

Comunicazioni Multimediali

Report di fine Progetto

A.A. 2020/2021

Mario Sorrentino, Luca Campana

Edoardo Maines, Alan Cesaro

19 Febbraio 2021



UNIVERSITÀ DEGLI STUDI DI TRENTO

Sommario

SOMMARIO.....	2
1. INTRODUZIONE.....	3
1.1 ABSTRACT	3
1.2 PREMESSA.....	3
1.3 OBIETTIVO.....	4
1.4 FASI	4
2. LINGUAGGI E LIBRERIE	5
3. PROGETTO	6
3.1 GESTIONE	6
3.2 APPLICAZIONE.....	6
3.3 DATASET.....	7
4. ALGORITMI	8
4.1 IMPLEMENTAZIONE ED APPLICAZIONE	8
4.2 CONFRONTO TRAMITE COLORI.....	8
4.2.1 <i>Iistogramma.py</i>	8
4.2.2 <i>Estrazionefeaturesisto.py</i>	9
4.2.3 <i>Comparatorelsto.py</i>	9
4.2.4 <i>Applicazione.py</i>	9
4.3 CONFRONTO TRAMITE CONTORNI	10
4.3.1 <i>Sift.py</i>	10
4.3.2 <i>EstrazioneSift.py</i>	10
4.3.3 <i>ComparatoreSift.py</i>	11
4.3.4 <i>Mixer.py</i>	11
4.4 CONSIDERAZIONI	12
5. INTERFACCIA.....	13
5.1 INTEGRAZIONE E TESTING	14
5.1.1 <i>Variazioni apportate dopo testing</i>	14
6. CONCLUSIONI	16
6.1 PROBLEMATICA RISCONTRATE	16
6.2 LIMITI SOFTWARE	18
6.3 USABILITÀ.....	19
6.4 RESOCONTI FINALE	22

1. Introduzione

1.1 Abstract

Il crescente utilizzo di apparecchi elettronici ha portato ingenti cambiamenti in diversi aspetti della nostra vita quotidiana.

Quelli che oggi chiamiamo contenuti multimediali sono la conseguenza dell'evoluzione della tecnologia, che ha stravolto il modo con cui ci interfacciamo ad attività un tempo più laboriose.

Se anni fa era consuetudine stampare fotografie e racchiuderle in album fotografici, ora quasi tutti i contenuti multimediali vengono memorizzati e archiviati su cloud o dischi rigidi.

Il continuo aggiornamento delle raccolte digitali ha reso necessario l'implementazione di strumenti che permettano di catalogare, ordinare ed estrapolare i contenuti ricercati dall'utente.

Da questa necessità, nasce l'idea di realizzare un software locale in grado di ricercare e catalogare, in modo veloce ed efficace, moli di immagini presenti in una libreria.

1.2 Premessa

Il sistema da noi realizzato si pone come uno strumento di facile utilizzo, adatto per tutti coloro i quali hanno esigenza di gestire e ricercare file immagini all'interno di datasets, non implicando una conoscenza di base in materia di programmazione.

1.3 Obiettivo

Dalle necessità precedentemente riportate, nasce l'idea di implementare una libreria di immagini e di sviluppare algoritmi di ricerca, capaci di restituire le foto più simili a quella di un'immagine campione, scelta dall'utente.

Questo è stato implementato andando a capire quali funzionalità l'utente richiede e desidera da un prodotto di questo genere, con un occhio vigile alla correttezza di risultato.

1.4 Fasi

Il progetto si struttura principalmente in 9 fasi, di seguito anche la ripartizione dei compiti per attività svolta:

- Definizione dei requisiti e del linguaggio di programmazione (tutti i membri)
- Mock-up (tutti i membri)
- Reperimento e adattamento del dataset (Alan)
- Apprendimento (tutti i membri)
- Sviluppo software degli algoritmi (Luca ed Alan)
- Sviluppo software della GUI (Edoardo e Mario)
- Integrazione (Luca ed Edoardo)
- Test case e valutazioni performance (Alan)
- Stesura del report (Mario)

2. Linguaggi e librerie

Il sistema è stato sviluppato in Python per la semplicità di sintassi e la sua varietà di librerie, in particolare è stato utilizzato OpenCV.

