# Analyse et Implémentation de Réseaux de Neurones Profonds Binarisés

**Roan Rubiales** and **Quentin Luquet de Saint-Germain**
Polytechnique Montréal
{roan.rubiales, quentin.luquet-de-saint-germain}@polymtl.ca
**Dépot GitHub** : https://github.com/Krumsky/BinnaryNeuralNetWork_INF8225

## Abstract

*La demande croissante pour des modèles d'intelligence artificielle déployables sur des appareils aux ressources limitées (smartphones, systèmes embarqués) a stimulé la recherche sur la compression de réseaux de neurones profonds. La binarisation, une forme extrême de quantification, représente une approche prometteuse en contraignant les poids et/ou les activations à des valeurs binaires (+1, -1), réduisant drastiquement l'empreinte mémoire et le coût computationnel. Ce rapport présente notre analyse et implémentation de réseaux de neurones binarisés (BNN). Nous explorons l'impact de la binarisation sur différentes architectures (MLP, CNNs) et datasets (MNIST, CIFAR-10). Nous discutons des défis rencontrés, présentons nos résultats expérimentaux et proposons une analyse critique de notre démarche d'apprentissage et d'implémentation, basée sur l'entraînement conscient de la quantification (QAT) avec des couches personnalisées en PyTorch.*

## 1 Introduction

Les réseaux de neurones profonds (DNN) ont atteint des performances remarquables dans de nombreux domaines, mais leur taille et leur complexité computationnelle limitent souvent leur déploiement sur des plateformes aux ressources contraintes comme les appareils mobiles ou les systèmes embarqués. La compression de modèles est donc devenue un axe de recherche majeur. Parmi les techniques de compression, la quantification, et plus particulièrement sa forme extrême, la **binarisation**, suscite un intérêt croissant.

La binarisation consiste à représenter les poids synaptiques et/ou les activations neuronales en utilisant seulement 1 bit, typiquement avec les valeurs {-1, +1} [Hubara *et al.*, 2016]. Cette approche offre des avantages théoriques considérables : une réduction drastique de la taille mémoire (jusqu'à 32x par rapport aux flottants 32 bits) et la possibilité de remplacer les coûteuses multiplications par des opérations logiques XNOR, beaucoup plus rapides et économes en énergie [Rastegari *et al.*, 2016]. Ces gains rendent les réseaux de neurones binarisés (BNN) particulièrement attractifs pour les applications en périphérie (edge AI).

Plusieurs travaux fondateurs ont exploré la binarisation. Hubara et al. [Hubara *et al.*, 2016] ont introduit les "Binarized Neural Networks" (BNN), démontrant leur faisabilité sur des tâches de classification d'images simples et proposant une méthode d'entraînement basée sur un estimateur direct (Straight-Through Estimator - STE) pour gérer le problème des gradients nuls des fonctions de binarisation. Rastegari et al.[Rastegari *et al.*, 2016] ont proposé XNOR-Net, qui binarise à la fois les poids et les entrées des couches convolutives et introduit des facteurs d'échelle pour améliorer la précision sur des tâches plus complexes comme ImageNet. D'autres approches comme DoReFa-Net [Zhou *et al.*, 2016] ou des méthodes plus récentes comme FracBNN [Zhang *et al.*, 2021] ont cherché à affiner le processus de quantification et d'entraînement pour combler l'écart de performance avec les réseaux pleine précision.

Cependant, la binarisation pose des défis significatifs. La perte d'information due à la quantification extrême entraîne souvent une dégradation notable de la précision par rapport aux modèles flottants [Hubara *et al.*, 2016; Rastegari *et al.*, 2016]. L'entraînement est également plus complexe et sensible aux hyperparamètres.

L'objectif de notre projet était d'analyser et d'implémenter des réseaux de neurones profonds binarisés afin de comprendre en profondeur l'impact de cette technique. Nous avons cherché à :

- Implémenter les mécanismes de binarisation des poids et des activations dans un framework moderne (PyTorch) en utilisant l'approche QAT (Quantization-Aware Training).
- Évaluer l'effet de la binarisation sur la performance de différentes architectures (MLP, CNNs comme VGG, Resnet-18 et MobileNetV2).
- Mener des expériences sur des datasets standards (MNIST, CIFAR-10).
- Analyser le compromis entre efficacité (taille/calcul) et précision.

