

ALHABAJ Mahmod
BELDA Tom
INGARAO Adrien
MAGGOUH Naoufal

DI 4
2020-2021

Professeur Référent: *CONTE Donatello*

Projet Génie Logiciel

Découverte Nvidia DeepStream SDK



Table des matières:

Introduction	5
I - Explications générales de DeepStream	6
II - Installation	8
0 - Prérequis :	8
1 - Vérification de la version du driver nvidia et de cuda	8
2 - Installation de Cuda 10.2	8
3 - Installation de CuDNN	11
4 - Installation de TensorRT	13
5 - Installation des dépendances	14
6 - Installation Deepstream 5.0	15
7 - Installation Deepstream Python Apps	16
III - Développement d'un modèle d'IA	18
1- Training de l'IA	18
2 - Validation	18
3 - Test	18
4 - Précision sur l'entraînement cas RetinaNet	19
5 - Feature map	19
6 - Anchor box	20
7 - Subnet	21
8 - Focal loss	21
IV - Entraînement d'un modèle RetinaNet	22
Partie 1 :	22
1 - Préparation des données	22
1 - Cloner le dépôt	23
2 - Demarrer le docker NGC PyTorch	23
3 - Téléchargement depuis Open Images	24
4 - Explications et exemple	24
5 - Conversion des "validation annotations" en JSON	26
6 - Conversion des "training annotation" en JSON	27
7 - Copie des images training et validation	28
2 - Entraînement du modèle RetinaNet :	29
1 - Cloner RetinaNet depuis le dépôt NVIDIA sur GitHub	29
2 - Modification des hyperparameters (Optionnel)	29
3 - Installation de RetinaNet (OTDK)	29
4 - Entraînement du modèle	30
5 - Evaluation	31
3 - Exporter en ONNX	32

Partie 2 :	33
1 - Construction de l'engine de TensorRT	33
1 - Construction de l'API RetinaNet en C++	33
2 - Génération du fichier "engine" avec TensorRT	34
2 - Déploiement avec DeepStream	35
1 - Construire le parseur personnalisé pour RetinaNet	35
2 - Paramétrer et lancer l'application	37
 V - Problèmes rencontrés	 42
 Sources	 45

Introduction

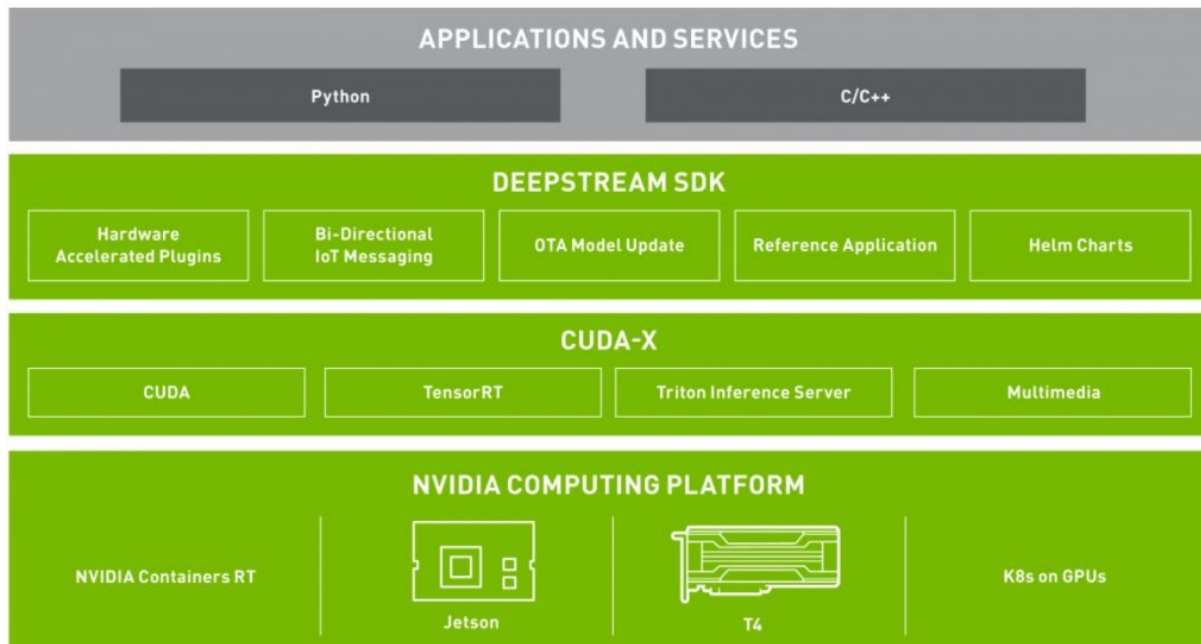
Avec l'évolution de l'intelligence artificielle et l'utilisation de celle-ci pour améliorer notre quotidien, nous rendons compte du potentielle de cette technologie.

Ce projet constitue une découverte d'un outil d'analyse d'images et de vidéos en temps réel fourni par Nvidia: Nvidia DeepStream SDK. Ce SDK (Software Development Toolkit) utilise une IA (Intelligence artificielle) pour augmenter la capacité de détection de l'application. Notre groupe d'étudiants de 4ème année à Polytech Tours en Informatique est constitué de ALHABAJ Mahmod, BELDA Tom, INGARAO Adrien ainsi que de MAGGOUH Naoufal et nous avons pour but principal de découvrir et de comprendre comment fonctionne cet outil. De plus, nous devons aussi créer une application en C++ ou en Python qui utilise DeepStream. Nous devons aussi rédiger un guide d'utilisation et une documentation claire pour pouvoir réutiliser le projet plus tard.

Dans un premier temps, nous allons expliquer concrètement ce qu'est DeepStream pour mieux comprendre l'enjeu du projet. Ensuite, nous présentons un tutoriel permettant d'installer ce SDK dans votre machine. Pour utiliser l'IA de deepstream, nous verrons comment fonctionne un modèle d'IA puis comment l'entraîner. Enfin, nous évoquerons les problèmes que nous avons rencontré lors du projet.

I - Explications générales de DeepStream

Deepstream est un closed source SDK, une toolbox complète qui fournit des outils de traitement de vidéo et la vision par ordinateur. Elle est destinée à construire des applications et des services IVA (Intelligent Video Analytics).



La librairie permet le développement en C/C++ ou, via Python Bindings, en Python3. De plus, celle-ci est accompagnée de plusieurs exemples d'applications pour aider les débutants (*fichiers samples*).

DeepStream contient de base plusieurs plugins dont différents plugins d'accélération matérielle tels que VIC, GPU, DLA, NVDEC ou NVENC. En effectuant toutes les opérations lourdes à l'aide des accélérateurs correspondants, DeepStream atteint un ratio précision/performance très intéressant en temps réel pour la plupart de types de modèles d'IA. Il permet de faire fonctionner un modèle d'intelligence artificielle sur un flux vidéo tout en optimisant l'encodage, le décodage et la modification du format de l'image pour permettre une augmentation générale des performances.

Nous retrouvons un tableau de données depuis le site de DeepStream SDK montrant les statistiques de performances obtenu pendant l'exécution de DeepStream sur une Jetson Xavier/Nano (mini carte-pc Nvidia --même idée que raspberryPI-- utilisée pour le développement d'outils IA).

Nous remarquons un "Model Accuracy" allant de 80% à 96% dans le cas de l'utilisation du modèle ResNet, avec en moyenne, au moins 10 fps pour la plus faible Jetson Nano sur un flux vidéo de 1080p/30fps et le modèle ResNet34 en INT8 avec une résolution d'inférence de 960x544.

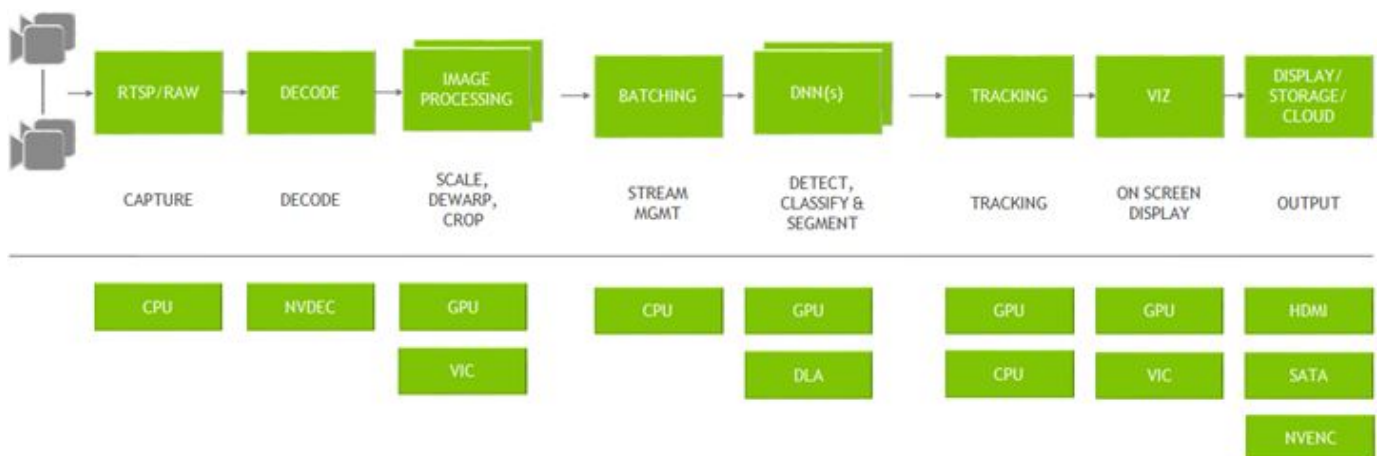
Model Architecture	Inference Resolution	Precision	Model Accuracy	Jetson Nano	Jetson Xavier NX			Jetson AGX Xavier			T4
				GPU (FPS*)	GPU (FPS)	DLA1 (FPS)	DLA2 (FPS)	GPU (FPS)	DLA1 (FPS)	DLA2 (FPS)	GPU (FPS)
PeopleNet-ResNet18	960x544	INT8	80%	14	218	72	72	384	94	94	1105
PeopleNet-ResNet34	960x544	INT8	84%	10	157	51	51	272	67	67	807
TrafficCamNet-ResNet18	960x544	INT8	84%	19	261	105	105	464	140	140	1300
DashCamNet-ResNet18	960x544	INT8	80%	18	252	102	102	442	133	133	1280
FaceDetect-IR-ResNet18	384x240	INT8	96%	95	1188	570	570	2006	750	750	2520

DeepStream a aussi accès à plusieurs librairies NVIDIA, on peut par exemple citer TensorRT, un SDK d'optimisation de l'inférence pour le deep learning, CUDA ou encore Triton.

DeepStream supporte plusieurs modèles d'IA passant par la détection d'objets jusqu'à la classification et segmentation d'images avec des méthodes out of the box tels que SSD, YOLO, FasterRCNN, et MaskRCNN. Il permet aussi l'utilisation des fonctions et librairies d'OpenCV. Ces différents modèles peuvent être déployés sur des frameworks comme PyTorch ou TensorFlow.

Grâce à tous ces outils, DeepStream permet à l'utilisateur de choisir lui-même les différents éléments qui composeront sa solution tout en rendant leur utilisation simplifiée.

DeepStream est basé sur le framework GStreamer. Le schéma ci-dessous représente l'architecture typique d'une application d'analyse vidéo faite sous DeepStream.



