Convolutional Neural Networks for embedded vision

Lucas Fernandez Brillet

▶ **To cite this version:**

Lucas Fernandez Brillet. Convolutional Neural Networks for embedded vision. Artificial Intelligence [cs.AI]. Université Grenoble Alpes [2020-..], 2020. English. NNT : 2020GRALM043 . tel-03101523

HAL Id: tel-03101523
https://theses.hal.science/tel-03101523

Submitted on 15 Feb 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques, sciences et technologies de l'information, informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Lucas FERNANDEZ BRILLET

Thèse dirigée par **Stéphane MANCINI**, Maître de Conférence, Université Grenoble Alpes et codirigée par **Marina Nicolas**, STMicroelectronics

préparée au sein du **Laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Réseaux de neurones CNN pour la vision embarquée

Convolutional Neural Networks for embedded vision

Thèse soutenue publiquement le **28 septembre 2020**,
devant le jury composé de :

Monsieur STEPHANE MANCINI
MAITRE DE CONFERENCES HDR, UNIVERSITE GRENOBLE ALPES, Directeur de thèse
Monsieur FRANÇOIS BERRY
PROFESSEUR DES UNIVERSITES, UNIVERSITE CLERMONT AUVERGNE, Rapporteur
Monsieur MICHEL PAINDAVOINE
PROFESSEUR DES UNIVERSITES, UNIVERSITE BOURGOGNE FRANCHE-COMTE, Rapporteur
Madame VIRGINIE FRESSE
MAITRE DE CONFERENCES HDR, UNIVERSITE JEAN MONNET - SAINT-ETIENNE, Examinatrice
Monsieur ROMAIN COUILLET
PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES, Président

Acknowledgements

This Ph.D. thesis has been funded though the CIFRE program from ANRT (Association Nationale de la Recherche Technique) in a collaboration between STMicroelectronics and the TIMA laboratory, Grenoble INP, Université Grenoble Alpes. The author would like to acknowledge all these institutions, and specially the people who have enabled and contributed to this research.

Abstract

Recently, Convolutional Neural Networks have become the state-of-the-art soluion to most computer vision problems. Although very interesting CNN-based computer vision solutions are constantly emerging, it is very difficult to deploy them in the real-world, and particularly in heavily computationally-constrained embedded systems. This thesis first details multiple fundamental issues and properties of CNNs impeding this goal.

Briefly, the main bottleneck is that, in order to achieve high accuracy rates, CNNs require a high parameter count, as well as a high number of operations. This greatly complicates the deployment of such solutions in embedded systems, which strive to reduce memory size, since this greatly reduces energy consumption, area, and cost of the solution. As an indicative figure, while most embedded systems are typically in the range of a few KBytes of memory, CNN models from the state-of-the-art usually account for multiple MBytes, or even GBytes in model size.

Throughout this thesis, multiple novel ideas allowing to ease this issue are proposed. This requires to jointly design the solution across three main axes : Application, Algorithm and Hardware.

First and foremost, clearly defining the applicative specifications helps exploit parameter efficiency by reducing the task's complexity, i.e. variability in the samples. Throughout this manuscript, the main levers allowing to tailor computational complexity of a generic CNN-based object detector are identified and studied. Since object detection requires scanning every possible location and scale across an image through a fixed-input CNN classifier, the number of operations quickly grows for high-resolution images, thereby obstructing their implementation in embedded systems. In order to perform object detection in an efficient way, the detection process is divided into two stages. The first stage involves a region proposal network which allows to trade-off recall (i.e. the percentage of identified regions containing an interesting object) for the number of operations required to perform the search, as well as the number of regions passed on to the next stage. Techniques such as bounding box regression also greatly help reduce the dimension of the search space. This in turn simplifies the second stage, since it allows to reduce the task's complexity to the set of possible proposals. Therefore, parameter counts can greatly be reduced.

Furthermore, at the algorithm level, CNNs exhibit multiple properties that contribute to their over-dimensionment. This over-dimensionement is one of the key success factors of CNNs in practice, since it eases the optimization process by allowing a large set of equivalent solutions. However, this also greatly increases computational complexity, and therefore complicates deploying the inference stage of these algorithms on embedded systems. In order to ease this problem, we proposed a CNN compression method which is based on *Principal Component Analysis* (PCA). PCA allows to find, for each layer of the network independently, a new representation of the set of learned filters by expressing them in a more appropriate *PCA basis*. This PCA basis is hierarchical, meaning that basis terms are ordered by importance, and by removing the least important basis terms, it is possible to optimally trade-off approximation error for parameter count. Through this method, it is possible to compress, for example, a *ResNet-32* network by a factor of $\times 2$ both in the number of parameters and operations with a loss of accuracy $<2\%$.

It is also shown that the proposed method is compatible with other previously reviewed state-of-the-art methods which exploit other CNN properties in order to reduce computational complexity, mainly *pruning*, *winograd* and *quantization*. Through this method, we have been able to reduce the size of a *ResNet-110* from 6.88Mbytes to 370kbytes, i.e. a x19 memory gain with a 3.9 % accuracy loss.

Through all this knowledge, we propose an efficient CNN-based solution for a consumer face detection scenario. The solution consists of just 29.3k parameters, with 6.3k parameters on the proposal network, and 23k parameters on the classification network. Since weights and activations are quantized to 8-bit, the model size is 29.3kBytes. This is x65 smaller than other state-of-the art CNN face detectors, while providing equal detection performance and lower number of operations on datasets representative of our applicative use-case. Our proposed face detector is also compared to a more traditional Viola-Jones face detector, exhibiting approximately an order of magnitude faster computation for the same detection performance, as well as the ability to scale to higher detection rates by slightly increasing computational complexity.

Both networks are finally implemented in STMicroelectronics' *ASMP*, a custom embedded multiprocessor. This allows to verify that PCA compression doesn't add any computing overheads, and that theorical computation gains stay mostly consistent when implemented in hardware. Furthermore, parallelizing the PCA compressed network over 8 *PEs* achieves a x11.68 speed-up with respect to the original network running on a single *PE*. The Proposal network's efficiency is also evaluated for different input resolutions. In this case, efficiency in terms of MAC/cycle, as well as parallelization speed-ups both increase as the input resolution gets larger. We also approximately evaluate the total cycle count required for the proposed applicative scenario.

As a general conclusion, the results from this thesis reinforce the importance of jointly designing algorithms and hardware for this domain. The effectiveness of the proposed methodologies is demonstrated throughout this thesis, ultimately leading to a CNN-based face detection solution which is able to run efficiently on low-cost hardware.

Résumé Étendu

Les réseaux de neurones convolutionnels CNNs comportent maintenant la solution de l'état de l'art pour la plupart des problématiques de vision par ordinateur. Des nouvelles solutions apparaissent constamment dans l'état de l'art, cependant, leur déploiement dans le monde réel est complexe, spécialement dans des systèmes embarqués qui présentent des grandes contraintes de calcul. Malgré tout, le déploiement de ces solutions dans ce type de systèmes constraints est essentiel pour la sustainabilité à long terme et l'adoption du marché de ces technologies.

Les sources de ces difficultés sont diverses :

— Pour atteindre des hauts taux de performance, les CNNs nécessitent d'un nombre de paramètres très élevé, qui surpasse largement la capacité des mémoires utilisées dans l'embarqué. Ceci est un point crucial, puisque la taille de la mémoire détermine en grande mesure la consommation d'énergie, et le coût de la solution. Dans un CNN, le nombre de paramètres dépend fortement de la complexité de la tâche à résoudre, et c'est pour cela que le cadre applicatif doit être bien spécifié, comme expliqué au Chapitre 2. De plus, les CNNs possèdent des propriétés inhérentes qui contribuent à leur surdimensionnement. Dans ce sens, des techniques de compression de CNN présentent un intérêt primordial, comme ceci est montré au Chapitre 3.

— En fonction du type d'application visée par le CNN, le nombre d'opérations peut aussi devenir encombrant. Dans ce sens, une méthode générique permettant d'adapter la complexité computationnelle d'un algorithme de détection d'objets est détaillé au Chapitre 2, et ensuite mise en pratique pour une application spécifique de détection de visages au Chapitre 4.

— Finalement, la solution doit être évaluée en hardware pour vérifier que les méthodes proposées sont bien amenés à la parallélisation, et que les gains algorithmiques sont consistents avec les gains mesurés. Ceci est expliqué au Chapitre 5.

0.1 Compression des CNN

Le Chapitre 1 fournit un ample contexte nécessaire à la compréhension des CNN, et évidence le surdimensionnement des solutions basées sur des CNN. Ce surdimensionnement facilite la phase d'optimisation [92, 93], et est considéré comme l'un des facteurs clé dans le succès des CNN. De même, ce surdimensionnement complique énormément leur implémentation sur des systèmes embarqués. C'est pour cela que des méthodes permettant de compresser un CNN, une fois qu'une solution satisfaisante ayant été trouvée, présentent un grand intérêt.

Plusieurs méthodes dans l'état de l'art permettent d'aborder ce problème en exploitant certaines propriétés inhérentes aux CNNs. Les plus effectives concernent la quantification [61–64], le pru-

ning [67–69] où bien Winograd [112]. Ces méthodes sont largement compatibles entre elles, et permettent de réduire considérablement la complexité computationnelle des CNNs.

Dans ce manuscrit, une nouvelle méthode de compression basée sur l'Analyse en composantes principales (PCA) est proposée. PCA nous permet pour l'ensemble des filtres dans chaque couche d'un CNN de trouver une nouvelle *base PCA*,$\mathbf{v}_k$, qui est plus appropriée pour la représentation de ces filtres, qui ont été appris auparavant, suite à l'optimisation du CNN. Dans cette base, chacun des filtres, X_i comportant la couche i, peut être exprimé comme une combinaison linéaire des filtres de base $\mathbf{v}_k$, leurs coordonées dans cette base λ_k^i, et le filtre moyen μ, i.e. $\mathbf{X}_i = \sum_{k=0}^{d-1} \lambda_k^i \mathbf{v}_k + \mu$.

Cette base PCA est hierarchique, c'est à dire, les termes de la base sont ordonnées par importance. Notamment, il existe une relation entre le nombre de termes de base employés et l'erreur quadratique moyen d'approximation de chaque filtre. Grâce à cela, PCA garantie de trouver des compromis optimales entre le nombre de paramètres, et l'erreur d'approximation en employant un sous-ensemble de $\mathbf{v}_k$, tel que $k \in [0, L-1]$,avec $L < d$. L'ensemble des nouveaux filtres est obtenu à partir de $\hat{\mathbf{X}}_i = \sum_{k=0}^{L-1} \lambda_k^i \mathbf{v}_k + \mu$. La Figure 1 montre un example de cette méthode appliquée à un réseau de l'état de l'art. La partie supérieure de la figure montre une reconstruction parfaite de l'esemble des filtres originaux en gardant la totalité des filtres de base, 27 dans ce cas. La partie inférieure montre une approximation obtenue en gardant uniquement les 11 premiers filtres de base. Cela produit une erreur quadratique moyenne de l'ordre de 6.83×10^{-5} tandis que le nombre de parametres est réduit d'un facteur $\times 2.02$. Le rémplacement dans le CNN des filtres originaux par cette version réconstruite ne produit aucune dégradation en termes de précision de détection dans ce cas.

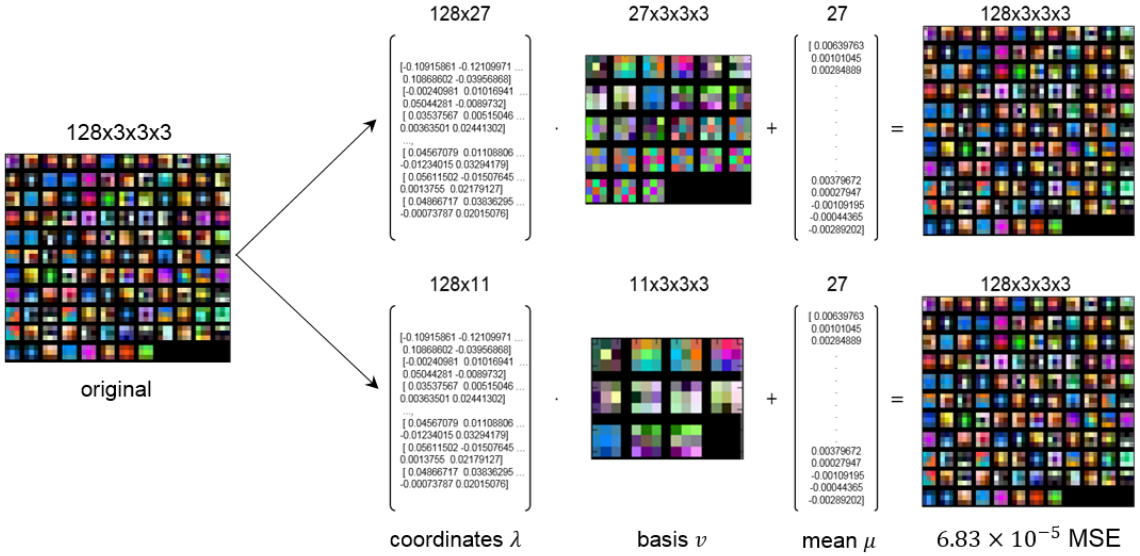


Figure 1 – Decomposition par PCA pour un ensemble de filtres d'un réseau de l'état de l'art.

Cependant, quand cette méthode est appliquée à travers toutes les couches du réseau, il devient compliqué de choisir le dégré d'approximation idéal pour chacune des couches individuelles. De plus, l'accumulation d'erreures causée par la composition de couches approximatives, produit des grandes pertes en termes de précision de détection.

L'une des solutions effectives pour réduire la dégradation en précision de détection, c'est de réaliser une légère étape de réentraînement. Lors de ce réentrainement, l'ensemble de filtres de base $\mathbf{v}_k$ reste fixe, et juste l'ensemble des coordonées λ_k^i est modifiable. Cette méthode se montre effective

pour plusieurs familles d'architectures de réseaux, dégrée de complexité de tâches, et bases de donées. La figure 2 montre l'efficacité de cette phase de réentraînement pour un réseau du type ResNet-32, sur la base de données CIFAR-10. Grace à cette méthode,il est possible, par example, de réduire le nombre de paramétres dans ce réseau de 0.453M à 0.226M (x2 gain mémoire), pour une perte en précision de détection inférieure à 2

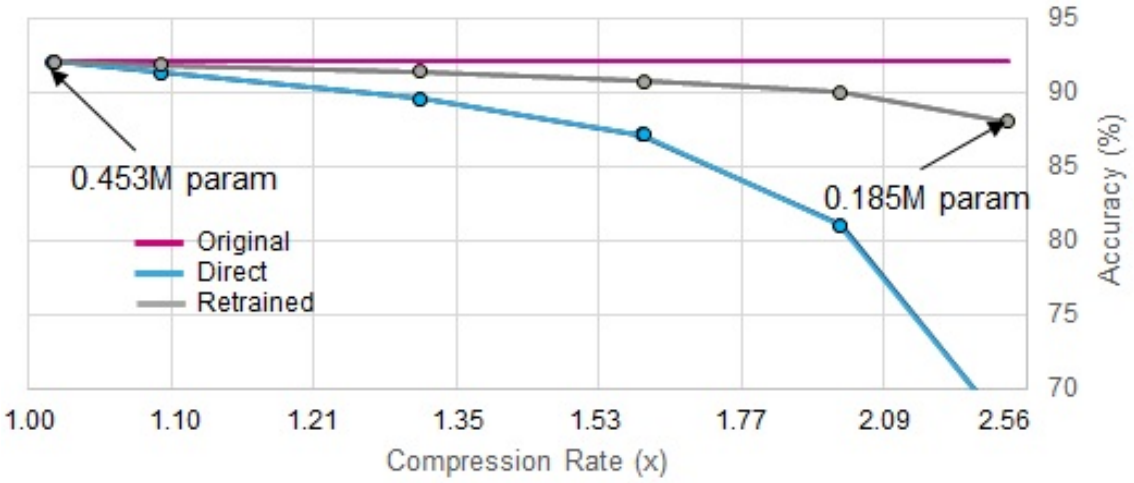


Figure 2 – Effet du réentrainement comparé à l'application directe de PCA sur un réseau du type ResNet-32 sur CIFAR-10.

Par ailleurs, une réduction équivalente en nombre d'opérations est obtenue grâce à PCA. Cela émerge du fait que PCA et la convolution soient des opérateurs linéaires. Cela permet de réaliser les convolutions uniquement avec les termes de base, pour ensuite recombinér les *feature maps* résultants à partir des coordonnées PCA correspondantes.

La compatibilité de la méthode proposée avec des autres méthodes de compression est ensuite vérifié. Par exemple, en combinant PCA, pruning et quantification, la taille mémoire d'un réseau du type ResNet-110 est réduite de 6.88Mbytes à 370kbytes, i.e. une réduction mémoire de x19 causant une perte en précision de détection de l'ordre de 3.9 %.

0.2 Détection d'objets efficiente avec CNN

Les techniques discutées précédemment ont pour objectif la réduction de la taille mémoire, ce qui est d'une importance primordiale pour l'implementabilité dans des systèmes embarqués. Cependant, en fonction de l'application visée, le nombre d'opérations peut augmenter drastiquement et devenir encombrant. Par exemple, des méthodes de l'état de l'art en détection d'objets telles que YOLO [57] ont une taille mémoire dans l'ordre des 194 MBytes et requièrent environ 62 millards d'opérations par image (résolution VGA). Ces chiffres sont complètement inabordables dans des systèmes embarqués. C'est pour cela que, dans le Chapitre 4, une méthode générique permettant d'adapter la complexité computationnelle d'un détecteur d'objets est proposé. Les étapes principales de cette méthode sont résumés dans la Figure 3.

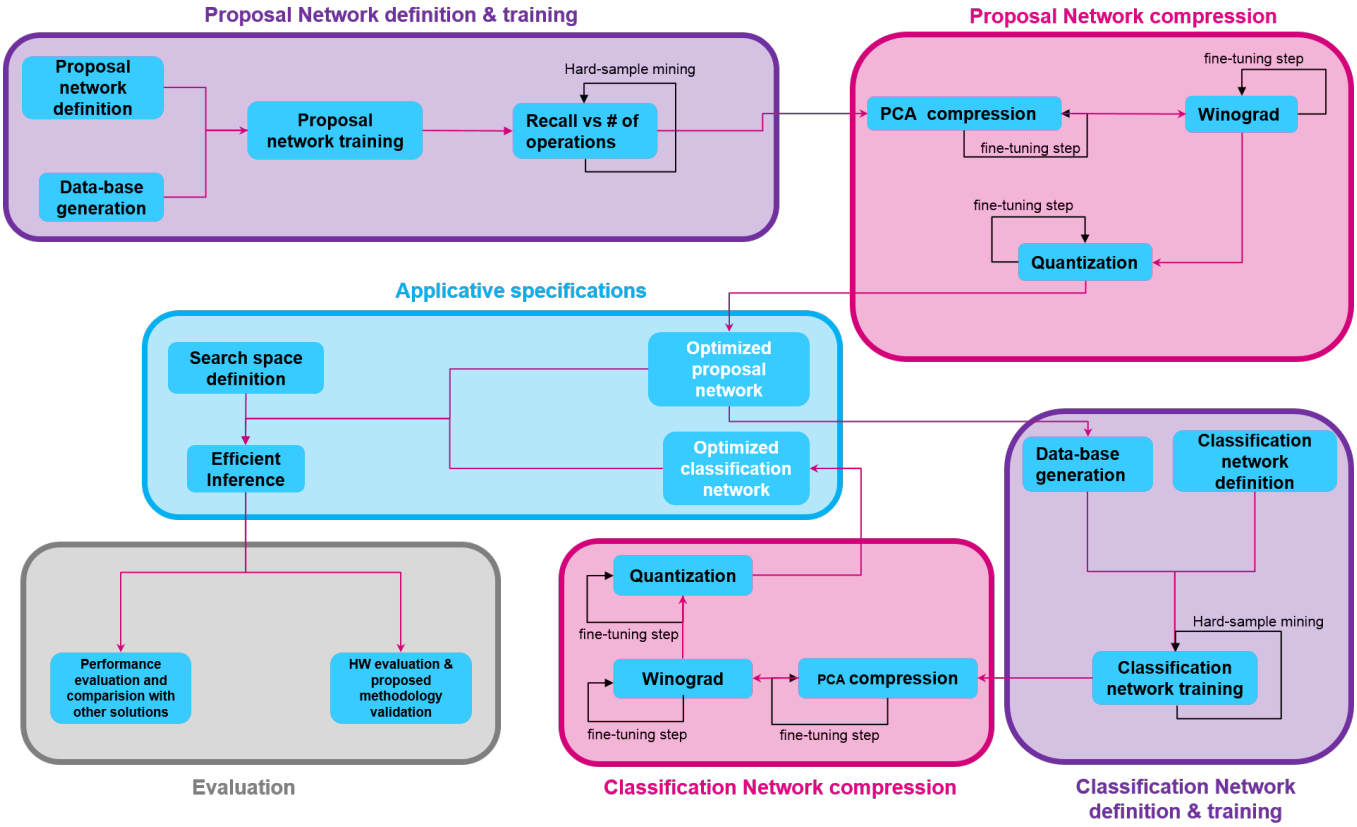


Figure 3 – Aperçu global de la méthode.

Concrètement, le processus de détection est décomposé en deux étapes. Un premier CNN sélectionne les régions d'intérêt dans l'image. Ensuite, un deuxième CNN classifie les régions proposées pour confirmer la présence ou l'absence des objets d'intérêt. Cela permet d'ajuster de manière très flexible le nombre d'opérations par rapport au taux de détection.

En effet, le réseau des propositions est conçu pour maximiser le nombre de régions d'intérêt détectés, en minimisant le nombre de paramètres employés, le nombre d'opérations, et le nombre de propositions passées au deuxième réseau. Pour un nombre de paramètres fixé, le nombre de régions d'intérêt détectés augmente lorsque le seuil de décision est abaissé. En revanche, le nombre d'opérations du deuxième réseau augmente en proportion. C'est pour cela que ces seuils doivent être soigneusement fixés en fonction des spécifications de l'application.

De plus, le réseau des propositions ayant une taille d'entrée fixe, il doit être appliqué à travers toutes les positions et échelles de l'image. Cela implique que le nombre d'opérations nécessaire pour générer les propositions (zones d'intêret) dépend fortement de la complexité de l'espace de recherche, i.e. ; de la résolution de l'image d'entrée, ainsi que du nombre d'échelles intermédiaires. On montre que les réseaux de neurones ont la capacité de réduire cet espace de recherche (et par consequént le nombre d'opérations) à travers des techniques telles que la régression des coordonnées, comme illustré dans la Figure 4.

Le réseau de classification est ensuite appliqué à l'ensemble des propositions. Dans ce réseau, l'enjeu est d'obtenir le plus haut taux de vrais positifs avec le plus bas taux de faux positifs sur l'ensemble de vrais visages, propositions générées, et non-visages. Puisque la classification précise à travers les CNNs nécessite d'un grand nombre de paramètres, ce réseau comporte la plupart des paramètres de la solution. C'est pour cela que la réduction en nombre de paramètres de ce réseau

est d'une grande importance, surtout lorsque l'application requiert des hauts taux de détection, et le seuil de décision du réseau des propositions est levé. Dans ce régime, le coût de classification des propositions surpasse largement le coût de génération de celles-ci par le réseau des propositions. C'est pour cela qu'un bloc de compression est proposé, comme illustré dans la Figure 3. Ce bloc est constitué d'une séquence de méthodes compatibles : PCA, quantification et winograd, tous avec leurs respectifs entrainements. Dans le cas du réseau des proposition, ces méthodes de compression peuvent aussi être appliqués, mais l'intérêt est plutôt une réduction en nombre d'opérations, plutôt que des paramètres, surtout dans un régime où le nombre de propositions est faible, et le nombre d'opérations sur les deux réseaux sont équilibrés.

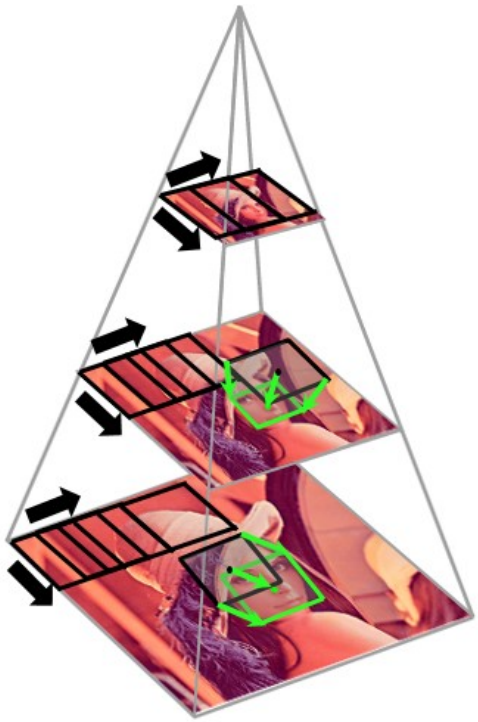


Figure 4 – La prédiction de boîtes englobantes (en vert) permet de réduire la complexité de l'espace de recherche par rapport à une méthode traditionnelle (en noir).

0.3 Détection de visage

Cette méthode de conception de détecteurs d'objets efficients est ensuite illustrée et mise en pratique dans le Chapitre 4. Le contexte applicatif choisi est la détection de visage dans un cadre collaboratif avec l'utilisateur, comme par exemple à travers la caméra frontale d'un smartphone. Notamment, pour une image en entrée en format VGA (résolution 640x480), seulement les visages de taille 64x64 doivent être détectés.

Pour cela, une solution comportant 82.7k paramètres est initialement proposée. Le flux d'exécution est détaillé dans le diagramme de la Figure 5. Dans ce diagramme, on voit que la solution ne requiert pas d'une analyse multirésolution intensive, en effet, trois niveaux pour l'analyse multirésolution suffisent. Cela est possible grâce au dégree élevé d'equivariance à l'échelle obtenu dans le réseau de propositions lors de l'entraînement, à travers de méthodes telles que la régression des

coordonnées des boîtes englobantes, comme illustré dans la Figure 4.

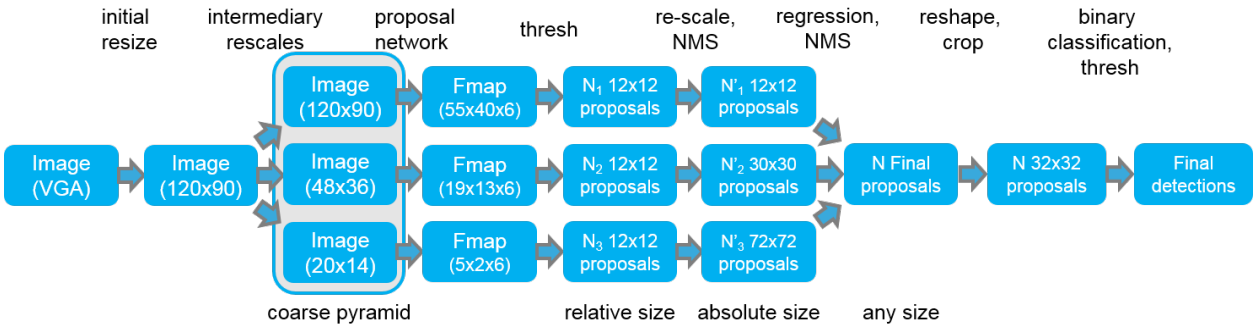


Figure 5 – Diagrame d'execution de la solution.

La Figure 6 ci-dessous, vérifie cette hypothèse et montre que, avec le réseau proposé, le nombre de régions d'intérêt retrouvés par rapport au nombre de propositions moyennes reste approximativement équivalent pour des différents niveaux de résolution (3,5 et 11 dans l'image).

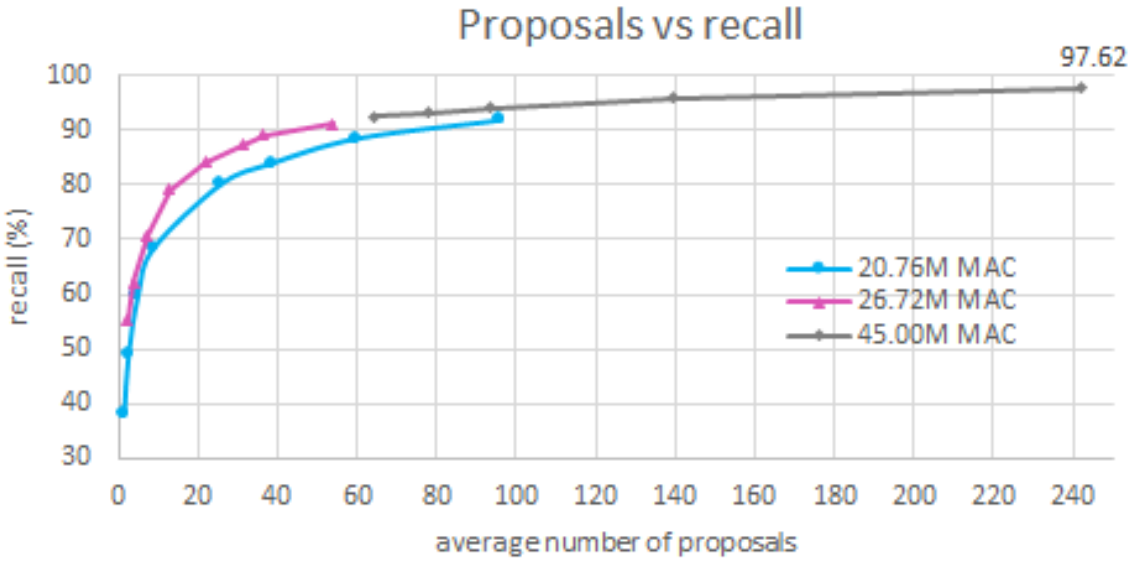


Figure 6 – Performance du réseau des propositions pour des différents niveaux de détail dans l'analyse multi-résolution.

La Figure 6, montre aussi, que, quand le seuil de décision du réseau des propositions est abaissé, le nombre de régions d'intérêt retrouvés augmente, tout en augmentant aussi le nombre moyen de propositions générées. Cela déséquilibre le coût global de la solution en termes de nombre d'opérations : le réseau de propositions avec 3 échelles intermédiaires nécessite de 20.76M MAC, pendant que le réseau de classifications nécessite de 5.73M MAC par proposition. C'est pour cette raison que plusieurs compromis sont ensuite étudiés à travers de PCA, comme le montre la Figure 4.5.

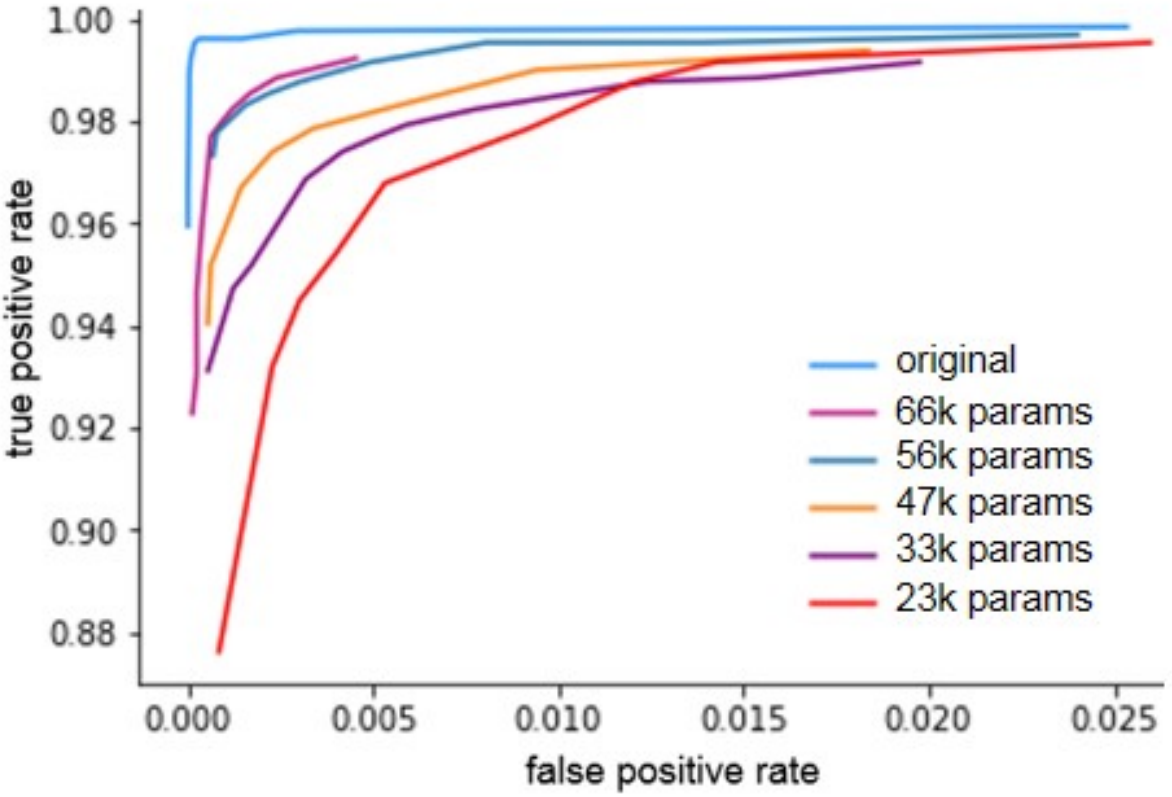


Figure 7 – Plusieurs compromis à travers de PCA entre taux de vrai possitifs, taux de faux possitifs et nombre de paramétres.

Concrètement, un seuil d'énergie de 60% est appliqué sur toutes les couches du réseau de classification original, et réduit à 23k paramètres. Cela représente un gain x3.27 en termes de nombre de paramètres et opérations. La solution globale est alors réduite à 29.3k paramètres, avec 6.3k pour le réseau des propositions, et 23k pour le réseau de classification. Les poids et activations étant quantifiés sur 8-bit, la taille mémoire du modèle est uniquement 29.3kBytes. Ceci, représente une réduction de l'ordre de x65 par rapport à des méthodes de l'état de l'art telles que MTCNN [103]. Les performances de la solution proposée sont ensuite évalués sur la base de données CelebA [113], qui est bien représentative du cadre applicatif proposé. Plusieurs régimes de performance sont montrés dans la Figure 8, ainsi qu'une comparaison avec MTCNN et un algorithme classique basé sur Viola-Jones. Le CNN proposé, nécessite d'environ un ordre de magnitude moins de calculs que Viola-Jones, et en même temps, montre une habilité de mise à l'échelle pour des scénarios applicatifs plus compliqués, qui est beaucoup plus coûteuse à atteindre dans le cas de Viola-Jones.

Le réseau des proposition est aussi évalué sur l'ASMP, pour plusieurs résolutions d'image en entrée. Comme l'illustre l'image 9, l'efficience en termes de MAC/cycles augmente lorsque la résolution augmente. Approximativement, un taux de détection de 93.77% est obtenu dans la base de données CelebA [113] avec 1.674 million de cycles par frame.

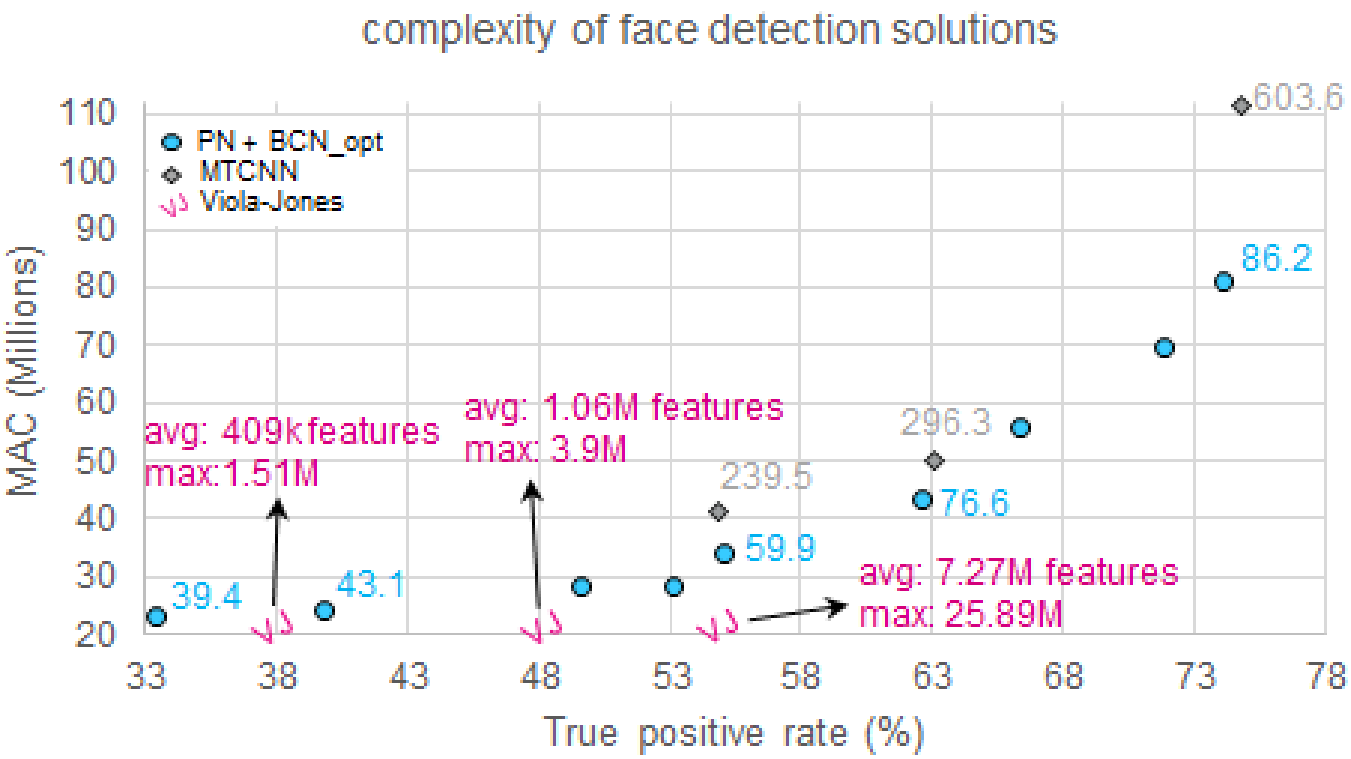


Figure 8 – Comparaison de la complexité computationelle de plusieurs méthodes de détection de visage.

0.4 Évaluation hardware

Dans le Chapitre 5 de ce manuscrit, les réseaux proposés sont ensuite évalués sur l'ASMP, un multiprocesseur embarqué conçu par ST. L'objectif est de mesurer et comparer le gain obtenu par PCA par rapport au gain théorique, ainsi que la performance des réseaux proposés pour la solution de détection de visages.

Les résultats pour le réseau de classification comprimé par PCA sont reportés dans le tableau 1 ci-dessous, qui confirme un gain approximativement consistent avec le gain théorique. Dans ce cas, le gain théorique obtenu par PCA est de x3.27, tandis que le gain mesuré sur un seul PE est de x2.63 par rapport au réseau original. Cela montre des effets de bord produits par l'irrégularité des dimensions des filtres de base PCA, concrètement entre le nombre de filtres d'entrée et sortie. De plus, cette table montre que, lorsque le réseau est parallélisé sur 8 PEs, une accélération de l'ordre x4.44 par rapport au réseau sur 1 seul PE est obtenue. Le gain obtenu en parallélisant le réseau comprimé par PCA sur 8 PEs, comparé au réseau original sur un PE s'élève à x11.68.

# PEs	1	2	4	8
Original network				
# cycles	1,386,170	782,452	461,247	290,702
MAC/cycle	4.40	7.80	13.24	21.00
Parallel speed-up	x1.00	x1.77	x3.01	x4.77
After PCA + retraining ; fused basis and recombination				
# cycles	625,193	355,677	211,755	139,802
MAC/cycle	2.99	5.26	8.83	13.38
Alg. speed-up	x2.22	x2.20	x2.18	x2.08
Parallel speed-up	x1.00	x1.76	x2.95	x4.47
After PCA + retraining ; fused basis and recombination + alignment				
# cycles	527,214	298,276	177,839	118,637
MAC/cycle	3.75	6.62	11.11	16.66
Alg. speed-up	**x2.63**	**x2.62**	**x2.60**	**x2.45**
Parallel speed-up	**x1.00**	**x1.77**	**x2.96**	**x4.44**
Total speed-up	**x2.63**	**x4.65**	**x7.79**	**x11.68**

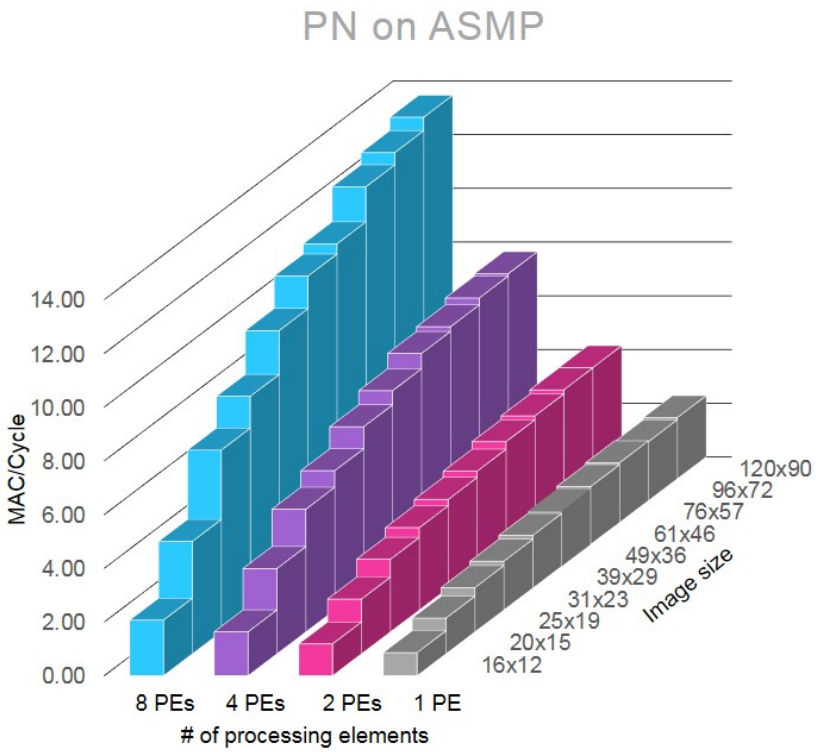


Figure 9 – Performance du réseau de propositions sur l'ASMP.

0.5 Conclusion génerale

À travers de cette thèse, plusieurs méthodologies sont proposées, qui montrent l'efficacité d'une approche de conception conjointe entre l'algorithme et l'architecture hardware. Notamment, l'algorithme détermine en grande mesure l'implentabilité de la solution dans des systèmes embarqués. Ainsi, le but primordial consiste à trouver des algorithmes qui sont capables de réduire le nombre de calculs et paramètres d'une solution de vision par ordinateur, tout en prenant en compte des caractéristiques bas niveau.

Pour cela, une méthodologie générale permettant d'adapter la complexité du classifieur CNN lui-même à été proposée. Ensuite, une autre méthodologie générale est proposée. Elle permet d'appliquer ce classifieur de manière efficace à travers une image pour réaliser de la détection d'objets. Même si cela introduit de légers effets de bord, les gains sont importants et les méthodes restent largement amenables à la parallelisation.

La diffculté c'est donc qu'il faut avoir de diverses connaissances, qu'il y a de nombreux paramètres à régler, et qu'il est nécessaire d'établir des méthodes pour aboutir à une solution efficace et cohérente.

Table des matières

Abstract **3**

Résumé Étendu **6**

0.1 Compression des CNN . . . 6
0.2 Détection d'objets efficiente avec CNN . . . 8
0.3 Détection de visage . . . 10
0.4 Évaluation hardware . . . 13
0.5 Conclusion génerale . . . 15
Table of contents . . . 18

Introduction **1**

0.5.1 Motivation . . . 1
0.5.2 Problem formalization . . . 3
0.5.3 Contributions and thesis outline . . . 6

1 Deep Learning Overview **8**
1.1 Neural Network reminder . . . 9
1.1.1 Artificial Neural Networks . . . 9
1.1.2 Convolutional Neural Networks . . . 10
1.1.3 Activation Layer . . . 13
1.1.4 Pooling/subsampling Layer . . . 14
1.1.5 Fully Connected Layer . . . 15
1.1.6 Softmax Layer . . . 15
1.1.7 Learning Features . . . 16
1.1.8 Feature interpretability . . . 17
1.1.9 Loss Function . . . 17
1.1.10 Minimizing the loss function . . . 18
1.1.11 Topology of the loss function . . . 20
1.1.12 Training and test distributions . . . 22
1.1.13 Overfitting and Underfitting . . . 23
1.1.14 Deep Learning vs traditional machine learning . . . 25
1.1.15 Deep Learning Frameworks . . . 25
1.1.16 Deep Learning in practice . . . 26
1.2 CNN architecture comparison . . . 27
1.2.1 AlexNet . . . 28
1.2.2 ZFNet . . . 28
1.2.3 VGG . . . 28
1.2.4 Network in Network . . . 28
1.2.5 GoogleNet/Incenption . . . 28
1.2.6 HighWay Networks . . . 28

1.2.7 ResNets . . . 29
1.2.8 Wide ResNet . . . 29
1.2.9 ResNext . . . 29
1.2.10 DenseNet . . . 29
1.2.11 Squeeze-and-Extraction . . . 29
1.2.12 FractalNet . . . 29
1.2.13 MobileNets . . . 29
1.2.14 ShuffleNets . . . 30
1.2.15 SqueezeNet . . . 30
1.2.16 NASNet . . . 30
1.3 Realistic CNN applications (as of now, 2019) . . . 31
1.3.1 Wake-up systems . . . 31
1.3.2 Image Classification . . . 31
1.3.3 Object Detection . . . 31
1.3.4 Depth Estimation . . . 32
1.3.5 Image Segmentation . . . 32
1.3.6 Gesture Recognition . . . 33
1.3.7 Image reconstruction/extrapolation/compression/noise reduction . . . 33
1.3.8 Natural Language Processing . . . 34
1.3.9 Image Understanding . . . 34
1.3.10 Autonomous Navigation . . . 34
1.4 CNN hardware accelerators . . . 36
1.5 Neuromorphic computing . . . 38

2 Methodology to adapt the computational complexity of CNN-based object detection for efficient inference in an applicative use-case 40
2.1 Computational Complexity of object detection . . . 41
2.2 State of the art in object detection . . . 43
2.2.1 RCNN . . . 43
2.2.2 SPPNet . . . 43
2.2.3 Faster RCNN . . . 44
2.2.4 YOLO & SSD . . . 44
2.3 Proposed methodology to optimize CNN object detector complexity for applicative use-case . . . 45
2.3.1 Applicative use-case specifications . . . 48
2.3.1.1 Search Space definition . . . 49
2.3.1.2 Efficient inference . . . 49
2.3.2 Proposal Network definition and training . . . 50
2.3.2.1 Proposal Network definition . . . 50
2.3.2.2 Database generation for PN . . . 52
2.3.2.3 Proposal Network training . . . 54
2.3.2.4 Tuning trade-off between recall and number of operations . . . 55
2.3.3 Proposal Network compression . . . 55
2.3.3.1 PCA compression . . . 56
2.3.3.2 Winograd . . . 56
2.3.3.3 Quantization . . . 57
2.3.4 Binary Classification Network definition and training . . . 57
2.3.4.1 Binary Classification Network definition . . . 58
2.3.4.2 Database generation for BCN . . . 58
2.3.4.3 Binary Classification Network training . . . 58
2.3.5 Classification Network compression . . . 58

2.3.6 Evaluation . . . 59
2.3.7 Algorithm benchmarking . . . 59
2.3.8 Hardware evaluation . . . 60
2.3.9 Conclusion and Perspectives . . . 61

3 CNN compression **63**
3.1 CNN compression methods . . . 65
3.1.0.1 Quantization . . . 65
3.1.0.2 Pruning . . . 66
3.1.0.3 Entropy Encoding . . . 68
3.1.0.4 Weight Sharing . . . 68
3.1.0.5 Distillation . . . 68
3.1.0.6 Filter Parametrization & Transformation . . . 69
3.2 PCA for CNN compression . . . 70
3.2.1 PCA for memory reduction . . . 70
3.3 PCA for computation reduction . . . 72
3.4 Experiments . . . 73
3.4.1 Uniform compression . . . 74
3.4.2 Progressive compression . . . 76
3.4.3 Trained reconstruction . . . 77
3.4.4 PCA for filter removal . . . 79
3.5 PCA combined with other compression methods . . . 81
3.5.1 PCA combined with pruning . . . 81
3.5.1.1 Structured pruning . . . 81
3.5.2 PCA combined with pruning and quantization . . . 83
3.5.3 PCA combined with winograd . . . 83
3.5.3.1 Example with ResNet110 on CIFAR-10 . . . 84
3.5.4 PCA combined with binary quantization . . . 85
3.6 Conclusion . . . 85
3.7 Perspectives . . . 86

4 Cascaded and compressed CNNs for fast and lightweight face detection **88**
4.1 State of the art in face detection . . . 90
4.1.1 Viola-Jones . . . 90
4.1.2 CNN Cascade . . . 92
4.1.3 MTCNN . . . 92
4.1.4 LightweightCNN . . . 92
4.2 Fast and Lightweight Face Detection through CNNs . . . 93
4.2.1 Proposed solution . . . 93
4.2.2 Proposal Network Optimization . . . 95
4.2.2.1 Proposal/recall trade-off . . . 95
4.2.2.2 Proposal network speed-up . . . 97
4.2.3 Binary Classification Network optimization . . . 98
4.2.3.1 Binary Classification Network compression and speed-up . . . 98
4.2.4 Comparison with other solutions . . . 100
4.3 Conclusions . . . 102
4.4 Perspectives & possible improvements . . . 102
4.4.1 General object detection . . . 102
4.4.2 Near-Infrared image sensors . . . 102
4.4.3 Event-based sensors . . . 103

5 Hardware evaluation on embedded multiprocessor **105**
5.1 State of the art CNN accelerators . . . 107
5.1.1 Sparse accelerator . . . 107
5.1.2 Winograd accelerator . . . 108
5.1.3 Sparse Winograd accelerator . . . 108
5.1.4 Binary/ternary network implemenations . . . 109
5.2 PCA evaluation on ASMP . . . 110
5.2.1 ASMP overview . . . 110
5.2.2 Compressed BCN evaluation on ASMP . . . 111
5.3 Proposal network evaluation on ASMP . . . 114
5.4 Face Detection solution evaluation on ASMP . . . 115
5.5 Conclusion and Perspectives . . . 116

Thesis Conclusion & Perspectives **118**

Bibliography **I**

Table des figures **VI**

Liste des tableaux **XIII**

Acronyms

NN	Neural Network
CNN	Convolutional Neural Network
CV	Computer Vision
NLP	Natural Language Processing
AI	Artificial Intelligence
SOTA	State-of-the-art
VJ	Viola-Jones
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ASMP	Application Specific Multi-Processor
MAC	Multipply Accumulate
DRAM	Dynamic Random Access Memory
SRAM	Static Random Access Memory
PCA	Principal Component Analysis
HW	Hardware
PE	Processing Engine
RF	Register File
NOC	Network On Chip
ANN	Artifivial Neural Network
SNN	Spiking Neural Network
RNN	Recursive Neural Network
LSTM	Long-Short Term Memory
GAN	Generative Adversial Networks
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
ISP	Image Signal Processor
DCT	Discrete Cosine Transform
RLC	Run Length Coding

Introduction

0.5.1 Motivation

Ever since the seminal work of [10], *Neural Networks* (NNs), and more concretely *Convolutional Neural Networks* (CNNs), have greatly regained interest from the research community. This has lead to major breakthroughs mostly in the areas of *Computer Vision* (CV) and *Natural Language Processing* (NLP), and a new wave of NN-centered *Artificial Intelligence* (AI) research has emerged.

