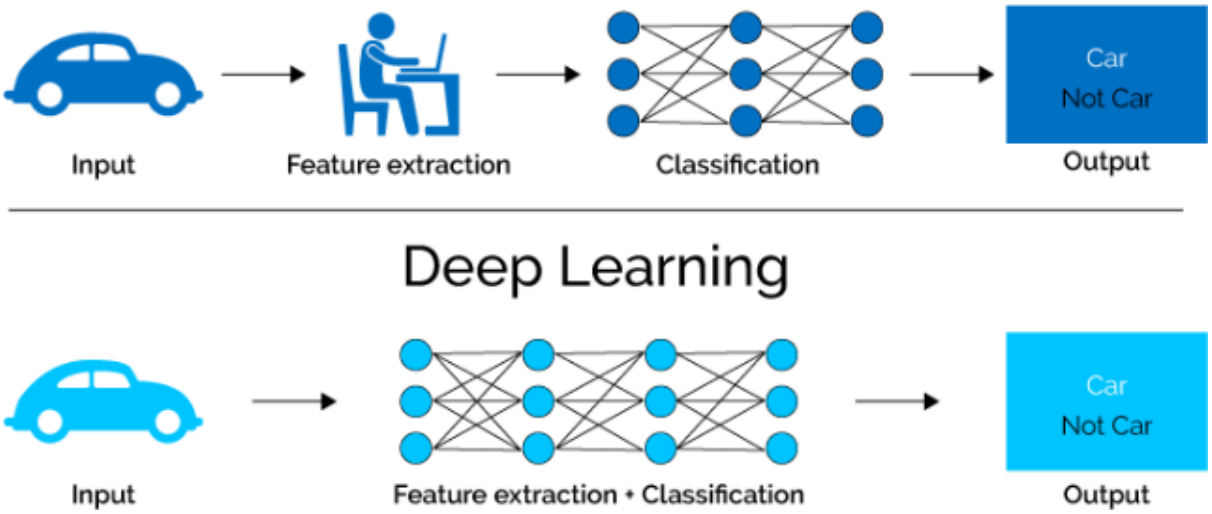
VISION

-

CLASSIFICATION D'IMAGES PAR DEEP LEARNING



Florent Muret

Jean-Baptiste Rambaud

Introduction

Dans ce TP nous mettre en œuvre une autre technique de détection d'image, à partir de darknet qui recherche des images en fonction de classes (objets référencés), ce qui nous rappel les algorithmes de haar cascades.

Nous allons donc mettre en place cette technique, l'appliquer sur des photos, des vidéos et sur un flux webcam.

Prise en main locale de darknet

Questions :

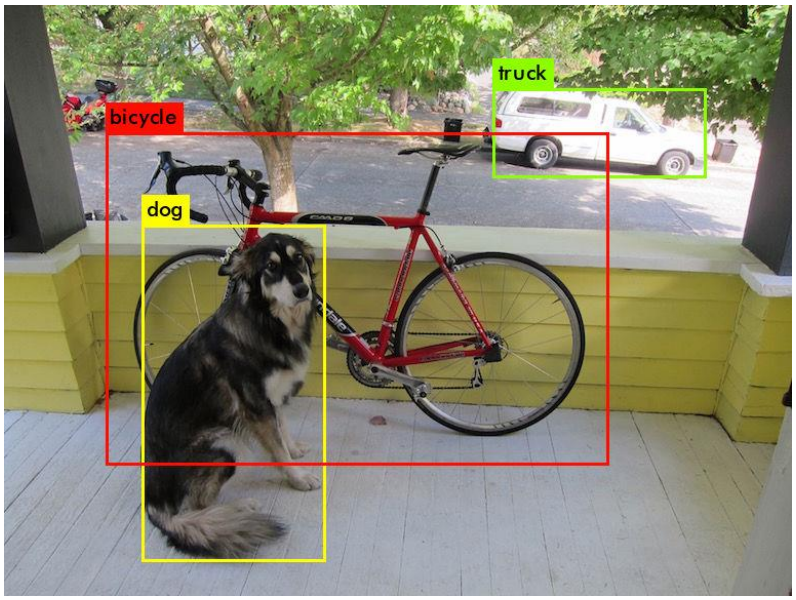
- *Tester les différents modèles pré entraînés proposés aux téléchargements*
- *Trouver des images pertinentes pour ces différentes classes.*
- *Comparer les performances*
- *Si possible trouver aussi une vidéo pertinente pour une partie de ces classes.*

On commence donc par suivre le lien suivant https://pjreddie.com/darknet/yolo/ , et on effectue les différentes commandes afin de récupérer le répertoire ainsi que le « pre-trainnig weight ». On lance ensuite le détecteur comme demandé avec le premier détecteur « yolov3 ».

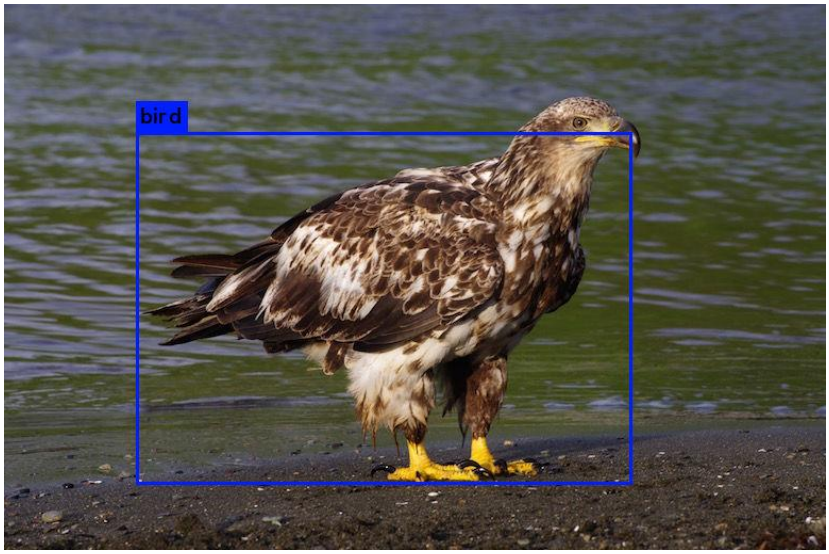
```
  89 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
  90 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
  91 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
  92 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
  93 conv    255  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 255  0.377 BFLOPs
  94 yolo
  95 route  91
  96 conv    128  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 128  0.095 BFLOPs
  97 upsample            2x    38 x  38 x 128   ->    76 x  76 x 128
  98 route  97 36
  99 conv    128  1 x 1 / 1    76 x  76 x 384   ->    76 x  76 x 128  0.568 BFLOPs
 100 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
 101 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
 102 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
 103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
 104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
 105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255  0.754 BFLOPs
 106 yolo
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 61.122622 seconds.
dog: 100%
truck: 92%
bicycle: 99%
florent.muret@tpb08:~/OpenCV_Muret_JB/TP 3 /darknet$
```

Détection effectuée avec « yolov3 » - terminal :

On peut remarquer que le terminal affiche le temps de calcul ainsi que le seuil de « confiance » avec lequel il détecte un objet.