Ce rapport détaille notre approche théorique, nos implémentations, les résultats expérimentaux obtenus, et une analyse critique de notre travail.

## 2 Approche Théorique

La base théorique de notre projet repose sur la binarisation des réseaux de neurones et l'entraînement conscient de la quantification (QAT).

### 2.1 Binarisation des Weights et Activations

Le principe fondamental de la **binarization** dans les réseaux de neurones est de contraindre la représentation numérique des paramètres (**poids**) et/ou des signaux intermédiaires (**activations**) à seulement deux valeurs possibles. L'idée centrale est de remplacer les valeurs réelles, typiquement stockées en format floating point 32 bits ($w \in \mathbb{R}, a \in \mathbb{R}$), par des valeurs binaires appartenant à l'ensemble $\{-1, +1\}$ ($w_b \in \{-1, +1\}, a_b \in \{-1, +1\}$) [Hubara *et al.*, 2016].

Cette transformation est généralement réalisée lors du forward pass de l'entraînement à l'aide d'une fonction déterministe. La plus courante est la fonction **sign** :

$$\text{sign}(x) = \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{sinon} \end{cases}$$

La valeur réelle $x$ (weight ou activation) est alors remplacée par $\text{sign}(x)$. Les avantages de cette approche sont considérables :

- **Réduction du Memory Footprint :** Chaque weight ou activation ne nécessite plus qu'un seul bit de stockage, contre 16 ou 32 bits pour les formats **half precision** ou **single precision**. Cela représente une compression mémoire potentielle allant jusqu'à 32x pour les weights.
- **Efficacité Computationnelle :** Si les weights *et* les activations sont binarisés, l'opération dominante (**Multiply-Accumulate**) peut être remplacée par des opérations **bit-wise XNOR** (pour la partie multiplication) suivies d'accumulations simples. Ces opérations sont nativement beaucoup plus rapides et moins énergivores sur le hardware (CPUs, FPGAs) [Rastegari *et al.*, 2016]. La binarisation des weights seule réduit déjà la mémoire, mais l'accélération majeure provient de la binarisation conjointe poids-activations.

Dans le cadre de notre projet, nous avons implémenté ce **forward pass** de binarisation via nos fonctions `WeightBinarizeFunc` et `ActivationBinarizeFunc` qui appliquent donc cette logique de $\text{sign}(x)$ pour obtenir les valeurs $w_b$ et $a_b$ utilisées dans les calculs d'inférence du réseau.

### 2.2 Quantization-Aware Training (QAT)

L'entraînement des réseaux de neurones binarisés présente un défi majeur : la fonction de binarisation (comme `sign(x)`) possède une dérivée nulle presque partout. Or, l'algorithme standard de rétropropagation repose sur le calcul des gradients pour mettre à jour les poids du réseau. Un gradient nul bloque ce processus d'apprentissage.

Pour surmonter cet obstacle, la technique la plus répandue est le **Straight-Through Estimator(STE)** [Hubara *et al.*, 2016]. Le principe du STE est de gérer différemment les passes avant et arrière :

1. **Forward Pass :** Utiliser les valeurs binarisées ($w_b, a_b$) pour tous les calculs, comme décrit précédemment. Cela garantit que le réseau opère avec la représentation cible (binaire).
2. **Backward Pass :** Lors du calcul des gradients destinés à mettre à jour les poids, ignorer la dérivée nulle de la fonction de binarisation et utiliser à la place un gradient **substitut**. Ce gradient substitut est souvent celui d'une fonction plus simple et différentiable :
    - **Identité :** Considérer que le gradient de $\text{sign}(x)$ est 1. C'est l'approche la plus simple.
    - **Clipping :** Considérer que le gradient est 1 seulement si l'entrée $x$ de la fonction de binarisation était dans un intervalle défini ([-1, 1]), et 0 sinon. Cela évite de propager des gradients pour des poids ou activations qui sont déjà "saturés".