La scelta del linguaggio di programmazione ci ha permesso attraverso l'utilizzo delle librerie Qt5, di implementare una UX che traduca le nostre aspettative.

Il fatto che i due moduli siano stati implementati con lo stesso linguaggio ha reso inutile l'ausilio di un ponte tra i linguaggi.

Di seguito elenchiamo le librerie utilizzate:

- Numpy
- Cv2
- Imutils
- Argparse
- Glob
- Json
- Csv
- Pandas
- Scipy
- Re
- Qt5

3. Progetto

3.1 Gestione

Nella parte iniziale del progetto è stato utilizzato Google Colab, per permettere a tutti i membri un facile accesso alle modifiche e al monitoraggio dello stato di avanzamento del lavoro.

In una fase successiva, abbiamo utilizzato Visual Studio Code e Zoom.

Per la parte di definizione e gestione delle scadenze abbiamo usato Tableau, software che tramite l'appoggio su un foglio di calcolo Excel, ha reso possibile la creazione e personalizzazione del Gantt diagram.

3.2 Applicazione

La nostra applicazione riceve un'immagine in input e successivamente ritorna un numero n (a discrezione dell'utente tra 1 e 20) di immagini simili ordinate sulla base di uno score di somiglianza.

Per riuscire ad avere una ricerca completa e performante abbiamo deciso di implementare due algoritmi, il cui utilizzo è complementare:

- Ricerca per colore → algoritmo per istogrammi
- Ricerca per forma → algoritmo Sift

Inoltre abbiamo deciso di fornire due tipologie alternative di ricerca:

- Tramite immagini in input
- Tramite labels

Questa nostra seconda scelta consente all'utente di avere delle ricerche predefinite e sviluppate in termini di performance: l'immagine associata alla label è stata scelta dopo numerosi test e inoltre viene scelto a priori l'algoritmo più adatto a quella categoria. Tutte le alternative presentate possono essere scelte a discrezionalità dell'utente.

3.3 Dataset

In linea con il nostro obiettivo di creare un programma funzionale per la ricerca di immagini, il nostro dataset è stato compost principalmente da un portfolio di un fotografo professionista di nostra conoscenza.

Le immagini sono all'incirca 200, selezionate accuratamente in base alle caratteristiche funzionali del programma; sono state selezionate più immagini per e nello stesso contesto.

Le immagini scattate avevano un peso elevato (nell'ordine dei 10MB per scatto) e dunque è stato necessario svolgere un lavoro di ridimensionamento uno ad uno per evitare calcoli computazionali dispendiosi che avrebbero generato matrici di dimensioni troppo grandi e complesse da gestire.

I risultati ottenuti sono stati un contenimento del range di peso che va da 500KB a 3MB. Tutte le immagini sono in formato .jpg.

Di seguito elenchiamo le labels da noi scelte:

- Tramonti
- Edifici
- Calcio
- Mare

4. Algoritmi

4.1 Implementazione ed applicazione

In questo paragrafo spiegheremo la parte di background dell'applicazione: abbiamo deciso di suddividere questa parte in 2 programmi separati ed indipendenti, salvo poi, come vedremo in seguito, unire le 2 parti per avere delle maggiori performance nella ricerca per contorni

Entrambi i programmi sono divisi in 2 macrosegmenti concettuali:

- Il primo estrae le features dal dataset e le alloca in un file specifico.
- Il secondo compara le features precedentemente estratte con la query.

Ora vedremo più in dettaglio ogni classe scelta [4.2. 4.3]

4.2 Confronto tramite colori

4.2.1 Istogramma.py

In questa parte del programma viene implementata la classe Istogramma che si compone di 3 funzioni:

- La funzione di inizializzazione;
- La funzione “histogram”: funzione che calcola l’istogramma 3D di una maschera data in input, in seguito l’output viene appiattito per facilitarne la scrittura su file e l’estrazione;
- La funzione “describe”: in questa funzione l’immagine data in input viene convertita in formato “HSV” e viene suddivisa in 4 maschere, ad ognuna delle quali viene applicata la funzione histogram.

I risultati vengono salvati in un array features.

