

# ÉCOLE CENTRALE LYON

MOD 3.2 - APPRENTISSAGE PROFOND  
ET INTELLIGENCE ARTIFICIELLE : UNE INTRODUCTION  
TD  
COMPTE RENDU

---

## TD2 - IA embarquée

---

*Elèves :*  
Akilhoussen ONALY  
Antony PARODI

*Enseignant :*  
Mr. Thomas DUBOUDIN

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Utilisation d'un modèle CNN pré-entraîné</b>	<b>2</b>
2.1	Étude du code de <i>run1.py</i> et des résultats obtenus . . . . .	2
2.2	Modification du code pour utiliser la caméra . . . . .	2
2.3	Inférence sur la carte avec les images de la caméra . . . . .	3
2.4	Inférence sur l'ordinateur avec les images de la caméra . . . . .	4
2.5	Comparaison des résultats avec d'autres réseaux de neurones . . . . .	5
<b>3</b>	<b>Transfer Learning</b>	<b>5</b>
3.1	Étude du code de <i>run3.py</i> et des résultats obtenus . . . . .	6
3.2	Comparaison de la durée d'entraînement de la carte par rapport à l'ordinateur	8
3.3	Inférence sur la carte avec les images de la caméra . . . . .	9
3.4	Inférence sur l'ordinateur avec images de la caméra . . . . .	11
<b>4</b>	<b>Conclusion</b>	<b>14</b>

## 1 Introduction

L'objectif de ce TD est de découvrir le framework de deep learning PyTorch et de mettre en oeuvre un modèle de classification d'images sur un dispositif embarqué qui est dans notre cas la carte NVIDIA Jetson Nano équipée d'une caméra.

## 2 Utilisation d'un modèle CNN pré-entraîné

Dans la première partie, nous allons utiliser un modèle pré-entraîné (Resnet50) pour essayer de classifier des images qui proviennent d'une caméra. Le set de labels utilisés est 'imagenet-simple-labels', qui contient un nombre important de labels capables de décrire beaucoup d'objets différents.

### 2.1 Étude du code de *run1.py* et des résultats obtenus

Le code commence par importer une image, ici l'image d'un Golden Retriever, et les labels nécessaires. Une fois que cela est fait, il transforme l'image pour qu'elle soit utilisable par le modèle CNN pré-entraîné de PyTorch, charge ce dernier, et lui demande de prédire ce que c'est. Le résultat obtenu est le suivant :

```
embedded@abos io-desktop:~/onaly$ date
mercredi 23 octobre 2019, 19:00:47 (UTC)
embedded@abos io-desktop:~/onaly$ python3
Predicted class is: Golden Retriever
embedded@abos io-desktop:~/onaly$ █
```

FIGURE 1 – Reconnaissance du chien

La classe prédictée est bel et bien correcte.

### 2.2 Modification du code pour utiliser la caméra

La prochaine étape est de modifier le code pour qu'à la place d'étudier l'image du Golden Retriever, on puisse lui fournir en entrée l'image de la caméra branchée à la carte NVIDIA. Pour cela, nous avons choisi de demander à l'algorithme d'ouvrir la fenêtre de capture de la caméra, d'appuyer sur un bouton pour prendre des photos, les enregistrer puis analyser ces photos avec le modèle. Le code correspondant se trouve dans *src/cnn.py*.

---

```
1 def gstreamer_pipeline(
2     capture_width=3280,
3     capture_height=2464,
4     display_width=820,
5     display_height=616,
6     framerate=21,
7     flip_method=0,
8     ):
9     return (
```

```

10         "nvarguscamerasrc ! "
11         "video/x-raw(memory:NVMM), "
12         "width=(int)%d, height=(int)%d, "
13         "format=(string)NV12, framerate=(fraction)%d/1 ! "
14         "nvvidconv flip-method=%d ! "
15         "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
16         "videoconvert ! "
17         "video/x-raw, format=(string)BGR ! appsink"
18     % (
19         capture_width,
20         capture_height,
21         framerate,
22         flip_method,
23         display_width,
24         display_height,
25     )
26 )
27
28
29 def take_picture():
30
31     cap = cv2.VideoCapture(gstreamer_pipeline(), cv2.CAP_GSTREAMER)
32     if cap.isOpened():
33         cv2.namedWindow("Test", cv2.WINDOW_AUTOSIZE)
34         while cv2.getWindowProperty("Test", 0) >= 0:
35             ret, img = cap.read()
36             cv2.imshow("Test", img)
37             keyCode = cv2.waitKey(30) & 0xFF
38             # Stop the program on the ESC key
39             if keyCode == 27:
40                 break
41             if keyCode == 32:
42                 cv2.imwrite(f"images/{uuid4()}.png", img)
43     cap.release()
44     cv2.destroyAllWindows()
45 else:
46     print("Unable to open camera")

