UNIVERSITÁ DEGLI STUDI DI BRESCIA
FACOLTÁ DI INGEGNERIA

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA DELLE TELECOMUNICAZIONI

Laboratorio specialistico Campi/TLC

Face Detection

Studenti:
Mohamad Farran Matricola: **73606**
Fabrizio Pedersoli Matricola: **72816**

ANNO ACCADEMICO 2009/2010

Indice

1 Introduzione

Quest'esperienza di laboratorio si focalizza su di un particolare ed importante algoritmo implementato all'interno delle Open Source Computer Vision Library (OpenCV).

OpenCV sono delle librerie open-source sviluppate da Intel, scritte in C e progettate per ottenere un'elevata efficienza computazionale con particolare interesse alle applicazione real-time. Queste forniscono all'utente una semplice interfaccia attraverso cui, con poco sforzo, è possibile realizzare sofisticate applicazioni di computer vision.

Con computer vision si intende il processo di trasformazione dei dati rappresentativi di un'immagine od un video in una "decisione" oppure in un nuovo tipo di rappresentazione. L'algoritmo studiato in questa esperienza è un esempio fondamentale, nonché di grande interesse, di questo tipo di applicazioni. L'algoritmo di Viola-Jones permette infatti il riconoscimento dei volti all'interno di un'immagine (face-detection).

2 Face Detection

OpenCV fornisce un face-detector pronto all'uso chiamato Haar Classifier.

Data un'immagine od un flusso video, il face-detector esamina ogni posizione e classifica la rispettiva zona in "face" o "non-face". Siccome un'immagine può raffigurare volti di dimensioni differenti, l'algoritmo deve essere eseguito svariate volte utilizzando scale diverse (default 50×50 pixel), esso presenta delle ottimizzazione che rendono molto efficienti (descritte in sec. 2.1) tutte queste operazioni che sembrano richiedere un'elevato carico computazionale.

Il classificatore utilizza dei dati archiviati in appositi file XML per poter classificare una data porzione di immagine. OpenCV fornisce diversi file adatti per contesti diversi, in particolare è presente un file per il face-detection "frontale" oppure per il profilo, e non solo: include anche un non-face file e full, upper, lower body.

2.1 Viola-Jones

Il face-detector implementato nelle OpenCV utilizza il metodo sviluppato da Paul Viola e Micheal Jones pubblicato nel 2001 dal titolo "Rapid Object Detection using a Boosted Cascade of Simple Features". Quest'approccio combina principalmente quattro fattori chiave:

- Semplici features rettangolari definite Haar features
- L'immagine integrale per velocizzare la classificazione
- Il metodo AdaBoost
- Un classificatore in cascata per combinare efficientemente i risultati

Il metodo Viola-Jones utilizza per la classificazione delle features denominate "Haarlike" del tipo mostrato in fig. 1.

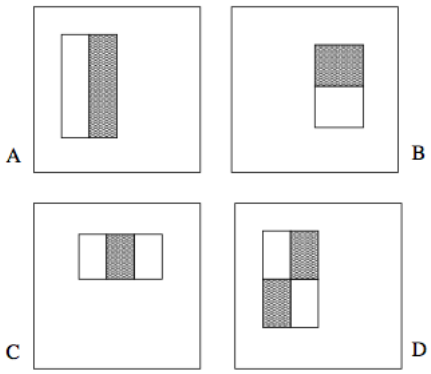


Figura 1: Haarlike features

La presenza di una Haar feature è determinata dalla sottrazione del valor medio dei pixel nella regione scura con il valor medio nella regione chiara. Se la differenza supera una certa soglia impostata durante la fase di training, allora tale features è presente. Per rendere più veloce la procedura di individuazione delle Haar features il metodo Viola-Jones propone l'utilizzo dell'immagine integrale, definita come:

$$ii(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y')$$

per mezzo dell'immagine integrale risulta molto veloce il calcolo dell'integrale in un qualsiasi rettangolo arbitrario, ad esempio in fig. 2

