RAPPORT DE PJI
Numéro : 39
Utilisation du deep learning pour la détection d'obstacles pour le véhicule autonome

BAAS Simon - MEINAS Julien
Master informatique – Université de Lille
Janvier 2020 - Mai 2020

Tuteur : DHERBOMEZ Gérald
Etablissement : Université de Lille

Sommaire

Introduction

Lors du deuxième semestre du master Informatique de l'université de Lille, nous avons dû réaliser un projet au cours de l'UE PJI.
Ce projet consistait à reprendre un travail déjà réalisé afin de l'améliorer ou bien de commencer un tout nouveau projet.
Nous nous sommes penché sur le projet suivant, "Utilisation du deep learning pour la détection d'obstacles pour le véhicule autonome".

Pourquoi ce choix ?

Nous avons été très intéressé par ce projet car il touche des technologies très récentes et peu étudiées en cours. En effet le deep learning et les voitures autonomes sont des sujets très travaillés ces dernières années ce qui a permis à de grandes avancées dans le domaine. Ces sujets vont donc continuer à évoluer ces prochaines années.

Ce projet a été alors pour nous une grande opportunité afin de découvrir de nouveaux domaines très intéressants de l'informatique. En particulier le deep learning qui est une branche de l'intelligence artificielle. C'est un domaine que nous avons survolé lors du semestre 1 avec l'UE PJE Twitter. Dans cette UE nous avons vu les algorithmes qui fondent la base de ce qu'est le deep learning.

Ainsi une fois le projet attribué, nous nous sommes penchés sur les différentes manières de résoudre celui-ci. Il est maintenant question de voir les enjeux d'un tel projet et d'étudier les principaux choix qui s'offrent à nous afin de le concevoir.

1 - Présentation du projet

Aujourd'hui avec le développement de l'informatique, le monde devient de plus en plus facile d'utilisation. L'un des principaux objectifs aujourd'hui, est de le rendre automatique et indépendant et ce grâce à l'intelligence artificielle.

1.1 - Introduction du véhicule autonome

Le domaine qui nous intéresse ici est l'automobile. Une échelle a été établie afin de voir les différents niveaux d'autonomies.

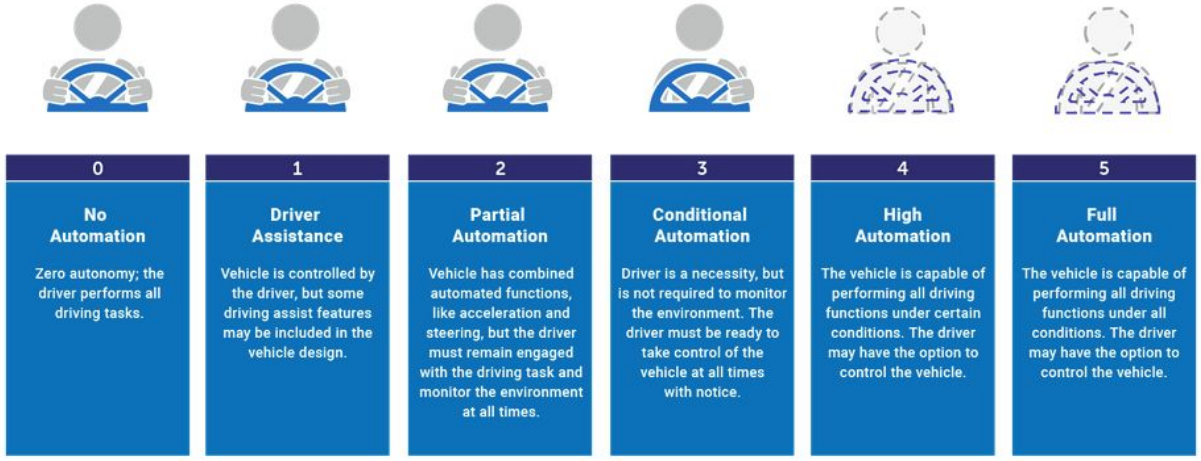

Figure 1 : Les niveaux d'autonomie du véhicule autonome

Les premières voitures qui ont été créé, étaient totalement dépendante du conducteur, on parle ici du niveau 0 d'autonomie d'une voiture.

Peu à peu les voitures de niveau 0 ont disparu pour laisser place à des véhicules toujours très dépendants du conducteur mais avec de petites assistances intéressantes. Ces assistances sont des aides à la conduite pour aider le conducteur à diriger, freiner ou accélérer. On parle ici du niveau 1 d'autonomie d'une voiture.

Le niveau 2 d'autonomie d'une voiture est très vite apparu, permettant de contrôler simultanément la direction et le freinage/l'accélération dans certaines circonstances. Cependant, le conducteur reste toujours au contrôle du véhicule.

Aujourd'hui la plupart des véhicules sont de niveau 3. À ce stade les voitures peuvent rouler en totale autonomie mais seulement dans des environnements bien spécifiques. Lorsque ce n'est pas le cas, la voiture demande au conducteur de reprendre la main.

C'est le cas notamment des voitures Tesla. Celle-ci possède une série d'équipements avancés permettant d'automatiser complètement la voiture. Ce sont les fonctionnalités d'Autopilot.

Au niveau des autres compagnies, il y a aussi Mobileye. Mobileye appartient à Intel et est le leader dans le développement de technologies pour assistance dans la conduite.

Figure 2 : Logo Mobileye

Sur le site de Mobileye nous retrouvons les différentes fonctionnalitées suivantes. Mobileye prend en charge une suite complète de fonctions ADAS, AEB, LDW, FCW, LKA, LC, TJA. Sur le site de Mobileye, on peut retrouver la définition de chacune d'entre elles.

ADAS (Advanced Driver Assistance Systems) Basé sur un spectre passif/actif. Un système passif alerte le conducteur d'un danger potentiel. Ainsi le conducteur peut effectuer une action afin de corriger l'erreur. Par exemple :

- LDW (Lane Departure Warning) Avertissement de sortie de voie. Avertit le conducteur d'un départ de voie involontaire
- FCW (Forward Collison Warning) Avertissement de collision avant. Indique dans la dynamique actuelle par rapport au véhicule qui précède, une collision est imminente. Le conducteur doit freiner pour éviter la collision.

