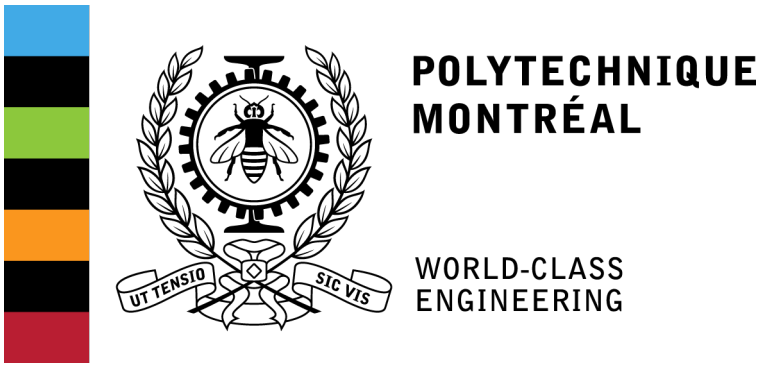
École Polytechnique de Montréal

Département de génie informatique et génie logiciel



INF8225 - Intelligence artificielle : techniques probabilistes et d'apprentissages

---

RAPPORT DU LABORATOIRE 2

Réseaux de neurones et Apprentissage automatique

Foromo Daniel Soromou 1759116

Gilles Eric Zagre 1146014

Chargé de laboratoire

Alexandre Piché

Dimance, 18 février 2018

Table des matières

1 Introduction

Dans ce laboratoire nous allons étudier la rétropropagation dans un réseau de neurones entièrement connecté *(fully connected)*, puis nous allons nous familiariser avec l'apprentissage machine avec différentes architectures de réseaux en utilisant la librairie Pytorch.

2 Partie I : Algorithme de rétropropagation avec plusieurs couches

Dans cette section nous décrivons l'algorithme de rétropropagation qui sert à ajuster les poinds W d'un réseau de neurones lors de la phase d'apprentissage, à partir de l'erreur de prédiction observée.

2.1 Pseudocode et calculs matriciels

Dans la description proposée, on suppose un réseau de neurones entièrement connecté avec D=100 neurones dans chaque couche cachée :

- x est un vecteur de taille Dx1
- $W^{(l)}$ est une matrice de taille DxD (*fully-connected,* $(l = 1...L)$)
- $b^{(l)}$ est un vecteur de taille Dx1 $(l = 1...L)$
- $h^{(l)}$ est un vecteur de taille Dx1 $(l = 1...L)$

Algorithm 1 Rétropropagation avec plusieurs couches

procedure APPRENTISSAGE(X, W) ▷ *L'apprentissage de W avec X*
  for $i \leftarrow 1, nbpochs$ **do**
    for $example(x)$ *in Training set* **do**
      $h^{(1)} = Sigmoide(\sum_{i=1}^{D} W_i^1 * x_i + b^{(1)}) = Sigm(W^1.x^T + b^{(1)})$
      for $l \leftarrow 2, L$ **do**
        $h^{(l)} = Sigmoide(\sum W^l.h^{(l-1)} + b^{(l)}) = Sigm(W^l.[h^{(l-1)}]^T + b^{(l)})$
      $\hat{y} = Sigmoide(\sum W^{L+1}.h^{(L)} + b^{(L+1)})$
      $\Delta_y = y_{example} - \hat{y}$ ▷ Δ *is array of DxL*
      $\Delta^{(L)} = h^{(L)} * (1 - h^{(L)}) * \sum W^{(L+1)} * \Delta_y$
      for $l \leftarrow L - 1, 1$ **do**
        $\Delta^{(l)} = h^{(l)} * (1 - h^{(l)}) * \sum W^{(l+1)} * \Delta[l + 1]$
      for $l \leftarrow 1, L + 1$ **do**
        $W^{(l)} = W(l) + \alpha \ * h^{(l)} * \Delta[l]$
  return W ▷ *Le meilleur ensemble de poids*

2.2 Optimisation du pseudocode

Pour un vaste ensemble de données (ex : N = 500 000), nous allons dans un premier temps subdiviser les données en ensemble d'entraînement (ex : 70%), de validation (ex : 15%) et de test (ex : 15%).

