Compte rendu TP 6-7
Deep learning

Elodie Difonzo
Roqyun Ko

1. **Considérant un seul filtre de convolution de padding p, de stride s et de taille de kernel k, pour une entrée de taille x × y × z quelle sera la taille de sortie ? Combien y a-t-il de poids à apprendre ? Combien de poids aurait-il fallu apprendre si une couche fully-connected devait produire une sortie de la même taille ?**
   La taille de sortie sera :
   $w_s = \frac{w + 2p - k}{s} + 1$
   $h_s = \frac{h + 2p - k}{s} + 1$
   Les poids pour la convolution : $p = (k \times k) \times z$
   Les poids pour une couche fully connected : $p = m \times n$
   avec $m = x \times yetn = w_s \times h_s$

2. **Quels avantages apporte la convolution par rapport à des couches fully-connected ? Quelle est sa limite principale ?**
   Les avantages de la convolution sont l'invariance par translation et par rotation. De plus, il y a beaucoup moins de poids pour la convolution donc l'apprentissage est plus rapide. Dans la pratique, cette méthode est efficace.
   La limite est que la convolution est une vision locale. Il est difficile d'avoir une vision globale sur l'image.

3. **Quel intérêt voyez-vous à l'usage du pooling spatial ?**
   Le pooling spatial réduit la taille de la sortie.On réduit la quantité de paramètres car l'image d'entrée est découpée en plusieurs rectangles de n pixels et on reforme une image intermédiaire plus petite.

4. **Supposons qu'on essaye de calculer la sortie d'un réseau convolutionnel classique (par exemple celui en Figure 2) pour une image d'entrée plus grande que la taille initialement prévue (224×224 dans l'exemple). Peut-on (sans modifier l'image) calculer tout ou une partie des couches du réseau ?**
   Ce n'est pas possible car la sortie sera d'une dimension différente.

5. **Montrer que l'on peut voir les couches fully-connected comme des convolutions particulières.**
   C'est une sorte de convolution de l'image entière.
   Démonstration :
   À l'entrée, on redimensionne l'image pour qu'elle soit de la taille
   $1 \times (longueur \cdot hauteur)$
   Soit $(longueur \cdot hauteur) = n_x$,

Soit la taille de sortie n_y,

$$y_i = \sum_{j=1}^{n_x} W_{x_{ij}} \cdot x_j + b_{y_i} \quad où \quad i \in \{1, \ldots, n_y\}$$

y_i est un produit de n_x poids et n_x pixels. C'est une sorte de convolution dont le *kernel* est de la taille n_x.

6. **Supposons que l'on remplace donc les fully-connected par leur équivalent en convolutions, répondre à nouveau à la question 4. Si on peut calculer la sortie, quelle est sa forme et son intérêt ?**
   Dans ce cas c'est possible car si l'image est plus grande il suffira de déplacer le filtre de convolution sur l'image pour obtenir la sortie. Tous les couches sont des filtres. Cela permet d'obtenir un une globale sur l'image. Par exemple, Fully Convolutional Network(FCN) est utilisé pour détecter les objets et pour segmenter l'image.

7. **On appelle champ récepteur (receptive field) d'un neurone l'ensemble des pixels de l'image dont la sortie de ce neurone dépend. Quelles sont les tailles des receptive fields des neurones de la première et de la deuxième couche de convolution ? Pouvez-vous imaginer ce qu'il se passe pour les couches plus profondes ? Comment l'interpréter ?**
   La taille d'un champ récepteur correspond à la taille du filtre de chaque couche.
   - la taille du champ récepteur de la 1ère couche est : 3x3
   - la taille du champ récepteur de la 2ème couche est : 3x3

   La taille du champ récepteur reste la même jusqu'à la 1ère couche Fully Connected (FC). La taille du champ récepteur de la couche FC est la taille de l'entrée entière. Elle devient de plus en plus grande (25088 -> 4096 -> 4096).

   La petite taille de récepteur donne une vision local alors que la grande taille de récepteur donne une vision globale.
8. **Pour les convolutions, on veut conserver en sortie les mêmes dimensions spatiales qu'en entrée. Quelles valeurs de padding et de stride va-t-on choisir ?**

   On trouve s = 1 et p = 2

$$w_s = \frac{w + 2p - k}{s} + 1 = \frac{32 + 2 \cdot 2 - 5}{1} + 1 = 32$$

$$h_s = \frac{h + 2p - k}{s} + 1 = \frac{32 + 2 \cdot 2 - 5}{1} + 1 = 32$$

9. **Pour les max poolings, on veut réduire les dimensions spatiales d'un facteur 2. Quelles valeurs de padding et de stride va-t-on choisir ?**

s = 2 et p = 0

$$w_s = \frac{w + 2p - k}{s} + 1 = \frac{32 - 2 \cdot 0 - 2}{2} + 1 = 16$$

$$h_s = \frac{h + 2p - k}{s} + 1 = \frac{32 - 2 \cdot 0 - 2}{2} + 1 = 16$$

10. **Pour chaque couche, indiquer la taille de sortie et le nombre de poids à apprendre. Commentez cette répartition.**

	Taille de sortie	Nombre de poids	Nombre de biais	Taille d'image en sortie
Couche de convolution 1	32 car il y a 32 convolutions faites	$(5 \cdot 5) \cdot 32$	$1 \cdot 32$	32 ×32
Pooling 1	32 *(la taille de la couche précédente)*	0	0	16 × 16
Couche de convolution 2	64	$(5 \cdot 5) \cdot 64$	$1 \cdot 64$	16 × 16
Pooling 2	64	0		8 × 8
Couche de convolution 3	64	$(5 \cdot 5) \cdot 64$	$1 \cdot 64$	8 × 8
Pooling 3	64	0	0	4 × 4
Fully-connected 4	1000	$64 \cdot (4 \cdot 4) \cdot 1000$	1000	1000
Fully-connected 5	10	$64 \cdot (4 \cdot 4) \cdot 10$	10	0

11. **Quel est donc le nombre total de poids à apprendre ? Comparer cela au nombre d'exemples.**

Le nombre total de poids à apprendre est :

$(5 \times 5 \times 32) + (5 \times 5 \times 64) + (5 \times 5 \times 64) + (64 \times 4 \times 4 \times 1000) + (64 \times 4 \times 4 \times 10) =$ ***1038240***

12. **Comparer le nombre de paramètres à apprendre avec celui de l'approche BoW et SVM.**

    - Le nombre de paramètres à apprendre pour **BoW** est le nombre de clusters, k pour l'algorithme de K-Means.
    - Il y a 2 paramètres à apprendre pour **SVM** : w et b
      $f(x) =< w, \phi(x) > +b$ où $\phi(x)$ est une *feature function*.

