Implémentation de la proposition 1 : SORT avec métrique de profondeur

* Aller dans le répertoire Git local de mon pc
* Copier le projet git de la proposition

git clone <https://github.com/nwojke/deep_sort.git>

* Télécharger le réseau neuronal convolutif sauvegardé en .pb « tensorflow »

<https://drive.google.com/drive/folders/1m2ebLHB2JThZC8vWGDYEKGsevLssSkjo>

Etant sous Windows 7 :

* Lancer le power shell windows
* Aller dans le répertoire python

cd « C:\Users\BOUILLETEAU\AppData\Local\Programs\Python\Python35 »

* Installer OpenCV

pip install opencv-python

* Installer scipy

pip install scipy

* Installer SKLearn

pip install scikit-learn

* Installer Tensorflow pour windows 7

python –m pip install https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.12.0-cp35-cp35m-win\_amd64.whl

ou

pip install https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.12.0-cp35-cp35m-win\_amd64.whl

Important, il faut la version 3.5 de python pour pouvoir installer tensorflow.

* Générer les détections

python tools generate\_detections.py --model=resources/networks/mars-small128.pb --mot\_dir=./MOT16/train --output\_dir=./resources/detections/MOT16\_train

Création du répertoire ressources/detection/MOT16\_train et du répertoire resources/networks.

Copie du réseau télécharger plus haut et coller dans le répertoire resources/networks.

PRB : your cpu support instructions that this tensorflow binary was not compiled to use : AVX2

bazel build -c opt --copt=-march=native --copt=-mfpmath=both --config=cuda -k //tensorflow/tools/pip\_package:build\_pip\_package

Potentiellement une réponse ?

Il va récupérer les images uns par uns pour chaque vidéo d’apprentissage issus de MOT2016.

Temps d’exécution sur CPU : 15 h 40 -> 16 h 15 soit environ 35 min

* Lancer le traqueur avec leur jeu de test

Récupérer les détections pré-détectées du jeu de test sur le lien : https://drive.google.com/drive/folders/1VVqtL0klSUvLnmBKS89il1EKC3IxUBVK

Ajouter les detections récupérées dans le repertoire ressources/detections/MOT16\_POI\_test.

python deep\_sort\_app.py

--sequence\_dir=./MOT16/test/MOT16-06

--detection\_file=./resources/detections/MOT16\_POI\_test/MOT16-06.npy

--output\_file=./tracking/hypotheses.txt

--min\_confidence=0.3

--nn\_budget=100

--display=True

Création du repertoire tracking/hypotheses.

## **Implémentation de l’algorithme de détection**

3 solutions ont été aperçues dont 2 ne sont pas retenus. La solution détectron qui nécessite d’avoir un GPU ainsi que la solution utilisée par MOT Challenge, DPM qui utilise un logiciel payant MATLAB.

La solution qui a été retenu est l’algorithme YOLO version 3 qui se trouve sur le site <https://pjreddie.com>

J’ai utilisé l’implémentation de l’algorithme d’une autre personne : <https://www.arunponnusamy.com/yolo-object-detection-opencv-python.html> que j’ai modifié afin de pouvoir avoir en sortie le fichier de détections .txt qui contient les boîtes englobantes des détections de chacune des images de la vidéo.

J’ai ajouté une fonction qui permet d’écrite dans un fichier texte les valeurs x, y, w, h et confidence pour chacune des détections.

J’ai aussi ajouté un filtre sur les détections afin qu’il ne garde que les détections de personnes.

J’ai ajouté du code pour qu’il parcourt un répertoire d’images plutôt qu’une image par une.

Temps d’exécution pour le 1\_test\_passage\_femme qui contient un dossier de 330 images : 662.1463148593903 sec, soit 11.04 min

Potentiellement, nouvelles méthodes : <https://stackoverflow.com/questions/44674517/yolo-darknet-detecting-only-specific-class-like-person-cat-dog-etc>

<https://github.com/pjreddie/darknet/issues/582>

## **Test de l’algorithme de détection**

L’algorithme modifié de détection a été lancé sur 1\_text\_passage\_femme et on obtient bien le fichier det/det.txt contenant les données des boîtes englobantes des détections de personnes dans chacune des images du répertoire données en entrée.

