

PAO DEEP LEARNING

Suivi des piétons basé sur CNN avec Keras

08/01/2019

Rodrigo Viano

Table des matières

1	Introduction	2
2	Contexte	2
3	Différentes approches	2
4	Approche utilisée	2
4.1	Qu'est-ce qu'un CNN est ?	2
4.2	Qu'est-ce que OpenCv est ?	2
4.3	Qu'est-ce que Keras est ?	3
5	Mise en oeuvre	3
6	Différents modèles	3
6.1	CNN 150x150 avec Kitti	3
6.1.1	Dataset	3
6.1.2	Structure du réseau	4
6.2	CNN 18x36 avec Daimler	4
6.2.1	Dataset	4
6.2.2	Pertes et précision	5
6.2.3	Structure du réseau	5
6.3	CNN 64x64 avec Daimler	6
6.3.1	Dataset	6
6.3.2	Pertes et précision	7
6.3.3	Structure du réseau	8
6.4	Code pour construire les réseaux de neurones	9
6.4.1	CNN 150x150	9
6.4.2	CNN 18x36	9
6.4.3	CNN 64x64	10
6.5	Entrainement des CNN	10
6.5.1	CNN 150x150	10
6.5.2	CNN 18x36	11
6.5.3	CNN 64x64	12
7	Suivi des piétons	13
8	Résultats	16
8.1	Capture de la vidéo originale	16
8.2	Capture de la vidéo après le 1er modèle	16
8.3	Capture de la vidéo après le 2eme modèle	17
8.4	Capture de la vidéo après le 3eme modèle	17

1 Introduction

D e nombreuses personnes meurent chaque année dans des accidents de la route. Et la raison de ces accidents c'est presque toujours le manque d'attention du côté du conducteur. Pendant ces dernières années il y a eu un grand intérêt dans le développement de systèmes de détection des piétons qui pourraient aider à réduire l'ampleur et l'impact de ces accidents. La plupart des systèmes proposés utilisent une caméra comme capteur, car les caméras peuvent fournir la résolution nécessaire pour une classification et une mesure de position précises. Les caméras peuvent également être partagées avec d'autres sous-systèmes d'assistance à la sécurité dans la voiture, tels qu'un système d'assistance de maintien de voie, améliorant ainsi le rapport prix / avantages de la caméra.

2 Contexte

C onstruire un modèle de réseaux de neurones capable de détecter et de suivre des piétons. Ce problème peut être abordé avec différentes approches

3 Différentes approches

- CNN avec sliding windows [7]
- CNN avec OpenCv
- Recurrent Convolutional Neural Network (R-CNN) [4]
- Fast Recurrent Convolutional Neural Network (Fast R-CNN) [4]
- Faster Recurrent Convolutional Neural Network (Faster R-CNN) [4]
- You Only Look Once (YOLO) [4] [10]

4 Approche utilisée

D ans ce travail j'ai utilisé un CNN avec OpenCv. Le CNN je l'ai utilisé pour faire un « image classifier » et OpenCv pour le traitement des images. Tout cela a été fait en Python et Keras pour le CNN

4.1 Qu'est-ce qu'un CNN est ?

E n deep learning, un réseau de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN ou ConvNet pour Convolutional Neural Networks) est un type de réseau de neurones artificiels, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont rangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilement multicouche de perceptrons¹, dont le but est de prétraiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

4.2 Qu'est-ce que OpenCv est ?

O penCV (pour Open Computer Vision) [8] est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel.

1. Le perceptron est un algorithme d'apprentissage supervisé de classificateurs binaires (c'est-à-dire séparant deux classes)

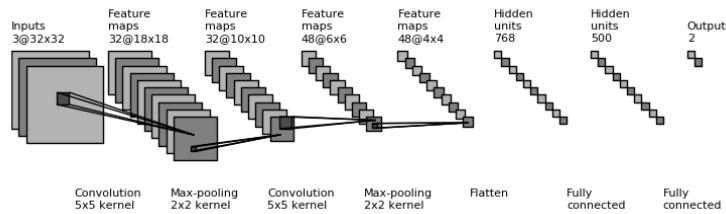


FIGURE 1: Example d'un CNN

4.3 Qu'est-ce que Keras est ?

Keras [3] est une bibliothèque de réseau de neurones open source écrite en Python. Il est capable de fonctionner sur TensorFlow [2], Microsoft Cognitive Toolkit ou Theano. Elle a été conçue pour permettre une expérimentation rapide avec des réseaux de neurones profonds.

5 Mise en oeuvre

Pour ce travail j'ai utilisé différents modèles de CNN entraînés avec différentes images de tailles différentes. Une fois le modèle entraîné, j'ai utilisé OpenCv pour lire une video (frame par frame), soustraire le fond du frame actuel et retrouver les contours des objects sur l'image. Une fois cela fait je modifie la taille de l'image pour pouvoir utiliser mon resseau de neurones et identifier si dans l'image il y a un piéton ou non. Dans le cas positif je dessine un bounding box autour du piéton identifié.

6 Différents modèles

J'ai entraîné 3 resseau de neurones différents avec des différentes images de différentes tailles. Chaque groupe d'images a été divisé en TrainingData et ValidationData (ValidationData = 10 % ou 20% du TrainingData).