Tout à gauche on trouve l'étape de capture de l'image, ensuite on peut voir les différentes étapes et plugins utilisés puis à la fin, l'affichage final. C'est sur ces différentes étapes utilisant différents matériels que DeepStream vient appliquer ses plugins d'optimisation en plus pour accélérer le processus.

II - Installation

0 - Prérequis :

- Ubuntu 18.04
- Python 3.6
- Avoir un GPU Nvidia compatible avec la version 450+ du driver

1 - Vérification de la version du driver nvidia et de cuda

Faire la commande suivante pour vérifier la version du driver nvidia et de Cuda installées sur l'ordinateur.

```
$ nvidia-smi
```

Nous aurons besoin du driver minimum nvidia 450 et de Cuda version 10.2.

Si vous n'avez pas la bonne version du driver Nvidia vous pouvez l'installer en suivant l'installation de Cuda normalement (voir tutoriel installation Cuda). Quant à Cuda, si vous avez une version trop vieille ou même plus récente, vous pouvez suivre le tutoriel ci-après afin d'avoir la version 10.2 nécessaire.

2 - Installation de Cuda 10.2

Rendez vous sur le site web de Nvidia à la page de cuda 10.2

(https://developer.nvidia.com/cuda-10.2-download-archive?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1804&target_type=deblocal) :

[Home](#) > [High Performance Computing](#) > [CUDA Toolkit](#) > [CUDA Toolkit Archive](#) > [CUDA Toolkit 10.2 Download](#)

CUDA Toolkit 10.2 Download

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX			
Architecture	x86_64	ppc64le				
Distribution	Fedora	OpenSUSE	RHEL	CentOS	SLES	Ubuntu
Version	18.04	16.04				
Installer Type	runfile (local)	deb (local)	deb (network)	cluster (local)		

Faire les instructions d'installation en lignes de commande données sur le site web :

```
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cuda-ubuntu1804.pin
$ sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget
https://developer.download.nvidia.com/compute/cuda/10.2/Prod/local_installers/cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
$ sudo dpkg -i
cuda-repo-ubuntu1804-10-2-local-10.2.89-440.33.01_1.0-1_amd64.deb
$ sudo apt-key add /var/cuda-repo-10-2-local-10.2.89-440.33.01/7fa2af80.pub
$ sudo apt-get update
```

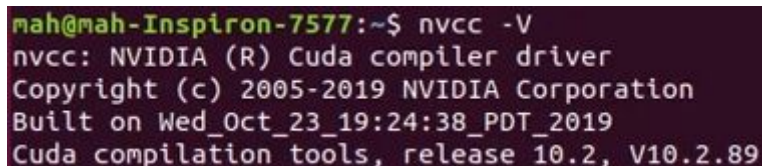
Ensuite selon votre situation, faite la première commande s'il vous manque le driver nvidia ou la seconde commande si vous avez déjà le driver nvidia et souhaitez directement installer cuda 10.2:

```
$ sudo apt-get install cuda
$ sudo apt-get install cuda-toolkit-10-2
```

Pour vérifier l'installation il suffit d'écrire les commandes ci-dessous :

```
$ sudo updatedb
$ nvcc -V
```

Si l'installation s'est bien effectuée vous devriez voir quelque chose comme ça :



```
mah@mah-Inspiron-7577:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Oct_23_19:24:38_PDT_2019
Cuda compilation tools, release 10.2, V10.2.89
```

Si la commande n'est pas reconnue, c'est que l'installation ne s'est probablement pas faite au bon endroit ou bien il faut ajouter la commande nvcc au PATH, donc continuer la suite du tutoriel pour pouvoir régler ce problème.

Faites la commande suivante :

```
$ /usr/local/cuda/bin/nvcc -V
```

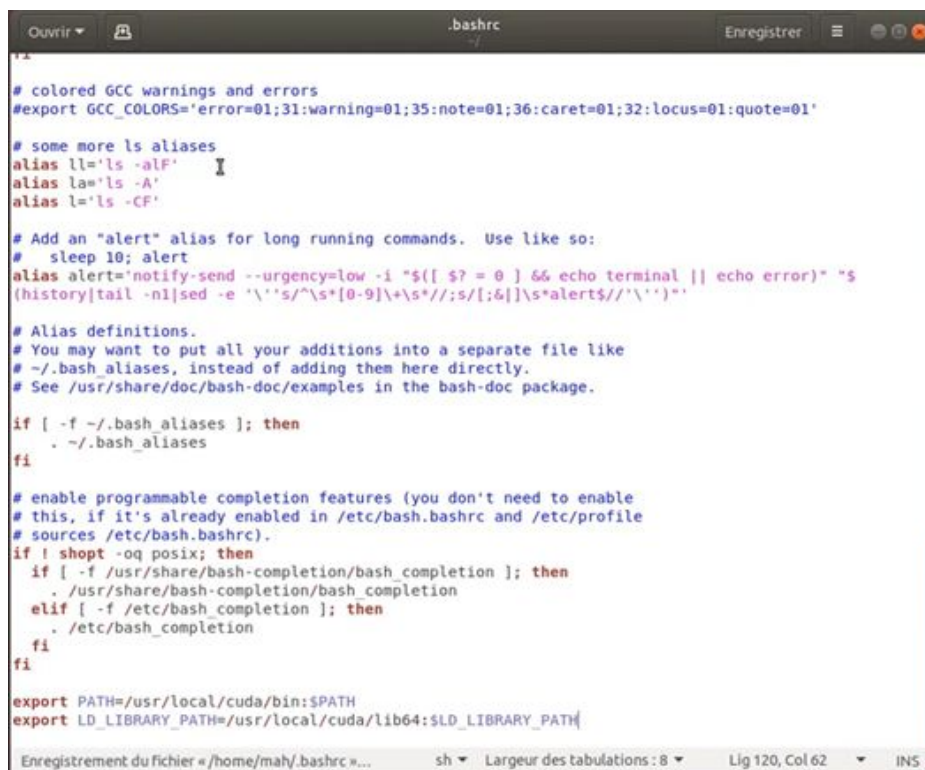
Si la commande affiche le résultat attendu il faut l'ajouter au PATH en éditant le fichier .bashrc :

```
$ sudo gedit ~/.bashrc
```

Le fichier en question s'ouvre alors. Vous devez ensuite coller ces deux lignes à la fin du fichier :

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Nous avons alors quelque chose qui ressemble à ça :



```
.bashrc
# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -lF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "${?} = 0" && echo terminal || echo error" "$
(history|tail -nl|sed -e '\''s/^\[0-9\]\+\s*//;s/[:&]\s*alert$/'\''')"

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Ce fichier s'exécute lors de la connexion au système. Pour éviter de redémarrer le système pour prendre en compte le changement effectué, nous pouvons directement exécuter la commande ci-dessous qui le permet aussi :

```
$ source ~/.bashrc
```

Maintenant vous pouvez retaper votre commande « nvcc -V » et ça devrait fonctionner ! Vous avez ainsi la version 10.2 de cuda.

Faire la commande suivante pour associer les nouvelles librairies au compilateur C :

```
$ sudo ldconfig
```

3 - Installation de CuDNN

Rendez-vous sur le site web de Nvidia à la page de CDNN Download pour télécharger la version 7.6.5 pour Cuda 10.2

(<https://developer.nvidia.com/rdp/cudnn-download>) :

cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☒ I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

Download cuDNN v8.0.5 [November 9th, 2020], for CUDA 11.1

Download cuDNN v8.0.5 [November 9th, 2020], for CUDA 11.0

Download cuDNN v8.0.5 [November 9th, 2020], for CUDA 10.2

Download cuDNN v8.0.5 [November 9th, 2020], for CUDA 10.1

Archived cuDNN Releases



Télécharger les fichiers bibliothèques correspondantes à la version 18.04 d'Ubuntu (normalement celle que vous avez actuellement) :

Download cuDNN v8.0.1 RC2 [June 26th, 2020], for CUDA 10.2

Download cuDNN v7.6.5 [November 18th, 2019], for CUDA 10.2

Library for Windows, Mac, Linux, Ubuntu and RedHat/Centos(x86_64architecture)

cuDNN Library for Windows 7

cuDNN Library for Windows 10

cuDNN Library for Linux

cuDNN Runtime Library for Ubuntu18.04 (Deb)

cuDNN Developer Library for Ubuntu18.04 (Deb)

cuDNN Code Samples and User Guide for Ubuntu18.04 (Deb)

cuDNN Runtime Library for Ubuntu16.04 (Deb)

cuDNN Developer Library for Ubuntu16.04 (Deb)

cuDNN Code Samples and User Guide for Ubuntu16.04 (Deb)



Installer ensuite **les 3 packages** à l'aide la commande « \$ sudo dpkg -i » suivie du nom du fichier dans l'ordre, comme ceci :

```
$ sudo dpkg -i libcudnn7_7.6.5.32-1+cuda10.2_amd.deb
$ sudo dpkg -i libcudnn7-dev_7.6.5.32-1+cuda10.2_amd64.deb
$ sudo dpkg -i libcudnn7-doc_7.6.5.32-1+cuda10.2_amd64.deb
$ sudo updatedb
```

Ensuite pour vérifier l'installation de CUDNN :

```
$ cd /usr/src/cudnn_samples_v7/mnistCUDNN
$ sudo make
$ ./mnistCUDNN
```

Si la commande se lance c'est que l'installation s'est bien effectuée sur cuda et sur cudnn.

Vous devriez alors avoir quelque chose qui ressemble à ça, avec une version de CUDNN 7.6.5 :

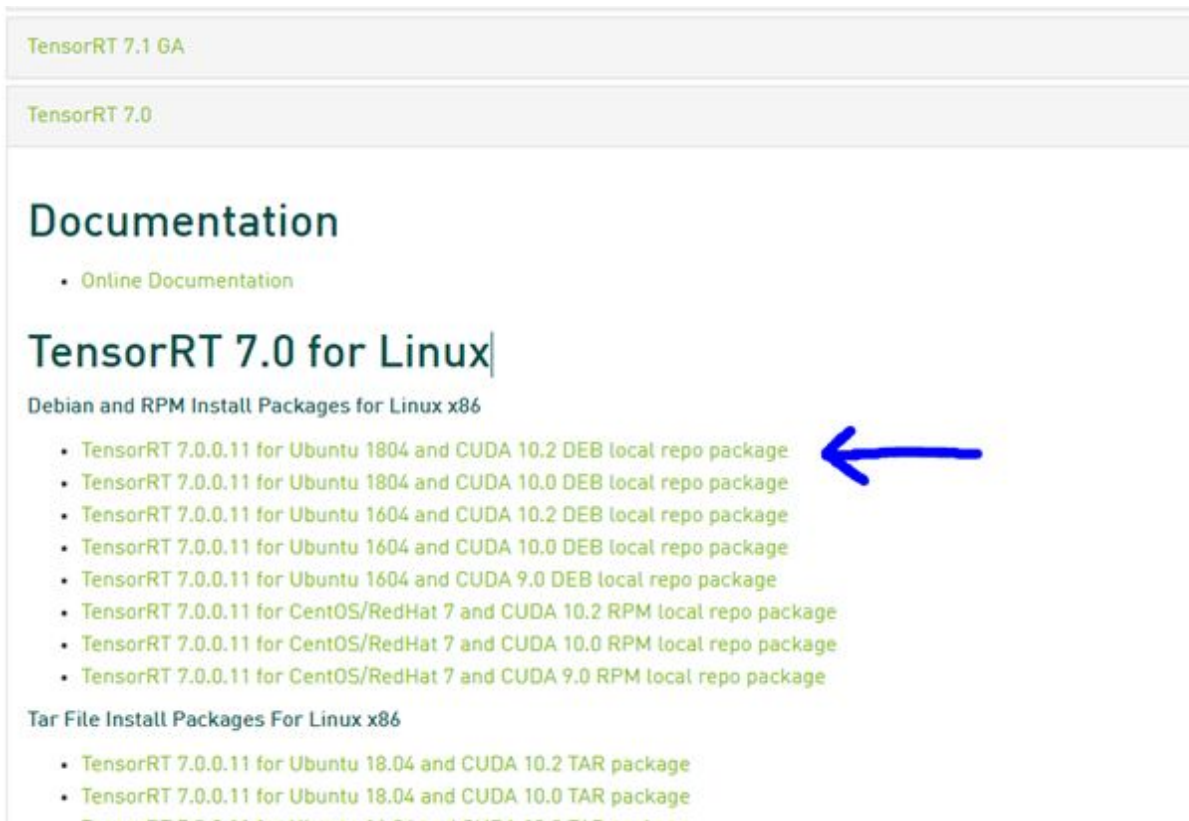
```
mah@mah-Inspiron-7577:/usr/src/cudnn_samples_v7/mnistCUDNN$ ./mnistCUDNN
cudnnGetVersion() : 7605 , CUDNN_VERSION from cudnn.h : 7605 (7.6.5)
Host compiler version : GCC 7.5.0
There are 1 CUDA capable devices on your machine :
device 0 : sms 10 Capabilities 6.1, SmClock 1341.5 Mhz, MemSize (Mb) 6078, MemClock 4004.0 Mhz, Ecc=0, boardGroupID=0
Using device 0

