Crucipuzzle solver report

# Presentazione generale

Il crucipuzzle solver è un programma per pc in grado di aiutare l’utente a rintracciare parole all’interno di un crucipuzzle.

Il suo funzionamento si basa sull’acquisizione di un’immagine, che può essere fornita tramite file oppure scattata sul momento utilizzando una camera di buona risoluzione collegata al computer. Una volta acquisita l’immagine all’utente verrà chiesto di inserire, da console di comando, una parola da cercare; immediatamente il programma visualizzerà sull’immagine originale la posizione di quella parola. Se l’utente lo desidera può continuare a cercare nuove parole che verranno disegnate progressivamente sulla foto, oppure può scegliere in qualunque momento di pulire la foto dalle ricerche già effettuate e continuare poi con nuove parole.

Una volta terminato, l’utente viene riportato al menù principale dove può scegliere da linea di comando se continuare con un nuovo puzzle oppure se chiudere l’applicazione.

# Descrizione generale del programma

Il risolutore di crucipuzzle si basa su un algoritmo composto dalle seguenti parti:

* Riconoscimento dell’area di interesse delimitata dalla cornice del puzzle   
  La cornice deve essere rettangolare e deve essere abbastanza grande all’interno dell’immagine affinchè sia chiaro che è proprio il puzzle il soggetto principale della foto. Inoltre, non sono supportati puzzle contenenti riquadri con foto all’interno.
* Estrapolazione del puzzle dalla foto.   
  Consiste nel correggere possibili distorsioni di prospettiva e rotazione del puzzle nella foto iniziale. Dopo aver corretto le distorsioni si ritaglia solamente l’area strettamente necessaria per proseguire nel riconoscimento del puzzle.
* Riconoscimento del puzzle

E’ una prima analisi del puzzle che ne riconosce le caratteristiche principali come numero di righe, numero di colonne e dimensioni in pixel dell’area del puzzle. Questa parte sfrutta già funzioni di riconoscimento testuale, ma al solo fine di ricavare informazioni generali.

* Riconoscimento delle singole lettere

Il riconoscimento avviene su piccole aree dell’immagine che contengono una lettera sola, e viene fatto per tutte le lettere una alla volta. Le lettere riconosciute vengono codificate anzichè come matrice contenente lettere, come immagine in scala di grigi contenente un pixel per ogni lettera del puzzle. Questo servirà per il successivo algoritmo di ricerca.

* Ricerca di una parola.   
  La procedura di ricerca sfrutta la codifica del puzzle e della parola in un’immagine in scala di grigio, in modo da eseguire una ricerca di un’immagine all’interno dell’altra. Per la precisione, la parola da cercare viene trasformata in un vettore di pixel, ne vengono poi fatte tante copie quante sono le possibili orientazioni che essa può assumere all’interno del puzzle, e vengono cercate queste immagini più piccole all’interno della matrice più grande.
* Evidenziazione della posizione di una parola.   
  La visualizzazione della parola trovata avviene tracciando una linea che si sovrappone ad essa sull’immagine precedentemente ritagliata. Più ricerche consecutive si accumuleranno sulla stessa immagine.
* Ricostruzione dell’immagine finale.   
  L’immagine ritagliata e distorta subisce un processo inverso e viene combinata assieme all’originale, applicando al puzzle e alle linee aggiuntive una trasformazione solidale, in modo che nell’immagine finale queste ultime rimangano sovrapposte alla parola corrispondente.

# Componenti del progetto:

Il progetto è gerarchicamente composto da un file principale, CrucipuzzleSolver, che contiene il main, e da altre componenti aggiuntive (chiamate libraries) dove abbiamo raggruppato e documentato le funzioni necessarie per la corretta esecuzione dell’algoritmo:

* plotLibrary
* frameLibrary
* ocrLibrary
* wordLibrary

## Crucipuzzle solver

Questo è il file che contiene il main. I menù del programma vengono gestiti da una semplice stampa su console e da una richiesta di input tramite str(input).