```

### 2.3 Inférence sur la carte avec les images de la caméra

On commence par exécuter le code sur la carte, en prenant des photos avec la caméra et en demandant à l'algorithme des prédictions. Voici des exemples de photos qui ont été prises pour cet exemple :



FIGURE 2 – Règle, reconnue comme une spatule



FIGURE 3 – Stylo, reconnu comme une seringue

Voici maintenant les prédictions obtenues par le réseau, avec le temps d'inférence associé (le premier n'est pas exact, car nous n'avions pas encore enlevé le chargement du réseau du calcul d'inférence) :

```
KeyboardInterrupt
embedded@abosio-desktop:~/onaly$ python3 run2.py
Predicted class time : 8.64 s for 5c4fecce-7be8-426c-8316-cea4a2fa2629.png is: cleaver
Predicted class time : 0.25 s for e64d0cc2-d7a1-4220-bc4f-5d451b1cca5d.png is: syringe
Predicted class time : 0.09 s for 5400bcfd-7a32-4528-933b-8cc679680327.png is: Band-Aid
Predicted class time : 0.09 s for 6333d6ef-2c30-4f44-831e-4088ab7a0aa2.png is: sunglasses
Predicted class time : 0.09 s for bdb5f643-9b30-4b1f-b065-1973e43e2955.png is: plastic bag
Predicted class time : 0.09 s for c646b47c-2214-4de9-b9c1-5416a9f3eb0d.png is: spatula
Predicted class time : 0.09 s for 15cfcb46-036c-4121-83d8-2436ae1eae59.png is: dumbbell
Predicted class time : 0.09 s for 7b976695-e185-4085-9dac-af0dfb8c9554.png is: knee pad
Predicted class time : 0.09 s for 7bf2f448-f218-4c2b-a929-0c063d228c04.png is: bathtub
Predicted class time : 0.09 s for b672be2b-430a-41f0-90e4-fdee228ad3d1.png is: remote control
embedded@abosio-desktop:~/onaly$
```

FIGURE 4 – Prédiction des images de la caméra

La plupart sont incorrectes, mais les photos sont aussi en général de mauvaise qualité. Les objets ne sont pas dans des conditions 'normales', la luminosité est plutôt mauvaise étant donné que les photos sont face à des néons, et il y a souvent une main qui vient tenir l'objet et rendre la lecture de l'image plus difficile pour le réseau. Ceci dit, même si les prédictions sont approximatives, le temps d'inférence est très rapide, environ un dixième de seconde.

## 2.4 Inférence sur l'ordinateur avec les images de la caméra

On fait de même mais sur notre ordinateur pour comparer les temps d'inférence. On obtient ces résultats :

```
['CNCCalculator.png', 'CNCard.png', 'CNNkeys.png', 'CNMouse.png',
'CNPen.png', 'CNNPhone.png', 'CNNPhoneFace.png', 'CNNRuler.png',
'CNNRuler2.png', 'CNNsunglasses.png', 'dog.png']
Predicted class is: remote control, in 0.23337578773498535
Predicted class is: plastic bag, in 0.19846892356872559
Predicted class is: spatula, in 0.1845078468322754
Predicted class is: bathtub, in 0.18949151039123535
Predicted class is: syringe, in 0.1934828758239746
Predicted class is: Band-Aid, in 0.1765270233154297
Predicted class is: knee pad, in 0.1795196533203125
Predicted class is: dumbbell, in 0.18547320365905762
Predicted class is: cleaver, in 0.2184152603149414
Predicted class is: sunglasses, in 0.18749785423278809
Predicted class is: Golden Retriever, in 0.1934821605682373
```

FIGURE 5 – Prédiction des images de la caméra sur l’ordinateur

Les résultats restent bien évidemment les mêmes, mais les temps d’inférence sont plus longs sur l’ordinateur, ce qui est logique. La carte sait mieux optimiser sa puissance avec l’utilisation du processeur graphique. Le temps d’inférence est en moyenne deux fois supérieur sur l’ordinateur.

