Big Data

Projet de reconnaissance d'images

-

Rapport 2ème partie

Membres du groupe : Jeffrey GONCALVES, Kévin LACOSTE

Enseignants : Claude BARRAS, Théophile SANCHEZ

Réponses aux questions

*-- Ligne 52, à quoi correspond une **epoch** ?*

Une epoch consiste en une itération du dataset d'entrainement fourni au réseau neuronal afin qu'il s'améliore à chaque itération. Dans notre cas, cela correspond aux données contenues dans le tenseur torch *train_data* que l'on fournit par défaut 10 fois au réseau pour qu'il s'entraine (10 Epoch).

-- Qu'est ce qu'un mini-batch et pourquoi l'utilise-t-on ?

Un mini-batch est un morceau du dataset complet des données d'entrainement. Une telle subdivision est utilisé afin d'accélérer la procédure d'entrainement en effectuant celle-ci morceau par morceau tout en vérifiant à chaque morceau le taux de succès obtenu et de modifier les poids du réseau en conséquence.

En effet, cela permet au réseau de s'améliorer plusieurs fois par epoch au lieu de devoir attendre la fin de l'entrainement pour pouvoir avoir un feedback sur le succès du réseau et de ne s'améliorer qu'à fin du traitement d'une epoch pour du traitement dans mini-batch, ce qui diminue considérablement la vitesse d'entrainement du réseau neuronal.

*-- À quoi correspondent les méthodes **__init__** et **forward** de la classe **CNN** ?*

La méthode **__init__** permet d'initialiser les convolutions et couches neuronales qui seront utilisées par le CNN pour le traitement (à noter que *CNN* hérite de *torch.nn.Module*).

Quand à la méthode **forward**, elle permet de faire traiter les données d'entrainement ou de test pour retourner un tableau composé, pour chaque donnée atomique traitée (une image ici), du “”score” obtenu pour chaque label / classe. Pour trouver les classes prédites par le réseau, il suffit alors, pour chaque image, de voir parmi les scores obtenus pour nos 10 labels le score le plus élevé : le label possédant le score le plus élevé est alors choisi comme label prédit pour l'image.

-- À partir de la ligne 13, à quoi servent ***`nn.Conv2d`*** *et* ***`nn.Linear`*** *?*

`nn.Conv2d` sert à créer les convolutions nécessaires au bon fonctionnement du CNN (rappel : **CNN** = **C**onvolutional **N**eural **N**etwork) avant le traitement neuronal.

`nn.Linear`, quant à lui, sert de traitement neuronal afin de réaliser la prédiction. En résumé, nos couches de neurones sont représentées par les ***`nn.Linear`***

-- À partir de la ligne 19, à quoi servent ***`F.relu`*** *et* ***`F.max_pool2d`*** *?*

`F.relu` est une application (au sens mathématique) qui renvoie 0 si une valeur négative lui est passée, et qui renvoie la valeur entrée sinon. Elle sert ici de fonction d'activation typique des CNN.

Quand à ***`F.max_pool2d`***, c'est une application qui effectue un pooling sur 2 dimensions afin de réduire la dimension des données entrées (par exemple, un pooling 2D sur une grille 2x2 transforme chaque groupe de 4 pixels formant un carré de 2x2 de l'image en un seul pixel). Elle est utilisée en combinaison avec des convolutions afin d'accélerer le traitement du réseau neuronal, un trait encore une fois caractéristique des CNN.

-- Ligne 50, à quoi sert l'`optimizer optim.SGD` *?*

L'optimiseur ***`optim.SGD`*** sert à fournir des poids au réseau neuronal afin d'améliorer les performances de celui-ci à chaque itération en mettant en avant les données intéressantes (patternes, etc.) et en réduisant l'importance des autres données moins importantes. Sans cet optimiseur, le réseau neuronal n'évolue pas et quel que soit le nombre d'epoch, les résultats de celui-ci restent inchangés et très faibles (< 10%).

-- Ligne 55, à quoi correspondent les prédictions faites par le réseau ?

Les predictions contenues dans *predictions_train* sont les différents “scores” obtenus pour chaque label, et ce pour chaque image d’entrainement du mini-batch traité. Le label choisi sera celui ayant pour l’image concernée le score le plus grand.

-- Ligne 57, qu’est ce que la `loss` ?