6.1 CNN 150x150 avec kitti

Le premier CNN je l'ai entraîné avec des images du dataset KITTI [6]. Ce dataset est composé de 7481 images de 1242x375 pixels qu'ont été transformées de RGB à noir et blanc. Dans le ValidationData j'ai mis 748 images et dans le TrainingData j'ai mis 6733. Finalement j'ai séparé chaque set de données en Pedestrian et NonPedestrian. Puis avant d'entrainer le resseau j'ai modifié la taille des images de 1242x375 pixels à 150x150 pixels, de cette façon l'entraînement se fait beaucoup plus rapide.

6.1.1 Dataset

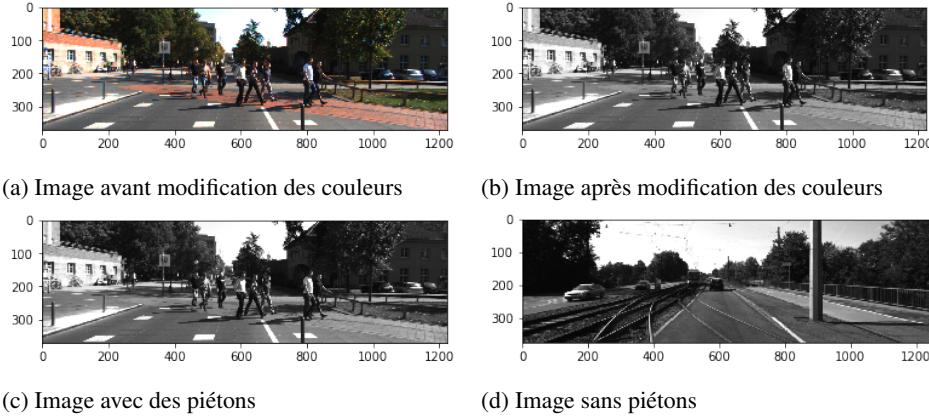


FIGURE 2: Images du dataset Kitti

6.1.2 Structure du resseau

```

File Edit View Bookmarks Settings Help
PyD: python - Konsole
Loading weights from kitti/savedweightsvgp-Kitti.h5
Loaded model and weights from disk
Layer (Type)          Output Shape      Param #
=====
conv2d_1 (Conv2D)    (None, 148, 148, 32) 896
activation_1 (Activation) (None, 148, 148, 32) 0
max_pooling2d_1 (MaxPooling2D) (None, 74, 74, 32) 0
conv2d_2 (Conv2D)    (None, 72, 72, 32) 9248
activation_2 (Activation) (None, 72, 72, 32) 0
max_pooling2d_2 (MaxPooling2D) (None, 36, 36, 32) 0
conv2d_3 (Conv2D)    (None, 34, 34, 64) 18496
activation_3 (Activation) (None, 34, 34, 64) 0
max_pooling2d_3 (MaxPooling2D) (None, 17, 17, 64) 0
flatten_1 (Flatten)  (None, 18496) 0
dense_1 (Dense)     (None, 64) 1183808
activation_4 (Activation) (None, 64) 0
dropout_1 (Dropout) (None, 64) 0
dense_2 (Dense)     (None, 1) 65
activation_5 (Activation) (None, 1) 0
=====
Total params: 1,212,513
Trainable params: 1,212,513
Non-trainable params: 0

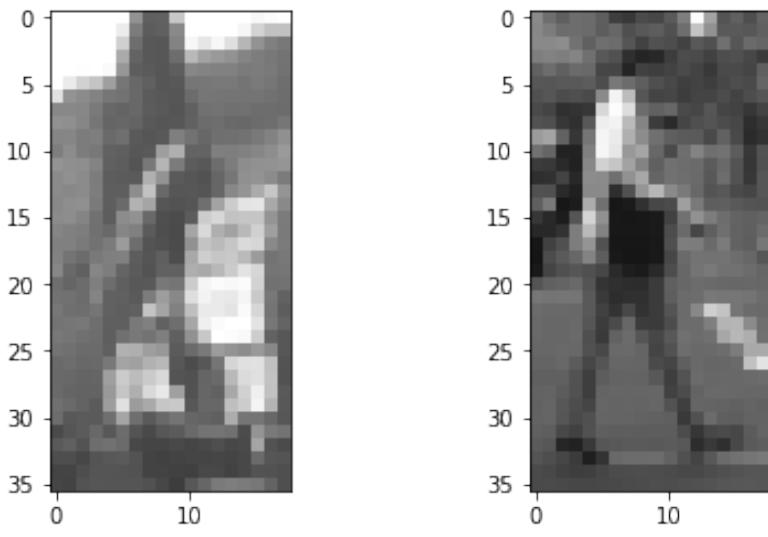
```

FIGURE 3: Structure du resseau

6.2 CNN 18x36 avec Daimler

Pour le 2eme CNN j'ai utilisé des images du dataset Daimler [5]. Ce dataset est composé de 7840 images dans le TrainingData et 1960 images dans le ValidationData. Dans ce cas, les images ont été aussi séparées entre Pedestrian et NonPedestrian dans chacun des sets des données. Les images de ce dataset sont des images de 18X36 pixels en noir et blanc (Figure 4). Les images n'ont pas été modifiées et avec ce modèle j'ai eu une précision de 80% environ (Figure 5)