Au contraire, les systèmes de sécurité actif prennent la main. AEB (Automatic Emergency Baking) Freinage d'urgence automatique, identifie la collision et les freins imminents sans aucune intervention du conducteur. ACC (Adaptive Cruise Control) Régulateur de vitesse adaptatif. LKA (Lane Keeping Assist) Assistant de

maintien de voie. LC (Lane Centering) Le centrage de voie. TJA (Traffic Jam Assist) Assistant d'embouteillage. TSR (Traffic Sign Recognition). IHC (Intelligent High-beam Control). Toutes ces fonctions sont prises en charge à l'aide d'une seule caméra montée sur le pare-brise.

Mobileye possède un système d'avertissement qui peut être installé sur n'importe quel véhicule existant. Mobileye se base uniquement sur les caméras.

Le niveau 3 d'autonomie est particulièrement dangereux du fait que lorsque la période d'autonomie se termine, le conducteur a peu de temps pour reprendre la main.

Les niveaux particulièrement recherchés par l'industrie automobile sont les niveaux 4 et 5. Nous avons ici des véhicules totalement autonomes en toutes circonstances et qui ne demanderont pas au conducteur de reprendre le volant en peu de temps.

L'objectif du projet est de rendre un véhicule autonome, c'est-à-dire de le faire passer au niveau 3, voir au niveau 4.
La recherche principale de notre projet est de mettre en place un algorithme compatible avec un système de caméras. Celui-ci sera placé sur le véhicule à différents endroits afin de détecter les obstacles sur la route telle que des véhicules ou bien des piétons, puis identifier les dangers potentiels.

Par la suite, en complément de la détection, le but sera de prévoir les mouvements potentiels de l'obstacle.

1.2 - Introduction du deep learning

Mais comment détecter, reconnaître un objet sur une image ?
L'une des grandes difficultés de notre projet était de mettre en place un algorithme prenant en entrée une image et donnant en sortie cette même image avec les objets détectés.
L'informatique "traditionnel" ne nous permet pas de faire cela. Nous avons ici des données d'entrée beaucoup trop complexes et des résultats possibles en sortie beaucoup trop nombreux.

Une bonne solution est alors d'utiliser l'apprentissage automatique ou intelligence artificielle. Une première possibilité consiste à utiliser le machine learning. C'est un domaine de l'IA qui peut permettre dans notre cas à apprendre à un réseau de neurones à reconnaître des objets sur une image.

Mais le problème est qu'il faut fournir à ce réseau de neurones des données en entrées qui caractérisent l'objet que l'on veut reconnaître. Par exemple pour reconnaître une voiture, il faut dire au réseau que l'objet à quatre roues, des fenêtres, etc.

Le problème ici est qu'il faut l'intervention d'un humain pour donner les caractéristiques de l'objet en question. Pour contourner ce problème, un nouveau domaine de l'informatique, appelé deep learning, est apparu en 2013 notamment grâce à Yann Le Cun.

Le deep learning fonctionne exactement comme le machine learning mis à part que nous ne devons pas lui fournir les caractéristiques de l'objet à reconnaître. L'algorithme va faire lui-même le travail en amont, grâce à un apprentissage sur une base de données.

De ce fait le plus important est la base de données afin d'effectuer l'apprentissage. Lors de cette manoeuvre, nous devions donner des images où les objets que l'on souhaite détecter sont bien identifié. Les images contenant l'objet désiré sont appelées, images positives. Il est aussi important d'avoir un apprentissage se basant sur des images négatives, des images ne contenant pas l'objet.

L'apprentissage nécessite tout de même un travail manuel en amont. Celui-ci demande beaucoup de rigueur afin d'être efficace.

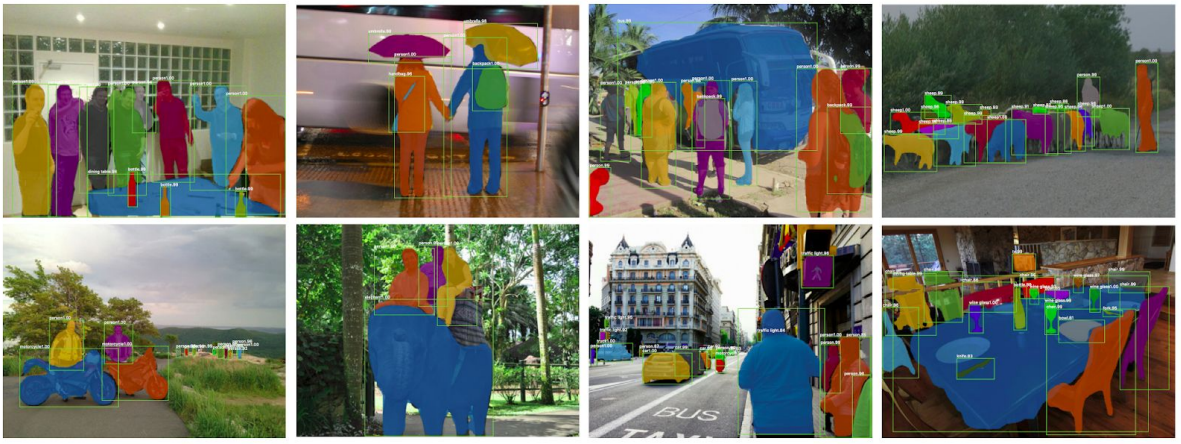

Figure 3 : Exemple de détection d'objets

À partir de la base de données, le réseau de neurones va alors apprendre la caractérisation des objets identifiés pour ensuite l'appliquer sur de nouvelles images. C'est pour cela que plus les images d'apprentissage seront correctement décrite et plus le réseau sera de qualité.

Pour ce qui est de l'algorithme de détection d'objets. Celui-ci utilise la méthode de Viola et Jones. Cette méthode de détection d'objet proposée par les chercheurs Paul Viola et Michael Jones en 2001, fait partie des premières méthodes capables de détecter efficacement et en temps réel, des objets dans une image.

À l'origine cette méthode a été conçue pour reconnaître des visages, mais elle peut également être utilisée pour reconnaître tout types d'objets.

Ainsi toute la difficulté dans un premier temps, consiste à élaborer une base d'apprentissage assez conséquente. Pour ensuite pouvoir l'utiliser dans un algorithme qui puisse traiter une image ou un flux vidéo afin d'afficher en sortie les objets détectés. Par la suite l'idée sera d'envisager les mouvements envisageables par l'objet détecté.

2 - Le développement d'un programme de détection d'obstacles

Étant donné que ce projet intègre des domaines nouveaux pour nous, la grande première partie de notre projet a été un travail de recherche.
Dans un premier temps nous avons découvert à travers nos premières recherches, une introduction au deep learning mais aussi à la voiture autonome.