13. **Lisez et testez le code fourni. Vous pouvez passer des paramètres en ligne de commande :**
    python base.py --help pour avoir la liste, par exemple
    python base.py --lr 1.0 pour un learning rate de 1.0

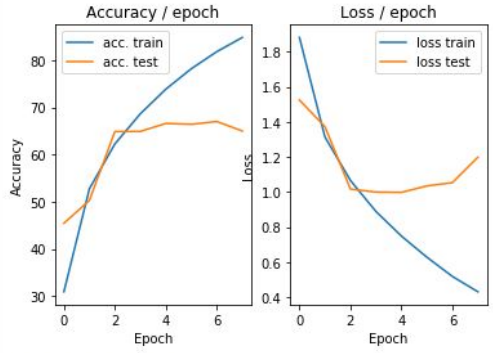
14. **Dans le code fourni, quelle différence importante y a-t-il entre la façon de calculer la loss et l'accuracy en train et en test (autre que les données sont différentes) ?**
    Pour calculer l'accuracy en train et en test les calculs sont pratiquement faits sur des réseaux différents. Certaines couches se comportent différemment pendant l'apprentissage et l'évaluation. Les poids est mis à jour à chaque epoch pour que la loss soit minimisée en utilisant la backpropagation. La précision est aussi influencée.

15. **Modifiez le code pour utiliser la base CIFAR-10 et implémenter l'architecture demandée ci-dessus. Attention à bien faire suffisamment d'époques pour que le modèle ait fini de converger.**

```
self.features = nn.Sequential(
    nn.Conv2d(3, 32, (5, 5), stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
    nn.Conv2d(32, 64, (5, 5), stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
    nn.Conv2d(64, 64, (5, 5), stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
)
# On défini les couches fully connected comme un groupe de couches
# `self.classifier`
self.classifier = nn.Sequential(
    nn.Linear(1024, 1000),
    nn.ReLU(),
    nn.Linear(1000, 10),
    # Rappel : Le softmax est inclus dans la loss, ne pas le mettre ici
)
```

16. **Quels sont les effets du pas d'apprentissage (learning rate) et de la taille de mini-batch ?**
    a. **Training n°1** : Learning rate η = ***0.1*** / la taille de mini-batch = ***32***

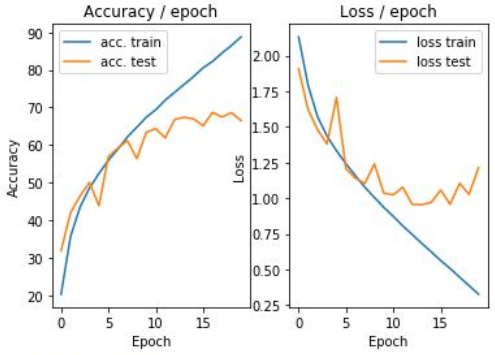
```
===============> Total time 16s Avg loss 0.4311 Avg Prec@1 84.83 %      Avg Prec@5 99.61 %

[EVAL Batch 000/313]    Time 0.087s (0.087s)    Loss 1.2706 (1.2706)    Prec@1  68.8 ( 68.8)    Prec@5  96.9 ( 96.9)
[EVAL Batch 200/313]    Time 0.013s (0.008s)    Loss 0.9726 (1.2095)    Prec@1  75.0 ( 65.3)    Prec@5 100.0 ( 95.6)

===============> Total time 2s  Avg loss 1.1984 Avg Prec@1 65.00 %      Avg Prec@5 95.68 %
```




```
==================
=== EPOCH 9 =====
==================
```

b. **Training n°2** : Learning rate η = ***0.010*** / la taille de mini-batch = ***32***

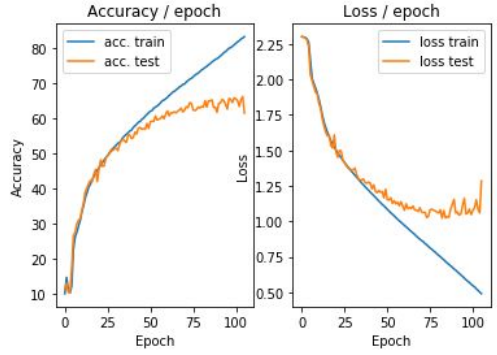
```
===============> Total time 17s Avg loss 0.3278 Avg Prec@1 88.78 %      Avg Prec@5 99.79 %

[EVAL Batch 000/313]    Time 0.107s (0.107s)    Loss 1.0028 (1.0028)    Prec@1  71.9 ( 71.9)    Prec@5  96.9 ( 96.9)
[EVAL Batch 200/313]    Time 0.011s (0.008s)    Loss 1.2225 (1.1896)    Prec@1  65.6 ( 67.0)    Prec@5 100.0 ( 96.8)

===============> Total time 2s  Avg loss 1.2144 Avg Prec@1 66.50 %      Avg Prec@5 96.75 %
```




```
=================
=== EPOCH 21 =====
=================
```

c. **Training n°3** : Learning rate η = ***0.001*** / la taille de mini-batch = ***32***