## **Test de l’algorithme de suivi**

Premier test : génération des détections via le modèle prédéfini

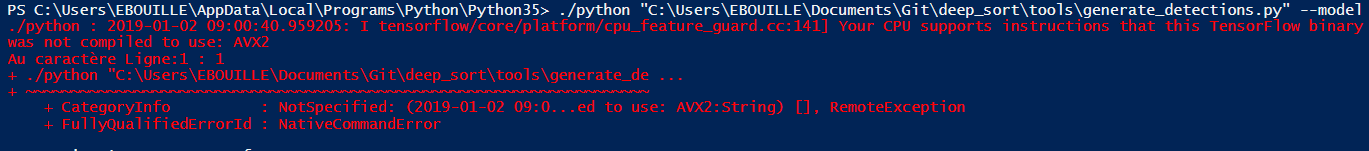
Prérequis :

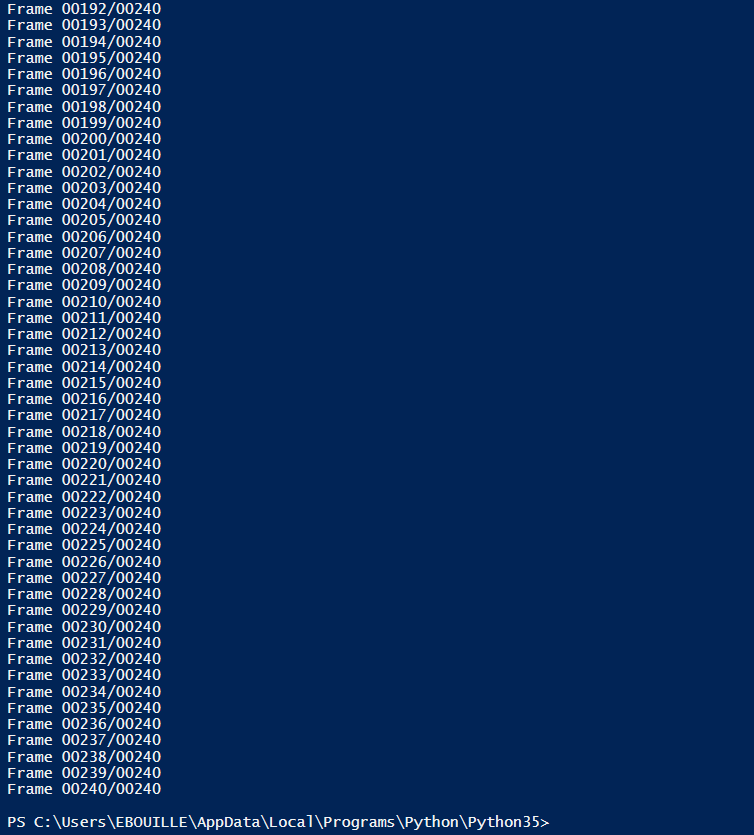
* Avoir le modèle prédéfini mars-small128.pb
* Avoir créé un jeu de test ressemblant au répertoire /MOT16/train
  + Contient autant de répertoire que de tests
  + Pour chaque répertoire test
    - Répertoire det avec le fichier det.txt
    - Répertoire img1 avec les images extraites de la vidéo
    - Le fichier seqinfo contenant les informations de la vidéo

On lance la commande suivante dans le powerShell :

python tools generate\_detections.py --model=resources/networks/mars-small128.pb --mot\_dir=./MOTFormat\_videos/test --output\_dir=./resources/detections/test\_1

Toujours l’erreur de tensorflow mais ça a l’air d’avoir fonctionné.





