UNIVERSITÀ DEGLI STUDI DI TORINO – DIPARTIMENTO DI INFORMATICA

Relazione del progetto di Elaborazione di Immagini e Visione Artificiale

Implementezione dell'algoritmo di training tratto dall'articolo "Robust Real-Time Face Detection" di Paul Viola e Michael J. Jones (2004)

Ruben Caliandro

Anno accademico 2013/2014

Il progetto consiste nell'implementazione di un algoritmo di training che costruisce una cascata di classificatori in grado di rilevare e distinguere la presenza dei volti all'interno delle immagini. Il training è stato eseguito su un dataset opensource di 6977 campioni. La cascata di classificatori è stata poi testata su un diverso dataset di immagini e sono stati raccolti e commentati i risultati.

Introduzione

Questo progetto consiste nell'implementazione dell'algoritmo di training presentato da Viola e Jones nel 2001 e pubblicato sull'International Journal of Computer Vision nel 2004.

L'articolo citato mostra un framework introdotto da Viola e Jones per il riconoscimento in real-time dei volti nelle immagini. Si parla di real-time perché già nel 2001 era possibile con questo metodo rilevare la presenza dei volti sulle immagini ad un frame rate di 15 fps (su un processore Intel Pentium III da 700 MHz, utilizzando immagini da 384x288 pixel).

Il punto chiave che permette un'elaborazione così rapida non consiste nell'algoritmo di rilevamento in sé, che è abbastanza semplice, sta invece in una fase di training piuttosto elaborata che mette insieme gli elementi per costruire un detector in grado di soddisfare i requisiti di velocità e detection rate prefissati.

In questa relazione illustro i principi che stanno alla base del framework di Viola e Jones, spiego l'implementazione dell'algoritmo di training che ho sviluppato per il corso di Elaborazione delle Immagini e Visione Artificiale e commento i risultati degli esperimenti che ho fatto per testare la correttezza dell'algoritmo.

Training

Scopo

L'algoritmo di training serve per scegliere un set di classificatori che dovrebbero poi essere in grado di distinguere quali sono le sottofinestre che contengono un volto all'interno di un'immagine.

Il principio che sta alla base è quello di far 'imparare' alla macchina come sono fatti i volti. Per fare questo, si utilizza un grosso insieme di immagini (il training-set) già classificate come positive o negative. Ogni immagine positiva contiene un volto visualizzato frontalmente e ritagliato all'altezza della fronte e del mento, mentre ogni immagine negativa contiene un qualunque particolare che non sia un volto. Tutte le immagini sono in scala di grigio e hanno tutte la stessa dimensione, tipicamente 24x24 o 19x19 pixel.

Quanto più è grande il training-set e quanto meno le immagini campione sono simili tra loro, tanto migliore sarà il risultato, poiché il computer impara a riconoscere volti che sono simili ai campioni positivi e impara a rifiutare immagini che sono simili ai campioni negativi.

Per il progetto è stato utilizzato un training-set di 6977 immagini campione da 19x19 pixel, di cui 2429 positive e 4548 negative. Tutte le immagini sono state prelevate dal database open-source CBCL Face Database #1 del MIT Center For Biological and Computation Learning.

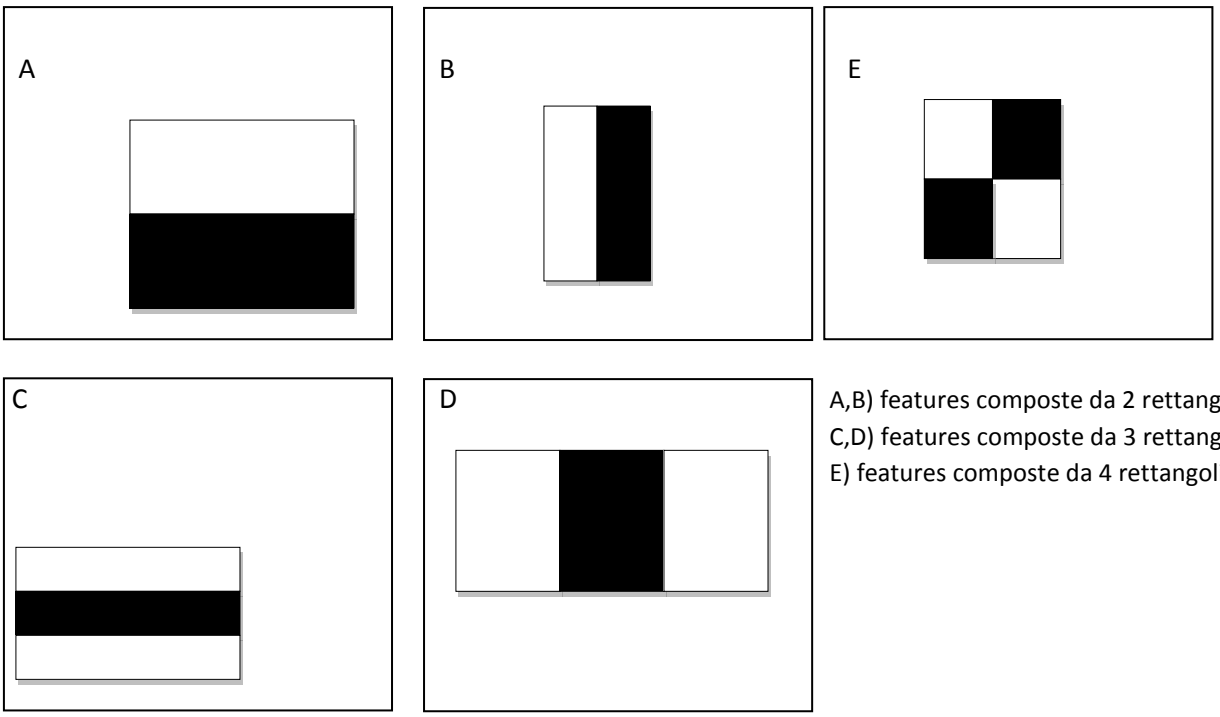
L'apprendimento avviene selezionando delle caratteristiche (features) che vengono riconosciute come comuni a tutti i volti e che verranno poi utilizzate per classificare le immagini da sottoporre a detection.

