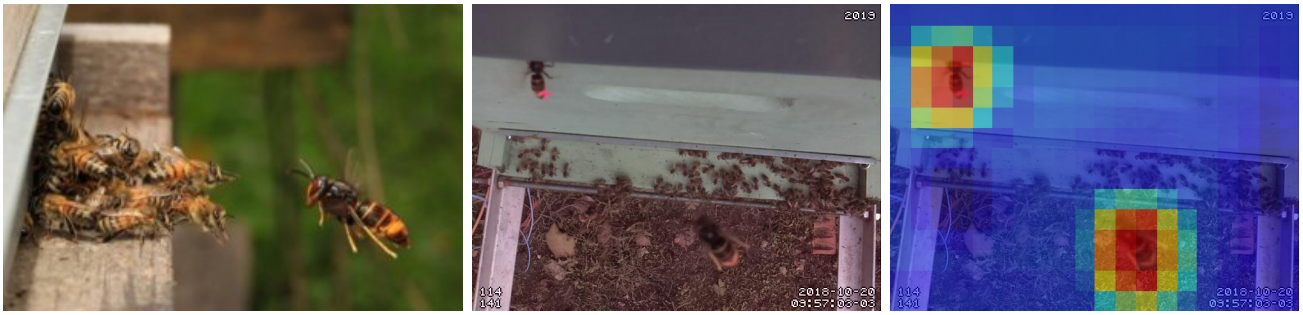
TP : Spécialisation d'un CNN pour la détection de frelons asiatiques

Guillaume Bourmaud

1 Contexte du TP

Le frelon asiatique est un prédateur d'abeilles très invasif qui serait arrivé en France par bateau dans une cargaison en provenance d'Asie au début des années 2000. Alors que cet envahisseur extermine petit à petit les abeilles de leurs ruchers, les apiculteurs n'ont que peu de moyens pour les combattre. Ce TP s'inscrit dans ce contexte et a pour objectif de développer une méthode permettant de détecter automatiquement la présence de frelons asiatiques dans une image issue d'une caméra montée à l'entrée d'une ruche (voir Figure 1).



(a) (b) (c)

Fig. 1: (1a) Un frelon asiatique attaquant des abeilles à la sortie d'une ruche. (1b) Un exemple d'image issue d'une caméra montée sur une ruche. (1c) Un exemple de prédiction de la localisation des frelons présents dans l'image.

2 Description du TP

2.1 Détecter la présence de frelons

L'objectif du TP est de spécialiser un réseau de neurones à convolution (CNN), en utilisant la bibliothèque PyTorch, afin de détecter la présence de frelons dans une image. Il s'agit donc d'un problème de classification (classe 0 : "absence de frelon" , classe 1 : "présence de frelons") pour lequel vous disposez d'une base de données annotées (une petite partie de cette base est disponible via ce lien, le reste de la base est déjà téléversé sur Google Drive comme vous le verrez par la suite).

Pour lancer un terminal ayant accès à un environnement où PyTorch (et d'autres paquets importants) sont installés, aller dans "Applications", puis "Autre", puis "conda_pytorch". Dans le terminal, taper "spyder &" pour lancer l'environnement

de développement.
Au cours du TP, les étapes suivantes devront être réalisées :

1. Prendre plusieurs minutes pour regarder les données.

2. Afin que PyTorch puisse efficacement gérer cette base de données, il est nécessaire d'utiliser une classe de PyTorch qui se chargera de définir un "dataset", par exemple en utilisant la classe "ImageFolder". Ainsi vous pourrez utiliser les fonctionnalités de PyTorch (notamment le "dataloader") pour générer et charger des "minibatches". Dans un premier temps, il est conseillé de réduire la résolution des images, par exemple d'un facteur deux, pour réduire le temps de calcul (fonction "torchvision.transforms.Resize"). Tester le "dataloader" en générant un "minibatch" et afficher des images de ce dernier.

3. Concernant l'architecture du CNN, il est conseillé d'utiliser un ResNet18 (dont l'architecture a été étudiée en cours). Cette architecture est disponible dans PyTorch (lire attentivement la documentation), ainsi vous n'avez pas à réaliser son implémentation. Il s'agit simplement de remplacer la dernière couche du réseau pour que ce dernier prédise uniquement deux scores au lieu de mille. Tester le réseau en mettant un minibatch en entrée du ResNet18 et vérifier la dimension du tenseur en sortie. Un exemple de code est fourni Fig.2, attention en fonction de la version de PyTorch utilisée, la première ligne peut nécessiter un changement en remplaçant 'pretrained=True' par 'weights='IMAGENET1K_V1'.

```
# Instancie un ResNet18 et télécharge les valeurs des paramètres issus d'un entraînement sur ImageNet
resnet = torchvision.models.resnet18(pretrained=True)

# Désactive les gradients des paramètres, dans le cas où on souhaite geler les paramètres du réseau
for param in resnet.parameters():
    param.requires_grad = False

# Remplace la dernière couche "Fully Connected" du ResNet18 par une autre "Fully Connected"
resnet.fc = nn.Linear(resnet.fc.in_features, 100)

# Test du réseau
images = torch.randn(64, 3, 224, 224)
outputs = resnet(images)
print (outputs.size())     # (64, 100)
```

