Inférence distribuée pour modèle de deep learning sur SoC-FPGA

Mathieu Hannoun[1,2] Stéphane Zuckerman[2] Olivier Romain[2]

[1]MADICOB

[2]Laboratoire ETIS, UMR 8051, CY Cergy Paris Universités, ENSEA, CNRS, F-95000, Cergy, France

Résumé – L'utilisation des réseaux de neurones profonds connaît une forte croissance depuis plusieurs années, accompagnée d'une forte augmentation de consommation d'énergie due à leur demande croissante en ressources de calcul. Pour atténuer leur impact environnemental et répondre aux préoccupations croissantes concernant la confidentialité des données, une tendance de plus en plus marquée consiste à traiter les données localement, à la périphérie, plutôt que de s'appuyer sur d'énormes *datacenters*. Les circuits reconfigurables (field-programmable gate array - FPGA), sobres en énergie et adaptés au calcul parallèle, répondent bien à ces exigences, malgré leurs ressources limitées qui restreignent la taille des DNN déployables. Cet article présente une méthodologie pour les DNN distribués sur plusieurs FPGA afin de prendre en charge des modèles qui ne sont généralement adaptés qu'à des FPGA de plus grande taille. Nous sommes en mesure de prendre en charge l'inférence d'un MobileNetV1 sur trois cartes Zedboard.

Abstract – Deep Neural Networks (DNNs) have experienced significant growth over the years, accompanied by a corresponding rise in energy consumption due to their escalating demand for computational resources. To mitigate the environmental impact of AI and address growing concerns over data privacy, a growing trend is to process data locally at the edge, rather than relying on large-scale data centers. Field-programmable gate arrays (FPGA), which are energy-efficient and well-suited for parallel computation, effectively meet these requirements, though their limited hardware resources restrict the size of deployable DNNs. This paper presents a methodology for distributed DNNs on multiple commodity FPGAs to support models that are usually only suited for larger FPGAs. We are able to support the inference for a MobileNetV1 on three Zedboards.

1 Introduction

L'inférence sur objet connecté de type *Edge* a gagné en popularité grâce à son efficacité énergétique, la localité et la confidentialité des données qu'ils procurent. Il y a une tendance à utiliser des objets *Edge* et *Fog* pour le calcul [3]. Celui-ci peut être effectué par une variété de matériels : smartphones [13], microcontrôleurs [8] et FPGA conduisant à l'essor de ce qui est appelé TinyML [1] (*Tiny Machine Learning*), soit le déploiement de modèles d'apprentissage automatique sur des dispositifs à ultra-basse consommation et à ressources limitées. Les microcontrôleurs sont une cible possible, mais s'ils répondent au besoin d'une consommation d'énergie très faible, ils n'ont pas la même capacité de calcul parallèle offerte par les FPGA, et souffrent des mêmes inconvénients d'avoir trop peu de ressources pour prendre en charge des modèles plus importants. Les FPGA ne sont pas adaptés aux réseaux de neurones convolutionnels ou réseaux profonds (DNN) à grande échelle, en raison du nombre massif d'opérations en virgule flottante qu'ils requièrent. Pour résoudre ce problème, de nombreuses techniques ont été développées ces dernières années dans des outils comme FINN [12], allant de la compression des poids par quantification jusqu'à l'utilisation d'un ou deux bits seulement pour les poids et les biais [5]. La conversion des opérations a également été développée pour tirer parti de ce niveau de compression [4, 10]. Dès lors, les FPGA haut de gamme (p.e. AMD Alveo) ont été explorés ces dernières années pour l'inférence de DNN, mais leur utilisation est principalement limitée aux data centers, vue leur consommation d'énergie et de leur prix. Les FPGA génériques offrent un compromis prometteur : ils allient une faible empreinte énergétique à un fort parallélisme pour les opérations matricielles des DNN. Cependant, les FPGA de base offrent des ressources matérielles limitées, ce qui rend la mise en œuvre des DNN difficile pour les modèles plus grands que quelques couches binarisées.

L'objectif de cet article est de démontrer la faisabilité de la division de modèles plus importants (*model splitting*), qui ne peuvent s'adapter pleinement qu'à des FPGA haut de gamme (UltraScale+, Alveo, *etc.*), sur un cluster de FPGA pris sur étagère, à l'aide de FINN. Cela pourrait éventuellement conduire à une solution distribuée à faible consommation, capable de s'adapter à une large gamme de DNN et de tâches.

