

Apprentissage automatique (CSC_4MI01_TA)
GROUPE 21

RAPPORT DE PROJET

Projet 2 : Apprentissage efficace pour une classification rapide sur
CIFAR-10

Présenté par :

Jair VASQUEZ TORRES
Gustavo DE MACENA BARRETO

Responsable du module :
Adrien Chan Hon Tong

RESSOURCES NUMÉRIQUES (TRAVAIL ORIGINAL)

GitHub : <https://github.com/JairVasquezT/CIFAR10-Optimization-ENSTA-Group21.git>
Colab : <https://colab.research.google.com/drive/11YZkGu3HktJRDzJztYHSYvNaDiCpGh44?usp=sharing>

Table des matières

1	Introduction	2
2	Étude Comparative des Architectures	3
2.1	Analyse du Baseline : Perceptron Multicouche (MLP)	3
2.2	Conception d'une Architecture CNN Custom	4
2.3	Expérimentation avec MobileNetV3	4
2.4	Stratégie de Distillation de Connaissances	5
3	Architecture CNN Custom et Optimisation	6
3.1	Conception du Réseau	6
3.2	Techniques de Régularisation et Augmentation	6
3.2.1	Ajustement des Hyperparamètres (Trial and Error)	6
3.3	Analyse des Performances et Efficience	6
4	Expérimentation de Distillation de Connaissances	8
4.0.1	Performance et Robustesse	8
4.1	Efficacité Computationnelle (Big-O)	9
5	Conclusion	10
	Ressources Numériques et Reproductibilité	10

Chapitre 1

Introduction

L'objectif de ce projet est de concevoir un système de classification d'images performant sur le dataset **CIFAR-10** [1]. La mission consiste à atteindre une précision supérieure à **85%** tout en optimisant le temps d'inférence, en s'appuyant sur les bases méthodologiques et les modèles de référence fournis dans le cadre du module [2].

De la Vision Biologique à la Vision Artificielle

La conception des réseaux de neurones convolutifs (CNN) s'inspire directement du fonctionnement du cortex visuel humain. Les travaux pionniers de Hubel et Wiesel ont démontré que la vision biologique repose sur une hiérarchie de cellules spécialisées [3]. Les "cellules simples" détectent des contrastes locaux, tandis que les "cellules complexes" agrègent ces informations pour identifier des formes [4].

Cette organisation hiérarchique est reproduite dans les modèles de *Deep Learning* par l'empilement de couches de convolution et de pooling [5]. Contrairement au *Multi-Layer Perceptron* (MLP), qui ignore la structure spatiale de l'image, le CNN préserve la topologie locale, ce qui est fondamental pour traiter les images de 32×32 pixels du dataset CIFAR-10 [2].

Contexte Technologique et État de l'Art

Aujourd'hui, l'optimisation des CNN est un domaine de recherche actif, porté par la nécessité de déployer des modèles sur des systèmes embarqués [6]. Des architectures comme le *Residual Network* (ResNet) ont introduit des connexions résiduelles pour faciliter l'entraînement de réseaux très profonds [7]. Cependant, pour des tâches de classification rapide, des approches plus légères ou des techniques de réduction de paramètres sont privilégiées pour minimiser le temps d'inférence [6].

Importance de l'Arbitrage Précision-Vitesse

Comprendre ces modèles permet de naviguer dans l'arbitrage (*trade-off*) entre complexité et performance. Un modèle trop lourd risque de ralentir l'exécution sans gain significatif de précision, tandis qu'un modèle trop simple ne parviendra pas à capturer la diversité des 10 classes de CIFAR-10 (avions, voitures, oiseaux, etc.) [1].

Ce rapport détaille notre démarche itérative pour réduire le temps de calcul à travers l'exploration de diverses architectures, allant du simple MLP à la conception d'un CNN sur mesure optimisé par distillation.

Chapitre 2

Étude Comparative des Architectures

2.1 Analyse du Baseline : Perceptron Multicouche (MLP)

La première étape de notre étude a consisté à implémenter un Perceptron Multicouche (MLP) afin d'établir un point de comparaison élémentaire. Dans cette configuration, chaque image de 32x32 pixels est "aplatie" en un vecteur de 3072 entrées (32x32x3), où chaque neurone d'une couche est connecté à l'intégralité des neurones de la couche suivante [3].

