Reconnaissance des formes pour l'analyse et l'interprétation d'images

Rapport TP 5–6 : Réseaux convolutionnels pour l'image

Etudiant :

DJEGHRI Amine

MAMOU Idles

Numéro :

3801757

3803676

Novembre 2020

Partie 1 – Introduction aux réseaux convolutionnels

1. En considérant un seul filtre de convolution

- La taille de sortie sera :

$$X' = \frac{x - k + 2p}{s} + 1$$

$$Y' = \frac{y - k + 2p}{s} + 1$$

- Nombre de poids à apprendre est: $K^2 \times Z + 1$ pour le bias
- Le nombre de poids qu'il aurait fallu apprendre si une couche fully-connected devait produire une sortie de la même taille : $x * y * z * X' * Y'$

2.
Les avantages apportés par la convolution par rapport à des couches fully-connected :
- Complexité inférieure en terme de poids (apprendre moins de poids)
- On est indépendant aux dimensions d'entrée
- Voir des bouts de l'image (notion de localité) et détecter les patterns

Sa limite est : La perte d'information

3. L'intérêt à l'usage du pooling spatial est : réduire la dimension des features maps par conséquent réduire le nombre de paramètres et aussi l'invariance aux transformations.

4. On peut calculer sans modifier l'image s'il y a au début du réseau **des couches de convolutions** car elles ne dépendent pas de la taille de l'entrée contrairement aux couches fully connected, **mais on va s'arrêter au passage de la couche de convolution à la couche fully connected**
Pour parier à ca : on peut faire un global pulling

5. On peut voir les couches fully-connected comme des convolutions particulières de taille 1x1 au nombre N (ont la même taille que l'image d'entrée) avec un padding nul

6. Si on remplace donc les fully-connected par leur équivalent en convolutions nous pourrons calculer la sortie mais la taille de cette dernière dépendra de l'image en entrée

7. Les tailles des receptive fields des neurones
de la première couche de convolution : kernel_conv1x kernel_conv1
de la deuxième couche de convolution : (kernel2 -1 + kernel1)²
Les couches profondes représenteront des formes complexes

Partie 2 – Apprentissage from scratch du modèle

2.1 Architecture du réseau

8. Les valeurs de padding et de stride à choisir sont :

La règle : $Taille_{entrée} = Taille_{sortie}$

$$X' = \frac{x - k + 2p}{s} + 1 = x$$

$$X' * s = x - k + 2p + s$$
$$p = \frac{X' * s - (x - k + s)}{2}$$

Exemple :
Stride = 1
Padding :

$$p = \frac{K-1}{2}$$

9. Pour réduire les dimensions spatiales d'un facteur 2 :
un padding = 0 , un kernel de taille= 2x2 , stride =2

10.

Couches	Taille de sortie	Nombre de poids
Entrée taille 32 x 32 x 3		
—32 conv1 : convolutions 5×5, suivie de ReLU	32x 32 x 32	(5 x 5 x3 +1) x 32= 2400 (le +1 pour le bias)
— pool1 : max-pooling 2×2	16x16x32	0
— conv2 : 64 convolutions 5×5, suivie de ReLU	16x16x64	(5x5x32+1)x64= 51200
— pool2 : max-pooling 2×2	8x8x64	0
— conv3 : 64 convolutions 5×5, suivie de ReLU	8x8x64	(5x5x64+1)x64= 102400
— pool3 : max-pooling 2×2	4x4 x 64	0
— fc4 : fully-connected, 1000 neurones en sortie, suivie de ReLU	1000	(4x4 x 64) x 1000= 1024000
— fc5 : fully-connected, 10 neurones en sortie, suivie de softmax	10	10 x 1000 = 10000

11. Nombre total de poids : 2432 + 51264 + + 102464+ 1024000 + 10000 = 1 190 160
- Nombre d'exemples 50k trop petit par rapport au nombre de paramètres, risque de overfitting