## 2.5 Comparaison des résultats avec d’autres réseaux de neurones

On essaie maintenant avec d’autres réseaux de neurones que Resnet50 pour comparer la précision et les temps d’inférence. Par exemple, en essayant avec AlexNet, on obtient les résultats suivants :

```
Predicted class is: toilet seat, in 0.12808942794799805
Predicted class is: carton, in 0.0329132080078125
Predicted class is: toilet seat, in 0.02693033218383789
Predicted class is: toilet seat, in 0.03391003608703613
Predicted class is: carton, in 0.02693033218383789
Predicted class is: toilet paper, in 0.03191256523132324
Predicted class is: toilet paper, in 0.028928518295288086
Predicted class is: mosquito net, in 0.024929285049438477
Predicted class is: shower cap, in 0.02593374252319336
Predicted class is: toilet seat, in 0.027926921844482422
Predicted class is: Sealyham Terrier, in 0.03391289710998535
```

FIGURE 6 – Prédiction des images de la caméra sur l’ordinateur avec le réseau AlexNet

On voit que les résultats sont nettement différents. Ce réseau semble accorder un peu plus d’importance au fond de l’image, car toutes les photos ont été prises avec le même fond, et les résultats sont plutôt consistants.

## 3 Transfer Learning

Dans cette partie, afin de permettre à un réseau d’apprendre à classifier les abeilles et les fourmis, on utilise un modèle pré-entraîné (ResNet18) comme extracteur de descripteurs et on remplace la couche de sortie par une couche fully-connected dont les poids seront mis à jour par entraînement sur le dataset contenant les images des abeilles et des fourmis. Les poids de l’extracteur de descripteurs, eux, resteront fixes.

### 3.1 Étude du code de *run3.py* et des résultats obtenus

Le code est divisé en plusieurs parties :

- Une première partie consiste entre autres à créer des fonctions qui utilisent des modules de Pytorch pour appliquer un prétraitement différent (normalisation, recadrage, flip, ...) aux images d'entrée du réseau selon si l'image appartient aux données d'entraînement ou aux données de test. L'idée est de reproduire le prétraitement qui a été appliqué initialement aux images qui ont servi à entraîner les modèles ImageNet tout en augmentant la variabilité des données d'entraînement. Certains paramètres (comme le batch size) y sont également définis.
- Une deuxième partie consiste à afficher quelques images du dataset.
- Une troisième et dernière partie consiste à créer une fonction qui va entraîner le modèle sur la carte et afficher la loss à chaque epoch. À la fin de l'entraînement, les poids qui auront donné les meilleurs résultats sur le jeu de validation sont ceux qui sont retenus. On appelle ensuite cette fonction (en plus d'autres fonctions) pour effectuer l'entraînement.

Remarque : Les deux premières parties du code sont celles qui constituent le code de *run2.py*

L'exécution du code permet d'afficher tout d'abord les images suivantes :

