

RELAZIONE SUL FEEDBACK DI RILEVANZA E MISURA DI SIMILARITÀ PER IL RECUPERO DI IMMAGINI CON RETI NEURALI SINERGICHE

MARIANI FILIPPO, PETROSILLI MARCO, ROSELLI PAOLO *

Introduzione. Con “image retrieval”, intendiamo un sistema che attraverso un input, e l’aiuto dallo user feedback, ci restituisca un insieme di elementi in quel dominio (in questo caso immagini) simili alla query. Con questa parola identifichiamo l’immagine che viene confrontata con le altre presenti nel database e permette al sistema di mostrare in output dei risultati, i quali vengono scelti per la loro somiglianza. Per “feedback di rilevanza” invece consideriamo il meccanismo interattivo con il quale l’utente aiuta il computer a trovare risultati sempre più accurati. Ad ogni esecuzione l’utente può cliccare su un risultato per comunicare al sistema il suo parere riguardo il grado di somiglianza, in questo modo al prossimo avvio del programma la query sarà più precisa.

Materiale usato. Per la realizzazione del progetto abbiamo utilizzato principalmente tre strumenti:

1)MATLAB, un ambiente scritto in C che permette di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi e creare interfacce utente. Per creare quest’ultima abbiamo utilizzato l’app-Designer. Attraverso la “codeView” abbiamo implementato il nostro codice relativo ad ogni componente precedentemente salvato nelle apposite classi nel software di Matlab (approfondimento nella sezione “implementazione gui”). Nella cartella contenente tutte le classi abbiamo inoltre salvato il database di immagini.



2)GITHUB, è un servizio utilizzato per il controllo delle versioni attraverso Git. Inizialmente abbiamo creato la repository dove abbiamo caricato il nostro progetto. Nel ramo principale del nostro archivio chiamato “master” abbiamo caricato tutte le modifiche che hanno portato alla realizzazione del software. Abbiamo suddiviso le modifiche in più commit, così siamo stati in grado di dividerci al meglio i compiti ed inoltre abbiamo potuto tener traccia delle modifiche e in caso di qualche “passo falso” poter tornare indietro annullando le modifiche attraverso il revert delle stesse.



3)PAINT, è un programma di grafica semplice e l’abbiamo utilizzato per ritagliare ed aggiustare il nostro database di immagini. Abbiamo reso ogni immagine quadrata e abbiamo pulito lo sfondo per un miglior funzionamento del software.



*Dipartimento di Ingegneria dell’Informazione (DII), Università Politecnica delle Marche (UnivPM), Via Brecce Bianche, I-60131 Ancona. Autori: Filippo Mariani, Marco Petrosilli, Paolo Roselli. Questo documento è aggiornato al giorno 13 gennaio 2023.

Relazioni matematiche principali. Per la realizzazione dell'algoritmo siamo partiti dal conoscere quali sono gli elementi che entrano in gioco (matrici, vettori, ecc.) e le conseguenti relazioni matematiche che li legano e ne consentono l'utilizzo. Dapprima bisogna introdurre le due matrici:

$$(1) \quad \mathbf{V}^+ = \left[(\boldsymbol{\nu}_1^+)' (\boldsymbol{\nu}_2^+)' \dots (\boldsymbol{\nu}_p^+)' \right]'$$

$$(2) \quad \mathbf{V} = \left[\boldsymbol{\nu}_1 \quad \boldsymbol{\nu}_2 \quad \dots \quad \boldsymbol{\nu}_p \right]$$

I singoli vettori di quest'ultime sono invece definiti come segue:

$$(\boldsymbol{\nu}_k^+ \cdot \boldsymbol{\nu}_i) = \delta_{ki} \text{ , con } \delta_{ki} \text{ funzione delta di Kronecker}$$

Ciò ci permette di concludere che le due matrici viste precedentemente sono legate in modo da essere l'una la pseudo-inversa dell'altra (pseudo- perché non si ha la garanzia che queste siano quadrate)

$$\begin{bmatrix} \boldsymbol{\nu}_1^+ \\ \boldsymbol{\nu}_2^+ \\ \dots \\ \boldsymbol{\nu}_p^+ \end{bmatrix} \begin{bmatrix} \boldsymbol{\nu}_1 & \boldsymbol{\nu}_2 & \dots & \boldsymbol{\nu}_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & 1 & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix} = \mathbf{I}$$

Grazie a questa relazione appena vista saremo in grado di ricavare i vettori della matrice (2) da quelli della (1). Quest'ultimi infatti vengono ricavati direttamente dalle immagini, grazie all'algoritmo CLM di cui si parlerà in seguito nell'apposita sezione.