6.2.1 Dataset



(a) Image sans piétons

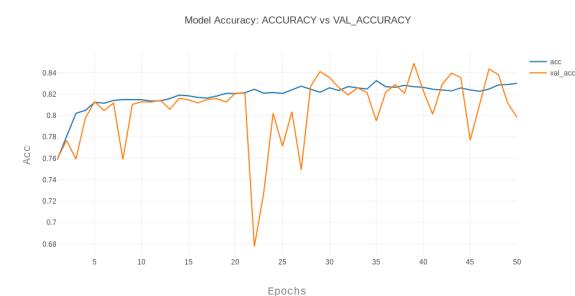
(b) Image avec des piétons

FIGURE 4: Images du dataset Daimler 18x36

6.2.2 Pertes et précision



(a) Perte



(b) Precision

FIGURE 5: Pertes et précision

6.2.3 Structure du resseau

```

File Edit View Bookmarks Settings Help
p_parallelism_threads for best performance.
Loading model from ped-18x36/modelcnn-18x36.json
Loading weights from ped-18x36/weights-18x36.h5
Loaded model and weights from disk

Layer (type)          Output Shape         Param #
=====
conv2d_1 (Conv2D)      (None, 18, 36, 16)    1216
batch_normalization_1 (Batch Normalization) (None, 18, 36, 16)    64
leaky_re_lu_1 (LeakyReLU) (None, 18, 36, 16)    0
max_pooling2d_1 (MaxPooling2D) (None, 9, 18, 16)    0
conv2d_2 (Conv2D)      (None, 9, 18, 32)       12832
batch_normalization_2 (Batch Normalization) (None, 9, 18, 32)    128
leaky_re_lu_2 (LeakyReLU) (None, 9, 18, 32)       0
max_pooling2d_2 (MaxPooling2D) (None, 4, 9, 32)     0
conv2d_3 (Conv2D)      (None, 4, 9, 64)        18496
batch_normalization_3 (Batch Normalization) (None, 4, 9, 64)    256
leaky_re_lu_3 (LeakyReLU) (None, 4, 9, 64)       0
max_pooling2d_3 (MaxPooling2D) (None, 2, 4, 64)     0
flatten_1 (Flatten)    (None, 512)           0
dense_1 (Dense)        (None, 128)          65664
activation_1 (Activation) (None, 128)          0
dropout_1 (Dropout)    (None, 128)           0
dense_2 (Dense)        (None, 1)            129
activation_2 (Activation) (None, 1)            0
=====
Total params: 98,785
Trainable params: 98,561
Non-trainable params: 224

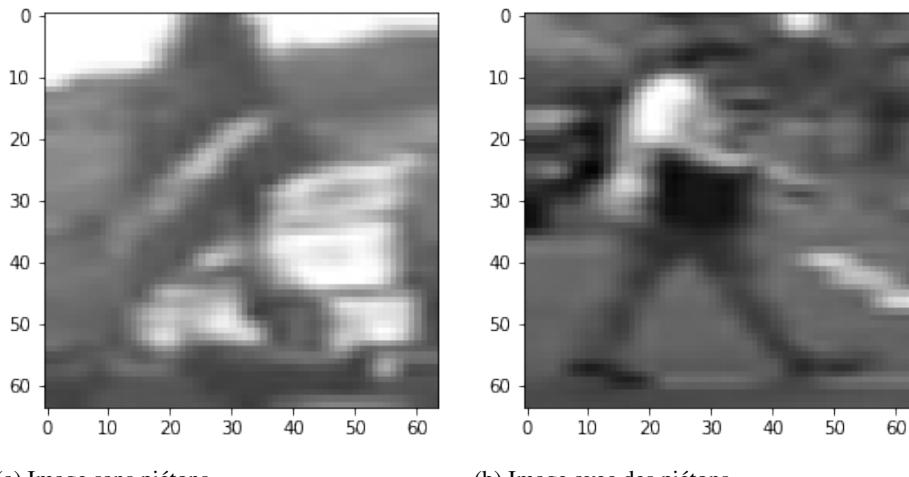
```

FIGURE 6: Structure du resseau

6.3 CNN 64x64 avec Daimler

Pour le 3eme et dernier CNN j'ai utilisé les mêmes images que pour le 2eme CNN mais avec une taille de 64X64 pixels (Figure 7) . Avec ce modèle j'ai eu une précision du 89% environ (Figure 8) .