Détection effectuée avec « yolov3 » :



Détection effectuée avec « yolov3 » sur une autre image :

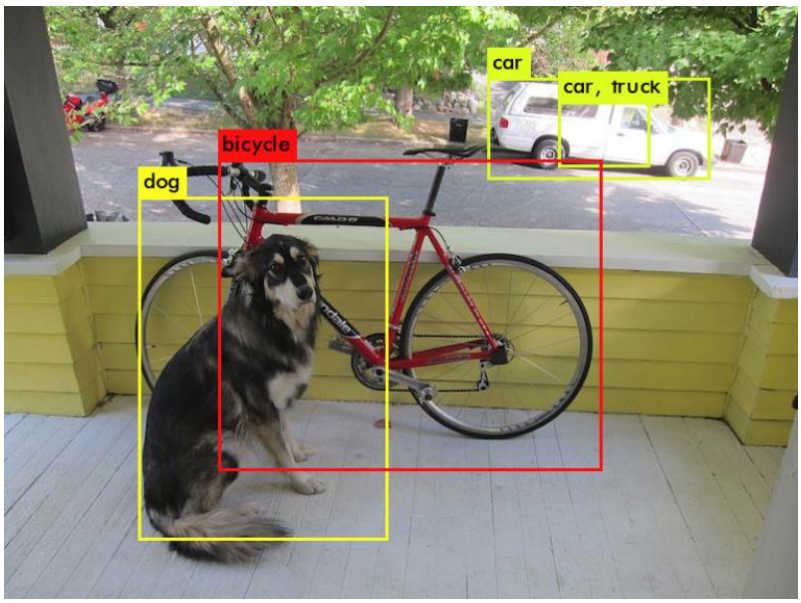
On peut en effet modifier ce seuil à l'aide de l'option « -thresh X » avec X la valeur que l'on souhaite donner au seuil afin d'affiner ou bien d'augmenter la tolérance de notre détecteur.

On télécharge par la suite la version Tiny de yolov3, qui nous sera utile par la suite car plus rapide à exécuter. Testons ce modèle :

```
florent.muret@tpb08:~/OpenCV_Muret_JB/TP 3 /darknet$ ./darknet detect cfg/yolov3-tiny.cf
g yolov3-tiny.weights data/dog.jpg
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16  0.150 BFLOPs
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32  0.399 BFLOPs
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64  0.399 BFLOPs
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128  0.399 BFLOPs
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256  0.399 BFLOPs
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
   13 conv    256  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 256  0.089 BFLOPs
   14 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   15 conv    255  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 255  0.044 BFLOPs
   16 yolo
   17 route  13
   18 conv    128  1 x 1 / 1    13 x  13 x 256   ->    13 x  13 x 128  0.011 BFLOPs
   19 upsample            2x    13 x  13 x 128   ->    26 x  26 x 128
   20 route  19 8
   21 conv    256  3 x 3 / 1    26 x  26 x 384   ->    26 x  26 x 256  1.196 BFLOPs
   22 conv    255  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x 255  0.088 BFLOPs
   23 yolo
Loading weights from yolov3-tiny.weights...Done!
data/dog.jpg: Predicted in 1.863593 seconds.
dog: 57%
car: 52%
truck: 56%
car: 62%
bicycle: 59%
```

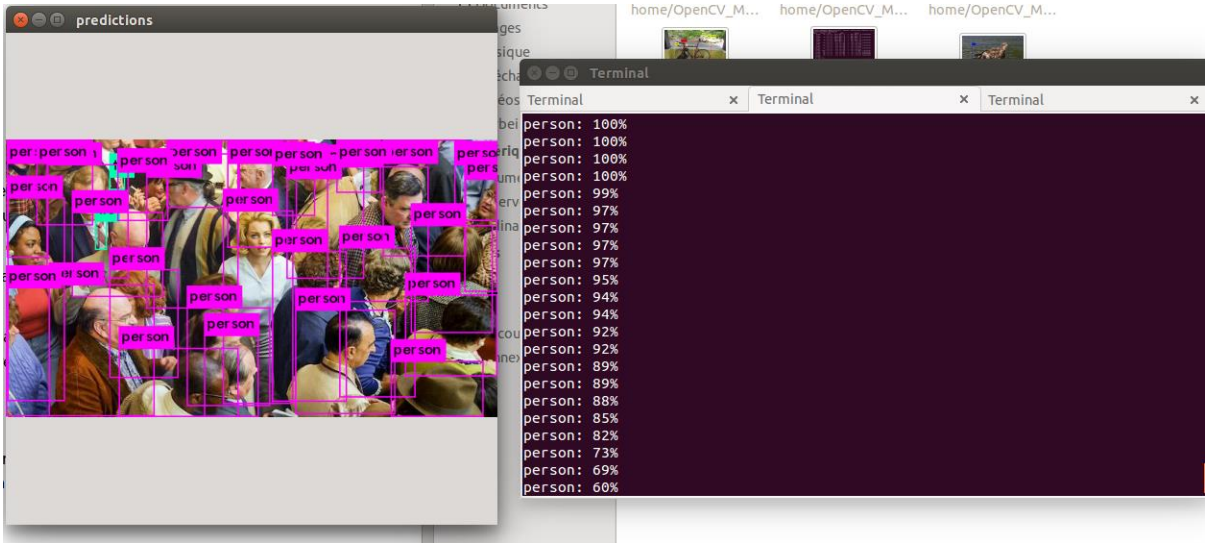
Détection effectuée avec « yolov3 -Tiny » - terminal :

On peut remarquer que le seuil de confiance est nettement inférieur à la version précédente (yolov3), cependant le temps de traitement est également bien inférieur : 1.8 secondes contre 61 secondes.

