Semestre 9 en Intelligence Artificielle

Analyse de Vidéos

Projet 2 : Aide à la préhension par des outils de vision par ordinateur aux amputés des membres supérieurs, porteurs de neuroprothèses

---

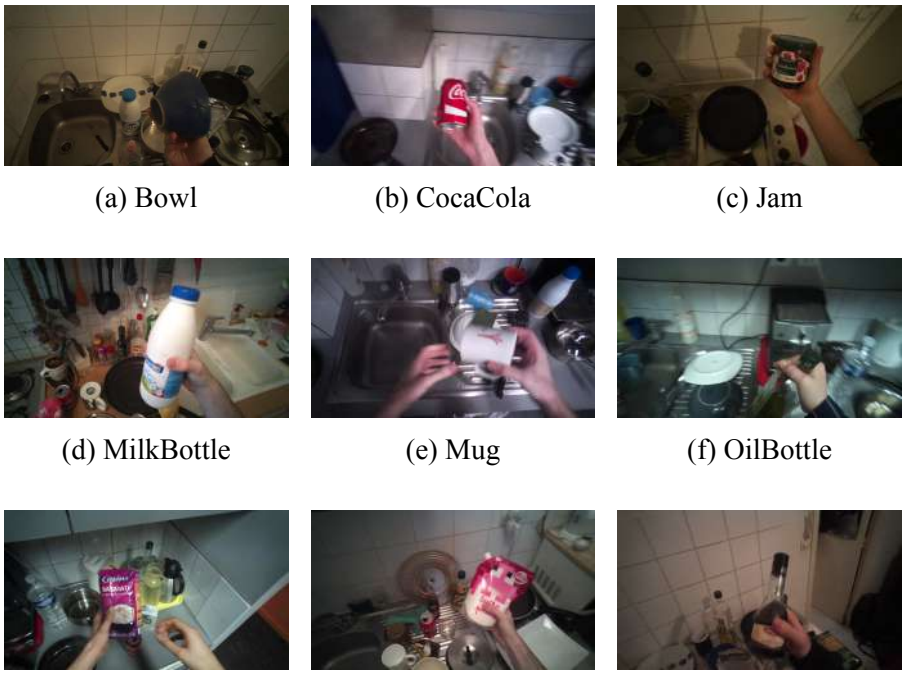
Auteurs
Bonhomme Romain
Pierre Romain

Professeur
Jenny Benois-Pineau
Assistant de Projet
Boris Mansencal

Année Académique 2024-2025

1 Descriptions des données

Le dataset utilisé est une version allégée du dataset *Grasping in the Wild (GITW)* [1]. Ce dataset contient des vidéos (au format *'.mp4'*) de personnes entrant dans différentes cuisines (7 différentes) et saisissant un objet déterminé (16 objets différents dans le dataset complet, 9 dans notre cas réduit). Les 9 classes présentes dans le dataset utilisé sont affichées sur la FIG.1.



(a) Bowl (b) CocaCola (c) Jam

(d) MilkBottle (e) Mug (f) OilBottle

(g) Rice (h) Sugar (i) VinegarBottle

Figure 1 – Présentation des 9 classes présentes dans le dataset utilisé.

Les autres données présentes sont des cartes de saillance (au format *.png*), réalisée par une détection automatique de l'objet avec une méthode de eye tracker. Ces cartes sont donc une image ne contenant que des pixels noirs, sauf pour la zone de l'image contenant l'objet souhaité (s'il est présent sur la frame de la vidéo). Un exemple de carte de saillance avec sa frame associée, peut être observé à la FIG.2.

(a) Image originale (b) Carte de saillance

Figure 2 – Frame de la vidéo et carte de saillance associée lors de la détection d'un bol.

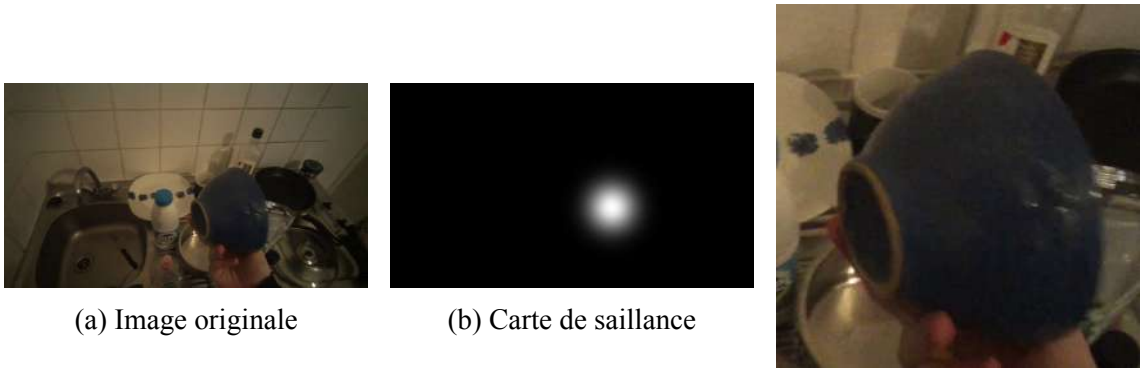
1. `https://universe.roboflow.com/iwrist/grasping-in-the-wild`

2 Description du Code

Dans le cadre de ce projet, nous explorons l'utilisation des données de fixation issues d'un eye tracker pour effectuer une classification d'objets à partir de vidéos. L'objectif est d'utiliser les points de fixation pour identifier les zones d'intérêt dans une séquence vidéo et de les associer aux objets correspondants.