6.3.1 Dataset



(a) Image sans piétons

(b) Image avec des piétons

FIGURE 7: Images du dataset Daimler 64x64 px

6.3.2 Pertes et précision

```

File Edit View Bookmarks Settings Help
245/245 [=====] - 215s 877ms/step - loss: 0.1432 - acc: 0.8026 - val_loss: 0.1275 - val_acc: 0.8526
Epoch 32/50
245/245 [=====] - 220s 899ms/step - loss: 0.1422 - acc: 0.8075 - val_loss: 0.1186 - val_acc: 0.8616
Epoch 33/50
245/245 [=====] - 228s 930ms/step - loss: 0.1384 - acc: 0.8091 - val_loss: 0.1185 - val_acc: 0.8700
Epoch 34/50
245/245 [=====] - 202s 823ms/step - loss: 0.1380 - acc: 0.8066 - val_loss: 0.1139 - val_acc: 0.8744
Epoch 35/50
245/245 [=====] - 235s 961ms/step - loss: 0.1374 - acc: 0.8079 - val_loss: 0.1138 - val_acc: 0.8658
Epoch 36/50
245/245 [=====] - 337s 1s/step - loss: 0.1329 - acc: 0.8172 - val_loss: 0.1114 - val_acc: 0.8601
Epoch 37/50
245/245 [=====] - 334s 1s/step - loss: 0.1296 - acc: 0.8223 - val_loss: 0.1108 - val_acc: 0.8627
Epoch 38/50
245/245 [=====] - 204s 832ms/step - loss: 0.1322 - acc: 0.8198 - val_loss: 0.1059 - val_acc: 0.8857
Epoch 39/50
245/245 [=====] - 195s 798ms/step - loss: 0.1294 - acc: 0.8209 - val_loss: 0.1140 - val_acc: 0.8652
Epoch 40/50
245/245 [=====] - 197s 802ms/step - loss: 0.1280 - acc: 0.8260 - val_loss: 0.1014 - val_acc: 0.8709
Epoch 41/50
245/245 [=====] - 216s 880ms/step - loss: 0.1281 - acc: 0.8221 - val_loss: 0.1017 - val_acc: 0.8711
Epoch 42/50
245/245 [=====] - 211s 867ms/step - loss: 0.1286 - acc: 0.8202 - val_loss: 0.1095 - val_acc: 0.8707
Epoch 43/50
245/245 [=====] - 244s 996ms/step - loss: 0.1263 - acc: 0.8291 - val_loss: 0.1041 - val_acc: 0.8742
Epoch 44/50
245/245 [=====] - 266s 1s/step - loss: 0.1267 - acc: 0.8241 - val_loss: 0.0998 - val_acc: 0.8822
Epoch 45/50
245/245 [=====] - 281s 1s/step - loss: 0.1263 - acc: 0.8226 - val_loss: 0.1015 - val_acc: 0.8796
Epoch 46/50
245/245 [=====] - 274s 1s/step - loss: 0.1263 - acc: 0.8241 - val_loss: 0.0969 - val_acc: 0.8878
Epoch 47/50
245/245 [=====] - 295s 1s/step - loss: 0.1259 - acc: 0.8245 - val_loss: 0.0988 - val_acc: 0.8856
Epoch 48/50
245/245 [=====] - 271s 1s/step - loss: 0.1247 - acc: 0.8259 - val_loss: 0.0963 - val_acc: 0.8848
Epoch 49/50
245/245 [=====] - 273s 1s/step - loss: 0.1216 - acc: 0.8362 - val_loss: 0.1070 - val_acc: 0.8771
Epoch 50/50
245/245 [=====] - 266s 1s/step - loss: 0.1207 - acc: 0.8356 - val_loss: 0.0966 - val_acc: 0.8969

```

FIGURE 8: Pertes et précision

6.3.3 Structure du resseau

```
File Edit View Bookmarks Settings Help
python tra      rodrigo@rodrigo-wesh:/media/rodrigo/Rodrigo/GIT/PAO
Using TensorFlow backend.
Loading model from ped-64x64/modelcnn-64x64.json
2019-01-06 17:12:42.964623: I tensorflow/core/platform/cpu_feature_
o use: SSE4.1 SSE4.2 AVX AVX2 FMA
2019-01-06 17:12:43.021244: I tensorflow/core/common_runtime/proces
p_parallelism_threads for best performance.
Loading weights from ped-64x64/weights-64x64.h5
Loaded model and weights from disk

Layer (type)          Output Shape         Param #
=====
lambda_1_input (InputLayer)  (None, 64, 64, 3)        0
lambda_1 (Lambda)        (None, 64, 64, 3)        0
cv0 (Conv2D)            (None, 64, 64, 16)       448
dropout_1 (Dropout)     (None, 64, 64, 16)       0
cv1 (Conv2D)            (None, 64, 64, 32)      4640
dropout_2 (Dropout)     (None, 64, 64, 32)       0
cv2 (Conv2D)            (None, 64, 64, 64)      18496
max_pooling2d_1 (MaxPooling2D) (None, 8, 8, 64)      0
dropout_3 (Dropout)     (None, 8, 8, 64)       0
fcn (Conv2D)            (None, 1, 1, 1)        4097
flatten_1 (Flatten)     (None, 1)                 0
=====
Total params: 27,681
Trainable params: 27,681
Non-trainable params: 0

rodrigo@rodrigo-wesh:/media/rodrigo/Rodrigo/GIT/PAO$ █
PAO:bash
```