Frame id de la frame dans det.txt / id max de la frame dans det.txt

Deuxième test : Traçage de personnes (depp\_sort)

Prérequis :

* Répertoire comme MOT16/test/MOT16\_06
  + Répertoire det avec le fichier det.txt
  + Répertoire img1 avec les images extraites de la vidéo
  + Le fichier seqinfo contenant les informations de la vidéo
* Le fichier de détection issu de l’algorithme précédent

python deep\_sort\_app.py

--sequence\_dir=./MOT16/test/MOT16-06

--detection\_file=./resources/detections/MOT16\_POI\_test/MOT16-06.npy

--output\_file=./tracking/hypotheses.txt

--min\_confidence=0.3

--nn\_budget=100

--display=True

Résultat : le test fonctionne. Les boîtes englobantes ne sont pas très précises.

## **Test de l’algorithme de calcul d’indicateurs MOT Challenge**

python evaluate\_motchallenge.py

--mot\_dir=./MOT16/test

--detection\_dir=./resources/detections/test

--output\_dir=./evaluate\_output

--min\_confidence=0.3

--nn\_budget=100

Résultat: rien de concluant

## **Génération de tous les tests**

Etape 1 : générer le répertoire img1 contenant toutes les images de la vidéo et générer le fichier seqinfo.ini

python generate\_MOTFormat\_File.py

--video\_file=./2\_test\_passage\_homme/2\_test\_passage\_homme.mp4

--output\_dir=./MOTFormat\_videos/test/2\_test\_passage\_homme

Etape 2 : générer les detections avec opencv yolo

python yolo\_opencv\_detections\_file.py

--images=./MOTFormat\_videos/test/2\_test\_passage\_homme/img1

--format=jpg

--config=./object-detection-opencv/yolov3.cfg

--weights=./object-detection-opencv/yolov3.weights

--classes=./object-detection-opencv/yolov3.txt

--path\_detections\_file=./MOTFormat\_videos/test/2\_test\_passage\_homme

Temps d’exécution pour le 2\_test\_passage\_homme qui contient un dossier de 292 images : 549.7598917484283 sec, soit 9.16 min

Temps d’exécution pour le 3\_test\_tempsdepause\_homme qui contient un dossier de 235 images : 471.0479152202606 sec, soit 7.85 min

Temps d’exécution pour le 4\_test\_passagehc\_femme qui contient un dossier de 989 images : 1936.918317079544 sec, soit 32.28 min

Temps d’exécution pour le 5\_test\_passagehc\_homme qui contient un dossier de 543 images : 1102.3738107681274 sec, soit 18.37 min

Temps d’exécution pour le 6\_test\_passagehc\_2hommes qui contient un dossier de 830 images : 1619.0835886001587 sec, soit 26.98 min

Temps d’exécution pour le 7\_test\_passagehc\_2hommes qui contient un dossier de 1562 images : 3565.782112121582 sec, soit 59.43 min

Temps d’exécution pour le 8\_test\_passagehc\_hommefemme qui contient un dossier de 412 images : 1138.0413007736206 sec, soit 18.96 min

Temps d’exécution pour le 9\_test\_passagehc\_hommefemme qui contient un dossier de 431 images : 980.167774438858 sec, soit 16.33 min

Etape 3 : génération des détections via le modèle prédéfini

python tools generate\_detections.py

--model=resources/networks/mars-small128.pb

--mot\_dir=./MOTFormat\_videos/test

--output\_dir=./resources/detections/test

Etape 4 : Traçage de personnes (deep\_sort)

python deep\_sort\_app.py

--sequence\_dir=./MOTFormat\_videos/test/2\_test\_passage\_homme

--detection\_file=./detections/test/2\_test\_passage\_homme.npy

--output\_file=./tracking/2\_hypotheses.txt

--min\_confidence=0.3

--nn\_budget=100

--display=True

## **Implémentation de l’algorithme de calcul de critères (MOTChallenge)**

Pour évaluer la méthode, nous avons besoin des boîtes englobantes véridique par image afin de les comparer avec les détections et le suivi. MOT16 et MOT17 contiennent un fichier texte de boîtes englobantes de vérité sur les données d’entrainement.

