



Semestre 9 en Intelligence Artificielle

Analyse de Vidéos

**Projets 2 et 3 : 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.

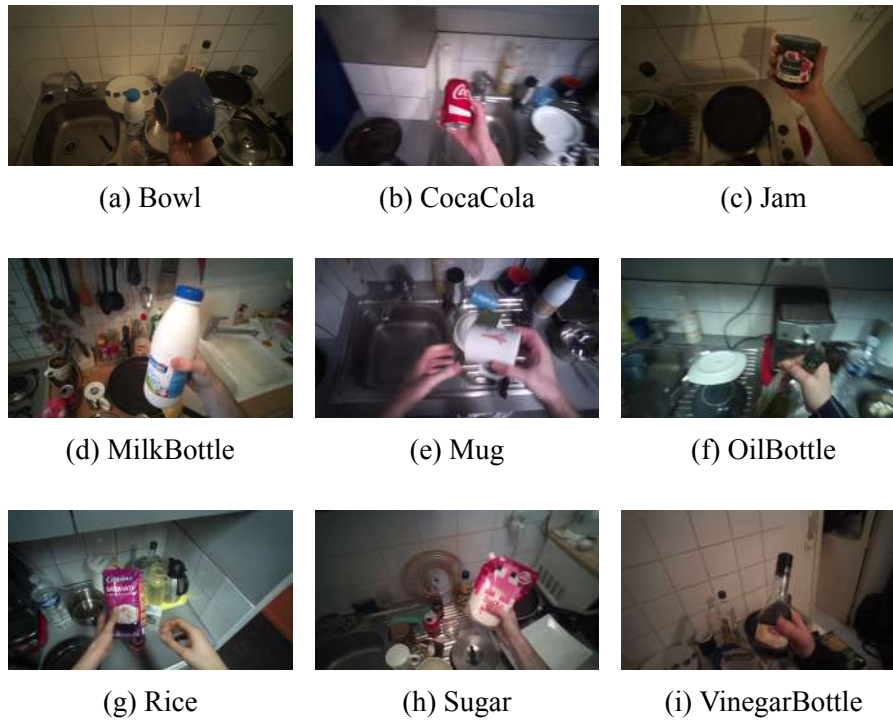


Figure 1 – Présentation des 9 classes présentes dans le dataset utilisé.

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.

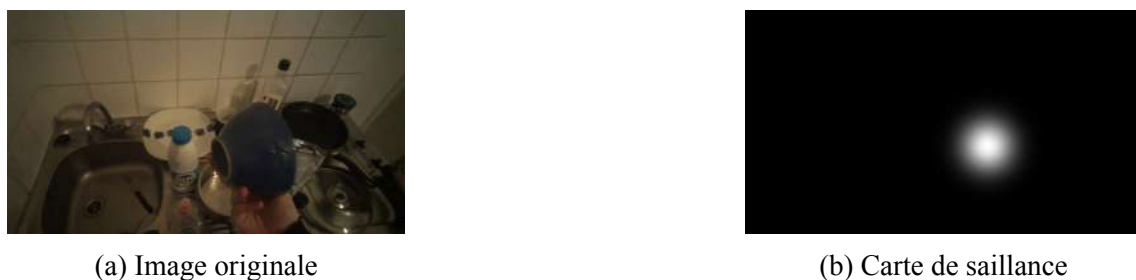


Figure 2 – Frame de la vidéo et carte de saillance associée lors de la détection d'un bol.

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.