Cet article propose un moyen de distribuer l'inférence des modèles de réseaux DNN sur plusieurs FPGA lorsqu'une seule carte n'est pas suffisante pour le contenir en entier. Nous tirons parti de la reconfiguration dynamique d'un système SoC-FPGA. Les communications sont gérées par le CPU, tandis que l'inférence est effectuée par le FPGA.

2 Contexte

2.1 FPGA et reconfiguration dynamique

Les FPGA (*Field-Programmable Gate Array*) sont des dispositifs matériels reconfigurables qui offrent une grande souplesse dans la réalisation de circuits numériques. Contrairement aux ASIC (*Application Specific Integrated Circuits*), les FPGA peuvent être reprogrammés après fabrication, ce qui les rend adaptés à un large éventail d'applications, du prototypage au déploiement dans des systèmes adaptatifs. L'un des principaux avantages des FPGA est leur capacité à être reconfigurables dynamiquement : ils peuvent modifier la logique matérielle en cours d'exécution sans arrêter le système. Ainsi, le système résultant peut s'adapter en en fonction de la charge de travail.

2.2 Le framework FINN

FINN [4] est un framework open source qui convertit un modèle ONNX (*Open Neural Network Exchange*) d'un réseau de neurones profond (DNN) en une implémentation FPGA. L'outil utilise plusieurs étapes pour convertir le modèle, résultant en de multiples représentations ONNX. Les étapes comprennent : la simplification du modèle ; la conversion des couches pour un fonctionnement adapté au matériel ; l'ajout de FIFO et de moteurs DMA personnalisés ; la génération de code de synthèse de haut niveau (HLS) pour chaque IP ; la création de projets Vivado et l'assemblage des différentes IP ; et la synthèse pour générer un *bitstream* puis, en option, un pilote Pynq.

3 Problématique

Nous cherchons à savoir comment intégrer les DNN dans les dispositifs embarqués tout en tirant parti de l'aptitude des FPGA à l'accélération matérielle et de leur faible consommation d'énergie. En outre, nous nous intéressons à la possibilité de remplacer un FPGA haut de gamme dédié par un ensemble de FPGA génériques basse consommation mis en réseau.

4 Méthodologie

Notre approche permet de déployer des DNN, généralement adaptés aux FPGA haut de gamme (p.e. Alveo U250) en raison de leur taille, sur des FPGA génériques en divisant le modèle en plusieurs sous-modèles. Ces sous-modèles sont distribués sur un réseau de SoC-FPGA. Notre méthodologie fournit une approche flexible et évolutive de l'inférence basée sur les FPGA, permettant l'exécution de modèles plus importants sur du matériel à faible coût. Bien qu'elle n'atteigne pas encore les performances des FPGA haut de gamme, elle représente une étape vers des solutions d'inférence distribuées plus accessibles et plus efficaces. Notre travail utilise FINN comme base pour l'implémentation matérielle afin d'exécuter les inférences. Comme expliqué dans la section 2.2, FINN peut convertir un modèle DNN pour cibler un FPGA. Il prendra un modèle ONNX représentant l'architecture DNN et ses poids, le convertira en de multiples représentations intermédiaires et produira un bitstream comprenant le modèle et les moteurs d'accès direct à la mémoire (DMA) sur mesure. Cependant, FINN est prévu pour prendre un modèle complet pour produire une configuration qui tiendra dans un seul FPGA. Notre objectif est de répartir le modèle en sous-modèles plus petits pouvant chacun tenir dans un FPGA générique et communiquer entre eux pour compléter le modèle global. Ainsi, nous divisons le modèle ONNX représentant le DNN, utilisons FINN pour générer un bitstream à partir de chaque sous-modèle, employons ces bitstreams pour configurer chaque carte, et établissons une communication avec chaque carte pour exécuter des inférences.

