DÉTECTEUR DE VIOLA JONES

Projet
INF 442

6 juillet 2016

Responsable du projet : Jean Baptiste Bordes
Felipe Garcia
Francisco Eckhardt

ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

TABLE DES MATIÈRES

1
PRÉSENTATION DU SUJET

L'objectif de notre projet est d'implémenter un détecteur de visages basé sur les caractéristiques introduites par Viola et Jones. Pour cela, on utilise des vecteurs de caractéristiques qui correspondent à un grand nombre de régions et tailles différentes possibles qu'on calculera pour plusieurs images. On les utilise à la fin dans la prise de décision (détection).

Dans le cadre de l'analyse des images nous avons utilisé la bibliothèque Open Source `CImg` pour accéder aux pixels et transformer les fichiers `.jpg` en matrices de `long` en C++. Ces matrices sont converties en Images Intégrales pour un traitement postérieur.

La deuxième partie de notre projet ce concentre sur la création des classifieurs qui discernent entre images de visages et de non visages. Nous avons utilisé l'algorithme du perceptron pour leur apprendre cette tâche. Ces résultats préliminaires sont ensuite pris par AdaBoost pour augmenter le pourcentage de détection de faces.

Dans la phase finale ce centre sur la discussion de la précision de notre classifieur final en fonction du paramètre extérieur θ par une analyse de faux positifs et faux négatifs. Puis nous faisons une extension de notre algorithme pour repérer des visages dans des images quelconques.

2
EXPLICATION DE L'ALGORITHME

2.1 Calcul de l'Image Intégrale

On ouvre l'image avec l'aide de la bibliothèque `CImg` comme une matrice de `long` , on la parcourt une seule fois (fonction **`SAT`**()), avec deux boucles `for` , l'une pour la hauteur et l'autre pour la largeur, en calculant l'image intégrale. On a utilisé la fonction **`printData`**() pour en tester le bon fonctionnement dans le fichier `test_q_1_1.cpp` .

2.2 Définition des caractéristiques en parallèle

On a créé la classe auxiliaire feature pour représenter une caractéristique, avec les champs représentant position, largeur, hauteur et type de caractéristique (conforme à la Figure 1).

Pour créer des différentes caractéristiques, on a imposé aux côtés des rectangles une taille minimale de 8 pixels et on a pris un incrément minimal de position et de taille de 4 pixels. La fonction **`distFeat`**() lance 4 threads responsables de créer les caractéristiques de chaque type ('a', 'b', 'c' et 'd'), puis les retourne dans un même vecteur. On a ainsi obtenu 128227 caractéristiques au total.

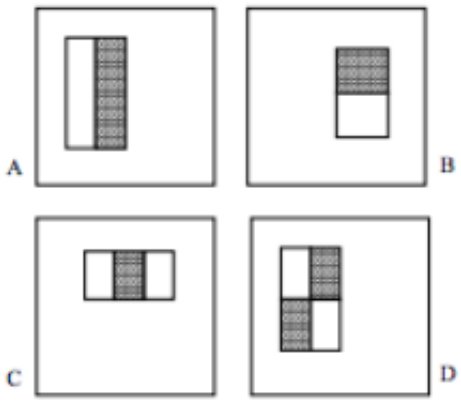

Figure 1 – Types des caractéristiques utilisées

2.3 Entraînement des classifieurs faibles en parallèle

À chaque caractéristique on associe un classifieur faible (h_i, représentés par la classe auxiliaire classifier, avec deux paramètres w_1 et w_2 et la fonction `calc(x)` qui renvoie $h(x)$, conforme défini dans le sujet). Chaque classifieur faible est entraîné en utilisant la méthode du Perceptron.

Les fonctions auxiliaires **`readImgs`**() et **`distII`**() sont responsables d'ouvrir et de calculer l'image intégrale des images d'un répertoire de manière parallélisée, ce qu'on a utilisé pour les trois répertoires "Train", "Validation" et "Test". De même pour l'entraînement : on distribue entre les `threads` différents classifieurs pour qu'ils les travaillent parallèlement.

2.4 Boosting et choix de classifieurs

On utilise l'algorithme Adaboost pour créer un classifieur final en utilisant une combinaison linéaire de classifieurs faibles précédemment définies. L'idée consiste en donner plus de poids pour les classifieurs les plus performants et ainsi obtenir un classifieur final qui les prend en compte de façon pondérée. Les étapes de choix du classifieur faible et de mise à jour des poids (λ_i pour $i \in \{1, \ldots, n\}$) sont faites en parallèle.

3
TEST ET ANALYSE DES RÉSULTATS

3.1 Variation en fonction du paramètre $\theta \in [-1, 1]$

On a créé un programme pour tester la performance de notre classifieur F en fonction de θ en calculant le taux de faux négatifs et faux positifs dans un intervalle $[\theta_{min}, \theta_{max}]$ choisi par l'utilisateur. Les autres paramètres peuvent aussi être modifiés dans le code.

3.2 Extension de l'algorithme

Notre algorithme est prêt à calculer l'Image Intégrale d'images de n'importe quelle taille. Pour l'adapter il faut donc suivre l'approche proposé par Viola et Jones ([4]) de redimensionner les caractéristiques au lieu de redimensionner les images et les faire parcourir toute l'image.

4
RÉFÉRENCES BIBLIOGRAPHIQUES