Testing single precision
Loading image data/one_28x28.pgm
Performing forward propagation ...
Testing cudnnGetConvolutionForwardAlgorithm ...
Fastest algorithm is Algo 1
Testing cudnnFindConvolutionForwardAlgorithm ...
^^^^ CUDNN_STATUS_SUCCESS for Algo 0: 0.017408 time requiring 0 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 2: 0.034816 time requiring 57600 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 1: 0.036864 time requiring 3464 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 4: 0.092160 time requiring 207360 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 5: 0.141312 time requiring 203008 memory
Resulting weights from Softmax:
0.0000000 0.9999399 0.0000000 0.0000000 0.0000561 0.0000000 0.0000012 0.0000017 0.0000010 0.0000000
000000
Loading image data/three_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000000 0.0000000 0.9999288 0.0000000 0.0000711 0.0000000 0.0000000 0.0000000 0.0000000
000000
Loading image data/five_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000000 0.0000000 0.0000002 0.0000000 0.9999820 0.0000154 0.0000000 0.0000012 0.0000006
000006

Result of classification: 1 3 5
Test passed!
```

4 - Installation de TensorRT

Rendez-vous sur le site web de Nvidia à la page de TensorRT Download pour télécharger la version 7.0, toujours pour Ubuntu 18.04 et CUDA 10.2 (<https://developer.nvidia.com/nvidia-tensorrt-7x-download>) :



Installer le package :

```
$ sudo dpkg -i  
nv-tensorrt-repo-ubuntu1804-cuda10.2-trt7.0.0.11-ga-20191216_1-1_amd64.deb  
$ sudo apt-key add  
/var/nv-tensorrt-repo-cuda10.2-trt7.0.0.11-ga-20191216/7fa2af80.pub
```

La deuxième commande devrait vous sortir un « OK » dans le terminal.

```
$ sudo apt update  
$ sudo apt install tensorrt  
$ sudo apt install python3-libnvinfer-dev  
$ sudo apt-get install uff-converter-tf
```


Pour vérifier l'installation de TensorRT 7.0 :

```
$ dpkg -l | grep TensorRT
```

Vous devez alors obtenir une liste bibliothèque « 7.0.0-1-cuda10.2 », comme ci-dessous :

```
mah@mah-Inspiron-7577:~/Téléchargements$ dpkg -l | grep TensorRT
ii graphsurgeon-tf                                7.0.0-1+cuda10.2          amd64      GraphSurgeon
for TensorRT package
ii libnvinfer-bin                                7.0.0-1+cuda10.2          amd64      TensorRT bina
ries
ii libnvinfer-dev                                7.0.0-1+cuda10.2          amd64      TensorRT deve
lopment libraries and headers
ii libnvinfer-doc                                7.0.0-1+cuda10.2          all        TensorRT docu
mentation
ii libnvinfer-plugin-dev                         7.0.0-1+cuda10.2          amd64      TensorRT plug
in libraries
ii libnvinfer-plugin7                           7.0.0-1+cuda10.2          amd64      TensorRT plug
in libraries
ii libnvinfer-samples                           7.0.0-1+cuda10.2          all        TensorRT samp
les
ii libnvinfer7                                   7.0.0-1+cuda10.2          amd64      TensorRT runt
ime libraries
ii libvonnxparsers-dev                          7.0.0-1+cuda10.2          amd64      TensorRT ONNX
libraries
ii libvonnxparsers7                             7.0.0-1+cuda10.2          amd64      TensorRT ONNX
libraries
ii libvnparsers-dev                             7.0.0-1+cuda10.2          amd64      TensorRT pars
ers libraries
ii libvnparsers7                               7.0.0-1+cuda10.2          amd64      TensorRT pars
ers libraries
ii python3-libnvinfer                           7.0.0-1+cuda10.2          amd64      Python 3 bind
ings for TensorRT
ii python3-libnvinfer-dev                       7.0.0-1+cuda10.2          amd64      Python 3 deve
lopment package for TensorRT
ii tensorrt                                     7.0.0-1+cuda10.2          amd64      Meta package
of TensorRT
ii uff-converter-tf                             7.0.0-1+cuda10.2          amd64      UFF converter
for TensorRT package
```

5 - Installation des dépendances

GStreamer.

Suivre simplement les lignes de commande suivantes :

```
$ sudo apt install libssl1.0.0
$ sudo apt install libssl-dev
$ sudo apt install libgstreamer1.0
$ sudo apt install gstreamer1.0-tools
$ sudo apt install gstreamer1.0-plugins-good
$ sudo apt install gstreamer1.0-plugins-bad
$ sudo apt install gstreamer1.0-plugins-ugly
$ sudo apt install gstreamer1.0-libav
$ sudo apt install libgstrtspserver-1.0
$ sudo apt install libjansson4
$ sudo apt update
```

Librdkafka:

Suivre simplement les lignes de commande suivantes :

```
$ sudo apt-get install librdkafka-dev
```

6 - Installation Deepstream 5.0

Rendez-vous sur le site web de Nvidia « Deepstream Getting Started » pour télécharger la version 5.0, toujours pour Ubuntu 18.04 et CUDA 10.2, de Deepstream (<https://developer.nvidia.com/deepstream-getting-started>) :

Downloads

☒ I Agree To the Terms of the **NVIDIA DeepStream SDK 5.0.1 Software License Agreement**

DeepStream 5.0.1 for Servers and Workstations

This release supports Tesla T4 and V100.

Download .tar

Download .deb

Get NGC Container for Data Center

DeepStream 5.0.1 for Jetson

This release supports Jetson TX1, TX2, Nano, NX and AGX Xavier.

Prerequisite: DeepStream SDK 5.0.1 requires the installation of **JetPack 4.4**.

Download .tar

Download .deb

Get NGC Container for Edge

Archived Version - Jetson

Archived Version - Tesla

Installer le package :

```
$ sudo mkdir -p /opt/nvidia/deepstream/deepstream-5.0/lib
$ sudo dpkg -i deepstream-5.0_5.0.1-1_amd64.deb
$ sudo updatedb
$ sudo cp /usr/lib/x86_64-linux-gnu/librdkafka*
/opt/nvidia/deepstream/deepstream-5.0/lib/
$ sudo ldconfig
$ sudo apt-get install libgstreamer-plugins-base1.0-dev
```

Il faut redémarrer ensuite l'ordinateur pour avoir un système fonctionnel après toutes ces installations.

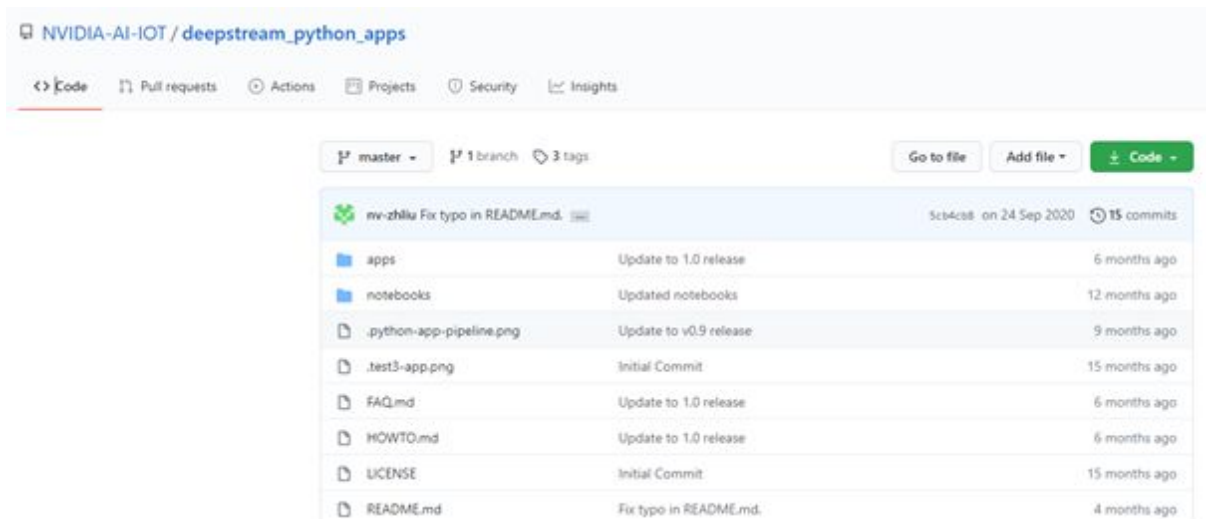
Vous pouvez ensuite tester un sample préinstallé de deepstream :

```
$ cd /opt/nvidia/deepstream/deepstream-5.0/samples/configs/deepstream-app/  
$ deepstream-app -c source1_usb_decinfer_resnet_int8.txt
```

Cet exemple se sert de la caméra de votre ordinateur comme source vidéo, coiffez-vous bien avant de lancer l'application ! (cf partie erreurs en cas de problème de lancement).

7 - Installation Deepstream Python Apps

Rendez-vous sur la page github de NVIDIA AI IOT pour cloner les applications python de deepstream (<https://developer.nvidia.com/deepstream-getting-started>) :



Pour les cloner suivez les commandes ci-dessous :

```
$ git clone https://github.com/NVIDIA-AI-IOT/deepstream_python_apps.git
```

Ensuite pour faire fonctionner ces applications, il faut installer ces dépendances :

```
$ sudo apt-get install python3-gi python-gst-1.0  
$ sudo apt-get install libgirepository1.0-dev  
$ sudo apt-get install libcairo2-dev gir1.2-gstreamer-1.0  
$ export GST_LIBS="-lgstreamer-1.0 -lgobject-2.0 -lglib-2.0"  
$ export GST_CFLAGS="-pthread -I/usr/include/gstreamer-1.0  
-I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include"
```


Installation de Gstreamer pour python:

```
$ git clone https://github.com/GStreamer/gst-python.git
$ cd gst-python
$ git checkout 1a8f48a
$ export PYTHON=/usr/bin/python3
$ pip3 install pycairo
$ pip3 install PyGObject
$ ./autogen.sh --disable-gtk-doc --noconfigure
$ sudo apt install python-gi-dev
$ ./configure --with-libpython-dir=/usr/lib/x86_64-linux-gnu
$ make
$ sudo make install
$ sudo ldconfig
```

Pour ensuite tester une des applications sample python, il faut aller dans le répertoire dans lequel vous les avez installé (il s'agissait de la commande `$ git clone https://github.com/GStreamer/gst-python.git`). Vous devrez donc être dans le dossier « `deepstream_pyton_apps` ». Ensuite lancer ces commandes :

```
$ cd apps/deepstream-test1/
$ python3
>> import sys
>> sys.path.append('../')
>> import gi
>> gi.require_version('Gst', '1.0')
>> from gi.repository import GObject, Gst
>> from common.is_aarch_64 import is_aarch64
>> common.bus_call import bus_call
>> import pyds
```

Pour tester le sample de la caméra de votre ordinateur portable :

```
$ cd deepstream_python_apps/apps/deepstream-test1-usbcam
$ python3 deepstream_test1_usb.py /dev/video0
```

III - Développement d'un modèle d'IA

Le développement d'une IA peut se séparer en 3 phases principales: tout d'abord, le training de l'IA, ensuite la validation puis enfin la phase de tests.

1- Training de l'IA

La première phase est la phase de training, c'est pendant cette phase que l'ia apprend. On donne à l'IA une grande quantité de données d'entraînement, ces données contiennent déjà la réponse recherchée par l'IA, celle-ci se contente d'observer les différents résultats et de trouver un pattern répétitif pour s'en servir de recette. Plus on donne de données à l'IA pour son entraînement, plus elle aura la chance de préciser sa recette.

2 - Validation

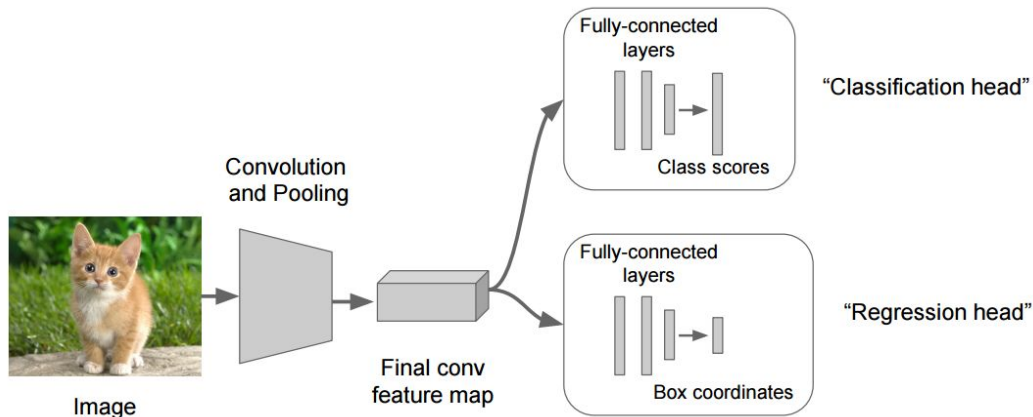
Une fois l'entraînement terminé, on passe à la phase de validation. Pendant cette phase, on donne premièrement la donnée sans résultat à l'IA qui applique sa recette puis on lui donne la bonne réponse et on voit si l'IA a réussi ou non. Tant que cette phase n'est pas réussie, l'IA retourne à la phase d'entraînement. Comme pour les données de test, les données de validation sont différentes des données d'entraînement pour éviter que la recette de l'IA soit uniquement basée sur la mémorisation.

3 - Test

Une fois que les résultats de la phase de validation vous convient, l'IA passe à la phase de test. On donne à l'IA de toutes nouvelles données sur lesquelles celle-ci appliquera ce qu'elle aura appris pendant toutes les phases de Training et de validation. A la fin de cette phase, si le résultat est satisfaisant, l'IA peut passer à la phase supérieure et commencer à être produite.

4 - Précision sur l'entraînement cas RetinaNet

Pour valider la présence ou non d'un objet sur l'image, les modèles d'IA doivent passer à travers plusieurs étapes pour avoir des résultats précis tout en restant relativement rapides.



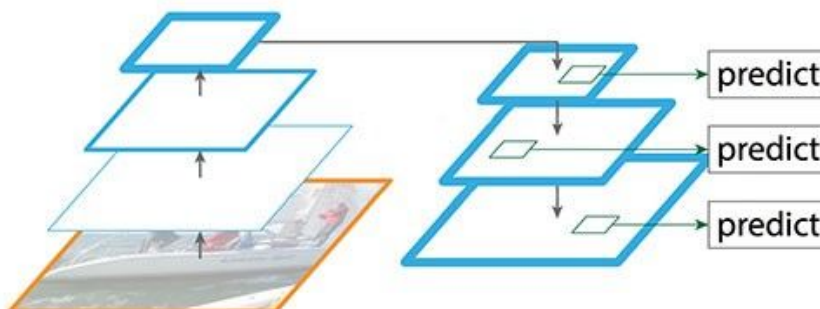
Ce schéma résume l'architecture générale de la recherche d'objet dans une image.

Premièrement, l'image passe dans un premier élément appelé le backbone. Le backbone prend l'image et génère une feature map. La deuxième partie du modèle peut être composée d'un ou plusieurs Subnet qui recherchent les éléments sur la feature map.

5 - Feature map

Les feature map sont des versions des images sur lesquelles sont passés plusieurs filtres avec pour objectif de faire ressortir les éléments importants de l'image. Cela permet de faire ressortir par exemple les ombres ou les couleurs dominantes d'une image rendant la reconnaissance de patterns plus simple.

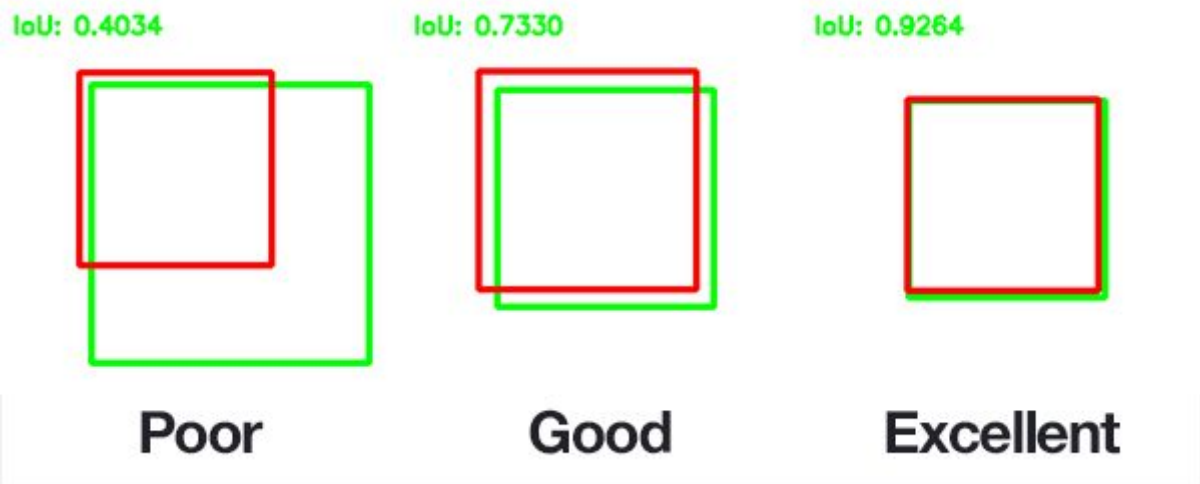
Dans le cas de RetinaNet, la partie qui génère les feature map est le backbone. Ce backbone génère les feature map d'une manière assez particulière car il utilise la méthode FPN (Feature Pyramid Network) qui consiste à générer des features maps de versions d'images de plus en plus petite et de moins en moins bonne qualité pour permettre le repérage d'objet de différentes taille.



6 - Anchor box :

Les anchor boxes sont des boîtes de formats différents. Ces boîtes sont disposées sur toute la surface des feature map et servent de zones de prédictions sur l'apparition d'un objet ou non. La disposition des anchor box sur les feature map est toujours la même.

Durant la phase d'entraînement, les anchor boxes sont comparées à des bounding boxes. Les bounding boxes sont les boîtes résultats (label), elles sont disposées autour d'un objet et sont accompagnées des informations sur l'objet. Plus l'élément trouvé par le modèle est proche de la bounding boxes, plus celui-ci est performant.



On peut voir sur l'image ci dessus la bounding box en rouge et les anchor boxes en vert. Plus celle-ci est proche de la bounding box, plus le résultat est valide. Pour comparer ces deux boîtes, on utilise la méthode IoU soit d' "Intersection over Union".

The diagram shows two overlapping blue rectangles. The top rectangle is slightly offset to the right and up from the bottom rectangle. The intersection of the two rectangles is a smaller blue rectangle in the center.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

7 - Subnet :

Les subnets sont représentés par des têtes qui parcourent les feature map pour repérer différents éléments.

Dans le cas de RetinaNet, il en existe deux : le "Classification subnet" et le "Box Regression subnet". Le "Classification subnet" donne pour chaque anchor box la probabilité d'apparition des objets alors que la "Box Regression subnet", elle, donne les coordonnées de cet objet dans l'anchor box. C'est grâce à ces deux mécanismes que le modèle peut retrouver des objets dans une image.

8 - Focal loss :

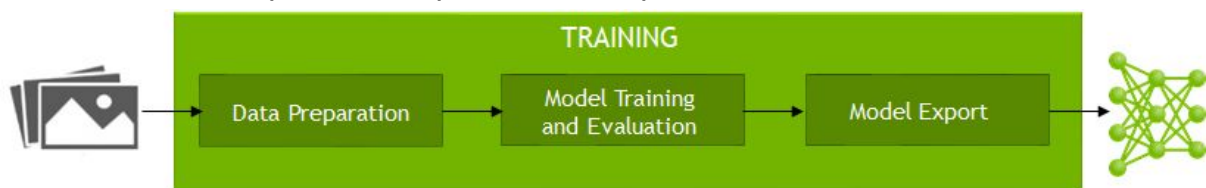
Le focal loss de RetinaNet accentue le travail du modèle sur les exemples compliqués et permet ainsi de réduire l'impact des exemples négatifs simples dans l'entraînement.

IV - Entraînement d'un modèle RetinaNet

Partie 1 :

Dans cette partie nous allons entraîner un modèle à reconnaître certains types d'objets en utilisant un docker PyTorch et une large base de données. Nous allons ensuite optimiser et mettre en application le modèle RetinaNet généré avec TensorRT et NVIDIA Deepstream.

Nous allons décomposer cette partie en 3 étapes :



Les étapes du Training

Dans la partie "Data Preparation" nous allons nous occuper de télécharger puis convertir dans un bon format les images nécessaires à l'entraînement de l'IA. Dans la partie "Model Training and Evaluation" nous allons voir comment entraîner le modèle Retinanet et évaluer ses résultats.

Enfin dans la partie "Model Export" nous allons exporter le modèle sous le format ONNX. C'est le format ONNX qui est utilisé pour le déploiement du modèle.

1 - Préparation des données :

Nous allons utiliser la banque d'images [Open Images v5](#) pour notre entraînement. Pour cet exemple, nous avons décidé que notre modèle RetinaNet allait reconnaître les bagages et les sacs (*luggage and bags* class). Il faut donc trier notre base d'images pour ne garder que ce qui nous intéresse.

Vous êtes libre du choix et nous expliquerons plus en détails comment choisir votre classe. Nous finirons cette partie en convertissant notre base d'images dans le format COCO.

Si vous rencontrez un problème lors de la conversion des training/validation/annotation en JSON (freeze de l'ordinateur/crash) ou dans les étapes à venir plus généralement, veuillez vous référer à la partie Problèmes matériels Ram dans la section problèmes.

1 - Cloner le dépôt

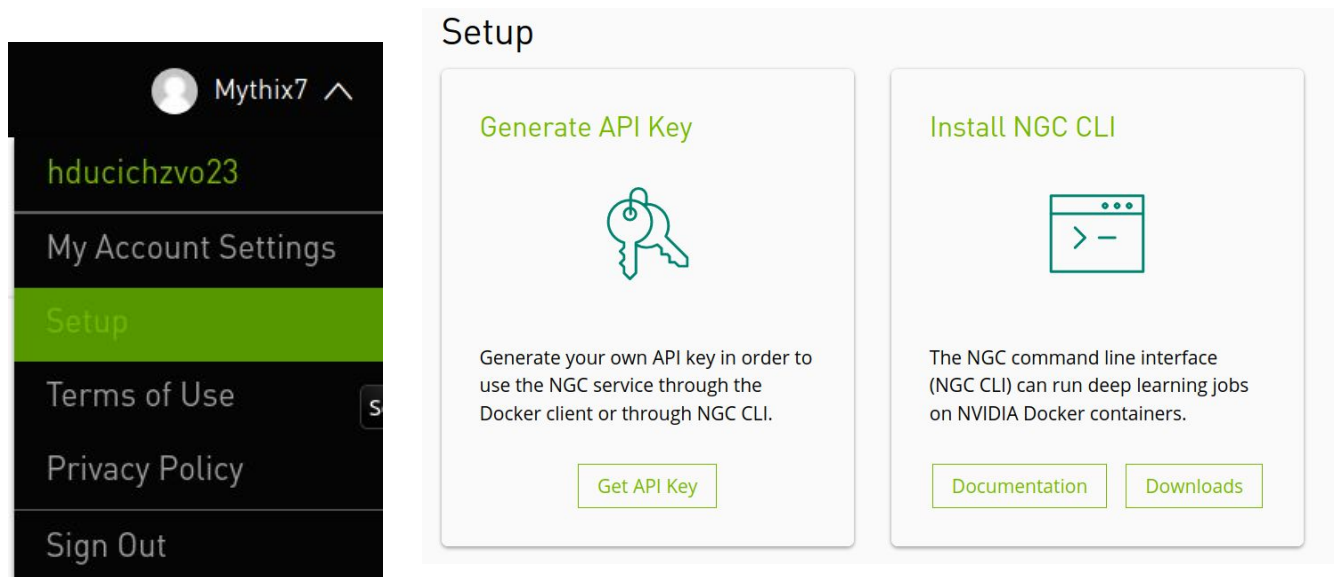
Ouvrez un terminal puis exécutez les commande suivantes dans le répertoire de votre choix :