4.2.2 Estrazionefeaturesisto.py

In questa parte del programma viene applicata la funzione “describe” a tutte le immagini del dataset scelto. I risultati vengono convertiti in stringa e trascritti in un file di tipo csv. Questa nostra scelta è data dalla precedente esperienza nel corso di Programmazione 2, in cui abbiamo brevemente trattato la scrittura su file di questo tipo.

Il Dataset e l’index file vengono dati in input grazie ad un Parser.

4.2.3 ComparatoreIsto.py

In questa parte del programma viene implementata la classe Comparatore che si compone di 3 funzioni:

- La funzione di inizializzazione;
- La funzione “chi2_distance”: funzione pensata proprio per confrontare istogrammi di colore;
- La funzione “compara”: che prende in input la features dell’immagine query. Questa funzione apre l’index file in lettura, e converte ogni riga (limitata dal carattere “,”) da stringa a float, in modo che le features possano essere confrontate con quelle della query. Le features verranno poi confrontate proprio tramite la funzione “chi2_distance” e i risultati verranno salvati nell’array associativo (tipo dictionary) “results”.

4.2.4 Applicazione.py

In questa parte del programma avviene il confronto vero e proprio tra le features e avviene la stampa dei risultati.

L’immagine query viene descritta tramite la funzione “describe” della classe Istogramma e le features ricavate vengono comparate tramite la funzione “compara” della classe “ComparatoreIsto”.

I risultati vengono riordinati, limitati e in seguito stampati a video.

4.3 Confronto tramite contorni

4.3.1 Sift.py

In questa parte del programma viene implementato l'algoritmo Sift. Questa nostra scelta è data in primis dalla semplicità di utilizzo di questo algoritmo e dalle maggiori prestazioni, almeno sulla carta, rispetto all'algoritmo orb. L'algoritmo è stato poi modificato rispetto alla forma originale per adattarlo alle nostre esigenze.

Si compone di una sola funzione:

- La funzione “describe”: prende in input un’immagine e la descrive in modo standard tramite l’algoritmo sift.

I descrittori, in seguito, sono stati “appiattiti” in un’unica grande matrice (128*20 valori), il perché lo vedremo in seguito nella funzione “ComparatoreSift.py”.

Successivamente abbiamo aggiunto degli zeri alla matrice, in quanto Sift, in caso i keypoints abbiano uguale importanza, ritorna entrambi.

Nel nostro dataset un’immagine ritornava addirittura 22 keypoints, quindi 128*22 valori; per questo abbiamo reso la dimensione massima pari a 128*23 valori.

Nel confronto è importante che ogni matrice abbia la stessa dimensione.

4.3.2 EstrazioneSift.py

Questa parte del programma è concettualmente uguale a quella precedentemente spiegata nel confronto tramite colore. Abbiamo dovuto però cambiare tipologia di index file per salvare le nostre features: il file di tipo csv non ci permetteva di ripristinare matrici nel processo di confronto, quindi abbiamo utilizzato file di tipo json.

Oltre alle differenze di scrittura dei 2 files, nell’EstrazioneSift.py viene applicato un resize all’immagine prima dell’estrazione delle features.

Questo perché il metodo di confronto che vedremo in seguito si basa sulla distanza euclidea, quindi immagini di differenti dimensioni ma uguale contenuto, avrebbero dato risultati assai diversi.

4.3.3 ComparatoreSift.py

In questa parte del programma viene implementata la classe ComparatoreSift che si compone di 3 funzioni:

- La funzione di inizializzazione;
- La funzione “euclidea”, che prende in input le features della query e le features dell’immagine da confrontare, ritornando in output la distanza euclidea delle nostre due matrici. Di norma con l’algoritmo sift si usano altri metodi di match, il più diffuso è sicuramente il brutal force matcher, che prende un singolo descrittore e lo confronta con tutti gli altri descrittori della seconda immagine. Questo però, oltre a rendere la scrittura su file assai più laboriosa (sarebbe servito un index file per ogni immagine, complicato da gestire anche nella parte di interfaccia) e la ricerca estremamente lenta, a volte superando i 10 secondi con il dataset usato come test (230 immagini, quindi molto piccolo).

Questo dato, unito alle performance non proprio ottimali dell’algoritmo stesso ci ha fatto virare sulla soluzione descritta in precedenza, che durante i testing ha mostrato prestazioni non inferiori al B.F.M e velocità assai superiore;

- La funzione “comparaSift” : lavora in modo simile alla 4.2.3, con opportune modifiche per la differente sorgente di estrazione.