12.
BoW 128 x 1001= 128000,
10 svms pour 10 classes = 10 parametres
Le réseau convolutionnel possède plus de paramètre

13. sur le notebook

2.2 Apprentissage du réseau

14. La différence entre la façon de calculer la loss et l'accuracy en train et en test :
- Dans le test on ne calcule pas le gradient, on ne fait pas de backpropagation et donc on ne met pas à jour les poids du modèle
- Dans le test on utilise model.eval() au lieu de model.train()

15. sur le notebook

16. Effets sur la convergence du modèle

Pas d'apprentissage (Learning rate) : c'est un hyper-paramètre

- Si le pas et trop petit, le modèle risque de mettre trop de temps à converger
- Si le pas est trop grand, risque de non convergence

La taille de mini-batch
batch size trop petit = mauvaise estimation du gradient + trop de bruit
trop grand = calcul long + la courbe moins bruité

17. L'erreur au début de la première époque correspond à un modèle pas encore entrainé et qui utilise des poids initialisés aléatoirement, les prédictions sont donc aléatoires

18.

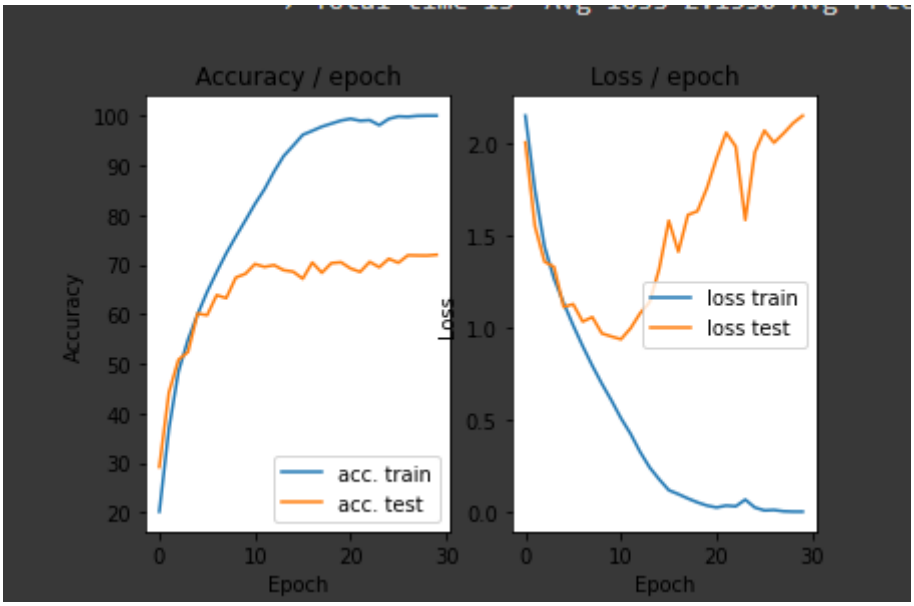


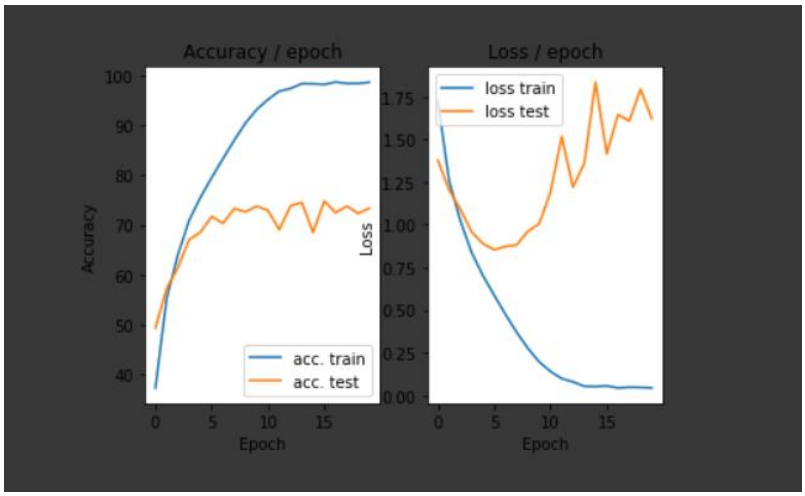
Figure 1 accuracy et loss de l'apprentissage du modèle sur le dataset CIFAR10

Nous remarquons que la loss du train diminue jusqu'à atteindre 0, cependant la loss du test diminue jusqu'à une certaine epoch ou elle remonte, on peut donc dire qu'il y a un **overfitting** (sur-apprentissage)
En ce qui concerne l'accuracy, nous remarquons que les deux accuracy du train et du test augmentent, mais à partir d'une certaine epoch, l'accuracy du train continue d'augmenter en revanche celle du test n'augmente plus et stagne.