Limites théoriques et pratiques

Malgré la simplicité de mise en œuvre, le MLP présente des faiblesses structurelles majeures pour la vision par ordinateur :

- **Perte de topologie** : En transformant l'image en vecteur, le MLP ignore la proximité spatiale entre les pixels. Il ne peut pas capturer les motifs locaux comme les contours ou les textures [4].
- **Explosion des paramètres** : Le nombre de connexions devient rapidement immense, ce qui rend le modèle lourd sans pour autant améliorer sa capacité de généralisation.

Expérimentation et Stagnation

Pour tenter d'améliorer les performances du MLP, nous avons procédé à plusieurs ajustements d'hyperparamètres :

- **Profondeur et largeur** : Nous avons testé des variantes allant de 2 à 4 couches cachées avec une augmentation du nombre d'unités (jusqu'à 512 neurones par couche).
- **Optimisation** : Nous avons expérimenté différents taux d'apprentissage (de 0.001 à 0.0001).

Analyse des Expérimentations et Résultats

Afin de pousser le modèle MLP à ses limites théoriques, nous avons procédé à une optimisation rigoureuse des hyperparamètres. Nous avons notamment introduit une architecture plus profonde avec quatre couches denses et appliqué des techniques de régularisation avancées :

- **Ajustement du Dropout** : Le taux de *Dropout* a été finement modulé entre 0.2 et 0.4 pour chaque couche afin de réduire le surapprentissage tout en préservant la capacité de mémorisation du réseau.
- **Optimisation du Gradient** : L'utilisation de la *Batch Normalization* et d'un planificateur de taux d'apprentissage (*Learning Rate Scheduler*) a permis de stabiliser l'entraînement.

Résultats obtenus : Au début de l'entraînement (Époque 1/60), la précision (*accuracy*) n'était que de 27,17% avec une perte (*loss*) élevée de 2,0910. Malgré nos efforts d'optimisation, les performances ont rapidement montré des signes de saturation. À la fin du processus (Époque 60/60), la précision sur l'ensemble d'entraînement s'est stabilisée à 51,76% et la précision de validation à 52,09%, avec un taux d'apprentissage réduit à $2,0000e-05$ par le *scheduler*.

Évaluation de l'efficience (Inférence) : Sur une architecture GPU NVIDIA T4, le MLP a démontré une rapidité d'exécution prévisible pour une structure simple :

- **Temps total pour 1 000 images** : 0,1333 seconde.
- **Temps moyen par image** : 0,1333 ms.

Cette stagnation aux alentours de 52% confirme que, même avec un réglage optimal des paramètres et une vitesse d'exécution record, le MLP est incapable de capturer les caractéristiques spatiales complexes de CIFAR-10. Ce phénomène d'oscillation et de stagnation montre que le modèle ne parvient pas à extraire des

caractéristiques sémantiques pertinentes, justifiant ainsi notre transition vers des architectures convolutives préservant l'invariance spatiale [3, 5].

