Università degli Studi dell'Aquila
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

---

Corso di Machine Learning

Riconoscimento facciale: Haar transform, bordi e EigenFaces

Angelo Damiani

Anno Accademico 2017/2018

Indice

Capitolo 1

Struttura del software

1.1 Linguaggio, librerie e repository utilizzate

Per la realizzazione di questo progetto è stato utilizzato come linguaggio di programmazione il Python con le seguenti librerie aggiuntive:

- Tensorflow: una libreria open source originariamente sviluppata dai ricercatori di Google per la ricerca sul deep learning. Fornisce delle API per diversi linguaggi di programmazione (Python, C++, Java, ...). È stata utilizzata per la compatibilità con la libreria Keras;
- Keras: è una high level API scritta in Python per diverse piattaforme (TensorFlow, CNTK e Theano). Originariamente è stata sviluppata per testare e sperimentare reti neurali nel più breve tempo possibile;
- Numpy: è una libreria matematica principalmente utilizzata per calcoli vettoriali e matriciali. Come per tensorflow, anche questa libreria è stata utilizzata per la compatibilità con Keras;
- PIL: è una libreria che si occupa di definire delle funzioni per la manipolazione delle immagini;
- PyWavelets: è una libreria che realizza i vari tipi di trasformata wavelet (tra cui la trasformata Haar);

- Scikit-Learn: è una libreria di machine learning per Python. Di questa libreria è stata usata solo l'implementazione dell'algoritmo PCA.

Il dataset utilizzato per il testing e la sperimentazione è la repository ORL Face dei laboratori della AT&T di Cambridge e, anche se per poca sperimentazione, il IMM Frontal Face Database della Technical University of Denmark.

1.2 Organizzazione del codice

Per la realizzazione di questo progetto sono state implementate 5 classi software:

- `MLP`: wrapper di Keras che si occupa di istanziare un multilayer perceptron, salvarlo, caricarlo da file, allenarlo e interrogarlo.
- `ImageProcessor`: classe che si occupa del processamento delle immagini. Tale processamento comprende il caricamento, salvataggio e conversione dell'immagine in scala di grigi da file, sfocatura, ricerca del contorno, trasformazione mediante trasformata di Haar e conversione da array a file immagine e viceversa.
- `ImageDatasetBuilder`: classe che si occupa della costruzione di dataset numerici a partire da cartelle con immagini (con l'ausilio della classe `ImageProcessor`).
- `DatasetProcessor`: modulo che si occupa di lettura e salvataggio di un dataset.
- `DataAnalyzer`: modulo che si occupa, generalmente, dell'estrazione di statistiche (per il momento solo la predizione esatta) tra due dataset (uno contenente le predizioni e uno contenente ciò che ci si aspetta dalla predizione).

Infine ci sono 2 file di script (`datasetbuilder.py` e `main.py`) che si occupano, rispettivamente, di orchestrare le classi che processano le immagini per la costruzione dei dataset e di allenare e testare la rete neurale.

Capitolo 2

Progettazione e definizione del problema

Ciò che si è realizzato è un sistema di classificazione (dunque riconoscimento) facciale tra diversi individui con l'utilizzo di immagini salvate in files.

2.1 Struttura dei dataset

Per la realizzazione di questo progetto si sono utilizzati 2 dataset:

- ORL Face Dataset (AT&T Laboratories in Cambridge);
- IMM Frontal Face Database (Technical University of Denmark).

Il primo è stato utilizzato per il testing di ogni metodologia che si vedrà nel seguito, mentre il secondo dataset è stato utilizzato per mostrare che un metodo non molto positivo su un genere di dati può essere davvero buono per un altro.

Ad ogni modo, ORL Face Dataset è un dataset composto da un totale di 400 foto scattate su 40 diversi individui tra il 1992 e il 1994 nei laboratori della AT&T. Ciascuna foto è in formato `.gpm` ed è una foto in primo piano sul volto della persona in questione (una foto d'esempio è in figura 2.1). Ogni file è un'immagine di dimensioni 92x112 (per un totale di 10304 pixel) in scala di grigi.