Haar-like features

La procedura introdotta da Viola e Jones classifica le immagini a partire dal valore di semplici features (dette *Haar-like features*). Queste features sono dei valori numerici calcolati a partire dai livelli di intensità di alcuni gruppi di pixel selezionati strategicamente. Il vantaggio di utilizzare le features invece dei singoli pixel sta nel fatto che la feature codifica in un unico valore un insieme di pixel mettendo in evidenza un certo tipo di correlazione che c'è tra di essi. Questo tipo di codifica semplifica notevolmente il costo computazionale.

Una feature è calcolata sommando e/o sottraendo tra di loro somme di pixel prese da regioni rettangolari.

Le features utilizzate per il progetto sono di cinque tipi, di cui vengono mostrati degli esempi in figura.



A,B) features composte da 2 rettangoli
C,D) features composte da 3 rettangoli
E) features composte da 4 rettangoli

Per calcolare il valore di una feature bisogna sottrarre la somma delle intensità dei pixel contenuti nei rettangoli scuri dalla somma delle intensità dei pixel contenuti nei rettangoli chiari.

Per il progetto sono state prese in considerazione solo le feature composte esclusivamente da rettangoli congruenti tra di loro, adiacenti e con area non nulla. In questo modo sono state individuate 63960 diverse features per una finestra di 19x19 pixel

Per far funzionare l'algoritmo è necessario che per ogni immagine vengano calcolate tutte le features. Data la grossa quantità di features e di immagini campione, calcolare ogni volta la somma dei pixel di ogni rettangolo è a dir poco proibitivo in termini di costo computazionale. Per questo motivo, invece dell'immagine originale a livelli di grigio, viene utilizzata una diversa rappresentazione che si ottiene facilmente dalla prima: l'immagine integrale.

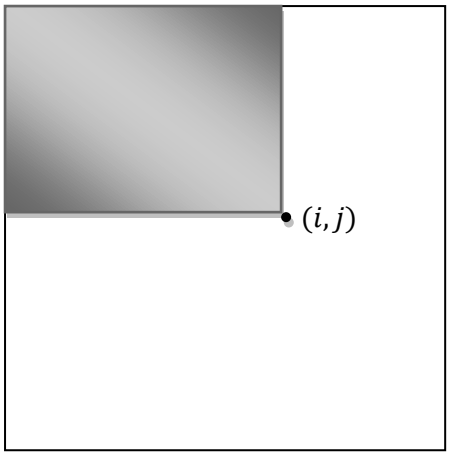
L'immagine integrale

L'immagine integrale è una rappresentazione in cui ogni pixel ha come valore la somma di tutti i pixel dell'immagine originale che si trovano in alto a sinistra rispetto alla sua posizione.

$$ii(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

Dove $f(x,y)$ è l'immgaine originale, e $ii(x,y)$ è l'immagine integrale. Per esempio:

$$f(x,y) = \begin{matrix} 3 & 2 & 5 \\ 1 & 6 & 2 \\ 1 & 2 & 1 \end{matrix} \rightarrow ii(x,y) = \begin{matrix} 3 & 5 & 10 \\ 4 & 12 & 19 \\ 5 & 15 & 23 \end{matrix}$$



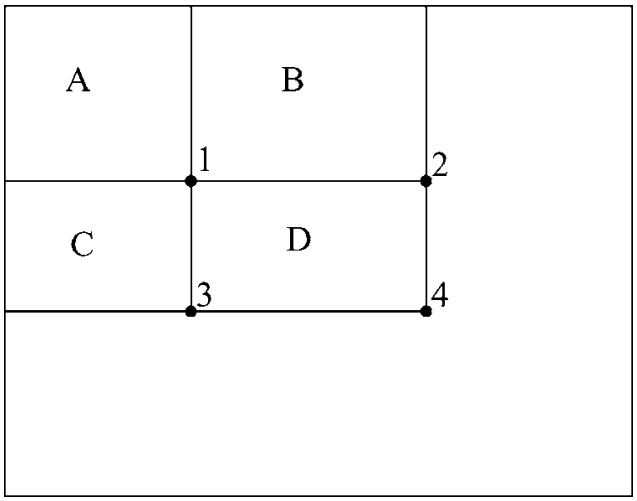
Il pixel in posizione (i,j) dell'immagine integrale ha come valore la somma di tutti i pixel dell'immagine originale che si trovano in alto a sinistra rispetto allo stesso.

Esiste un modo semplice per calcolare l'immagine integrale con complessità lineare. Basta scorrere l'immagine originale riga per riga e sommare i valori letti ai pixel già calcolati fino al passo precedente.

$$ii(i,j) = ii(i-1,j) + ii(i,j-1) - ii(i-1,j-1) + f(k,l)$$

Per semplificare alcune parti del codice si è deciso di creare delle immagini integrali 0-padded in alto a sinistra, quindi di dimensione 20x20.

Il vantaggio di utilizzare l'immagine integrale è che la somma di tutti i pixel contenuti in un rettangolo si può ottenere sommando quattro singoli pixel dell'immagine integrale.



Il valore dell'immagine integrale in posizione 1 è la somma dei pixel del rettangolo A.

Il valore dell'immagine integrale in posizione 2 è la somma dei pixel di A+B

Il valore dell'immagine integrale in posizione 3 è la somma dei pixel di A+C

Il valore dell'immagine integrale in posizione 4 è la somma dei pixel di A+B+C+D

La somma intena a D si può calcolare come
$D = 4 - 3 - 2 + 1$

(I numeri indicano il valore dell'immagine integrale nelle posizioni corrispondenti in figura)

Infatti $4 - 3 - 2 + 1 = (A + B + C + D) - (A + C) - (A + B) + A = D$

Questo permette di calcolare la somma dei pixel contenuti in un rettangolo utilizzando solo 4 accessi ad array e 3 somme.

Come accennato in precedenza, ogni immagine di 19x19 pixel contiene 63960 features, un numero parecchio maggiore del numero di pixel. Anche se ogni feature può essere calcolata in modo molto efficiente, calcolare l'intero set ogni volta che bisogna classificare un'immagine è troppo dispendioso.

In verità serve solo un sottoinsieme molto ridotto di queste features per formare un classificatore efficiente. La sfida principale è quella di trovare queste features.

Selezione delle features con AdaBoost

La selezione delle features avviene con una variante dell'algoritmo di apprendimento AdaBoost. Lo scopo è quello di ridurre notevolmente il numero di features necessarie per classificare le immagini, mantenendo quelle più informative e rimuovendo quelle ridondanti.

