Lab 4 - Jetson Nano – inférence

Contenu

4.0 Introduction

Dans ce laboratoire, nous allons analyser et tester un certain nombre d'applications avec des modèles NN préparés pour l'inférence. Toutes les applications sont développées avec le langage Python.
Nous commençons par la classification des images avec **ImageNet** et la localisation des objets avec **DetectNet** .

4.0.1 Classification des images avec `ImageNet`

Il existe plusieurs types de réseaux d'apprentissage en profondeur disponibles, notamment la reconnaissance, la détection/localisation et la segmentation sémantique. La première capacité d'apprentissage en profondeur que nous mettons en évidence dans ce laboratoire est la reconnaissance d'images à l'aide de réseaux de classification qui ont été formés pour identifier des scènes et des objets.

L'objet **`imageNet`** accepte une image d'entrée et génère la probabilité pour chaque classe. Après avoir été entraînés sur l'ensemble de données **ImageNet ILSVRC** de 1000 objets, les modèles **`GoogleNet`** et **`ResNet-18`** ont été automatiquement téléchargés lors de l'étape de construction.
D'autres modèles de classification peuvent également être téléchargés et utilisés.

4.0.2 Localisation d'objets avec `DetectNet`

Les exemples de reconnaissance d'image précédents ont des probabilités de classe de sortie représentant l'intégralité de l'image d'entrée. La deuxième capacité d'apprentissage en profondeur que nous mettons en évidence dans ce laboratoire est la **détection d'objets** et la recherche de leur **emplacement** dans la vidéo (c'est-à-dire l'extraction de leurs cadres de délimitation). Ceci est effectué à l'aide d'un **`detectNet`** - ou d'un réseau de détection/localisation d'objets.
L'objet **`detectNet`** accepte en entrée l'**image 2D** et génère une liste de coordonnées des cadres de délimitation détectés. Pour la détection d'objets, un modèle de pré-entraîné (comme **`ssd-mobilenet-v2`**) est utilisé par défaut avec des étiquettes de coordonnées de délimitation incluses dans l'ensemble de **données d'entraînement** en plus des images de source.

4.0.3 L'estimation de pose avec `PoseNet`

L'estimation de pose consiste à localiser diverses parties du corps (alias points clés) qui forment une topologie squelettique (alias liens). L'estimation de pose a une variété d'applications, y compris les gestes, AR/VR, **IHM** (interface homme/machine) et la correction de posture/démarche. Des modèles pré-entraînés sont fournis pour l'estimation de la pose du corps humain et de la main qui sont capables de détecter plusieurs personnes par image. Par défaut l'application utilise le modèle **`resnet18-body`**.

4.0.4 Profondeur monoculaire avec `DepthNet`

La détection de profondeur est utile pour des tâches telles que la cartographie, la navigation et la détection d'obstacles, mais elle nécessitait historiquement une **caméra stéréo** ou une caméra **RVB-D**.
Il existe maintenant des **DNN** capables de déduire une profondeur relative à partir d'une seule image monoculaire (alias **profondeur mono**). Une telle approche peut être accompli en utilisant des réseaux entièrement convolutifs (**FCN**). Le modèle (par défaut) pré-entraîné utilisé dans cette application est **`fcn-mobilenet`**.

4.0.5 Segmentation sémantique avec `SegNet`

La troisième capacité d'apprentissage en profondeur que nous pouvons mettre en évidence est la segmentation sémantique. La segmentation sémantique est basée sur la reconnaissance d'images, sauf que les classifications se produisent au niveau du pixel par opposition à la classification d'images entières comme avec la reconnaissance d'images.
Ceci est accompli en « convolutionnalisant » un modèle de reconnaissance d'images pré-entraîné (comme **Alexnet**), qui le transforme en un modèle de segmentation entièrement convolutif capable d'un étiquetage par pixel. Par défaut l'application utilise le modèle **`fcn-resnet18-voc.`**
Utile pour la détection de l'environnement et l'évitement des collisions, la segmentation produit une classification dense par pixel de nombreux objets potentiels différents par scène, y compris les avant-plans et les arrière-plans des scènes.

4.1 Classification des images avec `ImageNet`

Il existe plusieurs types de réseaux d'apprentissage en profondeur disponibles, notamment la reconnaissance, la détection/localisation et la segmentation sémantique. La première capacité d'apprentissage en profondeur que nous mettons en évidence dans ce laboratoire est la **reconnaissance d'images** à l'aide de **réseaux de classification** qui ont été formés pour identifier des scènes et des objets.

L'objet **`imageNet`** accepte une image d'entrée et génère la **probabilité pour chaque classe**. Après avoir été entraînés sur l'ensemble de données **ImageNet ILSVRC** de **1000** objets, les modèles **`GoogleNet`** et **`ResNet-18`** ont été automatiquement téléchargés lors de l'étape de construction.
Autres modèles de classification peuvent également être téléchargés et utilisés.

Comme exemples d'utilisation d'**`imageNet`**, nous utilisons **Python** :

- **`imagenet-console.py`**

Plus loin dans cette section, nous aborderons également les versions d'un programme de reconnaissance à partir d'une caméra ou d'un WebCam en mode direct :

- **`imagenet-camera.py`**

4.1.1 Utilisation du programme `console` sur Jetson

Tout d'abord, essayons d'utiliser le programme **`imagenet-console.py`** pour tester la reconnaissance **`imageNet`** sur quelques exemples d'images. Il charge une image, utilise **`TensorRT`** et la classe **`imageNet`** pour effectuer l'inférence, puis superpose le résultat de la classification et enregistre l'image de sortie.