FIGURE 9: Structure du resseau

6.4 Code pour construire les réseaux de neurones

6.4.1 CNN 150x150

```
model = Sequential()
model.add(Convolution2D(32, (3, 3), input_shape=(img_width, img_height, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
#Save the model
model_json = model.to_json()
with open("modelVggV2-Kitti.json", "w") as json_file:
    print("Saving model into: ", os.getcwd() + "/modelVggV2-Kitti.json")
    json_file.write(model_json)
```

6.4.2 CNN 18x36

```
def cnn(size):
    model = Sequential()
    model.add(Convolution2D(16, 5, 5, W_regularizer=l2(5e-4), border_mode='same', input_shape=(size[0], size[1], 3)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(32, 5, 5, W_regularizer=l2(5e-4), border_mode='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Convolution2D(64, 3, 3, W_regularizer=l2(5e-4), border_mode='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
    #Save the model
    model_json = model.to_json()
    with open("modelcnn-18x36.json", "w") as json_file:
        print("Saving model into: ", os.getcwd() + "/modelcnn-18x36.json")
        json_file.write(model_json)

return model
```

6.4.3 CNN 64x64

```
def cnn(inputShape=(64, 64, 3)):  
    model = Sequential()  
    # Center and normalize our data  
    model.add(Lambda(lambda x: x / 255., input_shape=inputShape, output_shape=inputShape))  
    # Block 0  
    model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu', name='cv0',  
                     input_shape=inputShape, padding="same"))  
    model.add(Dropout(0.5))  
  
    # Block 1  
    model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', name='cv1', padding="same"))  
    model.add(Dropout(0.5))  
  
    # block 2  
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', name='cv2', padding="same"))  
    model.add(MaxPooling2D(pool_size=(8, 8)))  
    model.add(Dropout(0.5))  
  
    # binary 'classifier'  
    model.add(Conv2D(filters=1, kernel_size=(8, 8), name='fcn', activation="sigmoid"))  
  
    return model
```

6.5 Entrainement des CNN

6.5.1 CNN 150x150

```
#VALUES  
  
base_path = '/media/rodrigo/Rodrigo/PAO/VGG-BdBxs/'  
train_data_dir = base_path + 'data/train'  
validation_data_dir = base_path + 'data/validation'  
nb_train_samples = 7481 #1779 pedestrian - 5702 notPedestrian  
nb_validation_samples = 748 #180 pedestrian/ 568 notPedestrian  
nb_epoch = 50  
  
#dimensions of our images.  
img = Image.open(train_data_dir + '/Pedestrian/' + "000000.png")  
img_width, img_height = img.size  
print(img_width, " ", img_height, " ", img.mode)  
##image resizing  
img_width, img_height = 150, 150
```

```

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

#PREPARE TRAINING DATA
train_generator = train_datagen.flow_from_directory(
    train_data_dir, #data/train
    target_size=(img_width, img_height), #RESIZE to 150/150
    batch_size=32,
    class_mode='binary') #since we are using binarycrossentropy need binary labels

#PREPARE VALIDATION DATA
validation_generator = test_datagen.flow_from_directory(
    validation_data_dir, #data/validation
    target_size=(img_width, img_height), #RESIZE 150/150
    batch_size=32,
    class_mode='binary')
labels = (train_generator.class_indices)
print(labels)

# In[ ]:

#START model.fit
history =model.fit_generator(
    train_generator, #train data
    samples_per_epoch=nb_train_samples,
    nb_epoch=nb_epoch,
    validation_data=validation_generator, #validation data
    nb_val_samples=nb_validation_samples)

print("saving weights to: " , os.getcwd() +"/savedweightsvgg-Kitti.h5" )
model.save_weights('savedweightsvgg-Kitti.h5')
# list all data in history
print(history.history.keys())

```

6.5.2 CNN 18x36

```

def main():
    nb_epochs=50
    nb_train_samples = 4000 + 3840 #3840 pedestrian - 4000 notPedestrian
    nb_validation_samples = 1000 + 960 #960 pedestrian/ 1000 notPedestrian
    images_dim = 18,36 #heights = 36, width = 18
    ...
    model = cnn(images_dim)
    train(model, nb_epochs, nb_train_samples, nb_validation_samples, images_dim)

```

```

def data_process(size, nb_train_samples, nb_validation_samples):
    path = '/media/rodrigo/Rodrigo/PAO/PedestrianTracking-V3/data/ped-18x36/'
    # Using the data Augmentation in training data
    datagen1 = ImageDataGenerator(
        rescale=1. / 255,
        shear_range=0.2,
        zoom_range=0.2,
        rotation_range=90,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True)

    datagen2 = ImageDataGenerator(rescale=1. / 255)

    train_generator = datagen1.flow_from_directory(
        path + 'train',
        target_size=(size[0], size[1]),
        batch_size=32,
        class_mode='binary')

    validation_generator = datagen2.flow_from_directory(
        path + 'valid',
        target_size=(size[0], size[1]),
        batch_size=32,
        class_mode='binary')

    return train_generator, validation_generator

def train(model, nb_epochs, nb_train_samples, nb_validation_samples, size):
    train_generator, validation_generator = data_process(size, nb_train_samples, nb_validation_samples)

    # Using the early stopping technique to prevent overfitting.
    earlyStopping = EarlyStopping(monitor='val_loss', patience=50, verbose=1, mode='auto')

    hist = model.fit_generator(
        train_generator, #train data
        samples_per_epoch=nb_train_samples,
        nb_epoch=nb_epochs,
        validation_data=validation_generator, #validation data
        nb_val_samples=nb_validation_samples, callbacks=[earlyStopping])

    # Saving the loss and acc during the training time.
    df = pd.DataFrame.from_dict(hist.history)
    df.to_csv('hist-18x36.csv', encoding='utf-8', index=False)
    model.save('weights-18x36.h5')
    plotHistory(hist)

