SY32 Projet : Détecteur de personnes

Luyue Lin, Matthieu Zins

22/05/2016

1 Introduction

L'objectif de ce projet est de réaliser un détecteur de personnes en utilisant l'environnement MATLAB.
Nous avons décomposé notre détecteur en 6 étapes principales :

- Préparation des images
- Extraction des boîtes positives et négatives
- Premier apprentissage
- Extraction des faux positifs
- Réapprentissage
- Détection

Par la suite, nous allons détailler plus précisement ces différentes parties.

2 Préparation des images

Le but de cette première étape est de préparer les images d'apprentissage et les images de test afin de les adapter à la détection de personnes. Dans un premier temps, nous avons chargé les labels des images d'apprentissage, les images de test et celles d'apprentissage dans des cellules. Les images (apprentissage et test) sont ensuite converties en niveaux de gris et normalisées. On obtient les cellules suivantes *images_train_n* et *images_test_n*. Par la suite, nous utiliserons uniquement ces images. Pour pouvoir afficher correctement des images normalisées, nous avons développé la fonction *aff_img_n*. De plus la matrice *labels* contient ligne par ligne les coordonnées et les tailles des personnes identifiées sur les images d'apprentissage.

3 Extraction des boîtes positives et négatives

Le but de cette partie est d'extraire des boîtes positives et négatives à partir des images d'apprentissage. La boîte la plus petite de l'ensemble d'apprentissage a une hauteur de 66 pixels. Pour respecter le rapport $\frac{hauteur}{largeur} = 2$ nous avons donc choisi d'utiliser des boîtes de largeur de 33 pixels et hauteur de 66 pixels.

3.1 Exemples positifs

L'extraction des boîtes positives se fait facilement à partir des labels des différentes images. Lorsque la boîte n'avait pas un rapport $\frac{hauteur}{largeur}$ exactement égal à 2, nous avons modifié la largeur et la coordonnée horizontale pour obtenir une boîte ayant une largeur deux fois plus grande que la hauteur tout en restant centré sur la personne. Il a également fallu gérer les coordonnées négatives et le fait que les boîtes peuvent dépasser de l'image. Une fois que l'on a extrait les boîtes positives, elles ont toutes le même rapport $\frac{hauteur}{largeur}$ et nous les avons donc mises à l'échelle afin qu'elles aient toutes la même taille (hauteur=66 et largeur=33). Enfin, nous avons extrait les caractéristiques HOG de chaque boîte avec la fonction **extractHOGFeatures**. Nous avons utilisé les paramètres suivants afin d'obtenir des vecteurs de taille acceptable (ni trop petit, ni trop grand) :

- taille des cellules = [8,8]
- nombre de bins = 9
- utilisation des orientations signées = true

3.2 Exemples négatifs

Pour extraire des boîtes négatives, nous avons généré des positions et des tailles aléatoires dans toute l'image. Nous avons également vérifié qu'elles ne recouvrent pas la boîte positive en vérifiant que :

$$\frac{Aire(boite_{aleat} \cap boite_{vraie})}{Aire(boite_{aleat} \cup boite_{vraie})} < 0.5$$

Ensuite, comme pour les boîtes positives, nous les avons redimensionnées à la taille 33x66 et extrait leurs caractérisitques HOG.

Ces caractéristiques HOG des boîtes positives et négatives sont enregistrées respectivement dans *boites_pos_hog* et *boites_neg_hog* ligne par ligne. Pour avoir plus d'images d'apprentissage, nous avons ajouté, pour chaque boîte, sa symétrie verticale. Cela nous permet d'obtenir 400 exemples positifs et 4000 exemples négatifs pour notre apprentissage.

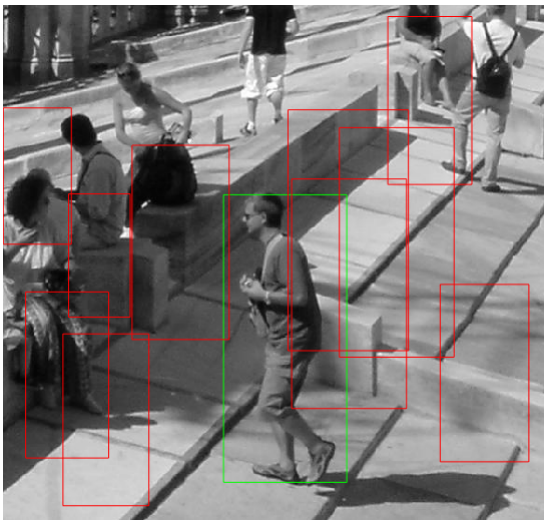

FIGURE 1 – Exemple d'extraction de la boîte positive (vert) et des boîtes négatives (rouge)

4 Premier apprentissage

Dans cette partie, nous avons appris un premier modèle en utilisant une approche par SVM. La première étape consiste à concaténer nos caractéristiques HOG positives et négatives en une seule matrice et de créer le vecteur contenant les étiquettes correspondantes. Ensuite, nous avons utilisé la validation croisée pour déterminer le meilleur règlage du paramètre **BoxConstraint**. Pour la validation croisée, nous avons séparé les données d'apprentissage en 5. En réduisant, petit à petit, l'intervalle de recherche nous avons obtenu un paramètre de 2^{-2} qui minimise l'erreur de classification (environ 3.14%). Une fois ce paramètre déterminé, nous avons appris le modèle sur toutes les données d'apprentissage.

