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 ad 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 precedenti vengono effettuati da detectSIFTFeatures che trova quali pixel saranno i keyPoint da 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 R^{128} e si usa la distanza euclidea per determinare quali keyPoint della foto sono simili a quelli del logo. Per il matching si usa matchFeatures, che trova per ogni keyPoint della foto quello più simile del logo; questa funzione ha un parametro 0<MatchThreshold<100 che indica la distanza massima tra le descrizioni per cui ci sia un match. Aumentando MatchThreshold si otterranno più match veri ma anche più match falsi.

Infine, si stima una $g: R^2 \rightarrow R^2$ con estgeotform2d che determina una trasformazione che ha come dominio i keyPoint del logo e come immagine quelli della foto. Questa funzione prende come parametro una stringa che può specificare il tipo di trasformazione si può scegliere tra lineare, affine, omografia e trasformazione similare a fine del report si trova una tabella che compara vari valori dei parametri MatchThreshold e Transformation type.

I match che non rispettano la trasformazione trovata ovvero i keyPoint della foto che non sono immagine di nessuno dei keyPoint del logo vengono scartati perché considerati outlier.

Per poter censurare il logo determiniamo un poligono che lo copra. Questo è possibile trovando l'immagine attraverso g dei 4 vertici del logo, ci darà 4 punti da cui possiamo costruire un poligono che conterrà il logo ($g(vertice)$ si trova con transformPointsForward). Poi si crea una maschera dal poligono e si applica una interpolazione alla foto attraverso la maschera (con regionfill). Il filtro interpolatore è computazionalmente più pesante di settare i pixel a 0 ma l'ho scelto perché permette di fare una censura più uniforme con il resto della foto.

Client Android in Kotlin e Server Matlab

Il client Android permette di scegliere una foto dal filepicker e inviarla al server in modo che questo ne censuri il logo. Ho scelto di utilizzare socket TCP per la comunicazione per l'affidabilità e la possibilità di stabilire una connessione rispetto a UDP, inoltre trasferendo solo una immagine il delay non è un problema significativo. Ho scritto la app in modo che gestisca il socket attraverso una coroutine asincrona (utilizzando i thread dedicati al IO grazie al Dispacther.IO di Kotlin) in modo che il trasferimento dell'immagine sia non bloccante. Una volta aperto il

filepicker la app ottiene l'URI della foto scelta e la legge in un oggetto Bitmap che poi viene compresso in PNG per il trasferimento. Appena viene letta l'immagine viene mostrata a schermo una preview.

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" come array di byte (uint8 in Matlab) 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`.

Il server 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. Dopo ogni chunk di byte letti il server controlla se esso termina con la ending sequence tramite checkFineStream, quando arriva questa ending sequence smette di leggere byte e comincia l'elaborazione dell'immagine come spiegato nella prima parte del report.

Osservazioni e conclusioni:

È presente un tradeoff tra falsi positivi e falsi negativi in base al valore di MatchThreshold: se si alza troppo il valore allora l'algoritmo trova match spuri in tutta l'immagine censurando pezzi della foto casuali e quindi aumentando i falsi positivi. Vice versa se si abbassa troppo MatchThreshold estgeotform2d non ha abbastanza match per poter determinare la trasformazione g e quindi non riesce a censurare il logo risultando in un maggior numero di falsi negativi. Di default Matlab utilizza MatchThreshold=1 ma siccome il nostro programma deve censurare loghi è preferibile avere falsi positivi che falsi negativi quindi ho scelto di utilizzare valori di MatchThreshold uguali a 5 o 10. Questa scelta è anche giustificata dalla seguente analisi:

MatchThreshold	tipo Trasf.	N. Errori	% errori
1	projective	443	80,25
10	projective	375	67,93
15	projective	382	69,20
20	projective	376	68,12
1	affine	217	39,31
10	affine	61	11,05
15	affine	63	11,41
20	affine	61	11,05

MatchThreshold	tipo Trasf.	N. Errori	% errori
1	projective	264	73,74
5	projective	128	35,75
10	projective	114	31,84
20	projective	121	33,80
1	affine	220	61,45
5	affine	65	18,16
10	affine	38	10,61
20	affine	40	11,17

Queste tabelle mostrano il numero di frame dove la censura non è riuscita rispetto ai valori dei parametri nei due video pocketVideo.mp4 e cameo.mp4. Quindi si vede come apparentemente aumentando MatchThreshold migliora il numero di frame censurati però si nota sperimentalmente che per valori > 10 vengono censurati molti frame dove non è presente il logo (vedi video "cameoM10err38affine.avi") quindi per un buon tradeoff ho scelto 5 o 10 come valori del parametro. Le migliori scelte sul tipo di trasformazione sono affine o omografia (projective), omografia è più robusta rispetto a cambi di prospettiva mentre affine è più precisa se la prospettiva rimane uguale.

Il progetto funziona bene se deve censurare immagini singole o video salvati in memoria (con la funzione censuraVideo) ma alcune scelte di progetto impediscono la elaborazione in tempo reale di video in quanto rallentano troppo la computazione, alcune modifiche che potrebbero portare a una velocità di esecuzione migliore potrebbero essere: utilizzare SURF o FAST al posto di SIFT che sono più veloci (sebbene siano meno precisi) e censurare assegnando ai pixel da coprire un valore costante invece di utilizzare l'interpolazione.

Nel server ho utilizzato degli oggetti Java all'interno di Matlab, grazie all'integrazione Java-Matlab, per scrivere l'immagine inviata da Android su un file PNG: questo mi ha permesso di non dover effettuare conversioni tra l'array di Byte in arrivo e formati intermedi di Matlab semplificando e velocizzando il server.

In Matlab non è possibile creare un server TCP che accetti richieste da più client quindi una sola app alla volta si può connettere al server. Se una app non riesce a connettersi l'errore viene catturato da un blocco try catch e permette di riprovare a connettersi in seguito. Una possibile soluzione per poter risolvere questo problema sarebbe cambiare linguaggio del server, ad esempio utilizzando Java e usare la integrazione di Matlab-Java per utilizzare SIFT.