Notes TP final robotique

**PROGRAMME DES TACHES A REALISER SUR LE PROJET**

**1 -** Programme capable de créer et de sauvegarder des modèles (avec data augmentation sur les données "panneau stop" du prof)

**2 -** Réaliser un max d'entrainements (en suivant un planning qu'on aurait mis en place par ex)

**3 -** Etre capable d'intégrer un modèle dans ROS (récupérable par le robot)

**4 -** Faire la documentation (10 pages pour expliquer ce qui a été fait, comparatif des modèles)

PLAN 17/01 – 19/01 🡺 Créer un programme capable de créer et sauvegarder des modèles (Pourra servir pour répartition tests lundi / mercredi)

**Intro** 🡺 Récupérer « code à compléter sur myges », l’extraire et exécuter « catkin\_make » (où se trouve le fichier README.md)

**A – Installation python dans robotic courses**

sudo apt-get install python3-pip python3-yaml

sudo pip3 install rospkg catkin\_pkg

//pip install ros-numpy 🡺 **NE FONCTIONNE PAS, IL VA FALLOIR VITE DEBUG**

sudo apt-get install python-catkin-tools python3-dev python3-numpy

//sudo apt-get install ros-melodic-ros-numpy 🡺 peut-être suffisant ? (je ne sais pas si la version est OK)

**B – Créer dossier spécial « robotics\_train\_sign\_data »** (<https://github.com/Dreamsplutox/robotics_train_sign_data>) récupérer les données de type « stop-sign » et les visualiser

**C – Créer un environnement python avec l’object detection API fonctionnelle**

(De mon côté j’ai clone mon environnement pour ne pas le perdre, et j’ai ajouté l’object detection)

conda create --clone <environment to clone> --name <new enviroment name>

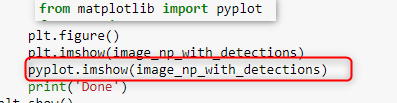
(install visual studio redis 2015 17 19 <https://support.microsoft.com/fr-fr/help/2977003/the-latest-supported-visual-c-downloads>)

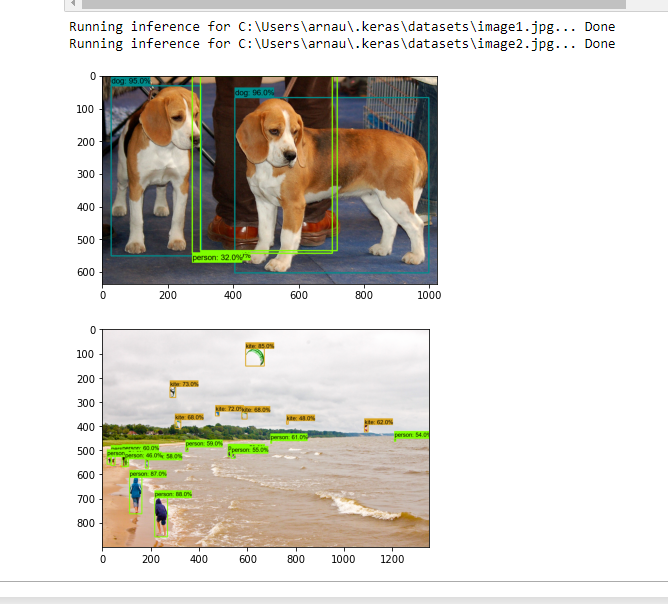
Installer CUDA 11.0 et CUDNN 11.1-windows-x64-v8.0.4.30 (utiliser word d’Antoine)

Commandes site du prof <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html#tensorflow-object-detection-api-installation>

Tester notebook basique utilisant l’object detection :

<https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/auto_examples/plot_object_detection_saved_model.html#sphx-glr-auto-examples-plot-object-detection-saved-model-py>

