Detecção de faces através do algoritmo de Viola-Jones

Túlio Ligneul Santos,
M. Sc. student at PESC – COPPE/UFRJ,
1st of July. 2011.

Abstract

This work was developed as a final project for the course *Introdução ao Processamento de Imagens* offered by the *Programa de Engenharia de Sistemas e Computação*(PESC) - COPPE/UFRJ in 2011/1st to graduate students. In this work, it is developed a face detector using the method proposed by Viola and Jones [1]. Therefore, we present such algorithm and expose the results obtained. Finally, conclusions are presented and future works are suggested.

Resumo

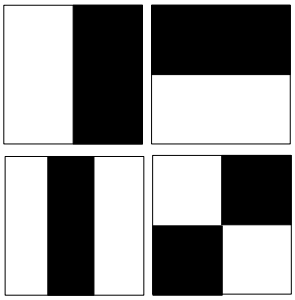
Este trabalho foi desenvolvido como projeto final para a disciplina Introdução ao Processamento de Imagens oferecida pelo Programa de Engenharia de Sistemas e Computação(PESC) - COPPE/UFRJ em 2011/1º aos alunos de pós-graduação. Neste trabalho, desenvolve-se um detector de faces utilizando o método proposto por Viola e Jones [1]. Assim sendo, apresenta-se tal algoritmo e se expõem os resultados obtidos. Por fim, conclusões são expostas e trabalhos futuros sugeridos.

1. Objetivo

Busca-se a implementação de um detector de faces em imagens. Para tanto, utiliza-se o método Viola-Jones [1] para a criação e execução de um detector. Inicialmente treina-se um detector, salvando-o em um arquivo XML para que este, em seguida, possa ser utilizado sempre que se deseje detectar faces em imagens. Adicionalmente, espera-se uma razoável taxa de verdadeiro-positivos nos conjuntos de treino e teste. Para arquivamento e posterior utilização, disponibiliza-se o código fonte do projeto no servidor Git do Laboratório de Computação Gráfica(LCG) – COPPE/UFRJ [4].

2. Fundamentos

As unidades básicas do método Viola-Jones são os denominados *features* retangulares**.** Eles possuem formatos específicos, estando os quatro padrões possíveis ilustrados na Figura 1. Contudo, à principio, podem ter dimensões e posições arbitrárias dentro de uma janela w.



(Figura 1 – As quatro configurações possíveis de um *feature*)

O valor de um *feature* sobre uma imagem pode ser calculado da seguinte forma:

$$f(w) = \sum^{w} p_{preto} - \sum^{w} p_{branco};\ \ onde:$$

$f(w)$: *valor do feature na janela* w;

$\sum^{w} p_{preto}$: *somatório dos pixels na região preta;*

$\sum^{w} p_{branco}$: *somatório dos pixels na região branca.*

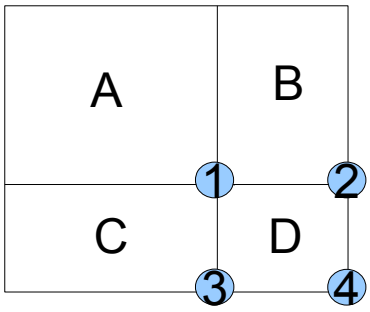
Para acelerar o cálculo do valor de um *feature,* é utilizada a representação da integral da imagem, definida por:

$$ii(x, y) = \sum_{x \leqslant x',\, y \leqslant y'} i(x', y');\ \ onde:$$

i(x' , y') : valor do pixel da imagem na coluna x' e linha y'.

ii(x , y): valor da integral da imagem até suas coluna x e linha y.

Ao certo, através desta representação, é possível calcular o valor do somatório dos valores dos pixels que ocupam determinada área retangular na imagem em O(1), como exibido na Figura 2. Consequentemente, também se obtém o valor de um *feature* com a mesma complexidade.



$$\sum^{D} pixels = ii(4) + ii(1) - (ii(2) + ii(3))$$

(Figura 2 – Avaliação da soma dos valores dos pixels na região D utilizando a integral da imagem.)

