Documents multimédia
Description et recherche automatique

# Système de classification d'images par apprentissage profond

## Compte rendu

**18 Avril 2021**

Chouaib Mounaime
M2GI Classique

# PLAN

# 1. Introduction

L'objectif de ce projet est de mettre en œuvre les principes d'apprentissage profond (deep learning), et plus précisément les réseaux neuronaux convolutifs,  dans le but de concevoir un système de classification d'images.

Pour la réalisation de ce projet, nous utiliserons **Pytorch** et le dataset **Cifar-10**. Nous utiliserons également un notebook **google Colab** qui propose des noeuds GPU (une alternative au Grid'5000), utiles pour la phase d'entraînement de notre réseau.

Durant ce projet, nous aborderons différents aspects concernant les réseaux CNN et nous réaliserons diverses expérimentations dans le but de maximiser les performances de notre r.

Dans un premier temps, nous nous baserons sur le modèle proposé dans le tutoriel pytorch pour réaliser et comparer 3 variantes :

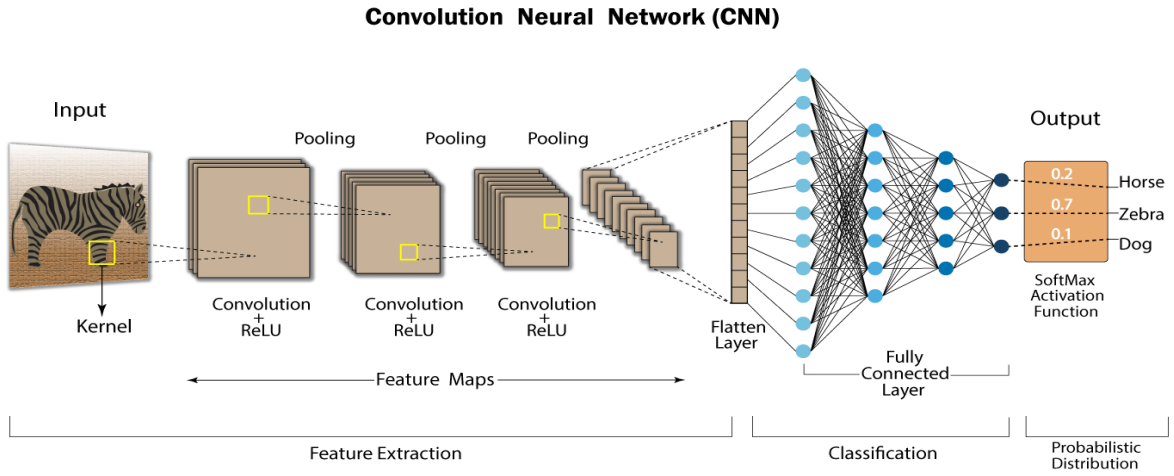
- **un modèle wide** : peu de couches mais beaucoup de filtres.
- **un modèle deep** : peu de filtres mais beaucoup de couches.
- **une combinaison des deux.**

Dans un second temps, nous nous focaliserons sur un des 3 modèles présentés et on essayera d'améliorer ses performances grâce à une série d'expérimentations avancées (**Batch normalization, data augmentation, dropout, batch size** ...).

**NB :** La largeur et/ou profondeur des modèles qui seront étudiés restent tout de même raisonnables et relatives à la taille limitée du modèle (1MB) afin de respecter les contraintes du projet :  nombre de paramètres (maximum 256k) et temps d'entraînement (maximum 1H).

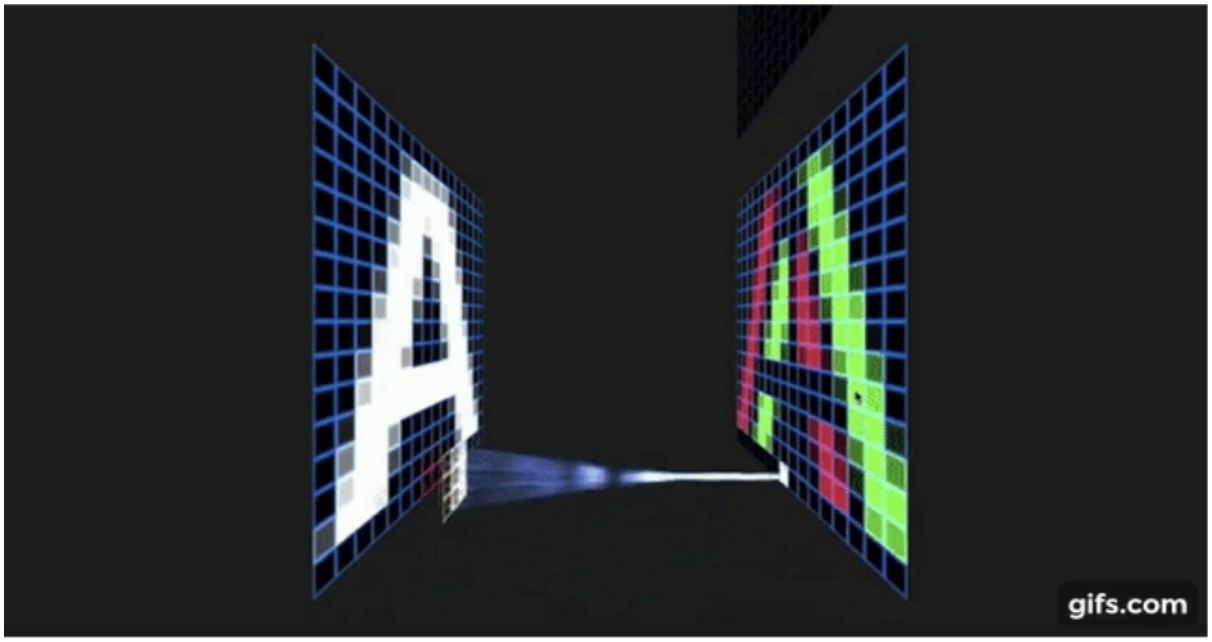
## 2. Petit rappel : un CNN, c'est quoi ?

Un réseau neuronal convolutif (CNN) est une sous-catégorie des réseaux de neurones, conçu spécifiquement pour le traitement et classification d'images. L'architecture d'un modèle CNN se compose de 2 parties principales : une partie de convolution et une partie de classification qui correspond à un réseau  MLP (multi layer perceptron).