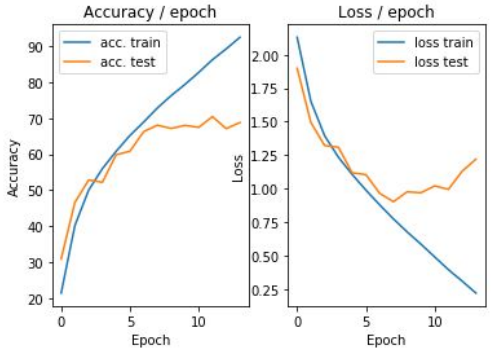
```
===============> Total time 16s Avg loss 0.4903 Avg Prec@1 83.29 %      Avg Prec@5 99.34 %

[EVAL Batch 000/313]    Time 0.105s (0.105s)    Loss 1.6507 (1.6507)    Prec@1  59.4 ( 59.4)    Prec@5  96.9 ( 96.9)
[EVAL Batch 200/313]    Time 0.002s (0.007s)    Loss 1.7063 (1.2763)    Prec@1  59.4 ( 61.8)    Prec@5  93.8 ( 96.3)

===============> Total time 2s  Avg loss 1.2847 Avg Prec@1 61.53 %      Avg Prec@5 96.26 %
```




```
=================
=== EPOCH 107 =====
=================
```

d. **Training n°4** : Learning rate η = ***0.100*** / la taille de mini-batch = ***128***

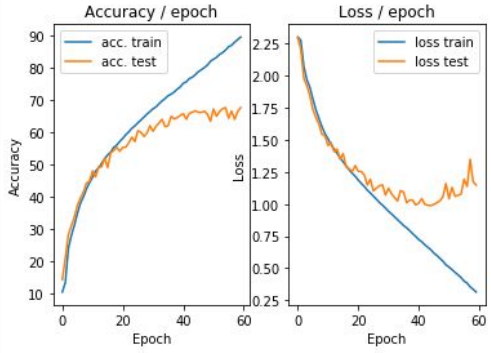
```
===============> Total time 9s  Avg loss 0.2203 Avg Prec@1 92.46 %      Avg Prec@5 99.93 %

[EVAL Batch 000/079]    Time 0.150s (0.150s)    Loss 1.2075 (1.2075)    Prec@1  65.6 ( 65.6)    Prec@5  96.1 ( 96.1)

===============> Total time 1s  Avg loss 1.2201 Avg Prec@1 68.81 %      Avg Prec@5 96.81 %
```




```
=================
=== EPOCH 15 =====
=================
```

e. **Training n°5** :Learning rate η = ***0.010*** / la taille de mini-batch = ***128***

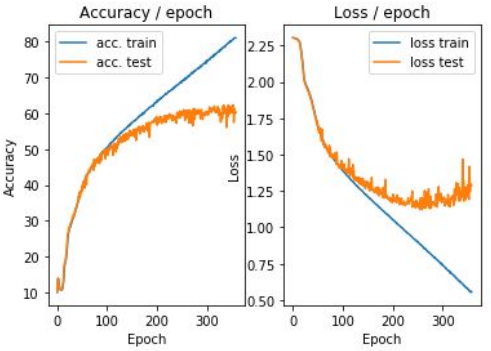
```
===============> Total time 9s  Avg loss 0.3149 Avg Prec@1 89.45 %      Avg Prec@5 99.78 %

[EVAL Batch 000/079]    Time 0.147s (0.147s)    Loss 1.0450 (1.0450)    Prec@1  71.1 ( 71.1)    Prec@5  96.9 ( 96.9)

===============> Total time 1s  Avg loss 1.1475 Avg Prec@1 67.59 %      Avg Prec@5 96.85 %
```




```
=================
=== EPOCH 61 =====
=================
```

f. **Training n°6** :Learning rate η = ***0.001*** / la taille de mini-batch = ***128***

```
===============> Total time 7s  Avg loss 0.5574 Avg Prec@1 81.02 %      Avg Prec@5 99.15 %

[EVAL Batch 000/079]    Time 0.123s (0.123s)    Loss 1.1970 (1.1970)    Prec@1  63.3 ( 63.3)    Prec@5  97.7 ( 97.7)

===============> Total time 1s  Avg loss 1.2865 Avg Prec@1 60.27 %      Avg Prec@5 95.59 %
```




```
=================
=== EPOCH 360 =====
=================
```

Benchmark des résultat des entraînements

Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°1	0.100	32	8	84.83 %	0.431	5	66.60 %	0.9978
n°2	0.010	32	20	88.78 %	0.328	14	67.41 %	0.9547
n°3	0.001	32	106	83.29 %	0.490	83	65.09 %	1.022
n°4	0.100	128	14	92.46 %	0.220	9	68.08 %	0.9025
n°5	0.010	128	61	89.45 %	0.315	41	65.67 %	1.0074
n°6	0.001	128	359	81.02 %	0.557	256	68.83 %	1.1210

Remarque : La meilleure époque est choisi en fonction du coût minimal

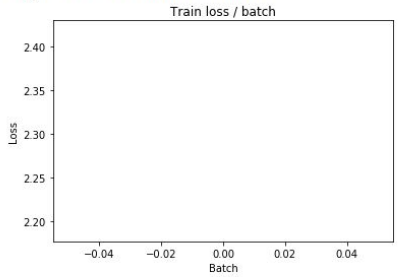
Conclusion :
On observe que la précision de test et le coût de test fluctuent brusquement pour les ηs d'une petite valeur. Par conséquent, on atteint le meilleur résultat après très peu de temps (5 epochs pour ***n°1*** et 9 epochs pour ***n°4***).
Pour les η des plus grandes valeurs, le modèle exige plus de temps pour optimiser le modèle.

De même, prendre un batch d'une grande taille prends plus longue durée pour optimiser modèle. Par exemple, pour la même valeur de η et les mini-batchs de différentes tailles, on atteint *le minimum à l'époque 83* dans ***n°3*** mais on atteint *le minimum à l'époque 256* dans ***n°6*** lors de l'évaluation.

