Détection D'objets en Mouvement

Monsieur Tchimanga Blaise

Thèse de Bachelor présentée pour l'obtention du titre
Bachelor of Science HES-SO en
Ingénierie des technologies de l'information
Orientation Multimédia, réseau, communication
Suivi par
Monsieur Andres Revuelta, professeur HES
Septembre 2017

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

printemps 2017
Session de bachelor

INGÉNIERIE DES TECHNOLOGIES DE L'INFORMATION

COMMUNICATIONS, MULTIMEDIA ET RESEAUX

DETECTION D'OBJETS EN MOUVEMENT

Descriptif :

La reconnaissance d'objets fait partie des technologies classiques liées au traitement du signal les plus employées. Une difficulté supplémentaire survient lorsque les objets à reconnaître sont en mouvement. Ors, le traitement de ce cas de figure est couramment nécessaire.

Des outils logiciels existent pour nous aider à traiter les différentes facettes de ce type de reconnaissance, notamment les outils de traitement d'images en temps réel.

Le présent travail propose d'explorer les solutions du marché et de l'appliquer à un cas d'étude à définir. L'objet choisi devra pouvoir aussi être détecté s'il est en mouvement dans une direction prédéterminée.

Travail demandé :

- Réalisation du cahier des charges détaillé.
- État de la technique du domaine.
- Proposition d'une solution adaptée au cas d'étude choisi.
- Réalisation d'un démonstrateur du cas d'étude.
- Le candidat fournira quatre tirages sur papier et l'accès aux codes sources, mémoire, résumé et présentation.

Le mémoire comprendra :

- L'énoncé du travail.
- Le cahier des charges
- L'étude théorique du sujet.
- Une description complète des logiciels/configurations.
- Le résultat complet du développement de la plateforme.
- La planification établie pour le projet et son déroulement effectif.
- Une table des matières, une conclusion.

Candidat :

M. Tchimanga Blaise

Filière d'études : ITI

Professeur(s) responsable(s) :

Revuelta Andrés

Travail de bachelor soumis à une convention de stage en entreprise : **non**
Travail de bachelor soumis à un contrat de confidentialité : **non**

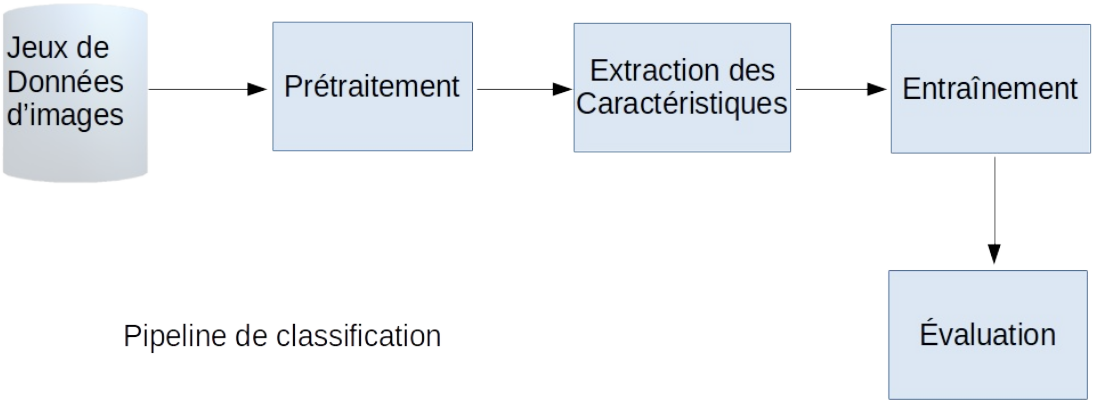
h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Printemps 2017
Session de bachelor

Résumé :

La détection automatique d'objets est une technologie qui devraient être de plus en plus sollicitée et se développer dans les années à venir pour peu que la demande en assistance personnel par les machines dans des domaines tels que, la conduite assistée de véhicules, la réalité augmenté, la robotique, etc, se poursuit.

Le présent travail propose d'explorer les solutions techniques qui permettent la localisation de manière robuste d'un objet prédéfini en mouvement.



Pipeline de classification

Candidat :

M. Tchimanga Blaise
Filière d'études : ITI

Professeur(s) responsable(s) :

REVUELTA ANDRES

En collaboration avec :
Travail de bachelor soumis à une convention de stage en entreprise : non
Travail de bachelor soumis à un contrat de confidentialité : non

Remerciement

Un grand merci à M. Revuelta pour m'avoir proposé ce projet, pour ses bons conseils et pour son suivi. Merci également à tous les enseignants des cours du soir pour leurs enseignements.

Une spéciale dédicace à Mm Grasshoff , pour son incroyable générosité et pour son soutient constant depuis de nombreuses années.

Guide du lecteur

chapitre 1 : Le chapitre 1 contient le cahier des charges ainsi que la planification du travail

chapitre 2 : Le chapitre 2 est une présentation de la vision par ordinateur et des bibliothèques utilisées

chapitre 3 : Le chapitre 3 comprend une étude des techniques de détections de mouvement

chapitre 4 : Le chapitre 4 porte sur les techniques de détections d'objets

chapitre 5 : Le chapitre 5 parle des réalisations pratiques

chapitre 6 : Le chapitre 6 évoque les difficultés rencontrées

chapitre 7 : Le chapitre 7 contient la conclusion

chapitre 8 : Le chapitre 8 contient les références

chapitre 9 : Le chapitre 9 contient les scipts d'installations et les codes

Table des matières

Table des figures

Première partie

Introduction

Chapitre 1

Contexte

1.1 Cahier des charges

— étude théorique de la technique
— faire un cahier des charges
— détecter le passage de cyclistes
— fournir scripts, codes et configurations utilisés
— écriture du mémoire
— faire un résumer
— faire une présentation

1.2 Planification du travail

Semaines/ Tâches	1	2	3	4	5	6	7	8	9	10	11
Recherche Détection de mouvement											
Recherche reconnaissance d'objets											
Réalisation pratique											
Ecriture du mémoire											

Planning effectif

Semaines/ Tâches	1	2	3	4	5	6	7	8	9	10	11
Recherche Détection de mouvement											
Recherche reconnaissance d'objets											
Réalisation pratique											
Ecriture du mémoire											

Chapitre 2

Présentation

2.1 Vision humaine et Vision par ordinateur

On peut se représenter la vision humaine en deux étapes. Premièrement la capture par les yeux des informations transportés par le flux lumineux après réflexion sur de la matière . Ce flux lumineux est transformé ensuite en flux nerveux à travers le nerf optique. La seconde étape plus mystérieuse est l'interprétation cognitif de ce flux nerveux par le cortex visuel à l'arrière du cerveau, pour reconnaitre formes, mouvements et objets. Notre système visuel est opérationnel dans les longueurs d'ondes comprises entre 400 et 700 nanomètres, il est plus sensible aux changements de luminosité qu'aux couleurs et les mouvements dans notre champs de vision sont facilement détectés. Dans la vision par ordinateur, les yeux sont remplacés par les capteurs et la lentille d'une caméra , le traitement d'image et l'analyse cognitif par le programme informatique. Pour le cerveau humain reconnaitre un chat ou une chaise est une tache effectuée par la plupart d'entre nous avec aisance . Pour l'ordinateur cette tâche n'est pas du tout évidente vu les différents types de formes, de positions, d'échelle ,de points de vue ou d'expositions que peuvent prendre des objets comme les chats ou les chaises. Lorsque l'objet en question est en partie occulté , la complexité devient encore plus grande. Sans parler des formes ambigües donnant lieu à des illusions d'optique. La vision par ordinateur cherche à trouver des méthodes ayant des propriétés plus ou moins semblable à notre système visuel pour rendre l'interprétations du monde extérieur plus efficace pour les machines. La détection d'objets repose sur des techniques de vision par ordinateur.

2.2 Image numérisée en 2D

Une image numérisée en 2D, est une image dont les éléments (pixels) sont discrètisés en valeur numérique sur 8 bits pour les variations de gris ou en 3x8 bits pour les images en couleurs (RGB). Faire du traitement ou de l'analyse d'image revient à manipuler des concepts d'algèbre linéaire tel que vecteurs et matrices. Chacun de ces pixels représente un point dans une matrice à coordonnée (x,y). En informatique cette matrice est stockée dans la mémoire comme un tableau à deux dimension.