Nella sua forma originale AdaBoost viene utilizzato per accelerare un algoritmo di apprendimento semplice. Questo viene fatto combinando una collezione di classificatori deboli per formare un classificatore più robusto.

Nel nostro caso un classificatore debole è definito come

$$h(x, f, p, \theta) = \begin{cases} 1 & se\ \ pf(x) > p\theta \\ 0 & altrimenti \end{cases}$$

Dove:

- x è un'immagine 19x19 pixel del training set.
- f è una Haar-like feature
- p è una polarità, che può assumere valori 1 o -1 e serve a determinare la direzione della disuguaglianza
- θ è un valore soglia (threshold)

Come si può dedurre dalla definizione, un classificatore debole distingue un'immagine come positiva se il valore della feature f calcolata sull'immagine x è maggiore di un certo valore soglia θ (o minore, se $p = -1$). Selezionare il miglior classificatore significa trovare f, p, θ che distinguono con il minore numero di errori tutte le immagini del training set.

L'algoritmo di boosting trova ad ogni iterazione il miglior classificatore debole. Ogni volta che viene scelto un classificatore, vengono aggiornati dei pesi applicati alle immagini, in modo che le immagini che sono state classificate erroneamente vengano pesate maggiormente. In questo modo, l'algoritmo all'iterazione successiva si concentrerà di più su quelle immagini che sono risultate più problematiche.

Quando si ottiene un sufficiente numero di classificatori deboli, si costruisce un classificatore forte come combinazione lineare dei classificatori deboli scelti.

Algoritmo:

1) Data una collezione di immagini campione $(x_1, y_1), \dots, (x_n, y_n)$ dove $y_i = 0,1$ rispettivamente per le immagini negative e positive.
2) Inizializzo i pesi $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ rispettivamente per $y_i = 0,1$. Dove m è il numero di campioni negativi e l il numero di positivi
3) Finchè non ho scartato una certa percentuale (scelta dall'utente) degli esempi negativi che sono rimasti
   a) Incremento il contatore t
   b) Normalizzo i pesi $w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
   c) Seleziono il miglior classificatore debole rispetto all'errore pesato (vedi sezione successiva)
   $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|$$
   d) Definisco $h_t(x) = h(x, f_t, p_t, \theta_t)$, dove f_t, p_t, θ_t sono i minimizzatori di ϵ_t
   e) Aggiorno i pesi
   $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

dove $e_i = 0$ se il campione x_i è stato classificato correttamente, $e_i = 1$ altrimenti.
dove $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

4) Il classificatore finale è definito come

$$C(x) = \begin{cases} 1 \ \ se \ \ \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 \quad altrimenti \end{cases}$$

dove $\alpha = log\left(\frac{1}{b_t}\right)$

La scelta del miglior classificatore debole

Per quanto riguarda il punto 3c) dell'algoritmo illustrato sopra, Viola e Jones danno alcune indicazioni per un'implementazione efficiente.

Nonostante AdaBoost sia un algoritmo efficiente, l'insieme di classificatori è comunque estremamente grande, dato che esiste un singolo classificatore per ogni coppia feature/threshold.

Il numero di feature è noto a priori (lo chiamiamo K), mentre per il numero di threshold bisogna fare un ragionamento in più:

Data una feature f_j, e dati tutti i campioni $x_i \ \ (i = 1..N)$ ordinati per il valore di $f_j(x_i)$:

per ogni coppia di campioni adiacenti x_i, x_{i+1}:

per ogni coppia di threshold θ_a, θ_b per cui $f_j(x_i) \leq \theta_k < f_j(x_{i+1}), k \in \{a; b\}$:

θ_a è equivalente a θ_b, cioè sia che io utilizzi il primo o il secondo threshold, entrambi dividono la lista ordinata di campioni nello stesso modo

Quindi, è sufficiente considerare un singolo threshold per ogni coppia di campioni x_i, x_{i+1} per individuare tutti i classificatori deboli possibili. Quindi il numero di threshold per ogni feature è N, cioè il numero di immagini campione.

Questo per dimostrare che in totale esistono KN classificatori deboli, dove K è il numero delle feature e N è il numero di immagini campione.

La complessità del punto 3c) dell'algoritmo è $O(KN^2)$ perché oltre a scorrere tutti i possibili classificatori, bisogna anche calcolare l'errore minimo per ogni classificatore, che significa controllare tutte le immagini campione per capire quante di esse sono state classificate correttamente.

La complessità dell'intero algoritmo di training, invece, dipende anche dal numero M di classificatori scelti, quindi risulta essere $O(MKN^2)$. Viola e Jones spiegano che è possibile ridurre la complessità dell'algoritmo a $O(MKN)$, e riescono a farlo sfruttando il fatto che in ogni ciclo l'intera dipendenza sulle features selezionate precedentemente può essere codificata in modo efficiente e compatto usando i pesi delle immagini campione. Questi pesi possono essere utilizzati per valutare un dato classificatore debole in un tempo costante.

Nell'implementazione del progetto, $M = 160, \ N = 6977$ *e* $K = 63960$ *quindi* $MKN^2 \approx 4{,}98 \cdot 10^{14}$
Utilizzando il metodo descritto da Viola e Jones, $MKN \approx 7{,}13 \cdot 10^{10}$

L'algoritmo di selezione del miglior classificatore debole procede come segue.

