Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

tonicarD: un'applicazione web per l'annotazione semiautomatica delle schede Caronti

Relatore:
Chiar.mo Prof.
Fabio Vitali

Correlatore:
Dott. Francesco Sovrano

Presentata da:
Luca Castagnotto

Sessione III
Anno Accademico 2021-2022

Introduzione

Durante il XIX secolo, il bibliotecario Andrea Caronti produsse a mano più di 150 000 schede per il catalogo della biblioteca dell'Università di Bologna, la quale ha deciso di digitalizzarle per preservarle dai danni del tempo. Data anche la grande quantità di materiale, è stata colta l'occasione per promuovere lavori di ricerca nel campo del riconoscimento del testo scritto a mano (HTR: *handwritten text recognition*) e creare così un sistema per automatizzare il processo di trascrizione di tali schede. È stato quindi realizzato un dataset iniziale composto da circa 330 immagini di schede, di cui più di 200 trascritte a mano, allo scopo di sviluppare un modello di riconoscimento automatico. Ad oggi, sono stati implementati svariati algoritmi basati su Machine Learning (ML) che risolvono con successo il problema dell'identificazione del testo scritto a mano. Come primo tentativo di trascrivere automaticamente le intere schede Caronti, è stato usato un modello open-source da zero (senza fine-tuning) basato su un'architettura transformer, che funziona sui dataset Bentham, IAM, Rimes, Saint Gall e Washington (gli ultimi due sono manoscritti storici) [1]. Considerato il fatto che tali dataset sono composti da linee di testo e non da intere pagine, non stupisce che i risultati ottenuti applicando il modello sulle schede Caronti non segmentate siano stati pessimi: il riconoscimento dei caratteri è di circa il 33% e quello delle parole del 25%. Le trascrizioni così ricavate sono illeggibili ed inutili e la limitata dimensione del dataset non permette di allenare il modello ulteriormente con la speranza di ottenere risultati migliori. Infatti, sebbene il problema del riconoscimento automatico del testo scritto a mano si consideri oramai risolto e superato, anche in riferimento ai documenti storici [2], affinché un programma di questo tipo funzioni, esso necessita di una grande quantità di dati annotati che servono per l'allenamento e la valutazione del modello.

Problema
Si individua la difficoltà di costruire un trascrittore per le schede Caronti nella mancanza di segmentazione del testo (riconoscimento delle regioni della scheda dove ci sono le scritte) e nella scarsa quantità di immagini che compongono il dataset. Risolvere il primo problema aiuterebbe automaticamente a risolvere il secondo, poiché per ogni scheda si creerebbero nuove immagini, ad esempio una in più per ciascuna riga di ciascuna scheda. Si ritiene che il numero di righe di cui deve essere composto un dataset affinché

un modello di riconoscimento di testo possa essere allenato con risultati soddisfacenti sia all'incirca di qualche migliaio. Nel contesto qui discusso, significherebbe aumentare approssimativamente del doppio il numero delle immagini, sulle quali poi si devono identificare le linee di testo e ritagliarle dall'immagine originale, in modo da avere un dataset costituito da foto di porzioni di schede, non più di schede intere. Tale insieme dovrebbe raggiungere una cardinalità di circa 3000 immagini, con la speranza che siano sufficienti per costruire un riconoscitore automatico funzionante. Una soluzione alternativa potrebbe essere invece quella di estendere il dataset con un numero inferiore di immagini ed usare la tecnica di fine-tuning per sfruttare un modello di riconoscimento di testo ben allenato su un dataset molto grande ed applicarlo al dataset Caronti. È stato infatti dimostrato che con questo metodo possono bastare poche centinaia di righe per costruire un riconoscitore valido [3]. Per entrambe queste soluzioni, si manifesta la necessità di segmentare le porzioni di testo nell'immagine, che è un'operazione manuale, o al più semiautomatica. Per questa ragione, è sfavorita la prima delle due opzioni, in quanto generare 3000 nuove annotazioni richiede molto lavoro. Infatti, l'annotazione è l'operazione congiunta di segmentazione e trascrizione e, quando è manuale, è estremamente tediosa e costosa in termini di tempo, tuttavia è un lavoro indispensabile poiché serve alla creazione del *ground-truth* (GT), ossia quell'insieme di dati sui quali il sistema di riconoscimento si allena per poter divenire capace di funzionare su input mai visti prima. Allo scopo di ridurre il carico di lavoro per chiunque debba eseguire le nuove annotazioni delle schede Caronti, si è deciso di propendere per la soluzione che applica il fine-tuning, il che però non libera dalla necessità di segmentare e trascrivere le schede Caronti, che rimane il problema centrale.