Le dépôt est livré avec quelques exemples d'images que vous pouvez utiliser.

Nous initialisons une session **docker** par :

```
$ cd jetson-inference
docker/run.sh
```

```
reading L4T version from /etc/nv_tegra_release
L4T BSP Version:  L4T R32.6.1
[sudo] password for jetson:
size of data/networks:  1269108476 bytes
CONTAINER:     dustynv/jetson-inference:r32.6.1
DATA_VOLUME:   --volume /home/jetson/jetson-inference/data:/jetson-inference/data --volume
/home/jetson/jetson-inference/python/training/classification/data:/jetson-inference/python/
training/classification/data --volume
/home/jetson/jetson-inference/python/training/classification/models:/jetson-inference/python/
training/classification/models --volume
/home/jetson/jetson-inference/python/training/detection/ssd/data:/jetson-inference/python/training/
detection/ssd/data --volume /home/jetson/jetson-inference/python/training/detection/ssd/models:/
jetson-inference/python/training/detection/ssd/models
USER_VOLUME:
USER_COMMAND:
V4L2_DEVICES:
localuser:root being added to access control list
xhost:  must be on local machine to add or remove hosts.
root@jetson-desktop:/jetson-inference#
```

Après avoir construit le dépôt, assurez-vous que votre terminal se trouve dans le répertoire :
`build/aarch64/bin :`

```
root@jetson-desktop:/jetson-inference#cd build/aarch64/bin
```

Vous vérifiez la présence des programmes (applications) dan le répertoire **`bin :`**

```
root@jetson-desktop:/jetson-inference/build/aarch64/bin# ls
```

```
camera-capture      detectnet             imagenet-console     segnet-camera
camera-viewer       detectnet-camera      imagenet-console.py  segnet-camera.py
camera-viewer.py    detectnet-camera.py   imagenet.py          segnet-console
```

```
cuda-examples.py     detectnet-console     images               segnet-console.py
cuda-from-cv.py      detectnet-console.py  my-detection.py      segnet.py
cuda-from-numpy.py   detectnet.py          my-recognition.py    segnet_utils.py
cuda-to-cv.py        gl-display-test       networks             video-viewer
cuda-to-numpy.py     gl-display-test.py    posenet              video-viewer.py
depthnet             imagenet              posenet.py
depthnet.py          imagenet-camera       segnet
depthnet_utils.py    imagenet-camera.py    segnet-batch.sh
```

Ensuite, classons une image avec le programme **`imagenet-console`** :

`imagenet-console` accepte 3 arguments de ligne de commande :

- le **chemin** (*path*) vers une **image d'entrée** (**jpg**, **png**, **tga**, **bmp**)
- **chemin facultatif** vers l'**image de sortie** (**jpg**, **png**, **tga**, **bmp**)
- indicateur optionnel **`--network`** qui modifie le modèle de classification utilisé (le réseau par défaut est **`GoogleNet`**).

Notez qu'il existe des paramètres de ligne de commande supplémentaires disponibles pour le chargement de modèles personnalisés. Lancez l'application avec l'indicateur **`--help`** pour recevoir plus d'informations sur leur utilisation.
Voici quelques exemples d'exécution du programme en Python :

```
$ ./imagenet-console.py --network=googlenet images/cat_0.jpg images/out_0.jpg
```

Remarques:
Le drapeau **`--network`** est facultatif. La première fois que vous exécutez le programme, **`TensorRT`** peut prendre jusqu'à **quelques minutes pour optimiser le réseau**.
Ce fichier réseau optimisé est **mis en cache** sur le disque après la première exécution, de sorte que les futures exécutions se chargeront plus rapidement.

4.1.2 Téléchargement d'autres modèles de classification

Par défaut, le référentiel est configuré pour télécharger les réseaux **`GoogleNet`** et **`ResNet-18`** lors de l'étape de construction.
Il existe d'autres modèles pré-entraînés que vous pouvez également utiliser, si vous choisissez de les télécharger :

Network	CLI argument	NetworkType enum
AlexNet	alexnet	ALEXNET
GoogleNet	googlenet	GOOGLENET
GoogleNet-12	googlenet-12	GOOGLENET_12
ResNet-18	resnet-18	RESNET_18
ResNet-50	resnet-50	RESNET_50
ResNet-101	resnet-101	RESNET_101
ResNet-152	resnet-152	RESNET_152
VGG-16	vgg-16	VGG-16
VGG-19	vgg-19	VGG-19
Inception-v4	inception-v4	INCEPTION_V4

Tab 1. Modèles pour la **classification** des objets.

Remarque : pour télécharger des réseaux supplémentaires, exécutez l'outil **Model Downloader** :

```
$ cd jetson-inference/tools
$ ./download-models.sh
```

En général, les réseaux les plus complexes peuvent avoir une plus grande précision de classification, avec une durée d'exécution (*run time*) accrue.

Attention : Le chargement de certains modèles de taille très important peut provoquer le blocage du système.

4.1.3 Utilisation de différents modèles de classification

Vous pouvez spécifier le modèle à charger en définissant l'indicateur **`--network`** sur la ligne de commande sur l'un des arguments CLI correspondants du tableau ci-dessus.
Par défaut, **GoogleNet** est chargé si l'indicateur facultatif **`--network`** n'est pas spécifié.

