SY32 - Projet détection de visage

Luc Varoqui - Groupe 6

Avril 2019

Table des matières

1 Introduction

Ce projet s'inscrit dans le cadre du cours de SY32 (Analyse et synthèse d'image). L'objectif est de mettre en application les concepts vus en cours, TD et TP et de créer un détecteur de visage. Pour cela nous utiliserons le langage de programmation Python et les librairies numpy (manipulation de matrice), scikit-image (manipulation d'image) et scikit-learn (apprentissage automatique).

Les classifieurs cités dans ce rapport se trouvent dans le dossier *classifiers*.

2 Fonctionnement du détecteur

Pour pouvoir détecter des visages dans des images, plusieurs étapes vont être nécessaires :

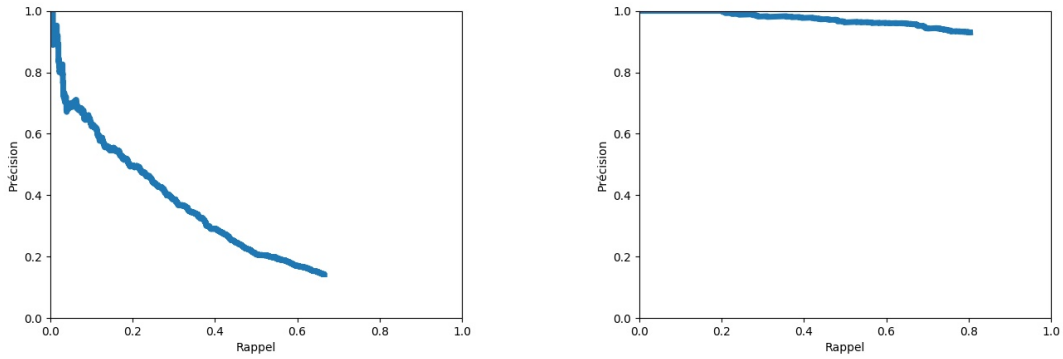
- Extraire tous les visages des images d'entraînement
- Générer des faux positifs à partir des images d'entraînement
- Générer les HOGs (vecteurs descripteurs) pour les vrais et les faux positifs
- Entraîner le classifieur intermédiaire avec ces données
- Scanner les images d'apprentissage avec ce classifieur intermédiaire et une fenêtre glissante
- Extraire les faux positifs du classifieur intermédiaire et ajouter leurs HOGs aux données d'apprentissage comme négatifs
- Entraîner le classifieur final à partir de ces données
- Scanner les images d'apprentissage avec le classifieur final et une fenêtre glissante

2.1 Vecteur descripteur

Pour pouvoir entraîner les classifieurs, il faut choisir un vecteur descripteur pour chaque échantillons, positifs et négatif. J'ai décidé d'utiliser le HOG (Histogram of Oriented Gradient) que nous avons vu en cours. Les visages dans l'ensemble d'entraînement ont un rapport hauteur/largeur moyen de 1,5. On extrait donc des échantillons de tailles 60x40 pixels. Ainsi tout HOG calculé sur un échantillon est de dimension 1215. Les résultats avec le HOG s'étant montrés plus que satisfaisant, je n'ai pas essayé d'autre type de vecteur descripteur.

2.2 Deux classifieurs ?

On peut voir que l'on utilise deux classifieurs. On utilise les erreurs du premier pour améliorer les résultats du second. Voici les courbes précision/rappel des classifieurs intermédiaire et final du détecteur *svm_ linear_ 3* :



(a) Classifieur intermédiaire

(b) Classifieur final

FIGURE 1 – Comparaison des courbes Précision/Rappel pour le détecteur *svm_ linear_ 3*

Ces deux courbes montrent bien que le classifieur final est beaucoup plus performant, il a une meilleure précision et un meilleur rappel que le classifieur intermédiaire.

2.3 Fenêtre glissante

Une fois le classifieur entraîné, on utilise une fenêtre glissante pour scanner les images. Les tailles des visages varient, il faut donc faire de multiples passage sur l'image avec différentes tailles de fenêtre. En pratique, on change la taille de l'image.

Dans un premier temps j'ai utilisé les paramètres suivant : *pas horizontal*, *pas vertical*, *diviseur maximum* et *pas de division*. J'obtenais de bon résultats en terme de précision mais de mauvais en terme de rappel. En effet, le diviseur maximum étant défini pour le lot d'image en entier, le régler trop haut provoquait des erreurs car certaines images était plus petite que la fenêtre de 60x40. Les visages les plus grand n'étaient donc pas détectés.

Pour palier ce problème, j'ai changé les paramètres *diviseur maximum* et *pas de division* en un seul : *nombre de division*. Ce paramètre défini le nombre de changement d'échelle à faire pour chaque image. Ainsi, le diviseur maximum est calculé pour chaque image, cela permet de fortement augmenter le rappel.