```
git clone
https://github.com/NVIDIA-AI-IOT/retinanet_for_redaction_with_deepstream.git
cd retinanet_for_redaction_with_deepstream.
```

Cela correspondra à votre répertoire de travail “Working Dir”.

2 - Demarrer le docker NGC PyTorch

Pour réaliser cet entraînement, vous avez besoin d’un compte sur [NGC](#). Créez vous un compte gratuitement si ce n’est pas déjà le cas. Générer ensuite une clef API en allant dans setup et en cliquant sur “GET API Key”. Puis sur “Generate API KEY” en haut à droite.



Notez votre identifiant ainsi que votre mot de passe.

Téléchargez le docker PyTorch 19.09 :

```
sudo docker pull nvcr.io/nvidia/pytorch:19.09-py3
```

Connectez vous au docker avec l’identifiant et mdp généré précédemment :

```
sudo docker login nvcr.io
```

Définissez votre espace de travail et l'endroit et de stockage de données :

```
DATA_DIR='path to your intended data dir' exemple: disque dur dans lequel se trouvent les images.
```

```
WORKING_DIR='path to working directory' exemple:le dossier sur le bureau dans lequel nous allons exécuter les scripts principaux.
```

Lancez le docker PyTorch :

```
sudo docker run -it --gpus all --rm --ipc=host -v $DATA_DIR:/data -v $WORKING_DIR:/src -w /src nvcr.io/nvidia/pytorch:19.09-py3  
cd /src
```

A partir de ce moment toutes les commandes seront exécutées à l'intérieur du docker

3 - Téléchargement depuis Open Images

Ce téléchargement est **très long** (~650 GO). Il est recommandé de le faire dans un environnement disposant d'une bonne connexion. L'extraction des données est également très longue et peut durer plusieurs jours. Il est donc recommandé de disposer d'un SSD et de patience...

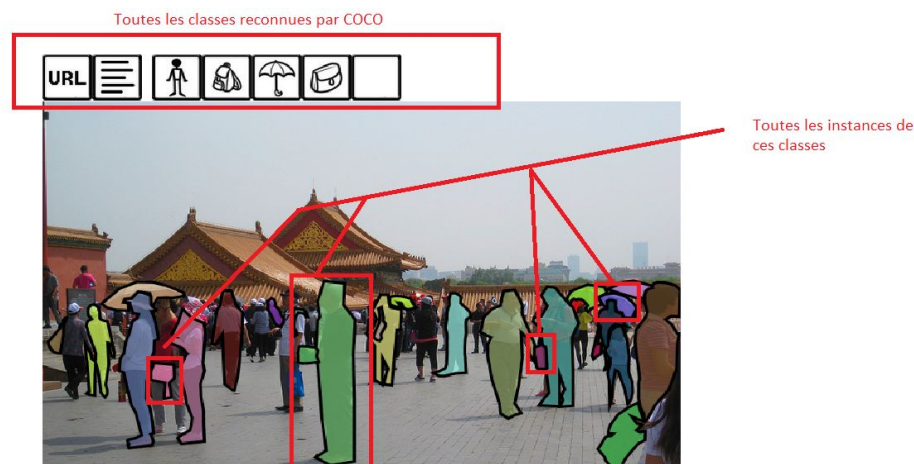
```
cd /data/open_images  
bash /src/open_images/download_open_images.sh  
bash /src/open_images/unzip_open_images.sh
```

4 - Explications et exemple

L'idée ici est de parser les annotations des images qui servent de validation au modèle dans un fichier JSON en utilisant un format COCO.

COCO est un outil à grande échelle de détection d'objets et segmentation d'une base d'images. COCO possède de nombreuses classes représentant des objets. Il identifie ensuite les instances(et leur position) de ses classes dans les images et les stock dans des fichiers annotations.

Exemple d'image :



Exemple d'annotations :

<https://cocodataset.org/#format-data>

C'est ce que nous souhaitons faire dans cet exemple avec la classe "luggage and bags". Néanmoins, il est possible de sélectionner la classe de votre choix, toutes les classes "parsables" sont dans le fichier /open_images/class-descriptions-boxable.csv et vous pouvez observer les images de votre classe sur le site [Open Images v5](#) dans "explore".

Dans notre cas, on exclut 2 types d'images lors de la conversion du dataset en COCO format. On exclut les images dont le ratio largeur/hauteur est supérieur à 2 ou si les images dont la redimension rend certains objets trop petits. On ne garde que les images de la classe qui nous intéressent.

5 - Conversion des “validation annotations” en JSON

Les fonctions utilisées dans cette section sont définies dans le fichier `/open_images/open_image_to_json.py`.

Créez un fichier `.py` dans votre répertoire `data` (`$ cd /data`) et remplissez le comme indiqué ci-dessous.

Commencez par indiquer les chemins des fichiers validation et annotations de Open images et choisissez un chemin pour enregistrer votre output. Nous vous recommandons de créer un dossier dans open images.

```
images_dir = '/data/open_images/validation'
annotation_csv = '/data/open_images/validation-annotations-bbox.csv'
category_csv = '/data/open_images/class-descriptions-boxable.csv'
output_json = '/data/open_images/nom_dossier/nom_fichier_validation.json'

# Read the Open Images categories and parse the data.

from data_tools.coco_tools import write_json
import open_images.open_image_to_json as oij
catmid2name = oij.read_catMIDtoname(category_csv) # Read the category names
oidata = oij.parse_open_images(annotation_csv) # This is a representation of our
dataset.

# Next, remove all the images that do not contain the class 'luggage and bags'.
set1 = oij.reduce_data(oidata, catmid2name, keep_classes=['Luggage and bags'])

# Finally convert this data to COCO format, using this as an opportunity to exclude two
sorts of annotations:
# 1. If the images were to be resized so that the longest edge was 864 pixels (set by
the max_size parameter), then exclude any annotations smaller than 2 x 2 pixels
(min_ann_size parameter).
# 2. Exclude any annotations (and the images they are associated with) if the width to
height ratio exceeds 2.0, set by the min_ratio parameter. This filters out annotations
that cover a large crowd of many faces.

# The annotations are not resized: RetinaNet handles that for you.

cocodata = oij.openimages2coco(set1, catmid2name, images_dir,
                              desc="Open Image validation data, set Luggage and bags.",
                              output_class_ids={'Luggage and bags': 1},
                              max_size=864, # Does not resize annotations!
                              min_ann_size=(2,2),
                              min_ratio=2.0)
write_json(cocodata, output_json)
```

Exécutez le script de votre fichier :

```
python3 nom_fichier_validation.py
```

6 - Conversion des “training annotation” en JSON

De manière similaire, allons maintenant convertir les “training annotations” en JSON avec le format COCO.

Créez un fichier .py dans votre répertoire data (\$ cd /data) et remplissez le comme indiqué ci-dessous.

```
import open_images.open_image_to_json as oij
from data_tools.coco_tools import write_json

# Definine paths
images_dir = ['/data/open_images/train_0%i'%oo for oo in range(9)] # There
are nine image directories.
annotation_csv = '/data/open_images/train-annotations-bbox.csv'
category_csv = '/data/open_images/class-descriptions-boxable.csv'
output_json = '/data/open_images/nom_dossier/nom_fichier_training.json'

# Read the category names
catmid2name = oij.read_catMIDtoname(category_csv)
# Parse the annotations
oidata = oij.parse_open_images(annotation_csv)

# Keep only luggage and bags
trainset1 = oij.reduce_data(oidata, catmid2name, keep_classes=['Luggage and
bags'])
cocodata = oij.openimages2coco(trainset1, catmid2name, images_dir,
desc="Open Image train data, set 1.",
                                output_class_ids={'Luggage and bags': 1},
                                max_size=880, min_ann_size=(1,1),
                                min_ratio=2.0)
write_json(cocodata, output_json)
```

Exécutez le script de votre fichier :

```
python3 nom_fichier_training.py
```

7 - Copie des images training et validation

L'idée ici est que jusqu'à présent nous avons trié toutes les annotations des images, nous savons donc quelles images utiliser. Et maintenant nous allons les séparer des autres images pour que l'entraînement du modèle ne se fasse que sur ces images.

Créez un fichier .py dans votre répertoire data (\$ cd /data) et remplissez le comme indiqué ci-dessous.