1) Per ogni feature, le immagini campione sono ordinate in base al valore di quella feature

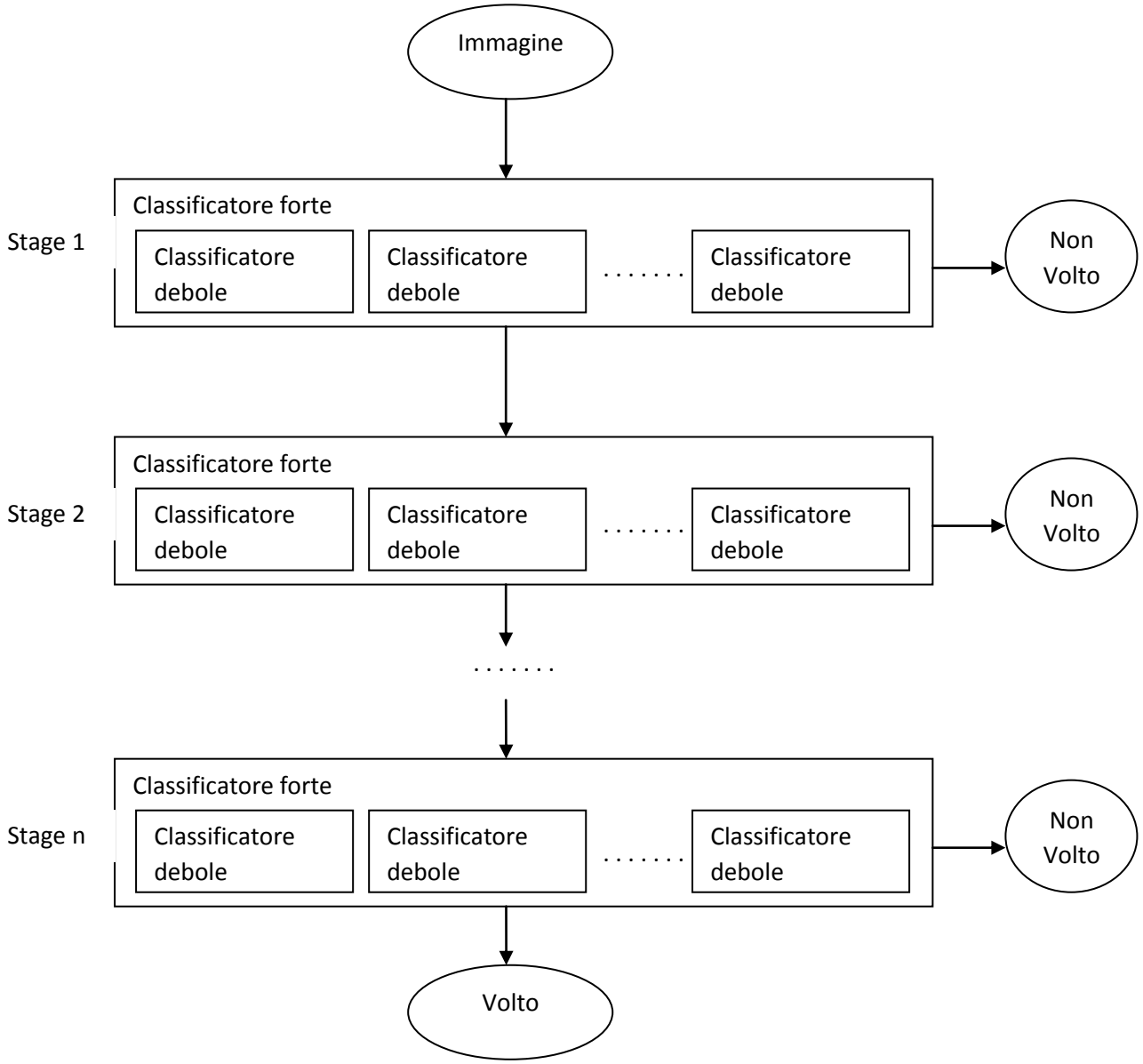
2) Per ogni elemento della lista ordinata, vengono mantenute e valutate quattro somme:
   T^+ - La somma dei pesi di tutte le immagini positive
   T^- - La somma dei pesi di tutte le immagini negative
   S^+ - La somma dei pesi di tutte le immagini positive che si trovano prima del campione corrente
   S^- - La somma dei pesi di tutte le immagini negative che si trovano prima del campione corrente
3) L'errore di un threshold che si trova nell'intervallo tra il campione precedente e quello corrente si può calcolare come:

$$e = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+))$$

La cascata di classificatori

L'algoritmo che è stato mostrato fino ad adesso crea un classificatore forte scegliendo alcuni classificatori deboli.

Viola e Jones dimostrano che si può accelerare il processo di detection creando invece un certo numero di classificatori forti che contengono meno classificatori deboli.
Questi classificatori forti vengono poi utilizzati in una "cascata" decisionale, come quella mostrata in figura.



Data una cascata di classificatori, il false positive rate è:

$$F = \prod_{i=1}^{K} f_i$$

Dove K è il numero di classificatori forti ed f_i è il false positive rate dell'i-esimo classificatore.

Il detection rate è:

$$D = \prod_{i=1}^{K} d_i$$

Dove d_i è il detection rate dell'i-esimo classificatore.

Quindi per avere un detection rate di 0.9 si può costruire una cascata di classificatori con 10 livelli, se ogni livello ha un detection rate di 0.99 (dato che $0.9 \approx 0.99^{10}$). Nonostante il detection rate richiesto sia alto, il vantaggio si ottiene dal fatto che per avere un false positive rate molto basso, il false positive rate di ogni classificatore può essere anche del 30% ($0.30^{10} \approx 6 \times 10^{-6}$)

Detection

Scansione

Il detector finale viene fatto scorrere sull'immagine da scansionare a diversi livelli di scaling. Invece di scalare l'immgine viene scalato il detector, perché grazie all'immagine integrale è possibile calcolare le features ad ogni livello di scaling con lo stesso costo. I risultati migliori si ottengono utilizzando livelli di scaling che si distanziano per un fattore di 1,25.

Il detector viene fatto scorrere traslando la finestra di un numero di pixel Δ. Questo processo di shifting è influenzato dal livello di scaling corrente. Se il livello di scaling corrente è s, la finestra viene traslata di $[s\Delta]$, dove [ ] è l'operatore di arrotondamento.

Per ogni posizione e livello di scaling della finestra, viene individuata una sottoimmagine. Questa sottoimmagine viene sottoposta a classificazione da parte della cascata di classificatori, che può scartarla o segnalarla come volto.

Dato che la presenza di un volto in una sottoimmagine è un evento raro, il detector organizzato a cascata accelera notevolmente il processo, poiché la gran parte delle finestre vengono scartate a partire dai primissimi stage. Solo i volti raggiungono l'ultimo stage della cascata.