2.3 Bibliothèques open source pour le traitement et l'analyse d'image

OpenCV (Open Computer Vision) est une bibliothèque open source pour la vision par ordinateur et le traitement d'images conçue comme un ensemble de modules fonctionnels d'algorithmes optimisés pour le traitement temps réel et qui peuvent être combinés ensemble pour créer des applications plus complexe. Elle est écrite en C/C++, et a aussi des interfaces pour Python , Java et Matlab. La dernière version stable à ce jour, maintenue par la société Intel est la version 3.2.0. Depuis 2010 NVidia développe des modules CUDA spécifique à ses processeurs

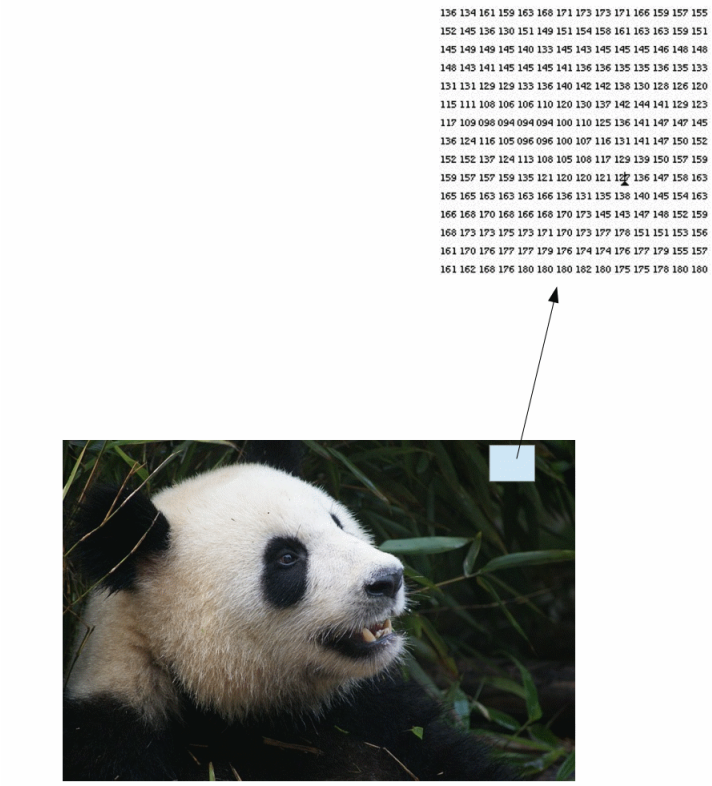


FIGURE 2.1 – Image numérique

FIGURE 2.2 – logo opencv

FIGURE 2.3 – scikit-learn et scikit-image

FIGURE 2.4 – dlib et boost

graphiques. On peut trouver les sources et la documentation sur opencv.org ou sur Github.

Scikit-image et Scikit-learn, sont également , d'autres bibliothèques open source d'algorithmes pour le traitement d'image et pour le machine learning. Elles sont écrite en python et maintenues par une communauté de volontaire.

Dlib est une bibliothèques c++ très performante qui contient des algorithmes open source pour le machine learning et la vision par ordinateur entre autre, dont des méthodes pour la détection d'objets, ce qui va être très intéressant pour nous. Elle fait fortement usage de la librairie Boost qui fournit des implémentations pour l'algèbre linéaire , le multithreading et les bases du traitement d'image. Il nous faudra donc l'installer avec Boost mais aussi avec Boost.Python qui assure la traduction entre les langages C++ et Python. On pourra ainsi combiné la vitesse de traitement de C++ avec la simplicité de Python.

Deuxième partie

Etude théorique de la technique

Chapitre 3

Techniques de détection de mouvement

3.1 introduction

D'après les recherches effectués au préalable , de nombreuses solutions sont théoriquement envisageable pour détecter le mouvement au sein d'une matrice de pixels. Techniquement c'est un problème à très large spectre qui dépend de ce que l'on cherche à détecter et du contexte. En pratique on retrouve le plus souvent les approches suivantes :

3.2 Calcul du flot optique

Le calcul du flot optique est un traitement d'images dit de bas niveau car basé sur le mouvement au niveau des pixels, permettant d'estimer le déplacement d'objets dans une scène. Plusieurs Méthodes comme la méthode Lucas-Kanade ou la méthode Horn-Schunck, sont utilisés.

3.3 Différences d'images successives

Les méthodes basées sur la différence entre images successive sont rapides et faciles à mettre en place. Elles s'adaptent bien aux changements brutaux du fond, mais sont très sensibles aux changements de l'intensité lumineuse.

3.4 Soustraction de fond

La soustraction de fond est une méthode fréquemment utilisée dans les cas où une caméra fixe cherche à segmenter un avant plan mobile en soustrayant l'arrière plan immobile. Dans le cas où on a une image de notre fond lorsque rien ne passe devant la caméra comme dans une pièce vide, il est facile de segmenter les objets mobiles en soustrayant simplement l'image du fond de la nouvelle image. Sinon il faut soustraire le fond d'images successives et cela se complique car même une ombre ou un reflet passe en avant plan comme un objet en mouvement. Plusieurs algorithmes sont conçu pour la soustraction de fond comme la Mixture de gaussienne, ou l'algorithme vumètre.

Chapitre 4

Techniques de détection d'objets

4.1 Introduction

Les systèmes de détection d'objets suivent un schéma général . Un descripteur pour permettre la classification de candidats . La classification de candidats consiste à établir une frontière qui sépare les objets de classes différentes afin de les distinguer. Pour obtenir cette frontière on utilise l'apprentissage automatique comprise sous le terme de machine learning. La classification d'objet permet de prendre une décision qui se matérialise par une bonding box , soit un rectangle entourant l'objet détecté dans une image.

4.2 Similarités entre images

Comme l'ordinateur ne voit dans une image qu'une matrice de pixels, la fonction élémentaire dans la détection d'objets et celle qui détermine si deux matrices de pixels ont des similarités ou non. Une façon de faire est de chercher une similitude basée sur une propriété du pixel comme la couleur, mais cette méthode a pour désavantage d'être très sensible aux petits changements et ne tient pas compte de la forme de l'objet. Il existe une méthode moins naïve et qui tient compte de la forme de l'objet : le Template Matching.

4.2.1 Template Matching

Le template matching est une technique pour trouver dans une image source, la zone qui est similaire à une image donnée comme modèle, soit l'objet que l'on veut détecter. L'algorithme du template matching calcule, par glissement du template de gauche à droite et de haut en bas, un métrique qui représente la similarité entre le model et la zone observée. Pour chaque emplacement de T(template) sur I(image) , le métrique est stocké dans une matrice R (Résultat). Soit chaque emplacement (x,y) dans R, contient la valeur du match. L'emplacement avec la plus haute valeur de similitude est considéré comme concordante. Une fonction pour calculer le maximum global est utilisée à cette fin.

4.3 Extractions de caractéristiques

La faiblesse du template matching, est que cette méthode n'est robuste ni aux rotations , ni aux distorsions, ni aux changement d'échelle, ni aux changements de luminosité, etc. En fait elle permet juste de reconnaitre une image connue, dans une plus grande image. C'est la qu'il faut introduire la notion d'extraction de caractéristiques. L'extraction de caractéristique d'une image par un descripteur, est le processus qui permet de créer un vecteur de caractéristiques.

4.4 Descripteur

Il existe deux types de descripteurs, les descripteur d'images et les descripteurs de caractéristiques. Les premiers sont globaux et à partir d'une image, génèrent un seul vecteur de caractéristiques qui peut être directement exploité pour la classification, les second sont locaux, plus robustes, mais génèrent plusieurs vecteurs de caractéristiques qui ont un cout en espace de stockage et demandent un travail supplémentaire pour les condenser en un seul. voici quelques descripteurs de caractéristiques que nous allons utiliser.

4.4.1 LBP

Local Binary Patterns, est un descripteur de caractéristiques qui donne une information relative au niveau de luminance entre un pixel et ses voisins. C'est un descripteur de texture qui obtient de bons résultats dans la reconnaissance de visage. Il s'applique sur des images en échelle de gris.

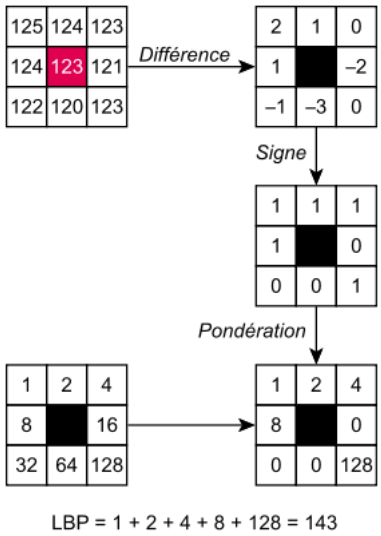


FIGURE 4.1 – descripteur lbp -image source wikipedia

4.4.2 Pseudo-Haar

Introduit par Viola et Jones en 2001, l'algorithme calcule la somme des différences entre pixels à l'intérieur d'une fenêtre de détection, à l'aide d'un masque de rectangles qui peuvent être à n'importe quelles positions et à n'importe quelle échelle à l'intérieur de la fenêtre de détection. On soustrait la somme des pixels sous la région blanche des rectangles de la somme des pixels sous la région noire. Combiné à la technique des images intégrale, ces caractéristiques permettent de faire de la détection en temps réel.