At the same time, devices such as *smart image sensors* can greatly benefit from the ability to perform the required computations locally on the sensor, as opposed to the current computing paradigm where NN-based algorithm's *inference* is run on the server-side. This motivates the interest for the integration of the CNN inference stage (without training) in *embedded systems*. Some of these benefits include :

— Privacy : since the processing is done directly inside the image sensor, the image never leaves the sensor. This poses a change of paradigm, as compared to the typical case, where the image needs to be transmitted to a server in order to carry out the processing.

— Efficiency : the energy consumption required to send an image to a server, as opposed to locally processing, is currently of the order of 1000 ops/sent pixel. Even with newer technologies such as 5G, this cost will remain high and clearly shows the advantage of a local processing scheme.

— Latency : since transmitting images is not necessary, data movement is minimized and only metadata would need to be transmitted, greatly reducing communication issues and processing latency.

— System scalability : since each sensor is able to process its input independently, a more distributed system could be deployed, whereas a centralized system would quickly run into processing bottlenecks.

— Cost : these sensors should be task-specialized and optimized for cost, driving down the global cost of the solution, as opposed to more general and high-end processing systems. This, clearly being one of the major keys to the democratization and full-scale deployment of AI solutions.

— Sustainability : Finally and most importantly, the current trends in AI for which it is claimed that a *deep learning* system's performance increases and scales well with data & computational complexity seems like a clear indicator of *over-fitting/memorization*. While this might help advance research in its current state, it doesn't seem sustainable in the long-term and clearly hints to the need for new concepts for "smarter" algorithms. Indeed, as will be discussed throughout this tesis, algorithm complexity almost completely dimensions the hardware complexity, and is the main driver of energy consumption. Therefore, more efficient algorithms are the most crucial aspect to work on.

For the above reasons, being able to integrate CNN solutions inside the sensor is a necessary path to follow, and it is the key to sustainable full-scale deployment and democratization of AI solutions.

While this has already been achieved for very specific kinds of applications through more traditional CV algorithms, CNNs are good candidates to replace them, since they bring improved accuracy, and can be seen as a generic tool that is able to scale to harder problems. The computations involved in CNNs are also rather suitable for hardware implementations due to the regularity of computations, and are therefore easily *parallelizable*. All these reasons strongly motivate the replacement of more traditional CV algorithms currently used in smart image sensors by CNN-based solutions.

However, the field is still in its infancy and greatly lacks theorical understanding, which makes of it a very empirical and subjectively benchmarked field. Although the state-of-the-art "seems" to advance very quickly, there currently is a great lack of work on the intersection of algorithms and hardware, with the AI/CV research relying on powerful hardware, and not adequately valuing the importance of efficient algorithms, while the lack of the theoretical competence on the hardware side easily leads to the implementation of over-dimensioned algorithms. This makes it very hard to bridge the two domains, and makes it complicated to clearly define and dimension real application scenarios, especially when processing efficiency is a priority and solutions need to be fully optimized. Indeed, it is clear that most CNN models from the state-of-the-art(SOTA) are greatly *over-parametrized*, leading to high *computational complexity* of CNN solutions, and making it impossible to efficiently implement CV solutions from the SOTA in constrained systems.

Furthermore, properly defining the task's complexity (according to an applicative use-case scenario), allows to bound the computational complexity of the CNN-based solution. In this sense, the acquisition of a representative data-base is also a crucial but non-trivial task.

Finally, since the neural network's outputs can be very vaguely explained, the promotion of a "black-box" solution also seems to be a blocking point. Although paradoxal, this mainly implies that, for now, targeted applications are reduced to those where a failure in the system does not have important implications, e.g. : wake-up systems. There are two main options to overcome this barrier : either the system is proven to perform better than a human expert, or new interpretability mechanisms need to be found, ideally both.

Therefore, it is clear that this is a very inter-disciplinary and complex field, and that, in order to achieve this goal, there needs to be both further breakthroughs and a very good synergy between *Applications*, *Algorithms* and *Hardware*, as summarized in Figure 10.

More concretely, Figure 10 shows the main metrics that need to be leveraged in each of the domains in order to achieve state-of-the-art efficient solutions :

— Algorithm : the main challenge consists in reaching good trade-offs between *accuracy* (i.e. percentage of correct predictions with respect to total predictions) and *computational complexity* (i.e. number parameters and operations). E.g. : Over-dimensioned CNN vs. compressed CNN vs. classical algorithm (e.g. Viola-Jones).

— Hardware : the main challenge consists in reaching good trade-offs between number of supported operations (*throughput*), *silicon area*, and *energy consumption*, as well as *cost* of the solution. E.g. : GPU vs. CPU vs. ASMP.

— Application : the main challenge consists in reaching good trade-offs between task complexity and computational complexity. E.g. : Face detection for *consummer applications* vs. *surveillance application*.

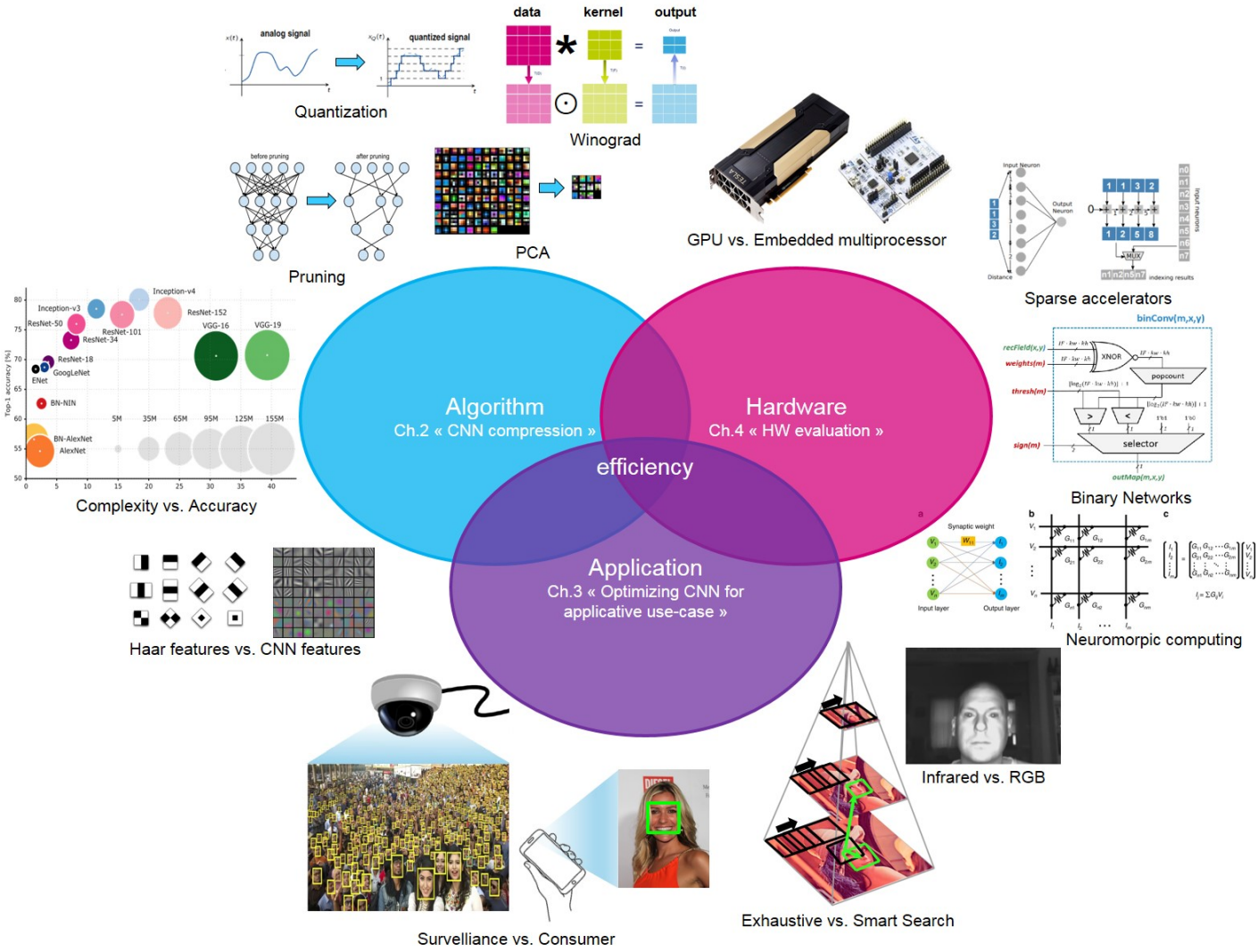


Figure 10 – Domains of competence necessary to achieve efficient CNN solutions

Figure 10 shows that these are mainly orthogonal domains, however, state-of-the-art efficient solutions require the co-adaptation and co-design of all of these domains. This will be demonstrated throughout this thesis in the specific case of face detection. On the other hand, treating any of the domains as a black-box will likely result in inefficient and over-dimensionned solutions.

0.5.2 Problem formalization

This section aims to illustrate the source of the problem for the processing of CNNs in constrained embedded systems. The main goal of the next paragraphs is to illustrate the basic operation in CNNs, which will allow to identify one of the sources of the problem, while CNNs will be overviewed in greater detail in section 1.1.2.

The basic operation in CNNs is the *convolution*, hence its name. Indeed, CNNs perform a *discrete convolution* operation between an *input* and a *kernel* in order to produce an output *feature map*. Figure 11 depicts a convolution between a 2D, single channel input image, I, and a single

2D kernel, K. This can mathematically be expressed as :

$$O(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Intuitively, a convolution consists on matching a pattern present in the kernel across all possible

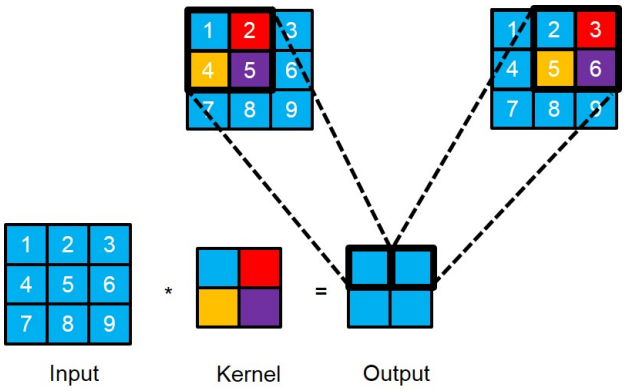


Figure 11 – Illustration of a discrete convolution between an 2D input image and a 2D convolution kernel.

positions in the image. In this sense, matching is an element-wise multiplication between the kernel and each possible position in the image. The element-wise product of each position is then summed to generate an output value. This is technically a correlation, as opposed to a convolution, since the kernel has not been flipped, but doesn't change anything with respect to CNNs, since a flipped version of the kernel would be learned.

Moreover, the main bottleneck in CNNs comes from memory access. Indeed, in order to compute convolutions, each MAC (multiply accumulation) requires the reading of 3 different memory values and 1 memory write, as shown in Figure 12.

For example, a network such as AlexNet [10] requires 724 million *multiply accumulates* (MACs) for each forward pass, which amounts to 2896 million memory accesses required to analyze a single 224x224 RGB image. Since weights in this network amount to $\approx$ 62 million (32 bit, 248 MBytes) they need to be stored in external memories such as DRAMs. However, the power consumtion is greatly dominated by data movement from & to DRAM when this strategy is adopted. This is why multiple works add a less expensive local memory layer within the processing engine and try to reuse the data loaded from main memory into these local memories as much as possible.

Moreover, our goal is to integrate CNN inference into *always-on & battery-powered* smart image sensors , i.e. sensors which constantly monitor the environment for specific events. Since these are battery-powered, energy consumption needs to be as low as possible. Therefore, the use of a power hungry memory such as DRAM is unthinkable. This greatly constrains available memory size and imposes strict boundaries on the type of CNN solutions that can be implemented. Furthermore, model size has direct impact on the silicon area and cost of the solution. Therefore techniques allowing the compression of such models are of primary interest, as will later be discussed in Chapter 3. Applicative context also plays a big role in the dimensioning of the solution, as will be shown in Chapter 4 through a realistic use-case scenario.

On the technology side, in order to build this sensor, the use of a *3D wafer stacking process* makes the most sense, i.e. the top die (circuit) contains the pixel array necessary for image acquisition, while the bottom die is mainly responsible for the memory & processing, as shown in Figure

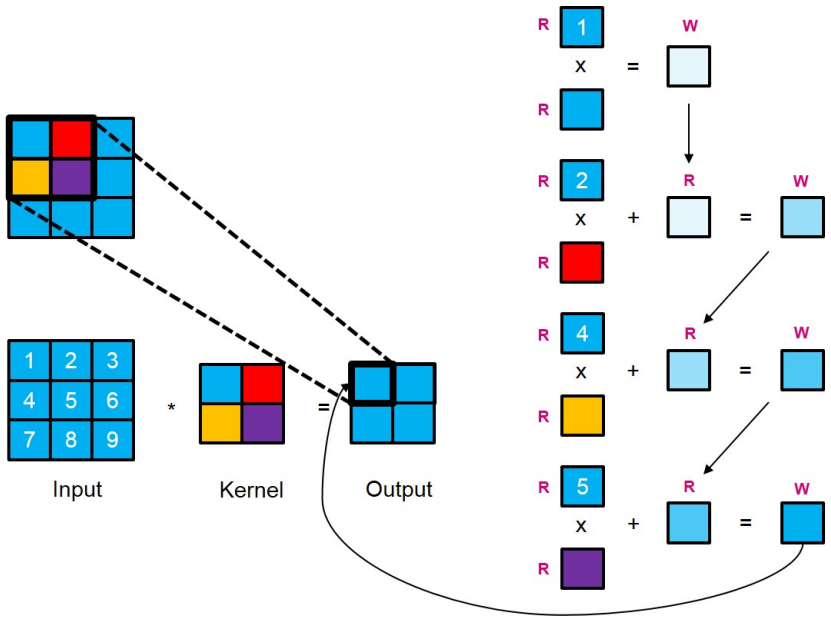


Figure 12 – Memory accesses required for convolution

13. Since these two dies are spatially close, memory access costs are greatly reduced, thereby reducing overall power consumption when pixels need to be accessed for processing.

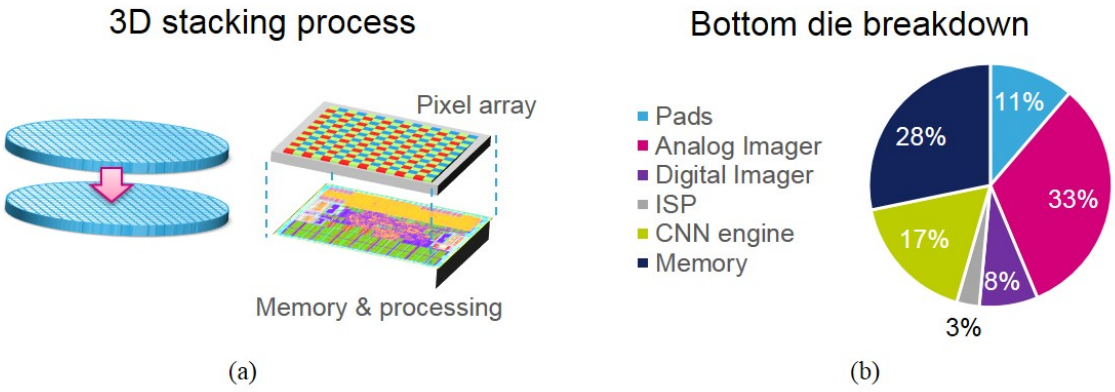


Figure 13 – 3D stacked image sensor breakdown

More concretely, the order of magnitude of the available memory budgets for a 2 Mpixel with a 2.8 μm pixel pitch 3D-stacked imager would be the following :

— Available memory with 40nm bottom die : 1.5 MBytes.

— Available memory with 28nm bottom die : 4.3 MBytes.

— Available memory with two 28nm bottom dies : 12 MBytes.

Of course, these numbers are merely indicative and greatly depend on the type of application. The lower these are, the lower the cost of the solution. Moreover, these numbers also show the mismatch between state-of-the-art CNN models and the memory budgets in our scope of applications, as will be discussed in Chapter 1.

0.5.3 Contributions and thesis outline

In this dissertation we tried to broadly look at possibilities in each of these domains allowing to jointly define a face detection solution with state-of-the-art detection rates, which is able to run in real-time on low-cost hardware.

On the algorithm side, we propose a CNN compression method [1], which is detailed in Chapter 4. The method is based on Principal Component Analysis (PCA) which is able to flexibly obtain trade-offs between memory size, number of operations, and accuracy. We also broadly review the positive and negative points of other interesting simplification methods, and check the compatibility of the proposed method with the most interesting ones.

On the application side, in Chapter 2, we propose a general methodology to adapt the computational complexity of a CNN-based object detector to an applicative use-case scenario. We show that task complexity is also a crucial dimensioning factor, and that it is imperative for the applicative scenario to be well-defined in order to fully optimize a solution. Through all this knowledge, in Chapter 4, we propose an efficient CNN-based Face Detection solution. The solution involves a cascaded approach which greatly lowers computational complexity and allows to very flexibly tune performance and complexity trade-offs. We then leverage CNN compression and speed-up methods, as well as applicative scenario knowledge in order to further reduce the computational complexity. Chapter 2 explains the general methodology and important concepts that need to be taken into account for efficiently. Chapter 4 demonstrates this methodology in a consumer face detection applicative use-case, and compares the proposed solution to other state-of-the-art CNN solutions, as well as a Viola-Jones based solution.

On the hardware side, we verify in [3] and Chapter 5 that the proposed compression method doesn't introduce new implementation challenges. We show that it greatly preserves parallelism by evaluating it on our custom embedded multiprocessor. Finally, we also separately evaluate each of the networks which make up the fully optimized face detection proposed solution in order to provide an approximate idea of the complexity of the HW solution.

1

Deep Learning Overview

The goal of this chapter is to quickly overview the main concepts in Deep Learning. We start with a quick reminder on Neural Networks and Convolutional Neural Networks. We will then provide a summarized overview and comparison of the important concepts involved in the main Neural Network architectures from the state-of-the-art. We will also review the main applications of CNNs, the datasets used for each of the tasks, and the main deep learning frameworks. Finally, we will review some of the HW accelerators present in the state-of-the-art.

Sommaire

0.1 Compression des CNN 6
0.2 Détection d'objets efficiente avec CNN 8
0.3 Détection de visage 10
0.4 Évaluation hardware 13
0.5 Conclusion génerale 15
Table of contents 18

1.1 Neural Network reminder

In this section, we quickly overview the main concepts relative to neural networks, allowing to understand both the *inference* and *training* stages. A much more detailed explanation of the important and subjacent concepts in this domain can be found in [6].

1.1.1 Artificial Neural Networks

The basic element of a *Neural Network* (NN) is the *neuron*. Neural networks are typically organized in *layers*. These layers are made up of a number of interconnected nodes (which represent the neurons) and whose purpose is to linearly combine the outputs of the neurons in the previous layer with which they share a connection through a set of *weighted connections*. These weighted connections are initialized to a -generally- random value, and will later be adjusted through a *learning/training* phase. A *bias* (acting as a threshold) is also learned for each neuron and is added to the result of the linear combination between the weights and the inputs. The output is then passed through an *activation function* in order to introduce *non-linearities*. This is shown in Figure 1.1 below. In the notation, the superscript refers to the layer number and the subscripts i and j to the neuron in the previous and current layers respectively. More concretely, the output of the j-th neuron in layer k, $y_j^{(k)}$ can be expressed as $y_j^{(k)} = f(\sum w_{i,j}^{(k)} y_i^{(k-1)} + b_j^{(k)})$, where $W_{i,j}^{(k)}$ is a matrix containing the connection weight between the j-th neuron in layer k and the i-th neuron in layer $k-1$. The term $b_j^{(k)}$ represents the bias value for the j-th neuron in layer k.

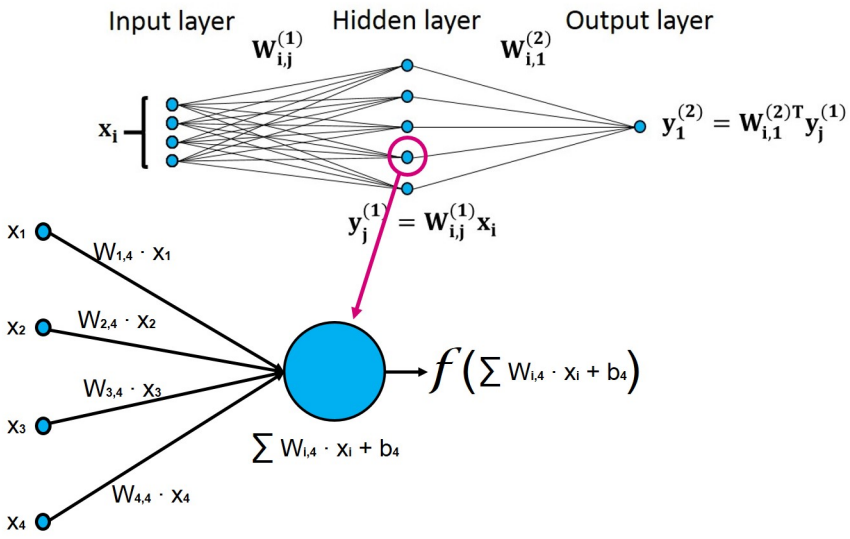


Figure 1.1 – Example showing the connection diagram for a simple one-layer fully connected network. In more detail, the operation carried out by neurons.

In a deep learning context there are multiple layers of neurons. In the case of *fully connected* NNs, as the one pictured in Figure 1.1, the neurons in a layer are connected to every neuron in the previous layer. As we will see later on, this is in fact the main difference with respect to other types of networks such as CNNs, RNNs or LSTMs. Once the *network architecture* has been fixed (i.e. number of layers, number of neurons per layer, type of connections, type of activation function, etc...), a learning phase begins in which the network is going to adjust its weights in order to minimize a *cost function*. This *cost function* measures the misfit between the *observed*(ground

truth, target...) and the *predicted* data.

At the beginning, weights are randomly initialized (with multiple possible initialization schemes). During the learning phase, patterns are presented to the network via the input layer. The network then makes a random guess as to what the output might be. It then sees how far its output was from the target output, which has previously been annotated as a ground truth in a database. The network then adjusts its connection weights. It repeats this process until the error(cost) measuring function is minimized through two algorithms known as *gradient descent* and *error back-propagation*.

1.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special case of Artificial Neural Networks (ANNs) in which the connections have been arranged in a way that produces a convolution operation, hence their name.

Convolutional neural networks have a special type of layer -the *convolutional layer* - where the convolution is produced, as was introduced in section 0.5.2. Convolution is in fact a very useful operator in the image processing domain and its understanding is crucial in order to also understand CNNs. Moreover, convolution allows to match a pattern present in the filter across the image. These filters are learned according to the task for which they have been trained. The output of the convolution (correlation really) of the image with a kernel is called a feature map, and each value of this matrix is obtained by taking the image values within a window (of the same size as the kernel) and multiplying them element-wise with the kernel, as was displayed in Figure 11. These are then summed in order to obtain a single value. The window of image values is then moved by a certain amount (*kernel stride*) and the element-wise multiplication and summation are repeated, as illustrated in Figure 1.2. Each of the values in the output feature maps therefore represent the absence or the presence of the filter's pattern inside the image.

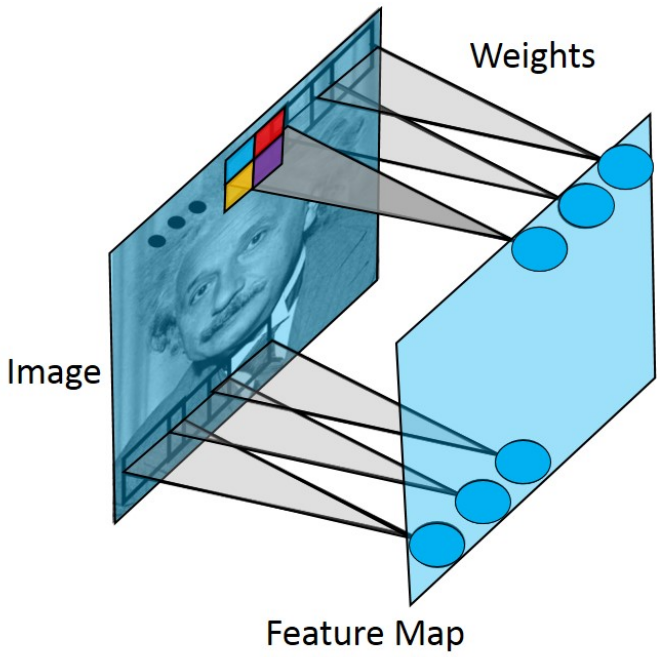


Figure 1.2 – Convolution through neural networks : Output neurons for a feature map (second layer) are repeatedly connected through the same set of weights to neurons containing shifted windows of the input image. These connections are displayed in greater detail in Figure 1.3.

To reproduce this operation through neural networks, the input image's pixel values are presented to the input of the neurons on the first layer and, for each kernel, the same set of weighted connections are repeated between the input neurons (image values) and the output neurons (feature map values) across the image. In other words, the weights for each of the neurons belonging to the same feature map must be the same as the weights of its neighbor, but with a shifted version of the input layer. Since each neuron is performing an element-wise product between the input and the weights, followed by an addition, the weights can be seen as the convolution kernel, as pictured in Figure 1.2. The connection diagram necessary to achieve convolution with a neural network is shown through an example in Figure 1.3.

There are some remarkable advantages to this connectivity scheme of neural networks :

— This implementation vastly reduces the amount of parameters in the network with respect to fully connected NNs since neurons representing the feature maps are just connected to a window of neurons in the input layer instead to all of them (as done in fully connected networks). What's more, experiments have shown that, even though a fully connected neural network could potentially learn the convolution connection scheme, it is not able to do so in practice [4, 5]. This highlights the importance of the right *a prioris* , i.e. beliefs about reasonable model choices before seeing any data, such as the weight connectivity for convolution and hierarchical composition of features in the case of images.

— Since the learned weights make up the convolution kernels, this means that during training, the best representations for the images are learned directly from the data. This is as opposed to other more traditional Computer Vision algorithms (such as Viola-Jones) which aim to generate interesting features by fixing the basis set (*haar functions* in the case of VJ), and which are indeed less appropriate than the features learned through data, as in CNN.

— This weight connection scheme mainly exploits the fact that an object may appear at different positions inside an image. By exhaustively searching for patterns across the image through convolution kernels, CNNs are inherently *translation invariant*, this being the main

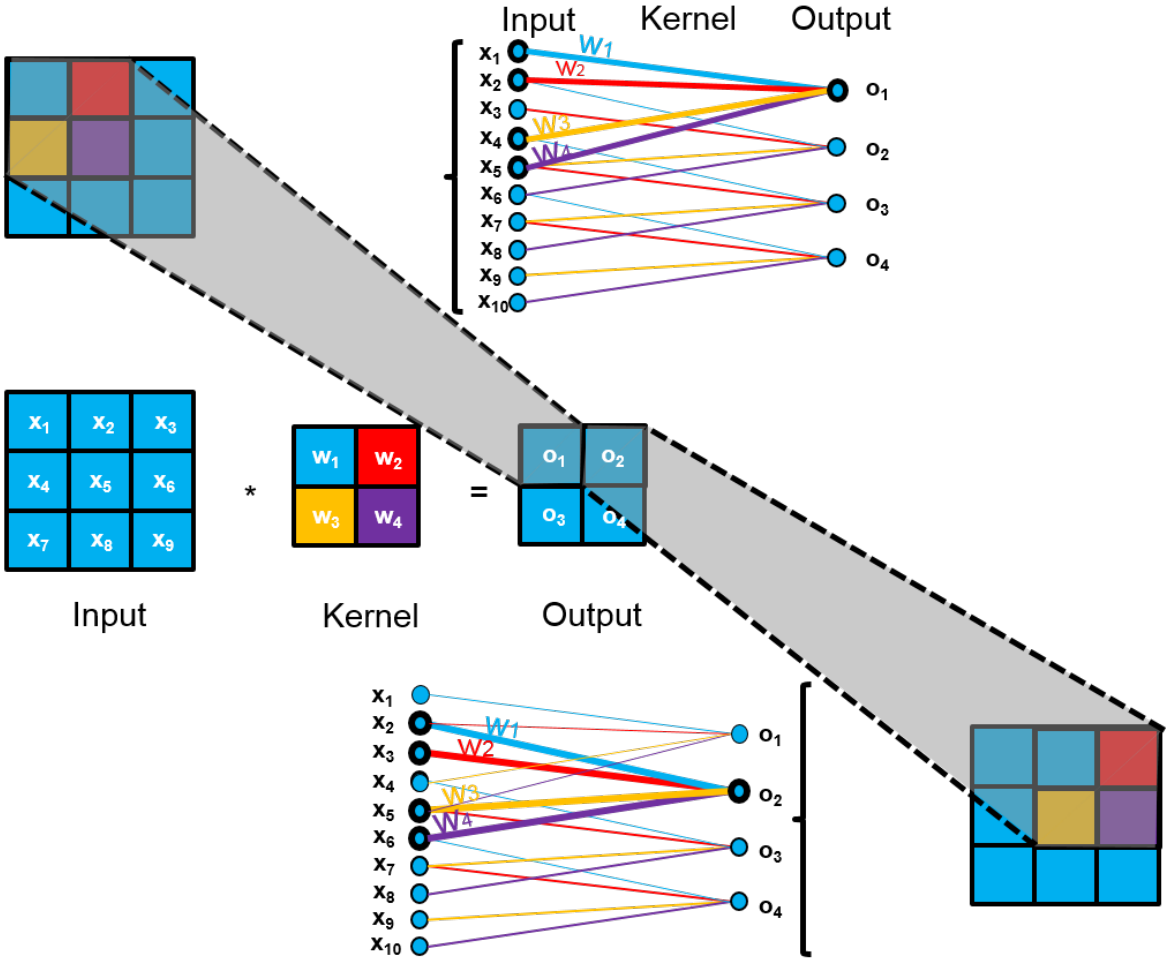


Figure 1.3 – Actual connection diagram for a neural network performing a 2×2 convolution on a 9×9 input image, highlighting the operation carried out in order to compute the output of the first two elements of the feature map.

reason why they perform well in vision tasks. Figure 1.4 also shows other forms of *invariance* which are important for Computer Vision, and which CNNs are good at, simply by adding these examples to the training dataset, or through additional mechanisms.

Formally, the function f is said to be *invariant* to T if $\forall T \in \mathscr{T}, f(x) = f(T(x))$. That is, applying transformations to the input does not change the output. Furthermore, a function $f(x)$ is *equivariant* to $T \in \mathscr{T}$ if $\forall T \in \mathscr{T}, T(f(x)) = f(T(x))$. That is, applying any transformation to the input of the function has the same effect as applying that transformation to the output of the function. As a good example, CNNs employed for classification (outputing a label) are mainly looking for translation and scale invariance scale invariance, while CNNs employed for object detection (outputing a bounding box) are mainly looking for translation and scale equivariance. Furthermore, Figure 1.4 shows the main forms of invariance which Computer Vision algorithms aim to achieve. Although CNNs can naturally achieve these properties from data, there are other ways to "hard-code" them through more traditional approaches. For example, illumination invariance could be more or less handled through histogram pre-processing techniques. As another example that will be discussed later on, size(scale) invariance could be handled through a multi-resolution pyramid, although Neural Networks are able to learn more efficient mechanisms when the right *a prioris* are introduced, such as bounding box regression in this case, as will be discussed in detail in

Chapter 2 & Chapter 4. Furthermore, this type of textita prioris (or even *a posteriori* in the case of our proposed compression algorithm) can be employed to identify certain CNN properties, such as more suitable basis functions, and will be discussed in detail in Chapter 3.

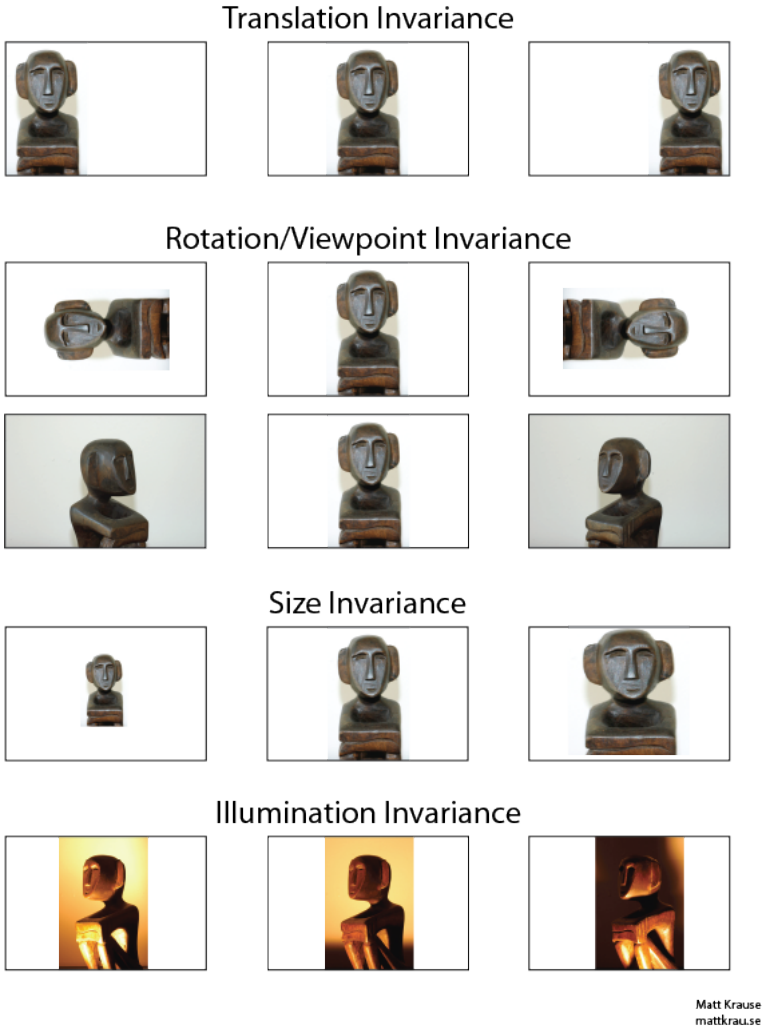


Figure 1.4 – Different forms of invariance necessary in Computer Vision tasks.

The CNN's architecture is then composed of a series of stacked layers, as is shown in Figure 1.5 below. Each of these different kinds of layer are over-viewed in subsequent sections.

1.1.3 Activation Layer

There are two main reasons to introduce activation layers :

— First and foremost, it is necessary to introduce non-linearities in-between two linear layers. If this is not the case, no matter how many layers are stacked, the final activation function of last layer is just a linear function of the input of the first layer, and could perfectly be achieved through a single linear layer.

— The second reason is biological in nature, and aims to model the firing threshold for individual neurons.

There are many different types of activation functions. Historically, the best known is the *sigmoid*, probably because it is bounded, and therefore, it acts as a regulator, and doesn't let some

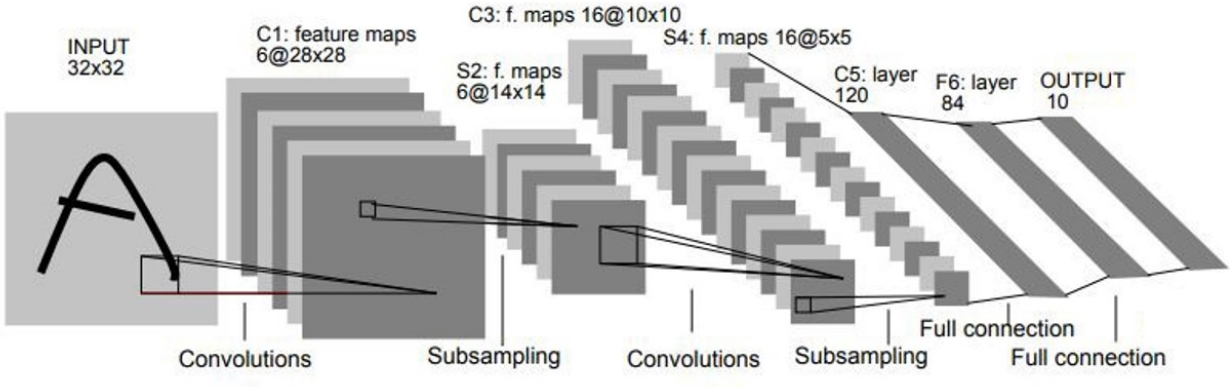


Figure 1.5 – Classical Convolutional Neural Network architecture as described in [11]. Image taken from [11]

parameters in the network blow up. It is also easily differentiable.

The Rectified Linear Unit(ReLU) has become very popular in the last few years. It computes the function $f(x) = max(0,x)$. In other words, we just throw away negative values. There are several pros and cons to using ReLUs :

— Compared to tanh/sigmoid neurons that involve expensive operations, the ReLU just thresholds values at zero and is therefore easily implemented in hardware. Therefore, in our digital context it becomes a clear choice.

— It was found to greatly accelerate the convergence of *stochastic gradient descent* compared to the sigmoid/tanh functions [10].

— On the negative side, ReLU units can "die" during training. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. A large portion of the network can be deactivated through the entire training dataset if the learning rate is set too high or the training batch is too small.

— ReLU has been shown to introduce parameter symmetries [105], leading to less efficient models. A solution to both these problems is to use a *leaky ReLU*, where instead of the function being zero when $x < 0$, we have a small negative slope (of 0.01, or so). However, this worsens the hardware implementation, since parameter count increases, and the dynamic range of the negative and positive elements of feature-maps becomes very inhomogeneous, impacting *quantization*.

1.1.4 Pooling/subsampling Layer

The idea is to replace the output at a certain location with a summary statistic (usually the maximum) of nearby outputs. The goal is to make the representation invariant to small translations of the input and therefore to detect the presence of a feature more than precisely locating it, as is shown in Figure 1.6.

Usually, pooling units have a stride higher than 1, since they represent a statistic for a neighborhood. Since the final effect on the feature map is just a subsampling that replaces a region by its maximum (for max pooling), this layer also improves the computational and memory efficiency (less inputs on the next layer). Notably, early layers in the network have higher resolution inputs

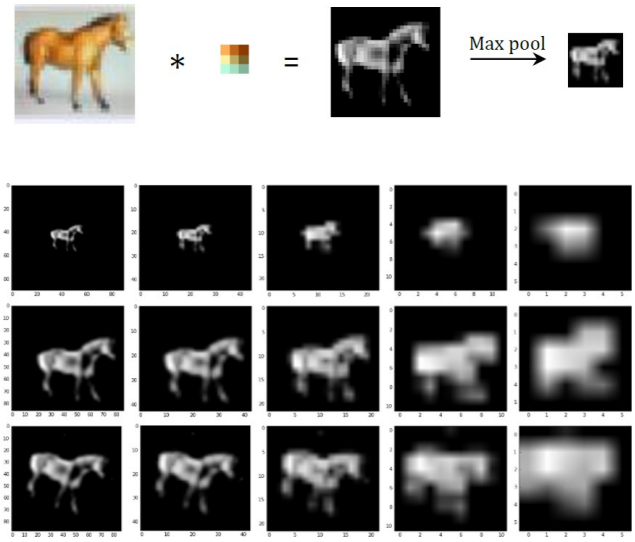


Figure 1.6 – The first row shows the effect of 2×2 *pooling on a feature map. The following rows show the effect of successive pooling on scaled, translated and rotated feature-maps. From these images we can see that pooling is an effective way to keep track of the presence of strongly activated features by a filter independently of their location, scale or orientation.*

and outputs, and adding pooling layers can help reduce memory requirements.

Max pooling makes the most sense in the vision context, because we expect that, when a pattern is found within an image, the region of the feature map corresponding to that pattern will return very high values. Keeping the maximum is therefore a good indicator of whether the filter has been matched in that region.

1.1.5 Fully Connected Layer

This is the classical layer of fully connected NNs, in which all neurons in a layer are connected to all neurons in the previous layer. It is usually employed for classification of the image from the features resulting from the convolutional and pooling layers, so it is placed at the end of the network, as is shown in Figure 1.5, and can be followed by more layers of the same type so that complex non-linear decision functions can be learned. Its function can also be achieved through 1x1 convolutions and recent works tend to completely remove fully connected layers.

1.1.6 Softmax Layer

This layer computes the softmax function (a.k.a normalized exponential), with the goal of converting a K-dimensional input vector $\mathbf{z}$ (which is the output of the last neural network layer) to another K-dimensional vector with values within the range $[0,1]$ and adding up to 1, so that each of the K components can be interpreted as the "probability" p_i of class i being the correct one :

$$\sigma(z_i) = p_i = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp(z_k)}.$$

1.1.7 Learning Features

The whole point with CNNs is to find "features" allowing to represent objects by learning them directly from data, instead of hand-crafting or manually selecting them. This is done by updating the weights (kernels in the image context) in an iterative manner and such that these updates help minimize an error measuring function. The adopted solution to this problem are two well known algorithms : gradient descent and error back-propagation, which will be described later.

Moreover, the goal of the neural network is to transform, or learn a representation of the input data, such that it becomes linearly separable, as is shown in Figure 1.7. This is also commonly referred to as feature extraction. The samples are then classified through linear *classifiers* (such as SVMs or fully connected layers) from the set of extracted features. Further details can be found in [12].

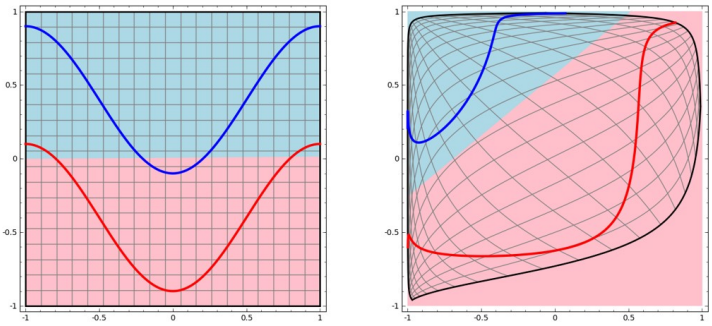


Figure 1.7 – Hierarchical feature construction by composition. Image from [12]

Indeed, a non-linear mapping may allow to linearly separate the data. More specifically, *Cover's theorem* states that a complex pattern-classification problem, cast in a high-dimensional space (nonlinearly), is more likely to be linearly separable than in a low-dimensional space. A clear example of this is shown in Figure 1.8, in which the data is not linearly separable in 2 dimensions, but, by adding a third dimension which measures the distance of the datapoints from the center, the data is now linearly separable. However, while a non-linear mapping may allow/ease data separability, it is not guaranteed that this will generalize to unseen data, and might result in *overfitting*, i.e. the classifier learns the appearance of specific instances and exceptions of our training dataset too well.

The above is a consequece of what is known as "the curse of dimensionality". Intuitively, if the amount of training data is fixed, then overfitting occurs when the problem is high-dimensional, since increasing the dimensionality introduces sparsity in the data. Furthermore, this sparseness not uniformly distributed. In order to ease overfitting, the amount of training data therefore needs to grow exponentially fast.

In practice, the performance of a classifier decreases when the dimensionality of the problem becomes too large, and data is limited. However, defining what "too large" means is not straightforward, and there is no principled way to do this in the case of Neural Networks. This is done in a very empirical way in practice and mainly depends on the amount of training data available, the complexity of the decision boundaries (task complexity), and the amount of useful a priori knowledge introduced into the classifier. This is a crucial issue in reaching our objective, since the complexity of the solution greatly determines its implementability in constrained hardware. Methods and heuristics allowing to address these issues in a more principled way, such that hardware implementation is eased will be discussed throughout this thesis.

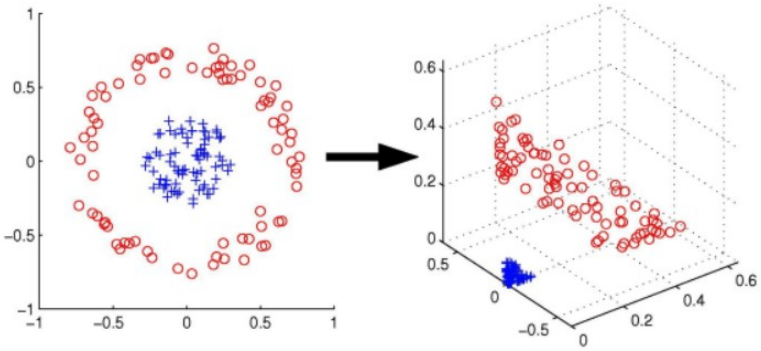


Figure 1.8 – A remapping of the data in higher dimensions allows for linear separability.

1.1.8 Feature interpretability

In the case of Convolutional Neural Networks, the hierarchical arrangement of convolutional layers allows to learn increasingly abstract types of features which are combined in a progressive fashion, allowing to learn abstract representations of objects. This is illustrated in the diagram on the left of Figure 1.9.

The right side of Figure 1.9 shows the actual features learned in each of the layers of a CNN. The first layer is visualized as a color image where simple patterns such as edges emerge, while latter layers are visualized through the work in [13] and seem to verify the hypothesis in which CNNs build higher-level, more abstract features as we go deeper into the network.

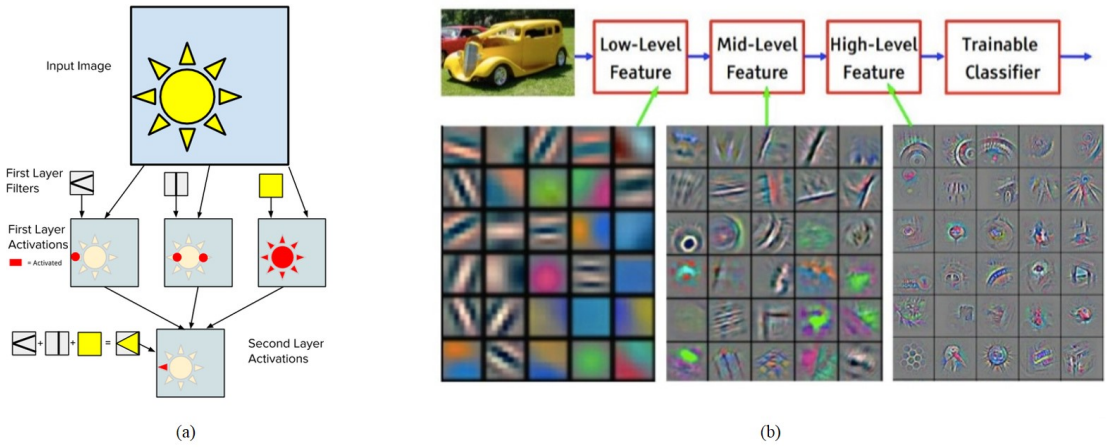


Figure 1.9 – (a) Schematic illustrating hierarchical feature construction by composition. (b) Visualization of strongest activation patterns for different layers of a state-of-the-art network. Features in the early layers strongly activate when edge or color patterns are presented, while later layers involve more abstract combinations of earlier features. Image from LeCun, Zeiler & Pete Warden

1.1.9 Loss Function

The metric used to measure the error defines the optimization problem we are trying to solve. For example adopting a Mean Squared Error (MSE) metric would be a good choice for a regres-

sion task, but by using other error functions, we can get additional information on how good the classification was. For example, the log-likelihood function, also known as cross-entropy function is a good choice for classification problems since it measures the accuracy of the classifier (vs. MSE just telling whether the classifier is correct or not).

Mathematically, this is expressed as :

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}),$$

where N denotes the total number of training samples, M is the total number of possible classes, y_{ij} is a binary indicator for the target class (e.g. if the first sample corresponds to the third class, then $y_{1,j} = 1 \iff j = 3, y_{1,j} = 0 \iff j \neq 3$), and p_{ij} is the model's probability of assigning label j to instance i, which corresponds to the softmax's output.

In the case of binary classification the previous formula simplifies to the following :

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_{i2} log(p_{i2}) + (1 - y_{i2}) log(1 - p_{i2}).$$

To illustrate the way this formula works, we can imagine the four extreme scenarios in a binary face classification scenario (face vs non-face) :

— Sample is a face ($y_{i2} = 1$) and the model's classification is very confident on the second class ($p_{ij} = [0.1\ 0.9]$).
In this scenario we should be pretty happy with the classification so the error for this sample shouldn't contribute much to the total cost function.
With log likelihood, this is in fact the case :
$1 \times log(0.9) + (1-1) \times log(1-0.9) \approx -0.0458$, so this sample doesn't contribute much to the total cost function.

— Sample is not a face ($y_{i2} = 0$) and the model's classification is sure it's on the first class ($p_i = [1.0\ 0.0]$).
In this scenario the classification is perfect so the error for this sample shouldn't contribute at all to the total cost function :
$0 \times log(1.0) + (1-0) \times log(1-0.0) = 0$, so this sample doesn't contribute at all to the total cost function.

— Sample is a face ($y_{i2} = 1$) and the model is confident it belongs to the first class ($p_i = [0.9\ 0.1]$).
In this scenario the classification is bad, so the error for this sample should contribute a lot to the total cost function : $(1) \times log(0.1) + (1-1) \times log(1-0.1) = -1$.

— Sample is not a face ($y_{i2} = 0$) and the model's classification is sure it's on the second class ($p_i = [0.05\ 0.95]$).
In this scenario the classification is completely wrong so the error for this sample should contribute as much as possible to the total cost function :
$0 \times log(0.95) + (1-0) \times log(1-0.95) \approx -1.301$.

1.1.10 Minimizing the loss function

The network's parameters (bias and weights) are randomly initialized and get iteratively modified so that we are able to reduce the cost function through these updates.

