

### Exercice 0.1 AlexNet (exercice 1)

1 - D'après le schéma de la figure 1 du TD, les images acceptées en entrée sont de taille  $224 \times 224$  avec 3 canaux (RGB). D'après la description de la première couche sur MATLAB, les images en entrée doivent être des images  $227 \times 227 \times 3$  : hauteur 227, largeur 227, 3 canaux (RGB pour des images couleur).

2 - Pour commencer, la taille des images en entrée n'est pas la même, comme on l'a remarqué dans la 1ère question. Aussi, dans l'article AlexNet, il n'est pas fait mention de *padding* alors que le réseau implémenté sur Matlab décrit un padding pour chaque couche (de convolution, mais pas que, pour le *max pooling* également).

3 - On teste deux images dans le fichier TD3.m (un oiseau "bulbul", et un chien "schipperke"). Voir fichiers Matlab.

4 - Matlab fournit `size(w) = 11, 11, 3, 96`, car la première couche de convolution consiste en 96 convolutions sur 3 canaux/features, ayant un noyau de taille  $11 \times 11$ . Autrement dit, on réalise 96 convolutions différentes (extraction de 96 *features*) sur les images en entrée (constituées donc de 3 canaux), et on utilise des noyaux de taille  $11 \times 11$ .

5 - On implémente le *stride* (pas) de 4 en extrayant une fraction du tableau obtenu en faisant la convolution de pas 1 (voir code Matlab). Ce faisant, pour l'image d'un "bulbul" (un oiseau) prise sur internet, on obtient 96 images de features de la première couche qui ressemblent très fortement (si ce n'est exactement - à quelques nuances plus sombres dans les *features* renvoyées par `conv2` près) aux images renvoyées et affichées grâce à la fonction `activations`. Voir pour cela la figure 1.

Les quelques différences entre les deux affichages peuvent notamment (mais pas que) être liées à la gestion des pixels négatifs renvoyés par les convolutions.

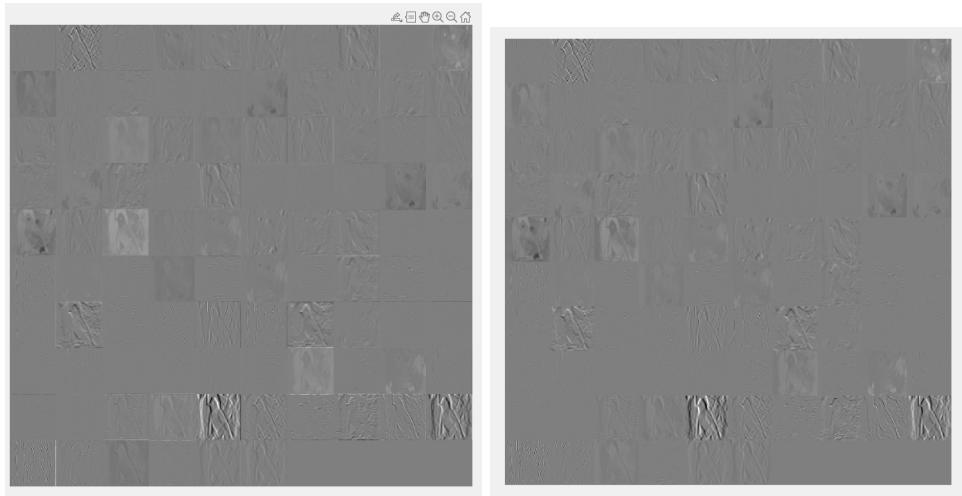


Figure 1 – A gauche : *features* obtenues "à la main" en appliquant la fonction `conv2` sur chacun des canaux de l'image. A droite : les *features* obtenues grâce à la fonction `activations`.

### Exercice 0.2 L'attrait des profondeurs

1 - Après entraînement, on obtient une précision sur la base de validation de 67,51% (et 98,44% sur le dernier MiniBatch de la base de test). Cette différence marquée, et le fait que la *loss* sur la base de validation augmente au bout de quelques *epoch* alors que l'erreur sur la base de test continue de diminuer, est typique d'une situation de **sur-apprentissage**. Voir la figure 2 représentant l'évolution de l'erreur et de la précision au cours de l'entraînement du réseau `layers_1`.

2 - Disclaimer : Pour éviter le sur-apprentissage, on fait du early stopping lorsque que l'on voit que l'erreur sur la base de validation commence à croître (cela a aussi l'avantage de réduire le temps d'apprentissage).

- Deuxième réseau de neurone (figure 3) :
- précision sur la base de validation : 72,81%.

