2020

Projet TAIV MIV 2020

DETECTION ET TRACKING DES OBJETS EN MOUVEMENT DANS UNE SCENE

NAIT KACI MOHAMED AGHILES & AMINE GUENAOUI

|

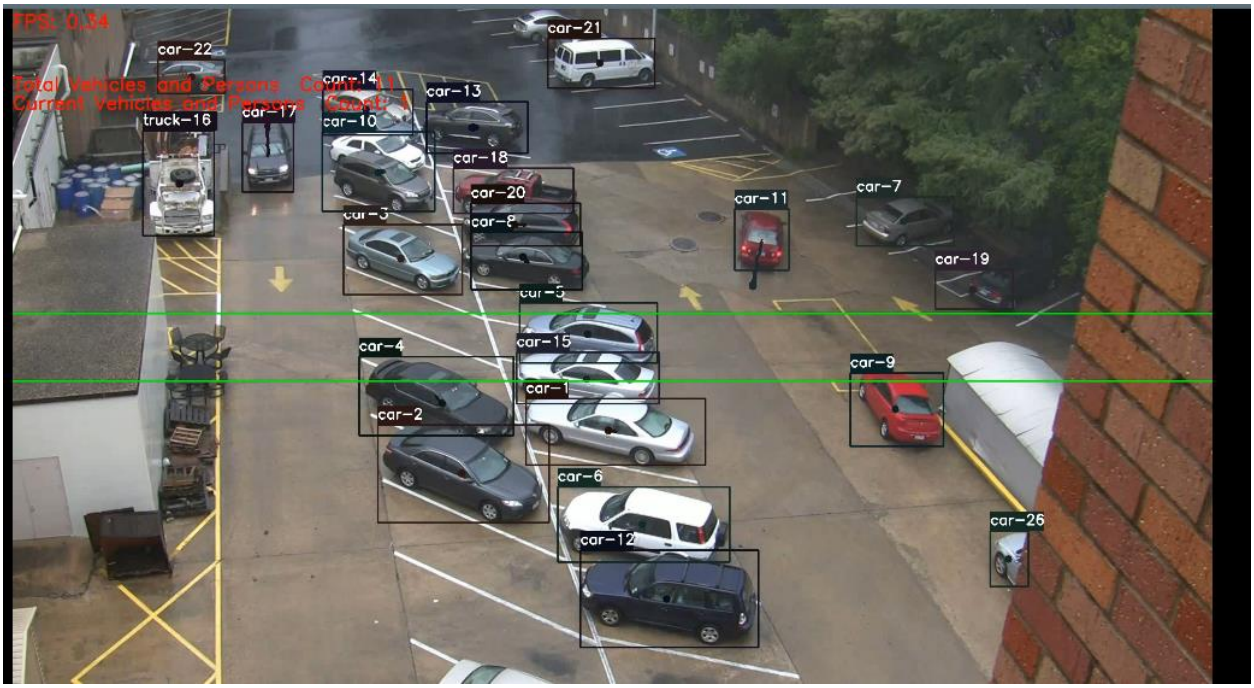
Introduction

Ce projet a était fait dans le but de détecter l'objet qui sont en mouvement, et leurs identifier, reconnaissance et suivi, (OBJECT DETECTION AND TRACKING)

Dans ce projet nous avons reçu à détecter les objets en mouvement, et appliquer la reconnaissance, aussi tracer la trajectoire de chaque objet et la afficher sur la vidéo donnée.

La sortie représente la vidéo avec les donnes de mouvement et le nom de chaque objet détecte.

Voici un exemple de la vidéo donnée, et le résultat du traitement.



Explication d'affichage

Dans la fenêtre chaque objet est englobe par un carre qui indique son nom, aussi son ID (qu'on appelle track_ID)

La trajectoire dessinée dans le centre de chaque objet indique sa trajectoire,

La ligne verte va compter combien d'objet son passée par la zone qui est affichée en haut à gauche,

Le nombre d'objets détectes et aussi afficher sur la fenêtre en haut à gauche.

Le FPS indique la vitesse de calcule.

Comment ça marche

Avant aller au code source, voici comment il contient quoi.

