predicted optical flow. The first one consists in adding some levels to the network (thus increasing the number of layers), the second in changing the architecture.

5.6.1.1 First approach : adding one or two levels

The first approach consists in adding levels to the network. A first option consists in adding one level to the FlowNetS network while keeping the same architecture. Consequently, the new output optical flow has a resolution divided by 2 instead of 4. This method still requires interpolation in order to reach the same resolution as the input images. A second option has also been examined. It consists in adding one more level so that the full-resolution is directly obtained without any interpolation.

The loss function defined in Section 5.2 is equal to the following quantity

$$Loss = \sum_{i=1}^{n} \lambda_i e_i \tag{5.3}$$

where n is the number of levels forming the network, λ_i is a weighting coefficient corresponding to the i-th level, and e_i is the AEE between of the output of the i-th level and the ground-truth

displacement sampled at the same resolution. This loss function was adapted to the proposed networks by keeping the same λ_i coefficients as in [12] for the levels corresponding to the FlowNetS levels, and affecting a coefficient of 0.003 to each new level. Compared to the strategy defined in Section 5.5.1, only the loss function is modified. Two training scenarios were used here. In the first one, the new networks were fine-tuned by updating all the weights of the networks. In the second scenario, only the weights of the new levels were updated. The remaining weights were the same as those obtained after applying the fine-tuning process described in the previous section.

By applying the first scenario for training the new network with one additional level only, the average AEE value defined in Section 5.5.2 increases from 0.107 to 0.141, which means that the results are worse. The same trend is observed with the MAE value deduced from the displacement map obtained with the Star images (MAE = 0.3334). The displacement map obtained in this case is shown in Figure 5.7. It can be seen that the error made is significant. Adding one more level and retraining all the network weights does not improve the quality of the results, which means that the first scenario is not a good option for training the networks.

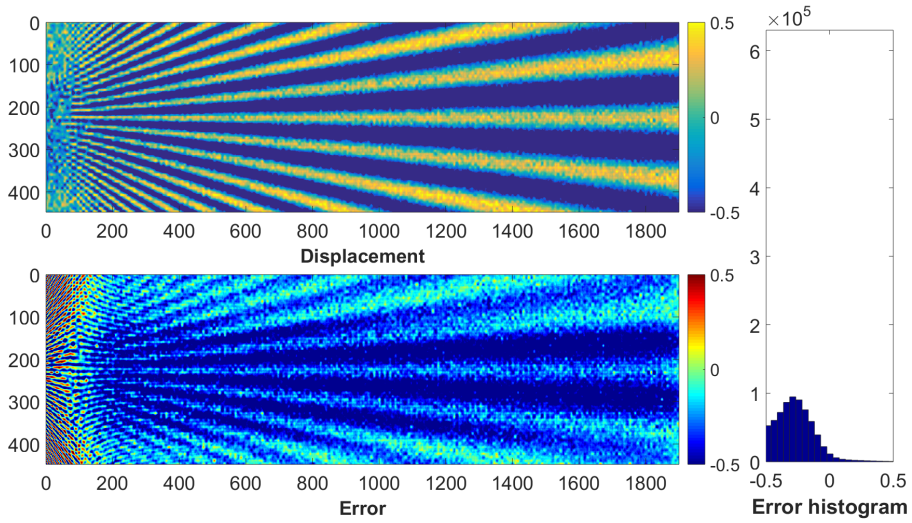


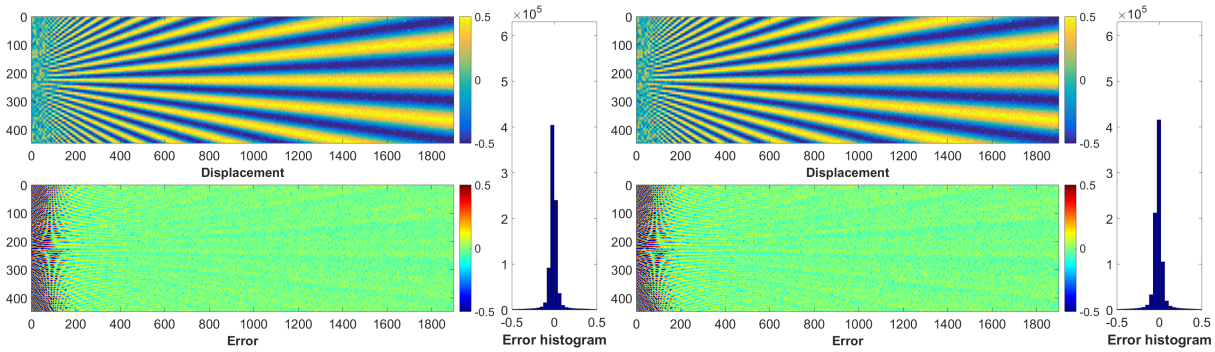
FIGURE 5.7 – Star displacement obtained by adding one level only to FlowNetS, performing interpolation to reach full resolution and updating all the weights, (MAE = 0.3334). All dimensions are in pixels.

On the contrary, the second scenario really improves the accuracy of the results when considering the random displacement fields used to generate the 363 deformed speckle images considered as test dataset, with a reduction of the average AEE of more than 50% compared to FowNetS-ft, as reported in Table 5.4. The average AEE concerns the 363 displacement fields retrieved from the 363 pairs of images of the test dataset. When considering now the Star displacement, we can see that the MAE reported in Table 5.4 for each of these two new networks is nearly the same as the MAE obtained with FlowNetS-ft. This is confirmed by visually comparing the Star displacement reported in Figure 5.8 which are obtained with these two new networks, and the reference Star displacement depicted in Figure 5.4b. Besides, it can be seen

in Figure 5.8 that almost the same accuracy as FlowNetS-ft is obtained when considering the Star displacement field of Figure 5.4b.

TABLE 5.4 – Comparison between *i*- FlowNetS-ft, *ii*- DIC with $2M+1=11$ pixels and *iii*- FlowNetS after adding one or two levels and updating only the weights of the new levels. Average AEE calculated over the whole images of the test dataset and MAE calculated with the Star displacement.

Metric	FlowNetS-ft	DIC, $2M+1=11$ pixels	Network with one additional level	Network with two additional levels
Average AEE	0.1070	-	0.0560	0.0450
MAE	0.0437	0.0365	0.0445	0.0471



(a) Option 1 : Adding one level to FlowNetS, MAE = 0.0445.

(b) Option 2 : Adding two levels to FlowNetS, MAE = 0.0471.

FIGURE 5.8 – Star displacement obtained by adding one or two levels to FlowNetS and updating only the weights of the new levels. All dimensions are in pixels.

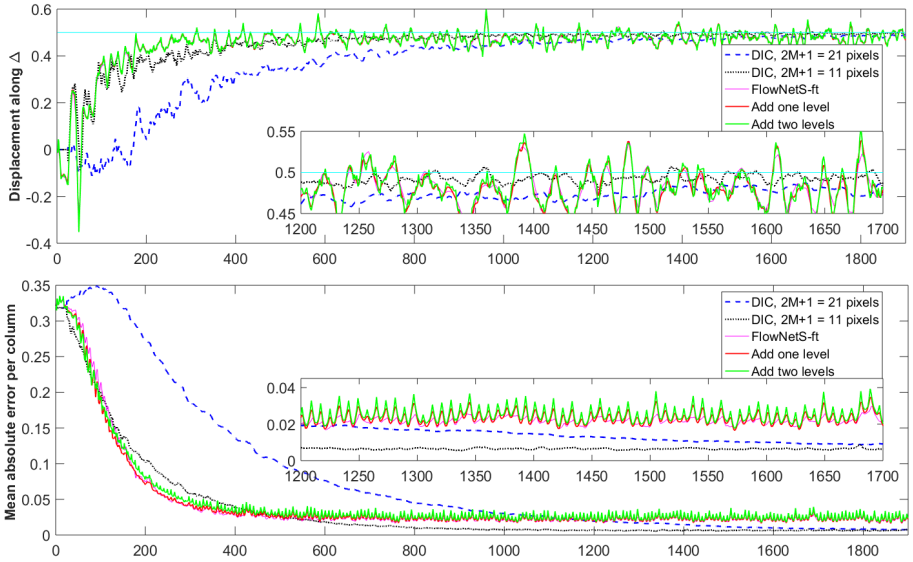


FIGURE 5.9 – Comparison between *i*- FlowNetS-ft, *ii*- FlowNetS after adding one or two levels and updating only the weights of the new levels, and *iii*- DIC with $2M+1=11$ and $2M+1=21$ pixels. Displacements along Δ and mean absolute error per column.

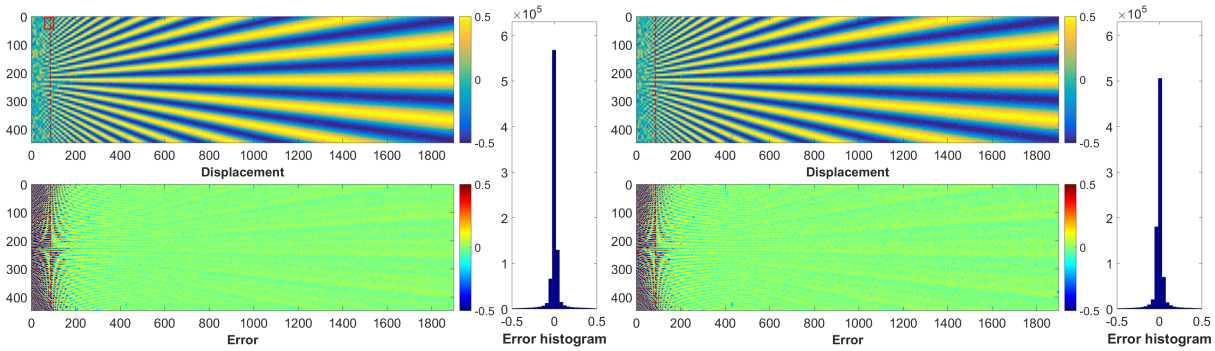
The displacements along the horizontal axis of symmetry Δ of the Star displacements which

are obtained with these two networks, FlowNetS-ft, and DIC ($2M+1=11$ and $2M+1=21$ pixels) are depicted in Figure 5.9 to more easily compare the different techniques. No real improvement is clearly observed with these curves. The mean absolute error estimated column-wise is lower for the proposed networks than for DIC ($2M+1=11$ pixels) for medium-range spatial frequencies, between $x=200$ and $x=400$, but this error becomes higher for $x>600$. The obtained results show that the networks proposed here enhance the learning accuracy on Speckle dataset 1.0 compared to FlowNetS-ft but no improvement is observed with the Star displacement.