```
import open_images.open_image_to_json as oij

oij.copy_images('/data/open_images/testbottle/val_vehicle.json',
                '/data/open_images/validation',
                '/data/open_images/nom_dossier/nom_sous_dossier_val')
images_dir = ['/data/open_images/train_0%i'%oo for oo in range(9)] # There are nine
image directories.
oij.copy_images('/data/open_images/testbottle/train_vehicle.json', images_dir,
                '/data/open_images/nom_dossier/nom_sous_dossier_train')
```

Exécutez le script de votre fichier :

```
python3 nom_fichier_copy.py
```

2 - Entraînement du modèle RetinaNet :

Maintenant que toutes les données sont dans le bon format, le plus dur est passé.

1 - Cloner RetinaNet depuis le dépôt NVIDIA sur GitHub

Déplacez vous dans votre environnement de travail. (\$ cd /src)

```
git clone
https://github.com/NVIDIA/retinanet-examples/tree/80f14c59fe343b25754fdbb4467
a85a9461c375d
cd retinanet-examples-80f14c59fe343b25754fdbb4467a85a9461c375d
```

2 - Modification des hyperparameters (Optionnel)

Avant d'installer RetinaNet, vous pouvez modifier quelques paramètres tels que les "hyperparameters" (pour plus d'informations cliquez [ici](#)) dans les fichiers model.py ou train.py. **Les hyperparamètres sont très importants et délicats, ils influent grandement sur l'entraînement et les résultats du modèle. Il est recommandé de bien se renseigner avant de les modifier.**

3 - Installation de RetinaNet (OTDK)

Avant l'installation veuillez télécharger le fichier ResNet50FPN [ici](#) et le coller dans <path_to>/retinanet_for_redaction_with_deepstream/retinanet-examples-80f14c59fe343b25754fdbb4467a85a9461c375d

et dans

<path_to>/retinanet_for_redaction_with_deepstream/retinanet-examples-80f14c59fe343b25754fdbb4467a85a9461c375d/retinanet

Performance

The detection pipeline allows the user to select a specific backbone depending on the latency-accuracy trade-off preferred.

ODTK **RetinaNet** model accuracy and inference latency & FPS (frames per seconds) for **COCO 2017** (train/val) after full training schedule. Inference results include bounding boxes post-processing for a batch size of 1. Inference measured at `--resize 800` using `--with-dali` on a FP16 TensorRT engine.

Backbone	mAP @[IoU=0.50:0.95]	Training Time on DGX1v	Inference latency FP16 on V100	Inference latency INT8 on T4
ResNet18FPN	0.318	5 hrs	14 ms; 71 FPS	18 ms; 56 FPS
MobileNetV2FPN	0.333		14 ms; 74 FPS	18 ms; 56 FPS
ResNet34FPN	0.343	6 hrs	16 ms; 64 FPS	20 ms; 50 FPS
ResNet50FPN	0.358	7 hrs	18 ms; 56 FPS	22 ms; 45 FPS
ResNet101FPN	0.376	10 hrs	22 ms; 46 FPS	27 ms; 37 FPS
ResNet152FPN	0.393	12 hrs	26 ms; 38 FPS	33 ms; 31 FPS

Installation

For best performance, use the latest [PyTorch NGC docker container](#). Clone this repository, build and run your own image:

```
git clone https://github.com/nvidia/retinanet-examples
docker build -t odtk:latest retinanet-examples/
docker run --gpus all --rm --ipc=host -it odtk:latest
```

Exécutez la commande suivante dans la répertoire `retinanet-examples-80f14...` que vous venez de cloner votre environnement de travail :

```
pip install --no-cache-dir .
```

4 - Entraînement du modèle

Nous allons procéder à l'entraînement (enfin) du modèle. Le temps d'entraînement varie fortement en fonction du modèle et du type de GPU (VRAM par exemple). Avec une Nvidia DGX-1 il est possible de réaliser un entraînement avec un batch de 80, 60 000 itérations en 8h. Nous avons réalisé avec une Nvidia GTX 1060 (6G VRAM) un entraînement avec un batch de 4, 30 000 itérations en 12h.

Vous pouvez surveiller sur un autre terminal avec la commande `nvidia-smi` l'évolution de l'utilisation de la VRAM et ajuster en conséquence les paramètres de l'entraînement (batch, iterations...).

Exécutez la commande suivante pour commencer l'entraînement:

```
odtk train outputModel.pth --backbone ResNet50FPN --fine-tune
retinanet_rn50fpn.pth --classes 1 --lr 0.00003 --batch 4 --images
/data/open_images/nom_dossier/nom_sous_dossier_train/ --annotations
/data/open_images/nom_dossier/nom_fichier_training.json --val-images
/data/open_images/validation --val-annotations
/data/open_images/nom_dossier/nom_fichier_validation.json --val-iters 3000
--resize 800 --max-size 880 --iters 30000 --milestones 2000 6000 | tee
outputModel.log
```

Voici une description non exhaustive de quelques hyperparameters :

- Le modèle est entraîné avec 30,000 iterations (--iters), avec une taille de batch de 4 (--batch).
- Les images sont redimensionnées pour que 800 pixels soit la taille minimum (--resize), et que la taille maximale des images ne dépasse pas 880 pixels (--max-size).
- Nous utilisons un “learning rate” ajustable qui commence à 0.00003 (--lr), qui est ensuite divisé par 10 après 2000 itérations, et re divisé encore après 6000 itérations, spécifié par --milestones options.
- Cet entraînement utilise un modèle prè-entraîné avec ResNet50 backbone (--fine-tune).

5 - Evaluation

Utiliser la commande otdk pour évaluer le modèle :

```
odtk infer outputModel.pth --images /data/open_images/validation  
--annotations /data/open_images/testbottle/val_vehicle.json
```

3 - Exporter en ONNX

Maintenant que le modèle est entraîné et évalué, l'étape finale de la partie 1 est d'exporter le modèle. Une fois que le modèle est exporté, il est prêt au déploiement.

Nous allons exporter le modèle dans le format "Open Neural Network Exchange (ONNX) standard, qui pourra ensuite être utilisé par n'importe quel NVIDIA GPU. Le format ONNX est facile à convertir en un fichier engine avec TensorRT pour faire de l'inférence.

Vous pouvez exporter le modèle en utilisant une précision de type FP32, FP16 ou INT8. Par défaut la précision utilisée par notre modèle est FP16, et le flux entrant du modèle est 864 x 512 pixels.

```
odtk export outputModel.pth outputModel.onnx --size 512 864 --batch 1
```

Pour plus d'information sur la [précision](#).

Pour exporter le modèle avec une précision différente vous pouvez exécuter la commande :

```
odtk export --help
```

Ou vous rendre dans le [readme](#) du dépôt sur RetinaNET

Partie 2 :

Dans cette partie, vous allez apprendre comment déployer le modèle obtenu à partir de la partie 1 dans une application de détection en temps réel. Ici, nous allons déployer ce modèle en utilisant une Nvidia GTX 1060 (6G). Néanmoins il est possible d'utiliser ce modèle sur tout appareil Nvidia (Jetson, datacenter..) en effectuant quelques petites modifications pendant le déroulement de cette partie.

Les commandes suivantes à exécuter ne relève plus du docker PyTorch.

1 - Construction de l'engine de TensorRT

Nous allons maintenant voir les étapes pour construire un engine TensorRT.

Il est fortement recommandé de faire une copie locale de votre environnement de travail (WORKING DIR).

1 - Construction de l'API RetinaNet en C++

Pour convertir le modèle ONNX en un engine TensorRT, vous devez d'abord générer un exécutable appelé **export**. Si vous souhaitez analyser plusieurs flux en même temps dans votre projet il vous est possible de modifier le "batch size" dans export.cpp avec la ligne

```
int batch = nombre_de_flux;
```

Rendez vous dans votre environnement de travail (WORKING_DIR) dans notre cas :

```
cd  
retinanet_for_redaction_with_deepstream/retinanet-examples-80f14c59fe343  
b25754fdbb4467a85a9461c375d/ )
```

Exécutez :

```
cd extras/cppapi
```

Nous allons générer l'API puis l'exécutable.

```
mkdir build && cd build  
sudo apt-get install cmake  
cmake -DCMAKE_CUDA_FLAGS="--expt-extended-lambda -std=c++11" ..
```

Si une erreur concernant opencv survient lors de l'exécution de la commande précédente. Veuillez exécuter les commandes suivantes :

```
cd
mkdir ~/opencv_build && cd ~/opencv_build
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
cd opencv
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D INSTALL_C_EXAMPLES=ON \
      -D INSTALL_PYTHON_EXAMPLES=ON \
      -D OPENCV_GENERATE_PKGCONFIG=ON \
      -D OPENCV_EXTRA_MODULES_PATH=~/opencv_build/opencv_contrib/modules \
      -D BUILD_EXAMPLES=ON ..
make j12 ( 12 étant le nombre de coeurs de votre processeur, si vous ne le
connaissiez pas, utilisez la commande nproc)
sudo make install
```

Si l'installation a bien fonctionné, la commande suivante devrait renvoyer une version:

```
pkg-config --modversion opencv4
```

Revenez dans votre environnement de travail

(retinanet_for_redaction_with_deepstream/retinanet-examples-80f14c59fe343b25754fdbb4467a85a9461c375d/extras/cppapi/build) la commande devrait fonctionner.

```
$ cmake -DCMAKE_CUDA_FLAGS="--expt-extended-lambda -std=c++11"..
$ make
```

2 - Génération du fichier “engine” avec TensorRT

TensorRT crée un fichier binaire “engine” ou “plan” optimisé, qui a pour but de réduire la latence et améliorer l'efficacité de l'inférence.

Copiez le modèle ONNX généré précédemment dans le dossier actuel.

Utilisez l'exécutable **export** généré à l'étape précédente pour convertir le modèle ONNX en un fichier “engine” de TensorRT. Dans notre cas, nous avons pris un batch de 1.