Gradient descent is a convex optimization method that allows us to find the fastest "route" to reach the (local or global) minimum of the function by moving in the opposite direction of the gradient (the gradient points directly uphill and its opposite directly downhill so we just have to move in the direction opposite to the gradient in order to reach a minimum). The parameter update rule is as follows :

$$w_{ij} = w_{ij} + \Delta w_{ij}, \text{ where } \Delta w_{ij} = -\eta \frac{\partial L}{\partial w_{ij}},$$

where :

— w_{ij} is the weight being updated.

— η is the learning rate.

— L is the cost function.

Figure 1.10 below depicts gradient descent for the simplest case, where the function to minimize $f(x) = \frac{1}{2}x^2$ is convex and depends on just one parameter.

In order to provide intuition on how gradient descent works, we take the case when the random initialization of x yields a negative value, i.e. $x < 0$. In this example, the minimum is located at $x = 0$ (as shown in Figure 1.10), so for $x < 0$, we must keep moving to the right, i.e. the opposite direction of the gradient. Indeed, in this case, since the derivative is $f'(x) = x \rightarrow -f'(x) > 0$. Furthermore, the gradient also keeps decreasing until it reaches the minimum (where it's well known that the derivative equals 0).

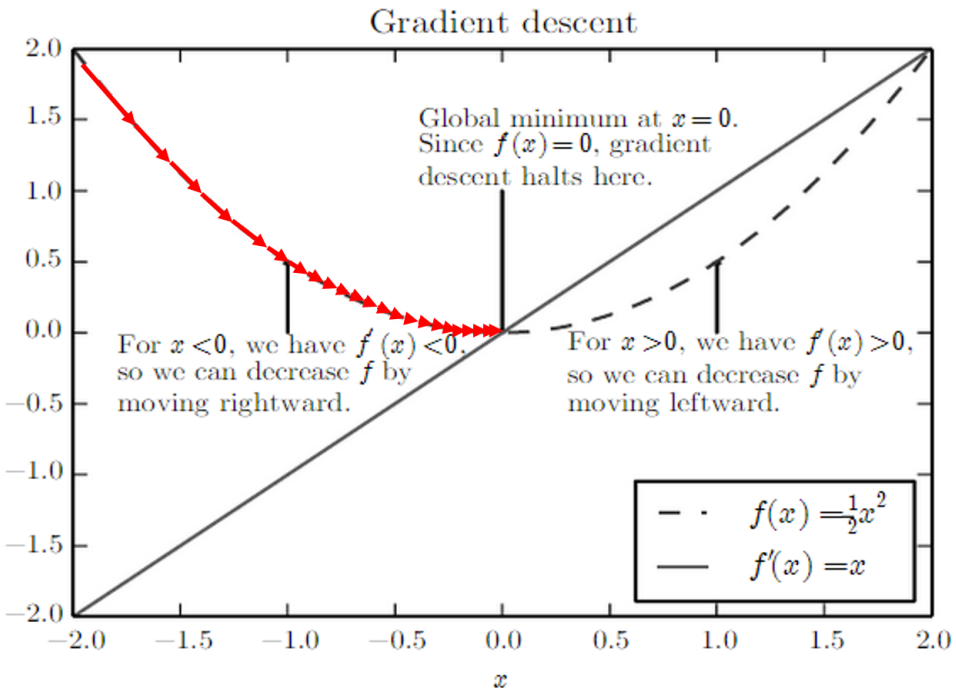


Figure 1.10 – Example where x is progressively updated by following the derivative of the function being minimized. [Image from Deep Learning Book [6]]

In the update rule there is also the learning rate term which basically determines how fast we go down the slope. In practice it's sometimes hard to find the right value for this parameter, but as a general rule, if the learning is too slow it should be increased and, if the cost function after the update oscillates too much, (overshooting in Figure 1.11 is a typical case of this) it should be decreased. The learning rate is also proportional to the batch size (number of samples used to compute the gradient).

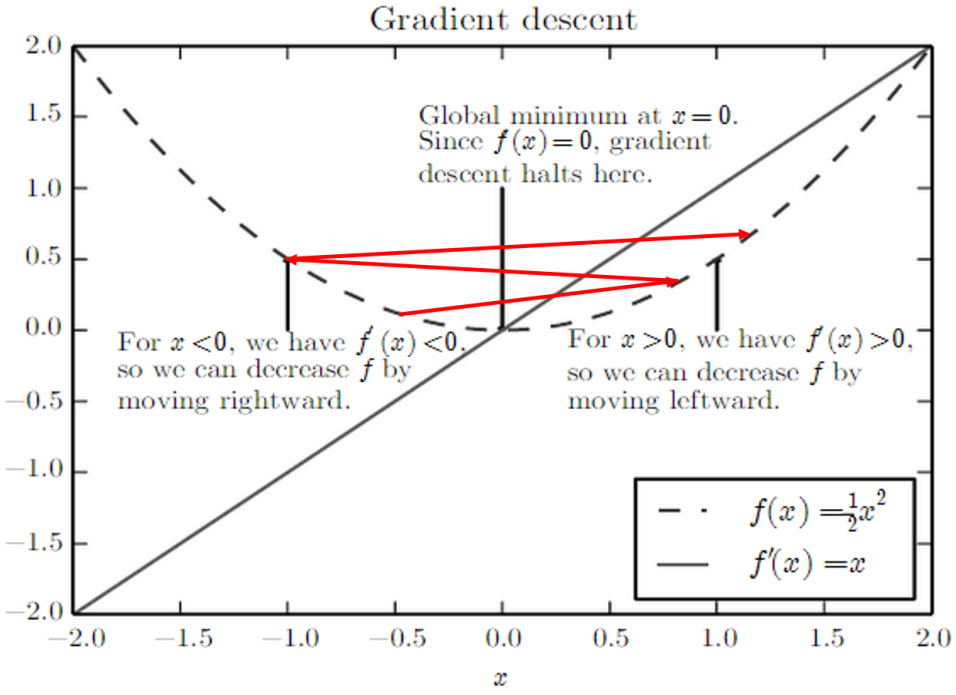


Figure 1.11 – Overshooting provoked by setting the learning rate too high. [Image from Deep Learning Book [6]]

Of course, in practice, unlike in the above example, cost functions are rarely convex, depend on many parameters (check where every partial derivative is zero) and it's hard to find the global minimum, so we often settle for non-optimal local minima. The example shows the perfect case, where the cost function has a unique minimum and the optimization problem is well-posed (unique, stable solution). Typically the topography of the cost function has a valley shape and the global optimum is not unique [93]. This induces a big part of this thesis' problematic, as will be discussed in later sections.

In fact, this used to be a problem in practice, and part of the reason why neural networks fell into disuse. However the availability of massive amounts of data and compute power in recent years has shown that, in practice the optimization problem is less significant and can be simplified by carefully bringing right a priori knowledge (i.e. right network architecture, clean datasets, convolution...) into the optimization problem.

1.1.11 Topology of the loss function

Mathematically, a CNN can be defined as a classifier between a set of n-dimensional attributes, and a set of M classes i.e. $\mathbf{CNN} : \mathbf{w} \in \mathbb{R}^n \longrightarrow C = \{c_1, ..., c_M\}$. The training consists in obtaining

the set of attributes $\mathbf{w}$ which fit a training set $T \subset \{I_1, c_1\}\{I_2, c_2\}...\{I_N, c_N\}$ where I_i are the images in our case, and C_i are the class labels (ground truths) for each of the N samples in the training set.

This fitting is done through the minimization of a loss function, in this case, the L2-norm :

$$L(\mathbf{w}) = \|\mathbf{CNN}(\mathbf{w}) - \mathbf{c}_{obs}\|_2,$$

where $\mathbf{c}_{obs} = \{c_1, ..., c_N\}$, and $\mathbf{CNN}(\mathbf{w})$ is a vector field whose components are $\{CNN_1(\mathbf{w}), ..., CNN_N(\mathbf{w})\}$, i.e. :

$$\mathbf{CNN} : \mathbb{R}^{\mathrm{n}} \longrightarrow \{\mathrm{c}_1, ..., \mathrm{c}_{\mathrm{N}}\} \in \mathbb{R}^{\mathrm{N}} \quad \{\mathrm{CNN}_1(\mathbf{w}), ..., \mathrm{CNN}_{\mathrm{N}}(\mathbf{w})\},$$

where $CNN_i(\mathbf{w})$ is the component function of **CNN** that predicts the i-th image of the training set. A regularization factor ε is usually also included in the loss function, such that a minimal norm (L1 here) solution is obtained :

$$L(\mathbf{w}) = \|\mathbf{CNN}(\mathbf{w}) - \mathbf{c}_{obs}\|_2 + \varepsilon^2 \|\mathbf{w}\|_1,$$

The set of observations $\mathbf{c}_{obs} = \{c_1, ..c_N\} \in \mathbb{R}^{\mathrm{N}}$ is an N-dimenisonal vector (contains N examples), and $\mathbf{w} \in \mathbb{R}^{\mathrm{n}}$ is an n-dimenisonal vector containing the parameters of the convolutional network. The goal is to minimize the following function :

$$\min_{\mathbf{w} \subset P \in \mathbb{R}^{\mathrm{n}}} L(\mathbf{w}).$$

Given an initial guess of a set of parameters, $\mathbf{w}_0$, it is possible to linearize the cost function through a Taylor expansion, i.e. :

$$L(\mathbf{w}) = L(\mathbf{w}_0) + \nabla L(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) + \mathrm{o}(\|\mathbf{w} - \mathbf{w}_0\|)$$

From this, it can be seen that the the loss function mainly depends on the term $\|\mathbf{CNN}(\mathbf{w}) - \mathbf{c}_{obs}\|_2$, which, when minimized, corresponds to the least squares solution of the non-linear system $\mathbf{CNN}(\mathbf{w}) = \mathbf{c}_{obs}$. As shown below, this problem has a topology influencing the cost function $L(\mathbf{w})$ both depending on the number of data (N, dimension of $\mathbf{c}_{obs}$), as well as the number of parameters (n, dimension of $\mathbf{w}$). To see this, it is possible to linearize $\mathbf{CNN}(\mathbf{w})$:

$$\mathbf{CNN}(\mathbf{w}) = \mathbf{CNN}(\mathbf{w}_0) + JCNN_{(\mathbf{w}_0)}(\mathbf{w} - \mathbf{w}_0) + \mathrm{o}(\|\mathbf{w} - \mathbf{w}_0\|),$$

where $JCNN_{(\mathbf{w}_0)}(\mathbf{w} - \mathbf{w}_0) \in M_{(N \times n)}(\mathbb{R})$ is the Jacobian matrix evaluated at point $\mathbf{w}_0$, i.e., $JCNN_{(\mathbf{w}_0)}(i, j) = \frac{\partial CNN_i}{\partial w_j}(\mathbf{w}_0)$. From the above equation, the cost function can then be expressed as :

$$L(\mathbf{w}) = \|\mathbf{CNN}(\mathbf{w}) - \mathbf{c}_{obs}\|_2 = \|(\mathbf{CNN}(\mathbf{w}_0) - \mathbf{c}_{obs}) + JCNN_{(\mathbf{w}_0)}(\mathbf{w} - \mathbf{w}_0) + \mathrm{o}(\|\mathbf{w} - \mathbf{w}_0\|)\|_2$$

Since $\mathbf{w}_0 \to \mathbf{w}$, the omicron, o, can be neglected, and :

$$min(L(\mathbf{w})) \Rightarrow min(\|(\mathbf{CNN}(\mathbf{w}_0) - \mathbf{c}_{obs}) + JCNN_{(\mathbf{w}_0)}(\mathbf{w} - \mathbf{w}_0)\|_2).$$

This corresponds to the least squares solution of the system

$$JCNN_{(\mathbf{w}_0)}(\mathbf{w} - \mathbf{w}_0) = (\mathbf{CNN}(\mathbf{w}_0) - \mathbf{c}_{obs}).$$

This clearly shows that, in any case, the solution to this system depends on the dimensions of the Jacobian $JCNN_{(\mathbf{w}_0)} \in M_{(N \times n)}(\mathbb{R})$, and therefore both the size of N(data) and n(parameters), as well as its rank, i.e. the degree of redundancy on both the set of parameters and data samples.

To understand why this problem admits multiple equivalent solutions, it is possible, for a set of parameters $\mathbf{w}$ to define an equivalence region M_{tol} in which the error is lower than a defined threshold ε_{tol}, i.e. $M_{tol} = \{\mathbf{w} : \|\mathbf{CNN}(\mathbf{w}) - \mathbf{c}_{obs}\|_2 < \varepsilon_{tol}\}$.

When gradient descent is applied to this system, the equivalence region implicitly becomes linearized in the following way : $L_{tol} = \{\mathbf{w} : \|JCNN_{(\mathbf{w}_0)}(\mathbf{w} - \mathbf{w}_0) - \mathbf{CNN}_{(\mathbf{w}_0)} + \mathbf{c}_{obs}\|_2 < \varepsilon_{tol}\}$.

The linearized equivalence region L_{tol} is an hyperquadric which is oriented by the matrix $JCNN^T_{(\mathbf{w}_0)} JCNN_{(\mathbf{w}_0)}$. This equivalence region is a flat valley whose size and shape depend on the rank of the Jacobian, the number of data, and the number of parameters. There are three main possible cases :

— N>n, with rank(JCNN)=n : purely overdetermined, in this case the hyperquadric is an ellipsoid whose largest axis is inversely protional to the lowest singular value of the Jacobian, and is oriented according to the eigenvectors of $JCNN^T_{(\mathbf{w}_0)} JCNN_{(\mathbf{w}_0)}$. In this case there is redundancy in the examples.

— n>N, with rank(JCNN)=N : purely underdetermined, in this case the hyperquadric is an infinite valley of equivalent solutions, and is oriented in the direction of the null space of the Jacobian.

— rank(JCNN)<min(N,n) : rank defficient, in this case there is redundancy both in the examples and parameters and there is also an infinite equivalence region. This is usually the case in practice, for which models are usually over-parametrized, and examples are redundant.

The degrees of freedom (underdeterminancy) corresponds to the number of parameters, n, minus the rank of the Jacobian. The conditioning of the system is defined as the largest singular value of the Jacobian, divided by the lowest non-zero singular value. When the number of parameters increase, the conditioning also increases since the lowest singular values become even lower. This makes the region of equivalence L_{tol} increase, as its axis are inversely proportional to the magnitude of the singular values.

Moreover we are interested in finding a region of overlap between M_{tol} and L_{tol} for which a solution is valid for both the non-linear, and linearized equivalence regions. In practice, this is verified at each iteration by evaluating the model through a forward pass in order to compute the loss, i.e. M_{tol}, and then performing gradient descent, i.e. L_{tol}. While the shape of L_{tol} greatly depends on the number of data and parameters, the shape of M_{tol} is bounded. In order to reach this overlap region, it is complicated to appropriately tune the learning rate to advance through the flat valley.

1.1.12 Training and test distributions

In machine learning, the minimization of the cost function implies a better performance when evaluating on the same dataset as in the training phase, i.e. we assume that training and test sets follow the same distributions. However, in practice, if the training set has a slightly different distribution than the test (unseen) set, it is possible that performance does not increase on the test set, even as the cost function is being minimized on the training set. For example, Figure 1.12 shows the importance of distribution consistency between training and test sets for deep learning models. This highlights the importance of a representative database, and the complexity of deploying machine learning models in the real world, where all variability is impossible to account for. These problems can be eased though engineering, a very good example of this being sensor fusion in self-driving cars. It also highlights the need for smarter algorithms, or need to introduce new concepts/mechanisms, approaching a more logical/causal reasoning.

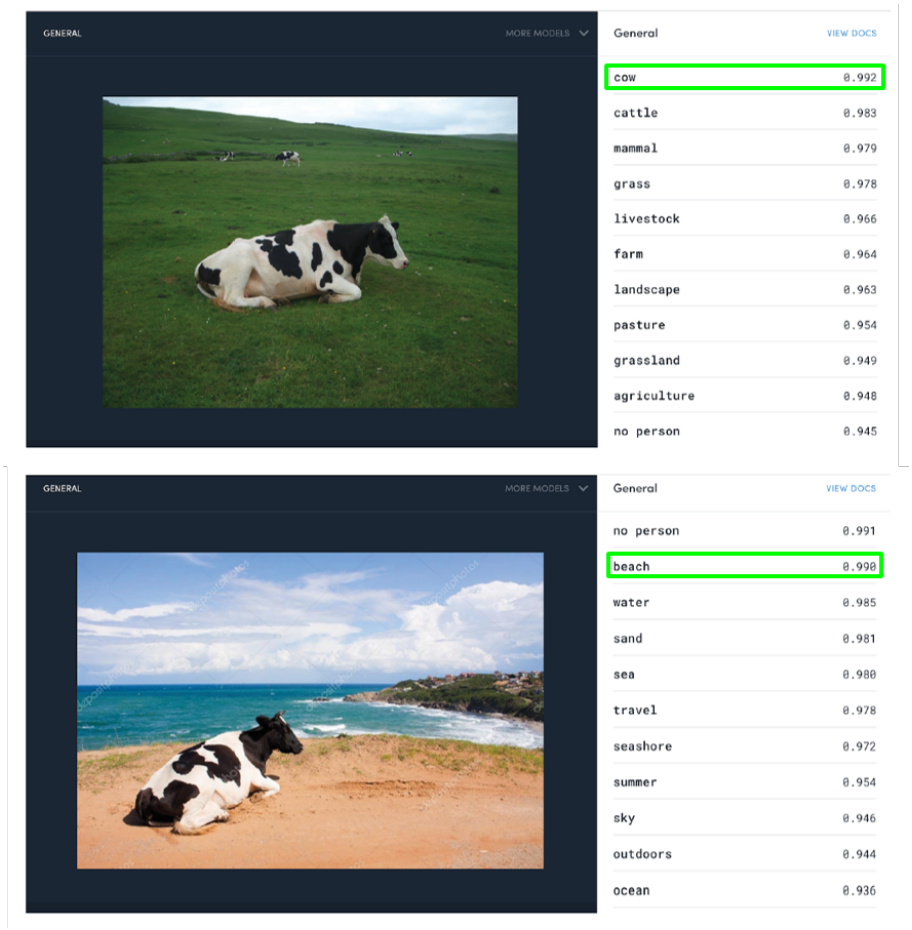


Figure 1.12 – Example of the effect of distribution missmatch between training and testing datasets.

1.1.13 Overfitting and Underfitting

Furthermore, even while assuming the same train and test distributions, it is very usually the case that train and test errors are not consistent. Moreover, the following cases can happen :

— Low training error (e.g. 1%) and high test error (e.g. 20%) : In statistics, this is known as a *high variance* model, i.e. a model that pays too much attention to data-points from the training set. This is generally known as *overfitting*, i.e. the model's complexity is high and the optimization of the loss function becomes an underdetermined problem (many equivalent solutions). Therefore, the solutions obtained on the training set are able to minimize the loss function, but the model is not able to generalize well to the testing set.

— High training error (e.g. 20%) and similarly high test error (e.g. 21%) : This model is said to have *low variance*, but *high bias*. This generally means that the model is too simple to capture the underlying patterns in the data, and is also known as *underfitting* .

— High training error (e.g. 20%)), but even lower test error (e.g. 30%) : This model is said to have *high variance* and *high bias*, it is simultaneously overfitting and underfitting.

— Low training error (e.g. 1%) and low test error (e.g. 2%) : This model is said to have low bias and low variance, and therefore it is performing well.

Formally, a predictive model requires adjusting a bias vs. variance trade-off. In the simplest case, we assume that the data we are trying to predict y can be explained from x through the function f, i.e. $y = f(x) + \varepsilon$, where we assume $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

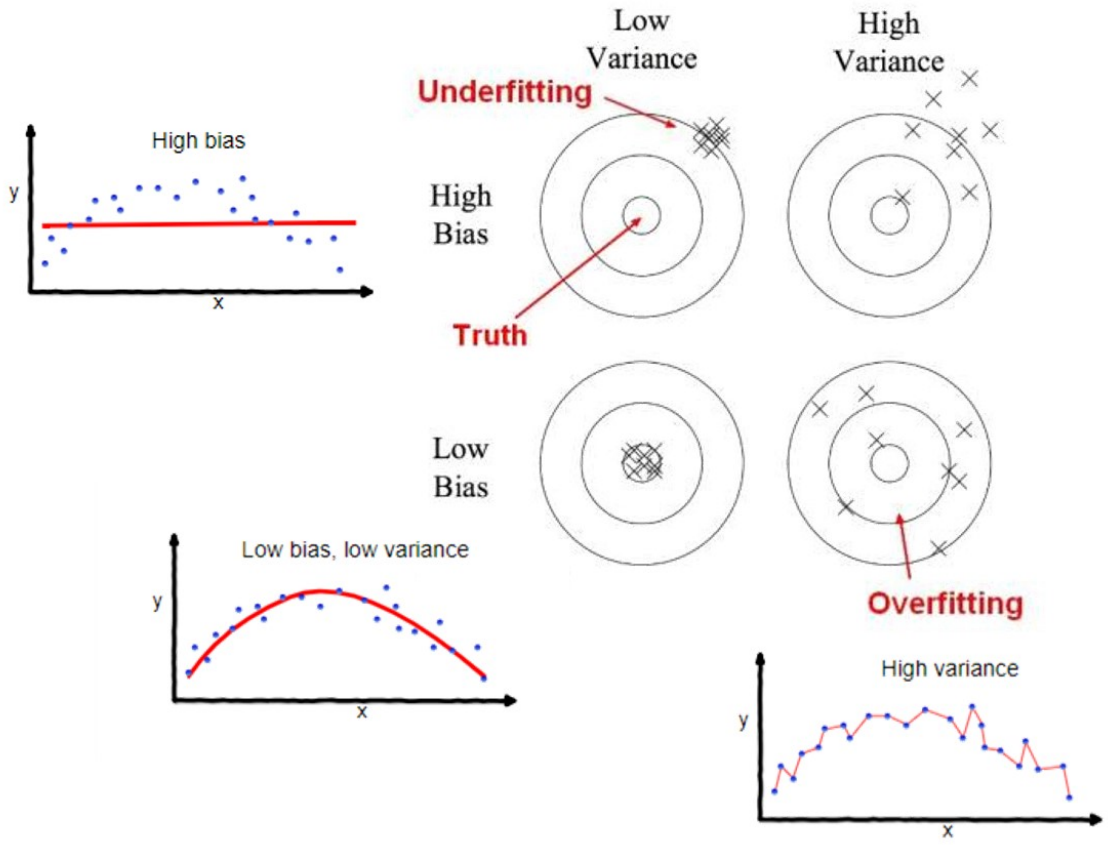


Figure 1.13 – Examples of possible bias/variance trade-offs. Image from [95].

For this, we propose a predictive model $\hat{f}(x)$, for which the expected error would be given by :

$$MSE : \mathbb{E}[(y-\hat{f})^2] = \mathbb{E}[(f+\varepsilon-\hat{f})^2] = \mathbb{E}[(f-\hat{f})^2 + 2(f-\hat{f})\varepsilon + \varepsilon^2] =$$
$$= \mathbb{E}[(f-\hat{f})^2] + 2\mathbb{E}[(f-\hat{f})\varepsilon] + \mathbb{E}[\varepsilon^2] = \mathbb{E}[(f-\hat{f})^2] + 2\mathbb{E}[(f-\hat{f})]\underbrace{\mathbb{E}[\varepsilon]}_{0} + \underbrace{\mathbb{E}[\varepsilon^2]}_{\sigma_\varepsilon^2} =$$
$$= \mathbb{E}[(f-\hat{f})^2] + \mathbb{E}[\varepsilon^2]$$

Furthermore, by introducing the term $\mathbb{E}[\hat{f}]$, the expression $\mathbb{E}[(f-\hat{f})^2]$ can be further developed to :

$$\mathbb{E}[(f-\hat{f})^2] = \mathbb{E}[((f-\mathbb{E}[\hat{f}]) - (\hat{f}-\mathbb{E}[\hat{f}]))^2] = \mathbb{E}[(f-\mathbb{E}[\hat{f}])^2 - 2(f-\mathbb{E}[\hat{f}]) =$$
$$= (\hat{f}-\mathbb{E}[\hat{f}]) + (\hat{f}-\mathbb{E}[\hat{f}])^2] = \mathbb{E}[(f-\mathbb{E}[\hat{f}])^2] + \mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])^2] - \underbrace{2\mathbb{E}[(f-\mathbb{E}[\hat{f}])(\hat{f}-\mathbb{E}[\hat{f}])]}_{2(f-\mathbb{E}[\hat{f}])\underbrace{\mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])]}_{\mathbb{E}[\hat{f}]-\mathbb{E}[\hat{f}]=0}}$$

Therefore, the MSE can be expressed as :

$$\mathbb{E}[(y-\hat{f})^2] = \underbrace{\mathbb{E}[(f-\mathbb{E}[\hat{f}])^2]}_{\text{model's bias}} + \underbrace{\mathbb{E}[(\hat{f}-\mathbb{E}[\hat{f}])^2]}_{\text{model's variance}} + \underbrace{\sigma_\varepsilon^2}_{\text{noise in data}}$$

However, with respect to modern CNNs, these trade-offs don't seem to hold too well, as there seems to also be a dependency on other factors. For example, a neural network's size can be increased in order to reduce bias, since this allows a large set of possible, equivalent solutions. However, regularization needs to be carefully tuned in order to avoid increasing variance at the same time.

By adding data, variance can also be reduced without affecting bias. As will be discussed in the rest of the Chapters from this thesis, selecting a model architecture that is well suited for a specific task also leads to simultaneously reduced bias and variance. Selecting such architecture, however, is far from trivial. Bringing a priori knowledge into the network, such as the importance of convolution also greatly helps reducing both bias and variance. Further interesting references on these topics include [95, 96].

1.1.14 Deep Learning vs traditional machine learning

It is also well known that deep learning models scale well with data, even on very complex tasks which are completely untargetable by traditional machine learning methods. Indeed, while traditional machine learning models reach an asymptotic performance, deep learning models are able to slightly increase performance by increasing both model size as well as data availability, as is shown in Figure 1.14.

However, this scenario is merely a caricature of the trend found in practice, since redundancy in both data and model's parameters play an important role in "effective" complexity, and are both difficult to estimate. Furthermore, this scenario still greatly reminds of over-fitting, and a lot of care must be taken when trading-off computational complexity and performance, especially in an embedded context. Multiple techniques allowing to fix these trade-offs will be discussed, as well as proposed in later chapters of this thesis. It will also be shown that indeed, at similar computational complexity, Neural Networks both perform and scale better than traditional algorithms.

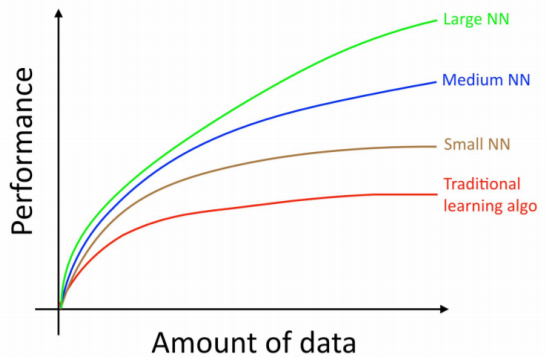


Figure 1.14 – Deep learning scalability through data availability and model size. Image from [96].

1.1.15 Deep Learning Frameworks

One of the strengths of frameworks such as Tensorflow [97] is their ability to decompose a Neural Network in a graph by composition of simpler, differentiable operators. This allows the automatic differentiation of a custom user-defined graph, abstracting the gradient descent and backpropagation phases from the user. This is a very powerful concept, which is illustrated below through a simple example in Figure 1.15.

The function $f(x,y,z) = (x+y)z$, while being simple enough to be differentiated directly, could also be decomposed into two simpler differetiable functions : $q = x+y$ and $f = qz$. Moreover, during backpropagation, we mainly care about how much does each of the input variables "influence" the output of the function, i.e. the derivative of the output with respect to the input variables. In our

example, this would correspond to the following : $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$. By having decomposed the function f into simpler components, i.e. $f = qp$, differentiation becomes trivial through the *chain rule* :

— $\frac{\partial f}{\partial x} = \frac{\partial (qz)}{\partial x} = z\frac{\partial q}{\partial x} + q\frac{\partial z}{\partial x} = z\frac{\partial q}{\partial x} + q \cdot 0,$
$\frac{\partial q}{\partial x} = \frac{\partial (x+y)}{\partial x} = \frac{\partial x}{\partial x} + \frac{\partial y}{\partial x} = 1 + 0 = 1 \implies \frac{\partial f}{\partial x} = z \cdot 1 = z.$

— $\frac{\partial f}{\partial y} = \frac{\partial (qz)}{\partial y} = z\frac{\partial q}{\partial y} + q\frac{\partial z}{\partial y} = z\frac{\partial (x+y)}{\partial y} + q \cdot 0 = z(\frac{\partial x}{\partial y} + \frac{\partial y}{\partial y}) = z(0+1) = z.$

— $\frac{\partial f}{\partial z} = \frac{\partial (qz)}{\partial z} = z\frac{\partial q}{\partial z} + q\frac{\partial z}{\partial z} = z \cdot (0+0) + q \cdot 1 = q.$

Intuitively, backpropagation can be thought of as gates communicating to each other (through the gradient signal) whether they want their outputs to increase or decrease (and how strongly), so as to make the final output value higher. In the example of Figure 1.15, we can see, that in order for the output to increase (from -12 to 5), we would need to modify both x and y with an equal contribution (gradient), which is proportional to z, while z has a contribution which is proportional to q. Modifying the parameters in the direction of the gradient leads to maximizing the function, while modifying them in the opposite direction leads to minimizing the function.

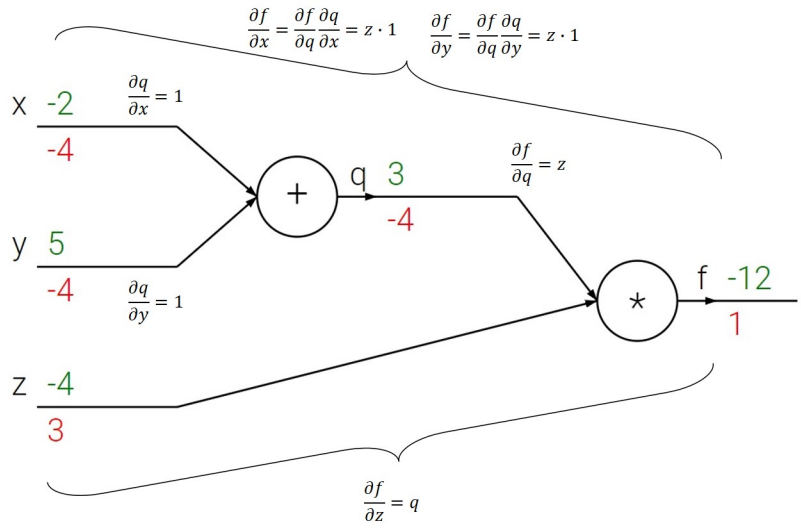


Figure 1.15 – Backpropagation of errors in a simple graph. Inputs and outputs to each node are in green, while gradients are in red. Image from CS231, Stanford.

1.1.16 Deep Learning in practice

Figure 1.16 summarizes the typical life-cycle of a deep learning solution. Nowadays, multiple frameworks (such as Tensorflow, Caffe, Torch or Theano) allow training neural networks. This mainly requires fixing CNN architecture choices and an image database. Once the training has converged to a satisfactory solution, the model can be frozen and used for inference.

Since deep learning methods scale well with data, this induces a virtuous cycle in which, once the model has been deployed, the user-base increases, also increasing data collection and therefore improving algorithm performance and attracting more users. In our case, further compression and trade-off possibilities must be explored in order to fit the solution into resource constrained embedded systems.

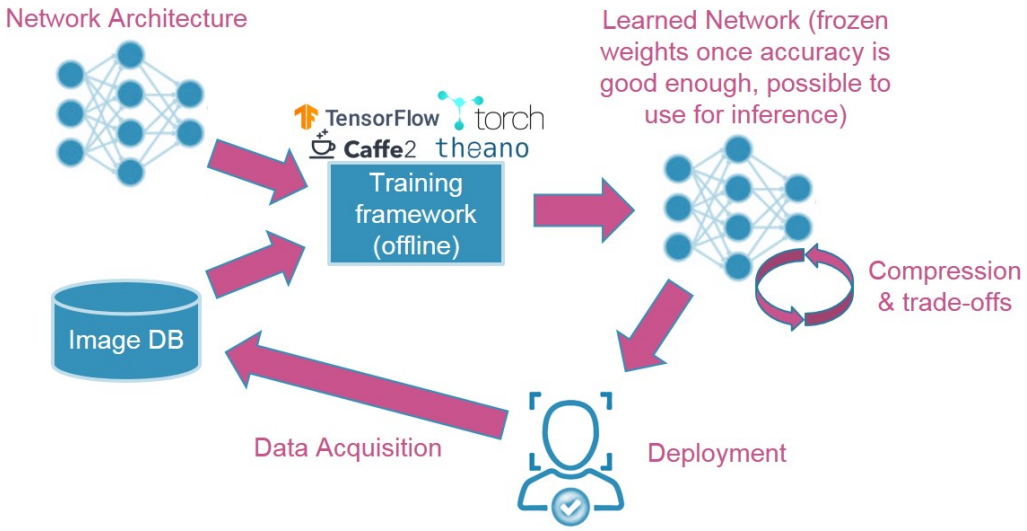


Figure 1.16 – Typical life cycle of machine learning solutions.

1.2 CNN architecture comparison

CNN architectures have "relatively" evolved over recent years. The main trend and focus being on accuracy in the early years (2012-2016) and focusing progressively more on efficiency in recent years (2017-2019). Different architectural choices have been progressively introduced throughout these years, culminating in automatic architecture search methods in actuality. The work by [7] and more recently [8] provide a good comparison of different CNN architectures in terms of accuracy and computational complexity, as is shown in Figure 1.17. Concretely, Figure 1.17.a compares different architectures' accuracies with respect to the number of operations, and number of parameters(size of the bubble). Figure 1.17.b compares network accuracy to parameter density, i.e. accuracy divided by number of parameters.

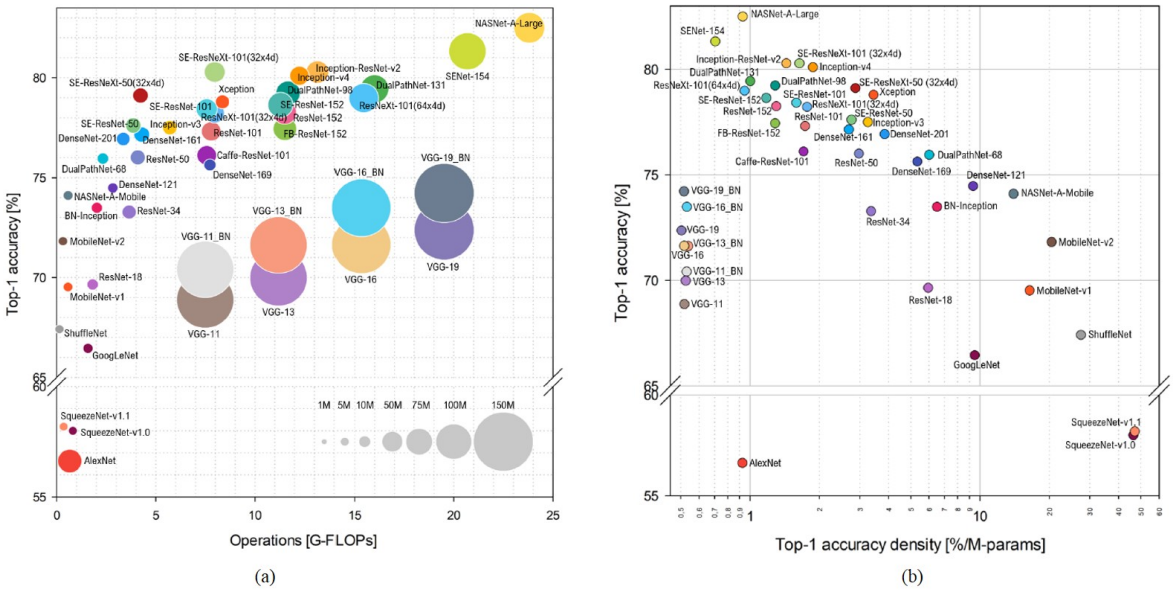


Figure 1.17 – Computational Complexity of state-of-the-art CNN solutions. Image from [8].

The following list provides a chronological overview of the most popular architectures and their main insights and contributions :

1.2.1 AlexNet

This is the first CNN-based winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9] with an 8-layer convolutional network as described in [10]. The main insight in this paper is that, by increasing the complexity of a CNN, it was able to scale to more difficult Computer Vision problems better than traditional algorithms. Code parallelization on multiple GPUs also played an essential role in being able to train such complex model($\sim$60 million parameters) on a large scale dataset($\sim$14 million images).

1.2.2 ZFNet

Described in [13], keeps the same architecture as AlexNet but improves accuracy by tweaking hyper-parameters such as kernel sizes after observation and analysis of the AlexNet's weights and feature maps.

1.2.3 VGG

Described in [14], achieved better results than AlexNet by greatly increasing computational complexity. It also proposed to use minimal filter sizes (3x3), since the combination of layers of these can emulate larger receptive fields. For example, two consecutive 3x3 layers effectively form a 5x5 receptive field. Thanks to the non-linearity between the two 3x3 layers, the representation is richer than a single linear 5x5 kernel.

1.2.4 Network in Network

Described in [15], it introduces the concept of "micronetworks" by adding multi-layer-perceptron within each conv layer in order to compute more abstract features within each of the layers.

1.2.5 GoogleNet/Incenption

Described in [16], introduces convolutional layers which contain multiple different kernel sizes. For example, a single layer may contain a 5x5, 3x3 and 1x1 receptive field simultaneously, for which the feature maps are concatenated at the end of the layer.

1.2.6 HighWay Networks

Described in [17], they introduce the idea of shortcut connections between convolutional layers for which the shortcut gating function is learnable at training time. This allows the network to automatically learn which features in previous layers could be useful. This is very similar in spirit to LSTMs (long short term memory networks).

1.2.7 ResNets

Described in [18], they are a special case of HighWay Networks in which feature map outputs are passed in an uniform way between two consecutive layers. They allow parameter reuse between layers and have been shown to simplify the loss function's landscape, as well as attenuating the vanishing gradient problem for deeper networks. However, intermediary memory requirements increase since we need to store two consecutive feature maps at inference time.

1.2.8 Wide ResNet

Described in [19], argues that depth is not as important as residuals and proposes to increase width (number of kernels and channels) instead of depth(layers).

1.2.9 ResNext

Described in [20], adds different kernel types within a residual block, much like Inception.

1.2.10 DenseNet

Described in [21], it proposes to densely connect all layers in the network with each other. However, intermediary data requirements greatly increase, since in order to evaluate a single layer in the network, the feature maps from all preceding layers in the network must be stored too.

1.2.11 Squeeze-and-Extraction

Described in [22], SENet adds a feature recalibration module to networks such as ResNets, allowing them to learn to re-weight feature maps instead of simply copying them between layers.

1.2.12 FractalNet

Described in [23], argues that residual connections might not be a necessary factor and proposes an architecture with multiple branches within each layer which can independently be desactivated during learning and obtaining similar accuracy as ResNets for similar parameter budgets. Since all branches are fixed and used at inference time, implemenation might also be more efficient than ResNets.

1.2.13 MobileNets

Described in [24], MobileNets employ Depthwise separable convolutions, i.e. each channel is convolved separtely with a filter, and the resulting feature-maps are then recombined in a learnable fashion, instead of a simple summation across channels, as is done in traditional CNNs. This allows to reach good trade-offs between model size and complexity.

1.2.14 ShuffleNets

Described in [25], they propose using pointwise group convolutions to reduce computation complexity of 1×1 convolutions as well as a channel suffle operation in order to overcome some side effects brought by group convolutions.

1.2.15 SqueezeNet

Described in [26], SqueezeNet employed an automatic model search approach by exploring different possible combinations of 3x3 and 1x1 filters on each layer. It achieved AlexNet level accuracy with a more efficient model, to which multiple compression methods were then applied.

1.2.16 NASNet

Described in [27], NasNet is a "Controller" network that learns to design a good network architecture through reinforcement learning. Moreover, a CNN architecture is sampled from search space. The architecture is trained, generating a "reward" related to accuracy. Controller parameters are updated such that the likelihood of a good architecture being sampled is increased and likelihood of bad architecture is decreased. However, this requires huge computational costs : [30] report the cost of running a NAS method to discover a state-of-the-art Natural Language Processing (NLP) network as generating x315 more Carbon Dioxide emissions than the equivalent for a single-passenger round-trip between New York and San Francisco by plane. Although scientific discoveries might seem more important than transportation, this shows that current trends are not sustainable and priority needs to be placed on efficient use of Deep Learning, more so since multiple works show that there is room for improvement.

1.3 Realistic CNN applications (as of now, 2019)

Deep learning has proven to scale better to complex problems and greatly benefits from data. This has effectively led to the replacement of more traditional machine learning techniques, and changing the paradigm from hand-crafting features to learning features automatically from data through neural networks.

Deep learning finds applications mainly in the domains of Computer Vision (images) and Natural Language Processing (audio). Although there are many interesting emerging applications of deep learning, most are completely out of the scope of embedded systems and therefore, in this section, we focus on showcasing some applications that could potentially fit an embedded implementation.

1.3.1 Wake-up systems

As was outlined in previous sections, CNNs are not reliable enough for most real-world scenarios. This is due to multiple reasons. While they have a slight, generalization ability, they mostly work by overfitting datasets, and therefore have a hard time with corner-case scenarios. Moreover, they need to see every possible scenario at training time in order to predict it correctly at inference. In real life, however the corner-cases are common and diverse. This restrains to applicative scenarios in which a slight error rate (such as a false positive) could be tolerated.

This issue is further magnified when accuracy/complexity trade-offs need to be made, as overviewed in Chapter 3. In this sense, CNNs can be used as a cascade of complexity increasing classifiers, in which interesting frames are detected on an edge device, which are then transmitted to a more complex algorithm which makes the final decision on the server-side. This allows to greatly lower computational complexity of CV solutions, while still providing high and reliable accuracy rates.

1.3.2 Image Classification

This is the task in which CNNs excel, with some examples being the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Most breakthroughs have come from this area of computer vision, as it is also the basis for other more complex tasks such as object detection, image understanding, etc.

— Face Identification : It's interest in an applicative context is clear. The work in [31] proposes state-of-the-art visual face recognition through a relatively lightweight CNN ($\sim$ 1 million parameters).

1.3.3 Object Detection

Object detection is one of the most useful applications of Computer Vision, and will be throughly covered in the next chapters of this thesis. Broadly speaking, general object detection is out of reach for embedded systems, however, by reducing the variability in the objects that need to be detected, some embedded applications could be considered, for example :

— Crowd Counting : the work in [32] presents a lightweight network allowing to estimate crowd counts through head location, achieving state-of-the-art results with 0.86M parameters. The network could further be compressed and implemented on resource constrained systems.

— Person detection : the work in [33] presents a shallow CNN conatining 3 convolutional layers (each with 32 5x5 filters $\sim$ 100 thousand parameters) which is able to outperform both HOG and haar-based pedestrian detection algorithms.

1.3.4 Depth Estimation

The work in [35] proposes an efficient and lightweight encoder-decoder network architecture allowing to perform depth estimation from a single RGB image. Network pruning is further employed to reduce computational complexity and latency, achieving close to state-of-the-art accuracy on the NYU Depth v2 dataset with $\sim$ 1 million parameters.

The interest on monocular depth estimation is mainly due to the relatively low cost and size of monocular cameras, however, even works such as this one are too computationally complex and results provide merely an estimation, as is shown in Figure 1.18. However, network complexity could possibly be lowered and results improved by merging this approach with other types of sensors such as RGB-Z (color and depth simultaneously).

Figure 1.18 – Depth estimation by a CNN through a single RGB image.

1.3.5 Image Segmentation

The work in [34] employs an asymmetric encoderdecoder architecture for the task of real-time semantic segmentation. The encoder adopts a ResNet as backbone network, while an attention pyramid network (APN) is employed in the decoder to further lighten the entire network complexity. This work achieves state-of-the-art results in terms of speed and accuracy trade-off on CityScapes dataset while employing $\sim$ 1 million parameters. This network could possibly also be merged with the approach described in 1.3.4, as well as RGB-Z sensors in order to both reduce computational complexity of the network, and also increase accuracy.

1.3.6 Gesture Recognition

Neural networks can also be employed to process outputs coming from a wide range of sensors such as accelerometers, time of flight... For example, Figure 1.19 shows an example of a commercial drone which is controlled by processing the data coming out of an accelerometer installed on the writst band. Given the robustness of neural networks to noise and their ability to generalize to unseen scenarios, it is likely that this processing is carried out by a neural network. The work in [36] employs a recurrent neural network with a total of 1.7 million parameters allowing to recognize sequences of gestures.

Figure 1.19 – Toy drone controlled through wrist accelerometer.Image from aliexpress.

1.3.7 Image reconstruction/extrapolation/compression/noise reduction

— Super Resolution : the work in [37] proposes a network consisting of only 20K parameters and 1.43 GMAC allowing to perform image super resolution.

— ISP bypass : works such as [38] propose to gain room for CNN processing by removing traditional parts of imagers such as the image signal processor (ISP). The problem being the differences in the distribution between the images generated by an imager with and without an ISP. To avoid a data capture stage in order to retrain the network, they propose slight hardware modifications by using logarithmic pixels and binned readouts (for gamma compression and pixel binning respectively) in order to approximate the data generating distribution of the ISP processed images.

— Compressed Sensing : The work in [39] shows the efficiency of Generative Adversial Networks (GANs) in a context of Compressed Sensing, and proves that fewer measurements (in the order of x5-10 times less) are required in order to reconstruct a signal with respect to a Lasso methods with a DCT (discrete cosine transform) basis. While the basic idea of compressed sensing is to reduce the number of required measurements by assuming sparsity inside a well-chosen basis, the paper doesn't rely on sparsity and instead, since generative models learn the distribution of the data, they suppose that measured vectors lie near the range of a generative model.

— Image compression : Several CNN-based solutions have been proposed, such as [40], in which an encoder CNN is used to learns optimal compact representation of an input image, encoded through a classical algorithm such as JPEG, while a decoder CNN is trained to reconstruct the compact decoded image. While promising, the interest of such methods is not clear since they introduce heavy computational complexity for marginal subjective quality gains. GANs have also been used in the state-of-the-art to massively compress images. Indeed works such as [41] prove that, although classical algorithms such as JPEG achieve better PSNR and SSIM with respect to GANs, the latter are more bitrate efficient, since they still achieve better performance on tasks such as semantic segmentation with such images, and display high potential for further improvements.

1.3.8 Natural Language Processing

This task is simpler and easier to fit into embedded system's constraints than image processing, since it deals with 1D signals (audio) instead of 2D signals (images). An example application that has been explored on microcontrollers is that of key-word spotting in order to wake-up a more complex system. For example, the work in [42] shows that an artificial neural network based keyword spotting system outperforms classical Hidden Markov Models (in both noisy and clean conditions) and also leads to a simpler implementation, reduced runtime computation, and smaller model size ($\sim$ 2 million parameters).

More recently, the work in [43] shows that employing a CNN instead of ANN as in [42] further boosts performance by $\sim$ 10% while requiring a similar number of parameters. Number of operations notably increases, but is still reasonable with $\sim$ 6 million operations per inference on the smallest model.

1.3.9 Image Understanding

This task consists in generating a description of the scene for any given image. This requires a combination of both a CNN for visual feature extraction and an LSTM for natural language processing and therefore it is an even computationally heavier task.

1.3.10 Autonomous Navigation

Self-driving cars are a clear example of the success of Deep Learning. It is also a clear example of the degree of computational complexity and complications of deploying such systems in the real world. For instance, all self-driving companies greatly struggle to achieve full autonomy due to the complexity of simulating all possible scenarios (which is the basis of Deep Learning), and therefore it is not possible to predict when such systems will be available. Furthermore, in the early models, GPU servers were implemented inside the car's trunk in order to keep up with the computational complexity demanded by such algorithms, as shown in Figure 1.20. Chip-makers have seen the opportunity for self-driving SoCs, such as the NVIDIA DRIVE AGX Pegasus, which is able to deliver 320 TOPs at 500 Watts TDP, or the TESLA FSD chip, delivering 140 TOPs at 72 Watts.

Although this shows the degree of complexity of current state-of-the-art Deep Learning solutions, it is also possible to find simpler and much more efficient autonomous systems which

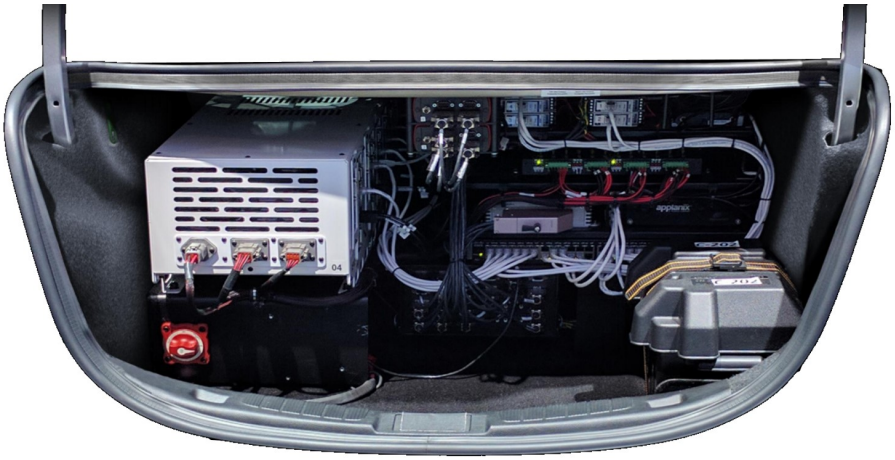


Figure 1.20 – GPU server inside a self-driving car's trunk.

employ Deep Learning, such as the work in [44], which implements a prototype of a CNN-based autonomous navigation system on a nano-drone. The complexity of such task is much lower than that of autonomous cars, but it still represents a real world applicative scenario that can be carried out in very constrained embedded systems, as the one shown in Figure 1.21.

In order to do this, an 8-layer ResNet consisting of 320K parameters, and described in [45] is trained on autonomous vehicle datasets in order to predict both steering angle and object collision probabilities. The precision of the original network is reduced from 32-bit floating point down to a 16-bit fixed point one employing 4 bits for the integer part (including sign) and 12 bits for the fractional part of both activations and weights ($\sim 10^{12}$ precision). The network is retrained with quantization in the loop.

Since the deployed camera is an ultra low power & low resolution (QVGA) grayscale camera, the dataset is extended with images directly coming from the camera (while the original dataset features high-resolution RGB images).