Une fois la partie théorie acquise, nous nous sommes lancé dans la recherche de projets répondant à nos attentes.
Nous avons pu trouver différents projets open source permettant de faire de la reconnaissance d'objets sur des images. Ces algorithmes utilisent de grandes bases de données pour la partie apprentissage des réseaux de neurones. Nous avons donc également recherché des bases d'images.

Voyons tout d'abord les principaux langages permettant de faire de reconnaissance d'obstacles.
Dès le début de nos recherches, nous avons très vite découvert un outil créé par Google, il s'agit de TensorFlow.

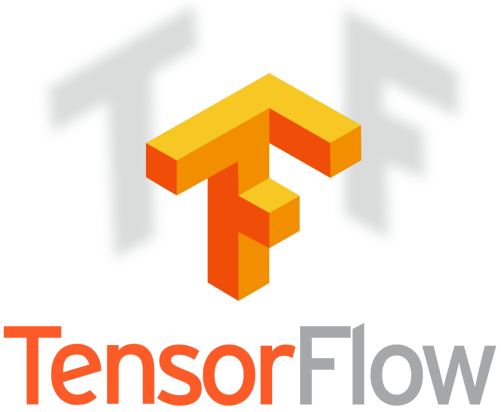

Figure 4 : Logo de TensorFlow

TensorFlow est une bibliothèque open source de deep learning créée par Google, permettant de développer et d'exécuter des applications de machine learning et de deep learning.

En poussant nos recherches, nous avons trouvé d'autres outils permettant de faire du deep learning. Par exemple Mixapel, Kount, Pyramid ou encore PyTorch.

2.1 - Principaux langages utilisés

En réalisant nos recherches nous avons très vite compris que la plupart de ces outils sont destinés aux entreprises et sont en conséquents payants.
Cependant deux des outils ont attiré notre attention, il s'agit TensorFlow et de PyTorch.

Figure 5 : TensorFlow vs PyTorch

TensorFlow qui a été créé en 2015 par Google reste aujourd'hui leader dans le marché du deep learning. Cette technologie permet de développer des architectures d'apprentissage et de les transformer en logiciels et ainsi apporter ses extensions.
TensorFlow regroupe un grand nombre de modèles et d'algorithmes de machine learning et de deep learning.
TensorFlow comporte de nombreux avantages tels que l'abstraction qui facilite l'implémentation d'algorithme et permet au développeur de se focaliser sur la logique générale d'une application. On y trouve aussi la personnalisation de l'interface ou encore la différenciation automatique.
Le grand avantage de TensorFlow reste sa notoriété dans le domaine du deep learning et ainsi la grande quantité de documentations.

PyTorch de son côté a été créé en 2016 par Facebook. L'outil a su se démarquer par la simplicité de son architecture simple et ses graphiques dynamiques. De plus, PyTorch avec son graphique de calcul défini à l'exécution, nous pouvons utiliser les outils de débogage Python tels que pdb, ipdb.

Depuis leurs sorties les deux logiciels tiennent le marché du machine learning et du deep learning, bien que TensorFlow en soit leader.

Au vu de nos recherches nous nous sommes plus dirigé vers TensorFlow qui est le plus connu et qui contient des ressources et du code open source extrêmement riche. Cela nous sera d'une très grande aide afin de trouver des projets open source permettant de faire de la reconnaissance d'obstacles sur des images.

En ajout à TensorFlow, nous allons utiliser la bibliothèque graphique OpenCV.

Figure 6 : Logo d'OpenCV

Grâce à OpenCV nous allons pouvoir effectuer du traitement sur des images et des vidéos. Cette bibliothèque nous servira principalement à traiter le flux venant d'une caméra ou bien d'un fichier vidéo.

2.2 - Les aspects positifs

Au niveau des points positifs nous avons tout d'abord remarqué la très grande utilisation de TensorFlow. Un grand nombre de personnes s'étant mises au deep learning/détection d'obstacle utilisent TensorFlow. Ainsi cela permet de trouver un très grand nombre de réponses par rapport aux différents problèmes rencontrés.

En plus d'être très largement utilisé, TensorFlow fournit une API et est compatible avec un grand nombre de frameworks. Lors de la création du projet, l'idée était de trouver une base afin de commencer le projet. Nous devions trouver une base car créer un réseau de neurones from scratch, c'est-à-dire de zéro, est très compliqué et demande beaucoup de temps et de connaissances.
Ainsi pour pallier à ce problème, Google a mis à disposition une API TensorFlow (TensorFlow Object Detection API) afin d'avoir à disposition quelques tutoriels et modèles pré entraînés.

En plus de cela l'API dispose de diverses explications notamment sur la hiérarchie du projet, la compilation et l'utilisation de l'API COCO.

À partir de cette API nous avons pu comprendre le tutoriel décrit et l'utiliser comme base de développement.

2.3 - Les aspects négatifs

L'API TensorFlow permet une bonne compréhension des différents sujets qu'elle traite. Néanmoins l'installation de celle-ci nécessite une certaine rigueur. Afin de l'utiliser, nous devions installer différentes librairies liées à pip et bien faire attention aux versions utilisées, notamment à celle de TensorFlow.

En plus d'effectuer les différentes étapes décrites dans l'installation de l'API, il fallait aussi installer différents packages afin de mettre à jour les drivers CUDA. Il faut donc suivre les différentes étapes décrites sur le tutoriel Nvidia en fonction de son architecture personnelle.

Une fois cela effectué, il fallait en plus bien suivre l'installation des différentes librairies python. Les différentes librairies s'installaient avec pip, il fallait donc aussi faire attention à la version de de pip.

Outre l'aspect installation, c'est la hiérarchie des modèles TensorFlow qui peut paraître un peu lourde. En effet le GitHub comprend des projets autres que la détection d'obstacles.

En réalisant l'installation de l'API TensorFlow nous avons pu voir un grand point négatif à cette technologie, en effet la mise en place demande l'installation d'un grand nombre de librairies diverses qui n'étaient parfois pas compatibles avec notre matériel qu'il fallait donc modifier en conséquence.

Une fois toutes les librairies installées il nous a fallu également installer différents packages afin de mettre à jour les drivers CUDA.

Après avoir réalisé tout cela, il nous a fallu suivre le tutoriel fourni avec l'API qui est heureusement très bien détaillé.

3 - Développement de notre projet

L'installation du projet est décrite dans l'annexe.