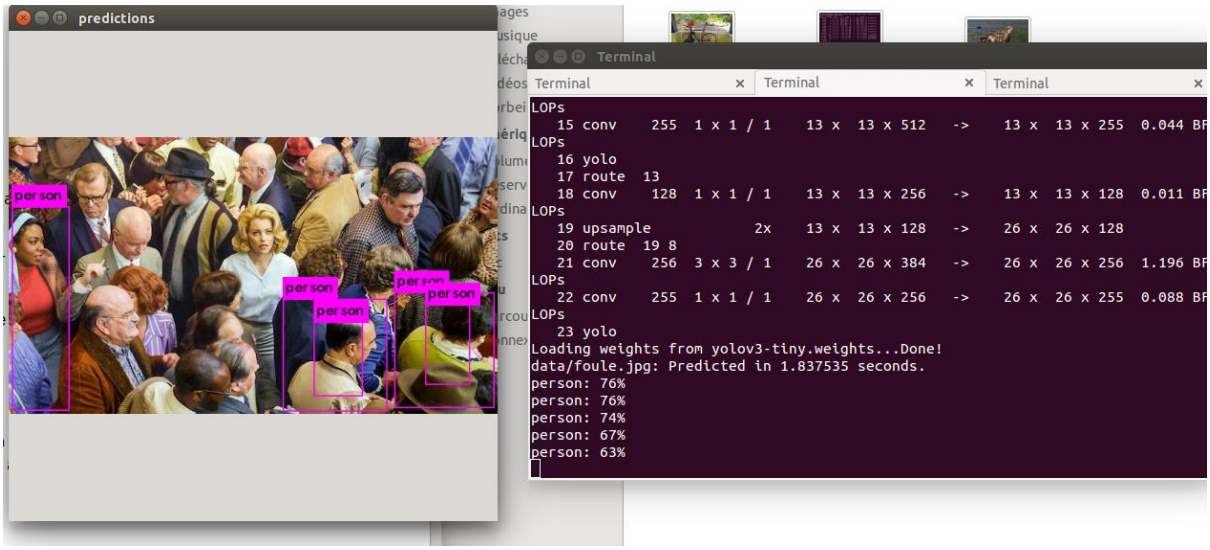


Détection effectuée avec « yolov3 - tiny » :

Essayons de trouver une meilleure photo pour effectuer une comparaison entre ces deux modèles. On va choisir ici une photo avec énormément de choses à détecter : prenons une photo d'une foule :



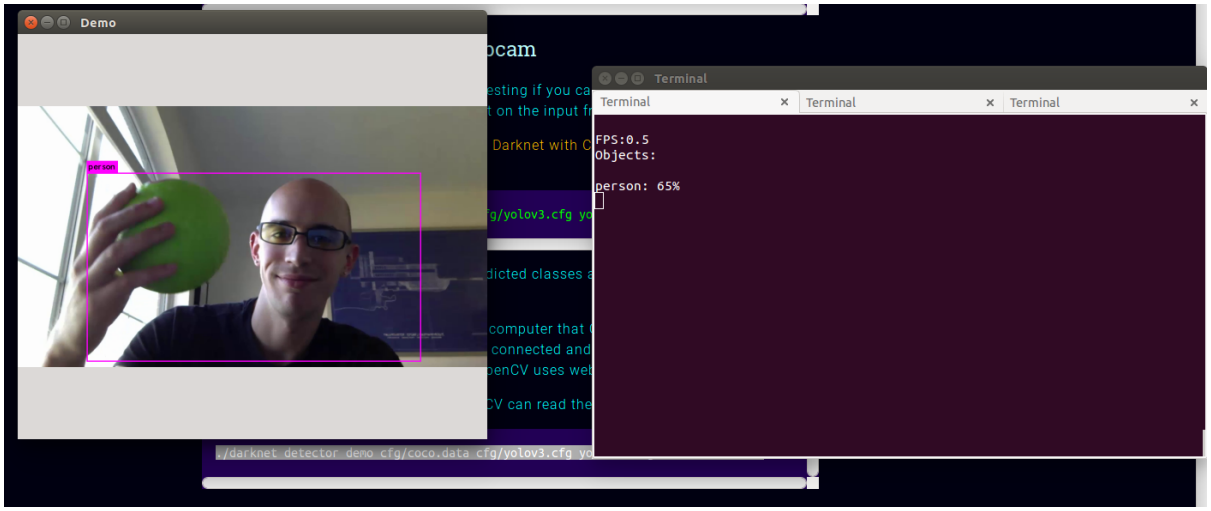
Détection effectuée avec « yolov3 » :



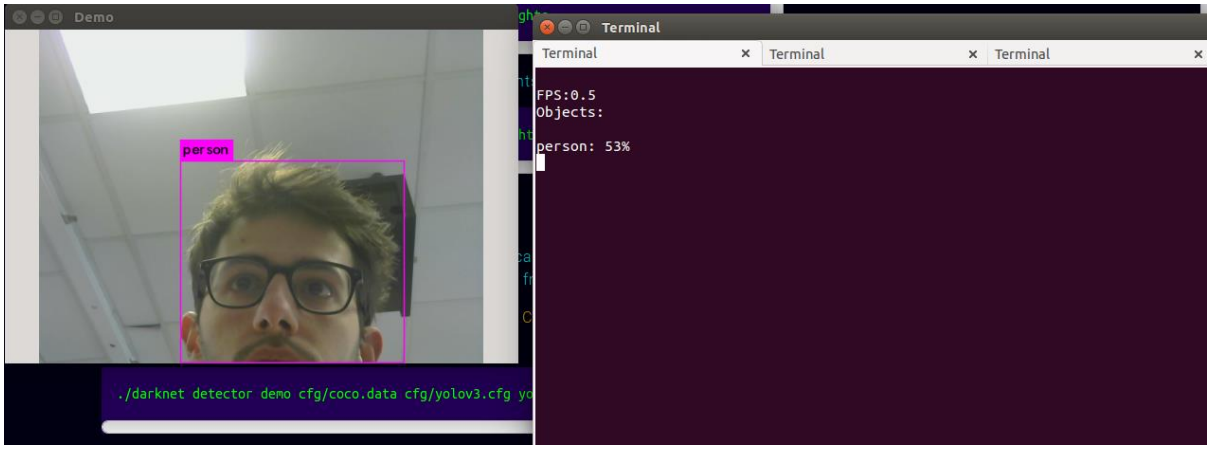
Détection effectuée avec « yolov3 - tiny » :

Le contraste entre les deux est ici frappant : en effet, sans toucher au « treshold », le premier modèle détecte quasiment tous les visages, avec des certitudes très élevées alors que le second en reconnait à peine quelques un et avec une certitude maximale de 76%. La version Tiny est donc beaucoup moins performante, mais est cependant beaucoup plus rapide.