Strumenti di annotazione

L'analisi di documenti come quelli antichi è particolarmente difficile perché essi presentano spesso layout complessi, calligrafie originali, elementi ornamentali, ecc. Ciascun dataset di vecchi manoscritti porta quindi con sé caratteristiche peculiari che li rendono unici e li distinguono da qualsiasi altro. Per queste ragioni, chiunque si sia imbattuto nella sfida di implementare un trascrittore di testi antichi, si è sempre prima dovuto preoccupare di sviluppare un sistema di annotazione, il che ha generato negli anni la proliferazione di diversi programmi finalizzati a questo scopo (vedi sezione 1.2). Si sono dunque presi in considerazione tali strumenti per svolgere la segmentazione e trascrizione delle schede Caronti, ma nessuno è stato adottato. Dapprima sono state escluse le applicazioni che, seppur ben note in letteratura, non sono più disponibili perché il progetto è stato abbandonato o perché non sono a libero accesso. Poi si è distinto fra sistemi di annotazione di qualsiasi tipo di immagine e sistemi specializzati per l'annotazione di testo di documenti scritti. Il primo gruppo non è stato considerato adatto perché tali applicazioni non offrono strumenti di riconoscimento automatico del testo e quindi non agevolano l'utente in alcun modo, né nella segmentazione né nella trascrizione

delle parti scritte. Alcuni strumenti del secondo gruppo, invece, sono stati considerati validi, tuttavia condividono lo stesso problema: hanno un'alta curva di apprendimento. Si tratta infatti di applicazioni che non si limitano all'annotazione dei documenti ma includono funzionalità per allenare dei nuovi modelli di riconoscimento automatico del testo, il che li rende molto sofisticati e questo si riflette nella complessità delle loro interfacce. D'altronde, la fase del progetto che si discute in questa tesi non riguarda già la realizzazione del trascrittore automatico, ma si limita alla fase precedente, ossia allo sviluppo di un'interfaccia volta all'annotazione delle schede Caronti, al fine di generare il ground truth.

Algoritmo di segmentazione automatica
Un altro aspetto su cui il progetto si è concentrato è stato lo sviluppo di un algoritmo di segmentazione automatica, capace di individuare nelle immagine le regioni di testo delle schede Caronti e di classificarle in base al tipo di informazione che rappresentano: autore, titolo, note di catalogazione o collocazione. In letteratura esistono sofisticate tecniche di analisi dell'immagine in grado riconoscere la presenza di oggetti o lettere all'interno di una foto, ma la loro implementazione non è semplice, soprattutto se si vogliono realizzare modelli molto performanti. Inizialmente, per questo progetto, era stato sviluppato un algoritmo che faceva uso di tali tecnologie allo scopo di segmentare le singole parole nelle schede Caronti, ma i risultati sono stati scoraggianti. Per risolvere il problema si è scelto allora un altro tipo di approccio: si è fatto uso del servizio Cloud Vision API di Google [4], che allo stesso tempo tempo forniva una soluzione sia per la segmentazione che per la trascrizione. Tale tecnologia ha permesso di fare un passo avanti con un solo comando, a nessun costo, verso due questioni distinte che ci si proponeva di risolvere automaticamente; l'inconveniente però è stato che anche i risultati di Google non sono molto affidabili su un input particolare come le schede Caronti. Si è deciso quindi di costruire attorno all'output di Cloud Vision un algoritmo che lo analizzasse e fosse in grado di correggere gli errori più comuni che si presentano nell'annotazione delle schede Caronti. Si tratta di una serie di istruzioni che unisce, cancella e modifica i poligoni descritti dalle annotazioni, realizzato sulla base di osservazioni di alcune segmentazioni dell'API e sull'impaginazione di un campione delle schede. L'algoritmo si concentra quasi esclusivamente sulla correzione delle segmentazioni, mentre interviene solo incidentalmente sulle trascrizioni, che rimangono quasi invariate da quelle ricevute da Cloud Vision.