Voici un exemple d'utilisation du modèle **`ResNet-18`** :

```
$ ./imagenet-console.py --network=resnet-18 jellyfish.jpg output_jellyfish.jpg
```

4.1.4 Classification avec une caméra/WebCam en direct

Nous avons une démo de caméra de reconnaissance d'image en temps réel disponible pour Python :

`imagenet-camera.py`

Semblable à l'exemple précédent de la **`console-imagenet`**, les applications de caméra sont créées dans le répertoire **`/aarch64/bin`**. Ils fonctionnent sur un flux de caméra en direct avec un rendu **OpenGL** et acceptent 4 arguments de ligne de commande facultatifs :

- **`--network`** flag définissant le modèle de classification (la valeur par défaut est **GoogleNet**)
  Voir Téléchargement d'autres modèles de classification pour les réseaux disponibles à utiliser.

- **`--camera`** flag définissant l'appareil photo à utiliser
  - Les caméras **MIPI CSI** sont utilisées en spécifiant l'index du capteur (**0** ou **1**, ect.)
  - Les caméras **`USB V4L2`** sont utilisées en spécifiant leur nœud **`/dev/video`** (**`/dev/video0`**, **`/dev/video1`**, ect.)
  - La valeur par défaut est le capteur **MIPI CSI 0** (**`--camera=0`**)

- Indicateurs **`--width`** et **`--height`** définissant la résolution de la caméra (défaut est **1280x720**)
  - La résolution doit être réglée sur un format pris en charge par la caméra.

Vous pouvez combiner l'utilisation de ces indicateurs selon vos besoins. Des paramètres de ligne de commande supplémentaires sont disponibles pour le chargement de modèles personnalisés.

Lancez l'application avec l'indicateur **`--help`** pour recevoir plus d'informations.

Voici quelques scénarios typiques de lancement du programme :

```
$ ./imagenet-camera.py      # using GoogleNet, default MIPI CSI camera (1280x720)
$ ./imagenet-camera.py --network=resnet-18  # using ResNet-18, default MIPI CSI camera (1280x720)
$ ./imagenet-camera.py --camera=/dev/video0 # using GoogleNet, V4L2 camera /dev/video0 (1280x720)
$ ./imagenet-camera.py --width=640 --height=480 # using GoogleNet, default MIPI CSI camera (640x480)
```

Pour le flux de caméra en direct, le nom de l'objet classifié et la confiance de l'objet classifié, ainsi que la fréquence d'images du réseau, sont affichés dans la fenêtre **OpenGL**. Sur Jetson Nano, vous devriez voir jusqu'à environ **75 FPS** pour **`GoogleNet`** et **`ResNet-18`**.

L'application peut reconnaître jusqu'à **1000 types d'objets différents**, car les modèles de classification sont entraînés sur l'ensemble de données **ILSVRC ImageNet** qui contient 1000 classes d'objets.

Le mappage des noms pour les 1000 types d'objets, vous pouvez trouver dans le repo sous

`data/networks/ilsvrc12_synset_words.txt`

4.2 Détection/localisation d'objets à partir de la ligne de commande/console

Pour traiter les images de test avec **`detectNet`** et **TensorRT** sur le Jetson, nous pouvons utiliser le programme **`detectnet-console`**.

`detectnet-console` accepte les arguments de ligne de commande représentant le **chemin d'accès à l'image d'entrée** et le **chemin d'accès à l'image de sortie** (avec les superpositions de cadre de délimitation rendues).

4.2.1 Lancement avec un modèle pré-entraîné

Alternativement, pour charger l'un des instantanés pré-entraînés fournis avec le référentiel, vous pouvez spécifier l'indicateur facultatif **`--network`** qui modifie le modèle de détection utilisé (le réseau par défaut est **`SSD-Mobilenet-v2`**).

Voici un exemple de localisation d'humains dans une image avec le modèle **`SSD-Mobilenet-v2`** par défaut :

```
$ python3 ./detectnet-console.py images/peds_1.jpg images/out_1.jpg
```

Vous pouvez consulter les images disponibles et générées dans :

```
~/jetson-inference/data/images
```

4.2.2 Détection/localisation avec des modèles pré-entraînée disponibles

Vous trouverez ci-dessous un tableau des réseaux de détection d'objets pré-entraînés disponibles au téléchargement, et l'argument -**`-network`** associé à **`detectnet-console`** utilisé pour charger les modèles pré-entraînés :

Model	CLI argument	NetworkType enum	Object classes
SSD-Mobilenet-v1	ssd-mobilenet-v1	SSD_MOBILENET_V1	91 (COCO classes)
SSD-Mobilenet-v2	ssd-mobilenet-v2	SSD_MOBILENET_V2	91 (COCO classes)
SSD-Inception-v2	ssd-inception-v2	SSD_INCEPTION_V2	91 (COCO classes)
DetectNet-COCO-Dog	coco-dog	COCO_DOG	dogs
DetectNet-COCO-Bottle	coco-bottle	COCO_BOTTLE	bottles
DetectNet-COCO-Chair	coco-chair	COCO_CHAIR	chairs
DetectNet-COCO-Airplane	coco-airplane	COCO_AIRPLANE	airplanes
ped-100	pednet	PEDNET	pedestrians
multiped-500	multiped	PEDNET_MULTI	pedestrians, luggage
facenet-120	facenet	FACENET	faces

Tab 2. Modèles pour la **détection** des objets.

4.2.3 Exécution de différents modèles de détection/localisation

Vous pouvez spécifier le modèle à charger en définissant l'indicateur **`--network`** sur la ligne de commande sur l'un des arguments CLI correspondants du tableau ci-dessus.
Par défaut, **`PedNet`** est chargé (**détection des piétons**) si l'option **`--network`** n'est pas spécifiée.