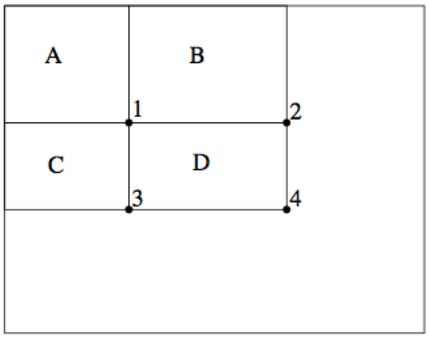


Figura 2: int

$$D = A + B + C + D - (A + B) - (A + C) + A$$

quindi per calcolare D viene effettuata solamente una semplice operazione tra interi:

$$D = (x_4, y_4) - (x_2, y_2) - (x_3, y_3) + (x_1, y_1)$$

Per selezionare l'Haar feature da utilizzare ed il relativo valore si soglia questo algoritmo adopera il metodo AdaBoost.

AdaBoost crea un classificatore "forte" utilizzando tanti classificatori "deboli" . Viola-Jones combina le classificazioni deboli in una catena di filtri del tipo in fig. 3

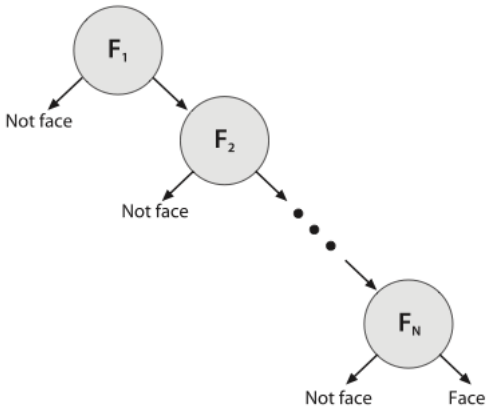


Figura 3: filtri

al fine di ottenere una classificazione efficiente. In pratica ogni filtro è un classificatore debole che rappresenta un Haar feature con un valore di soglia settato in modo tale da dare un risposta positiva per ogni immagine di training. Una regione è classificata "face" se passa tutta la catena di filtri. L'ordinamento dei filtri non è casuale ma è determinato dai pesi assegnati tramite AdaBoost, in pratica i filtri con più "peso" si trovano nelle prime posizioni così che le regioni "non-face" siano scartate il più velocemente possibile.

3 Analisi dell'implementazione in OpenCV

Come è stato visto precedentemente OpenCV offre un object-detector basato sul metodo viola-jones.

Come prima cosa però è necessario effettuare una fase di training, per cui si lascia agire il classificatore su di un database di qualche centinaio di oggetti di interesse. Questi oggetti di interesse possono essere di qualsiasi tipo: volti, auto, ... una volta effettuato il training il metodo funziona indistintamente dal tipo di oggetto da rilevare. Questo procedimento, nel caso di volti, lo esegue direttamente OpenCV fornendo degli appositi file XML.

Un volta che il classificatore è stato addestrato viene applicato alla regione di interesse nell'immagine test, esso restuisce "1" o "0" a seconda che la regione raffiguri o meno l'oggetto desiderato. A questo punto non resta altro che far scorrere la finestra di ricerca in ogni posizione dell'immagine ed eseguire il classificatore. Per essere in grado di rilevare oggetti a dimensioni differenti, il classificatore è stato progettato in modo da essere ridimensionato di conseguenza, così che sia più efficiente rispetto a ridimensionare l'immagine test. Pertanto per trovare un oggetto di dimensione incognita l'intera procedura di classificazione deve essere eseguita svariate volte a scale differenti.

Si ricorda inoltre che il classificatore è costituito da un cascata (cascade) di classificatori più semplici (stages), bastati sulle Haarlike feature ed applicati sequenzialmente alla regione di interesse, al primo stage che fallisce la procedura viene interrotta restituendo "0" (oggetto non trovato). La catena di classificatori è progettata tramite il metodo AdaBoost, in modo da posizionare nelle prime posizioni quelli più significativi.