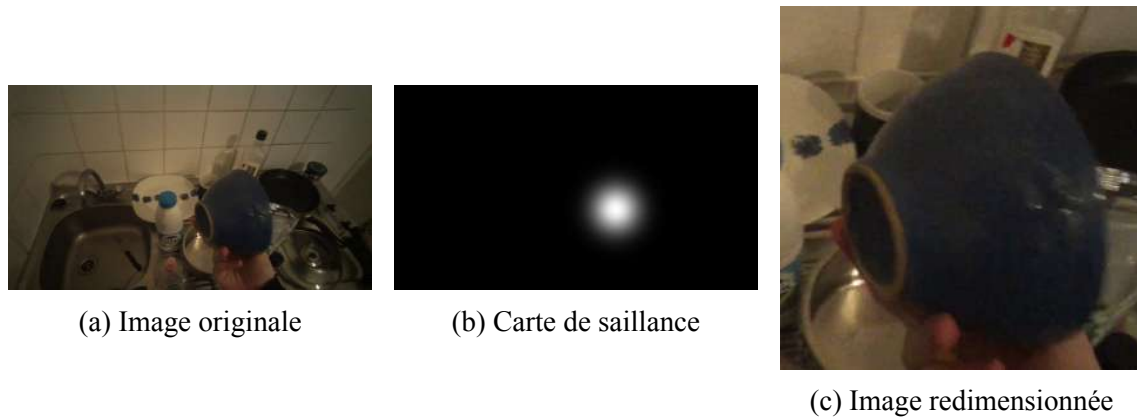


Figure 3 – Exemple d'application de l'extracteur d'image

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-entraînés 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>

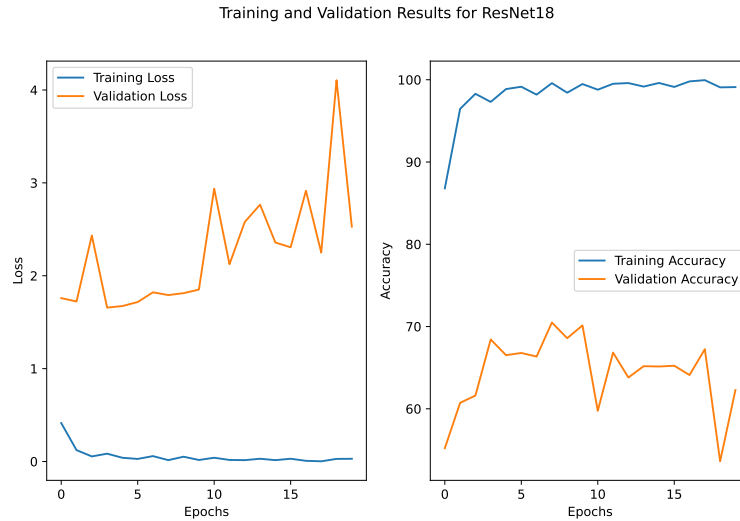
3. https://pytorch.org/vision/master/models/generated/torchvision.models.efficientnet_b0.html

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.

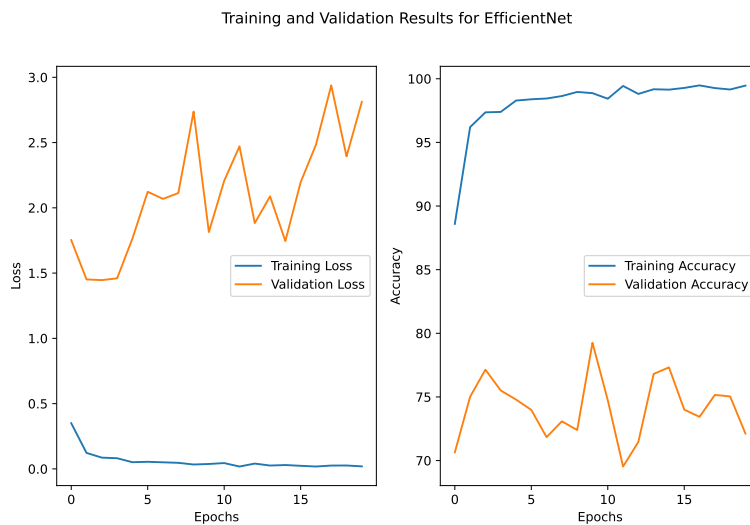
Pour l'entraînement, nous avons écrit une boucle d'entraînement ainsi qu'une boucle de validation. Les fine-tunings ont été fait sur 20 epochs tout en prenant le soin d'enregistrer la consommation à l'aide du module *codecarbon*.