Ci è poi utile introdurre il concetto di “vettore di ordine”, vettore popolato come segue prendendone altri due in entrata, le sue componenti ci daranno una prima misura di similarità tra i due vettori:

Vettore di ordine :

$$\mathbf{O}(\mathbf{a}, \mathbf{b}) = \mathbf{a} \circ \mathbf{b} = (a_1, a_2, \dots, a_n) \circ (b_1, b_2, \dots, b_n) = (a_1 b_1, a_2 b_2, \dots, a_n b_n)$$

$$E \text{ distinguiamo : } \mathbf{O}^{\text{pos}} = \{o_i \mid o_i > 0\} \text{ , } \mathbf{O}^{\text{neg}} = \{o_i \mid o_i < 0\}$$

Che sono rispettivamente la parte simile e quella dissimile delle due immagini. Tuttavia si avranno anche delle parti di quasi-zeri:

$$\mathbf{O}^{\text{zero}} = \{o_i \mid |o_i| < \delta\} \text{ con } \delta \text{ piccola costante positiva}$$

E per eliminare queste componenti insignificanti ridefiniamo il vettore di ordine come:

$$\mathbf{O}^\delta(\mathbf{a}, \mathbf{b}) = \{o_i^\delta(\mathbf{a}, \mathbf{b}), i = 1, \dots, m\} \quad \text{dove } o_i^\delta(\mathbf{a}, \mathbf{b}) = \begin{cases} a_i b_i - \delta & \text{se } a_i b_i > \delta \\ 0 & \text{altrimenti} \end{cases}$$

Introduciamo ora una funzione che, restituendo una grandezza scalare, ci permette di quantificare il grado di similarità di due immagini, questa prende infatti il nome di “Misura di similarità”:

$$(3) \quad S^\delta(\boldsymbol{\nu}_k, \mathbf{q}) = \sum_i o_i^\delta(\boldsymbol{\nu}_k^+, \mathbf{q})$$

Tuttavia questa funzione potrebbe portare a delle ambiguità, di conseguenza definiamo la “Misura di similarità normalizzata”, che ha comunque il medesimo fine di quella vista prima :

$$(4) \quad SN^\delta(\boldsymbol{\nu}_k, \mathbf{q}) = \left\{ \frac{S^\delta(\boldsymbol{\nu}_k, \mathbf{q})}{\sum_{k=1}^p S^\delta(\boldsymbol{\nu}_k, \mathbf{q})} \right\}$$

Ridefiniamo poi ulteriormente la (3) come segue, anche qui lo scopo di questa funzione resta lo stesso della “Misura di similarità”, ma in questo caso riusciamo a svolgere un’opera di ”filtraggio” delle informazioni rilevanti di q :

$$(5) \quad S^{\text{self-att}}(\boldsymbol{\nu}_r, \mathbf{q}) = \sum_i |o_i^\delta(\boldsymbol{\nu}_r^+, \boldsymbol{\nu}_r) - o_i^\delta(\boldsymbol{\nu}_r^+, \mathbf{q})|$$

Volgiamo ora l’attenzione sullo user feedback, seppure l’algoritmo operi tramite specifiche equazioni che consentono di calcolare la similarità tra immagini, questo non conosce tutti gli aspetti più sottili con i quali invece noi esseri umani notiamo la somiglianza tra due oggetti. Il programma deve quindi riaggiustare il suo calcolo con il passare delle ricerche, e questo proprio grazie ad un giudizio dell’utente, che basandosi sull’immagine recuperata come “più simile” gli assegna un peso di somiglianza secondo la seguente scala.

$$(6) \quad w_{\text{si}} = \begin{cases} 2 & \text{se molto simile} \\ 1.5 & \text{se più simile} \\ 1 & \text{se simile} \\ 0.5 & \text{se poco simile} \\ 0.1 & \text{se leggermente simile.} \end{cases}$$

