

Traitement d'images avec des réseaux de neurones pour détecter des personnes qui sont tombées



source:www.assystel.fr

Les personnes âgées sont fragiles, une simple chute pourrait les immobiliser ou même pire leur être fatal.

C'est pour cela que nous nous intéressons à la détection de chute.

n°00000

Nom Prénom

1

Thème 2021-2022 : santé, prévention

Sommaire

Problème à résoudre

1. Structure du projet
2. Problème de classification

Codage de l'algorithme de détection de chute

1. Explication du Machine Learning
2. Construction de la base de données
3. Entrainement et test du modèle
4. Présentation de l'algorithme YOLO
5. Construction de la base de données
6. Entrainement et test final du modèle

Détection sur caméra

1. Chrono sur une chute
2. Chrono et suivi sur plusieurs chutes

Structure du projet



Problème de classification

On veut pouvoir classifier une personne selon 2 états.

Classe 1 :

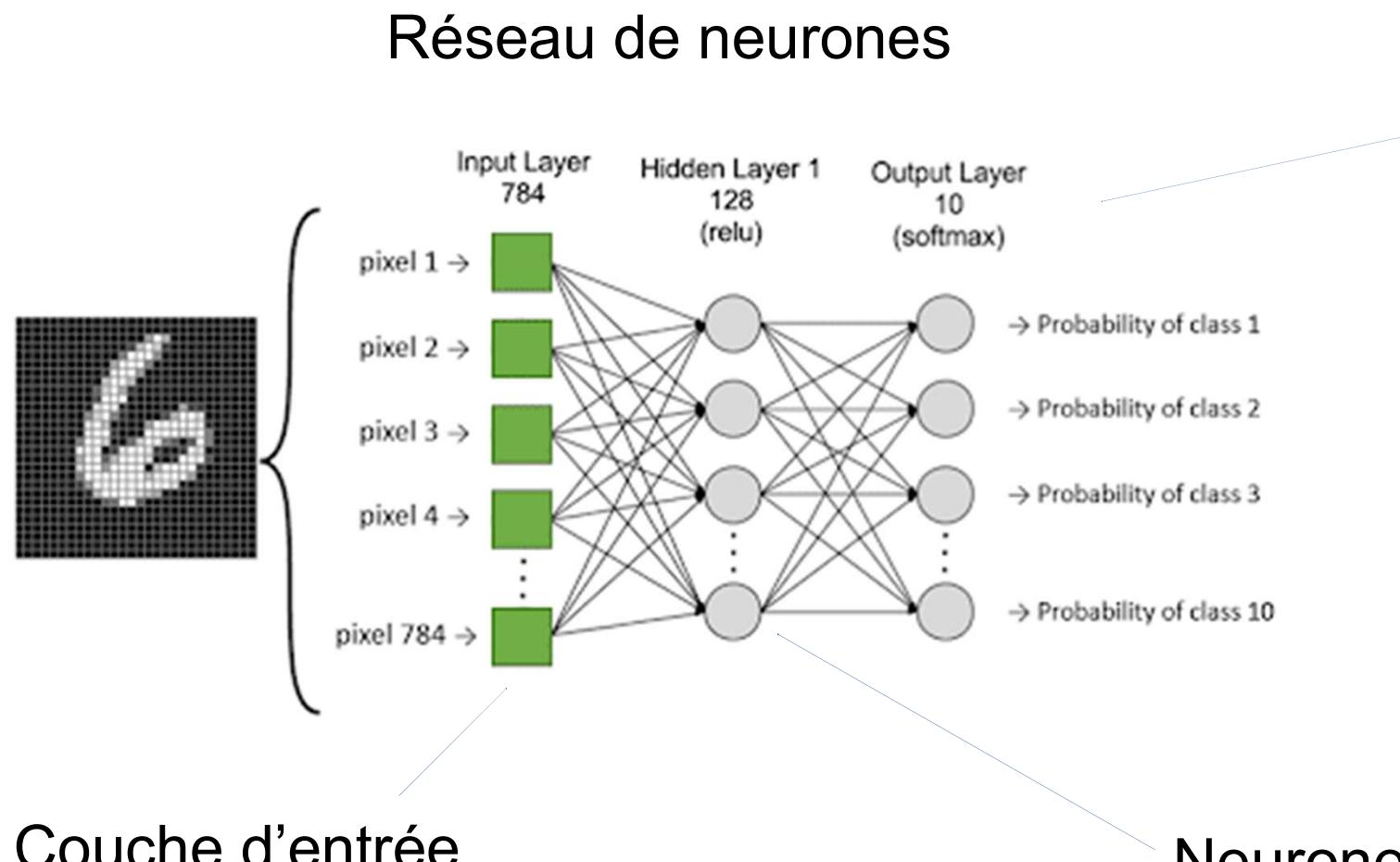
Personne tombée

Classe 2 :

Personne non-tombée

Le machine learning est une branche de l'intelligence artificielle qui est principalement utilisée pour des problèmes de classification, c'est ce que nous utiliserons ici.

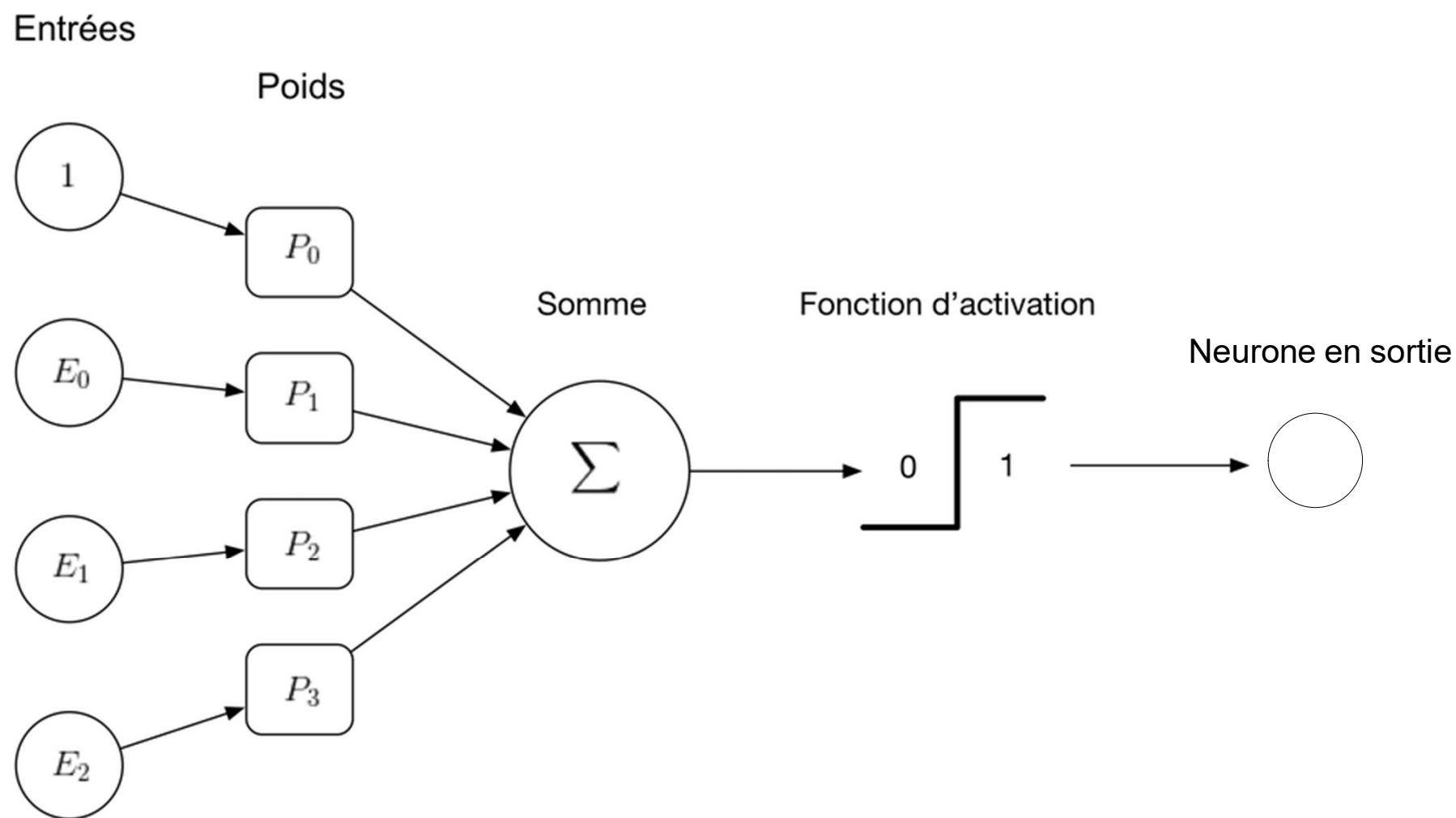
Explication Machine Learning



Un lien représente un poids/un paramètre.
On a pour ce réseau de neurones $784 \times 128 \times 10 = 1,003,520$ paramètres.
Un paramètre = un nombre réel

source:mxnet.apache.org/versions/1.4.1/tutorials/python/mnist.html

Explication Machine Learning

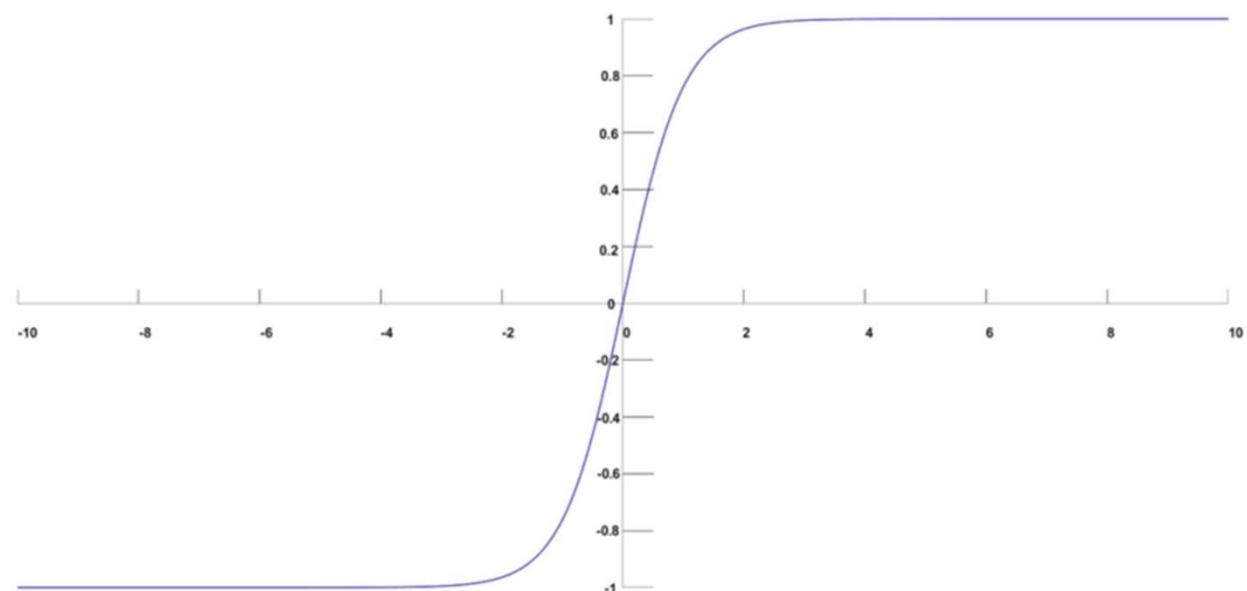


source:hal-sciencespo.archives-ouvertes.fr

Explication Machine Learning

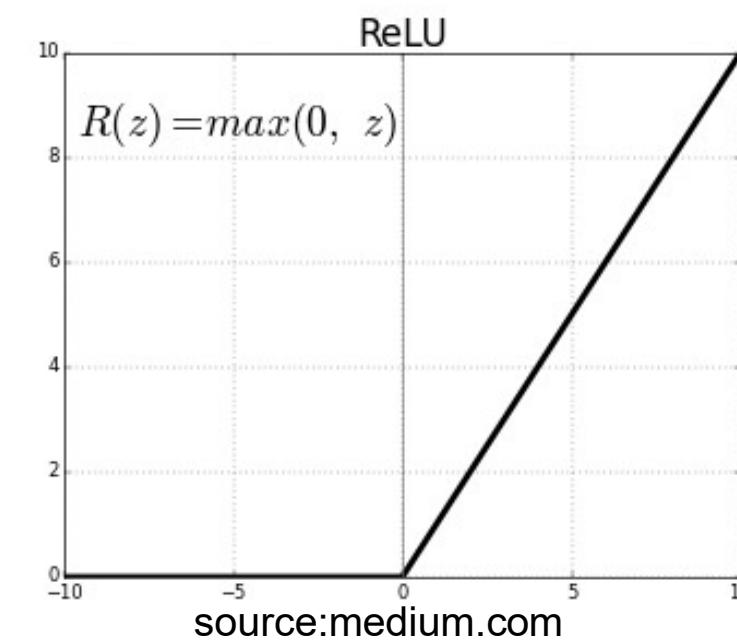
Fonctions de transfert :

$$\text{softmax}(x) = \begin{bmatrix} \frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}} \\ \frac{e^{x_2}}{\sum_{i=1}^n e^{x_i}} \\ \dots \\ \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}} \end{bmatrix}$$

