

**Due Date: 28nd February, 24:00**

Instructions

- *Ce devoir est difficile - commencez le bien en avance.*
- *Pour toute les questions, montrez votre travail!*
- *Soumettez votre rapport (PDF) et votre code électroniquement via la page Gradescope du cours.*
- *Vous pouvez utiliser le notebook Jupyter **main.ipynb** pour lancer vos expériences (facultatif).*
- *Pour les questions ouvertes (c.-à-d. les expériences où il n'y a aucun test associé), vous n'avez pas à envoyer de code - le rapport suffit.*
- *Les TAs pour ce devoir sont **Johan Obando** (IFT6135B) and **Jerry Huang** (IFT6135A)*

**Sommaire:**

Dans ce devoir, vous allez développer plusieurs modèles de classification d'images sur les données CIFAR-10.

CIFAR-10 est constituée de 60000  $32 \times 32$  images couleur réparties en 10 classes, et chaque classe compte 6 000 images. Nous avons donc 50000 images d'entraînement et 10000 images de test. Vous disposez d'une classe de données sur PyTorch (`torch.utils.data.Dataset`) intitulée `CIFAR10` à partir de la librairie `torchvision`. les répartitions `train`, `valid` et `test` sont fournies. Tout au long de ce devoir, la forme des données CIFAR10 est la suivante ( `Batch`, `Channels`, `Height`, `Width`).

Dans la première partie, vous êtes censés construire et entraîner un MLP. Dans la deuxième partie, vous allez élaborer et entraîner un ResNet18. Dans la troisième partie, vous aurez à construire et à entraîner une architecture récente appelée **MLPMixer**. Comme vous pouvez le constater, l'évolution de ces architectures reflète également la récente vague de recherche en vision par ordinateur. Cela a commencé avec les MLP, puis les CNN ont été les plus populaires, et maintenant nous découvrons que les MLP, lorsqu'elles sont bien conçues, ne sont pas si mal !

**Instructions de programmation** Vous devrez utiliser PyTorch pour répondre à toutes les questions. De plus, ce devoir nécessite l'exécution des modèles sur des GPUs (sinon cela prendra un temps considérable) ; si vous n'avez pas accès à vos propres ressources (par exemple, votre propre machine, un cluster), veuillez vous servir de Google Colab. Le principal fichier pour lancer vos expériences est `main.py` ou `main.ipynb`. Pendant le développement, vous pouvez utiliser les tests unitaires définis dans `test.py` pour vous aider. Rappelez-vous que ce ne sont que des tests de base, et que d'autres tests seront exécutés sur le Gradescope.

**Soumission** Vous devez soumettre les fichiers `mlp.py`, `resnet18.py`, `mlpmixer.py` et `utils.py` à gradescope pour l'auto-évaluation, ainsi qu'un rapport PDF pour les questions à caractère expérimental et ouvert.

## Problem 1

**Implémentation d'un MLP (5pts)** Dans ce problème, vous allez implémenter un MLP.

Un MLP est constitué de plusieurs couches linéaires suivies d'une fonction de non-linéarité. Vous trouverez la template de code dans `mlp.py`.

1. Vous devez compléter la `class Linear` dans le fichier `mlp.py`. La classe doit avoir deux attributs : `weight` et `bias`. Complétez la fonction forward correspondante
2. Vous pouvez maintenant passer à la `class MLP`. Dans `_build_layers`, vous devez mettre en place les couches cachées ainsi que la couche de sortie (classificateur). Dans `_initialize_linear_layer`, vous devez remplir les biais avec des zéros et utiliser l'initialisation normale de Xavier pour les poids. Essayez de vous référer à la documentation de pytorch pour cette partie.
3. Complétez le `activation_fn` afin qu'il puisse traiter les entrées en fonction de différents `self.activation`. Compléter la fonction forward selon la docstring.