Dopodichè questo verrà implementato ricalcolando una nuova visual keyword (combinando quella vecchia, con l’immagine più simile pesata tramite giudizio dell’utente) e ricalcolando la S self-attentive:

$$(7) \quad \boldsymbol{\nu}_{\text{new}} = \frac{w_{\text{si}} \cdot \boldsymbol{\nu}_{\text{si}} + \boldsymbol{\nu}_r}{1 + w_{\text{si}}}$$

$$(8) \quad S_{\text{new}}^{\text{self-att}}(\boldsymbol{\nu}_{\text{new}}, q) = \sum_i |o_i^\delta(\boldsymbol{\nu}_{\text{new}}^+, \boldsymbol{\nu}_{\text{new}}) - o_i^\delta(\boldsymbol{\nu}_{\text{new}}^+, \mathbf{q})|$$

Dove $\boldsymbol{\nu}_{\text{si}}$ è il vettore contenente le informazioni relative all’immagine recuperata come “più simile” dal software.

Possiamo infine introdurre tre indici, che ci permettono di stimare quanto il nostro sistema abbia lavorato in maniera efficiente e precisa, questi sono :

- R_n : Normalized recall
- P_n : Normalized precision
- L_n : Last place ranking

Dove R_n attribuisce un peso maggiore al successo nel recupero dei primi elementi, mentre P_n attribuisce uguale peso a tutti i recuperi. L_n indica invece il numero di pattern recuperati che un utente deve cercare per avere una ragionevole aspettativa di trovare tutti i pattern rilevanti. Questi sono definiti come segue:

$$R_n = 1 - \frac{\sum_{i=1}^n R_i - \sum_{i=1}^n i}{n(N - n)}$$

$$P_n = 1 - \frac{\sum_{i=1}^n (\log R_i) - \sum_{i=1}^n (\log i)}{\log(N!/n!(N - n)!)}$$

$$L_n = 1 - \frac{R_l - n}{N - n}$$

Dove R_i è il rango in cui il pattern rilevante i viene effettivamente recuperato, n è il numero totale di pattern rilevanti, N è la dimensione dell’intero database di immagini e R_l è il rango dell’ultima immagine rilevante. Gli indici appena visti assumono valori compresi tra 0 e 1, tanto più questi sono vicini a 1 tanto più il software avrà lavorato in maniera precisa e efficiente.

Complex Logarithm Mapping. Vediamo ora come vengono acquisite informazioni rilevanti per ogni immagine, come si arriva da quest'ultima così com'è ad avere un vettore contenente valori ad essa relativi.

Questa operazione è concretizzata grazie all'algoritmo CLM, Complex Logarithm Mapping.

Ma andiamo in ordine, inizialmente ci si occupa della fase di lettura e adattamento dell'immagine.

Sfruttando infatti il comando "imread" si va a leggere l'immagine, questo restituisce un vettore bidimensionale ($N \times M$) nel caso l'immagine sia in scala di grigi, altrimenti un array tridimensionale ($N \times M \times 3$) se l'immagine è a colori. Dopodiché ciò che si ottiene da "imread" viene ridimensionato con il comando "imresize", questo per lavorare in primis con le stesse dimensioni per tutte le immagini, e in secondo luogo anche per avere matrici quadrate.

L'ultimo passaggio infine, prima di invocare la funzione CLM da noi messa a punto, è sfruttare il comando "im2gray" che converte l'immagine specificata (la stessa che è stata precedentemente ridimensionata) in un'immagine in scala di grigi.

Successivamente inizia il CLM vero e proprio, viene chiamata infatti l'omonima funzione che fondamentalmente si suddivide in 4 step.

Sull'immagine, passata come parametro al CLM, viene utilizzato il comando "fft2", con il quale viene fatta la trasformata di Fourier della nostra matrice. Dopodiché su ciò che si ottiene viene invocato il comando "fftshift", ossia la trasformata di Fourier centrata, che riorganizza la trasformata che vuole come parametro, ponendo al centro dell'array la cosiddetta "componente a frequenza zero". Ciò che si ottiene viene poi passato come argomento alla funzione "log" (sommato di 1) e quello che infine si ottiene è una matrice contenente i valori su cui poi si andrà effettivamente a lavorare.