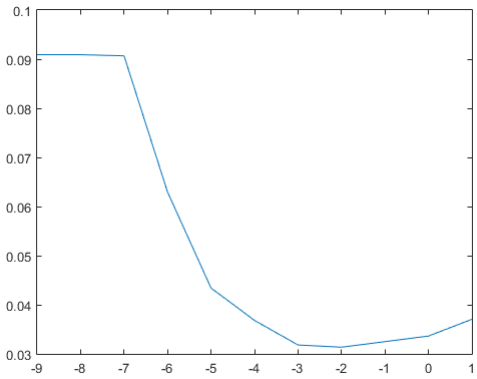

FIGURE 2 – Détermination du meilleur paramètre BoxConstraint par validation croisée

5 Extraction des faux positifs

Dans cette partie, nous avons utilisé le modèle appris précédemment sur nos images d'apprentissage. Nous avons utilisé la méthode de la fenêtre glissante sur chaque image avec différentes échelles. En comparant les hauteurs minimales et maximales des vraies boîtes, nous avons décidé d'utiliser les échelles suivantes : $\left[\frac{1}{13}, \frac{1}{12}, \frac{1}{11}, \frac{1}{10}, \frac{1}{9}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{3}{4}, 1\right]$. Pour le choix des pas de balayage horizontaux et verticaux, nous avons choisi de les calculer en fonction de l'échelle afin que la fenêtre puisse être déplacée 15 fois dans chaque direction. Pour chaque détection positive de personne, nous avons déterminé si la boîte recouvrait la vraie boîte à plus de 50%. Si c'est le cas, nous avons considéré qu'il s'agissait d'un vrai positif mais si ce n'était pas le cas, nous l'avons enregistrée comme faux positif. Nous avons donc pu enregistrer les caractérisitques HOG de toutes les boîtes fausses positives.

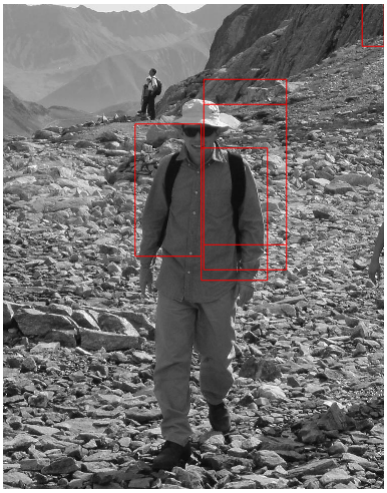

FIGURE 3 – Exemple d'extraction de faux positifs

6 Réapprentissage

Le but de cette partie est de réapprendre le modèle en ajoutant les boîtes fausses positives extraites dans la partie précédente en tant qu'exemples négatifs. Comme pour le premier apprentissage, nous avons fait la validation croisée pour déterminer le paramètre **BoxConstraint**. On obtient un paramètre de 2^{-2} qui minimise l'erreur de classification (environ 2.5%)

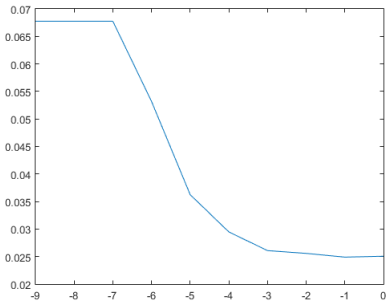

FIGURE 4 – Détermination du meilleur paramètre BoxConstraint par validation croisée pour le deuxième apprentissage

7 Détection

Dans cette dernière partie, il s'agit de faire la détection par fenêtre glissante sur les images de test en utilisant le modèle que l'on vient d'apprendre. Nous avons utilisé les mêmes échelles et les mêmes pas de balayage que lors de la phase d'extraction des faux positifs. Plusieurs détections ayant lieu, sur une image, nous avons gardé celle avec le score le plus élevé car elle correspond à celle pour laquelle le classificateur a le plus de certitudes.

FIGURE 5 – Exemple de détection pour une image de test

8 Conclusion

En affichant les zones détectées sur les images de test, on peut se rendre compte que notre détecteur fonctionne globalement bien. Dans les différentes étapes de notre détecteur, nous avons choisi des paramètres en fonction des ressources disponibles et en essayant d'obtenir des résultats dans un temps acceptable. Notre détecteur pourrait être amélioré avec plus de données d'apprentissage, des caractéristiques HOG plus détaillées, plus d'échelles testées et des pas plus petits lors de la fenêtre glissante.