Dans notre implémentation, la méthode `backward` de `WeightBinarizeFunc` implémente le STE basé sur l'identité (elle retourne simplement `grad_output`). Pour `ActivationBinarizeFunc`, sa méthode `backward` implémente une forme de STE avec **clipping**, en annulant le gradient (`out[ctx.saturation_mask] = 0`) si l'entrée originale avait une valeur absolue supérieure à 1.

Cette logique de STE est intégrée au sein de l'approche globale de **Quantization-Aware Training**. Dans notre framework, les couches personnalisées `QATLinear` et `QATConv2d` jouent un rôle clé :

- Elles maintiennent en interne une version des **poids** en précision flottante. Ce sont ces poids flottants qui sont mis à jour par l'**optimizer** (Adam, SGD) en utilisant les gradients approximés par le STE.
- Lors du **forward pass**, ces couches utilisent la fonction `weightBinarize` pour obtenir la version binarisée ($w_b$) des poids flottants internes, et ce sont ces poids binarisés qui sont utilisés pour le calcul de la sortie de la couche ($y = \text{Conv}(x, w_b)$ ou $y = \text{Linear}(x, w_b)$).
- Elles incluent également une fonction `clip` appliquée aux poids pendant l'entraînement. Cette fonction contraint les poids flottants à rester dans une plage dynamique pertinente pour la binarisation (par exemple, [-1, 1] pour `dbits=1`). Cela aide à stabiliser l'entraînement et à garantir que la valeur binarisée est une représentation fidèle du poids flottant sous-jacent.

L'essence de l'approche **QAT** est donc de simuler l'effet de la quantification (binarisation) pendant l'entraînement. En exposant le modèle à l'erreur de quantification lors de la passe avant tout en permettant une mise à jour des poids via le STE lors de la passe arrière, le modèle apprend à ajuster ses poids flottants latents pour devenir plus robuste à la perte d'information induite par la binarisation au moment de l'inférence.

### 2.3 Architectures Étudiées

Nous avons appliqué l'approche **QAT** décrite précédemment à différentes architectures de réseaux de neurones afin d'évaluer son impact sur diverses tâches et complexités de modèles :

- **Perceptron Multi-Couches (MLP) :** Un réseau simple entièrement connecté, utilisé comme première validation de notre implémentation QAT sur le dataset MNIST. Il utilise principalement la couche `QATLinear`.
- **Réseaux Convolutifs (CNNs) :** Pour la tâche de classification d'images plus complexe CIFAR-10, nous avons implémenté et étudié deux architectures CNN représentatives en utilisant notre approche QAT :
  - **VGG-style :** Une architecture inspirée de VGG, utilisant une séquence de couches `QATConv2d` suivies de max-pooling, et un classifieur basé sur `QATLinear`. Cela nous a servi de baseline pour une architecture CNN profonde classique.
  - **MobileNetV2 :** Une architecture reconnue pour son efficacité, notamment sur les appareils mobiles, grâce à ses blocs **inverted residual** et ses convolutions **depthwise separable** [Sandler *et al.*, 2018].
  - **FracBNN/ResNet :** Une implémentation appliquant les principes de FracBNN [Zhang *et al.*, 2021] (visant à améliorer la précision des BNNs) sur une base d'architecture ResNet modifiée.

## 3 Expériences et Résultats

### 3.1 Configuration Expérimentale

Nos expériences ont été conduites en utilisant le framework **PyTorch**.

**Datasets :**

- **MNIST :** Pour les expériences initiales avec le MLP. Dataset d'images de chiffres manuscrits en niveaux de gris (28x28 pixels, 10 classes). Le pré-traitement incluait la conversion en tenseur et la normalisation.
- **CIFAR-10 :** Pour l'évaluation des architectures CNN (VGG-style, MobileNetV2). Dataset d'images couleur (32x32 pixels, 10 classes). Le pré-traitement incluait la conversion en tenseur, la normalisation (avec les moyennes et écarts-types standards pour CIFAR-10), et des techniques de **data augmentation** pendant l'entraînement ('RandomHorizontalFlip', 'RandomCrop' avec padding).

Les données étaient chargées via les 'DataLoader' de PyTorch, gérés par notre classe `Builder` (dans `model_training/builder.py`).