Obiettivo
In linea con quanto scritto finora, l'obiettivo è stato quello di realizzare un'applicazione che servisse per la segmentazione e la trascrizione delle immagini delle schede Caronti, attraverso un'interfaccia intelligente, che fosse in grado di ridurre il carico di lavoro dell'utente e che fosse facile e veloce da usare. È stato quindi un requisito fondamentale

che il sistema fosse semiautomatico, ossia che fosse capace di suggerire all'utente una possibile soluzione del problema che è portato a risolvere, allo scopo di ridurre i tempi di completamento dell'annotazione. Ciò significa che, oltre all'applicazione in sé, è stato sviluppato un algoritmo che produce sia un'approssimativa segmentazione automatica dell'immagine della scheda, sia un'approssimativa trascrizione di ciascuna regione di testo identificata. Il vantaggio di creare un sistema specifico è stato quello di poter privilegiare le priorità che il progetto si prefiggeva, ossia velocità e facilità d'uso, a discapito magari di una maggior precisione nella segmentazione e senza complessità aggiuntive che si sarebbero manifestate nell'utilizzo delle applicazioni già esistenti, più ricercate ma più difficili da imparare ad usare. Il principio guida del design della nuova interfaccia è stato quello di ridurre il più possibile le interazioni dell'utente con il sistema (numero di click), senza però rinunciare ad un adeguato grado di precisione, in quanto la correttezza delle annotazioni è un requisito fondamentale per i modelli che devono fare affidamento su di un ground truth. Si è quindi realizzata un'interfaccia web ad hoc che implementa le seguenti operazioni:

- Caricamento dell'immagine da annotare
- Segmentazione e trascrizione automatiche della scheda
- Correzione manuale dei risultati automatici
- Salvataggio delle annotazioni

Lo spirito con cui si è progettata l'applicazione è stato quello di ridurre al minimo le features, implementando solo quelle necessarie, che facilitassero il compito dell'utente in maniera altamente significativa. Si è evitato appositamente di costruire un sistema estremamente sofisticato, con decine di funzionalità minori che sarebbero state sì utili, ma probabilmente solo in poche circostanze. Tutte le operazioni disponibili all'utente dovevano essere importanti, evidenti e visibili. In questo modo si è potuto mantenere un'interfaccia semplice e pulita, che favorisse la velocità di apprendimento. Il costo di un'applicazione progettata secondo questi principi è stato quello di una rinuncia ad un alto grado di precisione per quanto riguarda la segmentazione. I limiti di essa, per come è stata implementata l'interfaccia, consistono nell'avere annotazioni di regioni di testo multi-riga invece che riga per riga e dei semplici poligoni (tra l'altro esclusivamente quadrilateri) come bounding box al posto di poligoni perfettamente ritagliati attorno a ciascuna linea.

Risultati

Al termine dello sviluppo dell'applicazione, si è indagata la conformità ai requisiti di usabilità che nelle intenzioni originali l'interfaccia avrebbe dovuto soddisfare. Tale analisi è stata condotta attraverso tecniche di valutazione del software, con le quali si è messo

a confronto la nuova interfaccia con Transkribus, un'affermata applicazione di analisi dell'immagine di documenti e riconoscimento del testo. In base ai risultati ottenuti, ci si può ritenere soddisfatti del prodotto finale, in quanto dimostrano che un utente, senza una formazione tecnica, che deve annotare le schede Caronti, interagisce in maniera più veloce e più soddisfacente con l'app costruita ad hoc, piuttosto che con un software, più preciso nell'annotazione automatica, ma pieno di ulteriori funzionalità che lo disorientano. Si riesce quindi a semplificare il lavoro di colui che annota le immagini, al prezzo di una segmentazione non perfetta, ma che si ritiene sufficientemente valida per la costruzione del ground truth.