On effectue ensuite la lecture frame par frame d'un flux vidéo, tout d'abord d'une vidéo en locale, puis que l'on récupère via une webcam.



Détection effectuée avec « yolov3 - Tiny » sur une vidéo du TP 1 :



Détection effectuée avec « yolov3 - tiny » sur le flux webcam :

On utilise la version Tiny ici sinon nous ne pourrions pas lire la vidéo correctement dû à la configuration du pc. En effet, on peut voir ici que même avec la version Tiny qui accélère énormément le temps de traitement, le nombre de FPS est de 0.5 (ce qui signifie le nombre d'image par seconde, pour être fluide il faudrait avoir 30fps constant minimum).

1) En locale

Questions :

-Le but est de post-traiter le retour du détecteur pour créer l'équivalent du retour des boxes englobantes

-Rajouter ensuite une option pour n'afficher qu'une liste de classe configurée en début de code, prévoir aussi des couleurs différentes pour l'affichage des noms et des boîtes.

-Modifier le code pour travailler sur un flux webcam

-Bonus Sauvegarder le contenu des bounding box (portions d'images correspondant à des classes sous forme d'image jpg ssur Yolo :auvegardées dans un repertoire. Le nom de l'image doit contenir la classe et un « timestamp »

-Tester les codes server_fj et client disponible sur le campus.

-Creer un code basé sur client et votre code precedent pour traiter votre flux udp dans le détecteur darknet

1) Post traitement et sélection de paramètres

On souhaite maintenant effectuer le traitement de la détection par nous-même. On récupère le fichier python « detector_opencv_fj.py » et on l'exécute. On remarque que celui-ci renvoie dans le terminal une liste contenant les arguments de la détection, sous la forme : [(objet 1, pourcentage, coordonnées) (objet 2 …)].

On s'intéresse à la troisième détection, car c'est celle effectué à l'aide de Opencv.

```
Loading weights from /fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/d
arknet/cfg/yolov3.weights...Done!
 network loaded in GPU , Press Enter
test direct path for image :
[('dog', 0.999333381652832, (224.1796417236328, 378.48077392578125, 178.7485198
9746094, 328.2848815917969)), ('bicycle', 0.9916158318519592, (344.527984619140
6, 286.7609558105469, 486.1769714355469, 321.3677673339844)), ('truck', 0.91656
57162666321, (580.9124755859375, 125.0541000366211, 208.13795471191406, 87.2764
2822265625))]
test direct scipy read for image :
[('dog', 0.9987426996231079, (224.75650024414062, 376.8475341796875, 179.03170776367188,
 322.6524353027344)), ('bicycle', 0.9984409809112549, (343.85711669921875, 284.830017089
84375, 476.8623046875, 342.72528076171875)), ('truck', 0.952713668346405, (584.606323242
1875, 125.1603012084961, 213.17471313476562, 91.02485656738281))]
test direct opencv (cv2) read for image :
[('dog', 0.999333381652832, (224.1796417236328, 378.48077392578125, 178.74851989746094,
328.2848815917969)), ('bicycle', 0.9916158318519592, (344.5279846191406, 286.76095581054
69, 486.1769714355469, 321.3677673339844)), ('truck', 0.9165657162666321, (580.912475585
9375, 125.0541000366211, 208.13795471191406, 87.27642822265625))]
florent.muret@tpb08:~/OpenCV_Muret_JB/TP3/darknet$
```

Série de coordonnées récupéré via OpenCV :

On va donc récupérer ces paramètres, les traiter, et effectuer l'affichage des rectangles sur les classes repérés.

```python
arguments = sys.argv[1:]

liste = r
for index in liste :
  afficher = False
  structure = index
  position  = structure[2]
  nom = structure[0]
  pourcentage = structure[1]

  x = position[0]
  y = position[1]
  w = position[2]
  h = position[3]

  for k in range(len(arguments)):
    if nom == arguments[k] :
      afficher = True
      break


  if nom == 'dog' :
    color = (255,0,0)
  if nom == 'car' :
    color = (255,255,0)
  if nom == 'bicycle' :
    color = (0,255,0)
  if nom == 'truck' :
    color = (0,0,255)

  if afficher == True :
    cv2.putText(arr, nom, (int(x), int(h)), cv2.FONT_HERSHEY_SIMPLEX, 1.0, color)
    cv2.rectangle(arr,(int(x-(w/2)),int(y-(h/2))),(int(x+(w/2)),int(y+(h/2))),color,2)
    print('forme trouve = ' ,nom , pourcentage)

cv2.imshow(WINDOW_NAME, arr)
cv2.waitKey(0)
```

Dans le code ci-dessus, on peut voir que l'on récupère la liste de caractère renvoyé par OpenCv dans la valeur « liste ». Dans la boucle on effectue du cas par cas, afin d'identifier un objet et de le traiter (affichage ou non via les valeurs passées dans le terminal). Si un type passé en paramètre est identifié, alors on affiche son nom et on entoure son contour avec une couleur différente en fonction de son type, puis on affiche l'image.

Essayons maintenant ce code avec la photo du chien précédente. On rentre cette fois ci les arguments que l'on souhaite rechercher :

```
KeyboardInterrupt
florent.muret@tpb08:~/OpenCV_Muret_JB/TP3/darknet$ python python/detector_opencv_fj.py dog bicycle
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16  0.150 BFLOPs
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32  0.399 BFLOPs
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64  0.399 BFLOPs
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128  0.399 BFLOPs
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256  0.399 BFLOPs
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
```

Terminal avec exécution du programme selon les paramètres dog et bicycle :