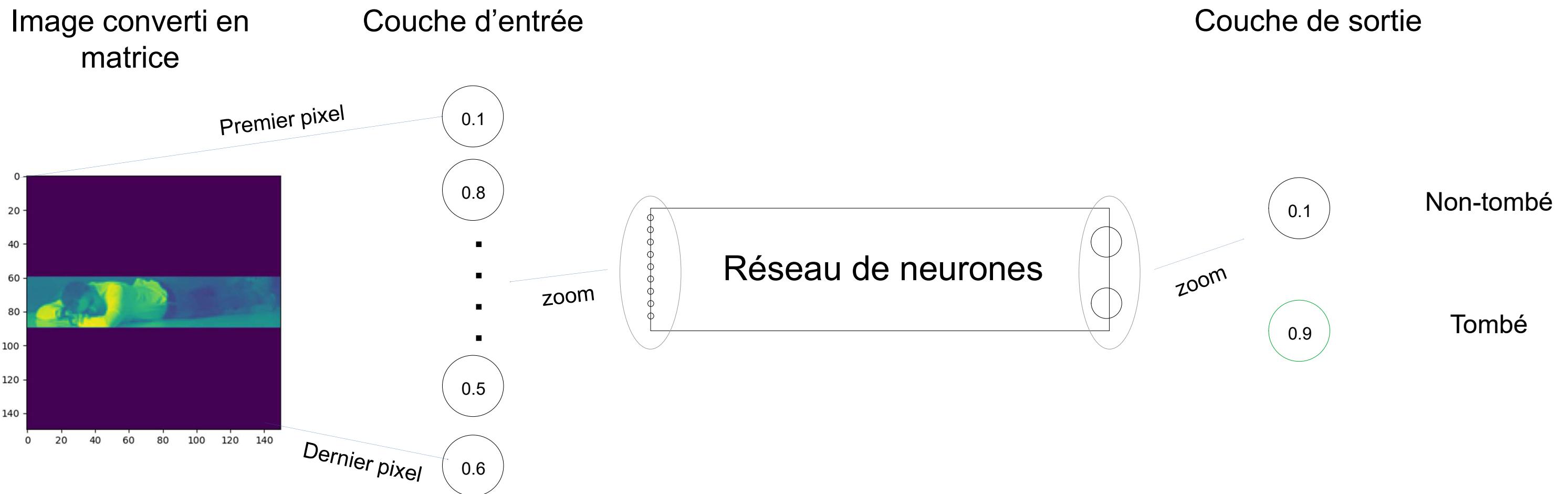


source:www.researchgate.net

ReLU



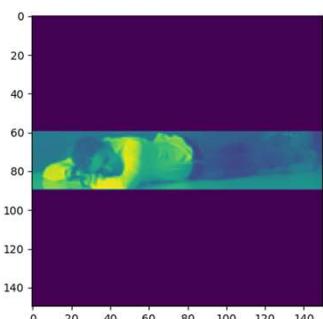
Explication Machine Learning



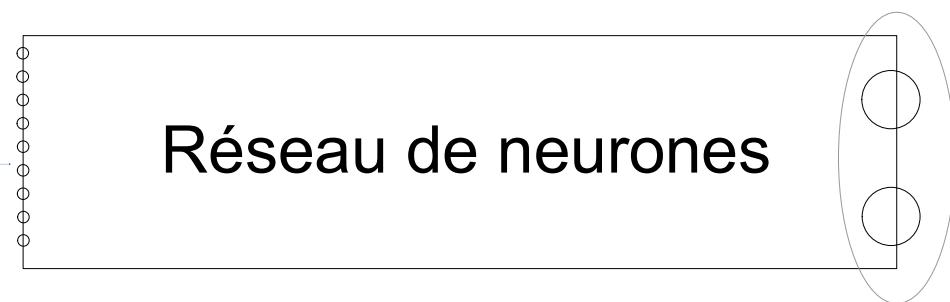
Explication Machine Learning

Entrainement du réseau de neurones

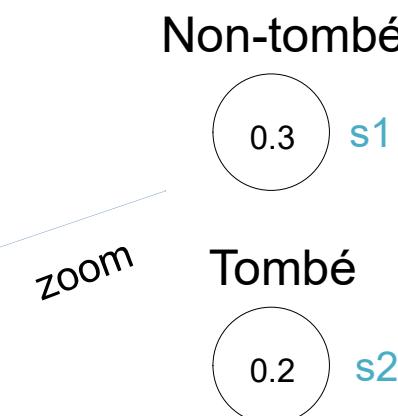
Couche d'entrée



Réseau de neurones



Couche de sortie



Couche de sortie
voulu

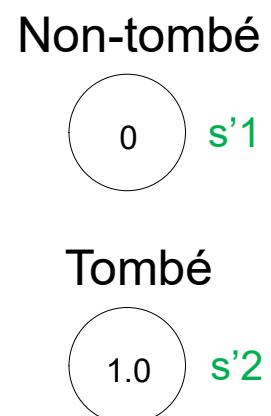


Image étiquetée tombée

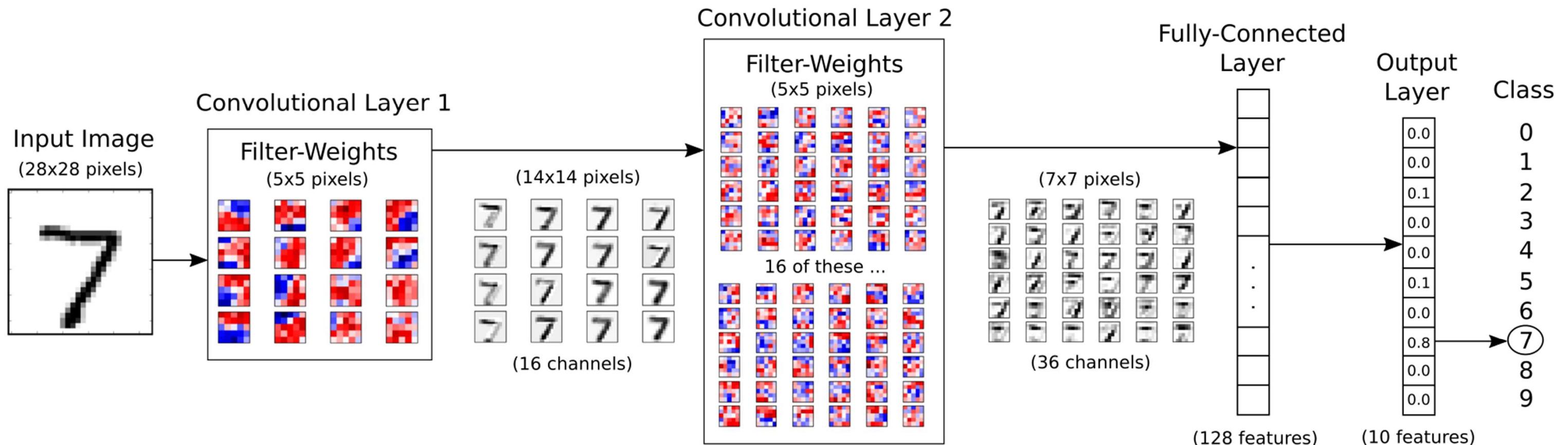
Par exemple dans notre exemple: $C(a_1, a_2, \dots, a_n) = (0.3 - 0)^2 + (0.2 - 1.0)^2 = 0.73$
 $= (s_1 - s'1)^2 + (s_2 - s'2)^2$

Fonction de coût : $C(a_1, a_2, \dots, a_n) = \sum_{k=1}^n (s_k - s'k)^2$

L'entraînement du réseau de neurones peut être assimilé à faire tendre cette fonction vers 0.

Explication Machine Learning

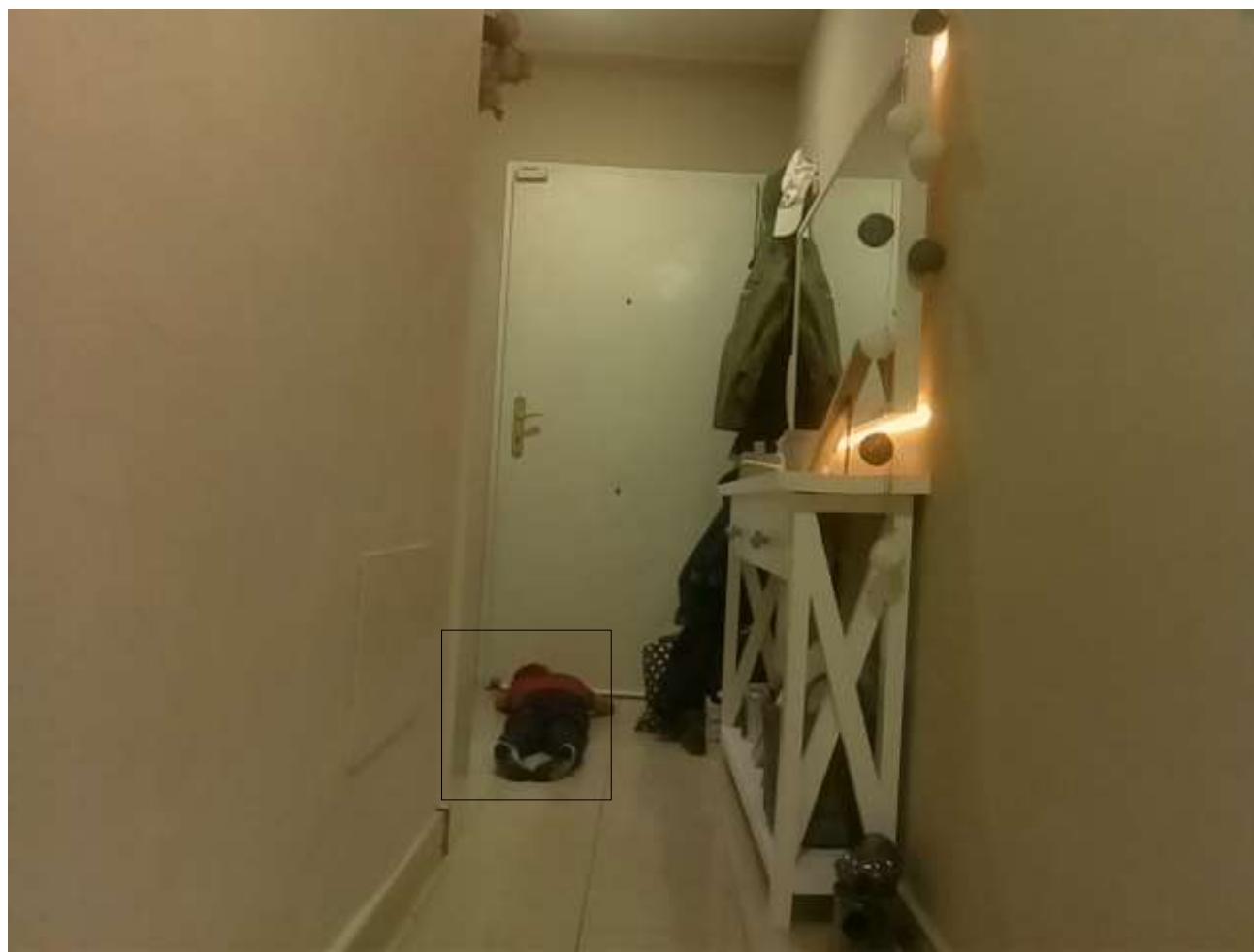
Réseau de neurones convolutif



source:datapeaker.com

Construction de la base de données

Base de données assez importante 7000 images annotées mais il y a un nombre déséquilibré (70 %) de personnes tombées et (30 %) de personnes non-tombées



Personnes étiquetées tombées dans ma base de données

Construction de la base de données



Personnes étiquetées non-tombées

Construction de la base de données

Récupération et formatage des données

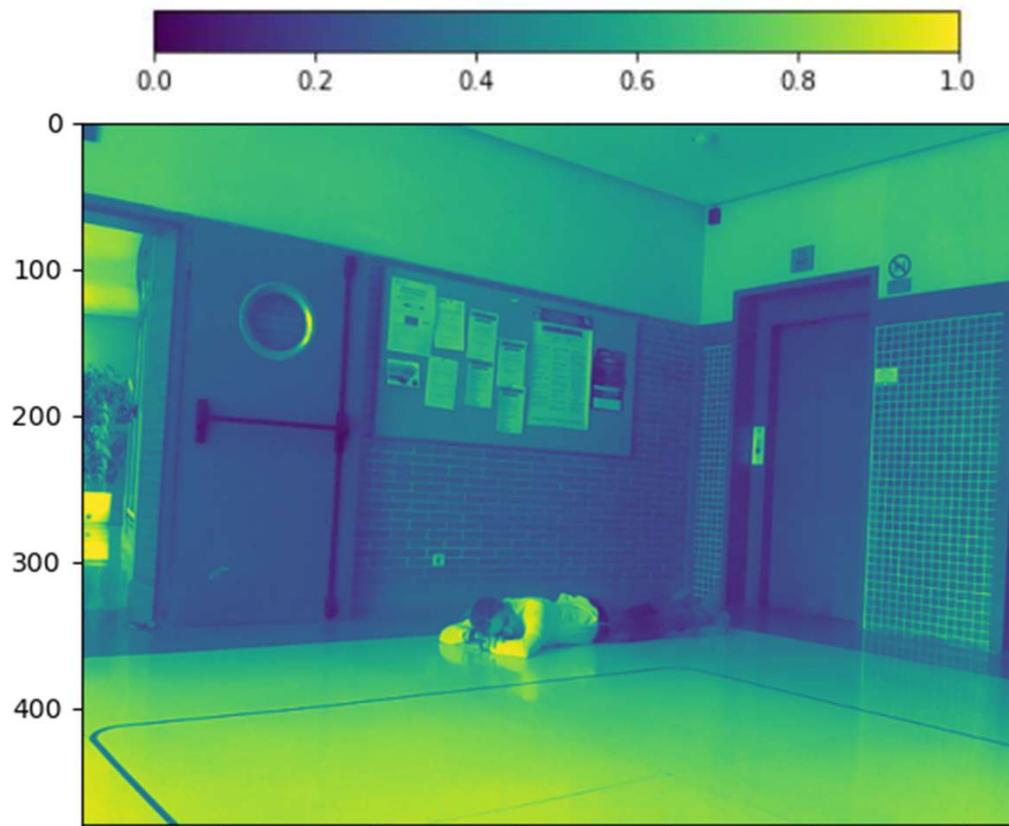
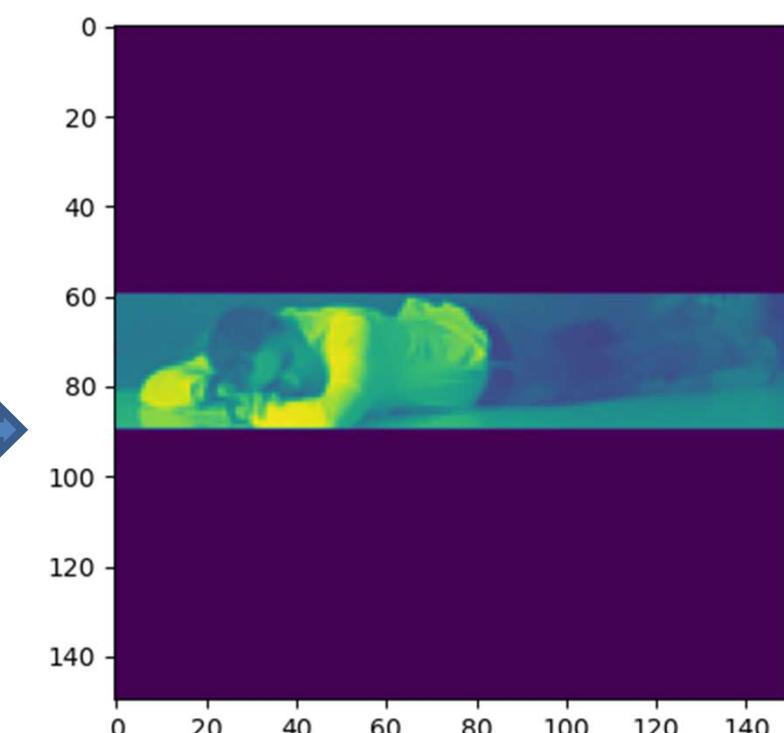
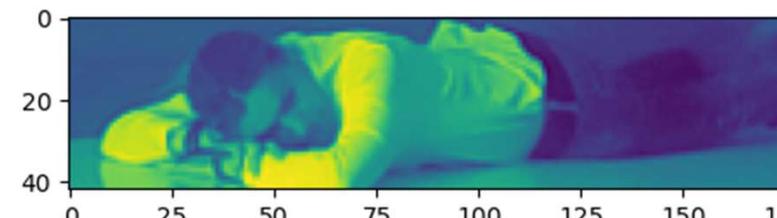