Indice

Capitolo 1

L'annotazione di immagini per la trascrizione di documenti

Il campo scientifico della visione artificiale (nota anche come *computer vision*) è la disciplina che studia come i computer possano ottenere una comprensione di alto livello da immagini o video digitali. Le funzioni che un sistema di visione artificiale deve fornire si distinguono tra:

- **Riconoscimento degli oggetti (o classificazione degli oggetti).** Il sistema assegna un'etichetta (classe di appartenenza) ad un oggetto presente nell'immagine in input.
- **Identificazione.** È la capacità di un sistema di individuare un'istanza specifica di un oggetto, come l'identificazione di un volto o delle lettere di un testo scritto a mano.
- **Rilevamento.** Il sistema scandisce un'immagine fino a che non individua il verificarsi di una condizione specifica, come la presenza di cellule anomale in immagini di natura medica.

L'annotazione è un processo fondamentale per la progettazione di modelli di visione artificiale. Essa consiste nell'etichettatura degli elementi che si vuole che il proprio sistema impari a riconoscere. Serve a popolare un dataset di immagini su cui un modello basato su machine learning si allena per divenire capace di identificare gli oggetti che si annotano. Di solito è un compito che viene svolto manualmente, a volte con il supporto di sistemi automatici. Nell'ambito del riconoscimento del testo, l'annotazione prevede una particolare forma di etichettatura, che consiste nella trascrizione delle scritte segmentate.

1.1 Background

1.1.1 Segmentazione dei documenti

La segmentazione è una delle operazioni di cui consiste l'annotazione. Il suo scopo è quello di dividere l'immagine nelle sue parti rilevanti, che, nel contesto del riconoscimento del testo, coincide con l'individuare esclusivamente le aree che contengono scritte. In pratica, si tratta di disegnare dei poligoni, detti *bounding boxes*, attorno alle porzioni di immagine entro cui stanno le scritte. Generalmente i sistemi OCR lavorano con singoli caratteri, parole o linee di testo come dati di input, ma modelli sofisticati sono in grado di operare su intere pagine di testo e/o regioni della pagina, proprio perché la pipeline completa di elaborazione di documenti di un modello di riconoscimento di testo include anche gli algoritmi di segmentazione automatica. Essi sono tipicamente classificati in tre gruppi [5]:

1. **Top-down**. La segmentazione inizia considerando la pagina intera che viene ripartita.

2. **Bottom-up**. La segmentazione inizia da un livello di granularità molto piccolo (singoli pixel) e si agglomerano gli elementi per ottenerne di maggiori fino ad arrivare all'ordine di grandezza della pagina stessa.

3. **Ibridi**.

Gli algoritmi di segmentazioni si possono però raggruppare anche a seconda del numero di layout su cui sono in grado di funzionare.

1. **Gruppo uno**. Di solito vengono usati per documenti con un layout specifico e predefinito, su cui si possono fare chiare assunzioni e dalle quali ricavare un insieme di regole utili a costruire la segmentazione. Hanno il grande vantaggio di poter essere utilizzati senza alcun allenamento, ma non hanno flessibilità.

2. **Gruppo due**. Questi algoritmi cercano di adattarsi a variazioni locali nella pagina allo scopo di poter segmentare diversi tipi di layout con un singolo algoritmo. Lo svantaggio che ne deriva è l'alto numero di parametri da sincronizzare e che potrebbe essere necessario un allenamento su dataset grandi. Si dividono in 3 sotto sottocategorie:

   (a) Algoritmi di clustering. Raggruppano gli elementi su basi geometriche, strutturali o di un insieme più generale di caratteristiche

   (b) Algoritmi basati su analisi di funzione. Dipendono da una funziona di ottimizzazione che cerca avvicinarsi il più possibile ad un certo valore.

(c) Algoritmi di classificazione. Sono allenati per distinguere diversi tipi di elementi in base a determinate caratteristiche.

3. **Gruppo tre**. Cercano di superare le limitazioni dei precedenti algoritmi combinandoli o con l'intelligenza artificiale, allo scopo di divenire capaci di funzionare su potenzialmente qualsiasi layout. Si suddividono in:

   (a) Algoritmi Ibridi. Combinano assieme vari algoritmi per sommarne la forza, ma tendono ad essere molto complessi.

   (b) Reti neurali. Apprendono autonomamente quali sono le features più significative per risolvere il compito, ma necessitano di un allenamento consistente.