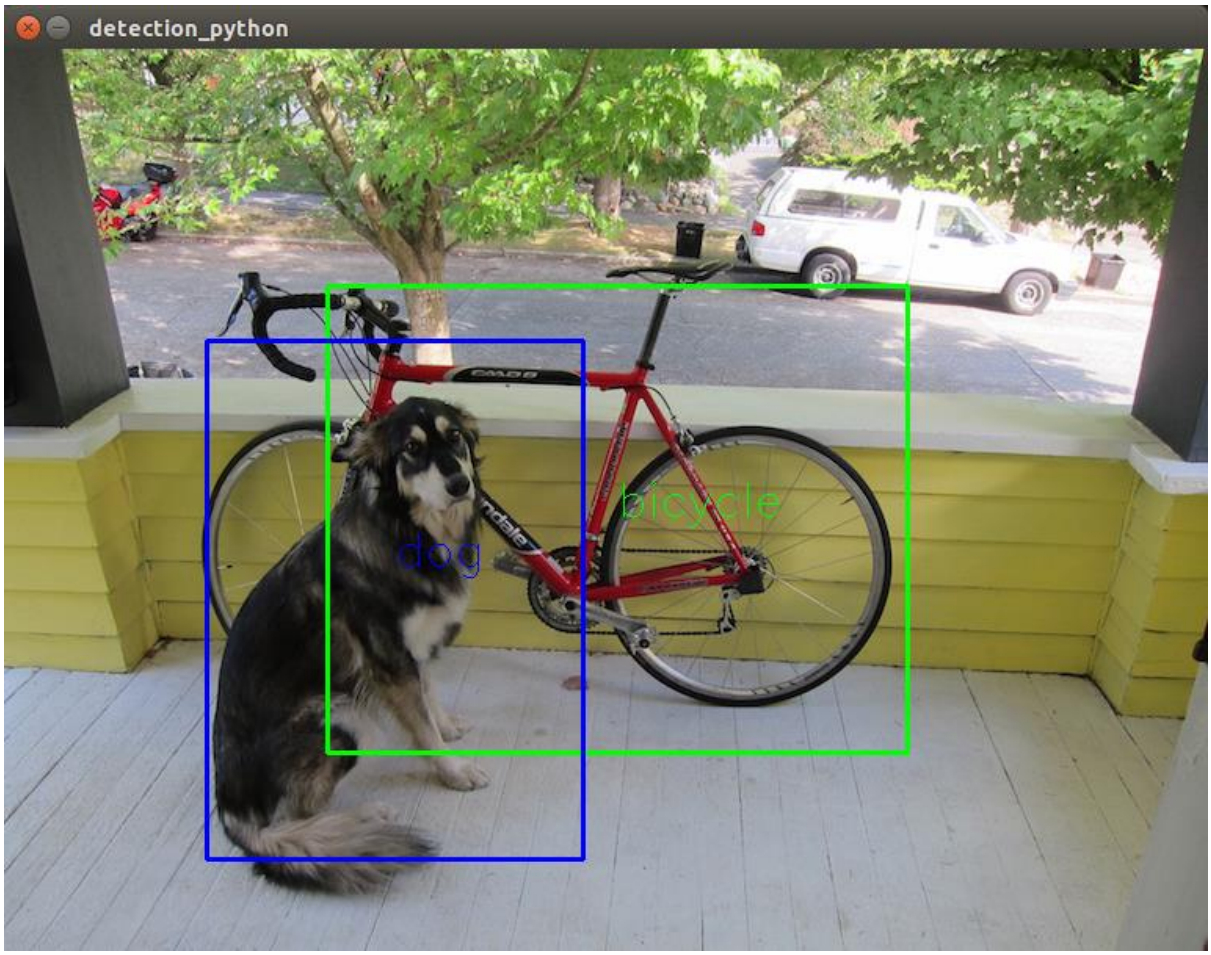


Photo affichée après traitement :

2) Flux webcam

On cherche maintenant à utiliser notre script python crée plus tôt, mais cette fois ci avec le flux de notre webcam. Cela consiste à lire la vidéo et à en faire la lecture et l'affichage frame par frame. On a pour cela écrit un script à part afin de le tester plus facilement :

```
import cv
import sys
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

A l'aide de ce code, nous somme capable de récupérer le flux webcam et de l'afficher sur une fenêtre. Nous allons maintenant intégrer ces bouts de codes à notre détecteur.

```
arguments = sys.argv[1:]

cap = cv2.VideoCapture(0)

while(True):
  # Capture frame-by-frame
  ret, arr = cap.read()

  net = dn.load_net("/fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/darknet/cfg/yolov3-tiny.cfg", "/fs03/share/users/
  meta = dn.load_meta("/fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/darknet/cfg/coco.data")

  im = array_to_image(arr)
  dn.rgbgr_image(im)
  r = dn.detectfj(net, meta, im)
  #print "test direct opencv (cv2) read for image :"
  #print r


  #traitement ========================================



  liste = r
  for index in liste :...
  afficher = True
  break...

  #cv2.rectangle(im,(x,y),(x+w,y+h),(0,255,0),2)
  cv2.imshow(WINDOW_NAME, arr)
  #traitement ========================================

  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

On peut voir sur le code ci-dessus l'intégration du code précédent avec celui de notre détecteur. On lance maintenant ce programme avec une recherche de personne en argument.

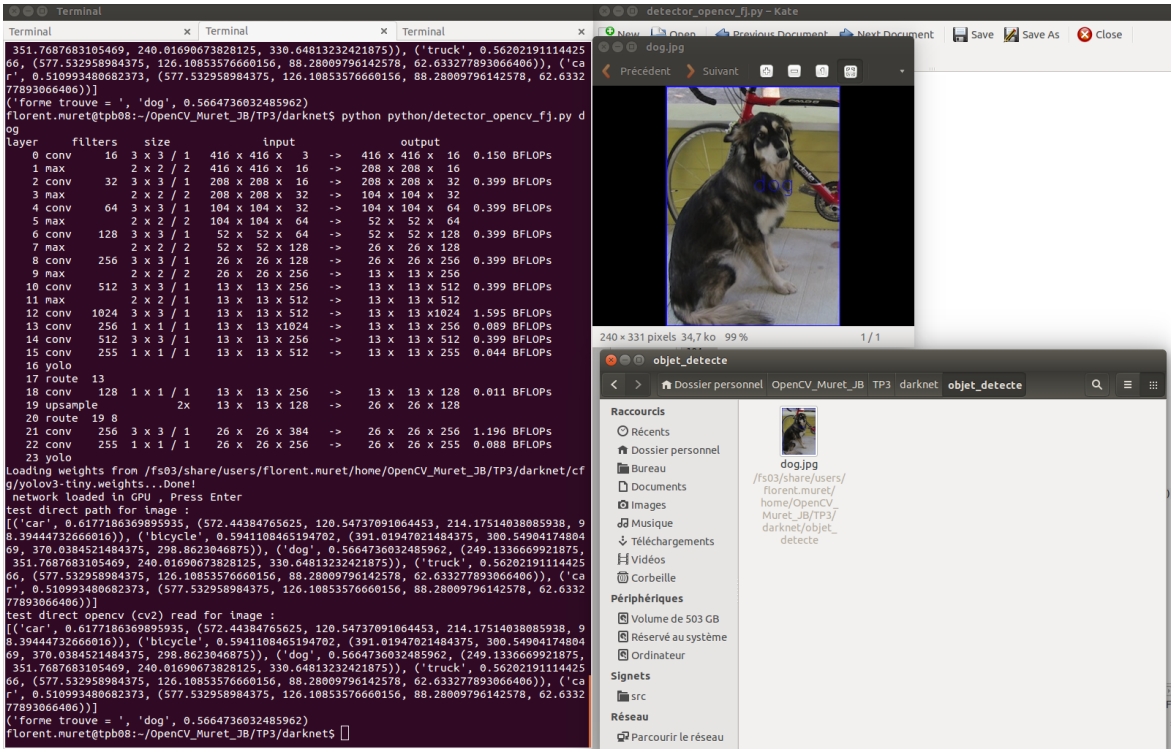
Encore une fois, on se retrouve avec un flux limité par les performances de notre ordinateur, et donc un nombre de FPS = 0.5. Cependant, on peut voir que notre algorithme de traitement reconnait bien les différentes personnes sur la photo, avec peu de précision (on peut lire dans le terminal 0.93, 0.75 et 0.68) car on utilise toujours le modèle Tiny de yolov3.