La ***`loss`*** est le tenseur crée via l’appel de *`F.nll_loss`* sur *predictions_train* et les vrais labels correspondant aux labels des images ayant été traitées pour créer *predictions_train*. Il est utilisé avec l’optimiseur et lui permet de connaitre les prédictions sur lequels le réseau s’est trompé afin que l’optimiseur ajuste les poids du réseau et que la précision des prédictions du réseau neuronal s’améliorent. Sans le ***`loss`***, l’optimiseur ne modifiera pas de poids et le réseau neuronal n’évoluera pas, ce qui revient à ne pas avoir d’optimiseur.

-- Que fait la ligne 58, `loss.backward()` ?

La ligne ***`loss.backward()`*** calcule le gradient du tenseur ***`loss`*** pour le mettre dans ce même tenseur. Cela est ensuite utilisé afin de fournir un gradient à l’optimiseur lui permettant de régler ses poids en fonction de ce qui a été un succès ou un échec.

-- Quel était le problème avec la valeur précédente du paramètre (pensez en terme de problème d’optimisation) ?

Le problème concernait l’hyperparamètre **`learning_rate`** et plus particulièrement la valeur qui lui était affectée. En effet, sa valeur était de 10 (ce qui est supérieur à 1), alors que la fonction du taux d’apprentissage est censé diminuer. Donner à **`learning_rate`** une valeur inférieure à 1 (comme 0.0001 par exemple) permet de résoudre ce problème.

Cela avait pour conséquence de faire stagner le réseau neuronal en terme de performance en lui faisant modifier ses poids au hasard au lieu de lui faire modifier ses poids de manière judicieuse.

-- Comparez la précision de l'algorithme sur les données d'entraıˆnement et de validation, que remarquez-vous, comment s'appelle ce phénomène ? Qu'elles peuvent être les causes de ce problème ?

Il est possible de remarquer que dans la très grande majorité des cas, les performances du classifieur sur les données d'entraînement sont meilleures que sur le set de test : nous constatons une différence qui peut varier de 10 à 35 %, voire plus. Cela peut être dû à deux raisons :

- Soit le set d'entraînement est trop petit, et ne couvre pas tous les patterns qui peuvent apparaître dans les images de test (**Under-fitting**)
- Soit le set d'entraînement est trop grand ou comprend un nombre d'epoch trop élevé, ce qui l'amène à ne se focaliser que sur les patterns spécifiques aux données d'entraînement, qui n'apparaissent pas forcément dans les données de test (**Over-fitting**)

Dans notre cas, cela doit être de l'under-fitting en raison du nombre réduit d'images d'entraînement au départ (100 images) par rapport au nombre d'images test (1000 images). Cela n'arrive plus si nous fournissons à notre modèle l'ensemble des données d'entrainement.

Evaluations des performances des réseaux neuronaux

-- Comparer les trois architectures (CNN, MLP et LeNet) en donnant leur meilleur score sur les données de validation et le score correspondant sur le jeu d'entraıˆnement. Vous donnerez aussi les temps d'exécution, la valeur des hyperparamètres choisies et le nombre de paramètres (poids et biais du réseau) pour les trois entraıˆnement. Quelle sont les avantages des CNNs par rapport aux MLP ?

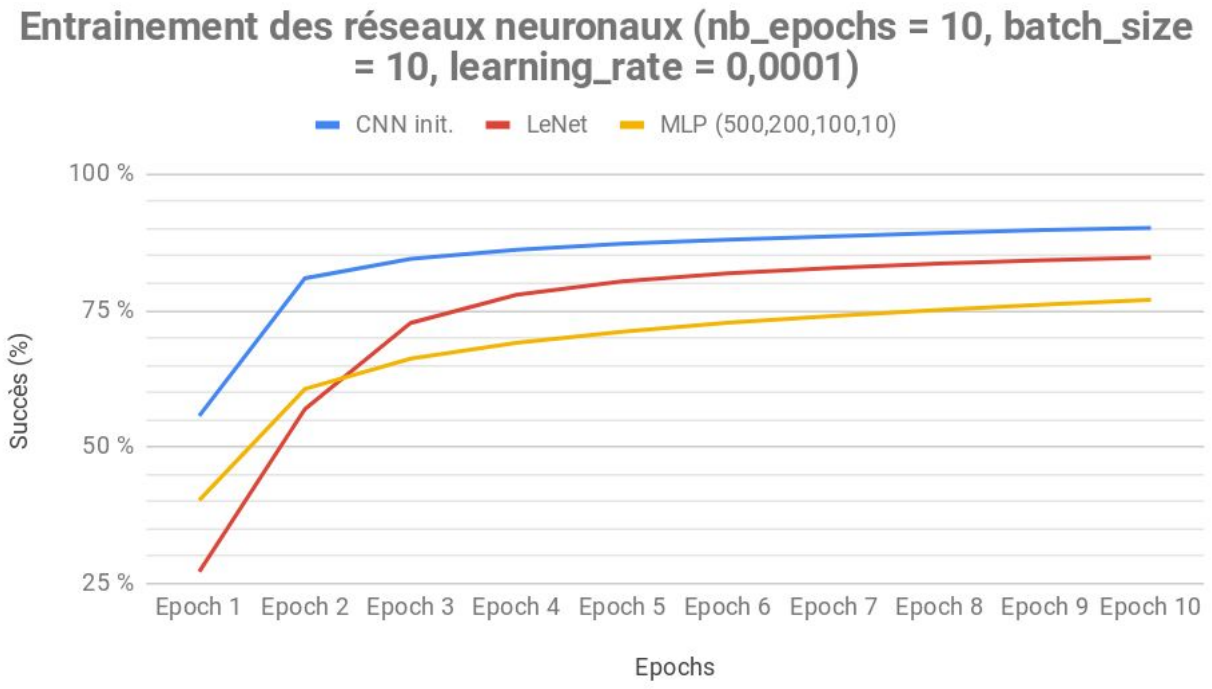
Pour cette question, les tests ont été effectués sur toutes les données de test et d'entrainement (73257 images du train et 26032 images de test) et en modifiant les epochs et la taille d'un mini-batch. 4 sets ont pu être réalisés :