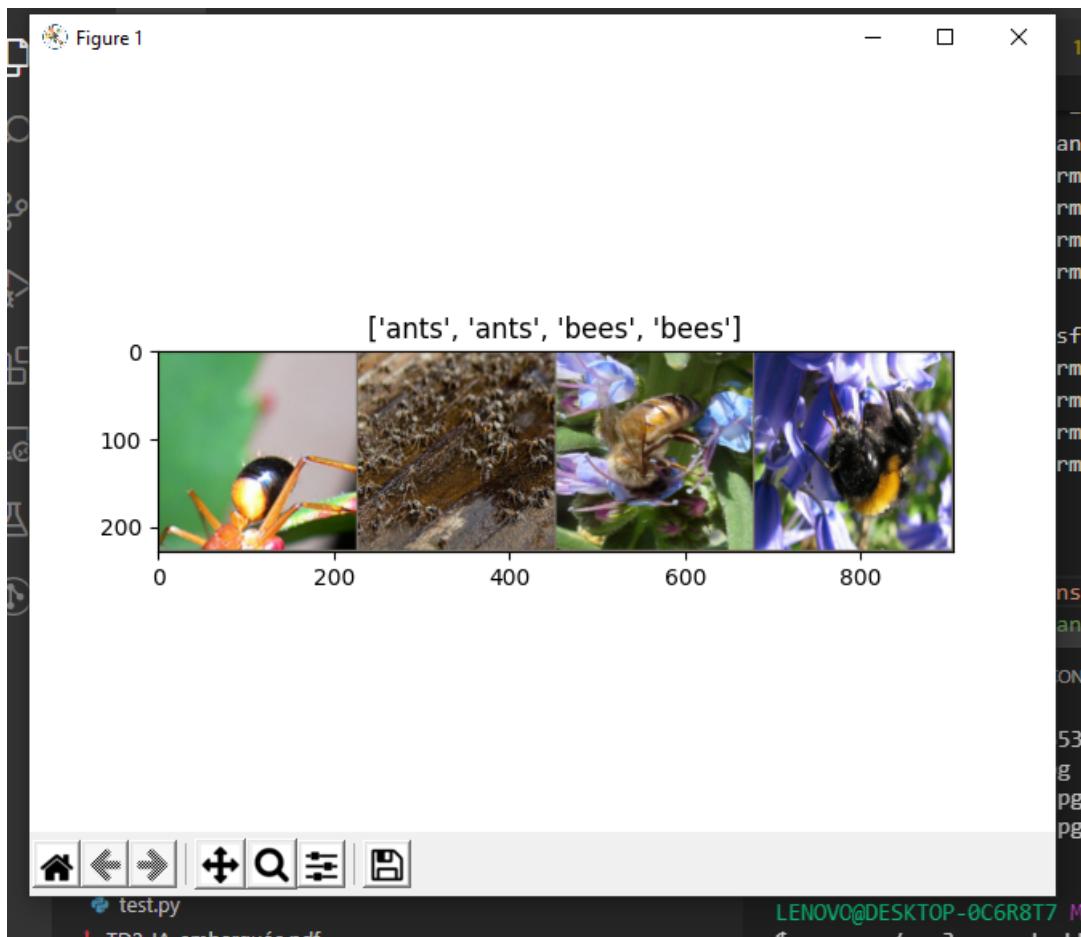


FIGURE 7 – Images affichées

Après l'affichage des images, l'entraînement démarre et lorsqu'il se termine, nous obtenons les résultats suivants :

```

val Loss: 0.1728 Acc: 0.9150
Epoch 5/10
-----
train Loss: 0.4464 Acc: 0.7951
val Loss: 0.2575 Acc: 0.8954

Epoch 6/10
-----
train Loss: 0.5786 Acc: 0.7787
val Loss: 0.2025 Acc: 0.9412

Epoch 7/10
-----
train Loss: 0.3854 Acc: 0.8361
val Loss: 0.1665 Acc: 0.9477

Epoch 8/10
-----
train Loss: 0.2629 Acc: 0.8811
val Loss: 0.1726 Acc: 0.9477

Epoch 9/10
-----
train Loss: 0.4364 Acc: 0.8197
val Loss: 0.1793 Acc: 0.9477

Epoch 10/10
-----
train Loss: 0.3626 Acc: 0.8320
val Loss: 0.1638 Acc: 0.9542

-> Training complete in 2m 13s
Best val Acc: 0.954248
embedded@abosio-desktop:~/onaly$ █

```

FIGURE 8 – Entraînement du classifieur d’abeilles et fourmis sur la carte

On observe que l’entraînement dure 2 minutes et 13 secondes et on obtient une précision de 95% sur le jeu de validation, ce qui est une très bonne performance.

### 3.2 Comparaison de la durée d’entraînement de la carte par rapport à l’ordinateur

Quand on exécute le code sur l’ordinateur, nous obtenons les performances suivantes :

```

-----
train Loss: 0.3027 Acc: 0.8648
val Loss: 0.2105 Acc: 0.9281

Epoch 10/10
-----
train Loss: 0.4055 Acc: 0.8279
val Loss: 0.1706 Acc: 0.9412

Training complete in 5m 25s
Best val Acc: 0.954248

```

FIGURE 9 – Entraînement du classifieur d’abeilles et fourmis sur l’ordinateur

On constate que la précision du réseau reste la même ce qui est tout à fait attendu et logique. En revanche, l'entraînement dure 5 minutes et 25 secondes. L'entraînement sur la carte en utilisant un processeur graphique permet donc de diminuer grandement la durée d'entraînement et illustre l'intérêt d'un dispositif embarqué de ce type.