Le due principali funzioni di partenza sono l’acquisizione di immagine tramite camera e tramite caricamento da file.

L’acquisizione tramite camera fa partire una cattura video, con apposito flag per farla comparire in primo piano, e continua finché l’utente non preme il tasto P, scelto come otturatore. Ciò scaturisce l’acquisizione sotto forma di immagine dell’ultimo frame visualizzato, che verrà utilizzato dalle successive componenti dell’algoritmo.

Il secondo metodo di acquisizione è tramite un dialog di scelta del file. Questo viene aperto grazie alla libreria tkinter, dalla quale importiamo la classe filedialog e filtriamo i file visualizzati e selezionabili in base alle estensioni riconducibili a immagini. In caso di problemi nell’acquisizione dell’immagine lanciamo un’eccezione di tipo file not found che viene gestita dal main.

Il main si occupa successivamente di una prima fase di elaborazione dell’immagine appena acquisita per l’estrapolazione dei dati di cui necessitiamo, e poi della ricerca delle parole richieste dell’utente, fino a che egli non decide di interrompere la ricerca per passare ad un altro puzzle o terminare l’esecuzione del programma.

## Plot library

Questo modulo si occupa principalmente di visualizzare le immagini in parallelo all’esecuzione del programma, e dell’evidenziazione della posizione delle parole trovate nel puzzle.

La classe *steadyImage* inizializza un nuovo thread grazie al modulo *threading*, e mostra un’immagine mentre l’esecuzione del programma prosegue. L’immagine mostrata è quella del puzzle e viene progressivamente aggiornata per mostrare le nuove parole trovate, oppure ripristinato all’immagine originale, cambiando l’immagine associata al thread.

Al momento di interrompere la visualizzazione si interrompe il loop di visualizzazione e si fa una join del thread.

La funzione di *drawLine* serve invece a rappresentare sulla ROI ritagliata del puzzle una parola appena trovata. Questo avviene disegnando una linea gialla che parte dalla prima lettera della parola fino all’ultima.

## Frame Library

Questo modulo si occupa di riconoscere la cornice del puzzle, correggerne le distorsioni e ritagliarne la ROI per il successivo riconoscimento. Esegue poi il procedimento inverso che da una ROI che contiene le linee colorate delle parole trovate lo riporta alle proporzioni originali ed aggiunge la parte di immagine precedentemente scartata.

Il riconoscimento della ROI avviene individuando una cornice rettangolare all’interno dell’immagine. Inizialmente si lavora sull’immagine con del blur per rimuovere il rumore, adaptive thresholding per avere un’immagine binaria, anche con condizioni di luce non omogenea, e delle aperture e chiusure per rimuovere eventuale rumore e tappare buchi.

(Nota su questa parte di pre-processamento: per arrivare a scegliere i parametri più adatti e le operazioni più adatte da svolgere o da non svolgere, ci siamo scritti un programma che apre un certo campione di immagini e le visualizza una alla volta tramite un flusso video, con possibilità di passare immediatamente da un’immagine ad un’altra, e per modificare i vari parametri in real time abbiamo sfruttato delle trackbars)

Dopo il preprocessamento, il riconoscimento del rettangolo viene eseguito grazie alla funzione di *findContours* dal cui risultato identifichiamo il perimetro di un quadrilatero che abbia l’area maggiore possibile. Se l’area del puzzle è troppo piccola e non supera una soglia minima, l’immagine viene scartata come non valida. A questo punto identifichiamo il quadrilatero appena riconosciuto tramite i suoi 4 vertici.

Per correggere le distorsioni di prospettiva e rotazione dell’immagine sfruttiamo la funzione di warpAffine su tutta l’immagine sfruttando come punti di controllo i 4 vertici prima identificati, e facendo una stima di quale vogliamo che sia la loro posizione a seguito del warp. Dopodichè viene ritagliata la ROI delimitata dalla nuova posizione dei 4 vertici.

La classe processedPicture contiene l’immagine dopo il warp, l’immagine cropped ed i punti di controllo; dati utili per proseguire nell’elaborazione e per eseguire poi la trasformazione inversa.

