Logo Removal con SIFT

Il progetto è composto da 1. Funzione in Matlab per censurare loghi da immagini in varie condizioni di illuminazione, risoluzione e prospettiva con l’algoritmo SIFT 2. Un server che riceve le immagini via TCP e rimuove i loghi 3. Un Client Android che permette di inviare le immagini da censurare al server.

SIFT è un algoritmo per estrazione e matching di features da immagini che restituisce un sottoinsieme di punti rappresentativi di un’immagine (keyPoints) con delle corrispondenti descrizioni dei punti che possono essere utilizzate per trovare corrispondenze tra immagini diverse, SIFT è invariante a cambiamenti di scala e rotazione grazie a degli step di normalizzazione che avvengono durante l’elaborazione dell’immagine.

SIFT è solo uno dei possibili algoritmi per estrazione di feature e l’ho scelto per la elevata accuratezza in confronto a algoritmi dello stesso genere come SURF o FAST (che però hanno tempi di esecuzione minori)

Passaggi importanti di SIFT e scelte di progetto:

Data J l’immagine originale per ottenere la scale-invariance nelle feature si crea lo scale space ovvero si scala J a N risoluzioni diverse e da ogni versione scalata si ottiene un set, chiamato ottava, applicando gaussian blur con diversi σ. Alla fine avremo N ottave ognuna con M immagini. Poi ricaviamo la DOG (Difference of Gaussians): per ogni ottava sottraiamo dalle foto con poco blur quelle con più blur. Per ottenere i keyPoint cerchiamo gli estremi locali nella DOG: è considerato estremo locale un pixel maggiore (o minore) dei pixel adiacenti nella stessa immagine e maggiore (o minore) dei pixel adiacenti nell’ottava precedente e successiva.

Dal set di keyPoints così ottenuto si rimuovono i punti a basso contrasto (con un threshold=0.03) e i punti lungo bordi di oggetti (per eliminare eventuali punti di sella che non sono estremi locali) che non danno buone feature. In Matlab questi passaggi vengono effettuati da detectSIFTFeatures che trova quali punti usare per estrarre le feature. Per la descrizione del keyPoint si usa extractFeatures che calcola il gradiente approssimato in quadrati 4x4 in una regione 16x16 attorno al pixel: si ottiene un vettore di 128 elementi a cui si sottrae il gradiente calcolato nel keyPoint stesso in modo da rendere SIFT invariante alle rotazioni.

Queste operazioni precedenti vengono effettuate sia sul logo che sulla foto da censurare, per poter trovare il logo bisogna fare un matching tra le feature del logo e della foto. Ogni keyPoint ha un vettore di feature in e si usa la distanza euclidea per determinare quali keyPoint della foto sono simili a quelli del logo. In Matlab

Le feature per ogni keyPoint sono dei vettori in R<sup>128</sup> quindi è possibile utilizzare la distanza euclidea per determinare per ogni keyPoint della foto a quale keyPoint del logo è più vicino in termini di feature. La funzione corrispondente in Matlab è `matchFeatures(featureLogo,featureFoto)`.

Un parametro importantissimo in questa funzione è `MatchThreshold`: se la distanza tra le feature è > di `MatchThreshold`il match viene scartato, ciò permette di gestire la quantità di falsi positivi/falsi negativi.

Sperimentalmente ho trovato che per ottenere il minor numero di falsi negativi il miglior valore di questo parametro è 10 o 11. Questo matching si vede nella figure 3 di `mainSIFT.m`.

Infine si rimuovono i match outlier: si stima il tipo di trasformazione avvenuto al logo, scegliendo tra lineare, affine o omografia tramite `estgeotform2d(matchLogo,matchFoto,tipoTrasformazione)`.

I match che non rispettano la trasformazione determinata vengono scartati e rimangono solo i match inlier figure 4 su Matlab.

### ROI e censura del logo

Per poter cancellare il logo determiniamo una zona della foto che verrà censurata.

Nel punto precedente abbiamo determinato la trasformazione che porta dal logo alla foto quindi prendiamo come controimmagini i 4 vertici del logo e applicando la trasformazione a questi 4 punti otterremo i vertici di un poligono nella foto (l'immagine attraverso la trasformazione dei 4 punti nel dominio).