3.1 - Utilisation de l'API TensorFlow

Une fois toutes les installations et préparations effectuées, nous avons pu élaborer le tutoriel donné afin d'avoir un script utilisant l'API TensorFlow pour réaliser la détection et l'identification d'objets sur une image donnée.

L'étape 1 du script consiste à récupérer un modèle pré entraîné afin d'avoir une base de données pour identifier les éléments. Un lien décrit dans l'annexe permet de choisir le modèle COCO désiré.

```
# What model to download.
# MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
# MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'
# MODEL_NAME = 'ssd_inception_v2_coco_2017_11_17'
MODEL_NAME = 'ssd_inception_v2_coco_2018_01_28'
# MODEL_NAME = 'faster_rcnn_inception_v2_coco_2018_01_28' # SLOW
# MODEL_NAME = 'mask_rcnn_inception_v2_coco_2018_01_28' # SLOW
# MODEL_NAME = 'faster_rcnn_inception_v2_coco_2018_01_28' #SLOW
# MODEL_NAME = 'inference_graph' # OWN MODEL

MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
```

Figure 7 : Récupération d'un modèle

À ces images doivent être associée la description des éléments sur chaque image. Cela est très important, plus la description est précise et plus le modèle d'apprentissage sera performant. Par exemple une image avec un label décrivant la présence de quatre personnes sur l'image sera beaucoup moins intéressante qu'un label décrivant la description de chaque personne (jeune, âgé...) avec la position exacte.

```
# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
# PATH_TO_LABELS = 'training/labelmap.pbtxt' #OWN LABELMAP

# NUM_CLASSES = 90
NUM_CLASSES = 5 #OWN CLASSES
```

Figure 8 : Nombre de classes labellisées

Ici par exemple, le modèle compte 90 classes différentes qui pourront être détectable sur les prochaines images.

Une fois le modèle sélectionné, si celui-ci n'est pas présent dans le répertoire, une fonction permet de le télécharger.

```
# ## Download Model

# In[5]:
print("Downloading model")

opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())

print ("Loading frozen model into memory")
```

Figure 9 : Fonction de téléchargement du modèle

Puis nous mettons en mémoire le modèle préalablement téléchargé au chemin PATH_TO_CKPT.

```
# In[6]:

detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name = '')
```

Figure 10 : Mise en mémoire du modèle

Une fois tout cela effectué, nous avons un modèle TensorFlow qui à partir de sa base d'apprentissage, est capable d'effectuer une détection d'obstacles pour une image donnée.

```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 3) ]
```

Figure 11 : Chemin vers le dossier d'images de test

Cela est très intéressant mais pour la détection d'obstacles qui sera utilisé sur la voiture autonome il est nécessaire d'effectuer la détection sur un flux vidéo.

En effet le script tutoriel TensorFlow permet uniquement de détecter des obstacles sur des images données.
Alors pour contourner ce problème, nous avons utilisé la librairie OpenCV. Ainsi on permet au script de prendre un flux vidéo en entrée. En entrée il interprète le flux vidéo comme une image qui s'actualise toutes les x secondes.
Si les captures d'écran sont réalisées dans un intervalle petit alors la détection d'obstacles se fait de manière fluide sur une caméra.

```
# In[10]:

with detection_graph.as_default():
  with tf.Session(graph = detection_graph) as sess:
    while (capture.isOpened()):
      ret, image_np = capture.read()
      # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
      image_np_expanded = np.expand_dims(image_np, axis = 0)
      image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
      # Each box represents a part of the image where a particular object was detected.
      boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
      # Each score represent how level of confidence for each of the objects.
      # Score is shown on the result image, together with the class label.
      scores = detection_graph.get_tensor_by_name('detection_scores:0')
      classes = detection_graph.get_tensor_by_name('detection_classes:0')
      num_detections = detection_graph.get_tensor_by_name('num_detections:0')
      # Actual detection.
      (boxes, scores, classes, num_detections) = sess.run(
          [boxes, scores, classes, num_detections],
          feed_dict = { image_tensor : image_np_expanded })
      # Visualization of the results of a detection.
      vis_util.visualize_boxes_and_labels_on_image_array(
          image_np,
          np.squeeze(boxes),
          np.squeeze(classes).astype(np.int32),
          np.squeeze(scores),
          category_index,
          use_normalized_coordinates = True,
          line_thickness = 8)
```

Figure 12 : Fonction principale de détection

Nous avons ici une boucle while qui effectue à chaque fois une capture du flux vidéo suivi du traitement de l'image par le modèle.