5.6.1.2 Second approach : Changing the architecture of FlowNetS

In FlowNetS, the feature maps are down-sampled 6 times (6 strides of 2 pixels), which means that the resolution of the last feature maps is 2^6 smaller than the resolution of the images which are processed. Then, optical flow estimation and up-sampling processing are performed only 4 times, which leads to output an optical flow with a resolution divided by 4. We also propose two alternative networks with this second approach. The first one, named StrainNet-f, is a full-resolution network obtained by applying 4 down-samplings followed by 4 up-samplings. The second network, named StrainNet-h, is a half-resolution network obtained by applying 5 down-samplings followed by 4 up-samplings. The same loss function as in FlowNetS is used in both cases. These two networks are trained by using the weights of the corresponding levels of FlowNetS-ft as starting values, and then by fine-tuning all the network weights. The same training strategy as that described in Section 5.5.1 was adopted. The average AEE and MAE values reported in Table 5.5 clearly show that these two proposed networks outperform the previous ones, even though no real difference is visible to the naked eye between the Star displacements reported in Figure 5.10 and those reported in Figure 5.8. This is clearer when observing the displacements along Δ reported in Figure 5.11 and comparing them with their counterparts in Figure 5.9. Indeed the sharp fluctuations visible in the closeup view embodied in Figure 5.11 are much smaller and smoother than those shown in Figure 5.9. In addition, the MAE per column is lower in the latter than in the former at high frequencies (for about $200<x<400$). Finally, the main conclusion of the different metrics given in Tables 5.4 and maps or curves showed in Figure 5.10 and 5.11 is that the last two networks perform better than DIC ($2M+1=11$ and 21 pixels, 1st order) at high spatial frequencies, and that they provide comparable results at low frequencies. Let us now examine how to improve further the results by changing the dataset used to train the networks.

Different methods were investigated in this section in order to improve the network. Table 5.6 gathers all these methods and the corresponding results in order to help the reader compare them and easily find the results obtained in each case. The results can also be improved by changing the dataset. This is the aim of the next section.



(a) First network : half-resolution network StrainNet-h, MAE = 0.0350.

(b) Second network : full-resolution network StrainNet-f, MAE = 0.0361.

FIGURE 5.10 – Star displacement obtained after changing the architecture of FlowNetS with two options : (a) half-resolution and (b) full resolution, and updating all the weights. All dimensions are in pixels. The abscissa of the vertical red line is such that the period of the sine wave is equal to 16 pixels, thus twice the size of the regions used to define the displacement maps gathered in Speckle dataset 1.0. The red square at the top left is the zone considered for plotting the closeup view in Figure 5.12.

TABLE 5.5 – Comparison between *i*- FlowNetS-ft, *ii*- DIC with $2M + 1 = 11$ pixels, and *iii*- FlowNetS after changing the architecture with two options : half-resolution (StrainNet-h) and full resolution (StrainNet-f), and updating all the weights. Average AEE and MAE (in pixels), calculated over the whole images of the test dataset.

Metric	FlowNetS-ft	DIC, $2M + 1 = 11$ pixels	Half-resolution network	Full-resolution network
Average AEE	0.1070	-	0.0352	0.0211
MAE	0.0437	0.0365	0.0350	0.0361

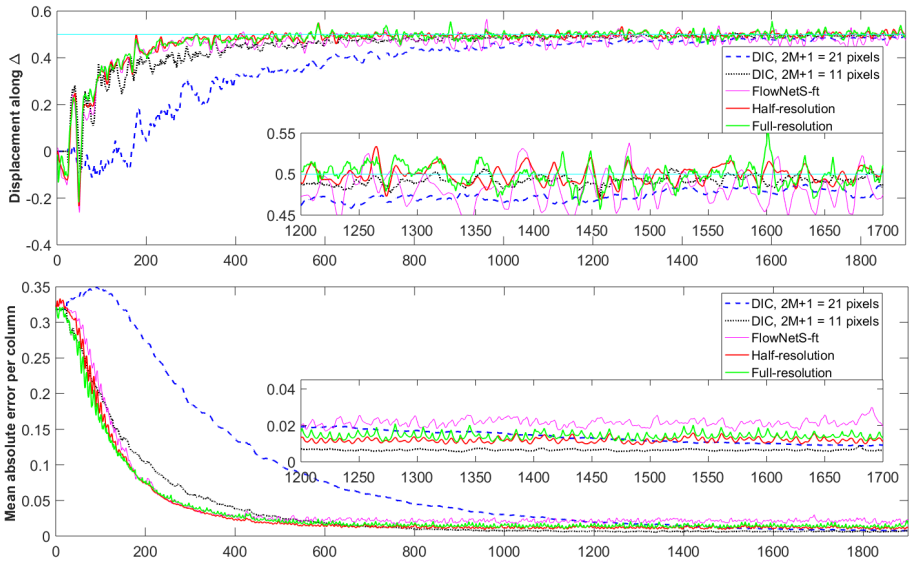


FIGURE 5.11 – Comparison between *i*- FlowNetS-ft, *ii*- DIC with $2M + 1 = 11$ and 21 pixels, and *iii*- FlowNetS after changing the architecture with two options : half-resolution and full resolution, and updating all the weights. Displacements along Δ and mean absolute error per column. All dimensions are in pixels.

TABLE 5.6 – Methods and options investigated to improve the network in this study.

Approach	Option	Training scenario	Results
1- Adding Levels	one level	Updating all the weights	Fig. 7
1- Adding Levels	one level	Updating new weights only	Table 4, Fig. 8a
1- Adding Levels	two levels	Updating new weights only	Table 4, Fig. 8b
2-Changing architecture	Half-resolution (StrainNet-h)	Updating all the weights	Table 5, Fig. 10a, Fig. 11
2- Changing architecture	Full-resolution (StrainNet-f)	Updating all the weights	Table 5, Fig. 10b, Fig. 11

5.6.2 Improvement of the training dataset

5.6.2.1 Procedure

An interesting conclusion of the preceding simulations is that, regardless of the influence of sensor noise propagation to final displacement maps, the networks proposed in the first approach (see Section 5.6.1.1) enhance the average accuracy over the test dataset. Nevertheless, the accuracy of the Star displacement field returned by these networks is not improved, especially when high-frequency displacements are concerned. This suggests limited generalization capability of the proposed network, probably caused by the nature of the training dataset.

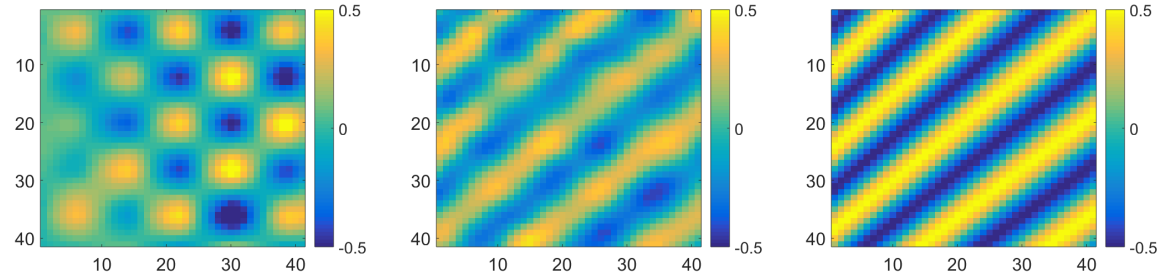
We now discuss how to change the speckle dataset so that a lower noise and a lower bias can be obtained with both StrainNet-f and -h for the highest spatial frequencies of the Star displacement.

5.6.2.1.1 Observing the bias at high frequencies in the Star displacement map

Before improving the dataset, let us examine in detail the results obtained in the preceding section. For instance, the error maps depicted in Figure 5.10 show that the error suddenly increases for the highest frequencies (those on the left). Interestingly, the location of this sudden change in response of the network substantially corresponds to the zone of the displacement map for which the spatial period of the vertical sine wave is equal to twice the size of the region used in Speckle dataset 1.0, namely $8 \times 2 = 16$ pixels. This is confirmed by superimposing to these maps a vertical red line at an abscissa which corresponds to this period of 16 pixels. The explanation is that the displacement field considered in Speckle dataset 1.0 are linear interpolation of random values 8 pixel apart. They are therefore increasing or decreasing over an 8 pixel interval, and cannot correctly represent sine waves of period lower than 16 pixels.

In the same way, let us now enlarge the displacement field obtained with StrainNet-h trained on Speckle dataset 1.0. The zone under consideration is a small square portion of the displacement field (see precise location with the red square in Figures 5.10a and 5.13a). The result is given in Figure 5.12a. It can be observed that the network is impacted by the fact that the data-

set used for training purposes is generated from 8×8 independent regions. Indeed, the network predicts the displacement at one point per region and then interpolates the results to obtain the full optical flow. This phenomenon is confirmed by down-sampling a predicted displacement with a factor of 8 and then up-sampling the result with the same factor. The resulting displacement is practically the same as the one given by StrainNet-h trained with Speckle dataset 1.0. The main conclusion is that the network cannot correctly predict the displacements on the left of the Star displacement because they occur at higher spatial frequencies than those used in Speckle dataset 1.0.



(a) StrainNet-h trained on Speckle dataset 1.0

(b) StrainNet-h trained on Speckle dataset 2.0

(c) Reference displacement

FIGURE 5.12 – Results obtained with StrainNet-h trained on Speckle dataset 1.0 and 2.0 : closeup view at high spatial frequencies area of the Star displacement (see precise location in Figures 5.10 and 5.13). (a) StrainNet-h trained on Speckle dataset 1.0. (b) StrainNet-h trained on Speckle dataset 2.0. (c) Reference displacement. All dimensions are in pixels.

A consequence of the remarks above is that square regions smaller in size than 8 pixels should also be included in the speckle dataset to be able to retrieve displacement fields involving spatial frequencies higher than 1/8 pixel^{-1}. A second and more suitable dataset called Speckle dataset 2.0 was therefore generated, as explained below.

5.6.2.1.2 Generating Speckle dataset 2.0

Speckle dataset 2.0 was generated with the same principle as Speckle dataset 1.0, but by changing three design rules. First, regions of various sizes instead of uniform size of 8×8 pixels were considered to define the random ground truth displacement fields. On the one hand, the preceding remark motivates the use of smaller regions. On the other hand, less accurate estimation than with DIC for low-frequency displacements (see for instance Figure 5.11) motivates the use of larger regions. We therefore considered regions of size equal to 4×4, 8×8, 16×16, 32×32, 64×64 and 128×128 pixels.

Second, bilinear interpolation used in Speckle dataset 1.0 was replaced by a bicubic one to limit potential interpolation bias. Third, a noise was added to all the images of the dataset in order to simulate the effect of sensor noise which always corrupts digital images, while only noiseless images were considered in Speckle dataset 1.0. This noise was heteroscedastic to mimic typical sensor noise of actual linear cameras [47, 43]. With this model, the variance v of the

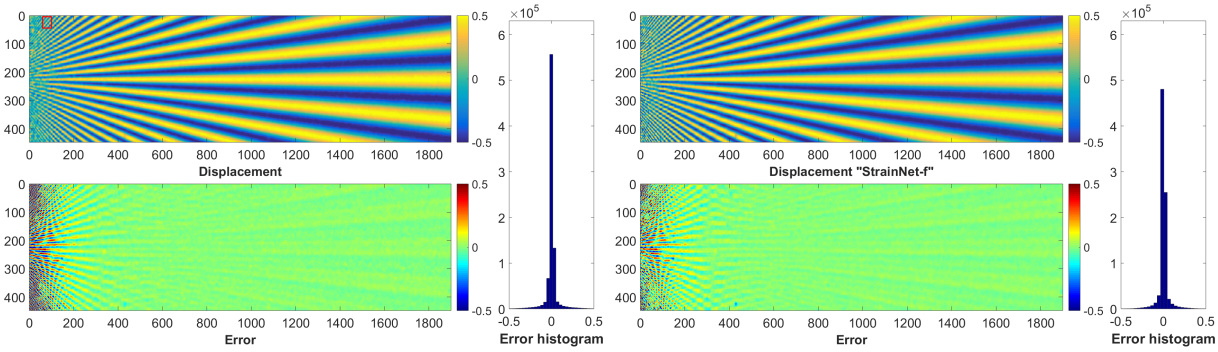
camera sensor noise is an affine function of the brightness s, so $v = a \times s + b$. We chose here a=0.0342 and b=0.2679 to be consistent with the values used to generate the noisy images used in the DIC Challenge 2.0 [4].