The CNN model is then mapped on a GAP8 multi-core microcontroller, an ASMP equivalent based on the open-source PULP platform and featuring 9 RISC-V cores, featuring an on-chip 64 kByte L1 TCDM memory along with a 512 kByte L2 SRAM as well as an off-chip 8 MByte DRAM / 16 MByte Flash. The GAP8 is then ported on an open-source CrazyFlie 2.0 nano-quadrotor.

The power consumption of all the electronics aboard the original drone amounts to 277 mW leaving 7.3 W for the motors. In addition to that, introducing the PULP-Shield, increases the peak power envelope by 64 mW (at 6fps) or 284 mW (at 18fps).

Figure 1.21 – Crazyflie with a GAP8 microcontroller on top of it for autonomous navigation. Image.

1.4 CNN hardware accelerators

As illustrated in section 0.5.2, the main bottleneck for processing energy consumption in CNNs is in the memory access. In order to reduce the energy cost of data movement, several levels of local memory hierarchy with different energy access cost need to be introduced.

Accelerators need to then be designed to support specialized processing dataflows that leverage this memory hierarchy. Concretely, the dataflow needs to efficiently manage data movements between the different levels of memory and decide the order in which the data is processed. Since there is no randomness in the processing of CNNs, it is possible to design a fixed dataflow that can adapt to the CNN shapes and sizes and optimize for the best energy efficiency.

The optimized dataflow therefore needs to minimize accesses from the more energy consuming levels of the memory hierarchy. Since large memories that can store a significant amount of data consume more energy than smaller memories, every time a piece of data is moved from an expensive level to a lower cost level, we want to reuse that piece of data as much as possible to minimize subsequent accesses to the expensive levels.

The challenge, however, is that the storage capacity of these low cost memories is limited. This clearly shows the cruciality of CNN complexity reduction techniques, as well as the need to explore different dataflows that maximize reuse under concrete applicative constraints.

The three main forms of data reuse in CNNs are explained below :

— Filter reuse : multiple input feature maps are processed at once, i.e. the same filter weights are used multiple times across input features maps.

— Feature map reuse : multiple filters are applied to the same feature map, i.e. the input feature map activations are used multiple times across filters.

— Convolutional reuse : the same input feature map and filter weights are used within a given channel, just in different combinations for different partial sums.

The goal is therefore to leverage these three main types of data reuse through different kinds of dataflows :

— Output stationary : illustrated in Figure 1.22, it is designed to minimize the energy consumption of reading and writing the partial sums by keeping the accumulation of partial sums for

the same output activation value local in the register file(RF). One common implementation for this is to stream the input activations across the processing element(PE) array and broadcast the weight to all PEs in the array.

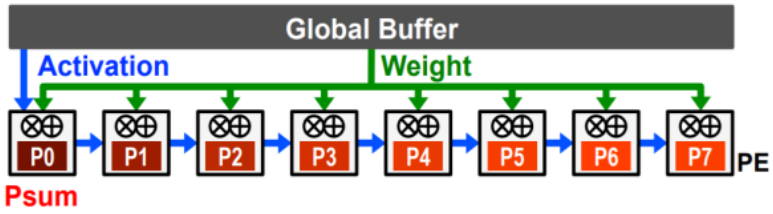


Figure 1.22 – Output stationary dataflow. Image from [46].

An example of this dataflow is the work in [47]. It was one of the first accelerators proposed for the embedded domain. The authors also try to reduce energy consumption by reducing memory access and therefore remove the DRAM memory. The size of the SRAM is only around 288 KB. The architecture is compossed of three SRAM memories ; two are used to store input and output feature maps, while the other is used to store filter weights. The Neural Functional Unit (NFU) is a 2D array of PEs, where each PE is able to carry out a single MAC per cycle.

— Weight stationary : illustrated in Figure 1.23, it is designed to minimize the energy consumption of reading filter weights. Each weight is read into the register file (RF) of each processing engine (PE) and stays there for further accesses. The dataflow runs as many MACs that use the same weight as possible while the weight is present in the RF, effectively maximizing filter weight reuse. An example of this dataflow is the work in [48], in which a comparison with [47] is also done.

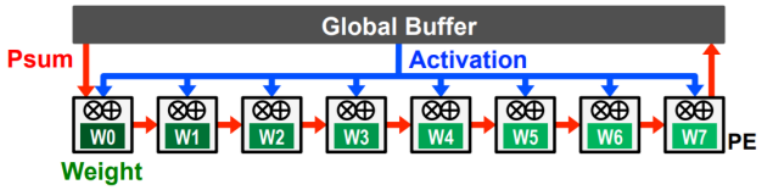


Figure 1.23 – Weight stationary dataflow. Image from [46].

— Row stationary : illustrated in Figure 1.24, it is designed to maximize the reuse and accumulation at the RF level for all types of data (weights, pixels, partial sums). In this dataflow, a row of filters and activations are loaded on each PE , as illustrated in Figure 1.24.

The data in the global buffer is accessible by all PEs through a multicast system. When data is read from the global buffer, the multicast allows to send it to a subset of selected PEs through a tag allowing to identify each PE. Some control logic allows to select relevant data and manage partial sums accumulation between layers of the network.

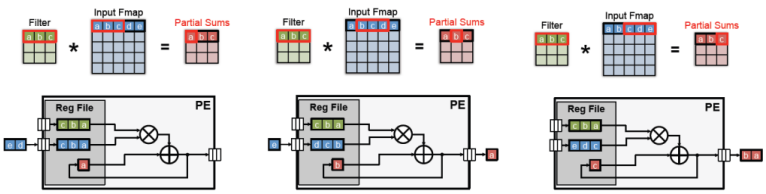


Figure 1.24 – 1 dimensional row stationary dataflow. Image from [46].

The work in [49] shows that this is an optimal dataflow for overall energy efficiency, by measuring a x1.4-2.5 energy reduction with respect to other dataflows. However, each PE

requires larger internal memory and control logic required for scheduling also becomes more complex. Furthermore, the authors employ a « zero-gating » mechanism for energy efficiency, which allows to exploit sparsity by skipping a MAC operation when either a weight or activation equals zero. The resulting sparse data is then encoded through a run length coding (RLC) mechanism for which the first 5 bits allow to encode a maximum of $2^5 = 32$ consecutive zeros and the following 16 bits encode the next non-zero value.

A follow-up work [50] tries to better adapt to the wide variety of emerging CNN architectures, since these have increasingly more irregular shapes and therefore not a single type of dataflow is best in all cases. For example, Weight stationary is the most efficient with a lot of input channels, while Output Stationary works best for large feature maps. The authors try to leverage this by modifying the proposed Row Stationary dataflow by adding a "network on chip" (NOC) router to better distribute PE workloads by trading-off data bandwidth and reusability.

— No local reuse : illustrated in Figure 1.25, it is designed to improve efficiency in terms of area instead of energy. No local storage is allocated to the PE and instead all that area is allocated to the global buffer to increase its capacity. This results in the worst energy efficiency of all mentioned techniques.

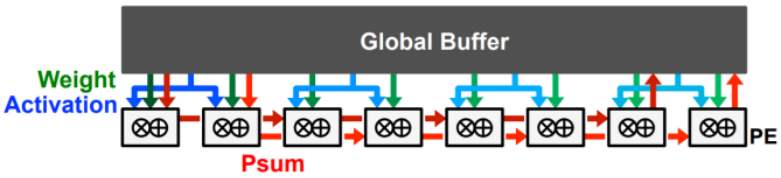


Figure 1.25 – Dataflow without any data reuse at PE level. Image from [46].

1.5 Neuromorphic computing

This is another promising (biologically inspired) concept in the field, in which the brain's functions are emulated in the analog domain. The approach is different from digital Artificial Neural Networks and employs the concept of Spiking Neural Networks (SNN), which are a more event-based type of processing, the main interest being orders of magnitude better energy efficiency as compared to classical Von Neuman computer architectures, as well as the ability to scale beyond Moore's Law.

Indeed, ANNs are mostly non-linear continuous function approximators that operate on a common clock cycle, whereas biological neurons compute with asynchronous spikes that signal the occurrence of some characteristic event through action potentials. Binary neural networks (later overviewed in section 3.1.0.1) share the binary and sparse communication scheme with SNNs. However, such networks are typically executed in a synchronized manner as opposed to the event-driven (asynchronous) nature of SNNs. The work in [51] does a good work in reviewing the main concepts in SNNs and comparing them to their ANN counterpart. Furthermore, the work in [52] provides a theoretical framework allowing to convert classical Neural Networks to Spiking Neural Networks by integrating activations during execution time at inference.

2

Methodology to adapt the computational complexity of CNN-based object detection for efficient inference in an applicative use-case

In this chapter we start by illustrating the main drivers and levers of computational complexity in CNNs when applying them to the specific task of object detection. First, we provide a quick overview of the state-of-the-art in CNN-based object detection. We then outline our approach and explain how it allows us to more flexibly tune computational complexity of the solution for an applicative use-case scenario with respect to existing approaches. The method is then illustrated and validated in Chapter 4 for a face detection applicative use-case.

Sommaire

0.5.1 Motivation 1
0.5.2 Problem formalization 3
0.5.3 Contributions and thesis outline 6

2.1 Computational Complexity of object detection

The task of *object detection* consists in jointly identifying the type and location of an object inside an image, as illustrated in Figure 2.1. The main challenges in such task involve the wide variety in the aspect of objects in the real world, as well as an extensive *search space* , i.e. searching for variable-sized objects inside high-resolution images.

Figure 2.1 – Goal of object detection algorithms.

As explained previously (sections 0.5.2 & 1.2), just being able to fit the *classifier*'s parameters inside *on-chip memory* is already a very challenging task. Furthermore, the number of operations required for object detection can also rapidly increase and become a burden. This is because, the main goal of object detection, is to achieve both translation and scale equivariance. I.e. if the object in the scene is scaled or translated, the CNN's output should undergo the same transformation, as was explained in section 1.1.2.

Indeed, since classifiers are usually trained at a fixed input size and location, the main mechanism to achieve these equivariance properties is to perform a *sliding window* search through an *image pyramid* , i.e. scanning all positions of the input image across multiple resolutions, as pictured in Figure 2.2.

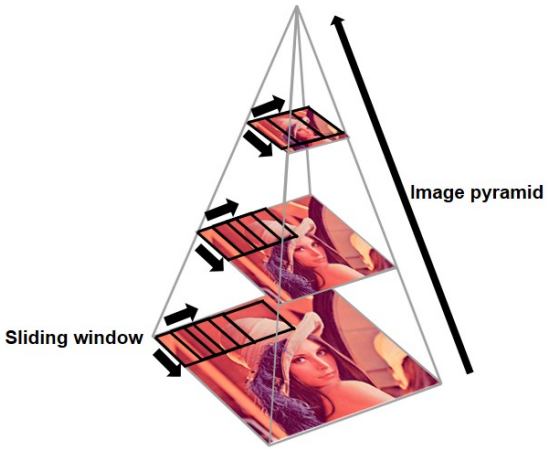


Figure 2.2 – Sliding window search approach for object detection across multiple resolutions.

Concretely, in order to find matches between any image region and the patterns detected by a N×N fixed-input classifier, we would need to match every N×N region of the image against the classifier, while simultaneously applying it at the right scale too, as pictured in Figure 2.3. This can rapidly increase the search space, in turn greatly increasing computational complexity of object detection.

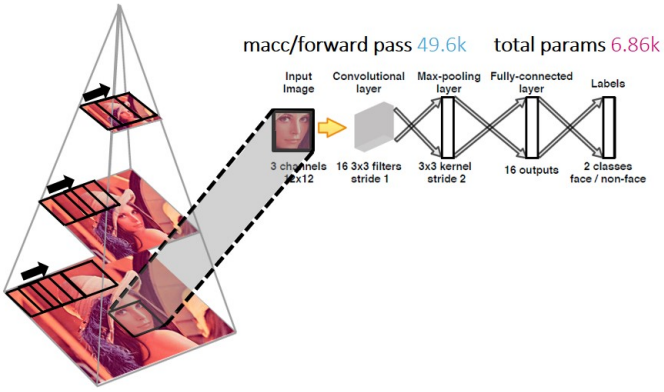


Figure 2.3 – Sliding window across multiple resolutions for object detection.

As an example, Figure 2.4 shows that, densely scanning a QVGA(320x240) input image in search of 12x12 faces -even through a minimal 16-filter network (which requires 49.6 KMAC per forward pass, as shown in Figure 2.2)- would take around 3.8 GMAC, while bigger faces such as 40x40 in QVGA would take approximately 249M MAC. All intermediary sizes would also need to be taken into account, as is shown in Figure 2.4. This clearly shows that even minimal CNN architectures can get quite expensive when performing object detection in a classical way and therefore, it is necessary to explore more efficient solutions. In this sense, allowing the CNN to learn scale invariance properties can greatly reduce the computational complexity of the object detector, as will be discussed later in section 2.3.2.1.

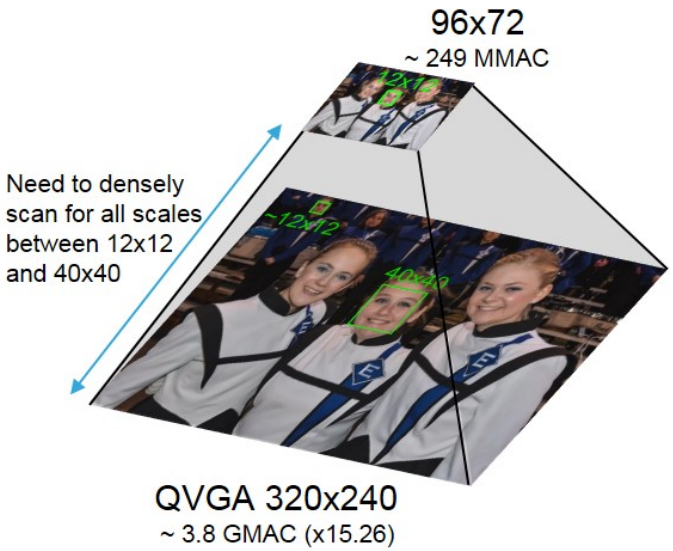


Figure 2.4 – Computational cost of classical object detection through minimalistic neural network. Searching for 12x12 faces in the original image would take 3.8G macc and 1.3M values of intermediary data (grayscale). By reducing the image size by a factor of x3.3 (96x72), the 40x40 face is now 12x12 and it takes 249M MAC to find (x15.26 less).

2.2 State of the art in object detection

The aforementioned issue becomes even more critical when a more complex CNN is used for classification. For instance, AlexNet [10] requires 2.5G operations per 227x227 window and therefore it becomes unfeasible to scan all possible positions of a high-resolution image across different scales with this network, even outside of the embedded domain.

2.2.1 RCNN

In this scope, RCNN [53] first generates around 2000 *region proposals* (i.e. bounding box proposals) through hand-crafted selective search methods, which indicate regions of the input image likely to contain an object of interest. Those proposed regions are then cropped and transformed to a fixed-size 227x227 image (AlexNet input size).

AlexNet is then run through each crop in order to extract features. These features are then classified through a Support Vector Machine (SVM) in order to obtain the class prediction. Finally, multiple class-specific bounding box regressors are also trained to refine the bounding box proposals from the extracted features. These regressed bounding boxes then define the regions of the image with detected objects. Figure 2.5 depicts this execution flow.

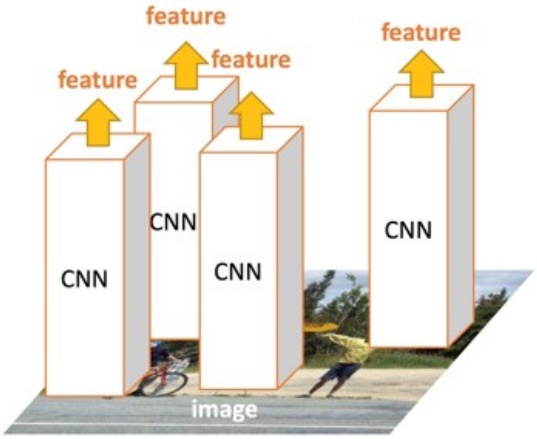


Figure 2.5 – Main idea for RCNN. Image taken from [53]

2.2.2 SPPNet

SPPNet [54] proposed to compute feature extraction just once per image, as shown in Figure 2.6. The main insight is that convolution produces feature maps where both the strength of the response to a filter, as well as spatial location are preserved, and therefore localization can also be achieved from feature maps. A multi-resolution pooling pyramid is also performed on the resulting feature maps. These are then scanned through a selective search method, allowing to generate RoIs. These RoIs are then passed on to fully-connected layers which are responsible for both classification and bounding box regression.

The same approach is then followed by Fast RCNN [55], with the main difference being the removal of the feature map multi-resolution pooling (replaced by single scale pooling), and

allowing the network to be trainable in an end-to-end fashion.

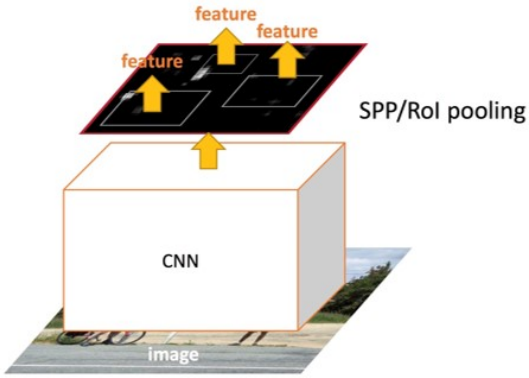


Figure 2.6 – Main idea for SPPNet/Fast RCNN. Image taken from [54]

2.2.3 Faster RCNN

This work [56] is an incremental improvement to the work in [55]. The main novelty is the replacement of the hand-crafted selective search algorithm by a neural network based region proposal, as shown in Figure 2.7.

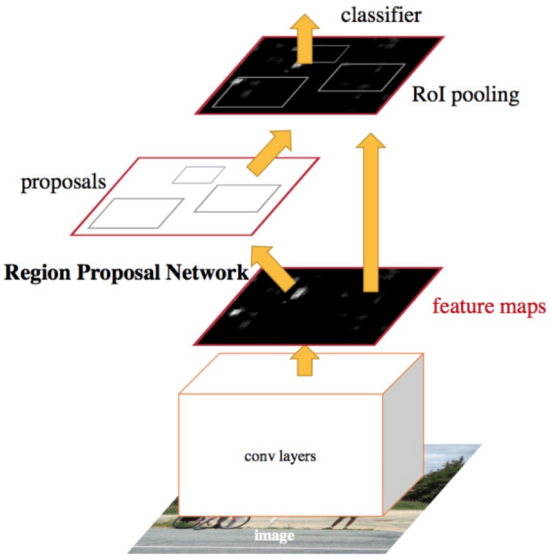


Figure 2.7 – Execution flow of Faster RCNN. Image taken from [56].

2.2.4 YOLO & SSD

The previously reviewed methods are commonly refered to as *two stage detectors*, since they comprise a region of interest proposal stage, followed by a classification stage.

On the other hand, single stage detectors methods such as YOLO [57] and SSD [58], skip the region proposal stage and run detection directly over a dense sampling of possible locations (e.g. prior boxes). In the literature, single stage detectors are commonly believed to be faster and simpler for object detection, while slightly lowering detection performance. This is

wrong, since the proposal stage can much more flexibly be tunned to trade-off computational complexity and detection performance, as will be shown in Chapter 4.

YOLO [57] proposes to analyze information from the entire image at once, instead of classifying image regions separately. For this, YOLO divides the input image into a grid where each grid cell contains a set of different sized, pre-defined *prior boxes*. Each of these prior boxes are parametrized by their coordinates (x,y,w,h), an object probability score, as well as an object class score. These coordinates are then regressed from the globality of the output feature map, which is why the network is said to globaly reason about the scene. This is as opposed to region proposal methods, which classify regions of the image independently. Since there aren't any non-differentiable operations (such as cropping proposals), the method is also trainable in an end-to-end fashion, while two-stage detectors are usually trained in different stages.

2.3 Proposed methodology to optimize CNN object detector complexity for applicative use-case

The works previously reviewed all involve very similar and interesting ideas. However, they are mostly targeted to a more general kind of solution. That is, the *backbone classification network* (e.g. VGG, ResNet, etc...) is generally trained on ImageNet [9], and performance is measured on datasets such as MSCOCO [59] and PASCAL VOC [60].

While these type of tasks are the ultimate goal of object detection, the proposed works are far too complex and completely out of reach for the embedded context. For example, the processing required to generate the results shown for the image in Figure 2.1, which is based on a YOLO approach, amounts to 62.938.253.312 floating point operations for a single VGA(640x480) frame. Furthermore, model size amounts to 194 MBytes ($\sim$ 50M parameters, 32-bit), while the layer with the most intermediary data amounts to 12.938.240 values (also coded in 32-bit).

Therefore, it is very clear that a Neural Network-based solution must be tailored and optimized to a concrete use-case scenario in order to fit the strict constraints of the embedded domain.

Furthermore, as will be shown through the rest of this Chapter, this is a very iterative process, during which a lot of different stages come into play. These stages also have strong dependencies, and need to be co-adapted in order to reach an efficient solution.

For this, an initial specification of the use-case first needs to be provided, in order to study its feasibility. Then, a preliminary solution needs to be proposed in order to approximately dimension the solution. Further trade-offs will then need to be thoroughly studied in order to tailor an efficient solution to the applicative use-case. This whole process will be overviewed in a general way in this chapter, and will also then be demonstrated in Chapter 4 for a concrete face detection use-case.

In this chapter, we aim at providing a principled methodology for designing and trading-off computational complexity of CNN-based object detectors on concrete applicative use-cases, allowing to adapt the solution to an embedded system's constraints. Then, in Chapter 4, we demonstrate the methodology on an applicative use-case for consumer face detection.

Moreover, the method consists of a cascaded approach in which first, a compact network coarsely scans the input image and generates region proposals, which are then further pro-

cessed by a second, more complex classifier network, as shown in Figure 2.8. Computational complexity is therefore greatly reduced, since there is no need to apply a complex classification network across all positions and scales of the input image. These CNNs are then further compressed through generic methods that allow to properly adapt computational complexity to the application's specifications. An outline of the method is now provided, further details can be found after this section.

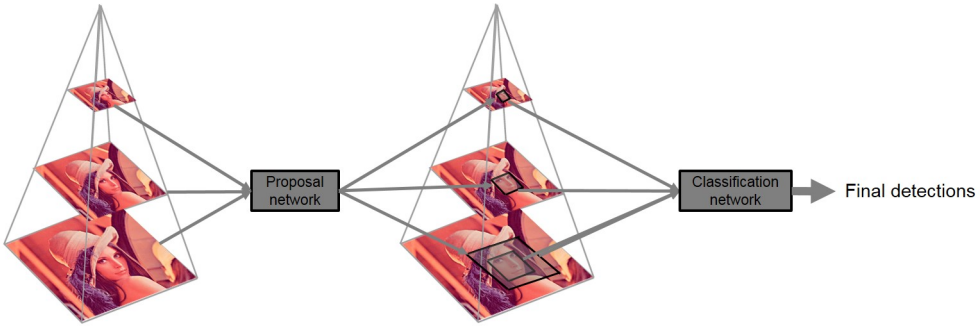


Figure 2.8 – Proposal and classification stages.

First, a *Proposal Network* (PN) is trained by heavily relying on *bounding box regression* i.e. linear regressors trained to predict bounding box coordinates from the CNN's feature maps, with respect to a ground truth bounding box. This allows to gain stronger scale and location equivariance, and therefore also allows to greatly simplify the search space, effectively reducing the number of operations and memory footprint. The proposal network therefore needs to be optimized to successfully identify interesting regions, while letting through an as low as possible average number of proposals.

Proposals then need to be accurately classified by a more complex network, a proposal classification network. This classification network is both trained to recognize objects, as well as correcting the proposal network's outputs.

Since accurate classification through neural networks requires more parameters than proposal generation, computational complexity vs. accuracy must be thoroughly studied for this network. Therefore, applying CNN compression methods on the classification network is of great interest.

Concretely, we propose a compression pipeline consisting of a combination of methods. First, our PCA compression method [111] (explained in detail in Chapter 3) allows to optimally trade-off both the number of parameters and operations of a learned neural network for detection performance. Then, the PCA basic filters can further be transformed into the frequency domain in order to perform convolution through the minimal filtering algorithm described in [89], as will be explained in Chapter 3. Finally, quantization as described in [61] is applied to all nodes in the network. The details, as well as complimentarity of such compression methods, are later discussed in Chapter 3.

Once the computational complexity and accuracy of the optimized networks have been tuned to fit the applicative specifications (in terms of number of parameters, maximum intermediary data, number of operations, TPR and FPR), the search space then needs to be defined (in terms of stride and number of re-scaling levels). The main dimensioning factors being the size of objects to be detected, the difficulty of such (in terms of variety of appearance, poses, occlusions etc...), as well as required detection rates.

Once all these design considerations have been taken into account with respect to an applica-

tive use-case scenario, the networks can be frozen to achieve an efficient inference solution. The solution then needs to be evaluated and benchmarked on a dataset which is representative of the use-case, as is done in Chapter 4 in the case of face detection for consumer applications. Finally, the solution needs to be evaluated on hardware, as is done in Chapter 5.

The diagram in Figure 2.9 provides a summarized overview of the main stages allowing to tune and leverage computational complexity of the solution. Each of these stages will now be explained in their respective sections.

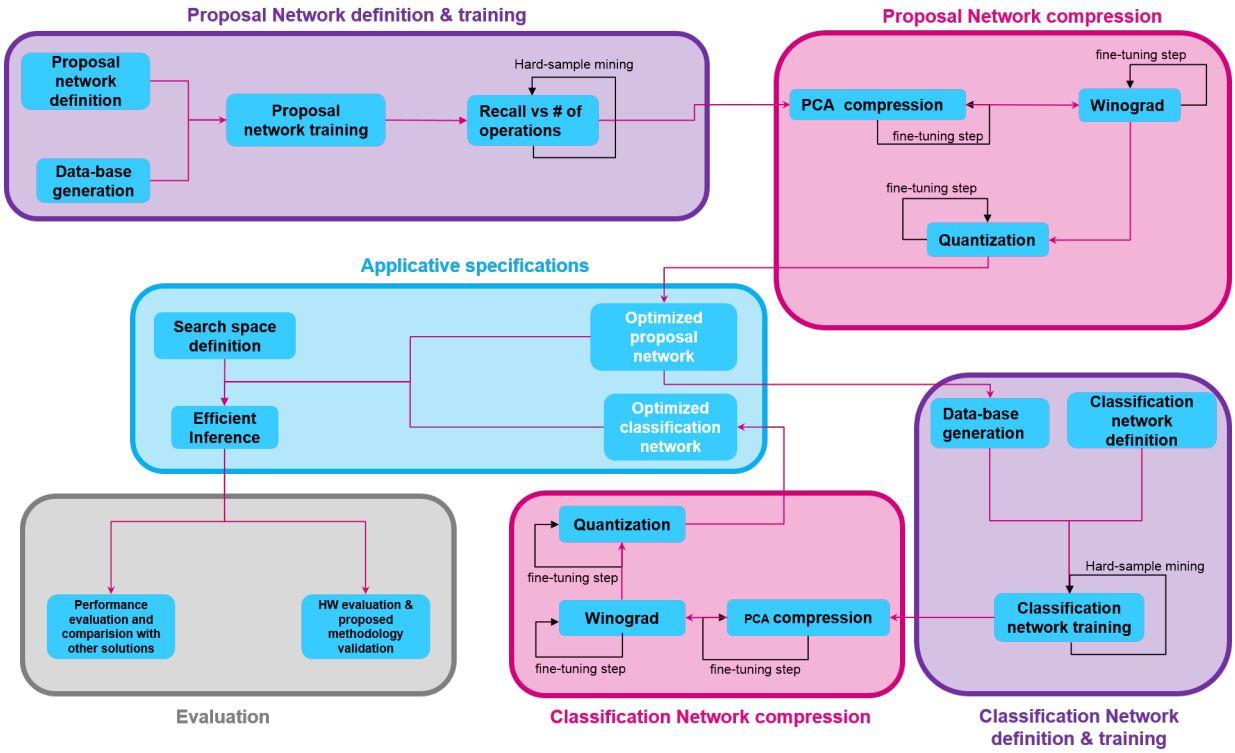


Figure 2.9 – Global overview of the proposed workflow to flexibly optimize CNN-based object detector complexity on a clearly defined applicative use-case.

Throughout the rest of this chapter, each of the different blocks from Figure 2.9 are described in their respective subsection. At the same time, each block is also subdivided into multiple sub-stages, for which the following elements are provided :

— A description of the overall goal of the stage is first provided.

— The main performance metrics that indicate the variables that need to be monitored when defining the corresponding stage of the solution in order to achieve the desired targets.

— The inputs that need to be defined by the user for this stage.

— The outputs resulting from this stage, which is usually an input to the following stage, indicated by the arrows in the diagram.

— The action to be carried-out during this stage.

— The hyper-parameters that need to be investigated by the user in order to achieve different results.

— The parameters defined by the user, mainly according to the applicative use-case specifications.

2.3.1 Applicative use-case specifications

Proper definition of an applicative use-case is crucial when dimensioning a detection algorithm to embedded systems. Moreover, the main interest in face detection in a consumer applicative context, is the ability to process the detection region afterwards, in order to obtain additional information, for example identity.

This greatly reduces the task's complexity, since it allows to exclude occluded, out-of-focus and, most importantly, smaller faces. This task complexity reduction, in turn, also allows to reduce the network's computational complexity, allowing to better fit the embedded system's constraints.

Moreover, once the applicative context has been thoroughly defined, the networks can go on to the learning phase. Compression algorithms exploiting certain properties of CNNs (as will be thoroughly discussed in Chapter 3), can then be employed to further reduce the solution's computational complexity, and ease HW implementation.

A summary of the main stages in this phase are shown in Figure 2.10, and detailed in the subsections following immediately below.

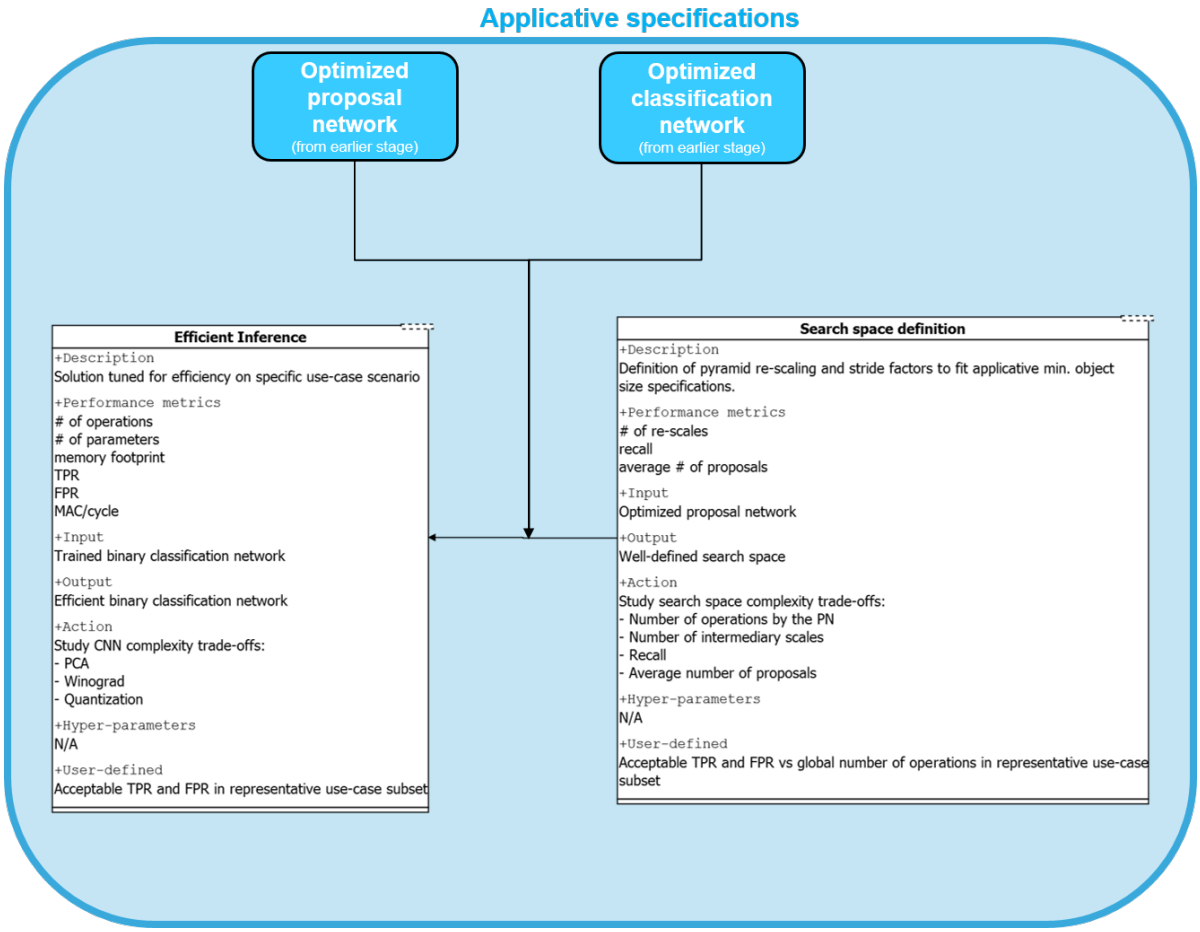


Figure 2.10 – Detailed applicative specification method.

2.3.1.1 Search Space definition

Face size is a crucial dimensioning factor, since the search space increases as objects get further away from the sensor, as Figure 2.11 shows. This translates into an increase of the number of positions to be analyzed and therefore requires higher computational budgets. Furthermore, regression plays a fundamental role in reducing the number of pyramid scales required during this search, as will be shown in Chapter 4.

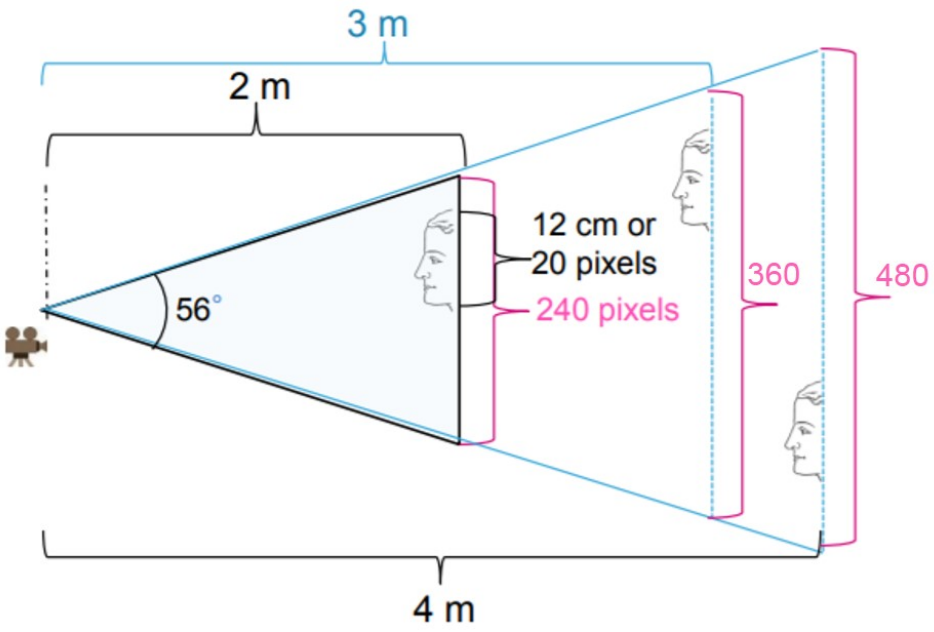


Figure 2.11 – Growth of the search space as objects get further away from the sensor. Detecting an average sized face ($>$ 12cm) 4 meters away from a sensor with a 56° FoV would require scanning 20 pixel windows inside a 480 pixel image.

2.3.1.2 Efficient inference

Once all these trade-offs have been properly tuned to an applicative use-case it is possible to freeze the model and deploy it for inference.

2.3.2 Proposal Network definition and training

The proposal network needs to be a shallow/compact network specialized in identifying interesting regions within an image. The diagram in figure 2.12 below shows in detail the main concepts and levers involved in the definition of the proposal network. A more in-depth explanation is provided for each block in the different parts of this section.

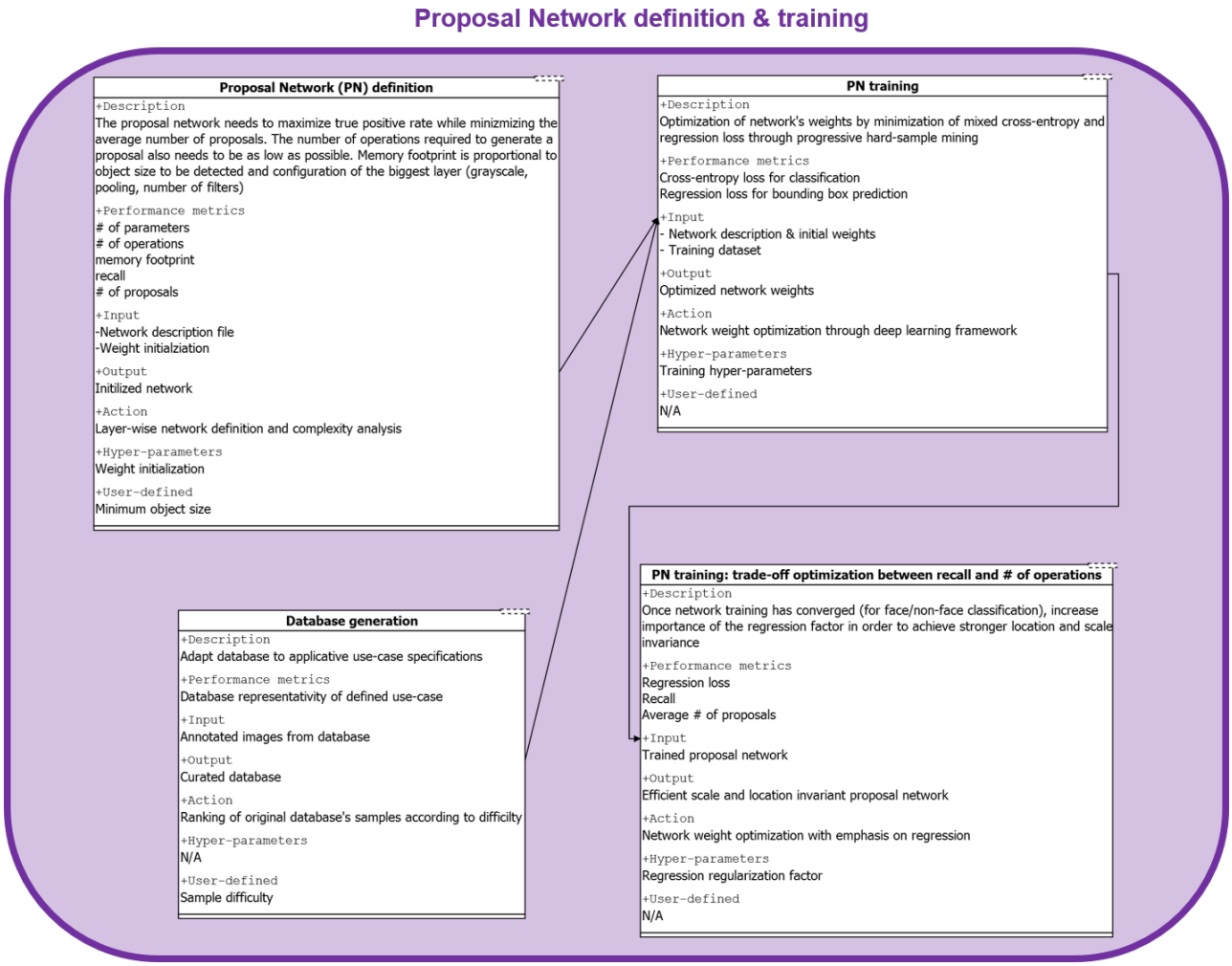


Figure 2.12 – Detailed proposal network definition and training method.

2.3.2.1 Proposal Network definition

Moreover, the ideal proposal network would need to identify all regions of interest in an image both with the lowest number of operations (most importantly) and also parameters. Indeed, as Figure 2.2 showed, even minimal networks (i.e. 1 conv. layer with 16 filters) can be very expensive when applied throughout all possible locations and scales in the image. This is because the network is trained with a fixed input size, and therefore it needs to fall exactly at the right location and scale in the image in order to properly classify the sample, as is shown in Figure 2.13 for a single, fixed image scale. Although some equivariance to these can be gained simply by increasing the variety of scale and location of face crops in the database, it is not enough and smarter mechanisms need to be taken into account.

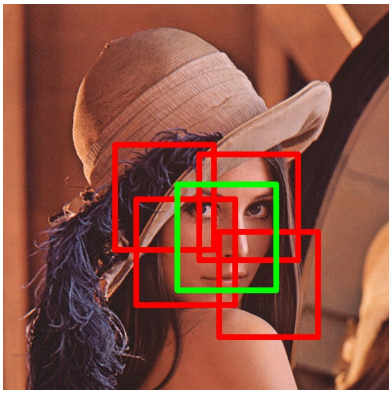


Figure 2.13 – A traditional sliding window search introduces great computational complexity for object detection, since the window being analyzed needs to exactly match the ground truth. This problem is also replicated accross the multiple scales of the image pyramid.

As explained earlier, bounding box regression is a very nice solution to this issue : by training the network to predict bounding box coordinates, the network is now able to predict both location and size of objects, as is shown in Figure 2.14. In the figure, it is possible to see the difference between a traditional sliding window apporach, and bounding box prediction. Concretely, when a more traditional sliding window approach is employed, the analyzed window needs to exactly match the ground truth in order to produce a detection. On the other hand, bounding box regression allows to perform detection without such strict conditions. This allows to increase both the stride and most importantly, the pyramid re-scaling factor, which greatly reduces the number of operations carried out during the proposal stage and is further discussed in section 2.3.2.3.

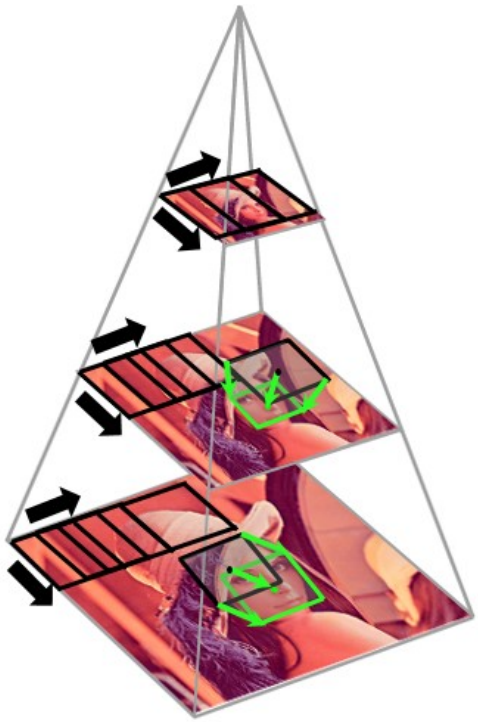


Figure 2.14 – Advantage of bounding box regression as opposed to a traditional sliding window approach.

Memory footprint is also an important design consideration, since the proposal network's input consists of a full-resolution image. Since the processing of a convolutional layer requires

to store the input as well as the layer's weights and output feature maps inside memory, the biggest memory requirements are usually on the first layer. This is because the input to the first layer is a whole image, and output feature map resolution is the same, but also multiplied by the number of output channels. Therefore, the dimensioning of this layer has a big impact on the hardware architecture. It is thereby crucial to clearly define the minimum object size to detect, as this will allow an initial downsizing of the input image, in turn reducing intermediary memory requirements, as will be shown and discussed in section 2.3.1.1. Furthermore, the effect on computational complexity of different design choices such as grayscale inputs(or even infrared as discused in section 4.4.2), or pooling layers can also greatly lower the memory footprint without accuracy degradation, and needs to be studied.

2.3.2.2 Database generation for PN

The amount of difficulty and variability in the samples also influences the complexity of the network.

Since the main mechanism in neural networks to gain invariance to such properties is an increase in network complexity (number of parameters) and variety in the dataset, it is crucial to properly define the applicative use-case in order to design an efficient solution.

Concretely, in our case we contemplate a consumer/collaborative use-case, as opposed to a more surveillance-type of use-case. The difference between these applicative scenarios is clearly pictured in Figure 2.15, where the consumer/collaborative scenario (Figure 2.15.a) is less complex, and more cooperative with the user. This also greatly reduces the search space and therefore the number of operations, as will be discussed in section 2.3.1.1.

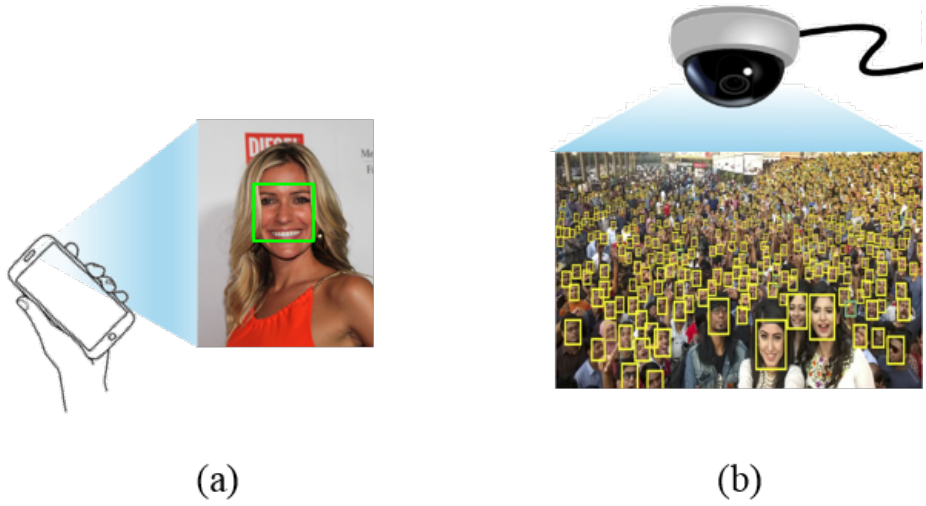


Figure 2.15 – Consumer applicative scenario (a) as opposed to a surveillance scenario (b).

Therefore, acquiring a representative dataset is of clear importance. For example, datasets such as WIDER FACE contain very extreme conditions in terms of occlusions, expressions, make-up or illumination, and are more suited to a surveillance applicative context, as shown in Figure 2.16. On the other hand, datasets such as CelebA pictured in figure 2.17 contain mostly a single, centered and high resolution face and are therefore more suited to a consumer scenario.

Figure 2.16 – Variety in appearance of faces in the WIDER FACE dataset. Image from http ://shuoyang1213.me/WIDERFACE/.

Figure 2.17 – Variety in appearance of faces in the CelebA dataset.

The solution therefore needs to be optimized to be efficient on the proposed use-case, and both the training and testing subsets must also be representative of the applicative scenario. In order to rank the samples according to difficulty, there can be different strategies :

— Finding more adapted datasets. For example, FFHQ or CelebA vs. Wider Face.

— Using flags available in datasets such as in WIDER FACE, indicating the degree of blurness, occlusion, etc...

— Employing pre-trained more complex neural networks in a spirit similar to that of distillation [78] to rank sample difficulty.

— Generating a new database in which conditions accurately represent the applicative use-case. This is a crucial phase, and is not trivial at all. Image acquisition and annotation mechanisms must be put in place in a first stage and need to evolve as the solution improves, as was mentioned in Figure 1.16.

Finally, a manual verification of samples is also very relevant, independently of any of the above-mentioned methods, and is guaranteed to improve results.

2.3.2.3 Proposal Network training

Since the Proposal Network(PN) is a fully convolutional network, the size of the crop samples employed for training must be equal to the PN's receptive field, as shown in Figure 2.18. In order to train the PN, it is necessary to have three types of samples :

— Positive samples : Full face crops.

— Negative samples : Non-face crops.

— Partial samples : Crops containing a small portion of the face along with the coordinates pointing to the ground-truth.

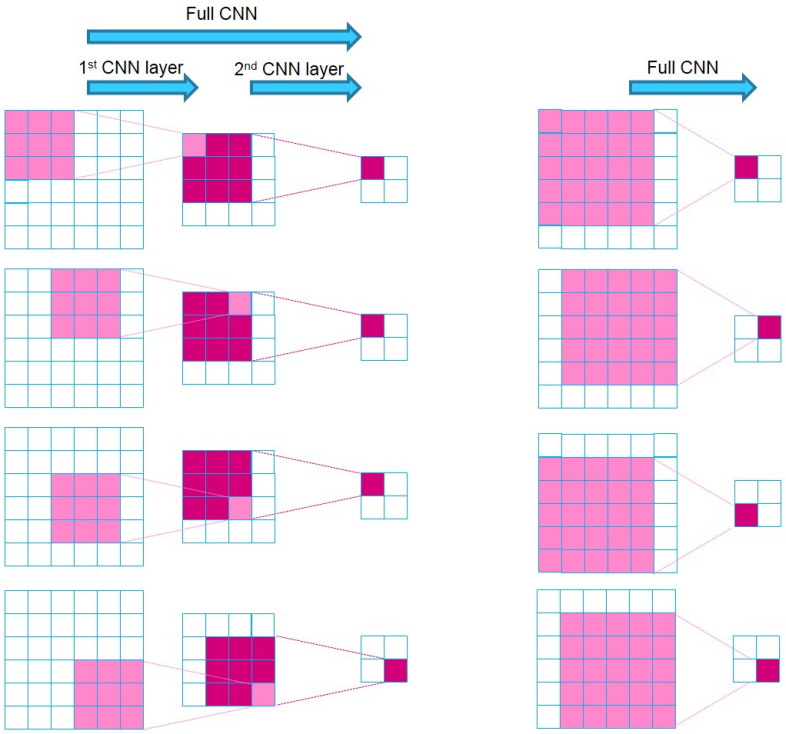


Figure 2.18 – A whole image can be fed to a neural network during inference (a), instead of carrying out a sliding window (b). During training, it might be more convenient to use crops with the receptive field size (b), instead of storing whole images(a).

As indicated earlier, regression plays a crucial role during training in order to obtain location and scale invariance. Moreover, once the network has learned to properly classify face and non-face crops, this is done by showing it only partial crops with annotated displacement coordinates . Concretely, the loss to optimize contains both a cross-entropy term (for binary classification) as well as a regression term (for bounding box prediction) and can be expressed as the following :

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} \lambda_{class}\,[y_i^{class} log(p_i(\mathbf{w})) + (1-y_i^{class}) log(1-p_i(\mathbf{w}))] + \lambda_{reg}\,[\|\hat{y}_i^{reg}(\mathbf{w}) - y_i^{reg}\|_2^2],$$

where :

— λ_{class} is the class regularization factor. This defines the importance of classification as opposed to regression during the optimization phase. In the beginning, it should be higher than regression, so that the network learns to correctly classify face vs. non-face samples.