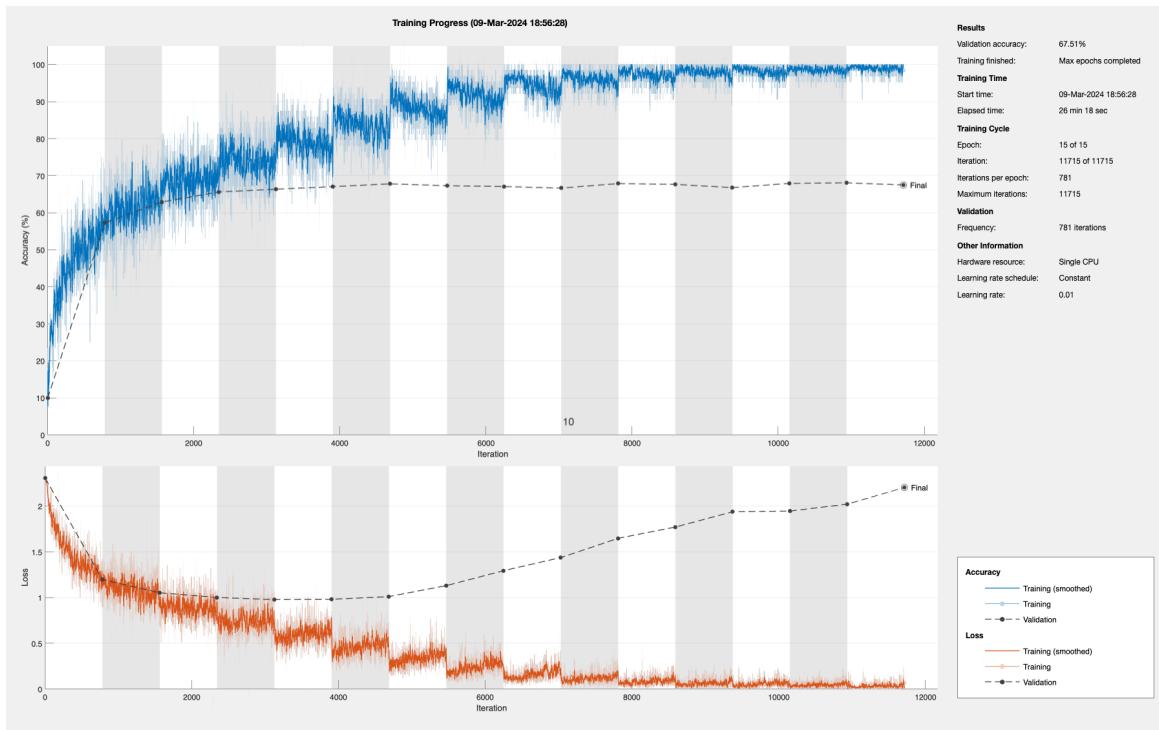


Figure 2 – Evolution de l'erreur et de la précision au cours de l'entraînement du réseau layers\_1.

- early stopping à la 9e epoch (sur-apprentissage à partir de la 6e époque)
- Troisième réseau (figure 4) :
  - précision sur la base de validation : 76,55%.
  - early stopping à la 13e epoch (sur-apprentissage à partir de la 10e environ)

On observe majoritairement deux effets visiblement liés à la profondeur du réseau de neurones (tout autre paramètre restant égal par ailleurs) :

- Plus le réseau est profond, mieux il apprend sur les données. En pratique on sait aussi que si le réseau est **trop** profond alors ça n'est plus le cas (au contraire le réseau apprend moins bien), mais comme le résume bien l'article décrivant AlexNet : "*the depth really is important for achieving our results*"
- d'autre part, on remarque que, plus le réseau est profond, plus le sur-apprentissage intervient tardivement (la valeur de la fonction coût sur la base de validation se remet à augmenter à la 6e, 9e puis 10e époque environ, respectivement). Ainsi il semblerait qu'approfondir le réseau aide à limiter le sur-apprentissage.

**3 -** Théoriquement, avec un learningRate de 0.1 (donc 10 fois plus important que 0.01), on peut s'attendre à ce que la convergence vers un minimum d'erreur sur la base de test soit mauvaise, avec des oscillations dues à un pas d'apprentissage trop élevé pour l'algorithme de descente du gradient. Et en pratique, on observe effectivement que le modèle apprend très mal sur les données avec un pas d'apprentissage de 0.1 : dès la 2<sup>e</sup> époque, l'accuracy et l'erreur (que ce soit sur la base de test ou sur la base de validation) stagnent à des niveaux assez mauvais (33% pour la précision sur la base de validation par exemple, contre plus de 60% pour le premier modèle à la même epoch) ; la descente de gradient se fait "trop vite", et elle n'est pas efficace lorsque le pas d'apprentissage est trop élevé, ce qui semble être le cas ici - on passe "à côté" du minimum visé par le gradient. La figure 5 présente l'apprentissage sur quelques epoch pour un pas d'apprentissage de 0.1 (modèle layers\_3).