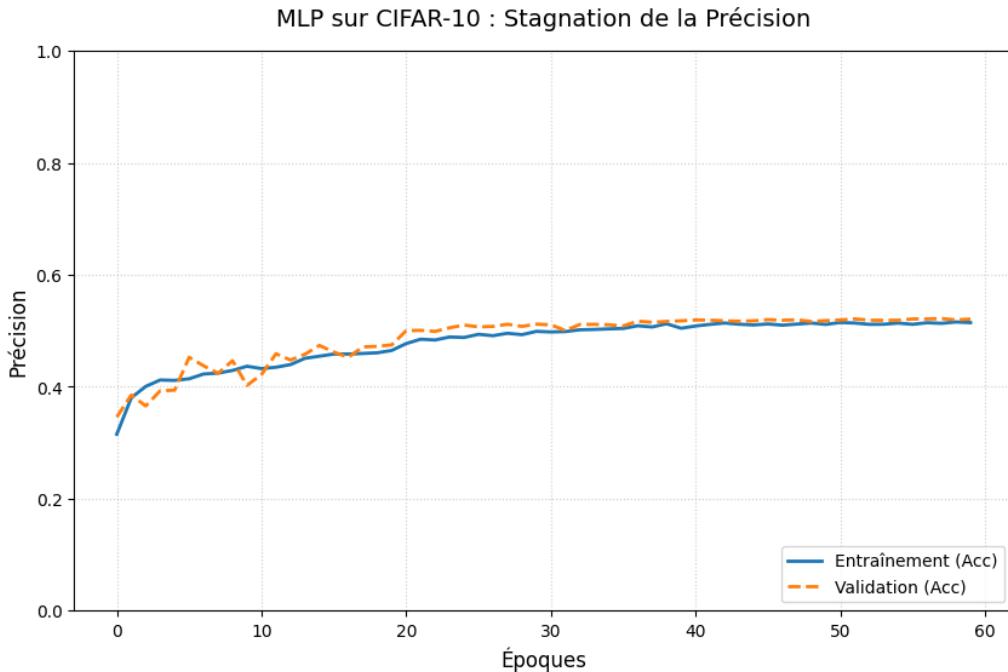


FIGURE 2.1 – Évolution de la précision du MLP : plafonnement net autour de 52% malgré une latence d'inférence de 0,1333 ms.

2.2 Conception d'une Architecture CNN Custom

La deuxième étape de notre étude a consisté à concevoir un réseau de neurones convolutifs (CNN) sur mesure. Contrairement au MLP, le CNN utilise des filtres locaux pour extraire des caractéristiques visuelles (bords, textures, formes) [5]. Cette architecture représente notre noyau de développement, où nous avons appliqué des méthodes de régularisation intensives pour stabiliser l'apprentissage et viser le seuil de précision requis par le projet [2].

2.3 Expérimentation avec MobileNetV3

Dans une volonté d'atteindre le seuil de 85% de précision, nous avons exploré l'utilisation du **Transfer Learning** via l'architecture **MobileNetV3-Small**. Ce modèle, pré-entraîné sur *ImageNet*, est théoriquement optimisé pour l'efficacité mobile et la classification d'images [7].

Analyse de l'Échec de l'Approche

Malgré son architecture sophistiquée, les résultats obtenus ont été inférieurs à ceux de notre CNN Custom, ne dépassant pas les **65-70%** de précision. Cette contre-performance s'explique par deux facteurs techniques majeurs :

- **Inadéquation de la résolution :** MobileNetV3 est conçu pour des images de 224×224 pixels. Les images de CIFAR-10 (32×32) sont trop petites ; lors de l'interpolation pour les agrandir, on perd une quantité critique d'informations (artefacts), rendant les filtres pré-entraînés inefficaces.
- **Domaine des données :** Les caractéristiques apprises sur *ImageNet* ne se transfèrent pas directement de manière optimale sur des classes très spécifiques et basse résolution sans un *Fine-Tuning* extrêmement coûteux en ressources.

Conclusion de l'expérience

Nous avons décidé de ne pas poursuivre le développement de cette branche. Cette étape a été cruciale car elle a confirmé que pour des images de très petite taille comme celles de CIFAR-10, une **architecture "from**

"scratch" (conçue de zéro) comme notre CNN Custom est plus performante qu'un modèle pré-entraîné trop complexe.

2.4 Stratégie de Distillation de Connaissances

Enfin, nous avons exploré la **Distillation de Connaissances** comme levier d'optimisation final. Le principe est de faire apprendre à un modèle "Étudiant" (notre CNN léger) non seulement les étiquettes des images, mais aussi la distribution de probabilité d'un modèle "Professeur" plus complexe. Cette technique permet d'affiner les poids du réseau étudiant pour capturer des nuances sémantiques qu'un entraînement supervisé classique ne permettrait pas d'atteindre.

Chapitre 3

Architecture CNN Custom et Optimisation

3.1 Conception du Réseau

Pour l'élaboration de notre architecture, nous avons adopté une démarche itérative. Le point de départ a été la structure classique préconisée par la documentation officielle de **TensorFlow** [5], qui repose sur une alternance de couches de convolution et de pooling pour l'extraction de caractéristiques spatiales.