Partie 3 – Améliorations des résultats

3.1 Normalisation des exemples

19.



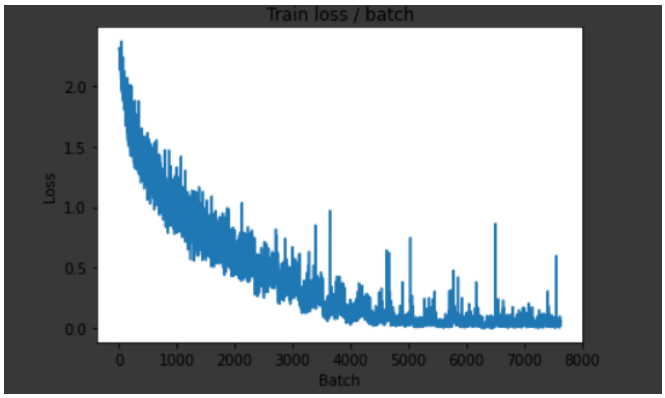


Figure 2 : Accuracy et loss après la normalisation des exemples

Le modèle apprend plus vite, il a atteint les 100% en train (a peu prêt) à la 11ème époch, et 70% en test, cependant Il y a toujours un sur-apprentissage du modèle même après l'utilisation de la normalisation

20. Car nous n'avons pas accès aux données du test et nous ne devons pas les utiliser
21. L'algorithme ZCA utilise la décomposition par composantes principales PCA modifiée permettant d'avoir les avantages de la méthode PCA tout en gardant des images semblables à la forme d'origine

3.2 Augmentation du nombre d'exemples d'apprentissage par data augmentation

22.

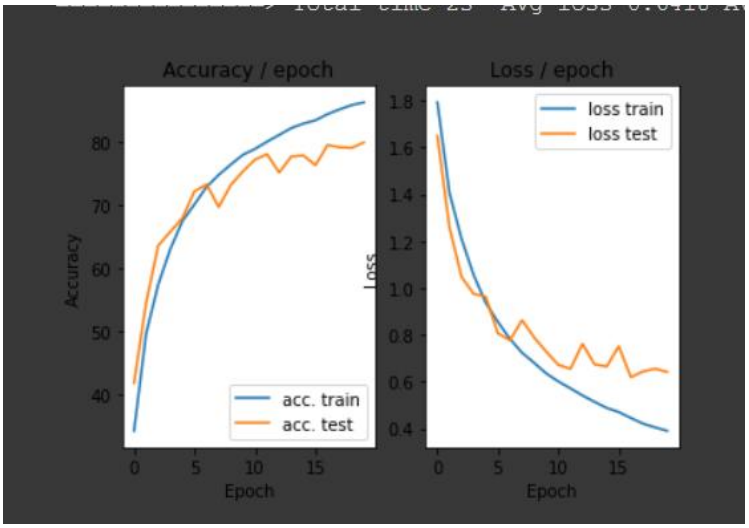


Figure 3 – accuracy et loss après la data augmentation

On remarque que la data augmentation est efficace, en effet notre modèle sur-apprend beaucoup moins et l'accuracy lors du test atteint les 80%

23. Non, elle n'est pas utilisable sur tout type d'image.
Elle peut être utilisée sur les images d'animaux, par exemple, et ne peux pas être utilisée sur des images représentants des chiffres ou des lettres.