## Problem 2

**Implémentation de ResNet18 (15pts)** ResNet propose de faire usage des connexions résiduelles afin que le réseau puisse être entraîné avec beaucoup plus de couches. Il est démontré que l'augmentation de la profondeur permet aux réseaux de neurones d'obtenir de meilleures performances dans plusieurs applications de vision. Dans ce problème, vous allez implémenter l'architecture de ResNet18 dans `resnet18.py`. Vous pouvez trouver les détails de l'architecture dans la Figure 1.

1. Nous fournissons le template de code pour le bloc ResNet dans `class BasicBlock`. Vous pouvez voir qu'il y a des points d'interrogation dans la fonction d'initialisation pour les arguments manquants. Veuillez compléter les arguments de façon à ce que la forme de chaque composant ait un sens. Ensuite, complétez la fonction forward selon le schéma de la Figure 1.
2. Complétez l'architecture ResNet18 dans `class ResNet18`. Remplissez les arguments manquants et complétez la fonction forward.

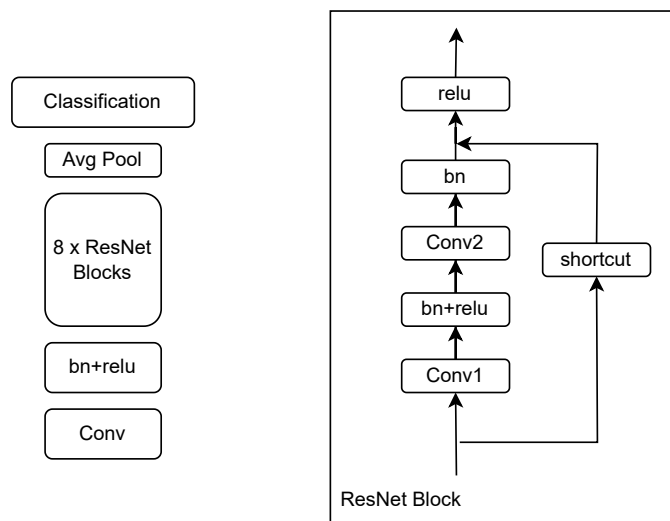


Figure 1: L'architecture ResNet18. **Gauche** : Le diagramme d'architecture d'un ResNet18. Il commence par une couche de convolution 3x3. Le corps principal est constitué de huit blocs ResNet. À la fin, on trouve le pooling moyen global et la tête de classification. **Droite** : Diagramme d'un bloc ResNet. Dans une branche, les entrées passent par deux couches de convolution avec des normalisations par batch et des fonctions de non-linéarité. Dans l'autre branche, les entrées passent par un module de raccourci, qui peut être une fonction d'identité si la taille correspond. Enfin, les sorties des deux branches sont additionnées. La branche de raccourci est également appelée *connexions résiduelles*.

## Problem 3

**Implémentation de MLP Mixer (20pts)** Dans cette partie, vous implémenterez l'architecture de MLP Mixer. Veuillez fournir votre réponse dans le fichier `mlpmixer.py`. MLP Mixer est un article récent dont les auteurs constatent que, si elles sont bien faites, les architectures MLP seules peuvent également être très compétitives. L'architecture est montrée dans la Figure 2.

1. La première composante que vous allez construire est `class PatchEmbed`. Contrairement aux architectures précédentes, MLP Mixer divise l'image entière en petits patches, et effectue une transformation linéaire commune sur ces patches. Par exemple, si la taille de l'image est de  $256 \times 256$ , et la taille du patch est de  $8 \times 8$ . Nous avons alors  $32 \times 32 = 1024$  patches. En supposant que notre image possède des chaînes RGB et que nous intégrons chaque patch à un vecteur de 512 dimensions, nous avons besoin d'une couche linéaire de forme  $(192, 512)$ . La procédure ci-dessus est assez complexe, mais en pratique, nous pouvons réaliser l'opération ci-dessus avec une simple couche de convolution. Vous allez l'implémenter. Veuillez compléter l'argument manquant de `self.proj`.
2. Implémentez le `class MixerBlock` et le `class MLP Mixer` en vous appuyant sur le schéma d'architecture.