Image en noir et blanc



Matrice à envoyer dans le
réseau de neurones

Entrainement et test du modèle

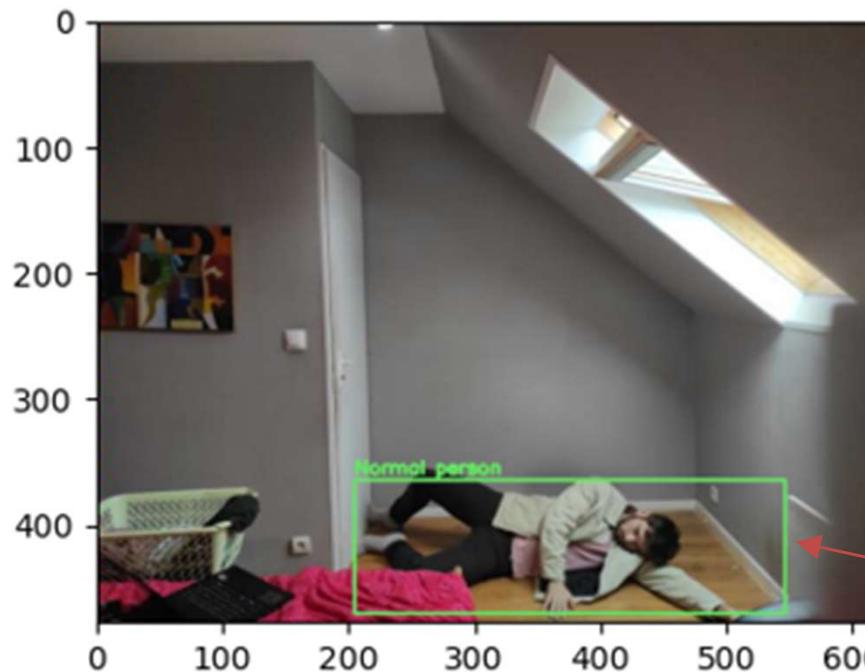
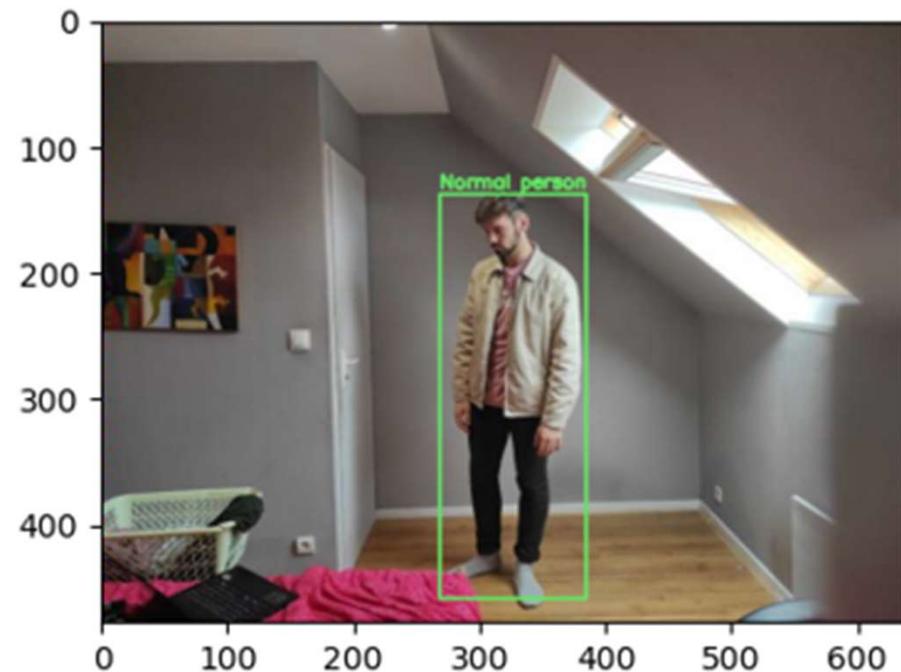
- Réseau de neurones convolutif

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(30, kernel_size=(10,10), activation="relu", padding="same", input_shape=(WIDTH,HEIGHT,1)),
    tf.keras.layers.MaxPool2D(pool_size=(3,3)),
    tf.keras.layers.Conv2D(10, kernel_size=(10, 10), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(3,3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(20,activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(2,activation="softmax")
])
```

« petit » réseau de neurones

Total params: 66,902
Trainable params: 66,902
Non-trainable params: 0

- Mais les résultats sur les images de test sont assez décevant



(60% de réussite sur des images qu'il n'a jamais vu)

Personne qui ne peut pas être confondu avec une personne non-tombée

Construction de la base de données

Problème : manque de données ?

mpii_human_pose_v1

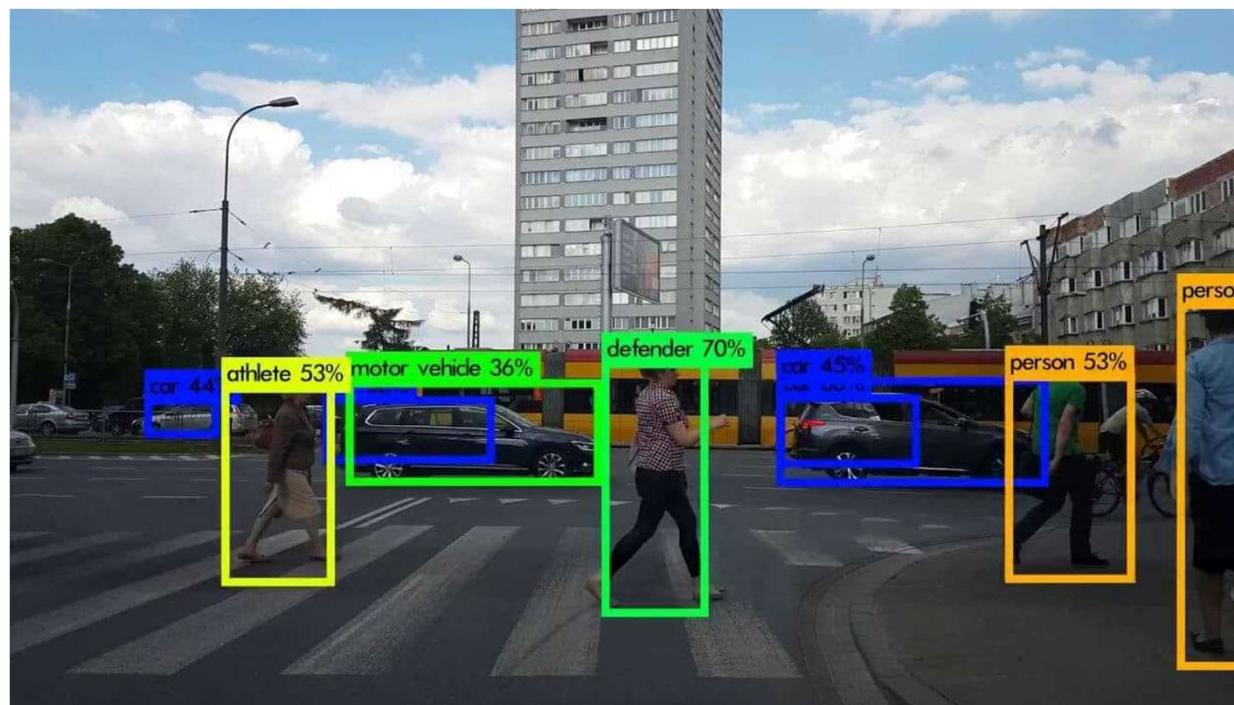


source:human-pose.mpi-inf.mpg.de

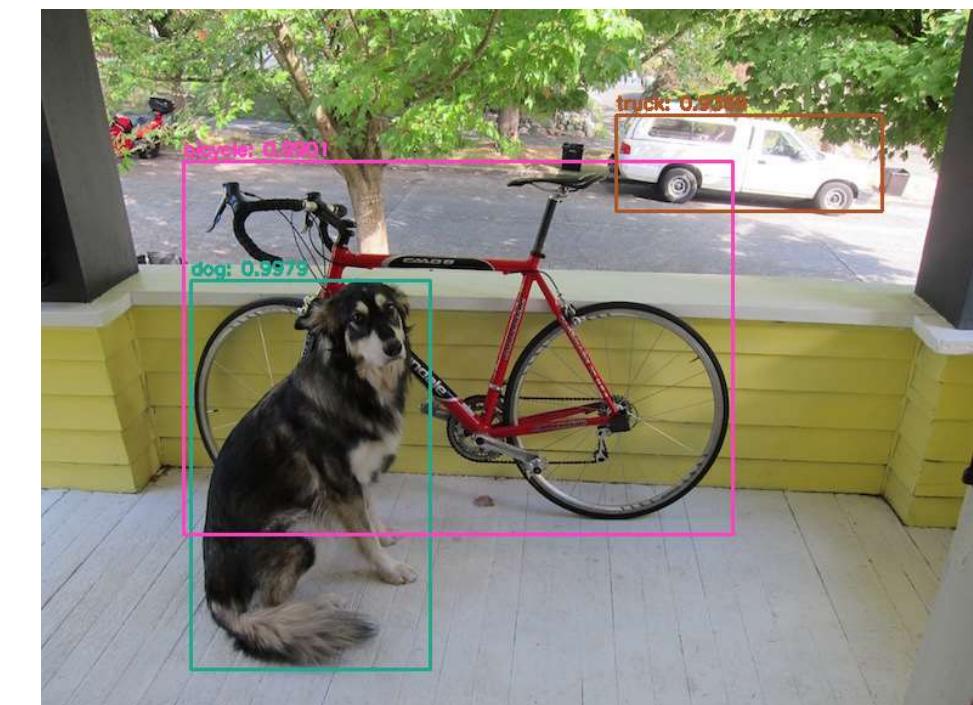
Présentation de l'algorithme YOLO

Utilisation de YOLO (You Only Look Once)

- L'algorithme prend en entrée une image et renvoie une liste de position de rectangle contenant des objets qu'il a reconnu (dont des humains)



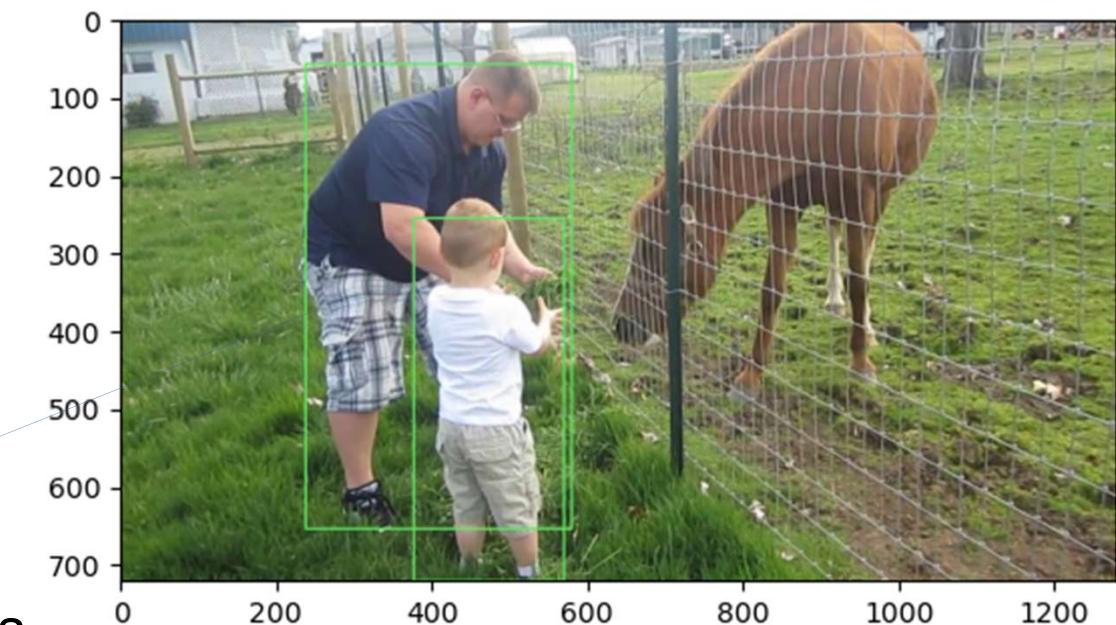
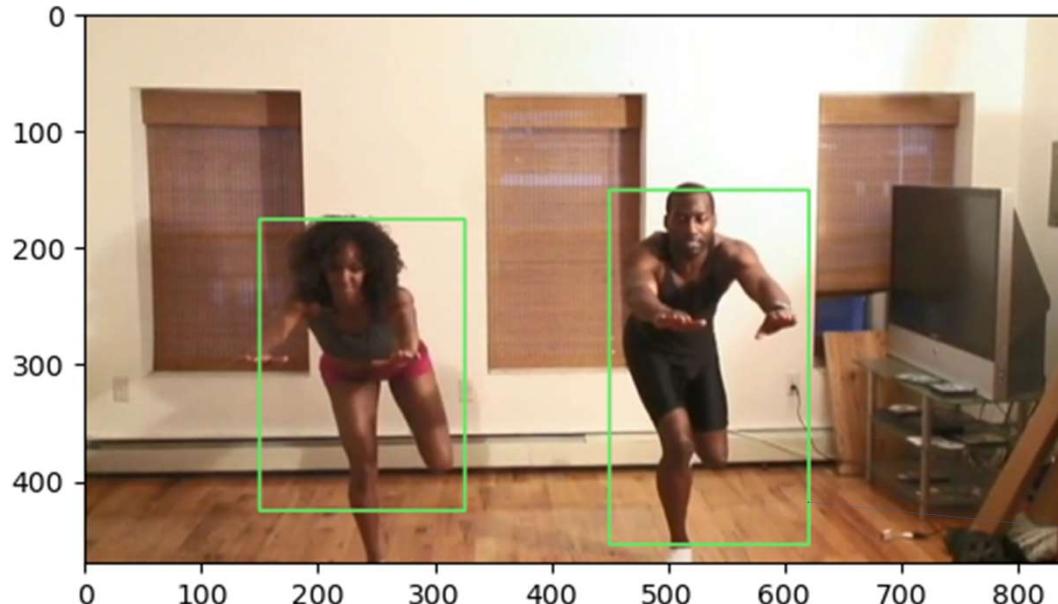
source: www.v7labs.com/blog/yolo-object-detection



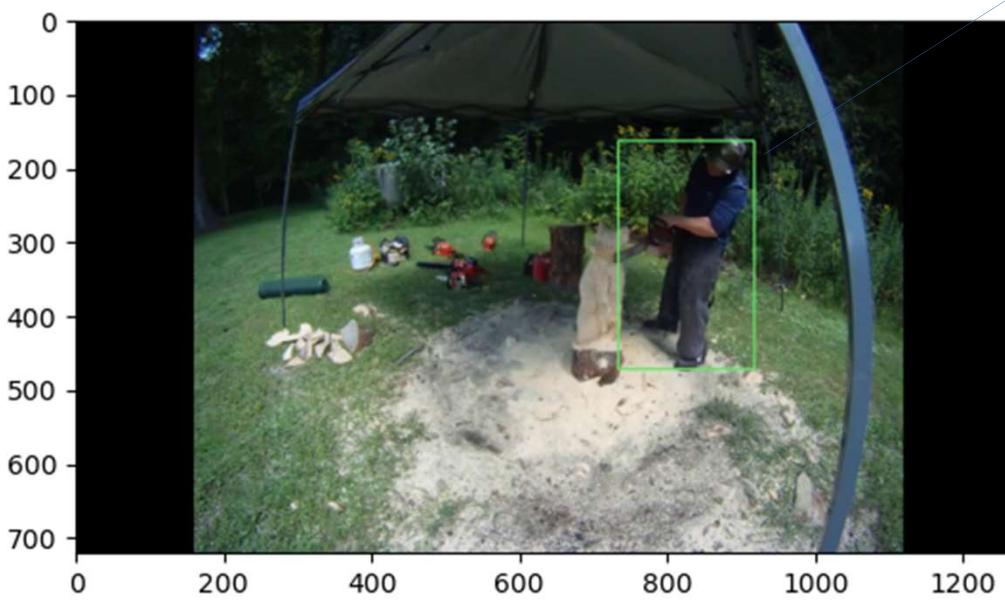
source: www.v7labs.com/blog/yolo-object-detection

Construction de la base de données

Annotation des nouvelles données



Détourage des personnes avec YOLO