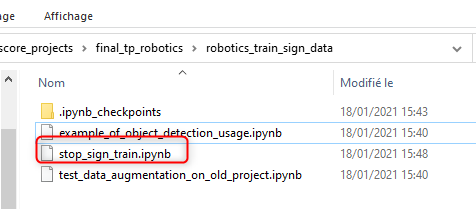


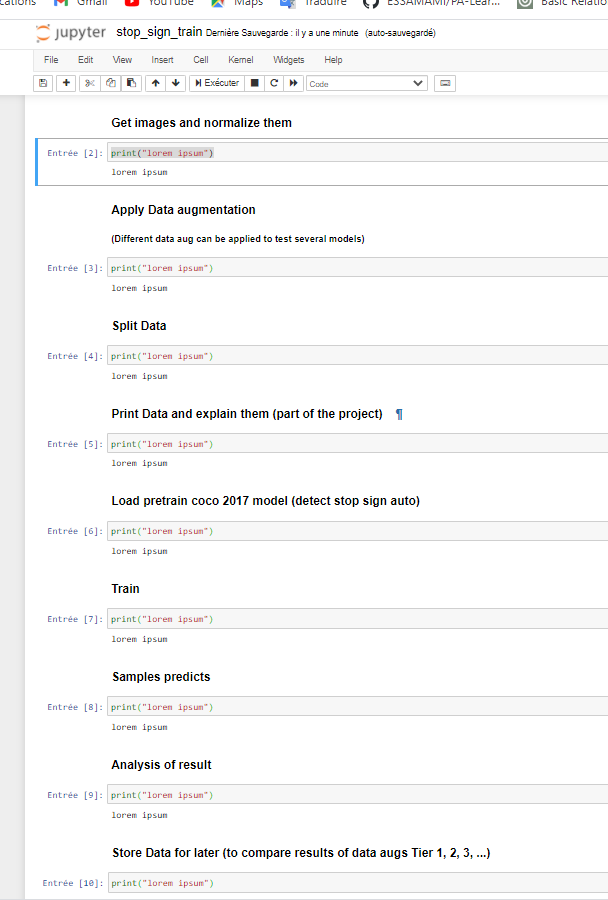


**D – Créer un notebook python (squelette)**

il sera utile pour réaliser l’ensemble des étapes (split, data aug, train, stock résultats dans des dossier pour un future analyse)

Notebook stop sign train :





**E – Récupération des données**

**Cloner ce projet Git**

<https://github.com/EscVM/OIDv4_ToolKit>

**Modifier fichier classes.txt**, supprimez tout et écrivez : **Stop sign**

**Téléchargez le dataset train des stop sign ainsi que le dataset validation puis mergez les 2 dans un dossier**

python3 main.py downloader --classes classes.txt --type\_csv train

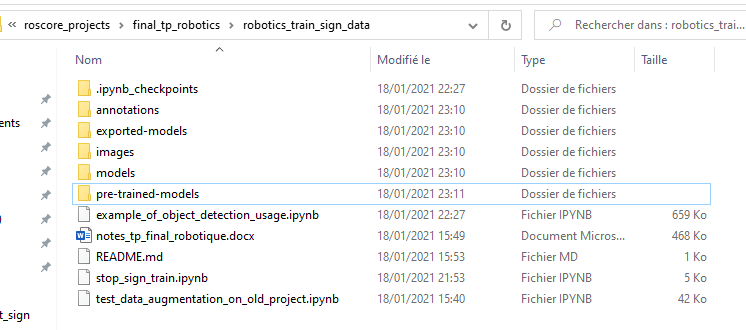
python3 main.py downloader --classes classes.txt --type\_csv test

Les images seront disponibles dans le dossier OID/dataset

*NB : Si vous voulez + de data, j’ai aussi récup 71 panneaux stop sur kaggle, demandez si vous êtes intéressés*

**F – Notebook en suivant les recommandations du tuto donné dans le TP** (<https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#partition-the>)

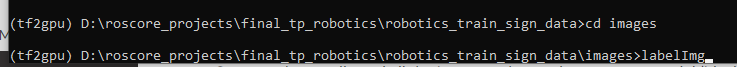
Architecture :



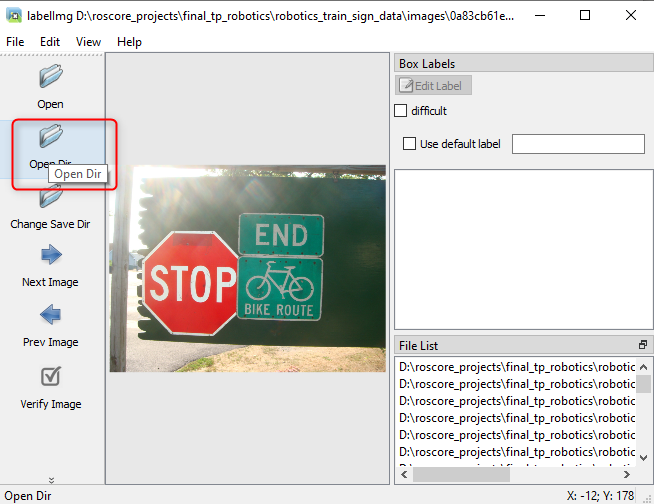
Installez LableImg en version 1.8.3

Placer les images dans le dossier « images »

Démarrez labelImg en tapant « labelImg » dans une CMD (dans le dossier images)



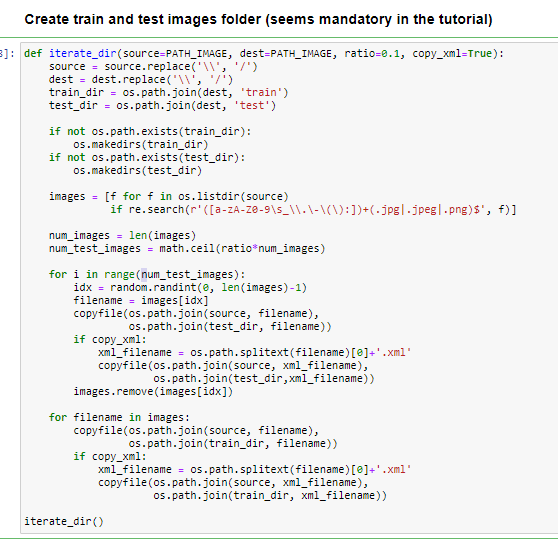
Cliquez sur Open Dir :