24. Les limites sont :
- Les images générées peuvent ne pas être utiles à l'apprentissage si les transformations ne sont pas cohérentes, ou ne sont pas assez représentatives. Aussi, un risque de biaiser le Dataset

25. D'autres techniques : GANs, rotation, ajouter du bruit, changer la luminosité

3.3 Variantes sur l'algorithme d'optimisation

26.

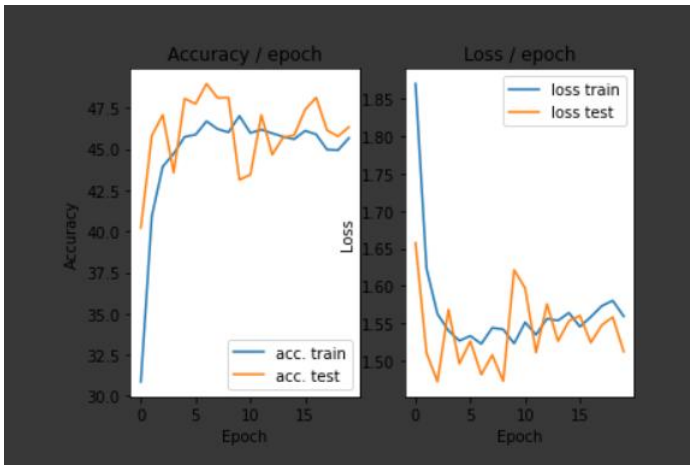


Figure 4.1 : accuracy et loss avec un momentum sans LR scheduler

Nous avons augmenté le nombres d'époch (de 15 à 30)

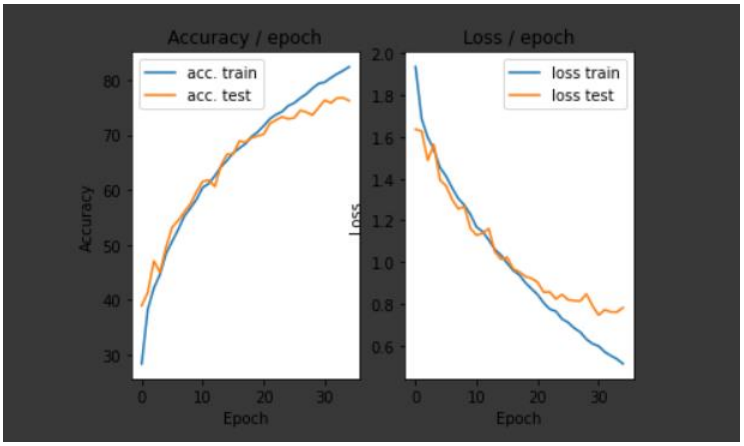


Figure 4.2 : Accuracy et loss après l'ajout d'un momentum et d'un LR Scheduler

Figure 4.1 : En rajoutant un momentum seulement, nous pouvons voir que le modèle trouve des difficultés à converger (train et test au tour des 45%)

Figure 4.2 : En rajoutant un momentum de 0.9 et d'un learning rate scheduler en utilisant une décroissance exponetielle avec un coefficient de 0.95 on peut remarquer que l'apprentissage est nettement amélioré, le modèle n'a plus de problème de convergence même si l'entrainement prend un plus de temps. Après 30 épochs le modèle atteint les 80% en train et 75% en test (et n'a toujours pas fini de converger)

27.

- Le momentum sert à accélérer la convergence et réduire les oscillations, il augmente le pas de la descente du gradient quand on a la même direction et réduit ce dernier quand la direction change. Il ajoute le gradient calculé à l'étape précédente en le pondérant par une valeur <1 (0.9 dans notre cas)
- Le learning rate scheduler adapte le pas d'apprentissage, au début on a un grand pas d'apprentissage pour approcher rapidement le minimum, puis au fur et à mesure de notre convergence il réduit ce pas d'apprentissage afin de converger.