4.3.4 Mixer.py

Concettualmente identica all’ “Applicazione.py” della ricerca tramite colori. Qui però è presente un ciclo for in più, in cui le distanze euclidean e le distanze chi-squared con lo stesso imageID sono sommate, con apporto medio del 20% da parte della funzione Istogrammi e 80% da parte della funzione Sift.

In seguito, i risultati totali vengono riordinati e limitati, per poi essere stampati a video.

4.4 Considerazioni

Abbiamo deciso di presentare 2 metodi di ricerca, quello per Istogrammi e la soluzione mista. Non abbiamo optato per una soluzione che lavori esclusivamente con Sift, in quanto le prestazioni non sarebbero state soddisfacenti.

Dal punto di vista della UX poco importano l'algoritmo utilizzato e l'implementazione, poichè l'utente desidera un programma che offra prestazioni adeguate sia in termini di qualità che di velocità.

Le soluzioni che abbiamo presentato soddisfano, a nostro parere, questi requisiti.

5. Interfaccia



Le uniche funzionalità che non abbiamo ancora presentato, sono rappresentate dai seguenti pulsanti presenti nella parte alta dell’interfaccia:

- Pulsante browse per la scelta dell’immagine di input
- Pulsante browse per la scelta degli index file
- Pulsante browse per la scelta del dataset

Attraverso l’utilizzo del pulsante Browse per l’Index file, è possibile selezionare il file.csv, in automatico sarà di default selezionato anche il file.json.

Tramite i menu a tendina, all’utente è lasciata la scelta di selezionare l’algoritmo desiderato o la label.

Premendo il tasto CERCA, un pop-up richiede all’utente il numero di immagini di output che si desiderano visualizzare.

Il numero deve essere compreso tra 1 e 20, numeri più grandi verranno trattati come se l'utente avesse inserito 20, analogamente numeri minori di 1 verranno trattati come se l'utente avesse scelto 1.

Nel caso in cui non venga selezionato alcun algoritmo, una finestra di errore segnalera il problema all'utente.

Le immagini in output vengono visualizzate in una scroll box in cui navigare nei risultati.

Passando con il cursore sulle immagini viene visualizzato la percentuale di somiglianza con l'immagine di input.

Nel caso in cui si esegua il RESET, solo le immagini in input e output verranno ripulite, mentre per quanto riguarda index file e dataset non verranno resettati.

5.1 Integrazione e Testing

Per quanto concerne l'integrazione dei due moduli sviluppati, questa non ha creato problemi grazie alla loro semplicità costruttiva e logica.

Abbiamo svolto due principali tipologie di testing:

- Test in fase di progettazione, in cui verificavamo il corretto funzionamento dei vari aggiornamenti apportati;
- Test finali, utili per scegliere e correggere il dataset.

5.1.1 Variazioni apportate dopo testing

Iistogrammi:

- Inizialmente la nostra scelta era di suddividere l'immagine in 9 maschere, che teoricamente ne avrebbe aumentato la precisione, ma in fase di testing abbiamo riscontrato delle maggiori performance da parte della versione a 4 maschere nel nostro Dataset.

Sift:

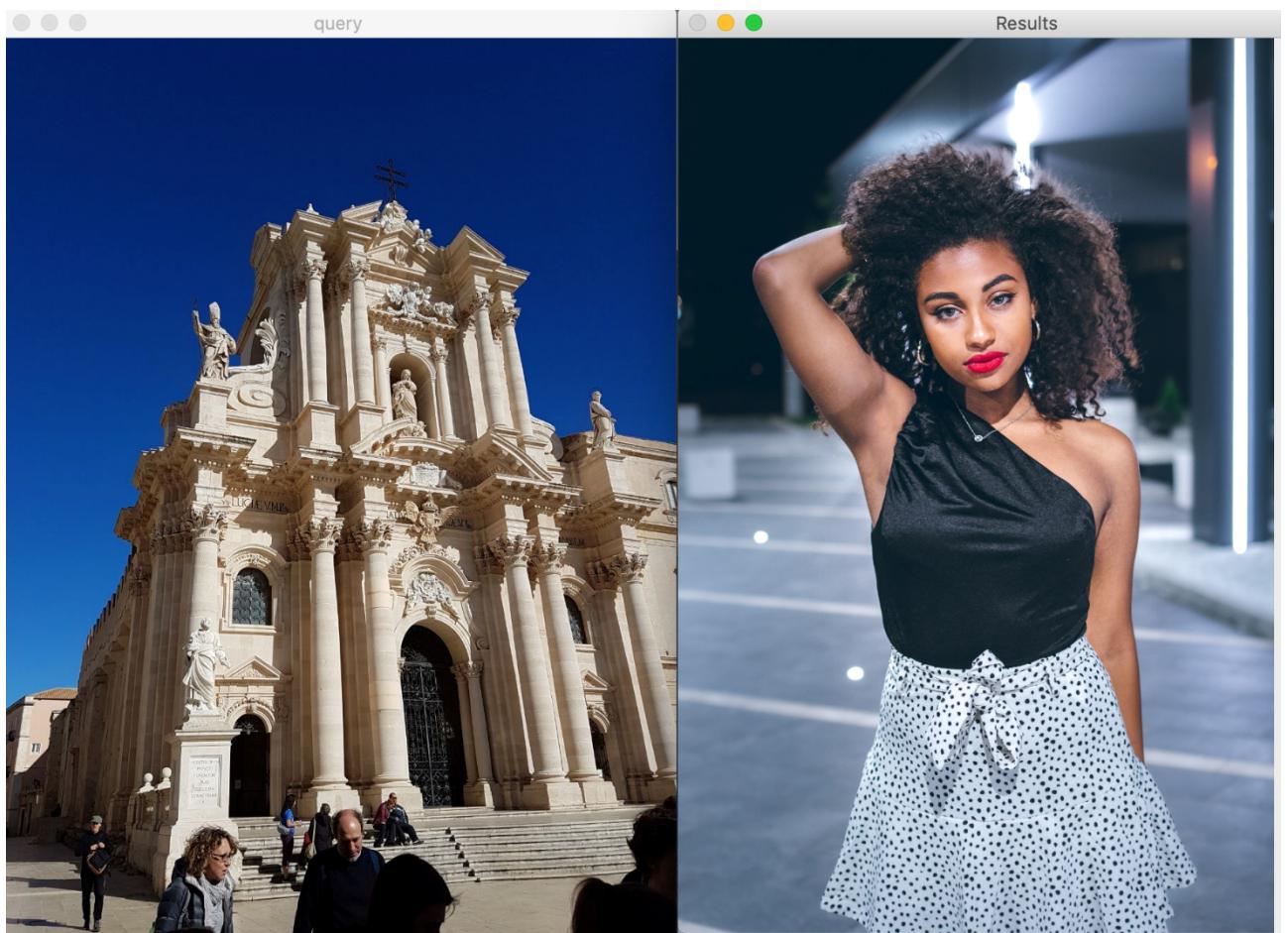
- Abbiamo deciso in fase di testing di prendere i migliori 20 keypoints e i relativi descrittori in quanto si è rivelato un buon compromesso tra velocità e precisione;
- Attraverso dei test empirici, abbiamo posto il contributo dell'algoritmo istogrammi intorno al 20%, in quanto tale valore eliminava i bug delle principali ricerche.