## OCR Library

Questo modulo è responsabile per il riconoscimento del puzzle vero e proprio, diviso in 2 fasi: riconoscimento delle caratteristiche generali e riconoscimento delle singole lettere.

Per poter ritagliare con precisione le singole lettere una a una per poterle riconoscere abbiamo bisogno di sapere quanto è grande il puzzle. Per scoprirlo partiamo dalla ROI del puzzle identificata dalla frame Library, binarizziamo l’immagine con un adaptive thresholding e poi eseguiamo un riconoscimento di caratteri grazie a *pytesseract*. Pytesseract è uno strumento di riconoscimento ottico di caratteri per python. In questo procedimento scegliamo di eseguire una ricerca sull’intera immagine ottenendo per ogni carattere riconosciuto anche il suo buonding box. Il riconoscimento è limitato alle sole lettere maiuscole dalla “A” alla “Z”.

Una volta ottenuti i bounding box filtriamo via tutti gli outliers con dimensioni troppo grandi o troppo piccole, nell’ottica che tutte le lettere riconosciute dovranno avere un bounding box di dimensioni simili. Le informazioni delle singole lettere sono immagazzinate in una struttura dati di tipo *CharacterWrapper*. A questo punto si sfruttano le lettere rimanenti dopo il filtraggio per costruire 2 matrici raggruppando le lettere per righe o per colonne. Queste struttura sono usate per calcolare le dimensione precise del puzzle ed il numero di colonne. Per il numero di righe abbiamo scelto di eseguire una vera e propria lettura per righe con pytesseract che abbiamo visto darci dei risultati migliori.

Una volta ottenute queste informazioni abbiamo potuto ritagliare le lettere una ad una ed eseguire singolarmente il riconoscimento con pytesseract. Scendendo nel dettaglio, per ogni immagine abbiamo applicato un “adaptive thresholding progressivo”, ovvero partendo da un valore iniziale del parametro di offset da sottrarre alla compitazione del valore di value per ogni pixel abbiamo progressivamente incrementato questo valore quando l’algoritmo di riconoscimento non era in grado di distinguere una sola lettera nell’immagine. Un lavoro a partee è stato fatto per la lettera “I” che risulta più difficile da riconoscere, perciò abbiamo provato a riconoscerla come rettangolo, come fatto similmente in precedenza colla cornice del puzzle.

Se dopo un certo numero di tentativi una lettera non viene riconosciuta viene lanciata un’eccezione e l’intera immagine è considerata non valida.

Per concludere, tutti i caratteri riconosciuti sono salvati in una matrice.

## Word library

Quest’ultimo modulo si occupa della ricerca di una parola nel puzzle e della codifica del puzzle e della parola da cercare, in funzione dell’algoritmo di ricerca.

La codifica delle lettere avviene tramite un dizionario nel quale viene fatta corrispondere ad ogni lettera maiuscola dalla “A” alla “Z” un multiplo di 10 tra 0 e 250. Questo permette di interpretare una singola lettera come un valore di grigio in una rappresentazione a 8 bit, e perciò la matrice di lettere del puzzle può essere vista come un’immagine in scala di grigio dove ogni lettera è un pixel. Similmente può essere fatto colla parola da cercare.

L’algoritmo di ricerca sfrutta la funzione di matchTemplate con machera. Vengono definiti 8 template da cercare all’interno del puzzle: parola in orizzontale, in verticale, in diagonale e in antidiagonale, e poi altri 4 template colla parola scritta al contrario. La presenza delle maschere è resa necessaria dalla ricerca delle parole in diagonale, siccome il template è quadrato, ma noi siamo interessati a trovare solo i valori in una certa posizione del template.

Se la funzione di ricerca da esito positivo si calcola la posizione di inizio e fine della parola, per poterla successivamente evidenziare disegnandoci sopra una riga.

Questo modulo si occupa anche di verificare la validità delle stringhe inserite dall’utente, utilizzando una regular expression grazie all’apposito modulo *re*.