3) Sauvegarde des box

Dans cette partie nous allons sauvegarder les parties d'images qui contiennent les objets trouvés dans un répertoire appelé « objet_detecte ». Pour cela on rajoute le code suivant :

```
    color = (0,255,0)
  if nom == 'truck' :
    color = (0,0,255)

  if afficher == True :
    cv2.putText(arr, nom, (int(x), int(h)), cv2.FONT_HERSHEY_SIMPLEX, 1.0, color)
    cv2.rectangle(arr,(int(x-(w/2)),int(y-(h/2))),(int(x+(w/2)),int(y+(h/2))),color,2)
    print('forme trouve = ' ,nom , pourcentage)

    first  = x - (w/2)
    second = y - (h/2)

    sub_image = arr[ second:second+h, first:first+w]
    cv2.imwrite('/fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/darknet/objet_detecte/dog.jpg',sub_image)

#cv2.rectangle(im,(x,y),(x+w,y+h),(0,255,0),2)
cv2.imshow(WINDOW_NAME, arr)
cv2.waitKey(0)
```

On test ce code sur le détecteur prévu pour les images et non pour le flux vidéo, sinon cela est trop contraignant au niveau des images sauvegarder et du temps de traitement. Reprenons notre photo première photo, et demandons-lui de sélectionner le chien.



Enregistrement du chien.jpg :

Dans cette partie nous nous sommes limités à un cas simple, mais en théorie nous devrions changer le nom du fichier en modifiant le dernier chemin du Path, avec un compteur s'incrémentant à chaque fois qu'un type a été trouvé : par exemple s'il y a plusieurs chien le code aurait ressemblé à ceci :

```
Directory = '/fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/darknet/objet_detecte/dog.jpg'
Path=  Directory+ 'nom_ '+ str(compteur)+'.jpg'
cv2.imwrite('path',sub_image)
```

Avec un tableau de compteur contenant un compteur pour chaque classe à initialisé à 1 afin d'avoir dans un dossier : chien_1.jpg, chien_2.jpg, etc. On aurait pu également trouvé un moyen plus « propre » d'importer les classes dans notre code, et de définir la couleur à partir de random.

4) Server_fj et client

Dans cette partie, nous allons essayer de prendre en main un code de gestion client/serveur, en locale (sur la même machine), afin de pouvoir l'appliquer plus tard sur une machine distante plus puissante afin de palier à la limite du nombre de FPS.

On récupère tout d'abord les deux scripts python sur l'e-campus. En lisant les codes écrit en python, on comprend qu'il va falloir passer des adresses IP, port et nom de réseau en argument. On regarde donc à l'aide de la commande ifconfig des différents périphériques ou

réseau connectés à l'ordinateur, et de celui dont nous allons nous servir. On remarque qu'il n'y a pas de périphérique eth0.

```
florent.muret@tpb08:~$ ifconfig
cpe       Link encap:Ethernet  HWaddr 74:46:a0:8e:1c:17
          inet adr:134.214.51.155  Bcast:134.214.51.255  Masque:255.255.254.0
          adr inet6: fe80::7646:a0ff:fe8e:1c17/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:562656 erreurs:0 :0 overruns:0 frame:0
          TX packets:430519 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:549747303 (549.7 MB) Octets transmis:256074880 (256.0 MB)
          Interruption:20 Mémoire:f7200000-f7220000

docker0   Link encap:Ethernet  HWaddr 02:42:29:7b:46:94
          inet adr:172.17.0.1  Bcast:172.17.255.255  Masque:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Packets reçus:0 erreurs:0 :0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          Octets reçus:0 (0.0 B) Octets transmis:0 (0.0 B)

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          Packets reçus:29389 erreurs:0 :0 overruns:0 frame:0
          TX packets:29389 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1
          Octets reçus:91910596 (91.9 MB) Octets transmis:91910596 (91.9 MB)

virbr0    Link encap:Ethernet  HWaddr 32:12:a9:e3:7f:b4
          inet adr:192.168.122.1  Bcast:192.168.122.255  Masque:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Packets reçus:0 erreurs:0 :0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:0 (0.0 B) Octets transmis:0 (0.0 B)

vm0       Link encap:Ethernet  HWaddr 68:05:ca:17:ad:4d
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Packets reçus:0 erreurs:0 :0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:0 (0.0 B) Octets transmis:0 (0.0 B)
          Interruption:16 Mémoire:f71c0000-f71e0000
```

On lance maintenant le fichier python *server_fj.py* en lui passant en argument « cpe », qui est le nom du réseau que nous allons utiliser.