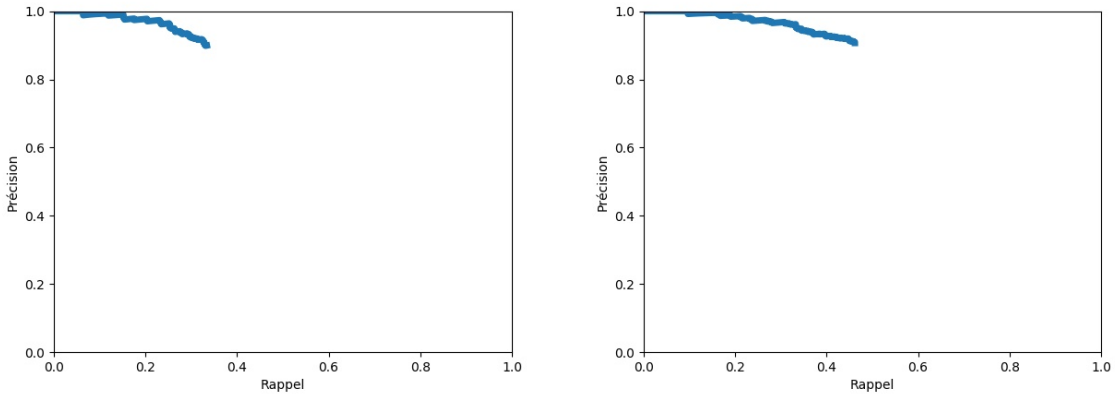
Les scores des détecteurs *svc_ kernel_ rbf_ 1* et *svc_ kernel_ rbf_ 2* ci-dessous illustrent la monté en performance dû à ce changement. Toutes choses égales par ailleurs, le premier utilise l'ancien système de changement d'échelle et l'autre le nouveau. C'est ce changement qui distingue mon premier et deuxième envoi sur les données de test.

	Ancienne version	Nouvelle version
Précision	0.98	0.98
Rappel	0.46	0.81
F-Score	0.63	0.88

TABLE 1 – Comparaison des systèmes de changement d'échelle

Pour déterminer mes paramètres (*pas horizontal*, *pas vertical* et *nombre de division*) j'ai fait de l'essai erreur pour aboutir à des pas horizontaux et verticaux de 8 et un nombre de division de 10.

Voici, toutes choses égales par ailleurs, les courbes de précision/rappel avec des pas de 15 et de 10 (ces deux test utilisaient encore l'ancien système de changement d'échelle). Comme on pouvait s'y attendre, un pas plus petit augmente le rappel, cependant cela augmente aussi le temps d'exécution.



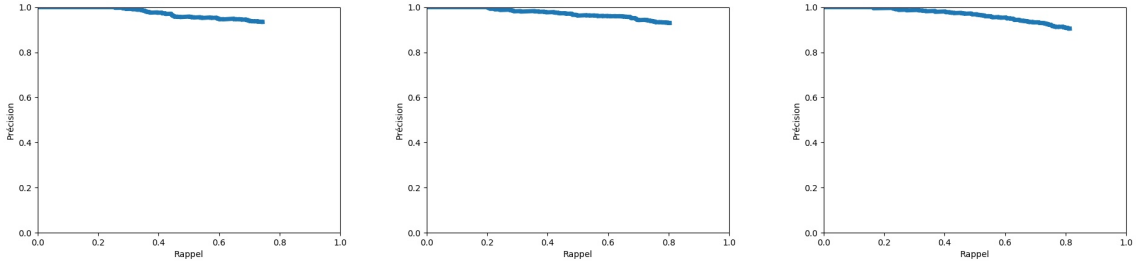
(a) Pas horizontal et vertical de 15 (*svm_linear_0*) (b) Pas horizontal et vertical de 10 (*svm_linear_1*)

FIGURE 2 – Comparaison des courbes Précision/Rappel en fonction du pas de la fenêtre glissante

2.4 Choix du classifieur

Pour choisir un classifieur, je me suis basé sur deux métrique : les résultats de validation croisée et les courbes de précisions/rappels.