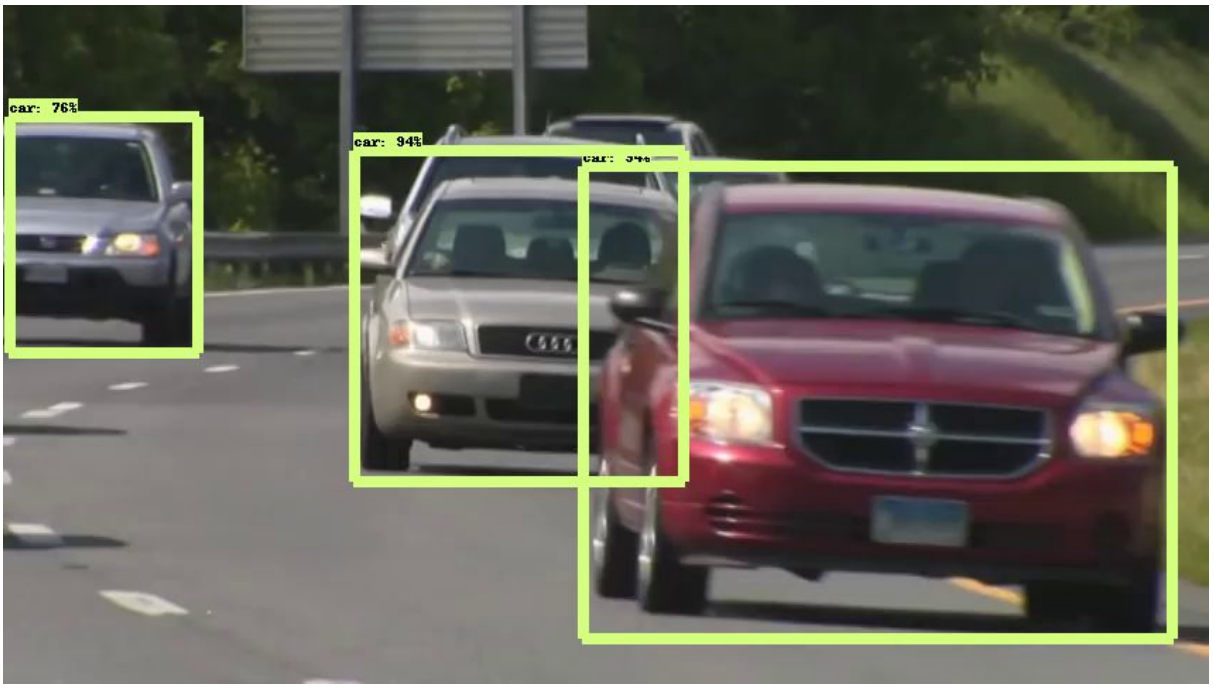

Figure 13 : 1er exemple de l'algorithme sur une vidéo avec un modèle COCO

Une fois ceci réalisé, il est possible d'améliorer le script en utilisant une base d'apprentissage plus riche et plus détaillé. La détection d'obstacles sera de meilleur qualité et plus précise.
Par exemple dans notre modèle nous pouvons détecter les voitures, mais si dans notre base de données d'apprentissage, nous identifions les voitures avec leur marque. Alors à l'avenir nous pourrions avoir un modèle capable de détecter une voiture et sa marque.
Les possibilités d'amélioration sont très nombreuses et tout cela est possible en améliorant la base d'apprentissage.

3.2 - Création d'un modèle

Pour créer un modèle il est nécessaire d'avoir une certaine quantité d'image. Une fois ces images rassemblées, il faut toutes les redimensionner à la même taille. Ensuite 80% des images servent d'images d'entraînements et le reste d'images de tests.

Si celles-ci ne sont pas labellisées, il est possible de le faire à l'aide d'un logiciel. Une fois fait, chaque image est enregistrée sous un format xml. Nous devons

ensuite convertir tous ces .xml en deux fichiers au format csv à l'aide d'un script. Le script est déjà présent dans répertoire de l'API sous le nom . L'exécution de ce script nous renvois les deux fichiers suivants, test_labels.csv et train_labels.csv.

Il faut ensuite modifier le fichier generate_tfrecords.py en fonction des classes utilisées dans la labéllisation.
À partir de ce fichier python et des deux fichiers csv, on peut générer deux fichiers, train.record et test.record qui vont nous servir à entraîner notre modèle.

Avant ça il nous faut créer un fichier labelmap contenant les différents noms de nos classes. Celui-ci sera placé dans un dossier training.

Il est nécessaire d'avoir un fichier de configuration pour entraîner son modèle. Nous allons nous baser sur une configuration d'un modèle prédéfini. Il faut en conséquence changer quelques éléments dans le fichier tel le nombre de classe et différents chemins.

Ensuite nous pouvons entraîner le modèle à l'aide d'un script déjà présent dans l'architecture. Il est donc possible de relancer l'entraînement plusieurs fois afin d'avoir un modèle de plus en plus performant.

Une fois le modèle entraîné, il faut l'exporter sous forme de graphe afin qu'il soit lu par l'algorithme de détection d'obstacles.
Le modèle peut ensuite être testé par l'algorithme.

Nous avons voulu utiliser une petite base d'apprentissage afin de voir les performances que l'on pouvait obtenir.

Nous avons donc cherché par nous-même, des images contenant différents types de véhicules. Nous nous sommes basées sur 5 classes, les voitures, les piétons, les camions, les motos et les bus. Nous avons récupéré 20 images et labellisé chacunes d'entre elles à l'aide du logiciel LabelImg.

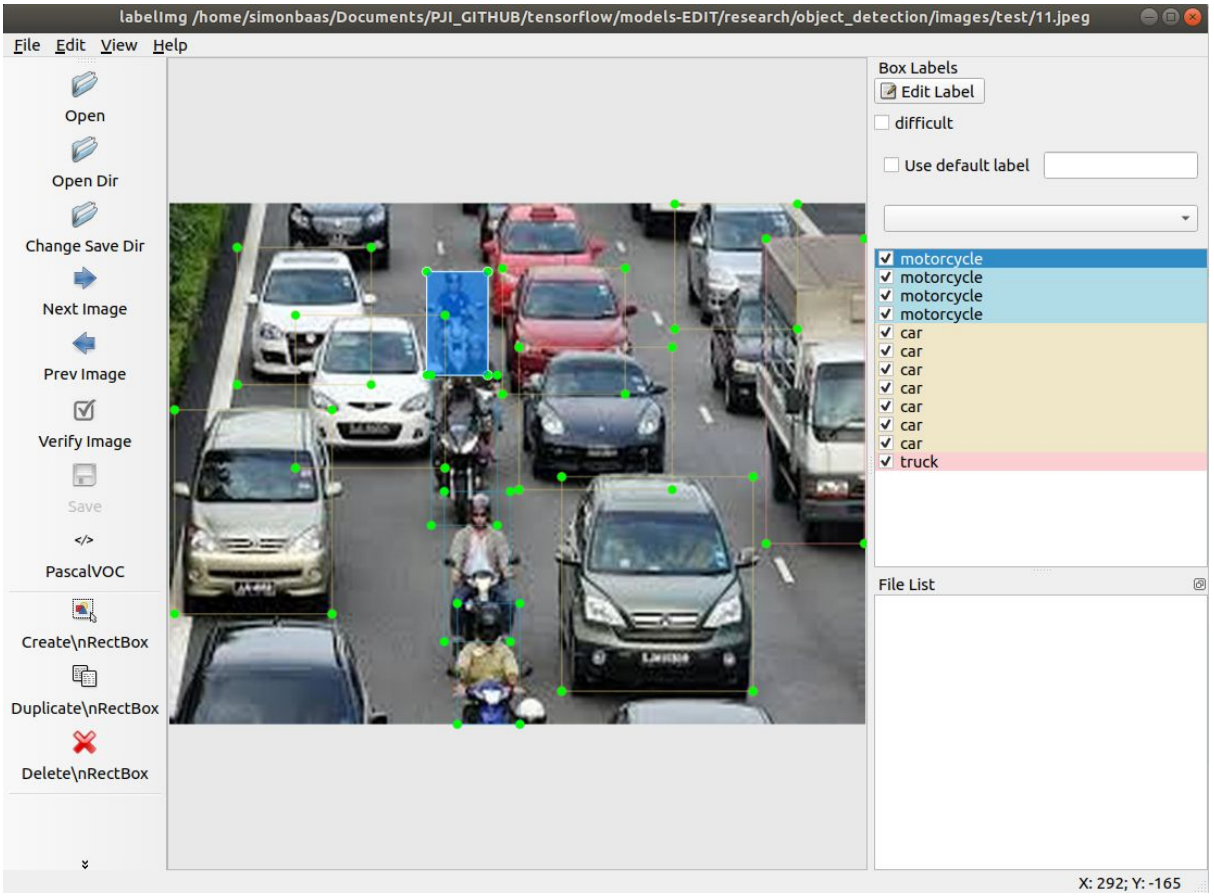

Figure 14 : Exemple d'utilisation du logiciel LabelImg

Nous avons ensuite suivi les différentes étapes décrites plus haut.

