Documentation Deep Learning

Baptiste BEHELLE

Septembre 2021

Table des matières

1 Prérequis à installer

Pour faire fonctionner la partie sur le deep learning, il faut installer plusieurs logiciels et bibliotèque :

Tensorflow : qui va permettre de créer des réseaux de neurones, d'entrainer des réseaux et de charger des réseaux pour reconnaître des objets dans des images.
`https://www.tensorflow.org/install?hl=fr`

OpenCV : qui va permettre de faire la passerelle entre une image et un objet mathématique exploitable en data vision
`https://opencv.org/`

label image : Un logiciel tel que LabelImg pour préparer les images avant l'entrainement du réseau et mettre un cadre et un label sur chaque objet que le vont reconnaître sur chaque image que l'on va transmettre pendant l'entrainement

lib Nvidia : Les différentes bibliothèques et outils de Nvidia pour pouvoir utiliser le mode GPU de Tensorflow (cf procédure d'installation) ce n'est pas nécessaire, mais en GPU on aura de meilleures performances que sur CPU.

2 Préparation de l'entrainement

Avant de lancer l'entrainement, il faut télécharger les outils de object detection depuis le gtihub de tensorflow et build le dossier pour pouvoir accéder aux différents programmes.

```
git clone https://github.com/tensorflow/models.git
cd models/research
# Compile protos.
protoc object_detection/protos/*.proto --python_out=.
# Install TensorFlow Object Detection API.
cp object_detection/packages/tf2/setup.py .
python -m pip install .
```

Maintenant que cela est fait, on peut passer à l'acquisition de données, c'est-à-dire récupérer un dataset d'images déjà prêt sur internet ou le faire nous même.

Pour faire un dataset, on commence par récupérer le plus d'images possibles. Puis, on les labels toutes avec LabelImg :

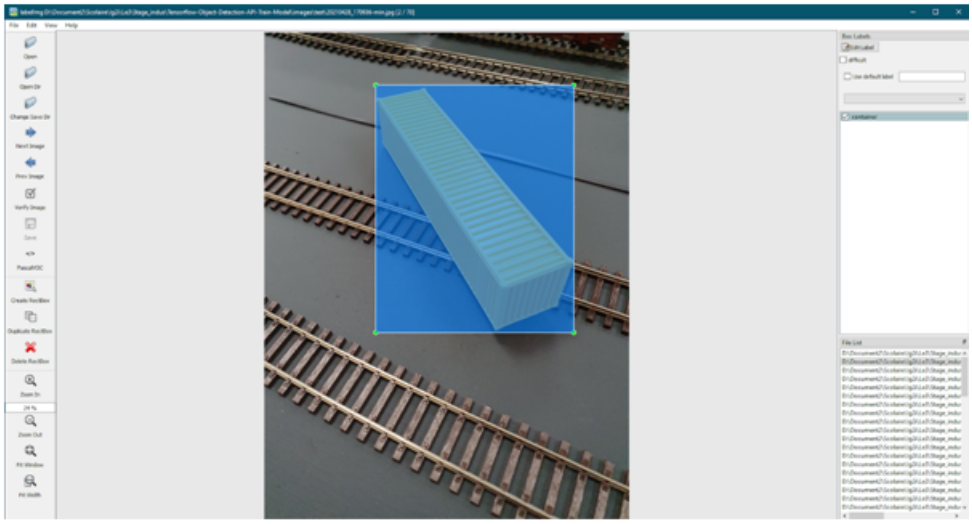

FIG. 1 – *label d'une image de conteneur*

On obtient alors un dossier constitué d'images et de fichiers en .xml contenant la position du rectangle dans l'image.

Pour l'entrainement, il est recommandé de mettre 80% des images dans un dossier train et les 20% restant dans un dossier test pour la validation.

On utilise un outil pour transformer tous ça en .csv

```
python xml_to_csv.py
```

Puis, on modifie le fichier generate_tfrecord pour dire les classes que l'on veut reconnaître :

```python
def class_text_to_int(row_label):
    if row_label == 'container':
        return 1
    else:
        return None
```

On peut alors lancer le programmes sur nos 2 fichiers csv

```
python generate_tfrecord.py --csv_input=images/train_labels.csv
--image_dir=images/train --output_path=train.record
```

```
python generate_tfrecord.py --csv_input=images/test_labels.csv
--image_dir=images/test --output_path=test.record
```

La préparation est presque terminée, il faut maintenant créer le fichier labelmap.pbtxt qui contient la liste des classes à reconnaître :

```
item {
    id: 1
    name: 'container'
}
```

On peut alors télécharger un modèle tf2 puis modifier son fichier de config pour préparer l'entrainement :

class_number : le nombre de classes à reconnaître

batch_number : plus la valeur est faible moins analyse d'images en même temps

fine_tune_checkpoint : le chemin vers le ckpt-0 du models

input_path : pour indiquer le chemin vers le train.record ou le test.record

3 L'entrainement

On peut lancer alors l'entrainement de la façon suivante et attendre le temps que l'on veut pour faire tourner le réseau selon le niveau d'apprentissage voulu.

```
python model_main_tf2.py --pipeline_config_path=training/models.config
--model_dir=training --alsologtostderr
```

On peut tous de même surveiller la bonne exécution de l'entrainement et voir l'évolution du taux d'apprentissage etc grâce à un petit utilitaire :

```
tensorboard --logdir=training/train
```

Il va ouvrir une page sur le port 6006 de loacalhost et on pourra y voir des infos sur le réseau :

4 Exporter le modèle généré

Une fois que le modèle est assez entrainé, on peut juste kill le programme puis lancer ce dernier qui va récupérer le dernier modèle et en générer une version exploitable :

```
python exporter_main_v2.py --trained_checkpoint_dir=training
--pipeline_config_path=training/models.config
--output_directory models_generate
```

On pourra par la suite l'utiliser dans nos autres projets tels que sur ROS à partir du flux vidéo de la caméra du Rapsberry.