The number of frames was the same for each region size. It was equal to $363 \times 10 = 3,630$ (363 reference images deformed 10 times). Since six different region sizes were considered, Speckle dataset 2.0 contains $6 \times 3630 = 21780$ different pairs of images.

5.6.2.2 Results obtained with the Star images

5.6.2.2.1 Results obtained with noiseless images

StrainNet-h and StrainNet-f were trained again, but this time on Speckle dataset 2.0 instead of Speckle dataset 1.0. Processing then the noiseless Star images with these two networks gives the results illustrated in Figure 5.13-a and -b, respectively. It is worth remembering that StrainNet-f anf -h are used here to determine a displacement field from noiseless frames after being trained on noisy frames.



(a) StrainNet-h, MAE = 0.0305 (b) StrainNet-f, MAE = 0.0266

FIGURE 5.13 – Star displacement obtained with StrainNet-f and StrainNet-h trained on speckle dataset 2.0. The red square at the top left is the zone considered for plotting the closeup view in Figure 5.12. All dimensions are in pixels.

Results obtained at the right-hand side of the displacement map of Figure 5.13 are rather smooth. In addition, bearing in mind that the colorbars used in Figures 5.10 and 5.13 are the same, it can be seen by comparing the error maps that the high spatial frequencies are rendered in a more accurate way with the networks trained on Speckle dataset 2.0, in particular the high-frequency components in the left-hand side of the displacement maps.

Figure 5.14 compares the results obtained by StrainNet-h and StrainNet-f trained in turn on Speckle datasets 1.0 and 2.0. The results obtained by these networks trained on Speckle dataset 2.0 are also compared in Figure 5.15 with DIC (subset size $2M + 1 = 11$ and 21 pixels). It is clear that the results obtained after training on Speckle dataset 2.0 are better and have less fluctuations of the error. Furthermore, the results shown in Figure 5.15 show that StrainNet-h and StrainNet-f have a similar accuracy to DIC at low frequencies and a better one at high fre-

quencies.

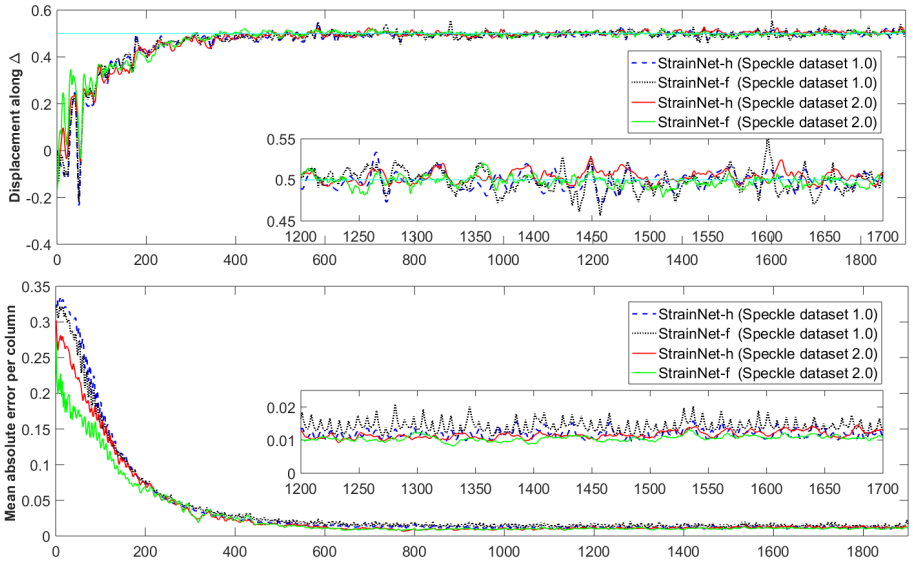


FIGURE 5.14 – Comparison between the networks trained on Speckle dataset 1.0 and Speckle dataset2.0.

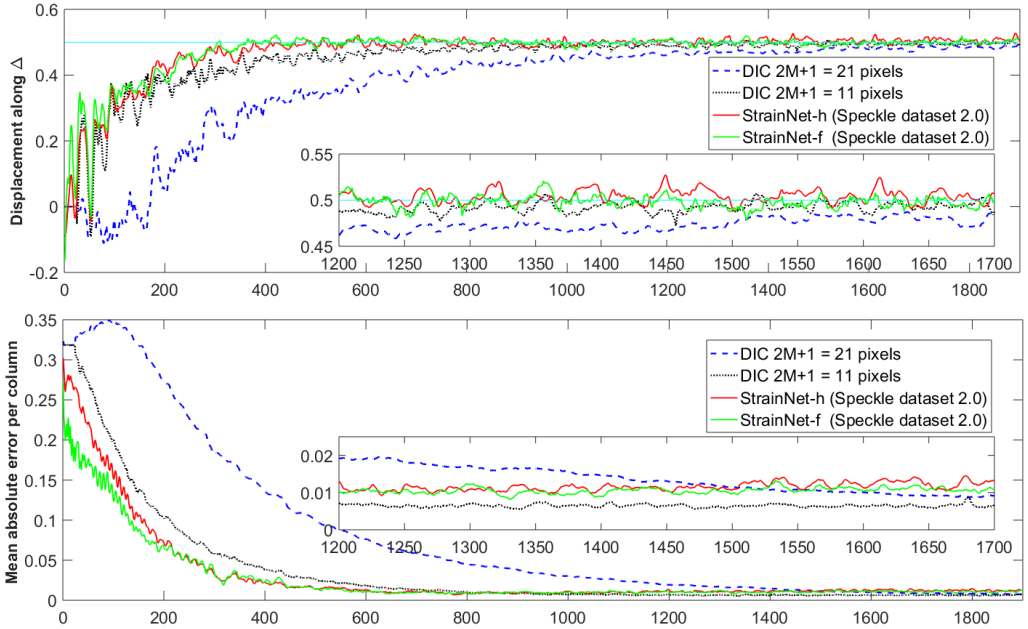
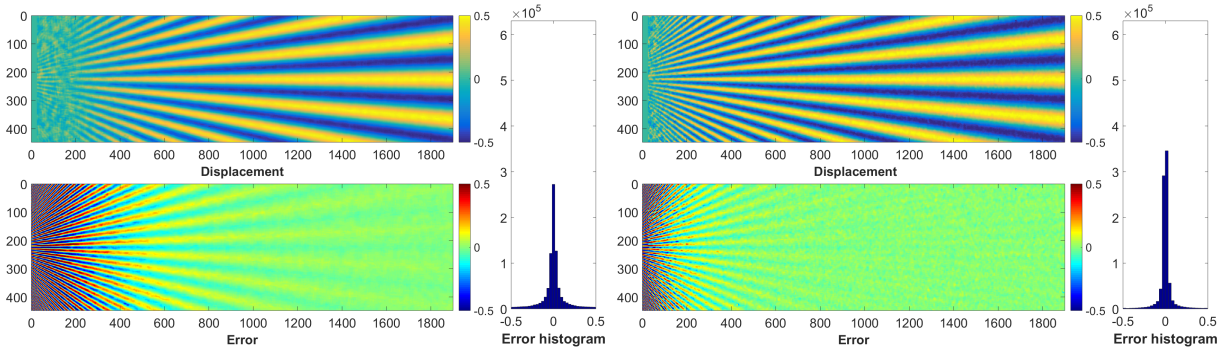


FIGURE 5.15 – Comparison between the networks trained on Speckle dataset 2.0 and DICs ($2M+1 = 11$ and 21 pixels). Top : displacements along Δ. Bottom : mean absolute error per column. All dimensions are in pixels.

Finally, it can be concluded that training the network with Speckle dataset 2.0 instead of Speckle dataset 1.0 leads to better results with the Star images. Let us now examine the influence of image noise on the results.

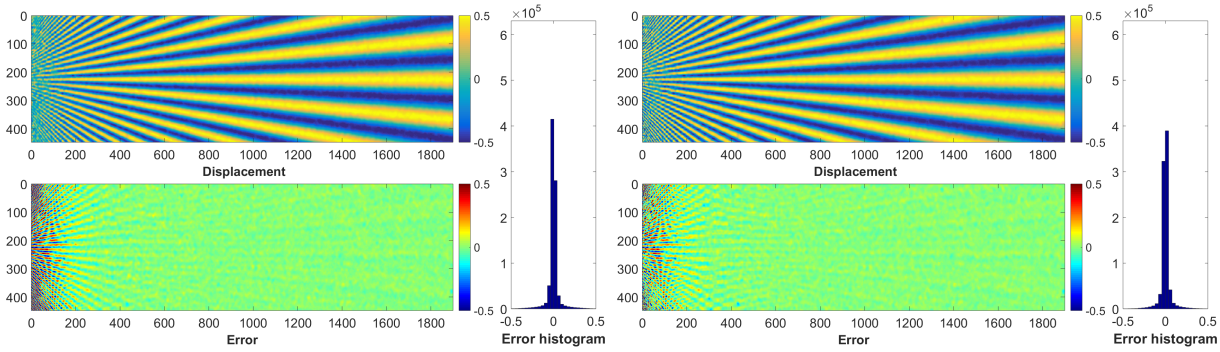
5.6.2.2.2 Results obtained with noisy images

The previous results were obtained with noiseless images. We now consider noisy images to evaluate the dependency of the maps provided by StrainNet-f and -h to image noise. The Star images used in this case were obtained by adding a heteroscedastic noise similar to the one discussed above to the noiseless Star images.



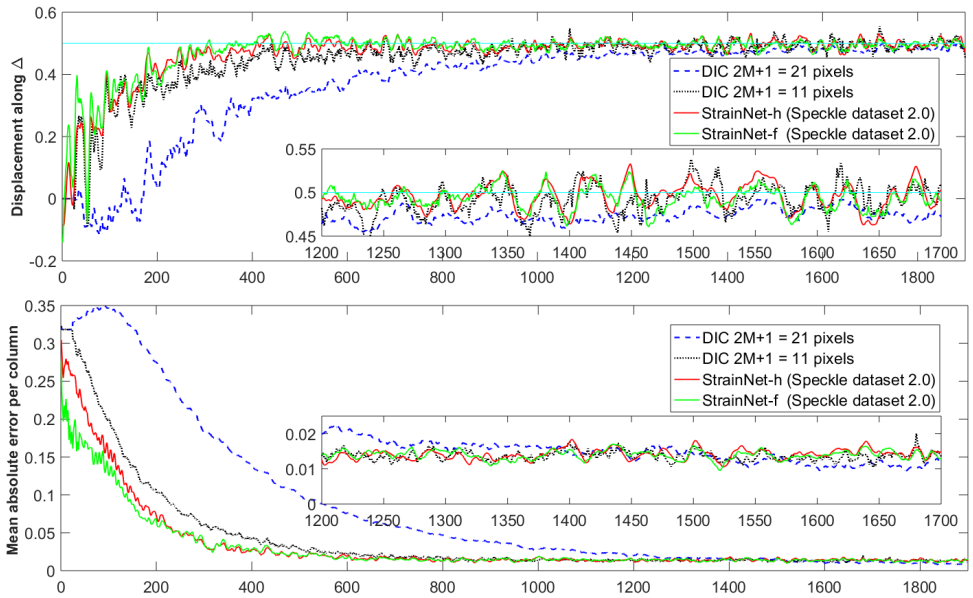
(a) DIC, $2M + 1 = 21$, MAE = 0.0834

(b) DIC, $2M + 1 = 11$, MAE = 0.0417



(c) StrainNet-h, MAE = 0.0333

(d) StrainNet-f, MAE = 0.0299



(e) Displacement along Δ and mean absolute error per column. All dimensions are in pixels.