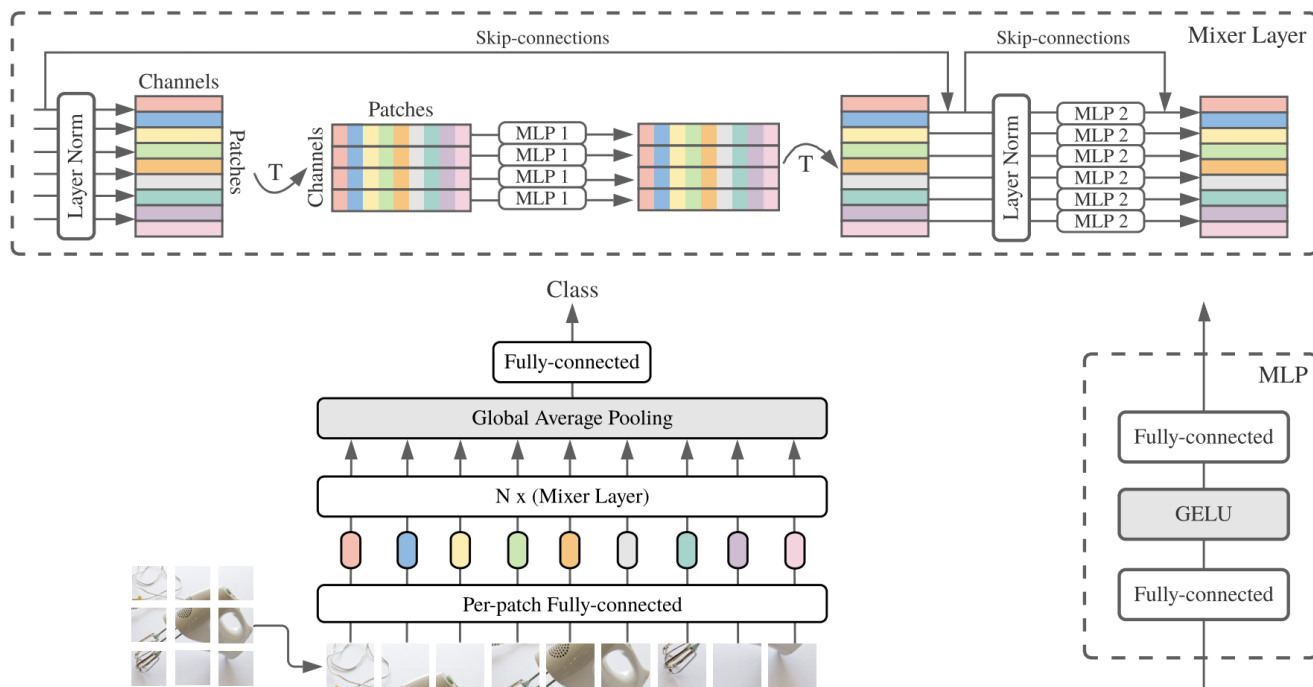


Figure 2: Reproduite de l'article de MLP-Mixer. MLP-Mixer se compose des embeddings linéaires par patch, des couches Mixer et d'une tête de classification. Les couches de mixage contiennent un MLP de mixage de tokens et un MLP de mixage de chaînes, chacun étant composé de deux couches entièrement connectées et d'une non-linéarité GELU. Les autres composants comprennent : des connexions de saut, le dropout, et la norme de couche sur les chaînes.