4.1 Division du réseau

Pour permettre aux modèles DNN de taille moyenne de tenir sur des FPGA à faible empreinte, il est nécessaire de les diviser pour s'adapter aux contraintes de ressources. FINN [4] est utilisé pour générer un modèle prétraité et une estimation initiale des ressources. Le fractionnement qui en résulte est basé sur ces modèles. L'objectif de notre séparateur automatisé est de maximiser l'occupation des ressources pour réduire le nombre de bitstreams, car la reconfiguration dynamique et les opérations réseau sont les plus coûteuses en temps (détaillées dans la section 5.3). Pour l'instant, notre seul objectif est de maximiser l'occupation des ressources afin de réduire le nombre de bitstreams (à l'avenir, d'autres mesures et objectifs pourront être ciblés, tels que la quantité de données transmises entre les couches afin de réduire le trafic sur le réseau). Pour trouver une coupe appropriée, le graphe ONNX est parcouru, en commençant par la première couche. Pour chaque nœud, l'estimation des ressources nécessaires (LUT, BRAM, DSP) est ajoutée, jusqu'à ce qu'elle dépasse les ressources disponibles pour une carte donnée. Les manipulations ONNX sont ensuite utilisées pour diviser le modèle prétraité. Les modèles résultants sont ensuite transmis à FINN pour générer une conception FPGA ainsi qu'un bitstream (dont l'empreinte est négligeable en mémoire) pour chaque sous-modèle.

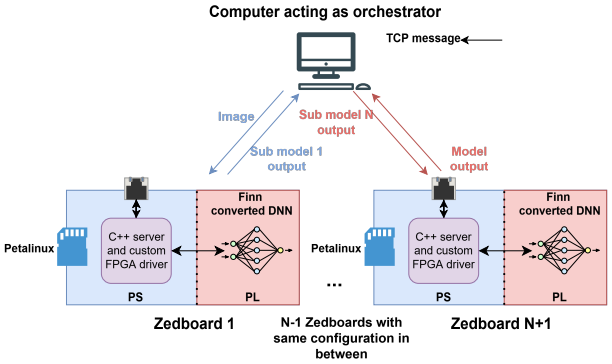

FIGURE 1 : Communication complète pour un DNN divisé en 3 bitstreams, d'une entrée d'image à la classification du modèle (sortie du sous-modèle 3), de droite à gauche séquentiellement.

4.2 Conception du système

Chaque carte dispose de son propre serveur. Un client unique joue le rôle d'orchestrateur, qui lancera de nouvelles demandes de reconfiguration et de transferts de données d'entrée, et collectera les résultats de tous les travailleurs-FPGA interrogés (*cf.* Fig. 1). Un client peut attribuer un DNN à la volée en demandant à la carte sélectionnée de reconfigurer sa logique programmable (PL) avec le sous-modèle souhaité. Les sous-modèles sont générés au préalable (*cf.* section 4.1) et chaque carte a tous les sous-modèles stockés localement. Chaque carte peut être interrogée indépendamment par l'orchestrateur. Les communications sont gérées par un serveur TCP écrit en C++ et fonctionnant du côté du système de traitement (PS), comme le montre la Fig. 1. Le serveur interagit directement avec la partie PL pour déclencher des inférences réseau par l'intermédiaire du processeur ARM.

Les DNN sont entièrement exécutés sur le FPGA, sans contribution du processeur. Le client peut déclencher à distance la reconfiguration dynamique du PL, à condition qu'il existe un bitstream du DNN requis sur la carte. Tous les bitstreams sont conservés localement sur chaque carte. Le client peut également envoyer des données à traiter par le DNN implémenté sur la partie PL et recevoir en retour la sortie du DNN du serveur. Pour l'instant, le système est centralisé, avec un orchestrateur unique interrogeant chaque carte et centralisant les résultats.