```
./export outputModel.onnx outputModel_b1.plan
```

2 - Déploiement avec DeepStream

1 - Construire le parseur personnalisé pour RetinaNet

DeepStream utilise TensorRT pour l'inférence. DeepStream vous donne l'option d'utiliser soit le fichier ONNX directement ou le fichier "engine" de TensorRT généré à l'étape précédente. Dans notre cas, nous avons choisi d'utiliser le fichier "engine".

De peur de perdre le sens original de l'explication à travers la traduction de mots n'ayant pas forcément d'équivalent. Nous avons choisi d'expliquer ce paragraphe en anglais :

"TensorRT takes the input tensors and generates output tensors. To detect an object, tensors from the output layer must be converted to X,Y coordinates or the location of the bounding box of the object. This step is called bounding box parsing."

Par défaut, DeepStream fournit un parseur pour le modèle DetectNet_v2. Pour les autres modèles il faut construire un parseur personnalisé. Pour notre modèle RetinaNet le code pour parser les "bounding box" (rectangles entourant les objets détectés) se trouve dans `nvdsparsebbox_retinanet.cpp` :

(/retinanet_for_redaction_with_deepstream/retinanet-examples-80f14c59fe343b25754fdbb4467a85a9461c375d/extras/deepstream/deepstream-sample).

La fonction utilisée par DeepStream pour parser les box s'appelle `NvDsInferParseRetinaNet` (cf [source](#) et [code source](#)).

Pour d'autres modèles tels que YOLO, FasterRCNN, et SSD, Deepstream propose des exemples.

Pour construire le parseur personnalisé :

```
cd extras/deepstream/deepstream-sample/
```

Le parseur à générer dépend de l'architecture du matériel sur lequel il sera déployé. Voici un tableau avec quelques noms de code d'architectures Nvidia : voir [site](#).

Fermi	Kepler	Maxwell	Pascal	Volta	Turing	Ampere	Hopper
sm_20	sm_30	sm_50	sm_60	sm_70	sm_75	sm_80	sm_90*?
	sm_35	sm_52	sm_61	sm_72		sm_86	
	sm_37	sm_53	sm_62				

Gardez en mémoire l'architecture de votre matériel, dans notre cas nous disposons d'une GTX 1060 -> Architecture Pascal -> sm_61 (voir ci dessous):

Pascal (CUDA 8 and later)

- SM60 or **SM_60, compute_60** –
Quadro GP100, Tesla P100, DGX-1 (Generic Pascal)
- SM61 or **SM_61, compute_61** –
GTX 1080, GTX 1070, GTX 1060, GTX 1050, GTX 1030 (GP108), GT 1010 (GP108) Titan Xp, Tesla P40, Tesla P4, Discrete GPU on the NVIDIA Drive PX2
- SM62 or **SM_62, compute_62** –
Integrated GPU on the NVIDIA Drive PX2, Tegra (Jetson) TX2

Éditez le fichier CMakeLists.txt.

Modifier la ligne **set(ARCH "sm_votreCodeArchitecture")**

```
mkdir build && cd build
cmake -DCMAKE_CUDA_FLAGS="--expt-extended-lambda -std=c++11" ..
make
```

Modifier le fichier **nvdsparsebbox_retinanet**.

Changer la ligne `#include "nvdsinfer_custom_impl.h"` en incluant le lien vers le dossier `sources/includes/` de votre installation deepstream. Par exemple :

```
#include
"/opt/nvidia/deepstream/deepstream-5.0/sources/includes/nvdsinfer_custom_impl.h
```

```
/**
 * Copyright (c) 2020, NVIDIA CORPORATION. All rights reserved.
 *
 * NVIDIA Corporation and its licensors retain all intellectual property
 * and proprietary rights in and to this software, related documentation
 * and any modifications thereto. Any use, reproduction, disclosure or
 * distribution of this software and related documentation without an express
 * license agreement from NVIDIA Corporation is strictly prohibited.
 */

#include <cstring>
#include <iostream>
#include "/opt/nvidia/deepstream/deepstream-5.0/sources/includes/nvdsinfer_custom_impl.h"

#define MIN(a,b) ((a) < (b) ? (a) : (b))

/* This is a sample bounding box parsing function for the sample Resnet10
 * detector model provided with the SDK. */

/* C-linkage to prevent name-mangling */
extern "C"
```

Cette opération créer un fichier `.so` (shared object) `libnvdsparsebbox_retinanet.so`.

Ce fichier sera utilisé par l'application DeepStream ultérieurement lors du parsing des bounding box.

2 - Paramétrer et lancer l'application

Nous sommes sur le point de lancer l'application, mais avant il va falloir mettre en place un **fichier config** de DeepStream dans lequel nous préciserons des paramètres dont:

- Setup notre source (webcam, lien vidéo)
- Setup le model engine pour qu'il pointe sur notre model TensorRT
- Setup le chemin du parser de bounding box à notre parser .so libnvdsparsebbox_retinanet.so

Pour chaque application utilisant deepstream, il est nécessaire de réaliser un fichier config dans lequel nous allons fournir l'ensemble des ressources nécessaires à l'exécution du programme et à la détection des objets. Ici, nous allons utiliser un **fichier modèle** appelé valiseEngineFile_b4.**plan** généré à l'aide de RetinaNet (le résultat d'un l'entraînement tel que vu précédemment) et qui sert à détecter des valises depuis une webcam.

Fichier config **dstest1_pgjie_Valise.txt**:

Avant toute chose, il est conseillé de lire les fichiers config samples fournis par deepstream à l'emplacement d'installation (pour notre cas, c'est `/opt/nvidia/deepstream/deepstream-5.0/samples/configs/`)

Ici, nous pouvons trouver de nombreux exemples couvrant la plupart des cas d'applications générales. Nous prenons en exemple le fichier utilisé pour le sample de la caméra intégrée (ou usb). Il est conseillé également de lire les commentaires nvidia en début de fichier pour comprendre quelques règles d'écriture dans le fichier config.

Voici donc notre fichier modifié pour la détection de valises, on le remplit avec les informations suivantes (en fonction de notre utilisation):

Il existe plusieurs groupes de paramètres dans un fichier config, et pour chaque groupe, on indique ceux que nous voulons lui appliquer. Ici, nous avons seulement le groupe “property” et “class-attrs-all”:

```
# propriétés de l'application

[property]
gpu-id=0
net-scale-factor=0.0039215697906911373
# Lien vers notre fichier .plan résultant des étapes suivis précédemment
model-engine-file=/home/mah/bottlesDetector/src/redaction_b4.plan
batch-size=1
## 0=FP32, 1=INT8, 2=FP16 mode
network-mode=1
num-detected-classes=1
interval=0
gie-unique-id=1
is-classifier=0
# Fonction de parsing comme donnée précédemment dans la partie Etape 2
(fichier .so).
# C'est une fonction custom, mais deepstream fournit des fonctions de
parsing out of the box pour quelques modèles.
# Dans notre cas, on utilise retinaNet donc nous devons fournir la
nôtre:
parse-bbox-func-name=NvDsInferParseRetinaNet
custom-lib-path=/home/mah/bottlesDetector/src/libnvdsparsebbox_retinanet
.so
output-blob-names=conv2d_bbox;conv2d_cov/Sigmoid

[class-attrs-all]
threshold=0.2
group-threshold=0
```

Pour d'autres applications, il peut exister des fichiers configs beaucoup plus complexes et même certains sont séparés en plusieurs fichiers.

Néanmoins, pas de panique! il suffit seulement de lire la remarque suivante:

Remarque: Dans cette exemple se trouve seulement les balises que nous jugeons intéressants pour notre application, néanmoins il existe une multitude de paramètres que vous pouvez retrouver dans ce lien: [NVIDIA DeepStream SDK Developer Guide : Reference Application Configuration | NVIDIA Docs](#) et [DeepStream Reference Application - deepstream-app — DeepStream DeepStream Version: 5.0 documentation \(nvidia.com\)](#)

Il suffit de sélectionner le groupe de paramètres et de lire la liste des paramètres et leurs descriptions et fonctions.

Ainsi, nous avons vu comment se constituent le(s) fichier(s) config d'un projet DeepStream.

Pour l'application en elle-même, il faut savoir que le code entre python et c++ est extrêmement similaire du côtés des noms des fonctions deepstream et gstreamer.

Pour des exemple:

→ Si votre projet est en C/C++:

Alors il est conseillé de prendre exemple sur le repertoire “/opt/nvidia/deepstream/deepstream-5.0/sources/apps/sample_apps” (dans le cas de ce tuto, c'est ici que se trouve l'installation deepstream) et de consulter les différents exemples donnés en fichiers sources. Par exemple, le dossier deepstream-test1 contient le fichier source deepstream_test1_app.c pour lancer une application deepstream de reconnaissance d'objets (voiture) via webcam. Pour compiler celle-ci, il suffit d'ouvrir le terminal, et de faire la commande: “sudo make”.

Si vous rencontrez des problèmes pour exécuter cette commande, merci de consulter la section “Problèmes rencontrés” à la fin de ce tutoriel pour d'éventuelles solutions.

Ensuite, on lance l'application via terminale avec la commande:

```
./deepstream-test1-app <h264_elementary_stream comme /dev/video0 pour la webcam>
```

Attention: Ces commandes peuvent différer d'un sample à l'autre. Merci de consulter le README à l'intérieur de chaque sample pour voir les étapes pour compiler le sample en question.

Ainsi, nous pouvons voir le programme se lancer.

→ Si votre projet est en Python:

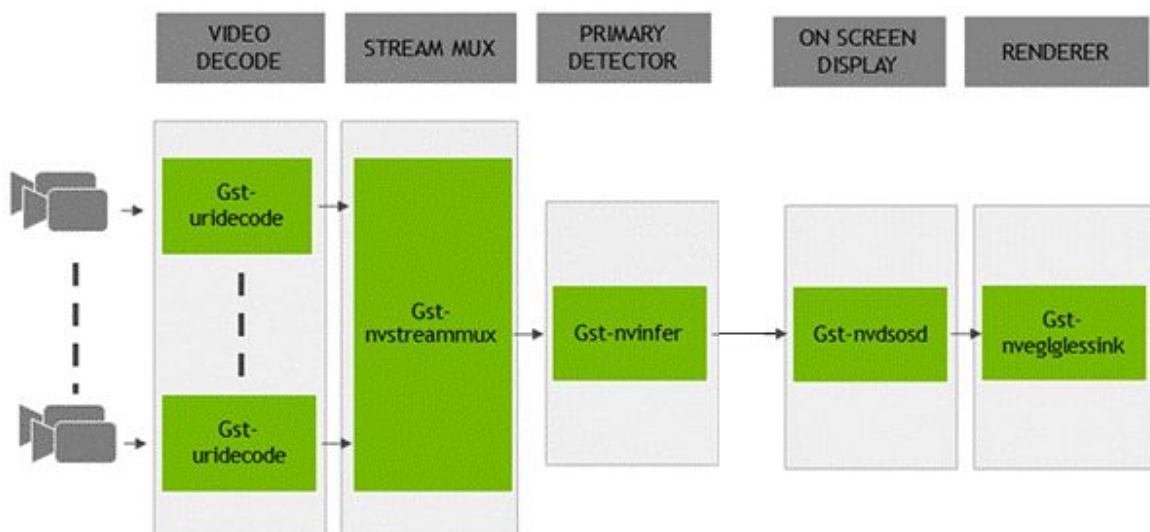
On consulte dans le dossier `deepstream_python_apps` (le python bindings que nous avons téléchargé à l'étape de l'installation) le dossier `apps/` dans lequel se trouve un grand nombre d'exemples tout comme la partie décrite de c++ plus haut. On prend par exemple le sample "deepstream-test1-usbcam" et dans le dossier, on retrouve le fichier `conf`, ainsi que le fichier python que nous pourrions exécuter avec la commande:

```
python3 deepstream_test_1_usb.py <v4l2-device-path comme /dev/video0 pour webcam>
```

Pour chaque sample, la structure du code est décrite dans le fichier `README` présent dans leur dossier respectif. On peut également retrouver des informations dans les sites dont les liens ont été donnés plus haut (la doc Nvidia deepstream metropolis).

Pour résumer notre exemple de détection de valise:

- Nous retrouverons dans le fichier python une première fonction qui gère le OSD (la partie informations présente en haut à gauche de l'écran).
- Le main dans lequel nous allons instancier et créer les pipes nécessaire à gstreamer et deepstream selon le schéma donnée :



Extrait du main() du code python:

```
# videoconvert to make sure a superset of raw formats are supported
vidconvsrc = Gst.ElementFactory.make("videoconvert", "converter_src1")
if not vidconvsrc:
    sys.stderr.write(" Unable to create videoconvert \n")