**Modèles :**

- Un **MLP** simple avec des couches `QATLinear`.
- Une architecture **VGG-style** utilisant des couches `QATConv2d`, `BatchNorm2d`, `ReLU` et `MaxPool2d`, avec un classifieur basé sur `QATLinear`.
- L'architecture **MobileNetV2_QAT** qui intègre les couches `QATConv2d` et `QATLinear` au sein des blocs **Inverted Residual** et du classifieur.
- Une implémentation basée sur les principes de **FracBNN** appliquée à une architecture **ResNet** modifiée.

Tous les modèles héritent de notre classe `ModelBase` et utilisent nos modules QAT personnalisés (`qnn/modules.py`, `qnn/models/`).

**Entraînement :**

- **Optimiseur :** Choix entre **Adam** et **SGD** (avec momentum), configurable via le fichier TOML.
- **Fonction de Coût :** Entropie croisée, standard pour la classification multi-classes.
- **Planificateur de Taux d'Apprentissage :** Utilisation de `MultiStepLR` (réduction du learning rate à des époques prédéfinies) configurable via le fichier TOML.
- **Hyperparamètres :** Les paramètres cruciaux comme le learning rate initial, le nombre d'epochs, la taille du batch sont définis dans des fichiers de configuration au format **TOML** (par exemple, `config/mobilenetv2_cifar10.toml`).
- **QAT :** L'entraînement simule la quantification à chaque itération grâce aux couches QAT.

**Métriques enregistré:**

- La **loss** moyenne sur l'ensemble d'entraînement.
- La **loss** moyenne sur l'ensemble de validation.
- L'**accuracy Top-1** et **Top-5** sur l'ensemble de validation.

**Environnement Matériel et Logiciel :** Les entraînements ont été accélérés en utilisant des GPUs NVIDIA. L'environnement logiciel était géré par Conda.

### 3.2 Résultats Obtenus

Cette section présente les résultats quantitatifs et qualitatifs obtenus lors de nos différentes expériences d'entraînement avec l'approche **QAT**.

**MLP sur MNIST**

Notre première série d'expériences visait à valider l'implémentation QAT sur une tâche simple avec un modèle MLP avec des couches `QATLinear` sur MNIST.

| Métrique (MLP MNIST, dbits=1) | Valeur Finale |
|---|---|
| Validation Loss | 2.078 |
| Validation Accuracy (Top-1) | 98.57 % |

Table 1: Performances finales du MLP QAT (dbits=1) sur MNIST.

**Analyse :** L'entraînement du MLP binarisé sur MNIST a permis au modèle de converger avec succès, atteignant une accuracy finale sur l'ensemble de validation de 98.57%. Cette performance est très proche de la baseline (>98-99%) pour

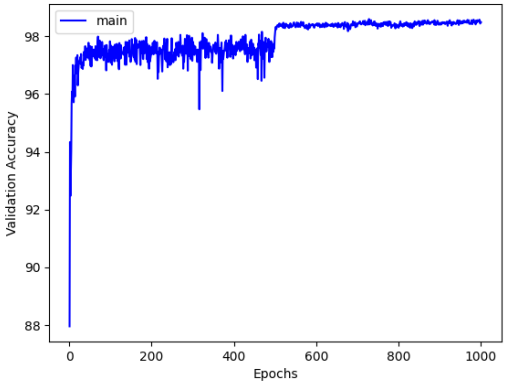


Figure 1: Évolution de l'accuracy sur MNIST Validation pour le MLP QAT (dbits=1).

des MLPs standards non quantifiés, démontrant l'efficacité de notre approche QAT pour cette tâche et cette architecture simples, même avec une binarisation extrême.

La dynamique d'apprentissage a montré une montée rapide à 97% d'accuracy, suivie d'une longue phase de stagnation. Ce plateau est probablement dû à la complexité du paysage d'optimisation induit par QAT/STE. Le saut de performance observé tardivement (après l'époque 500) pour dépasser les 98% coïncide vraisemblablement avec une réduction du taux d'apprentissage déclenchée par le scheduler utilisé avec l'optimiseur SGD, permettant de raffiner la solution dans une région plus prometteuse.

### VGG-style sur CIFAR-10

Nous avons ensuite évalué une architecture CNN plus profonde de type VGG, intégrant les couches `QATConv2d` et `QATLinear`, sur le dataset CIFAR-10.