```
Ajout des métadonnées aux images du dossier mpii_human_pose_v1
0 : mpii_human_pose_v1/images/086196676.jpg
mpii_human_pose_v1/images/086196676.txt
1 : mpii_human_pose_v1/images/040330989.jpg
mpii_human_pose_v1/images/040330989.txt
2 : mpii_human_pose_v1/images/015061095.jpg
mpii_human_pose_v1/images/015061095.txt
3 : mpii_human_pose_v1/images/005318534.jpg
mpii_human_pose_v1/images/005318534.txt
4 : mpii_human_pose_v1/images/033615966.jpg
mpii_human_pose_v1/images/033615966.txt
5 : mpii_human_pose_v1/images/057213775.jpg
mpii_human_pose_v1/images/057213775.txt
6 : mpii_human_pose_v1/images/084118871.jpg
mpii_human_pose_v1/images/084118871.txt
7 : mpii_human_pose_v1/images/001521653.jpg
mpii_human_pose_v1/images/001521653.txt
8 : mpii_human_pose_v1/images/062385932.jpg
mpii_human_pose_v1/images/062385932.txt
9 : mpii_human_pose_v1/images/017514063.jpg
```

Construction de la base de données

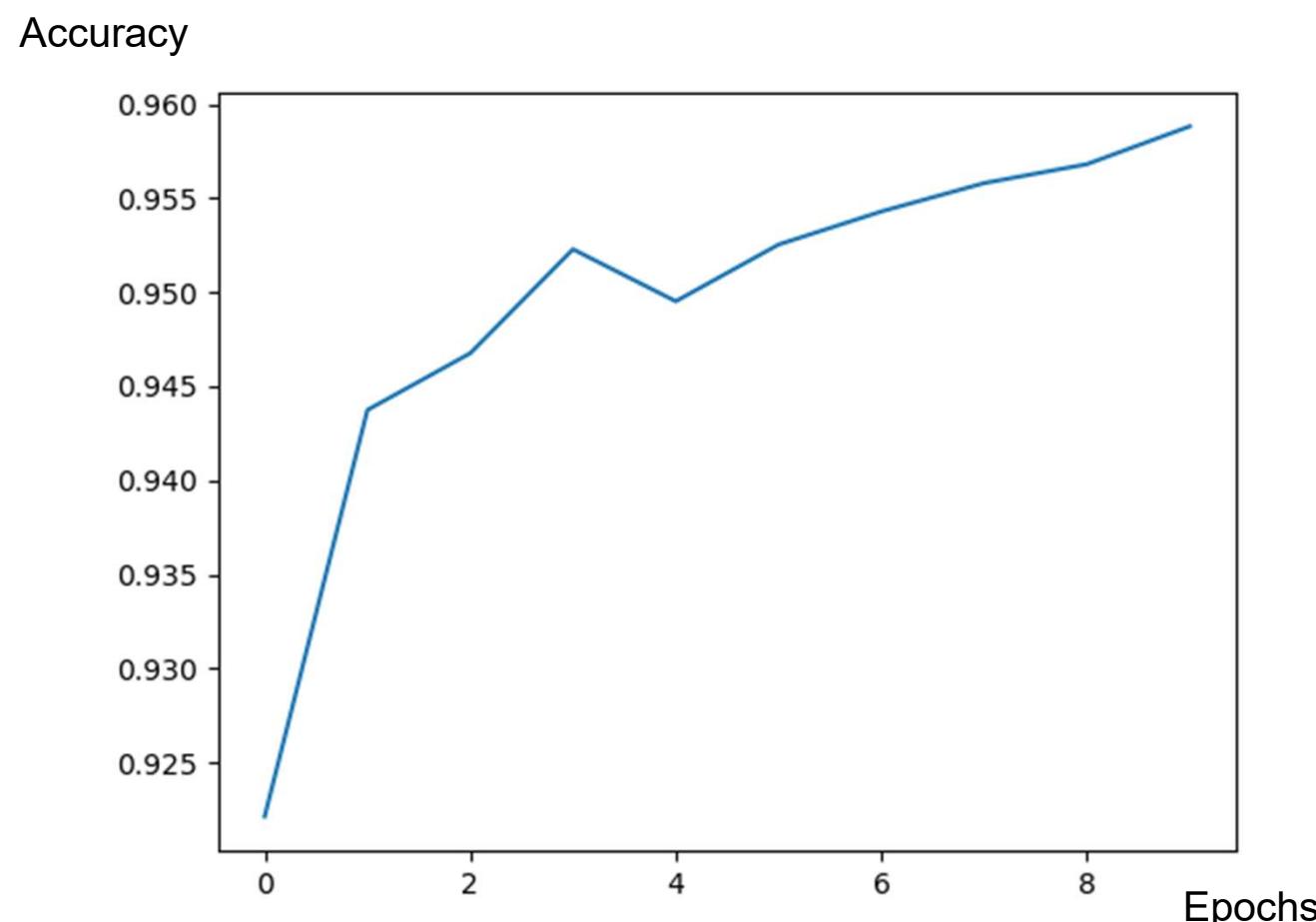
- Elle contient maintenant à peu près 10 000 images étiquetées avec 50/50 % de personnes qui sont tombées et qui sont non-tombées
- On relance le même modèle avec la nouvelle base de données

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(30, kernel_size=(10,10), activation="relu", padding="same", input_shape=(WIDTH,HEIGHT,1)),
    tf.keras.layers.MaxPool2D(pool_size=(3,3)),
    tf.keras.layers.Conv2D(10, kernel_size=(10, 10), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(3,3)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(20,activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(2,activation="softmax")
])
```

La base de données est assez variée et assez grande pour donner de meilleurs résultats

Entrainement et test final du modèle

Précision lors de l'apprentissage



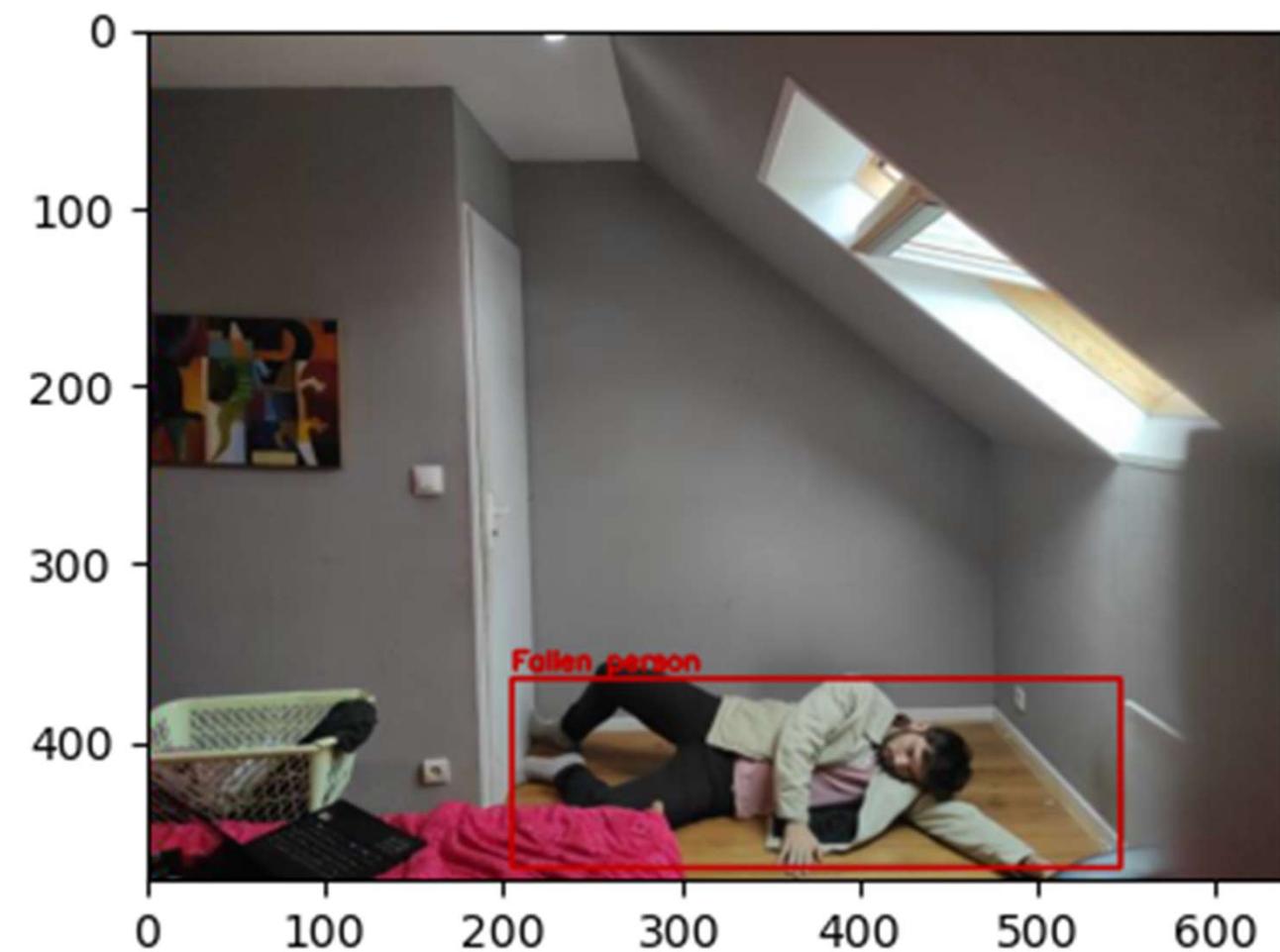
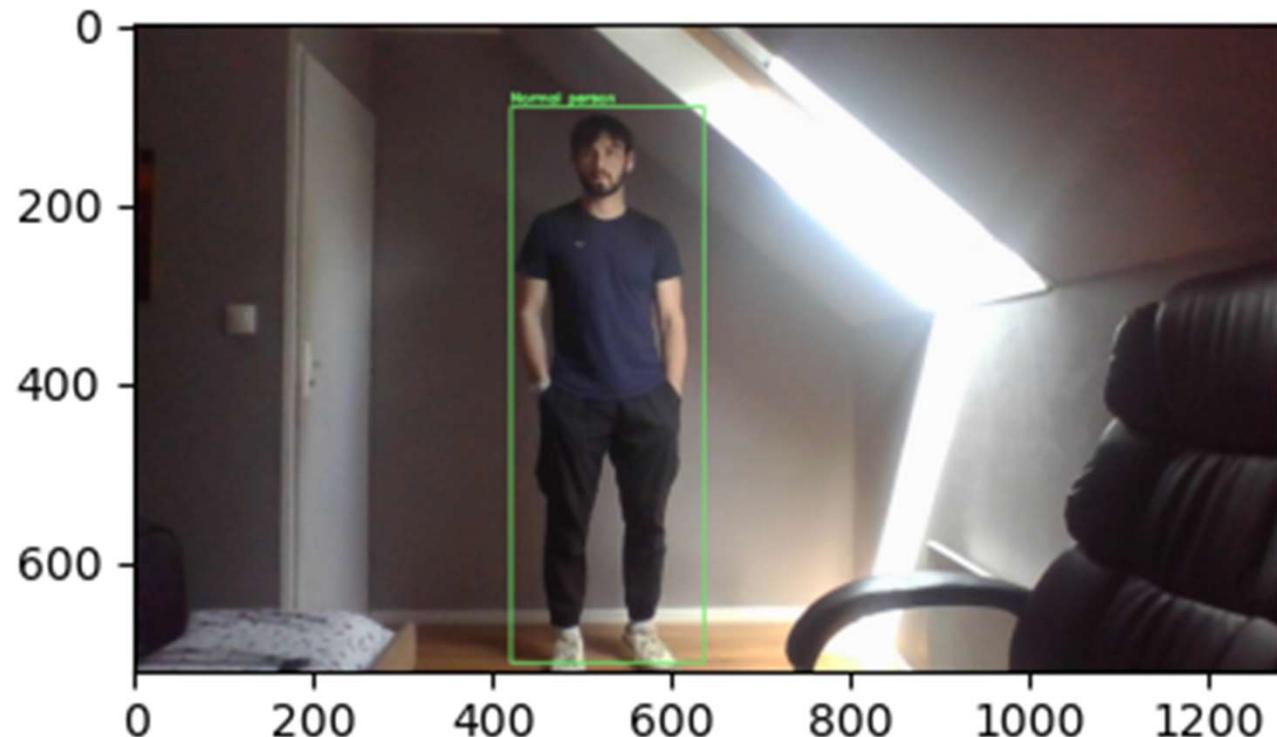
Temps d'entraînement et de chargement des données 25min

```
entraînement
Epoch 1/10
125/125 [=====] - 65s 514ms/step - loss: 0.2525 - accuracy: 0.9221
Epoch 2/10
125/125 [=====] - 71s 566ms/step - loss: 0.1886 - accuracy: 0.9437
Epoch 3/10
125/125 [=====] - 58s 465ms/step - loss: 0.1804 - accuracy: 0.9468
Epoch 4/10
125/125 [=====] - 84s 671ms/step - loss: 0.1589 - accuracy: 0.9523
Epoch 5/10
125/125 [=====] - 91s 725ms/step - loss: 0.1531 - accuracy: 0.9495
Epoch 6/10
125/125 [=====] - 92s 740ms/step - loss: 0.1401 - accuracy: 0.9525
Epoch 7/10
125/125 [=====] - 84s 676ms/step - loss: 0.1254 - accuracy: 0.9543
Epoch 8/10
125/125 [=====] - 65s 523ms/step - loss: 0.1193 - accuracy: 0.9558
Epoch 9/10
125/125 [=====] - 67s 533ms/step - loss: 0.1166 - accuracy: 0.9568
Epoch 10/10
125/125 [=====] - 88s 704ms/step - loss: 0.1010 - accuracy: 0.9588
Precision: 0.9508032202720642
Erreurs: 0.13723856210708618
Affichage apprentissage
Test sur valeur inconnue
enregistrement du réseau de neurones
```