Essayons d'exécuter certains des autres modèles **COCO** :

```
$ ./detectnet-console.py --network=coco-dog dog_1.jpg output_1.jpg
```

4.2.4 Exécution de la démonstration de détection/localisation en direct

Nous avons une démo de caméra/WebCam de détection d'objets en temps réel disponible pour Python :

detectnet-camera.py

Semblable à l'exemple de **detectnet-console** précédent, ces applications de caméra utilisent des réseaux de détection, sauf qu'elles traitent un flux vidéo en direct à partir d'une caméra. **detectnet-camera** accepte 4 paramètres de ligne de commande facultatifs :

- --**network** flag définissant le modèle de classification (la valeur par défaut est **PedNet**)
  - Voir Modèles de détection pré-entraînés disponibles pour les réseaux disponibles à utiliser.
- --**camera** flag définissant l'appareil photo à utiliser
  - Les caméras **MIPI CSI** sont utilisées en spécifiant l'index du capteur (**0** ou **1**, ect.)
  - Les caméras **USB V4L2** sont utilisées en spécifiant leur nœud **/dev/video** (**/dev/video0**, **/dev/video1**, ect.)
  - La valeur par défaut est d'utiliser le capteur MIPI CSI 0 (**--camera=0**)
- Indicateurs **--width** et **--height** définissant la résolution de la caméra (la valeur par défaut est **1280x720**)
  - La résolution doit être réglée sur un format pris en charge par la caméra.

Vous pouvez combiner l'utilisation de ces indicateurs selon vos besoins, et des paramètres de ligne de commande supplémentaires sont disponibles pour le chargement de modèles personnalisés. Lancez l'application avec l'indicateur **--help** pour recevoir plus d'informations, ou consultez le fichier d'exemples *readme.*

Voici quelques scénarios typiques de lancement du programme :

```
$ ./detectnet-camera.py                          # using PedNet,  default MIPI CSI camera (1280x720)
$ ./detectnet-camera.py --network=facenet        # using FaceNet, default MIPI CSI camera (1280x720)
$ ./detectnet-camera.py --camera=/dev/video0     # using PedNet,  V4L2 camera /dev/video0 (1280x720)
$ ./detectnet-camera.py --width=640 --height=480 # using PedNet,  default MIPI CSI camera (640x480)
```

4.2.4.1 Visualisation

Dans la fenêtre **OpenGL** on affiche le flux de caméra en direct superposé avec les cadres de délimitation des objets détectés. Notez que les modèles à base de SSD (**Single-Shot Detector**) ont actuellement les performances les plus élevées. En voici un utilisant le modèle **coco-dog** :

```
$ ./detectnet-camera.py -network=coco-dog
```

4.3 Estimation de la pose avec `posenet.py` et `resnet18-body`

L'estimation de pose consiste à localiser diverses parties du corps (alias points clés) qui forment une topologie squelettique (alias liens).Par défaut l'application utilise le modèle **`resnet18-body`**.

Voici le code complet de **`posenet.py`**.

```
import jetson.inference
import jetson.utils
import argparse
import sys
# parse the command line
parser = argparse.ArgumentParser(description="Run pose estimation DNN on a video/image stream.",
                                 formatter_class=argparse.RawTextHelpFormatter,
epilog=jetson.inference.poseNet.Usage() +
                                 jetson.utils.videoSource.Usage() + jetson.utils.videoOutput.Usage()
+ jetson.utils.logUsage())

parser.add_argument("input_URI", type=str, default="", nargs='?', help="URI of the input stream")
parser.add_argument("output_URI", type=str, default="", nargs='?', help="URI of the output stream")
parser.add_argument("--network", type=str, default="resnet18-body", help="pre-trained model to load
(see below for options)")
parser.add_argument("--overlay", type=str, default="links,keypoints", help="pose overlay flags (e.g.
--overlay=links,keypoints)\nvalid combinations are:  'links', 'keypoints', 'boxes', 'none'")
parser.add_argument("--threshold", type=float, default=0.15, help="minimum detection threshold to
use")
try:
        opt = parser.parse_known_args()[0]
except:
        print("")
        parser.print_help()
        sys.exit(0)

# load the pose estimation model
net = jetson.inference.poseNet(opt.network, sys.argv, opt.threshold)
# create video sources & outputs
input = jetson.utils.videoSource(opt.input_URI, argv=sys.argv)
output = jetson.utils.videoOutput(opt.output_URI, argv=sys.argv)
# process frames until the user exits
while True:
    # capture the next image
    img = input.Capture()
    # perform pose estimation (with overlay)
    poses = net.Process(img, overlay=opt.overlay)
    # print the pose results
    print("detected {:d} objects in image".format(len(poses)))
    for pose in poses:
        print(pose)
        print(pose.Keypoints)
        print('Links', pose.Links)
    # render the image
    output.Render(img)
    # update the title bar
    output.SetStatus("{:s} | Network {:.0f} FPS".format(opt.network, net.GetNetworkFPS()))
    # print out performance info
    net.PrintProfilerTimes()
    # exit on input/output EOS
    if not input.IsStreaming() or not output.IsStreaming():
        break
```

Pour observer votre position lancer le programme avec :

```
./posenet.py -camera=/dev/video0
```

Notez que les paramètres - positions de votre corps sont affichés sur le terminal avec les coordonnées **`x`**, **`y`**.

4.4 Profondeur monoculaire avec `DepthNet`

La détection de profondeur est utile pour des tâches telles que la cartographie, la navigation et la détection d'obstacles, mais elle nécessitait historiquement une **caméra stéréo** ou une caméra **RVB-D**.