```

6.5.3 CNN 64x64

```

def main():
    ...

    nb_epochs=50
    nb_train_samples = 4000 + 3840 #3840 pedestrian - 4000 notPedestrian
    nb_validation_samples = 1000 + 960 #960 pedestrian/ 1000 notPedestrian
    #images_dim = 64
    images_dim = 64 # heigth = 64, width = 64

    sourceModel = cnn()

    ...

    # Adding fully-connected layer to train the 'classifier'
    x = sourceModel.output
    x = Flatten()(x)
    model = Model(inputs=sourceModel.input, outputs=x)
    print(model.summary())
    model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

    ...

    #Save the model
    model_json = model.to_json()
    with open("modelcnn-64x64.json", "w") as json_file:
        print("Saving model into: ", os.getcwd() + "/modelcnn-64x64.json")
        json_file.write(model_json)

    ...

    #train the model ...
    train(model, nb_epochs, nb_train_samples, nb_validation_samples, images_dim)

```

```

def data_process(size, nb_train_samples, nb_validation_samples):
    path = '/media/rodrigo/Rodrigo/PAO/PedestrianTracking-V3/data/ped-64x64/data/'
    # Using the data Augmentation in training data
    datagen = ImageDataGenerator(
        rescale=1. / 255,
        shear_range=0.2,
        zoom_range=0.2,
        rotation_range=90,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True)

    datagen2 = ImageDataGenerator(rescale=1. / 255)

    train_generator = datagen1.flow_from_directory(
        path + 'train',
        target_size=(size, size),
        batch_size=32,
        class_mode='binary')

    validation_generator = datagen2.flow_from_directory(
        path + 'valid',
        target_size=(size, size),
        batch_size=32,
        class_mode='binary')

    return train_generator, validation_generator

def train(model, nb_epochs, nb_train_samples, nb_validation_samples, size):
    train_generator, validation_generator = data_process(size, nb_train_samples, nb_validation_samples)

    #checkpoint
    checkpointer = ModelCheckpoint(filepath='weights-64x64.h5',
                                    monitor='val_acc', verbose=0, save_best_only=True)

    hist = model.fit_generator(
        train_generator, #train data
        samples_per_epoch=nb_train_samples,
        nb_epoch=nb_epochs,
        validation_data=validation_generator, #validation data
        nb_val_samples=nb_validation_samples, callbacks=[checkpointer])

    # Saving the loss and acc during the training time.
    df = pd.DataFrame.from_dict(hist.history)
    df.to_csv('hist-64x64.csv', encoding='utf-8', index=False)
    model.save('weights-64x64.h5')
    plotHistory(hist)

```

7 Suivi des piétons

Pour le suivi des piétons j'ai utilisé openCV et mon CNN. Tout d'abord j'ouvre une vidéo et j'enregistre chaque frame dans une variable.

```

def track(video, iou):
    camera = cv2.VideoCapture(video)
    res, frame = camera.read()
    y_size = frame.shape[0]
    x_size = frame.shape[1]

```

Après je charge le modèle à utiliser et j'enlève le background du frame (Figure 10).

```

# Load CNN classification model
#model = loadModelAndWeights("ped-18x36/modelcnn-18x36.json","ped-18x36/weights-18x36.h5")
#model = loadModelAndWeights("kitti/modelVggV2-Kitti.json","kitti/savedweightsvgg-Kitti.h5")
model = loadModelAndWeights("ped-64x64/modelcnn-64x64.json","ped-64x64/weights-64x64.h5")
#model = loadModelAndWeights("ped-36x18/modelcnn-36x18.json","ped-36x18/weights-36x18.h5")
# Definition of MOG2 Background Subtraction
bs = cv2.createBackgroundSubtractorMOG2(detectShadows=True)
history = 20
frames = 0
counter = 0

```



FIGURE 10: Image du frame sans background

Puis je fais un peu de traitement d'image sur le frame (expansion et débruitage) et je trouve les contours des objets présents sur le frame (findContours).

```

# Expansion and denoising the original frame
th = cv2.threshold(fg_mask.copy(), 244, 255, cv2.THRESH_BINARY)[1]
th = cv2.erode(th, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)), iterations=2)
dilated = cv2.dilate(th, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (8, 3)), iterations=2)
image, contours, hier = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

```

Finalement j'extrais les objects sur l'image et j'utilise mon réseau de neurones pour classifier s'il y a un piéton ou pas.

```

x, y, w, h = cv2.boundingRect(c)
if cv2.contourArea(c) > 3000:
    # Extract roi
    frame2 = frame[y:y+h, x:x+w, :]
    img = frame[y:y+h, x:x+w, :]