3 Analyse des Résultats

3.1 Fonctions de Perte et Accuracies



(a) ResNet18



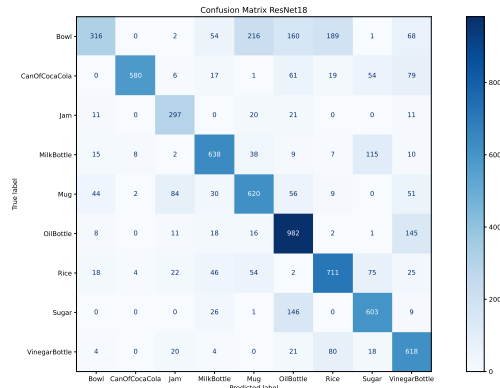
(b) EfficientNetB0

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).

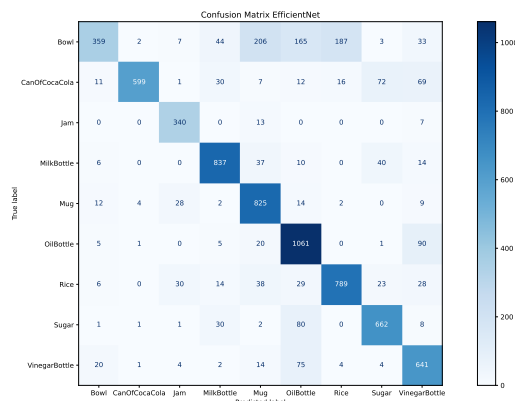
En regardant la FIG.4, on remarque que les 2 modèles ont des comportements semblables. En effet, dans les 2 cas, la loss d'entraînement atteint rapidement une valeur proche de 0 tandis que celle de validation ne décroît pas et tend plutôt à augmenter avec le nombre d'époques. En terme d'accuracy, les 2 modèles atteignent une performance maximale avant 10 époques, 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) et atteint 31.4115% d'exemples bien classifiés sur la classe 'Bowl' tandis que le modèle *EfficientNetB0* classifie la classe 'Jam' au mieux (avec 94.4444% d'exemples bien classifiés) et atteint 35.6859% d'exemples bien classifiés sur la classe 'Bowl'. 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
- Emissions 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
- Emissions 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 performances significativement meilleures. Les émissions de CO2 restent toutefois faibles grâce au mix énergétique peu carboné en Nouvelle-Aquitaine.

4 Améliorations

4.1 Modifications Effectuées

4.1.1 Augmentation et Normalisation des Données

Afin d'améliorer les performances de nos modèles, nous avons mis en œuvre deux stratégies principales de prétraitement :

La normalisation des données a été raffinée en utilisant la moyenne et l'écart-type du jeu de données de pré-entraînement, en cohérence avec les modèles préentraînés. Cette approche permet de garantir une mise à l'échelle cohérente des données d'entrée.

Nous avons intégré une stratégie d'augmentation de données pour enrichir notre ensemble d'apprentissage et favoriser la généralisation du modèle. Les transformations appliquées incluent :

- Recadrage aléatoire redimensionné (224 pixels, échelle 0.8-1.0)
- Retournement horizontal aléatoire (probabilité 0.5)
- Rotation aléatoire (± 15 degrés)
- Variations colorimétriques :
 - Luminosité ($\pm 20\%$)
 - Contraste ($\pm 20\%$)
 - Saturation ($\pm 20\%$)
 - Teinte ($\pm 10\%$)
- Conversion aléatoire en niveaux de gris (probabilité 0.1)

La normalisation finale utilise les paramètres standards :

$$\text{Normalize}(\mu = [0.485, 0.456, 0.406], \sigma = [0.229, 0.224, 0.225])$$

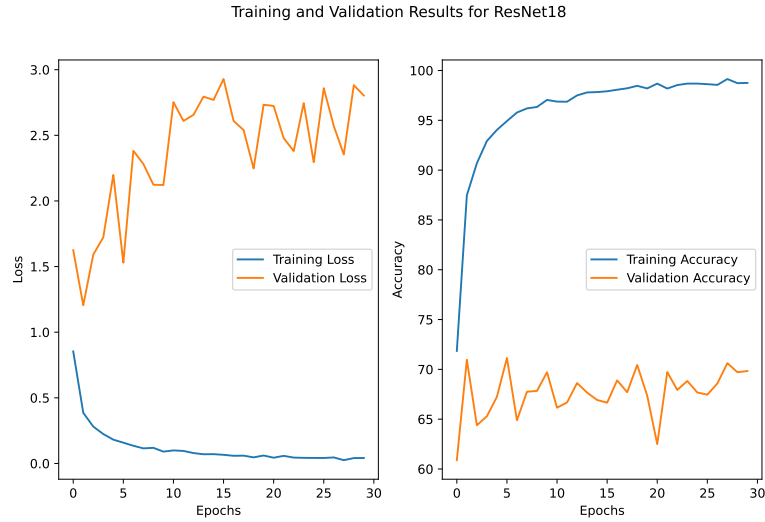
Ces techniques visent à renforcer la robustesse et la capacité de généralisation de notre modèle.