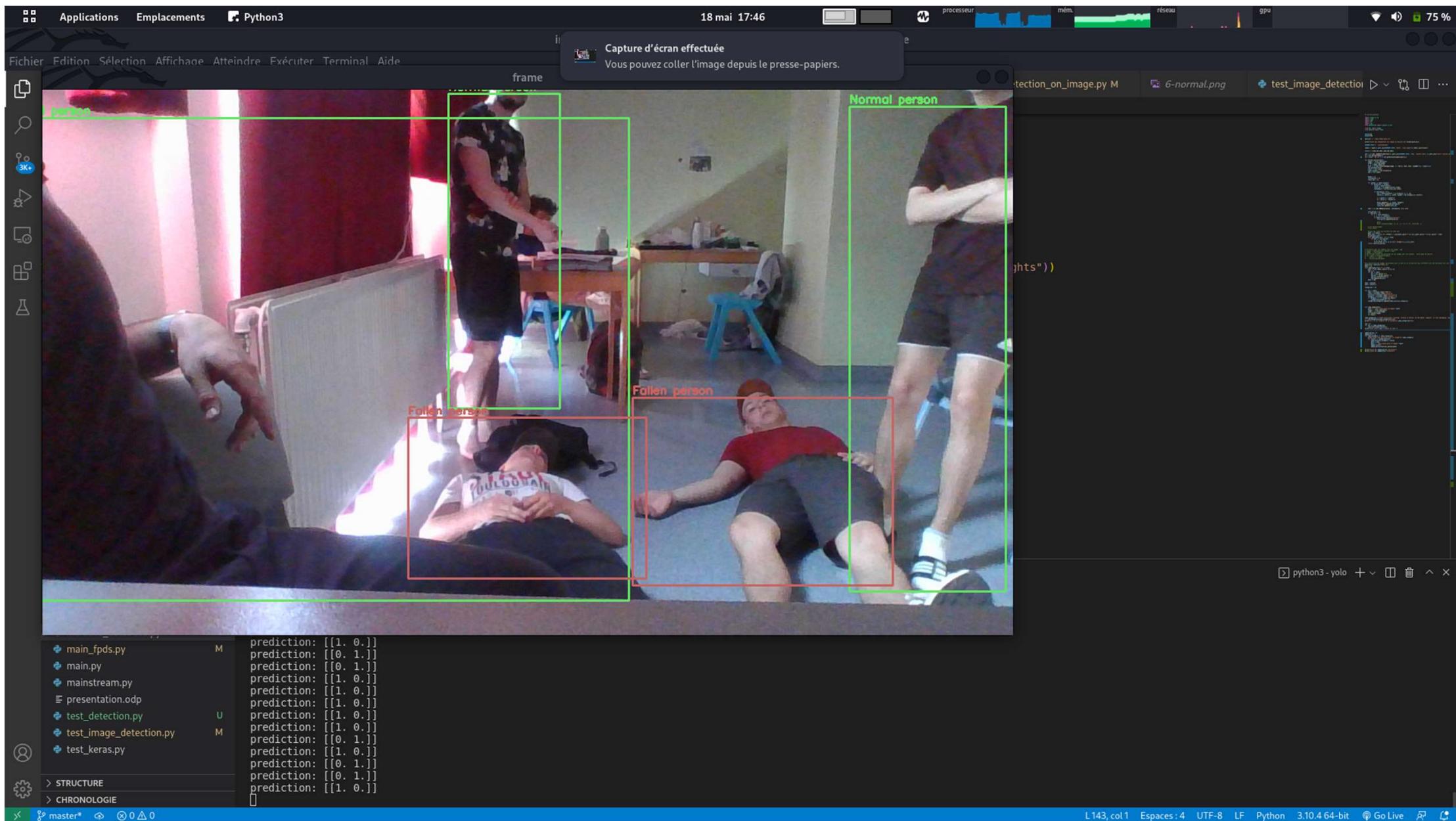
86.3% de classification correcte sur des données qu'il n'a jamais vu

Entrainement et test final du modèle

Résultats sur des images de test



Détection sur caméra



L'algorithme YOLO nous permet de détecter les personnes en temps réel

Le temps de calcul de l'algorithme est assez faible (240ms) donc on peut le faire fonctionner en direct sur une caméra

duration_frame:0.241, FPS:4.1

Détection sur caméra

Ajout d'un chrono sur la première chute détectée pour éviter de déclencher l'alarme sur des chutes qui ne sont pas grave.

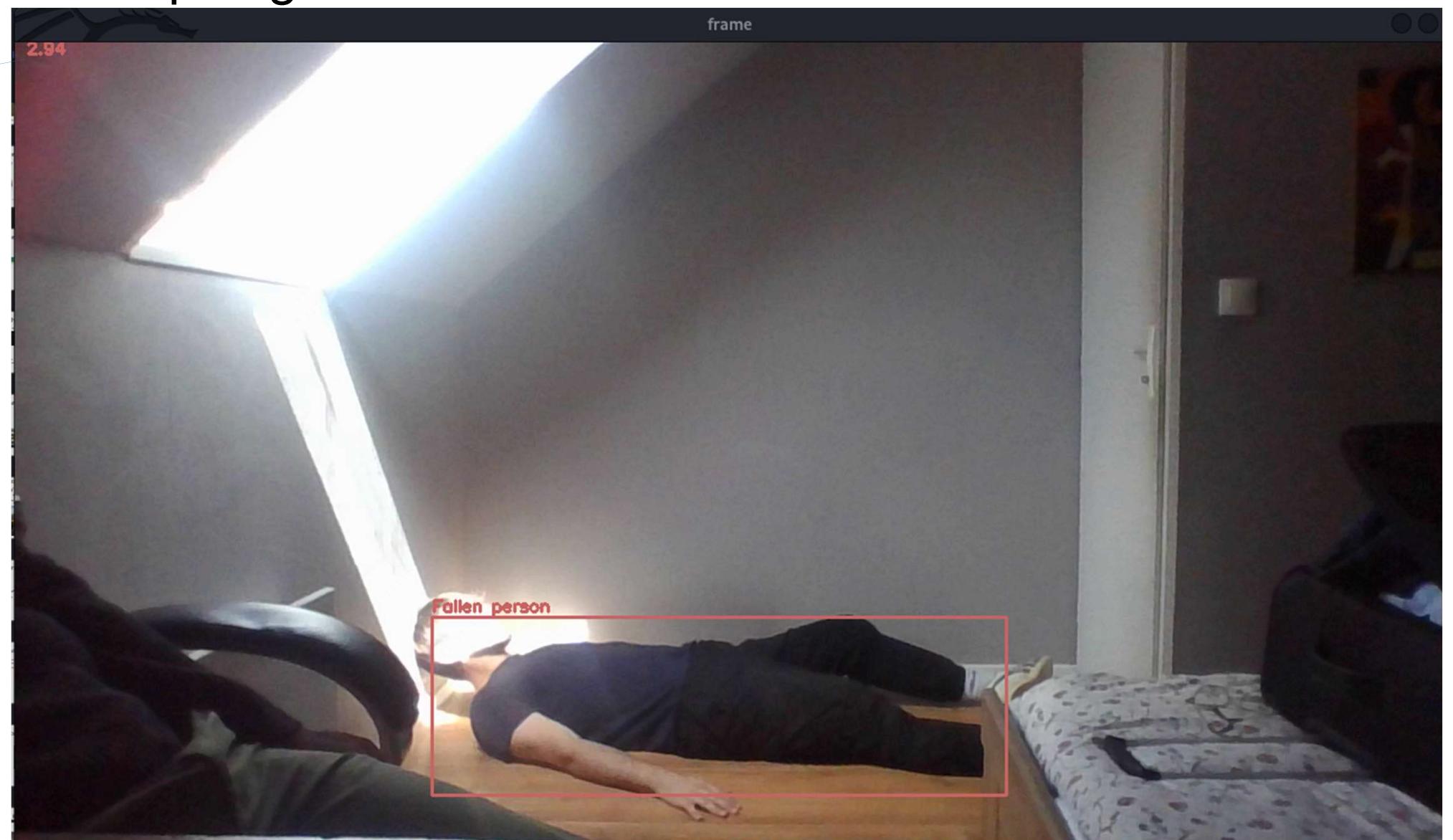
Temps de chute

```
#detection de chute qui dure dans le temps
if is_a_fall:
    if start_fall==None:
        start_fall = time.time()
else:
    start_fall = None

if start_fall!=None:
    fall_duration = time.time()-start_fall
    if fall_duration>ALARM_TIMEOUT:
        alarme()
```

Production personnelle

Si la durée de la chute dépasse le temps en seconde mis dans ALARM_TIMEOUT, on déclenche l'alarme.



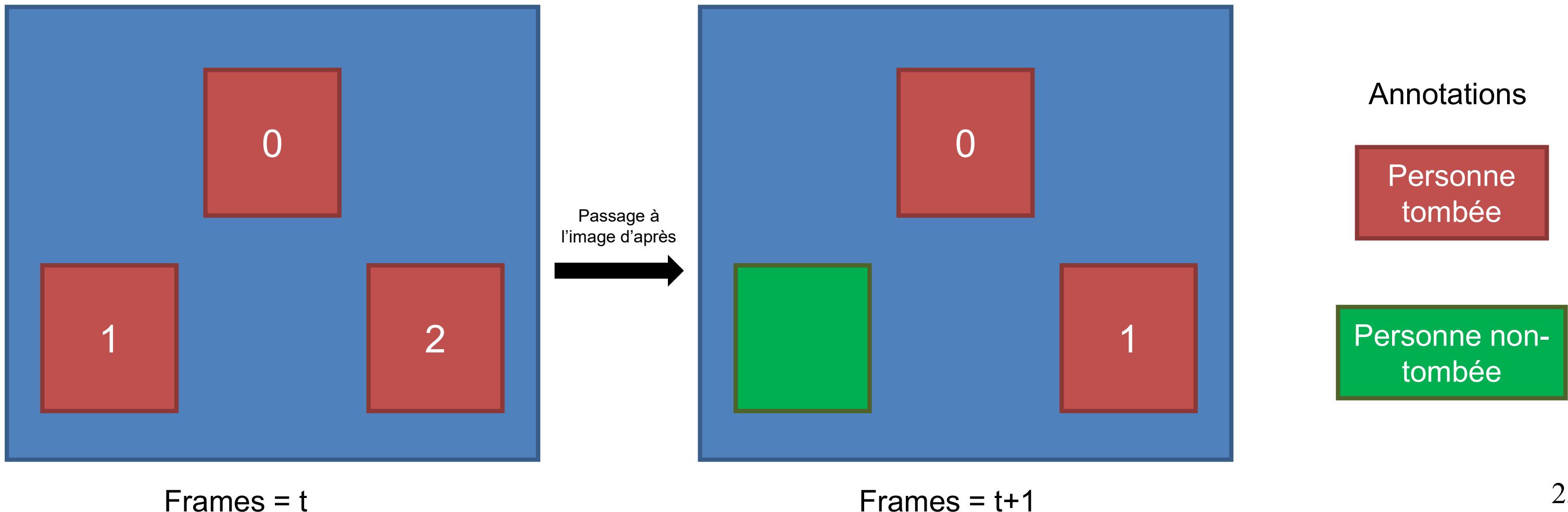
Détection sur caméra

Gérer plusieurs personnes en même temps

Problème : les personnes qui sont marquées tombées sont stockées dans une liste or cette liste n'assure en aucun cas le suivi d'une image à une autre.

Code python de la liste
fallen_person=[elt0,elt1,...]

Mise en situation du problème



Détection sur caméra

Gérer plusieurs personnes en même temps

- Hypothèse : Une personne tombée bouge très peu alors la distance du rectangle de détection d'une image à une autre reste faible

Algorithme de suivi de détection

```
#Suivi de la chute entre 2 images
#permutation
permutation_falls = []
if len(prev_falls)!=0:
    for box1 in start_falls:
        distances = []
        for i in range(len(prev_falls)):
            box2 = prev_falls[i]
            distances.append(distanceBox(box1,box2))
        #on récupère l'indice de la distance la plus petite
        min_i,min_value = getMinIndiceAndValueOfMin(distances)
        if min_value > MAX_DISTANCE:
            min_i = None
        permutation_falls.append(min_i)
else:
    permutation_falls = [None]*len(start_falls)
```

Annotations

i = indice du rectangle de détection sur la nouvelle image

j = permutation_falls[i]

j = indice du rectangle de détection sur l'ancienne image

En particulier si j==None c'est que le rectangle d'indice i est une nouvelle détection

Détection sur caméra

Explication de deux fonctions importantes

```
def distanceBox(box1,box2):
    #on récupère la position et la taille des rectangles
    x1,y1,xw,yh = box1
    w1,h1 = xw-x1,yh-y1
    x2,y2,xw,yh = box2
    w2,h2 = xw-x2,yh-y2

    #distance de manhattan sur la position
    sum = abs(x1-x2)+abs(y1-y2)

    #distance de manhattan sur la taille
    sum += abs(w1-w2) + abs(h1-h2)

    #on divise par 4 car on a l'addition de la distance sur 4 dimensions (x,y,w,h)
    return sum/4.0
```

Fonction qui calcule la différence de « ressemblance » entre deux rectangles
(si elle renvoie 0, les deux rectangles sont identiques)