Risultati del progetto

Il detector finale

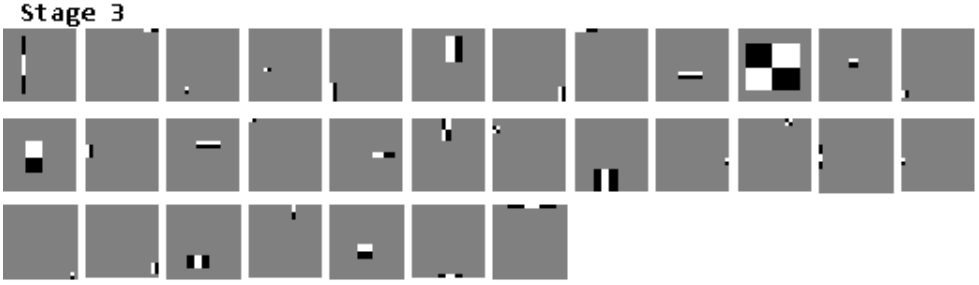
Il training è stato eseguito su un training-set di 6977 immagini campione da 19x19 pixel, di cui 2429 positive e 4548 negative, prelevate dal *CBCL Face Database #1*. Per facilitare il training tutte le immagini sono state normalizzate a media 0 e varianza 1 (come consigliato da Viola e Jones). Questo ovviamente richiede che anche in fase di detection le immagini vengano normalizzate.

La durata del training è stata di circa 40 ore, al termine delle quali è risultata una cascata di 10 stage (classificatori forti), per un totale di 160 classificatori deboli.

Stage 1

Stage 2

Stage 3



Stage 4

Stage 5

Stage 6

Stage 7

Stage 8

Stage 9

Stage 10

Notare che il numero di stage e classificatori deboli è molto ridotto rispetto a quello riportato da Viola e Jones: il loro detector era costituito da 38 stage per un totale di 6060 classificatori deboli. Ulteriori spiegazioni in merito a questi risultati verranno date nella prossima sezione.

Ogni classificatore forte è stato allenato in modo da riconoscere il 100% delle immagini positive del training set, e in modo da potare **almeno** il 50% delle immagini negative ancora non scartate dagli stage precedenti.

Il numero di stage non è stato deciso a priori, ma è risultato dal fatto che dopo il decimo stage tutte le immagini erano già state classificate correttamente.

Questo è stato un problema, perché in base alle formule introdotte precedentemente, il false positive rate teorico, calcolato come $F = \prod_{i=1}^{K} f_i$, risulta essere $F = 0.5^{10} \approx 9.77 \times 10^{-4}$, che è quasi 1000 volte più alto di quello consigliato da Viola e Jones (10^{-6}).

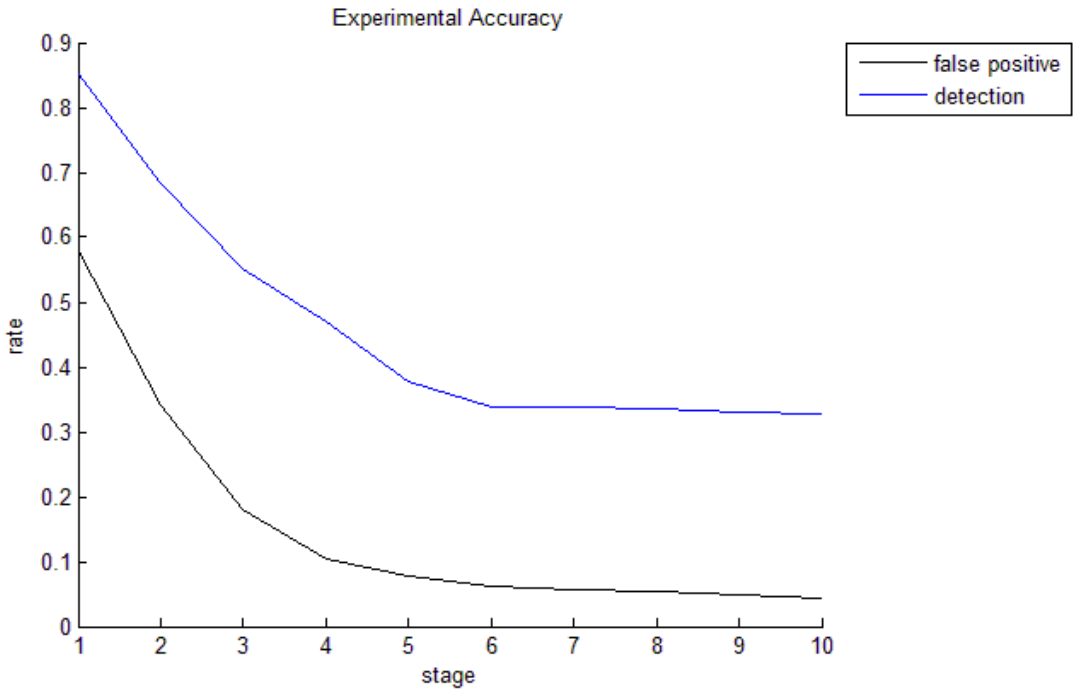
Per quanto riguarda il detection rate, quello teorico è perfetto, perché i classificatori sono stati allenati per riconoscere il 100% delle immagini positive del training set. Tuttavia, come è facile intuire, la bontà del detection rate sperimentale dipende fortemente dalla quantità e dalla varietà delle immagini del training set.

Nel prossimo paragrafo discuterò i risultati sperimentali, mettendoli a confronto con quelli teorici.

Testing e risultati sperimentali

Il detector è stato testato su un **data-set di test** di 24045 immagini, di cui 472 positive e 23573 negative, prelevato anch'esso dal *MIT Center For Biological and Computation Learning*.

Sono stati raccolti i dati di accuratezza del detector in base ai falsi positivi e ai falsi negativi riscontrati via via che i campioni venivano processati dagli stage della cascata.