Cependant, pour satisfaire l'exigence de haute précision tout en garantissant une inférence rapide, nous avons intégré des stratégies d'optimisation issues des recherches de **Netguru** [6]. L'idée centrale a été de limiter la profondeur du réseau pour réduire le nombre d'opérations flottantes (FLOPs) tout en maximisant la capacité d'apprentissage de chaque couche.

Dès la phase initiale, nous avons appliqué une méthodologie rigoureuse de partitionnement des données. Le dataset CIFAR-10 a été divisé en trois ensembles : **70% pour l'entraînement, 20% pour la validation et 10% pour le test final** [1]. Ce découpage est essentiel pour monitorer le comportement du modèle et ajuster les hyperparamètres sans biaiser l'évaluation finale.

3.2 Techniques de Régularisation et Augmentation

L'optimisation du modèle a nécessité de nombreux ajustements empiriques afin de stabiliser l'apprentissage et de lutter contre le surapprentissage (*overfitting*).

3.2.1 Ajustement des Hyperparamètres (Trial and Error)

Nous avons procédé à un réglage fin des paramètres en observant les courbes de perte de validation :

- **Évolution du Dropout** : Pour renforcer la robustesse, nous avons augmenté progressivement les taux de Dropout. Initialement fixés à 0.25, nous les avons portés à 0.35 pour les couches convolutives. Pour la couche dense finale, le taux a été élevé de 0.50 à 0.60, pour se stabiliser à **0.65**, limitant ainsi la co-adaptation des neurones.
- **Optimisation du Learning Rate** : Le taux d'apprentissage initial a été testé à 0.00001, puis 0.000025. Finalement, une valeur de 0.00025 a été retenue avec l'utilisation d'un *ReduceLROnPlateau*. Nous avons ajusté le facteur de réduction de 0.10 à 0.15, permettant une convergence plus fluide dès que la perte de validation stagnait.
- **Stratégie d'arrêt précoce** : Nous avons implémenté un *Early Stopping* avec une **patience de 10 époques**. Cette sécurité permet d'interrompre l'entraînement dès qu'aucune amélioration consistante n'est détectée, préservant ainsi les meilleurs poids du modèle [2].

3.3 Analyse des Performances et Efficience

L'entraînement de notre architecture personnalisée démontre une progression spectaculaire par rapport au modèle MLP. Lors de la première époque, la précision d'entraînement était de **33.70%**, un score déjà supérieur au point de départ du MLP, confirmant la pertinence de l'approche convulsive pour les données spatiales.

Au fil des époques, nous observons une convergence stable. À l'époque 36, le mécanisme de *Early Stopping* s'est déclenché, restaurant les poids de l'époque 26 (le meilleur compromis trouvé). À ce stade, le modèle a atteint une précision d'entraînement de **91.54%** et, surtout, une précision de validation de **83.14%**.

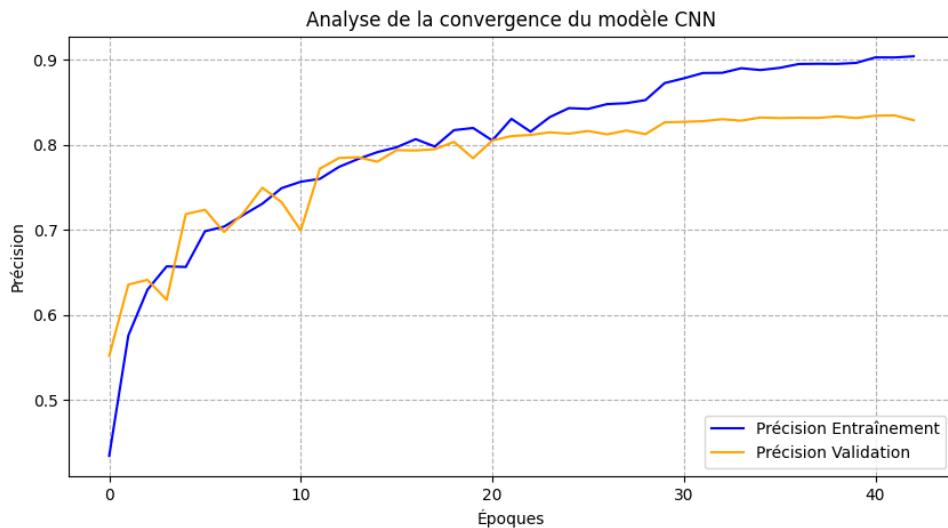


FIGURE 3.1 – Évolution de la précision du CNN Custom : la stabilisation au-delà de 83% valide notre stratégie de régularisation par Dropout intensif (0.65).