FIGURE 5.16 – Results given by DIC and StrainNet with noisy images.

Figure 5.16 shows the maps obtained with StrainNet-h and StrainNet-f with the Star images, as well as those obtained with DIC for comparison purposes ($2M+1=11$ and 21 pixels). It is worth noting that StrainNet-h and StrainNet-f outperform DIC $2M+1=11$ pixels at both high and low spatial frequencies. Indeed, the bias is lower for the high spatial frequencies and a smoother displacement distribution (thus a lower noise) is obtained.

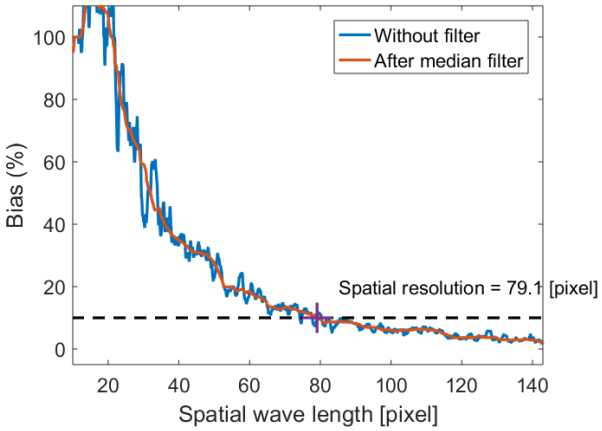
5.7 Spatial resolution and metrological performance indicator

5.7.1 Spatial resolution

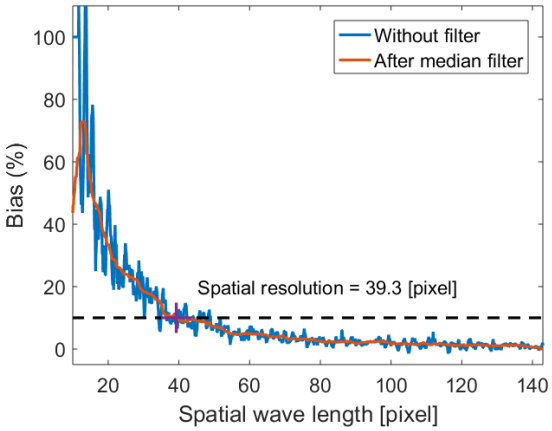
As in other studies dealing with the metrological performance of full-field measurement techniques [10, 46], the spatial resolution of each algorithm is estimated here by considering the displacements along the horizontal axis of symmetry Δ. The reference value is 0.5 pixels all along this line, but the measured one becomes lower when going toward the high frequencies of the star displacement, thus toward the left of the map. The attenuation of the signal, which directly reflects the bias, depends on the spatial frequency. The spatial resolution, denoted here by d, is generally defined as the inverse of the spatial frequency obtained for a given attenuation of the amplitude of the vertical sine displacement. This attenuation is generally equal to 10%=0.1. In the present case, it means that the spatial resolution is the period of the vertical sine displacement for which the amplitude is equal to $0.5-0.5\times0.1=0.45$ pixel. The value of d must be as small as possible to reflect a small spatial resolution, thus the ability of the measuring technique of interest to distinguish close features in displacement and strain maps, and return a value of the displacements and strains in these regions with a small bias. In certain cases, the displacement resolution can be predicted theoretically from the transfer function of the filter associated to the technique (Savitzky-Golay filter for DIC [5, 6], Gaussian filter for the Localized Spectrum Analysis [48]).

In the present case of CNN however, no transfer function has been identified so far, so d can only be determined numerically or graphically, by seeking the intersection between the curve representing the error obtained along Δ with noiseless images on the one hand, and a horizontal line of equation $y=10\%$ on the other hand, see Figure 5.17. Note that the curves were smoothed with a rectangular filter to remove the local fluctuation of the bias that could potentially disturb a proper estimation of this quantity. The spatial resolution found in each case is directly reported in each subfigure. We also considered here second-order subset shape functions, this case being more favorable for DIC [49]. Only the case $2M+1=21$ pixels is reported here, DIC diverging at some points with $2M+1=11$ pixels.

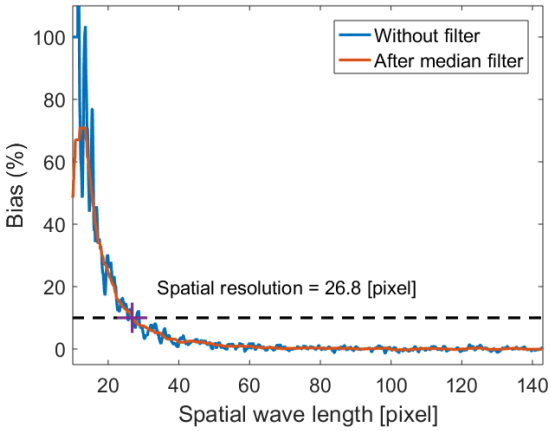
The value of d is smaller with both StrainNet-f and -h than with DIC used with first-order subset shape functions, even for $2M+1=11$ pixels, while d is nearly the same with DIC used with second-order subset shape functions. Indeed, Figure 5.18 (top) shows that the displacement along Δ is similar between StrainNet-f or -h on the one hand, and DIC with second-order subset



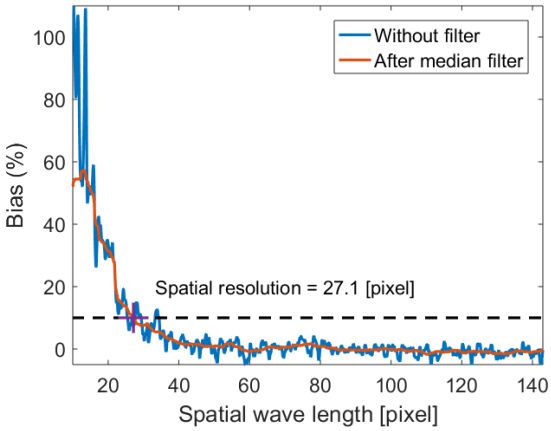
(a) DIC, $2M + 1 = 21$ pixels, first-order subset shape functions



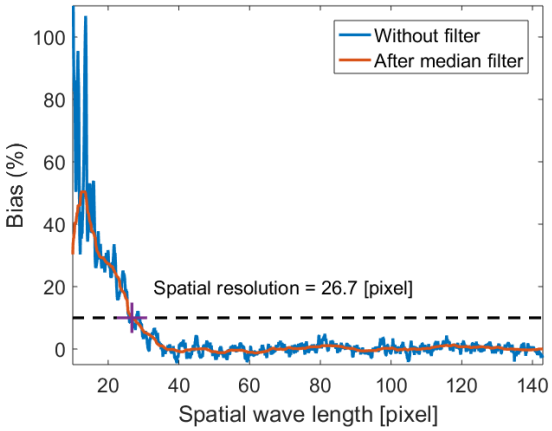
(b) DIC, $2M + 1 = 11$ pixels, first-order subset shape functions



(c) DIC, $2M + 1 = 21$ pixels, second-order subset shape functions



(d) StrainNet-h



(e) StrainNet-f

FIGURE 5.17 – Seeking the spatial resolution of each technique. The bias given here is a percentage of the displacement amplitude, which is equal to 0.5 pixel

.

shape functions on the other hand.

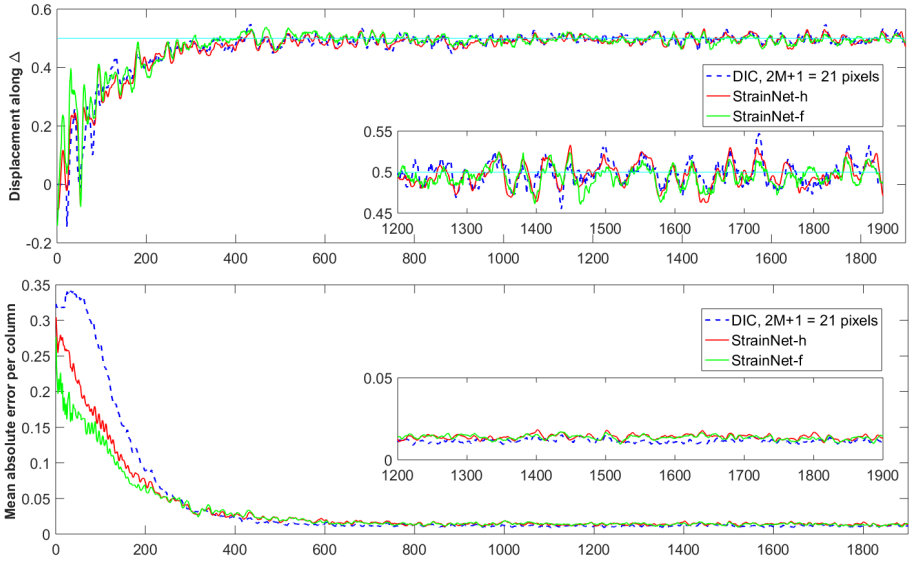
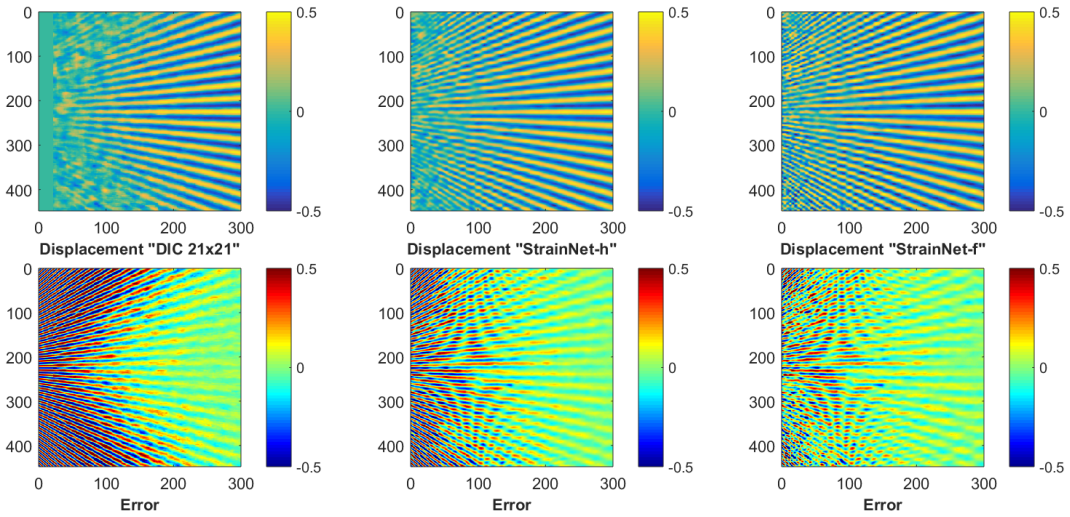


FIGURE 5.18 – Comparison between StrainNet and DIC ($2M+1=21$ pixels) with second-order subset shape function (noisy images). Top : displacements along Δ. Bottom : mean absolute error per column. All dimensions are in pixels.



