Lab4 - Utilisation et exécution de réseaux CNN pour la reconnaissance d'images/vidéos

La carte Jetson Nano est fournie avec un OS et l'ensemble de biblithèques et APIs pour le développement des applications d'IA.

Pour commncer vous pouvez tester quelques exemple précompilés dans le répertoire : **`jetson-inference/build/aarch64/bin`.**

Il s'agit de programmes concus en C++ et ompilés, et de programmes en Python permettant de lancer des applications type : réconnaisance d'image ou d'un objet dans la vidéo captée à partir de la camera associée ou à partir d'une Webcam haute définition.

Exercice

Exétez un programmes en Python et **analysez** l'affichage **(important!) - identifiez le type de API, le type du modèle CNN, analysez les résultats de réconnaissance, etc..**

```
python3 imagenet-console images/horse_0.jpg
bako@bako-desktop:~/jetson-inference/build/aarch64/bin$ python3 imagenet-
console.py images/cat_0.jpg
jetson.inference.__init__.py
jetson.inference -- initializing Python 3.6 bindings...
jetson.inference -- registering module types...
jetson.inference -- done registering module types
jetson.inference -- done Python 3.6 binding initialization
jetson.utils.__init__.py
jetson.utils -- initializing Python 3.6 bindings...
jetson.utils -- registering module functions...
jetson.utils -- done registering module functions
jetson.utils -- registering module types...
jetson.utils -- done registering module types
jetson.utils -- done Python 3.6 binding initialization
[image] loaded 'images/cat_0.jpg'  (481 x 640, 3 channels)
jetson.inference -- PyTensorNet_New()
jetson.inference -- PyImageNet_Init()
jetson.inference -- imageNet loading network using argv command line params
jetson.inference -- imageNet.__init__() argv[0] = 'imagenet-console.py'
jetson.inference -- imageNet.__init__() argv[1] = 'images/cat_0.jpg'

imageNet -- loading classification network model from:
         -- prototxt     networks/googlenet.prototxt
         -- model        networks/bvlc_googlenet.caffemodel
         -- class_labels networks/ilsvrc12_synset_words.txt
         -- input_blob   'data'
         -- output_blob  'prob'
         -- batch_size   1
[TRT]   TensorRT version 5.1.6
[TRT]   loading NVIDIA plugins...
[TRT]   Plugin Creator registration succeeded - GridAnchor_TRT
[TRT]   Plugin Creator registration succeeded - NMS_TRT
[TRT]   Plugin Creator registration succeeded - Reorg_TRT
[TRT]   Plugin Creator registration succeeded - Region_TRT
[TRT]   Plugin Creator registration succeeded - Clip_TRT
[TRT]   Plugin Creator registration succeeded - LReLU_TRT
[TRT]   Plugin Creator registration succeeded - PriorBox_TRT
[TRT]   Plugin Creator registration succeeded - Normalize_TRT
[TRT]   Plugin Creator registration succeeded - RPROI_TRT
[TRT]   Plugin Creator registration succeeded - BatchedNMS_TRT
[TRT]   completed loading NVIDIA plugins.
[TRT]   detected model format - caffe  (extension '.caffemodel')
[TRT]   desired precision specified for GPU: FASTEST
[TRT]   requested fasted precision for device GPU without providing valid
calibrator, disabling INT8
[TRT]   native precisions detected for GPU:  FP32, FP16
```

```
[TRT]   selecting fastest native precision for GPU:  FP16
[TRT]   attempting to open engine cache file
networks/bvlc_googlenet.caffemodel.1.1.GPU.FP16.engine
[TRT]   loading network profile from engine cache...
networks/bvlc_googlenet.caffemodel.1.1.GPU.FP16.engine
[TRT]   device GPU, networks/bvlc_googlenet.caffemodel loaded
[TRT]   device GPU, CUDA engine context initialized with 2 bindings
[TRT]   binding -- index   0
               -- name    'data'
               -- type    FP32
               -- in/out  INPUT
               -- # dims  3
               -- dim #0  3 (CHANNEL)
               -- dim #1  224 (SPATIAL)
               -- dim #2  224 (SPATIAL)
[TRT]   binding -- index   1
               -- name    'prob'
               -- type    FP32
               -- in/out  OUTPUT
               -- # dims  3
               -- dim #0  1000 (CHANNEL)
               -- dim #1  1 (SPATIAL)
               -- dim #2  1 (SPATIAL)
[TRT]   binding to input 0 data  binding index:  0
[TRT]   binding to input 0 data  dims (b=1 c=3 h=224 w=224) size=602112
[TRT]   binding to output 0 prob  binding index:  1
[TRT]   binding to output 0 prob  dims (b=1 c=1000 h=1 w=1) size=4000
device GPU, networks/bvlc_googlenet.caffemodel initialized.
[TRT]   networks/bvlc_googlenet.caffemodel loaded
imageNet -- loaded 1000 class info entries
networks/bvlc_googlenet.caffemodel initialized.
class 0103 - 0.017395  (platypus, duckbill, duckbilled platypus, duck-billed
platypus, Ornithorhynchus anatinus)
class 0231 - 0.016022  (collie)
class 0232 - 0.020340  (Border collie)
class 0239 - 0.023575  (Bernese mountain dog)
class 0240 - 0.025116  (Appenzeller)
class 0264 - 0.015289  (Cardigan, Cardigan Welsh corgi)
class 0281 - 0.066711  (tabby, tabby cat)
class 0282 - 0.177002  (tiger cat)
class 0283 - 0.014145  (Persian cat)
class 0285 - 0.048584  (Egyptian cat)
class 0338 - 0.026627  (guinea pig, Cavia cobaya)
class 0463 - 0.011139  (bucket, pail)
class 0470 - 0.014305  (candle, taper, wax light)
class 0572 - 0.011772  (goblet)
class 0611 - 0.037415  (jigsaw puzzle)
class 0711 - 0.010757  (perfume, essence)
class 0773 - 0.016205  (saltshaker, salt shaker)
class 0868 - 0.028229  (tray)
class 0898 - 0.025696  (water bottle)
class 0899 - 0.057495  (water jug)
image is recognized as 'tiger cat' (class #282) with 17.700195% confidence


[TRT]   ------------------------------------------------
[TRT]   Timing Report networks/bvlc_googlenet.caffemodel
[TRT]   ------------------------------------------------
[TRT]   Pre-Process   CPU   0.06911ms  CUDA   0.26167ms
[TRT]   Network       CPU  14.82312ms  CUDA  14.45927ms
[TRT]   Post-Process  CPU   0.40599ms  CUDA   0.60948ms
[TRT]   Total         CPU  15.29823ms  CUDA  15.33042ms
[TRT]   ------------------------------------------------

[TRT]   note -- when processing a single image, run 'sudo jetson_clocks' before
                to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
```