Il secondo dataset, invece, è una repository di foto scattate nel 2005 presso la Technical University of Denmark. A differenza dalla repository ORL, questo

dataset contiene 120 foto a colori scattate su 12 diversi individui e ciascuna di esse è scattata a mezzobusto. Strutturalmente ciascun file è un'immagine `.jpg` di

Figura 2.1: Foto d'esempio del dataset ORL

dimensioni 800x600 (per un totale di 480000 pixel). Un esempio del dataset è dato dalla figura 2.2.

Figura 2.2: Foto d'esempio del dataset IMM

In entrambi i casi si ha un gran numero di pixel e un apprendimento pixel per pixel risulterebbe davvero oneroso in termini di tempo e spazio. Per cui c'è bisogno di estrarre delle feature fondamentali. Nel seguito si parlerà di manipolazione di immagini pensando a queste come delle matrici; con la codifica RGB, in realtà, le matrici sono 3 ma nel progetto è stato effettuata una conversione in scala di grigi ber operare solo su di una.

2.2 Feature extraction: Eigenfaces o trasformata di Haar

I due metodi utilizzati in questo progetto per l'estrazione delle feature sono:

- Principal Component Analysis (che si traduce, nel caso delle immagini, nel metodo delle **eigenfaces**)
- Trasformata di Haar (discreta)

2.2.1 Eigenfaces

Il metodo delle eigenfaces permette di estrarre un numero arbitrario di feature (ovviamente varia la precisione dell'algoritmo di Machine Learning) minore rispetto alla dimensione della foto stessa. Dato un dataset, l'algoritmo per il calcolo delle eigenfaces di ordine K è il seguente:

1: Si converte ogni immagine I_i in un vettore monodimensionale Γ_i
2: Si calcola il vettore medio: $\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i$
3: Si definisce, per ogni i: $\Phi_i = \Gamma_i - \Psi$
4: Definita $A = [\Phi_1 \Phi_2 ... \Phi_M]$ si calcolano gli M autovettori destri v_i di $A^T A$
5: Si calcolano e normalizzano gli autovettori u_i di AA^T tali che $u_i = Av_i$
6: Si prendono i K autovettori u_i relativi agli autovalori più alti di AA^T

A questo punto, ogni $\Phi_i = \Gamma_i - \Psi$ può essere visto come una combinazione lineare: $\Phi_i \simeq \sum_{i=1}^{K} w_i u_i$. Dunque, conoscendo gli autovettori si possono estrarre le K feature w_i; ovviamente più K è grande più accurata sarà la combinazione. Se una nuova immagine Γ arriva nel sistema, è facile calcolare le K feature come segue:

$$\Phi = \Gamma - \Psi \simeq \sum_{i=1}^{K} w_i u_i \rightarrow w_i = u_i^T \Phi \tag{2.1}$$

2.2.2 Trasformata di Haar (discreta)

La trasformata di Haar è un caso particolare della trasformata wavelet. Con la trasformata di Haar discreta, nel caso unidimensionale, è possibile trasformare una qualsiasi sequenza $(a_1, a_2, ..., a_{2n})$ in una sequenza di egual lunghezza strutturata come segue:

$$(\frac{a_1 + a_2}{2}, \frac{a_3 + a_4}{2}, ..., \frac{a_{2n-1} + a_{2n}}{2}, \frac{a_1 - a_2}{2}, \frac{a_3 - a_4}{2}, ..., \frac{a_{2n-1} - a_{2n}}{2}) \tag{2.2}$$

Si definisce allora $s_k = \frac{a_{2k-1}+a_{2k}}{2}$ e $d_k = \frac{a_{2k-1}-a_{2k}}{2}$. E dunque si può scrivere la nuova sequenza, **trasformata**, come:

$$(s_1, s_2, ..., s_n, d_1, d_2, ..., d_n) \tag{2.3}$$

In questo modo la prima metà della nuova sequenza contiene il nucleo più importante del segnale iniziale mentre la seconda parte contiene i dettagli per la ricostruzione. Infatti si può ricostruire il segnale originario, **antitrasformata**, effettuando:

$$a_{2k-1} = s_k + d_k \tag{2.4}$$

$$a_{2k} = 2s_k - a_{2k-1} \tag{2.5}$$

Solitamente la trasformazione avviene più volte, ogni volta sulla prima metà dell'iterazione precedente.

