Date de remise: 21 octobre (23h00), 2022

Rapport partie pratique

Hamed Nazim MAMACHE

Question 3. Implémentation d'un CNN et comparaison avec les MLP utilisant PyTorch

4. Tracer des courbes de perte d'entraînement, de précision d'entraînement, de perte de validation et l'exactitude de la validation à travers les époques et l'inclure dans votre rapport (ReLU activation, `xavier_normal` initialization).

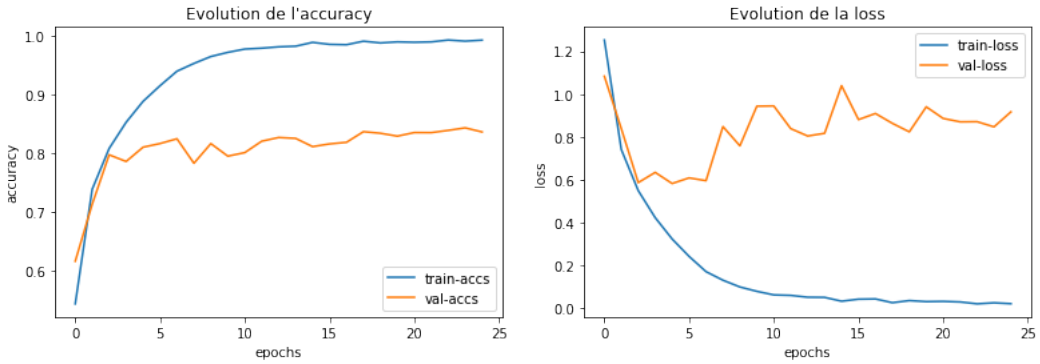

FIGURE 1 – Evolution de la perte et de l'exactitude de l'entraînement et de la validation

Observations :
Concernant les courbes de l'entraînement, on obtient un résultat attendu : l'accuracy augmente très vite durant les 10 premières epochs, puis converge vers 1 par la suite. De plus, la loss diminue très vite durant les 10 premières epochs, puis converge vers 0 par la suite.
Concernant les courbes de la validation, l'accuracy stagne autour de 80% à partir de la 5ème epoch environ.
Concernant la loss, on retrouve bien l'aspect d'une courbe convexe : entre l'epoch 0 et 3, on semble être en sous apprentissage, entre l'epoch 3 et 6 (environ) on semble être dans un apprentissage plutôt correct, et à partir de la 7 ème epoch, on semble être en surapprentissage. En effet, à partir de la 7ème epoch, la loss de la validation augmente tandis que celle de l'entraînement continue de converger vers 0.

5. Visualisez quelques filtres de la première et de la dernière couche de convolution dans le modèle formé. Inclure les images dans votre rapport et commentez les fonctionnalités apprises par les premier et dernier calques de convolution. En quoi diffèrent-elles ?

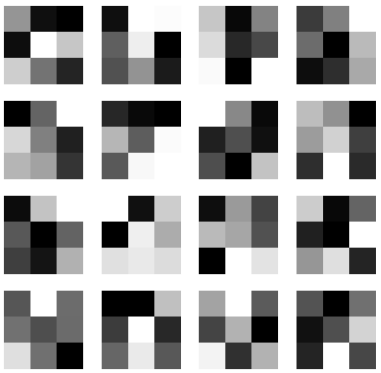

FIGURE 2 – Visualisation de quelques filtres de la première couche de convolution

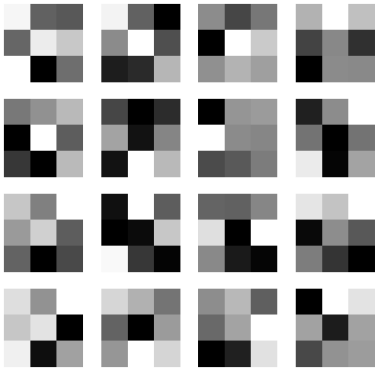

FIGURE 3 – Visualisation de quelques filtres de la dernière couche de convolution

Observations :
Les différents filtres de convolution extraient différentes caractéristiques d'une image. De plus, ils sont également une représentation des poids qui sont mis à jour pendant l'entraînement.
En effet, les valeurs du filtre sont considérées comme des poids. Ainsi, les filtres sont mis à jour (optimisés) à mesure que les poids sont optimisés pendant l'entraînement.
Il existe plusieurs façons de choisir les filtres, dans notre cas, on procède à l'initialisation des poids aléatoire (xavier_normal).
Ici, les filtres de la première couche de convolution diffèrent avec celles de la dernière couche de convolution puisque le modèle cherche à extraires des caractéristiques différentes de l'image en entrée.