2.1 Extracteur d'images

La première brique de notre projet est donc un extracteur d'images qui va venir traiter chaque vidéo *'.mp4'* et la décomposer en frames, au format *'.png'* tout en préservant le label, i.e. la classe à laquelle la vidéo appartient. Ensuite, à l'aide de la carte de saillance qui lui est attribuée, nous redimensionnons l'image autour de l'objet d'intérêt. Cette transformation peut être observée à la FIG.3.



(a) Image originale (b) Carte de saillance (c) Image redimensionnée

Figure 3 – Exemple d'application de l'extracteur d'image

2.2 Modèles et entraînements

Notre projet est implémenté en *PyTorch* et contient 2 modèles, à savoir un *ResNet18* [2] et un*EfficientNetB0* [3]. Les modèles sont pré-entrainés sur *ImageNet* et fine-tunés sur notre dataset.

ResNet18 est un réseau de neurones convolutionnel composé de 18 couches. Son architecture repose sur des blocs résiduels, chaque bloc comportant deux couches convolutives et une connexion d'identité ajoutée à la sortie. Cela permet de résoudre le problème de la dégradation des performances dans les réseaux très profonds. Le modèle commence par une couche convolutive de 7×7, suivie d'un max pooling. Il est ensuite constitué de quatre groupes de blocs résiduels (64, 128, 256, 512 canaux). Enfin, une couche de Global Average Pooling est appliquée avant une couche fully connected pour la classification.

2. `https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html`

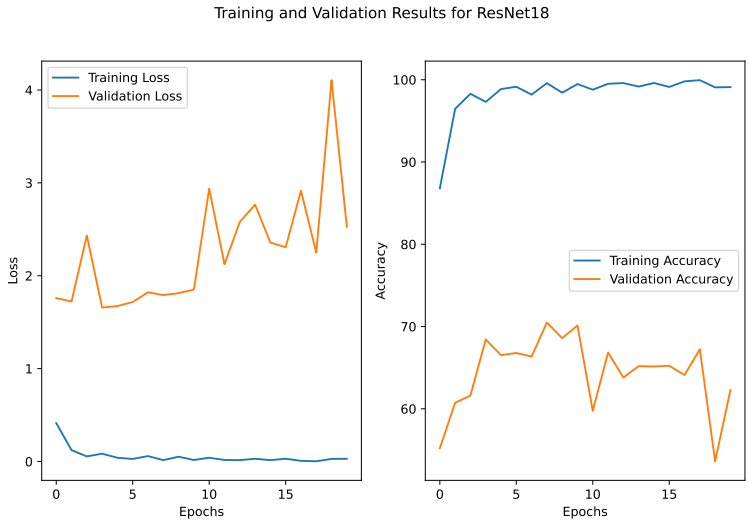
3. `https://pytorch.org/vision/master/models/generated/torchvision.models.efficientnet_b0.html`

EfficientNetB0 est un modèle de réseau neuronal convolutionnel qui utilise une approche d'optimisation appelée "compound scaling" pour équilibrer l'augmentation de la profondeur, de la largeur et de la résolution de l'image d'entrée. Il repose sur des blocs convolutifs de type MBConv, utilisant des convolutions depthwise et pointwise, et une activation Swish. Le modèle commence par une couche convolutive standard, suivie de plusieurs blocs MBConv, et termine par une couche de classification. *EfficientNetB0* est conçu pour être à la fois léger et précis, offrant un excellent compromis entre efficacité et performance.

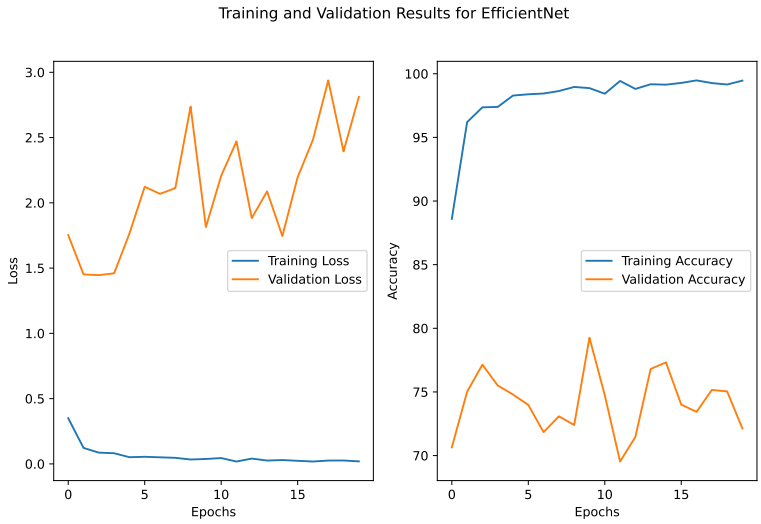
Pour l'entraînement, nous avons écrit une boucle d'entraînement ainsi qu'une boucle de validation. Les fine-tunings ont été fait sur 20 epochs tout en prenant le soin d'enregistrer la consommation à l'aide du module *codecarbon*.