Per quanto riguarda le immagini, esse sono vettori bidimensionali, delle matrici di pixel. In tal caso la trasformata di Haar non è altro che l'applicazione di quest'ultima prima per ogni riga e poi per ogni colonna. In questo modo l'immagine viene settorializzata in quattro quadranti (figura 2.3).

Il quadrante in alto a sinistra indica l'approssimazione, ossia il contenuto principale dell'immagine. Il quadrante in alto a destra prende il nome di dettaglio orizzontale, quello in basso a sinistra prende il nome di dettaglio verticale e, infine, quello in basso a destra prende il nome di dettaglio diagonale.

Come si può notare la dimensione dell'immagine diventa $\frac{1}{4}$ dell'originale. Così facendo si riduce il numero di feature ma può non bastare in caso di immagini grandi come questa. Per questa ragione è necessario un preprocessing.

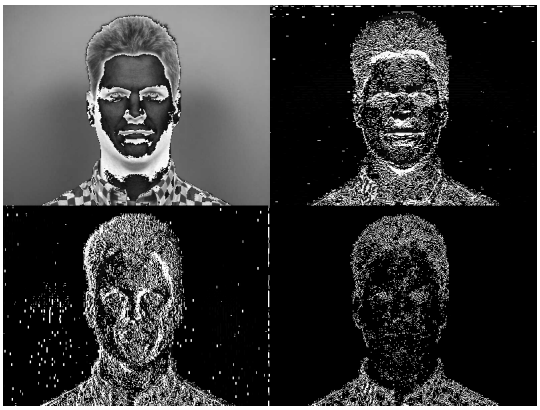

Figura 2.3: Trasformata di Haar della figura 2.2

2.3 Preprocessing: ridimensionamento, sfocatura e ricerca dei bordi

In questo progetto sono state pensate (e utilizzate) 3 tecniche di preprocessing delle immagini. Il ridimensionamento è stato utilizzato per mezzo della libreria PIL mentre le funzioni di sfocatura e di ricerca dei bordi sono state pensate e realizzate.

2.3.1 Sfocatura

L'algoritmo di sfocatura pensato e realizzato è il seguente:

1: **for** $i<K$ **do**
2: **for** each p_i dell'immagine **do**
3: il valore di p_i diventa la media dei valori degli 8 pixel circostanti
4: **end for**
5: $i++$
6: **end for**

Scegliendo $K = 20$ il risultato è mostrato in figura 2.4. La sfocatura è utile affinchè immagini somiglianti (ma con una leggera differenza) risultino molto simili tra loro e, dunque, raffiguranti la stessa persona.


Figura 2.4: Comparazione tra la foto originale e quella dopo la sfocatura

2.3.2 Ricerca dei bordi

Un'altra forma di preprocessing che si è pensato potesse essere utile riguarda il rilevamento dei contorni. Come si vedrà nella sperimentazione, invece, l'algoritmo sviluppato risulta essere poco performante (sia in termini computazionali ma soprattutto in termini di risultati). Ad ogni modo l'algoritmo è il seguente:

1: Inizializza un'immagine vuota della stessa dimensione del target
2: **for** each pixel p_i dell'immagine **do**
3: **if** $p_i > \rho\mathbb{E}$[degli 8 pixel circostanti] **then**
4: il valore di p_i della nuova immagine diventa 255
5: **else**
6: il valore di p_i della nuova immagine diventa 0
7: **end if**
8: **end for**

Il risultato di questo algoritmo per $\rho = 0.88$ si ha l'immagine in figura **??**.

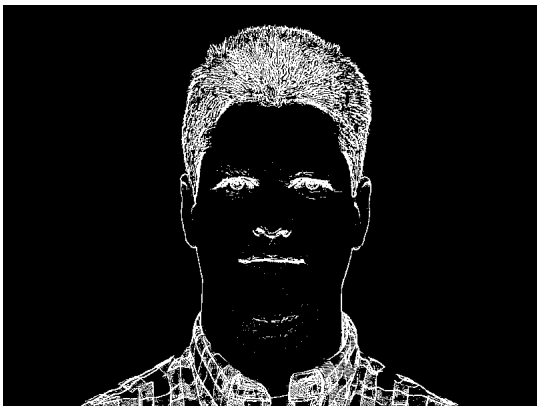

Figura 2.5: Ricerca dei bordi della figura 2.2

2.4 Combinazione delle tecniche

Nella sperimentazione sono state utilizzate delle combinazioni dei metodi precedenti. Nel particolare sono state fatte le seguenti sperimentazioni sul dataset ORL Face:

- Resize (a 28x34 pixel) → sfocatura → eigenfaces (a 50, 100, 150, 200 e 250 componenti)
- Resize (a 28x34 pixel) → ricerca dei bordi → eigenfaces (a 50, 100, 150, 200 e 250 componenti)
- Resize (a 28x34 pixel) → eigenfaces (a 50, 100, 150, 200 e 250 componenti)
- Sfocatura → eigenfaces (a 50, 100, 150, 200 e 250 componenti)
- Ricerca dei bordi → eigenfaces (a 50, 100, 150, 200 e 250 componenti)
- Eigenfaces (a 50, 100, 150, 200 e 250 componenti)
- Trasformata di Haar discreta

Mentre per il dataset IMM sono state effettuate solo due sperimentazioni:

- Resize (a 64x48 pixel) → Trasformata di Haar discreta
- Resize (a 64x48 pixel) → eigenfaces (a 50, 100, 150, 200 e 250 componenti)

Capitolo 3

Sperimentazione

Le sperimentazioni effettuate sono state realizzate allenando reti neurali con 5 layers.

Si sottolinea che per questo progetto non è stata utilizzata alcuna forma di codifica di input (al di fuori del preprocessing affrontato nel capitolo precedente). Per quanto riguarda l'output, essendo un problema di classificazione, è stata effettuata una codifica one-hot vector. Infatti, in uscita dalla rete neurale sarà presente un vettore di dimensione 40 (nel caso di ORL) o 12 (nel caso di IMM); il valore più alto di questo vettore viene convertito in 1 e gli altri in 0. Infine, l'indice del vettore in cui è presente l'1 sarà il numero associato all'individuo predetto.

Inoltre, per ogni dataset è stato estratto il 90% e usato come training set (9 foto su 10 di ogni individuo) lasciando il rimanente 10% come testset. Il training set è stato usato, per ogni sperimentazione, per 200 epoch. I risultati risultano essere tutti ottimi con una media dei risultati sopra al 90% con alcuni casi che arrivano anche al 100%. Questi risultati, forse, sono dovuti alle dimensioni ridotte delle immagini per cui, per immagini più grandi, alcuni algoritmi di preprocessing potrebbero risultare migliori di altre e secondo la letteratura attuale, uno dei metodi migliori risulta essere quello delle Eigenfaces.

I risultati sono racchiusi nelle seguenti tabelle. Nel campo *Rete neurale* viene mostrato il numero di unità per ogni layer della rete neurale (viene escluso il layer di output sempre uguale al numero di classi del dataset). Per quanto riguarda l'algoritmo *Eigenfaces* viene posto come parametro il numero di componenti con cui è stata fatta la sperimentazione.

Dataset	Algoritmi	Accuratezza	Rete neurale
ORL	Resize (28x34px) Eigenfaces (50)	90%	50 → 100 → 150 → 100 → 50
ORL	Resize (28x34px) Sfocatura Eigenfaces (50)	97.5%	50 → 100 → 150 → 100 → 50
ORL	Resize (28x34px) Ricerca dei bordi Eigenfaces (50)	**87.5%**	50 → 100 → 150 → 100 → 50
ORL	Resize (28x34px) Eigenfaces (100)	95%	100 → 200 → 300 → 200 → 100
ORL	Resize (28x34px) Sfocatura Eigenfaces (100)	97.5%	100 → 200 → 300 → 200 → 100
ORL	Resize (28x34px) Ricerca dei bordi Eigenfaces (100)	92.5%	100 → 200 → 300 → 200 → 100
ORL	Resize (28x34px) Eigenfaces (150)	95%	150 → 300 → 450 → 300 → 150
ORL	Resize (28x34px) Sfocatura Eigenfaces (150)	97.5%	150 → 300 → 450 → 300 → 150
ORL	Resize (28x34px) Ricerca dei bordi Eigenfaces (150)	92.5%	150 → 300 → 450 → 300 → 150
ORL	Resize (28x34px) Eigenfaces (200)	97.5%	200 → 400 → 600 → 400 → 200
ORL	Resize (28x34px) Sfocatura Eigenfaces (200)	**100%**	200 → 400 → 600 → 400 → 200
ORL	Resize (28x34px) Ricerca dei bordi Eigenfaces (200)	95%	200 → 400 → 600 → 400 → 200