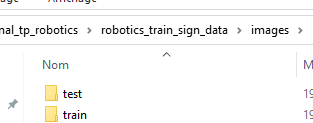


Attribuez des classes pour chaque image en sélectionnant le panneau stop (touche w)

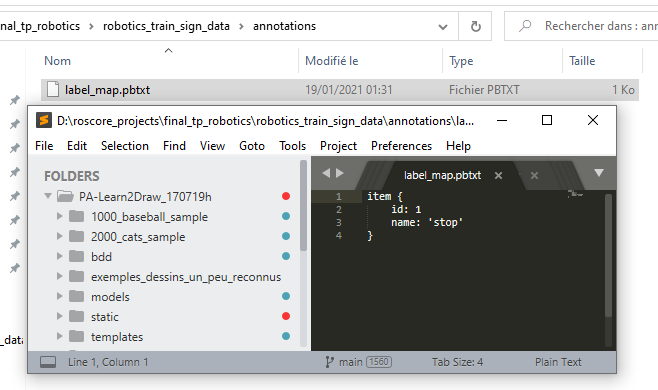
Je l’ai fait pour la partie OIDV4 (388 images)

Créer ensuite une fonction python qui va mettre en place les dossiers d’image train et test (Tuto sans les inputs CMD)

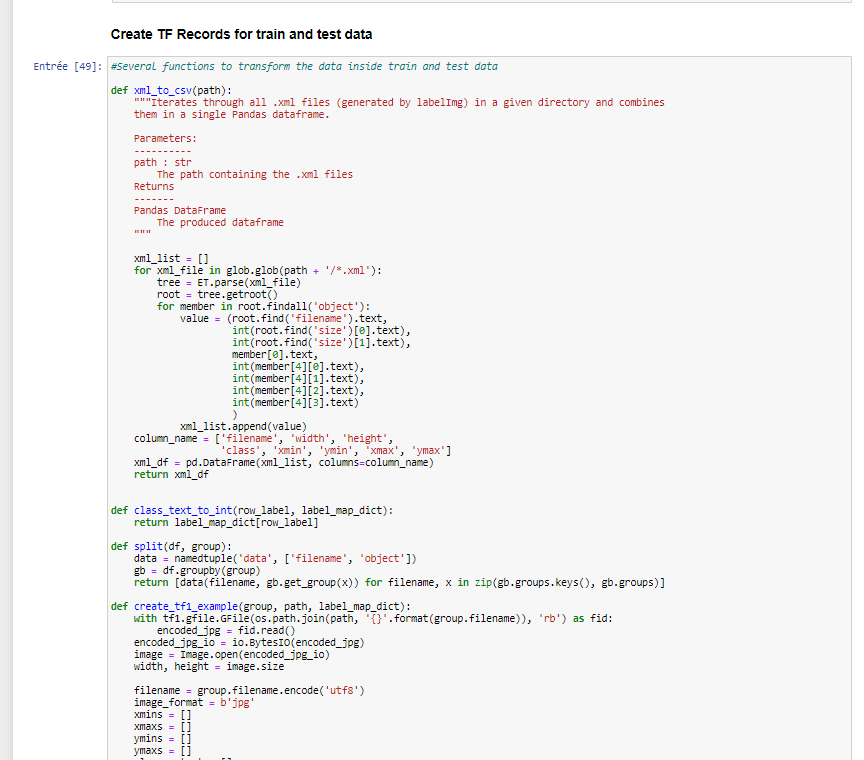


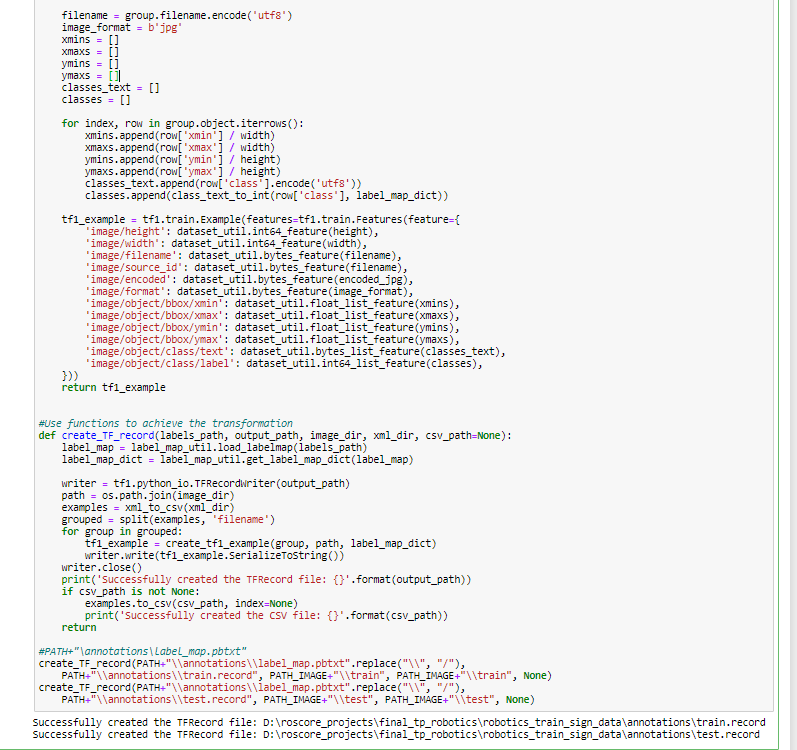


Créer fichier label map qui associe chaque label à un nombre, ici on a qu’un seul label 🡺 stop



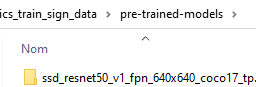
Création de tensorflow records à partir des données test et train



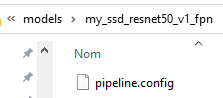


Téléchargement d’un modèle pré entrainé (parmi ceux du TF2 detection model zoo)

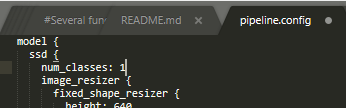
MODEL :  [SSD ResNet50 V1 FPN 640x640](http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz)



Configuration de la pipeline d’entrainement (Créer nouveau dossier dans models puis copier le .config du resnet)