Il existe maintenant des **DNN** capables de déduire une profondeur relative à partir d'une seule image monoculaire (alias **profondeur mono**). Une telle approche peut être accompli en utilisant des **réseaux entièrement convolutifs** (**FCN**).

Le modèle (par défaut) pré-entraîné utilisé dans cette application est **`fcn-mobilenet`**.

```
import jetson.inference
import jetson.utils
import argparse
import sys
from depthnet_utils import depthBuffers
# parse the command line
parser = argparse.ArgumentParser(description="Mono depth estimation on a video/image stream using
depthNet DNN.",
                                 formatter_class=argparse.RawTextHelpFormatter,
epilog=jetson.inference.depthNet.Usage() +
                                 jetson.utils.videoSource.Usage() + jetson.utils.videoOutput.Usage()
+ jetson.utils.logUsage())

parser.add_argument("input_URI", type=str, default="", nargs=&apos;?&apos;, help="URI of the input
stream")
parser.add_argument("output_URI", type=str, default="", nargs=&apos;?&apos;, help="URI of the output
stream")
parser.add_argument("--network", type=str, default="fcn-mobilenet", help="pre-trained model to load,
see below for options")
parser.add_argument("--visualize", type=str, default="input,depth", help="visualization options (can
be &apos;input&apos; &apos;depth&apos; &apos;input,depth&apos;")
parser.add_argument("--depth-size", type=float, default=1.0, help="scales the size of the depth map
visualization, as a percentage of th$
parser.add_argument("--filter-mode", type=str, default="linear", choices=["point", "linear"],
help="filtering mode used during visualiza$
parser.add_argument("--colormap", type=str, default="viridis-inverted", help="colormap to use for
visualization (default is &apos;viridis-inv$
                                 choices=["inferno", "inferno-inverted", "magma", "magma-inverted",
"parula", "parula-inverted",
                                          "plasma", "plasma-inverted", "turbo", "turbo-inverted",
"viridis", "viridis-inverted"])

try:
        opt = parser.parse_known_args()[0]
except:
        print("")
        parser.print_help()
        sys.exit(0)

# load the segmentation network
net = jetson.inference.depthNet(opt.network, sys.argv)
# create buffer manager
buffers = depthBuffers(opt)
# create video sources & outputs
input = jetson.utils.videoSource(opt.input_URI, argv=sys.argv)
output = jetson.utils.videoOutput(opt.output_URI, argv=sys.argv)
# process frames until user exits
while True:
    # capture the next image
    img_input = input.Capture()
    # allocate buffers for this size image
    buffers.Alloc(img_input.shape, img_input.format)
    # process the mono depth and visualize
    net.Process(img_input, buffers.depth, opt.colormap, opt.filter_mode)
    # composite the images
    if buffers.use_input:
        jetson.utils.cudaOverlay(img_input, buffers.composite, 0, 0)
       if buffers.use_depth:
        jetson.utils.cudaOverlay(buffers.depth, buffers.composite, img_input.width if
buffers.use_input else 0, 0)

    # render the output image
    output.Render(buffers.composite)
```

```
    # update the title bar
    output.SetStatus("{:s} | {:s} | Network {:.0f} FPS".format(opt.network, net.GetNetworkName(),
net.GetNetworkFPS()))
    # print out performance info
    jetson.utils.cudaDeviceSynchronize()
    net.PrintProfilerTimes()
    # exit on input/output EOS
    if not input.IsStreaming() or not output.IsStreaming():
        break
```

Vous pouvez lancer ce programme comme suit :

```
./depthnet.py  --camera=/dev/video0  --height=480  --width=640
```

4.5 Segmentation sémantique avec `SegNet`

La troisième capacité d'apprentissage en profondeur que nous pouvons mettre en évidence est la **segmentation sémantique**.

La segmentation sémantique est basée sur la reconnaissance d'images, sauf que **les classifications se produisent au niveau du pixel** par opposition à la classification d'images entières comme avec la reconnaissance d'images.

Ceci est accompli en «convolutionnalisant» un modèle de reconnaissance d'images pré-entraîné (comme **`Alexnet`**), qui le transforme en un modèle de segmentation entièrement convolutif capable d'un étiquetage par pixel. Par défaut l'application utilise le modèle **`fcn-resnet18-voc.`**