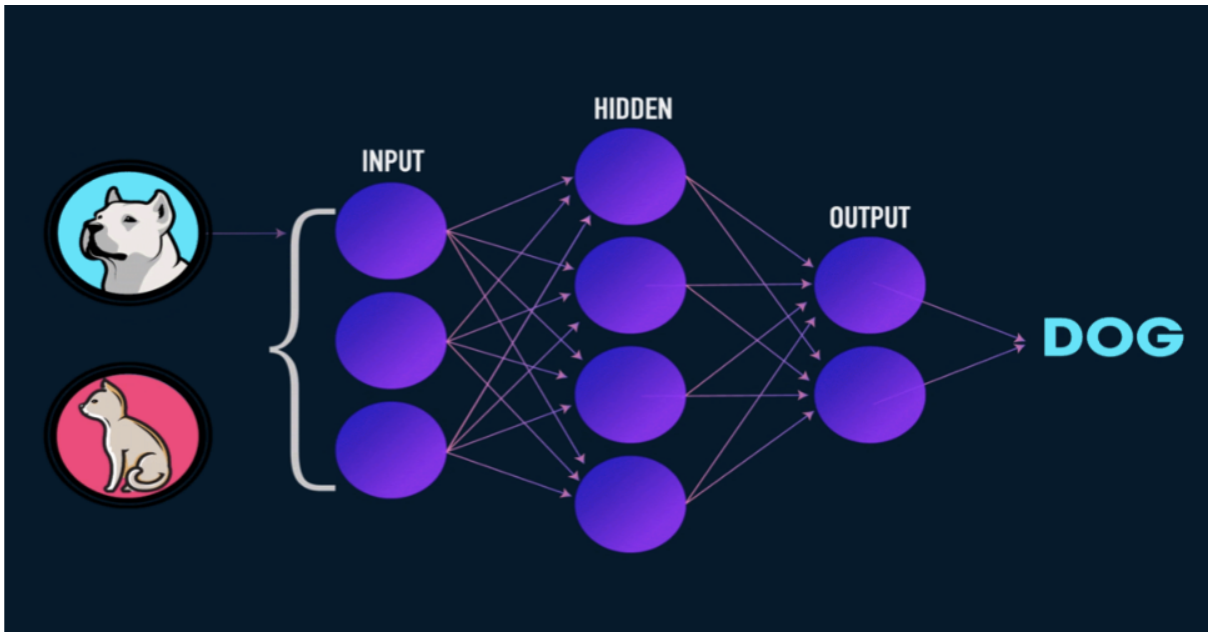

### 2.1. Partie de convolutions

L'objectif de cette partie est de repérer et d'extraire les descripteurs (features) d'une image passée en entrée du modèle en y appliquant une succession de filtres (kernels).



## 2.2 Partie de classification

Le résultat de la partie de convolution est fourni en entrée (sous forme d'un vecteur à une dimension) à un modèle constitué de couches entièrement connectées qui appliquent successivement des opérations linéaires et d'activation dans le bus de classer l'image. La sortie correspond à un vecteur dans lequel chaque élément représente la probabilité d'appartenance de l'image a une classe.



# 3. Etude préliminaire

## 3.1 Modèle 1 : réseau large

Ce modèle est caractérisé par ses couches de convolution (3 couches) et ses 2 couches fully connected, et comptabilise un nombre total de paramètres entraînables de 245.546.
Le nombre d'opérations **MACs (multiply-accumulate opérations)** est estimé à plus de 13 millions d'opérations.
L'avantage de ce modèle est que chacune des couches de convolution possède un nombre important de **kernels** (48-64). Ces filtres sont de taille 3x3 avec un **stride** de 1 et **padding** de 1.
Un opération de max **pooling** 2x2 est réalisée après chaque couche de convolution.

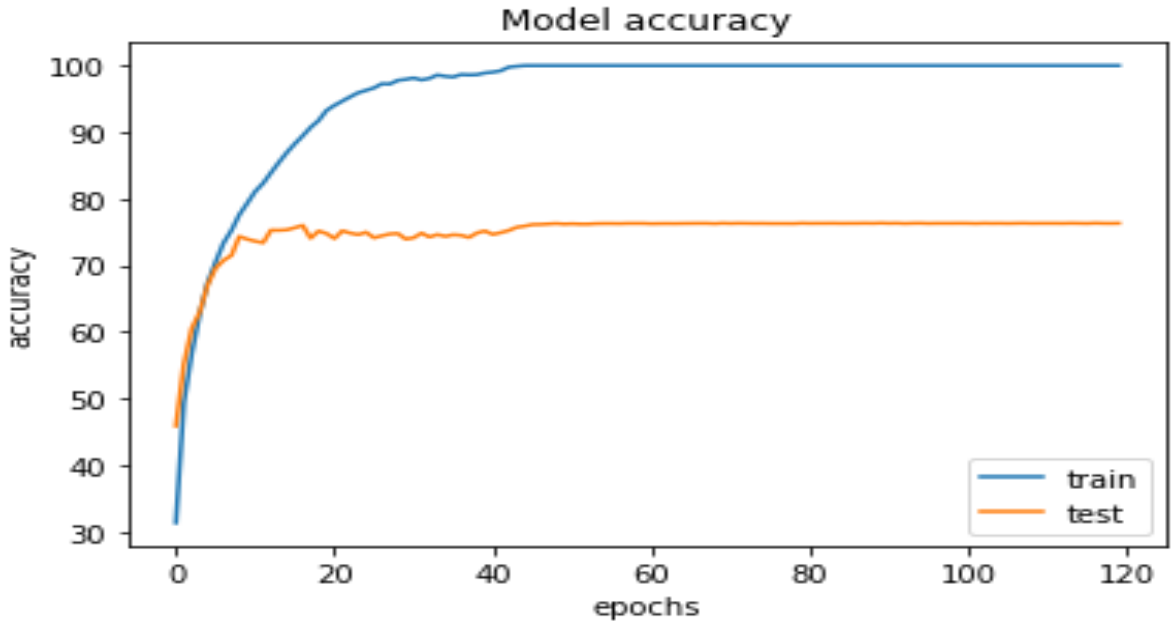
```
Net(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=768, out_features=200, bias=True)
  (fc2): Linear(in_features=200, out_features=10, bias=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

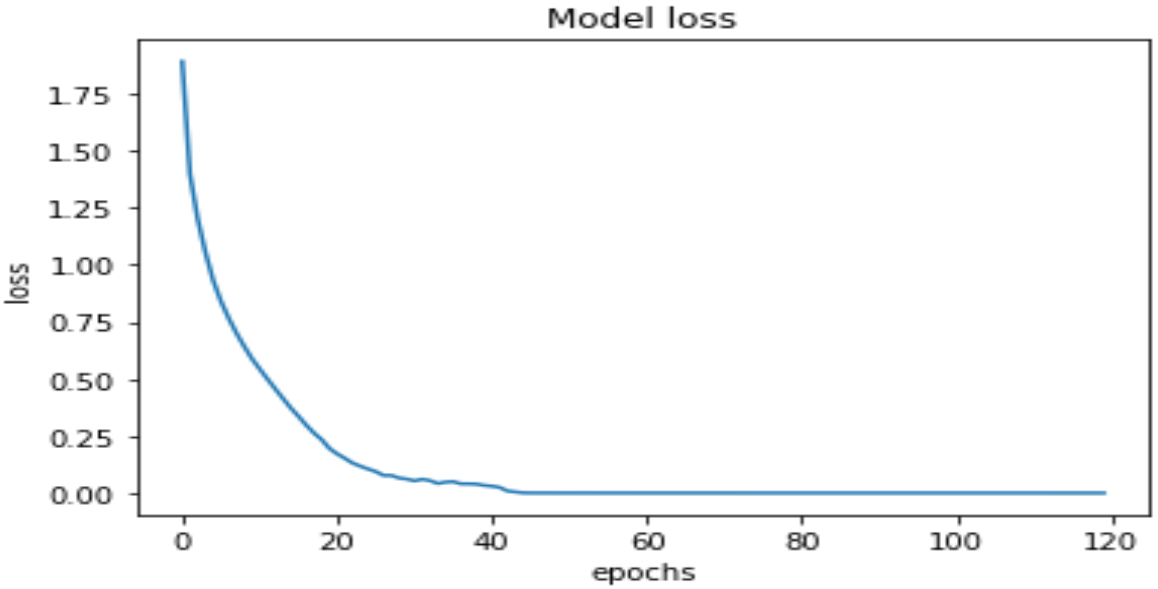
Ci dessous la description complète de  l'architecture du réseau  avec toutes ses couches, formes des kernels, tailles des tenseurs d'entrée/sortie ainsi que le nombre de paramètres  et de MACs de chaque couche.
A noter qu'une fonction d'action (non visible dans la description) est appliquée après chaque convolution ou couche linéaire.