### 3.3 Inférence sur la carte avec les images de la caméra

Dans cette partie, nous allons modifier le code pour prédire à l'aide d'images prises par la caméra. Cela nous permettra de quantifier le temps d'inférence de la carte et la performance du réseau en "temps réel". Le code correspondant se trouve dans *src/run3.py*.

Pour cela, nous modifions la fonction de prise de photo utilisée précédemment, et l'ajoutons à la fin du code :

---

```

1 import cv2
2 from uuid import uuid4
3
4 def gstreamer_pipeline(
5     capture_width=3280,
6     capture_height=2464,
7     display_width=820,
8     display_height=616,
9     framerate=21,
10    flip_method=0,
11):
12    return (
13        "nvarguscamerasrc ! "
14        "video/x-raw(memory:NVMM), "
15        "width=(int)%d, height=(int)%d, "
16        "format=(string)NV12, framerate=(fraction)%d/1 ! "
17        "nvvidconv flip-method=%d ! "
18        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
19        "videoconvert ! "
20        "video/x-raw, format=(string)BGR ! appsink"
21        %
22            capture_width,
23            capture_height,
24            framerate,
25            flip_method,
26            display_width,
27            display_height,
28        )
29    )
30
31
32 def take_picture():
33

```

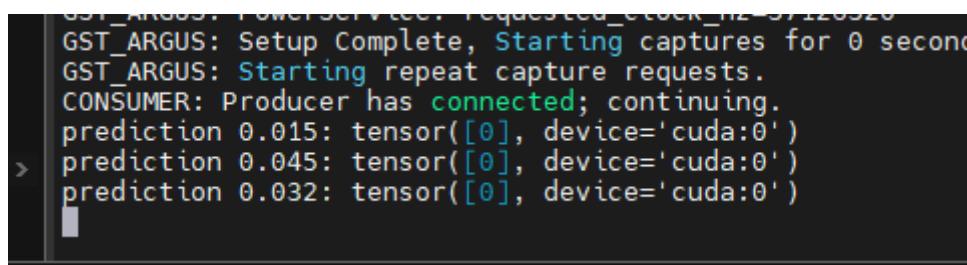
```

34     cap = cv2.VideoCapture(gstreamer_pipeline(), cv2.CAP_GSTREAMER)
35     if cap.isOpened():
36         cv2.namedWindow("Test", cv2.WINDOW_AUTOSIZE)
37         while cv2.getWindowProperty("Test", 0) >= 0:
38             ret, img = cap.read()
39             cv2.imshow("Test", img)
40             keyCode = cv2.waitKey(30) & 0xFF
41             # Stop the program on the ESC key
42             if keyCode == 27:
43                 break
44             if keyCode == 32:
45                 start_time = time.time()
46                 cv2.imwrite(f"test_transfer_learning/{uuid4()}.png", img)
47                 test_img = Image.fromarray(img.astype('uint8'), 'RGB')
48                 test_image = data_transforms['val'](test_img).unsqueeze(0).cuda()
49                 out = model(test_image)
50                 _, pred = torch.max(out, 1)
51                 end_time = time.time()
52                 print(f"prediction {end_time-start_time:.3f}:", pred)
53                 cap.release()
54                 cv2.destroyAllWindows()
55             else:
56                 print("Unable to open camera")
57
58     take_picture()

```

Cette partie du code consiste à créer une fonction qui va ouvrir une fenêtre montrant l'image de la caméra et ensuite exécuter quelques lignes quand la touche Espace est appuyée. Comme on peut le voir à partir des lignes 44 à 53, quand la touche Espace sera appuyée, l'image en cours sera enregistrée et servie au modèle pour effectuer une prédiction ( sachant que 0 représente les fourmis et 1 les abeilles). Nous calculons et affichons également le temps d'inférence.