Utile pour la détection de l'environnement et l'**évitement des collisions**, la segmentation produit une classification dense par pixel de nombreux objets potentiels différents par scène, y compris les avant-plans et les arrière-plans des scènes.

```
import jetson.inference
import jetson.utils
import argparse
import sys
from segnet_utils import *
# parse the command line
parser = argparse.ArgumentParser(description="Segment a live camera stream using an semantic
segmentation DNN.",
                                 formatter_class=argparse.RawTextHelpFormatter,
epilog=jetson.inference.segNet.Usage() +
                                 jetson.utils.videoSource.Usage() + jetson.utils.videoOutput.Usage()
+ jetson.utils.logUsage())

parser.add_argument("input_URI", type=str, default="", nargs='?', help="URI of the input stream")
parser.add_argument("output_URI", type=str, default="", nargs='?', help="URI of the output stream")
parser.add_argument("--network", type=str, default="fcn-resnet18-voc", help="pre-trained model to
load, see below for options")
parser.add_argument("--filter-mode", type=str, default="linear", choices=["point", "linear"],
help="filtering mode used during visualiza$
parser.add_argument("--visualize", type=str, default="overlay,mask", help="Visualization options
(can be 'overlay' 'mask' 'overlay,mask'$
parser.add_argument("--ignore-class", type=str, default="void", help="optional name of class to
ignore in the visualization results (def$
parser.add_argument("--alpha", type=float, default=150.0, help="alpha blending value to use during
overlay, between 0.0 and 255.0 (defau$
parser.add_argument("--stats", action="store_true", help="compute statistics about segmentation mask
class output")

is_headless = ["--headless"] if sys.argv[0].find('console.py') != -1 else [""]

try:
        opt = parser.parse_known_args()[0]
except:
        print("")
        parser.print_help()
        sys.exit(0)

# load the segmentation network
net = jetson.inference.segNet(opt.network, sys.argv)
# set the alpha blending value
net.SetOverlayAlpha(opt.alpha)
# create buffer manager
buffers = segmentationBuffers(net, opt)
# create video sources & outputs
input = jetson.utils.videoSource(opt.input_URI, argv=sys.argv)
output = jetson.utils.videoOutput(opt.output_URI, argv=sys.argv+is_headless)
# process frames until user exits
while True:
        # capture the next image
        img_input = input.Capture()
        # allocate buffers for this size image
        buffers.Alloc(img_input.shape, img_input.format)
        # process the segmentation network
        net.Process(img_input, ignore_class=opt.ignore_class)

        # generate the overlay
```

```
        if buffers.overlay:
                net.Overlay(buffers.overlay, filter_mode=opt.filter_mode)
        # generate the mask
        if buffers.mask:
                net.Mask(buffers.mask, filter_mode=opt.filter_mode)
        # composite the images
        if buffers.composite:
                jetson.utils.cudaOverlay(buffers.overlay, buffers.composite, 0, 0)
                jetson.utils.cudaOverlay(buffers.mask, buffers.composite, buffers.overlay.width, 0)
        # render the output image
        output.Render(buffers.output)
        # update the title bar
        output.SetStatus("{:s} | Network {:.0f} FPS".format(opt.network, net.GetNetworkFPS()))
       # print out performance info
        jetson.utils.cudaDeviceSynchronize()
        net.PrintProfilerTimes()
    # compute segmentation class stats
        if opt.stats:
                buffers.ComputeStats()
        # exit on input/output EOS
        if not input.IsStreaming() or not output.IsStreaming():
                break
```

Vous pouvez lancer l'exécution de ce programme en statique – une photo (**`segnet-console.py`**), ou en dynamique avec la saisie de la vidéo par une caméra – WebCam (**`segnet-camera.py`**).

Par exemple :

```
./segnet-camera.py -camera=/dev/video0  --width=640  --height=480
```

4.6 Mini projet 1 - `my-recognition.py`

Codage de votre propre programme de reconnaissance d'image (Python)

Dans l'étape précédente, nous avons exécuté une application fournie avec le dépôt **`jetson-inference`**. Maintenant, nous allons parcourir la création d'un nouveau programme à partir de zéro en Python pour la reconnaissance d'image appelé **`my-recognition.py`**.

Ce script chargera une image arbitraire à partir du disque et la classera en utilisant l'objet **`imageNet`**.

Pour votre commodité et référence, la source complète est disponible dans le fichier python **`/examples /my-recognition.py`** du dépôt, mais le guide ci-dessous agira comme s'il résidait dans le répertoire personnel de l'utilisateur ou dans un répertoire arbitraire de votre choix.

4.6.1 Mise en place du projet

Vous pouvez stocker l'exemple **`my-recognition.py`** que nous allons créer où vous le souhaitez sur votre Jetson.
Pour plus de simplicité, ce guide le créera avec quelques images de test dans un répertoire sous le répertoire personnel de l'utilisateur situé à **`~ /my-reconnaissance-python`**.

Exécutez ces commandes à partir d'un terminal pour créer le répertoire et les fichiers requis:

```
$ cd ~/$ mkdir my-recognition-python$ cd my-recognition-python
$ touch my-recognition.py
$ chmod +x my-recognition.py
$ wget https://github.com/dusty-nv/jetson-inference/raw/master/data/images/black_bear.jpg
$ wget https://github.com/dusty-nv/jetson-inference/raw/master/data/images/brown_bear.jpg
$ wget https://github.com/dusty-nv/jetson-inference/raw/master/data/images/polar_bear.jpg
```

Certaines images de test sont également téléchargées dans le dossier avec les commandes wget ci-dessus. Ensuite, nous ajouterons le code Python du programme au fichier source vide que nous avons créé ici.

4.6.2 Code source

Ouvrez my-recognition.py dans l'éditeur de votre choix (e.g. gedit **`my-recognition.py`**).

4.6.2.1 Importation de modules

Ajoutez des instructions d'importation pour importer les modules **`jetson.inference`** et **`jetson.utils`** utilisés pour reconnaître les images et le chargement d'images. Nous chargerons également le package standard **`argparse`** pour analyser la ligne de commande.

```
import jetson.inference
import jetson.utils
import argparse
```

4.6.2.2 Analyse de la ligne de commande (`parser`)

Ensuite, ajoutez du code pour analyser le nom de fichier de l'image et le paramètre **`--network`** :

```
# parse the command line
parser = argparse.ArgumentParser()
parser.add_argument("filename", type=str, help="filename of the image to process")
parser.add_argument("--network", type=str, default="googlenet", help="model to use, can be:
googlenet, resnet-18, ect. (see --help for others)")
opt = parser.parse_args()
```

Cet exemple charge et classe une image spécifiée par l'utilisateur. Il devrait être exécuté comme ceci:

```
$ ./my-recognition.py my_image.jpg
```

Le nom de fichier d'image à charger doit être remplacé par **`my_image.jpg`**. Vous pouvez également spécifier le paramètre **`--network`** pour modifier le réseau de classification utilisé (la valeur par défaut est **GoogleNet**):

```
$ ./my-recognition.py --network = resnet-18 my_image.jpg
```

4.6.2.3 Chargement de l'image à partir du disque

Vous pouvez charger des images dans la mémoire GPU à l'aide de la fonction **`loadImageRGBA`**(). Les formats pris en charge sont : JPG, PNG, TGA et BMP.