On peut ensuite modifier les paramètres du modèle :



On doit changer le nombre de classes gérées (ici 1)

 (mieux vaut mettre 8)

Changer le batch size si nécessaire

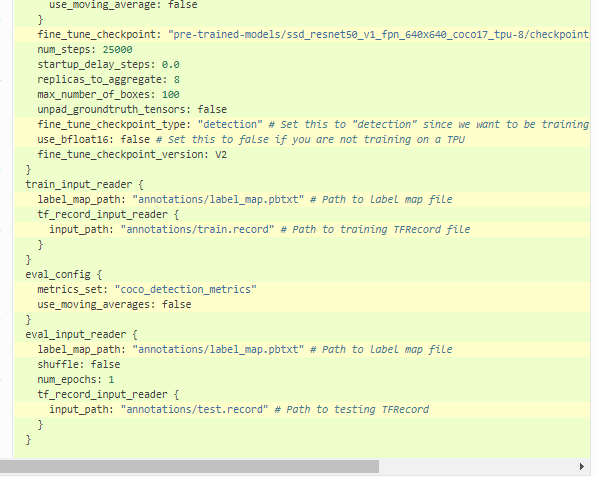
Changer de modèle dans fine\_tune\_checkpoint

Change le type de checkpoint

Ne pas utiliser de bfloat16 (TPU only)

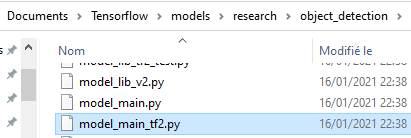
Préciser les chemins du label path et des TF record

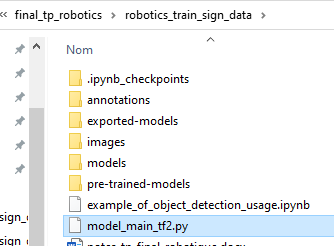
Configuration de la metric, supprimer les moyennes mouvantes :



ATTENTION 🡺 Changez le nombre de steps, 25 000 steps ça pourrait prendre un temps monstrueux (testez avec 400 – 900 steps déjà)

Nous allons maintenant entrainer le modèle, pour ce faire copiez le fichier « model\_main\_tf2.py » du dossier object detection dans le répertoire courant :





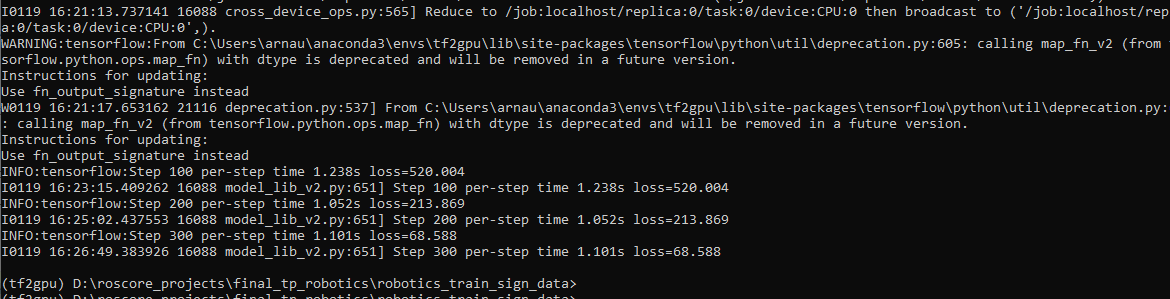
Executez le script suivant dans un terminal (ou via commande spéciale du notebook) :