O inicio da classificação de um a janela (ou sub-janela) como contenedor de uma face ou não ocorre através dos denominados classificadores fracos. Um classificador fraco pode ser definido pela função:

$$h(w, p, f, \theta) = \begin{cases} 1, & se\ pf(w) < p\theta \\ 0, & caso\ contrário \end{cases};\ \ onde:$$

w: *sub-janela de* 24x24 *pixels*

f: *feature;*

p: *polaridade;*

θ: *theshold.*

Em um nível acima, se encontram os classificadores fortes. Estes, compostos por diversos classificadores fracos, são definidos pela função:

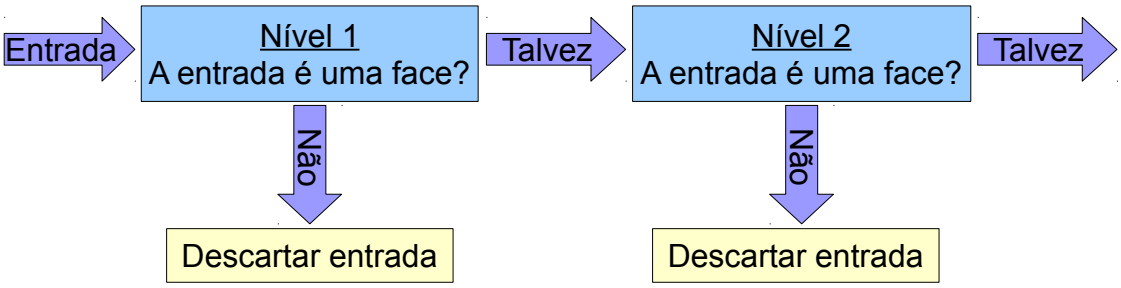
$$C(w)=\begin{cases}1, & se \sum_{t=1}^{T} \alpha_t h_t(w) \leqslant \frac{1}{2}\sum_{t=1}^{T} h_t(w) \\ 0, & caso\ contrário\end{cases}; \quad onde:$$

α_t: *constante calculada durante o treinamento.*

$h_t(w)$*:valor do t-ésimo classificador fraco;*

T*: número de classificadores fracos;*

Por fim, constrói-se uma cascata de classificadores onde, cada camada/nível possui um conjunto distinto de classificadores fortes. Uma representação do funcionamento da cascata de classificadores pode ser vista na Figura 3. Seu uso objetiva a detecção de casos negativos rapidamente, acelerando a execução do algoritmo. De fato, casos negativos não passam por todos os níveis da cascata, sendo rejeitados antes do último nível da cascata. De modo contrário, para uma face ser detectada é preciso que se percorra todas as camadas.



(Figura 3 – Representação do funcionamento da cascata de classificadores.)

3. Treinamento

O treinamento de um classificador forte que contenha T classificadores fracos é realizado pelo algoritmo AdaBoost, detalhado abaixo:

▪ Dadas as imagens exemplo (x_1, y_1), ..., (x_n, y_n) de 24x24 pixels, onde $y_i = 0,1$ para imagens negativas e positivas respectivamente:

▪ Inicializar os pesos $\omega_i = 1/2m,\ 1/2l$ para $y_i = 0,1$ respectivamente, tendo m exemplos negativos e l exemplos positivos.

▪ Para $t = 1,...,T$:

- Normalizar os pesos: $\omega_{t,i}=\frac{\omega_{t,i}}{\sum_{j=1}^{n}\omega_{t,j}}$

- Selecionar o melhor classificador fraco em relação ao erro:

$$\epsilon_t = min_{f,p,\theta}\left(\sum_i^n \omega_i |h(x_i, f, p, \theta) - y_i|\right)$$

- Definir $h_t(x) = h(x, f_t, p_t, \theta_t)$ onde f_t, p_t, e θ_t são os minimizadores de ϵ_t.

- Atualizar os pesos: $\omega_{t+1,i} = \omega_{t,i}\beta_t^{1-e_i}$, usando $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ e tendo e_i = 0 se o i-ésimo exemplo foi classificado corretamente ou 1, caso contrário.