Du fait du très grand nombre d'exemples d'entrainement, le stockage en mémoire peut être problématique. Ainsi, il sera important d'opter pour l'approche par minibatch. Malheuresement, lorsque nous avons un grand nombre de mini-batchs, la convergence peut-être difficile a attiendre. Ainsi, une solution a ce problème sera de normaliser les mini-batchs.
Il se peut aussi que le gradient soit coincé dans un optimum local après plusieurs itérations, une solution possible est d'augmenter le taux d'apprentissage pendant quelques itérations afin de sortir de cet optimum local. Ceci pourrait avoir pour effet de dégrader pendant quelque temps la solution (meilleur poids trouvés), toutefois cela nous permettra de retrouver un autre optimum local qui pourrait être meilleur. Pour ne pas cycler (retomber dans le même optimum), On pourrait interdire de retomber dans les solutions déja explorées en interdisant une direction de descente déja empruntée à partir d'un point de l'espace de recherche, ce serait une exploration suivant la méthode Tabou.

3 Partie II : expérimentation avec les données Fashion Mnist

Dans cette partie du laboratoire, nous utilisons la librairie Pytorch pour expérimenter l'apprentissage d'un réseau de neurones avec les données Fashion Mnist. Nous essaierons plusieurs types d'architecture de réseau puis sous en sélectionnerons un pour lequel nous vérifierons la performance avec l'ensemble de test.

3.1 Essais avec des *réseaux de neuronnes linéaires*

Dans cette section nous allons essayer plusieurs architectures de réseaux de neurones avec des couches cachées entièrement connectées. Nous ferons varier plusieurs paramètres tout en observant les performances en aprentissage et en validation.

3.1.1 Réseau avec une couche *Fully-Connnected*

Paramètres : *(Voir fichier Network1.py)*
- Couche de sortie avec fonction d'activation soft-max.
- Couche cachée fully-connected avec 512 neurones
- Taux d'apprentissage : 0.01, 0.001 et 0.0001

La Figure 1 présente les résultats obtenus pour ce réseau avec différents taux d'apprentissage. On constate que le taux de 0.001 offre la meilleure performance. La précision maximale obtenue avec ce réseau est de 90%.

Cette prémière expérience montre plus en détail l'impact du taux d'apprentissage. On remarque qu'avec un taux d'apprentissage de 0.01 le modèle ne s'améliore pratiquement jamais. Par contre avec un taux d'apprentissage de 0.0001, le modèle s'améliore mais très lentement. De plus avec un taux d'apprentissage très pétit, le modèle peut stagner dans un minimum local sans jamais en sortir. Finalement avec un taux d'apprentissage de 0.001, le modèle converge rapidement et nous n'avons pas de sur-apprentissage.

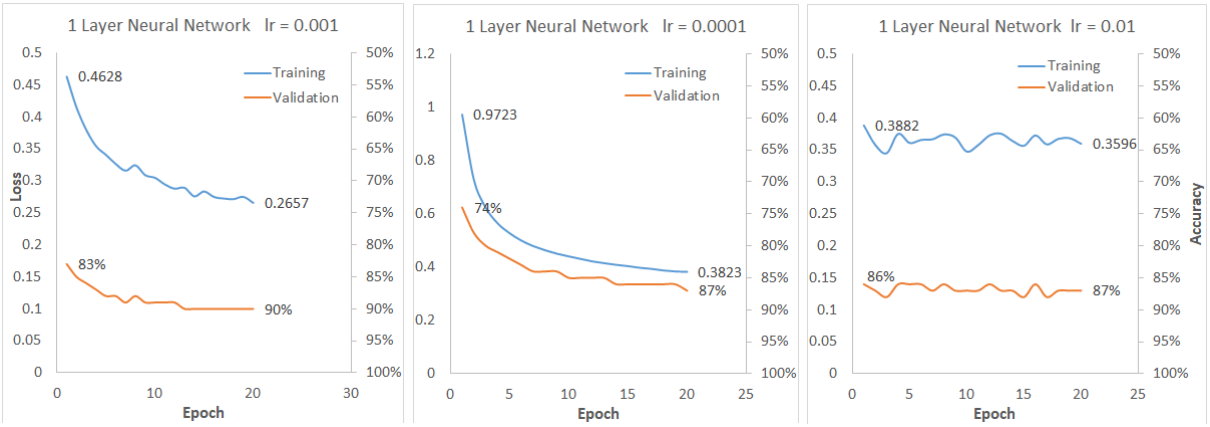


FIGURE 1 – Apprentissage et Validation avec une couche fully-connected et différents taux d'apprentissage

3.1.2 Réseau avec deux couches *Fully-Connnected*

Paramètres : *(Voir fichier Network2.py)*
— Couches cachées fully-connected avec 512 neurones et 256 neurones
— Taux d'apprentissage : 0.01, 0.001 et 0.0001

La Figure 2 présente les résultats obtenus pour ce réseau avec différents taux d'apprentissage. On constate que le taux de 0.001 offre encore la meilleure performance. La précision maximale obtenue avec ce réseau est de 90%. L'ajout d'une couche n'a donc pas significativement amélioré la performance du réseau.

