Telecom ENSEIRB-MATMECA

Spécialisation d'un CNN pour la détection de frelons asiatiques

Salma AZZOUZI
Cyril Hannier

Module : TS228
2021/2022

Table des matières

1 Introduction

L'apprentissage profond (ou deep learning) regroupe une famille de modèles d'apprentissage automatique, décrits sous forme de réseaux de neurones organisés en couches. Parmi ces modèles, on cite le réseau neuronal convolutif (CNN) et qui a des grandes applications dans la reconnaissance d'image. Dans ce Projet, nous serons amener à implémenter un réseau CNN afin de détecter la présence de frelon asiatique dans des images prises à l'entrée des ruches.

2 Détection de la présence de frelon :

Le problème de détection de frelon se ramène à un problème de classification où la classe 0 :"absence de frelon" et la classe 1 :"présence de frelons". On dispose donc d'une base de données annotée pour l'entraînement du CNN et une autre base pour le test.
En utilisant la classe "ImageFolder" combinée à la fonction "dataloader", on charge un minibatch de taille 4 dont les images ont été réduite d'un facteur de deux par la fonction "transforms.Resize". On affiche les quatre image comme suit :

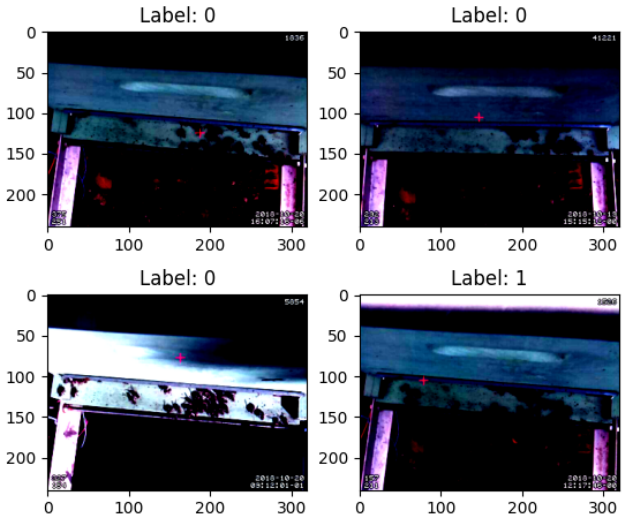


FIGURE 1 – Les images réduites d'un minibatch de taille 4

Pour l'architecture CNN, on utilise un ResNat18 dont la dernière couche a été remplacée par une couche linéaire (2x1). On vérifie ainsi la sortie du réseau en prenant à l'entrée le minibatch défini en-haut et on obtient à la sortie un tenseur $(4x2)$ où chaque colonne corresponde à une image et la ligne à une classe.

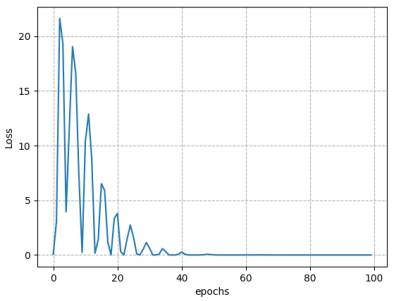
```
    ...: outputs = resnet(train_features)
    ...:
    ...: outputs_lab= torch.argmax(outputs,dim=1)
    ...: print ('La sortie du ResNet18 est ',outputs.size())
La sortie du ResNet18 est  torch.Size([4, 2])

In [15]:
```

FIGURE 2 – La sortie de RssNet18 pour une entrée de minibatch = 4

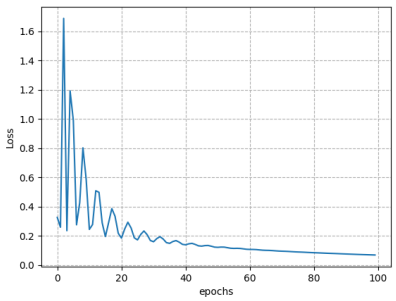
Pour l'entraînement du réseau, on choisit pour fonction de coût *Categorical Cross-Entropy* étant donné le problème est un problème de classification multi-class à simple label de sortie.

On lance l'apprentissage sur un seul minibatch en choisissant l'algorithme d'optimisation "Adam" et un pas d'apprentissage de $1e-3 < lr < 1e-1$, on affiche ensuite l'évolution de la fonction de coût au cours de l'entraînement (i.e epochs).