## Problem 4

**L'entraînement des modèles (60pts)** Dans ce problème, vous allez entraîner l'architecture implémentée ci-dessus et effectuer une recherche d'hyperparamètres. Veuillez consulter `main.py` ou `main.ipynb`. Pour chaque architecture, nous fournissons les fichiers de configuration json du modèle dans `model_configs`. Pour chacune des expériences, entraîner pour au moins 10 époques. Veuillez soumettre un rapport PDF pour répondre aux questions suivantes:

- (5pts) Complétez la fonction `cross_entropy_loss` dans `utils.py` **sans** recourir aux fonctions préexistantes `torch.nn.CrossEntropyLoss` ou `torch.nn.functional.cross_entropy`.
- (10pts) Pour l'architecture MLP, étudiez l'effet de la non-linéarité tout en conservant les autres hyperparamètres par défaut. Vous devez fournir quatre figures correspondant à la fonction de coût d'apprentissage, à la fonction de coût de validation, à la précision d'apprentissage et à la précision de validation, l'axe des x indiquant le nombre d'époques. Pour chaque figure, utilisez la légende pour indiquer la non-linéarité utilisée. Déterminez quelle non-linéarité est la meilleure et donnez votre explication. Nous fournissons facultativement la fonction utilitaire de visualisation dans `utils.py`.

3. (10pts) Pour l'architecture ResNet18, étudiez l'effet du taux d'apprentissage avec l'optimiseur Adam. Choisissez des taux d'apprentissage de 0,1, 0,01, 0,001, 0,0001, 0,00001. Fournissez les graphiques et expliquez vos résultats.
4. (15pts) Pour MLP Mixer, examinez l'effet de la taille du patch. Aucune valeur recommandée n'est donnée, et vous devez effectuer au moins 3 expériences. N'oubliez pas qu'il n'y a que quelques valeurs valables pour la taille du patch pour une taille d'image donnée. Veuillez fournir des graphiques et expliquer vos résultats. Expliquez également par écrit l'effet sur le nombre de paramètres du modèle ainsi que sur le temps d'exécution.
5. (10pts) Trouvez votre meilleur modèle ResNet18 en jouant avec les hyperparamètres. Mentionnez les hyperparamètres dans votre rapport. Visualisez les noyaux de la première couche, qui a un poids de forme (out\_channel, in\_channel, kernel\_size, kernel\_size). Vous pouvez modifier le fichier `main.py` ou ajouter une cellule supplémentaire dans `main.ipynb` pour la visualisation. Puisque nous avons 64 chaînes de sortie et 3 chaînes d'entrée ( RGB ), on peut voir cela comme soixante-quatre  $3 \times 3$  petites images, où chaque image représente le noyau correspondant à cette chaîne de sortie. Notez qu'il s'agit d'une question ouverte. Vous pouvez effectuer différents prétraitements pour la visualisation, par exemple, normaliser les valeurs de poids, faire la moyenne entre les chaînes pour obtenir des images en échelle de gris, etc. Vous pouvez voir plus de détails et d'exemples dans ce [blogpost](#). Veuillez décrire votre démarche de visualisation dans votre rapport.
6. (5pts) Définissez la taille du patch à 4, et trouvez les hyper-paramètres pour votre meilleur modèle de MLP Mixer. Fournissez les hyper-paramètres complets dans votre rapport. Visualisez les poids (uniquement la première couche) du MLP de mélange de tokens dans le premier bloc comme décrit dans la Figure 5 du document [MLP Mixer](#). Commentez et comparez vos résultats avec les visualisations de la convolution. Expliquez ce qui, selon vous, justifie le succès du MLP Mixer, notamment par rapport aux MLP normaux.
7. (10pts) • (10pts) Étudiez comment la largeur des couches MLP de mélange de tokens et de mélange de canaux affecte la capacité de généralisation du modèle. Entraînez le modèle MLP-Mixer avec différentes largeurs pour les couches MLP de mélange de tokens et de mélange de canaux (par exemple, 256, 512, 1024). Comparez les précisions d'entraînement et de validation pour différentes largeurs. Analysez comment la capacité des couches MLP influence le surajustement ou le sous-ajustement.
8. (5pts) Comparez le flux de gradient (par exemple, les normes des gradients à différentes couches) lors de la rétropropagation pour les trois architectures (MLP, ResNet18 et MLP-Mixer). Analysez et comparez le comportement des gradients dans chaque architecture.