Comme l'expérience précédente, il facile de remarquer l'impact du taux d'apprentissage. Cette fois-ci, nous allons plus aborder l'effet combiné du nombre de couches. Avec une croyance naive, on pourrait penser que plus nous avons des couches plus le modèle est efficace. Mais dans ce cas présent, on remarque que le fait de doubler notre couche, n'améorile en aucun cas le réseau. Ainsi la conclusion de cette expérience, est qu'il est préferable d'avoir un modèle plus pétit mais qui est éfficace et qui converge vite. Car, lorsqu'un modèle est grand, il aura beaucoup de paramètre et en plus l'apprentissage prendra beaucoup plus de temps.

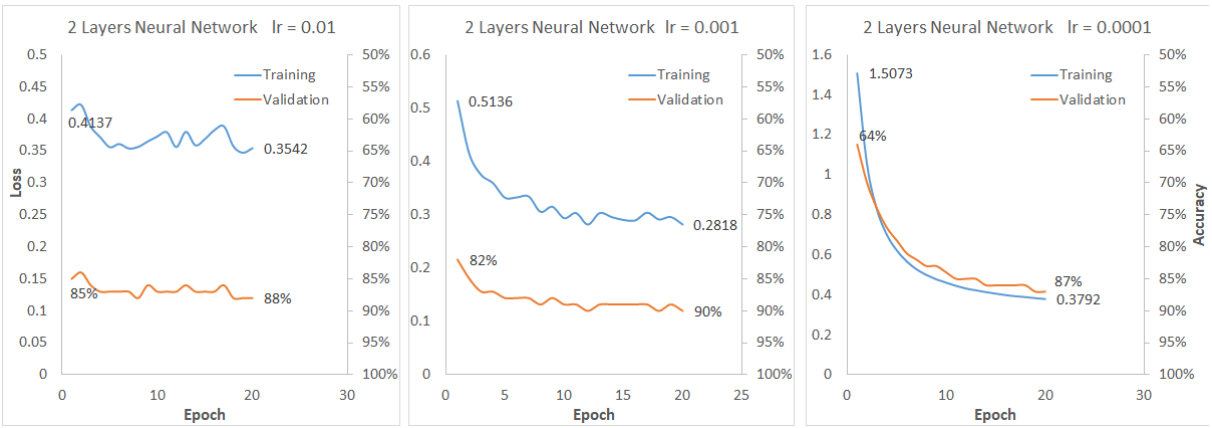


FIGURE 2 – Apprentissage et Validation avec 2 couches fully-connected et différents taux d'apprentissage

3.1.3 Réseau avec trois couches *Fully-Connnected*

Paramètres : *(Voir fichier Network3.py)*
— Couches cachées fully-connected avec 512, 256 puis 128 neurones
— Taux d'apprentissage : 0.01, 0.001 et 0.0001

La Figure 3 présente les résultats obtenus pour ce réseau avec différents taux d'apprentissage. On constate que le taux de 0.001 offre encore la meilleure performance. La précision maximale obtenue avec ce réseau est de 89%. L'ajout d'une autre couche n'a donc pas amélioré la performance du réseau.