Lors de l'exécution de l'algorithme avec notre modèle, nous avons pu constater les résultats au travers de la détection.

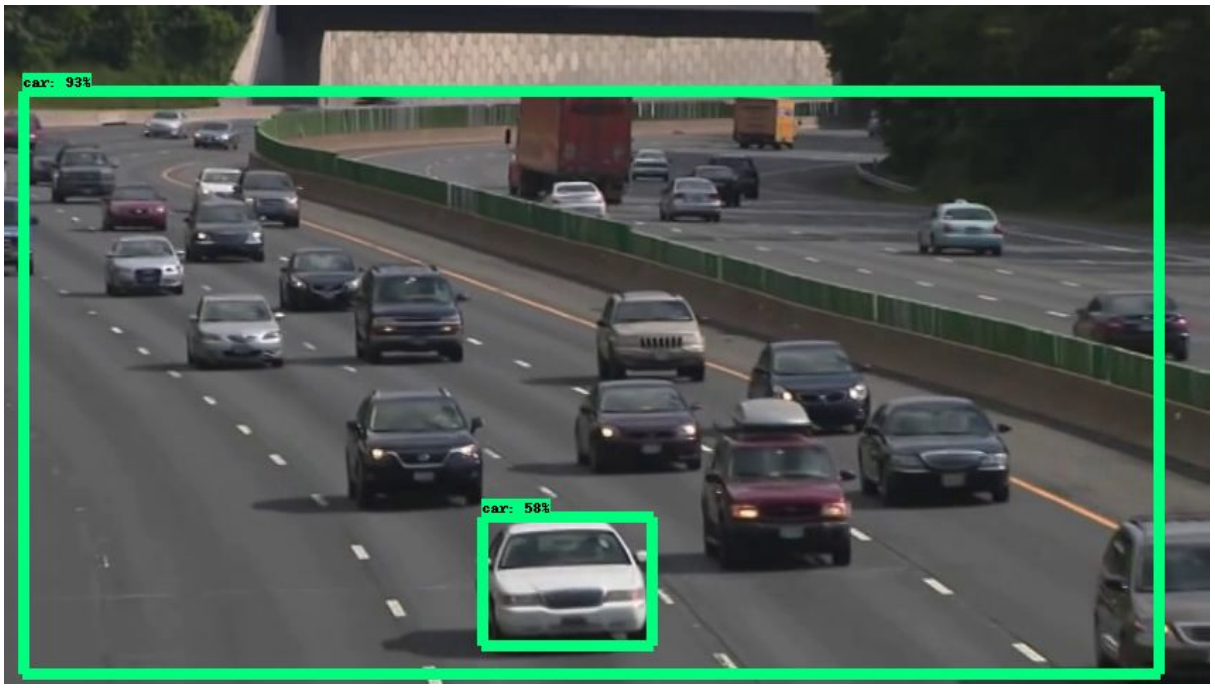

Figure 15 : 1er exemple de l'algorithme sur une vidéo avec notre propre modèle

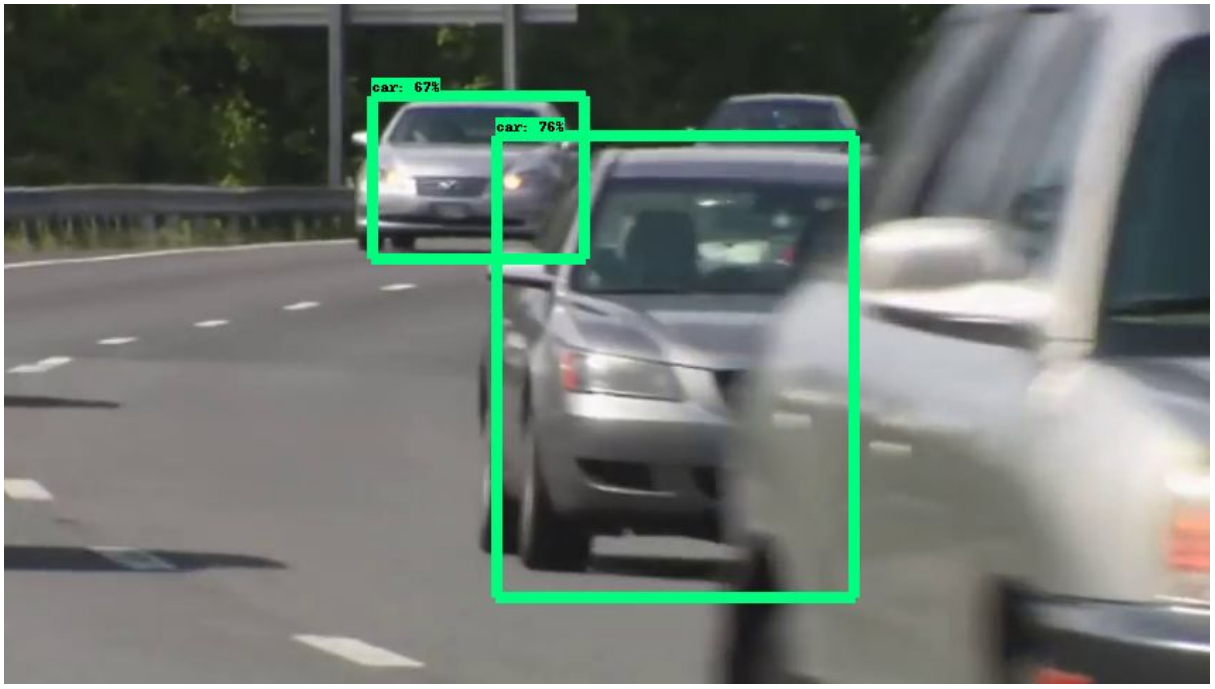

Figure 16 : 2ème exemple de l'algorithme sur une vidéo avec notre propre modèle

Rien que visuellement on peut voir que très peu de véhicules sont détectés et avec pour chacun un pourcentage faible.
Les résultats ne sont pas très bons. Le nombre d'images étant très petit, 20 images au total. Le modèle n'utilise pas non plus beaucoup de classes différentes à savoir 5 (voitures, piétons, motos, camions et bus). Il n'était aussi peut-être pas suffisamment

entraîné. Ainsi pour la suite des tests, nous avons opté pour le modèle prédéfini par COCO, “ssd_inception_v2_coco_2018_01_28”. Ce modèle comporte 90 classes.