Nonostante esistano numerosi validi metodi di segmentazione automatica, tutti sono inclini ad errori, specialmente quando applicati su documenti storici. Conseguenza di ciò è che nello sviluppo di un modello di riconoscimento del testo scritto a mano è quasi sempre necessaria una fase di segmentazione manuale, ossia durante la generazione del ground truth. Di solito, al momento dell'annotazione manuale, sia essa fatta dall'utente da zero oppure a partire da un'approssimazione automatica, è richiesto di escludere i falsi positivi e delimitare le aree contenenti esclusivamente testo, aggiustandone i contorni; non è necessario disegnare bounding box attorno alle singole parole o, ancora peggio, ai caratteri, che sarebbe sì eccesivamente estenuante, bensì è sufficiente individuare le linee di testo o anche solo intere aree multi-riga. Avendo ottenuto in questo modo delle regioni più pulite, il modello avrà vita più facile nel distinguere linee, parole e caratteri all'interno di esse.

1.1.2 Modelli OCR

Il riconoscimento automatico dei caratteri nei documenti (OCR: *optical character recognition*) è da decenni un attivo campo di ricerca, poiché permette di risparmiare tempo e fatica nell'operazione di digitalizzazione dei testi (libri e documenti di ogni tipo). Creare un archivio digitale di scritti di cui esistono solo copie cartacee è una necessità molto comune, le cui ragioni principali di solito risiedono nell'accesso e nella consultazione rapida delle opere, oltre che nella conservazione a lungo termine delle informazioni che contengono. Questo vale in particolare per i documenti storici, che inevitabilmente esistono solo su carta e sono a rischio di serio degrado. I sistemi OCR hanno raggiunto risultati diversi con velocità diverse a seconda che lavorino su testi stampati o su testi scritti a mano. I primi sono infatti più facili da trascrivere e modelli di riconoscimento sono stati perfezionati con esiti più che soddisfacenti, mentre i secondi hanno richiesto più anni per raggiungere risultati peggiori. Ciò è inevitabile se si considera quanto è variabile la calligrafia umana, unica per ciascun individuo, rispetto a quella delle macchine e quanto ciò complichi le cose nel momento in cui si deve istruire un modello di riconoscimento. Il sistema allenato su documenti stampati ha buone possibilità di funzionare bene

su documenti stampati di un dataset diverso da quello di allenamento, ma il sistema allenato su documenti scritti a mano (solitamente da una o poche persone) ha poche speranze di trascrivere bene documenti scritti da autori diverso da quelli del dataset di partenza. La differenza è estremamente più accentuata quando si tratta di manoscritti storici, che portano con sé complicazioni ulteriori, quali stili di scrittura e layout del testo unici. I riconoscitori automatici di testo si dividono in due gruppi: online ed offline. I primi usano dati di input dinamici basati sul movimento della punta della penna di cui si considera velocità, angolo di proiezione, posizione e punto di localizzazione; quelli offline invece usano come input delle immagini scansionate. Relativamente al progetto Caronti, serve sviluppare un riconoscitore del secondo tipo e negli ultimi decenni sono state create diverse valide soluzioni con metodi basati sul machine learning [6]:

- **Metodi kernel**. Esempi: support vector machines (SVMs), Kernel Fisher Discriminant Analysis (KDA), Kernel Principal Component Analysis (KPCA). Questi algoritmi eseguono una mappatura dei dati in uno spazio di caratteristiche multidimensionali, al fine di trovare un iperpiano che separa linearmente le classi con il maggior margine possibile. Prima dell'esplosione di popolarità del deep learning, queste erano fra le più robuste tecniche per il riconoscimento di testo scritto, classificazione di immagini, rilevamento di volti e oggetti e classificazione del testo. Ci sono ricercatori che ritengono che la tecnologia SVM ancora sia migliore della maggior parte delle alternative in ambito OCR.

- **Metodi statistici** Si dividono in parametrici e non-parametrici.