Ajoutez cette ligne pour charger l'image avec le nom de fichier spécifié à partir de la ligne de commande:

```
img, width, height = jetson.utils.loadImageRGBA (opt.filename)
```

L'image chargée sera stockée dans la mémoire partagée qui est mappée à la fois au CPU et au GPU.
Étant donné que le processeur et le GPU intégré de Jetson partagent la même mémoire physique, les copies de mémoire (**`cudaMemcpy()`**) entre les devices ne sont pas nécessaires.
Notez que l'image est chargée au format **`float4 RGBA`**, avec des valeurs de pixels comprises entre **`0,0`** et **`255,0`**.

4.6.2.4 Chargement du réseau de reconnaissance d'images

À l'aide de l'objet **imageNet**, le code suivant chargera le modèle de classification souhaité avec **TensorRT**.
À moins que vous n'ayez spécifié un réseau différent à l'aide de l'indicateur **`--network`**, par défaut, il chargera **GoogleNet**, qui était déjà téléchargé lorsque vous avez initialement créé le répertoire jetson-inference (le modèle **ResNet-18** a été également téléchargé).

Tous les modèles de classification disponibles sont pré-entraînés sur l'ensemble de données **ImageNet ILSVRC**, qui peut reconnaître jusqu'à **1000 classes d'objets différentes**, comme différents types de fruits et légumes, de nombreuses espèces animales différentes, ainsi que des objets de tous les jours comme des véhicules , mobilier de bureau, équipement sportif, ect.

```
# charger le réseau de reconnaissance
net = jetson.inference.imageNet(opt.network)
```

4.6.2.5 Classification de l'image

Ensuite, nous allons classer l'image avec le réseau de reconnaissance en utilisant la fonction **`imageNet.Classify()`**:

```
# classer l'image
class_idx, confidence = net.Classify (img, width, height)
```

`imageNet.Classify()` accepte l'image et ses dimensions et effectue l'inférence avec **TensorRT**.
Il renvoie un tuple contenant l'**index entier de la classe** d'objets sous laquelle l'image a été reconnue, ainsi que la **valeur de confiance** (en virgule flottante) du résultat.

4.6.2.6 Interprétation des résultats

Dans la dernière étape, nous récupérons la description de la classe et imprimons les résultats de la classification:

```
# trouver la description de l'objet
class_desc = net.GetClassDesc (class_idx)
# imprimer le résultat
print("l'image est reconnue comme '{: s}' (classe # {: d}) avec {: f}% de
confiance".format (class_desc, class_idx, confidence * 100))
```

`ImageNet.Classify()` renvoie **l'index** de la classe d'objets reconnue (entre **0 et 999** pour ces modèles qui ont été entraînés sur **ILSVRC**).
Étant donné l'index de classe, la fonction **`imageNet.GetClassDesc()`** renvoie la chaîne contenant la description textuelle de cette classe. Ces descriptions sont automatiquement chargées à partir de **`ilsvrc12_synset_words.txt`**.

Et voilà ! C'est tout le code Python dont nous avons besoin pour la classification des images.

4.6.2.7 Code source

Pour être complet, voici la source complète du script Python que nous venons de créer.

```
import jetson.inference
import jetson.utils
import argparse
# parse the command line
parser = argparse.ArgumentParser()
parser.add_argument("filename", type=str, help="filename of the image to process")
parser.add_argument("--network", type=str, default="googlenet", help="model to use, can be:
googlenet, resnet-18, ect. (see --help for others)")
opt = parser.parse_args()
# load an image (into shared CPU/GPU memory)
img, width, height = jetson.utils.loadImageRGBA(opt.filename)
# load the recognition network
net = jetson.inference.imageNet(opt.network)
# classify the image
class_idx, confidence = net.Classify(img, width, height)
# find the object description
class_desc = net.GetClassDesc(class_idx)
# print out the result
print("image is recognized as '{:s}' (class #{:d}) with {:f}% confidence".format(class_desc,
class_idx, confidence * 100))
```

Le nom de fichier d'image à charger doit être remplacé par **`my_image.jpg`**. Vous pouvez également (en option) spécifiez le paramètre **`--network`** pour modifier le réseau de classification utilisé (la valeur par défaut est **GoogleNet**) :

```
$ python3 ./my-recognition.py --network=resnet-18 images/cat_0.jpg
..
image is recognized as 'tabby, tabby cat' (class #281) with 8.544922% confidencejetson.utils --
freeing CUDA mapped memoryPyTensorNet_Dealloc()

$ python3 ./my-recognition.py --network=GoogleNet images/cat_0.jpg
..
image is recognized as 'tiger cat' (class #282) with 17.700195% confidencejetson.utils -- freeing
CUDA mapped memoryPyTensorNet_Dealloc()
```

Figure 1 Image **`cat_0.jpg`**

4.7 Mini projet 7 - `my-detection.py`

Dans cette partie du lab, nous allons parcourir la création d'une application pour la **détection d'objets** en temps réel sur un flux de caméra (ou Webcam) en seulement 10 lignes de code Python.
Le programme chargera le réseau de détection avec l'objet **detectNet**, capturera des images vidéo et les traitera, puis affichera les noms (labels) des objets détectés à l'écran.