— λ_{reg} is the bounding box regression regularization factor. As the training progresses and the network learns to differentiate between face and non-face samples, the regression factor should increase in importance, and gradually becoming the main focus towards the last phase of the training. This allows to focus on efficiency of the proposal stage, by monitoring recall vs. average number of proposals vs. number of operations.

— y_i^{class} is the one-hot encoded ground-truth class vector for each sample i, in this case $y_i^{class} \in \{0, 1\}$, while p_i represents the output predicted by the network for each sample i, such that $p_i \in [0, 1]$.

— y_i^{reg} represents the ground-truth bounding box coordinates of the sample i, containing left, top, height and width, and therefore $y_i^{reg} \in \mathbb{R}^4$. The term $\hat{y}_i^{reg}$ represents the coordinates predicted by the network.

Furthermore, *hard sample mining* is also employed during training. This allows the network to learn in a progressive manner, and has been proven to obtain better performance in tasks such as face detection [104]. For this, as training progresses, the loss value of each batch of samples is first computed through a forward propagation step, and backward propagation and weight updates are only carried out if the batch's loss exceeds a certain threshold.

2.3.2.4 Tuning trade-off between recall and number of operations

As previously mentioned, regression is quite important during training. Moreover, we need to optimize the network in such a way that *recall* (i.e. percentage of relevant instances found by the detector) is as high as possible, while minimizing the number of average proposals for a given decision threshold. Furthermore, these need to be monitored and compared across a varying number of pyramid re-scales in order to verify scale invariance, as will be shown in the results provided in Chapter 4.

However, this is a very tedious and "blind" task since at training time, the PN only sees crops, while the PN needs to be evaluated over full-image datasets with annotations and therefore evaluation is done separately from training. Setting up more rigorous metrics or adding new terms to the loss function in order to evaluate the ability to perform bounding box prediction could be of great interest and could potentially improve results. For example, partial crops could be ranked according to intersection over union (IoU) with respect to the ground truth. That would allow to accurately evaluate regression ability, as well as hard sample mining : samples with a higher IoU would be shown at the beginning of the regression training phase, while harder samples (lower IoU) would be shown towards the end.

2.3.3 Proposal Network compression

Since we need to simplify the computational complexity of the solution as much as possible, CNN compression techniques are of great interest, since they allow to trade-off complexity for accuracy. Moreover, we mainly leverage the combination of the following compression techniques : PCA, winograd and quantization. This topic will be discused in great detail in Chapter 3, while this section overviews the main interests and levers of such techniques,

as shown in Figure 2.19. The main trade-offs achieved in our face detection use-case are described in Chapter 4, while their hardware evaluation is described in Chapter 5.

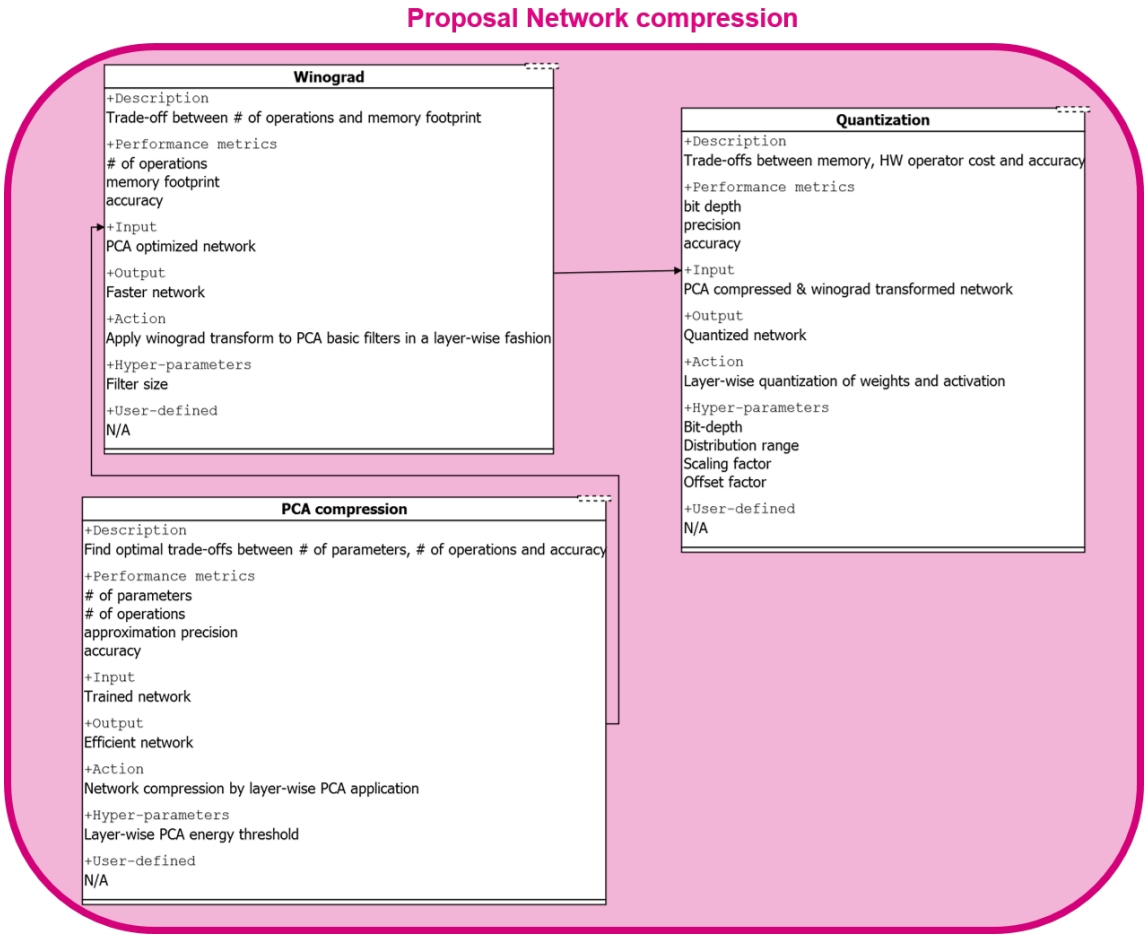


Figure 2.19 – Detailed proposal network compression method.

2.3.3.1 PCA compression

Principal Component Analysis can also be applied to reduce neural network complexity, as described in our work [111]. It allows to find optimal trade-offs (in the MSE sense) between the number of operations, parameters and network accuracy. Further fine-tuning allows for better trade-offs between computational complexity and accuracy and will be thoroughly explained in section 3.2.

2.3.3.2 Winograd

The Winograd algorithm is described in [89]. It is a minimal filtering algorithm and consists on performing convolution in the frequency domain (multiplication). It can be combined with PCA by applying it to the set of basic filters, as will be further discussed in section 3.5.3.

2.3.3.3 Quantization

A neural network quantization method is described in [61], it is a well studied topic in CNNs and can be combined with all the previous discussed methods. As is the case in PCA, better trade-offs are achieved by an additional fine-tuning step.

Results from applying this compression workflow are discussed in Chapter 3 for more general cases (10-way classification), while results for the case of face detection are shown in Chapter 4.

2.3.4 Binary Classification Network definition and training

In this section, the main considerations needed to define the classification network are described and shown in Figure 2.20. Concretely, this network needs to deliver a very accurate classification with a low as possible computational complexity, as will be overviewed in the following sections. Moreover, since this network needs to correct the outputs of the proposal network, the database used for training must take into account the outputs generated by the optimized network obtained at the end of section 2.3.3.

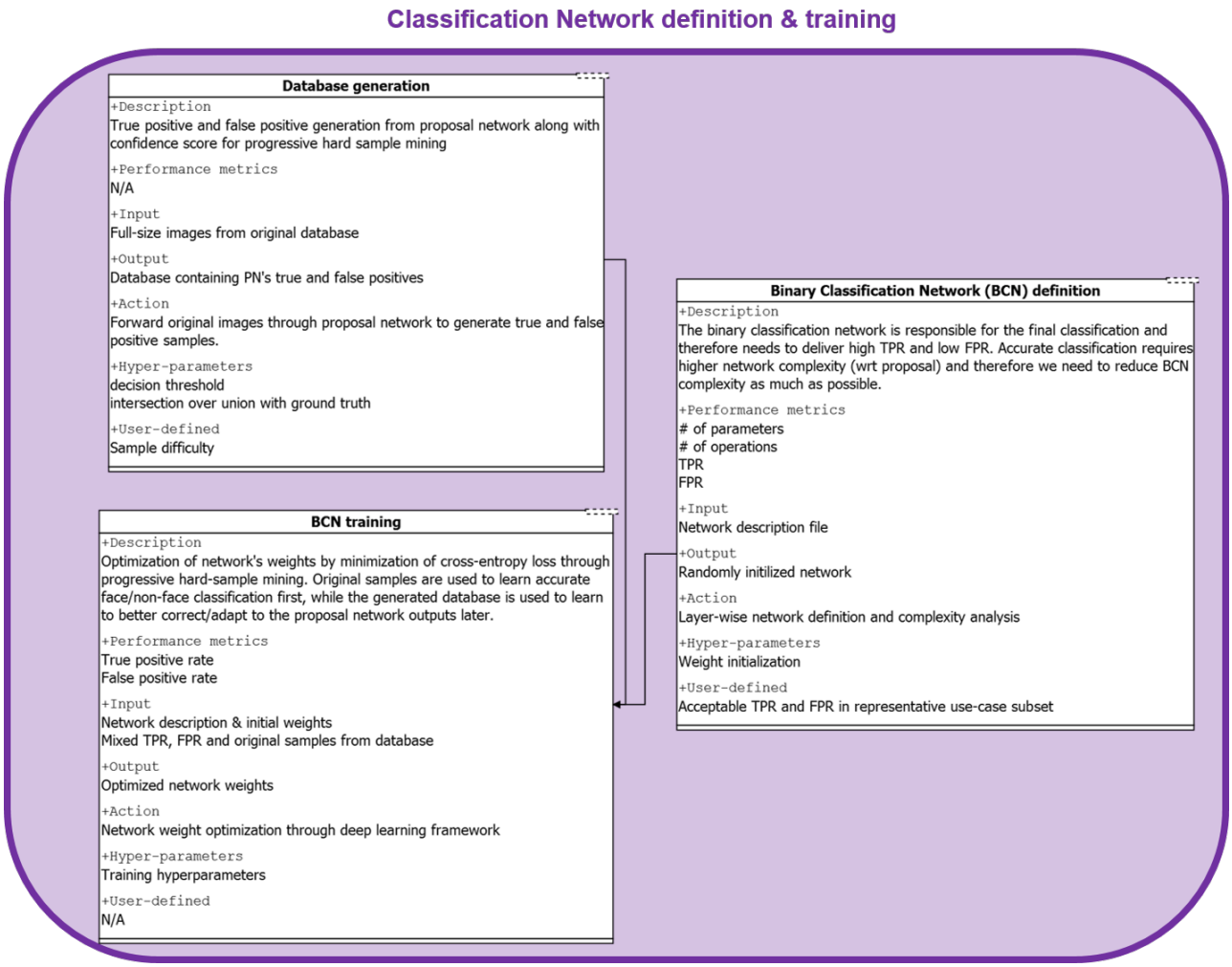


Figure 2.20 – Detailed classification network definition and training method.

2.3.4.1 Binary Classification Network definition

Once an efficient proposal network has been defined, a more complex network needs to achieve high *precision* on the set of proposals (i.e. out of all proposals classified as positive, what percentage is really correct). Ideally, this network needs to provide a high as possible *true positive rate* (TPR, i.e. percentage of actual positives which are correctly identified) and a low as possible *false positive rate* (FPR, i.e. percentage of wrong positives which are incorrectly identified). Since this needs to be a more accurate task than the proposal stage, network complexity is also higher and amounts for most of the total number of parameters.

2.3.4.2 Database generation for BCN

The BCN needs to accurately identify faces, but most importantly it also needs to be tuned to learn to correct the PN's outputs. In order to achieve this, a new dataset needs to be generated by scanning full-sized images with the PN. Proposals with high confidence scores and high IoU with respect to the ground truths are kept as positive training samples, while proposals not matching a ground truth (or with low IoU) are kept as negative training samples. The confidence score generated by the PN is also kept as a measure of sample difficulty for hard sample mining. Positive and negative samples from the PN dataset are also employed during training.

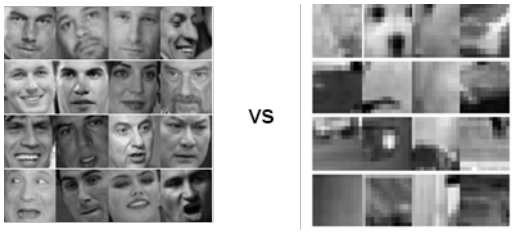


Figure 2.21 – Face versus non-face samples.

2.3.4.3 Binary Classification Network training

The BCN training also consists in minimizing a cross-entropy loss function. The network is first trained to recognize face and non-face samples from the dataset defined in section 2.3.2.2. The network is then fine-tuned on the set of proposals generated in section 2.3.4.2. Hard-sample mining is also employed throughout the training phase.

2.3.5 Classification Network compression

Moreover, the classification network is more complex than the proposal network (both in terms of number of parameters and operations), and the number of operations carried out by the detector for each frame is proportional to the number of generated proposals. Therefore, reducing its computational complexity by trading-off accuracy is of very high importance, as displayed in the diagram from Figure 2.22.

Furthermore, high recall rates on the PN usually generate a high average amount of proposals passed on to the classification network, as will be shown in Chapter 4. This means that

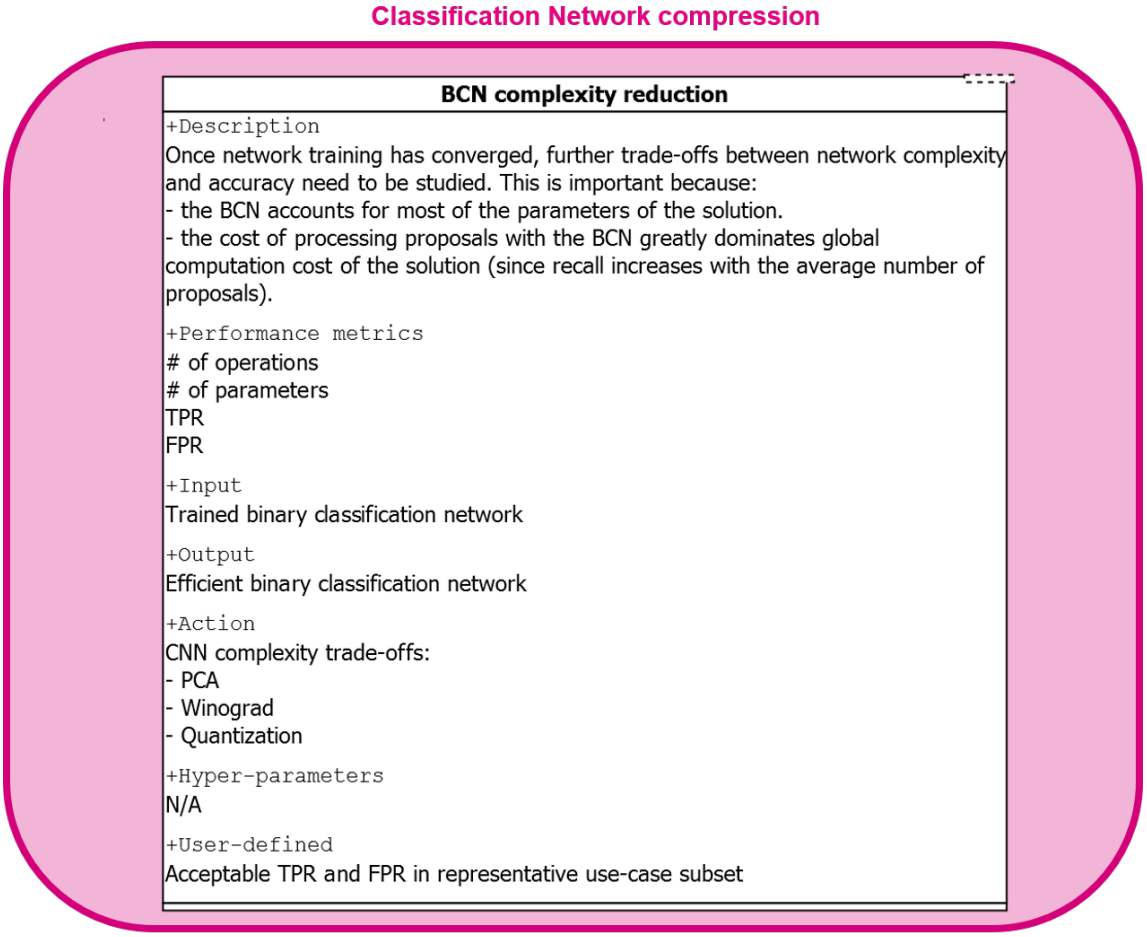


Figure 2.22 – Detailed classification network compression method.

high recall rates in complicated datasets such as WIDER FACE induce a high number of operations to be carried out by the classification network, quickly outweighing computational cost of the proposal network. Compression methods therefore become of even greater interest for this network, as will be discussed in Chapter 4.

2.3.6 Evaluation

The last step involves both the algorithm evaluation, as well as hardware implementation of the proposed algorithm as well as evaluation, as is shown in Figure 2.23.

2.3.7 Algorithm benchmarking

The solution then needs to be benchmarked on all key performance indices against other state-of-the-art solutions. This is done for the applicative scenario studied for face detection in Chapter 4. In our case, the most important performance metrics are the following :

— Number of parameters.

— Memory footprint.

— Number of operations.

Figure 2.23 – Detailed evaluation block.

— Final true positive rate.

— Final false positive rate.

— Hardware friendliness at implementation.

2.3.8 Hardware evaluation

Finally, the proposed solution needs to be evaluated on hardware in order to verify that the proposed optimizations still preserve parallelization ability. Chapter 5 shows results relative to this point.

2.3.9 Conclusion and Perspectives

In this chapter we have overviewed the main methods from the state-of-the-art in CNN-based object detection, and identified the main levers and ideas allowing to tailor computational complexity of a CNN-based object detection solution.

We have then proposed a generic workflow, allowing to easily fix trade-offs between detection performance and computational complexity. Concretely, decomposing the detection problem into a proposal and classification stage allows to very flexibly tune computational complexity and accuracy of each of the networks making up the solution.

Further trade-offs are then made available through other network compression methods. These compression methods will be discussed in greater detail in Chapter 3.

The method is then put into practice for a real face detection application scenario in Chapter 4. The generated solution is then evaluated in hardware in Chapter 5.

3

CNN compression

In this chapter we will start by reviewing the main techniques from the state-of-the-art allowing to reduce the computational complexity of CNNs. We will also provide an outlook of the main implications when implementing each of the methods. We will then present our proposed compression method, which is based on Principal Component Analysis (PCA) and which allows for flexible trade-offs between model accuracy, number of parameters and number of operations. The method is then evaluated on multiple datasets and network architectures. We also show the compatibility of the proposed method with other compression and speed-up techniques from the state-of-the-art.

Sommaire

1.1 Neural Network reminder 9
1.1.1 Artificial Neural Networks 9
1.1.2 Convolutional Neural Networks 10
1.1.3 Activation Layer 13
1.1.4 Pooling/subsampling Layer 14
1.1.5 Fully Connected Layer 15
1.1.6 Softmax Layer 15
1.1.7 Learning Features 16
1.1.8 Feature interpretability 17
1.1.9 Loss Function 17
1.1.10 Minimizing the loss function 18
1.1.11 Topology of the loss function 20
1.1.12 Training and test distributions 22
1.1.13 Overfitting and Underfitting 23
1.1.14 Deep Learning vs traditional machine learning 25

1.1.15 Deep Learning Frameworks . . . 25
1.1.16 Deep Learning in practice . . . 26
1.2 CNN architecture comparison . . . 27
1.2.1 AlexNet . . . 28
1.2.2 ZFNet . . . 28
1.2.3 VGG . . . 28
1.2.4 Network in Network . . . 28
1.2.5 GoogleNet/Incenption . . . 28
1.2.6 HighWay Networks . . . 28
1.2.7 ResNets . . . 29
1.2.8 Wide ResNet . . . 29
1.2.9 ResNext . . . 29
1.2.10 DenseNet . . . 29
1.2.11 Squeeze-and-Extraction . . . 29
1.2.12 FractalNet . . . 29
1.2.13 MobileNets . . . 29
1.2.14 ShuffleNets . . . 30
1.2.15 SqueezeNet . . . 30
1.2.16 NASNet . . . 30
1.3 Realistic CNN applications (as of now, 2019) . . . 31
1.3.1 Wake-up systems . . . 31
1.3.2 Image Classification . . . 31
1.3.3 Object Detection . . . 31
1.3.4 Depth Estimation . . . 32
1.3.5 Image Segmentation . . . 32
1.3.6 Gesture Recognition . . . 33
1.3.7 Image reconstruction/extrapolation/compression/noise reduction 33
1.3.8 Natural Language Processing . . . 34
1.3.9 Image Understanding . . . 34
1.3.10 Autonomous Navigation . . . 34
1.4 CNN hardware accelerators . . . 36
1.5 Neuromorphic computing . . . 38

In this Chapter we present a novel CNN compression and speed-up method which is based on Principal Component Analysis (PCA) [111], which is first benchmarked on multiple CNNs and datasets from the state of the art as well as for a real application such as face detection, as will be presented in Chapter 4. The compressed network is then implemented on an embedded multiprocessor in Chapter 5, showing the effectiveness and interest of the proposed method to target a hardware implementation.

The chapter is organized as follows : section 3.1 gives a broad overview of the main CNN simplification techniques employed in the state of the art as well as their positive and negative aspects when considering HW implementation. Section 3.2.1 formalizes the proposed method in a theorical way and shows how memory gains are obtained through PCA, while section 3.3 discusses how compute gains can be obtained. Section 3.4 shows experimental results and multiple variants of the proposed algorithm. Finally, section 3.5 explores the compatibility of the proposed PCA compression with other techniques reviewed in section 3.1.

3.1 CNN compression methods

As overviewed in section 1.2, the design space of CNNs is large. Recently, efficiency is becoming a CNN architecture design trend, and more recent models tend to be more compact. Consequently, the CNN model should also be carefully considered prior to compression, as this will give more or less room for simplification. Task complexity is also an important factor, since neural networks seem to greatly learn by memorization and therefore, tasks involving a large degree of variety in their distribution also require higher computational complexity.

Once the model architecture has been chosen, there are multiple approaches to further reduce its computational complexity, allowing to fit CNN models in hardware implementations. The most popular and well-known techniques are reviewed in this section.

3.1.0.1 Quantization

This means representing data, namely weights and activations in the case of neural networks, by different levels of precision or number of bits. The benefits of lower precision are both a reduced storage cost, as well as reduced computation requirements. It is a well developed area both in the field of NNs and embedded systems.

The goal is therefore to convert the network from a floating point representation, which is used during training, to a fixed point format, used for inference. Since floating point representations are rare in embedded systems, quantization is essential.

Fixed point representation can either be dynamic or static. In the static case, the position of the decimal point is the same for all values, which makes it the simplest form of representation. However, in the context of CNNs, different layers have different dynamic ranges and also require different levels of precision. As a consequence, a differentiated fixed point

representation, where each layer of weights or feature maps have their own fixed point representation, usually improves the network accuracy.

Furthermore, since CNN weights are not uniformly distributed, a strategy is to elaborate quantization schemes which are well adapted to these distributions. For example, the work in [61] explores a quantization method which is not centered around zero and which allows for a scale factor different than a power of 2 (as is the case in fixed point). Most works also leverage training and quantization in order to reach more interesting trade-offs between accuracy and arithmetic precision.

A special case of quantization is that of binary networks. In these, all values are represented by -1 and +1. As discussed in [62–64], this quantization can be performed on the weights with low impact in network accuracy, allowing to reduce weight size by a factor of 32 with respect to a 32-bit implementation. Since weights have binary values, multiplication is no longer necessary and convolution can be computed through addition and substraction uniquely, leading to a x2 computation speed-up. However, these networks require an increase in parameter count of the order of $\times 5$ in order to reach the accuracy of 8-bit precision networks [66].

Furthermore, when activations (ie feature maps) are also binary, convolution can be very efficiently computed by XNOR gates and bit counting operations, leading to a x58 computation speed-up. However, accuracy drops considerably in this case and training is not trivial because stochastic gradient descent is not compatible with non-differentiability of binary variables. In order to alleviate this issue, [63] proposes multiplying the outputs with a scale factor to recover the dynamic range (where the scale factor is the average of the absolute values of the weights in the filter), they keep earlier layers in the network at higher precision and also employ batch normalization in order to reduce dynamic range of activations. Still, the accuracy of binary network is poor and degradation is much greater than 10%, relative to a fixed point network of same architecture and size.

Another solution is to increase the set of possible values by adding an extra bit and allowing the network to use the zero value, i.e. the weights can have -1, 0 and 1 values. Sparsity can then be exploited in order to reach performances close to that of binary networks, as shown in [65].

Finally, [66] propose to decompose the set of weights and activations as a linear combination of some binary basis functions and their corresponding coefficients. If only one basis term is employed, this is equivalent to [63]. However, when a higher number of basis terms is employed (5), much more flexible trade-offs are obtained, with almost no accuracy degradation, the only downside being that memory gains are instead equivalent to that of 5-bit quantization. The network can still be implemented through XNOR gates if both weights and activations have been decomposed into linear combinarions of binary basis functions, while also preserving accuracy.

3.1.0.2 Pruning

Pruning allows to reduce effective parameter count by replacing low values with zeros [67]. Indeed, zero values in a convolution kernel indicate the absence of a pattern and although necessary, can easily be removed from the network. Most networks from the state of the art allow for large amounts of weights to be removed by comparing the magnitude of individual weights to the distribution in their layer. Accuracy is slightly sacrificed but can be recovered

by further retraining phases as [68] shows. This retraining can be seen as a training phase in which the network automatically learns to identify important parameters. However, this creates holes in the network and requires indexing the position of parameters that can be discarded. This carries along several downsides :

— Additional hardware such as [69] is required in order to efficiently decode and skip calculations with zero values.

— The irregular access patterns reduce parallelism effectiveness.

— The sparsity generates workload imbalance since processing elements do not have the same amount of computations.

— Room for pruning decreases on more efficient models and therefore memory and indexing overheads become more substantial and reduce effectiveness of this technique.

— It is not trivial to combine with Winograd [89], since transforming filters and feature maps into the Winograd domain destroys sparsity. The work in [71] therefore proposes to move the ReLu activation function into the transformed domain, as well as pruning filters in the transformed domain as shown in Figure 3.1.c.

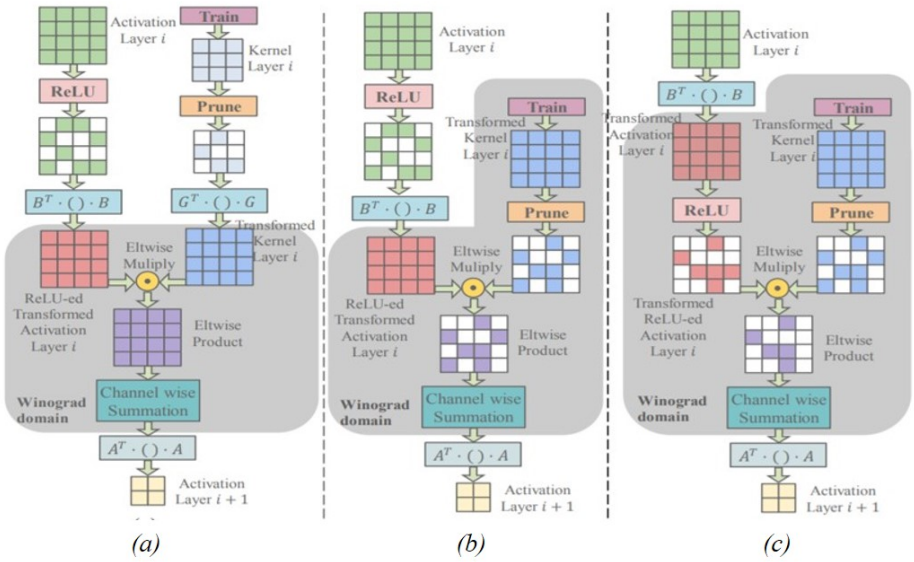


Figure 3.1 – Combination of pruning with Winograd. Figure from [71]

The same philosophy holds for feature maps, which are usually sparse thanks to the use of the ReLU activation function and which can also further be pruned. However, unlike weights, feature maps vary at inference time and it is therefore hard to guarantee compression gains. The work in [50] exploits these properties to design a hardware accelerator which is able to deal with sparsity of both weights and activations. Furthermore, [74] also tries to leverage pruning and energy efficiency by adding a "hardware-in-the-loop" phase. During this phase, CNN architecture design choices directly reflect the impact in terms of energy in order to guide the pruning process. As shown in Figure 3.2, the authors in [74] also propose a feature-map-based type of pruning in which the goal is to prune filters in such a way that the produced feature-map is close to the original one. A further fine-tuning phase allows to reach even better trade-offs.

Similar to the previous idea, the work in [75] propose to stop processing once its is useless to continue computing some branches of the network (completly empty feature maps, etc...). In

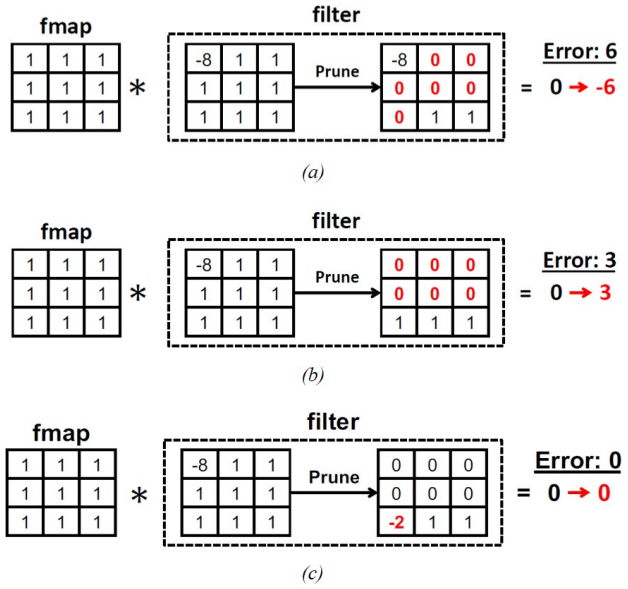


Figure 3.2 – Different approaches to pruning : (a) Magnitude-based filter pruning. (b) Feature-map-based pruning. (c) Feature-map-based pruning with fine-tuning.

a similar spirit, the work in [76] proposes to add a classifier in between each network layer, in order to early classify the sample without carrying out the whole network's processing. Also similarly, but opposite, the work in [77] proposes to only add neurons to the network on-demand, and training the added neurons independently of the whole network.

3.1.0.3 Entropy Encoding

This is a well known lossless compression method that relies on the idea that more common symbols should be represented through shorter codes. It has been applied to CNNs and combined with pruning in works such as [68], however it also requires specialized hardware in order to decode the weights and offers a lower compression gain than in the case of pruning.

3.1.0.4 Weight Sharing

The work in [68] reduces weights storage requirements by clustering sets of weights and assigning them to a common centroid. The weights therefore become an index pointing towards a centroid look-up table which is stored in memory. A retraining step is also involved in order to allow the network to adjust centroids to reach better accuracy/compression trade-offs. However, this method also requires specialized hardware and compression gains are also lower than in the case of pruning.

3.1.0.5 Distillation

This method consists in transferring the knowledge learned by some complex model (teacher) to a simpler model (student) as described in [78]. Distillation can in some cases lead

to a better performance in terms of accuracy than training a simpler model from scratch and could also be involved in the retraining phase of methods such as pruning or binary networks. It has also been used to transfer the knowledge from an ensemble of networks into a single network [79].

3.1.0.6 Filter Parametrization & Transformation

Observation of the filters learned by neural networks has shown interesting properties that could potentially be exploited by hardware implementations.

For example, filters in the first layer of networks such as [10] look like Gabor functions, which are known to be present in the human visual system [80]. The work in [81] takes advantage of this observation by modulating learned filters by a set of Gabor basis filters, in order to avoid learning filters with redundant orientation and scale information. As Gabor filters are tuned by both a frequency and an orientation parameters, their representation is more compact. However, this is mostly only valid for early layers of the network, and hardware complexity to generate Gabor filters from their parameters is too high.

In a similar spirit, the work in [105] tries to exploit symmetries in the filters learned by CNNs which are caused by the ReLU activation function. It does this by slightly modifying the activation fuction. The work in [83] tries to generalize the concept of translation equivariance inherent in CNNs to other kind of symmetries with the goal of also reducing parameter count. The works in [84] and [85] provide a way to learn affine equivariances but do not address the fact that objects can also undergo local transformations. To solve this, instead of building models that are globally equivariant to affine transformations, [86] and [87] rely on the assumption that complex objects can be decomposed into simpler objects. Since object parts exhibit less variance in terms of appearance than full objects, they should be easier to learn and objects can then be recognized from their parts and poses.

Other works also aim at representing the learned CNN in a different, more appropriate domain. For example, the work in [88] argues that the spectral domain provides a powerful and efficient representation for CNNs. In the same line of work, [89] shows that it is also possible to transform a commonly learned CNN into the spectral domain on the fly in order to exploit the computational gain obtained by performing multiplication instead of convolution, specially for small filters, where FFTs are inefficient.

The work in [90] shows that it is possible to design neural networks for which a large portion of the weights are randomly set, and only a small portion are tuned. They show comparable performance to CNNs in simple object recognition benchmarks. The work is interesting, since a great amount of weights could be randomly generated while only storing a small portion of the learned weights in memory.

In a similar approach, the work in [91] aims at exploiting the redundancy present in CNNs by finding an appropriate low-rank approximation and is similar to the approach taken in this chapter.

Eventually, the reader may observe that most of the presented CNN optimizations aim at exploiting different properties inherent to CNNs, and are therefore highly complementary, leading to very efficient CNN implementations when combined.

3.2 PCA for CNN compression

From the previous section, it is evident that CNNs are hugely over-parametrized models. Several works [92] claim this to be a key factor for the success of deep learning, in the sense that this over-parametrization allows us to solve a highly non-convex optimization problem through convex optimization methods such as Stochastic Gradient Descent (SGD).

Indeed, the over-parametrization of such models induces a highly underdetermined system of equations, i.e. an infinite number of equivalent solutions. Intuitively, this generates huge flat equivalence regions on the loss landscape [93], to which SGD would be more likely attracted, as was explained in section 1.1.11.

However, this over-parametrization is clearly a nuisance when trying to implement such models into embedded systems, for which computation capability is very constrained compared to other general-purpose systems such as desktop computers.

In this work, we aim at finding a way to identify important parameters of a learned CNN. For this, we propose to exploit the redundancy in each of the CNN's layers by finding a more appropriate set of basis filters by the means of Principal Component Analysis (PCA). The idea is to take as a starting point a learned CNN, to then compress it through PCA, in order to retain only the relevant parameters. Experiments on state of the art CNNs and a custom CNN show the effectiveness of the method. Variants of the method are then proposed, and we show that the naïve compression scheme benefits from a slight retraining phase, which remarkably recovers the compressed CNN's accuracy.

3.2.1 PCA for memory reduction

Principal Component Analysis [111] is a popular dimensionality reduction technique. It is here applied to CNNs in order to decompose a set of learned filters into a linear combination of lower-dimensional basis filters and their corresponding coordinates.

The idea is then to apply this transformation in a layer-wise fashion throughout the network, allowing us to select only effective parameters in the network.

More concretely, for any given layer consisting of N filters of size $d = H_f \times W_f \times C_f$, we stack each of the N 3-D flattened filters in that layer $\mathbf{X}_i \in \mathbb{R}^d$, in order to form $X = [\mathbf{X}_0, \mathbf{X}_1, ..., \mathbf{X}_{N-1}]^T \in M_{N\times d}$.

The empirical covariance matrix of the filters in that layer, $C_X \in M_{d\times d}$, is computed by $C_X = X_c^T X_c$, where $X_c = X - \mu \mathbb{1}_{N\times d}$, and $\mu = \frac{1}{N}\sum_{i=0}^{N-1} \mathbf{X}_i \in \mathbb{R}^d$. Note that, in the context of deep learning, matrix-vector additions are allowed, and are done in the same way as the *broadcasting* operation supported in libraries such as *Numpy*.

The goal of PCA is to find P, such that $Y_c = X_c P$, where Y_c is a new representation of the learned filters, such that C_{Y_c} is diagonal (covariance is 0 in this new representation of the data).

This is done by diagonalizing the mean-centered empirical covariance matrix C_X. Since C_X is a positive semi-definite symmetric matrix, it admits an orthogonal diagonalization, that is, $C_X = QDQ^T$, such that Q is orthogonal, its columns are formed by the eigenvectors $\mathbf{v}_k \in \mathbb{R}^d, k \in [0, d-1]$, and their corresponding eigenvalues are contained in the diagonal

matrix D, and are all real and non-negative.

Furthermore, if we choose the transformation matrix to be $P = Q$, it can be seen that, under this transformation, Y_c is a new set of filters which are indeed uncorrelated :

$$\begin{aligned} C_{Y_c} =& Y_c^T Y_c = (X_c P)^T (X_c P) = P^T X_c^T X_c P = P^T C_x P = P^T Q D Q^T P = \\ =& P^T P D P^T P = P^{-1} P D P^{-1} P = D \Leftrightarrow C_{Y_c} = D. \end{aligned}$$

That is, PCA allows us to find an orthonormal basis P of the learned filter space in which the set of filters are now uncorrelated. Rows of Y_c represent the coordinates λ_d^i in the orthogonal basis P and conversely, it is possible to write any of the original filters as $\mathbf{X}_i = \sum_{k=0}^{d-1} \lambda_k^i \mathbf{v}_k + \mu$. Equivalently, in matrix form we have $X = Y_c P^{-1} + \mu \mathbb{1}_{N \times d}$.

Furthermore, since PCA is an orthogonal transformation, the energy content (Frobenius norm of the transformation matrix) is preserved throughout the transformation and is contained in the first few eigenvalues :

$$\|Y_c\|_F = \|X_c\|_F = \sqrt{trace(X_c^T X_c)} = \sqrt{trace(C_X)} = \sqrt{trace(QDQ^T)} =$$

$$= \sqrt{trace(DQ^T Q)} = \sqrt{trace(DQ^{-1} Q)} = \sqrt{trace(D)}.$$

This property enables dimensionality reduction by approximating the original set of filters through a subset of the L eigenvectors associated to the L largest eigenvalues such that :

$$\|X_c - \hat{X}_{c\{0:L-1\}}\|_F = \ e \ , \text{with e} \ \propto \sqrt{trace(D_{\{0:L-1\}})}.$$

That is, we can optimally approximate (in the MSE sense) any of the original filters up to an arbitrary mean squared error (MSE) by choosing a subset of the first L eigenvectors and their corresponding coordinates through $\hat{\mathbf{X}}_i = \sum_{k=0}^{L-1} \lambda_k^i \mathbf{v}_k + \mu$ or equivalently, in matrix form $\hat{X} = Y_{c\{0:L-1\}} {P^{-1}}_{\{0:L-1\}} + \mu \mathbb{1}_{N \times d}$.

The energy threshold for choosing the size of the PCA basis is computed by comparing the energy content of the reconstructed version of the filters and the energy content of the original filter matrix, that is :

$$E(\%) = 100 \times \frac{\|\hat{X}_{c\{0:L-1\}}\|_F}{\|X_c\|_F} = 100 \times \frac{\sqrt{trace(D_{\{0:L-1\}})}}{\sqrt{trace(D)}}.$$

More concretely, direct application of the PCA algorithm allows to decompose a set of filters in any given layer into a hierarchical set of basis filters along with a set of coordinates. This enables the reconstruction of the original set of filters up to an arbitrary MSE, which decreases with the number of basis filters kept for the reconstruction. For example, Figure 3.3 shows a 128-filter layer being reconstructed by its first 11 principal components with a MSE of 6.83×10^{-5}. In this case, direct substitution of the original set of filters by their reconstructed version doesn't produce any noticeable degradation in terms of network accuracy.

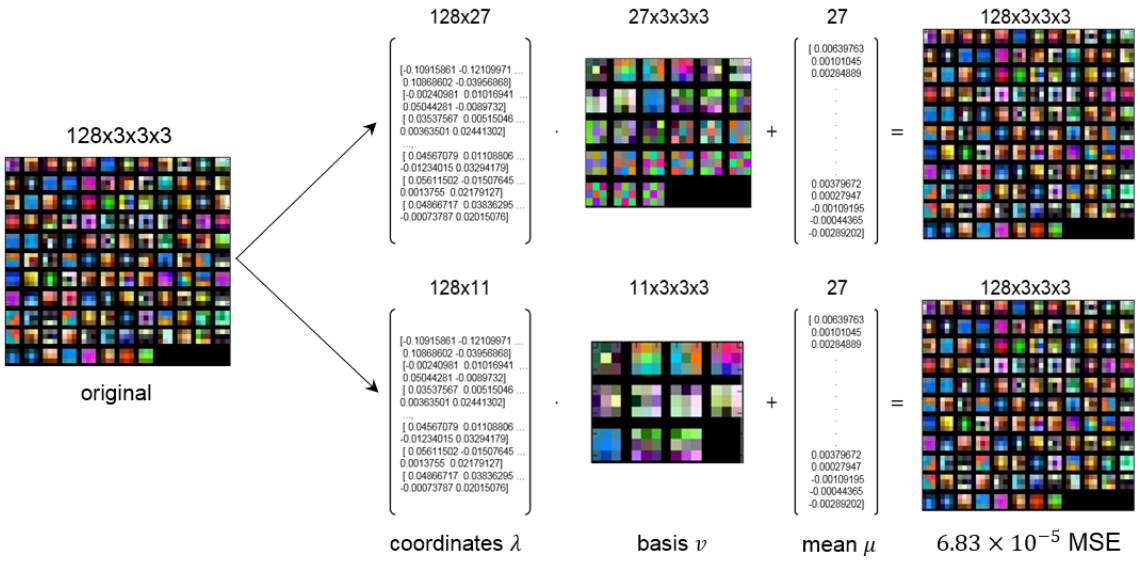


Figure 3.3 – PCA decomposition for a set of filters inside a convolution layer. The top figure shows a perfect reconstruction of the original filters by keeping all basis filters, while the second figure shows an approximation obtained by keeping only the 11 most important basis filters. Best viewed in color.

3.3 PCA for computation reduction

The PCA compression scheme presented in the previous section is also a powerful way of reducing the number of operations performed by convolution. Indeed, convolutions can be performed directly in the reduced PCA basis space and results can then be efficiently recombined. This property emerges naturally from the linear nature of the PCA transformation and associativity property of convolution.

More concretely, Figure 3.4.a shows the reconstruction of the first $3 \times 3 \times 3$ filter from Figure 3.3 as a linear combination of a reduced set of 11 coordinates and $11 \times 3 \times 3 \times 3$ basis filters plus a mean $3 \times 3 \times 3$ filter.
In the context of CNNs, this combination can be achieved through 1x1 convolutions, and the filter reconstruction can be expressed as :

$$f_{reconstructed} = f_{basis} * f_{coord}, \tag{3.1}$$

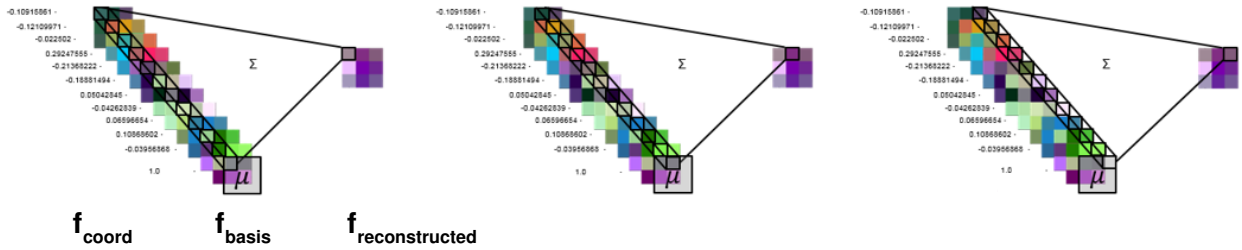
where $*$ denotes the convolution operation, f_{basis} represents the set of basis filters $\mathbf{v}_{k\in\{0:L-1\}}$ along with the mean filter μ, and f_{coord} is a 1x1 kernel built from the set of coordinates λ_k^i, $i \in [0:N-1]$ and $k \in [0:L-1]$, as shown in figure 3.4.b .

Since convolution is associative, for a given input I, we can write :

$$I * f_{reconstructed} = I * (f_{basis} * f_{coord}) = (I * f_{basis}) * f_{coord}. \tag{3.2}$$

This means that, instead of directly convolving the basis filters with the coordinates to regenerate the reconstructed filters $f_{reconstructed}$ and applying this later to the input, we can

(a)



(b)

Figure 3.4 – Filter reconstruction viewed as a 1x1 convolution between the basis filters f_{basis} and the coordinates f_{coord}. A kernel is reconstructed from the kernels of the PCA basis filter and its coordinates in that basis.

directly apply the basis filters f_{basis} to the input feature map I, and convolve the result by f_{coord} in the so-called recombination layer.

This leads to a reduction in the number of operations of :

$$\text{Operation gain} = \frac{N \times H_f \times W_f \times C_f}{(L+1) \times (H_f \times W_f \times C_f + N)}, \tag{3.3}$$

where N is the number of original filters (before PCA), L is the number of basis filters kept through PCA and $H_f \times W_f \times C_f$ correspond to the original size of the filter's convolution kernels, which is also the size of $f_{reconstructed}$.

Applying convolutions in the PCA basis leads to a virtual intermediate feature map in the PCA space, as is shown in Figure 3.5, which is then recombined in order to get back to the original feature map space. This intermediate feature map has no memory cost when implemented on hardware, since the recombination layer is computed on the fly.

Furthermore, PCA is also very well suited for a hardware implementation, since the number of basis filters to be kept in each layer can easily be tuned without noticeable accuracy degradation. This allows to conveniently tune layer sizes (in an optimal way) in order to better fit hardware architectures. In turn, this allows to lower overheads and therefore increases hardware efficiency in terms of MAC/Cycle, as will be shown in future sections.

3.4 Experiments

A naïve implementation of the PCA compression proves this method to be an effective approach for compressing CNNs as will be shown in this section from experiments on different networks and datasets. The first strategy involves compressing all layers in a CNN with an uniform energy threshold, however this is not ideal and further improvements are proposed in latter sections.

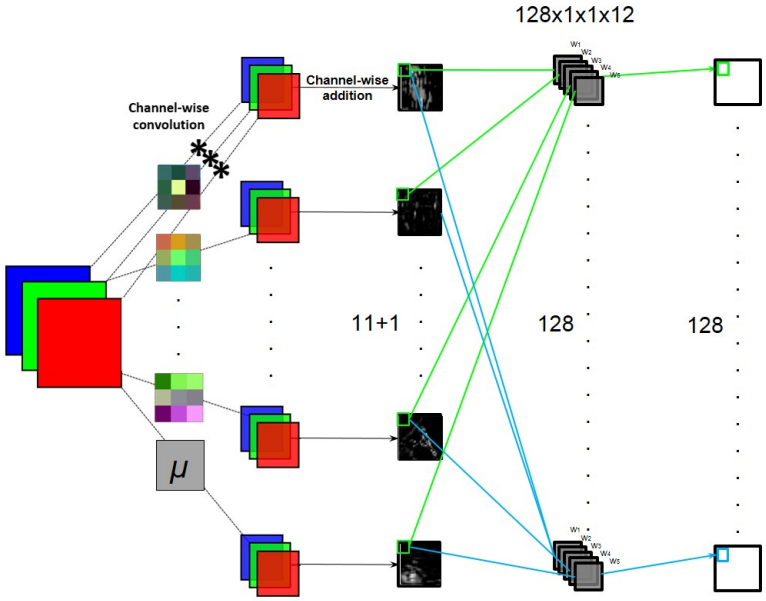


Figure 3.5 – Computational Complexity is reduced by only convolving the input feature maps with a subset of the PCA basis filters and then recombining the resulting feature maps from the set of PCA coordinates.

For our experiments we benchmark three different networks from the state of the art. All of them are fully convolutional and employ batch normalization and ReLU activations along with a few max pooling stages. Moreover, we choose an overparametrized model such as VGG consisting of 9 layers and 7.331M parameters as an upper bound and on the opposite extreme, a deeper and more efficient 32 layer ResNet model consisting of 0.453M parameters and a similar accuracy level. We also employ an intermediary 110 layer ResNet model consisiting of 1.723M parameters.

We employ CIFAR-10 and CIFAR-100 [94] as dataset benchmarks. They are well-established computer vision datasets used for object recognition. They are particularly useful as a proof of concept for new ideas and practical because they allow to explore the minimum network requirements for such tasks. CIFAR-10 consists of 60,000 32x32 RGB images containing one of 10 object classes, with 6,000 images per class. 50,000 images are used for training and the remaining 10,000 are used for testing. In the case of CIFAR-100 [94] the 60,000 images are divided into 100 different classes.

3.4.1 Uniform compression

The experiments summarized in Table 3.1 show the effectiveness of the PCA compression : for various energy thresholds ($E(\%)$, both the compressed network accuracy, as well as parameter count are measured. To note that, for each column, the energy threshold is applied uniformely throughout the network's layers : the set of original filters X is replaced by its reconstructed version $\hat{X}$. CNN accuracy is then measured for CIFAR-10 and CIFAR-100. Table 3.1 shows that a compression rate of 1.94 times can be achieved with only a drop of 0.81% of accuracy (CIFAR-10/VGG ; E(%)=70). The detailed analysis of these numbers helps to improve the method as will be shown in the next sections.

Table 3.1 – Multiple trade-off points showing the effect of replacing all layers in various networks by their approximation, for different thresholds and datasets. The following stand for : E : Energy threshold, P : Parameters, CR : Compression Rate, A : Accuracy, RE : Relative Error. 100% energy represents the baseline model.