```

```

#rimg = cv2.resize(img, (36, 18), interpolation=cv2.INTER_CUBIC)
#if using model 36x18
#rimg = cv2.resize(img, (18, 36), interpolation=cv2.INTER_CUBIC)
#if using model 64x64
rimg = cv2.resize(img, (64, 64), interpolation=cv2.INTER_CUBIC)
#if using model kitti
#rimg = cv2.resize(img, (150, 150), interpolation=cv2.INTER_CUBIC)
image_data = np.array(rimg, dtype='float32')
image_data /= 255.
roi = np.expand_dims(image_data, axis=0)
flag = model.predict(roi)

```

S'il y a un piéton je dessine un bounding box autour de l'object et s'il n'y a pas de piéton je ne dessine pas de bounding box

```

.... if flag[0][0] > 0.5:
....     e = Entity(counter, (x, y, w, h), frame)
.... 
.... # Exclude existing targets in the tracking list
.... if track_list:
....     count = 0
....     num = len(track_list)
....     for p in track_list:
....         if overlap((x, y, w, h), p.windows) < iou:
....             count += 1
....     if count == num:
....         track_list.append(e)
....     else:
....         track_list.append(e)
.... counter += 1

```

Après je montre le frame avec ou sans le bounding box dessiné

```

.... frames += 1
.... out.write(frame)
.... cv2.imshow("detection", frame )

```

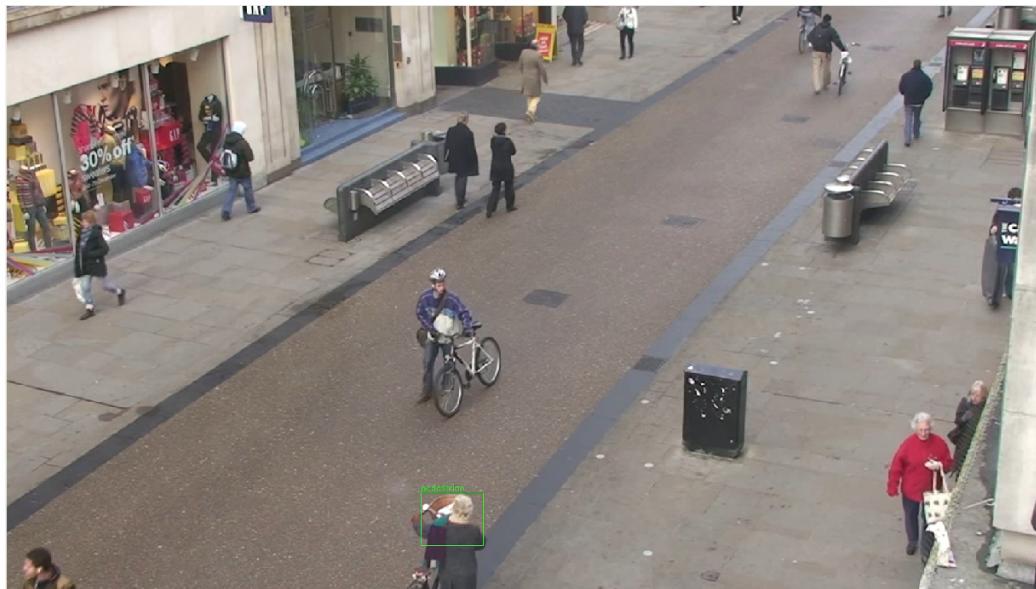
8 Résultats

Pour tester la fonctionnalité des modèles, j'ai utilisé la vidéo de « town center video » avec les 3 modèles.