Tuttavia sopra si parlava di applicare quelle equazioni e relazioni su dei vettori, ognuno dei quali derivante da una singola immagine, l'ultimo passaggio della funzione CLM è infatti applicare il comando "reshape" alla matrice di cui si è parlato poc'anzi, quest'ultimo permette di trasformare una matrice in un unico vettore riga, così facendo infatti, all'atto pratico ogni immagine ci permetterà di ottenere un vettore

Riportiamo di seguito:

- Codice per lettura dell'immagine

```
1      imageSize = StoreData.GetImageSize();

3      Vpiu = zeros(nfiles, imageSize.^2);

5      imdata = imread(fullName);
      imdata = imresize(imdata, [imageSize imageSize]);
7      imdata = im2gray(imdata);

9      VpiuKRow = clm(imdata);
```

- Corpo della funzione CLM:

```
1      function X = clm(A)
      F = fft2(A);

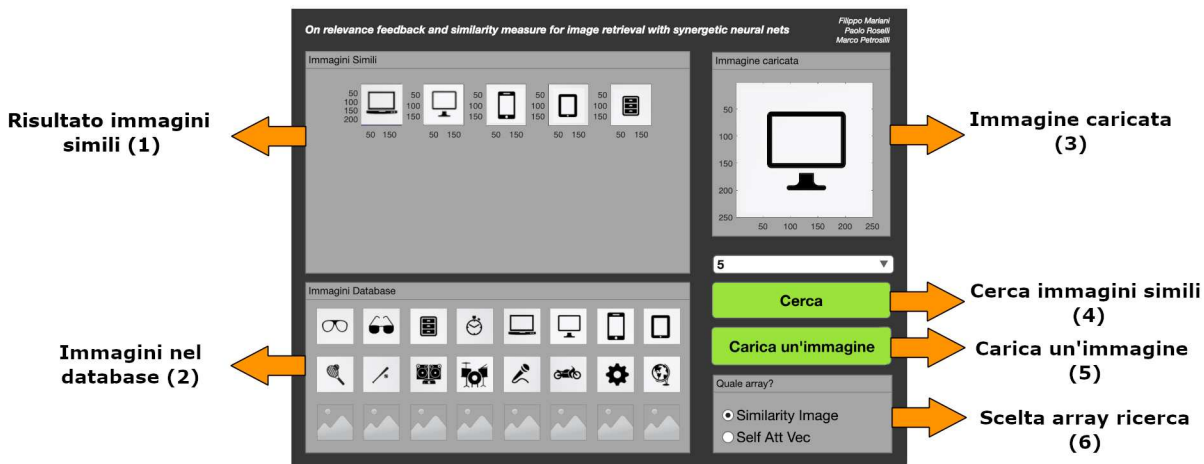
3      Fsh = fftshift(F);

5      Vpiu = log(1+Fsh);

7      X = reshape(Vpiu, 1, []);

9      end
```

Implementazione GUI. Attraverso l'app designer di Matlab abbiamo costruito la sottostante parte grafica.



Risultato immagini simili (1) = In questo pannello vengono mostrate un numero variabile di immagini a seconda della preferenza dell'utente. Durante l'esecuzione del programma e prima di premere sul pulsante cerca, l'utente può decidere quante foto simili stampare tra 5,10,15 attraverso la selezione nel dropdown. Le immagini stampate sono il risultato del sistema.

Immagini nel database (2) = In questo pannello vengono visualizzate 24 immagini tra quelle presenti nel database.

Immagine caricata (3) = Qui viene visualizzata l'immagine che l'utente ha deciso di utilizzare come query di ricerca precedentemente caricata (5).

Cerca immagini simili (4) = Questo bottone avvia la ricerca delle immagini simili, infatti collegato a quest'ultimo c'è il callback, il quale esegue la main function del programma.

Carica un'immagine (5) = Questo bottone permette all'utente di scegliere un'immagine, dovrà selezionare dal finder di sistema un'immagine a suo piacimento, la quale verrà in seguito usata come query e confrontata con le altre nel database.

Scelta array ricerca (6) = In questo pannello sono contenuti due radio button che permettono di scegliere o una o l'altra opzione, di default viene selezionato l'array di ordine (vedere order vector) altrimenti l'utente può decidere di utilizzare il Self-att*, in base alla scelta verranno eseguiti due procedimenti diversi nella parte principale di codice.