FIGURE 4.2 – rectangles pseudo haar- image source wikipedia

4.4.3 Histogramme de gradient orienté

Un histogramme de gradient orienté est un descripteur qui utilise la distribution de l'intensité des gradient locaux et la direction des contours pour la reconnaissance de forme et de texture. Ceci peut être fait en divisant l'image en des régions adjacentes de petite taille, appelées cellules, et en calculant pour chaque cellule l'histogramme des directions du gradient ou des orientations des contours pour les pixels à l'intérieur de cette cellule. La combinaison des histogrammes forme alors le descripteur HOG. Pour de meilleurs résultats, les histogrammes locaux sont normalisés en contraste, en calculant une mesure de l'intensité sur des zones plus larges que les cellules, appelées des blocs, et en utilisant cette valeur pour normaliser toutes les cellules du bloc. Le descripteur résultant de cette normalisation est plus robuste aux changements de luminosité.

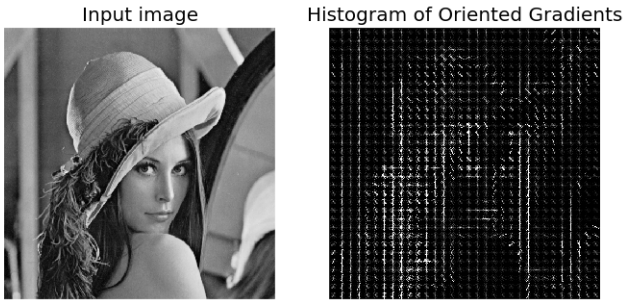


FIGURE 4.3 – Histogramme de gradients orientés- image source sckit-image

4.4.4 ORB

ORB est une implémentation propre à openCV. Inspiré essentiellement du détecteur de points clé FAST et du descripteur de caractéristique BRIEF, comme alternatif à SIFT. ORB utilise FAST pour détecter les coins dans une image, les mets dans un vecteur de caractéristiques, puis calcule les angles et les distances entre les coins pour mesurer la correspondance entre vecteurs, par la distance de Hamming. ORB peut être utilisé dans la reconnaissance d'images.

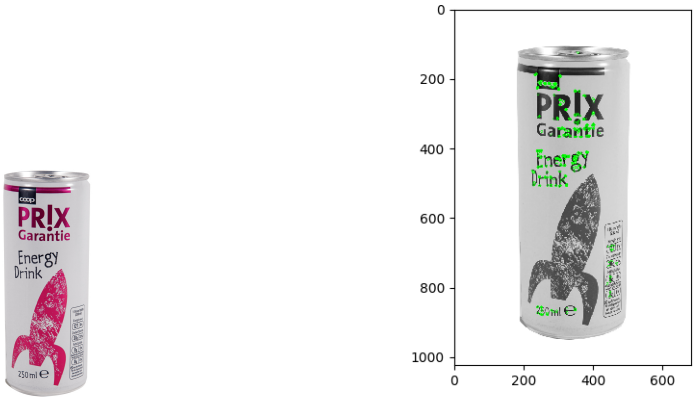


FIGURE 4.4 – ORB test

4.5 Vecteur de caractéristiques

Obtenu par l'extraction de caractéristiques globales ou locales, un Vecteur de caractéristiques, est un vecteur colonne. Cette suite de nombres décrit pour l'ordinateur, la variété des propriétés d'une image et permet la classification de l'objet.

4.6 La classification d'objet

La classification d'objet consiste à mettre un label à un objet pour dire de quelle classe il fait parti. On fait pour cela appel au machine learning.

4.7 Machine learning

On considère trois type d'algorithmes d'apprentissage :

4.7.1 L'apprentissage supervisé

L'apprentissage supervisé est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage contenant des exemples de cas généralement déjà traités et validés. Exemple d'algorithme : Boosting , arbre de décision, machine à vecteur de support, KNN, Réseau de neurones, classification naïve bayesienne, Régression logistique, etc.

4.7.2 L'apprentissage non supervisé

L'apprentissage non supervisé est une méthode d'apprentissage automatique qui consiste à diviser un groupe hétérogène de données, en sous groupe de manière que les données considérées comme les plus similaires soient associées au sein d'un groupe homogène et que les données considérées comme différentes se retrouvent dans d'autres groupes distincts. L'objectif étant de permettre une extraction de connaissance organisée à partir de ces données.

4.7.3 L'apprentissage par renforcement

L'apprentissage par renforcement est une méthode d'apprentissage automatique dont le but est d'apprendre à partir d'expériences, ce qui convient de faire en différente situations, de façon à optimiser une récompense quantitative au cours du temps.

Dans le cas qui nous intéresse, nous utilisons les méthodes d'apprentissages supervisées.

4.7.4 Boosting

Boosting est une famille d'algorithmes dont le but est de construire un classificateur fort à partir d'un ensemble de classificateurs faibles. Adaboost est la variante utilisée avec le descripteur pseudo-haar.

4.7.5 SVM (machine à vecteur de support)

Les SVM font parti de la famille des classificateurs linéaire. Pour notre problématique, ils sont utilisé afin de discriminer deux échantillons en établissant un hyperplan qui sépare avec la plus grande marge possible les caractéristiques des vecteurs donnés en entré. A la sortie nous obtenons un classificateur pour l'objet recherché, qui peut être très performant en fonction du jeux de données à la base de l'entraînement.

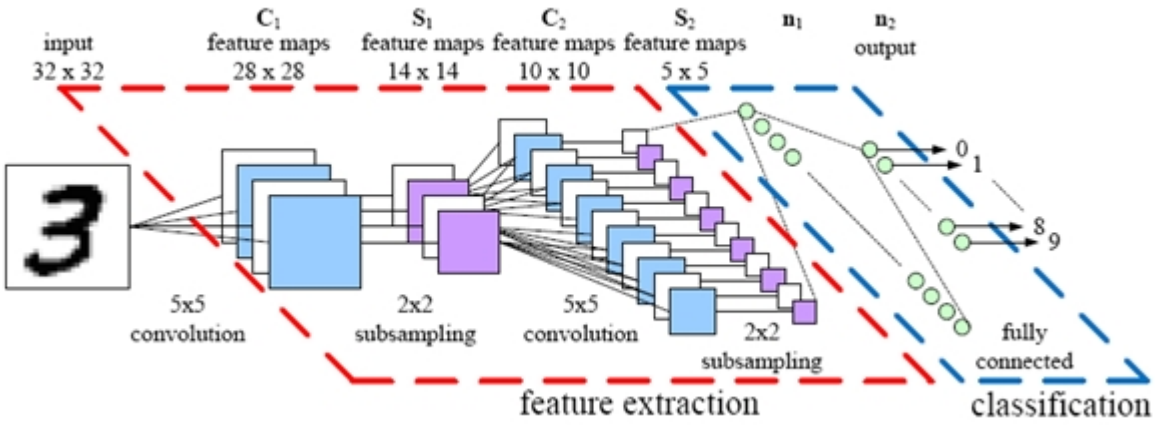


FIGURE 4.5 – architecture CNN

4.8 Réseaux de Neurones à convolution (CNN)

La tendance actuelle est aux réseaux de neurones convolutifs. Ces derniers peuvent être utilisés pour un tas d'applications allant de la reconnaissance d'images au traitement du langage naturel. L'idée est d'apprendre automatiquement les descripteurs, au lieu de les désigner manuellement. Cette méthode est intéressante à l'échelle du million d'échantillons d'images données en entrée et s'appuie sur la puissance de calcul des GPU. Un réseau de neurones convolutifs est un type de réseau de neurones artificiel inspiré par les connections entre neurones dans le cortex visuel des animaux qui se chevauchent lors du pavage du champ visuel. Les CNN consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations . La dernière couche est un classificateur entraîné pour des tâches complexes.