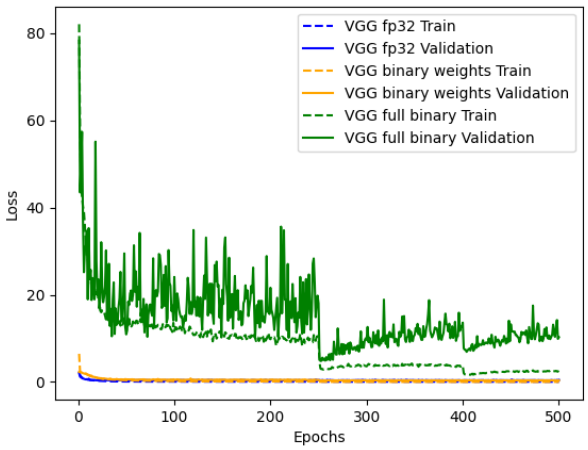


Figure 2: Évolution de la loss (Train/Validation) pour VGG sur CIFAR-10.

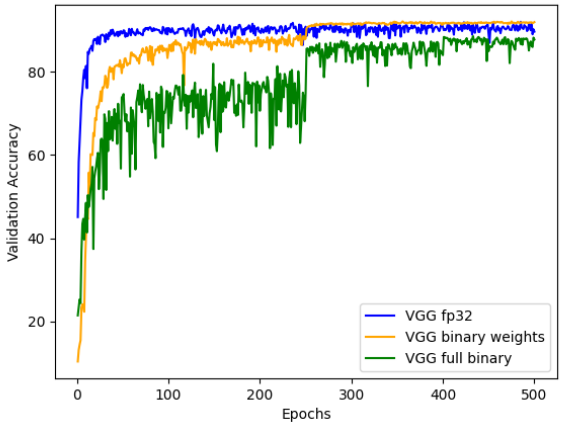


Figure 3: Évolution de l'accuracy Top-1 sur CIFAR-10 Validation pour VGG.

**Analyse :** On utilise un optimiseur SGD avec un learning rate de 0.1 initial et divisé par 10 à l'epoch 250, et à l'epoch 400. On obtient une précision maximale de 92% (Fig. 3), proche de l'état de l'art, mais pas exactement car nous n'utilisons pas de dropout. La binarisation des poids atteint une précision de 91% avec plus d'instabilité. Cependant, le VGG utilisant une binarisation totale atteint 88%, une précision dégradée par rapport à la baseline, avec une instabilité d'entraînement bien plus grande.
De tels résultats nous encouragent à chercher des méthodes afin de contourner la perte de précision liée à la binarisation, comme celles implémentées par Zhan et al. [Zhang *et al.*, 2021].

### MobileNetV2 sur CIFAR-10

Nous avons testé notre implémentation `MobileNetV2_QAT` sur CIFAR-10, en nous concentrant particulièrement sur la binarisation des poids, sans les activations, afin d'observer les performances de modèles à connexions résiduelles. Nous présentons ici les résultats obtenus après 500 epochs d'entraînement avec l'optimiseur Adam.

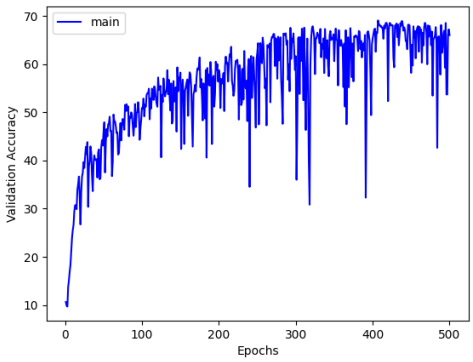


Figure 4: Évolution de l'accuracy Top-1 sur CIFAR-10 Validation pour `MobileNetV2_QAT` (dbits=1, Adam, 500 époques).

**Analyse :** Les résultats obtenus montrent une dynamique d'apprentissage, mais confirment les défis de la binarisation extrême.