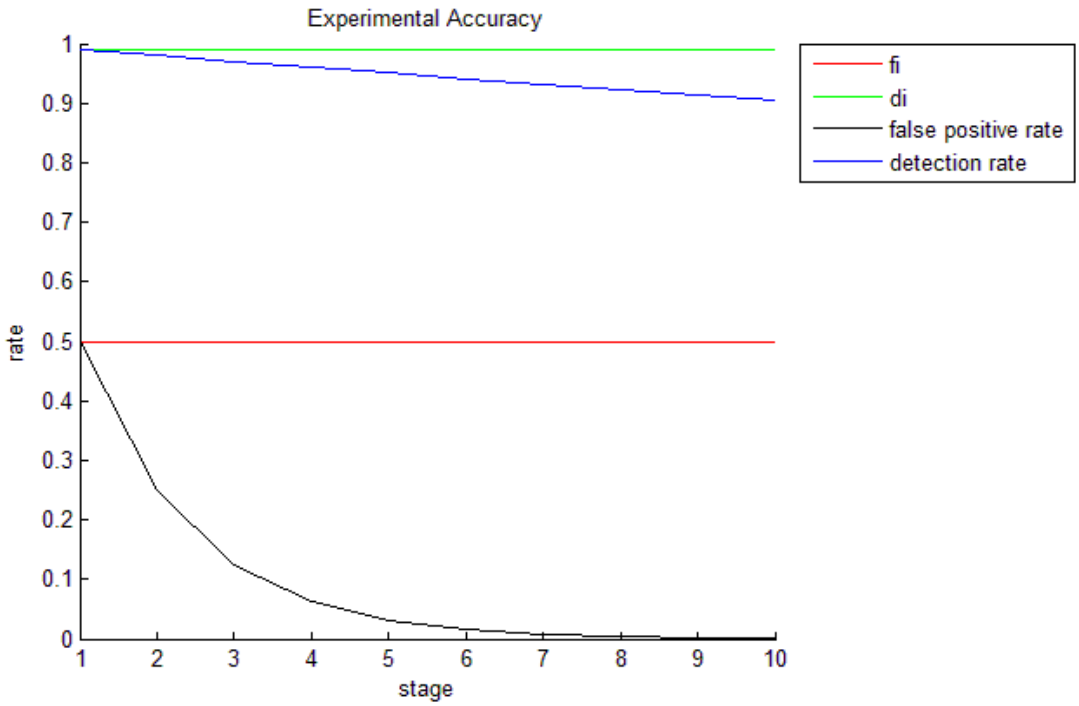
Come si può vedere dal grafico, i risultati ottenuti sono molto lontani da quello che ci si aspettava. Per questo motivo, ho deciso di analizzare più a fondo i dati ottenuti.

L'analisi è stata fatta mettendo a confronto lo stesso insieme di dati preso da tre diverse fonti:

1. I **dati teorici**, calcolati a mano dalle formule implementate nell'algoritmo, assumendo che:
    a. $d_i = 0.99$ per ogni i, dove d_i è il detection rate dello stage i-esimo
    b. $f_i = 0.5$ per ogni i, dove f_i è il false positive rate dello stage i-esimo
    c. Solo 10 stage nella cascata di classificatori (ne servirebbero di più per avere buoni risultati, ma ho preso solo quelli utili al confronto dei dati)
2. I **dati sperimentali**, ottenuti dal test eseguito sul **data-set di test**
3. I **dati sperimentali**, ottenuti da un test eseguito sul **training-set**, aspettandosi che:
    a. $d_i = 1$ per ogni i, perché ogni stage è stato allenato a riconoscere **tutte** le immagini del training-set.
    b. $f_i \leq 0.5$ per ogni i, perché l'algoritmo è fatto in modo che ogni stage poti almeno il 50% dei campioni negativi rimasti.

Fonte 1 – dati teorici

	f_i	d_i	False positive rate	Detection rate
Stage 1	0.5	0.99	0.5	0.9900
Stage 2	0.5	0.99	0.25	0.9801
Stage 3	0.5	0.99	0.125	0.9703
Stage 4	0.5	0.99	6.25×10^{2}	0.9606
Stage 5	0.5	0.99	3.13×10^{-2}	0.9510
Stage 6	0.5	0.99	1.56×10^{-2}	0.9415
Stage 7	0.5	0.99	7.81×10^{-3}	0.9321
Stage 8	0.5	0.99	3.91×10^{-3}	0.9227
Stage 9	0.5	0.99	1.95×10^{-3}	0.9135
Stage 10	0.5	0.99	9.77×10^{-4}	0.9044



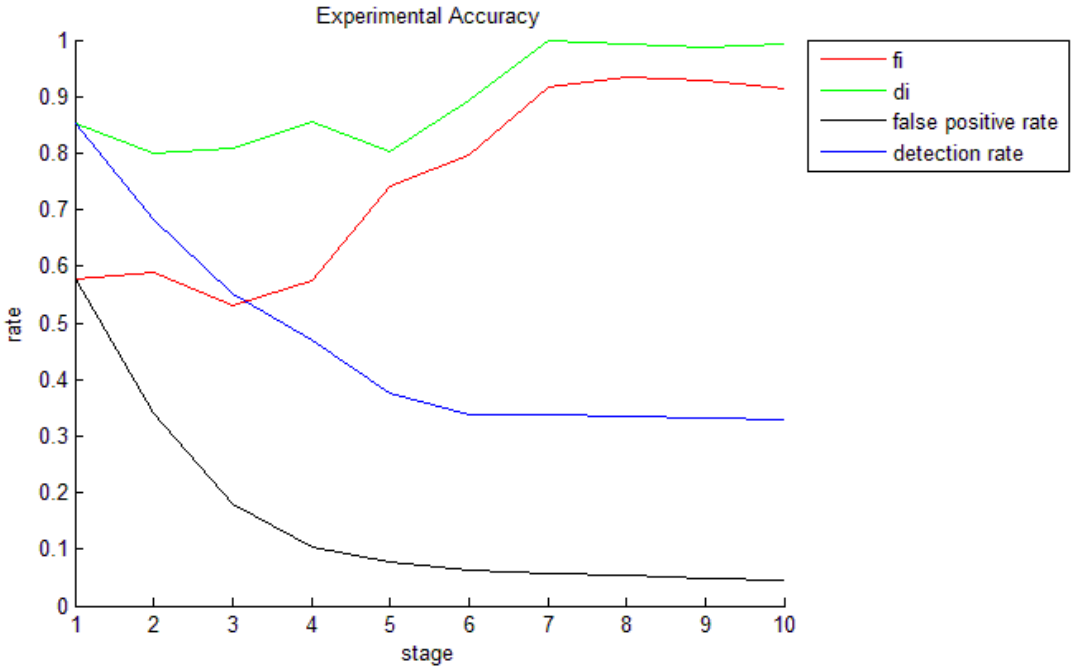
Questi dati teorici servono per dare un'idea di cosa ci dovremmo aspettare da un detector allenato correttamente.