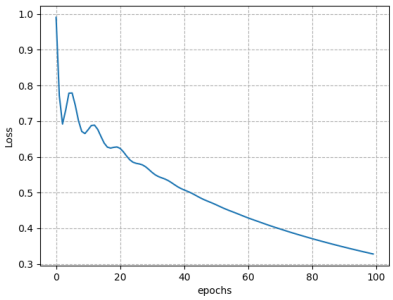
On remarque que pour un pas d'apprentissage de 0.1, la fonction de coup fluctue beaucoup puis converge vers 0.
Pour un pas d'apprentissage de 0.01, la fonction est moins fluctuante et converge plus lentement vers 0.
Enfin, pour un pas d'apprentissage de 0.001, la fonction ne fluctue plus et tend à converger très lentement vers 0.



(a) Lr= 0.1



(b) Lr= 0.01



(c) Lr= 0.001

FIGURE 3 – Allure de la fonction de coût en fonction du Learning rate

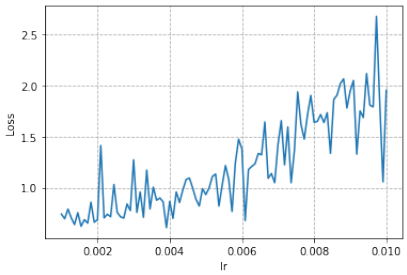


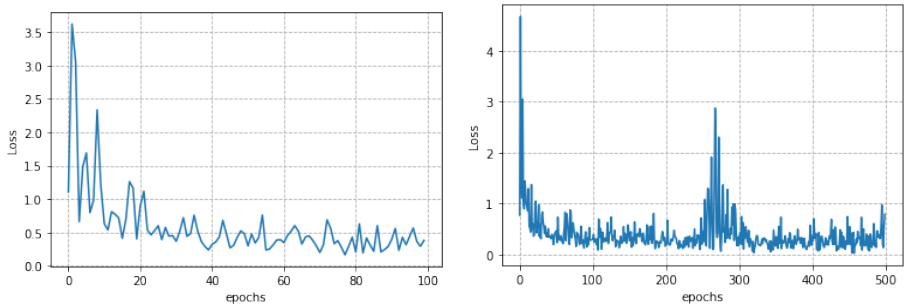
FIGURE 4 – La fonction du coût en fonction du pas Lr

Sur la figure 4, nous faisons varier le pas d'apprentissage entre 0.001 et 0.01 à chaque itération, la fonction du coût varie donc et il a tendance à croître tout en étant minimale pour un pas égal à 0.001.

On conclut que le choix du Learning rate sera définit en fonction du nombre d'epoch à notre disposition. En effet, si nous avons un nombre d'epoch plutôt faible (entre 40 et 100), nous opterons davantage pour un lr égal à 0.1 ou 0.01. A l'inverse, si nous avons à notre disposition davantage d'epoch, nous pourrons choisir un lr égale à 0.001.

3 Apprentissage du ResNet18 initialisé avec ImageNet :

Puur utiliser le CNN ResNet18 pré-entraîné, on met l'option *pretrained* = *True*.Ensuite, nous lançons l'entraînement en choisissant comme hyper-paramètres : epochs = 500, lr = 0.01, mini-batch = 25. Nous traçons par la suite l'évolution de la fonction du coût au cours de l'entraînement .



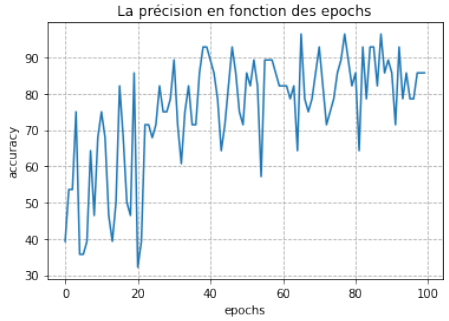
(a) Loss au cours de 100 itérations (b) Loss au cours de 500 itérations

FIGURE 5

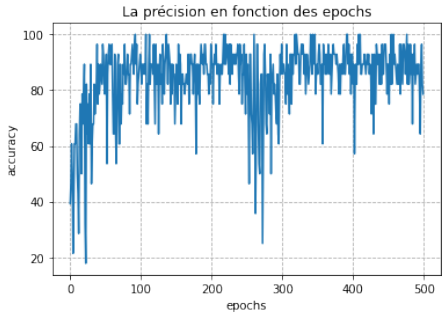
Pour les deux courbes, le coût a tendance de décroître tout en présentant des fluctuations.

A propos de la fonction de coût pour 500 itérations, on peut remarquer que la fonction de coût converge relativement rapidement vers 0 pour les 200 premiers epochs avant de diverger brutalement. Ensuite, la fonction de coût se remet à converger de manière progressive vers 0.5 environ. Nous pouvons faire certaines conclusions à partir de ces 2 courbes. En effet, on peut donc se contenter des 200 premiers epochs pour utiliser le ResNet18 afin d'éviter une divergence de la fonction de coût.