	Net	E(%)	100	90	80	70	60
CIFAR-10	VGG-inspired	P(M)	7.331	6.485	4.965	**3.778**	2.821
		CR(x)	1.00	1.13	1.48	**1.94**	2.60
		A(%)	92.20	92.31	92.27	**91.45**	90.21
		RE(%)	0.0	0.0	0.0	**0.81**	2.16
	ResNet-110	P(M)	1.723	1.548	**1.26**	1.028	0.825
		CR(x)	1.00	1.11	**1.37**	1.68	2.09
		A(%)	92.91	92.61	**91.48**	89.75	86.44
		RE(%)	0.0	0.32	**1.54**	3.40	6.96
	ResNet-32	P(M)	0.453	**0.423**	0.35	0.288	0.232
		CR(x)	1.00	**1.07**	1.29	1.57	1.95
		A(%)	92.04	**91.29**	89.51	87.12	81.02
		RE(%)	0.0	**0.81**	2.75	5.75	11.97
CIFAR-100	ResNet-110	P(M)	1.723	1.585	**1.318**	1.09	0.894
		CR(x)	1.00	1.09	**1.31**	1.58	1.93
		A(%)	71.3	68.3	**55.1**	32.2	8.4
		RE(%)	0.0	4.21	**22.72**	54.84	88.22
	ResNet-32	P(M)	0.453	**0.435**	0.367	0.308	0.255
		CR(x)	1.00	**1.04**	1.23	1.47	1.78
		A(%)	68.1	**56.9**	29.1	4.2	1.6
		RE(%)	0.0	**16.45**	57.27	93.83	97.65

From these results it is clear that the size of the initial model impacts the compression rate, as smaller models such as ResNet-32 are more difficult to compress, and show a faster accuracy degradation than larger models such as VGG. This can be seen when comparing ResNet-32 at 90% energy and VGG at 70% energy, for which both have the same relative error (-0.81%) but compression rates are 1.07 and 1.94 respectively. Furthermore, when the compression rate increases, accuracy degrades faster in the case of a more compact model such as ResNet-32 than in the case of VGG.

The complexity of the task (i.e. the number of classes) also seems to impact the accuracy degradation, as can be seen when comparing between a 100-way and 10-way classification. Indeed, for the same energy thresholds, relative error is higher and compression rate is lower in the case of CIFAR-100 as compared to the same networks but applied for CIFAR-10 classification.

As a summary, the results above show that PCA compression is an effective way to identify important parameters within the network and allows for a flexible approach to find trade-offs between model size and performance. Improvements to better tune this trade-off are

discussed in the next sections.

3.4.2 Progressive compression

An explanation to the rapid decay of accuracy of some of the tested networks is that PCA compression does not induce the same error at each layer for the same energy threshold and the impact of these errors on the network accuracy also depends on the layer. Some experiments highlight this last phenomenon and show that the position of the layer in the network greatly impacts the CNN accuracy, as shown in Figure 3.6. In this figure, PCA is progressively and cumulatively applied to each of the 32 layers on the ResNet-32 network on CIFAR-10 with a 70% energy threshold. Along the curve *'All layers'*, the first point corresponds to the original network, the second point shows the effect of applying PCA solely to the first layer of the network (all other layers remain unchanged), the third point shows the effect of applying PCA to only the first layer and the second layer, etc... At last, the rightmost point of the curve shows the trade-off between compression rate and network accuracy when PCA is applied to all of the network's layers.

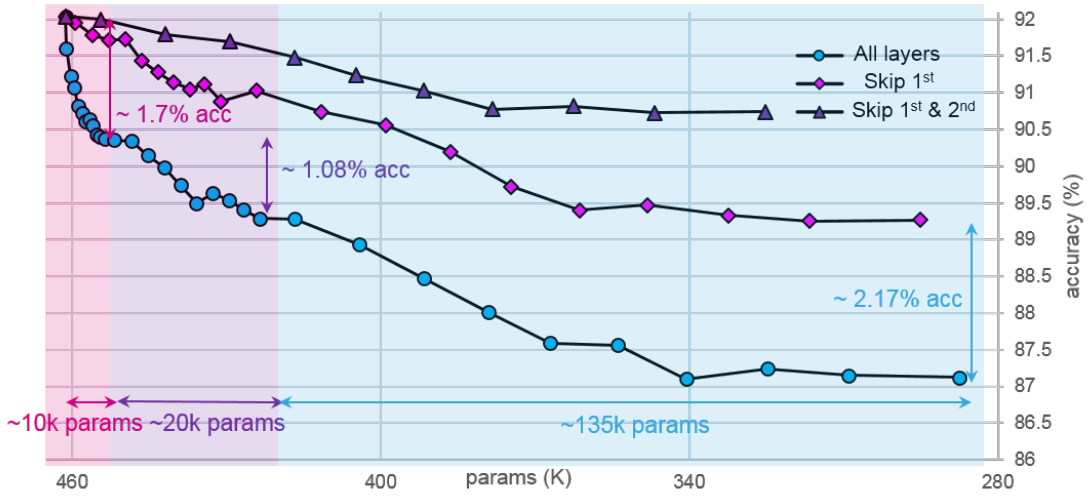


Figure 3.6 – Effect of cumulatively applying PCA to different layers of a ResNet-32 network on CIFAR-10.

From Figure 3.6, the curve *'All layers'* shows that the last layers of the network are more robust to the PCA transformation : applying PCA to the last 10 layers leads to a ∼135k parameter reduction with a ∼2.17% accuracy loss, while applying PCA to the first 10 layers in the network leads to a ∼10k parameter reduction with a ∼1.7% accuracy loss. This could possibly be due to the accumulation of errors, since approximation errors in the first layers propagate through a larger number of layers. Also, the number of parameters in the last layers of the network is usually higher (16 filters in the first layers vs. 64 in the last in the case of ResNet), leaving more room for compression.

This assumption is reinforced by the two other curves obtained by compressing layers in different orders and combinations. For example, the curve *'Skip* 1^{st}*'* from Figure 3.6 shows the result obtained by skipping the first block of 10 layers in the network : only the last two blocks of 20 layers are compressed, the first 10 layers are not compressed. In the same way, the curve *'Skip* 1^{st} *&* 2^{nd}*'* shows the result of skipping the first two blocks of 20 layers and

only applying PCA to the last block of 10 layers. It can be observed that accuracies of these curves are higher than the curve *'All layers'* : a vertical cut shows that, for example, there is a way to obtain a network of 340k parameters with an accuracy of 91%.

These experiments show the possibility to apply different compression rates to different layers of the network in order to better control the possible trade-offs. A second finding is that after compressing a layer, it should be possible to adapt the rest of the network through an additional re-training step. More precisely, the next section shows that a light retraining (fine-tuning) allows to recover most of the accuracy, for all the tested networks and databases.

3.4.3 Trained reconstruction

In this section, we show that an additional retraining phase in the PCA basis space is an effective way to overcome the limitations observed in sections 3.4.1 & 3.4.2. More concretely, the idea is to fine-tune the set of coordinates in each layer in order to improve the accuracy, the starting point being the values computed by PCA. Experiments show that retraining allows a severe improvement in accuracy for a given compression rate with respect to direct application of PCA.

In order to achieve this, the filter reconstruction through PCA is explicitly learned by replacing the original TensorFlow [97] graph by the decomposed one, as illustrated in Figure 3.7.b. Contretely, each layer in the new network is decomposed into the PCA basis, the reconstruction coordinates and the mean. These are initialized at the same values as computed through PCA in section 3.4.1. As shown in Figure 3.7.b, only the reconstruction coordinates are trainable in order to recover from the accuracy drop, while the rest of the parameters in the network remain fixed.

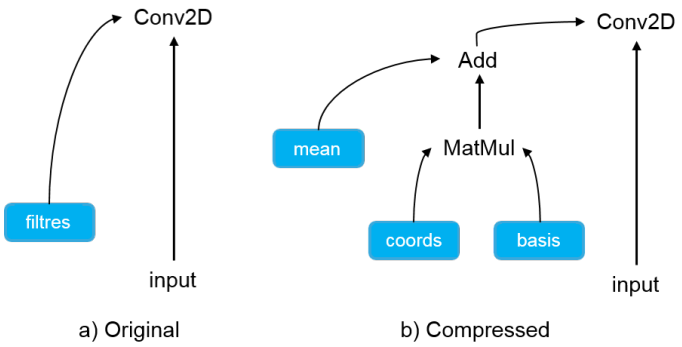
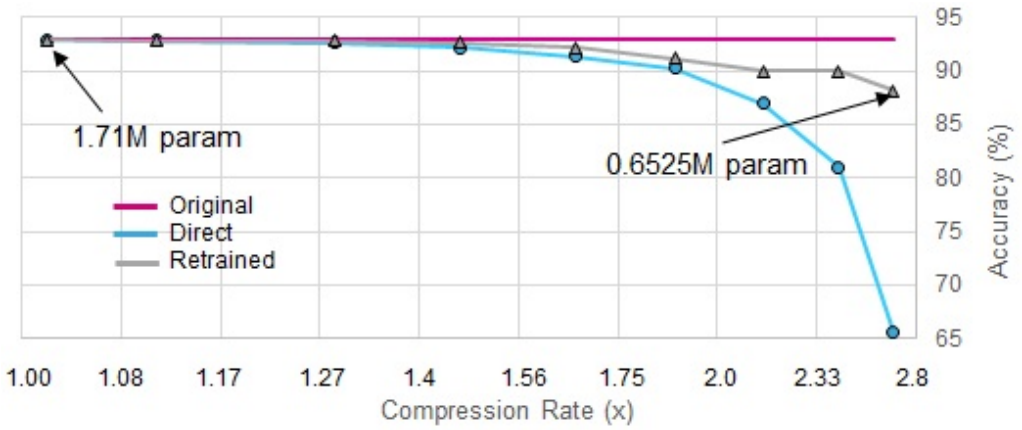


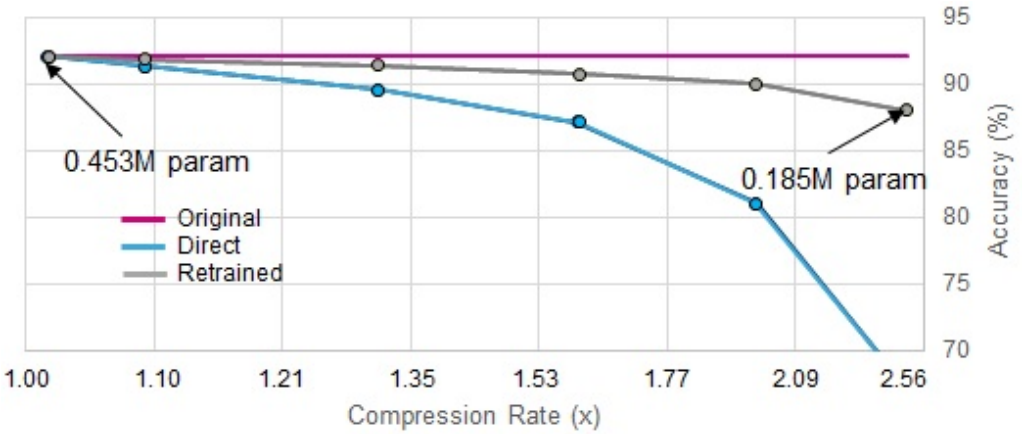
Figure 3.7 – Implementation of PCA reduction into the tensorflow graph (b) as compared to the original (a). The variables in (b) are initialized to the values previously obtained through PCA.

The method is now applied to the same networks as in section 3.4.1 and results are compared to each-other. Figures 3.8.a & 3.8.b show the effect of retraining as compared to direct PCA compression (Table 3.1) for both ResNet-110 and ResNet-32 respectively on the CIFAR-10 dataset. As an example, in Figure 3.8.a, the CNN of original size 0.46 MParams is compressed to 0.23 MParams (x2 compression rate) with 89.97% accuracy (2.07% drop), instead of 81.02% obtained by applying PCA directly.

Figure 3.8.a shows a higher robustness to PCA compression when comparing to Figure



(a) ResNet-110 on CIFAR-10



(b) ResNet-32 on CIFAR-10

Figure 3.8 – Effect of retraining as compared to directly applying PCA on two ResNet models on CIFAR-10.

3.8.b, since the initial model is larger and there is higher margin for compression. However, better final trade-offs are obtained with a more efficient model as starting point, as can be seen by comparing model sizes in Figure 3.8.b to Figure 3.8.a for a fixed accuracy level.

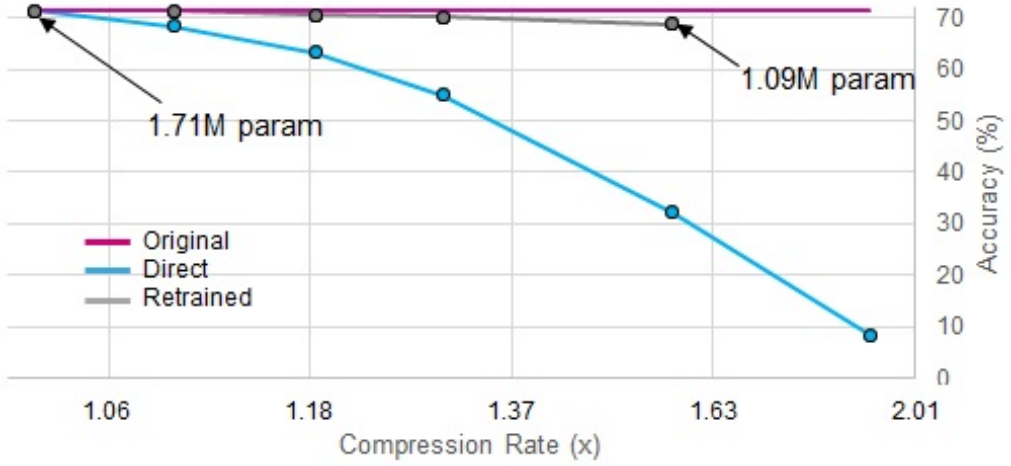
The same experiments are then performed on the CIFAR-100 dataset, resulting in Figure 3.9. These results clearly show the benefit of retraining. In the case of CIFAR-100 the retraining step is crucial in order to preserve accuracy.

Task complexity (i.e. number of output classes) still seems to be an important factor, since the same network (ResNet-32) at the same energy threshold (60) yields a x2 compression rate for a 2.07% accuracy drop in the case of CIFAR-10 as compared to a x1.78 compression rate for a 3.97% drop for CIFAR-100 after retraining.

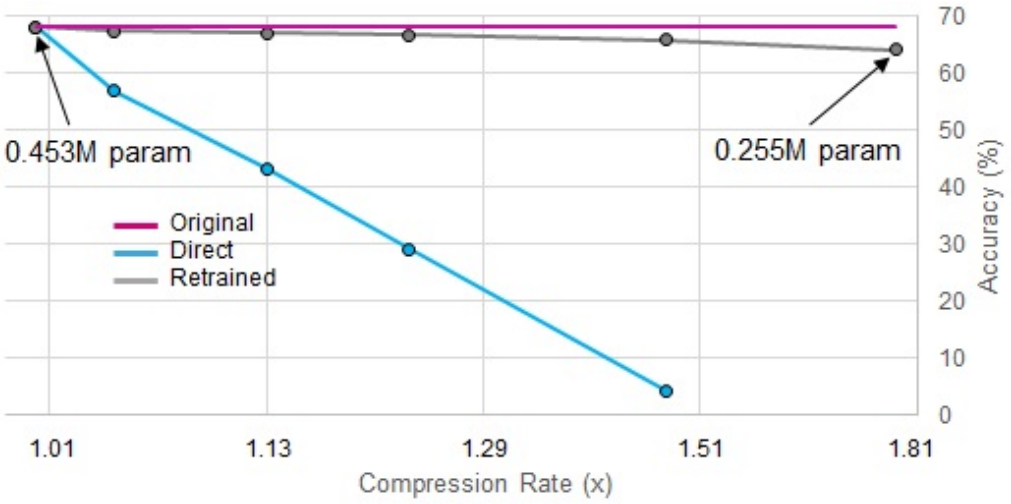
As a conclusion, the results presented in this section show that PCA is an effective method for selecting important parameters within a network, but an additional fine-tuning step is greatly beneficial and allows the network to recover most of the accuracy. Some other interesting properties we have observed throughout our experimentation are the following :

— Training convergence is obtained within a small number of epochs ($\sim$4) and only a small subset of training samples suffices to recover accuracy.

— Only a very small subset of parameters (PCA coordinates) are trainable, while the rest



(a) ResNet-110 on CIFAR-100



(b) ResNet-32 on CIFAR-100

Figure 3.9 – Effect of retraining as compared to directly applying PCA on two ResNet models on CIFAR-100.

(PCA basis) remain fixed.

— The retraining phase only produces small adjustments in these trainable parameters, preserving the initial distribution of the weights computed by PCA.

3.4.4 PCA for filter removal

Another plausible idea is that the PCA basis filters already contain all the necessary information and therefore the introduced recombination layer could be advantageously replaced by the next layer of the network. Indeed, as a theoretical hypothesis, if the network was entirely linear, then for the next layer performing a combination of the input feature map to produce a new one, it would be possible to fuse it with the previous recombination layer. But, in reality, since the recombination layer and the next filter layer are separated by the non linear activation function, this fusion becomes mathematically impossible. As training has proved effective to compensate for deviations, then it could be of interest to check that the PCA basis is sufficient. The following experiments show that this strategy can lead to

over 11x compression rate for specific layers, at negligible accuracy drop. However, applying the method to all layers in the network leads to worse results than in the case where the recombination layer is kept.

In the following experiments we completely remove the recombination layer (only keeping the basis filters) and randomly initialize the next layer, while the rest of the network's layers remain fixed. We then proceed to train the randomly initialized layer exclusively. Once some degree of convergence has been achieved, PCA is applied on the newly learned layer and the process is repeated throughout the network.

Figure 3.10 shows that every layer in the VGG-inspired network is able to learn to recombine the PCA basis layer independently. In order to generate these curves, PCA has been applied separately to each layer in the network. The energy threshold is fixed at 90% in every layer, and SGD (Stochastic Gradient Descent) hyper-parameters are constant : 0.1 learning rate, 3000 iterations, 200 batch size. In this example, the layers at the beginning and end of the network are able to almost completely recover accuracy, while layers in the middle suffer a higher accuracy degradation. In the case of the first layer, the 128 filters are replaced by the 11 first basis filters, leading to a x11.64 gain in the number of parameters, operations and feature map amount in both this layer and the following one, without any loss of accuracy. Since this layer accounts for the highest feature map data amount, the maximum storage needed for a single layer could be reduced by a factor of x11.64 without noticeable impact on the CNN accuracy.

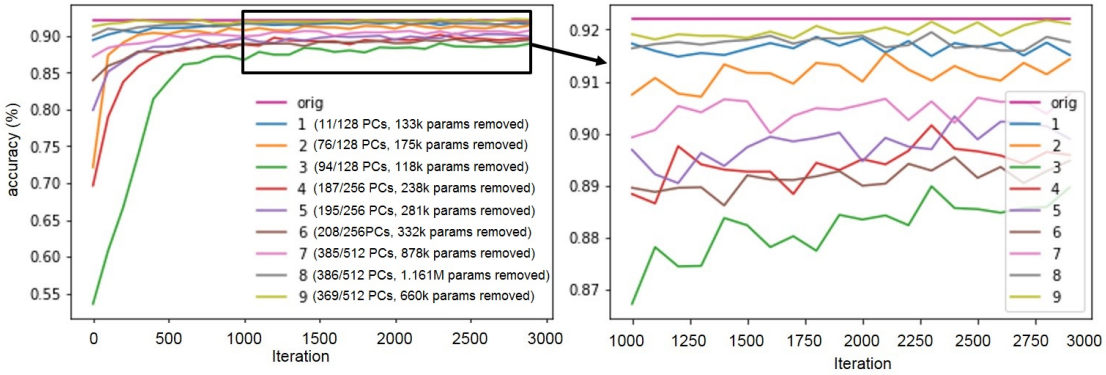


Figure 3.10 – Learning curves after replacing the filters by their PCA basis, without recombination.

3.5 PCA combined with other compression methods

From observation of Figure 3.3 we can see that, although PCA employs an optimally reduced set of parameters to approximate the set of original filters, the patterns in the filters before and after PCA reconstruction are still very similar. This means that further redundancy could potentially be exploited through other well known methods such as pruning, quantization, symmetry removals, etc... For example, if there are prunable parameters in the set of original filters in Figure 3.3, it should also be the case in the reconstructed filters. In this section we will study the combination of the PCA compression only with the most popular and promising methods. Other options could also be considered.

3.5.1 PCA combined with pruning

Since convolutional networks try to maximize response to patterns inside the learned filters (filter matching), this means that lower values are less important and can be easily removed from the network. Figure 3.11 shows a comparison between pruning and PCA as well as the result of their combination. As opposed to PCA, pruning requires the indexing of zero values and therefore memory and computation gains are not directly proportional to the number of removed parameters. As the figure shows, pruning gains are lower for more compact starting models (lower PCA thresholds). This means that the overhead mechanisms introduced for indexing can outweigh computation gains, and therefore the main interest of pruning on compact models would be memory size reductions, which can easily be exploited in hardware through run-length encoding (RLC).

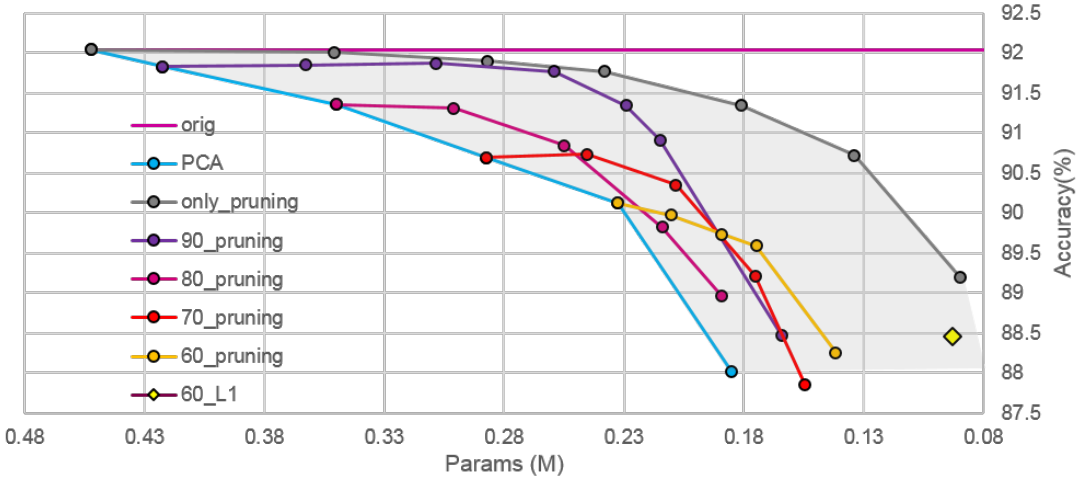


Figure 3.11 – Results of further pruning PCA compressed models.

3.5.1.1 Structured pruning

Furthermore, PCA allows a very natural way of performing structured pruning (such that it doesn't need indexing overheads). Indeed, it is possible to rank the set of recombination coefficients in each layer to then remove the least important ones, effectively removing a whole feature map channel in the current and following layers, as is displayed in Figure 3.12. For example, the point '60_L1' on Figure 3.11 shows the results of combining a PCA

decomposition and retrain with an uniform energy threshold of 60% per layer, plus a structured pruning phase on the set of coordinates, allowing to reach similar results to simple pruning, while completely removing operations from the network (instead of indexing).

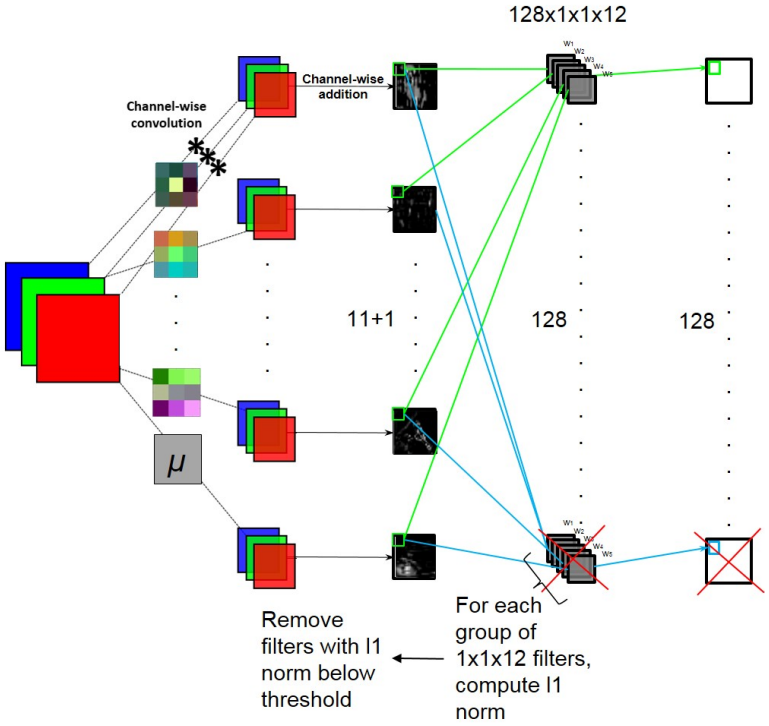


Figure 3.12 – Structured pruning through PCA.

The table in Figure 3.13 below shows two possible trade-offs in which the L1 norm threshold is modified through the parameter "range" in order to completely remove filters from the network on every layer. The impact of L1 pruning is first shown through the first "accuracy" field and then in the "retrained" field after a retraining phase has been applied. The basis filters are then also pruned in a classical way by a threshold "basis_std", and the final accuracy after retraining is reported in the lats "retrained" field of the table. This shows that L1 pruning effectively allows to completely remove a substantial number filters and feature maps from the network ; for example, 254 out of the 896 filters and fmaps can be removed after PCA, while still achieving 88.95% accuracy on CIFAR-10, while classical pruning on the basis filters allows to further reduce non-zero parameter count by a factor of x2.5 with reasonable accuracy (88.45%). More aggressive thresholds can also produce interesting results. For example, more than half of the network's filters (525 out of 896) can be removed through L1 pruning on the set of recombination coordinates with reasonable degradation in accuracy ($\sim$ 5%), while the total number of parameters can be reduced by a factor of x3.32 by pruning the set of basis filters, still achieving a 85.02% accuracy on the CIFAR-10 dataset.

# filters	gain	range	Accuracy(%)	Retrained(%)	basis_std	gain	Model Size(M)	Accuracy(%)	Retrained(%)
254/896	0.29	0.4	80.60	**88.95**	0.8	2.5	**0.093**	78.37	**88.45**
525/896	0.59	0.6	35.75	**86.34**	1	3.32	**0.070**	71.10	**85.02**

Figure 3.13 – Results of structured pruning on a PCA-compressed (60% energy threshold) ResNet-32 on CIFAR-10.

3.5.2 PCA combined with pruning and quantization

Furthermore, quantization can also be applied to a network which has been compressed by PCA and pruned. As shown in the diagram from Figure 3.14, each layer in the network is decomposed into the set of PCA coordinates and basis filters. These are then masked according to a threshold which is proportional to the standard deviation of the layer (i.e weights in the interval $[-\lambda\sigma, +\lambda\sigma]$ are removed). Quantization is performed as described in [61] for each of the masked terms. The PCA recombination is then performed and another quantization node is further applied. Results showing the combination of these methods are reported in section 3.5.3.1.

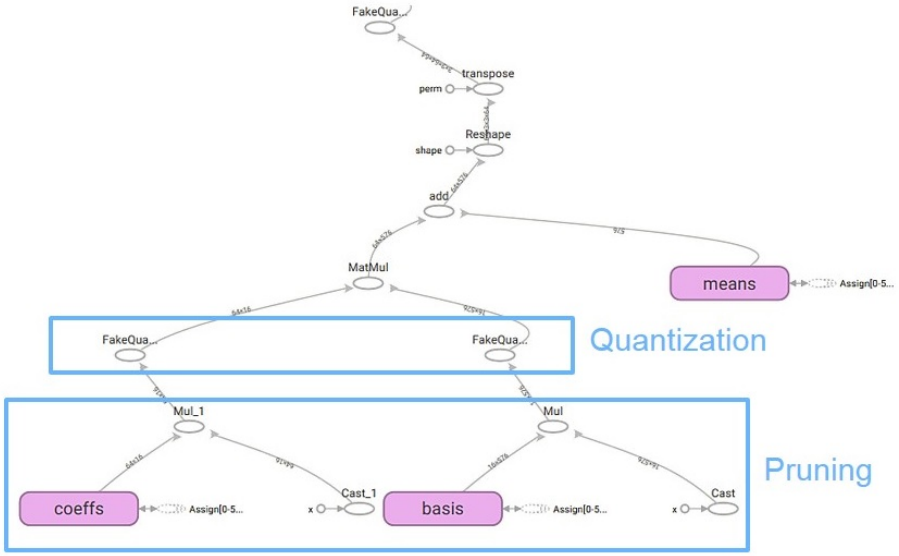


Figure 3.14 – Tensorflow graph of the layer of a network which has been decomposed by PCA, pruned and quantized.

3.5.3 PCA combined with winograd

Since both PCA and Winograd are linear transforms, it is also possible to apply Winograd to the set of basis filters and then recombine the results from the set of PCA coordinates :

$$T_w(\hat{X}) = G\hat{X}G^T = G[\sum_{k=0}^{L-1} \lambda_k^i \mathbf{v}_k + \mu]G^T = \sum_{k=0}^{L-1} \lambda_k^i [G\mathbf{v}_k G^T] + G\mu G^T = \sum_{k=0}^{L-1} \lambda_k^i [T_w(\mathbf{v}_k)] + T_w(\mu),$$

where :

— G is the winograd transformation matrix as defined in [89].

— $\mathbf{v}_k$ is the set of PCA basis filters, and λ_k^i is the set of PCA coordinates, as explained in section 3.3.

Intuitively, we can look at the PCA decomposition as two independent layers, i.e. a basis layer followed by a recombination layer, as was shown in Figure 3.5. This means that the winograd transformation could be applied to every basis layer independently, as would be done with any other network.

3.5.3.1 Example with ResNet110 on CIFAR-10

As a starting point, we take the PCA compressed model from Figure 3.8.a which has been reduced from 1.71M parameters to 0.6525M by applying a uniform 30% energy threshold to every layer, while producing an accuracy drop of 3.67%. We further apply a pruning stage with different thresholds, as is shown in Figure 3.15. During the pruning stage, we choose the point providing a $\sim$ x1.7 parameter gain ($\sim$ 370K params, 87.34 % accuracy). The network is then retrained to reach 88.78% accuracy.

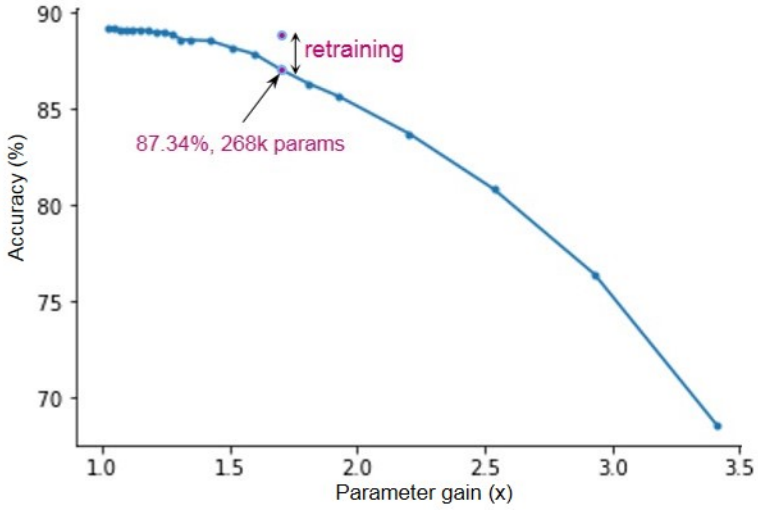


Figure 3.15 – Effect of further pruning the set of PCA basis and coordinates without retraining.

The network is then quantized to 8-bits with an accuracy of 88.42%, as shown in Figure 3.16. It is then retrained to achieve 88.94% accuracy. Cumulatively, the network is compressed from 6.88Mbytes originally to around 370Kbytes (x19 memory gain for a 3.9% accuracy loss). Further applying winograd across each layer of the network would provide an additional x2.25 theorical speed-up, however care must be taken in order to make it compatible with pruning, as was mentioned in section 3.1.0.2.

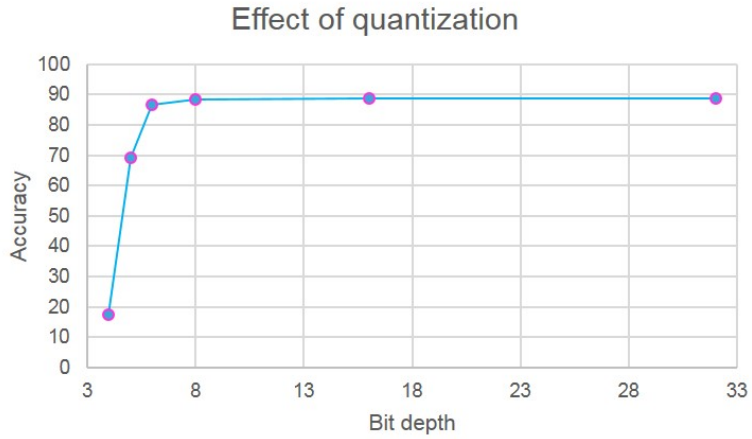


Figure 3.16 – Effect of further quantizing the set of pruned PCA basis and coordinates without retraining.

3.5.4 PCA combined with binary quantization

Furthermore, more extreme quantization schemes can be employed, such as the one described in [66]. The goal is to further decompose a set of filters, in this case the PCA basis filters X_{basis} into a linear combination of a set of binary basis filters, B_m, and a set of coefficients, α_m. That is, we are trying to approximate $X_{basis} \in \mathbb{R}^{N\times H_f\times W_f\times C}$, through $X_{basis} \approx \alpha_1 B_1 + \alpha_2 B_2 + ... + \alpha_m B_m$, where $B_1, B_2, ..., B_m \in \{-1,1\}^{N\times H_f\times W_f\times C}$ and $\alpha_1, \alpha_2, ..., \alpha_m \in \mathbb{R}$.

We therefore need to solve the following optimization problem :

$$\min_{\alpha,B} \|X_{basis} - \sum_{k=0}^{m} \alpha_i B_k\|^2, \text{ such that } B_k(i,j) \in \{-1,1\}.$$

In order to allow the network to learn these coefficients through backpropagation, the $B_k's$ on every layer are initially fixed as $B_k = sign[X_{basis} - \text{mean}(X_{basis}) + u_k \text{std}(X_{basis})]$, where u_k is a learnable 'shift parameter' allowing to better approximate the gaussian distributions of the original set of weights, X_{basis}.

This idea was applied to the PCA compressed network from Figure 3.8 at 60% energy threshold ($\sim$ 0.226M parameters, $\sim$ 90.1% accuracy) without loss of accuracy by choosing a basis consisting of 5 terms, i.e m=5. Memory gains are therefore equivalent to that of 5-bit compression. Computations are also greatly simplified since all convolutions now involve uniquely -1 and +1 filter weights and therefore multiplication is no longer necessary, and only addition and subtraction operations are required. Although it was not thoroughly explored (because of time constraints), applying the same approach to the activations (in order to achieve convolution through bitwise operations) produced heavy accuracy degradation which was not recoverable during training.

3.6 Conclusion

In this chapter, the most interesting SOA methods allowing to reduce computational complexity of CNNs have been reviewed. The main advantages and inconveniences of the most promising methods have been discussed, as well as the potential compatibility between the different approaches.

A CNN compression method was then proposed. This method is based on Principal Component Analysis (PCA), which allows for flexible trade-offs between model accuracy, number of parameters and number of operations. For each layer of the network, PCA allows us to replace learned filters by a more appropriate set of basis filters. This basis is hierarchical, which allows to remove an arbitrary amount of basis filters from the layer in order to approximate the original set of filters, without great impact on accuracy. However, the network's accuracy suffers from error accumulation from applying this transformation across every layer of the network. Multiple solutions to this problem are proposed in the chapter, the most succesful one being a retraining phase in which the set of PCA basis filters remains fixed, while the PCA coordinates undergo a slight fine-tuning phase.For example, this PCA compression method enables to reduce by a factor of $\times 2$ the computational complexity of a ResNet-32, both in the number of parameters and operations, with an accuracy drop below

2% .

The method is then evaluated on multiple datasets and network architectures. We also propose different variants of the algorithm, and show the compatibility of the proposed method with multiple compression and speed-up techniques from the state-of-the-art, notably pruning, quantization and winograd. For example, by the combination of such methods, a ResNet-110 is reduced from 6.88Mbytes to 370kbytes, i.e. a x19 memory gain with a 3.9 % accuracy loss.

3.7 Perspectives

Although the proposed method provides a more theorical apporach to fixing the CNN's architecture, it is not trivial to choose the approximation error of PCA for each layer. While retraining has proven to be an effective solution to this issue, it is still a very empirical and hard to explain solution to the problem.

In this sense, the work in [98] experimentally shows that the total MSE in the last layer is additive. That means that, the total accumulated output error of a network for which each individual layer's weight and activations have been approximated is the sum of the approximation error in each layer. This allows the authors to formulate a framework that adjusts the quantization bit-depths across the layers of weights and activations in order to minimize the total output error of the neural network. This is done by finding the *Pareto-optimum* bit allocation by using a *Lagrangian formulation*. By employing this strategy, they are able to optimally allocate the bit-depth of both weights and activations of each layer without need for retraining.

Since PCA is an optimal way to trade-off parameters for reconstruction MSE within a layer, this framework could be directly applied in our case to optimally allocate the number of principal components throughout all layers of the network. This would give a more principled way to to fix the network's computational complexity, as well as removing the need for the retraining phase. Although this seems very promising, due to time constraints, it was not possible to explore the combination of both these methods.

4

Cascaded and compressed CNNs for fast and lightweight face detection

In this chapter, we validate the method outlined in Chapter 2 in a face detection applicative scenario in order to fully co-optimize detection performance and computational cost of the solution. We start by providing a review of the state-of-the-art face detection solutions, and explaining their main issues when targeting an efficient implementation. Results supporting the main claims in Chapter 2 are then discussed. The face detector is finally compared to both a traditional computer vision algorithm and a state-of-the-art CNN.

Sommaire

2.1 Computational Complexity of object detection 41
2.2 State of the art in object detection 43
2.2.1 RCNN 43
2.2.2 SPPNet 43
2.2.3 Faster RCNN 44
2.2.4 YOLO & SSD 44
2.3 Proposed methodology to optimize CNN object detector complexity for applicative use-case 45
2.3.1 Applicative use-case specifications 48
2.3.2 Proposal Network definition and training 50
2.3.3 Proposal Network compression 55
2.3.4 Binary Classification Network definition and training 57
2.3.5 Classification Network compression 58
2.3.6 Evaluation 59
2.3.7 Algorithm benchmarking 59

2.3.8 Hardware evaluation 60
2.3.9 Conclusion and Perspectives 61

4.1 State of the art in face detection

Face detection is a specific case of object detection, and therefore the same issues as the ones discussed in Chapter 2 are encountered. Concretely, the main difficulties involve the high variability in the appearance of human faces and the extensive search space of possible face locations and sizes over full images.

Moreover, the main interest in face detection in an applicative context is the ability to process the detection region afterwards. This allows to obtain additional information, such as identity. This greatly reduces the task's complexity, since it allows to exclude occluded, out-of-focus and smaller faces, this last being the biggest dimensioning factor, as was discussed in section 2.1.

Indeed, as is shown in Fig. 4.1, face size is a crucial dimensioning factor, since the search space increases as objects get further away from the sensor. This translates into an increase of the number positions to be analyzed, and therefore requires higher computational budgets.

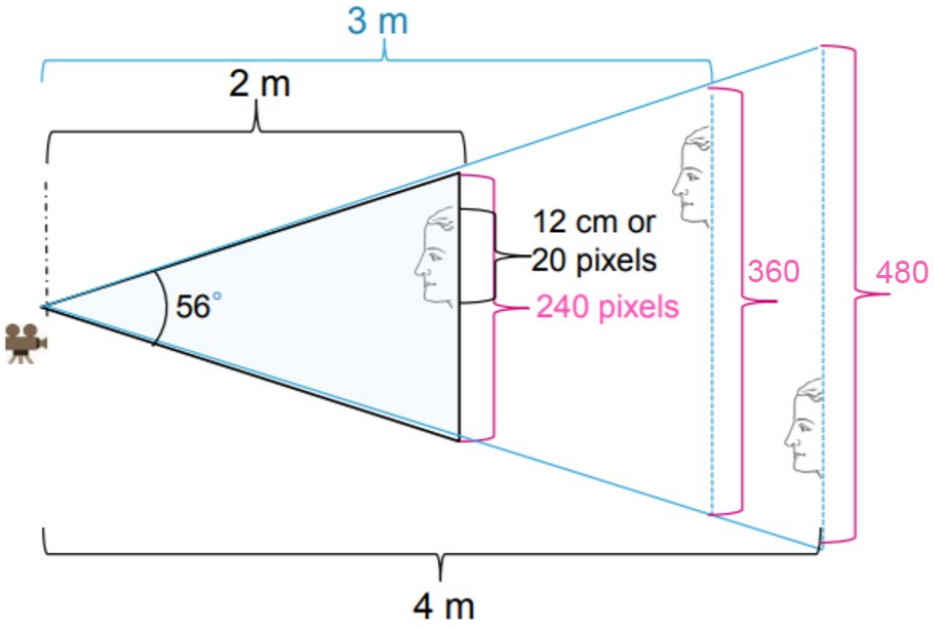


Figure 4.1 – Growth of the search space as objects get further away from the sensor. Detecting an average sized face ($>$ 12cm) 4 meters away from a sensor with a 56° FoV would require scanning 20 pixel windows inside a 480 pixel image.

This section outlines the main solutions present in the state-of-the-art of face detection, with special attention to computational complexity and detection performance.

4.1.1 Viola-Jones

One of the earliest and most popular real-time solutions to face detection was the Viola-Jones (VJ) algorithm [99], which is based on a cascade of simple Haar-like features (generated through Haar wavelet basis functions). These features are depicted in Figure 4.2.a. Haar features are also well suited for a hardware implementation, as they are simple and fast to evaluate. Furthermore, these features are evaluated in a cascaded fashion, allowing to

discard true negative regions at early stages, while letting through potential true positives to more discriminative "boosted" [100] features.

Further improvements such as representing the original image by an "integral image" lead to very efficient HW implementations with good detection performance on frontal faces. However, detection performance quickly degrades in an uncontrolled environment (varied poses, lighting or face expressions). From early times, CNNs have been known to be a more robust solution [101], but at the cost of increased complexity.

Interestingly, some of the features learned on the first layer of a Convolutional Neural Network trained for face detection (Figure 4.2.c) closely resemble the set of basic Haar features (Figure 4.2.a). As is shown in Figure 4.2.b, this is also the case when a PCA basis is computed on a set of centered facial crops, i.e. the same algorithm as in section 3.2.1, but now applied to a set of facial crops as opposed to the set of filters in a layer. Indeed, from Figure 4.2, we can see that most filters learned by the CNN are also present on the PCA basis, while the set of haar features resemble a quantized version of the PCA basic elements.

However, even though these methods have all very strong similarities, the ability of the CNN to learn more invariant features through mechanisms such as bounding box regression (explained in section 2.3.2.1), clearly makes it a more advantageous solution. For example, bounding box regression allows the proposal network to perform a coarser search, in terms of number of pyramid rescales and strides, as will be shown later in this chapter.

On the other hand, methods such as VJ and PCA would need to convolve each of the basis' patterns (i.e. haar/PCA set of basis functions) with the input image accross multiple locations and scales in order to exactly match the pattern. In other words, in order to find matches against an N×N basis term, we would need to match every possible N×N region of the image against the basis term, at every possible scale too.

Moreover, although some invariance can be gained from the training database, the main mechanism to gain scale invariance for both these algorithms would be a multi-resolution analysis such as the one displayed in Figure 2.2. The ability of CNNs to gain scale invariance through bounding box regression is therefore not only a great advantage in the number of operations of the solution, but also in the number of generated pyramid levels. This has great impact on the hardware dimensionment, and will be discussed in latter sections of this chapter by comparing detection rates of different methods for similar computational budgets.

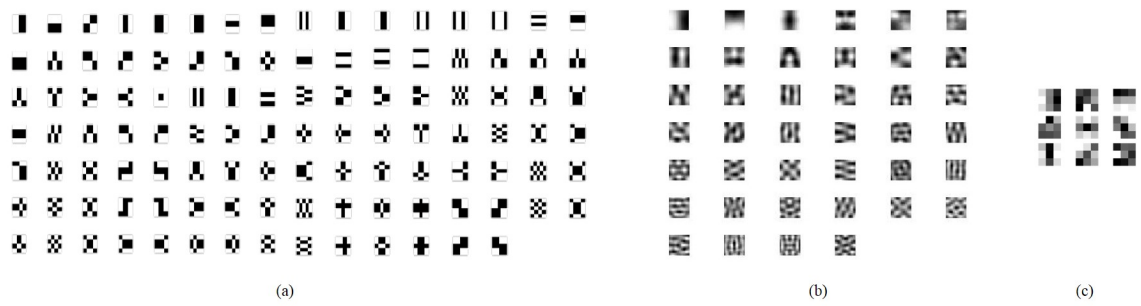


Figure 4.2 – (a) Haar basic functions employed in VJ. (b) PCA basic functions obtained on a set of centered face crops. (c) Filters obtained in the first layer of the CNN solution presented in this chapter.

4.1.2 CNN Cascade

Recently, the work in [102] proposes a CNN face detection solution with the same philosophy as in Viola-Jones, i.e. rejecting true negatives early on. For this, they use a cascade of three increasing complexity networks, allowing to run at 14 FPS on an Intel Xeon E5 CPU, as shown in Table 4.1.

However, two major drawbacks of this solution are both a dense scanning of the input image across multiple scales and positions, as well as a large model size ($>$ 50 MBytes at 32-bit, mainly due to fully-connected layers). For example, densely scanning an image for 12x12 faces inside a QVGA image with their minimal 16-filter network would take 3.8G MAC, while bigger faces such as 40x40 in QVGA would take approximately 249M MAC. All intermediary sizes would also need to be taken into account.

Therefore, this clearly shows that even minimal CNN architectures can get quite expensive when performing object detection in a classical way, and therefore it is necessary to explore more efficient solutions.

4.1.3 MTCNN

In this scope, MTCNN [103] applies the same cascaded approach as in [102] but with fully convolutional networks that are also able to perform bounding box regression. The paper reports 100fps on a NVIDIA Titan Black GPU when searching for 20x20 faces inside VGA input images, with SOA results in detection performance for multiple datasets. However, model size is also large with 1.89 MBytes (32-bit).

4.1.4 LightweightCNN

Finally, the work in [104] provides a binary face classifier consisting of 76,375 parameters and similar detection rates to MTCNN. However, it is only able to run at 31 FPS when searching for 64x64 faces in VGA input on a GTX 1080 GPU. Nevertheless, this shows that binary classification can be achieved very accurately with a reduced number of parameters, providing a good upper-bound on model complexity.

Table 4.1 – Comparison of state-of-the-art face detectors.

Model	Model Size	Min. face size	Perf.	Target
Cascade [102]	$>$50 MB	80x80 [VGA]	14fps	Xeon E5 [CPU]
MTCNN [103]	1.89 MB	20x20 [VGA]	100fps	titan black [GPU]
LW [104]	305.5 KB	64x64 [VGA]	31fps	GTX 1080 [GPU]
LFFD (Ours)	29.3 KB	20x20 [VGA]	$>$100fps	titan black [GPU]

4.2 Fast and Lightweight Face Detection through CNNs

In this section we will leverage the concepts outlined in Chapter 2 in order to define a CNN solution for an applicative scenario of face detection with hardware considerations in mind. This leads us to a state-of-the-art face detection solution both in terms of number of parameters (29.3k, 8-bit) and operations, as shown in Table 4.1. The proposed solution, LFFD (Lightweight & Fast Face Detector) is then bench-marked against the other solutions reviewed in section 4.1. Detection rates are on-par with MTCNN when evaluated on a database which is representative of out use-case scenario, while requiring a $\times 65$ smaller model size, and noticeably less operations.

In this chapter, we apply the general methodology presented in Chapter 2 to the specific problem of face detection. In section 4.2.1 we start by thoroughly describing the networks composing the solution, as well as intermediary processing steps.

Then, each of the networks is analyzed individually, in order to identify important dimensioning network design choices, as overviewed in Chapter 2. This is first done for the Proposal Network in section 4.2.2, and then for the Binary Classification Network in section 4.2.3.

Finally, detection performance is compared in section 4.2.4 across multiple SOA face detection solutions on common datasets, and with fixed computational budgets (in terms of number of operations).

4.2.1 Proposed solution

The face detection process is divided into two main phases : a region proposal phase followed by a binary classification phase, as was detailed by the diagram in Figure 2.9 in section 2.3.

First, a very compact proposal network (PN) is derived from [103]. This network estimates the presence or absence of a face inside each 12x12 region of the input image, and also outputs a bounding box prediction for the regions containing a face. Its architecture and computational complexity are shown in Table 4.2. The goal of this proposal network is to discard true negative regions in the image, while maximizing recall i.e. finding relevant instances, in this case regions where there is a face. The PN needs to have a very low computational cost (around 55K MAC per 12x12 window), since it has to be applied across all positions and scales of the input.

Table 4.2 – Proposal network architecture.

Layer	Kernel	Filters	Input	Output	Params	MAC
conv1	3x3	10	hxwx1	$(h-2)$x$(w-2)$x10	90	$(h-2)$x$(w-2)$x10x3x3x1
max	2x2	stride=2	$(h-2)$x$(w-2)$x10	$\frac{(h-2)}{2}$x$\frac{(w-2)}{2}$x10		
conv2	3x3	16	$\frac{(h-2)}{2}$x$\frac{(w-2)}{2}$x10	$(\frac{(h-2)}{2}-2)$x$(\frac{(w-2)}{2}-2)$x16	1440	$(\frac{(h-2)}{2}-2)$x$(\frac{(w-2)}{2}-2)$x16x3x3x10
conv3	3x3	32	$(\frac{(h-2)}{2}-2)$x$(\frac{(w-2)}{2}-2)$x16	$(\frac{(h-2)}{2}-4)$x$(\frac{(w-2)}{2}-4)$x32	4608	$(\frac{(h-2)}{2}-4)$x$(\frac{(w-2)}{2}-4)$x32x3x3x16
conv4	1x1	6	$(\frac{(h-2)}{2}-4)$x$(\frac{(w-2)}{2}-4)$x32	$(\frac{(h-2)}{2}-4)$x$(\frac{(w-2)}{2}-4)$x6	192	$(\frac{(h-2)}{2}-4)$x$(\frac{(w-2)}{2}-4)$x32x3x3x6

As will be shown in latter sections, by training the network to predict both the location and size of faces through bounding box regression (Figure 2.14), the stride and re-scaling factor of the search space pyramid can be increased. This leads to a less exhaustive search space

since the PN is able to propose face sizes at neighbor scales from the pyramid. Therefore, a coarse multi-resolution pyramid can be employed. This dramatically reduces the number of operations, as it is no longer necessary to apply a complex classification network across all positions and scales of the input image.