Dans un premier temps j'ai comparé le classifieur LinearSVC avec différentes valeurs de C. Mes trois tests ont donné des résultats très proches :



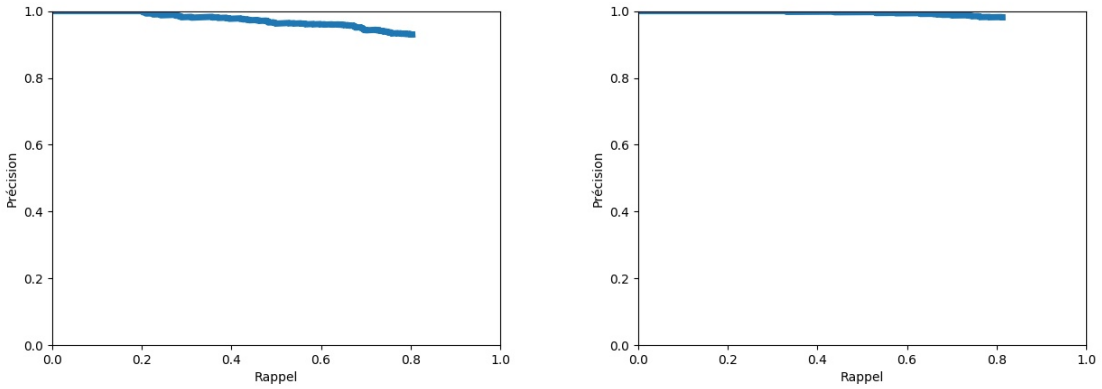
(a) C = 0.1 (*svm_linear_4*) (b) C = 1 (*svm_linear_3*) (c) C = 3 (*svm_linear_5*)

FIGURE 3 – Comparaison des courbes Précision/Rappel pour un classifieur linéaire avec différentes valeurs de C

	C=0.1	C=1	C=3
Score de validation croisée moyen	0.9831	0.9881	0.9890
Precision	0.9344	0.9047	0.9047
Rappel	0.7437	0.8138	0.8138
F-Score	0.8282	0.8569	0.8569

TABLE 2 – Comparaison des scores pour un classifieur linéaire avec différentes valeurs de C

J'ai ensuite testé le classifieur SVC avec un noyau RBF. Il s'est montré extrêmement performant. C'est donc celui-ci que j'ai utilisé pour l'ensemble de test. Comme chaque test prend beaucoup de temps j'ai décidé d'arrêter là mes essais. Voici les résultats du classifieur SVC avec noyau RBF comparé au LinearSVC avec C = 1.



(a) LinearSVC (C = 1) (*svm_linear_3*) (b) SVC (noyau RBF) (*svc_kernel_rbf_2*)

FIGURE 4 – Comparaison des courbes Précision/Rappel entre classifier linéaire et à noyau

	SVC linéaire (C = 1)	SVC (noyau RBF)
Score de validation croisée moyen	0.9881	0.9880
Precision	0.9047	0.9812
Rappel	0.8138	0.8130
F-Score	0.8569	0.8892

2.5 Performances

La recherche de visage par fenêtre glissante est lente, le HOG et la prédiction du classifieur doivent être calculé de multiple fois pour chaque image, pour l'ensemble de test mon algorithme mettait environ 10s à traiter chaque photo. J'ai donc décidé de paralléliser l'étape de fenêtre glissante et d'utiliser mon ordinateur de bureau, resté à Compiègne, pour faire les calculs à distance via SSH. Cela m'a permis de diviser par 10 le temps nécessaire à l'exécution de la fenêtre glissante sur un lot d'image, j'ai donc pu faire plus de test.

3 Instructions pour l'exécution du code

Mon code nécessite les paquet Python suivant : numpy, scikit-learn, scikit-image, joblib (sauvegarde du classifieur), tqdm (barre de progression).

Lors de l'appel avec python de mes programmes, sous certaines version de scimage des warning peuvent apparaître. Je recommande l'utilisation de *-W ignore* dans ce cas.

L'entraînement du classifieur se fait grâce au fichier *train.py*. Avant de le lancer, il faut modifier la configuration grâce à la section CONFIG dans le code, le paramètre "config_name" doit être modifié pour éviter les conflits avec d'autres configurations du dossier *classifiers*. Il suffit ensuite d'exécuter le fichier avec python3 en ligne de commande en veillant à ce que le dossier *classifiers* existe. Le programme y créera automatiquement un nouveau dossier avec le nom de la configuration.
Note importante : J'ai renommé l'image et le label numéro 1000 en numéro 0 pour que les numéro d'image correspondent à leurs index dans python.

Pour exécuter un classifieur sur l'ensemble de test, il suffit de spécifier le chemin vers le classifieur dans le fichier *test.py* et de l'exécuter avec python3 en ligne de commande. Le fichier *detections.txt* sera automatiquement créé à la racine du projet. Le classifieur ayant donné le dernier résultat posté sur le site est *classifiers/svc_kernel_rbf_2/clf.joblib*.

4 Conclusion

J'ai beaucoup aimé travailler sur ce projet, j'ai pu mettre en application les différents concepts du cours : apprentissage automatique, validation croisée, courbes précision/rappel, fenêtre glissante... J'ai également progressé en Python, découvert de nouvelles librairies et appris à paralléliser des tâches. Mon détecteur a obtenu de bons résultats sur l'ensemble de test. Le temps d'exécution reste un problème majeur et il serait intéressant de travailler à son optimisation.