Nous obtenons les résultats suivants suite à des tests sur quelques images :



```

GST_AUGUS: PowerService requested clock_id: 37120520
GST_ARGUS: Setup Complete, Starting captures for 0 seconds
GST_ARGUS: Starting repeat capture requests.
CONSUMER: Producer has connected; continuing.
> prediction 0.015: tensor([0], device='cuda:0')
> prediction 0.045: tensor([0], device='cuda:0')
> prediction 0.032: tensor([0], device='cuda:0')

```

FIGURE 10 – Performances d'inférence du réseau sur la carte

On constate que la carte met en moyenne 0,03 secondes par image soit environ 33 images par seconde. En ce qui concerne les performances du réseau, nous ne pouvons pas conclure car le réseau semble prédire la même chose ([0]) alors que nous avons varié entre

une abeille et fourmi sur les 3 images prises. Il se pourrait aussi que l'image ne soit pas correctement chargée avec les fonctions que nous avons utilisé notamment à la ligne 47. Par manque de temps, nous n'avons pas pu débuguer cette partie. Cependant, comme les performances prédictives du réseau ne dépendent a priori pas du matériel à l'exception de la caméra et que la caméra de la webcam et la caméra fournie avec la carte ne sont pas si différentes nous allons pouvoir tester cela sur l'ordinateur avec la webcam.

### 3.4 Inférence sur l'ordinateur avec images de la caméra

Afin de comparer les performances d'inférence de la carte avec celles de l'ordinateur, nous allons modifier la fonction *take\_picture* précédente pour qu'elle prenne les images de la webcam de l'ordinateur et nous avons également changé la manière dont nous servons l'image au réseau. Au lieu de donner directement l'image en cours, nous l'enregistrons d'abord puis nous la rechargeons pour la servir au modèle à l'aide de la fonction *Image.open* car cela avait fonctionné dans la partie précédente (dans *cnn.py*). Le code de cette partie se trouve dans *src/run3local.py*.

Voici le code correspondant :

---

```

1  def take_picture():
2
3      cap = cv2.VideoCapture(0)
4      if cap.isOpened():
5          cv2.namedWindow("Test", cv2.WINDOW_AUTOSIZE)
6          k = 0
7          while cv2.getWindowProperty("Test", 0) >= 0:
8              _, img = cap.read()
9              cv2.imshow("Test", img)
10             keyCode = cv2.waitKey(30) & 0xFF
11             # Stop the program on the ESC key
12             if keyCode == 27:
13                 break
14             if keyCode == 32:
15                 path = os.getcwd()+"\\data\\localcam\\img{k}.png"
16                 cv2.imwrite(path,img)
17                 start_time = time.time()
18                 test_img = Image.open(path)
19                 test_image = data_transforms['val'](test_img).unsqueeze(0)
20                 out = model(test_image)
21                 _, pred = torch.max(out, 1)
22                 pred = int(pred)
23                 end_time = time.time()
24                 if pred == 0:
25                     print(f"Prédiction img{k}: Fourmi")
26                 else:
27                     print(f"Prédiction img{k}: Abeille")
28             print(f"prediction time {end_time-start_time:.3f} s")
```

```

29             k += 1
30         cap.release()
31         cv2.destroyAllWindows()
32     else:
33         print("Unable to open camera")
34
35 take_picture()

```

---

Voici les images que nous avons prises à l'aide de la webcam :