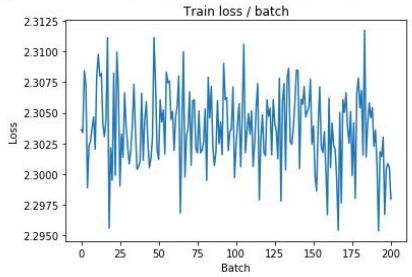
Le taux d'apprentissage permet de modifier plus finement les poids pour que le modèle soit plus précis. Un taux d'apprentissage d'une grande valeur risque de mal optimiser le modèle (un phénomène de "overshoot") alors qu'il permet au modèle d'apprendre rapidement.
On constate que les traces de test évoluent de la manière plus stable les mini-batch de plus grande taille.

17. A quoi correspond l'erreur au début de la première époque ? Comment s'interprète-t-elle ?

```
================
=== EPOCH 1 =====
================

[TRAIN Batch 000/391]   Time 0.145s (0.145s)    Loss 2.3036 (2.3036)
<Figure size 432x288 with 0 Axes>
```



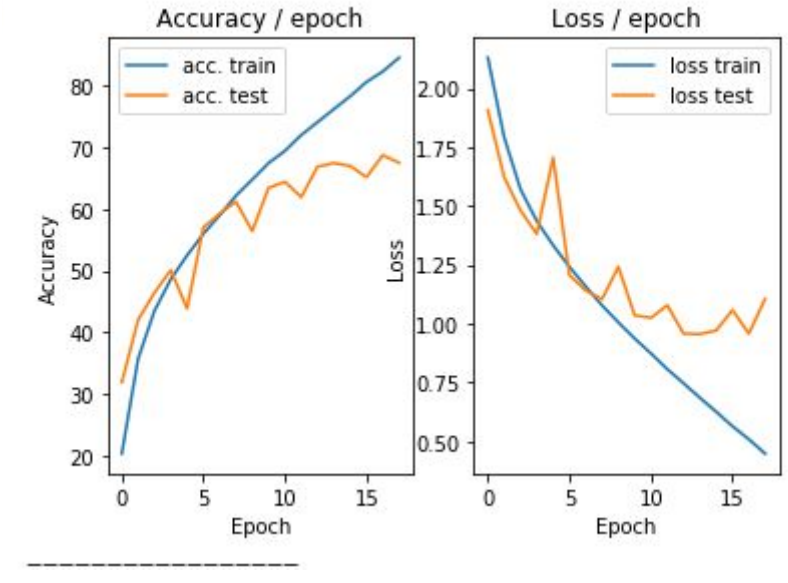

```
[TRAIN Batch 200/391]   Time 0.020s (0.018s)    Loss 2.2979 (2.3037)
```




```
===============> Total time 6s  Avg loss 2.3035 Avg Prec@1 10.05 %

[EVAL Batch 000/079]    Time 0.124s (0.124s)    Loss 2.3078 (2.3078)
```

La première époque est le début de l'entrainement. Le modèle n'a pas appris du tout pour pouvoir classifier correctement les images, donc il donne une erreur de grande valeur.

18. Interpréter les résultats. Qu'est-ce qui ne va pas ? Quel est ce phénomène ?

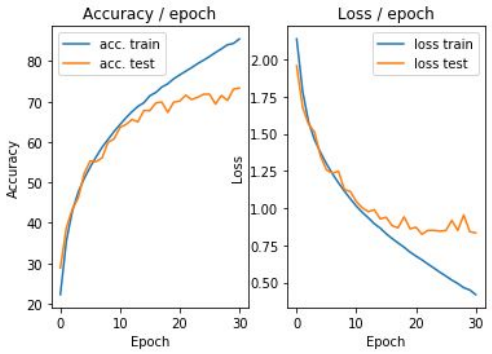


On observe qu'à partir de la 20ème epoch la précision obtenue avec les données de train augmentent pour atteindre 90% tandis que la précision obtenue avec les données de test se stabilise autour de 60%. Il s'agit du phénomène de sur-apprentissage.

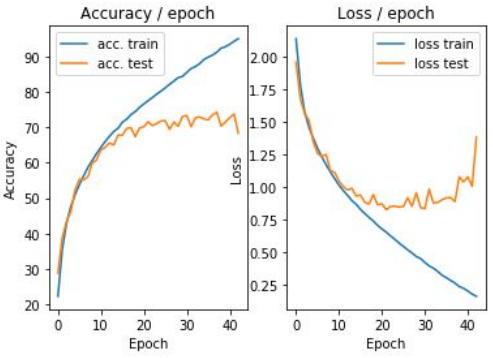
19. Décrire vos résultats expérimentaux.

Training n°7 : Learning rate η ***= 0.01*** / la taille de mini-batch ***= 128 /*** *normalization appliquée* μ *= [0.491, 0.482, 0.447] et* σ *= [0.202, 0.199, 0.201]*

```
===============> Total time 12s Avg loss 0.4193 Avg Prec@1 85.49 %      Avg Prec@5 99.56 %

[EVAL Batch 000/079]    Time 0.153s (0.153s)    Loss 0.6873 (0.6873)    Prec@1  82.0 ( 82.0)    Prec@5  96.9 ( 96.9)

===============> Total time 2s  Avg loss 0.8342 Avg Prec@1 73.34 %      Avg Prec@5 98.06 %
```




```
=================
=== EPOCH 32 =====
=================

===============> Total time 13s Avg loss 0.1603 Avg Prec@1 95.02 %      Avg Prec@5 99.95 %

[EVAL Batch 000/079]    Time 0.137s (0.137s)    Loss 1.0817 (1.0817)    Prec@1  77.3 ( 77.3)    Prec@5  98.4 ( 98.4)

===============> Total time 2s  Avg loss 1.3831 Avg Prec@1 68.31 %      Avg Prec@5 97.94 %
```




```
=================
=== EPOCH 44 =====
=================
```