  - **Parametrici**. Esempi: Hidden Markov Model (HMM), Linear Discriminant Analysis (LDA), Logistic Regression (LR). Utilizzano un numero fissato di parametri, in quanto presuppongono che l'insieme dei dati di addestramento possa essere modellato da una distribuzione di probabilità che ha un insieme fisso di parametri. Nel campo OCR, i caratteri sono solitamente classificati sulla base di regole di decisione come quello della massima verosimiglianza o dell'inferenza bayesiana. Gli algoritmi appartenenti a questa classe sono generalmente veloci nell'apprendimento e possono lavorare con dataset piccoli, tanto che c'è chi ritiene che in questi casi il metodo HMM fornisca risultati migliori.
  - **Non-parametrici**. Esempi: k nearest neoghbor (kNN), Decision Trees (DT). Questi algoritmi non richiedono alcuna conoscenza a priori e il numero di parametri che utilizzano cresce all'aumentare del training set. Il metodo kNN è stato usato per più di un decennio nel riconoscimento dei caratteri e si ritiene che raggiunga ancora buoni risultati.

- **Tecniche template matching**. Esempi: deformable template matching, rigid template matching. Consiste in un approccio tale per cui piccole porzioni di un'im-

magine vengono abbinate a dei modelli predefiniti e si misura la corrispondenza tra le due parti con funzioni apposite (distanza euclidiana, city block distance, correlazione incrociata, ecc.).

- **Riconoscimento dei pattern strutturali**. Questa tecnica cerca di classificare gli oggetti in base alle relazioni fra i propri tratti costitutivi (forme, segni, linee) ricavati da alcuni tratti primitivi. Si dividono in *graphical methods* e *grammar based methods*. Per funzionare l'algoritmo necessita di un'immagine in formato binario e contorni molto ben definiti; per queste ragioni, i sistemi OCR che sfruttano questa tecnica non sono molto adatti per il riconoscimento di testo scritto a mano.

- **Reti neurali** Esempi: Recurrent Neural Networks (RNN), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) networks. Le reti neurali sono costituite da elementi chiamati neuroni connessi gli uni agli altri, dove ad ogni connessione è associato un valore, chiamato peso. Questi neuroni lavorano assieme per manipolare i dati in input e ottenere in output una classe di appartenenza tra quelle predefinite. Negli ultimi anni le reti neurali si sono affermate come il miglior metodo di classificazione, anche in ambito OCR. Il prezzo da pagare per una così elevata precisione è un alto costo computazionale e un training set di dimensione consistente.

1.1.3 Applicazioni web

Le applicazioni web sono tutti quei programmi cui si interagisce dal browser. Sono estremamente popolari in quanto è facile accedervi, non richiedono installazioni, configurazioni o sistemi hardware specifici, il che permette a chiunque sia fornito di un browser e di una connessione internet di usufruirne. Inoltre, le tecnologie che supportano lo sviluppo di applicazioni di questo tipo sono innumerevoli e semplificano molto il lavoro dei programmatori nella costruzione sia dell'interfaccia del sistema sia delle funzionalità che implementa.

1.1.3.1 Applicazioni multi-page

Un'applicazione multi-page consiste in un tradizionale sito web, il cui contenuto è suddiviso in multiple pagine. Ciascuna di esse risiede sullo stesso server ed il client, di volta in volta, chiede di poterne visualizzare i contenuti; il server elabora la richiesta e restituisce al client la pagina con le relative risorse. La navigazione avviene quindi fra pagine diverse, identificate dall'URL, dove ognuna porta con sé il proprio codice HTML e Javascript. Lo scambio di dati tra il server ed il client, che avviene ad ogni caricamento pagina, per quanto grande, non è un problema dal momento che la tecnologia AJAX (Asynchronous JavaScript and XML) garantisce una comunicazione veloce tra le due parti. Secondo il paradigma multi-page, il rendering è server-side, poiché è il server che

invia la pagina corretta da visualizzare al client, il quale ne attende il caricamento ogni volta che la richiede. Questo modello viene usato per applicazioni con molti contenuti tra cui navigare ed è comodo per la gestione SEO, ma aggiunge complessità allo sviluppo sia del front-end che del back-end, che devono essere strettamente connessi.

1.1.3.2 Applicazione single-page

L'approccio *Single-page application* (SPA) è invece quello più moderno, che consiste nello sviluppo di una singola pagina, il cui contenuto viene riscritto in maniera dinamica dal client, invece che essere caricato dal server. In risposta alle azioni che l'utente esegue sul sito, si attiva del codice Javascript che modifica il contenuto visualizzato. Il pregio di questo paradigma è la velocità di navigazione, poiché il rendering è client-side (non si attende che la pagina venga fornita dal server), il debugging è più semplice e si può riutilizzare il codice del back-end per costruire applicazioni mobile. D'altra parte, il fatto che l'URL sia sempre lo stesso, dato che la pagina non cambia mai, è fonte di due principali problemi: è più difficile fare ottimizzazione per i motori di ricerca e rende disfunzionale l'uso della cronologia browser (frecce avanti ed indietro).