3.3 - Étude des différents tests réalisés

Au niveau des tests, deux principaux ont été effectués. Ces séries de tests, ont été réalisées sur 10 images. Ces images étant toutes à la même résolution et contenant au total 5 catégories d’obstacles, voiture, piéton, moto, camion et bus. Ces images sont présentes dans l’annexe. Bien sûr, plus le nombre d’images est important et plus les résultats de tests seront précis.

Les tests ont été effectués à l’aide du modèle COCO suivant, “ssd_inception_v2_coco_2018_01_28”. Nous avons automatisé les tests à l’aide d’algorithmes. Ces algorithmes prennent en entrée un tableau contenant chaque véhicule présent à l’instant t avec son pourcentage de détection.

Pour chacun des tests, nous avons fait varier le seuil de détection. Un véhicule ayant un seuil de détection bas va alors être plus prudent et détecter plus d’obstacles. Néanmoins le pourcentage de détection des obstacles se fera en fonction du seuil et ainsi il est possible que les obstacles ne soient pas toujours pertinents à détecter. De plus, la conduite du véhicule en sera affectée et celle-ci ne sera pas fluide.

Tandis que pour un seuil de détection plus élevé, le véhicule va alors détecter les obstacles essentiels et aura une conduite plus fluide. En conséquence, l’algorithme peut ne pas prendre en compte des obstacles qui sont tout de même non négligeables. Par exemple, on peut trouver une vidéo d’un accident avec un véhicule autonome Uber dans la section bibliographie. Un article détaillant une piste d’explication et lui aussi fourni.

1ère série de test

Nous avons tout d’abord calculé une moyenne de détection, soit le pourcentage de détection obtenu pour chaque classe de véhicule, divisé par le nombre de véhicules de même classe détecté.

	Voiture	Piétons	Moto	Camion	Bus
Moyenne de détection (Seuil de détection : 10%)	0.6333	0.7689	0.7569	0.6896	0.6312
Moyenne de détection (Seuil de détection : 20%)	0.6333	0.7689	0.7569	0.6896	0.6312

Moyenne de détection (Seuil de détection : 30%)	0.6333	0.7689	0.7569	0.6896	0.6312
Moyenne de détection (Seuil de détection : 40%)	0.6947	0.8025	0.7569	0.7503	0.9339
Moyenne de détection (Seuil de détection : 50%)	0.7641	0.8329	0.7569	0.8201	0.9339
Moyenne de détection (Seuil de détection : 60%)	0.8374	0.8586	0.7569	0.8201	0.9339
Moyenne de détection (Seuil de détection : 70%)	0.8790	0.8586	0.9117	0.9433	0.9339
Moyenne de détection (Seuil de détection : 80%)	0.9014	0.9183	0.9117	0.9433	0.9339
Moyenne de détection (Seuil de détection : 90%)	0.9117	0.9607	0.9288	0.9671	0.9339

2ème série de test

Pour le deuxième test, nous avons au préalable défini à la main le nombre de véhicules de même classe présent sur l'image pour ensuite calculer un pourcentage de véhicule détecté.

	Pourcentage de véhicule détecté
Seuil de détection : 10%	0.9473
Seuil de détection : 20%	0.9473
Seuil de détection : 30%	0.9473
Seuil de détection : 40%	0.8955
Seuil de détection : 50%	0.8578
Seuil de détection : 60%	0.7375
Seuil de détection : 70%	0.5900
Seuil de détection : 80%	0.5190
Seuil de détection : 90%	0.3789

Ce qui nous donne le graphe suivant

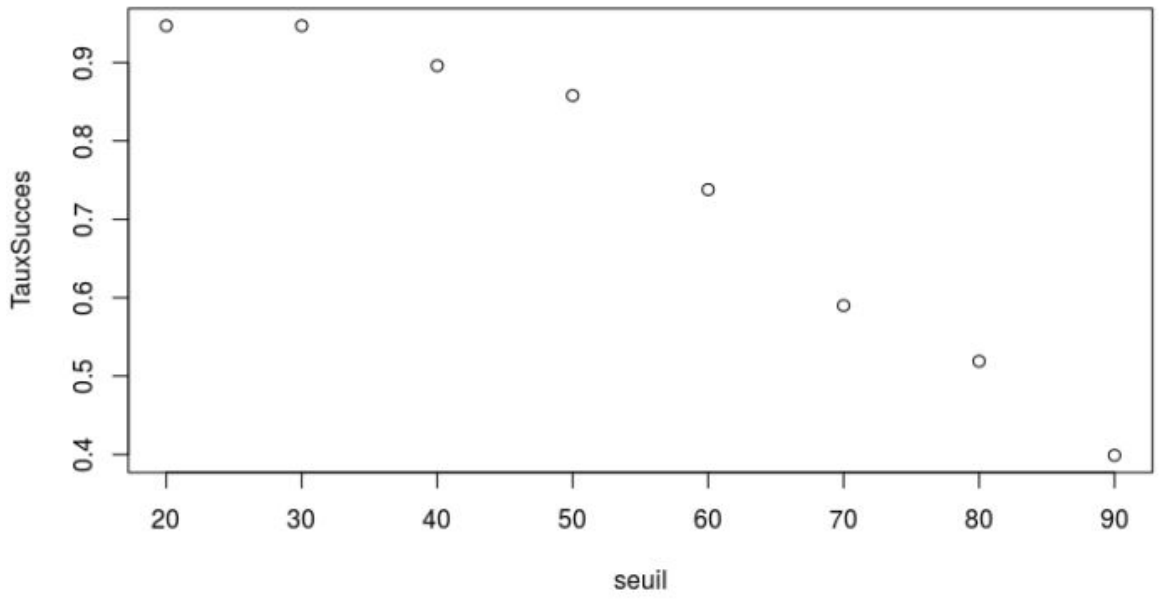

Figure 17 : Graphique illustrant le pourcentage de véhicule détecté en fonction du seuil de détection

Ainsi avec ce graphique, on peut apercevoir une baisse du pourcentage de véhicule détecté suite à l'augmentation du seuil de détection. Ce sont des résultats cohérents car certains véhicules n'ont pas un pourcentage de détection très élevé. Par rapport à la photo, ceux-ci peuvent être loin ou encore difficilement visibles.