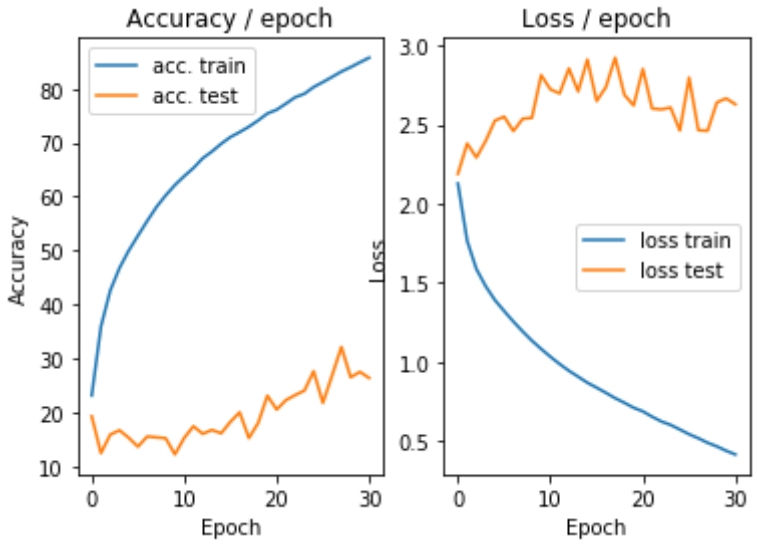
Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°5	0.010	128	61	89.45 %	0.315	41	65.67 %	1.0074
n°7	0.010	128	43	95.02 %	0.160	31	73.34 %	0.8342

On constate 3 avantage de normalisation par rapport au modèle entraîné avec les données non-normalisées.

1. Le temps nécessaire pour obtenir le meilleur modèle est réduit (41 époque => 31 époque).
2. La précision de test augmente de 7,67 % (65.67 % => 73.34 %).
3. L'erreur de test diminue de 0.1732 (1.0074 => 0.8342).

20. Pourquoi ne calculer l'image moyenne que sur les exemples d'apprentissage et normaliser les exemples de validation avec la même image ?

Normaliser les exemples de validations avec les mêmes valeurs est important pour conserver l'échelle. C'est-à-dire, normaliser de la même manière donne la cohérence des caractéristiques dans les images. Sans normaliser, il peut y avoir des écarts sérieux entre l'échelle des images qui ne permettent pas d'entraîner correctement le modèle.
Training n°8 : Learning rate η ***= 0.01*** / la taille de mini-batch ***= 128 /*** *normalization appliquée (seulement au train)* μ *= [0.491, 0.482, 0.447] et* σ *= [0.202, 0.199, 0.201]*



Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°8	0.010	128	35	89.39 %	0.0314	1	19.29 %	2.1889

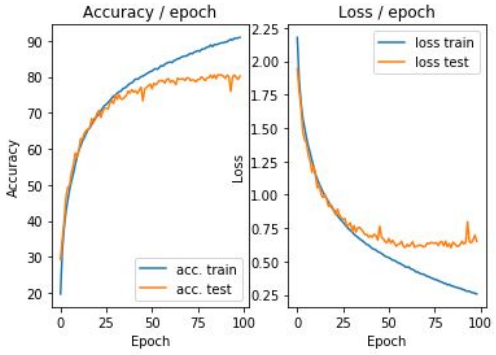
On ne constate que le sur-apprentissage sans normalisation.

21. Bonus : Il existe d'autres schémas de normalisation qui peuvent être plus efficaces comme la normalisation ZCA. Essayer d'autres méthodes, expliquer les différences et comparer les à celle demandée.

Dans la partie suivante du TP, on souhaite augmenter le nombre de données d'entraînement de notre système en appliquant des transformations aux images déjà présentes dans le jeu de données.

22. Décrire vos résultats expérimentaux et les comparer aux résultats précédents.
Training n°9 : Learning rate η ***= 0.01*** / la taille de mini-batch ***= 128 /*** *normalization appliquée (seulement au train)* μ *= [0.491, 0.482, 0.447] et* σ *= [0.202, 0.199, 0.201] / crop aléatoire (28 x 28) pour train / crop centré (28 x 28) pour test / symétrie horizontale aléatoire*

```
===============> Total time 16s Avg loss 0.2566 Avg Prec@1 90.94 %      Avg Prec@5 99.84 %
[EVAL Batch 000/079]    Time 0.189s (0.189s)    Loss 0.6307 (0.6307)    Prec@1  81.2 ( 81.2)    Prec@5  96.9 ( 96.9)
===============> Total time 2s  Avg loss 0.6501 Avg Prec@1 80.20 %      Avg Prec@5 98.70 %
```




```
================
=== EPOCH 100 =====
================
```

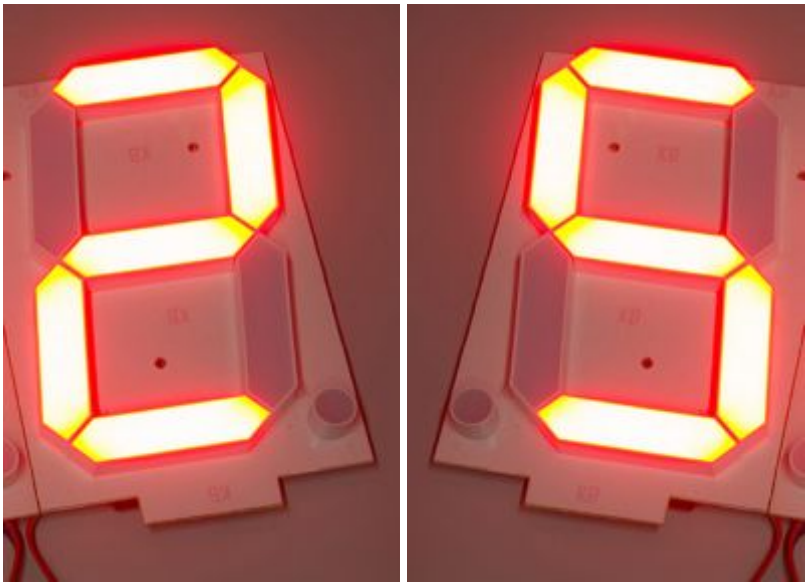
Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°9	0.010	128	100	90.94 %	0.2566	85	80.52 %	0.6032

La précision de test augmente à 80.52 % et l'erreur diminue à 0.6032. Le sur-apprentissage et le phénomène de "overshoot" sont moins évidents par rapport aux résultats précédents.