```
==========================================================================
           Kernel Shape     Output Shape   Params  Mult-Adds
Layer
0_conv1   [3, 64, 3, 3]  [1, 64, 32, 32]   1.792k  1.769472M
1_pool                -  [1, 64, 16, 16]        -          -
2_conv2  [64, 64, 3, 3]  [1, 64, 16, 16]  36.928k  9.437184M
3_pool                -    [1, 64, 8, 8]        -          -
4_conv3  [64, 48, 3, 3]    [1, 48, 8, 8]  27.696k  1.769472M
5_pool                -    [1, 48, 4, 4]        -          -
6_fc1        [768, 200]          [1, 200]   153.8k     153.6k
7_fc2         [200, 10]           [1, 10]    2.01k       2.0k
--------------------------------------------------------------------------
                         Totals
Total params           222.226k
Trainable params       222.226k
Non-trainable params        0.0
Mult-Adds            13.131728M
==========================================================================
```

Après 120 époques d'entraînement (41min), la précision sur le trainset enregistre un pourcentage de 100%, tandis que sur le testset elle plafonne autour des 56%.

```
[epoch 115] train accuracy: 100%  test accuracy: 76%  loss: 0.000  duration: 20.65
[epoch 116] train accuracy: 100%  test accuracy: 76%  loss: 0.000  duration: 20.79
[epoch 117] train accuracy: 100%  test accuracy: 76%  loss: 0.000  duration: 19.65
[epoch 118] train accuracy: 100%  test accuracy: 76%  loss: 0.000  duration: 20.59
[epoch 119] train accuracy: 100%  test accuracy: 76%  loss: 0.000  duration: 20.65
[epoch 120] train accuracy: 100%  test accuracy: 76%  loss: 0.000  duration: 20.49

Training finished in 00h 40min 53sec
```



En comparant ces résultats des graphiques ci-dessus, on remarque donc que notre modèle tombe rapidement (après 20 époques seulement) dans un cas **d'overfitting** (**sur-apprentissage**) : il s'agit du fait que le modèle s'adapte aux données d'entraînement ce qui le rend bon en mémorisation mais très mauvais en généralisation.

On constate également qu' au bout de 40 époques, la fonction de perte converge vers 0, c'est a dire que notre modèle réussi à prédire parfaitement les données du trainset, ce qui explique la stagnation des précisions.

Dans la seconde partie de cette étude (voir section 4), nous présenterons quelques techniques qui permettent d'éviter l'overfitting/underfitting sur les données de test ainsi que l'amélioration des performances.

## 3.2 Modèle 2 : réseau profond

Ce modèle est caractérisé par ses multiples couches de convolution (6 couches conv. ) organisées sous forme de 3 blocs de 2 convolutions par bloc + 1 pooling, et ses 3 couches fully connected, et comptabilise un nombre total de paramètres entraînables de 127 724. et un nombre de **MACs** de plus de 2.5 millions.

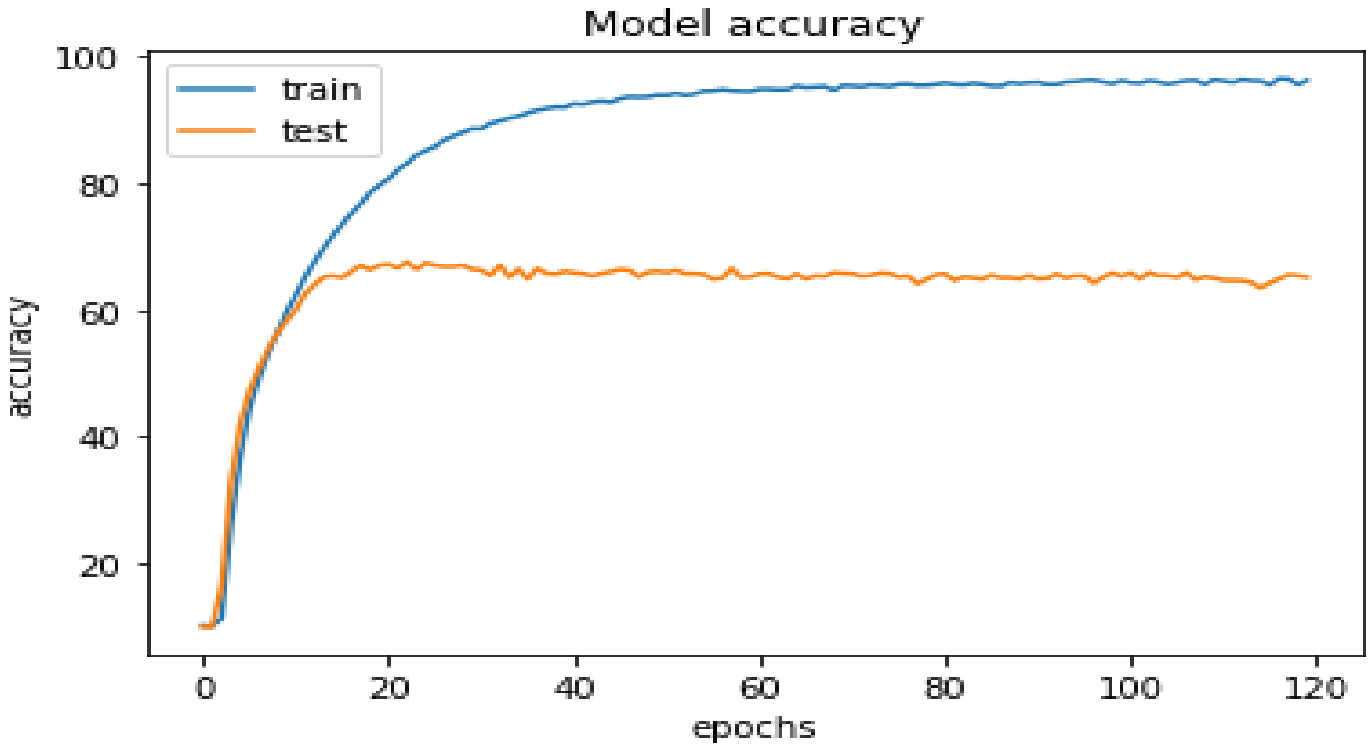
Le nombre de filtres de chacune de ces couches convolutionnelles est de 8, 16 ou 32 filtres de tailles 3x3 avec un **stride** de 1 et **padding** de 1.
Un opération de **max pooling** 2x2 est réalisée après la deuxième, la 4eme et la dernière couche conv.

```
Net(
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv6): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=512, out_features=180, bias=True)
  (fc2): Linear(in_features=180, out_features=90, bias=True)
  (fc3): Linear(in_features=90, out_features=10, bias=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

```
==========================================================
           Kernel Shape     Output Shape  Params Mult-Adds
Layer
0_conv1     [3, 8, 3, 3]   [1, 8, 32, 32]   224.0  221.184k
1_conv2     [8, 8, 3, 3]   [1, 8, 32, 32]   584.0  589.824k
2_pool                 -   [1, 8, 16, 16]       -         -
3_conv3    [8, 16, 3, 3]  [1, 16, 16, 16]  1.168k  294.912k
4_conv4   [16, 16, 3, 3]  [1, 16, 16, 16]   2.32k  589.824k
5_pool                 -    [1, 16, 8, 8]       -         -
6_conv5   [16, 32, 3, 3]    [1, 32, 8, 8]   4.64k  294.912k
7_conv6   [32, 32, 3, 3]    [1, 32, 8, 8]  9.248k  589.824k
8_pool                 -    [1, 32, 4, 4]       -         -
9_fc1         [512, 180]          [1, 180]  92.34k    92.16k
10_fc2         [180, 90]           [1, 90]  16.29k     16.2k
11_fc3          [90, 10]           [1, 10]   910.0     900.0
----------------------------------------------------------
                       Totals
Total params          127.724k
Trainable params      127.724k
Non-trainable params       0.0
Mult-Adds             2.68974M
==========================================================
```

Après 120 époques d'entraînement (48min), la précision sur le trainset enregistre un pourcentage de 96%, tandis que sur le testset elle dépasse à peine 56%.

```
[epoch 115] train accuracy: 96%  test accuracy: 64%  loss: 0.117  duration: 22.80
[epoch 116] train accuracy: 96%  test accuracy: 64%  loss: 0.143  duration: 22.65
[epoch 117] train accuracy: 97%  test accuracy: 65%  loss: 0.107  duration: 22.96
[epoch 118] train accuracy: 96%  test accuracy: 66%  loss: 0.114  duration: 22.63
[epoch 119] train accuracy: 96%  test accuracy: 65%  loss: 0.139  duration: 23.00
[epoch 120] train accuracy: 96%  test accuracy: 65%  loss: 0.118  duration: 22.99

Training finished in 00h 47min 02sec
```




De la même manière que pour le modèle, ce modèle sur-apprend (overfit) les données de test en 20 époques seulement.
Quant aux performances, ce modèle enregistre une précision médiocre comparée à celle du modèle 1 (65% contre 76% sur le testset).