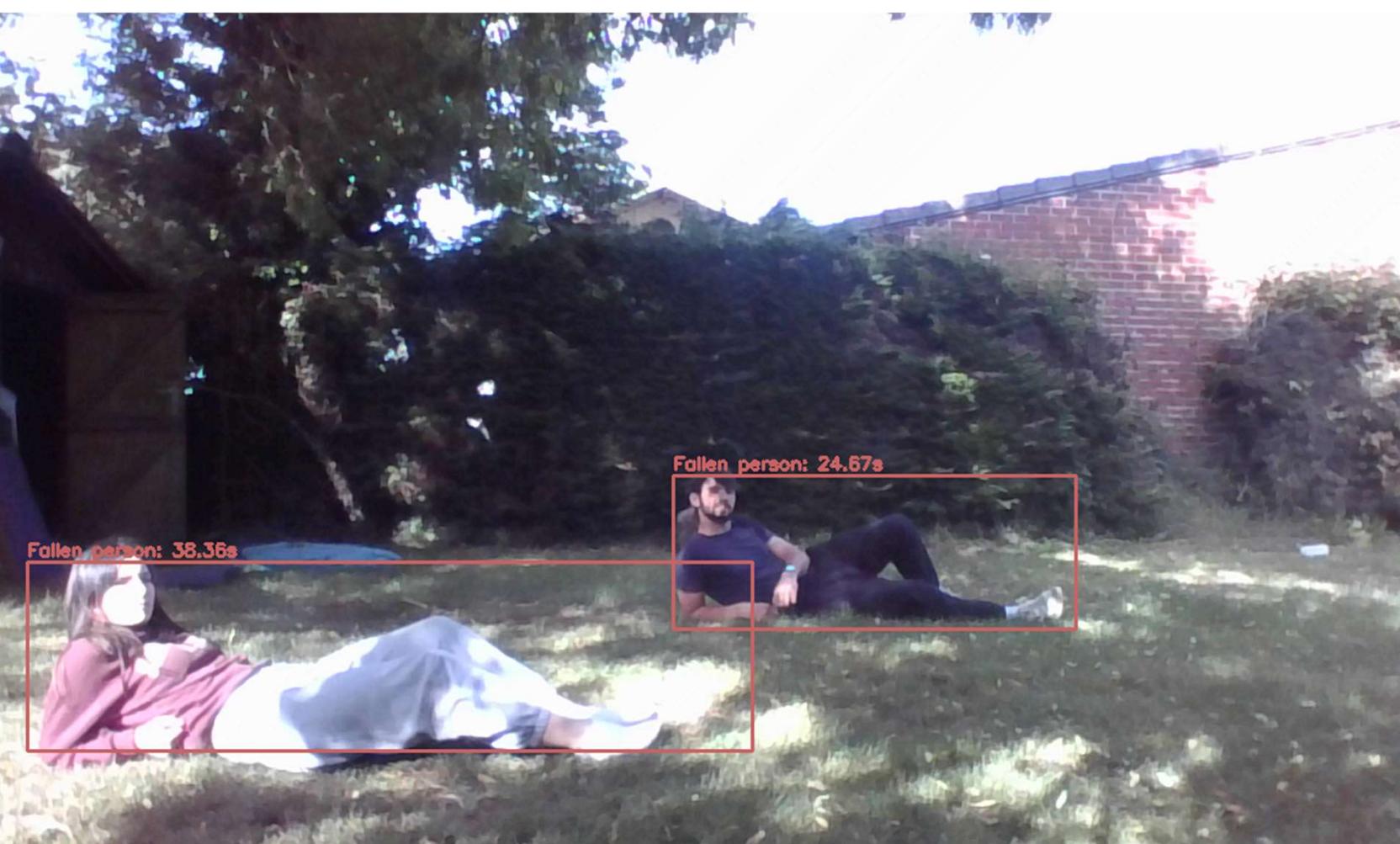
box1 et box2 sont de la forme [x,y,x+w,y+h] et la variable liste est une liste des différences d'un rectangle avec les rectangles de la nouvelle image

```
def getMinIndiceAndValueOfMin(liste):
    min_i = 0
    min_liste = liste[0]
    n = len(liste)
    for i in range(1,n):
        elt = liste[i]
        if min_liste>elt:
            min_i = i
            min_liste = elt
    return min_i,min_liste
```

Fonction qui renvoie l'indice de l'élément qui est le plus petit ainsi que l'élément en lui même

Détection sur caméra

Gérer plusieurs personnes en même temps



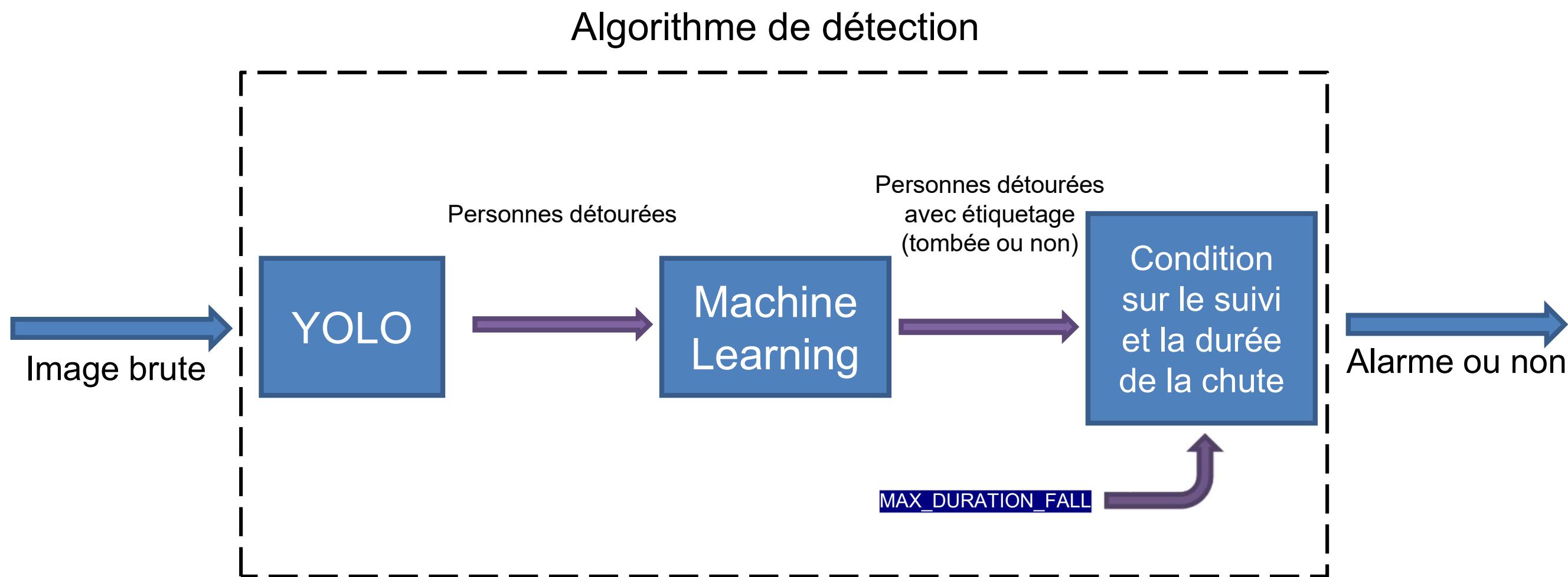
Temps de calcul de l'algorithme

```
2 fallen person  
Person fallen n°0 duration:38.36s  
Person fallen n°1 duration:24.67s  
duration frame:0.379, FPS:2.6
```

l'algorithme requiert beaucoup de ressource
Et a une complexité en $O(n^2)$ avec n le
nombre de personnes tombées

Détection sur caméra

Schéma du fonctionnement de l'algorithme final :



Annexes

1. main_fpds.py = Programme qui entraîne le réseau de neurone
2. test_image_detection.py = Programme qui affiche les images de test avec l'étiquetage qui est fait par notre réseau de neurone
3. detection.py = Programme qui lance la détection sur la caméra avec le chrono sur la première personne tombée
4. detection_timer.py = Programme qui lance la détection sur la caméra avec le chrono sur plusieurs personnes en même temps si besoin
5. increase_database.py = Programme qui augmente la base de données actuelle avec les données d'une autre base de données

Annexes

1. main_fpds.py

```
main_fpds.py M
main_fpds.py > ...

1 #!/usr/bin/python3
2 import numpy as np
3 from matplotlib import pyplot as plt
4 from PIL import Image
5 from pathlib import Path
6
7 import tensorflow as tf
8 from sklearn.model_selection import train_test_split
9 from keras.utils import np_utils
10
11 #taille de la matrice d'entrée du réseau de neurones de détection
12 WIDTH=150
13 HEIGHT=150
14
15
16 #le nombre maximale de donnée de fichier qu'il faut ajouter pour compléter la base de donnée de base
17 max_file = 1500
18
19
20 #fonction qui vient préparer l'image pour l'envoyer au réseau de neurones de détection des chutes
21 def load_image(path, box):
22
23     #on manipule un peu l'image
24     im=Image.open(path)
25     im = im.convert("L")
26     im=im.crop(box)
27     h,w = np.shape(np.array(im))
28
29     if w>h:
30         a,b = 1, h/w
31     else:
32         a,b = w/h, 1
33
34     #on change sa taille
35     im=im.resize((int(a*WIDTH),int(b*HEIGHT)))
36     im=np.array(im)
37     dtype = im.dtype
38
39     #on ajoute le padding si besoin
40     padding_w = int((WIDTH*(1-a))/2)
41     padding_h = int((HEIGHT*(1-b))/2)
42
43     #height padding
44     if padding_h!=0:
45         im = np.vstack([np.zeros((padding_h,WIDTH), dtype),im,np.zeros((padding_h,WIDTH), dtype)])
46         h,w = im.shape
47         if h!=HEIGHT:
48             im = np.vstack([im, np.zeros((HEIGHT-h, WIDTH), dtype)])
```

```
47
48     if padding_w!=0:
49         im = np.column_stack([np.zeros((HEIGHT,padding_w), dtype),im,np.zeros((HEIGHT,padding_w), dtype)])
50         h,w = im.shape
51         if w!=WIDTH:
52             im = np.column_stack([im, np.zeros((HEIGHT, WIDTH-w), dtype)])
53
54     #correction pixel par pixel
55     while im.shape != (HEIGHT,WIDTH):
56         h,w = im.shape
57         if w!=WIDTH:
58             im = np.column_stack([im, np.zeros((HEIGHT, 1), dtype)])
59         if h!=HEIGHT:
60             im = np.vstack([im, np.zeros((1, WIDTH), dtype)])
61
62     if im.shape != (HEIGHT,WIDTH):
63         print("ERRREEEUUURRRR")
64
65     return im
66
67 def load_path(i_label, i_content):
68     return contents[i_label][i_content],labels[i_label]
69
70 #lis les fichiers dans le dossier qui contient notre première base de données
71 print("on recupere les donnees")
72 print("recuperation 1")
73 images = Path("train/").rglob("*.png")
74 images = list(images)
75
76 #on les arrange aux hasards
77 images = np.array(images)
78 np.random.shuffle(images)
79
80 #on récupère les annotations qui vont avec les images
81 #ce sont des fichiers txt qui portent le même noms que les images
82 labels=[]
83 contents=[]
84 for path in images:
85     path = str(path)
86     sep = path.split("/")
87     assert len(sep)==3
88
89     text_path = "{}/{}/{}.txt".format(sep[0],sep[1],sep[2].split(".")[0])
90     #read annotations
91     lines = []
92     with open(text_path,"r") as file:
93         lines = file.read().split("\n")[:-1]
94
95     for line in lines:
```

Annexes

```
94     raw_data = line.split(" ")
95     assert len(raw_data) == 5
96     fall, left, right, top, bot = raw_data
97     box = (int(left), int(top), int(right), int(bot))
98     im = load_image(path, box)
99     #on ajoute les infos
100    contents.append(im)
101    fall = int(fall)
102    if fall<0:
103        fall = 0
104    labels.append(fall)
105
106 #2eme base de données
107 #recuperation de la base mpii_human_pose_v1 avec annotations
108 print("recuperation 2")
109 textes= Path("mpii_human_pose_v1/images/").rglob("*.txt")
110 textes = list(textes)
111 print("len recuper 2:", len(textes))
112 #on veille à ne pas dépasser un nombre trop grand de données en plus
113 if len(textes)>=max_file:
114     textes = textes[:max_file+1]
115
116 #on récupère les annotations en plus
117 #dans les fichiers txt portant le même nom sur le même principe que la première base de données
118 added_person = 0
119 for path in textes:
120     path = str(path)
121     sep = path.split("/")
122     image_path = "{}/{}.jpg".format("/".join(sep[:-1]),sep[-1].split(".")[0])
123     #read annotations
124     lines = []
125     with open(path,"r") as file:
126         lines = file.read().split("\n")[:-1]
127     for line in lines:
128         added_person+=1
129         raw_data = line.split(" ")
130         assert len(raw_data) == 5
131         fall, left, right, top, bot = raw_data
132         box = (int(left), int(right), int(top), int(bot))
133
134     #on prépare les données à envoyer au réseau de neurones
135     im = load_image(image_path, box)
136
137     #on ajoute les infos
138     contents.append(im)
139     fall = int(fall)
140     if fall<0:
141         fall = 0
142         labels.append(fall)
143     #on affiche le nombre de personne ajouté avec les fichiers en plus
144     #car il peut y avoir plus d'une personne par image
145     print("nombre de personne ajouté: ",added_person)
146
147     #on normalise les images pour que chaque valeur de chaque pixel soit compris entre 0 et 1
148     contents = np.array(contents, dtype="float32")/255.0
149     #on converti une valeur d'étiquetage en une liste d'étiquetage
150     labels = np_utils.to_categorical(labels, 2)
151     print("----\n",labels)
152     #on forme la bonne dimension pour notre liste
153     contents = np.expand_dims(contents, axis=3)
154     #on sépare notre base de données en 2, valeurs d'entraînement et valeurs de test qu'il n'aura jamais vu
155     x_train, x_test, y_train, y_test = train_test_split(contents, labels, test_size=0.2, shuffle=True)
156
157     #on affiche une valeur de test pour voir les données
158     print("affichage test")
159     plt.title(y_train[0])
160     plt.imshow(x_train[0])
161     plt.show()
162
163
164     #on regarde le nombre d'image que notre réseau de neurones aura pour s'entraîner avec
165     print("Taille :", len(x_train))
166
167     #on regarde la mixité des valeurs
168     a,b = 0,0
169     for fall in labels:
170         if fall[0]:
171             b+=1
172         else:
173             a+=1
174
175     #on affiche le pourcentage de présence de chaque catégorie
176     print("personne tombé: {} et non-tombé:{}.".format(a,b))
177     print("pourcentage de personne tombé:{} et non-tombé:{}.".format(a/len(labels),b/len(labels)))
178
179
180     #on crée enfin le modèle
181     print("creation du modele")
182
183     # model = Sequential()
184     # model.add(Conv2D(10, kernel_size=(10, 10), activation='relu', input_shape=(WIDTH,HEIGHT,1)))
185     # model.add(MaxPool2D(pool_size=(3, 3)))
186     # model.add(Dropout(0.25))
187     # model.add(Conv2D(10, kernel_size=(10, 10), activation='relu'))
```