4.7.1 Code source

Tout d'abord, ouvrez l'éditeur de texte de votre choix et créez un nouveau fichier. Nous supposerons que vous l'enregistrerez dans le répertoire personnel de votre utilisateur sous le nom : **`my-detection.py`**.

4.7.1.1 Importation de modules

En haut du fichier source, nous importerons les modules Python que nous allons utiliser dans le script.
Ajoutez des instructions d'importation pour charger les modules **`jetson.inference`** et **`jetson.utils`** utilisés pour la détection d'objets et la capture de caméra.

```
import jetson.inference
import jetson.utils
```

4.7.1.2 Chargement du modèle de détection

Ensuite utilisez la ligne suivante pour créer une instance d'objet **`detectNet`** qui charge le modèle **SSD-Mobilenet-v2** avec 91 classes:

```
# charger le modèle de détection d'objet
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
```

Notez que vous pouvez remplacer la chaîne de modèle pour charger un modèle de détection différent. Nous avons également défini le seuil de détection par défaut à **0,5** à des fins d'illustration - vous pouvez le modifier plus tard si nécessaire.

4.7.1.3 Ouverture du flux de caméras

Pour vous connecter à l'appareil de capture vidéo pour le streaming, nous allons créer une instance de l'objet **`gstCamera`**:

```
camera = jetson.utils.gstCamera(1280,720,"/dev/video0") # en utilisant V4L2
```

Son constructeur accepte 3 paramètres - la largeur, la hauteur et le périphérique vidéo à utiliser.
Remplacez l'extrait suivant selon que vous utilisez une caméra MIPI CSI ou une caméra USB V4L2, ainsi que la résolution préférée:
Les caméras **MIPI CSI** sont utilisées en spécifiant l'indice du capteur ("0" ou "1", etc.)

```
camera = jetson.utils.gstCamera(1280,720,"0")
```

Les **Webcams USB** **`V4L2`** sont utilisées en spécifiant leur nœud : ("**`/dev/video0`**","**`/dev/video1`**",etc.)

```
camera = jetson.utils.gstCamera(1280,720,"/dev/video0")
```

Si nécessaire, modifiez **1280** et **720** ci-dessus à la largeur/hauteur souhaitée

4.7.1.4 Boucle d'affichage

Ensuite, nous allons créer un affichage **OpenGL** avec l'objet **`glDisplay`** et créer une boucle principale qui s'exécutera jusqu'à ce que l'utilisateur quitte:

```
display = jetson.utils.glDisplay()
while display.IsOpen():
# main loop will go here
```

Notez que le reste du code ci-dessous doit être mis en retrait sous cette boucle **`while`**.

4.7.1.5 Capture de la vidéo

La première chose qui se passe dans la boucle principale est la capture de la prochaine image vidéo-.
`camera.CaptureRGBA()` - attendra que la prochaine image ait été envoyée par la caméra, et une fois acquise par le Jetson, il la convertira au format en virgule flottante **RGBA** résidant dans la mémoire du GPU.

```
img,width,height = camera.CaptureRGBA()
```

Le résultat renvoyé est un tuple contenant une référence aux données d'image sur le GPU, ainsi que ses dimensions.

4.7.1.6 Détection d'objets

Ensuite, le réseau de détection traite l'image avec la fonction **`net.Detect()`**. Il prend l'image, la largeur et la hauteur fournies par la **`caméra.CaptureRGBA()`** et renvoie une liste de détections:

```
detections = net.Detect(img, width, height)
```

Cette fonction affiche automatiquement les résultats de détection au-dessus de l'image d'entrée.
Si vous le souhaitez, vous pouvez ajouter les coordonnées, la confiance et les informations de classe a imprimer sur le terminal pour chaque résultat de détection.
Consultez également la documentation **detectNet** pour obtenir des informations sur les différents membres de la structure **`Detection`** qui sont retournés pour y accéder directement dans une application personnalisée.

4.7.1.7 Le rendu

Enfin, nous allons visualiser les résultats avec **OpenGL** et mettre à jour le titre de la fenêtre pour afficher les performances actuelles:

```
display.RenderOnce(img,width,hight)
display.SetTitle ("Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

La fonction **`RenderOnce()`** retournera automatiquement le **`backbuffer`** qui et est utilisée lorsque nous n'avons qu'une seule image à rendre.

4.7.1.8 Code source complet

Pour être complet, voici la source complète du script Python que nous venons de créer:

```
import jetson.inference
import jetson.utils
net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.gstCamera(1280, 720, "/dev/video0") # using V4L2
display = jetson.utils.glDisplay()
while display.IsOpen():
  img, width, height = camera.CaptureRGBA()
  detections = net.Detect(img, width, height)
  display.RenderOnce(img, width, height)
  display.SetTitle("Object Detection | Network {:.0f}  FPS".format(net.GetNetworkFPS()))
```

Notez que cette version suppose que vous utilisez une **Webcam USB V4L2**. Reportez-vous à la section Ouverture du flux de caméras ci-dessus pour obtenir des informations sur sa modification pour utiliser une caméra CIP MIPI ou prendre en charge différentes résolutions.

4.7.1.9 Exécution du programme

Pour exécuter l'application que nous venons de coder, lancez-la simplement à partir d'un terminal avec l'interpréteur Python:

```
$ python my-detection.py
```

Pour modifier les résultats, vous pouvez essayer de modifier le modèle chargé avec le seuil de détection.

Table of Contents