**(Il peut être intéressant de lancer l’évaluation pendant l’entrainement dans un autre terminal étant donne que les analyses sont realises toutes les 300 sec / puis attendent l arrivee d un checkpoint, voir plus bas)**



python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config –checkpoint\_every\_n=1

Si tout fonctionne bien, vous pourrez « récupérer la main » sur votre terminal :

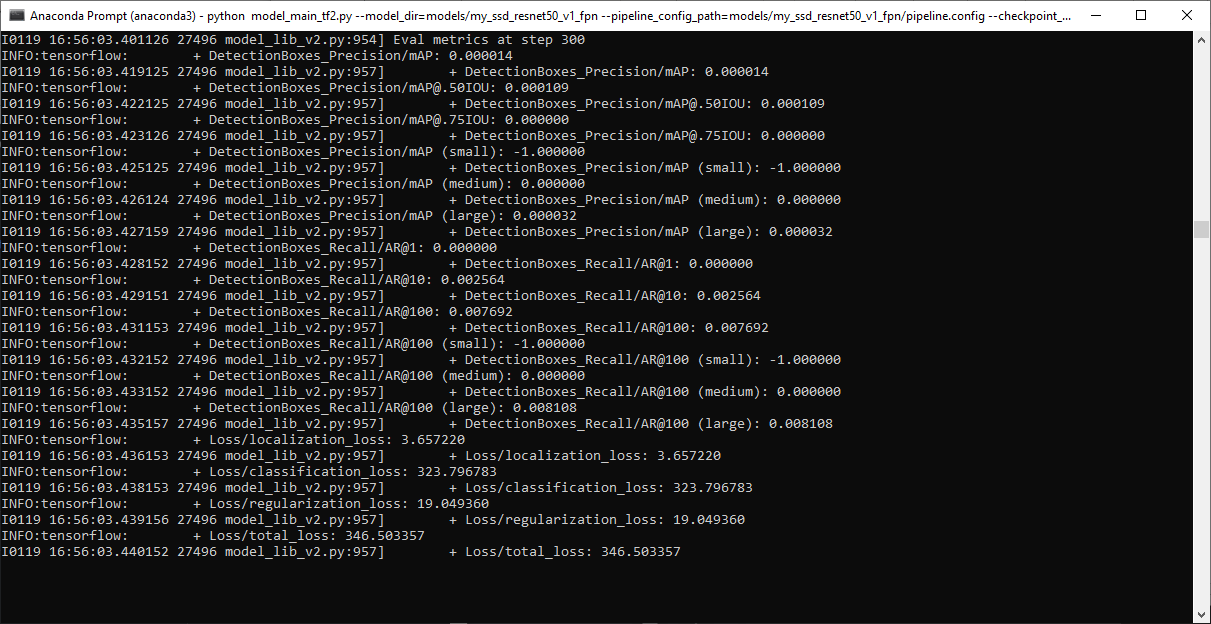


Maintenant nous pouvons évaluer le modèle, pour ce faire exécutez la commande suivante :

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_v1\_fpn

(pour éviter bug éventuel, refaire pip install numpy avec version + vieille, il sera toujours possible de l’update à nouveau plis tard)