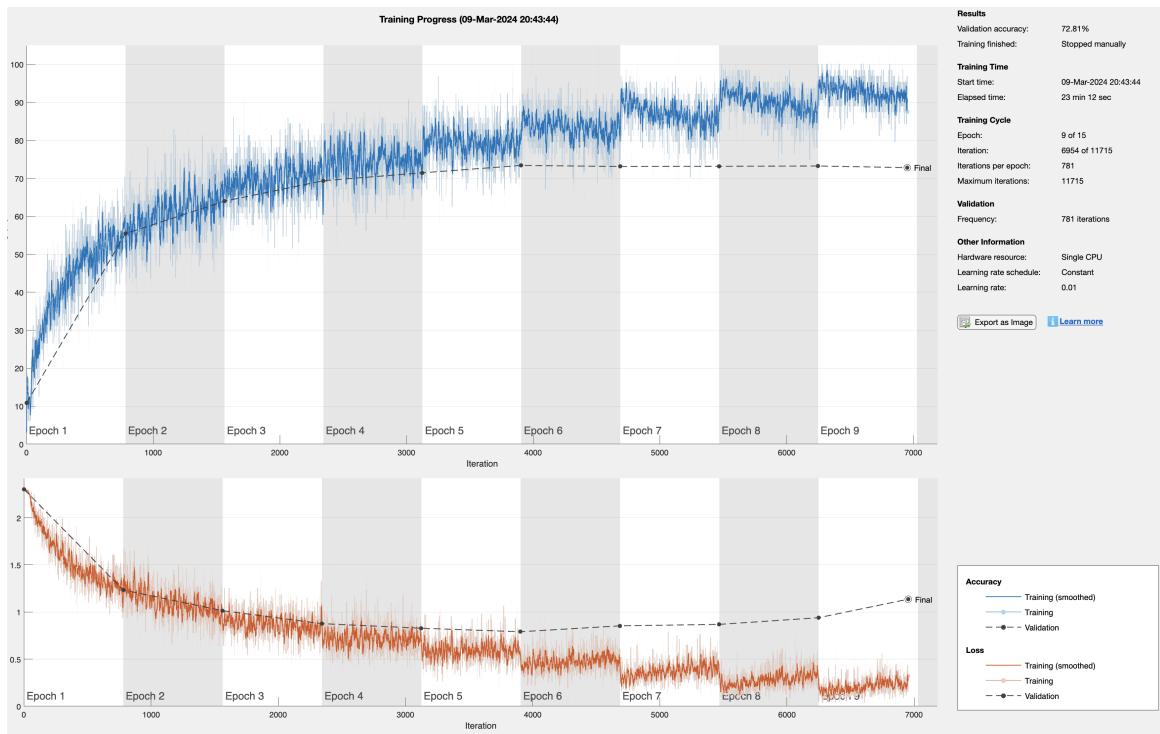


Figure 3 – Evolution de l'erreur et de la précision au cours de l'entraînement du réseau layers\_2.