Funzionamento. L'implementazione dell'Image Retrieval è basata sull'applicazione delle equazioni citate in precedenza. Si parte andando a costruire la matrice V_{piu} , ricavata applicando la procedura descritta nella sezione "Complex Logarithm Mapping" calcolata su ogni immagine utilizzata per la ricerca. Quindi ogni riga della matrice V_{piu} corrisponde ad una visual keyword. Il passo successivo è quello di ricavare la matrice V , inversa di V_{piu} . Essendo la matrice V_{piu} non quadrata nella maggior parte dei casi, si ricava la sua inversa calcolando la pseudo inversa di Moore-Penrose. A questo punto si hanno tutte le informazioni per classificare le immagini selezionate. Andremo ora a confrontare una ad una le immagini del database con la nostra matrice V_{piu} . Per ogni immagine si applica il CLM, "Complex Logarithm Mapping", così da avere i dati nella stessa forma per poterli confrontare. A seconda del metodo di classificazione che andremo a scegliere si differenzieranno i procedimenti svolti nella parte centrale dell'esecuzione:

1. Order Vector

- L'order vector si basa sull'implementazione della sola equazione

$$O^\delta(\nu_k, q) = \{o_i^\delta(\nu_k, q), i = 1, \dots, m\} \quad \text{dove } o_i^\delta(\nu_k, q) = \begin{cases} \nu_{k,i}q_i - \delta & \text{if } \nu_{k,i}q_i > \delta \\ 0 & \text{altrimenti} \end{cases}$$

Proseguiamo facendo la sommatoria di tutti i componenti del vettore in modo tale da ottenere uno scalare.

$$(3) \quad S^\delta(\nu_k, q) = \sum_i o_i^\delta(\nu_k^+, q)$$

Questo valore rappresenta la similarità con l'immagine k-esima tra le immagini usate per la ricerca, le visual keywords. Per estrarre la classifica vengono fatti dei passaggi comuni ad entrambi i procedimenti che verranno spiegati nel prossimo punto.

2. Self-att

- Il self-att si basa sull'applicazione della seguente equazione

$$S^{\text{self-att}}(\nu_r, q) = \sum_i |o_i^\delta(\nu_r^+, \nu_r) - o_i^\delta(\nu_r^+, q)|$$

dove i vettori O^δ vengono calcolati usando l'equazione rappresentata al punto precedente. Il risultato di questa equazione è uno scalare che rappresenta il valore di somiglianza della foto del database con la k-esima tra quelle usate per la ricerca. Il processo di normalizzazione per ottenere un valore compreso tra 0 e 1 viene spiegato nel prossimo paragrafo.

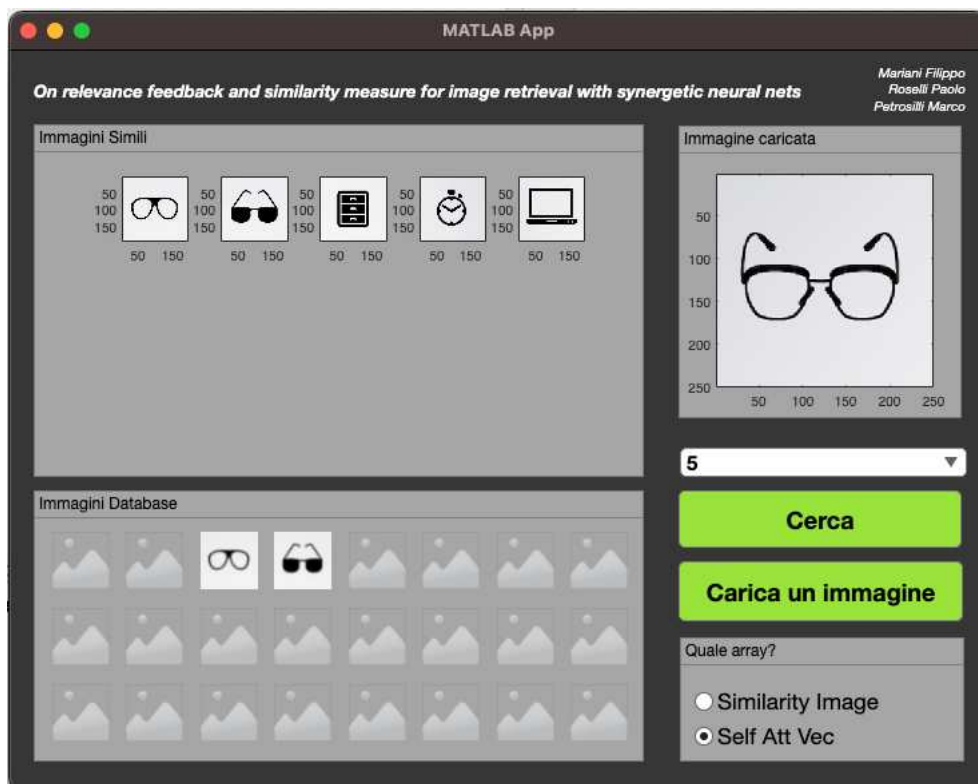
- Il self-att può essere utilizzato per ricalibrare la query utilizzando il feedback dell'utente. L'utente esprime un giudizio sulla foto seguendo i criteri illustrati alla (6). Si prosegue ricalcolando la matrice V sostituendo al ν_r il nuovo vettore ν_{new} , si calcola la pseudo-inversa ottenendo la nuova V_{piu} . A questo punto si riesegue l'equazione sopra-illustrata utilizzando come valore per la ricerca ν_{new} (8). A questo punto abbiamo il nuovo valore di similarità e possiamo proseguire stilando la nuova classifica. Per ottenere un risultato ottimale è fondamentale un continuo ricalibramento del modello attraverso lo user-feedback.