pip install numpy==1.17.4



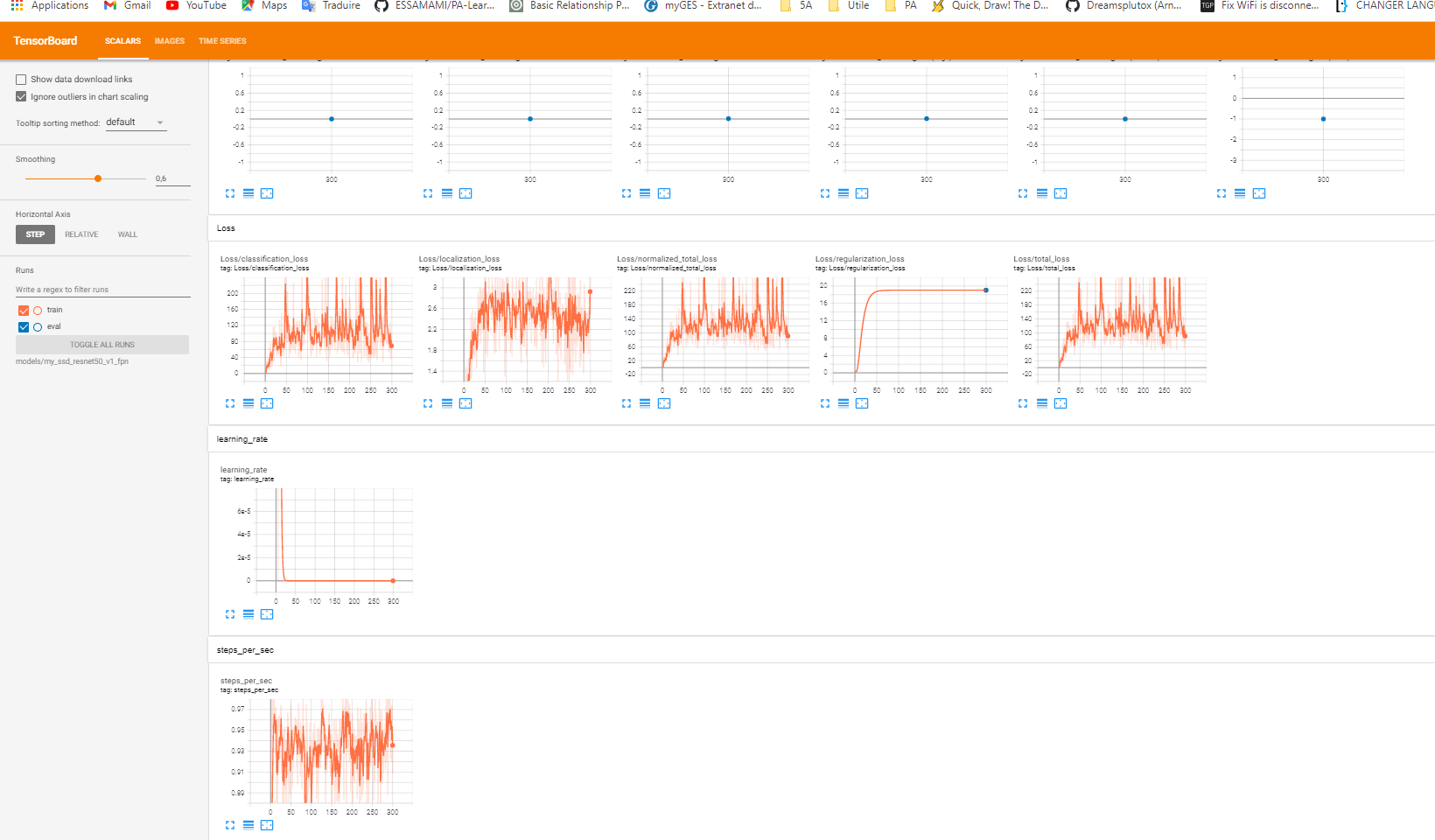
(Il peut être intéressant de lancer l’évaluation pendant l’entrainement dans un autre terminal)

Pour bien monitorer la progrfession de l’entrainement, nous pouvons utiliser tensorboard.

Il suffit de créer un nouveau terminal et d’exécuter cette commande :

tensorboard --logdir=models/my\_ssd\_resnet50\_v1\_fpn

Tensorboard devrait alors être accessible (localhost:6006 par défaut)



En l’occurrence le model ne semble pas très bien fonctionné, il faudrait surement faire + de steps

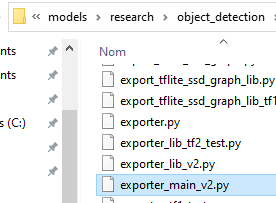
Partie image tensorboard

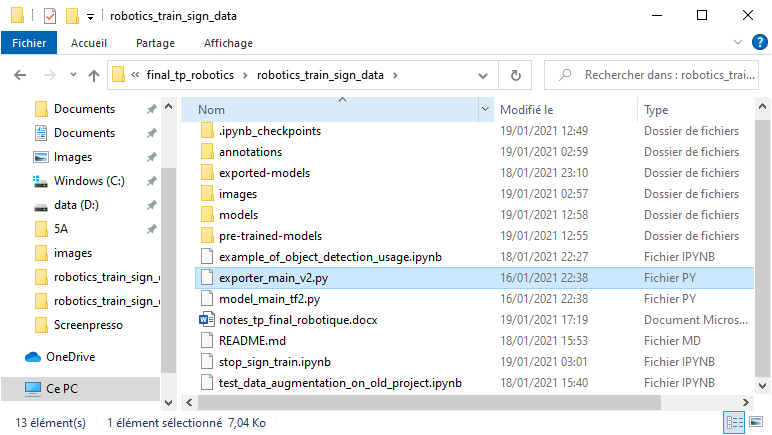


Il faudra très probablement relancer tensorboard + le script d’évaluation + l’entrainement en même temps, sur pas mal de steps (peut être changer la catégorie stop par Stop sign ?)

Export du modèle

Pour exporter le modèle, copiez le fichier exporter\_main\_v2.py (d’object detection) dans le dossier du projet et exécutez la commande suivante :

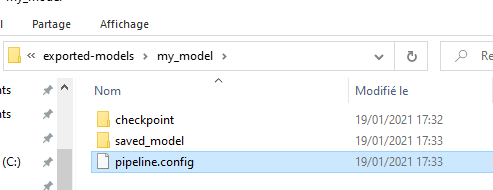




Commande :

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_v1\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_v1\_fpn\ --output\_directory .\exported-models\my\_model

Résultat :



Maintenant que nous avons sauvegardé notre modèle, nous pouvons l’utiliser dans un jupyter notebook (en reprenant a peu près ce tutoriel <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/auto_examples/plot_object_detection_saved_model.html#sphx-glr-auto-examples-plot-object-detection-saved-model-py>) :