Annexes

```
188 # model.add(Dropout(0.25))
189 # model.add(Conv2D(5, kernel_size=(6, 6), activation='relu'))
190 # model.add(Dropout(0.25))
191 # model.add(MaxPool2D(pool_size=(3, 3)))
192 # model.add(Dropout(0.25))
193 # model.add(Flatten())
194 # model.add(Dropout(0.4))
195 # model.add(Dense(10, activation='relu'))
196 # model.add(Dropout(0.4))
197 # model.add(Dense(2, activation='softmax'))
198
199 #réseau de neurone de convolution assez petit ~ 65,000 paramètres à faire varier
200 model = tf.keras.Sequential([
201     tf.keras.layers.Conv2D(30, kernel_size=(10,10), activation="relu", padding="same", input_shape=(WIDTH,HEIGHT,1)),
202     tf.keras.layers.MaxPool2D(pool_size=(3,3)),
203     tf.keras.layers.Conv2D(10, kernel_size=(10, 10), activation='relu'),
204     tf.keras.layers.MaxPool2D(pool_size=(3,3)),
205     tf.keras.layers.Flatten(),
206     tf.keras.layers.Dense(20,activation="relu"),
207     tf.keras.layers.Dropout(0.2),
208     tf.keras.layers.Dense(2,activation="softmax")
209 ])
210
211 print("entraînement")
212
213
214 #on configure l'entraînement avec des paramètres usuels d'entraînement
215 model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
216 history = model.fit(x_train, y_train, epochs = 3, verbose = 1)
217
218 #on affiche la courbe de précision de notre algorithme
219 print("affichage apprentissage")
220 plt.plot(history.history["accuracy"])
221 plt.show()
222
223
224 #on test sur des valeurs inconnus
225 print("test sur valeur inconnu")
226 evaluation = model.evaluate(x_test,y_test, verbose=0)
227 #erreur de classification
228 print("Erreur:", evaluation[0])
229 #erreur de précision de classification
230 print("Precision:", evaluation[1])
231
232 #on enregistre notre réseau de neurone entraîné dans un fichier pour le réutiliser dans d'autre programme
233 print("enregistrement du réseau de neuronne")
234
```

```
235     model.save("./weights/weights")
236
237
238 #on essaye sur des images aux hasards et on affiche les résultats
239 print("test sur une image dans la banque de test au hasard")
240 def get_prediction(im):
241     im = np.array([np.expand_dims(im, axis=2)])
242     predictions = model.predict(im)
243     fallen = np.argmax(predictions[0])
244     return predictions[0]
245
246 for x in x_test:
247     plt.title(get_prediction(x))
248     plt.imshow(x)
249     plt.show()
```

Annexes

2. test_image_detection.py

```
test_image_detection.py M
test_image_detection.py > ...
1 #!/usr/bin/python3
2
3 import numpy as np
4 import time
5 import cv2
6 import os
7
8 from matplotlib import pyplot as plt
9
10 from PIL import Image
11
12 import tensorflow as tf
13
14
15 #taille de la matrice d'entrée de mon réseau de neurones
16 WIDTH=150
17 HEIGHT=150
18
19 #creation du detecteur
20 print("creation du modèle")
21 model = tf.keras.models.load_model('./weights/weights')
22
23
24 #fonction qui vient détourner la personne sur une image et la compléter avec du noir pour qu'elle puisse entrer dans la matrice
25 #de taille WIDTH et HEIGHT
26 def add_padding(im, box):
27     #on la met en noir et blanc
28     im = im.convert("L")
29     #on la coupe sur le rectangle de detection
30     im=im.crop(box)
31     #on récupère la taille
32     h,w = np.shape(np.array(im))
33     #on complète en haut et en bas
34     if w>h:
35         a,b = 1, h/w
36     #on complète à droite et à gauche
37     else:
38         a,b = w/h, 1
39     #on la met à la bonne taille
40     im=im.resize((int(a*WIDTH),int(b*HEIGHT)))
41     im=np.array(im)
42     dtype = im.dtype
43     #on ajoute le padding si besoin
44     padding_w = int((WIDTH*(1-a))/2)
45     padding_h = int((HEIGHT*(1-b))/2
46
47     #height padding
48     if padding_h!=0:
```

```
49         im = np.vstack([np.zeros((padding_h,WIDTH), dtype),im,np.zeros((padding_h,WIDTH),dtype)])
50     h,w = im.shape
51     if h!=HEIGHT:
52         im = np.vstack([im, np.zeros((HEIGHT-h, WIDTH), dtype)])
53
54     if padding_w!=0:
55         im = np.column_stack([np.zeros((HEIGHT,padding_w), dtype),im,np.zeros((HEIGHT,padding_w), dtype)])
56     h,w = im.shape
57     if w!=WIDTH:
58         im = np.column_stack([im, np.zeros((HEIGHT, WIDTH-w), dtype)])
59
60     #petite erreur car pas exactement de taille HEIGHT,WIDTH
61     #correction pixel par pixel
62     while im.shape != (HEIGHT,WIDTH):
63         h,w = im.shape
64         if w!=WIDTH:
65             im = np.column_stack([im, np.zeros((HEIGHT, 1), dtype)])
66         if h!=HEIGHT:
67             im = np.vstack([im, np.zeros((1, WIDTH), dtype)])
68     #condition de test qui ne doit pas apparaitre
69     if im.shape != (HEIGHT,WIDTH):
70         print("ERRREEEUUURRRR")
71
72     return im
73
74 #fonction qui va donner l'étiquetage de l'image
75 def get_prediction(im, box):
76     im = add_padding(im,box)
77
78     #prediction
79     im = np.array([np.expand_dims(im, axis=2)])
80     predictions = model.predict(im)
81     fallen = np.argmax(predictions[0])
82     print("prediction:",predictions)
83     return int(fallen)
84
85 #chemin de l'algorithme yolo
86 DARKNET_PATH = './yolo/darknet'
87
88 #objet reconnaissable par YOLO
89 labels = open(os.path.join(DARKNET_PATH, "data", "coco.names")).read().splitlines()
90
91 #couleur des détection
92 colors = [(100,230,100),(200,0,0)]
93
94 #chargement du modèle de prédiction YOLO
95 net = cv2.dnn.readNetFromDarknet(os.path.join(DARKNET_PATH, "cfg", "yolov3.cfg"), os.path.join("yolo", "yolov3.weights"))
```

Annexes

```
96 ln = net.getLayerNames()
97 ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
98
99
100 #fonction d'affichage de l'image après passage par YOLO et par la prédiction de mon modèle
101 def display(path):
102     #préparation de l'image
103     image_prediction = Image.open(path)
104     image = np.array(image_prediction)
105     h, w = image.shape[:2]
106     blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True, crop=False)
107     net.setInput(blob)
108     layer_outputs = net.forward(ln)
109
110     #récupération de la détection de YOLO
111     boxes = []
112     confidences = []
113     class_ids = []
114     for output in layer_outputs:
115         for detection in output:
116             scores = detection[5:]
117             class_id = np.argmax(scores).item()
118             confidence = scores[class_id].item()
119
120             if confidence > 0.3:
121                 box = detection[0:4] * np.array([w, h, w, h])
122                 center_x, center_y, width, height = box.astype(int).tolist()
123
124                 #on prend la position à partir du centre et de la taille de l'objet
125                 x = center_x - width//2
126                 y = center_y - height//2
127
128                 #on ajoute la détection dans une liste
129                 boxes.append([x, y, width, height])
130                 confidences.append(confidence)
131                 class_ids.append(class_id)
132
133     #on récupère les indices des détection des objets les plus sûrs
134     idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.5)
135
136
137     if len(idxs) > 0:
138         for i in idxs.flatten():
139             x, y, w, h = boxes[i]
140             box = [x,y,x+w,y+h]
141             #on récupère uniquement la détection de personne
142             if labels[class_ids[i]]=="person":
```

```
143
144     #on utilise notre prédiction
145     fallen = get_prediction(image_prediction, box)
146
147     #on marque ensuite les personnes selon leurs classification
148     if fallen:
149         text="Fallen person"
150     else:
151         text="Normal person"
152
153     #on dessine sur l'image un rectangle de détection puis le texte
154     cv2.rectangle(image, (x, y), (x + w, y + h), colors[fallen], 2)
155     cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, colors[fallen], 2)
156
157     #on affiche l'image avec pyplot
158     plt.imshow(image)
159
160     #on affiche les deux images de test en même temps pour pouvoir effectuer des tests plus rapidement
161     fig = plt.figure(figsize=(10,5))
162     fig.add_subplot(1,2,1)
163     display("perso_data/0-normal.png")
164     fig.add_subplot(1,2,2)
165     display("perso_data/1-falling.png")
166
167     plt.show()
```

Annexes

3. detection.py

```
detection.py M >
yolo > detection.py > ...
1 #!/usr/bin/python3
2 import numpy as np
3 import time
4 import cv2
5 import os
6 from PIL import Image
7 import tensorflow as tf
8
9 #jouer des sons
10 import simpleaudio as sa
11
12 #on initialise des variables pour pouvoir jouer du son
13 wave_object = sa.WaveObject.from_wave_file('alarme.wav')
14 print('playing sound using simpleaudio')
15 play_file = None
16
17 #on fixe la condition de chute sur 5secondes pour déclencher l'alarme
18 ALARM_TIMEOUT = 5
19 start_fall = 0
20
21 #la fonction qui est lancé quand il y a une chute
22 def alarme():
23     global play_file
24     print("ALARMEEEEEEE")
25     if play_file == None or not play_file.is_playing():
26         play_file = wave_object.play()
27
28 #la taille de la matrice d'entrée pour le réseau de neurones de détection de chute
29 WIDTH=150
30 HEIGHT=150
31
32 #creation du detecteur
33 print("creation du modele")
34 model = tf.keras.models.load_model('../weights/weights')
35
36 #fonction qui vient formater notre image comme il faut pour pouvoir entrer dans notre réseau de neurones de détection
37 #chute
38 def add_padding(im, box):
39     #manipulation de couleur, de taille, ...
40     im = im.convert("L")
41     im=im.crop(box)
42     h,w = np.shape(np.array(im))
43     if w>h:
44         a,b = 1, h/w
45     else:
46         a,b = w/h, 1
47         im=im.resize((int(a*WIDTH),int(b*HEIGHT)))
48         im=np.array(im)
49         dtype = im.dtype
50         #on ajoute le padding si besoin
51         padding_w = int((WIDTH*(1-a))//2)
52         padding_h = int((HEIGHT*(1-b))//2)
53
54         #height padding
55         if padding_h!=0:
56             im = np.vstack([np.zeros((padding_h,WIDTH), dtype),im,np.zeros((padding_h,WIDTH), dtype)])
57             h,w = im.shape
58             if h!=HEIGHT:
59                 im = np.vstack([im, np.zeros((HEIGHT-h, WIDTH), dtype)])
60
61             if padding_w!=0:
62                 im = np.column_stack([np.zeros((HEIGHT,padding_w), dtype),im,np.zeros((HEIGHT,padding_w), dtype)])
63                 h,w = im.shape
64                 if w!=WIDTH:
65                     im = np.column_stack([im, np.zeros((HEIGHT, WIDTH-w), dtype)])
66
67             #correction pixel par pixel
68             while im.shape != (HEIGHT,WIDTH):
69                 h,w = im.shape
70                 if w!=WIDTH:
71                     im = np.column_stack([im, np.zeros((HEIGHT, 1), dtype)])
72                 if h!=HEIGHT:
73                     im = np.vstack([im, np.zeros((1, WIDTH), dtype)])
74
75             if im.shape != (HEIGHT,WIDTH):
76                 print("ERRREEEUUUURRRR")
77
78         return im
79
80 #fonction qui renvoie la prediction d'une chute où non d'une personne
81 def get_prediction(im, box):
82     im = add_padding(im,box)
83
84     #prediction
85     im = np.array([np.expand_dims(im, axis=2)])
86     predictions = model.predict(im)
87     fallen = np.argmax(predictions[0])
88     print("prediction:",predictions)
89
90     return int(fallen)
```

Annexes

```
96 #on récupere la caméra de mon ordinateur avec opencv2
97 cap = cv2.VideoCapture(0)
98 #cap = cv2.VideoCapture("http://192.168.1.68:8080/video/mjpeg")
99
100 #chemin d'accès à l'algorithme YOLO
101 DARKNET_PATH = 'darknet'
102
103 #objet que YOLO peut reconnaître
104 labels = open(os.path.join(DARKNET_PATH, "data", "coco.names")).read().splitlines()
105
106 #couleur des rectangles de détection possible
107 colors = [(100,230,100),(100,100,200)]
108
109
110 #chargement de l'algorithme YOLO
111 net = cv2.dnn.readNetFromDarknet(os.path.join(DARKNET_PATH, "cfg", "yolov3.cfg"), "yolov3.weights")
112 ln = net.getLayerNames()
113 ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
114
115
116 #on crée une boucle infini qui permet de gérer image par image de la caméra
117 while True:
118     #timer pour pouvoir calculer le temps d'exécution de notre algorithme
119     start_frame = time.time()
120     #video live
121     ret, image = cap.read()
122     if ret == False:
123         print("Erreur")
124         break
125
126     #on récupère l'image dans un autre format pour pouvoir la manipuler
127     image_prediction = Image.fromarray(image)
128     image = np.array(image_prediction)
129     image = np.array(image)
130
131     #on prépare l'image pour l'envoyer dans YOLO
132     h, w = image.shape[:2]
133     blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True, crop=False)
134
135     #on la passe dans l'algorithme YOLO
136     net.setInput(blob)
137     layer_outputs = net.forward(ln)
138
139
140     #on récupère chaque détection de YOLO
141     boxes = []
142     confidences = []
```

```
143 class_ids = []
144
145 for output in layer_outputs:
146     for detection in output:
147         scores = detection[5:]
148         class_id = np.argmax(scores).item()
149         confidence = scores[class_id].item()
150
151         if confidence > 0.3:
152             #information de détection donnée par YOLO
153             box = detection[0:4] * np.array([w, h, w, h])
154             center_x, center_y, width, height = box.astype(int).tolist()
155
156             #on récupère la position du rectangle avec la taille et la position du centre
157             x = center_x - width//2
158             y = center_y - height//2
159
160             #on ajoute les données dans des listes
161             boxes.append([x, y, width, height])
162             confidences.append(confidence)
163             class_ids.append(class_id)
164
165     #on filtre les détection qui sont en dessous de 60% de confiance
166     idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.6, 0.6)
167     is_a_fall = False
168     if len(idxs) > 0:
169         for i in idxs.flatten():
170             x, y, w, h = boxes[i]
171             box = [x,y,x+w,y+h]
172
173             #on récupère uniquement la détection de personne
174             if labels[class_ids[i]]=="person":
175                 #on prédit avec notre algorithme la chute ou non de la personne
176                 fallen = get_prediction(image_prediction, box)
177                 if fallen:
178                     text="Fallen person"
179                     is_a_fall = True
180                 else:
181                     text="Normal person"
182
183             #on dessine sur l'image pour annoter les chutes détecter
184             cv2.rectangle(image, (x, y), (x + w, y + h), colors[fallen], 2)
185             cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, colors[fallen], 2)
186
187             #detection de chute qui dure dans le temps
188             if is_a_fall:
189                 if start_fall==None:
190                     start_fall = time.time()
```

Annexes

```
190     else:
191         start_fall = None
192
193     if start_fall!=None:
194         fall_duration = time.time()-start_fall
195         if fall_duration>ALARM_TIMEOUT:
196             alarme()
197             cv2.putText(image, str(round(fall_duration,2)), (30, 30), cv2.FONT_HERSHEY_SIMPLEX,0.5, (colors[1]), 2)
198
199 #affichage de l'image finale
200 cv2.imshow("frame", image)
201 #calcul du temps de calcul et du nombre de FPS
202 duration_frame = time.time()-start_frame
203 print("duration_frame:{}, FPS:{}".format(round(duration_frame,3),round(1.0/duration_frame,1)))
204 if cv2.waitKey(1) & 0xFF == ord("q"):
205     break
206
```