23. Est-ce que cette approche par symétrie horizontale vous semble utilisable sur tout type d'images ? Dans quels cas peut-elle l'être ou ne pas l'être ?

Non cette approche par symétrie horizontale ne semble pas utilisable sur tout type d'images. Pour des images d'animaux, de plantes, des fruits ou d'humains par exemple on en voit l'intérêt car des images semblables peuvent effectivement exister de façon symétrique à l'image originale. Cependant, pour des images de livres ou possédant des écritures par exemple, il n'y a pas d'intérêt.

Lorsque l'orientation de l'image est importante. L'image originale à gauche ci-dessous signifie 2 mais elle signifie donc 5 (à droite) lorsqu'elle est inversée. Donc, la symétrie horizontale risque de mener à la classification mauvaise lorsque l'orientation de l'image est non-négligeable.



(Source : https://learn.sparkfun.com/tutorials/large-digit-driver-hookup-guide)

En revanche, si l'orientation de l'image ne change pas de signification de l'image, alors cette approche est utilisable.

24. Quelles limites voyez-vous à ce type de data augmentation par transformation du dataset ?

Il faut assurer que cette méthode conserve l'intégralité des informations dans une image. Sinon, l'augmentation des données peut mal optimiser le modèle à cause d'un mauvais apprentissage.

Une des limites est le fait qu'il faut que les images sélectionnées pour le jeu de données initial soient suffisamment précises. En effet, il faut que ces images correspondent au contexte d'application pour que les transformations appliquées soient utiles à l'apprentissage.

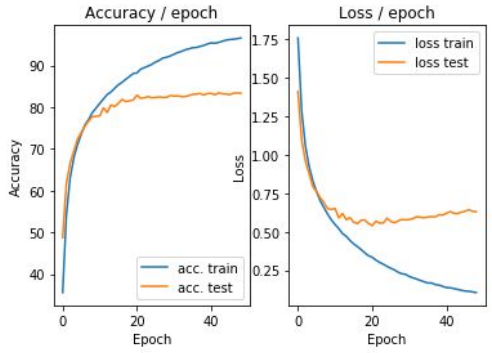
25. Bonus : D'autres méthodes de data augmentation sont possibles. Chercher lesquelles et en tester certaines

Il existe d'autres méthodes de data augmentation que la symétrie horizontale et la réduction d'image (crop).
On peut modifier les images en appliquant une translation ou une rotation. On peut compléter des pixels manquants après une rotation d'image en appliquant du padding. On peut également changer la luminosité, le contraste ou le bruit des images.
Toutes ces transformations permettent d'augmenter la base de données de test.

26. Décrire vos résultats expérimentaux et les comparer aux résultats précédents, notamment la stabilité de l'apprentissage.

Training n°10 : Learning rate η ***= 0.01*** / la taille de mini-batch ***= 128 /*** *normalization appliquée (seulement au train)* μ *= [0.491, 0.482, 0.447] et* σ *= [0.202, 0.199, 0.201] / crop aléatoire (28 x 28) pour train / crop centré (28 x 28) pour test / symétrie horizontale aléatoire / SDG Momentum =* ***0.9 /*** *Exponential Learning Rate Scheduler* γ **= 0.95**

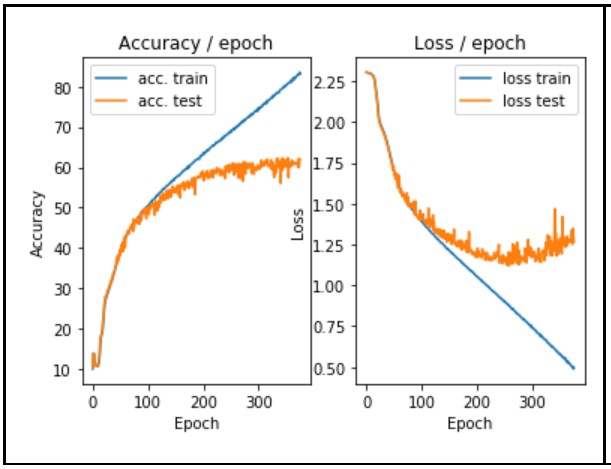
```
===============> Total time 16s Avg loss 0.1055 Avg Prec@1 96.56 %      Avg Prec@5 99.98 %

[EVAL Batch 000/079]    Time 0.188s (0.188s)    Loss 0.5673 (0.5673)    Prec@1  86.7 ( 86.7)    Prec@5  98.4 ( 98.4)

===============> Total time 2s  Avg loss 0.6300 Avg Prec@1 83.34 %      Avg Prec@5 98.95 %
```