6. Visualiser les cartes de caractéristiques à partir des première et dernière couches de convolution (pour quelques filtres) dans le modèle formé pour l'image fournie dans la variable `vis_image`. Inclure les images dans votre rapport . Commentaires sur les caractéristiques de l'entrée détectée par des unités spécifiques qui pourraient aider à classer l'image.

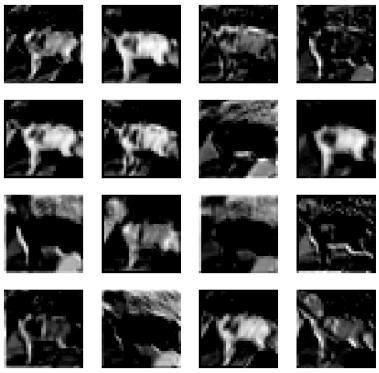

FIGURE 4 – Visualisation de quelques features map de la première couche de convolution

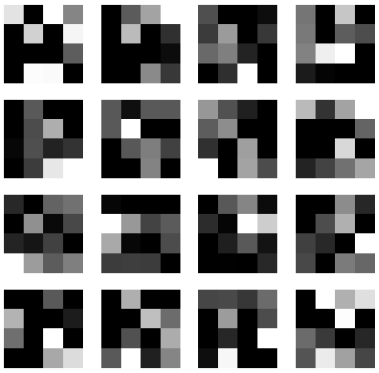

FIGURE 5 – Visualisation de quelques features map de la dernière couche de convolution

Observations :
Comme on peut le voir sur la première figure (celle de gauche), les caractéristiques de l'entrée détectées semblent être la silhouette de l'animal. En effet, on observe que le modèle cherche à comprendre les contours de la silhouette.
Concernant la deuxième figure, on observe la représentation de motifs partiels : comme on peut le voir la feature map est de dimension 4×4.

7. Considérez les fonctions d'activation: (i) ReLU et (ii) tanh, et les schémas d'initialisation des poids: (i) une Glorot uniforme (`init.xavier_uniform_`), (ii) Glorot normale (`init.xavier_normal_`) and (ii) He normal (`init.kaiming_normal_`). Effectuez une recherche par grille sur ces hyperparamètres. Signaler l'ensemble des hyperparamètres qui donnent la plus grande précision de validation, et fournir également les valeurs de l'exactitude et de la perte de validation.

   Le tableau suivant présente les meilleurs valeurs d'exactitudes (les plus élevées) et de perte (les plus petites) obtenus avec les données de validation. De plus, il présente les epochs à laquelle ces valeurs ont été atteintes.

Hyperparamètres	Plus grande exactitude de validation	epoch numero (pour l'exactitude)	Plus basse perte de validation	epoch numero (pour la perte)
ReLU et xavier_normal	0.8457	21	0.5585378231127051	3
ReLU et xavier_uniform	0.8436	23	0.6173864987832082	7
ReLU et kaiming_normal	0.8429	10	0.5945973456660404	2
Tanh et xavier_normal	0.7653	22	0.7785018392001526	7
Tanh et xavier_uniform	0.7671	18	0.7146756147282033	6
Tanh et kaiming_normal	0.758	21	0.7874026762533791	6

Le tableau suivant présente simplement les valeurs d'exactitude et de perte obtenus sur les données de validation au bout de 25 epochs.

Hyperparamètres	Exactitude de validation	Perte de validation
ReLU et xavier_normal	0.8138	1.0378625543811653
ReLU et xavier_uniform	0.8355	0.8641215140306497
ReLU et kaiming_normal	0.8329	0.9151209781441507
Tanh et xavier_normal	0.7558	1.3395114669316932
Tanh et xavier_uniform	0.7383	1.4788974568813662
Tanh et kaiming_normal	0.7562	1.3846288896814178

Observations :
Comme on peut le voir sur les deux tableaux, les paramètres qui offrent la plus grande précision de validation sont la fonction d'activation ReLU et le schéma d'initialisation des poids xavier_normal.