De la même façon, nous avons tracé l'évolution de la précision au cours de l'entraînement.



(a) Précision au cours de 100 itérations



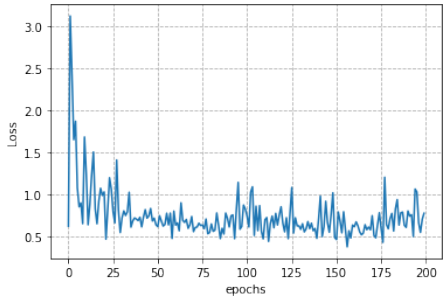
(b) Précision au cours de 500 itérations

FIGURE 6

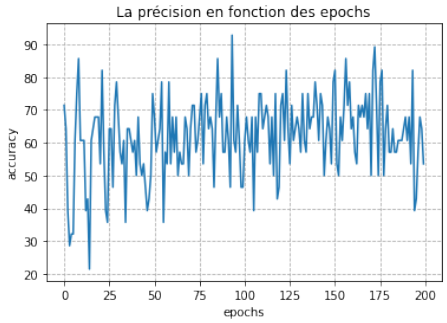
D'une manière assez cohérente au vu de la fonction de coût, la précision augmente au cours des epochs tout en ayant certaines fluctuations plus au moins importantes. On observe là aussi une nette variation de la précision au même nombre d'epochs que la fonction de coût. En effet, la précision chute très nettement là où la fonction de coût augmente fortement. Cette grande variation est observable pour un nombre d'epochs compris entre 250 et 280 environ. Après cette baisse, la précision remonte pour converger de nouveau vers 90 %

4 Apprentissage du ResNet18 initialisé aléatoirement :

Dans cette deuxième partie, nous avons initialisé le paramètre *pretrained* = *False* et de la même manière nous avons représentés l'évolution du coût et de la précision au cours des itérations. Nous avons utilisé ce réseau sur 200 epochs.



(a) Loss au cours de 200 itérations



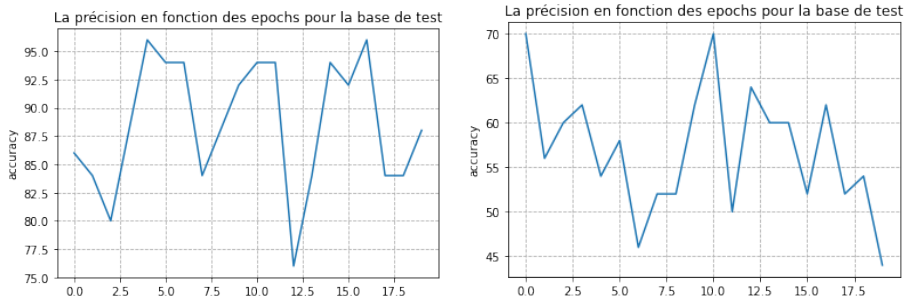
(b) Précision au cours de 200 itérations

FIGURE 7

On remarque que la fonction de coût converge assez rapidement de manière fluctuante vers 0.5 tandis que la précision semble beaucoup varier. En effet, la précision est très fluctante pour les plus petites valeurs d'epochs avant de fluctuer un peu moins par la suite. Pour autant, la précision moyenne semble moins elevée que precedemment, tournant autour de 60 % en moyenne.

5 Application des deux réseaux sur la base Test :

Après apprentissage des deux réseaux, on test leurs performances en utilisant à l'entrée des deux réseaux des minibatchs de taille = 50 et en comparant les prédictions à la sorties avec les labels corrects.La précision obtenu pour 20 minibatchs est représentée sur la figure suivante :



(a) Précision obtenue avec le réseau pré-entraînée

(b) Précision obtenue avec le réseau initialisé aléatoirement

FIGURE 8

Le premier réseau permet une précision supérieur à 77% et qui peut atteindre 98% .Contrairement au deuxième réseau pour lequel la précision peut atteindre 44% et ne dépasse pas 70%. Le premier réseau obtenu avec pré-entraînement est donc nettement supérieur en terme de précision par rapport au réseau initialisé aléatoirement.

6 Conclusion

L'utilisation d'un réseau Resnet18 permet la détection de frelons. A travers ce TP, nous avons mis en place ce réseau tout en initialisant de deux manières différentes afin d'effectuer des comparaisons. La première initialisation a consisté à utiliser un réseau pré-entraîné alors que la deuxième méthode se base sur une initialisation aléatoire. Les résultats obtenus nous ont permis de mettre en avant la plus grande efficacité du réseau basé sur une initialisation avec un réseau pré-entraîné.