# nvvideoconvert to convert incoming raw buffers to NVMM Mem (NvBufSurface API)
nvvidconvsrc = Gst.ElementFactory.make("nvvideoconvert", "converter_src2")
if not nvvidconvsrc:
    sys.stderr.write(" Unable to create Nvvideoconvert \n")

caps_vidconvsrc = Gst.ElementFactory.make("capsfilter", "nvmm_caps")
if not caps_vidconvsrc:
    sys.stderr.write(" Unable to create capsfilter \n")

# Create nvstreammux instance to form batches from one or more sources.
streammux = Gst.ElementFactory.make("nvstreammux", "Stream-muxer")
if not streammux:
    sys.stderr.write(" Unable to create NvStreamMux \n")

# Use nvinfer to run inferencing on camera's output,
# behaviour of inferencing is set through config file

pgie = Gst.ElementFactory.make("nvinfer", "primary-inference")
if not pgie:
    sys.stderr.write(" Unable to create pgie \n")

# Use converter to convert from NV12 to RGBA as required by nvosd
nvvidconv = Gst.ElementFactory.make("nvvideoconvert", "converter")
if not nvvidconv:
    sys.stderr.write(" Unable to create nvvidconv \n")

# Create OSD to draw on the converted RGBA buffer
nvosd = Gst.ElementFactory.make("nvsosd", "onscreendisplay")

if not nvosd:
    sys.stderr.write(" Unable to create nvosd \n")

# Finally render the osd output
if is_aarch64():
    transform = Gst.ElementFactory.make("nvegltransform", "nvegl-transform")
```

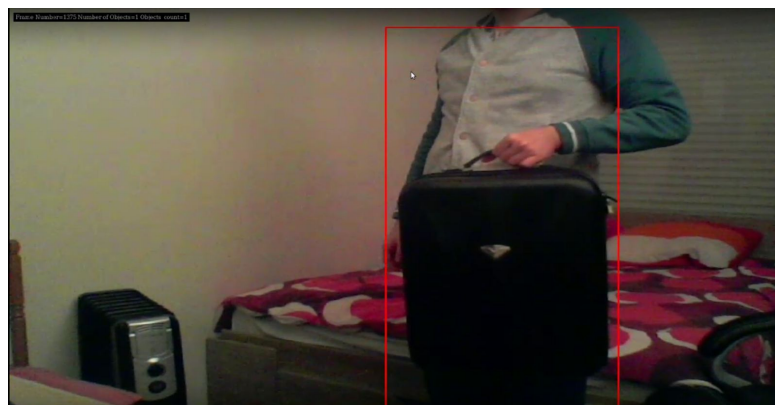
Creation des éléments nécessaires au pipeline gst.

```
print("Creating EGLSink \n")
sink = Gst.ElementFactory.make("nveglglessink", "nvvideo-renderer")
if not sink:
    sys.stderr.write(" Unable to create egl sink \n")

print("Playing cam %s " %args[1])
caps_v4l2src.set_property('caps', Gst.Caps.from_string("video/x-raw, framerate=30/1"))
caps_vidconvsrc.set_property('caps', Gst.Caps.from_string("video/x-raw(memory:NVMM)"))
source.set_property('device', args[1])
streammux.set_property('width', 1920)
streammux.set_property('height', 1080)
streammux.set_property('batch-size', 1)
streammux.set_property('batched-push-timeout', 4000000)
pgie.set_property('config-file-path', "dstest1_pgie_config.txt")
# Set sync = false to avoid late frame drops at the display-sink
```

Parametres config gst et deepstream

Pour plus d'informations sur l'architecture de base de l'application reference deepstream-app, vous pouvez consulter: [DeepStream Reference Application - deepstream-app — DeepStream DeepStream Version: 5.0 documentation \(nvidia.com\)](https://docs.nvidia.com/deeplearning/deepstream/docs/index.html#deepstream-app) .



V - Problèmes rencontrés

Nous avons suivi un tutoriel pour la partie training néanmoins des informations sont erronées/plus à jour.

Problème Lancement Sample Python :

Si vous rencontrez un problème lors de l'exécution du sample (en rapport avec un certain fichier "resnet10.caffemodel_b1_gpu0_int8.engine" manquant, il suffit d'aller dans le fichier config "dstest1_pgie_config.txt" par exemple et mettre la ligne avec la balise "model-engine-file=" en commentaire (on met en # au début de la ligne) tel que:

```
#model-engine-file=../../../../../samples/models/Primary_Detector/resnet10.caffemodel_b1_gpu0_int8.engine
```

Problème versionning : write_json

Dans la partie 1.4 Conversion de la partie validation en JSON mais aussi dans la partie 1.5 Conversion des "training annotation" en JSON . Lors de l'exécution du fichier.py que nous créons, la méthode **oij.write_json(cocodata, output_json)** n'est pas définie.

```
images_dir = '/data/open_images/validation'
annotation_csv = '/data/open_images/validation-annotations-bbox.csv'
category_csv = '/data/open_images/class-descriptions-boxable.csv'
output_json = '/data/open_images/processed_val/val_faces.json'

# Read the Open Images categories and parse the data.

import open_images.open_image_to_json as oij
catmid2name = oij.read_catMIDtoName(category_csv) # Read the category names
oidata = oij.parse_open_images(annotation_csv) # This is a representation of our dataset.

# Next, remove all the images that do not contain the class 'human face'.
set1 = oij.reduce_data(oidata, catmid2name, keep_classes=['Human face'])

# Finally convert this data to COCO format, using this as an opportunity to exclude two sorts of annotations:
# 1. If the images were to be resized so that the longest edge was 864 pixels (set by the max_size parameter), then e
# 2. Exclude any annotations (and the images they are associated with) if the width to height ratio exceeds 2.0, set

# The annotations are not resized: RetinaNet handles that for you.

cocodata = oij.openimages2coco(set1, catmid2name, images_dir,
                             desc="Open Image validation data, set 1.",
                             output_class_ids={'Human face': 1},
                             max_size=864, # Does not resize annotations!
                             min_ann_size=(2,2),
                             min_ratio=2.0)
oij.write_json_data(cocodata, output_json)
```

capture d'écran du tutoriel que nous avons suivi

Il faut la remplacer par `write_json(cocodata, output_json)` et rajouter dans les import : `from data_tools.coco_tools import write_json`.

Problème matériel : RAM

Un second problème rencontré est celui du matériel. N'ayant à disposition que 11 Go de RAM ce qui n'est pas suffisant pour la partie "parsing training data". Nous avons dû étendre la pagination du système (swap memory) et celle Docker pour palier au manque de ram en utilisant ce lien :

<https://linuxize.com/post/how-to-add-swap-space-on-ubuntu-18-04/>

Une fois cela réalisé on lance le docker PyTorch en utilisant la commande suivante :

```
sudo docker run -it --memory="9g" --memory-swap="30g" --gpus all --rm
--ipc=host -v $DATA_DIR:/data -v $WORKING_DIR:/src -w /src
nvcr.io/nvidia/pytorch:19.09-py3
```

Problème versionning : GIT

Avec le Git Actuel, l'exécution de la commande:

```
pip install --no-cache-dir .
```

provoque une erreur à l'étape "Building wheels for collected packages: odtk". Nous n'avons pas compris ce problème mais il s'agit à coup sûr d'un problème de versionning. La résolution de ce problème s'est faite par l'utilisation de la branche de git du 20 avril plutôt que celle actuelle.

Problème Include : “Gst/gst.h” ou autre dans certains des .cpp:

Il arrive quelques fois lors de la compilation de certains samples, des includes soient invalides.

Par exemple, dans le dossier samples de deepstream, quand on fait “make”, on nous indique qu’un include “Gst/gst.h” est introuvable dans le fichier .cpp. Notre solution au problème était de faire le suivant:

Installer le package “libjson-glib-dev” avec apt-get install et exécuter la commande suivante: “pkg-config --cflags json-glib-1.0”.

Pour une autre erreur, vous pouvez essayer de faire la commande :

“sudo updatedb” puis “locate <le nom du fichier recherché>”. Cette commande va renvoyer l’emplacement du fichier si il se trouve sur votre système. On copie ce lien et on le colle dans le .cpp à la place de l’ancien include.

Exemple: Erreur à la ligne: include “fichier1.h introuvable”.

Alors on fait sudo updatedb puis on fait “locate fichier1.h”. Si il se trouve bien dans l’ordinateur, alors un lien vers ce fichier sera renvoyé dans la console. On copie ce lien, et on va dans le cpp remplacer la ligne include “fichier1.h” par include “/opt/exemple/fichier1.h”.

Si “locate” ne renvoie rien, c’est que le fichier n’est pas ici. Il faut donc aller chercher sur internet 🤔🤔🤔

Sources:

[NVIDIA DeepStream SDK Developer Guide : DeepStream overview | NVIDIA Docs](#)

[NVIDIA DeepStream SDK | NVIDIA Developer](#)

[Setup Nvidia Deepstream development Environment | by Amar Kumar | Oct. 2020 | Medium](#)

[Setup Docker in Ubuntu 18.04. I know the challenges when people stuck... | by Amar Kumar | Oct. 2020 | Medium](#)

[NVIDIA DeepStream SDK FAQs : Frequently Asked Questions | NVIDIA Docs](#)

[NVIDIA DeepStream SDK Developer Guide : Reference Application Configuration | NVIDIA Docs](#)

[DeepStream Python — Deepstream ##DeepStream_VERSION## documentation](#)

[Building Intelligent Video Analytics Apps Using NVIDIA DeepStream 5.0 \(Updated for GA\) | NVIDIA Developer Blog](#)

[NVIDIA DeepStream SDK Developer Guide : Performance | NVIDIA Docs](#)

[Getting Started with NVIDIA DeepStream-5.0 | LaptrinhX](#)

[Creating a Human Pose Estimation Application with NVIDIA DeepStream | NVIDIA Developer Blog](#)

[Understanding the basics of AIXPRT precision settings | BenchmarkXPRT Development Community Blog](#)

[Intersection over Union \(IoU\) for object detection - PyImageSearch](#)

[What's the Difference Between Deep Learning Training and Inference? | The Official NVIDIA Blog](#)

[Anchor Boxes — The key to quality object detection | Towards Data Science](#)

[DeepStream Development Guide — DeepStream DeepStream Version: 5.0 documentation](#)

[Riding NVIDIA's slipstream. with Python | darlingevil.com](#)

[DeepStream Reference Application - deepstream-app — DeepStream DeepStream Version: 5.0 documentation](#)

[NVIDIA/retinanet-examples: Fast and accurate object detection with end-to-end GPU optimization \(github.com\)](#)

https://docs.nvidia.com/metropolis/deepstream/5.0DP/dev-guide/index.html#page/DeepStream_Development_Guide/deepstream_app_config.3.2.html

[NVIDIA Transfer Learning Toolkit | NVIDIA Developer](#)

[GStreamer](#)

[Classes - Gst 1.0 \(lazka.github.io\)](#)

[Install Deepstream SDK on Nvidia RTX 2060 GPU | by Tunggal MAT | Deepstream Tutorial | Medium](#)

[CUDA Toolkit 10.2 Download | NVIDIA Developer](#)

[Latest Intelligent Video Analytics/DeepStream SDK topics - NVIDIA Developer Forums](#)

[The intuition behind RetinaNet. The end goal of this blog post is to... | by Prakash Jay | Medium](#)

[Anchor Boxes for Object Detection - MATLAB & Simulink](#)