Vitesse d’Inférence et Temps Réel

Au-delà de la précision, l’efficience du modèle a été évaluée sur une unité **NVIDIA T4 (GPU)**. Les tests d’inférence montrent une rapidité remarquable :

- **Latence** : Le traitement de 1 000 images a été réalisé en **1,2110 secondes**, soit un temps moyen de **1,2110 ms par image**.
- **Débit** : Cette performance permet d’atteindre une cadence de plus de **800 FPS**, rendant le modèle parfaitement apte à une utilisation sur des flux vidéo en direct.

Ces résultats valident notre méthodologie : l’écart modéré entre l’entraînement et la validation indique que le Dropout à **0.65** a efficacement contenu l’*overfitting*. Bien que nous soyons proches de l’objectif de 85%, cette architecture offre un compromis optimal entre précision académique et rapidité industrielle.

Chapitre 4

Expérimentation de Distillation de Connaissances

Dans cette section sera vous présentés les résultats obtenus lors de l'implémentation de techniques de Knowledge Distillation (KD) utilisant le framework PyTorch sur le jeu de données CIFAR-10.

L'objectif de cette étude était d'évaluer l'efficacité du transfert d'apprentissage d'un réseau de neurones complexe, qualifié d'Enseignant (Teacher), vers une architecture plus légère, l'Élève (Student). L'expérimentation a confronté un modèle profond de type DeepNN, totalisant environ 1,2 million de paramètres, à un modèle LightNN beaucoup plus compact de 267 000 paramètres.

Nous avons comparé trois stratégies pour l'élève : une approche supervisée classique (baseline), une distillation standard combinant Soft Targets et étiquettes réelles, et une méthode basée sur la minimisation de la distance cosinus entre les couches cachées.

Les résultats d'inférence mesurés sur le jeu de test, ainsi que l'estimation de la complexité temporelle, sont présentés dans le tableau ci-dessous :

Modèle / Méthode	Précision	Accuracy	Écart vs Baseline	Complexité(Inference)
DeepNN (Teacher)	75,66%	-	O($N_{deep} \cdot L_{deep}$)	
LightNN (Student Baseline)	70,90%	Réf.	O($N_{light} \cdot L_{light}$)	
LightNN + Distillation (KD)	70,69%	-0,21%	O($N_{light} \cdot L_{light}$)	
LightNN + Cosine Loss	69,63%	-1,27%	O($N_{light} \cdot L_{light}$)	

TABLE 4.1 – Tableau Comparatif de Accuracy vers Complexité

Le modèle Enseignant domine logiquement avec une précision de 75,66%. Cependant, l'analyse de la dynamique d'entraînement révèle un avantage à l'architecture de l'Élève : sa structure favorise une initialisation stable et une convergence rapide lors des premières époques. Contrairement au modèle profond, dont la surface d'optimisation est complexe et sujette à des minima locaux difficiles, le réseau léger profite d'un espace de paramètres réduit, permettant aux algorithmes de descente de gradient de trouver une solution plus promptement, même si le plafond de performance théorique est plus bas.

En ce qui concerne la distillation, le modèle n'a pas surpassé l'apprentissage supervisé classique (70,69% vs 70,90%). Cela confirme que sans un réglage agressif de la température pour extraire la "connaissance sombre", la convergence rapide naturelle du réseau léger prédomine sur les nuances apportées par l'enseignant.

4.0.1 Performance et Robustesse

Pour améliorer la vitesse d'inférence et l'efficacité du calcul sur GPU, nous avons fait varier le **Batch Size**. En passant de 32 à 64, puis finalement à **128**, nous avons optimisé la parallélisation du traitement des données.