4.1 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.

4.1.1 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.

4.1.2 Code source

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

4.1.2.1 Importation de modules

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

```
importer jetson.inference
importer jetson.utils
importer argparse
```

4.1.2.2 Analyse de la ligne de commande

Ensuite, ajoutez du code passe-partout pour analyser le nom de fichier de l'image et un paramètre **`--network`** (facultatif):

```
# 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 éventuellement 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
```

4.1.2.3 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**.

4.1.2.4 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)
```

4.1.2.5 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.

4.1.2.6 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.

4.1.2.7 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))
```

The desired image filename to be loaded should be substituted for **`my_image.jpg`**. You can also optionally specify the **`--network`** parameter to change the classification network that's used (the default is **`GoogleNet`**):

```
$ python3 ./my-recognition.py --network=resnet-18 images/cat_0.jpg
..
image is recognized as 'tabby, tabby cat' (class #281) with 8.544922% confidence
jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
$ python3 ./my-recognition.py --network= GoogleNet images/cat_0.jpg
..
image is recognized as 'tiger cat' (class #282) with 17.700195% confidence
jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
```

Figure 1.1 Image **`cat_0.jpg`**

Lab 4.2 Codage de votre propre programme de détection d'objets

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.

4.2.1 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`**.

4.2.1.1 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
```

4.2.1.2 Chargement du modèle de détection

Utilisez ensuite 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.

4.2.1.3 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")
```

La **largeur** et la **hauteur** doivent être une résolution prise en charge par la caméra. Recherchez les résolutions disponibles avec les commandes suivantes:

```
$ sudo apt-get install v4l-utils
$ v4l2-ctl –list-formats-ext
```

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

4.2.1.4 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.**

4.2.1.5 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.

4.2.1.6 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 **Deection** qui sont retournés pour y accéder directement dans une application personnalisée.

4.2.1.7 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, largeur, hauteur)
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.

4.2.1.8 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**.**

4.2.3 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**.