La courbe d'évolution de l'accuracy Top-1 (Figure 4) progresse rapidement lors des 100 premières époques (passant de 10 % à plus de 50 %), puis ralentit autour de 65–68 % à la 300e époque. On observe également des fluctuations importantes qui reflètent l'instabilité induite par la binarisation extrême au sein des blocs inverted residual. Malgré ces variations, le modèle parvient à maintenir une précision moyenne plutôt basse autour de 67 % (comparé au baseline à 94 % sur CIFAR-10), montrant l'imcopatibilité des connexiosn résiduelles telles quelles avec la binarisation. L'instabilité extrême montre aussi les limites de la binarisation naïve des modèles.

L'accuracy Top-5, atteignant 98.7%, montre toutefois que le modèle réussit à généraliser sur le jeu de données, mais manque d'une capacité à s'ajuster de manière fine.

| Métrique | Valeur Finale (Époque 500) |
|---|---|
| Validation Loss | 1.04 |
| Validation Accuracy (Top-1) | 68.3 % |
| Validation Accuracy (Top-5) | 98.7 % |

Table 2: Performances finales de `MobileNetV2_QAT` (dbits=1, Adam) sur CIFAR-10 après 500 époques.

### Exploration de FracBNN

Face aux limitations de précision rencontrées avec l'approche QAT binarisée standard, particulièrement sur CIFAR-10 avec MobileNetV2 ou VGG Full binarized, nous avons exploré une technique plus avancée : FracBNN (Fractional Binary Neural Networks) [Zhang *et al.*, 2021]. Cette méthode vise à améliorer l'accuracy des BNNs en utilisant notamment des activations dites "fractionnaires" (représentées jusqu'à 2 bits via une convolution binaire sparse additionnelle) pour augmenter la capacité de représentation du modèle.

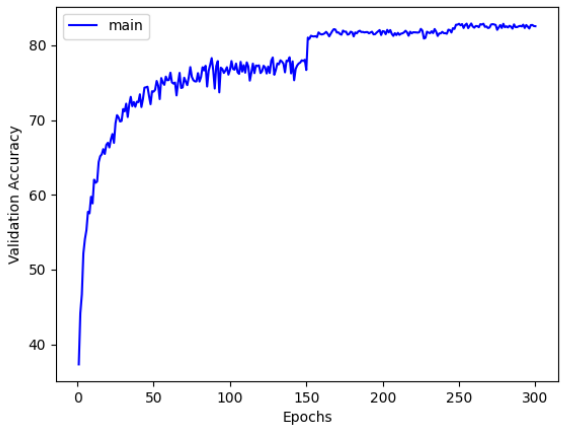


Figure 5: Évolution de l'accuracy Top-1 sur CIFAR-10 Validation pour `FracBNN` (dbits=1, Adam, 300 époques).

**Analyse :** L'expérimentation avec une approche inspirée de FracBNN sur une base ResNet montre une dynamique d'apprentissage et des performances encourageantes par rapport à notre QAT standard sur MobileNetV2. La courbe d'accuracy (Figure 7) indique une convergence rapide au début, atteignant plus de 75% avant l'époque 100. Un second saut de performance notable se produit autour de l'époque 150, potentiellement lié à un ajustement du taux d'apprentissage par le scheduler, permettant au modèle de dépasser les 80%. L'accuracy semble ensuite plafonner autour de 82-83% durant les 100 dernières époques.

Cette performance finale de 82-83% représente une amélioration substantielle par rapport aux 68.3% obtenus avec MobileNetV2 QAT standard, suggérant que les techniques FracBNN (comme les activations fractionnaires) ou l'architecture ResNet elle-même sont plus propices à la binarisation que notre implémentation QAT de base sur MobileNetV2. Cependant, cet accuracy reste encore en deçà de la baseline d'un ResNet-18 pleine précision sur CIFAR-10 (typiquement >93%). Le plateau observé suggère que, bien que FracBNN aide à récupérer une partie de la précision perdue, des limitations dues à la binarisation persistent ou que l'optimisation pourrait être encore affinée. Cette exploration confirme néanmoins que des techniques QAT plus sophistiquées peuvent effectivement réduire l'écart de performance des BNNs.