Annexes

4. detection_timer.py

```
detection_timer.py ×
yolo > detection_timer.py > ...
1 #!/usr/bin/python3
2
3 import numpy as np
4 import time
5 import cv2
6 import os
7
8 from PIL import Image
9
10 import tensorflow as tf
11
12 #jouer des sons
13 import simpleaudio as sa
14
15 wave_object = sa.WaveObject.from_wave_file('alarme.wav')
16 print('playing sound using simpleaudio')
17 play_file = None
18
19 #variable pour la détection de plusieurs personnes
20 #condition sur le temps maximal de chute
21 ALARM_TIMEOUT = 5
22 MAX_DISTANCE = 60
23
24 start_falls = []
25 start_timer = []
26
27 #fonction à activer quand une chute est détectée
28 def alarme():
29     global play_file
30     print("ALARMEEEEEEE")
31     if play_file == None or not play_file.is_playing():
32         play_file = wave_object.play()
33
34 #fonction de calcul de différence entre 2 rectangles de détection
35 def distanceBox(box1,box2):
36     #on récupère la position et la taille des rectangles
37     x1,y1,xw,yh = box1
38     w1,h1 = xw-x1,yh-y1
39     x2,y2,xw,yh = box2
40     w2,h2 = xw-x2,yh-y2
41
42     #distance de manhattan sur la position
43     sum = abs(x1-x2)+abs(y1-y2)
44
45     #distance de manhattan sur la taille
46     sum += abs(w1-w2) + abs(h1-h2)
47
48     #on divise par 4 car on a l'addition de la distance sur 4 dimensions (x,y,w,h)
49     return sum/4.0
50
51
52 #fonction qui renvoie l'indice de la valeur minimum dans une liste ainsi que la valeur du min
53 def getMinIndiceAndValueOfMin(liste):
54     min_i = 0
55     min_liste = liste[0]
56     n = len(liste)
57     for i in range(1,n):
58         elt = liste[i]
59         if min_liste>elt:
60             min_i = i
61             min_liste = elt
62     return min_i,min_liste
63
64
65 #taille de la matrice d'entrée de notre réseau de neurones de détection de chute
66 WIDTH=150
67 HEIGHT=150
68
69 #création du détecteur
70 print("creation du modèle")
71 model = tf.keras.models.load_model('../weights/weights')
72
73 #préparation de l'image pour qu'elle puisse entrer dans le réseau de neurones de détection de chute
74 def add_padding(im, box):
75     #manipulation de couleur, de taille ....
76     im = im.convert("L")
77     im=im.crop(box)
78     h,w = np.shape(np.array(im))
79     if w>h:
80         a,b = 1, h/w
81     else:
82         a,b = w/h, 1
83     im=im.resize((int(a*WIDTH),int(b*HEIGHT)))
84     im=np.array(im)
85     dtype = im.dtype
86     #on ajoute le padding si besoin
87     padding_w = int((WIDTH*(1-a))/2)
88     padding_h = int((HEIGHT*(1-b))/2)
89
90     #height padding
91     if padding_h!=0:
92         im = np.vstack([np.zeros((padding_h,WIDTH), dtype),im,np.zeros((padding_h,WIDTH), dtype)])
93
94
95
```

Annexes

```
96     h,w = im.shape
97     if h!=HEIGHT:
98         |   im = np.vstack([im, np.zeros((HEIGHT-h, WIDTH), dtype)])
99
100    if padding_w!=0:
101        im = np.column_stack([np.zeros((HEIGHT,padding_w), dtype),im,np.zeros((HEIGHT,padding_w), dtype)])
102    h,w = im.shape
103    if w!=WIDTH:
104        |   im = np.column_stack([im, np.zeros((HEIGHT, WIDTH-w), dtype)])
105
106    #correction pixel par pixel
107    while im.shape != (HEIGHT,WIDTH):
108        h,w = im.shape
109        if w!=WIDTH:
110            |   im = np.column_stack([im, np.zeros((HEIGHT, 1), dtype)])
111            if h!=HEIGHT:
112                |   im = np.vstack([im, np.zeros((1, WIDTH), dtype)])
113    if im.shape != (HEIGHT,WIDTH):
114        print("ERRREEEUUURRRR")
115
116    return im
117
118 #fonction qui fait passer l'image demande dans notre algorithme et qui renvoie la prédition de chute ou non
119 def get_prediction(im, box):
120     im = add_padding(im,box)
121
122     #prediction
123     im = np.array([np.expand_dims(im, axis=2)])
124     predictions = model.predict(im)
125     fallen = np.argmax(predictions[0])
126     #print("prediction:",predictions)
127     return int(fallen)
128
129 #on recupere la caméra de l'ordinateur
130 cap = cv2.VideoCapture(0)
131 #cap = cv2.VideoCapture("http://192.168.1.68:8080/video/mjpeg")
132
133 #chemin d'accès à l'algorithme YOLO
134 DARKNET_PATH = 'darknet'
135
136 #objet reconnaissable par YOLO
137 labels = open(os.path.join(DARKNET_PATH, "data", "coco.names")).read().splitlines()
138
139 #couleur des rectangles de détection de chute
140 colors = [(100,230,100),(100,100,200)]
141
142 #chargement de l'algorithme YOLO
143 net = cv2.dnn.readNetFromDarknet(os.path.join(DARKNET_PATH, "cfg", "yolov3.cfg"), "yolov3.weights")
144 ln = net.getLayerNames()
145 ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
146
147 #boucle infini qui permet de gérer le flux vidéo
148 while True:
149     #chrono qui permet de calculer la durée de calcul de notre algorithme sur une image
150     start_frame = time.time()
151     #video live
152     ret, image = cap.read()
153     if ret == False:
154         print("Erreur")
155         break
156
157     #récupération de l'image dans une autre forme pour pouvoir effectuer des manipulations dessus
158     image_prediction = Image.fromarray(image)
159     image = np.array(image_prediction)
160     image = np.array(image)
161
162     #préparation de l'image à envoyer à YOLO
163     h, w = image.shape[:2]
164     blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True, crop=False)
165     #envoie de l'image à YOLO
166     net.setInput(blob)
167     layer_outputs = net.forward(ln)
168
169
170     #on récupère les informations que YOLO nous renvoie
171     boxes = []
172     confidences = []
173     class_ids = []
174
175     for output in layer_outputs:
176         for detection in output:
177             scores = detection[5:]
178             class_id = np.argmax(scores).item()
179             confidence = scores[class_id].item()
180
181             if confidence > 0.3:
182                 #rectangle de détection
183                 box = detection[0:4] * np.array([w, h, w, h])
184                 center_x, center_y, width, height = box.astype(int).tolist()
185
186                 #on récupère la position avec la position du centre ainsi que la taille du rectangle
187                 #de détection
188                 x = center_x - width//2
189                 y = center_y - height//2
```

Annexes

```
190
191     #ajout des données formatées dans des listes
192     boxes.append([x, y, width, height])
193     confidences.append(confidence)
194     class_ids.append(class_id)
195
196     #on récupère les rectangles de détection au dessus de 60% de confiance
197     idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.6, 0.6)
198     prev_falls = start_falls
199     start_falls = []
200     if len(idxs) > 0:
201         for i in idxs.flatten():
202             x, y, w, h = boxes[i]
203             box = [x,y,x+w,y+h]
204             #on récupère uniquement la détection de personne
205             if labels[class_ids[i]]=="person":
206                 #étiquetage de la personne avec
207                 #notre réseau de neurones
208                 fallen = get_prediction(image_prediction, box)
209                 if fallen:
210                     #suivi plus complexe avant affichage de la chute
211                     start_falls.append(box)
212                 else:
213                     #affichage de la personne si elle n'est pas tombé
214                     cv2.rectangle(image, (x, y), (x + w, y + h), colors[fallen], 2)
215                     cv2.putText(image, "Normal person", (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,0.5, colors[fallen], 2)
216
217     #Suivi de la chute entre 2 images
218     #permutation
219     permutation_falls = []
220     if len(prev_falls)!=0:
221         for box1 in start_falls:
222             distances = []
223             for i in range(len(prev_falls)):
224                 box2 = prev_falls[i]
225                 distances.append(distanceBox(box1,box2))
226             #on récupère l'indice de la distance la plus petite
227             min_i,min_value = getMinIndiceAndValueOfMin(distances)
228             if min_value > MAX_DISTANCE:
229                 min_i = None
230                 permutation_falls.append(min_i)
231             else:
232                 permutation_falls = [None]*len(start_falls)
233
234     #on effectue les permutations comme il se doit
235     prev_start_timer = start_timer
236
237     start_timer = []
238     new_falls = []
239     #permutation de i0 (indice dans start_falls) vers il(indice dans prev_falls) si il != None
240     for i0,il in enumerate(permutation_falls):
241         #si on a aucune permutation avec l'ancienne liste c'est que c'est une nouvelle detection
242         if il==None:
243             start_timer.append(time.time())
244             new_falls.append(start_falls[i0])
245         #on a une permutation
246         else:
247             #on récupère l'ancienne
248             start_timer.append(prev_start_timer[il])
249             new_falls.append(start_falls[i0])
250
251     start_falls = new_falls
252
253     #on dessine les rectangles avec les informations récupérées
254     nbr_fallen = len(start_falls)
255     print("{} fallen person".format(nbr_fallen))
256     for i in range(nbr_fallen):
257         x1, y1, x2, y2 = start_falls[i]
258         timer = start_timer[i]
259         duration = time.time()-timer
260         cv2.rectangle(image, (x1, y1), (x2, y2), colors[1], 2)
261         if duration > ALARM_TIMEOUT:
262             alarme()
263             cv2.putText(image, "Fallen person: {}s".format(round(duration,2)), (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX,0.5, colors[1], 2)
264             #affichage du nombre de personne tombée et la durée
265             print("Person fallen n°{} duration:{}s".format(i,round(duration,2)))
266
267     duration_frame = time.time()-start_frame
268     cv2.imshow("frame", image)
269     print("duration_frame:{}, FPS:{}".format(round(duration_frame,3),round(1.0/duration_frame,1)))
270     if cv2.waitKey(1) & 0xFF == ord("q"):
271         break
272
273
```

Annexes

5. increase_database.py

```
#!/usr/bin/python3
import numpy as np
import cv2
import os
from matplotlib import pyplot as plt
from PIL import Image
#taille de la matrice d'entrée de notre réseau de neurone
WIDTH=150
HEIGHT=150
#chemin où se trouve les images non-annotées
path_dir = "./mpii_human_pose_v1/"
print("Ajout des metadonnées aux images du dossier {}".format(path_dir))
#chemin où se trouve l'algorithme YOLO
DARKNET_PATH = './yolo/darknet'
#éléments reconnaissable par YOLO
labels = open(os.path.join(DARKNET_PATH, "data", "coco.names")).read().splitlines()
#couleur des rectangles de detection (r,g,b)
colors = [(100,230,100),(100,100,200)]
#récupération de l'algorithme YOLO
net = cv2.dnn.readNetFromDarknet(os.path.join(DARKNET_PATH, "cfg", "yolov3.cfg"), os.path.join("yolo","yolov3.weights"))
#on récupère les dernières couches
ln = net.getLayerNames()
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
liste = []
def extract_person(path):
    image = Image.open(path)
    image = np.array(image)
    h, w = image.shape[:2]
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True, crop=False)
    net.setInput(blob)
    layer_outputs = net.forward(ln)
    boxes = []
    confidences = []
    class_ids = []
    for output in layer_outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores).item()
            confidence = scores[class_id].item()
            if confidence > 0.3:
                box = detection[0:4] * np.array([w, h, w, h])
                center_x, center_y, width, height = box.astype(int).tolist()
                x = center_x - width//2
                y = center_y - height//2
                boxes.append([x, y, width, height])
                confidences.append(confidence)
                class_ids.append(class_id)
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.6, 0.6)
    list_person = []
    if len(idxs) > 0:
        for i in idxs.flatten():
            x, y, w, h = boxes[i]
            if labels[class_ids[i]]=="person":
                list_person.append(boxes[i])
    #test
    #cv2.rectangle(image, (x, y), (x + w, y + h), colors[0], 2)
    # plt.imshow(image)
    # plt.show()
    #ajout des infos aux fichiers du même nom
    path = str(path)
    path_text = "{}.txt".format("/".join(path.split("/")[:-1]),(path.split("/")[-1]).split(".")[0])
    print(path_text)
    with open(path_text, "w") as file:
        for box in list_person:
            x, y, w, h = box
            file.write("{} {} {} {} {} {} \n".format(-1,x,y,x+w,y+h))
```

Annexes

```
94     global liste
95     liste.append(path_text)
96     return len(list_person)
97
98
99
100 # #récupère tous les chemins vers les images .jpg
101 # images = Path(path_dir).rglob("*.jpg")
102 # images = list(images)
103 # #on extrait toutes les personnes de ces images pour les ajouter à notre base de données
104 # for i,path in enumerate(images):
105 #     print(i,".",path)
106 #     extract_person(path)
107
108
109 #on sélectionne des images spécifiques dans la base qu'on ne pourrait pas confondre avec des personnes qui sont tombées
110 path_file = path_dir+"data.csv"
111 data = []
112 with open(path_file,"r") as file:
113     data = file.read().split("\n")[:-1]
114     tmp = []
115     for elt in data:
116         elt = elt.split(",")
117         new_elt = elt[0].split(",")
118         new_elt.extend(elt[1:])
119         tmp.append(new_elt)
120     data = tmp
121
122 keys = data[0]
123 data = data[1:]
124
125 categories = {}
126 liste_images = []
127 for elt in data:
128     name = elt[keys.index("NAME")]
129     scale = elt[keys.index('Scale')]
130     activity = elt[keys.index('Activity')]
131     category = elt[keys.index('Category')]
132     liste_images.append(name)
133     if category not in categories.keys():
134         categories[category] = []
135     categories[category].append([name,activity,category])
136
137 def load_image(path):
138     path = "./mpii_human_pose_v1/images/"+path
139     image = Image.open(path)
140     image = np.array(image)

141     plt.imshow(image)
142     plt.show()
143
144     #name_categories = ["home activities","sitting, talking in person, on the phone, computer, or text messaging, light effort","standing, talking in church","postal carri
145     name_categories = list(categories.keys())
146     n = len(categories[name_categories[-1]])
147     print("il y a",n,"éléments de la catégorie",name_categories[-1])
148
149     sum = 0
150     for elt in name_categories:
151         sum+=len(categories[elt])
152     print("Nous avons",sum,"éléments en tout.")
153
154     added_person = 0
155     added_file = 0
156
157     for name in liste_images:
158         path = name
159         path = "./mpii_human_pose_v1/images/"+path
160         if path not in liste:
161             added_file+=1
162             added_person+=extract_person(path)
163
164
165         print("Ajout de",added_person,"personnes")
166         print("Ajout de",added_file,"fichiers")
167
168
169
170
171     liste = list(set(liste))
172     print(len(liste))
```