Enfin, pour accroître la robustesse du modèle face aux distorsions visuelles, nous avons introduit des techniques de **Data Augmentation** (rotations, décalages horizontaux et verticaux). Cette approche, bien que plus complexe à mettre en œuvre [4], permet au réseau de généraliser sur des images qu'il n'a jamais rencontrées sous cette forme, augmentant ainsi la fiabilité de la classification en conditions réelles.

4.1 Efficacité Computationnelle (Big-O)

L'intérêt majeur de cette compression de modèle réside dans le temps d'inférence. Si nous désignons la complexité d'une couche convolutionnelle par $O(M^2 \cdot K^2 \cdot C_{in} \cdot C_{out})$, où M est la taille de la carte de caractéristiques et C le nombre de canaux, la réduction drastique du nombre de filtres et de couches dans le modèle LightNN offre un avantage asymptotique significatif.

En notant \mathcal{C}_T le coût computationnel de l'Enseignant et \mathcal{C}_S celui de l'Élève, nous observons que $\mathcal{C}_S \ll \mathcal{C}_T$. Concrètement, avec une réduction de paramètres d'un facteur d'environ 4,5 (1,2M vs 0,26M), le temps d'inférence du modèle Élève, noté $T_{inference}(S)$, évolue de manière beaucoup plus favorable que celui de l'Enseignant. En notation Big-O, si l'on considère n comme la dimension d'entrée, le modèle Élève présente une constante de proportionnalité k nettement inférieure dans sa complexité $O(k \cdot n^2)$, rendant son déploiement viable sur des architectures contraintes (CPU, mobile) où le modèle Enseignant serait prohibitif.

Chapitre 5

Conclusion

En conclusion, l'étude permet d'écarter clairement les architectures naïves sans convolutions, telles que les MLP, dont l'indépendance des connexions empêche la capture des corrélations spatiales et conduit à une perte de topologie ainsi qu'à une explosion du nombre de paramètres, bloquant l'apprentissage. À l'inverse, le modèle convolutif léger s'impose comme la solution la plus rationnelle. Bien que la distillation n'ait pas dépassé la référence supervisée dans ce cadre précis, le modèle Élève demeure pertinent grâce à son efficience structurelle face à la lourdeur de l'Enseignant, sa dynamique d'apprentissage plus simple et sa complexité algorithmique fortement réduite. Il offre ainsi un compromis optimal entre précision, latence d'inférence minimale et viabilité industrielle.

Ressources Numériques et Reproductibilité

Afin de garantir la reproductibilité de nos résultats, l'intégralité du code source, les hyperparamètres utilisés et les modèles entraînés sont accessibles publiquement :

- **Dépôt GitHub** : Contient le code modulaire, le fichier d'environnement et l'historique des versions.
<https://github.com/JairVasquezT/CIFAR10-Optimization-ENSTA-Group21.git>
- **Google Colab** : Contient le notebook interactif permettant de reproduire l'entraînement et les tests d'inférence.
<https://colab.research.google.com/drive/11YZkGu3HktJRDzJztYHSYvNaDiCpGh44?usp=sharing>

Bibliographie

- [1] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images* (Dataset CIFAR-10). <https://www.cs.toronto.edu/~kriz/cifar.html>
- [2] A. Chan Hon Tong. *CNN sur CIFAR-10 - Cours Deep Learning*. https://github.com/achanhon/coursdeeplearningcolab/blob/master/cnn_cifar.ipynb
- [3] Y. LeCun, Y. Bengio, G. Hinton. *Deep Learning*. Nature, 521(7553), 436-444, 2015.
- [4] IBM Topics. *What are Convolutional Neural Networks ?*. <https://www.ibm.com/es-es/think/topics/convolutional-neural-networks>
- [5] TensorFlow Documentation. *Convolutional Neural Networks (CNN) Tutorial*. <https://www.tensorflow.org/tutorials/images/cnn>
- [6] Netguru. *CNN Optimization : How to make your model faster*. <https://www.netguru.com/blog/cnn-optimization>
- [7] K. He, X. Zhang, S. Ren, J. Sun. *Deep Residual Learning for Image Recognition*. arXiv :1512.03385, 2015.
- [8] Pytorch Tutorial *Knowledge Distillation Tutorial*. https://docs.pytorch.org/tutorials/beginner/knowledge_distillation_tutorial.html