(a) DIC, $2M+1=21$ pixels, second-order subset shape functions.

(b) StrainNet-h

(c) StrainNet-f

FIGURE 5.19 – Closeup view of the error map in pixels (for the high spatial frequencies) obtained with StrainNet and DIC with second-order subset shape functions.

Figure 5.19, where closeup views of the error map for the highest spatial frequencies are depicted, shows however that the way d is estimated is more favorable for DIC with second-order subset shape functions than for StrainNet-f and -h. Indeed, an additional bias occurs with DIC when going vertically away from the axis of symmetry Δ, as clearly illustrated in Figure 5.19a. Figures 5.19b-c show that it is not the case for StrainNet-f and -h. This phenomenon is not

taken into account when estimating d since only the loss of amplitude along Δ is considered. Consequently, when considering the mean absolute error per column as in Figure 5.18 (bottom), it can be observed that this error is lower for StrainNet-f and -h than for DIC with second-order subset shape functions.

5.7.2 Metrological performance indicator

A general remark holds for full-field measurement systems : the lower the value of d, the higher the noise in the displacement map. The noise level is quantified by the standard deviation of the noise. This quantity, denoted by σ_u, reflects the displacement resolution of the technique. Thus there is a link between d and σ_u. It has been rigorously demonstrated in [48, 43] that this product is constant for the Localized Spectral Analysis, a spectral technique which minimizes the optical residual in the Fourier domain. It has also been observed in subsequent studies that it was also the case for DIC [7, 46], which minimizes the same residual, but in the spatial domain. Hence estimating the product between d and σ_u is a handy way to compare measurement systems for a given bias, but independently from any other parameter chosen by the user such as the subset size for DIC. This product, denoted by α and named “metrological performance indicator" in [7, 46], has been calculated here for the different algorithms. The value of σ_u is merely estimated by calculating the standard deviation of the difference between the displacement fields found by processing noisy and noiseless Star images. The values of α found for each algorithm is reported in Figure 5.20.

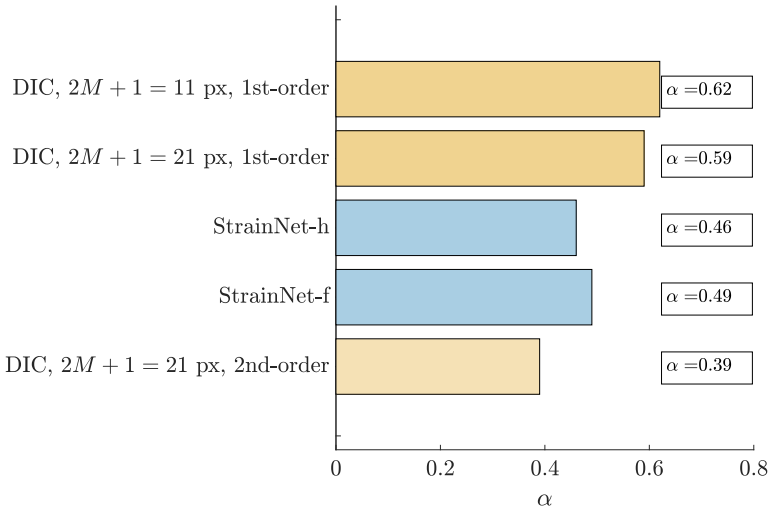
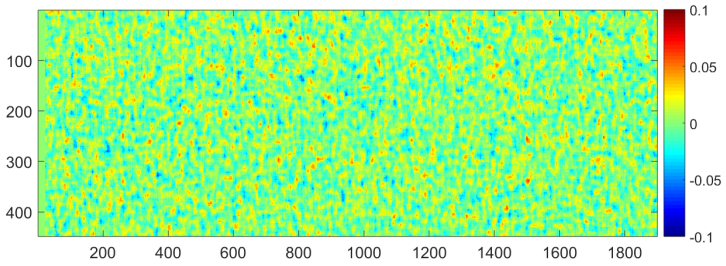


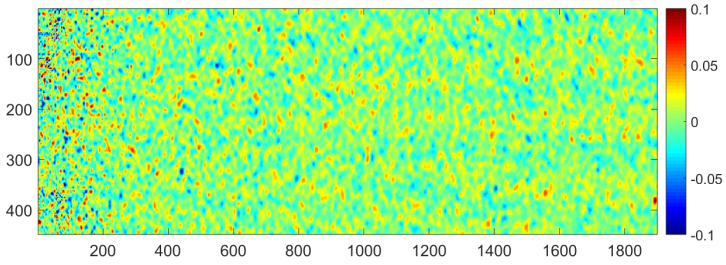
FIGURE 5.20 – Metrological efficiency indicator α for DIC (1st and 2nd subset shape functions), StrainNet-h and StrainNet-f.

This indicator is nearly the same for DIC used with $2M+1=11$ pixels and 21 pixels (1st order), which is consistent with the conclusions of [7]. It is also almost identical for StrainNet-f and -h. Both lie between DIC used with first- and second-order subset shape functions. Since the spatial resolution estimated with d is nearly the same with DIC used with second-order subset shape functions and StrainNet, it means that the noise level is higher in the latter case. This can

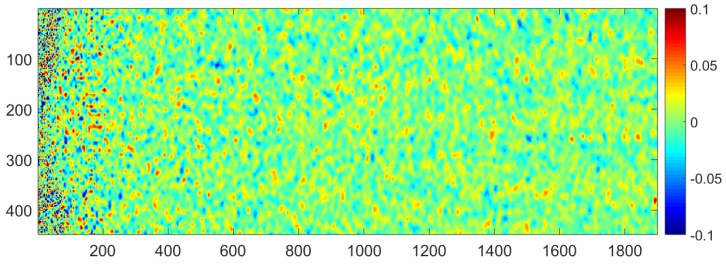
be observed in the noise maps depicted in Figure 5.21. In particular, the shape of the wave can be guessed in Figure 5.21b-c. A higher difference can also be observed on the left, so for the highest spatial frequencies. It means that a slight bias is induced by noise in these two cases. Further investigations should be undertaken to see how to eliminate this phenomenon, by changing the dataset and/or the layers of the network itself.



(a) DIC, $2M + 1 = 21$ pixels, 2nd-order subset shape functions



(b) StrainNet-h



(c) StrainNet-f

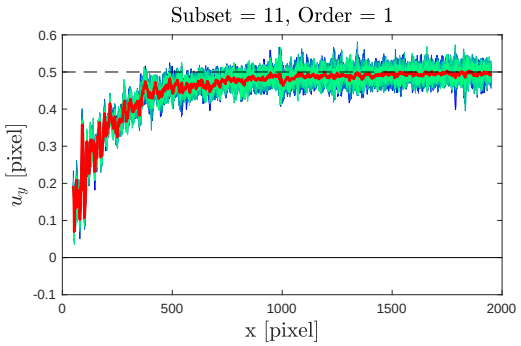
FIGURE 5.21 – Difference between displacement fields obtained with noisy and noiseless speckle images (in pixels). All dimensions are in pixels.

5.7.3 Pattern-induced bias

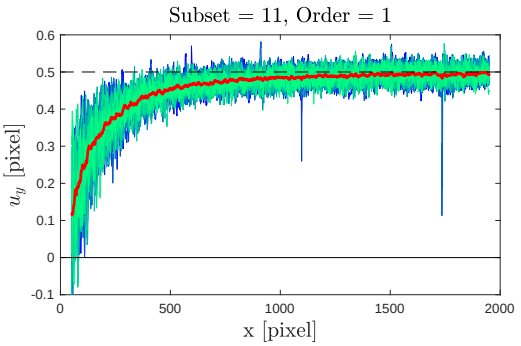
Pattern-induced bias (PIB) is a phenomenon, which has only recently been observed and described in the DIC literature [50, 51]. This bias is induced by the pattern texture itself. It manifests itself by the presence of random fluctuations in the displacement and strain maps. These random fluctuations shall not be confused with sensor noise propagation since it is due to different causes, mainly the image gradient distribution and the difference between the true displacement field and its local approximation by subset shape functions, see [6] where a model for this phenomenon is proposed. These spatial fluctuations are randomly distributed because spe-

ckle patterns are randomly distributed. The aim here is to briefly examine whether displacement fields retrieved by StrainNet are also prone to this phenomenon. We performed for this the same two experiments as in [51]. These experiments also rely on synthetic images deformed through the Star displacement. The first one consists in considering a unique pair of reference/deformed speckle images, adding 50 times a different copy of heteroscedastic noise, retrieving the 50 corresponding displacement fields, and plotting the mean distribution along the horizontal axis of symmetry Δ of the displacement map. The second one consists in considering 50 different pairs of reference/deformed speckles images, adding heteroscedastic noise to each image, retrieving the 50 corresponding displacement fields, and plotting again the mean distribution along Δ. With the first experiment, the random fluctuations due to sensor noise are averaged out (or at least decrease in amplitude by averaging effect). However, PIB is constant over the dataset since the displacement is the same and the speckles are similar from one image to another, the only difference between them being due to noise. In the second experiment, all the speckles are different and they are noisy, so both the random fluctuations due to sensor noise on the one hand, and due to the random fluctuations cause by PIB on the other hand, are averaged out. Comparing the curves obtained in each of these two cases enables us to numerically illustrate the effect of PIB on the displacement field, and to study its properties.

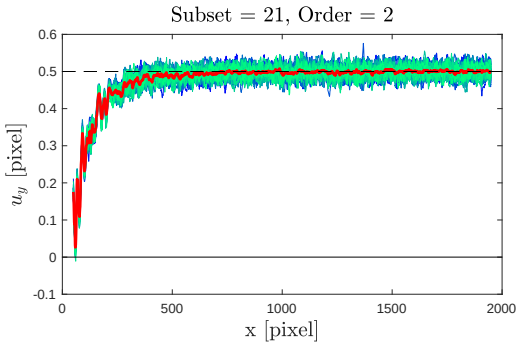
Figure 5.22 shows on the left and on the right the curves obtained on average in the first and second cases, respectively. They are plotted in red. The curves obtained for the 50 different pairs of images are superimposed in each case to give an idea of the fluctuations which are observed with each pair of images. The results obtained with DIC ($2M + 1 = 11$ pixels, 1st order and $2M + 1 = 21$ pixels, 2nd order) are also shown for comparison purposes. It is worth remembering that exactly the same sets of images are processed here by DIC and StrainNet. The main remark is that PIB also affects the results given by StrainNet, but averaging the results provided by 50 different patterns less improves the results than for DIC. The effect of PIB seems therefore to be less pronounced for StrainNet than for DIC, other phenomena causing these fluctuations. It is worth remembering that StrainNet-f and -h were trained on Speckle dataset 2.0, and that the deformed images contained in this dataset were obtained by interpolation. It would therefore be interesting to train StrainNet-f and -h with images obtained with the Speckle generator described in [39]. Indeed this latter generator does not perform any interpolation, so we could see if the errors observed in Figure 5.22-f are due to the pattern alone or to both the pattern and the bias due to the interpolation used when generating the deformed images. A larger dataset with another random displacement generation scheme as in Speckle dataset 2.0 could also help smoothing out this bias.