### 3.3 Modèle 3 : réseau large-profond

Pour ce modèle, on essayera de combiner les 2 approches précédentes, afin de tirer les avantages de chacune d'elles.
On reprendra donc l'architecture du modèle 2, composée de 6 couches de convolution, organisées sous forme de 3 blocs de 2 convolutions + 1 pooling, et 3 couches fully connected.

Il comptabilise un nombre total de paramètres entraînables de 253 638 et un nombre de **MACs** de plus de 12 millions.
Le nombre de filtres de chacune de ces couches convolutionnelles est de 16 ou 32 filtres de tailles 3x3 avec un **stride** de 1 et **padding** de 1.
Un opération de **max pooling** 2x2 est réalisée après la deuxième, la 4eme et la dernière couche conv.

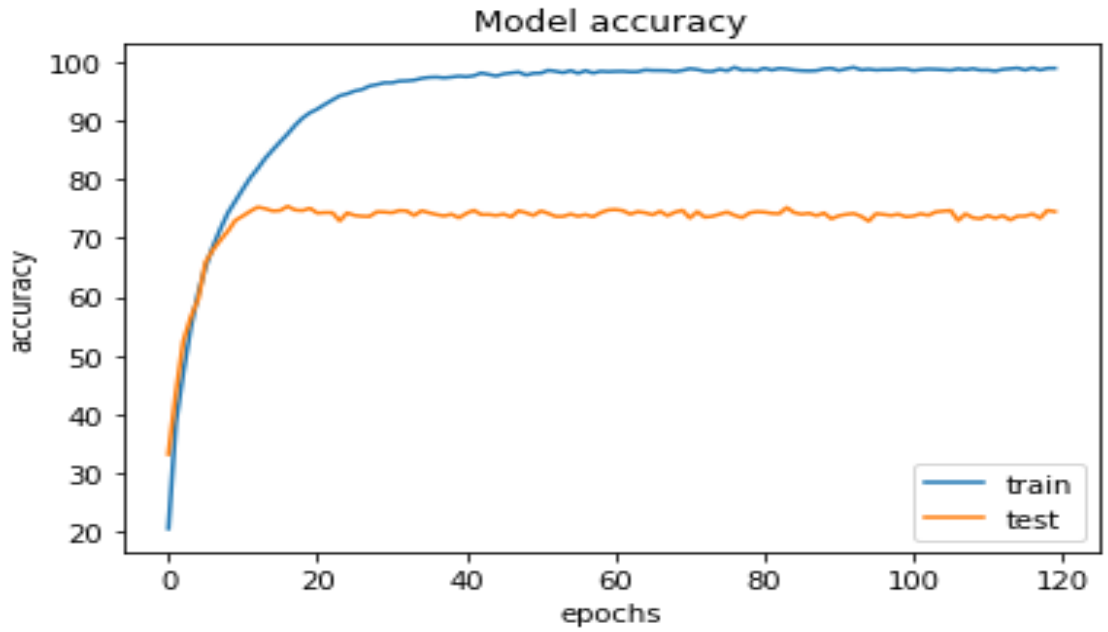
```
Net(
  (conv1): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(24, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(32, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv6): Conv2d(48, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=768, out_features=220, bias=True)
  (fc2): Linear(in_features=220, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

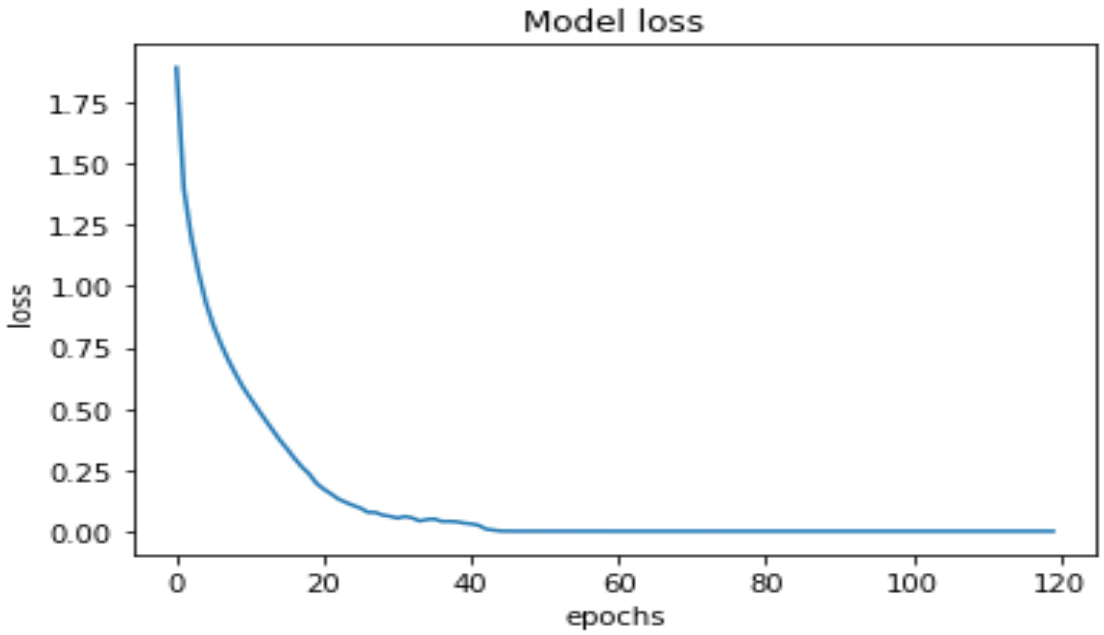
| Layer | Kernel Shape | Output Shape | Params | Mult-Adds |
|---|---|---|---|---|
| 0_conv1 | [3, 24, 3, 3] | [1, 24, 32, 32] | 672.0 | 663.552k |
| 1_conv2 | [24, 24, 3, 3] | [1, 24, 32, 32] | 5.208k | 5.308416M |
| 2_pool | - | [1, 24, 16, 16] | - | - |
| 3_conv3 | [24, 32, 3, 3] | [1, 32, 16, 16] | 6.944k | 1.769472M |
| 4_conv4 | [32, 32, 3, 3] | [1, 32, 16, 16] | 9.248k | 2.359296M |
| 5_pool | - | [1, 32, 8, 8] | - | - |
| 6_conv5 | [32, 48, 3, 3] | [1, 48, 8, 8] | 13.872k | 884.736k |
| 7_conv6 | [48, 48, 3, 3] | [1, 48, 8, 8] | 20.784k | 1.327104M |
| 8_pool | - | [1, 48, 4, 4] | - | - |
| 9_fc1 | [768, 220] | [1, 220] | 169.18k | 168.96k |
| 10_fc2 | [220, 120] | [1, 120] | 26.52k | 26.4k |
| 11_fc3 | [120, 10] | [1, 10] | 1.21k | 1.2k |

| Totals | |
|---|---|
| Total params | 253.638k |
| Trainable params | 253.638k |
| Non-trainable params | 0.0 |
| Mult-Adds | 12.509136M |

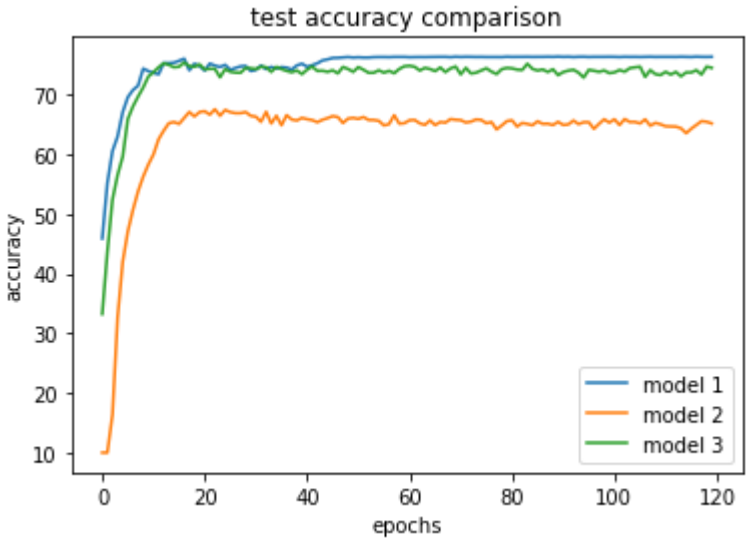
```
[epoch 115] train accuracy: 99%  test accuracy: 74%  loss: 0.040  duration: 22.54
[epoch 116] train accuracy: 99%  test accuracy: 74%  loss: 0.051  duration: 22.55
[epoch 117] train accuracy: 99%  test accuracy: 74%  loss: 0.036  duration: 22.27
[epoch 118] train accuracy: 99%  test accuracy: 73%  loss: 0.050  duration: 22.09
[epoch 119] train accuracy: 99%  test accuracy: 75%  loss: 0.037  duration: 21.95
[epoch 120] train accuracy: 99%  test accuracy: 74%  loss: 0.038  duration: 22.78