3.1 Boosted cascade classifier

Di seguito saranno descritte le strutture che implementano il classificatore delle OpenCV

3.1.1 CvHaarFeature

```
#define CV_HAAR_FEATURE_MAX 3

typedef struct CvHaarFeature
{
  int titled;
  struct
  {
    CvRect r;
    float weight;
    rect[CV_HAAR_FEATURE_MAX];
  }
} CvHaarFeature;
```

definisce una haar feature, orizzontale/verticale oppure ruotata di 45°, a seconda del valore di *titled*, che consiste in 2-3 rettangoli con il relativo penso.

3.1.2 CvHaarClassifier

```
typedef struct CvHaarClassifier
{
  int count;
  CvHaarFeature* haar_feature;
  float* threshold;
  int* left;
  int* right;
  float* alpha;
}
CvHaarClassifier;
```

un singolo albero di classificazione, valutato in una particolare posizione dell'immagine test, che riporta il risultato delle haar feature testate nei rispettivi rami (*left* e *right*).

3.1.3 CvHaarStageClassifier

```
typedef struct CvHaarStageClassifier
{
  int count;
  float threshold;
  CvHaarClassifier* classifier;
  int next;
  int child;
  int parent;
}
CvHaarStageClassifier;
```

costituisce il classificatore stage: continente un'array di cvHaarClassifier ed il valore di soglia a livello di stage.

3.1.4 CvHaarClassifierCascade

```
typedef struct CvHaarClassifierCascade
{
  int flags;
  int count;
  CvSize orig_window_size;
  CvSize real_window_size;
  double scale;
  CvHaarStageClassifier* stage_classifier;
  CvHidHaarClassifierCascade* hid_cascade;
}
CvHaarClassifierCascade;
```

cascata dei classificatori stage, definisce parametri quali: le dimensioni delle finestra di ricerca, il fattore di scale ed una versione ottimizzata del cascade.

3.2 Funzioni per l' object detection

In questa sottosezione si vedranno le principali funzioni costituenti l'object-detector

3.2.1 cvHaarDetectObjects

```
CvSeq*
cvHaarDetectObjects(
        const CvArr* image,
        CvHaarClassifierCascade* cascade,
        CvMemStorage* storage,
        double scale_factor=1.1,
        int min_neighbors=3,
        int flags=0,
        CvSize min_size(0,0))
```

richiede come parametri

- *image* immagine test.
- *cascade* Haar classifier cascade
- *storage* area di memoria utilizzata per immagazzinare la sequenza di rettangoli candidati a contenere l'oggetto di interesse
- *scale_factor* fattore di scala attraverso cui la finestra di ricerca viene riscalata nelle scansioni successive
- *min_neighbors* numero minimo di rettangoli "vicini" che creano un'oggetto. Se questo parametro viene settato a zero, non viene effettuato alcun raggruppamento, per cui la funzione restituirà tutti i rettangoli candidati.
- *flags* un flag che indica la modalità di funzionamento, praticamente l'unico che può essere specificato, `CV_HAAR_DO_CANNY_PRUNING`. Se questo è settato la funziona utilizza un Canny edge detector per rifiutare particolari regioni che contengono troppi, o troppo pochi edge, e che pertanto si ritiene che non conterraneo l'oggetto ricercato.
- il valore minimo delle finestra di ricerca, di default coincide che le dimensioni dei campioni utilizzati durante il training.

Questa è la funzione principale, trova le regioni rettangolari nell'immagine test che soddisfano il classificatore, e ritorna tali zone come una sequenza di rettangoli. Questa funzione scansiona l'immagine diverse volte a differenti scale, ogni volta considera regioni sovrapposte ed esegue il classificatore attraverso la funzione *RunHaarClassifierCascade*. Può applicare inoltre qualche euristica, tipo il Canny pruning, al fine di ridurre le regioni da analizzare velocizzando la scansione.