Tout de même, le pourcentage de véhicule détecté baisse considérablement au fur et à mesure que l'on augmente le seuil. Les véhicules autonomes se doivent tout de même d'avoir un seuil de détection assez élevé, afin de prendre en compte les obstacles essentiels.

C'est pourquoi, on peut considérer le modèle utilisé comme insuffisant au niveau des performances obtenues. En plus des points évoqués précédemment, à savoir l'entraînement du modèle et les différentes classes qui le compose. On peut aussi prendre en compte par la suite l'aspect matériel dans la détection. Ainsi en plus de caméras qui donneraient le flux vidéo à l'algorithme, on pourrait par exemple imaginer la présence de différents capteurs sur le véhicule. Ces différents points amélioreront grandement les résultats obtenus.

Conclusion

L'objectif de notre projet était de se lancer dans la recherche de l'intelligence artificielle pour les voitures autonomes avec le deep learning. Puis mettre en place un outil permettant de détecter les véhicules présents sur la route.

Dans un premier temps nous avons réalisé un travail de recherche afin de nous familiariser avec le projet.
Ensuite nous avons recherché les outils de détection d'obstacles existant sur le marché, nous en avons découvert différents avec pour chacun, leurs points forts et leurs points faibles. Suite à cette recherche, nous avons choisi de nous pencher vers TensorFlow.

Nous avons donc mis en place l'API TensorFlow contenant un algorithme tutoriel, que nous avons modifié afin qu'il puisse détecter les obstacles à partir d'une vidéo.
Il a aussi été question de voir les différentes étapes de la mise en place de notre propre modèle et d'en voir les limites.

Nous avons ensuite réalisé nos tests sur des ensembles d'images, de là nous avons pu voir la fiabilité de notre outil avec différents seuils de détection.

Nous avons donc été au bout de notre projet en mettant en place un outil de détection d'obstacle qui a des taux de détection jusqu'à 95%.
Dans le futur, nous pourrions utiliser des ensembles de données plus grands et plus précis afin d'avoir des résultats encore plus satisfaisants, et mettre en place un système permettant de trouver l'obstacle le plus dangereux pour la voiture.

Table des figures

Bibliographie

Méthode de Viola et Jones

https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Viola_et_Jones

Technologies de Mobileye

https://www.mobileye.com/our-technology/

Niveaux d'autonomie du véhicule autonome

https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Site de TensorFlow

https://www.tensorflow.org/

Site de PyTorch

https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Site de OpenCV

https://opencv.org/

Libraire COCO (PythonAPI et Librairie d'images)

https://github.com/cocodataset/cocoapi

Site de COCO

http://cocodataset.org/#home

Librairie ImageNet

http://image-net.org/

Modèles pré entraînés de TensorFlow

https://github.com/tensorflow/models/tree/master/research/object_detection

Accident avec un véhicule autonome Uber

https://www.youtube.com/watch?v=Vqi9mLumWnA

Article détaillant une piste d'explication

https://actu-moteurs.com/2019/11/12/auto/actualite/142_a/accident-mortel-avec-une-volvo-autonome-uber-les-resultats-de-lenquete-sont-connus/

Article évoquant l'abaissement du seuil de détection

https://www.lesechos.fr/industrie-services/tourisme-transport/voiture-autonome-une-procureure-blanchit-uber-pour-laccident-mortel-de-2018-996191

Annexe

Lien vers le GitHub

https://github.com/simonbaas-gif/PJI-BAAS-MEINAS

Installation des librairies

Nous avons dû dans un premier temps, réaliser l'installation des différentes librairies nécessaires.
Premièrement **Protobuf,** cette librairie est un mécanisme extensible indépendant de Google pour la stérilisation des données structurées.
Installation :

```
sudo apt-get install protobuf-compiler
```

```
python-pil python-lxml python-tk
```

Ensuite nous avons eu besoin de **Cython.** Elle nous permet une puissance de Python et C combiné afin d'écrire nos applications en Python en ayant la puissance du langage C ou C++.
Installation :

```
pip install --user Cython
```

contextlib2 est une librairie fournissant des commandes afin de simplifier des tâches courantes. Par exemple la commande "with" qui sera utilisée dans le script principal.
Installation :

```
pip install --user contextlib2
```

Jupyter Notebook est un environnement de développement interactif basé sur le web. Cela peut s'avérer assez intéressant car nous pouvons configurer et organiser l'interface utilisateur pour prendre en charge un large éventaille en apprentissage automatique.
Installation :

```
pip install --user jupyter
```

Enfin **Matplotlib** qui est une librairie permettant de créer des visualisations statiques, animées et interactives en python. Cela nous sera très utile pour encadrer les objets identifiés sur une image.

Installation :

```
pip install --user matplotlib
```

Nous avons ensuite besoin Python-tk. C'est une librairie intégrée à python qui fournit un jeu d'outils robustes et indépendants de la plateforme pour gérer des fenêtres.

Pour l'installation de TensorFlow :
Pour le CPU

```
pip install tensorflow
```

Pour le GPU

```
pip install tensorflow-gpu
```

Dans le projet fourni par Google utilisant l'API, on retrouve un fichier de tutoriel permettant de détecter des objets sur deux images.

Télécharger le répertoire de modèles TensorFlow :

```
git clone https://github.com/tensorflow/models.git
```

Le projet se base donc sur un modèle prédéfini. Ces modèles sont téléchargeables grâce à l'API de COCO. Ainsi il est possible de changer facilement de modèle parmi ceux proposés mais il est aussi possible d'améliorer un modèle.

Installer l'API COCO pour Python :

```
git clone https://github.com/cocodataset/cocoapi.git
cd cocoapi/PythonAPI
make
cp -r pycocotools <path_to_tensorflow>/models/research/
```

Puis il faut utiliser Protobuf afin de configurer le modèle, et donc compiler les fichiers .proto.

```
# From tensorflow/models/research/
protoc object_detection/protos/*.proto --python_out=.
```

À la fin, il suffit d'ajouter les librairies à sa variable d'environnement PYTHONPATH.

```
# From tensorflow/models/research/
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

Les différents modèles COCO

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Les différentes images de test