4.1.2 GradCam

Afin de visualiser les relations que déterminent nos modèles nous utilisons la méthode **GradCam** sur la dernière couche. Cette méthode utilise les gradients de la couche de sortie pour générer une carte de chaleur qui met en évidence les régions de l'image les plus importantes pour la classification.

Plus précisément, GradCam calcule le gradient de la classe prédite par rapport aux cartes de caractéristiques de la dernière couche convolutive. Ces gradients sont ensuite moyennés spatialement et utilisés comme poids pour produire une carte de localisation grossière qui souligne les zones influentes pour la décision du modèle.

4.2 Résultats



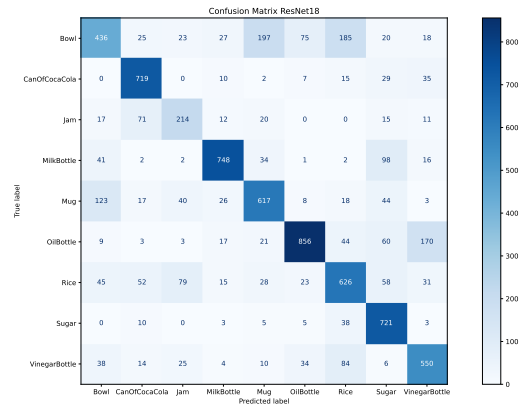
(a) ResNet18



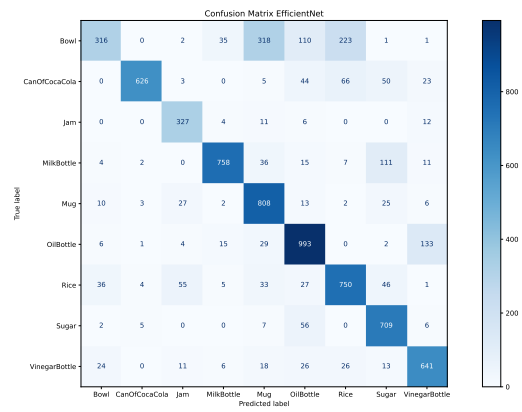
(b) EfficientNetB0

Figure 6 – 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).

Là encore, les modèles overfit rapidement avec la loss de validation qui augmente et celle de training qui diminue. Les résultats sont légèrement meilleurs qu'au TP2 pour Resnet18 mais rien de très significatif. Par contre ceux d'EfficientNet sont même inférieures. Les évolutions apportées n'ont pas forcément été bénéfiques. Cela peut venir du fait que le rognage à l'aide de cartes de saillance est imprécis et ces données de mauvaise qualité ne permettent pas à la data augmentation de réellement alimenter le jeu d'entraînement d'exemples pertinents.



(a) ResNet18



(b) EfficientNetB0

Figure 7 – 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.7, on remarque d’abord qu’à nouveau, l’ensemble des objets sont bien classifiés sauf pour la classe ‘Bowl’ qui comme précédemment n’a même pas la moitié de ses éléments bien classifiés. Concernant la comparaison des méthodes, le modèle *ResNet18* classe la classe ‘Sugar’ au mieux (avec 91.85% d’exemples bien classifiés) et atteint 43.3399% d’exemples bien classifiés sur la classe ‘Bowl’ tandis que le modèle *EfficientNetB0* classe la classe ‘Jam’ au mieux (avec 90.83% d’exemples bien classifiés) et atteint 31.4115% d’exemples bien classifiés sur la classe ‘Bowl’.