Troisième partie

Réalisation

Chapitre 5

Réalisation

5.1 Mise en place de l'environnement de travail

Le système d'exploitation utilisé est Ubuntu 16.04 . Pour le langage de programmation, python 2.7 au lieu de C++, car il y a une grande communauté de gens qui utilisent python dans le domaine de l'intelligence artificielle et de la vision par ordinateur. C'est un langage avec une syntaxe qui le rend très lisible et qui permet de se concentrer plus sur la logique d'un programme et moins sur des concepts de plus bas niveau. La vie étant trop courte pour gérer des problèmes de compilateur, python est un bon choix et nous pouvons grâce au binding profiter de la rapidité des bibliothèques C++. Il y a en annexe un script qui permet d'installer openCV 2.4 et de faire la liaison pour python. D'autres bibliothèques viendront s'installer par la suite, les scripts seront aussi en annexe. Pour l' IDE , tantôt "gedit" pour un coup d'oeil rapide , tantôt "Pycharm", IDE spécialisé pour la production de code en python. Nous nous fixons comme cadre d'étude, la détection du passage de cyclistes dans une vidéo ou à l'aide d'une caméra fixe.

5.2 Techniques utilisant la détection de mouvement

5.2.1 Soustraction de l'arrière plan

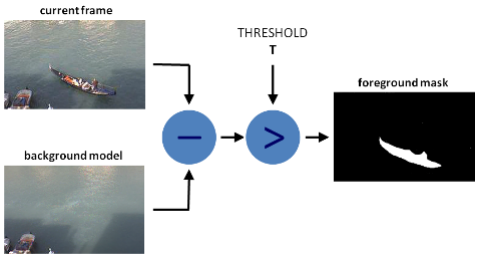


FIGURE 5.1 – soustraction arrière plan

A la lecture de l'énoncé du mémoire, "Détection d'objets en mouvement ", On peut se dire intuitivement que pour détecter un objet en mouvement, il faut d'abord trouver un algorithme qui distingue le mouvement du statique d'un flux vidéo, puis voir de là si on peut reconnaitre une forme et appliquer une méthode pour extraire un contour. openCV 2.4 possède une méthode basée sur la mixture de gaussienne nommée "BackgroundSubtractorMOG()" qui permet d'extraire l'avant plan en mouvement en blanc, de l'arrière plan statique en noir. Le code en annexe ne prend que quelques lignes.

Algorithm 1 pseudocode BackgroundSubtractor

```
cap ←captureVideo()
fgbg ←BackgroundSubstractorMOG()
while esc ≠ 1 do
    trame ←lecture(cap)
    masque ←fgbg(trame)
    affichage ←masque
end while
liberer ←cap
detruire ←affichage
```

Résultat

Le moindre changement de lumière ou le moindre reflet est perçu comme un mouvement. L'application d'algorithmes de détection de contour, comme le "hough circles", a généré une multitude de cercles imprévisibles. Après quelques tests infructueux, il a paru plus judicieux de s'intéresser aux techniques détection d'objets dans une image. Après tout une vidéo est une succession d'images.

5.3 Techniques utilisant la classification

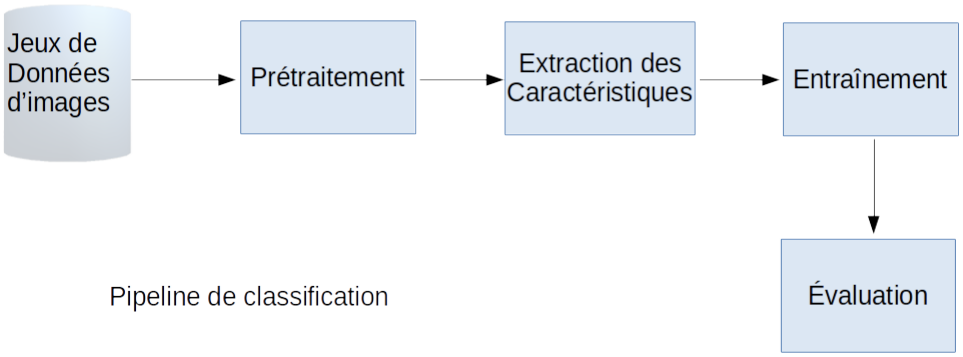


FIGURE 5.2 – Pipeline de classification

5.3.1 Haar Cascades et Boost (openCV)

En 2001, Paul viola et Michael Jones, ont publié un papier nommé "Rapid object detection using a boosted cascade of simple features" proposant un framework, pour la détection (et non la reconnaissance) de visages, en utilisant les caractéristiques Haar et l'algorithme d'apprentissage Adaboost qui construit un classificateur fort avec une cascade de classificateur faible.
Ce framework a été l'état de l'art pendant des années pour la détections de visages et est parfois employé pour la détection de divers objets.
La bibliothèque openCV implémente ce framework et met à disposition dans son répertoire data/haarcascades , des classificateurs sous forme de fichiers xml , pour la détection de visages , de parties du corps, ou de corps en entier. On peut utiliser ces fichiers pour identifier par un rectangle de couleur, des formes humanoïdes dans une vidéo avec un code comme celui en annexe.

Algorithm 2 pseudocode haarcascade opencv

```
classificateur ←Cascadehaar(fichierXML)
cap ←captureVideo()
while cap =ouvert do
  while esc ≠ 1 do
    trame ←lecture(cap)
    objet ←detectionAplusieursEchelle(classificateur)
    if il ya un objet then
      dessineRectangleAutour
    end if
    affichage(image)
  end while
end while
liberer ←cap
detruire ←affichage
```

La conversion en niveau de gris permet de réduire un peu le temps de traitement en enlevant de l'information non pertinente. On peut aussi réduire la dimension des trames pour gagner un peu de vitesse.

Résultat

Hormis quelques faux positifs et une petite latence, le résultat est plutôt bon dans le cas où on cherche à détecter des piétons placé à une distance allant jusqu'à quelques mètre dans le plan de la caméra. La seule marche de manoeuvre ici est la méthode detectMultiscale() qui comme son nom l'indique, sert à détecter des objets à différentes tailles dans l'image. Le premier paramètre est la matrice image, le deuxième paramètre spécifie de combien la taille de l'objet est réduite . Le troisième paramètre précise le nombre de voisins que chaque candidat (rectangle) peut avoir. Partant de là, nous avons voulu savoir comment générer un classificateur haarcascade. Pour cela nous avons suivi les étapes présenté dans le blog coding-robin.de, inspiré par le tutoriel de Naotoshi Seo. (voir références)

Jeux de donnée cyclistes

La première chose à faire consiste à constituer deux dossiers. Un dossier dit positif , contenant l'objet que l'on cherche à reconnaître et un dossier dit négatif ou background, contenant ce qui n'est pas notre objet. Il est possible d'utiliser des bases de données d'université ayant un laboratoire de recherche sur la vison par ordinateur. En général ils proposent un dossier contenant les images et un dossier d'annotations contenant les coordonnées (x,y) de la région d'intérêt ROI de l'image correspondante.
Avec un téléphone portable environ 70 photos de cyclistes on été prise et redimensionner autour de la région d'intérêt pour ne pas avoir à faire d'annotations.

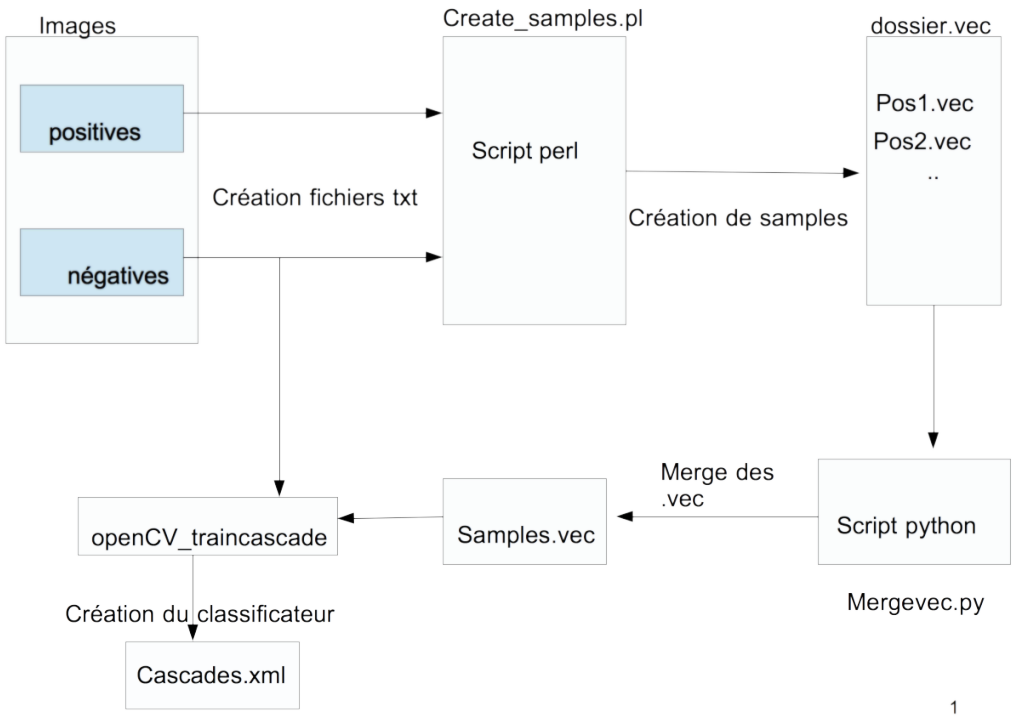


FIGURE 5.3 – Schéma création haarcascades

Puis environs 230 images quelconques sans cyclistes, laissé tel quel. Les images ont été respectivement mis dans un dossier "positives" et "négatives".