(Voir notebook stop train)

MALHEUREUSEMENT, LE MODELE NE DETECTE RIEN SUR LES IMAGES TEST, on réessaye avec un modèle + entrainé

MODELE ENTRAINE DEMO 🡺

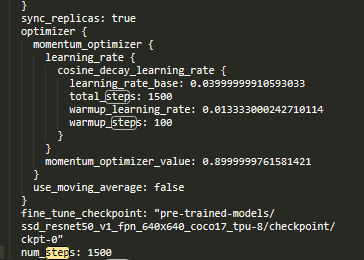


MODELE PERSO (0 detection) :



NOUVEL ESSAI AVEC MODELE PLUS LONGTEMPS ENTRAINE (1500 steps)

Nouveau dossier model, copie pipeline config, modif nb steps :



Remplacer tous les v1 par des v2 (pour les features extractor



Démarrer simultanément : 🡺 FINALEMENT IMPOSSIBLE DE LANCER SIMULTANEMENT, MEMOIRE INSUFFISANTE, on va quand même démarrer tensorboard

**Train**

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v2\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v2\_fpn/pipeline.config –checkpoint\_every\_n=1

**Eval**

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v2\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v2\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_v2\_fpn

**Tensorboard**

tensorboard --logdir=models/my\_ssd\_resnet50\_v2\_fpn

A la fin de l’entrainement, **exporter le modèle**

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_v2\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_v2\_fpn\ --output\_directory .\exported-models\my\_model\_v2

Regénérer fichiers record avec code du python du tuto :

*python generate\_tfrecord.py -x D:\\roscore\_projects\\final\_tp\_robotics\\robotics\_train\_sign\_data\\images\\train -l D:\\roscore\_projects\\final\_tp\_robotics\\robotics\_train\_sign\_data\\annotations\\label\_map.pbtxt -o D:\\roscore\_projects\\final\_tp\_robotics\\robotics\_train\_sign\_data\\annotations\\train.record*

*python generate\_tfrecord.py -x D:\\roscore\_projects\\final\_tp\_robotics\\robotics\_train\_sign\_data\\images\\test -l D:\\roscore\_projects\\final\_tp\_robotics\\robotics\_train\_sign\_data\\annotations\\label\_map.pbtxt -o D:\\roscore\_projects\\final\_tp\_robotics\\robotics\_train\_sign\_data\\annotations\\test.record*

TEST 25K avec new TF RECORDS

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config –checkpoint\_every\_n=1

Eval

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_v1\_fpn

Export

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_v1\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_v1\_fpn\ --output\_directory .\exported-models\my\_model

Fonctionne en changeant le min\_score\_thresh :



TEST 11K avec new TF RECORDS, (il faut stopper vers 11000 quand la loss est basse)

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet\_50\_test\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet\_50\_test\_fpn/pipeline.config –checkpoint\_every\_n=1

Eval

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet\_50\_test\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet\_50\_test\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_test\_fpn

Export

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_test\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_test\_fpn\ --output\_directory .\exported-models\my\_ssd\_resnet50\_test\_fpn

**LES EVAL ET EXPORT NE FONCTIONNE PAS SUR AUTRE CHOSE QUE MY SSD RESENET 50 V1, ON VA TENTER DE REMPLACER LA V1 PAR NOTRE CONTENU (sauvegarder model v1, copiez contenu notre modele a l interieur)**

Eval

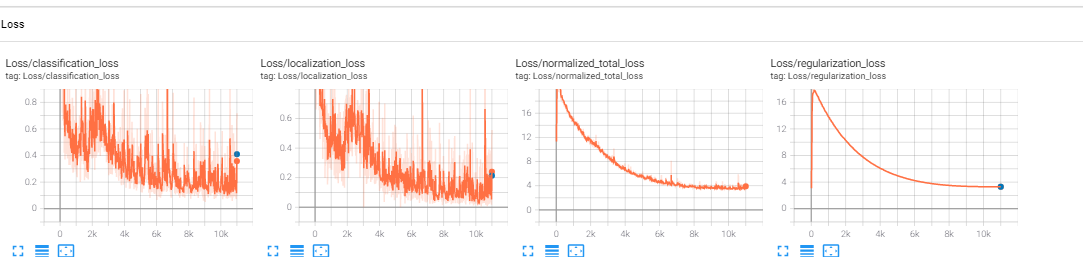
python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_v1\_fpn

Export

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_v1\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_v1\_fpn\ --output\_directory .\exported-models\my\_ssd\_resnet50\_test\_fpn

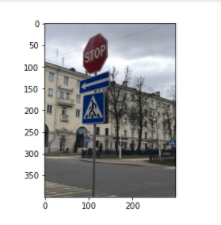
Tensorboard

tensorboard --logdir=models/my\_ssd\_resnet50\_v1\_fpn



Une fois le nouveau modèle obtenu, resauvegarder tout dans un dossier avec le bon nom (ici test plutôt que v1)