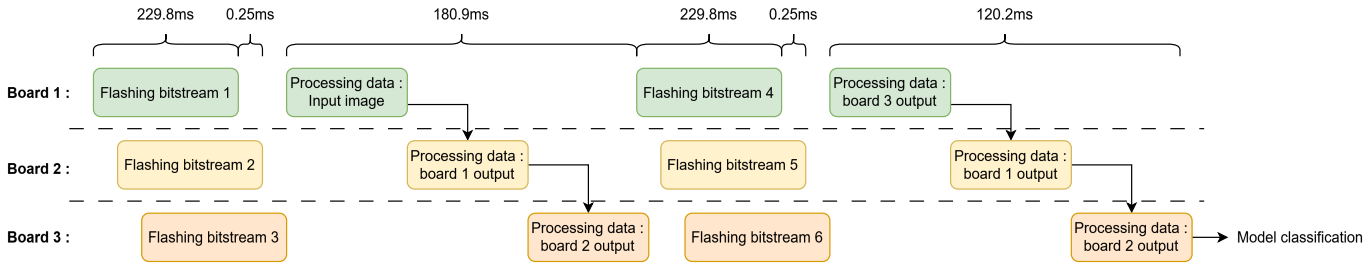

FIGURE 2 : Tâches nécessaires au traitement d'une image pour le MobileNetV1 à l'aide de trois cartes pour six bitstreams.

TABLE 1 : DNN : Ressources requises (après le prétraitement FINN utilisant la configuration de pliage par défaut de FINN) *vs.* disponibilité des ressources sur différents FPGA.

Modèle | Plateforme / Resource	CnvW2A2	MobileNetV1	Zedboard	Alveo U280
BRAMs	202 × 18 Kb	1058 × 18 Kb	140 × 36 Kb	1490 × 36 Kb
LUTs	18 836	67 090	53 200	1 082 000
DSPs	0	748	220	8490

5 Résultats expérimentaux

5.1 Base d'essai expérimentale

Notre système est composé de 3 Zedboards (utilisant un Zynq 7020) connectées à un switch Ethernet 100 Mbps. Un PC portable sert d'orchestrateur. Le Zynq 7020 est un système hétérogène avec un processeur bi-cœur (Cortex-A9) couplé à un FPGA (Artix-7). Chaque carte est configurée avec une version identique de Petalinux v2024.1. Chaque carte est indépendante, avec un serveur pouvant être interrogé pour l'inférence d'un sous-modèle qu'il héberge sur sa PL. Un PC portable joue le rôle de client et envoie des données à chaque carte de manière séquentielle (*cf.* Fig. 1 et 2). Le temps a été mesuré côté client pour tenir compte de toutes les transmissions sur le réseau. Les temps totaux de traitement sont bien plus importants que les temps de transfert.

5.2 CnvW2A2

Le premier DNN testé dans le cadre de notre configuration est un réseau de neurones binaires (BNN) appelé CnvW2A2. Il s'agit d'un réseau de neurones convolutif quantifié tel que ses poids et ses activations ont une résolution de 2 bits. Seules son entrée et sa sortie sont sur 8 bits. Le modèle comprend 6 couches de convolution et 3 couches entièrement connectées. Il a été entraîné avec l'ensemble de données CIFAR-10 [9]. Ses entrées sont des images de dimension $32 \times 32 \times 3$. Ce DNN a été adapté à une carte Pynq, et peut donc tenir sur une Zedboard (*cf.* Tab. 1 pour les ressources requises). Ce modèle a été utilisé uniquement comme preuve de concept de notre architecture. Il a été séparé manuellement en deux sous-réseaux à la jonction entre les couches convolutives et les couches entièrement connectées. Deux cartes sont utilisées ,chacune exécutant une partie du DNN. Cette architecture offre un débit de 123,8 inférences par seconde (IPS). Notre approche atteint un débit inférieur à celui de Fiscaletti [6] qui atteignent 719.56 IPS avec 3 Pynq. Avec l'optimisation du placement et du routage de la logique sur le FPGA, la maximisation des ressources et une topologie de réseau plus directe, nous pourrions atteindre des performances plus élevées ; ce point sera abordé dans des travaux futurs.

5.3 MobileNetV1

MobileNetV1 [7] est un CNN plus important avec 3,22 millions de paramètres. Le modèle est également moins quantifié (4 bits pour les poids et l'activation contre seulement 2 pour CnvW2A2). Les ressources nécessaires pour ce DNN sont indiquées dans la table 1. Les ressources de l'Alveo U280 sont divisées en 4 tuiles FPGA ; le nombre donné est le total des ressources. L'Alveo U280 dispose également de 960×288 Kb UltraRAMs.

En utilisant notre méthode de découpage, nous avons obtenu 6 bitstreams qui peuvent chacun tenir dans un seul FPGA. N'ayant que 3 cartes dans notre cluster, nous avons choisi d'inclure la reconfiguration dynamique dans le processus d'inférence (*cf.* Fig. 2).