Notare quanto è importante che ogni stage abbia un detection rate molto alto per garantire un detection rate globale accettabile. Per quanto riguarda il false positive rate, invece, quello che conta di più è avere un

numero sufficiente di stage per far sì che la curva abbia modo di avvicinarsi a 0 a causa della sua natura esponenziale.

Fonte 2 – test sul data-set di test

	f_i	d_i	False positive rate	Detection rate
Stage 1	0.5780	0.8517	0.5780	0.8517
Stage 2	0.5888	0.8010	0.3403	0.6822
Stage 3	0.5315	0.8075	0.1809	0.5508
Stage 4	0.5760	0.8538	0.1042	0.4703
Stage 5	0.7402	0.8018	0.0771	0.3771
Stage 6	0.7981	0.8933	0.0616	0.3369
Stage 7	0.9159	1.0000	0.0564	0.3369
Stage 8	0.9338	0.9937	0.0526	0.3347
Stage 9	0.9283	0.9873	0.0489	0.3305
Stage 10	0.9132	0.9936	0.0446	0.3284



La prima cosa che si evince dal grafico, è che ad abbattere drasticamente il detection rate sono i contributi d_i dei primi 6 stage. Il motivo per cui il detection rate dei primi stage è così basso non si può sapere con certezza assoluta senza andare ad analizzare direttamente i campioni, ma di certo dipende fortemente da come sono stati allenati i classificatori.

Il fatto stesso che il detector sia formato da 160 classificatori deboli (contro i 6060 di Viola e Jones) fa presupporre che gli stage della cascata effettuino una classificazione abbastanza "grossolana" delle immagini.

E' molto probabile che il training-set non contenesse abbastanza immagini e/o che le immagini contenute non fossero sufficientemente varie. In pratica quello che è certo è che il training-set non ha messo abbastanza "in difficoltà" l'algoritmo di training, che ha scelto pochi classificatori, alcuni anche simili tra di loro.

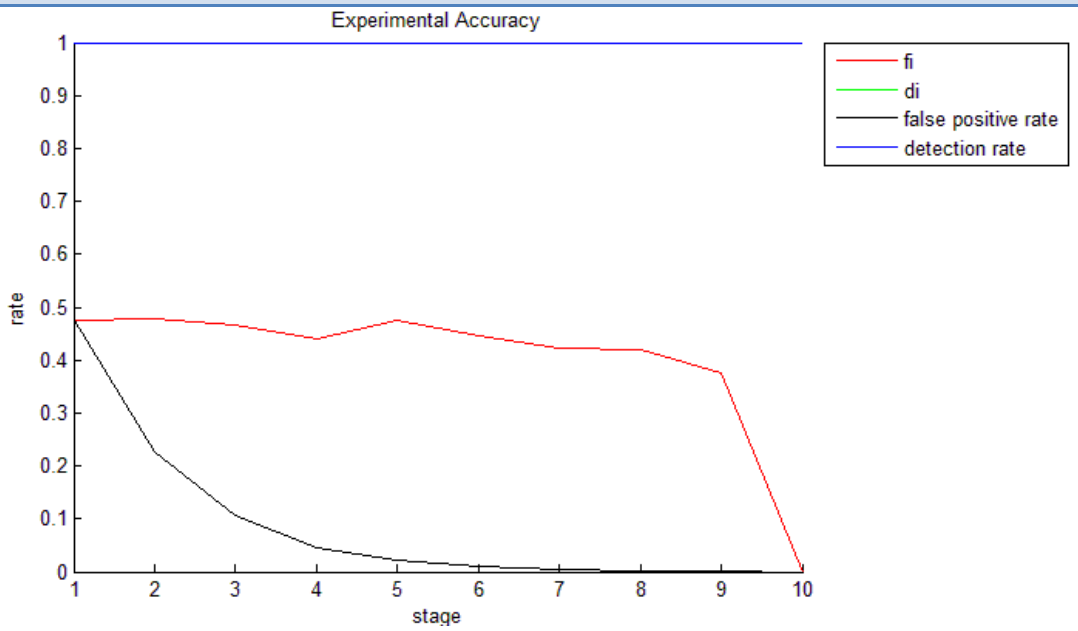
Stessa cosa si può dire per quanto riguarda i valori f_i, che partono bene con i primi stage (di poco superiori a 0.5) e finiscono per assumere valori molto alti verso il fondo della cascata. Purtroppo, anche se la curva

nera non sembra essere troppo lontana dai dati teorici (a vista), è proprio quella che comporta più problemi, perché un false positive rate di 0.0446 comporta un numero di falsi positivi molto maggiore del numero di falsi negativi dovuti al detection rate molto basso. Questo perché la presenza di un volto in un'immagine è un evento raro, che quindi causa pochi errori, mentre la NON presenza di un volto in un'immagine è un evento molto comune, quindi causa tantissimi errori se non viene riconosciuto adeguatamente. Di questo aspetto si avrà un riscontro visivo più avanti.

Per concludere le osservazioni su questo test, si può dire che il fallimento sia dovuto ad un'eccessiva selettività del detector negli stadi iniziali (scarta troppe immagini, anche positive) ed una scarsa selettività negli stadi finali (non scarta quasi niente, neanche le negative), causate probabilmente da un training-set non adeguatamente selezionato.

Fonte 3 – test sul training-set

	f_i	d_i	False positive rate	Detection rate
Stage 1	0.4749	1	0.4749	1
Stage 2	0.4782	1	0.2271	1
Stage 3	0.4666	1	0.1060	1
Stage 4	0.4398	1	0.0466	1
Stage 5	0.4764	1	0.0222	1
Stage 6	0.4455	1	0.0099	1
Stage 7	0.4222	1	0.0042	1
Stage 8	0.4211	1	0.0018	1
Stage 9	0.3750	1	0.0007	1
Stage 10	0	1	0	1