Questo si può fare su Matlab con `transformPointsForward(trasformazione,poligonoLogo)`,dal poligono ottenuto possiamo ricavare una region of interest sulla figura con `drawpolygon` e trasformarla in una maschera con `createMask(roi)` poi applicheremo un filtro interpolatore sulla foto con questa maschera con `regionfill(immagine,mask)` che andrà a censurare il logo nella foto. Il filtro interpolatore tende a funzionare meglio in zone di colore uniforme e senza riflessi (esempio logo hollister) ma in ogni caso riesce a nascondere il logo.

| ![logo in zona con riflessi](immaginiREADME/hollister.jpg) |

| :--: |

|Il logo viene eliminato senza lasciare segni sulla maglia|

| ![logo in zona con riflessi](immaginiREADME/s3Removal.jpg) |

| :--: |

|Il logo viene eliminato ma si nota la censura a causa dei riflessi della scatola|

\_La funzione `SIFT\_frame\_funzione` esegue tutti i passaggi di `mainSIFT.m` senza mostrare le figure in modo da poterla usare nella demo della `censuraVideo` e nel server.\_

## App android e Server Matlab

### Client Android

L'app android è scritta in Kotlin e utilizza il toolkit `Jetpack Compose` per il rendering della UI e funzioni asincrone e socket per l'invio delle immagini via TCP.

| ![applicazione](immaginiREADME/layoutApp.jpg) |

| :--: |

|Layout dell'applicazione|

L'applicazione è composta da una preview della foto da inviare, un pulsante per mandare l'immagine al server e un pulsante che apre un filepicker della galleria per scegliere la foto.

Dopo aver scelto la foto dal filepicker l'app android ottiene l'URI corrispondente alla foto e legge il file in un oggetto `Bitmap foto` tramite `BitmapFactory.decodeStream()` quando `foto` viene cambiata JetPack Compose osserva come lo stato `mutableState` della variabile sia cambiato e quindi forza un recompose della UI disegnando a schermo la immagine appena letta.

Quando viene premuto il pulsante "Invia messaggio" faccio partire una coroutine asincrona che gestirà il socket TCP per l'invio della foto questa coroutine è assegnata ai thread riservati all'IO (tramite `Dispatcher.IO`) così non bloccherà la normale esecuzione dell'app.

Per inviare la foto comprimo la `Bitmap foto` in formato PNG tramite `foto.compress(Bitmap.CompressFormat.PNG, 100, stream)` .

Kotlin è completamente interoperabile con Java e quindi per l'invio del file ho usato i Socket da `java.net.Socket`

Per l'invio dell'immagine ho scelto un protocollo molto semplice: invio tutta la foto in un unico `write` seguita da 32 byte di `NULL` (32 byte a 0) come ending sequence per avvisare il server che la trasmissione dell'immagine è terminata e può procedere con la censura.

Il motivo per cui ho scelto questa ending sequence è che viene ignorata direttamente quando scrivo il file PNG in quanto è composta da `NULL`.

### Server MATLAB

Il server di Matlab ascolta sulla porta 4316 TCP e una volta che il client si connette comincia a ricevere i byte e a concatenarli nel vettore `dataread` in modo da poter ricomporre l'immagine.

Il vettore `dataread` è di tipo `uint8` corrispondente al tipo `byte` di Kotlin e Java.

Dopo ogni chunk di byte letti il server controlla se il chunk termina con 32 byte di zeri tramite la funzione `checkFineStream`, quando arriva questa ending sequence smette di leggere byte e comincia l'elaborazione dell'immagine.

Da Matlab posso utilizzare degli oggetti Java per scrivere l'immagine ricevuta su un file `outputFile.png`:

```

outputFile=javaObject("java.io.File","outputFile.png");

outputStream=javaObject("java.io.FileOutputStream",outputFile);

javaMethod("write",outputStream,dataread)

javaMethod("close",outputStream)

```

Dopo aver salvato l'immagine posso procedere con l'elaborazione tramite SIFT come spiegato nel paragrafo precedente per ottenere l'immagine censurata che viene scritta in `censurataOutput.png`.

---

Link e reference utili usati durante il progetto:

\* [[2]Introduction to SIFT( Scale Invariant Feature Transform)](https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40).

\* [[3]Computer vision Toolbox Mathworks](https://www.mathworks.com/products/computer-vision.html)

\* [[4]Android Studio Documentation](https://developer.android.com/docs)