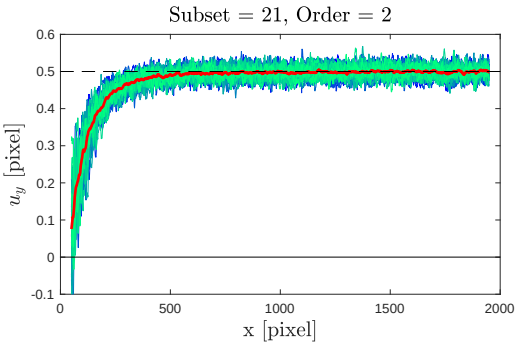
(a) DIC, $2M+1=11$ pixels, 1st-order subset shape functions, 1 pattern with 50 noises



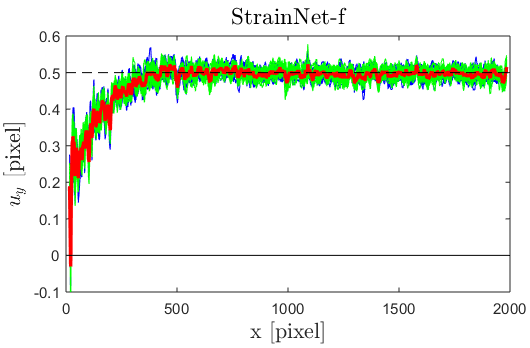
(b) DIC, $2M+1=11$ pixels, 1st-order subset shape functions, 50 patterns with a different noise



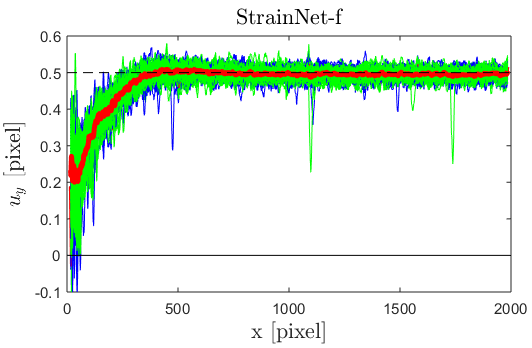
(c) DIC, $2M+1=21$ pixels, 2nd-order subset shape functions, 1 pattern with 50 noises



(d) DIC, $2M+1=21$ pixels, 2nd-order subset shape functions,50 patterns with a different noise



(e) StrainNet-f, 1 pattern with 50 noises



(f) StrainNet-f, 50 patterns with a different noise

FIGURE 5.22 – Pattern-induced bias. Comparison between results obtained with DIC ($2M+1=11$ pixels, 1st order and $2M+1=21$ pixels, 2nd order) and StrainNet-f.

5.8 Assessing the generalization capability

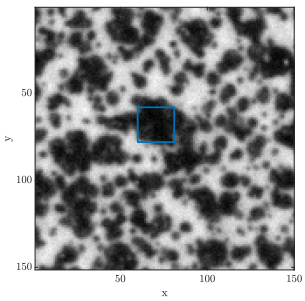
In deep learning, a key point is to validate any CNN with images different from those used to train the network in order to ensure good generalization capability. In the preceding sections, we took care to use StrainNet on speckle images deformed with the Star displacement while this network was trained with Speckle dataset 2.0 which does not contain any image deformed in a similar way. This is, however, not sufficient because both the reference images in Speckle dataset 2.0 and the reference Star image were generated with the same speckle generator [39].

Two other examples were therefore considered in this study. Both involve speckles, which are different from those obtained with the speckle generator [39] which generates the reference frames in the speckle datasets, and both the reference and deformed Star images. The first example concerns images of synthetic speckles from Sample 14 of the DIC Challenge 1.0 [4], the second real speckle images taken during a compression test performed on a wood specimen, as described in [44].

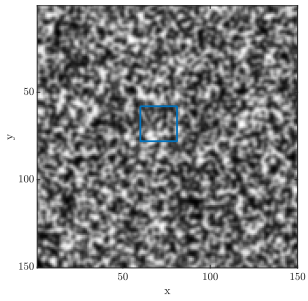
5.8.1 Example 1 : Sample 14 of the DIC Challenge 1.0

In this section, we consider a pair of images from Sample 14 of the DIC Challenge 1.0 [4, 3]. The speckle pattern is obtained with TexGen, a speckle generator described in [52]. This pattern is deformed by using a standard FFT expansion. It can be checked that the visual aspect is different from that of the images used so far in this paper. In particular, large dots over which the image gradient is nearly null can be observed, see Figure 5.23a where a typical subset size of $2M+1=21$ pixels is superimposed. For comparison purposes, Figure 5.23b represents a closeup view of one the speckle images of Speckle dataset 2.0. The displacement used to artificially deform the images is a sine wave along the horizontal axis, but with a frequency which increases when going to the right, the amplitude being constant and equal to 0.1 pixel. This example is discussed in [3], so the same colorbar as in this reference is used here to make comparison easier. Figures 5.23c-j show the displacement fields u along the horizontal direction obtained with the images from the “Sample 14 L5 Amp0.1" file available in [4]. DIC ($2M+1=11$ and 21 pixels, first-order subset shape functions), StrainNet-h, and -f are used here. The mean value along the vertical direction of the horizontal displacement, denoted here by $\overline{u}_x$, is also represented.

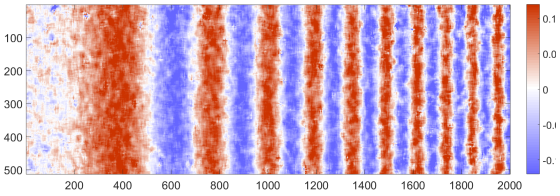
It can be seen that results obtained with DIC and $2M+1=21$ pixels is the less affected by noise. Noise increases with $2M+1=11$ pixels, which is logical since the subset size is smaller. The displacement field obtained with StrainNet-h is similar to the one obtained with DIC and $2M+1=11$ pixels. Figure 5.24 shows the strain fields ε_{xx} corresponding to the displacement fields shown above. They were obtained by convolving the corresponding displacement fields with a derivative kernel, which enables us to perform at the same time both a smoothing of the noisy data and differentiation. This kernel is the x-derivative of a Gaussian window of standard deviation equal to 6 pixels. Again, similar maps are obtained with StrainNet-f and -h trained on Speckle dataset 2.0 on the one hand, and DIC run with $2M+1=11$ pixels on the other hand. However, the main conclusion here is not really that StrainNet-f and -h give displacement and strain maps similar to DIC, but that these networks trained with speckles obtained with the speckle generator described in [39] are able to determine the displacement field used to deform speckle images obtained with another generator.



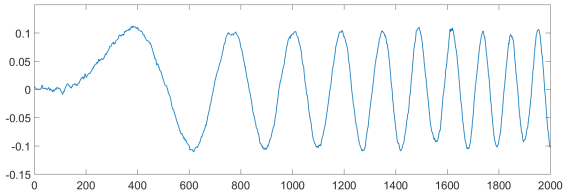
(a) Close-up view of the images of Sample 14 of the DIC challenge 1.0. A subset used in DIC is superimposed (size : $2M+1=21$ pixels).



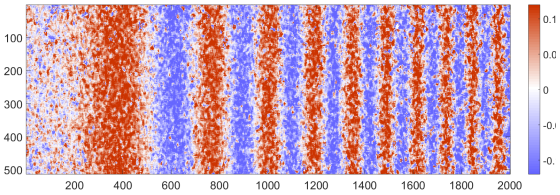
(b) Close-up view of one of the images of Speckle dataset 2.0 used to train the network.



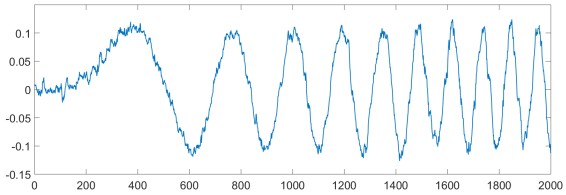
(c) u_x, DIC, $2M+1=21$ pixels



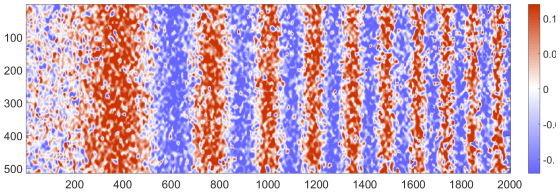
(d) $\overline{u}_x$, DIC, $2M+1=21$ pixels



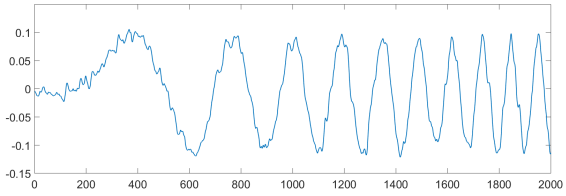
(e) u_x, DIC, $2M+1=11$ pixels



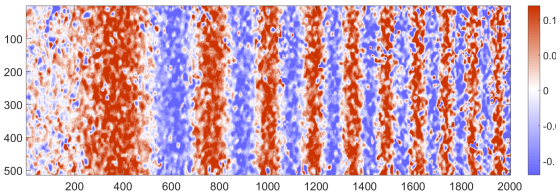
(f) $\overline{u}_x$, DIC, 11, $2M+1=11$ pixels



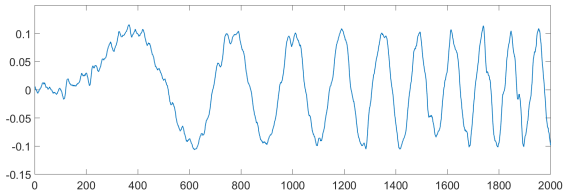
(g) u_x, StrainNet-h



(h) $\overline{u}_x$, StrainNet-h

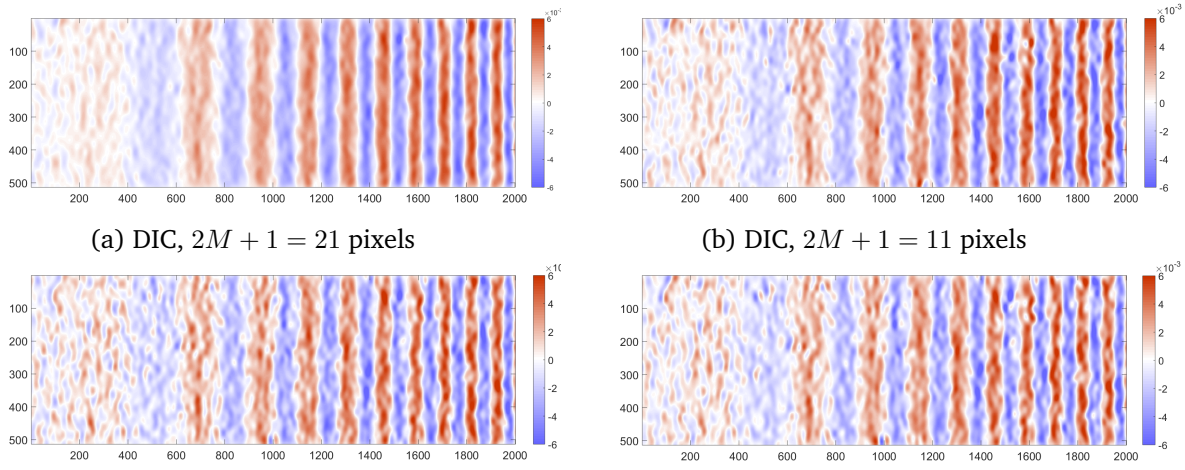


(i) u_x, StrainNet-f



(j) $\overline{u}_x$, StrainNet-f

FIGURE 5.23 – Results obtained by processing images from Sample 14 of the DIC Challenge [4]. Top : closeup view of the speckle. Left : displacement fields. Right : profile of the average displacement over all the columns of the maps. See [3] to compare these maps to those obtained with other DIC packages. All dimensions are in pixels.