1.1.4 Elementi di interfacce

Si inserisce qui un breve elenco di componenti tipiche delle interfacce web, di cui è stato fatto uso nel progetto, che potrebbero non essere note a tutti:

1. **Carosello**. Anche noto come slider, galleria o slideshow, è un elemento dell'interfaccia dove vengono esibite grafiche, immagini o video tra cui scorrere. Si usa quando si ha una collezione di elementi omogenei (ad esempio immagini), in modo tale che se ne possa visualizzare uno (o qualcuno) alla volta, ma con la possibilità di passare a quelli successivi quando lo si desidera.

2. **Tooltip**. Si tratta di un messaggio che compare in primo piano sullo schermo quando il cursore si posa su di un elemento della pagina.

3. **Toast**. Il toast è un elemento che svolge una funzione di feedback per l'utente, per segnalare l'occorrenza di un evento. È un oggetto che compare in primo piano sullo schermo, di solito su di un bordo della finestra dell'interfaccia, che tipicamente scompare da solo.

4. **Badge**. Consiste in un piccolo numero o simbolo che si aggiunge ad una riga di testo, o più spesso ad un pulsante, per notificare un cambio di stato. Se è un numero allora fornisce un sistema di conteggio, altrimenti serve per evidenziare l'elemento cui è associato.

1.2 Applicazioni per l'annotazione di immagini

Nel corso degli ultimi 25 anni sono state proposte diverse applicazioni per fare annotazione manuale di immagini. In questo capitolo si elencano i lavori più rilevanti in letteratura, anche se la maggior parte di essi non sono mai stati open-source o non sono più disponibili per essere testati. Man a mano che saranno presentati i lavori più recenti, si approfondirà meglio l'aspetto dell'usabilità dei programmi e quindi la loro interfaccia.

1.2.1 Annotatori di immagini

Il tipo più generico di annotatore di immagini di solito prevede di avere a che fare con foto scattate in scenari molto comuni, come la stanza di una casa o una strada, con persone, animali e oggetti di vario genere, ma sempre in un contesto di usuale normalità. Permettono all'utente di disegnare forme di ogni tipo per ritagliare un qualsiasi elemento dell'immagine e gli richiedono di associare a ciascuna segmentazione un'etichetta (ad esempio: divano, persona, cane, ecc.). Queste annotazioni servono poi a creare vasti dataset usati per allenare modelli di riconoscimento di immagini. Seguono alcune tra le applicazioni più importanti ed interessanti.

1.2.1.1 LabelMe, 2005

LabelMe [7] è un programma di annotazione di immagini storicamente tra i più importanti. Nato nel 2005 come progetto open-source, è stato aggiornato fino ad oggi ed è possibile installarlo liberamente [8]. Non è pensato per i documenti, bensì per foto qualsiasi. L'interfaccia è semplice e completa: si presenta con una barra verticale a sinistra con i comandi, al centro l'immagine su cui si lavora e a destra delle finestre con l'elenco delle associazioni etichette-colore e la lista dei poligoni creati. Il testo destro permette di aprire un menù alternativo con tutte le opzioni relative al disegno. Per disegnare bisogna cliccare uno dei pulsanti tasti del menù, in base a quale tipo di forma geometrica si vuole usare: poligono, cerchio, linea, punto, rettangolo. Nel primo caso si cliccherà una volta per ciascun vertice, negli altri casi invece è sufficiente cliccare e trascinare il mouse. Una volta completata la figura, compare in primo piano una finestra popup in cui specificare l'etichetta da associarvi, che apparirà nel menù a destra. Per modificare le segmentazioni è necessario cliccare lo specifico pulsante che accede alla modalità di editing e a quel punto si potranno ridimensionare, ridefinire e spostare i poligoni; per cancellarli invece esiste un'altra modalità apposita da selezionare.