28. De nombreuses autres variantes de la SGD existent et de nombreuses stratégies de planification de learning rate existent, par exemple : ADAM, Adagrad

3.4 Régularisation du réseau par dropout

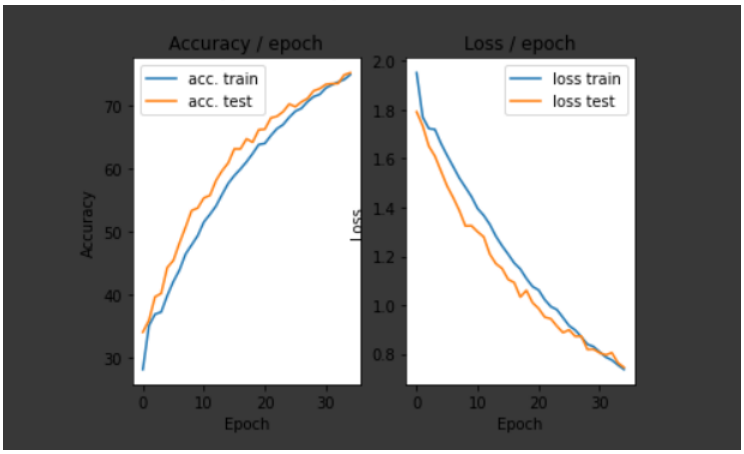


Figure 5 : Accuracy et loss après l'ajout de Dropout

29. En rajoutant un dropout, nous pouvons voir que nous somme dans une situation où les performances en tests sont (équivalentes) voir meilleures que celles en train, grâce au dropout, au bout de 30 epochs nous sommes à 80% accuracy en train et en test (le modèle n'a pas fini de converger)
Les épochs sont plus rapides car il y a moins de poids

30. La régularisation vise à limiter le surapprentissage en pénalisant la complexité du modèle et contrôler l'erreur de type variance pour aboutir à de meilleures performances.

31. Le dropout permet de désactiver aléatoirement à chaque couche certaines neurones de notre réseau pour diminuer le nombre de paramètres. Ceci évite le sur-apprentissage du fait qu'on empêche que certains neurones soient plus influents et qu'elles aient plus d'importance que d'autres. En désactivant ces neurones durant l'apprentissage on oblige les autres neurones de prendre de l'importance eux aussi.

32.L'hyper-paramètre est P : la probabilité qu'un neurone soit désactivé
Si P est trop petit, on perd l'avantage du dropout et on risque de sur-apprendre
Si P est trop grand, les neurones activés auront un poids supérieur et le réseau sera instable
De préférence P= 0.5

33. Pendant l'apprentissage, la couche dropout désactive aléatoirement les neurones, et pendant l'évaluation, le dropout est désactivé, elle ne désactive aucun neurone

3.5 Utilisation de batch normalization

34.

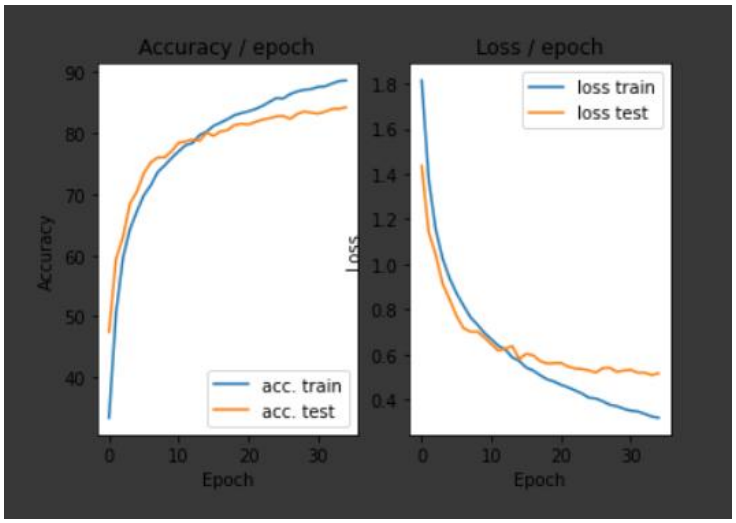


Figure 6 : Accuracy et loss après batch norm

Nous pouvons voir que rajouter la batch normalization accélère l'apprentissage, le modèle atteint les 80% d'accuracy en test et en train en 10 épochs seulement