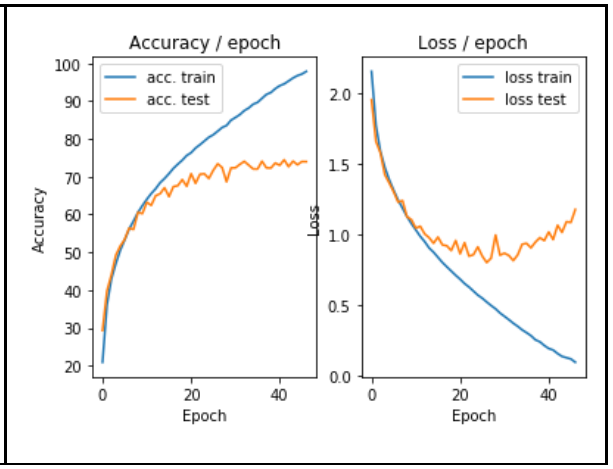


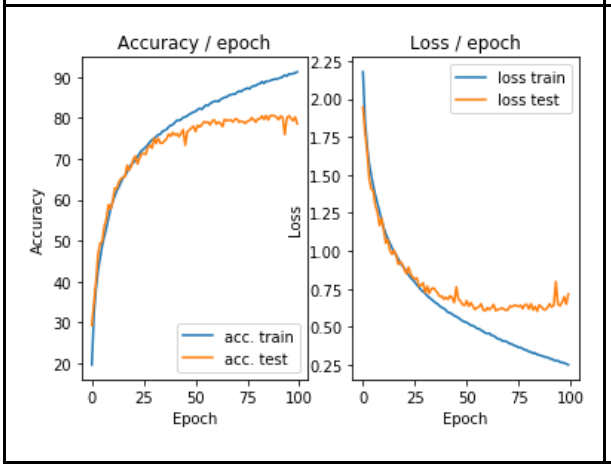
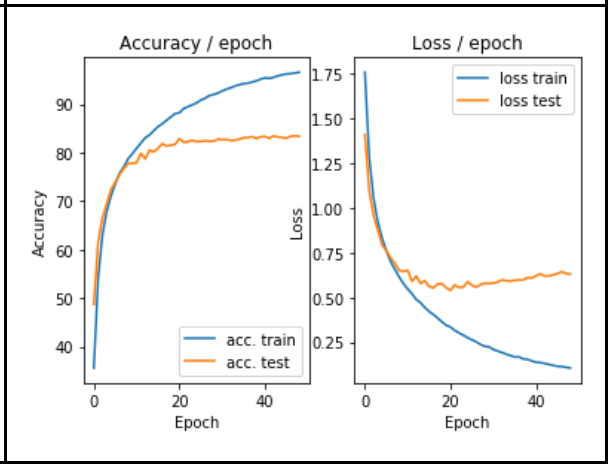

```
=================
=== EPOCH 50 =====
=================
```

Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°10	0.010	128	50	96.56 %	0.1055	21	82.87 %	0.5392

Sans prétraitement (n°6)	+ **Normalisation (n°7)**





+ Augmentation des données (n°9)	+ LR Schedule / Momentum (n°10)
	

L'évolution de la précision est très instable sans prétraitement au début. Elle devient un peu plus stable avec la normalisation. C'est-à-dire, l'oscillation est moins fréquent jusqu'à l'optimisation du modèle. L'augmentation des données n'a pas d'effet significatif sur la stabilité mais l'ajout d'un momentum et d'un "learning rate scheduler" améliore beaucoup la stabilité. Les traces de précisions et erreurs est quasi lisses.

Les résultats sont améliorés à chaque entraînement (sauf pour le ***n°8***)

27. Pourquoi ces deux méthodes améliorent l'apprentissage ?

Ne pas totalement oublier la direction du gradient peut améliorer la stabilité. Le SGD mini-batch est un compromis entre la version classique et la version online et garde la caractéristique de la version online. C'est-à-dire, le gradient peut se diriger vers une direction "noisy". Le momentum sert à fixer la direction du gradient jusqu'à un certain point.

Learning rate scheduler implémenté dans notre modèle diminue le taux d'apprentissage à chaque époque. Au début, on varie largement les poids pour que le modèle s'entraîne vite et puis on fait un réglage de l'optimisation lorsque le modèle est plus ou moins optimisé. A ce point, avoir un petit taux d'apprentissage varie finement les poids pour le modèle soit mieux

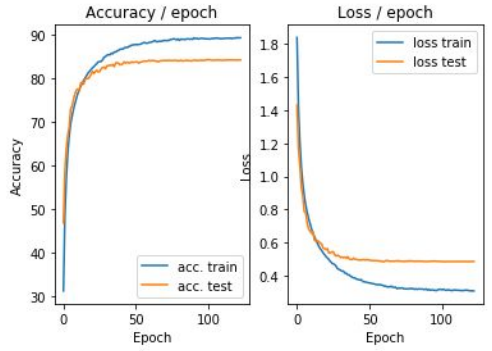
optimiser sans avoir “overshoot”. En effet, on observe que la précision de test reste plat dès que le modèle est optimisé dans le ***training n°10***.

28. Bonus : De nombreuses autres variantes de la SGD existent et de nombreuses stratégies de planification de learning rate existent. Lesquelles ? En tester certaines.

Comme variante de la SGD, on peut citer l’algorithme RMSprop. Cet algorithme a pour objectif de régler le problème des tailles de gradients qui diffèrent et qui rend difficile de trouver un learning rate global.
On peut aussi parler de l’algorithme Adam qui est une combinaison des algorithmes SGD et RMSprop.

29. Décrire vos résultats expérimentaux et les comparer aux résultats précédents.
Training n°11 : Learning rate η ***= 0.01*** / la taille de mini-batch ***= 128 /*** *normalization appliquée (seulement au train)* μ *= [0.491, 0.482, 0.447] et* σ *= [0.202, 0.199, 0.201] / crop aléatoire (28 x 28) pour train / crop centré (28 x 28) pour test / symétrie horizontale aléatoire / SDG Momentum =* ***0.9 /*** *Exponential Learning Rate Scheduler* γ **= 0.95** / Dropout = 80 %

```
===============> Total time 14s Avg loss 0.3073 Avg Prec@1 89.25 %      Avg Prec@5 99.70 %
[EVAL Batch 000/079]    Time 0.162s (0.162s)    Loss 0.3702 (0.3702)    Prec@1  87.5 ( 87.5)    Prec@5  98.4 ( 98.4)
===============> Total time 2s  Avg loss 0.4855 Avg Prec@1 84.15 %      Avg Prec@5 99.27 %
```




```
=================
=== EPOCH 124 =====
=================
```