Création de samples

Une fois les dossiers mis à leur place , un fichier .txt qui liste les images contenues dans chacun des répertoires est crée, en tapant dans un terminal :

```
1 #!/bin/bash
2 find ./negatives -iname "*.jpg" > negatives.txt
3 find ./positives -iname "*.jpg" > positives.txt
```

Ces fichiers textes sont passés au script perl create samples.pl qui génère des samples positifs en combinant chaque images positives avec une image négative prise au hasard en utilisant la méthode opencv createsamples.
opencv cratesamples génère des samples sous forme d'images vectorielles et applique des transformations et des distorsions sur les images.

```
1 #!/bin/bash
2 createsamples.pl positives.txt negatives.txt samples 1500\
3 "opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\
4 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 40"
```

Cela a pour effet de mettre dans le dossier samples , les images vectorielle en noir et blanc légèrement distordue de 80 pixel de large et 40 de haut.

Fusion des fichiers .vec

Le script python Mergevec permet de fusionner tous les fichier .vec du dossier samples en un seul fichier samples.vec qui sera donner à la méthode opencv traincascade.

```
1 #!/bin/bash
2 mergevec.py -v samples/ -o samples.vec
```

Entraînement de cascades

```
1 #!/bin/bash
2  opencv_traincascade -data classifier -vec samples.vec -bg negatives.txt\
3 -numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos
      1000\
4 -numNeg 600 -w 80 -h 40 -mode ALL -precalcValBufSize 1024\
5 -precalcIdxBufSize 1024
```

la méthode prend comme paramètres : -data : le nom du dossier de sorti pour le fichier xml final, -vec : le fichiers de samples, -bg : le background donc notre listing txt du dossier négatif. Les paramètres les plus importants, sont

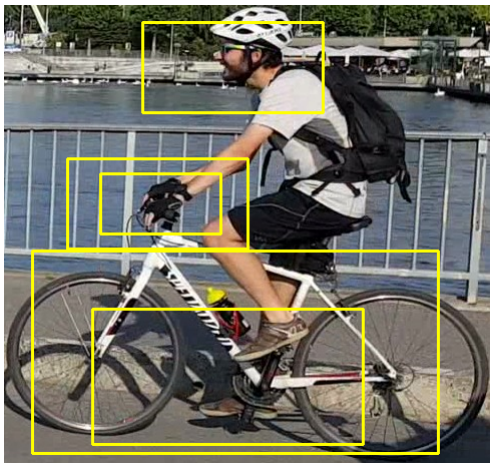


FIGURE 5.4 – Résultat haar avec 20 cascades

ici -numPos : le nombre de samples positif que l'on possède pour entrainer notre cascades ainsi que -w et -h qui sont les dimensions de nos samples et doivent donc avoir exactement la même dimension que lors de leur création, et bien sûr -numStages qui est le nombre de cascades que l'on souhaite entrainer. -minHitRate : le taux de succès minimum à chaque étapes. -maxFalseAlarmRate : taux maximum de fausse alarme à chaque étape. -precalcValBufSize : est la taille du tampon précalculé. -precalcldxBufSize : taille du tampon pour les indices de caractéristiques précalculées, plus on a de mémoire disponible, plus le processus d'entrainement est rapide. Après quelques jours de calculs , nous obtenons finalement notre descripteur sous forme de fichier xml. Le type de descripteur par défaut est haar. Si on souhaite faire une cascade avec LBP au lieu de haar, il faut ajouter après l'option -precalcldxBufSize, -featureType LBP.

Résultat

Le classificateur, n'était pas assez entrainé avec le nombre d'images prises ,ça a malgré tout pris plusieurs jours et il y avait beaucoup trop de faux positif et une superposition de boites d'encadrement comme on peut le voir sur l'image ci dessus. On peut toujours tâcher d'améliorer la détection en améliorant la qualité et la quantité des datas.

5.3.2 HOG et SVM (openCV)

En 2005 Navneet Dalal et Bill Triggs deux chercheurs de l'INRIA , ont publié un papier qui fait toujours référence aujourd'hui, sur l'utilisation des histogramme à gradients orientés avec une machine linéaire à vecteur de support, pour la détection de piétons. Leur but était de proposer un framework assez robuste pour la détection d'objets . (voir références)

OpenCV donne par défaut une implémentation calquée sur la méthode mise en oeuvre par Dalal et Triggs pour la détection de piétons, écrit en dur dans le fichier source hog.cpp du module objdetect. En modifiant l'exemple donné par opencv pour la détection de personnes dans une image, il est possible de l'exploiter pour la détection de personnes dans une vidéo (voir annexe)

Résultat

Le résultat est meilleur que avec le détecteur de piéton haarcascade donné par défaut, avec également une latence dans les trames . Mais de nouveau, hormis la methode detectMultiScale, pas beaucoup de manoeuvres dans le réglage du détecteur. Comme pour haarcascade, nous avons voulu connaître les moyens donnés par opencv pour la création d'un détecteur d'objet avec HOG et SVM. Opencv offre peu de flexibilité pour créer ce type de détecteur. D'autres bibliothèques comme scikit-image et scikit-learn offrent plus d'options pour fabriquer un détecteur sur mesure.

5.3.3 HOG SVM (scikit)

A l'aide de l'excellent blog de vision par ordinateur "pyimagesearch.com" de Adrian Rosebrock, nous apprenons que le pipeline de création comprend plusieurs étapes.

Samples

Comme pour haarcascades il faut fournir à l'algorithme d'apprentissage des samples positifs avec la localisation de la zone à extraire (ROI) acronyme pour region of interest, ainsi que des samples négatifs qui ne contiennent pas du tout l'objet. Un algorithme extraira sur chaque images les histogrammes de gradient orientés puis les metra à la suite dans un vecteur.

Calcul du descripteur

La bibliothèque Scikit-image donne une méthode appelée "hog" pour créer le descripteurs HOG . Elle prend en paramètre, l'image, le nombre d'orientations qui est le nombre de boite de l'histogramme , le nombre de pixels par cellules , le nombre de cellules par blocs et une option de visualisation du résultat. Par

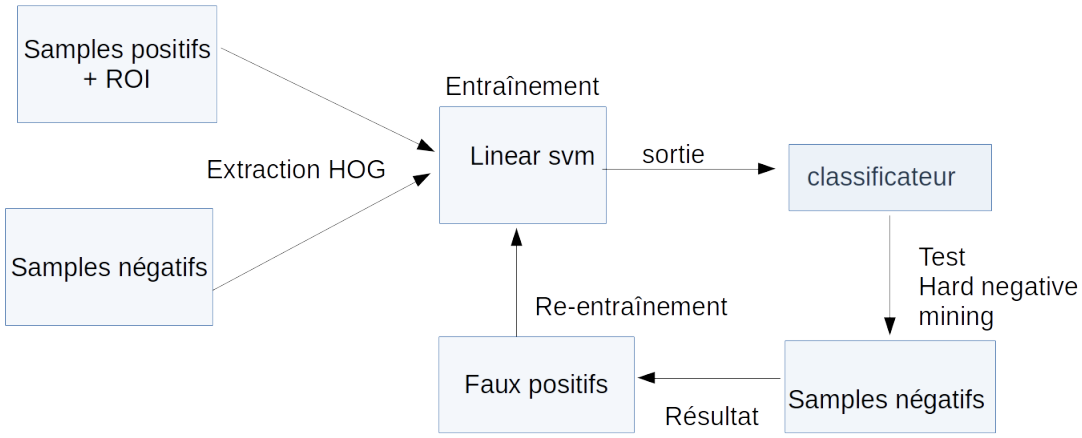


FIGURE 5.5 – Schema entrainement hog svm

défaut la méthode de normalisation de block est L2-Hys, dans la version 0.13 de scikit-image.