- Calcular: $\alpha_t = \log(\frac{1}{\beta_t})$

Sabendo como ocorre o treinamento dos classificadores fortes, em seguida, é preciso conhecer como é treinada/formada a cascata de classificadores. Para tanto, define-se a notação: F_i para a taxa de falso-positivos até o i-ésimo nível da cascata, D_i para a a taxa de detecção (verdadeiro-positivos) até o i-ésimo nível da cascata e i, para o número de níveis na cascata.

Antes do algoritmo ser iniciado é preciso que o usuário defina F_{target} ,a taxa alvo de falso-positivos; *f*, a taxa de falso-positivos por camada e *d*, a taxa minima de detecção por camada. A seguir, o treinamento é iniciado com os conjuntos P e N dos exemplos positivos e negativos, respectivamente, $F_0 = 1$, $D_0 = 1$ e i = 0, seguindo, então, o algoritmo:

▪ Enquanto $F_i > F_{target}$:

- i+=1; $n_i = 0$; $F_i = F_{i-1}$;

- Enquanto $F_i > f * F_{i-1}$:

  ▫ n_i+=1;

  ▫ Usar N e P para treinar 1 classificador (forte) com n_i *features*;

  ▫ Inserir classificador no final da cascata;

  ▫ Avaliar cascata no conjunto de validação e determinar D_i;

  ▫ Enquanto $D_i < d * D_{i-1}$:

· Reduzir o *threshold* do i-ésimo classificador e recalcular D_i;

▫ Avaliar cascata no conjunto de validação e determinar F_i.

▫ Se $F_i \leq f * F_{i-1}$:

· Remover o i-ésimo classificador da cascata.

- N = Ø;

- Se $F_i > F_{target}$:

▫ Avaliar o detector no conjunto de imagens negativas, colocando todos os falso-positivos em N;

Como visto, para a efetiva execução do treinamento são necessários os conjuntos de imagens negativas e positivas. No caso dos exemplos negativos, eles são simplesmente sub-janelas aleatórias de 24x24 pixels retiradas de 1192 imagens negativas obtidas pela navegação no Google imagens. Já os exemplos positivos foram obtidos no *Face Detection Data Set and Benchmark* (FDDB) [2]. Contudo, ainda foi necessário utilizar o detector de faces do OpenCv para detectar as faces nas imagens, as salvando em imagens separadas e ainda remover manualmente do conjunto de imagens qualquer falso-positivo que possa ter sido detectado pelo OpenCv. Uma amostra do conjunto de imagens positivas utilizado para o treinamento pode ser visto na Figura 4. Por fim, cada imagem, negativa ou positiva, foi convertida para escala de cinza, escalada para 24x24 pixels e normalizada com média nula e variância unitária antes de ser utilizada.

(Figura 4 – Amostra do conjunto de imagens positivas utilizado no treinamento.)

Tendo a imagens já preparadas para o uso, foram utilizados para o treinamento 4916 exemplos positivos e 6000 negativos. Enquanto isso, para a validação utilizou-se 5000 imagens positivas e 10000 negativas, sendo que o conjunto positivo de treino está contido no positivo de validação.

Por fim, antes de iniciar o treinamento definiram-se outros parâmetros: para o treino dos classificadores fortes, estabeleceu-se que, para cada classificador fraco inserido nele durante o treinamento, busque-se o *feature* que apresenta o menor erro dentre 400 *features* gerados aleatóriamente. Além disso, optou-se por construir um detector de 10 camadas com taxa máxima de falso-positivos por camada f = 0.39 e taxa mínima de detecção (verdadeiro-positivos) por camada d = 0.93, de modo que $F_{total} \leq (0.39)^{10} \approx 6.2 * 10^{-5}$ e $D_{total} \geq (0.93)^{10} \approx 0.48$.

4. Detecção

Após ter criado um detector, este poderá ser carregado e utilizado para detectar faces em qualquer imagem, seguindo os passos abaixo. Uma demonstração visual do processo de detecção pode ser visto em *OpenCV Face Detection: Visualized* [3].