6. Conclusioni

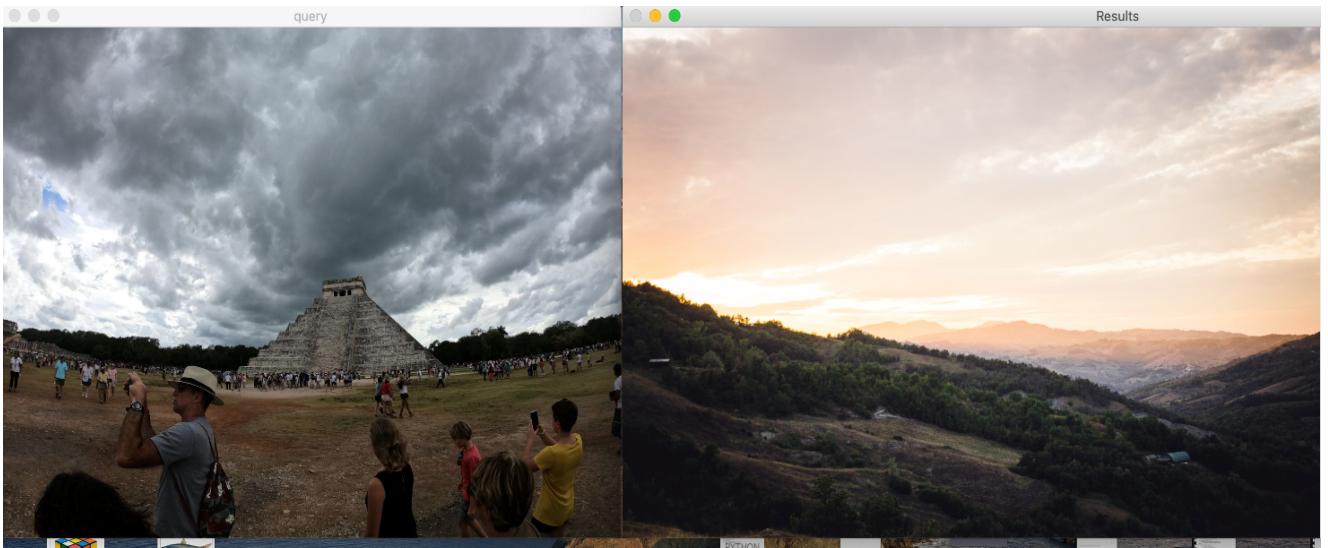
6.1 Problematiche riscontrate

- Bug;
- Necessaria conoscenza elementare dei criteri di selezione del metodo di ricercar;
- Sift (salvataggio keypoints);
- Sift (efficacia reale);
- Visual Studio Code;
- Iniziale difficoltà nell'installazione della libreria xfeatures2d, che ci avrebbe reso possibile implementare funzioni come surf o brief.

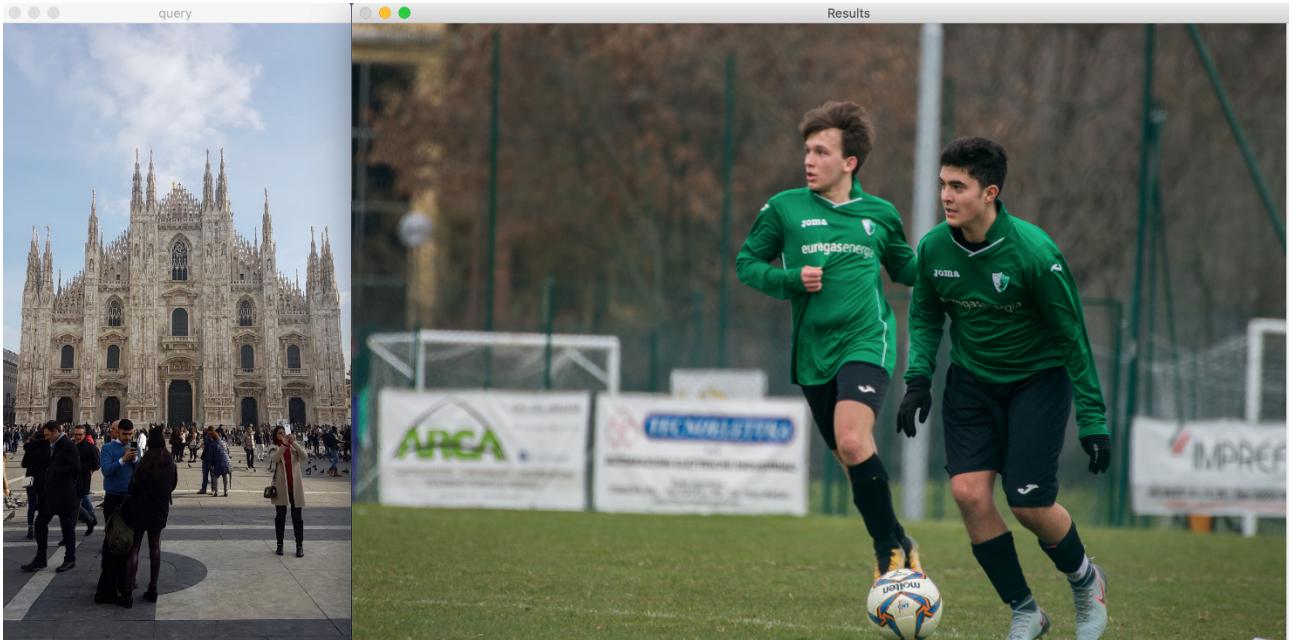
Alcuni risultati dei test effettuati attraverso l'algoritmo sift:



(immagine a destra compariva in output come quarto risultato)



(immagine a destra compariva in output come quarto risultato)



(immagine a destra compariva in output come terzo risultato)

6.2 Limiti software

- Gli algoritmi implementati risultano efficaci con il dataset costituito, tuttavia la scelta di un altro dataset potrebbe porre in evidenza problematiche affrontate durante lo sviluppo del programma.
In caso di cambio del dataset bisogna, rigenerare i file index manualmente tramite riga di comando;
- Il programma permette di visualizzare al massimo 20 immagini in output. Nel caso in cui se ne volessero visualizzare di più la modifica deve essere effettuato nel codice;
- Bisogna installare tutte le librerie essenziali per il funzionamento del programma;
- Tutte le immagini devono essere dello stesso formato/estensione.

6.3 Usabilità

Input Immagine

premere per inserire foto, index file e dataset -->

Scelta tecnica di confronto...

...

CERCA

Immagine Input

Immagini Output

Browse pic Browse index Browse dataset

Labels

Scelta label...

...

CERCA



RESET

Input Immagine

premere per inserire foto, index file e dataset -->

Scelta tecnica di confronto...

...

CERCA

Immagine Input



Immagini Output

Browse pic Browse index Browse dataset

Labels

Scelta label...

...

CERCA



RESET

Input Immagine

Scelta tecnica di confronto...

Confronto per istogrammi
Tecnica Sift

CERCA

premere per inserire foto, index file e dataset -->

Browse pic Browse index Browse dataset

Immagine Input

Immagini Output

Labels

Scelta label...

...

CERCA

4 IMMAGINI
1 GRUPPO

RESET

Input Immagine

Scelta tecnica di confronto...

Confronto per istogrammi

CERCA

premere per inserire foto, index file e dataset -->

Browse pic Browse index Browse dataset

Immagine Input

Immagini Output

Labels

Scelta label...

...

CERCA

4 IMMAGINI
1 GRUPPO

RESET

Input Immagine

premere per inserire foto, index file e dataset -->

Immagine Input

Immagini Output

Browse pic Browse index Browse dataset

Scelta tecnica di confronto...

Confronto per istogrammi

CERCA




Labels

Scelta label...

CERCA

Edifici
Calcio
Tramonto
Mare

RESET

**4 IMMAGINI
1 GRUPPO**

Input Immagine

premere per inserire foto, index file e dataset -->

Immagine Input

Immagini Output

Browse pic Browse index Browse dataset

Scelta tecnica di confronto...

...

CERCA




Labels

Scelta label...

CERCA

Tramonto

RESET

**4 IMMAGINI
1 GRUPPO**

6.4 Resoconto finale

Il gruppo di lavoro è stato in grado di completare tutti gli obiettivi inizialmente fissati.

Tuttavia, il raggiungimento completo e ottimale di questi, ha comportato un leggero ritardo come evidenziato dalle modifiche apportate alla tabella dei tempi (Gantt).

Nonostante questo ritardo, il gruppo è stato in grado di coordinarsi tra le varie fasi di lavoro e di creare strategie alternative volte a risolvere o aggirare problemi e ostacoli.

La suddivisione dei ruoli e conseguente parallelizzazione fra sviluppo interfaccia e sviluppo algoritmi ha funzionato piuttosto bene.

Caricando un'immagine in input, vengono mostrate in output le immagini del dataset più simili a quella caricata, in numero equivalente a quello scelto dall'utente. Ogni immagine del dataset presenta una percentuale che va ad indicare la similitudine con l'immagine caricata.

Malgrado la situazione sanitaria abbia reso la gestione e realizzazione del progetto più difficile, siamo riusciti attraverso una proficua collaborazione e coordinazione a rispettare i task e presentare un lavoro in linea con le nostre aspettative.