Training finished in 00h 45min 00sec
```

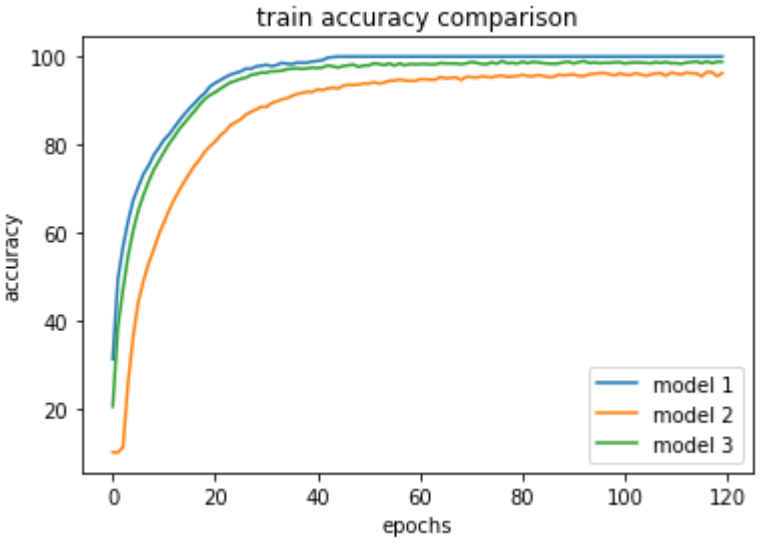
Comme pour les 2 modèles précédents, la perte converge vers 0 au bout d'environ 40 époques, tandis qu'il suffit de 20 époques pour que l'overfitting du testset se forme.




## 3.4. Comparaison des performances



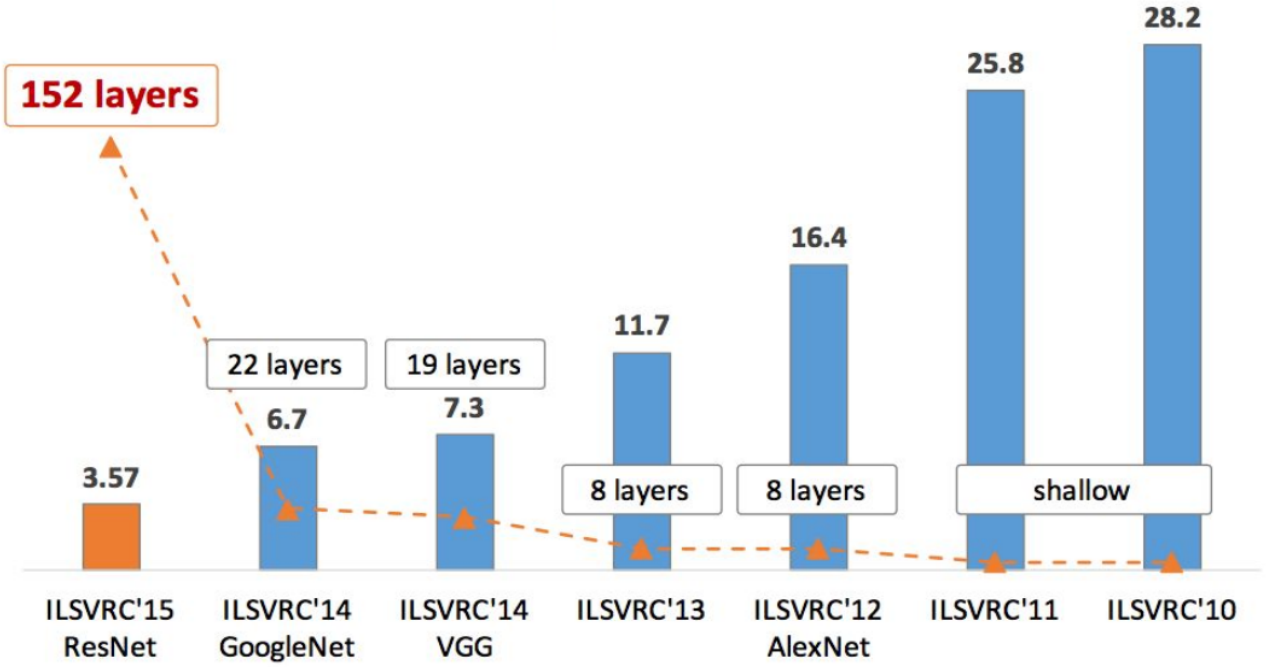




En comparant les performances des 3 réseaux, on constate que le 1er et 3eme garantissent des performances assez similaires tandis que le 2eme est un peu moins performant.
Notons cependant que le 2eme modèle réalise 4 fois moins d'opérations, et possède beaucoup moins de paramètres.

| | # params | # MACs |
|---|---|---|
| **Modèle 1** | 222K | 13.1M |
| **Modèle 2** | 127K | 2.6M |
| **Modèle 3** | 253K | 12.5M |

## 4. Etude avancée

Après avoir étudié 3 approches différentes d'architecture de CNN, nous nous focaliserons à présent sur un des 3 modèles présentés dans le but d'améliorer ses performances et de remédier à l'overfitting.




Comme le nombre de couches a significativement progressé au fil des années et au vu du succès des réseaux profonds, on s'intéressera donc dans cette partie à notre 3eme réseau (voir modèle 3) qui a la particularité d'être à la fois large et profond.

Dans cette partie on abordera les techniques les plus populaires d'amélioration de réseaux neuronaux convolutifs tels que le **data augmentation**, **batch normalization** ou encore le **dropout.**

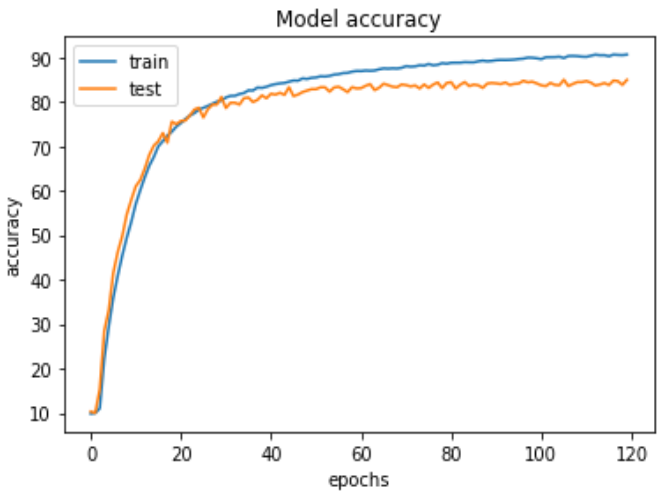
On essayera également de trouver les valeurs optimales des hyperparamètres liés à l'algorithme d'entraînement tels que la taille des batchs **(batch size).**

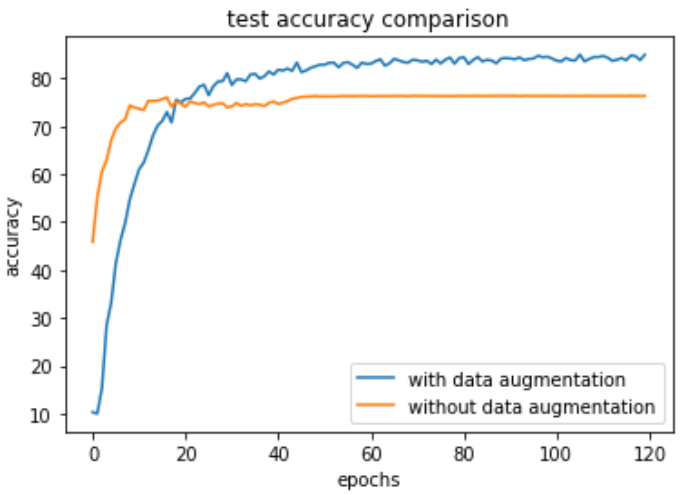
## 4.1. Data augmentation

Cette technique consiste à augmenter la quantité de données d'entraînement en appliquant des transformations sur les images (flip, crop, zoom shift ...) afin de permettre au modèle d'extraire et d'apprendre davantage des features à partir des nouvelles images générées, et ainsi augmenter ses capacités de généralisation.

Après avoir entraîné notre réseau pendant 120 époques (54min), on constate une amélioration significative des performances.
En effet, grâce à l'augmentation de données, on passe de **74%** (voir section 3.3) à de **85%** sur les données de test.
On constate également une réduction importante de l'overfitting.