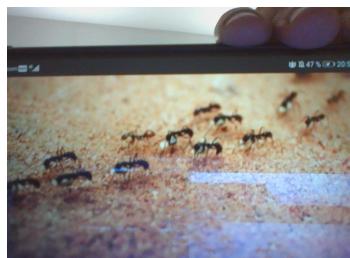


FIGURE 11 – img0.png



FIGURE 12 – img1.png

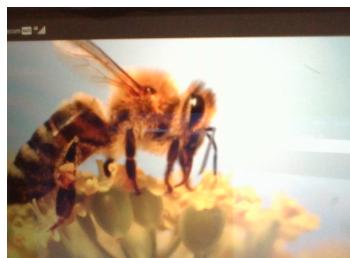


FIGURE 13 – img2.png

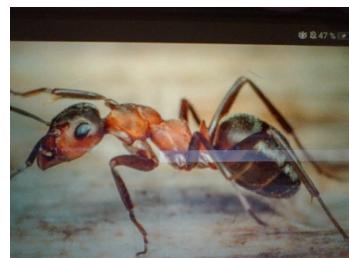


FIGURE 14 – img3.png

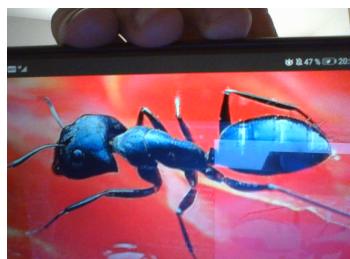


FIGURE 15 – img4.png

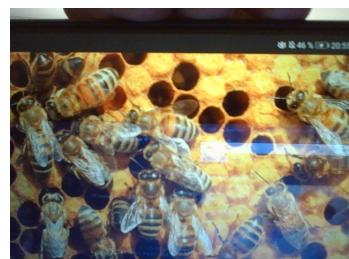


FIGURE 16 – img5.png



FIGURE 17 – img6.png



FIGURE 18 – img7.png



FIGURE 19 – img8.png



FIGURE 20 – img9.png



FIGURE 21 – img10.png

Avec ces images, nous obtenons les performances suivantes sur l'ordinateur après entraînement sur 2 epochs :

```
Epoch 1/2
-----
C:\Users\LENOVO\AppData\Local\Pro
  UserWarning: Detected call of `l
r, you should call them in the op
to do this will result in PyTorch
s at https://pytorch.org/docs/sta
  warnings.warn("Detected call of
train Loss: 0.6501 Acc: 0.6557
val Loss: 0.3409 Acc: 0.8301

Epoch 2/2
-----
train Loss: 0.6798 Acc: 0.6844
val Loss: 0.1962 Acc: 0.9281
```

FIGURE 22 – Entraînement sur 2 epochs du réseau

PROBLÈMES	SORTIE	CONSOLE DE DÉBOGAGE	TERMINAL
			<pre>Prédiction img0: Fourmi prediction time 0.105 s Prédiction img1: Abeille prediction time 0.085 s Prédiction img2: Abeille prediction time 0.098 s Prédiction img3: Fourmi prediction time 0.116 s Prédiction img4: Fourmi prediction time 0.100 s Prédiction img5: Fourmi prediction time 0.100 s Prédiction img6: Fourmi prediction time 0.114 s Prédiction img7: Fourmi prediction time 0.084 s Prédiction img8: Abeille prediction time 0.122 s Prédiction img9: Fourmi prediction time 0.117 s Prédiction img10: Fourmi prediction time 0.100 s</pre>

FIGURE 23 – Performances d’inférence du réseau sur l’ordinateur

On constate qu’en moyenne le réseau met 0.1 seconde pour réaliser la prédiction, ce qui est environ 3 fois supérieur au temps d’inférence de la carte et on observe que le réseau se trompe 2 fois sur les 11 images notamment sur *img5.png* et *img7.png*. Cela pourrait venir soit de la qualité de l’image qui n’est pas exactement celle des données qui ont servi à l’entraînement, soit de certaines caractéristiques de ces images qui induisent le réseau en erreur soit tout simplement parce que le réseau n’a pas été suffisamment entraîné. Une analyse plus fine de la performance sur les images de test pourrait nous donner plus d’indications sur l’origine de ces erreurs. Nous pouvons quand même considérer que le réseau est suffisamment performant car il arrive à réaliser une prédiction correcte 9 fois sur 11 et notamment sur les images *img9.png* ou *im10.png* qui ne sont pas si "évidentes" à classifier. On peut donc dire que les couches en amont du réseau (celles issues du réseau pré-entraîné) ont permis d’extraire les bons descripteurs et ces descripteurs sont suffisamment généraux et robustes pour permettre à un réseau de se spécialiser dans la reconnaissance de n’importe quel objet et cela illustre bien l’intérêt du Transfer Learning.

## 4 Conclusion

Pour conclure, nous avons pu observer des comportements de différents classificateurs à base de réseaux de neurones, que ce soit sur une carte dédiée ou sur un ordinateur, avec des images provenant d’une caméra. Nous avons pu constater que, comme prévu, les résultats arrivent plus rapidement là où la puissance de calcul est plus optimisée, c'est à

dire sur la carte, qui exploite le parallélisme et le calcul avec le processeur graphique.