## 3.3 Comparaison et Discussion des Résultats

Afin d'évaluer l'impact de notre approche QAT et de la binarisation (`dbits=1`), nous comparons les performances obtenues pour les différentes architectures aux performances de référence (baselines) non quantifiées. Le tableau 3 résume ces performances en termes d'accuracy Top-1 sur l'ensemble de validation.[Papers With Code, 2024b; Papers With Code, 2024a; Liu, 2018]

| Architecture | Dataset | Résultat | Val Acc (Top-1) |
|---|---|---|---|
| MLP | MNIST | QAT | 98.57 % |
| MLP | MNIST | baseline | 98-99+ % |
| VGG-style | CIFAR-10 | QAT | 91 % |
| VGG-style | CIFAR-10 | baseline | 92-94 % |
| MobileNetV2 | CIFAR-10 | QAT | 68.3 % |
| MobileNetV2 | CIFAR-10 | baseline | 94-94.5 % |
| FracBNN/ResNet | CIFAR-10 | QAT | 82-83 % |
| ResNet-18 | CIFAR-10 | baseline | 93-95 % |

Table 3: Comparaison des performances Top-1 finales QAT (`dbits=1`) vs. Baselines.

**Analyse Comparative :**

La comparaison des résultats met en lumière l'impact variable de notre approche QAT binarisée selon la complexité de la tâche et de l'architecture.

Sur MNIST, le MLP QAT (98.57%) performe remarquablement près de sa baseline (98-99+%), validant l'efficacité de l'approche QAT pour les tâches et modèles simples.

Sur la tâche plus complexe CIFAR-10, la binarisation QAT

pose un défi plus important. Le VGG-style QAT (91%) montre une robustesse notable, restant relativement proche de sa baseline (92-94%). En revanche, le MobileNetV2 QAT chute significativement à 68.3% (baseline 94-94.5%), suggérant une plus grande sensibilité de cette architecture efficace à la perte d'information binaire, malgré une extraction de caractéristiques pertinente indiquée par un Top-5 élevé (98.7%).

L'exploration de l'approche FracBNN sur une base ResNet montre une amélioration substantielle, atteignant 82-83% d'accuracy Top-1. Ce résultat, supérieur a MobileNetV2, indique le bénéfice potentiel de techniques QAT plus avancées ou de backbones différents comme ResNet, bien qu'un écart demeure par rapport à la baseline ResNet pleine précision (93-95%).

En synthèse, nos expériences confirment que si le QAT/STE permet l'entraînement de BNNs, la binarisation stricte engendre une perte de précision notable sur les tâches complexes, variable selon l'architecture. Les approches plus sophistiquées comme FracBNN peuvent atténuer cet écart, mais le compromis entre l'efficacité théorique de la binarisation et la précision demeure un défi central.

## 4 Analyse Critique de l'Approche

Ce projet a permis une compréhension pratique des défis liés à l'implémentation et à l'entraînement des BNNs via QAT. Notre framework modulaire en PyTorch, intégrant des couches QAT personnalisées et un STE, a facilité l'expérimentation mais a aussi mis en lumière des limites et pistes d'amélioration.

- **Approximation STE :** Notre implémentation QAT utilise un STE simple (identité/clipping). Bien que fonctionnel, c'est une approximation perfectible du gradient réel. Des STE plus avancés (ex: dérivés de 'hardtanh', seuils de clipping adaptatifs) existent et pourraient améliorer la stabilité et la performance finale, mais dépassaient le cadre de ce projet initial.
- **Sensibilité aux Hyperparamètres et Optimisation :** Les BNNs sont très sensibles aux hyperparamètres (LR, optimiseur, scheduler). Notre exploration, bien que facilitée par les configurations TOML, est restée limitée. L'optimisation dans cet espace non lisse nécessiterait une recherche plus systématique (grid/random search, etc.) pour de meilleurs résultats, ce qui était coûteux en temps de calcul.
- **Entraînement "From Scratch" vs Fine-tuning :** L'entraînement à partir de zéro est plus difficile pour les modèles QAT à faible nombre de bits. Partir de poids pré-entraînés en pleine précision et affiner via QAT (*fine-tuning*) facilite souvent la convergence et aurait pu améliorer nos résultats, mais complexifiait la gestion des poids.

## 5 Conclusion

Notre étude a permis d'implémenter et d'évaluer l'entraînement conscient de la quantification (QAT) pour des réseaux de neurones binarisés (`dbits=1`) sur diverses architectures (MLP, VGG, MobileNetV2) et datasets (MNIST, CIFAR-10). Nous avons confirmé la faisabilité de l'entraînement via QAT et STE, et observé le potentiel d'efficacité théorique des BNNs.