Finally, a very precise binary classification network (BCN), derived from [104], and shown in Table 4.3 is applied to the regions selected by the PN. The BCN confirms whether or not a face is indeed contained inside each of the porposals generated by the PN. The BCN therefore needs to deliver the highest possible true positive rate (TPR) while minimizing false positive rate (FPR) on the set of proposals generated by the PN. Since this network outweights the PN both in the total number of parameters and operations, it imperatively needs to be optimized to find good trade-offs between model size, TPR and FPR.

Table 4.3 – Binary classification network architecture.

Layer	Kernel	Filters	Input	Output	Parameters	MAC
conv1	3x3	24	32x32x1	30x30x24	216	0.194M
conv2	4x4	24	30x30x24	28x28x24	9216	1.806M
max	2x2	str=2	28x28x24	14x14x24		
conv3	4x4	32	14x14x24	11x11x32	12288	1.486M
conv4	4x4	48	11x11x32	8x8x48	24576	1.572M
conv5	4x4	32	8x8x48	5x5x32	24576	0.614M
conv6	4x4	16	5x5x32	3x3x16	4608	0.041M
conv7	3x3	2	3x3x16	1x1x2	288	288

Figure 4.3 shows a full diagram of the proposed solution. For each 12x12 window, at each level of the coarse pyramid, the PN outputs a feature map with depth of 6, representing face & non-face probability as well as bounding box regression coordinates : left, top, height and width. Proposals having a face probability above a certain threshold are kept, non maximum suppression (NMS) is performed and the proposed bounding boxes are scaled back to the original resolution. NMS is again performed on the whole set of proposals and bonding box regression is then applied in order to compute the final proposals. Because the BCN processes 32x32 inputs, the retained proposed bounding boxes are cropped from the original image, resized to 32x32 size and passed on to the BCN. Final detections are found by applying a threshold to the output of the BCN.

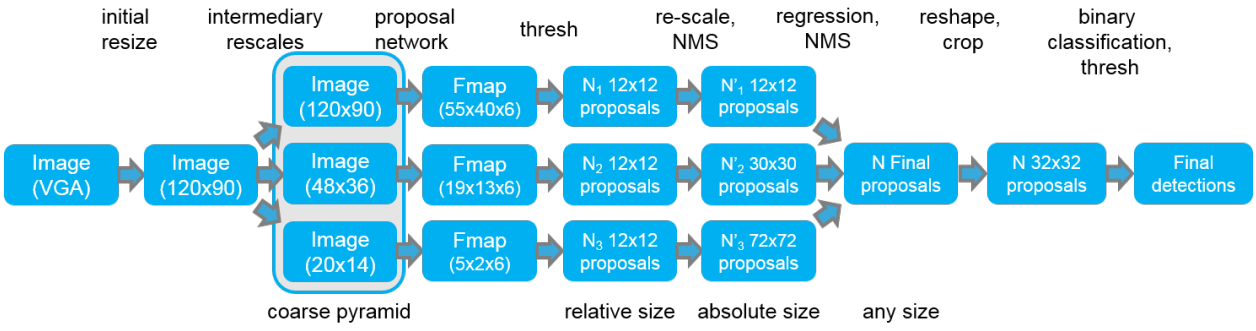


Figure 4.3 – Detailed execution diagram of the described solution.

4.2.2 Proposal Network Optimization

Even though the PN is already a minimal 4 layer network with 6.3k parameters, its input is a whole image. This means that the first layer needs to store a number of temporary data proportional to the size of the input image and the number of input and output channels, as explained in section 2.3.2.1. These numbers can quickly grow with high resolution inputs. Concretely, the size of the memory required to store all these values is given by the following : $[H_{im} \times W_{im} \times C_{im}] + [H_{fmap} \times W_{fmap} \times N_{ker}]$, where :

— H_{im}, W_{im}, C_{im} are the height, width and number of channels of the input image.

— H_{fmap}, W_{fmap} are the height and width of the feature maps.

— N_{ker} is the number of filters in the layer (same as number of output feature maps).

By restraining the application scenario to the detection of larger faces exclusively, it is possible to downscale the original input image in order to reduce the number of operations and temporary values. For example, detecting 64x64 faces inside a VGA input is equivalent to resizing the input image to 120x90 resolution, while searching for 12x12 faces in that image. This implies a reduction in the number of operations and temporary values of the order of x5.33. Furthermore, working with grayscale inputs allows for an additional x3 gain in temporary data reduction, while an additional x2 gain is obtained by using max pooling right after the first layer. Further applying 8-bit quantization to these values allows for an additional x4 memory gain. The 6.3k weight parameters of the network are also quantized to 8-bits.

Therefore, for the PN, the maximum number of intermediate values to be stored in a single layer throughout the network is $120*90+\frac{120-2}{2}\text{x}\frac{90-2}{2}\text{x}10 = 36,760$ bytes in this applicative scenario for which only 64x64 faces in VGA need to be detected.

Finally, the original network employs a parametric ReLU activation function. This is good since some works [105] have shown that the use of the ReLU function introduces symmetries in the network's filters which can be avoided by this sort of parametric ReLU activation function. However, this comes at the cost of increased number of parameters and complexifies the HW implementation of the network. In our experiments for this concrete case, we have been able to replace the activations in every layer with ReLU without loss of accuracy thanks to a slight re-training phase, in a similar spirit as [78]. This also eases HW implementation and increases feature-map sparsity. This last point could be further exploited by considering HW support for zero skipping, as is done in [106].

4.2.2.1 Proposal/recall trade-off

There is a close relationship between the average number of proposals generated by the PN, its recall, and the total computational cost of the solution(PN + BCN). As shown in Table 4.3, 5.72M MAC are required to run a single proposal through the BCN. Indeed, the total computational cost can quickly become very expensive if a large number of proposals are generated by the PN.

Therefore, the PN needs to be trained to maximize the recall while minimizing the number of proposals. Since the cost of generating proposals with the PN can greatly increase when a finer pyramid is employed, a trade-off between the number of pyramid levels, recall and average number of proposals needs to be determined. As was outlined in sections 2.3.2.1

& 2.3.2.4 of Chapter 2, bounding box regression plays a very important role in improving these trade-offs.

Table 4.4 shows the computational cost of searching for various face sizes with the PN. For example, searching for 12x12 faces in a 120x90 input image takes ∼18.32M MAC, while looking for 12x12 faces at a 16x12 resolution requires only ∼55K MAC.

Furthermore, the coarse pyramid proposed for our applicative scenario in Figure 4.3 employs a 0.4 scaling factor, which generates the following resolutions : 120x90, 49x36, 20x15. From Table 4.4, this amounts to a total computational cost of $18.32 + 2.29 + 0.15 \approx 20.76$M MAC, while a 0.9 scaling factor would require a total of 45M MAC.

Table 4.4 – Cost of proposals as a function of absolute face size (relative to 120x90 input).

Abs. Face size	12	15	19	23	29	37	46	57	72	89
Resized Resolution	120x90	96x72	76x57	61x46	49x36	39x29	31x23	25x19	20x15	16x12
MAC	18.32M	11.18M	6.73M	3.98M	2.29M	1.28M	680K	337K	150K	55K

Thanks to bounding box regression, the network has the ability to guess faces at neighboring scales and locations and we are able to achieve a high degree of scale invariance. Indeed, Figure 4.4 proves this point, since it shows little difference in terms of recall and number of proposals between a 0.4 and 0.9 scaling factor, thereby justifying a coarse pyramid.

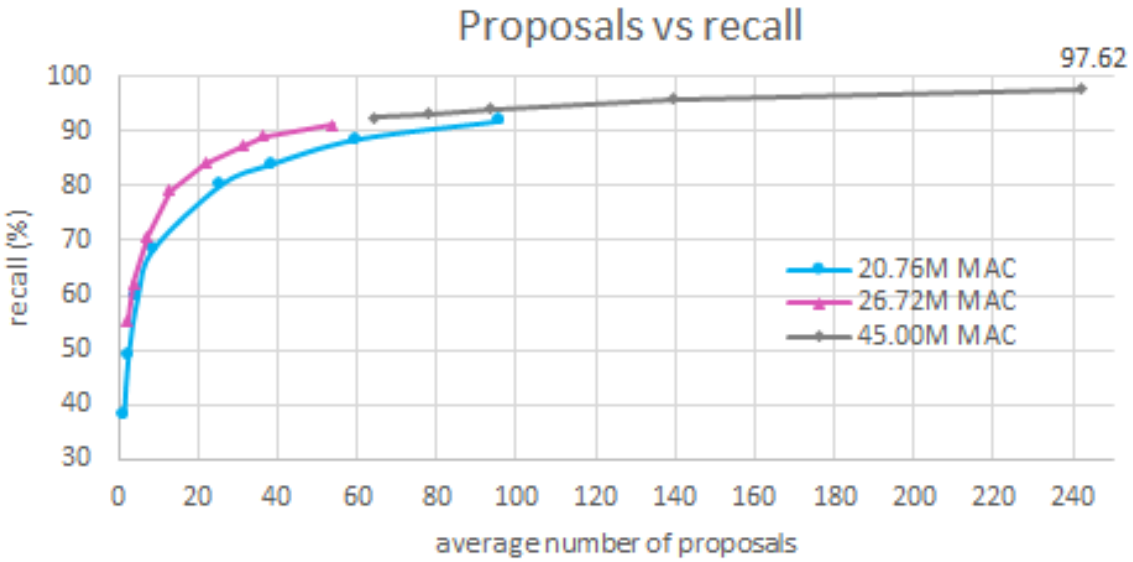


Figure 4.4 – Recall and average number of proposals for different scaling factors on 1613 validation images containing faces larger than 100x100 (∼64x64 in VGA) from the challenging WIDER FACE dataset [107].

Furthermore, recall increases when the decision threshold of the PN is lowered. However, this can dramatically increase the number of proposals to be analyzed by the BCN, and therefore the total computational cost of the solution.

For example, a recall of 56.14% is obtained in the WIDER FACE dataset with a 3-level pyramid (20.76M MAC, 0.4 scaling factor). This generates an average of 3 proposals per image, and, since a single forward pass on the BCN requires 5.72M MAC, this would mean an additional $5.72 * 3 \approx 17.16$M MAC to be computed by the BCN on average.

By employing a 4-level pyramid (26.72M MAC, 0.6 scaling factor) and lowering the de-

cision threshold we obtain a recall of 91.22% with an average of 54 proposals per image ($5.72 * 54 \approx 308.9$M MAC). These are summarized in Table 4.5.

Table 4.5 – Possible trade-offs between model complexity and performance.

Pyramid Level	Scale	PN MAC	avg. # of proposals	BCN MAC	Total MAC	Recall
3	0.4	20.76M	3	17.16M	37.92M	56.14%
5	0.6	26.72M	54	308.9M	335.62M	91.22%
10	0.9	45.00M	242	1.381G	1.381G	97.62%

Low recall rates are miss-leading due to the difficulty of samples in the WIDER FACE dataset, as was shown in Figure 2.16, from section 2.3.2.2. Indeed, we have observed that high decision thresholds tend to discard harder samples, whereas frontal faces are rarely missed. This is a positive characteristic in terms of application and therefore higher thresholds should be favored for an efficient solution.

4.2.2.2 Proposal network speed-up

From Table 4.5 it can be seen that when a higher decision threshold is chosen (first row of the table), the operation cost of the PN and BCN are balanced and therefore it becomes interesting to reduce the number of operations of the PN.

Although the PN is a very compact network with only 6.3k parameters, its input is a whole image. As can be seen in Table 4.5, this means that, although the number of operations per forward pass is constant, it is higher than in the case of the BCN (where it depends on the number of proposals). This is a good opportunity to envision other speed-up techniques which trade-off memory for computation, such as Winograd [112]. In the case of a more strict configuration (such as the last line in Table 4.5), the cost of the BCN heavily outweights the PN and therefore this kind of optimization would become less interesting.

Concretely, considering 3x3 filters, it would be possible to further reduce the number of MACs in a Winograd configuration of (2x2,3x3) by a factor of x2.25. All previously mentioned trade-off points shown in Table 4.5 and Figure 4.4 can therefore be improved by performing multiplications in the Winograd domain instead of convolutions in the temporal domain.

Moreover, the Winograd transform is applied in a layer-wise fashion to all filters of the PN as well as their corresponding input feature-maps without any loss of accuracy. Further quantizing these values to 8 bits and retraining results in less than 1% loss of recall. However, the number of proposals generated slightly increases and a retraining step could be of interest.

PCA compression could also be envisaged for this network, but since the gains would not be significative with respect to the BCN, it was not a priority and therefore not explored. Since the parameter count in the PN is low (6.3k parameters), the main interest of applying PCA would be to obtain an additional computational speed-up, more importantly than a parameter reduction.

4.2.3 Binary Classification Network optimization

As outlined in section 2.3.4, the goal of the BCN is to maximize the final TPR, while minimizing the final FPR on the set of proposals generated by the PN. For this, the BCN is trained with crops of faces from multiple datasets [107–110] as positive training samples, while false positives generated by the proposal network are used as negative training samples, as was explained in section 2.3.4.3.

These datasets have different degrees of complexity. For example, FFHQ [108] contains mostly frontal faces with considerable variation in age, ethnicity and accessories. This makes it representative of our applicative scenario and therefore, most images used for training are taken from this dataset (65k out of 80k). FDDB and AFLW [109, 110] are more challenging datasets with some partially occluded, blurred and rotated faces and only some of the samples are kept as harder training samples. WIDER-FACE [107] contains very extreme conditions and most samples are far from our scope of application.

In order to generate 275k false positives, which are used as negative samples, we run the PN through the WIDER FACE training dataset with a relatively low threshod (0.5). From these, 75k are used for training and 200k are kept for testing. Around 5k true positives are also kept in order to allow the BCN to adapt to the PN's outputs.

4.2.3.1 Binary Classification Network compression and speed-up

The architecture of the BCN is described in Table 4.3. To fit embedded system contraints, the input is also converted to grayscale and PReLU is also replaced by ReLU, exactly as was discussed in section 4.2.2.

Due to the computational cost of processing proposals by the BCN (5.72M MAC each), the total number of operations can rapidly increase and is proportional to the amount of proposals generated by the PN. The BCN also accounts for most of the weights (76.38k in the BCN vs. 6.3k in the PN), and therefore reducing its computational complexity is prioritary, since it greatly reduces the overall computational cost of the solution.

In order to trade-off computational complexity for detection rate, we apply PCA compression [111], which has been described in detail in Chapter 3. In order to do this, PCA with re-training is applied to the network with different compression rates, resulting in Figure 4.5. This figure shows the different trade-offs that can be made at a constant number of weights (params). For each of the curves, the BCN's decision threshold is varied, and FPR and TPR are recorded for each threshold. High decision thresholds lead to low FPR, while low decision thresholds lead to high TPR. This threshold needs to be fixed according to the application, in some applications such as wake-up systems, false positives are acceptable for example. As an example, this figure shows that we are able to obtain a x3.27 gain both in the number of parameters and operations with acceptable performance in terms of true and false positive rates.

Thanks to the PCA-based dimensionality reduction method proposed in [111] and discussed in detail in Chapter 3, model size is decreased by a factor of x3.27 both in the number of parameters and operations. Concretely, the number of parameters is reduced from 76.38k to 23k, and the number of operations per proposal is reduced from 5.72M MAC to 1.865M MAC.

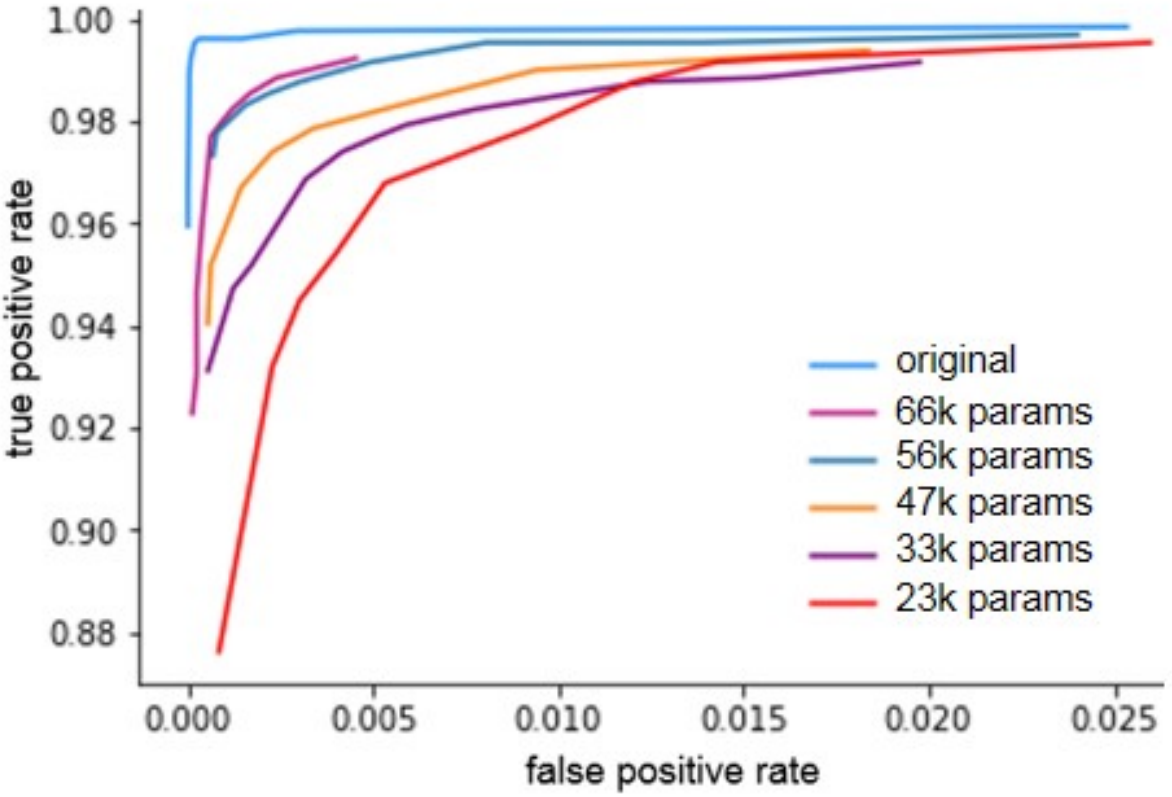


Figure 4.5 – Multiple trade-off points between TPR, FPR and # of parameters obtained through PCA. The database used for retraining consists of the first 13k face crops from the AFLW dataset [109] along with 13k negative samples. The remaining face crops are used for testing along with 50k negative samples.

The weights and activations on each layer are then also quantized to 8 bits without any loss of accuracy by employing the method described in [61]. The network is then fine-tuned on the data-set which has been generated as described at the beginning of section 4.2.3. Figure 4.6 shows the performance of the original BCN as compared to both the optimized and MTCNN versions for various datasets.

Furthermore, the small size of the global optimized solution (6.3k parameters on the PN, 23k on the BCN and 37k values as maximum temporary data, all quantized to 8-bits), makes it again the perfect scenario to further envision other speed-up techniques such as Winograd, which trade-off memory for computation. Concretely, for 4x4 filters, a (2x2,4x4) winograd configuration yields an additional theoretical reduction of x2.56 operations. Winograd is also compatible with PCA compression, as was explained in section 3.5.3. However, because of time constraints, this point was not thoroughly explored.

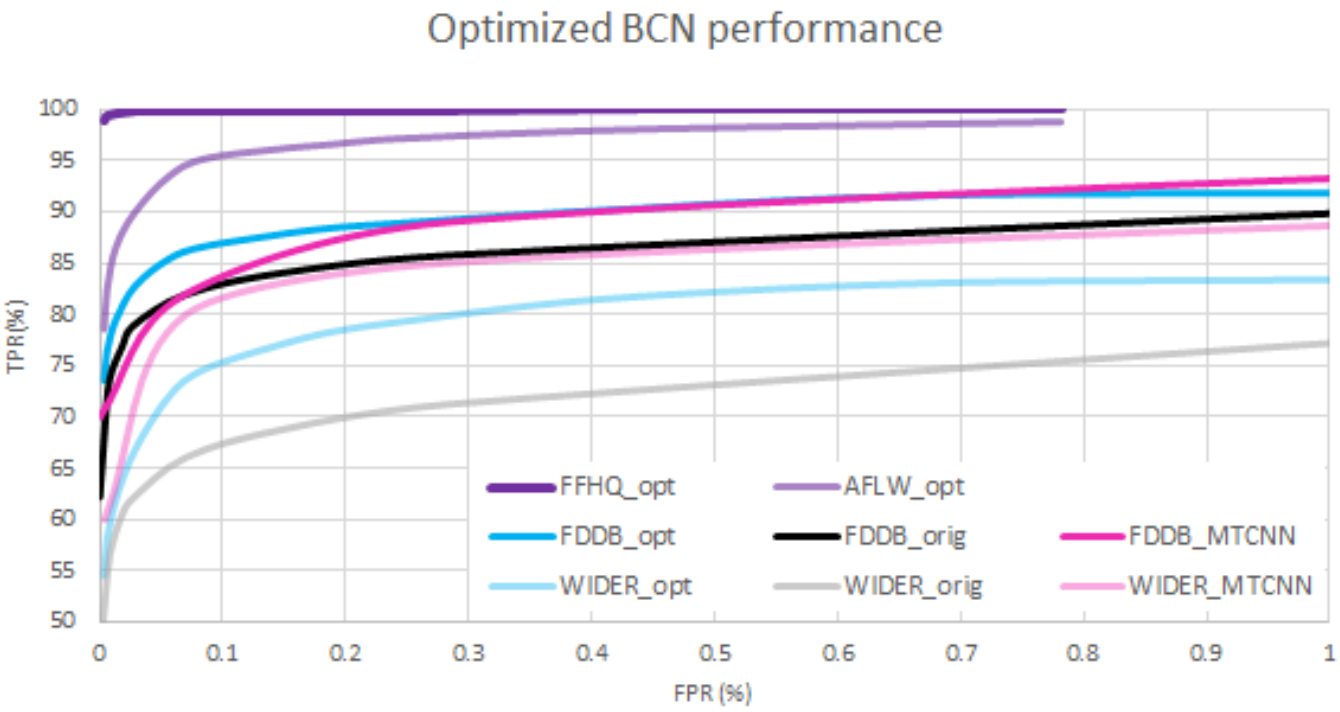


Figure 4.6 – Comparison between the original(orig) and the optimized(opt) models, as well as MTCNN for multiple datasets : FFHQ, AFLW, FDDB and WIDER FACE.

4.2.4 Comparison with other solutions

In order to determine the quality and interest of the proposed solution, LFFD (Lightweight & Fast Face Detector), we compare it to both our own VJ [99] based HW implemenation as well as MTCNN [103]. All these solutions are compared on a dataset which is representative of the applicative use-case scenario both in terms of their computational cost and detection performance. For this, only face sizes above 100 pixels from the validation subset of WIDER FACE are selected, which is equivalent to detecting 64x64 faces inside VGA inputs. The cycle counts in this section are merely approximative and are meant to give an order of magnitude of the complexity of the solutions. More detailed analysis for our proposed method is given in Chapter 5.

As can be seen in Figure 4.7, in the case of VJ, a TPR of 38.47% is obtained by evaluating 409.513 features per frame on average, with a maximum number of 1.51M evaluations for a single frame. Our VJ implementation takes approximately 50 clock cycles to evaluate a single feature on average, meaning an average of 20.48M cycles per frame (120x90 pixels).

In the case of LFFD, a TPR of 49.62% is obtained with an average of 28.01M MAC per frame. Since our convolution accelerator is able to deliver around 16 MAC/cycle, this means that an average of 1.75M clock cycles are needed per frame. More accurate measurements for different configurations are given in Chapter 5.

This clearly shows the superiority of a CNN-based solution, by outperforming the VJ-based solution by 10% in detection ability with approximately an order of magnitude faster computation.

Another key benefit of CNN is its ability to scale to harder application scenarios. For example, as can be seen in Figure 4.7, VJ is only able to reach a 55.05% TPR by increasing the average number of feature evaluations to 7.27M, while LFFD is able to reach a similar TPR of 55.13% by slightly increasing the number of MACs per frame to 33.81M.

We also compare LFFD to MTCNN in Figure 4.7. We show that, for similar TPR and FPR, LFFD performs a lower average number of MACs per frame, and much lower in the maximum number of MACs per frame. LFFD also employs a x64.5 smaller model than MTCNN, i.e. 29.3KBytes coded in 8-bit vs. 1.89MBytes coded in 32-bit. Maximum intermediary data is also lower by a factor of x6.352.

Finally, LFFD is evaluated on CelebA [113]. This dataset is quite representative of our applicative use-case, since it consists of images containing mostly a single high resolution face (>100 pixels as was shown in Figure 2.17, Chapter 2). It is also large, as it is comprised of 202,599 images. In this dataset we are able to obtain, for example, a 93.77% TPR with a total of 1205 false positives ($< 10^{-3}$ % FPR, of which most are unannotated and partial detections) for an average computational budget of 50M MAC per frame, enabling very efficient and real-time face detection on low-cost HW.

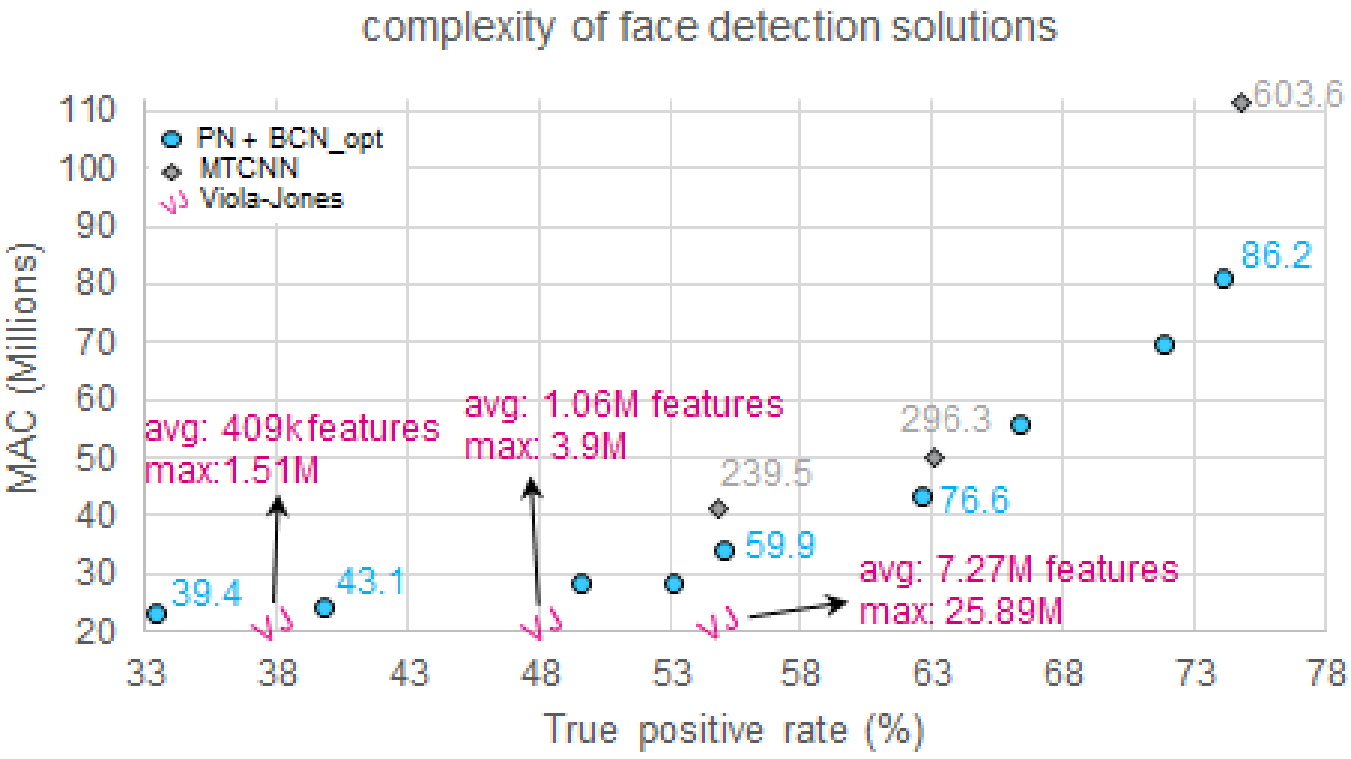


Figure 4.7 – Complexity comparison of various face detection methods. Points correspond to the avg. number of MACs per frame, while annotations correspond to the max. number of MACs on a single frame. All points have similar FPRs.

4.3 Conclusions

Throughout this chapter, the methodology that was proposed and explained in Chapter 2 has been put to practice in the case of face detection. The neural network compression methods proposed in Chapter 3 have also been applied and validated in this chapter for the case of binary classification. By splitting the detection process into a proposal and classification stage, and carefully designing each of these networks, we have been able to more flexibly trade-off computational complexity for detection performance.

The amount of parameters of the obtained solution consists of only 29.3KBytes (x64.5 smaller than MTCNN), and inference is also faster than both Viola-Jones and MTCNN for similar detection rates. Although Viola-Jones allows for very efficient hardware implementations, we estimate our CNN-based solution to require around an order of magnitude less computation at similar detection rates. Furthermore, we have also shown the ability of the CNN to scale to harder applicative scenarios without greatly incrementing computational complexity. This is also a great advantage over the Viola-Jones based solution.

The obtained networks then need to be evaluated in hardware, in order to guarantee that the theorical speed-ups match the measured speed-ups. This will be discussed in the next chapter of this thesis.

4.4 Perspectives & possible improvements

4.4.1 General object detection

As preseneted in Chapter 2, the methodology is generic and could be applied to more complex tasks, such as multi-class object detection. For example, the method could be applied to 10-way object detection. For this, it would suffice to train a 10-way proposal network on datasets such as PASCAL VOC or MS-COCO. An optimized classification network trained on a dataset such as CIFAR-10 was already obtained in section 3.5.4. This 10-way classification network consisted of around 130kBytes (binary weights) and provided around 90% accuracy. No experiments were carried on a the proposal network due to lack of time. Intuitively, the proposal network should not be much more complex than the one proposed in this chapter in the case of face detection . This would allow moderately reliable multiple-object detection through simple neural networks which are able to run on most embedded systems.

4.4.2 Near-Infrared image sensors

Machine vision applications such as ADAS (advanced driver assistance systems) often require illumination that goes beyond the visible light spectrum (e.g. night vision). Advances in digital near-infrared (NIR) imaging have revolutionized these night-vision capabilities. This approach also often requires less power consumption. Furthermore, these types of sensors can help reduce computational complexity of CV algorithms : from Figure 4.8 we can see that in low light environments, the subject, stands out. This could greatly help in selec-

ting a Region of Interest (RoI), therefore reducing the possible locations to be analyzed by the detection algorithm presented in this Chapter.

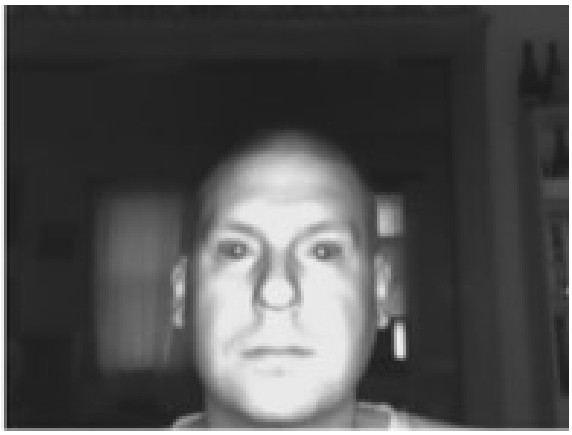


Figure 4.8 – Example of infrared face image in an indoor environment. The subject naturally stands out in this scenario and reduces the search space.

Even RGB sensors could benefit from this observation. As Figure 4.9 shows, samples with relatively flat backgrounds result in mostly sparse feature maps (very often > 75%). This could be exploited by hardware which is able to skip computations involving zero values.

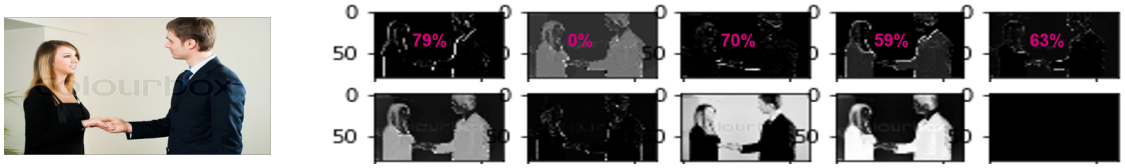


Figure 4.9 – Feature map sparsity on the proposed solution for a flat background example image.

4.4.3 Event-based sensors

The proposal stage in the workflow could greatly be simplified by specialized HW, or by different approaches to vision such as IR cameras (previous section), or event-based cameras. Event-based cameras are asynchronous sensors that pose a paradigm shift in the way visual information is acquired. This is because they sample light based on the scene dynamics, instead of relying on a fixed clock that has no relation to the scene. They are data-driven sensors : their output depends on the amount of motion or intensity changes in the scene. This generally implies a lower power consumption.

Moreover, each pixel memorizes the log intensity each time it sends an event. Intensity levels for each pixel are then continuously monitored. When the intensity for a pixel exceeds an increment threshold, an event is sent. These events communicate the location (x,y), the time (t), and the polarity (p) of the change (i.e., intensity increase (1) or decrease (0)).

Another important aspect is that this apporach is also much more compatible with a neuromorphic approach (discussed in section 1.5) and is also biologically inspired. Since maturity of event-based sensors is not yet on point, comparisons are complicated, but they certainly show potential for more efficient, near-sensor processing. Notably, only high-level, non-redundant information is transmitted, thus reducing bandwidth, latency and power consumption.

5

Hardware evaluation on embedded multiprocessor

In this chapter we evaluate the proposed methods (both PCA compression and face detection solution) on hardware. We start by giving a quick overview of the state-of-the-art of CNN hardware accelerators. We then show the results of implementing the PCA-optimized Binary Face Classifier network from section 4.2.3.1 on a custom embedded multicore, thereby verifying the effectiveness and interest of the proposed PCA compression algorithm, as well as its parallelization ability. We then evaluate the fully optimized proposed face detection solution.

Sommaire

3.1 CNN compression methods 65
3.2 PCA for CNN compression 70
3.2.1 PCA for memory reduction 70
3.3 PCA for computation reduction 72
3.4 Experiments 73
3.4.1 Uniform compression 74
3.4.2 Progressive compression 76
3.4.3 Trained reconstruction 77
3.4.4 PCA for filter removal 79
3.5 PCA combined with other compression methods 81
3.5.1 PCA combined with pruning 81
3.5.2 PCA combined with pruning and quantization 83
3.5.3 PCA combined with winograd 83
3.5.4 PCA combined with binary quantization 85
3.6 Conclusion 85

3.7 Perspectives 86

5.1 State of the art CNN accelerators

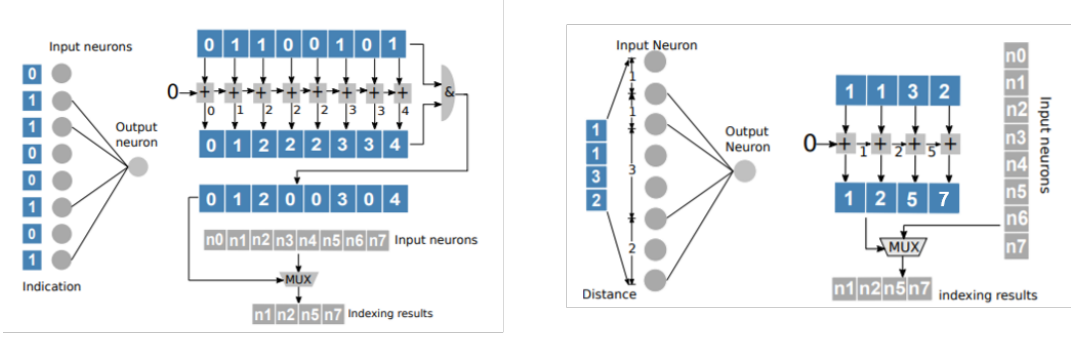
5.1.1 Sparse accelerator

As was discussed in section 3.1.0.2, pruning is a very good way to remove parameters from a neural network with very low accuracy degradation. However, doing so creates holes in the network, and therefore, in order to be able to take advantage from the parameter reduction, the hardware needs to index the position of zero values inside the network. This, evidently lowers memory gains due to encoding overheads, but also complicates execution and parallelism, as was explained in section 3.1.0.2.

The work in [69] proposes two interesting ways to deal with the aforementioned issues :

— Direct indexing : this method is the most straightforward, and consists in applying a binary mask to each of the weights. This concept is shown on the left part of Figure 5.1.a. Its hardware implementation is shown in the right part of Figure 5.1.a, in which first, each bit in the binary indexing string is added to obtain an accumulated string. Each element in the accumulated string, which indicates the location of the corresponding connection. After enforcing an "AND" operation between the accumulated string and the original string, the indexes of each connected neuron can be obtained, and the output neuron's value can be retrieved through a multiplexer.

— Step indexing, is shown on Figure 5.1.b, and consists in recording the distance between nonzero values. For its implementation, a distance string between consecutive nonzero neurons is stored. In the same way, each element in the distance string is added in order to generate an accumulated string, which directly indicates the position of nonzero neurons. The output neuron's value can then be retrieved through a multiplexer.



(a) (b)

Figure 5.1 – Two different approaches for sparse accelerators : (a) direct indexing, (b) step indexing. Images taken from [69].

The work in [69] also compares both approaches across multiple networks and datasets, and reports the costs of the step indexing as always smaller than that of the direct indexing both in terms of area and power.

Compressed Sparse Row (CSR) is another sparse matrix encoding format, in which the sparse matrix is represented by three submatrices, as is shown in Figure 5.2 :

— The matrix A contains all non-zero values.

— The matrix IA contains the cummulative total number of non-zero values per row.

— The matrix JA contains the ordered column index of each non-zero value. It may contain an extra end element which is set to the number of non zero entries in the original matrix (full non-zero rows).

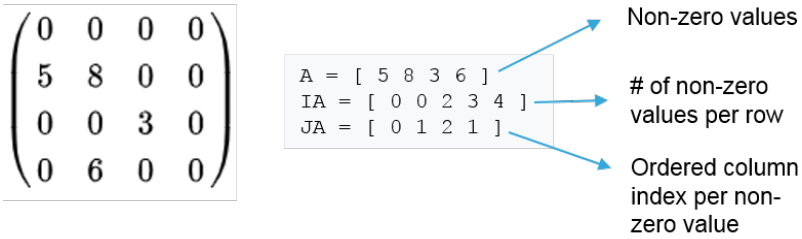


Figure 5.2 – Compressed sparse row representation.

5.1.2 Winograd accelerator

The work in [70] explores a Winograd implementation in the Intel/Movidius Myriad2 platform, which is similar to the ASMP's architecture, and which will also be later described in section 5.2.1. Its architecture is shown in Figure 5.3 and contains :

— A general purpose processor.

— Vector processing units (VPUs) which support SIMD (single instruction multiple data) and MAC operations efficiently.

— An internal SRAM, which is accessible by all processors and is partitioned in slices, offering a faster access to specific processors. Data transfers and management must therefore be optimized.

— An external DDR memory.

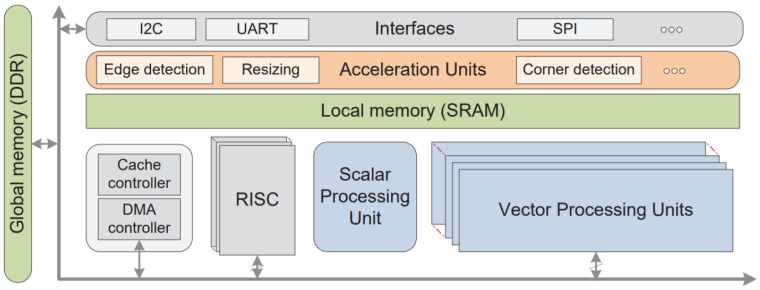


Figure 5.3 – Intel/Movidius Myriad2 architecture. Image taken from [70].

Results show that their Winograd implementation scales almost linearly with the number of VPUs. They observe that, although power increases with the number of active processing units, the energy consumption is dominated by the execution time, rather than by the power. Since the execution time significantly drops with the number of active processing units, energy decreases as well. However, performance greatly depends on the layer's configuration, as will be later discussed in section 5.3.

5.1.3 Sparse Winograd accelerator

As was explained in section 3.1.0.2, the work in [71] proposes to combine the computational savings of sparse weights and activations with the savings of the Winograd transform by making two modifcations to conventional CNNs :

— the networks are trained and pruned directly in the Winograd domain.

— the ReLU non-linear operation is employed after the Winograd transform to make the activations sparse at multiplication.

The work in [72] implements an FPGA CNN accelerator which is able to support both the CSR sparse encoding format, as well as the the winograd transformations. The authors report a 2.9x3.1x speedup for a VGG-16 and YOLO networks on a Xilinx ZC706 platform.

5.1.4 Binary/ternary network implemenations

-XNOR

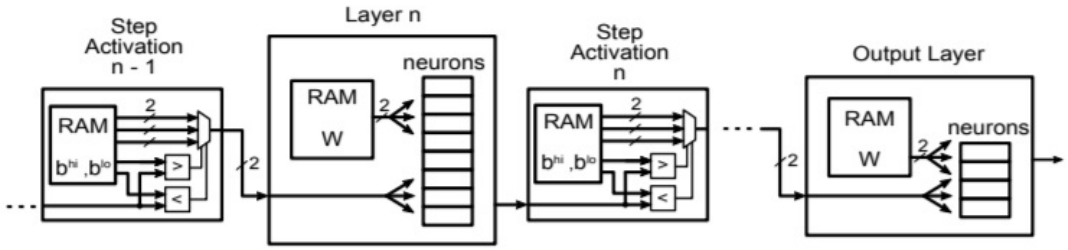


Figure 5.4 – Ternary network hardware implementation. Image taken from

5.2 PCA evaluation on ASMP

Although more specialized hardware is of clear interest and needs to be further studied, one of the goals of the Ph.D. was to benchmark the solution on ST's ASMP (Application Specific Multiprocessor). This platform is quickly overviewed in section 5.2.1. Results relative to PCA's performance are shown and discussed in section 5.2.2.

5.2.1 ASMP overview

The ASMP is a STMicroelectronics multicore cluster made from STMicroelectronics STxP70 processors. Its schematic is shown in Figure 5.5 and described in [115]. Code parallelization is managed by adding OpenMP directive annotations. Each PE (processing element) is made up of a STxP70 core along with extensions, which allows adding specific instructions in order to speed-up specific processing. For example, the CVA8 extension allows the speed-up of common computer vision instructions such as multiply accumulates (MAC) through vectorial processing.

The ASMP contains a shared data memory as well as a shared program cache between all PEs. It is possible to configure the shared memory TCDM (tightly-coupled data memory) by choosing its size, data width and banking factor.

Two other memories can be added in an ASMP configuration : a SRAM and a DDR. These memories form a configurable 3 levels memory hiearchy ; TCDM is L1, SRAM is L2 and DDR is L3. L1 is small and fast, while L2 has medium size and access, and L3 is large and slow. Generally L1 contains less than 1 Mbyte, L2 a few Mbytes and L3 can go up to 4 Gbytes. The DMA (Direct memory access) allows to transfer data between the different levels of memory, usually between L1 and L2/L3.

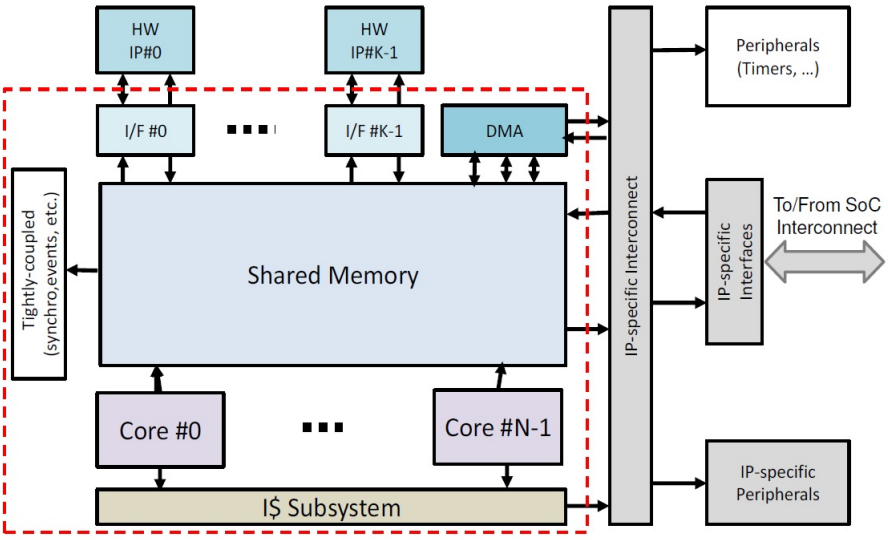


Figure 5.5 – Schematic of the ASMP.

5.2.2 Compressed BCN evaluation on ASMP

In this section, we start by implementing the optimized Binary Classification Network (BCN) obtained in section 4.2.3.1 on the ASMP. The goal of this is twofold :

— First, we aim to verify that the PCA compression algorithm proposed in Chapter 3 and applied to face classification in section 4.2.3.1 is well suited for a hardware implementation, and mainly preserves parallelism.

— Second, we also aim to benchmark the optimized network on the ASMP for multiple possible configurations.

As a measure of efficiency, Table 5.1 provides the number of clock cycles to run the BCN from a single PE and up to 8 PEs, for the following different versions of the network :

— The original, as described in Table 4.3.

— The optimized version, obtained after PCA compression. The basis and recombination layers are fused in this hardware implementation, as explained in section 3.3.

— The optimized version with filter sizes aligned to multiples of 8 through PCA, as explained in section 3.3.

Table 5.1 – Binary face detection network performance on ASMP multiprocessor. ***Algorithmic speed-up*** *is measured relatively to the same architecture and number of PE.*

# PEs	1	2	4	8
Original network				
# cycles	1,386,170	782,452	461,247	290,702
MAC/cycle	4.40	7.80	13.24	21.00
Parallel speed-up	x1.00	x1.77	x3.01	x4.77
After PCA + retraining ; fused basis and recombination				
# cycles	625,193	355,677	211,755	139,802
MAC/cycle	2.99	5.26	8.83	13.38
Alg. speed-up	x2.22	x2.20	x2.18	x2.08
Parallel speed-up	x1.00	x1.76	x2.95	x4.47
After PCA + retraining ; fused basis and recombination + alignment				
# cycles	527,214	298,276	177,839	118,637
MAC/cycle	3.75	6.62	11.11	16.66
Alg. speed-up	**x2.63**	**x2.62**	**x2.60**	**x2.45**
Parallel speed-up	**x1.00**	**x1.77**	**x2.96**	**x4.44**
Total speed-up	**x2.63**	**x4.65**	**x7.79**	**x11.68**

Table 5.1 shows the effectiveness of the proposed method, which allows to reduce the number of cycles by a factor of x2.63 over a single PE, while parallelization over 8 PEs yields a x4.44 speed-up. Combining compression and parallelization over 8 PEs therefore amounts to a $2.63 \times 4.44 = 11.68$ speed-up compared to the original network running on a single

PE. It takes only 0.118M clock cycles to process a proposal when running the optimized network on the ASMP with 8 PEs.

However, these results could be improved since they show a low number of MAC per cycle as compared to the original network. Indeed, from Table 5.1, it can be seen that theoretical and measured speed-ups are not consistent. Specifically, the optimized network theorically reduces the number of operations by a factor x3.27, while execution on the ASMP with a single PE takes x2.63 less cycles to complete than the original.

This is due to side effects when parallelizing tasks of small iteration counts, as well as irregular layer sizes (low number of channels, filters or fmap sizes), since both the size and number of channels of feature maps on the optimized network are quite low. Indeed, as can be seen from Figure 5.6, layers with the highest number of MACs (conv2,4,3 & 5 respectively) obtain better performance in terms of MAC/cycle with respect to smaller layers (conv1,6 & 7). When iteration counts are low, overheads and synchronization mechanisms become prohibitively expensive, lowering parallelization performance. This phenomenon is known as Amdahl's law, and can explain an increasing performance gap between the original and optimized network as the number of PEs increases, as can be seen in Figure 5.1.

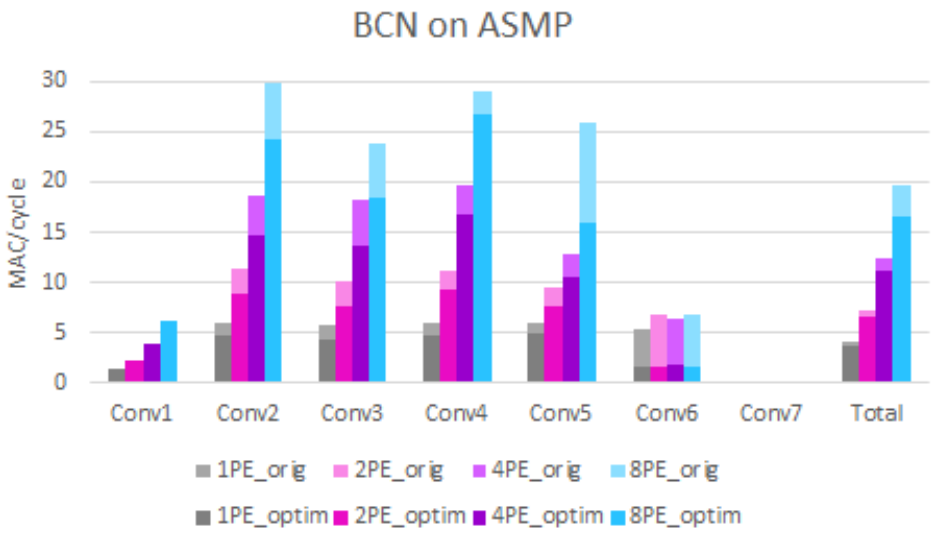


Figure 5.6 – Layer-wise comparison of parallelization effect on the original and optimized BCN across different number of PEs.

To gain better insight, Figure 5.7 shows a more detailed analysis. Concretely, Figure 5.7.b shows the decomposition of a filter in its basis and its coordinates terms, and both are measured independently. As an example, the layer 'Conv2.1' indicates the convolution in the PCA basis and 'Conv2.2' the one with the coordinates, a.k.a. 'recombination' layer. In 5.7.c, the intermediate feature maps are suppressed and the convolutions with the coordinates on the recombination layer are performed on the fly.