(a) DIC, $2M+1=21$ pixels

(b) DIC, $2M+1=11$ pixels

(c) StrainNet-h

(d) StrainNet-f

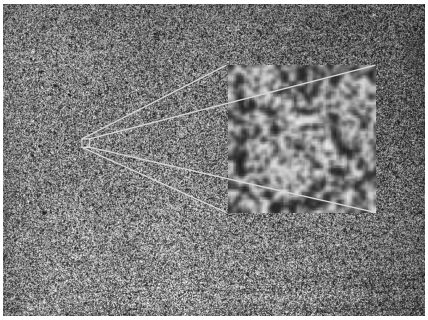
FIGURE 5.24 – Strain map ε_{xx} deduced from the displacement fields depicted in Figure 5.23. All dimensions are in pixels.

5.8.2 Example 2 : Compression test on a wood specimen

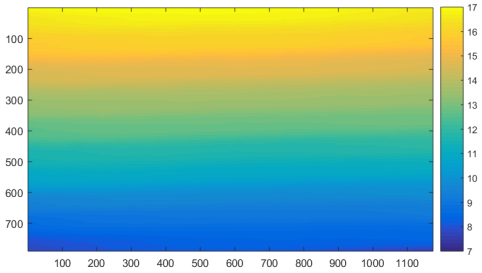
In this second example, we consider a real test performed on a wood specimen shown in Figure 5.25a, see [44] for more details about this specimen and the testing conditions. The interest here is twofold. First real speckles instead of artificial ones are considered. Second, the constitutive material of the specimen is heterogeneous since this is a stack of early and late wood. A consequence is that the stiffness spatially changes, and so does the strain distribution if the rings are perpendicular to the loading force, which is the case here. We consider a typical pair of frames and applied StrainNet-f to determine the displacement field. Results obtained with DIC with $2M+1=21$ pixels (1st-order subset shape functions) are shown for comparison purposes. A convolution with a Gaussian derivative filter is then applied in all cases to deduce the vertical strain map ε_{yy}.

It can be seen that similar maps are obtained but again, a more refined analysis should be performed to discuss the possible damping of the actual details in the strain map, as in [44] but this is out of the scope of the present paper. The main point here is that StrainNet is able to extract displacement and strain fields featuring rather high spatial frequencies from images different from those obtained with Speckle dataset 2.0 since this is here a real speckle pattern. It must be noted that the displacement is greater than one pixel over most of the front face of the specimen. This displacement was therefore split into two quantities. The first one is the displacement with a quite rough pixel resolution. The second one is the subpixel displacement. The images are therefore processed piecewise, in such a way that the round value of the displacement is the same throughout each of the elements of the mesh. A mesh of 11 horizontal bands is considered here. This round value for the displacement can easily be found by cross-correlation for instance. This point is however not really challenging and we applied here a rough DIC to get this integer value for the sake of simplicity, only the subpixel determination of the displacement

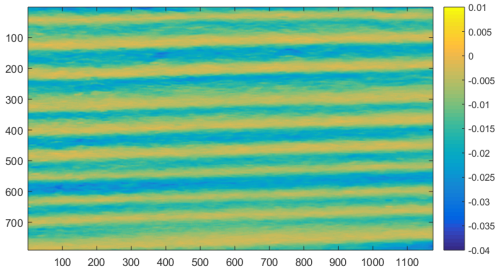
(a) Specimen before spray-painting, after [44]



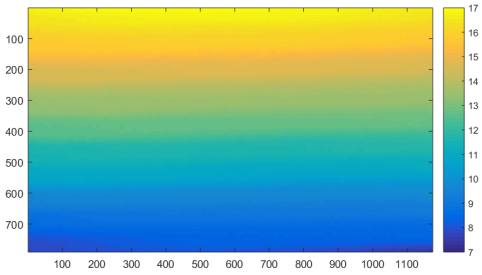
(b) Speckled surface of the specimen after spray painting, after [44]



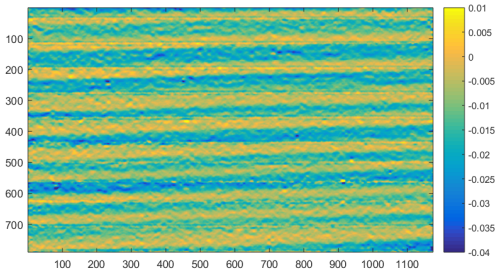
(c) DIC, $2M + 1 = 21$ pixels



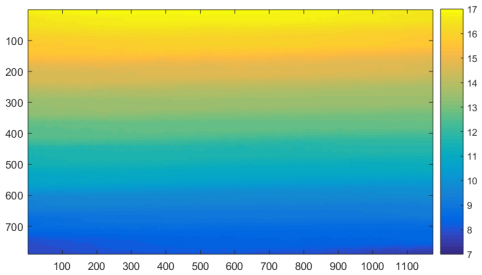
(d) DIC, $2M + 1 = 21$ pixels



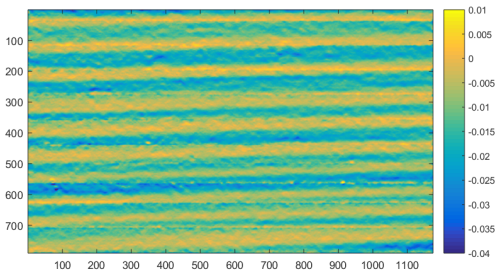
(e) StrainNet-h



(f) StrainNet-h



(g) StrainNet-f



(h) StrainNet-f

FIGURE 5.25 – Results obtained by processing real images. Left : v displacement field in pixels. Right : ε_{yy} strain map. All dimensions are in pixels.

being of interest in the present study. A consequence is the fact that on close inspection, slight straight lines can be seen in the strain maps, along the border between some of the elements. Nothing can be visually detected at the same place in the displacement maps.

5.9 Computing time

Finally, we give some information on the computing time needed to perform the calculations. CNNs are well suited to be run on Graphical Processing Units (GPUs), which was the case here. This is even necessary to achieve training in a reasonable amount of time, as mentioned in Section 5.5.1. Once StrainNet was trained, we used it on the same GPU. The computing time needed to estimate the subpixel displacement field is reported in Table 5.7. Two typical examples are considered here. The first one corresponds to one pair of Star images, the second to Sample 14 of the DIC Challenge 2.0. The number of pixels of the frames and the computing time are also reported in this table. The number of Points Of Interest per second (POI/s) is also given in each case. This quantity represents the number of points per second at which a measurement is provided by the measuring system. It is used in [53] in order to measure the calculating speed of GPU-based DIC. In our case, using this quantity is a handy way to normalize the results obtained with different techniques and different frame sizes, and to fairly compare them.

TABLE 5.7 – Computing time and Points Of Interest per second (POI/s) for Examples 1 and 2.

Case	Frame size	StrainNet-h		StrainNet-f	
		Computing time (s)	POI/s	Calculating time (s)	POI/s
Example 1	2000×501	0.0081	1.24E+08	0.077	1.30E+07
Example 2	2048×589	0.0088	1.37E+08	0.092	1.31E+07

For StrainNet-f and -h, the value of POI/s is about ten times lower for the “h" version than for the “f" one while the resolution before final interpolation to reach full resolution is 4 times lower. Interestingly, Ref [53] reports a POI/s equal to $1.66 \cdot 10^5$ and $1.13 \cdot 10^5$ for a parallel DIC code implemented on a GPU, which is nearly two orders of magnitude below. These values are given for information only : the GPU used in [53] (NVIDIA GTX 760, 2.3 TFLOPs) is indeed less powerful than the GPU used in the present study (NVIDIA TESLA V100, 114 TFLOPs). In addition, the reader should bear in mind that CNNs must be trained with a suitable dataset, which generally represents heavy calculations. Further work should therefore be undertaken to fairly compare StrainNet and a GPU-based DIC in terms of computing time. The conclusion is, however, that StrainNet provides pixel-wise displacement maps (and thus strain maps by convolution with suitable derivative filters) at a rate compatible with real-time measurements.

5.10 Conclusion

This paper presented a CNN dedicated to the measurement of displacement and strain fields. As for DIC, the surface to be investigated was marked with a speckle pattern. Various strategies deduced from the similar problem of optical flow determination were presented, and the best one has been adapted to give a network named StrainNet. This network was trained with two versions of a specific speckle dataset. The main result was to demonstrate, through some relevant

examples, the feasibility of this type of approach for accurate pixelwise subpixel measurement over full displacement fields. As for other problems tackled with CNNs in engineering, the main benefit here is the very short computing time to obtain the sought quantities, here the displacement fields. Further studies remain necessary to investigate various problems, which are still open after this preliminary work. For instance, the dataset used in order to train the network directly influences the quality of the final results. The dataset developed here led to valuable results for speckles different than those of the images forming the dataset, in particular experimental ones. This observation should however be consolidated by considering a wider panel of speckles and thus by trying to reduce noise and bias in the final displacement maps. The generator free from any interpolation, which was used here only for generating the deformed Star images for a matter of time, could also be employed for the images of the dataset despite the computing cost. The networks discussed here were obtained by enhancing FlowNetS. A complete redesign should also be undertaken, in particular in order to simplify the network. This would certainly reduce both the training and the processing times. Finally, a model able to deal with displacements larger than one pixel while still giving accurate subpixel estimation should also be investigated further, for instance by training the network on a dataset containing deformed images involving displacements greater than one pixel.

Acknowledgments

This work has been sponsored by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01).

Bibliographie

[1] M. Sutton, J.J. Orteu, and H. Schreier. *Image Correlation for Shape, Motion and Deformation Measurements. Basic Concepts,Theory and Applications*. Springer, 2009.

[2] B. Pan, K. Qian, H. Xie, and A. Asundi. Two-dimensional digital image correlation for in-plane displacement and strain measurement : a review. *Measurement Science and Technology*, 20(6) :062001, 2009.

[3] P. L. Reu, E. Toussaint, E. Jones, H. A. Bruck, M. Iadicola, R. Balcaen, D. Z. Turner, T. Siebert, P. Lava, and M. Simonsen. DIC challenge : Developing images and guidelines for evaluating accuracy and resolution of 2D analyses. *Experimental Mechanics*, 58(7) :1067–1099, 2018.

[4] DIC challenge : http ://sem.org/dic-challenge/.

[5] H. W. Schreier, M. Sutton, and A. Michael. Systematic errors in digital image correlation due to undermatched subset shape functions. *Experimental Mechanics*, 42(3) :303–310, 2002.

[6] F. Sur, B. Blaysat, and M. Grédiac. On biases in displacement estimation for image registration, with a focus on photomechanics. Submitted for publication, 2020.

[7] M. Grédiac, B. Blaysat, and F. Sur. A critical comparison of some metrological parameters characterizing local digital image correlation and grid method. *Experimental Mechanics*, 57(6) :871–903, 2017.

[8] G.F. Bomarito, J.D. Hochhalter, T.J. Ruggles, and A.H. Cannon. Increasing accuracy and precision of digital image correlation through pattern optimization. *Optics and Lasers in Engineering*, 91 :73 – 85, 2017.

[9] M. Grédiac, B. Blaysat, and F. Sur. Extracting displacement and strain fields from checkerboard images with the localized spectrum analysis. *Experimental Mechanics*, 59(2) :207–218, 2019.