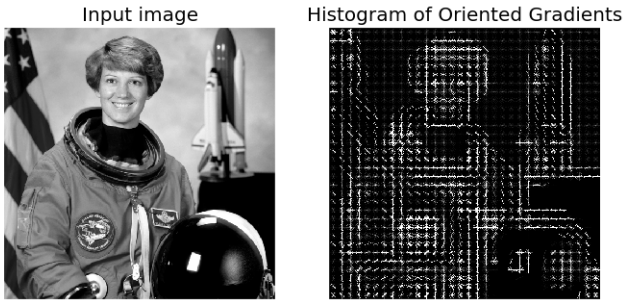


FIGURE 5.6 – calcul HOG -image source scikit-image

Entraînement

La machine à vecteur de support , prend alors en entrée un vecteur positif et un vecteur négatif et applique un hyperplan pour distinguer l' objet de ce qui n'est pas l' objet. le résultat en sortie est un classificateur

Test hard negative mining

Un test nommé hard negative mining est alors nécessaire pour affuter le classificateur. Il consiste à tester un dossier de samples négatives et extraire les faux positifs pour re-entrainer le détecteur d'objet. Il faut au moins le faire une fois pour avoir de bons résultats.

Test

Que ce soit sur des images négatives pour le hard negative mining ou sur des images positives. Un test fait appel à une combinaison de deux algorithmes pour extraire les histogrammes de gradient orienté et comparer les vecteurs avec le classificateur pour décider si on est en présence de l'objet recherché ou non :

— Fenêtre glissante
— Pyramide

Fenêtre glissante

La fenêtre glissante parcours une image de gauche à droite et de haut en bas. La taille des fenêtres est un paramètre important pour ne pas manquer l'objet. Il faut qu'elle soit adapté aux proportion de l' objet. Par exemple dans le cas d'un visage, une fenêtre carrée est appropriée. Dans le cas d'une voiture la fenêtre est un rectangle plus large que haut, la proportion est d'environ 1/3.(voir référence)

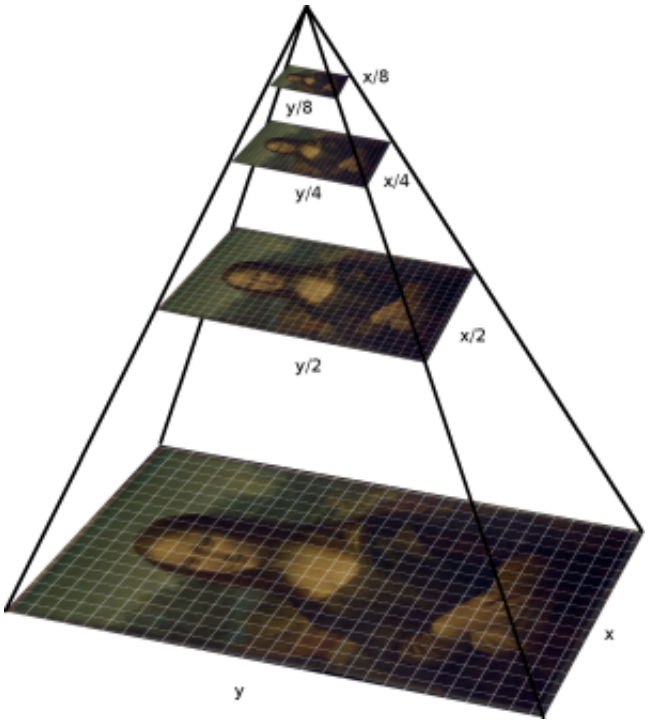


FIGURE 5.7 – Pyramide-image source pyimagesearch blog

Pyramide

La pyramide est une représentation à différentes échelles de l' image. Associé à la fenêtre glissante, ces deux algorithme permettent de détecter l' objet à différentes échelles et locations. à chaque étape l'image d'origine est réduite jusqu'à une échelle minimale fixée comme paramètre. (voir référence)

Extraction des histogrammes de gradient orientés

Les deux paramètres les plus importants sont le nombre de pixels par cellules et le nombre de cellules par blocs. La fenêtre glissante doit être divisible par ces valeurs sinon le descripteur ne sera pas ajusté aux dimensions de la fenêtre à extraire.
L'extraction des vecteurs de caractéristiques peut demander énormément de ressources mémoire. Le format hdf5 offre des solutions pour stocker d'énormes quantité de données (voir références)

Chevauchement des boîtes d'encadrement

Une fois l'objet détecté , un problème se pose du fait qu'il y a plusieurs boites d'encadrement superposés au même endroit. Pour ne garder que la plus grande

boite, il faut appliquer un algorithme comme "nms" Non maximum suppression. (voir référence)

Algorithm 3 pseudocode test HOG

```
for A in Pyramide do
  for B in FenetreGlissante do
    vecteur = extraitHOG()
    classificateur(vecteur)
    if decisionPositif then
      dessineRectangleAutour
      appliqueNMS
    end if
    affichage(trame)
  end for
end for
```

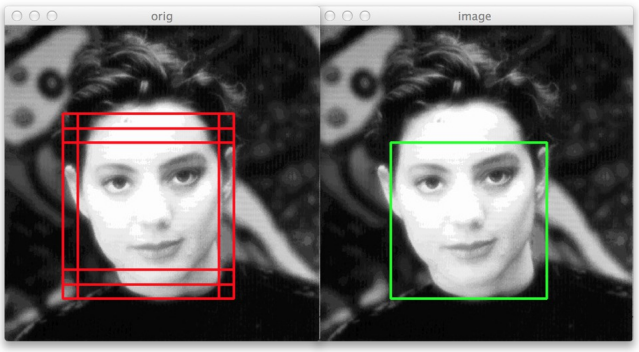


FIGURE 5.8 – nms-image source pyimagesearch blog

Resultat

Cela marche bien lorsque l'on veut détecter un objet dans une image fixe. Le problème est que ces librairies en python ne sont pas performante pour le traitement video avec un détecteur de ce type.

5.3.4 HOG et SVM (dlib)

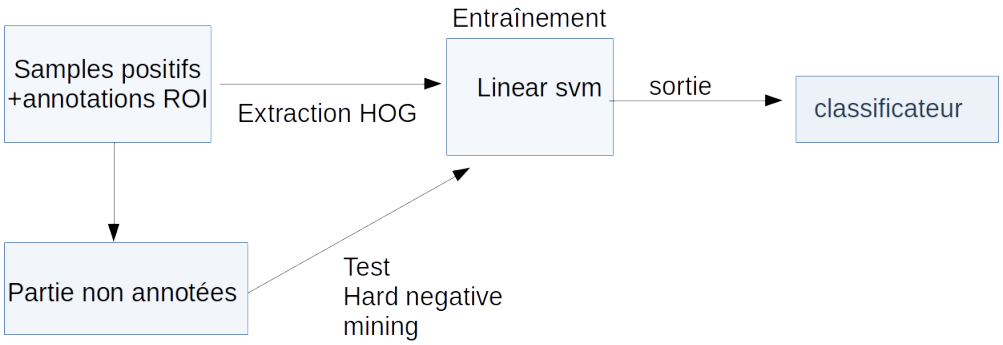


FIGURE 5.9 – Max-Margin

En 2015, Davis King créateur de la bibliothèques dlib, a proposé un autre framework pour la création d'un détecteur d'objet HOG + SVM, dans son article "Max-Margin Object Détection". Il n'y a plus besoin de donner des images négatives pour l'entraînement. Seuls les samples positifs avec les annotations des coordonnées des zones d'intérêts sont nécessaires et obligatoires. Les parties non annotées des images sont automatiquement utilisée pour faire le hard negative mining. C'est cette solution qui est implementé dans dlib.

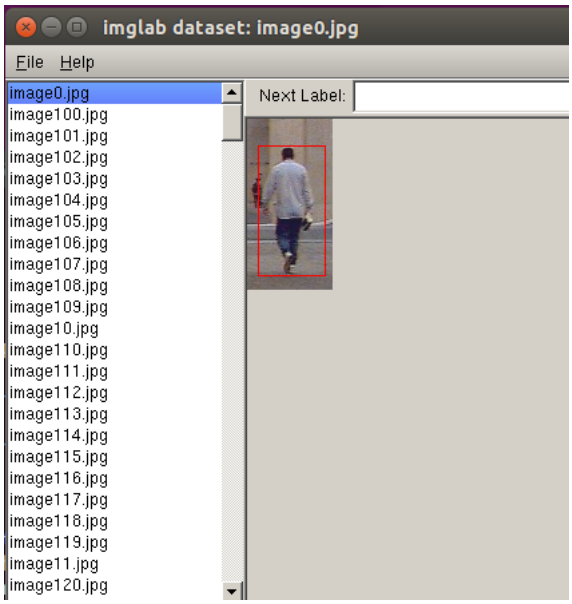


FIGURE 5.10 – outil graphique imglab

Entrainement avec dlib

Dlib propose un outil graphique nommé "imglab" qui permet de faire des annotations d'images, afin d'entrainer son détecteur d'objet. La compilation de cet outil ce fait en exécutant Cmake (voir références).
En reprenant le dossier d'images de cyclistes pris précédemment, l'enregistrement des annotations se fait sous la forme d'un fichier xml. Ce fichier xml est donnée à la méthode "train simple object detector" de dlib avec quelques options pour l'extraction, (voir annexe). Le programme en sortie donne un classificateur svm prêt à être tester sur une vidéo.