YOLO

Dans ce code nous avons utilisé YOLO, qui permet la détection des objets et leurs reconnaitre en même temps, il applique un réseau de neurone qui permet de prédire les boîtes englobantes et la classification multi-étiquettes.

Deep SORT

Nous avons aussi utilisé Dee SORT tell que :

- Sort applique le filtre de Kalmar et la méthode de HungArian pour gérer la prédiction de mouvement et l'association de données.
- Les modèles d'objet contiennent la position de l'objet, l'échelle, le rapport de la boîte englobante et la prédiction de mouvement pour l'image suivante.
- SORT résout l'association de données en calculant l'intersection de la boîte englobante sur la distance d'union (Intersection Over Union)
- Sur la base de la distance IOU, l'affectation finale est effectuée.
- est rapide et simple et ont une précision et une précision élevées.
- ne peut pas gérer très bien l'occlusion car SORT repose sur un modèle de mouvement simple.

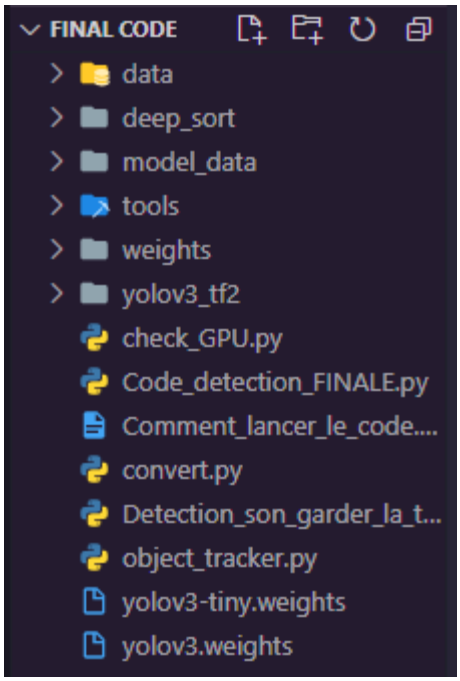
Comment Yolo et DeepSort marchent

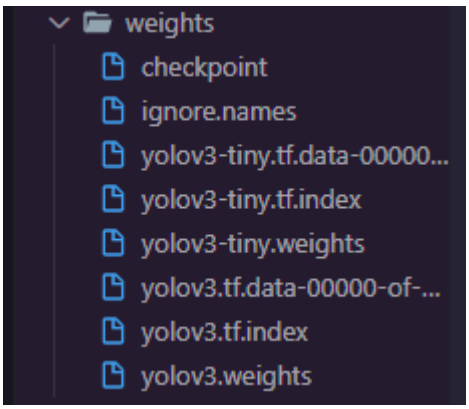
Ça marche en trois étapes :

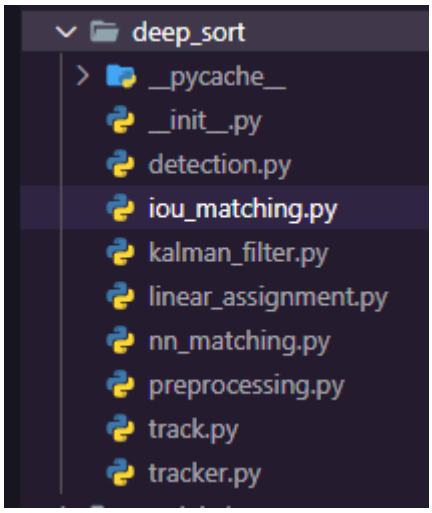
1. Détection et reconnaissance d'objets.
    a. Yolo ou RNC.
2. Prédiction de mouvement et génération de fonctionnalités.
    a. Un modèle d'estimation est une phase intermédiaire avant l'association des données, qui utilise l'état de chaque piste .
    b. Le filtre de Kalman est utilisé pour modéliser ces états et effectuer une prédiction de mouvement.
    c. La génération d'entités ou les descripteurs de boîte englobante sont calculés à l'aide d'un RNC préformé.
3. Tracking (suivie)
    a. Compte tenu des états prédits du filtrage de Kalman :
        i. dans l'image précédente.
        ii. boîte nouvellement détectée dans la trame actuelle.
    b. une association est faite pour la nouvelle détection dans l'image courante avec les anciennes pistes d'objets dans l'image précédente.
    c. La distance d'entité cosinus, la distance IOU et la distance d'état de Kalman sont calculées pour la mise à jour correspondante.