I valori di similarità sopra ottenuti vengono poi normalizzati per poterli classificare facilmente utilizzando la seguente equazione:

$$SN^\delta(\nu_k, q) = \left\{ \frac{S^\delta(\nu_k, q)}{\sum_{k=1}^p S^\delta(\nu_k, q)} \right\}$$

dove più in generale al numeratore abbiamo il singolo valore ottenuto nella query con ν_k e al denominatore abbiamo la somma di tutti valori ottenuti confrontando con tutte le visual keywords.

Andando a riordinare i valori normalizzati in modo decrescente otteniamo la classifica delle immagini più simili a quella scelta per la ricerca.



Risultato. L'articolo racconta letteralmente i passi, e gli elementi necessari, per arrivare a sviluppare una misura di similarità per il riconoscimento delle immagini, sfruttando il concetto di rete neurale sinergica. La tecnica ottenuta, oltre a fornire buone prestazioni di recupero, offre una metodologia di filtraggio delle caratteristiche ritenute irrilevanti e del "rumore". In aggiunta il sistema messo a punto è in grado di regolare in maniera autonoma l'insieme delle caratteristiche discriminanti grazie a meccanismi di recupero auto-attentivo e feedback di pertinenza. Bisogna però dire che per il corretto funzionamento dell'algoritmo implementato bisognerebbe adottare determinate accortezze:





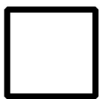

1. Il sistema è fortemente sensibile allo stile delle immagini, dove per stile si intendono ad esempio tipo e spessore del bordo, tonalità della figura e dimensioni dei soggetti, dunque per un funzionamento ottimale le immagini nel database dovrebbero avere tutte queste caratteristiche alla base simili tra loro
2. Ad ogni esecuzione il sistema dovrebbe poter salvare e riutilizzare lo stesso modello, in modo da non perdere tutti i progressi e le informazioni apprese tramite feedback dell'utente, così da poter "allenarsi" e diventare più accurato mano a mano che si utilizza il software, infatti al momento il sistema ogni volta che lo si rimanda in esecuzione perde tutto ciò che ha appreso in precedenza ed è costretto ad imparare da 0, non permettendo così di sfruttare il suo reale potenziale
3. Il database di immagini su cui lavora il programma dovrebbe avere dimensione molto maggiore di quello da noi usato, cosicché il sistema possa allenarsi su un panorama che offre una varietà maggiore di immagini così da renderlo più sensibile alle "piccole differenze"

Per quanto invece riguarda le conclusioni in un'ottica più personale, possiamo dire che non abbiamo riscontrato difficoltà notevoli nella fase di comprensione dell'articolo e nell'individuazione dei suoi punti cardine, che hanno permesso poi di implementare l'algoritmo in MATLAB. D'altronde quest'ultimo è un ambiente di sviluppo che si presta bene al lavoro con grandezze vettoriali e matriciali, mettendo a

disposizione numerose funzioni e costrutti, che rendono l'implementazione di tali procedure piuttosto agevole. Abbiamo infine testato l'efficienza del programma grazie ai criteri di valutazione introdotti precedentemente nella trattazione, la tabella contenente i risultati è riportata nella pagina di seguito.

La tabella è stata creata utilizzando la ricerca self-att con la selezione di 5 immagini.

Tabella 0.1: Tabella valutazione efficienza

Immagine	Rn	Pn	Ln
	0,8156	0,6898	0,6818
	0,8176	0,7052	0,6837
	0,8176	0,7052	0,6838
	0,8194	0,7179	0,6856
	0,8206	0,7254	0,6869
	0,8192	0,7163	0,6853