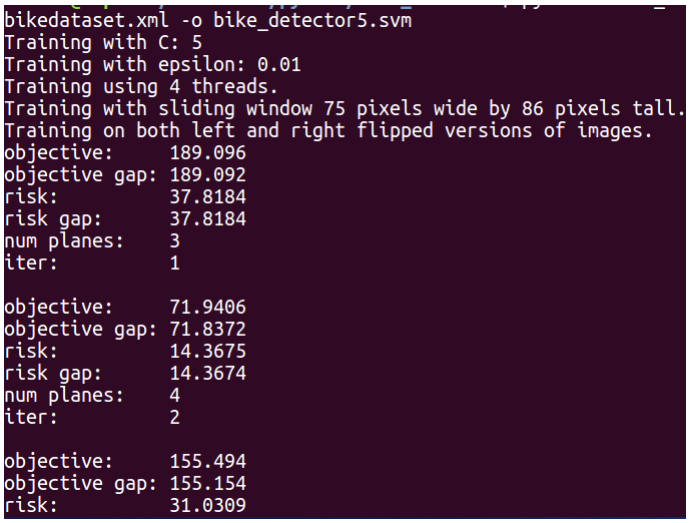


FIGURE 5.11 – Console entrainement

Test Video

Pour tester le détecteur sur une video , on exécute le script en annexe qui prend en argument d'entrée le chemin vers le classificateur crée , ainsi que le chemin vers la video de test. L'objet "dlib simple object detector" prend alors le classificateur donné et parcours chaque trames en faisant le glissement de fenêtre et la pyramide.

Résultat

Aucun faux positif et pratiquement tous les cyclistes sont détectés. La vitesse d'exécution ne souffre d'aucune latence , elle est en temps réel. Le but initial est finalement atteint .

Nouvelles données pour l'entrainement avec dlib

Le labo de vision par ordinateur du California institute of technologie offre sur son site "vision.caltech.edu" un dossier compressé d' images d'objets de 101 catégories, avec leurs annotations correspondante en format matlab. Les catégories sont assez variées. Après avoir téléchargé et extrait le dossier .tar, le dossier "car side" est chosi pour entrainer un détecteur. Il faut d'abord extraire la région d'intérêt de chaque image du dossier grâce aux annotations correspondante pour cette catégorie dans le dossier d'annotation. Les annotations sont sous format matlab. Pour lire ce format le paquet "scipy" est utile. Pour l'installer, il suffit de taper dans la terminal la commande :

```
1 pip install scipy
```

Le script en annexe, prend en entrée les chemins vers les dossiers images et annotations et le chemin de sortie où déposer le classificateur.

Résultat

Aucun faux positif, quelques voitures passent entre les gouttes, mais surtout la vitesse de traitement est en temps réels. Aucune latence comme avec les détecteurs données par opencv. Le résultat est satisfaisant. On peut utiliser les même script pour chacune des 101 catégories de caltech101 en jouant sur les paramètres pour coller au mieux au jeux de données d'entrainement .

5.4 Jeux de données d'images

Il est crucial que le jeux de données d'images utilisés pour l'entrainement du classificateur soit assez similaire aux conditions du test. Un détecteur de piétons debout ne servira pas pour détecter des personnes assises, ainsi qu'un détecteur de cycliste vu de profil ne marchera pas pour des cyclistes vu de face ou de derrière. Il est aussi important d'avoir une bonne cohérence dans le choix des images d'entrainement. Elle devront montrer la même classe d'objet dans une position similaire. Le classificateur fera un choix en fonction de la représentation moyenne qu'il aura d'après le jeux de donnée pour l'entrainement. On ne fait pas de miracles avec de mauvaises données.

5.5 Bilan technique

La méthode de soustraction de l'arrière plan et détection de contours s'est révélée trop instable pour être utilisés dans notre contexte. La méthode haar cascades peut être envisagée , mais cela demande un gros travail de récoltes et de prétraitement des images . Plusieurs milliers d'images semble être nécessaires pour entraîner un bon classificateur. Obtenir le fichier xml final peut prendre dans ces conditions plusieurs semaines de calculs. La combinaison HOG SVM pour la détection d'un objet donne de très bonnes performances. Opencv offre un détecteur de piéton par défaut mais peu de documentation sur comment créer son propre détecteur d'objet. Scikit donne beaucoup de possibilité pour faire ce détecteur sur mesure mais ne permet pas le traitement en temps réel. Pour la détection d'un objet en mouvement, la bibliothèque dlib offre un très bon compromis entre facilité d'implémentation, Nombre d' échantillons nécessaire , temps d'entrainement, vitesse de traitement sur les trames et robustesse.

Chapitre 6

Difficultés rencontrées

La première difficulté , a été de trouver les informations permettant d'avoir une vision d'ensemble du domaine, dans le le web anglophone. Puis de trouver et d'apprendre à utiliser les bibliothèques open source pour réaliser un détecteur d'objet robuste. J'ai passé plusieurs semaines à chercher comment créer un détecteur HOG SVM avec opencv alors que ça se fait plus facilement avec scikit et dlib. Il faut trouver un équilibre entre poursuivre sur une idée et essayer autre chose. L'extraction de caractéristiques peut être très couteux en mémoire, il m'est arrivé plusieurs fois en extrayant les HOG d'un jeux d'images négatives, de faire planter la machine car j'avais dépassé la taille max de mémoire allouée. La récolte et l'annotation d'images est quelque chose qui peut prendre énormément de temps et les bibliothèques d'images ne sont pas toujours adaptées à ce que l'on cherche à détecter et ne sont pas nécessairement annotées.

Chapitre 7

Conclusion

A la fin de ce travail de Bachelor, j'ai découvert un vaste champ d'études fort intéressantes autour de la vision par ordinateur et du machine learning avec de nombreuses applications à développer. J'ai acquis des compétences avec les librairies opencv ,scikit, dlib et j'ai augmenté mes connaissance en python. Tout ceci me sera utile pour des projets personnels ou professionnels.

Chapitre 8

Références

Bibliothèque Dlib site principal
Dlib training options pour le détecteur d'objet
opencv train cascades
Article Dalal and Triggs HOG for human detection
Outil dlib imglab
wikipedia hog
opencv hog
scikit hog
Bibliotheque hdf5 pour le stockage de donnée avec python
sckit-image site principal
sckit-learn site principal
opencv site principal
Bibliothèque boost
Bibliothèque boost python
Article max-Margin object detection
pyimagesearch nms
pyimagesearch pyramide
pyimagesearch fenêtre glissante
Article viola-jones
Blog coding robin
Cours détection d'objets coursera
site opencv soustraction de l'arrière plan

Chapitre 9

Annexes

9.1 Scripts d'installation

```
#installation de openCV 2.4

wget -O opencv-2.4.10.zip http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.10/opencv-2.4.10.zip/download
unzip opencv-2.4.10.zip
cd opencv-2.4.10

mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_NEW_PYTHON_SUPPORT=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON ..
make -j4
sudo make install
sudo ldconfig
```

```
#installation de dépendence python 2.7
sudo apt-get install python2.7-dev

pip install numpy

pip install scipy matplotlib
#installation de scikit image et scikit learn
pip install scikit-learn
pip install -U scikit-image
```

```
## mise a jour et installation de divers paquet de boost et de boost-python

sudo apt-get update
sudo apt-get upgrade

sudo apt-get install build-essential cmake pkg-config

sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install libgtk2.0-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libatlas-base-dev gfortran
sudo apt-get install libboost-all-dev
wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
```

```
### installation dlib sur ubuntu, aller sur la home page chercher le paquet
# l'extraire

tar xvjf dlib-19.4.tar.bz2
cd dlib-19.4/python_examples/
mkdir build
cd build
cmake ../../tools/python
cmake --build . --config Release --target install
cd ..
ls -l dlib.so
```

9.2 Code soustraction arrière plan

```
# installer le paquet numpy pour la manipulation de matrices
import numpy as np
import cv2

#capture d'une video ou d'une camera IP avec VideoCapture

cap = cv2.VideoCapture("adresse_IP_ou_chemin_vers_la_video")

# création du filtre
fgbg = cv2.BackgroundSubtractorMOG()

while(1):
```

```
13      # lecture
14      ret, frame = cap.read()
15      # application du masque
16      fgmask = fgbg.apply(frame)
17      # affichage
18      cv2.imshow('frame',fgmask)
19      k = cv2.waitKey(30) & 0xff
20      if k == 27: #ESC
21          break
22
23  cap.release()
24  cv2.destroyAllWindows()
```