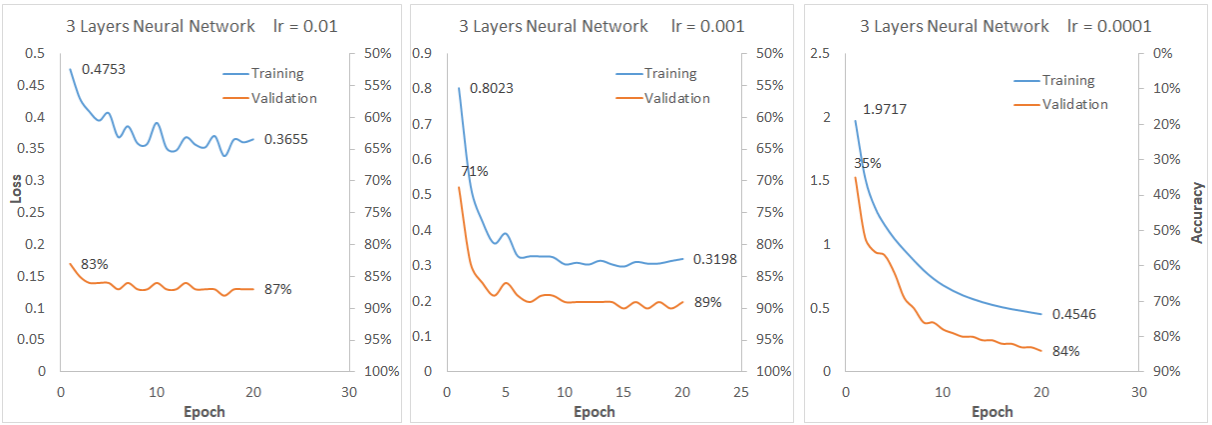


FIGURE 3 – Apprentissage et Validation avec 3 couches fully-connected et différents taux d'apprentissage

Comme l'expérience précédente, nous avons les mêmes effects combinés du taux d'apprentissage et du nombre de couches. Mais cette fois-ci, nous avons 3 couches fully connected. Mais cette fois-ci, on remarque un cas de sur-apprentissage lié au nombre de couches. Car avec 3 couches fully connected, le modèle apprend tous les pétits détails ainsi que les bruits de notre exemple d'apprentissage. Ainsi, notre modèle aura du mal à généraliser notre exemple car les bruits pour chaque exemplaire ne s'applique pas aux autres. Ce qui explique le fait que nous avons une précision maximale (89%) inférieure à la précision maximale de l'exprérience précédente (90%)

3.2 Essais avec des réseaux de neuronnes convolutifs

Un réseau de neural convolutif (CNN) est constitué d'une ou de plusieurs couches convolutives, puis il s'en suivra plusieurs couches entièrement connectés (fully connectted layer(s)). Les CNN ont été concu pour des structures 2D ou 3D(séquences d'image). Les CNN utilisent des filtres afin d'extraire des caractéristiques invariants des connexions locales. De plus, un avantage des CNN est qu'il est plus facile d'avoir moins de paramètres que les réseaux de neurones entièrement connectés avec le même nombre de couches cachées (hidden layer).

Dans cette section nous essayons des réseaux de neurones avec des couches de convolution pour étudier leurs impacts.

3.2.1 Réseau avec une couche de convolution

Paramètres : *(Voir fichier Network4.py)*
— Couche de convolution : Pas=1, Padding=1, Taille du filtre=3x3

La Figure 4 présente les résultats obtenus pour ce réseau. La précision maximale obtenue avec ce réseau est de 90%.
Comme nous pouvons le remarqué, avec une seul couche de convolution notre modèle converge rapidement avec une précision élévée. Mais on remarque qu'à partir de la 20ième itération, le modéle d'apprend quasiment plus.