Lien vers le git : <https://github.com/cheind/py-motmetrics>

./python "C:\Users\EBOUILLE\Documents\Git\pred\Developpement\MOTChallenge\_metrics.py" --hypotheses\_file "C:\Users\EBOUILLE\Documents\Git\pred\Developpement\MOT17\train\MOT17-02-DPM\det\det.txt" --ground\_truth\_file "C:\Users\EBOUILLE\Documents\Git\pred\Developpement\MOT17\train\MOT17-02-DPM\gt\gt.txt" --output\_file "C:\Users\EBOUILLE\Documents\Git\pred\Developpement\evaluate\_output"

pip install motmetric

On va évaluer l’algorithme sur les données MOT16 qui contiennent les boîtes englobantes véridique.

IDF1 IDP IDR Rcll Prcn GT MT PT ML FP FN IDs FM MOTA MOTP

MOT16-02 22.8% 72.5% 13.5% 16.8% 90.4% 54 3 14 37 318 14830 31 118 14.9% 0.233

MOT16-04 40.7% 73.8% 28.1% 35.7% 93.8% 83 6 41 36 1123 30570 102 567 33.1% 0.201

MOT16-05 37.6% 77.9% 24.8% 29.7% 93.4% 125 5 47 73 143 4792 11 86 27.5% 0.224

MOT16-09 49.5% 67.9% 39.0% 50.8% 88.5% 25 4 17 4 347 2586 39 122 43.5% 0.246

MOT16-10 38.8% 76.8% 25.9% 32.3% 95.6% 54 5 16 33 181 8340 25 168 30.6% 0.237

MOT16-11 50.8% 73.2% 39.0% 50.1% 94.0% 69 6 26 37 293 4580 21 69 46.7% 0.204

MOT16-13 19.2% 86.5% 10.8% 11.7% 93.9% 107 3 16 88 88 10107 8 76 10.9% 0.251

OVERALL 37.3% 74.2% 24.9% 31.3% 93.3% 517 32 177 308 2493 75805 237 1206 28.9% 0.215

MOT17

.\python "C:\Users\EBOUILLE\Documents\Git\py-motmetrics\motmetrics\apps\eval\_motchallenge.py" "C:\Users\EBOUILLE\Documents\Git\pred\Developpement\MOT17\train" "C:\Users\EBOUILLE\Documents\Git\deep\_sort\resources\tracking\MOT17\_train(DPM)"

IDF1 IDP IDR Rcll Prcn GT MT PT ML FP FN IDs FM MOTA MOTP

MOT17-02-DPM 22.0% 72.5% 13.0% 16.2% 90.4% 62 3 14 45 318 15578 31 118 14.3% 0.233

MOT17-04-DPM 40.7% 73.8% 28.1% 35.7% 93.8% 83 6 41 36 1123 30570 102 567 33.1% 0.201

MOT17-05-DPM 37.2% 77.9% 24.4% 29.3% 93.5% 133 5 47 81 142 4890 11 86 27.1% 0.225

MOT17-09-DPM 49.3% 68.1% 38.6% 50.4% 88.9% 26 3 18 5 336 2643 39 123 43.3% 0.247

MOT17-10-DPM 37.7% 77.0% 24.9% 31.1% 95.9% 57 5 16 36 170 8850 25 167 29.6% 0.235

MOT17-11-DPM 49.9% 73.2% 37.9% 48.7% 94.1% 75 6 26 43 290 4839 22 71 45.4% 0.204

MOT17-13-DPM 18.9% 86.5% 10.6% 11.5% 93.9% 110 3 15 92 88 10299 8 76 10.7% 0.251

OVERALL 36.9% 74.2% 24.5% 30.8% 93.3% 546 31 177 338 2467 77669 238 1208 28.4% 0.215

REFERENCE

<https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>

<https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/>

<https://github.com/arunponnusamy/object-detection-opencv>

<https://github.com/cheind/py-motmetrics>

OUVERTURE

A propos de la camera PixyCam... <https://www.youtube.com/watch?v=EcCbEWiyiQY>