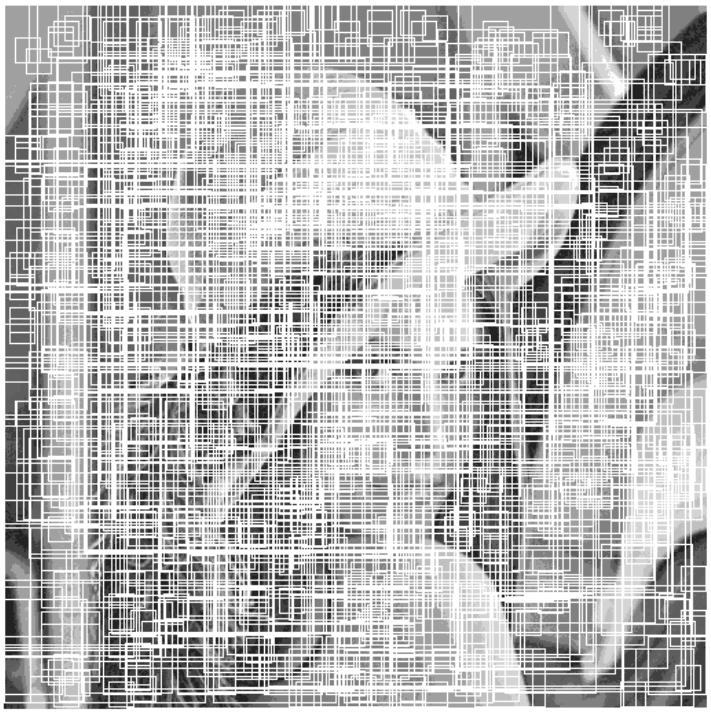
Uno dei motivi principali per cui ho deciso di testare il detector sul training-set è stato per accertarmi del fatto che funzionasse a dovere. La tabella e il grafico confermano l'effettiva correttezza del training, e confermano anche il fatto che il detector sia ottimo per il training-set, poiché termina con un false positive rate di 0 e un detection rate di 1.

Una cosa interessante da osservare è il fatto che i valori f_i sono sempre < 0.5, questo perché l'algoritmo imponeva che ogni stage potasse almeno la metà dei campioni negativi rimasti. Il fatto che nel grafico precedente non ci sia neanche un valore di $f_i < 0.5$ mette in risalto la debolezza del detector.

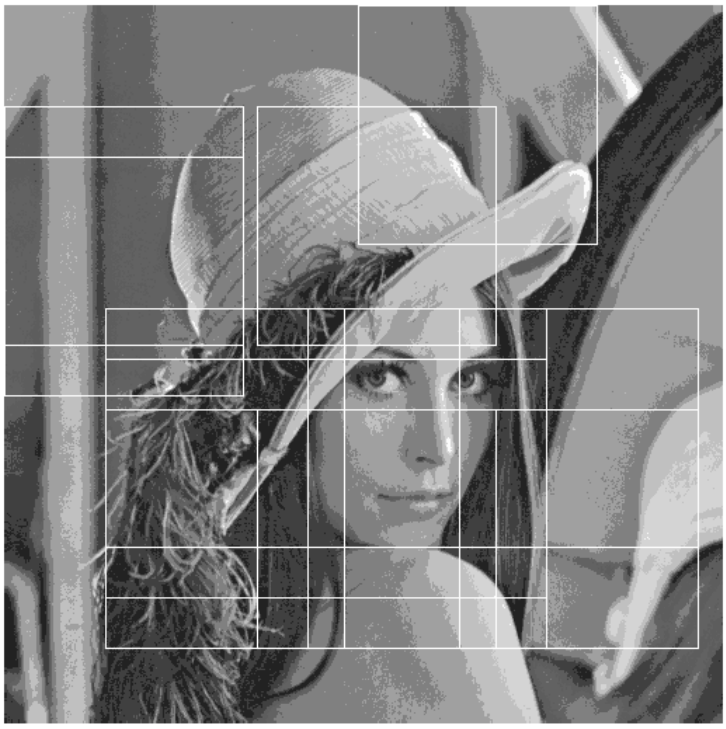
Il detection rate è sempre 1 perché l'algoritmo di training parte considerando tutte le immagini come positive, ed è progettato per scegliere dei classificatori che non le scartino mai.

Riscontro su un'immagine di prova

Testando il detector su un'immagine con un volto il risultato è stato abbastanza confusionario:



Nonostante ciò che si vede possa sembrare del tutto casuale, se si isolano le detections per un singolo valore di scaling, si comincia a capire qualcosa di più. Per esempio con scaling=9:



Dall'immagine si capisce che tutto sommato la gran parte delle finestre vengono scartate. Ne rimangono una piccola percentuale, che dovrebbe rispecchiare il false positive rate calcolato con i test.

Il numero totale di finestre calcolate sull'immagine di prova è $N = 20573$

Il numero totale di detection è $V = 1217$, di cui solo una è giusta.

Il false positive rate su quest'immagine è: $F = \frac{V-1}{N} = 0.0591$

Il false positive rate calcolato con i test era: $F_{test} = 0.0446$

Conclusioni

Sia i test che il riscontro finale fanno capire quanto sia importante che il false positive rate sia molto basso per ottenere dei risultati accettabili (Viola e Jones consigliano 10^{-6}). Anche se il valore ottenuto sperimentalmente potrebbe sembrare non troppo alto, l'elevato numero di finestre valutate su un'immagine fanno sì che una scarsa accuratezza si noti parecchio.

Purtroppo per il training-set utilizzato l'accuratezza del detector non può essere migliorata, perché riconosce già tutte le immagini. Possono essere migliorate o peggiorate le prestazioni, regolando il false positive rate dei singoli stage, ma il numero di stage è inversamente proporzionale al false positive rate, quindi non si avrebbe un miglioramento in termini di accuratezza. Il miglioramento si avrebbe utilizzando un training-set studiato apposta per questo tipo di algoritmo, che a quanto pare è particolarmente esigente in termini di quantità di immagini.

L'ideale sarebbe utilizzare un training-set simile a quello che hanno utilizzato Viola e Jones, composto da 4916 immagini positive e circa 350 milioni di immagini negative, di dimensione 24x24 pixel.

Inoltre un loro ulteriore accorgimento è stato di allenare ogni stage separatamente, prelevando casualmente dal training-set un massimo di 6000 immagini negative per ogni stage.

Il codice del progetto è disponibile open-source all'indirizzo https://github.com/Chosko/img-face-detection

Riferimenti

CBCL Face Database #1. (s.d.). Tratto il giorno 15 Gennaio 2014 da MIT Center For Biological and Computation Learning: http://www.ai.mit.edu/projects/cbcl

Freund, Y., & Schapire, R. E. (1995). A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting.

Viola, P., & Jones, M. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision 57(2)* , p. 137-154.