Dataset	Algoritmi	Accuratezza	Rete neurale
ORL	Resize (28x34px) Eigenfaces (250)	**100%**	250 → 500 → 750 → 500 → 250
ORL	Resize (28x34px) Sfocatura Eigenfaces (250)	**100%**	250 → 500 → 750 → 500 → 250
ORL	Resize (28x34px) Ricerca dei bordi Eigenfaces (250)	**82.5%**	250 → 500 → 750 → 500 → 250
ORL	Eigenfaces(50)	90%	50 → 100 → 150 → 100 → 50
ORL	Sfocatura Eigenfaces (50)	95%	50 → 100 → 150 → 100 → 50
ORL	Ricerca dei bordi Eigenfaces (50)	**85%**	50 → 100 → 150 → 100 → 50
ORL	Eigenfaces (100)	97.5%	100 → 200 → 300 → 200 → 100
ORL	Sfocatura Eigenfaces (100)	95%	100 → 200 → 300 → 200 → 100
ORL	Ricerca dei bordi Eigenfaces (100)	85%	100 → 200 → 300 → 200 → 100
ORL	Eigenfaces (150)	95%	150 → 300 → 450 → 300 → 150
ORL	Sfocatura Eigenfaces (150)	**100%**	150 → 300 → 450 → 300 → 150
ORL	Ricerca dei bordi Eigenfaces (150)	**87.5%**	150 → 300 → 450 → 300 → 150

Dataset	Algoritmi	Accuratezza	Rete neurale
ORL	Eigenfaces (200)	97.5%	200 → 400 → 600 → 400 → 200
ORL	Sfocatura Eigenfaces (200)	95%	200 → 400 → 600 → 400 → 200
ORL	Ricerca dei bordi Eigenfaces (200)	**80%**	200 → 400 → 600 → 400 → 200
ORL	Eigenfaces (250)	95%	250 → 500 → 750 → 500 → 250
ORL	Sfocatura Eigenfaces (250)	95%	250 → 500 → 750 → 500 → 250
ORL	Ricerca dei bordi Eigenfaces (250)	**87.5%**	250 → 500 → 750 → 500 → 250
ORL	Trasformata di Haar	**72.5%**	100 → 200 → 300 → 200 → 100
IMM	Resize (64x48px) Trasformata di Haar	**100%**	100 → 200 → 300 → 200 → 100
IMM	Resize (64x48px) Eigenfaces (50)	**100%**	50 → 100 → 150 → 100 → 50
IMM	Resize (64x48px) Eigenfaces (100)	**100%**	100 → 200 → 300 → 200 → 100
IMM	Resize (64x48px) Eigenfaces (150)	**100%**	150 → 300 → 450 → 300 → 150
IMM	Resize (64x48px) Eigenfaces (200)	**100%**	200 → 400 → 600 → 400 → 200
IMM	Resize (64x48px) Eigenfaces (250)	**100%**	250 → 500 → 750 → 500 → 250

3.1 Sperimentazione finale

Oltre a quelle citate è stata fatta una sperimentazione più "concettuale" finale. Il sistema era stato progettato per prendere dimestichezza con le tecnologie di preprocessing disponibili per le foto. Una volta fatte le sperimentazioni è stato provato (poi parzialmente rimosso) un approccio più utile. Mentre nelle prime sperimentazioni il sistema doveva essere solo in grado di distinguere tra un numero fisso di persone, in questa sperimentazione si è provato a distinguere persone conosciute (primi 30 individui del dataset ORM) da persone sconosciute (restanti 10). Tale approccio può essere visto nell'ottica di un riconoscimento di sicurezza in una azienda o simili.

Dunque per questa sperimentazione si è operato come in precedenza ma affibiando la stessa classe (classe 8 per convenzione interna ma sta a simboleggiare l'individuo sconosciuto) agli ultimi 10 individui del dataset. Dunque si è effettuato un resizing delle immagini a 28x34 pixel, è stato utilizzato l'algoritmo delle eigenfaces con 50 componenti e in più si è modificato il modo di interpretare l'output della rete neurale (questa modifica è stata poi rimossa). L'output, in questo contesto, doveva essere certo (o quasi) della persona della foto e dunque l'output non sarà più la posizione in cui il sigmoide di uscita ha valore più alto ma è stata fissata una soglia (0.8) sotto la quale l'individuo sarà reputato estraneo (ossia con la classe 8).