- Nombre d'epochs = 10, taille d'un mini-batch = 10, learning rate = 0,0001
- Nombre d'epochs = 10, taille d'un mini-batch = 5, learning rate = 0,0001
- Nombre d'epochs = 5, taille d'un mini-batch = 10, learning rate = 0,0001
- Nombre d'epochs = 5, taille d'un mini-batch = 5, learning rate = 0,0001

De plus, par la suite, et avec les paramètres énoncés ci-dessus, les 3 réseaux neuronaux utilisés ici seront comparés en fonction de :

- leur courbe d'apprentissage
- leurs performances finales sur le dataset d'entrainement
- leurs performances finales sur le dataset de test

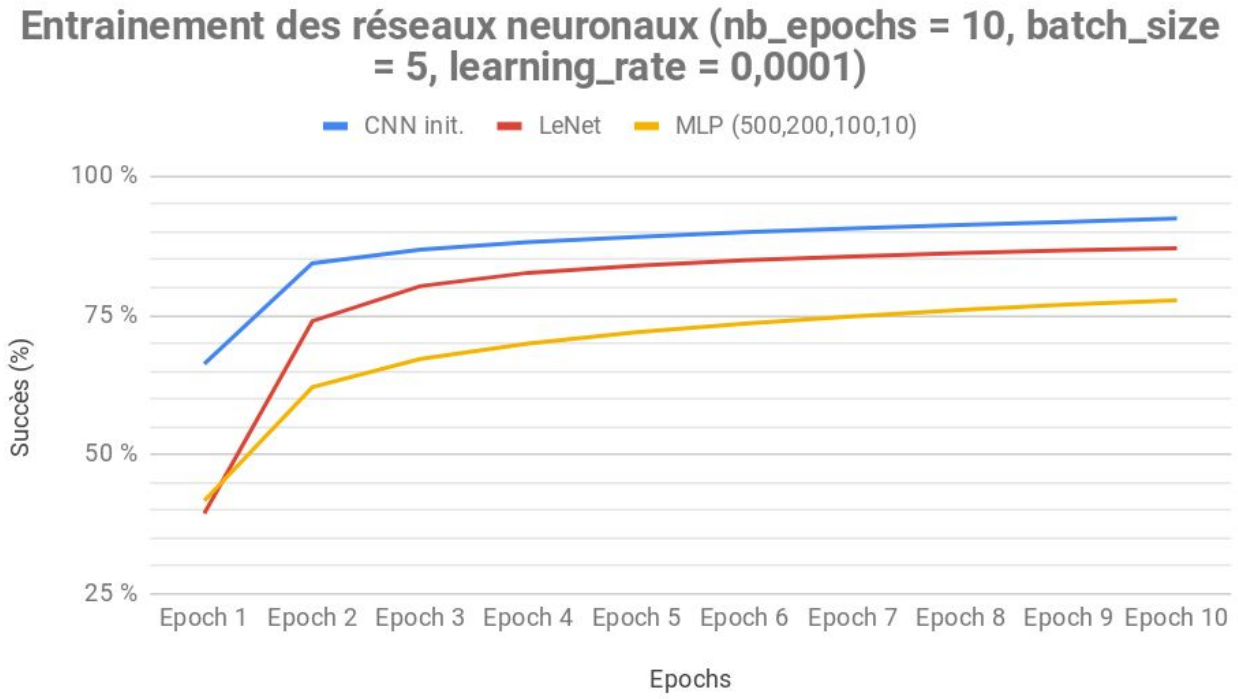
I. Nombre d'epochs = 10, taille d'un mini-batch = 10, learning rate = 0,0001