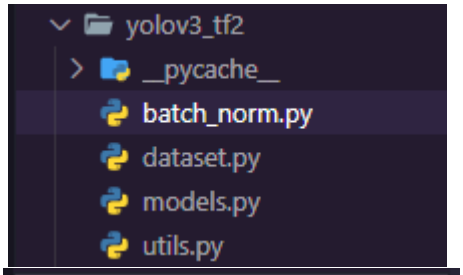
Code Source

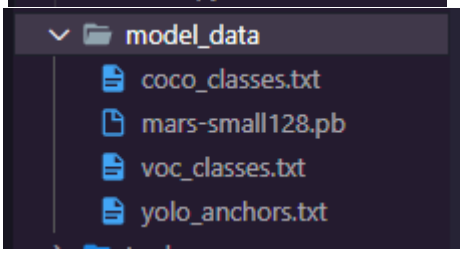
Répertoire du code source :


FINAL CODE
data
deep_sort
model_data
tools
weights
yolov3_tf2
check_GPU.py
Code_detection_FINALE.py
Comment_lancer_le_code....
convert.py
Detection_son_garder_la_t...
object_tracker.py
yolov3-tiny.weights
yolov3.weights


weights
checkpoint
ignore.names
yolov3-tiny.tf.data-00000...
yolov3-tiny.tf.index
yolov3-tiny.weights
yolov3.tf.data-00000-of-...
yolov3.tf.index
yolov3.weights


deep_sort
__pycache__
__init__.py
detection.py
iou_matching.py
kalman_filter.py
linear_assignment.py
nn_matching.py
preprocessing.py
track.py
tracker.py