Dunque la sperimentazione ha prodotto le seguenti predizioni:

```
Using TensorFlow backend.
Hello, this is a basic software for training and quering a neural network
What do you want to do? ('T' to train a new one, 'V' to test an already existing neural network, anything else to close: t
Please insert the neural network name (it's only used to SAVE the neural network): ORL-Custom-28-50
Insert the path to dataset that will be used to train the network: files/datasets/ORL-Custom/28/50/normal
How many layers should the neural network have? 5
Insert the number of units for layer 0: 50
Insert the number of units for layer 1: 100
Insert the number of units for layer 2: 150
Insert the number of units for layer 3: 100
Insert the number of units for layer 4: 50
With how many samples do you want to train the network? ALL
For how many epoch do you want to train the network? 200
On how many samples the network should be tested? ALL
Neural network trained
Test performed
[27, 28, 29, 30, 29, 8, 8, 8, 8, 8, 29, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 30, 8, 8, 8, 8, 8, 2, 8, 29, 8, 8]   Predizioni
[27.0, 28.0, 29.0, 30.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0]
```

Labels

Quindi si è testata la rete su un test set di 30 sconosciuti e 4 volti noti. I volti noti sono stati pienamente riconosciuti. Gli sconosciuti, invece, sono stati "scambiati" per volti noti 6 volte su 30 (20% delle volte). Nel complesso l'accuratezza risulta essere del 85.3% ma in alcuni contesti ciò può essere molto basso, soprattutto dal

punto di vista della sicurezza (magari di un edificio). Per tali contesti bisognerebbe alzare la soglia e/o utilizzare altre tecniche.

3.2 Conclusioni

Dunque questo progetto mi ha portato a conoscere vari algoritmi di preprocessing delle immagini (principalmente eigenfaces e trasformata di Haar) e a doverli comprendere ed implementare. La mia realizzazione, però, ha numerosi vincoli. Il principale sta nell'architettura del software; con tutta onestà è stato continuato lo sviluppo a partire dagli homework durante il corso e dunque molti errori sono stati portati dietro da questi. Inoltre, personalmente, ho reputato più importante cercare di comprendere degli algoritmi di processamento piuttosto che trattare la qualità del software, compito che secondo me deve essere demandato a un altro tipo di corsi. Dunque il principale sviluppo futuro di questo progetto potrebbe essere la ricostruzione più flessibile del programma e delle classi e un ingegnerizzazione di tutto il software. La principale delle aggiunte da fare è la possibilità di interrogare la rete neurale. In questo progetto la rete è stata interrogata solo dopo il training e usando il dataset poichè, a causa delle eigenfaces, risulterebbe complesso il salvataggio della base per trasformare la query in una combinazione lineare. Per questa ragione è stata rimossa la possibilità di effettuare query.

Un'altra nota dolente del progetto sta nella qualità e nella quantità delle immagini. Purtroppo dove vi sono molte foto (dataset ORL) la qualità delle foto è scarsa (visi quasi in primissimo piano in bassa risoluzione) mentre dove vi sono fotografie ad una risoluzione soddisfacente (dataset IMM) ci sono pochi individui (e foto per singoli individui). Inoltre il processamento di immagini ad alta risoluzione (anche 1920x1080) risulta più onerosa per gli algoritmi proposti senza un precedente resizing. Dunque un altro possibile continuo di questo progetto potrebbe essere la raccolta di più fotografie di visi ad alta risoluzione e il tentativo di ottimizzare gli algoritmi per un'esecuzione efficiente (senza o con ridimensionamento limitato delle immagini).

Infine si aggiunge un'ultima considerazione, quasi una battuta, ogni algoritmo visto in precedenza presenta un'accuratezza buona (nel contesto di questo tipo di immagini) ma uno è principe negativo ed è uno dei due sviluppato dal sottoscritto: la

ricerca dei bordi. I risultati per quell'algoritmo sono persino inferiori all'inserimento dei dati grezzi delle immagini all'interno della rete ma ne è valsa la pena tentare.