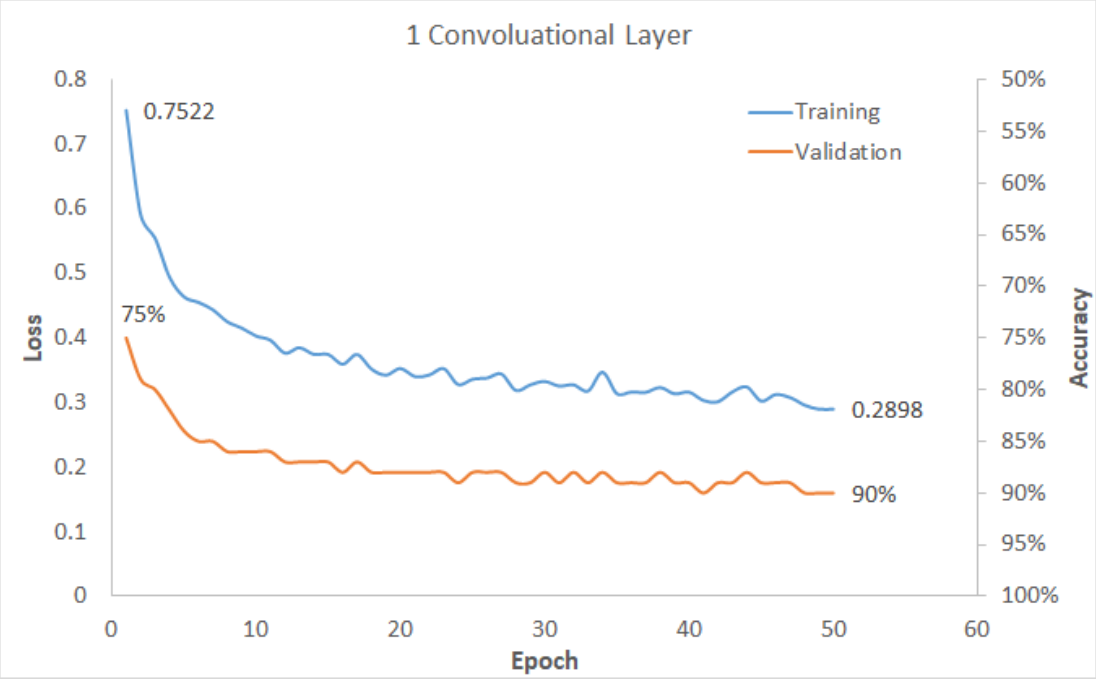


FIGURE 4 – Apprentissage et Validation avec une couche de convolution

3.2.2 Réseau avec 2 couches de convolution

Paramètres : *(Voir fichier Network5.py)*
— Couches de convolution : Pas=1, Padding=1, Taille du filtre=3x3

La Figure 5 présente les résultats obtenus pour ce réseau. La précision maximale obtenue avec ce réseau est de 90%.

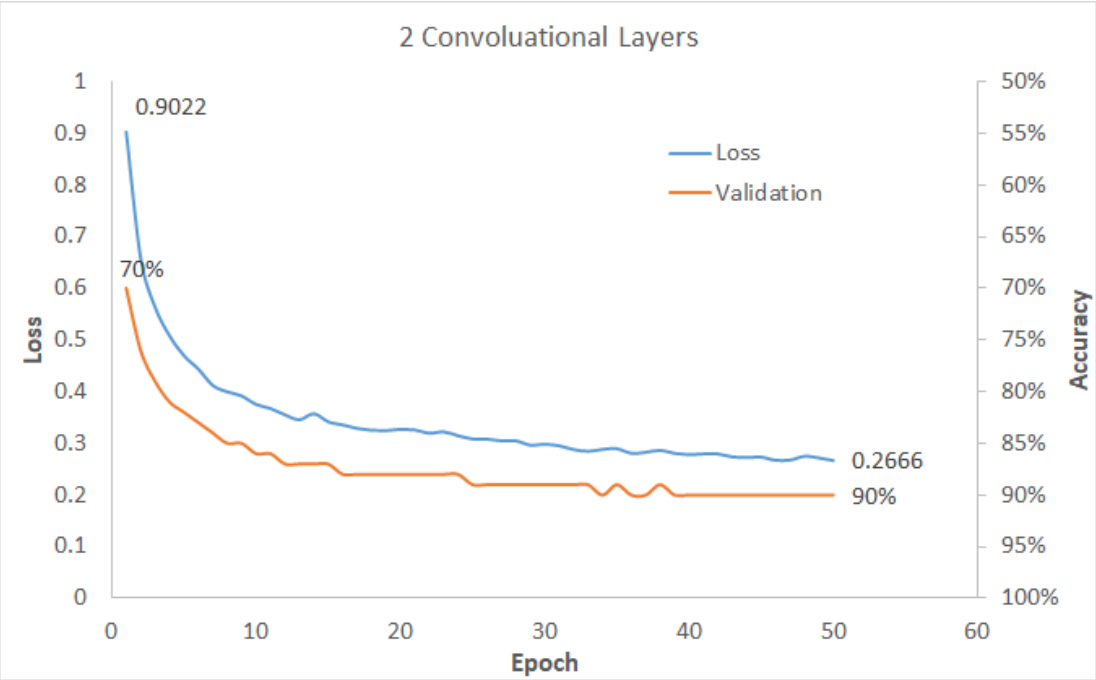


FIGURE 5 – Apprentissage et Validation avec 2 coucheS de convolution

3.2.3 Réseau avec 3 couches de convolution

Paramètres : *(Voir fichier Network6.py)*
— Couches de convolution : Pas=1, Padding=1, Taille du filtre=3x3

La Figure 6 présente les résultats obtenus pour ce réseau. La précision maximale obtenue avec ce réseau est de 90%.