yolov3_tf2
__pycache__
batch_norm.py
dataset.py
models.py
utils.py


model_data
coco_classes.txt
mars-small128.pb
voc_classes.txt
yolo_anchors.txt

Code_detection_FINALE.py

```python
from absl import flags
import sys

FLAGS = flags.FLAGS
FLAGS(sys.argv)

import time
import numpy as np
import cv2
import matplotlib.pyplot as plt

import tensorflow as tf
from yolov3_tf2.models import YoloV3
from yolov3_tf2.dataset import transform_images
from yolov3_tf2.utils import convert_boxes

from deep_sort import preprocessing
from deep_sort import nn_matching #for the association matrix
from deep_sort.detection import Detection
from deep_sort.tracker import Tracker
from tools import generate_detections as gdet #to help detect objects

#Loading wieghts
class_names = [c.strip() for c in open('data/labels/coco.names').readlines()]
yolo = YoloV3(classes=len(class_names))
yolo.load_weights('weights/yolov3.tf')

max_cosine_distance = 0.5 # features similarity
nn_budget = None #to create the libraries
nms_max_overlap = 0.8 #to avoid many detections on the same object

#application using nearest neighbor as metric for the tracker of features
model_filename = 'model_data/mars-small128.pb'
encoder = gdet.create_box_encoder(model_filename, batch_size=1)
metric = nn_matching.NearestNeighborDistanceMetric('cosine',max_cosine_distance,nn_budget)
tracker = Tracker(metric)

#Loading the video

vid = cv2.VideoCapture('./data/video/test2.mp4')

codec = cv2.VideoWriter_fourcc(*'XVID')
vid_fps = int(vid.get(cv2.CAP_PROP_FPS))
vid_width,vid_height = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)),int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
out = cv2.VideoWriter('./data/video/result.avi',codec,vid_fps,(vid_width,vid_height))

#trajectoire
from _collections import deque
pts = [deque(maxlen=30) for _ in range(1000)]
```

```python
pts = [deque(maxlen=30) for _ in range(1000)]

#counter
counter = []


while True:
    _,img = vid.read()
    if img is None:
        print('Completed')
        break
    #pretraitement de l'image
    img_in = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img_in = tf.expand_dims(img_in,0) #adding more dimension to the original image
    img_in = transform_images(img_in,416)


    t1 = time.time()
    #prediction using YOLO
    boxes,scores,classes,nums = yolo.predict(img_in)

    #boxes has 3D shape (1, 100, 4)
    #scores has 2D shape  (1, 100)
    # classes has  2D shape( 1 ,100)
    #nums has 1D shape (1,)
    #getting the classes
    classes = classes[0]
    names = []
    for i in range(len(classes)):
        names.append(class_names[int(classes[i])])
    #getting the names and boxes and extracting features
    names = np.array(names)
    converted_boxes = convert_boxes(img, boxes[0])
    features = encoder(img, converted_boxes)

    detections = [Detection(bbox, score, class_name, feature) for bbox, score, class_name, feature in zip(converted_boxes, scores[0], names, features)]
    boxs = np.array([d.tlwh for d in detections])
    scores = np.array([d.confidence for d in detections])
    calsses = np.array([d.class_name for d in detections])
    indices = preprocessing.non_max_suppression(boxs,classes,nms_max_overlap,scores)
    detections = [detections[i] for i in indices]
    #predictions and then update  using KALMAN
    tracker.predict()
    tracker.update(detections)

    cmap = plt.get_cmap('tab20b')
```

```
    tracker.update(detections)

    cmap = plt.get_cmap('tab20b')
    colors = [cmap(i)[:3] for i in np.linspace(0,1,20)]

    #objects that are passing by
    current_count = int(0)
    #for each tracked object draw the box and it's path
    for track in tracker.tracks:
        if not track.is_confirmed() or track.time_since_update > 1:
            continue
        bbox = track.to_tlbr()
        class_name = track.get_class()
        color = colors[int(track.track_id) % len(colors) ]
        color = [i * 55 for i in color]

        cv2.rectangle(img,(int(bbox[0]),int(bbox[1])),(int(bbox[2]),int(bbox[3])), color, 2)
        cv2.rectangle(img,(int(bbox[0]),int(bbox[1]-30)),(int(bbox[0])+(len(class_name)+len(str(track.track_id)))*17,int(bbox[1])), color, -1)
        cv2.putText(img,class_name+"-"+str(track.track_id),(int(bbox[0]),int(bbox[1]-10)),0,0.75,(255,255,255),2)

        #hisotry part / trajectoire
        center = (int(((bbox[0])+(bbox[2]))/2), int(((bbox[1])+(bbox[3]))/2) )
        pts[track.track_id].append(center)
        #drawing the PATH

        for j in range(1 , len(pts[track.track_id])):
            if pts[track.track_id][j-1] is None or pts[track.track_id][j] is None:
                continue
            thickness = int(np.sqrt(64/float(j+1))*2)
            cv2.line(img, (pts[track.track_id][j-1]), (pts[track.track_id][j]), color , thickness)

        #drawing a line that tells how many objects have passed by zone or line
        height, width , _ = img.shape
        cv2.line(img, (0,int(3*height/6+height/20)), (width,int(3*height/6+height/20)), (0,255,0),thickness=2)
        cv2.line(img, (0,int(3*height/6-height/20)), (width,int(3*height/6-height/20)), (0,255,0),thickness=2)

        #for each box check if it's center is passing by the line

        center_y = int( ( (bbox[1])+(bbox[3]) ) / 2)

        if center_y <= int(3*height/6+height/30) and center_y >= int(3*height/6-height/30):
            if class_name == 'car' or class_name == 'truck' or class_name == "person":

                counter.append(int(track.track_id))
                current_count +=1

    #related to the objects that passed by a line
```

```
                current_count +=1

     #related to the objects that passed by a line
     total_count = len(str(counter))
     cv2.putText(img , "Current Vehicles and Persons  Count: " + str(current_count), (0,160), 0, 1 , (0,255,0), 2)
     cv2.putText(img , "Total Vehicles and Persons  Count: " + str(total_count), (0,130), 0, 1 , (0,255,0), 2)
     #passed
     #counting fps
     fps = 1./(time.time()-t1)
     cv2.putText(img,"FPS: {:.2f}".format(fps), (0,30),0,1,(0,255,0),2)
     #showing the frame and writing it on the file
     cv2.imshow('output',img)
     out.write(img)

     if cv2.waitKey(1) == ord('q'):
         break

vid.release()
out.release()
cv2.destroyAllWindows()
```

FIN