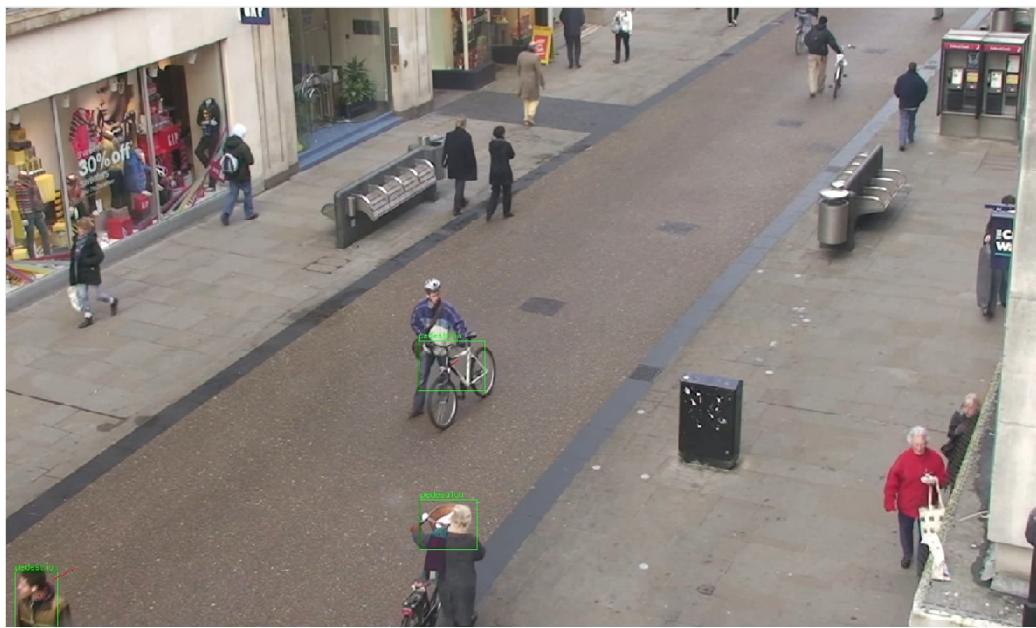
8.1 Capture de la vidéo originale



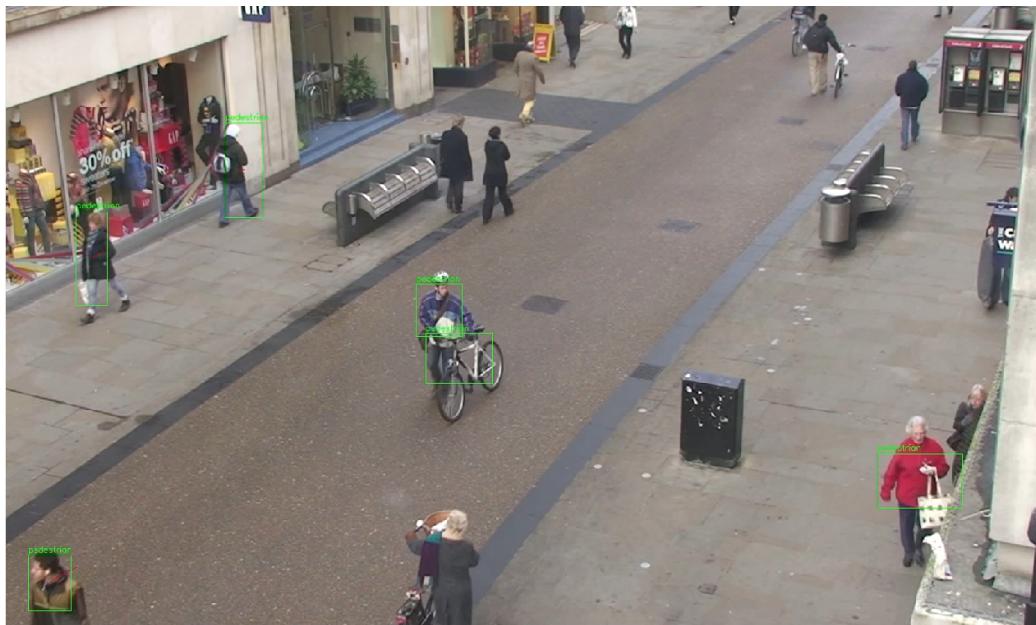
8.2 Capture de la vidéo après le 1er modèle



8.3 Capture de la vidéo après le 2eme modèle



8.4 Capture de la vidéo après le 3eme modèle



Références

- [1] T. B. M. V. ASSOCIATION AND S. FOR PATTERN RECOGNITION, *Pedestrian tracking*. <http://www.bmva.org/apps:pedestrian>.
- [2] G. BRAIN, *Tensorflow*. <https://www.tensorflow.org/>, 2015.
- [3] F. CHOLLET, *Keras*. <https://keras.io/>, 2015.
- [4] R. GANDHI, *R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms*. <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>, 2018.
- [5] D. M. GAVRILA, *Pedestrian detection*. http://www.gavrila.net/Research/Pedestrian_Detection/pedestrian_detection.html, 2001-2013.
- [6] A. GEIGER, P. LENZ, AND R. URTASUN, *Are we ready for autonomous driving ? the kitti vision benchmark suite*. http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d, 2012.
- [7] D. HOIEM, *Object Category Detection : Sliding Windows*. https://courses.engr.illinois.edu/cs543/sp2012/lectures/Lecture%203%20-%20Sliding%20Window%20Detection%20-20Vision_Spring2012.pdf, 2012.
- [8] INTEL, *Open Source Computer Vision Library*. <https://opencv.org/>, 2017.
- [9] F.-F. LI, J. JOHNSON, AND S. YEUNG, *Convolutional neural networks for visual recognition*. <https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM81jYj-zLfQRF3E08sYv>, 2017.
- [10] J. REDMON AND A. FARHADI, *Yolov3 : An incremental improvement*. <https://pjreddie.com/darknet/yolo/>, 2018.
- [11] A. ROSEBROCK, *Deep learning for computer vision with Python*.
- [12] STANFORD UNIVERSITY, *Slides 2017 cs231n*. <http://cs231n.stanford.edu/slides/2017/>, 2017.
- [13] M. SZARVAS, A. YOSHIZAWA, M. YAMAMOTO, AND J. OGATA, *Pedestrian detection with convolutional neural networks*. https://www.researchgate.net/publication/4171908_Pedestrian_detection_with_convolutional_neural_networks., 2005.
- [14] S. UNIVERSITY, *Convolutional neural networks for visual recognition*. <http://cs231n.stanford.edu/2017/>, 2017.