Fig. 2: Exemple de code permettant de charger un ResNet18 pré-entraîné et de modifier la dernière couche.

Attention : comme nous l'avons vu en cours, il faut que les statistiques des images placées en entrée du ResNet18 soient similaires à celles qui ont été utilisées pour pré-entraîner ce ResNet18 sur ImageNet. Ces statistiques sont disponibles dans la documentation du ResNet18. Il faut donc rajouter une transformation de la forme "torchvision.transforms.Normalize(mean=[..., ..., ...], std=[..., ..., ...])" au "dataloader".

4. Choisir une fonction de coût et justifier ce choix.

5. Choisir un algorithme d'optimisation (par exemple Adam) et un pas d'apprentissage (pour commencer entre 1e-2 et 1e-3). Avant de lancer un apprentissage complet, il est conseillé de déboguer votre code en effectuant un apprentissage sur un seul minibatch : tant que votre code n'est pas capable de faire baisser le coût d'apprentissage proche de zéro, cela ne sert à rien de passer sur la base de données complète. Prenez le temps de réaliser des affichages pour contrôler l'évolution de l'apprentissage au cours du temps.

6. Vous pourriez maintenant lancer un apprentissage complet, mais les itérations sont lentes car l'exécution se fait sur CPU. Les ordinateurs n'ayant pas de GPU compatible, vous devrez utiliser un GPU à distance mis à disposition par Google Colab **en créant un compte spécialement pour le TP**. Pour accélérer cette partie, la base de données à déjà était mise sur Google Drive ici. Après avoir cliqué sur ce lien, le dossier "imgs_frelon" apparaît dans votre drive dans l'onglet "Partagés avec moi". Il suffit alors de faire un clic droit sur ce dossier et de sélectionner "Ajouter un raccourci dans Drive" pour que ce dossier soit visible dans votre drive de la même manière que si vous l'aviez vous-même téléversé. Une fois l'environnement Colab maîtrisé (importation des données dans "Google Colab", utilisation du GPU, etc.) il est demandé de fournir le résultat d'au moins deux apprentissages :

   - Un premier apprentissage où les valeurs des paramètres du ResNet18 obtenus après un apprentissage sur ImageNet sont utilisées comme initialisation. Il s'agit donc d'une spécialisation du ResNet18 pour la détection de frelons. Information sur le résultat attendu : vous devriez obtenir un taux de bonne classification supérieur à 90%.
   - Un second apprentissage où **tous les paramètres** du ResNet18 sont initialisés aléatoirement. Il ne s'agit donc pas d'une spécialisation du ResNet18 pour la détection de frelons mais d'un apprentissage classique. Information sur le résultat attendu : vous devriez obtenir un taux de bonne classification supérieur à 90%.
   - Comparer les résultats des deux apprentissages précédents. Les courbes d'apprentissage sont-elles identiques ? Les deux méthodes permettent-elles d'atteindre les mêmes performances ? La spécialisation présente-t-elle des avantages ? Si oui, lesquels ? Si non, à votre avis pour quelles raisons ?
   - Pour information, grâce à l'utilisation du GPU, une "epoch" dure moins de trois minutes (à l'exception de la première "epoch" qui est généralement plus lente probablement car les données doivent être transférée depuis le "Google Drive" vers "Google Colab"). En fonction de votre avancement, vous pouvez également tester des techniques d'augmentation de données, différents algorithmes d'apprentissage, différentes architectures, etc.

2.2 Localiser les frelons (optionnel)

Après avoir spécialisé un CNN, ce dernier est capable de prédire si des frelons sont présents dans l'image ou non. Cependant, en cas de détection de frelons, le réseau ne nous dit pas où se situent ces frelons dans l'image. Une manière d'obtenir la localisation consiste à inspecter la dernière couche du ResNet18 comme expliqué dans l'article Learning Deep Features for Discriminative Localization. Il est demandé de réaliser l'implémentation de cette technique de localisation. Un exemple de résultat est présenté sur la figure .1c.