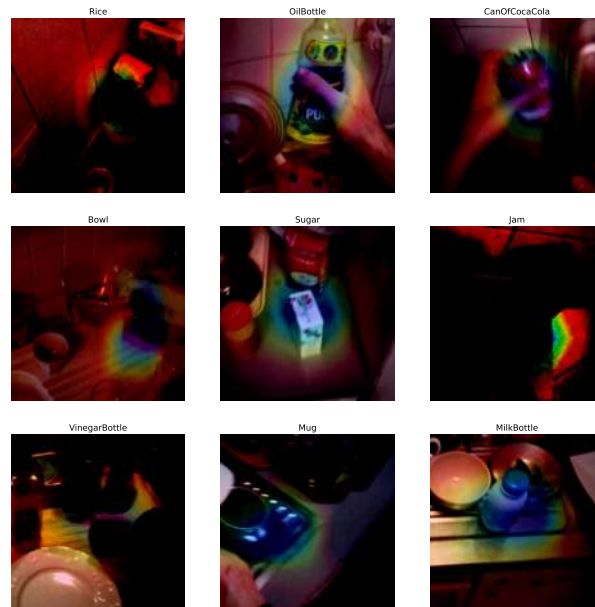
La Table1 montre la précision de classification pour les 4 modèles entraînés, pour la pire classe (la classe ‘Bowl’ dans tous les cas), la meilleure classe et une précision de classification moyenne. De manière assez surprenante, les améliorations amenées n’ont absolument pas le même effet sur les 2 modèles. En effet, le modèle *ResNet18* obtient de meilleures performances pour les 3 valeurs calculées, tandis que le modèle *EfficientNetB0* a quant à lui régressé dans les 3.

	Pire Classif (%)	Meilleure Classif (%)	Classif Moyenne (%)
<i>ResNet18</i> TP2	31.4115 (‘Bowl’)	83.0092 (‘Oil Bottle’)	70.4901
<i>ResNet18</i> TP3	43.3399 (‘Bowl’)	91.8471 (‘Sugar’)	72.0930
<i>EfficientNetB0</i> TP2	35.6859 (‘Bowl’)	94.4444 (‘Jam’)	80.3179
<i>EfficientNetB0</i> TP3	31.4115 (‘Bowl’)	90.8333 (‘Jam’)	69.6098

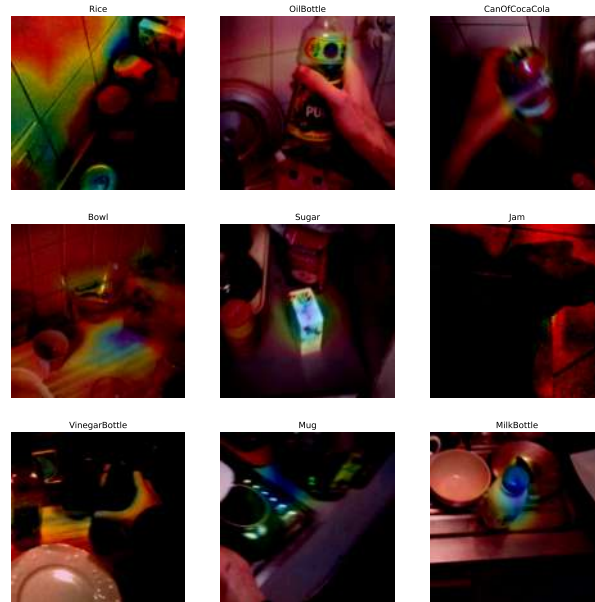
Table 1 – Précision de classification des différents modèles entraînés.

4.3 Expliquabilité

L’utilisation de GradCam nous a permis d’observer les points d’intérêt du modèle à un instant donné sur la dernière couche de convolution. Cette représentation est observable à la FIG.8. Comme attendu, ces zones d’intérêts se situent majoritairement sur les objets en question. Toutefois, on peut remarquer que *ResNet* semble englober la totalité de l’objet là où *EfficientNet* semble se concentrer sur une partie plus spécifique de l’objet. Cela peut nous laisser penser que les modèles se concentrent sur les objets et que l’entraînement a eu un comportement logique. Mais cela reste une interprétation de la dernière couche à l’aide de GradCam et ne permet pas d’affirmer quoi que ce soit.



(a) ResNet18



(b) EfficientNetB0

Figure 8 – Résultats GradCam pour les modèles *ResNet* et *EfficientNetB0*.