ICI CA FONCTIONNE, MEILLEURE PREDICTIONS MAIS MANQUE ENCORE DE PRECISION SUR DES CAS COMPLEXES :





MAINTENANT, PASSONS A LA DATA AUGMENTATION

Objectif 🡺 Créer beaucoup d’images modifiées pour augmenter la taille de notre dataset

Voila l’ensembles des étapes que nous allons réaliser :

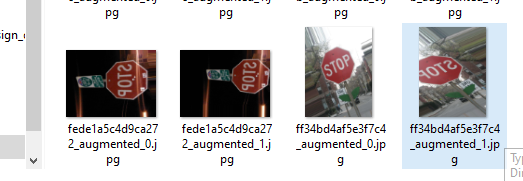
1 – Créer des images avec tensorflow

2 – Les sauvegarder dans un répertoire

🡺



50% des images ont 2 augmentations



3 – Les labelliser avec « LabelImg »

(Parfois les panneaux stop n’apparaissent pas dans les data aug (image trop tournée + décalage suffisant pour cacher panneau stop par exemple 🡺 Dans ce cas j’ai décidé de ne pas créer de fichiers XML, à voir si ça pose pb, si c’est le cas on mettre un stop random dans les images concernées)

🡺A la fin on a 772 éléments (perte de 4 photos, dans le doute on va les supprimer et tout stocker dans un dossier 🡺 code dans notebook, nous a donné

0026b117bb17880a\_augmented\_1.jpg et 574fe3de15a4c169\_augmented\_1.jpg )

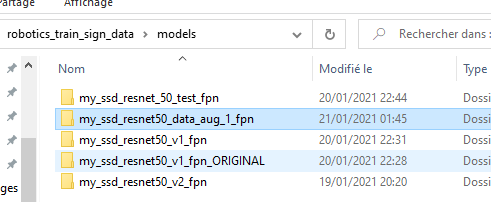
Nous finissons avec 384 images augmentées disponibles ! (avec leurs xml)

4 – Les rajouter dans le dossier images (On sauvegardera les images originales dans un autre dossier) 🡺 772 images dispo

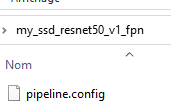
5 – Regénérer dossiers train et test, renommez les tf record pour les conserver, en générer de nouveaux

6 – Refaire un entrainement à 11 000 steps et comparer !

Créer dossier qui accueillera les données :



Placer les données de my\_ssd\_resnet50\_v1\_fpn dans le resnet test, le vider (sauf pipeline.config) 🡺il va accueillir le nouvel entrainement



Voici les commandes à entrer dans les CMD :

**Train**

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config –checkpoint\_every\_n=1

**Tensorboard (en même temps pour suivi logs)**

tensorboard --logdir=models/my\_ssd\_resnet50\_v1\_fpn

**Eval**

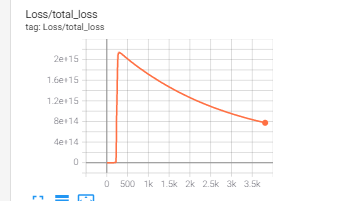
python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_v1\_fpn

**Export**

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_v1\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_v1\_fpn\ --output\_directory .\exported-models\my\_ssd\_resnet50\_data\_aug\_1\_fpn

Copiez ensuite le contenu de my\_ssd\_resnet50\_v1 dans my\_ssd\_resnet50\_data\_aug\_1

Testons :



Loss immense malheureusement, il faudrait surement tenter avec une toute petite data aug (uniquement zoom 0.2 + rotate 45 + width et height shift à 0.05, sans flip)

seulement 24 sur image simple 🡺 INSUFFISANT

Idée 🡺 Problème vient peut être du format des JPG nouvellement créés, si c’est un pb on créera les images en png et on les convertira sur le web en jpg

TEST DATA AUG + LEGERE AVEC 100 IMAGES SUR 11 000 steps

(Supprimer contenu folder images (les augmented précédentes ont été save dans un fichier))

Créer 100 images



Les labelliser

Reset resnet50\_v1 sauf pipeline config, en sauvegardant l’ancien résultat

Voici les commandes à entrer dans les **CMD** :

**Train**

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config –checkpoint\_every\_n=1

**Tensorboard (en même temps pour suivi logs)**

tensorboard --logdir=models/my\_ssd\_resnet50\_v1\_fpn

**Eval**

python model\_main\_tf2.py --model\_dir=models/my\_ssd\_resnet50\_v1\_fpn --pipeline\_config\_path=models/my\_ssd\_resnet50\_v1\_fpn/pipeline.config --checkpoint\_dir=models/my\_ssd\_resnet50\_v1\_fpn

**Export**

python .\exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path .\models\my\_ssd\_resnet50\_v1\_fpn\pipeline.config --trained\_checkpoint\_dir .\models\my\_ssd\_resnet50\_v1\_fpn\ --output\_directory .\exported-models\my\_ssd\_resnet50\_data\_aug\_2\_fpn