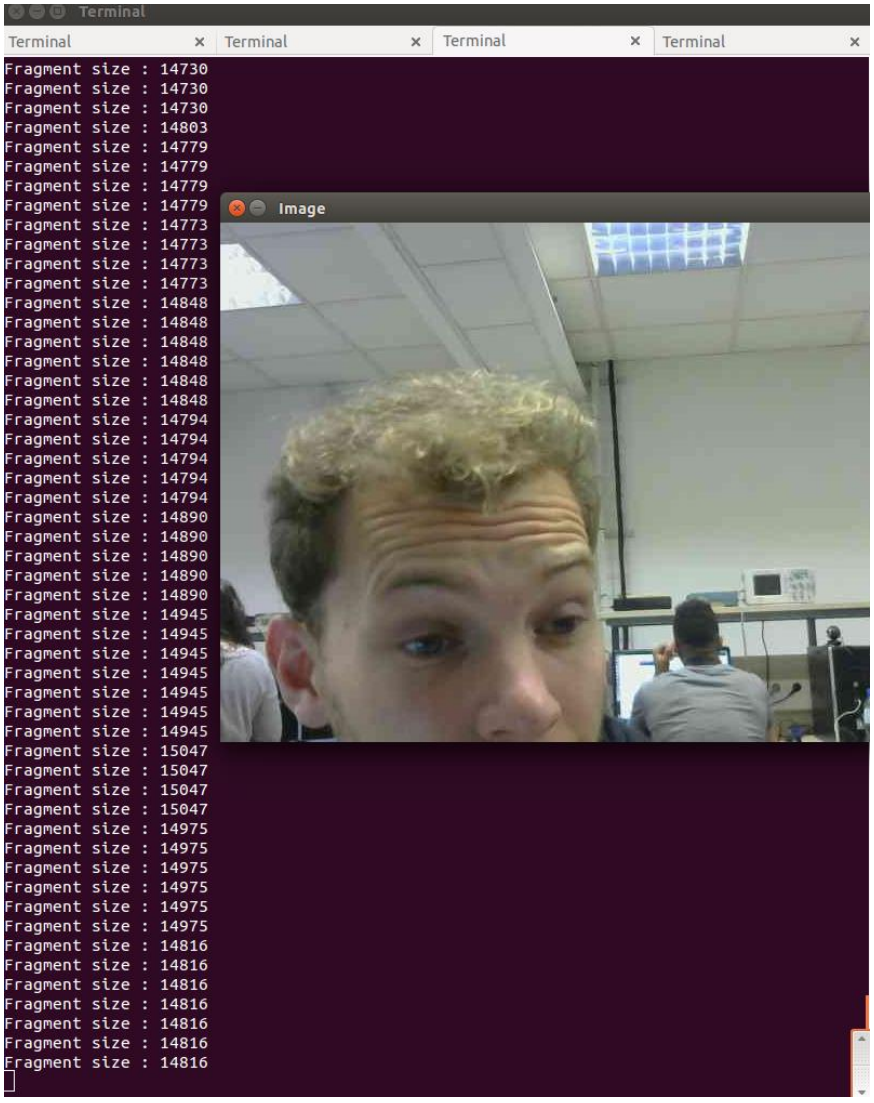
```
florent.muret@tpb08:~/OpenCV_Muret_JB/TP3/darknet$ python python/server_fj.py cpe
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
starting up on 134.214.51.155 port 1080
```

On récupère le numéro du port et l'adresse IP (les paramètres qui sont envoyés au client) :

- Port : 1080
- IP : 134.214.51.155

1080 est un port communément dédié au socks (protocole internet qui échanges des paquets entre un client et un serveur grâce à un serveur proxy).

On va donc lancer le client avec ces paramètres :



Lecture du flux de la webcam via le client :

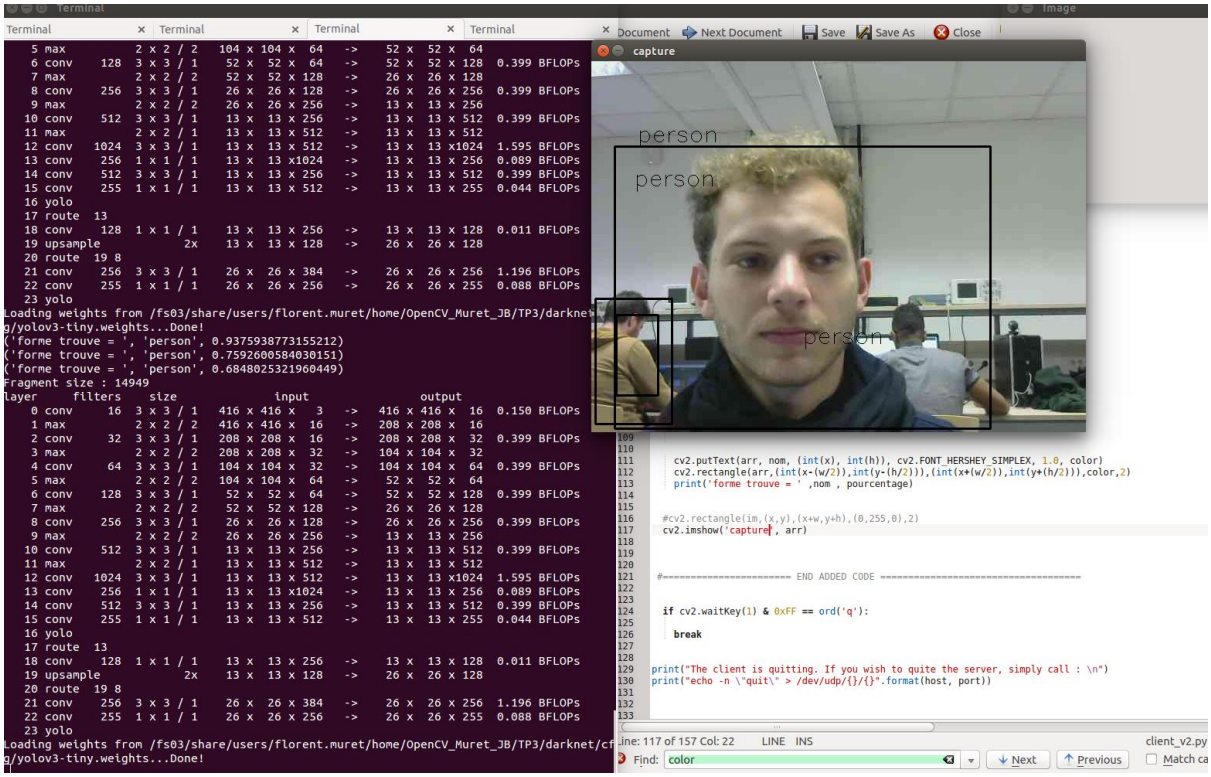
Le programme client.py est alors lancés en prenant pour paramètres le Port et l'IP trouvés précédemment. On récupère donc le flux de la webcam. En effet, le code du serveur l'envoie sur un réseau spécifique et à l'aide du code client, on vient récupérer l'information sur le port associé et on l'affiche.