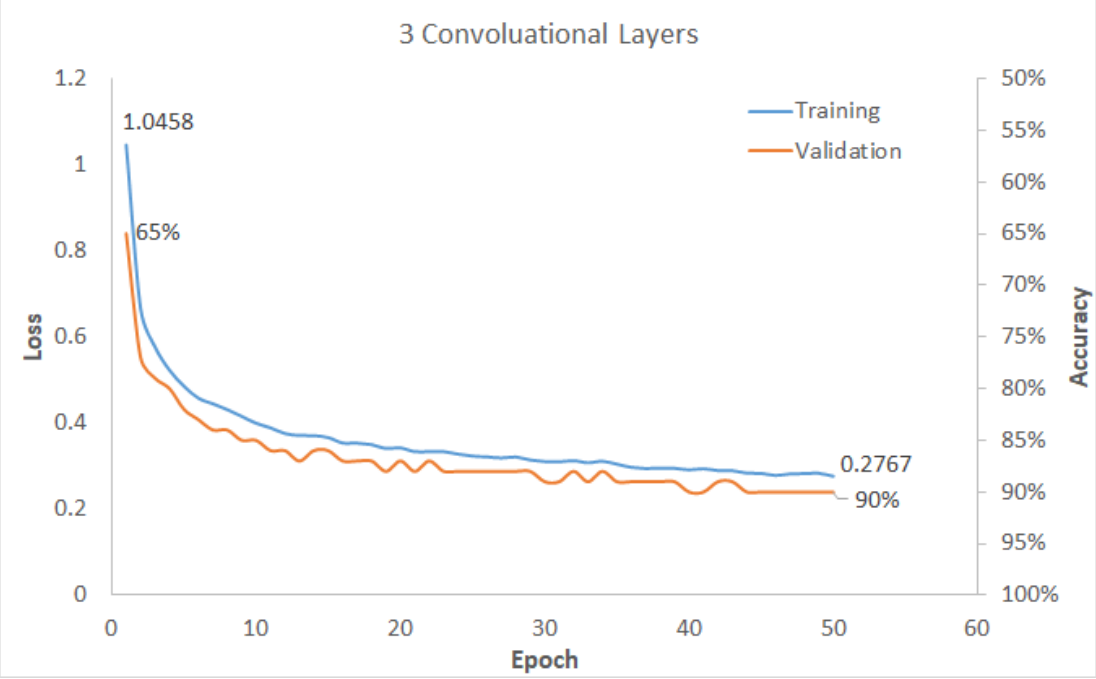


FIGURE 6 – Apprentissage et Validation avec 3 coucheS de convolution

3.2.4 Réseau avec 3 couches de convolution et une couche *dense*

Paramètres : *(Voir fichier Network7.py)*
— Couches de convolution : Pas=1, Padding=1, Taille du filtre=3x3

La Figure 7 présente les résultats obtenus pour ce réseau. La précision maximale obtenue avec ce réseau est de 92%.

Dans un monde idéal, cette architecture sera parfaite car elle converge (10 itérations) trés rapidement et avec trés peu de paramètre.