Les reconfigurations dynamiques (DR) peuvent être émises séquentiellement par le client, mais une fois envoyées, chaque DR sera exécutée en parallèle sur chaque carte (*cf.* la Fig. 2 pour la visualisation des séquences de tâches). L'envoi d'une demande de DR prend entre 0,2 ms et 0,3 ms. Chaque DR prend 229,8 ms pour être entièrement exécuté et pour que la confirmation soit envoyée au client. Deux lots de 3 DR doivent être exécutés pour chaque classification d'image, soit une durée totale de 459,6 ms. En comparaison, les six ordres de traitement ne prennent que 307,9 ms. Le temps de traitement moyen pour une image est de 767,9 ms. La Fig. 3 décompose le temps de calcul et de reconfiguration pour chaque bitstream. Cela montre le coût élevé de la reconfiguration. Il pourrait être réduit en groupant les images en lot, réduisant ainsi le nombre de reconfigurations à une par lot (au lieu d'une par image). Les performances s'en trouveraient considérablement accrues, mais la latence du système augmenterait, théoriquement jusqu'à $taille\,du\,lot \times 307,9ms + 459,6ms$. L'inférence basée sur le FPGA pour une seule image ne durant que 28,50 ms, l'optimisation des opérations de réseau pourrait être un facteur important d'amélioration des performances.

Pour l'instant, le débit ne convient qu'aux applications n'étant pas soumises à des contraintes de temps strictes. Néanmoins, il s'agit d'un système fonctionnel, évolutif et capable de prendre en charge des DNN nécessitant plusieurs fois leurs ressources individuelles.

TABLE 2 : Comparaison avec les travaux existants sur MobileNetV1

Référence	FPGA	Nb cartes	Débit moy. (img/s)
Nous	Zedboard	3	1.302
[11]	Zynq UltraScale+	1	17.85
[2]	Alveo U280	1	3741
[2]	Alveo U280	2	5923

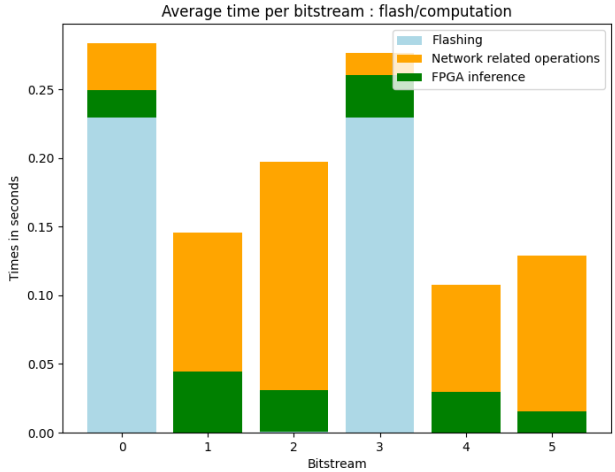

FIGURE 3 : Temps moyen par bitstream pour une inférence : reconfiguration, calcul FPGA (inférence de sous-modèle) et opérations de réseau.

6 État de l'art

Ficaletti et al. [6] ont exploité FINN pour diviser un réseau de CNN binarisé (CnvW2A2, *cf.* section 5.2). Ils dupliquent les couches de calcul lourdes sur deux cartes, puis envoient les sorties au reste des couches implémentées sur une troisième carte afin d'améliorer le débit global de l'architecture. Notre travail développe leur idée en prenant en charge des DNN plus importants. Leur travail sur l'optimisation de l'empreinte PL n'entre pas dans le cadre du présent document, mais sera exploré dans des travaux futurs. Alonso et al. [2] se sont concentrés sur le partitionnement et l'optimisation des ressources pour diviser un MobileNetV1 et un Resnet50 sur deux cartes haut de gamme (Alveo, *cf.* Tableau 1). Cette solution a un débit extrêmement élevé mais vise principalement les *data centers*.

De nombreux travaux traitent également l'implémentation de MobileNetV1 sur FPGA, mais se concentrent sur l'utilisation d'un matériel plus puissant. Par exemple, Sharma et al. [11] utilisent des cartes Zynq UltraScale+.