```
[epoch 117] train accuracy: 91%  test accuracy: 85%  loss: 0.265  duration: 27.22
[epoch 118] train accuracy: 91%  test accuracy: 85%  loss: 0.267  duration: 27.29
[epoch 119] train accuracy: 90%  test accuracy: 84%  loss: 0.263  duration: 27.20
[epoch 120] train accuracy: 91%  test accuracy: 85%  loss: 0.265  duration: 27.58

Training finished in 00h 54min 38sec
```



## 4.2. Batch normalization

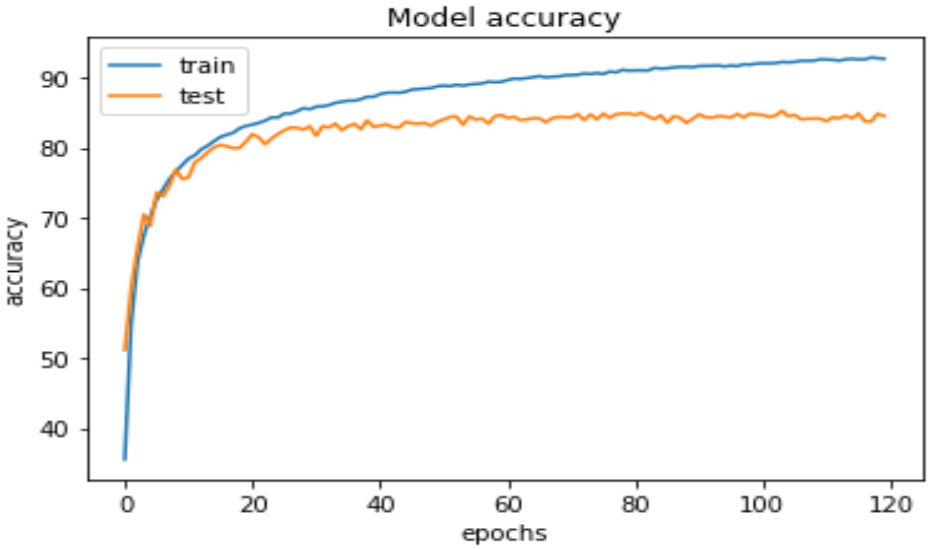
Il s'agit d'une technique qui permet de faciliter l'entraînement des réseaux de neurones, elle présente deux avantages majeurs : taux d'apprentissage plus élevé (apprentissage plus simple) et baisse de l'importance des paramètres d'initialisation du réseau. Ce qui veut dire que le réseau apprend plus vite avec moins de ressources consommées.

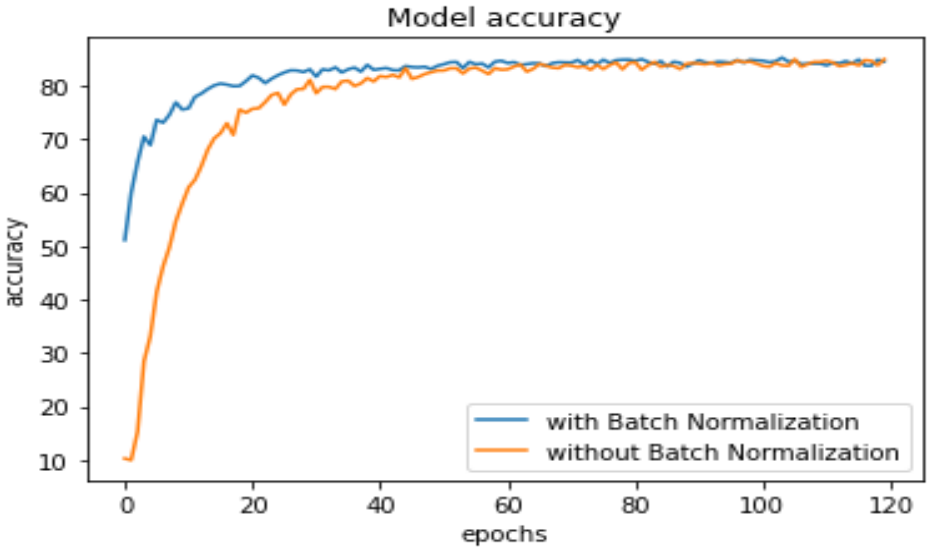
La couche de **batch normalization** appliquée sur un réseau convolutif en la couche de convolution et la couche d'activation (avant l'application de la fonction d'activation **relu** par exemple).

On ajoute donc une couche de batch normalization après la première convolution de chaque bloc : après **conv1**, **conv3** et **conv5**.