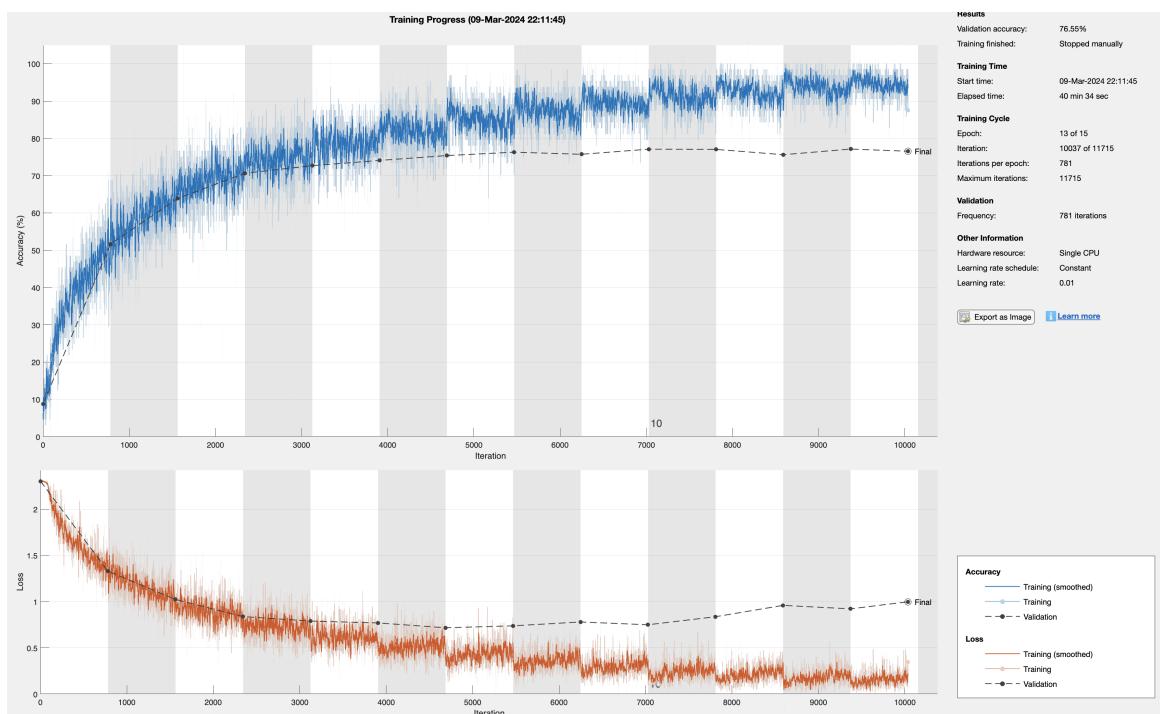


Figure 4 – Evolution de l'erreur et de la précision au cours de l'entraînement du réseau layers\_3.

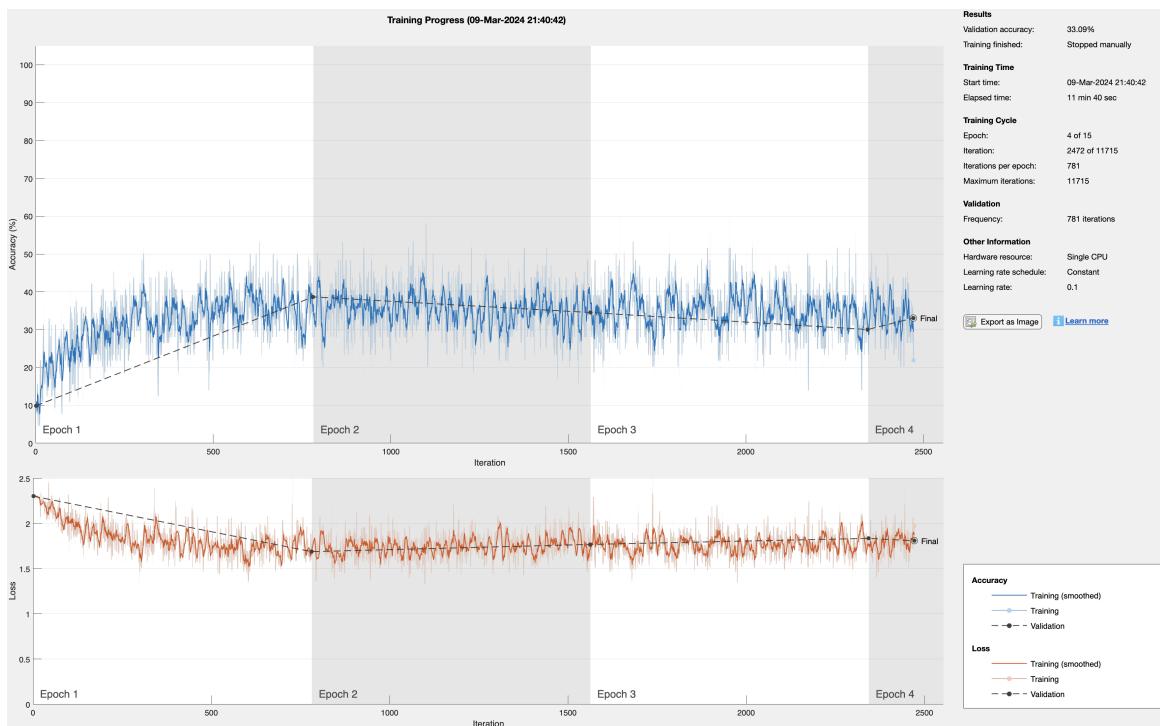


Figure 5 – Apprentissage du 3e réseau (layers\_3) pour un pas d'apprentissage de 0.1, sur 3 époques (early stopping avant la fin de la 4<sup>e</sup>). On voit que les métriques stagnent très vite à un niveau bien en-deçà des niveaux obtenus avec un pas de 0.01 pour n'importe lequel des 3 réseaux.