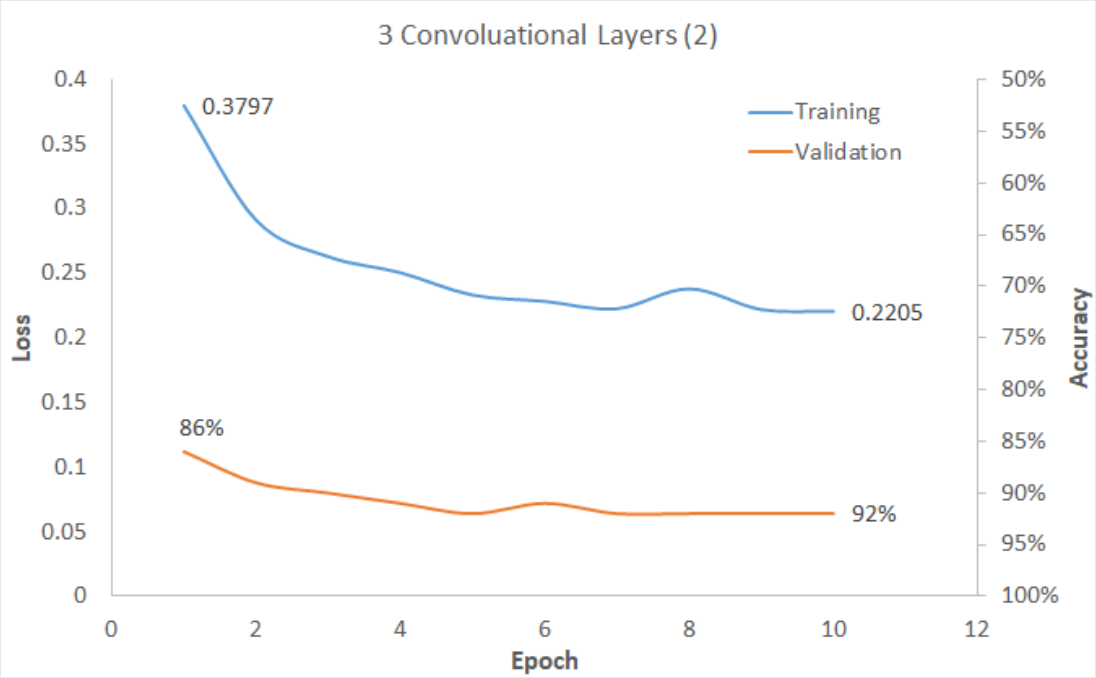


FIGURE 7 – Apprentissage et Validation avec 3 couches de convolution et une couche *dense*

3.2.5 Réseau avec une couche de convolution *mobilenet*

Cette fois-ci, nous nous sommes inspiré une architecture classique existant sous le nom de mobilenet pour faire une étude comparative avec nos architectures.
Paramètres : *(Voir fichier Network8.py)*
— Couches de convolution : Pas=1, Padding=1, Taille du filtre=3x3

La Figure 8 présente les résultats obtenus pour ce réseau. La précision maximale obtenue avec ce réseau est de 92%. Ce que nous voulons montrer ici, est que, bien qu'on ai 4 fois plus de couche que notre réseau précedent, nous avons la même précison. De plus, le notre converge plus vite que le mobilenet.

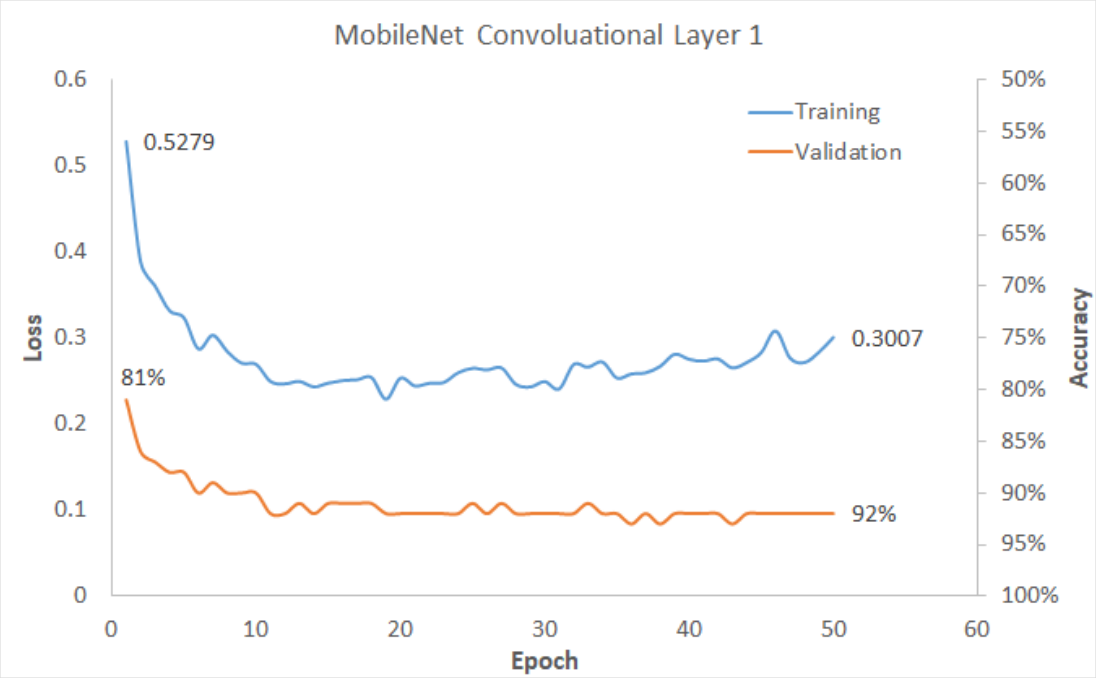


FIGURE 8 – Apprentissage et Validation avec une couche de convolution *mobilenet*