```
==========================================================================
            Kernel Shape     Output Shape   Params  Mult-Adds
Layer
0_conv1    [3, 24, 3, 3]  [1, 24, 32, 32]    672.0   663.552k
1_norm24            [24]  [1, 24, 32, 32]     48.0       24.0
2_conv2   [24, 24, 3, 3]  [1, 24, 32, 32]   5.208k  5.308416M
3_pool                 -  [1, 24, 16, 16]        -          -
4_conv3   [24, 32, 3, 3]  [1, 32, 16, 16]   6.944k  1.769472M
5_norm32            [32]  [1, 32, 16, 16]     64.0       32.0
6_conv4   [32, 32, 3, 3]  [1, 32, 16, 16]   9.248k  2.359296M
7_pool                 -    [1, 32, 8, 8]        -          -
8_conv5   [32, 48, 3, 3]    [1, 48, 8, 8]  13.872k   884.736k
9_norm48            [48]    [1, 48, 8, 8]     96.0       48.0
10_conv6  [48, 48, 3, 3]    [1, 48, 8, 8]  20.784k  1.327104M
11_pool                -    [1, 48, 4, 4]        -          -
12_fc1        [768, 220]         [1, 220]  169.18k    168.96k
13_fc2        [220, 120]         [1, 120]   26.52k      26.4k
14_fc3         [120, 10]          [1, 10]    1.21k       1.2k
--------------------------------------------------------------------------
                       Totals
Total params          253.846k
Trainable params      253.846k
Non-trainable params       0.0
Mult-Adds             12.50924M
==========================================================================
```

```
[epoch 117] train accuracy: 93%  test accuracy: 84%  loss: 0.207  duration: 29.53
[epoch 118] train accuracy: 93%  test accuracy: 84%  loss: 0.202  duration: 29.21
[epoch 119] train accuracy: 93%  test accuracy: 85%  loss: 0.202  duration: 29.20
[epoch 120] train accuracy: 93%  test accuracy: 85%  loss: 0.205  duration: 28.75

Training finished in 00h 58min 12sec
```



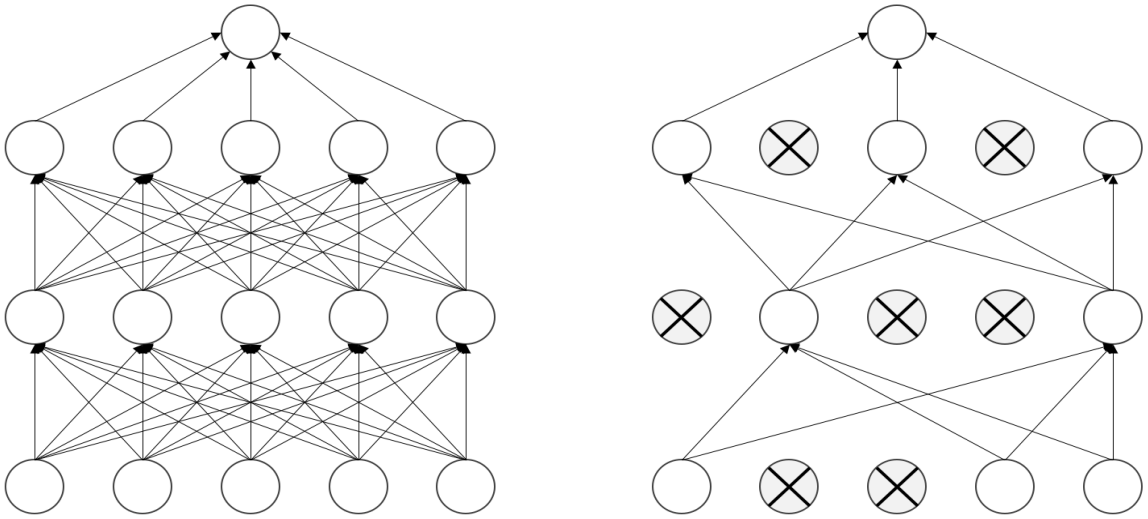

**NB :** les résultats ci-dessus prennent en compte le data augmentation.

Après avoir entraîné le réseau pendant 120 époques, on constate que les résultats ne sont pas tout à fait concluants : d'une part, la précision sur les données de test reste constante malgré l'ajout du batch normalization. D'une autre part, on remarque que l'overfitting est plus important à cause de l'augmentation de la précision sur le trainset mais pas sur le testset.

On ne retiendra donc pas cette technique dans notre réseau final.

### 4.3. Dropout

Cette technique de régularisation consiste à neutraliser (ignorer) des nœuds (neurones) lors de la phase d'apprentissage suivant une probabilité prédéfinie. Sur un réseau convolutif, cette technique n'est applicable que sur la partie classification (portion fully-connected).



Standard Neural Net

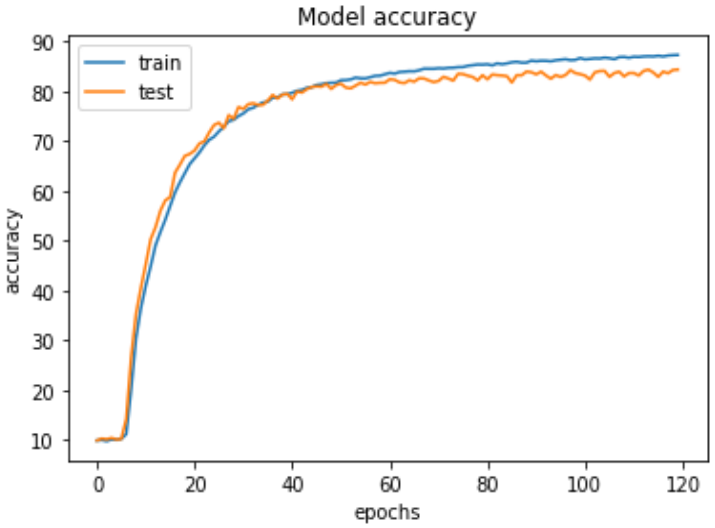
After applying dropout

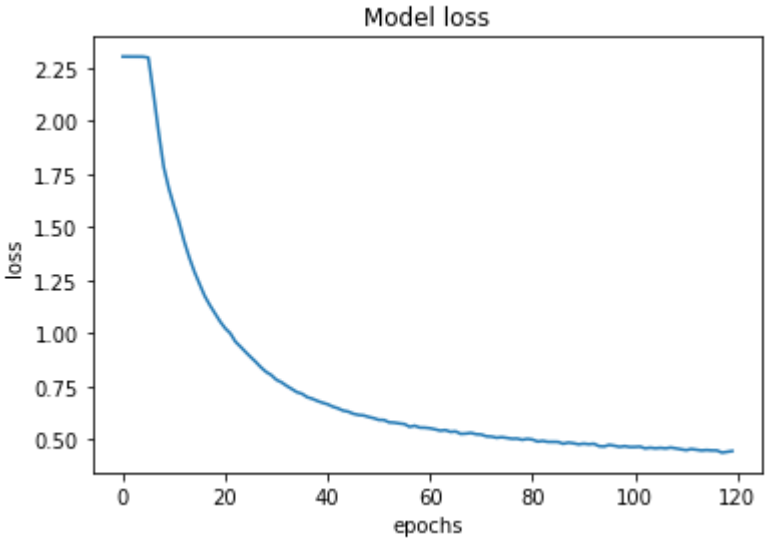
Après avoir testé plusieurs combinaisons de dropout (à différents endroits et différentes probabilités), il s'avère que le réseau suivant est celui qui offre la meilleur performance

```
==========================================================================
                Kernel Shape     Output Shape   Params  Mult-Adds
Layer
0_conv1          [3, 24, 3, 3]  [1, 24, 32, 32]    672.0   663.552k
1_conv2         [24, 24, 3, 3]  [1, 24, 32, 32]   5.208k  5.308416M
2_pool                       -  [1, 24, 16, 16]        -          -
3_conv3         [24, 32, 3, 3]  [1, 32, 16, 16]   6.944k  1.769472M
4_conv4         [32, 32, 3, 3]  [1, 32, 16, 16]   9.248k  2.359296M
5_pool                       -    [1, 32, 8, 8]        -          -
6_conv5         [32, 48, 3, 3]    [1, 48, 8, 8]  13.872k   884.736k
7_conv6         [48, 48, 3, 3]    [1, 48, 8, 8]  20.784k  1.327104M
8_pool                       -    [1, 48, 4, 4]        -          -
9_dropout10                  -          [1, 768]        -          -
10_fc1             [768, 220]          [1, 220]  169.18k    168.96k
11_dropout25                 -          [1, 220]        -          -
12_fc2             [220, 120]          [1, 120]   26.52k      26.4k
13_dropout25                 -          [1, 120]        -          -
14_fc3              [120, 10]           [1, 10]    1.21k       1.2k
--------------------------------------------------------------------------
                          Totals
Total params             253.638k
Trainable params         253.638k
Non-trainable params          0.0
Mult-Adds              12.509136M
==========================================================================
```

un dropout de 5% avant la 1ere couche fully connected et
un dropout de 25% après la 1ere et deuxième couche fully connected.