- Carregar detector treinado do aquivo XML que o contém.
- Definir o tamanho inicial da sub-janela em que o classificador atua como t = 24x24 pixels.
- Definir o fator de escala como s = 1.
- Enquanto tamanho da sub-janela < tamanho da imagem:
    - Varrer a imagem com a sub-janela, realizando deslocamentos de s pixels e avaliando se a sub-janela contém uma face. Se uma face for detectada deve-se salvar a sub-janela.
    - Escalar a sub-janela de modo que t = t * 1.25 e s = s * 1.25;
- Agrupar sub-janelas em conjuntos disjuntos.
- Descartar conjuntos com menos de 10 sub-janelas.
- Fundir as sub-janelas em cada conjunto restante.
- Marcar detecções na imagem original.

5. Exemplos

(Figura 5 – Exemplos de detecções de faces realizadas pelo detector criado.)

6. Conclusões

Podem ser apontadas algumas dificuldades encontradas durante o projeto, especialmente para o treinamento dos classificadores. Por exemplo, foi necessário estabelecer as taxas de detecção e de falso-positivos e o número de níveis da cascata de classificadores, tal como mencionado na seção 4. Ao certo, isto se mostrou uma barreira a medida que, à principio, não se tem um valor certo para tais valores. Assim sendo, foram escolhidos os valores que permitissem que o treinamento não fosse demasiado longo e nem comprometesse o funcionamento do detector, seja por o fazer ser mais lento ou produzir resultados incorretos.

Como outra dificuldade, pode ser apontada a geração de imagens negativas para o treino, pois à cada nível inserido na cascata de classificadores, torna-se mais difícil encontrar falso-positivos para treinar o nível seguinte. Isso fez com que apenas fossem utilizados poucos exemplos negativos para os últimos níveis da cascata.

Em comparação com o detector implementado por Viola e Jones [1], que possui 6060 *features* distribuídos em 38 camadas/níveis, o detector implementado se mostra, sem dúvida, mais

simples, não sendo tão eficiente, apesar de provar a validade do método. Outra diferença em relação à implementação original ocorre no modo com que os exemplos positivos foram gerados. Enquanto no de Viola-Jones as faces foram selecionadas à mão e seguindo um procedimento definido, neste trabalho elas foram escolhidas usando um detector já existente. Isto, de fato, pode ter prejudicado o desempenho do classificador criado. No estágio em que as sub-janelas são agrupadas também pode se perceber uma modificação. Alterou-se de 5 para 10 o número de sub-janelas em um grupo necessárias para caracterizar uma detecção. Isso foi feito como modo de compensar a alta taxa de falso-positivos obtida.

7. Trabalho Futuro

Propõe-se selecionar novas imagens positivas utilizando o procedimento sugerido por Viola e Jones [1], além de usar validação cruzada no treinamento, usando conjuntos de treino e teste independentes. Ademais, deveria ser treinado um novo detector, podendo esse processo ser dado de modo mais cuidadoso, procurando sempre detecções eficientes e maior velocidade. Por exemplo, as taxas máximas e mínimas de detecção e falso-positivos podem ser variadas de camada em camada da cascata de classificadores. Além disso, poder-se-ia avaliar diferentes classificadores para a mesma camada, como comparar o desempenho de classificadores com diferentes quantidades de *features*.

Por fim, uma melhoria na velocidade de execução do treinamento poderia ser adquirida através da paralelização do processo, podendo inclusive ter o processo realizado por placas gráficas (GPU's) . Analogamente, o mesmo raciocínio também poderia ser utilizado durante a detecção de faces.

8. Referências

[1] Viola, P., e Jones, M. - "*Robust Real-Time Face Detection"*, *International Journal of Computer Vision* 57(2), 137-154, 2004.

[2] University of Massachusetts - "*Face Detection Data Set and Benchmark* (FDDB)", http://vis-www.cs.umass.edu/fddb/, último acesso em 27/06/2011.

[3] Harvey, A. - "OpenCV Face Detection: Visualized", http://vimeo.com/12774628, último acesso em 28/06/2011.

[4] Santos, T. L. - "Repositório do projeto em servidor Git" - git@git.lcg.ufrj.br:tulio/imageprocessing.git, último acesso em 28/06/2011.