7 Conclusion

Dans cet article, nous avons présenté une méthodologie permettant de répartir les réseaux neuronaux sur plusieurs systèmes SoC-FPGA de base et de tirer parti de la reconfiguration dynamique pour prendre en charge des modèles plus importants sur un nombre limité de cartes. Cette solution prend en charge un nombre arbitraire de bitstreams et de cartes. Nos premiers résultats sont prometteurs, avec un débit de 123 inférences par seconde pour un DNN ultra léger et la prise en charge de MobileNetV1.

Les travaux futurs comprennent l'optimisation des communications grâce à une architecture en pipeline, la partition de réseaux neuronaux plus importants sur un plus grand nombre de puces FPGA, l'exploration des différents compromis pour découper de tels réseaux, p.e. l'utilisation maximale des ressources par carte, ou le type de ressources utilisées, *etc.*, ainsi que la contribution de différents FPGA à un même réseau de neurones plus important sur un système hétérogène.

Références

[1] Norah N. ALAJLAN et Dina M. IBRAHIM : TinyML : Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications. *Micromachines*, 13(6):851, juin 2022.

[2] Tobias ALONSO, Lucian PETRICA, Mario RUIZ, Jakoba PETRI-KOENIG, Yaman UMUROGLU, Ioannis STAMELOS, Elias KOROMILAS, Michaela BLOTT et Kees VISSERS : Elastic-df : Scaling performance of dnn inference in fpga clouds through automatic partitioning. *ACM Trans. Reconfigurable Technol. Syst.*, 15(2), décembre 2021.

[3] Hany F. ATLAM, Robert J. WALTERS et Gary B. WILLS : Fog Computing and the Internet of Things : A Review. *Big Data and Cognitive Computing*, 2(2), juin 2018.

[4] Michaela BLOTT, Thomas B PREUßER, Nicholas J FRASER, Giulio GAMBARDELLA, Kenneth O'BRIEN, Yaman UMUROGLU, Miriam LEESER et Kees VISSERS : FINN-R : An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfigurable Technol. Syst.*, 11(3):1–23, 2018.

[5] Matthieu COURBARIAUX, Itay HUBARA, Daniel SOUDRY, Ran EL-YANIV et Yoshua BENGIO : Binarized Neural Networks : Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1, mars 2016. arXiv :1602.02830.

[6] Giorgia FISCALETTI, Marco SPEZIALI, Luca STORNAIUOLO, Marco D. SANTAMBROGIO et Donatella SCIUTO : BNNsplit : Binarized neural networks for embedded distributed FPGA-based computing systems. *In 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 975–978, 2020. ISSN : 1558-1101.

[7] Andrew G HOWARD : Mobilenets : Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv :1704.04861*, 2017.

[8] Massimo MERENDA, Carlo PORCARO et Demetrio IERO : Edge Machine Learning for AI-Enabled IoT Devices : A Review. *Sensors*, 20(9), janvier 2020.

[9] Alex Krizhevsky Vinod NAIR et Geoffrey HINTON : Cifar-10 dataset. Accessed : 2025-01-28.

[10] Mohammad RASTEGARI, Vicente ORDONEZ, Joseph REDMON et Ali FARHADI : XNOR-Net : ImageNet Classification Using Binary Convolutional Neural Networks. *In* Bastian LEIBE, Jiri MATAS, Nicu SEBE et Max WELLING, éditeurs : *ECCV 2016*, pages 525–542, Cham, 2016.

[11] Aman SHARMA, Vijander SINGH et Asha RANI : Implementation of CNN on Zynq based FPGA for Real-time Object Detection. *IEEE Internet of Things Journal*, pages 1–7, juillet 2019.

[12] Yaman UMUROGLU, Nicholas J. FRASER, Giulio GAMBARDELLA, Michaela BLOTT, Philip LEONG, Magnus JAHRE et Kees VISSERS : FINN : A Framework for Fast, Scalable Binarized Neural Network Inference. FPGA '17, page 65–74, New York, NY, USA, 2017. ACM.

[13] Tahmina ZEBIN, Patricia J. SCULLY, Niels PEEK, Alexander J. CASSON et Krikor B. OZANYAN : Design and Implementation of a Convolutional Neural Network on an Edge Computing Smartphone for Human Activity Recognition. *IEEE Access*, 7:133509–133520, 2019.