```
[epoch 115] train accuracy: 87%  test accuracy: 84%  loss: 0.376  duration: 28.62
[epoch 116] train accuracy: 87%  test accuracy: 83%  loss: 0.373  duration: 28.08
[epoch 117] train accuracy: 87%  test accuracy: 84%  loss: 0.376  duration: 28.39
[epoch 118] train accuracy: 87%  test accuracy: 84%  loss: 0.370  duration: 28.17
[epoch 119] train accuracy: 87%  test accuracy: 84%  loss: 0.372  duration: 28.65
[epoch 120] train accuracy: 87%  test accuracy: 84%  loss: 0.367  duration: 28.12

Training finished in 00h 56min 49sec
```




On constate une baisse de 1% de la précision sur l'ensemble de tests, passant de 85% à 84%.
De la même manière, la précision sur l'ensemble d'apprentissage passe de 91% à 87% grâce au dropout, ce qui permet de réduire encore plus l'overfitting de notre réseau.

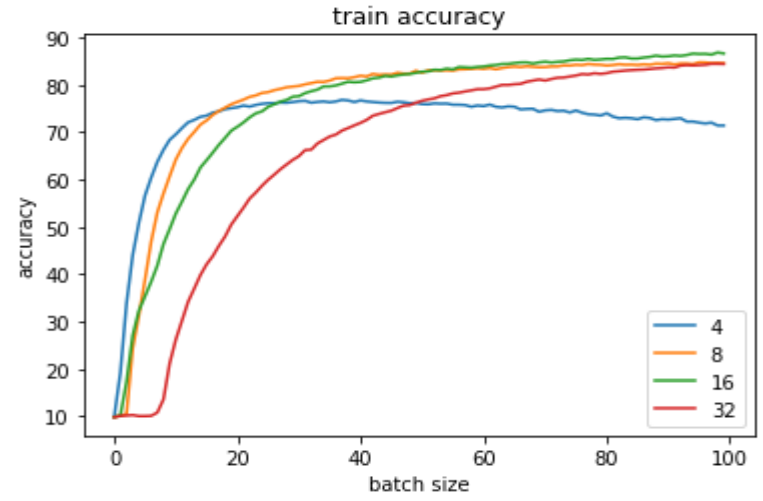
## 4.4. Hyperparameters tuning

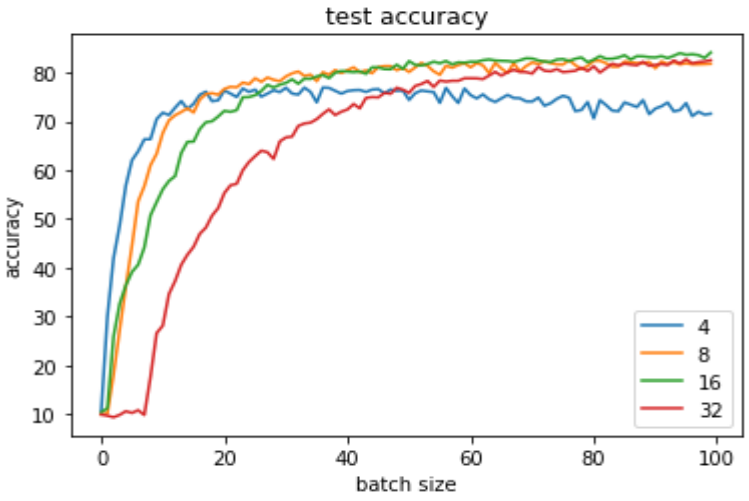
Dans cette partie on s'appuiera sur le réseau de la partie 4.3, qui utilise les techniques de **data augmentation** et de **dropout** (mais pas le batch normalization).

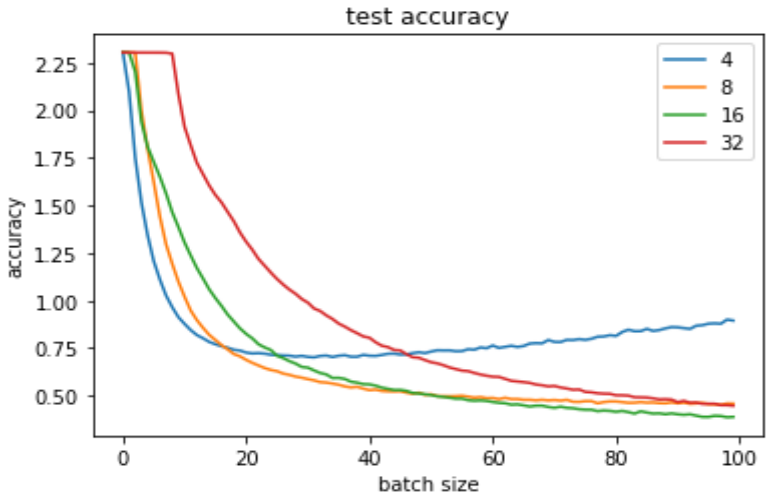
### 4.4.1. Batch size

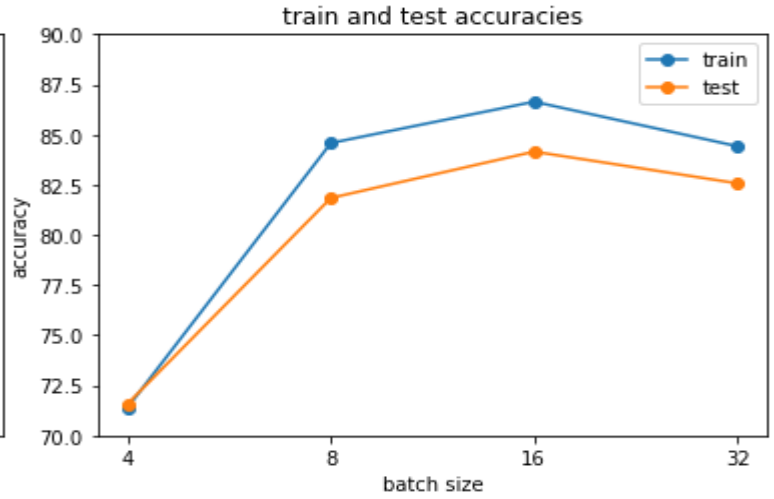
Dans cette partie nous testerons notre réseau avec différentes tailles de batchs afin de déterminer la taille qui fournit un résultat optimal.

Les figures ci-dessous comparent l'évolution des performances des différentes tailles de batchs : 4, 8, 16 et 32.



On remarque qu'en utilisant 8, 16 et 32 comme tailles de batchs les résultats sont assez proches avec un léger avantage de la taille 16, et une convergence plus lente pour la taille 32.

On remarque également que pour la taille 4, plus on avance dans les époques plus la précision se détériore. Il serait envisageable dans ce cas de figure d'utiliser la technique d'**arrêt prématuré (early stopping).**

# 5. Conclusion

Au cours de ce projet, nous avons étudié différents aspects des réseaux de neurones convolutifs a commencer par les couches de convolutions et de classification, les datasets, l'apprentissage et validation, les techniques de régularisation les plus connues et pour finir les batch size en tant que hyperparamètre.