[10] M. Grédiac, B. Blaysat, and F. Sur. On the optimal pattern for displacement field measurement : random speckle and DIC, or checkerboard and LSA ? *Experimental Mechanics*, 60(4) :509–534, 2020.

[11] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[12] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet : Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[13] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[14] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1–3) :185–203, 1981.

[15] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, volume 2, page 674–679, 1981.

[16] S. Guan, H. Li, and W. Zheng. Unsupervised learning for optical flow estimation using pyramid convolution lstm. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 181–186, 2019.

[17] A. Ahmadi and I. Patras. Unsupervised convolutional neural networks for motion estimation. In *IEEE International Conference on Image Processing (ICIP)*, pages 1629–1633, 2016.

[18] Y. Wang, Y. Yang, Z. Yang, L. Zhao, P. Wang, and W. Xu. Occlusion aware unsupervised learning of optical flow. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4884–4893, 2018.

[19] W.-S. Lai, J.-B. Huang, and M.-H. Yang. Semi-supervised learning for optical flow with generative adversarial networks. In *Neural Information Processing Systems (NIPS)*, 2017.

[20] Y. Yang and S. Soatto. Conditional prior networks for optical flow. In *IEEE European Conference on Computer Vision (ECCV)*, pages 282–298, 2018.

[21] J. Xu, R. Ranftl, and V. Koltun. Accurate optical flow via direct cost volume processing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5807–5815, 2017.

[22] J. Wulff, L. Sevilla-Lara, and M. J. Black. Optical flow in mostly rigid scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6911–6920, 2017.

[23] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow : Large displacement optical flow with deep matching. In *IEEE International Conference on Computer Vision*, pages 1385–1392, 2013.

[24] C. Bailer, B. Taetz, and D. Stricker. Flow fields : Dense correspondence fields for highly accurate large displacement optical flow estimation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 4015–4023, 2015.

[25] O. Ronneberger, P. Fischer, and T. Brox. U-net : Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015.

[26] P. Hu, G. Wang, and Y. Tan. Recurrent spatial pyramid CNN for optical flow estimation. *IEEE Transactions on Multimedia*, 20(10) :2814–2823, 2018.

[27] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0 : Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1647–1655, 2017.

[28] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2720–2729, 2017.

[29] T. Hui, X. Tang, and C. C. Loy. Liteflownet : A lightweight convolutional neural network for optical flow estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8981–8989, 2018.

[30] D. Sun, X. Yang, M. Liu, and J. Kautz. PWC-Net : CNNs for optical flow using pyramid, warping, and cost volume. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018.

[31] D.J. Butler, J. Wulff, G.B. Stanley, and M.J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision (ECCV)*, page 611–625, 2012.

[32] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015.

[33] S. Cai, S. Zhou, C. Xu, and Q. Gao. Dense motion estimation of particle images via a convolutional neural network. *Experiments in Fluids*, 60(4) :73, 2019.

[34] S. Cai, J. Liang, Q. Gao, C. Xu, and R. Wei. Particle image velocimetry based on a deep learning motion estimator. *IEEE Transactions on Instrumentation and Measurement*, 2019.

[35] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016.

[36] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1) :1–31, 2010.

[37] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving ? the KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[38] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016.

[39] F. Sur, B. Blaysat, and M. Grédiac. Rendering deformed speckle images with a Boolean model. *Journal of Mathematical Imaging and Vision*, 60(5) :634–650, 2018.

[40] C. Pinard. A reimplementation of FlowNet using PyTorch. `https://github.com/ClementPinard/FlowNetPytorch`, 2017.

[41] S. Niklaus. A reimplementation of PWC-Net using PyTorch. `https://github.com/sniklaus/pytorch-pwc`, 2018.

[42] S. Niklaus. A reimplementation of LiteFlowNet using PyTorch. `https://github.com/sniklaus/pytorch-liteflownet`, 2019.

[43] M. Grédiac and F. Sur. Effect of sensor noise on the resolution and spatial resolution of the displacement and strain maps obtained with the grid method. *Strain*, 50(1) :1–27, 2014.

[44] M. Grédiac, B. Blaysat, and F. Sur. A robust-to-noise deconvolution algorithm to enhance displacement and strain maps obtained with local DIC and LSA. *Experimental Mechanics*, 59(2) :219–243, 2019.

[45] E.M.C Jones and M.A. Iadicola (Eds). *A Good Practices Guide for Digital Image Correlation*. International Digital Image Correlation Society, 2018. DOI : 10.32720/idics/gpg.ed1.

[46] B. Blaysat, J. Neggers, M. Grédiac, and F. Sur. Towards criteria characterizing the metrological performance of full-field measurement techniques. Application to the comparison between local and global versions of DIC. *Experimental Mechanics*, 60(3) :393–407, 2020.

[47] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian. Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 17(10) :1737–1754, 2008.

[48] F. Sur and M. Grédiac. Towards deconvolution to enhance the grid method for in-plane strain measurement. *Inverse Problems and Imaging*, 8(1) :259–291, 2014.

[49] L. Wittevrongel, P. Lava, S. V. Lomov, and D. Debruyne. A self adaptive global digital image correlation algorithm. *Experimental Mechanics*, 55(2) :361–378, 2015.

[50] R. B. Lehoucq, P. L. Reu, and D. Z. Turner. The effect of the ill-posed problem on quantitative error assessment in digital image correlation. *Experimental Mechanics*, 2017. in press.

[51] S. S. Fayad, D. T. Seidl, and P. L. Reu. Spatial dic errors due to pattern-induced bias and grey level discretization. *Experimental Mechanics*, 60(2) :249–263, 2020.

[52] J.-J. Orteu, D. Garcia, L. Robert, and F. Bugarin. A speckle texture image generator. *Proceedings SPIE : Speckle06 : speckles, from grains to flowers.*, 6341 :63410H 1–6, 2006.

[53] L. Zhang, T. Wang, Z. Jiang, Q. Kemao, Y. Liu, Z. Liu, L. Tang, and S. Dong. High accuracy digital image correlation powered by GPU-based parallel computing. *Optics and Lasers in Engineering*, 69 :7 – 12, 2015.

Chapitre 6

Conclusion générale et perspectives

6.1 Conclusion

Cette thèse a été menée en deux parties. Dans la première, nous avons visé l'accélération de l'interpolation qui est considérée comme le traitement clé dans la CIN. La seconde partie a présenté une nouvelle méthode de mesure basée sur les réseaux de neurones convolutifs pour déterminer les champs de déplacements et de déformations.

Concernant l'accélération de l'interpolation, nous avons apporté deux contributions. Dans la première, nous avons implémenté sur FPGA une architecture hétérogène de l'interpolation bi-cubique. Cette architecture repose sur une reformulation mathématique de l'interpolation bi-cubique afin de réduire la complexité du calcul. L'architecture proposée réduit le nombre de multiplicateurs de 25% par rapport aux références [1, 2]. Elle évite ainsi douze calculs redondants. Une largeur de 9 bits fractionnaires est sélectionnée afin d'atteindre un bon compromis entre consommation des ressources matérielles et précision des résultats. L'architecture a été implémentée en utilisant le VHDL. Elle fonctionne à une fréquence de 205 MHz sur le FPGA d'Intel Cyclone V.

La seconde contribution propose un ensemble d'algorithmes comme alternatives à l'interpolation bi-cubique. Ces algorithmes sont basés sur des combinaisons d'interpolations cubiques et linéaires. L'architecture de chaque algorithme a également été implémentée sur le FPGA d'Intel Cyclone V. L'avantage des algorithmes proposés est la réduction de la consommation des ressources matérielles. Nous avons ainsi réduit le nombre de multiplicateurs de plus de 65% par rapport aux références [1, 3]. En outre, les algorithmes proposés nécessitent moins d'ALMs et fonctionnent à une fréquence plus élevée que celle de [1]. Ces algorithmes se rapprochent également au mieux de l'interpolation bi-cubique. De plus, une étude de précision en fonction des ressources matérielles requises a été présentée pour chaque algorithme.

À la fin de cette partie, nous avons conclu que malgré toutes ces améliorations apportées pour réduire la consommation des ressources matérielles, l'implémentation de la CIN nécessite toujours plus de ressources que celles fournies par un seul FPGA. Dans la deuxième partie de ce travail, nous avons donc exploré la possibilité d'utiliser des réseaux de neurones convolutifs

(CNNs) pour déterminer les champs de déplacements et de déformations. Deux CNNs appelés StrainNet-f et -h dédiés à la mesure des champs de déplacements et de déformations ont donc été proposés. Ces deux CNNs ont été entraînés avec deux datasets (Speckele dataset 1.0 et 2.0 [1]) que nous avons développés dans le cadre de ce travail. La principale conclusion est StrainNet-f et -h présentent des performances largement concurrentielles voire supérieures à celles de la CIN en termes de performances métrologiques et de temps de calcul.

6.2 Perspectives

Dans la continuité de cette thèse, plusieurs travaux et voies de recherche peuvent être envisagés. Ceux-ci visent principalement le dataset, le réseau de neurones lui-même, et la plateforme matérielle.

— Afin d'étudier la possibilité d'améliorer les performances métrologiques, une première étape serait d'améliorer le dataset, en générant des images sans bais d'interpolation et plus riches en termes de types de speckles. Pour cela, une re-implémentation du générateur proposée dans [4] en CUDA peut être envisagée pour accélérer le temps de génération des images.

— Une deuxième piste serait d'implémenter la CIN en CUDA pour pouvoir fournir des comparaisons équitables entre les CNNs proposés et la CIN dans la même plateforme matérielle GPU.

— Une troisième piste viserait à optimiser le réseau StrainNet afin de pouvoir l'implémenter sur des plateformes embarquées à faible coût, comme la Nvidia Jetson Xavier NX [5]. Pour cela, nous envisageons de réduire la taille des noyaux de convolution, le nombre de couches et/ou le nombre des "feature maps" de chaque couche du réseau.

— Une autre piste serait d'étudier la possibilité d'utiliser des plateformes hybrides (FPGA et GPU) pour accélérer le temps de calcul afin d'avoir un bon compromis entre performances métrologiques et consommation d'énergie.

— Enfin, un modèle capable de traiter des déplacements supérieurs à un pixel tout en donnant une estimation précise d'ordre sous-pixellique devrait également être étudié. Pour cela, entraîner le réseau sur un dataset contenant des déplacements supérieurs à un pixel peut être envisagé. Une autre possibilité est de combiner un algorithme classique simple pour calculer les déplacements pixelliques et un CNN comme StrainNet pour estimer les déplacements sous-pixelliques.

1. Disponible sur : `github.com/DreamIP/StrainNet`

Bibliographie

[1] M. A. Nuno-Maganda and M. O. Arias-Estrada. Real-time FPGA-based architecture for bicubic interpolation : an application for digital image scaling. *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig05)*, 2005.

[2] Y. Zhang, Y. Li, J. Zhen, J. Li, and R. Xie. The hardware realization of the bicubic interpolation enlargement algorithm based on FPGA. *2010 Third International Symposium on Information Processing*, 2010.

[3] D. Q. Liu, G. Q. Zhou, X. Zhou, C. Y. Li, and F. Wang. FPGA-based on-board cubic convolution interpolation for spaceborne georeferencing. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-3/W10 :349–356, Jul 2020.

[4] F. Sur, B. Blaysat, and M. Grédiac. Rendering deformed speckle images with a Boolean model. *Journal of Mathematical Imaging and Vision*, 60(5) :634–650, 2018.

[5] NVIDIA Jetson Xavier NX. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/`.