3 Analyse des Résultats

3.1 Fonctions de Perte et Accuracies



(a) ResNet18



(b) EfficientNetB0

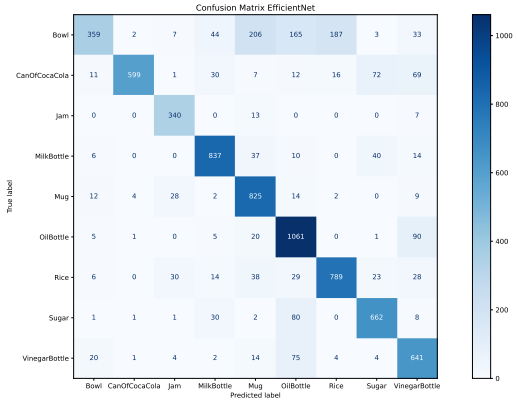
Figure 4 – Evolution de la fonction de perte (*CrossEntropy*) (sur les graphiques de gauche) et de l'accuracy de classification (sur les graphiques de droite) pour les modèles *ResNet* et *EfficientNetB0* sur les ensembles d'entraînement (en bleu) et de validation (en orange).

En regardant la FIG.4, on remarque que les 2 modèles ont des comportements semblables. En effet, dans les 2 cas, la loss d'entraînement atteint rapidement une valeur proche de 0 tandis que celle de validation ne décroît pas et tend plutôt à augmenter avec le nombre d'epochs. En terme d'accuracy, les 2 modèles atteignent une performance maximale avant 10 epochs, et semblent ensuite être en overfitting, au vu de la diminution de l'accuracy. En terme de comparaison de performances, il semble clair que le modèle *EfficientNetB0* est

plus performant que le *ResNet18*, étant donné qu'il atteint une précision maximale autour des 80%, tandis que l'autre est limité autour des 70%.

3.2 Matrices de Confusion

(a) ResNet18



(b) EfficientNetB0

Figure 5 – Matrices de Confusion de classification des modèles *ResNet18* (matrice du haut) et *EfficientNetB0* (matrice du bas). L'axe vertical représente les labels réels et l'axe horizontal représente les labels prédits par les modèles.

En regardant la FIG.5, on peut remarquer plusieurs choses. Premièrement, les 2 modèles prédisent majoritairement bien toutes les classes, sauf la classe 'Bowl' (qui contient plus d'exemples mal classifiés que d'exemples bien classifiés). Ensuite, le modèle *ResNet18* classifie la classe 'Oil Bottle' au mieux (avec 83.0092% d'exemples bien classifiés) tandis que le modèle *EfficientNetB0* classifie la classe 'Jam' au mieux (avec 94.4444% d'exemples bien classifiés). Enfin, l'observation précédente tend à confirmer les observations faites sur les graphiques de la FIG.4. En effet, le modèle *EfficientNetB0* surperforme le *ResNet18* pour toutes les classes, en classifiant plus d'exemples correctement.

3.3 Analyse de CodeCarbon

Résultats de l'entraînement sur 20 époques de **ResNet18** :

- Pays : France
- Région : Nouvelle-Aquitaine
- OS : Linux 6.12
- Python : 3.12
- CPU : Intel Xeon 1270
- Puissance CPU : 40 W
- GPU : Nvidia 3060
- Puissance GPU : 70 W
- RAM : 32 GB
- Puissance RAM : 11 W
- Durée : 2h 20m
- Energies consommées : 0.34 kWh
- Emissons de CO2 : 0.02 kg.CO2eq

Résultats de l'entraînement sur 20 époques de **EfficientNetB0** :

- Pays : France
- Région : Nouvelle-Aquitaine
- OS : Linux 6.8
- Python : 3.12
- CPU : Intel Xeon 1270
- Puissance CPU : 40 W
- GPU : Nvidia 3060
- Puissance GPU : 76 W
- RAM : 32 GB
- Puissance RAM : 11 W
- Durée : 2h 57m
- Energies consommées : 0.48 kWh
- Emissons de CO2 : 0.03 kg.CO2eq

L'analyse montre que l'entraînement de *ResNet18* ($2h20m$, 0.34 kWh, 0.02 kg.CO2eq) consomme moins d'énergie et génère moins d'émissions que *EfficientNetB0* ($2h57m$, 0.48 kWh, 0.03 kg.CO2eq). *EfficientNetB0*, bien que plus énergivore avec une puissance GPU de $76W$ contre $70W$ pour *ResNet18*, a atteint des performance significativement meilleurs. Les émissions de CO2 restent toutefois faibles grâce au mix énergétique peu carboné en Nouvelle-Aquitaine.