On peut observer sur la figure suivante l'évolution de la perte et de l'exactitude de la validation pour les différents hyperparamètres. Pour chaque combinaison d'hyperparamètres, les valeurs d'accuracy maximales et de loss minimales sont mis en valeur entourées par un cercle rouge sur les graphes.

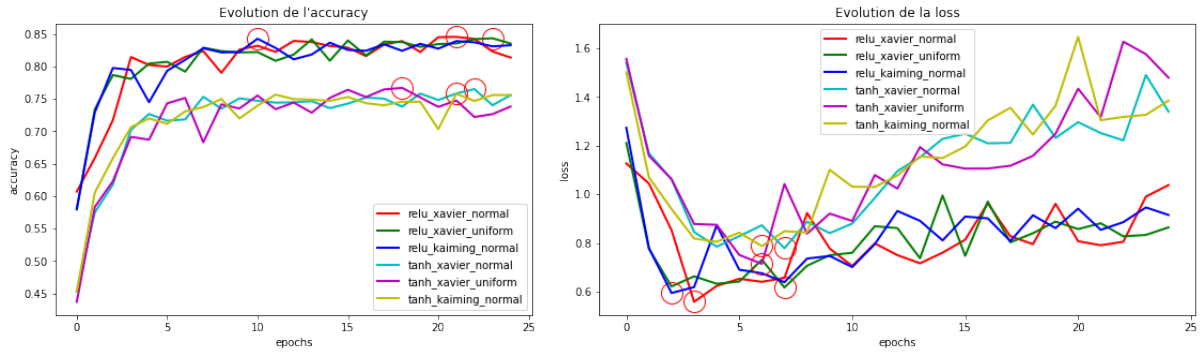

FIGURE 6 – Perte et exactitude de la validation pour les différents hyperparamètres

De plus, on remarque à travers les tableaux et les graphiques, une grande différence entre les mesures d'exactitude et de perte lorsqu'on compare la fonction d'activation ReLU à celle de Tanh.
En effet, l'accuracy est beaucoup mieux et la perte est beaucoup plus petite lorsqu'on utilise ReLU en comparaison à Tanh.
Concernant les choix d'initialisations, on ne remarque pas une différence aussi prononcée entre xavier_normal, xavier_uniform et kaiming_normal (lorsqu'on compare avec la différence observée entre ReLU et Tanh).

8. Construire un PLP avec des paramètres approximativement égaux comme le ResNet-18 et l'entraîner sur l'ensemble de données CIFAR-10. Inclure un tableau dans votre rapport comparant le MLP et le CNN – énumérer le nombre de paramètres, l'exactitude de la validation, la perte de validation et le temps de formation (nombre d'époques 25). Quel modèle fonctionne le mieux et pourquoi le pensez-vous ?

Observations :
Le modèle qui fonctionne le mieux est celui de ResNet-18 (celui codé avec Pytorch).
En effet, je pense que c'est grâce à l'optimisation des fonctions de la librairie torch et torch.nn notamment, comparé au MLP codé à partir de zéro en utilisant la librairie numpy.
PyTorch est conçu pour offrir une bonne flexibilité et des vitesses élevées pour la mise en oeuvre de réseaux de neurones profonds.
Si on souhaite comparer un peu plus, la principale différence entre un tenseur PyTorch et un tableau numpy est qu'un tenseur PyTorch peut s'exécuter sur la CPU ainsi que sur la GPU, ce qui n'est par le cas d'un tableau numpy, qui ne peut s'exécuter que sur la CPU.
Pour exécuter le tenseur PyTorch sur une GPU, il suffit simplement de convertir le tenseur en un type de données "CUDA" (ce qui a été fait durant ce devoir pratique).
Sachant que les calculs intensifs (utilisés notamment dans les réseaux de neurones profonds), sont plus rapides et optimisés en utilisant une GPU (en comparaison à la CPU), le modèle codé à l'aide de PyTorch est plus performant.