3.2.6 Réseau avec 3 couches de convolution + *Maxpooling* + *Batch Normalization* et *Augmentation de données*

Paramètres : *(Voir fichier Network9.py)*
— Couches de convolution : Pas=1, Padding=1, Taille du filtre=3x3

La Figure 9 présente les résultats obtenus pour ce réseau. Pour ce réseau les données sont augmentées en rajoutant la rotation des images à l'entrée. Bien que la convergence est un peu plus longue (près de 300 itérations), la précision maximale obtenue avec ce réseau est de 93%.

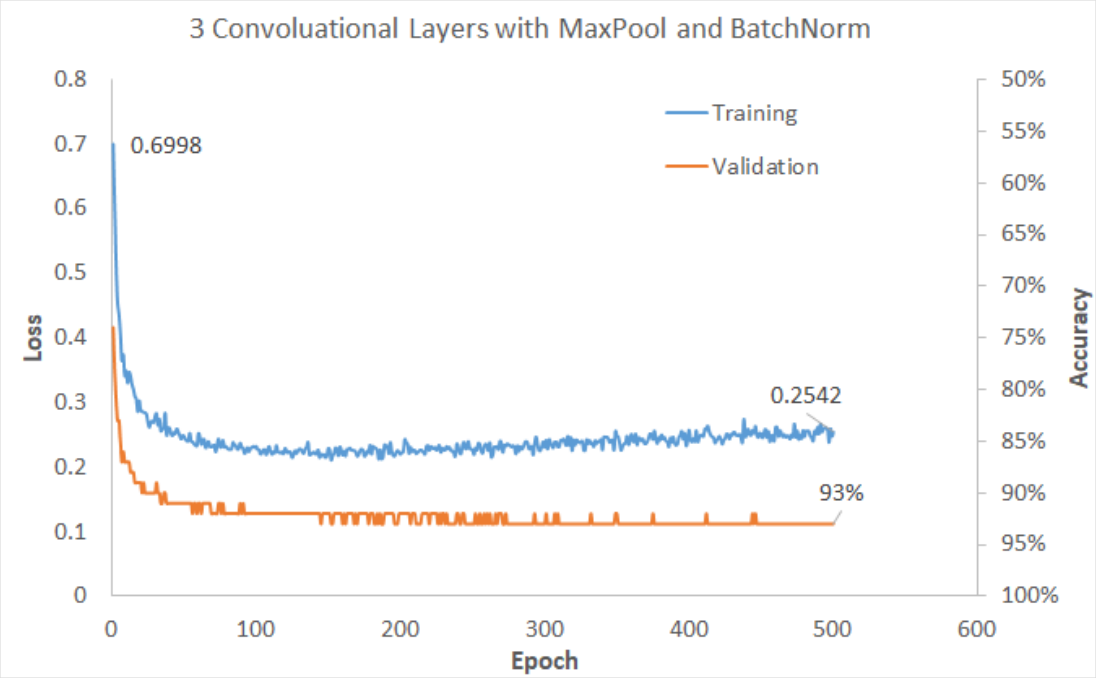


FIGURE 9 – Apprentissage et Validation avec 3 couches de convolution + *Maxpooling* et *BatchNorm* + *Data Augmentation*

3.3 Sélection et performance sur l'ensemble de test

La derniere architecture de réseau (3 couches de convolution + *Maxpooling* + *Batch Normalization* + *Data Augmentation*) offre la meilleure performance (93%). C'est donc cette architecture que nous retenons pour notre réseau. La performance sur de ce réseau sur l'ensemble de test a permi d'obtenir une **précision de prédiction de 93% sur l'ensemble de test**.

4 Conclusion

Dans ce laboratoire nous avons fourni une description détaillée du pseudo code pour l'algorithme de rétropropagation qui permet d'ajuster les poids dans un réseau de neurones.

Nous avons également essayé l'apprentissage à l'aide des donnée Fashion Mnist en utilisant pluiseurs architectures de réseau. Cela a permis de trouver une structure de réseau qui offre une performance de 93% lorsque testé sur l'ensemble de test.

Références

[1] Notes de cours INF8225 École Polytechnique de Montréal.

[2] https ://www.codeday.top/2017/10/20/50797.html, Consulté le 3 Février 2018