3.2.2 cvSetImageForHaarClassifierCascade

```
void
cvSetImagesForHaarClassifierCascade(
          CvHaarClassifierCascade* cascade,
          const Cvarr* sum,
          const CvArr* sqsum,
          const CvArr titled_sum,
          double scale)
```

richiede come parametri:

- *cascade* Haar classifier cascade
- *sum* immagine integrale a singolo canale a 32bit
- *sqsum* immagine intergrale degli elementi al quadrato in formato 64bit floating point
- *scale* fattore di scale per la finestra di ricerca

Questa funzione è utilizzata per preparare il cascade per il riconoscimento di oggetti di una particolare dimensione nell'immagine test.

3.2.3 RunHaarClassifierCascade

```
int
cvRunHaarClassifierCascade(
        CvHaarClassifierCascade* cascade,
        CvPoint pt,
        int start_stage=0)
```

richiede come parametri:

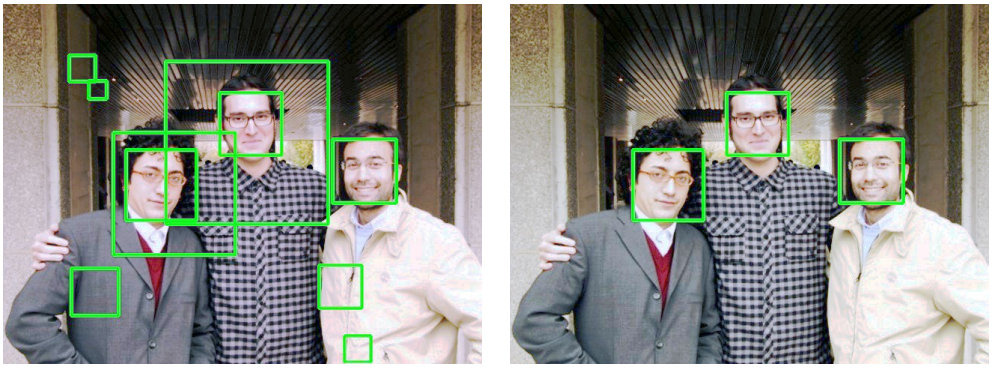
- *cascade* Haar classifier cascade
- *pt* angoli sinistro superiore della regione da analizzare
- *start_stage* indica l'indice della cascata da cui partire

Questa funzione esegue il classificatore cascade in una singola posizione nell'immagine test. Prima di utilizzare tale funzione è necessario aver settato l'immagine integrale e l'appropriata dimensione delle finestra di ricerca attraverso la funzione *SetImageForHaarClassifierCascade*. Viene ritornato un valore positivo se il rettangolo analizzato soddisfa l'intera cascata, zero od un valore negativo altrimenti.

4 Simulazioni

In questa sezione saranno commentati i risultati di face-detecion su di un'immagine test, di dimensione 640×480, al variare dei principali parametri della funzione *cvHaarDetectObjects*. Operativamente si andrà ad agire su *min_neighbors*, *flags* e *min_size*.

Il parametro *min_neighbors*, fig. 4, è fondamentale affinché non ci siano falsi riconoscimenti, infatti una zona in cui è presente una faccia tende solitamente a riportare riconoscimenti multipli. Settare questo parametro per esempio a 3 significa che viene riconosciuto un volto qualora ci siano almeno 3 regioni sovrapposte classificate come tale.