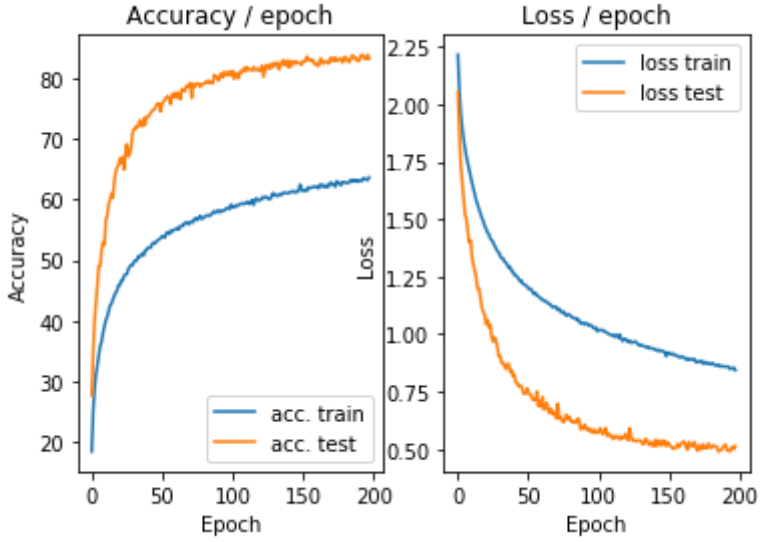
Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°11	0.010	128	123	89.25 %	0.3073	117	84.17 %	0.4849

Il y a moins d’écart entre les valeurs de précisions et d’erreurs en train et en test par rapport aux résultats précédents. Ces valeurs varient très peu dès qu’il est optimisé. Surtout, le tracé

d’erreurs reste plat. On peut remarquer que l’erreur commence à remonter et ne reste plus plat après l’optimisation dans le ***training n°10*** mais on peut constater que l’ajout d’un dropout peut résoudre ce problème.
On a obtenu la meilleure valeur de précision et loss. Alors que il y a peu de variation au niveau de la précision dès la 50ème époque, cela prend du temps pour obtenir un modèle légèrement plus optimisé

30. Qu’est-ce que la régularisation de manière générale ?
La régularisation est une technique visant à limiter le sur-apprentissage. Elle consiste à ajouter une contrainte lors de l’apprentissage pour simplifier le modèle et améliorer la généralisation de la solution. Dans le cadre du TP la couche dropout a pour but de diminuer le nombre de paramètres.

31. Cherchez et "discutez" des possibles interprétations de l’effet du dropout sur le comportement d’un réseau l’utilisant ?
Le dropout met aléatoirement des éléments de l’entrée à zéro. Donc, l’entrée devient bruitée. Alors que cela peut empêcher le modèle d’être entraîné correctement, cela peut généraliser les exemples de training. Donc, le dropout est une technique de régularisation.



(Une autre couche de dropout est ajoutée entre la FC5 et la couche de softmax pour mieux visualiser l’effet du dropout)
On constate que les valeurs de précisions de train restent très inférieures aux résultats précédents et aux valeurs de précision de test. De même, les valeurs d’erreurs ont la même tendance.
On voit bien que l’effet du dropout limite le sur-apprentissage.

32. Quelle est l’influence de l’hyper-paramètre de cette couche ?
L’hyper-paramètre, ***p*** en pytorch, est un taux de dropout. Cela détermine combien d’éléments doivent être mise à zéro.

33. Quelle est la différence de comportement de la couche de dropout entre l'apprentissage et l'évaluation ?

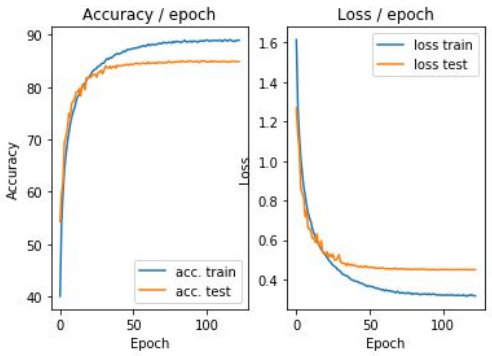
La couche de dropout est désactivé lors de l'évaluation.

34. Décrire vos résultats expérimentaux et les comparer aux résultats précédents.

Training n°12 : Learning rate η = ***0.01*** / la taille de mini-batch = ***128 /*** *normalization appliquée (seulement au train)* μ *= [0.491, 0.482, 0.447] et* σ *= [0.202, 0.199, 0.201] / crop aléatoire (28 x 28) pour train / crop centré (28 x 28) pour test / symétrie horizontale aléatoire / SDG Momentum =* ***0.9 /*** *Exponential Learning Rate Scheduler* γ = **0.95** / Dropout = 80 % / Batch normalization

```
self.features = nn.Sequential(
    nn.Conv2d(3, 32, (5, 5), stride=1, padding=2),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
    nn.Conv2d(32, 64, (5, 5), stride=1, padding=2),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0),
    nn.Conv2d(64, 64, (5, 5), stride=1, padding=2),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d((2, 2), stride=2, padding=0, ceil_mode=True),
)
```

```
===============> Total time 14s Avg loss 0.3167 Avg Prec@1 88.94 %      Avg Prec@5 99.66 %

[EVAL Batch 000/079]    Time 0.171s (0.171s)    Loss 0.4151 (0.4151)    Prec@1  86.7 ( 86.7)    Prec@5  99.2 ( 99.2)

===============> Total time 2s  Avg loss 0.4497 Avg Prec@1 84.89 %      Avg Prec@5 99.32 %
```




```
==================
=== EPOCH 124 =====
==================
```

Training	Learning rate η	mini-batch size	Sampled epoch (train)	Average Accuray @1 (train)	Average Loss (train)	Best Epoch (test)	Average Accuray @1 (test)	Average Loss (test)
n°11	0.010	128	123	89.25 %	0.3073	117	84.17 %	0.4849
n°12	0.010	128	123	88.94 %	0.3167	91	84.86 %	0.4488

Le training n°12 donne un légèrement meilleur résultat que les résultats précédent. La précision de test augmente et l'erreur de test diminue. Surtout, on a pu réduire le temps nécessaire pour pouvoir optimiser le modèle.