9.3 code houg circle

```
1   import cv2
2   import numpy as np
3
4
5   vc = cv2.VideoCapture("http://admin:cam6493@10.10.7.205/video2.mjpg")
6
7   while (vc.isOpened() == 1):
8
9       ret,frame = vc.read()
10      gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
11      blur = cv2.medianBlur(gray,5)
12      circles = cv2.HoughCircles(blur,cv2.HOUGH_GRADIENT,1.5,10)
13      #circles = cv2.HoughCircles(gray,cv2.HOUGH_GRADIENT,1.5,10)
14      circles = np.uint16(np.around(circles))
15      for i in circles [0,:]:
16          cv2.circle(frame,(i[0], i[1]),i[2],(255,0,0) ,2)
17          cv2.circle(frame,(i[0], i[1]) ,2,(0,255,0) ,5)
18
19      cv2.imshow("HoughCircles",frame)
20      key = cv2.waitKey(1)
21      if key == 27:
22          break
23
24  vc.release()
25  cv2.destroyAllWindows()
```

9.4 code ORB test

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('drink.jpg',0)
# Initiate ORB detector
orb = cv2.ORB_create()
# find the keypoints with ORB
kp = orb.detect(img,None)
# compute the descriptors with ORB
kp, des = orb.compute(img, kp)
# draw only keypoints location,not size and orientation
img2 = cv2.drawKeypoints(img, kp, None, color=(0,255,0), flags=0)
cv2.imshow('drink',img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

9.5 code haarcascade

```
import cv2
import numpy as np

people_classifier = cv2.CascadeClassifier('haarcascade_fullbody.xml')


cap = cv2.VideoCapture('video1.mp4')

while cap.isOpened():

    ret, frame = cap.read()
    # redimmentionnement de trames
    frame = cv2.resize(frame,None, fx=0.5, fy=0.5, interpolation=
    cv2.INTER_LINEAR)
    #Transformation de la trame couleur en niveau de gris
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    people = people_classifier.detectMultiScale(gray,1.5,2)

#dessine un rectangle jaune autour de l'objet détecté et affiche la trame
    for (x,y,w,h) in people:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)
        cv2.imshow('people',frame)
        # exit
    if cv2.waitKey(1) == 27:
```

```
26          break
27
28  cap.release()
29  cv2.destroyAllWindows()
```

9.6 code HOG SVM openCV

```
1   #!/usr/bin/env python
2
3
4   # Python 2/3 compatibility
5   from __future__ import print_function
6
7   import numpy as np
8   import cv2
9   import imutils
10
11  def draw_detections(img, rects, thickness = 1):
12      for x, y, w, h in rects:
13          # the HOG detector returns slightly larger rectangles than the real
                objects.
14          # so we slightly shrink the rectangles to get a nicer output.
15          pad_w, pad_h = int(0.15*w), int(0.05*h)
16          cv2.rectangle(img, (x+pad_w, y+pad_h), (x+w-pad_w, y+h-
                pad_h), (0, 255, 0), thickness)
17
18
19  if __name__ == '__main__':
20
21      hog = cv2.HOGDescriptor()
22      hog.setSVMDetector( cv2.HOGDescriptor_getDefaultPeopleDetector() )
23      cap = cv2.VideoCapture('videos.mp4')
24
25      while cap.isOpened():
26          ret, frame = cap.read()
27          frame = imutils.resize(frame,width=400)
28          gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
29
30          found, w = hog.detectMultiScale(gray, winStride=(8,8), padding
                =(32,32), scale=1.05)
31          draw_detections(frame, found)
32          cv2.imshow('people', frame)
33          ch = cv2.waitKey(30) & 0xff
34          if ch == 27: #Esc
```

```
35                 break
36         cap.release()
37         cv2.destroyAllWindows()
```

9.7 code dlib entrainement velo

```
1  #USAGE
2  # python train_bike_detector.py -a bikedataset.xml -o bike_detector.svm
3  # import des paquets
4  import os
5  import sys
6  import glob
7  import dlib
8  import argparse
9
10 # arguments du programme
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-a","--annotations",required = True,
13                 help="chemin vers le fichier xml")
14 ap.add_argument("-o","--output",required = True,
15                 help = "chemin et nom du fichier .svm a creer")
16 args = vars(ap.parse_args())
17
18 #chargement des options
19 option = dlib.simple_object_detector_training_options()
20
21 #pivoter les images a extraire
22 option.add_left_right_image_flips = True
23 #fixe la marge de decision
24 option.C = 5
25 #nombre de threads
26 option.num_threads = 4
27 #verbeux
28 option.be_verbose = True
29 # entrainement du detecteur de cyclistes
30 dlib.train_simple_object_detector(args["annotations"],args["output"],option)
```

9.8 code HOG svm extraction images caltech

```
1  # USAGE
2  # python train_detector.py --class car_side --annotations car_side_a
```

```
3  #        --output output/car_detector.svm
4  # import des paquets
5  from __future__ import print_function
6  from imutils import paths
7  from scipy.io import loadmat
8  from skimage import io
9  import argparse
10 import dlib
11
12 # arguments du programme
13 ap = argparse.ArgumentParser()
14 ap.add_argument("-c", "--class", required=True,
15         help="chemin vers la classe d'images")
16 ap.add_argument("-a", "--annotations", required=True,
17         help="chemin vers la classe d'annotations")
18 ap.add_argument("-o", "--output", required=True,
19         help="chemin pour enregitrer le detecteur")
20 args = vars(ap.parse_args())
21
22 # prise des options par defaut pour HOG + Linear SVM et initialise la
23 # liste d'images et de boites d'encadrement pour entrainer le classificateur
24 print("[INFO] recoltes images et boite d'encadrement...")
25 options = dlib.simple_object_detector_training_options()
26 images = []
27 boxes = []
28
29 # boucle sur la classe d'images
30 for imagePath in paths.list_images(args["class"]):
31         # extraction de ID des images et chargement des fichiers d'
              annotations
32         imageID = imagePath[imagePath.rfind("/") + 1:].split("_")[1]
33         imageID = imageID.replace(".jpg", "")
34         p = "{}/annotation_{}.mat".format(args["annotations"], imageID)
35         annotations = loadmat(p)["box_coord"]
36
37         # boucle sur les annotations et ajout de chaque annotation a la
              liste des boites
38         # d'encadrement
39         bb = [dlib.rectangle(left=long(x), top=long(y), right=long(w),
              bottom=long(h))
40                 for (y, h, x, w) in annotations]
41         boxes.append(bb)
42
43         # ajout a la liste d'images
44         images.append(io.imread(imagePath))
45
```

```
# entrainement du detecteur
print("[INFO] entrainement du detecteur...")
detector = dlib.train_simple_object_detector(images, boxes, options)
print("[INFO] options du detecteur...")
print(dir(detector))
# depot du classificateur dans le fichier de sortie
print("[INFO] depot du classificateur dans le fichier de sortie ... ")
detector.save(args["output"])

# visualisation du resultat
win = dlib.image_window()
win.set_image(detector)
dlib.hit_enter_to_continue()
```

9.9 code detecteur dlib test video

```
 #usage
# python object_detector.py --detector output/bike_detector.svm
# --video bike_vid/bikes1.mp4
# importation des paquets
import dlib
import cv2
import imutils
import argparse

# arguments du programme
ap = argparse.ArgumentParser()
ap.add_argument("-d","--detector",required = True,
                help="chemin vers le detecteur svm")
ap.add_argument("-v","--video",required = True,
                help = "chemin vers le fichier video")
args = vars(ap.parse_args())

# initialisation des objets
detector = dlib.simple_object_detector(args["detector"])

cap = cv2.VideoCapture(args["video"])

# lecture des trames
while cap.isOpened():

    ret, frame = cap.read()
    if not ret:
        break
```

```
29      # redimensionnement des trames
30      frame = imutils.resize(frame,width=400)
31      #application du detecteur a chaque trames
32      det = detector(frame)
33
34      # si objet, dessine rectangle d'encadrement en vert
35      for d in det:
36          cv2.rectangle(frame, (d.left(), d.top()), (d.right(), d.bottom()),
                (0, 255, 0), 2)
37
38      # affichage et touche de sortie
39      cv2.imshow("frame",frame)
40      if cv2.waitKey(1) & 0xFF == ord('q'):
41          break
42
43  #libere les ressources
44  cap.release()
45  cv2.destroyAllWindows()
```