As Figure 5.7.b shows, 1x1 convolutions are not well parallelized in our implementation and none of these layers performs better than 2.5 MAC/cycle. However, since they constitute a very small portion of the operations relative to the basis layers, overall performance is not greatly affected by these, as can be seen in Figure 5.7.c. In this latter figure the computation of basis and recombination layers are fused and we can see that overall performance is very similar to that of basis layers from Figure 5.7.b, confirming that recombination layers are

practically negligible. The basis layers however, are far from the performance shown in Figure 5.7.a. This could be explained by the low number of operations, as well as filter size misalignments caused by PCA :

— The low number of operations relatively increases the overheads of the ASMP architecture and OpenMP stack, causing parallelization to be less effective.

— The PCA decomposition generates irregular filter shapes in terms of number of input and output channels. For example PCA would typically decompose a 32 inputs, 48 outputs 3x3 filter (i.e. 48x3x3x32) into a basic 8x3x3x32 filter, followed by a 48x1x1x8 recombination filter. This kind of misalignment can cause PCA to become less effective.

In any case, a maximum imporvement of $(3.27-2.63)/3.27*100 = 19.57\%$ could be achieved though microarchitecture imporvements.

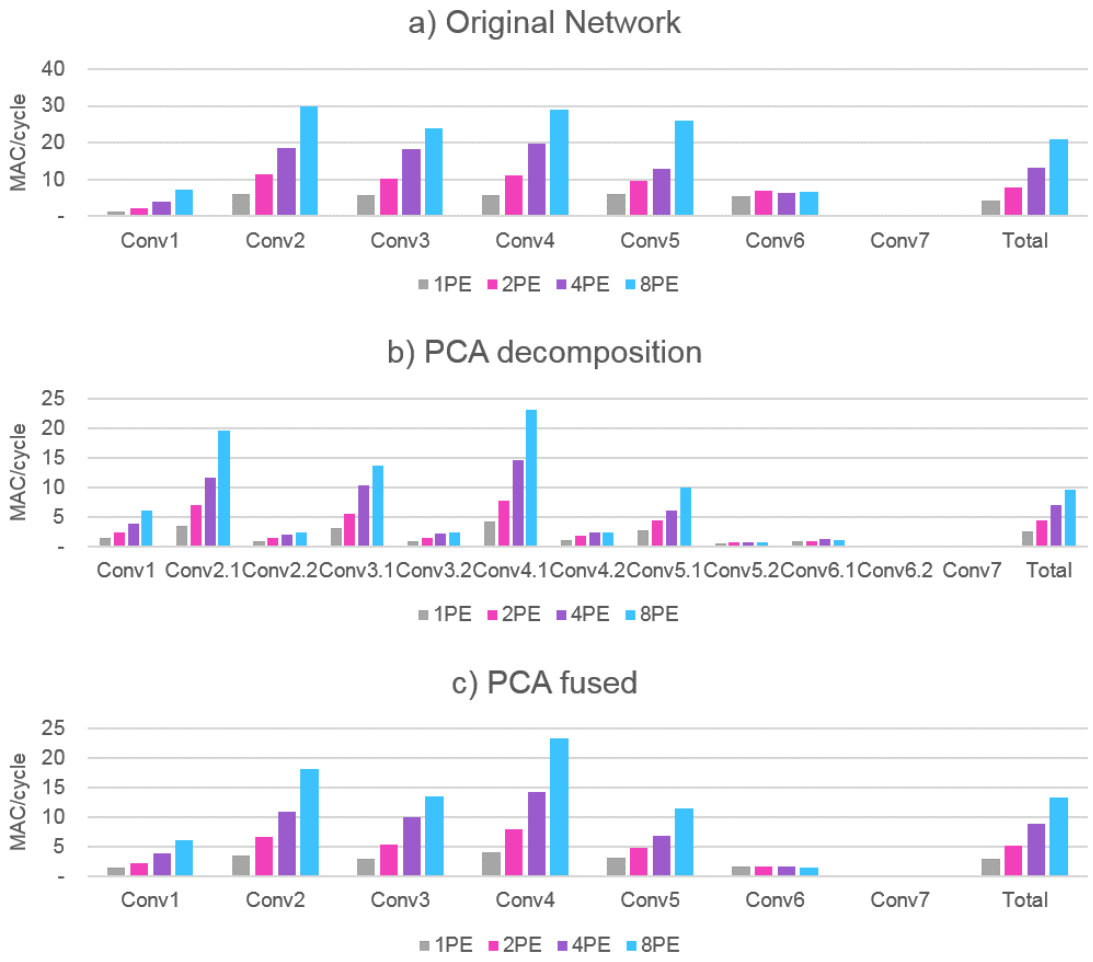


Figure 5.7 – Layer-wise performance of PCA optimization comparison for the original(a), decomposed(b) and fused(c) network implementations on the ASMP multiprocessor.

5.3 Proposal network evaluation on ASMP

The proposal network(PN) is also evaluated on the ASMP in order to get a better approximation of the global complexity of the solution. Since the PN gets multiple-sized inputs from the multi-resolution pyramid, it needs to be evaluated for all possible input image sizes.

Table 5.8 reports the results both in terms of number of cycles and MAC/cycle for all possible image sizes reported in Table 4.4 from Chapter 4. Only results generated by employing the CVA8 extension of the ASMP and a static fixed-point representation are reported, since these allow to obtain the best results in terms of number of cycles.

input size	120x90				96x72				76x57				61x46				49x36			
Total MAC	14,941,920				9,129,000				5,478,300				3,286,440				1,867,020			
# PEs	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
# cycles	6,556,190	3,610,045	1,957,653	1,110,035	4,156,619	2,212,930	1,209,741	704,528	2,462,955	1,380,303	751,790	438,293	1,527,691	859,323	462,354	293,683	908,237	527,337	285,634	172,780
MAC/cycle	2.28	4.14	7.63	13.46	2.20	4.13	7.55	12.96	2.22	3.97	7.29	12.50	2.15	3.82	7.11	11.19	2.06	3.54	6.54	10.81

input size	39x29				31x23				25x19				20x15				16x12			
Total MAC	1,103,670				592,890				327,990				143,460				48,600			
# PEs	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
# cycles	561,899	334,098	183,726	115,070	326,373	193,377	114,292	74,536	198,967	121,359	71,776	48,449	108,106	70,442	45,193	34,285	56,685	41,088	29,808	23,494
MAC/cycle	1.96	3.30	6.01	9.59	1.82	3.07	5.19	7.95	1.65	2.70	4.57	6.77	1.33	2.04	3.17	4.18	0.86	1.18	1.63	2.07

Figure 5.8 – Proposal network performance on ASMP for different input image sizes and number of PEs.

Figure 5.9 shows the same results as reported in Table 5.8 in a graphical way. By observation of this figure, it is very clear that as the input image gets larger, efficiency in terms of MAC/cycle improves over a single PE. Parallelism also improves as inputs become larger.

Indeed, as Table 4.4 shows, the higher the input resolution, the more operations need to be carried out. In this regime, the ASMP becomes more efficient. As can be seen in the figure, as the resolution increases, efficiency also always increases across a constant number of PEs. For example, at 8 PEs, an input resolution of 16x12 only achieves around 2.07 MAC/Cycle, while the same configuration with a 120x90 input resolution is able to achieve around 13.46 MAC/Cycle. Parallelism also becomes more effective as input resolution increases. For example, at 16x12 resolution, efficiency improves by a factor of x2.41 from 1 to 8 PEs, while at 120x90 resolution, efficiency improves by a factor of x5.90 from 1 to 8 PEs.

As a perspective, further significant computation gains could also be obtained in the Proposal Network by applying and parallelizing the Winograd algorithm on the ASMP. Preliminary experiments show approximately an additional x1.6 effective speed-up from Winograd on the ASMP. However, other preliminary experiments have shown a x10 increase on the average number of generated proposals when winograd and 8-bit quantization were combined on the proposal network. A retraining stage could probably help recovering from this increase in the number of proposals and is left for further experimentation.

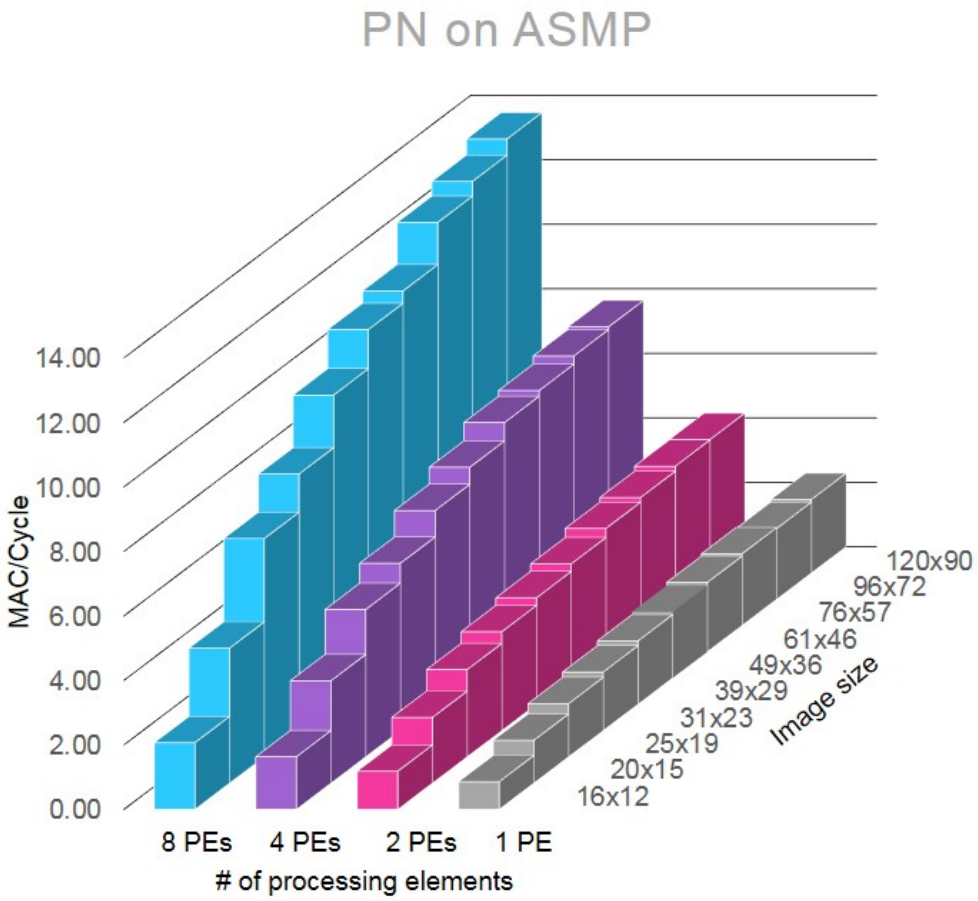


Figure 5.9 – Proposal network performance on ASMP graphically.

5.4 Face Detection solution evaluation on ASMP

In this section we aim to give an approximation of the complexity in terms of clock cycles for running the proposed face detection solution on the ASMP. This is approximative, since only the networks (PN and optimized BCN) are executed, while the computation cycles of other required operations, such as pyramid generation or non-maximum supression, etc. are ignored.

For example, the 3-level pyramid described in Table 4.5 from Chapter 4 would generate the following image sizes : 120x90, 49x36 and 20x15. From table 5.8, with an 8 PE configuration, this would mean a total of 1.11, 0.173 and 0.0342 million clock cycles respectively. From Table 5.1, with an 8 PE configuration it takes 0.119 million clock cycles to process a single proposal with the PCA optimized BCN. Since 3 proposals are passed on average to the optimized BCN, this would take an additional total of $0.119 * 3 = 0.357$ million clock cycles on average. Therefore, running all networks with 8 PEs on the ASMP would approximately take a total of $1.11 + 0.173 + 0.0342 + 0.357 = 1.674$ million clock cycles.

By considering 10 frames per second a real-time rate on the applicative use-case, the solution would need to run at a frequency of $1.674 * 10 = 16.74$ million cycles per second, or equivalently, 59.74 Mhz.

Finally, other estimations are shown in Table 5.2 for other different face detection configurations allowing to reach higher recall rates.

Table 5.2 – Possible trade-offs between model complexity and performance.

P.L.	Scale	PN MAC	PN cycles	avg. # p.	BCN MAC	BCN cycles	Total MAC	Total cycles	Recall
3	0.4	20.76M	**1.317M**	3	17.16M	**0.357M**	37.92M	**1.674M**	56.14%
5	0.6	26.72M	**1.735M**	54	308.9M	**6.426M**	335.62M	**8.161M**	91.22%
10	0.9	45.00M	**3.016M**	242	1.381G	**28.798M**	1.381G	**31.814M**	97.62%

5.5 Conclusion and Perspectives

The main goal of this chapter was to show that the proposed PCA compression algorithm is able to preserve computation gains, and run efficiently when implemented in hardware. For this, the PCA compressed Binary Classification Network proposed in section 4.2.3.1 from Chapter 4 is implemented on the ASMP, and the theorical speed-up is compared to the measured speed-up. Over a single PE, thanks to PCA, we are able to align the number of filters to multiples of 8 and obtain a x2.63 speed-up in the number of cycles. However, this is slightly inconsistent with the theoretical computation gain provided by PCA, which amounts to x3.27 in this case. This inconsistency is mainly caused by irregular filter shapes after PCA, i.e. the number of input and output channels after PCA are greatly imbalanced, causing the ASMP to be less effective.

Furthermore, when the execution of this network is parallelized over 8 PEs, we are able to obtain a x11.68 speed-up in the number of cycles with respect to the original network running over a single PE. This clearly showcases the effectiveness of our proposed approach.

The proposal network is also then evaluated on the ASMP for different input image resolutions. We observe that larger input resolutions run more efficiently in terms of MAC/cycle, and also benefit from greater speed-ups when parallelizing over multiple PEs.

We also approximately estimate the complexity of running the proposed face detection solution on a real-time scenario. For a 10fps analysis, the ASMP would need to run a frequency of around 60 Mhz. This is a relatively low frequency, and since lower frequencies correspond to lower energy consumption the our proposed solution is thereby proven to be efficient.

As a perspective, the whole solution should be implemented on the ASMP by including the rest of the operations (e.g. the multi-resolution pyramid) in order to obtain more accurate and reliable efficiency measurements.

Thesis Conclusion & Perspectives

Although very interesting deep learning-based computer vision solutions are constantly emerging, it is still very difficult to deploy these solutions in the real-world, more so in heavily computationally-constrained embedded systems. Throughout this thesis we have shown that there are multiple fundamental issues with CNNs in achieving this goal. In this thesis, multiple ideas allowing to ease these issues are proposed. However, this remains an open problem and therefore new ideas or approaches are interesting to explore or combine.

Furthermore, since this is a very broad and multi-disciplinary domain, it is not trivial to tell which approaches are fundamentally correct and should be pursued in the long-term. It is therefore extremely hard to establish clear statements about the strategic positioning with respect to the future of the field. This is why, throughout this thesis, we have also tried to keep a very broad outlook, providing a review and critical analysis of the different promising emerging domains. For each of these domains, we have tried to simultaneously analyze the strong and weak points of each approach, exposing the most promising techniques and ideas when approaching the embedded vision problem.

Indeed, most computer vision solutions from the state-of-the-art are heavily over-dimensioned, and implementing such models as a black-box therefore also leads to over-dimensioned hardware. This has great impact both in the cost of the solution and energy efficiency, which hinder the solution's market adoption, deployment, as well as sustainability. On the other hand, jointly designing hardware and algorithms is a clear path for efficient solutions, with algorithms clearly being the crucial dimensioning factor for hardware.

We have started this thesis by formalizing the problem that was faced, i.e. adapting a CNN's computational complexity to an embedded system's constraints. We have identified plenty of the multidisciplinary causes of such problem in order to properly define the most interesting problems which needed to be worked on. In this sense, three main work axis were identified, notably, Algorithms, Hardware and Application. While these can be seen as mostly orthogonal domains, a deep understanding of all of these is necessary in order to be able to design and identify the main levers leading to an efficient solution. In the first chapters of this work, we have thereby provided most of the relevant background knowledge that is required to correctly reason about this issue.

The over-dimensionnement of CNNs is double-edged : it eases the optimization process by increasing the set of equivalent parameterizations, but it also greatly increases computational complexity and therefore complicates deploying the inference stage of these algorithms on embedded systems. In order to ease this problem, we have started by over-viewing the main methods from the state-of-the-art allowing to exploit different CNN properties. From there, we have proposed a CNN compression method in Chapter 3, which is based on Principal Component Analysis (PCA). PCA allows us to find, for each layer of the network

independently, a new representation of the set of learned filters by expressing them in a more appropriate *PCA basis*, removing redundancies. This PCA basis is hierarchical, meaning that basis terms are ordered by importance, and by removing the least important basis terms, we are able to optimally trade-off approximation error for parameter count. However, it is not straightforward to determine the optimal approximation error for each layer. By fixing the PCA basis terms on every layer while applying a slight retraining phase to the set of *PCA coordinates*, the accuracy loss produced by the accumulation of approximation errors across all layers is recovered. Through this method, it is possible to compress, for example, an already efficient ResNet-32 network by a factor of $\times 2$ both in the number of parameters and operations with a loss of accuracy $<2\%$. While this method is effective, it is still very empirical. As a perspective, we outline an approach to optimally allocate approximation errors across layers. We also propose different variants of the algorithm allowing, in some cases, to completely remove up to a factor of $\times 11$ parameters in a single layer of a network.

We have also shown that the proposed method is compatible with other previously reviewed state-of-the-art methods exploiting different CNN properties which allow to reduce computational complexity. Concretely, we propose to combine PCA, pruning and quantization. Through this method, we have been able to reduce the size of a ResNet-110 from 6.88Mbytes to 370kbytes, i.e. a x19 memory gain with a 3.9 % accuracy loss. PCA was also succesfully combined with more agressive quantization schemes, such as binary values for the weights after PCA compression. For example, a ResNet-32 can be reduced by a factor of $\times 2$, and further compressed by a factor of $\times 6.4$ through binary quantization. Since weights are restrained to -1 and 1 values, addition is the only operation required for inference, thereby greatly simplifying its implementation on specialized hardware.

The previously mentioned techniques primarily aim at reducing the memory required to store a CNN model for inference on an embedded device. This is already a big challenge by itself, since memory size and type has a great impact on both the energy consumption and silicon area, and is therefore greatly limited for embedded devices. Furthermore, when the classifier is employed for complex tasks such as object detection, the number of operations, as well as memory requirements for processing high-resolution input images can also become a burden. In Chapter 2 of this thesis we discussed the main levers affecting computational complexity of a CNN-based object detector.

From there, we have proposed a general method that allows to flexibly tailor each of these trade-offs for a given applicative scenario. We argue that for object detection, splitting the process into a proposal phase followed by a classification phase allows to very flexibly trade-off computational complexity for detection performance.

Moreover, the proposal network needs to efficiently identify regions of interest in the image, which will then be classified by a more accurate classification network. This is efficient, since the majority of possible locations in the input image don't generally contain any objects of interest an can be discarded by a simpler analysis. Indeed, identifying regions of interest through a neural network is a much less computationally demanding task than accurately classifying every possible region of the image.

The ideal proposal network needs to deliver the highest possible recall, while minimizing both the average number of generated proposals, and the number of operations required for generating those proposals. For a fixed parameter count, recall increases when a lower decision threshold is employed. However, lowering the decision threshold also increases the number of proposals passed on to the classification network, and these need to be properly

tuned according to the application specifications. Furthermore, the number of operations employed to generate proposals for a fixed network architecture depends both on the size of the input image, and the number of pyramid levels. In this sense, bounding box regression allows us to greatly reduce the search space by gaining stronger scale and position equivariance.

The classification network is then applied on the set of generated proposals. This network needs to obtain the highest possible true positive rate and lowest possible false positive rate on the dataset composed of real faces, generated proposals and non-face background samples. Since Accurate classification through neural networks requires high parameter count, this network accounts most of the parameters of the solution, and therefore, reducing the parameter count of this network is the major concern. Furthermore, when high recall rates are required by the application, the decision threshold of the proposal network needs to be lowered, and the average number of generated proposals increases. In this regime, the cost of classifying the proposals greatly outweighs the cost of generating them. It is therefore important to study multiple trade-offs between computational complexity and accuracy through methods such as our proposed PCA compression, quantization, and winograd.

In the case of the proposal network, parameter count is much lower, and therefore the main interest of applying such methods is a reduction in the number of operations. This is especially interesting in a regime in which a low number of proposals is generated, since the cost of generating proposals and classifying them is quite equilibrated.

This proposed efficient object detection method is then brought to practice in Chapter 4 for a consumer face detection scenario. In this context, we initially propose a solution consisting of a total of 82.7k parameters. Multiple possible trade-offs between TPR, FPR and computational complexity are then studied through PCA, and the solution is then reduced to just 29.3k parameters, with 6.3k parameters on the proposal network, and 23k parameters on the classification network. Since weights and activations are quantized to 8-bit, the model size is 29.3kBytes. This is x65 smaller than other state-of-the art CNN face detectors such as MTCNN, while providing equal detection performance and lower number of operations on datasets such as CelebA. This dataset is large ($>$200k images) and representative of a consumer face detection applicative use case and therefore validates our approach. Our proposed face detector is also compared to a more traditional Viola-Jones face detector, exhibiting approximately an order of magnitude faster computation for the same detection performance, as well as the ability to scale to higher detection rates by slightly increasing computational complexity.

In Chapter 5, both networks are finally implemented in ST's ASMP, a custom embedded multiprocessor. This allows to verify that PCA compression doesn't add any computing overheads, and that theorical computation gains stay mostly consistent when implemented in hardware, as is indeed the case. Furthermore, parallelizing the PCA compressed BCN from Chapter 4 over 8 PEs achieves a x11.68 speed-up with respect to the original network running on a single PE. The PN's efficiency is also evaluated for different input resolutions. In this case, efficiency in terms of MAC/cycle, as well as parallelization speed-ups both increase as the input resolution gets larger. We also approximately evaluate the total cycle count required for the proposed applicative scenario. For example a recall of 56.14% can be obtained on the WIDER Face dataset, or equivalently a 93.77% on the CelebA dataset through an approximative 1.674 million cycles per frame.

Throughout this thesis the main levers allowing to efficiently implement a CNN-based object detector have been exposed. This requires simultaneously considering the application, algo-

rithm and hardware, and has been effectively demonstrated in this thesis for a face detection solution. Throughout this manuscript, it has been shown that both algorithm and application drastically determine the feasibility of implementing CNN solutions in embedded systems. The main goal is to modify the algorithm such that both the memory requirements and operation count are lowered, while also taking low-level hardware characteristics into account. The proposed CNN compression algorithm was proven effective in reducing computational complexity of multiple models from the SOA, while also preserving the regularity of operations, and allowing to flexibly align filter sizes to conveniently fit the hardware. The algorithms also need to be carefully tailored to the application, as is shown in the case of object detection, for which a cascaded approach proves to be very effective. While this reduces regularity of the execution pipeline, the gain both in terms of parameters and operations is very substantial. This approach also allows to very flexibly trade-off detection performance for computational complexity.

These methods are all combined and demonstrated throughout the manuscript, ultimately leading to a very efficient CNN-based face detection solution which is able to efficiently run on low-cost hardware. While multiple general methodologies have been proposed thorughout this thesis, the design of such systems remains a difficult and empirical task. It requires a lot of multidisciplinary background knowledge, as well as a large amount of design choices and parameter tuning in order to achieve a coherent solution.

Bibliographie

[1] Brillet, Lucas, Mancini, Stephane, Cleyet-Merle, Sébastien and Nicolas, Marina. Tunable CNN Compression Through Dimensionality Reduction. In Proceedings of the IEEE International Conference on Image Processing, pp. 3851-3855. 2019.

[2]

[3] Brillet, Lucas, Leclaire, Nicolas, Mancini, Stephane, Cleyet-Merle, Sébastien, Nicolas, Marina, Henriques, Jean-Paul and Delnondedieu, Claude. Speeding-up CNN inference through dimensionality reduction. In Proceedings of the IEEE International Conference on Design and Architectures for Signal and Image Processing, pp. 3851-3855. 2019.

[4] Urban, G., Geras, K. J., Kahou, S. E., Aslan, O., Wang, S., Caruana, R., ... Richardson, M. (2016). Do deep convolutional nets really need to be deep and convolutional ?. arXiv preprint arXiv :1603.05691.

[5] Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., Liao, Q. (2017). Why and when can deep-but not shallow-networks avoid the curse of dimensionality : a review. International Journal of Automation and Computing, 14(5), 503-519.

[6] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT press.

[7] Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." arXiv preprint arXiv :1605.07678 (2016).

[8] Bianco, S., Cadene, R., Celona, L., Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. IEEE Access, 6, 64270-64277.

[9] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International journal of computer vision 115.3 (2015) : 211-252.

[10] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

[11] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[12] https ://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

[13] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.

[14] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In proceedings of International Conference on Learning Representations, 2014.

[15] Lin, M., Chen, Q., Yan, S. (2013). Network in network. arXiv preprint arXiv :1312.4400.

[16] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[17] Srivastava, R. K., Greff, K., Schmidhuber, J. (2015). Highway networks. arXiv preprint arXiv :1505.00387.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778, 2016.

[19] Zagoruyko, S., Komodakis, N. (2016). Wide residual networks. arXiv preprint arXiv :1605.07146.

[20] Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K. (2017). Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1492-1500).

[21] Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely connected convolutional networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700-4708. 2017.

[22] Hu, J., Shen, L., Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).

[23] Larsson, G., Maire, M., Shakhnarovich, G. (2016). Fractalnet : Ultra-deep neural networks without residuals. arXiv preprint arXiv :1605.07648.

[24] Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets : Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv :1704.04861 (2017).

[25] Zhang, X., Zhou, X., Lin, M., Sun, J. (2018). Shufflenet : An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856).

[26] Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. "SqueezeNet : AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size." arXiv preprint arXiv :1602.07360 (2016).

[27] Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. "Learning transferable architectures for scalable image recognition." I

[28] Shlens, Jonathon. "A tutorial on principal component analysis." arXiv preprint arXiv :1404.1100, 2014.

[29] He, Yihui, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. "Amc : Automl for model compression and acceleration on mobile devices." In Proceedings of the European Conference on Computer Vision (ECCV), pp. 784-800. 2018.

[30] Strubell, Emma, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP.” Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (2019)

[31] Chen, Sheng, et al. "Mobilefacenets : Efficient cnns for accurate real-time face verification on mobile devices." Chinese Conference on Biometric Recognition. Springer, Cham, 2018.

[32] Ma, Xiangyu, Du, Shan and Liu, Yu. A Lightweight Neural Network For Crowd Analysis Of Images With Congested Scenes. In Proceedings of IEEE International Conference on Image Processing, pp. 979-983. 2019.

[33] Mounir Errami and Mohammed Rziza. Improving pedestrian detection using light convolutional neural network. In Proceedings of the International Symposium on Signal, Image, Video and Communications, pp. 71-75. 2018.

[34] Wang, Y., Zhou, Q., Liu, J., Xiong, J., Gao, G., Wu, X., Latecki, L. J. (2019). LEDNet : A Lightweight Encoder-Decoder Network for Real-Time Semantic Segmentation.In Proceedings of IEEE International Conference on Image Processing, 2019.

[35] Wofk, D., Ma, F., Yang, T. J., Karaman, S., Sze, V. (2019). FastDepth : Fast Monocular Depth Estimation on Embedded Systems. arXiv preprint arXiv :1903.03273.

[36] Kim, J. H., Hong, G. S., Kim, B. G., Dogra, D. P. (2018). deepGesture : Deep learning-based gesture recognition scheme using motion sensors. Displays, 55, 38-45.

[37] Shi, Wenzhe, et al. "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[38] Lubana, E.S., Dick, R.P., Aggarwal, V. and Pradhan, P.M., 2019, September. Minimalistic Image Signal Processing for Deep Learning Applications. In IEEE International Conference on Image Processing (ICIP) (pp. 4165-4169). 2019.

[39] Bora, Ashish, et al. "Compressed sensing using generative models." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

[40] Jiang, F., Tao, W., Liu, S., Ren, J., Guo, X., Zhao, D. (2017). An end-to-end compression framework based on convolutional neural networks. IEEE Transactions on Circuits and Systems for Video Technology, 28(10), 3007-3018.

[41] Löhdefink, Jonas, et al. "GAN-vs. JPEG2000 Image Compression for Distributed Automotive Perception : Higher Peak SNR Does Not Mean Better Semantic Segmentation." arXiv preprint arXiv :1902.04311 (2019).

[42] Chen, G., Parada, C., Heigold, G. (2014, May). Small-footprint keyword spotting using deep neural networks. In 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 4087-4091). IEEE.

[43] Zhang, Y., Suda, N., Lai, L., Chandra, V. (2017). Hello edge : Keyword spotting on microcontrollers. arXiv preprint arXiv :1711.07128.

[44] Palossi, D., Loquercio, A., Conti, F., Flamand, E., Scaramuzza, D., Benini, L. (2019). A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones. IEEE Internet of Things Journal.

[45] Loquercio, A., Maqueda, A. I., Del-Blanco, C. R., Scaramuzza, D. (2018). Dronet : Learning to fly by driving. IEEE Robotics and Automation Letters, 3(2), 1088-1095.

[46] Sze, Vivienne, et al. "Efficient processing of deep neural networks : A tutorial and survey." Proceedings of the IEEE 105.12 (2017) : 2295-2329.

[47] Du, Z., Fasthuber, R., Chen, T., et al. ShiDianNao : Shifting vision processing closer to the sensor. In ACM SIGARCH Computer Architecture News. ACM, 2015. p. 92-104.

[48] Cavigelli, Lukas et Benini, Luca. Origami : A 803-gop/s/w convolutional network accelerator. IEEE Transactions on Circuits and Systems for Video Technology, 2016, vol. 27, no 11, p. 2461-2475.

[49] Chen, Yu-Hsin, et al. "Eyeriss : An energy-efficient reconfigurable accelerator for deep convolutional neural networks." IEEE Journal of Solid-State Circuits 52.1 (2016) : 127-138.

[50] Chen, Yu-Hsin, et al. "Eyeriss v2 : A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices." arXiv preprint arXiv :1807.07928 (2018).

[51] Pfeiffer, Michael, and Thomas Pfeil. "Deep learning with spiking neurons : opportunities and challenges." Frontiers in neuroscience 12 (2018).

[52] Rueckauer, Bodo, et al. "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification." Frontiers in neuroscience 11 (2017) : 682.

[53] Girshick, R., Donahue, J., Darrell, T., Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[54] He, K., Zhang, X., Ren, S., Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence, 37(9), 1904-1916.

[55] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[56] Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster r-cnn : Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

[57] Redmon, Joseph, et al. "You only look once : Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[58] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., Berg, A. C. (2016, October). Ssd : Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[59] Lin, Tsung-Yi, et al. "Microsoft coco : Common objects in context." European conference on computer vision. Springer, Cham, 2014.

[60] Everingham, Mark, et al. "The pascal visual object classes (voc) challenge." International journal of computer vision 88.2 (2010) : 303-338.

[61] Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704-2713. 2018.

[62] Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect : Training deep neural networks with binary weights during propagations." In Advances in neural information processing systems, pp. 3123-3131. 2015.

[63] Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net : Imagenet classification using binary convolutional neural networks." In European Conference on Computer Vision, pp. 525-542. Springer, Cham, 2016.

[64] M. Courbariaux, Y. Bengio. "Binarynet : Training deep neural networks with weights and activations constrained to +1 or -1." In Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2016.

[65] F. Li and B. Liu, "Ternary weight networks," in NIPS Workshop on Efficient Methods for Deep Neural Networks, 2016.

[66] Lin, Xiaofan, Cong Zhao, and Wei Pan. "Towards accurate binary convolutional neural network." In Advances in Neural Information Processing Systems, pp. 345-353. 2017.

[67] LeCun, Y., Denker, J.S. and Solla, S.A., 1990. Optimal brain damage. In Advances in neural information processing systems (pp. 598-605).

[68] S. Han, H. Mao, and W.J. Dally, "Deep compression : Compressing deep neural network with pruning, trained quantization and huffman coding", 4th International Conference on Learning Representations, 2016.

[69] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. Cambricon-X : An Accelerator for Sparse Neural Networks Proceedings of the International Symposium on Microarchitecture (MICRO), 2016.

[70] A. Xygkis, D. Soudris, L. Papadopoulos, S. Yous, and D. Moloney. Efficient winograd-based convolution kernel implementation on edge devices. In 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pages 1–6, June 2018.

[71] Liu, X., Pool, J., Han, S., Dally, W. J. (2018). Efficient sparse-winograd convolutional neural networks. In the 2018 International Conference on Learning Representations.

[72] Lu, Liqiang, and Yun Liang. "Spwa : an efficient sparse winograd convolutional neural networks accelerator on fpgas." 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). IEEE, 2018.

[73] L. Lu et al. Evaluating Fast algorithms for Convolutional Neural Networks on FPGAs. In FCCM, 2017.

[74] Yang, Tien-Ju, Yu-Hsin Chen, and Vivienne Sze. "Designing energy-efficient convolutional neural networks using energy-aware pruning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[75] Liu, Lanlan, and Jia Deng. "Dynamic deep neural networks : Optimizing accuracy-efficiency trade-offs by selective execution." In Thirty-Second AAAI Conference on Artificial Intelligence. 2018. Fahlman, S. E., Lebiere, C. (1990). The cascade-correlation learning architecture. In Advances in neural information processing systems (pp. 524-532).

[76] Passalis, Nikolaos et al. "Adaptive Inference Using Hierarchical Convolutional Bag-of-Features for Low-Power Embedded Platforms." 2019 IEEE International Conference on Image Processing (ICIP) (2019) : 3048-3052.

[77] Fahlman, S. E., Lebiere, C. (1990). The cascade-correlation learning architecture. In Advances in neural information processing systems (pp. 524-532).

[78] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network", Deep Learning and Representation Learning Workshop, NIPS, 2014.

[79] Asif, Umar, Jianbin Tang, and Stefan Harrer. "Ensemble Knowledge Distillation for Learning Improved and Efficient Networks." arXiv preprint arXiv :1909.08097 (2019).

[80] Olshausen, B. A., Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature, 381(6583), 607-609.

[81] Luan, Shangzhen, Chen Chen, Baochang Zhang, Jungong Han, and Jianzhuang Liu. "Gabor convolutional networks." IEEE Transactions on Image Processing 27, no. 9 (2018) : 4357-4366.

[82] Shang, Wenling, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. "Understanding and improving convolutional neural networks via concatenated rectified linear units." In international conference on machine learning, pp. 2217-2225. 2016.

[83] Cohen, Taco, and Max Welling. "Group equivariant convolutional networks." In International conference on machine learning, pp. 2990-2999. 2016.

[84] Jaderberg, Max, Karen Simonyan, and Andrew Zisserman. "Spatial transformer networks." In Advances in neural information processing systems, pp. 2017-2025. 2015.

[85] Tai, Kai Sheng, Peter Bailis, and Gregory Valiant. "Equivariant Transformer Networks." arXiv preprint arXiv :1901.11399 (2019).

[86] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." In Advances in neural information processing systems, pp. 3856-3866. 2017.

[87] Kosiorek, Adam R., Sara Sabour, Yee Whye Teh, and Geoffrey E. Hinton. "Stacked Capsule Autoencoders." arXiv preprint arXiv :1906.06818 (2019).

[88] Rippel, Oren, Jasper Snoek, and Ryan P. Adams. "Spectral representations for convolutional neural networks." In Advances in neural information processing systems, pp. 2449-2457. 2015.

[89] Lavin, Andrew, and Scott Gray. "Fast algorithms for convolutional neural networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4013-4021. 2016.

[90] Huang, Guang-Bin, Zuo Bai, Liyanaarachchi Lekamalage Chamara Kasun, and Chi Man Vong. "Local receptive fields based extreme learning machine." IEEE Computational Intelligence Magazine 10, no. 2 (2015) : 18-29.

[91] Denton, Emily L., Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. "Exploiting linear structure within convolutional networks for efficient evaluation." In Advances in neural information processing systems, pp. 1269-1277. 2014.

[92] Zhang, Chiyuan, Qianli Liao, Alexander Rakhlin, Brando Miranda, Noah Golowich, and Tomaso Poggio. "Theory of deep learning IIb : Optimization properties of SGD." arXiv preprint arXiv :1801.02254 (2018).

[93] Fernández-Martínez, Juan Luis Fernández-Muñiz, Zulima Tompkins, Michael. (2012). On the topography of the cost functional in linear and nonlinear inverse problems. Geophysics. 77. 1-. 10.1190/geo2011-0341.1.

[94] A. Krizhevsky, and G. Hinton, "Learning multiple layers of features from tiny images", (Technical Report) University of Toronto, 2009.

[95] Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78-87.

[96] Andrew NG, "Machine Learning Yearning. Technical strategy for AI engineers, in the era of Deep Learning", 2018.

[97] M. Abadi, A. Agarwal, et al., "Tensorflow : Large-scale machine learning on heterogeneous distributed systems", 12th Symposium on Operating Systems Design and Implementation. 2016.

[98] Zhe, Wang, Jie Lin, Vijay Chandrasekhar, and Bernd Girod. "Optimizing the Bit Allocation for Compression of Weights and Activations of Deep Neural Networks." In 2019 IEEE International Conference on Image Processing (ICIP), pp. 3826-3830. IEEE, 2019.

[99] Viola, P. and Jones, M.J. (2004) Robust real-time face detection. *International journal of computer vision*, vol. 57, no.2, pp. 137-154.

[100] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory : Eurocolt '95, pages 23–37. Springer-Verlag, 1995.

[101] Garcia, C. & Delakis, M. (2004) Convolutional face finder : A neural architecture for fast and robust face detection. *IEEE Transactions on pattern analysis and machine intelligence*, vol. 26, no. 11, pp, 1408-1423.

[102] Li, H.and Lin, Z. and Shen, X. and Brandt, J. and Hua, G. (2015) A convolutional neural network cascade for face detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5325-5334.

[103] Zhang, K. and Zhang, Z.and Li, Z. and Qiao, Y. (2016) Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499-1503.

[104] Triantafyllidou, D. and Tefas, A. (2016) A fast deep convolutional neural network for face detection in big visual data. *INNS conference on big data*, pp. 61-70.

[105] Shang, W., Sohn, K., Almeida, D. and Lee, H. "Understanding and improving convolutional neural networks via concatenated rectified linear units." In international conference on machine learning, pp. 2217-2225. 2016.

[106] Albericio, J., et al. (2016) Cnvlutin : Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1-13.

[107] Yang, S., Luo, P., Loy, C. and Tang, X. (2016). Wider face : A face detection benchmark. *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 5525-5533.

[108] Karras, T., Laine, S., & Aila, T. (2018). A style-based generator architecture for generative adversarial networks. arXiv preprint arXiv :1812.04948.

[109] Koestinger, M., Wohlhart, P., Roth, P. M., & Bischof, H.(2011) Annotated Facial Landmarks in the Wild : A Large-scale, Real-world Database for Facial Landmark Localization. *Proc. IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*.

[110] Jain, V. and Learned-Miller, E. (2010) Fddb : A benchmark for face detection in unconstrained settings. *Technical Report UMCS-2010-009*.

[111] Fernandez Brillet, L., Mancini, S., Cleyet-Merle, S. and Nicolas, M. (2019). "Tunable CNN compression through dimensionality reduction". In Press *26th IEEE International Conference on Image Processing (ICIP)*.

[112] Lavin, A. and Gray, S. (2016) Fast algorithms for convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 4013-4021.

[113] Liu, Z., Luo, P., Wang, X., & Tang, X. (2018) Large-scale celebfaces attributes (celeba) dataset.

[114] Schwambach, V., Cleyet-Merle, S., Issard, A., Mancini, S. Estimating the potential speedup of computer vision applications on embedded multiprocessors. arXiv :1502.07446, 2015.

[115] Stoutchinin, A. and Benini, L., "StreamDrive : a Dynamic Dataflow Framework for Clustered Embedded Architectures. Journal of Signal Processing Systems", 91(3-4), 275-301, 2019.

Table des figures

1 Decomposition par PCA pour un ensemble de filtres d'un réseau de l'état de l'art. 7

2 Effet du réentrainement comparé à l'application directe de PCA sur un réseau du type ResNet-32 sur CIFAR-10. 8

3 Aperçu global de la méthode. 9

4 La prédiction de boîtes englobantes (en vert) permet de réduire la complexité de l'espace de recherche par rapport à une méthode traditionnelle (en noir). 10

5 Diagrame d'execution de la solution. 11

6 Performance du réseau des propositions pour des différents niveaux de détail dans l'analyse multi-résolution. 11

7 Plusieurs compromis à travers de PCA entre taux de vrai possitifs, taux de faux possitifs et nombre de paramétres. 12

8 Comparaison de la complexité computationelle de plusieurs méthodes de détection de visage. 13

9 Performance du réseau de propositions sur l'ASMP. 14

10 Domains of competence necessary to achieve efficient CNN solutions . . . 3

11 Illustration of a discrete convolution between an 2D input image and a 2D convolution kernel. 4

12 Memory accesses required for convolution 5

13 3D stacked image sensor breakdown 5

1.1 Example showing the connection diagram for a simple one-layer fully connected network. In more detail, the operation carried out by neurons. 9

1.2 Convolution through neural networks : Output neurons for a feature map (second layer) are repeatedly connected through the same set of weights to neurons containing shifted windows of the input image. These connections are displayed in greater detail in Figure 1.3. 11

1.3 Actual connection diagram for a neural network performing a 2×2 convolution on a 9×9 input image, highlighting the operation carried out in order to compute the output of the first two elements of the feature map. 12

1.4 Different forms of invariance necessary in Computer Vision tasks. 13

1.5 Classical Convolutional Neural Network architecture as described in [11]. Image taken from [11] 14

1.6 The first row shows the effect of 2×2 pooling on a feature map. The following rows show the effect of successive pooling on scaled, translated and rotated feature-maps. From these images we can see that pooling is an effective way to keep track of the presence of strongly activated features by a filter independently of their location, scale or orientation. 15

1.7 Hierarchical feature construction by composition. Image from [12] 16

1.8 A remapping of the data in higher dimensions allows for linear separability. 17

1.9 (a) Schematic illustrating hierarchical feature construction by composition. (b) Visualization of strongest activation patterns for different layers of a state-of-the-art network. Features in the early layers strongly activate when edge or color patterns are presented, while later layers involve more abstract combinations of earlier features. Image from LeCun, Zeiler & Pete Warden 17

1.10 Example where x is progressively updated by following the derivative of the function being minimized. [Image from Deep Learning Book [6]] 19

1.11 Overshooting provoked by setting the learning rate too high. [Image from Deep Learning Book [6]] 20

1.12 Example of the effect of distribution missmatch between training and testing datasets. 23

1.13 Examples of possible bias/variance trade-offs. Image from [95]. 24

1.14 Deep learning scalability through data availability and model size. Image from [96]. 25

1.15 Backpropagation of errors in a simple graph. Inputs and outputs to each node are in green, while gradients are in red. Image from CS231, Stanford. . 26

1.16 Typical life cycle of machine learning solutions. 27

1.17 Computational Complexity of state-of-the-art CNN solutions. Image from [8]. 27

1.18 Depth estimation by a CNN through a single RGB image. 32

1.19 Toy drone controlled through wrist accelerometer.Image from aliexpress. . . 33

1.20 GPU server inside a self-driving car's trunk. 35

1.21 Crazyflie with a GAP8 microcontroller on top of it for autonomous navigation. Image. 36

1.22 Output stationary dataflow. Image from [46]. 37

1.23 Weight stationary dataflow. Image from [46]. 37

1.24 1 dimensional row stationary dataflow. Image from [46]. 37

1.25 Dataflow without any data reuse at PE level. Image from [46]. 38

2.1 Goal of object detection algorithms. 41

2.2 Sliding window search approach for object detection across multiple resolutions. 41

2.3 Sliding window across multiple resolutions for object detection. 42

2.4 Computational cost of classical object detection through minimalistic neural network. Searching for 12x12 faces in the original image would take 3.8G macc and 1.3M values of intermediary data (grayscale). By reducing the image size by a factor of x3.3 (96x72), the 40x40 face is now 12x12 and it takes 249M MAC to find (x15.26 less). 42

2.5 Main idea for RCNN. Image taken from [53] 43

2.6 Main idea for SPPNet/Fast RCNN. Image taken from [54] 44

2.7 Execution flow of Faster RCNN. Image taken from [56]. 44

2.8 Proposal and classification stages. 46

2.9 Global overview of the proposed workflow to flexibly optimize CNN-based object detector complexity on a clearly defined applicative use-case. 47

2.10 Detailed applicative specification method. 48

2.11 Growth of the search space as objects get further away from the sensor. Detecting an average sized face ($>$ 12cm) 4 meters away from a sensor with a 56° FoV would require scanning 20 pixel windows inside a 480 pixel image. 49

2.12 Detailed proposal network definition and training method. 50

2.13 A traditional sliding window search introduces great computational complexity for object detection, since the window being analyzed needs to exactly match the ground truth. This problem is also replicated accross the multiple scales of the image pyramid. 51

2.14 Advantage of bounding box regression as opposed to a traditional sliding window approach. 51

2.15 Consumer applicative scenario (a) as opposed to a surveillance scenario (b). 52

2.16 Variety in appearance of faces in the WIDER FACE dataset. Image from http ://shuoyang1213.me/WIDERFACE/. 53

2.17 Variety in appearance of faces in the CelebA dataset. 53

2.18 A whole image can be fed to a neural network during inference (a), instead of carrying out a sliding window (b). During training, it might be more convenient to use crops with the receptive field size (b), instead of storing whole images(a). 54

2.19 Detailed proposal network compression method. 56

2.20 Detailed classification network definition and training method. 57

2.21 Face versus non-face samples. 58

2.22 Detailed classification network compression method. 59

2.23 Detailed evaluation block. 60

3.1 Combination of pruning with Winograd. Figure from [71] 67

3.2 Different approaches to pruning : (a) Magnitude-based filter pruning. (b) Feature-map-based pruning. (c) Feature-map-based pruning with fine-tuning. 68

3.3 PCA decomposition for a set of filters inside a convolution layer. The top figure shows a perfect reconstruction of the original filters by keeping all basis filters, while the second figure shows an approximation obtained by keeping only the 11 most important basis filters. Best viewed in color. . . . 72

3.4 Filter reconstruction viewed as a 1x1 convolution between the basis filters f_{basis} and the coordinates f_{coord}. A kernel is reconstructed from the kernels of the PCA basis filter and its coordinates in that basis. 73

3.5 Computational Complexity is reduced by only convolving the input feature maps with a subset of the PCA basis filters and then recombining the resulting feature maps from the set of PCA coordinates. 74

3.6 Effect of cumulatively applying PCA to different layers of a ResNet-32 network on CIFAR-10. 76

3.7 Implementation of PCA reduction into the tensorflow graph (b) as compared to the original (a). The variables in (b) are initialized to the values previously obtained through PCA. 77

3.8 Effect of retraining as compared to directly applying PCA on two ResNet models on CIFAR-10. 78

3.9 Effect of retraining as compared to directly applying PCA on two ResNet models on CIFAR-100. 79

3.10 Learning curves after replacing the filters by their PCA basis, without recombination. 80

3.11 Results of further pruning PCA compressed models. 81

3.12 Structured pruning through PCA. 82

3.13 Results of structured pruning on a PCA-compressed (60% energy threshold) ResNet-32 on CIFAR-10. 82

3.14 Tensorflow graph of the layer of a network which has been decomposed by PCA, pruned and quantized. 83

3.15 Effect of further pruning the set of PCA basis and coordinates without retraining. 84

3.16 Effect of further quantizing the set of pruned PCA basis and coordinates without retraining. 84

4.1 Growth of the search space as objects get further away from the sensor. Detecting an average sized face (> 12cm) 4 meters away from a sensor with a 56° FoV would require scanning 20 pixel windows inside a 480 pixel image. 90

4.2 (a) Haar basic functions employed in VJ. (b) PCA basic functions obtained on a set of centered face crops. (c) Filters obtained in the first layer of the CNN solution presented in this chapter. 91

4.3 Detailed execution diagram of the described solution. 94

4.4 Recall and average number of proposals for different scaling factors on 1613 validation images containing faces larger than 100x100 (∼64x64 in VGA) from the challenging WIDER FACE dataset [107]. 96

4.5 Multiple trade-off points between TPR, FPR and # of parameters obtained through PCA. The database used for retraining consists of the first 13k face crops from the AFLW dataset [109] along with 13k negative samples. The remaining face crops are used for testing along with 50k negative samples. . 99

4.6 Comparison between the original(orig) and the optimized(opt) models, as well as MTCNN for multiple datasets : FFHQ, AFLW, FDDB and WIDER FACE. 100

4.7 Complexity comparison of various face detection methods. Points correspond to the avg. number of MACs per frame, while annotations correspond to the max. number of MACs on a single frame. All points have similar FPRs. 101

4.8 Example of infrared face image in an indoor environment. The subject naturally stands out in this scenario and reduces the search space. 103

4.9 Feature map sparsity on the proposed solution for a flat background example image. 103

5.1 Two different approaches for sparse accelerators : (a) direct indexing, (b) step indexing. Images taken from [69]. 107

5.2 Compressed sparse row representation. 108

5.3 Intel/Movidius Myriad2 architecture. Image taken from [70]. 108

5.4 Ternary network hardware implementation. Image taken from 109

5.5 Schematic of the ASMP. 110

5.6 Layer-wise comparison of parallelization effect on the original and optimized BCN across different number of PEs. 112

5.7 Layer-wise performance of PCA optimization comparison for the original(a), decomposed(b) and fused(c) network implementations on the ASMP multiprocessor. 113

5.8 Proposal network performance on ASMP for different input image sizes and number of PEs. 114

5.9 Proposal network performance on ASMP graphically. 115

Liste des tableaux

1 Performance du réseau de classifications sur l'ASMP. 14

3.1 Multiple trade-off points showing the effect of replacing all layers in various networks by their approximation, for different thresholds and datasets. The following stand for : E : Energy threshold, P : Parameters, CR : Compression Rate, A : Accuracy, RE : Relative Error. 100% energy represents the baseline model. 75

4.1 Comparison of state-of-the-art face detectors. 92

4.2 Proposal network architecture. 93

4.3 Binary classification network architecture. 94

4.4 Cost of proposals as a function of absolute face size (relative to 120x90 input). 96

4.5 Possible trade-offs between model complexity and performance. 97

5.1 Binary face detection network performance on ASMP multiprocessor. **Algorithmic speed-up** is measured relatively to the same architecture and number of PE. 111

5.2 Possible trade-offs between model complexity and performance. 116

"Convolutional Neural Networks for Embedded Vision"

Résumé

Ici ... résumé en français.

Mots-clés : Mots clés

Abstract

Ici ... résumé en anglais.

Keywords : Keywords