ECOLE POLYTECHNIQUE

# Détection par "boosting"

Yuxiang LI, Nicolas MORIN

19 mai 2015

## 1 INTRODUCTION

La détection des visages ou plus généralement la reconnaissance des objets est un sujet extrêmement important aujourd'hui. Beaucoup d'articles ont été publiés pour essayer de trouver un détecteur à la fois efficient et efficace. Dans notre projet, nous avons implémenté la méthode proposée par Viola en utilisant 5233 images de tailles $112 \times 92$. Afin de traiter ces informations plus efficacement, nous avons utilisé la bibliothèque MPI pour mener un calcul distribué. Le résultat est assez prometteur sur les visages proches de notre base d'apprentissage, cependant il lui est difficile de reconnaître d'autres visages quand nous utilisons une photo quelconque.

## 2 CRÉATION D'UN CLASSIFIEUR

### 2.1 LES CARACTÉRISTIQUES

Une caractéristique est une zone rectangulaire définie par sa position $(x, y)$, sa taille $(w, h)$ et le calcul que nous faisons dessus ($type$). Nous utilisons un nombre gigantesque de caractéristiques pour décrire une image et celle qui s'approche plus d'un visage que d'un paysage sera classée comme un visage. Voici 4 types de caractéristiques :

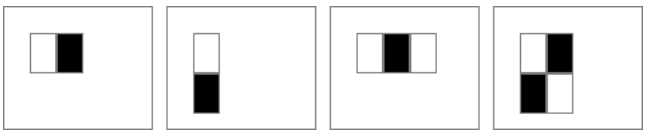

FIGURE 1 – La valeur d'une caractéristique égale la somme en blanc moins la somme en noir

Sur une image de $112 \times 92$, le nombre de caractéristiques est immense. Nous avons utilisé pour cela un incrément de 4 ou 8 (la position et les dimensions sont donc toutes des multiples de 4 ou 8), ce qui a donné 168336 ou 9834 caractéristiques.

## 2.2 Classifieurs faibles

A chaque caractéristique, on associe un classifieur composé de 2 nombre réels $w_{1i}$ et $w_{2i}$. Nous classifions une image avec la valeur $w_{1i}X_i + w_{2i}$ où $X_i$ est la i$^{\text{ème}}$ caractéristique de l'image. Nous avons entraîné les deux valeurs avec la méthode de perception.

Ce processus se parallélise facilement avec 2 méthodes distinctes :

1. Répartir les caractéristiques sur les machines ;
   Chacune travaille avec toutes les images qu'elle possède
   Ramasser tout le tableau des caractéristiques (Gather)
2. Chaque machine entraîne toutes les caractéristiques avec une partie des images ;
   Calculer la somme pour toutes les caractéristiques (Reduce)

Dans notre cas, nous avons utilisé la première méthode en appelant `Gatherv` à la fin pour ramasser tous les résultats. Ce choix est dû au nombre plus important des caractéristiques que des images.

### 2.2.1 Boosting des classifieurs faibles

Chacun de ces classifieurs sont faibles, car leur performance individuelle est très limitée, nous comprenons bien que nous ne pourrons rien dire sur une image si nous n'en voyons qu'une petite partie. Utiliser toutes les caractéristiques est aussi une mission impossible, étant donné leur nombre. Mais si nous voyons les yeux, ce sera plus facile. L'idée de l'algorithme Adaboost est donc de trouver les caractéristiques qui importent le plus et de les assembler en un seul classifieur. Dans notre implémentation, la caractéristique la plus importante est celle qui minimise l'erreur.

La complexité pour trouver une caractéristique est de $O(d * n)$ où $d$ est le nombre de caractéristiques et $n$ le nombre d'images. Une grande partie de cet algorithme n'est pas parallélisable à cause de la fonction exponentielle, mais nous pouvons toujours paralléliser le calcul de l'erreur :

1. Répartir les caractéristiques sur les machines
   Ramasser l'erreur pour avoir le minimum (Reduce)
2. Chaque machine calcule l'erreur sur une partie de la base de données
   Calculer la somme d'erreur (Gather)

Nous avons utilisé la deuxième méthode, parce que la recherche du minimum en gardant l'index du classifieur est plus complexe et que la deuxième parallélisation est aussi efficace que la première si la taille de la base de données est assez grande.

# 3 Organisation du code

## 3.1 `Image` & `imageUtils` & `mpiUtils`

La classe `Image` permet de lire une image et de stocker les pixels dans un tableau. Nous initialisons un objet `Image` avec le calcul de l'image intégrale, puis on appellent `GetFeatureAt` pour calculer la caractéristique demandée.

En-dehors de la classe, dans l'espace de noms `imageUtils`, nous avons implémenté quelques fonctions utiles pour décrire les caractéristiques et initialiser le programme :

- `InitFeatures` : trouver toutes les caractéristiques étant donnée la dimension de l'image
- `FeatureEncode` : remplacer la définition par 5 entiers $(x, y, z, h, type)$ par un hash
- `FeatureDecode` : retrouver les attributs d'une caractéristiques avec un hash
- `InitImages` : lire toutes les images dans chaque répertoire et les insérer dans 3 vectors.