(a) 1 (b) 3

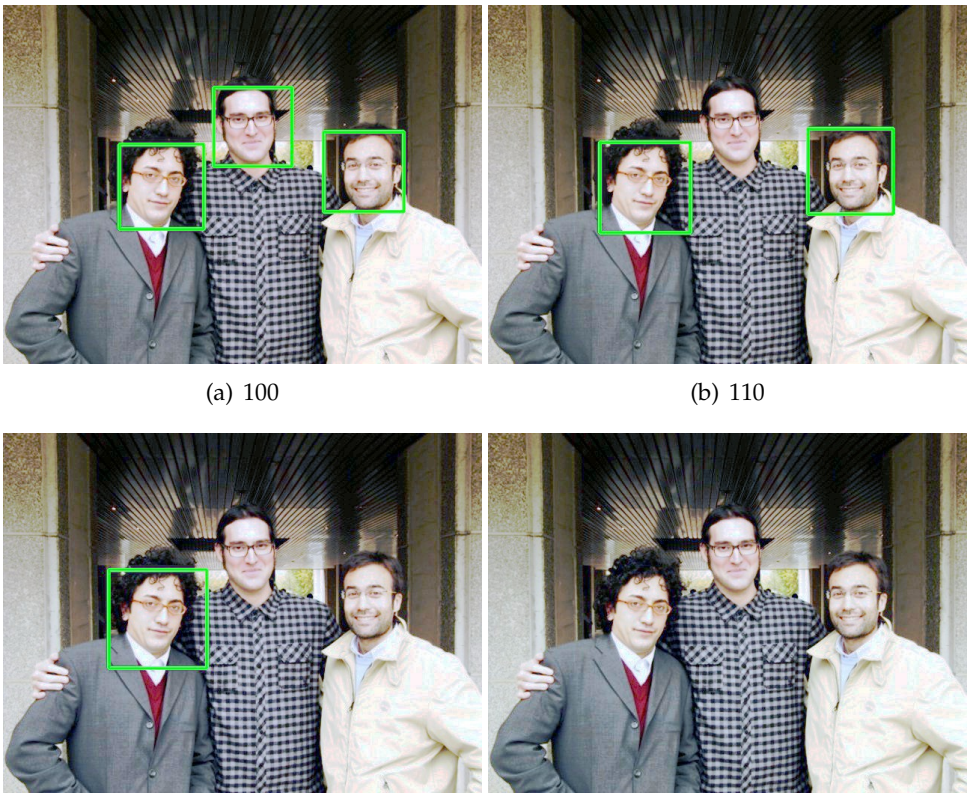
Figura 4: risultati al variare di *min_neighbors*

si noti come nel caso fig. 4(a) siano presenti delle false classificazioni, mentre in fig. 4(b) i volti vengono rilevati correttamente.

Modificano invece il valore di *min_size*, fig. 5 si indica il la dimensione minima della finestra di ricerca. Il valore di default coincide con la dimensione utilizzata in fase di training, che è specificata nel file XML utilizzato. La variazione di questo parametro permette di velocizzare la ricerca, anche di molto, se si ha un conoscenza a priori della dimensione delle facce da ricercare.

Si noti come in questo caso utilizzando una dimensione minima della finestra di 150, fig. 5(d), non si ha riconoscimento, con 120, fig. 5(c), si ha un riconoscimento parziale (una su tre) con 110, fig. 5(b), (due su tre), con 100, fig. 5(a), si ha un riconoscimento totale.

Attraverso il parametro *flags* si modifica in modo marginale il comportamento della funzione. Operativamente con `CV_HAAR_SCALE_IMAGE` si indica all'algoritmo di scalare l'immagine piuttosto che il classificatore, mentre con `CV_HAAR_FIND_BIGGEST_OBJECT` si fa si che la funzione ritorni solamente l'oggetto più grosso tra quelli trovati, fig. 5.



(a) 100 (b) 110

(c) 120 (d) 150

Figura 5: risultati al variare di *min_size*

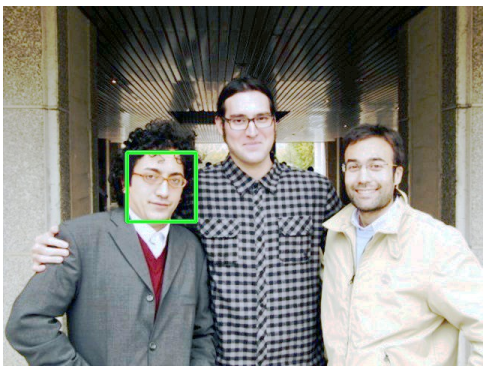


Figura 6: rilevazione dell'oggetto più grande