Taux de succès final sur les données de test :

- CNN : 86,44 %
- LeNet : 82,66 %
- MLP (500n -> 200n -> 100n -> 10n) : 70,84 %

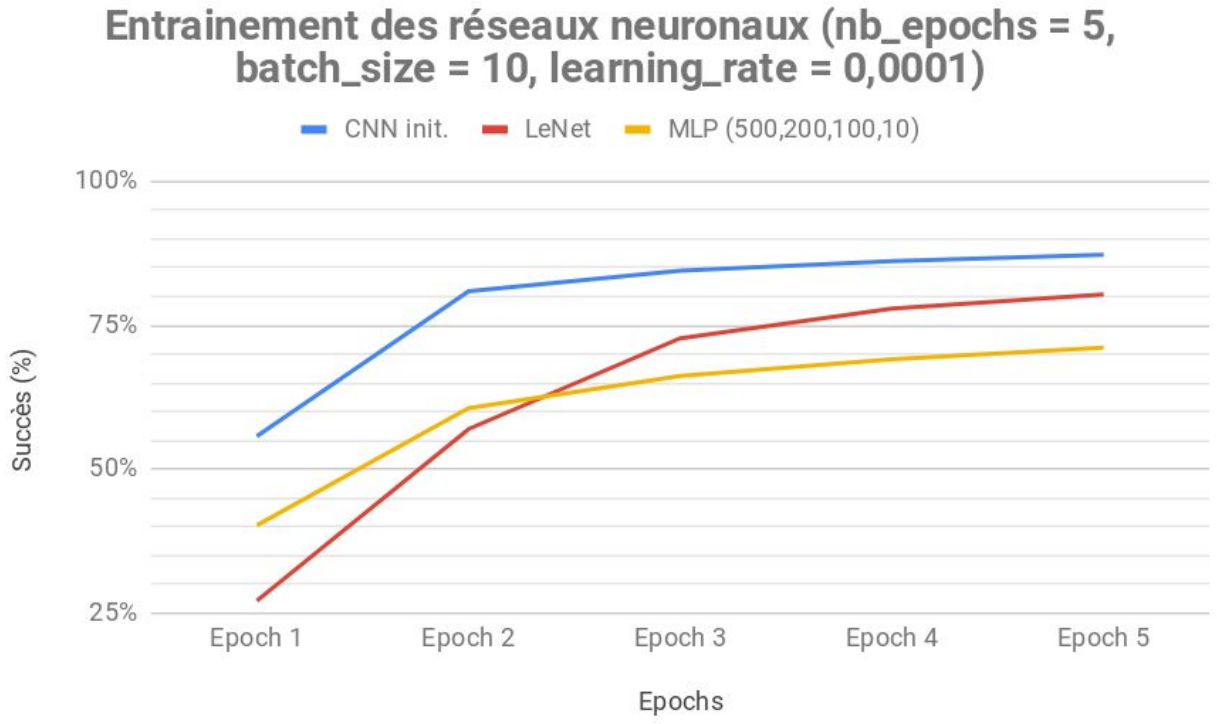
II. Nombre d'epochs = 10, taille d'un mini-batch = 5, learning rate = 0,0001



Taux de succès final sur les données de test :