Nos résultats expérimentaux démontrent cependant un fort contraste : alors que le MLP binarisé atteint une performance quasi-optimale sur MNIST, les architectures CNN binarisées (VGG, MobileNetV2) subissent une chute de précision significative sur la tâche plus complexe CIFAR-10 par rapport à leurs baselines pleine précision. L'exploration d'une approche inspirée de FracBNN sur une base ResNet a montré une amélioration notable par rapport à notre MobileNetV2 QAT standard, indiquant le potentiel de techniques QAT plus avancées ou de backbones différents pour atténuer cette perte.

Néanmoins, même avec FracBNN, un écart de performance subsiste par rapport aux modèles non quantifiés, soulignant la difficulté inhérente à la binarisation extrême (`dbits=1`). L'amélioration des performances nécessiterait une exploration plus poussée des méthodes QAT avancées, une optimisation rigoureuse des hyperparamètres, et potentiellement l'utilisation de pré-entraînement, comme discuté dans notre analyse critique.

## References


[Hubara *et al.*, 2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4107–4115, 2016.

[Liu, 2018] Kuang Liu. pytorch-cifar: 95.47% on cifar10 with pytorch. https://github.com/kuangliu/pytorch-cifar, 2018. GitHub repository.

[Papers With Code, 2024a] Papers With Code. CIFAR-10 Benchmark (Image Classification). https://paperswithcode.com/sota/image-classification-on-cifar-10, 2024.

[Papers With Code, 2024b] Papers With Code. MNIST Benchmark (Image Classification). https://paperswithcode.com/sota/image-classification-on-mnist, 2024.

[Rastegari *et al.*, 2016] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pages 525–542. Springer, 2016.

[Sandler *et al.*, 2018] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[Zhang *et al.*, 2021] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. Fracbnn: Accurate and fpga-efficient binary neural networks with fractional activations. In *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '21)*, FPGA '21, pages 171–182, New York, NY, USA, 2021. Association for Computing Machinery.

[Zhou *et al.*, 2016] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.


# 6 Annexe

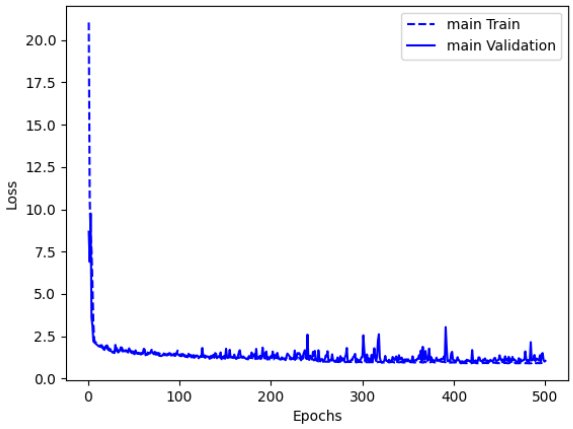


Figure 6: Évolution de la loss pour `MobileNetV2_QAT` (dbits=1, Adam, 500 époques) sur CIFAR-10.

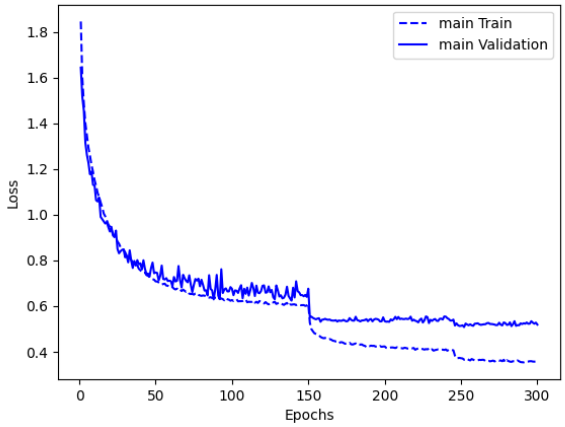


Figure 7: Évolution de l'accuracy Top-1 sur CIFAR-10 Validation pour `FracBNN` (dbits=1, Adam, 300 époques).