L'espace de noms `mpiUtils` stocke quant à lui les informations communes et nécessaires pour effectuer le calcul distribué comme `Gatherv`. Par exemple :

- `recvCounts` : tableau avec le nombre de caractéristiques à analyser sur chaque machine (utile au cas où ce nombre n'est pas divisible par le nombre de machines)
- `displs` : tableau avec l'index de la première caractéristique à traiter sur chaque machine

## 3.2 `classifier` & `adaboost`

L'espace de noms `classifier` permet d'initialiser toutes les caractéristiques, la méthode `Train` les entraîne et la méthode `Classify` classe une image avec une caractéristique.

Une fois que nous avons tous les classifieurs faibles, nous pouvons faire le boosting avec l'espace de noms `adaboost`. Comme l'espace précédent, la méthode `Iteration` fait le calcul et la méthode `Classify` classe l'image.

Dans tous les cas, nous avons aussi implémenté une méthode `Print` et une méthode `Read` pour éviter de faire le calcul à chaque test.

## 3.3 `Photo`

Cette dernière composante du programme est proche de la classe `Image`. La seule différence est que sa taille est réglable. La méthode `FindFace` parcourt tous les coins de la photo pour trouver les éventuels visages de toute taille. Pour ce faire, la méthode `Resize` élargit ou réduit une zone de photo et renvoie une instance d'`Image` pour passer dans le classifieur.

## 3.4 Un peu de Python

Afin de visualiser efficacement les résultats, nous avons utilisé Python pour faire des statistiques, mettre nos photos en gris, superposer les visages détectés sur la photo, etc..

# 4 RÉSULTATS

Les tests ont été menés sur une machine avec la configuration suivante :
- Processeur : i7-4700M, quad-core
- Mémoire : 8 Gb
- Système : Ubuntu 14.04 LTS
- Commande MPI : `mpirun -np 5 ...`
- Incrément des caractéristiques : 8 (par défaut)

## 4.1 CLASSIFICATION AVEC LES CARACTÉRISTIQUES SÉLECTIONNÉES

### 4.1.1 CLASSIFIEURS SÉLECTIONNÉS

Dans notre expérience, nous avons sélectionné 50 caractéristiques parmi 9834. Les 3 caractéristiques les plus importantes sont illustrées dans la figure ci-dessous. Nous voyons bien que même pour la machine, la zone des yeux est très particulière.

FIGURE 2 – Les 3 premières (5 à droite avec un incrément de 4) caractéristiques sélectionnées

### 4.1.2 TAUX DE FAUX POSITIFS

La vitesse de notre détecteur est assez grande, il arrive à classer 5233 images en seulement 15 ms. En faisant varier $\theta$ de $-1$ à 1, nous avons eu le résultat suivant :

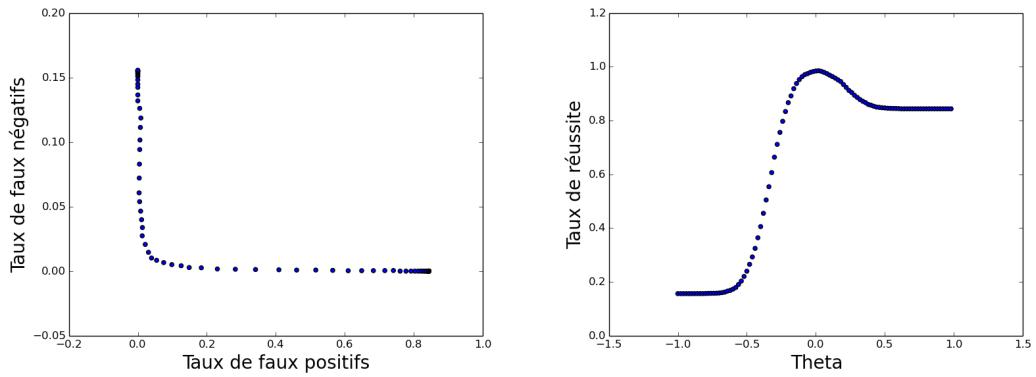


FIGURE 3 – Résultat avec $\theta \in [-1, 1]$

#### 4.1.3 RÉSULTAT SUR UNE PHOTO QUELCONQUE

Nous avons testé notre détecteur sur une image quelconque de dimensions :

— Largeur : 3636
— Hauteur : 2425
— Format : 8-bits Grayscale

Nous avons eu le résultat suivant :

FIGURE 4 – Résultat du détecteur : 18 secondes

Le résultat est assez pauvre, quelques visages sont retrouvés au prix de nombreux faux positifs. De plus, les visages un peu tournés ne sont pas du tout détectés.

## 5 CONCLUSION

La taille de la base de données importe. Comme nous l'avons vu dans la section Résultats, la performance de notre détecteur dépend beaucoup des visages qu'il a étudiés. Dans la pratique, on utilise plusieurs poses d'un même visage pour pouvoir détecter tous les visages même s'ils sont tournés. Mieux encore, nous pourrons utiliser le fait que le taux de faux positifs dépend d'une variable pour créer une série de détecteurs qui à chaque tour élimine les images les moins probables d'être un visage. Cette méthode a été proposée par Viola sous le nom de *cascade*.