- CNN : 86,81 %
- LeNet : 84,55 %
- MLP (500n -> 200n -> 100n -> 10n) : 70,84 %

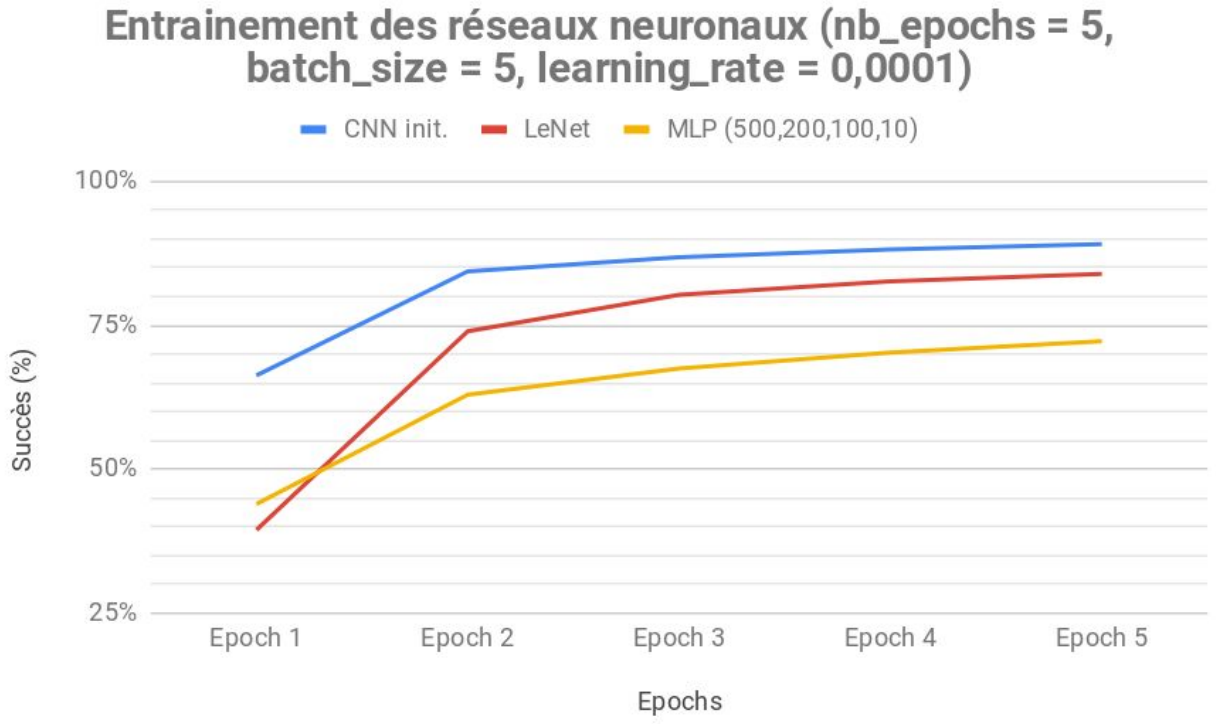
III. Nombre d'epochs = 5, taille d'un mini-batch = 10, learning rate = 0,0001



Taux de succès final sur les données de test :

- CNN : 84,98 %
- LeNet : 78,90 %
- MLP (500n -> 200n -> 100n -> 10n) : 68,66 %

IV. Nombre d'epochs = 5, taille d'un mini-batch = 5, learning rate = 0,0001



Taux de succès final sur les données de test :

- CNN : 86,08 %
- LeNet : 78,90 %
- MLP (500n -> 200n -> 100n -> 10n) : 68,32 %

V. Comparaison des résultats sur les datasets de test

	CNN	LeNet	MLP (500,200,100,10)
Nombre Epochs = 10 Taille mini_batch = 10	86,44	82,66	70,84
Nombre Epochs = 10 Taille mini_batch = 5	86,81	84,55	70,84
Nombre Epochs = 5 Taille mini_batch = 10	84,98	78,9	68,66
Nombre Epochs = 5 Taille mini_batch = 5	86,08	78,9	68,32

Il est possible ici de constater que le plus performant des réseaux neuronaux ici est le **CNN** qui nous a été fourni de base (en moyenne 85% de succès), suivi de près par le réseau **LeNet** (en moyenne 82% de succès), et en bon dernier nous avons le **MLP** (en moyenne 70% de succès).

De plus, nous constatons ici qu'effectuer 5 epochs avec des mini_batchs de 5 images suffit ici pour entrainer nos réseaux neuronaux avec toutes les images d'entrainement d'après les courbes d'évolution de l'entrainement de nos réseaux (voir courbes précédentes) et que le MLP ne pourra probablement jamais être égal aux CNNs même en augmentant le nombre d'epochs ou en diminuant la taille d'un mini-batch.

Enfin, il nous semble important de noter que le MLP garde malgré tout un avantage par rapport aux CNNs qui reste négligeable : le temps nécessaire pour l'entrainement. En effet, tandis que les CNNs mettent en moyenne 1h pour traiter les données d'entrainement (ce qui reste acceptable si le système est utilisé sur le long-terme), le MLP choisi ici est bien plus rapide, ne mettant en moyenne qu'une dizaine de minutes.

Pour conclure, l'avantage des CNNs sur les plus simples perceptrons multi-couches est principalement la capacité des CNNs à détecter des patternes récurrents dans les images et de repérer les dépendances spatiales à observer pour chaque label. Cela rend donc les CNNs adaptés au traitement d'images où les objets à détecter ont des formes semblables, comme dans le cas de chiffres.