5) Codage d'un client_v2 pour effectuer le traitement du flux de la webcam :

On part donc du code de client.py et on y intègre le code effectué précédemment afin d'effectuer le traitement de chaque frame.

```
sys.path.append(os.path.join(os.getcwd(),'python/'))


while(True):

#======================  recuperation du flux sur le reseau  ==============
  sent = sock.sendto("get", server_address)

  data, server = sock.recvfrom(65507)
  print("Fragment size : {}".format(len(data)))
  if len(data) == 4:
      # This is a message error sent back by the server
      if(data == "FAIL"):

          continue
  array = np.frombuffer(data, dtype=np.dtype('uint8'))
  arr = cv2.imdecode(array, 1)

#======================  ADDED CODE ====================================

  net = dn.load_net("/fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/darknet/cfg/yolov3-tiny.cfg", "/fs03/shar
  meta = dn.load_meta("/fs03/share/users/florent.muret/home/OpenCV_Muret_JB/TP3/darknet/cfg/coco.data")

  im = array_to_image(arr)
  dn.rgbgr_image(im)
  r = dn.detectfj(net, meta, im)

  liste = r
#======================== TRAITEMENT ====================================
  for index in liste :...
  cv2.imshow('capture', arr)

 #====================== END ADDED CODE ================================


  if cv2.waitKey(1) & 0xFF == ord('q'):...


print("The client is quitting. If you wish to quite the server, simply call : \n")
print("echo -n \"quit\" > /dev/udp/{}/{}".format(host, port))
```

En relançant le code du client on obtient le même résultat qu'avant c'est-à-dire la lecture du flux vidéo avec un FPS = 0.5.



Code client /server et traitement du flux webcam :

Nous allons maintenant passer à l'exécution de ce code sur une machine distante afin d'accélérer le traitement.

2) Sur la machine de calcul

On arrive finalement à la partie qui va nous permettre de palier au manque de performances de notre machine. En effet, nous allons utiliser le code précédent afin d'exécuter le traitement des frames sur une machine distante plus performante. Le principe est simple : avec un code serveur on va transférer le flux de notre webcam à la machine distante, et nous allons la traiter et lire sur flux via cette machine distante via le code client.

On va donc dans un premier temps se connecter à la machine distante et sur un port disponible (plusieurs connexions vont être effectuées par les différents binômes sur les différents ports).

On se connecte via ssh sur le port07 avec la commande suivante :

ssh -x guest_std07@deeplearning.cpe.fr

Une fois la connexion établie il va falloir transférer le code du client ainsi que tout le répertoire nécessaire à l'exécution du darknet afin que la machine distante puisse effectuer le traitement. On crée une archive de notre dossier en tar.gz.

Pour cela on utilise la commande **scp** dont le nom provient de secure cp(copy). Cette commande permet de copier des fichiers à travers une connexion ssh. La connexion étant cryptée, c'est une méthode sécurité de copier des fichiers entre ordinateurs.

scp darknet.tar.gz guest_std07

```
florent.muret@tpb08:~/OpenCV_Muret_JB/TP3/darknet$ cd ..
florent.muret@tpb08:~/OpenCV_Muret_JB/TP3$ scp darknet.tar.gz guest_std07@deeplearning:
guest_std07@deeplearning's password:
darknet.tar.gz                                100%  513MB  10.1MB/s   00:51
florent.muret@tpb08:~/OpenCV_Muret_JB/TP3$
```

On effectue ensuite l'extraction de l'archive sur le répertoire de la machine :

Tar zxvf votre_archive.tar.gz

On obtient alors notre répertoire darknet sur notre machine distante.

```
darknet/data/labels/115_7.png
darknet/obj/instance-segmenter.o
darknet/data/labels/85_6.png
darknet/data/labels/111_6.png
darknet/data/labels/46_1.png
guest_std07@deeplearning:~$ ls
darknet  darknet.tar.gz  examples.desktop
guest_std07@deeplearning:~$ cd darknet/
guest_std07@deeplearning:~/darknet$ ls
backup    include        LICENSE.gen   Makefile       results
cfg       libdarknet.a   LICENSE.gpl   obj            scripts
darknet   libdarknet.so  LICENSE.meta  objet_detecte  src
data      LICENSE        LICENSE.mit   python         yolov3-tiny.weights
examples  LICENSE.fuck   LICENSE.v1    README.md      yolov3.weights
guest_std07@deeplearning:~/darknet$
```

Il faut bien penser à modifier le makefile (à l'aide de la commande « nano ») afin de mettre gpu =1 ainsi que cudnn =1, puis de recompiler le projet (« make clean » puis « make »).

Ensuite on peut exécuter notre programme sur le serveur. On peut tout d'abord exécuter notre traitement d'image avant notre flux vidéo afin de voir si on a correctement modifié les path dans notre code, puis on peux passer à la vidéo.

Durant notre TP nous n'avons pas eu le temps d'allez plus loin dû à un problème sur la machine distante.

Dans cette partie on utilise udp (un protocole) mais on n'a pas de retour, c'est unidirectionnel. On aurait pu faire une requête http qui renverrai un JSON mais ce n'est pas le cas ici.

Entraîner votre propre classifier

Question

> Essayer de préparer un dataset, de préparer les fichiers de config au sens de Yolo darknet puis de lancer un apprentissage, d'abord sur votre machine locale puis nous pourrons tester dans les jours à venir sur la machine de calcul.

Dans cette partie il nous est demandé d'entrainer notre propre classe, c'est-à-dire un quelque chose à détecter par darknet. Cela peut être une forme ou un objet spécifique, plus détaillé, ou bien quelque chose qu'y n'est pas encore dans la librairie de classe proposé par darknet.

Pour cela on suit le tuto : https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

Pour cela on a besoin d'une bibliothèque d'image mais surtout d'un grand nombre de « training », ce qui consiste à répéter le processus en fonction du nombre d'itérations. Celle du répertoire « weight- file » sont des classes faites à partir de plus de 10000 itérations. Nous n'avons pas pu traiter cette partie par manque de temps.

Conclusion

Dans ce TP on a pu avoir un aperçu des capacités d'analyse dynamique que possède yolov3 darknet. On a pu comparer les performances entre yolov3 et yolov3-tiny, et ainsi en déduire les applications possibles. Nous avons également été confronté à l'un des problèmes majeurs du traitement d'image, qui est le temps de traitement des algorithmes. On a donc essayé d'utiliser une machine distante et un code adéquat à son exécution pour palier à ce problème.

On peut cependant avoir un aperçu du résultat que cela donnerait en regardant les résultats sur des machines plus performantes (https://www.youtube.com/watch?v=ZQlvjFOo9i8), et imaginer toutes les applications possibles pour la robotique (tracking d'objets spécifiques en fonction de leurs caractéristiques, détection d'objet etc…).