1.2.1.2 VGG Image Annotator (VIA), 2016

Pubblicato per la prima volta nel 2016 [9], ad oggi è uno dei software gratuiti di annotazione di immagini, audio e video più popolari di tutti. Funziona sul browser e nasce

con un chiaro intento progettuale, cioè essere facile da usare: è veloce, minimalista e non richiede installazioni, configurazioni o una formazione tecnica. Gli autori riportano di essersi trattenuti dall'aggiungere features per evitare di aumentare troppo la complessità in termini di usabilità ed implementazione. L'interfaccia è composta solamente di una barra dei comandi in alto e di una finestra che sale dal basso in cui ci sono i dettagli delle annotazioni, mentre il centro della pagina è lasciato all'immagine. Sopra il margine in basso della pagina, in primo piano, appare una finestrella in cui vengono dati dei suggerimenti all'utente su cosa fare, in base ai comandi che ha selezionato.

1.2.1.3 FluidAnnotation, 2018

Nel 2018 viene pubblicato un articolo che presenta FluidAnnotation [10], uno strumento per annotare le immagini per intero (senza escludere alcun elemento presente, nemmeno lo sfondo), che si prefigge come scopo quello di avere un'interfaccia naturale ed altamente efficiente che richieda molto meno sforzo manuale rispetto alle altre interfacce, come quella di LabelMe. Per farlo, si basa su un design costruito su 3 principi:

1. Annotazione automatica (segmentazione ed etichettatura) come base di partenza per l'utente, che può correggere manualmente.
2. Esecuzione dell'annotazione dell'immagine completa in un solo passaggio.
3. Eliminazione di una sequenza predeterminata di azioni: l'utente decide cosa e in quale ordine annotare gli oggetti.

Una volta caricata l'immagine segmentata automaticamente, l'utente può eseguire le seguenti azioni:

- **Aggiungere/Modificare un segmento**. L'utente non disegna mai poligoni, ma quando clicca in un qualsiasi punto, il sistema calcola alcune possibili forme contenenti tal punto, che probabilmente racchiudono l'oggetto che si vuole annotare; si può scorrere e visualizzare la lista delle proposte con la rotella del mouse e cliccare quella che meglio si adatta alla soluzione.
- **Rimuovere un segmento**. Si seleziona il poligono e si clicca il tasto destro.
- **Cambiare etichetta**. Si posiziona il mouse sopra il segmento e si clicca un qualsiasi tasto della tastiera: si apre un menù a discesa con l'elenco delle possibili etichettature, o in alternativa si digita il testo manualmente.
- **Cambiare l'ordine di sovrapposizione**. Può capitare che due forme si sovrappongano parzialmente; in questo caso si posiziona il mouse su una delle due figure e si scrolla con la rotella del mouse per cambiarne l'ordine.
- **Nascondere le annotazioni**. Per visualizzare meglio i dettagli dell'immagine originale, si può premere la barra spaziatrice in modo tale da nascondere i segmenti.

1.2.2 Annotatori di documenti

Segmentare e trascrivere immagini con testo è un compito che fin dagli anni '90 ha portato allo sviluppo di programmi specifici, che mettevano a disposizione degli strumenti per disegnare linee e poligoni allo scopo di evidenziare certe regioni della pagina ed associare ad ognuna una qualche etichetta o la trascrizione corrispondente. Soprattutto le prime applicazioni richiedevano hardware o sistemi operativi specifici per poter essere installati ed eseguiti e soprattutto si concentravano su immagini di testo stampato, non scritti a mano. Questi sono stati i principali limiti di tali primi sistemi, di cui segue l'elenco.

1.2.2.1 Pink Panther, 1998

Pink Panther [11] è stato il primo progetto di gran rilievo (non il primo in assoluto) che si è occupato di sviluppare un programma per l'annotazione di un'immagine di una pagina di testo. Permetteva agli utenti di disegnare con click del mouse zone rettangolari o poligonali per identificare regioni dell'immagine separate e di specificare per ciascuna di esse il tipo di quella zona e la relazione con le altre. Funzionava bene su documenti semplici, ma non altrettanto su documenti scritti a mano, in particolare quelli storici, che spesso hanno layout complessi.

1.2.2.2 TrueViz, 2000

TrueViz [12] consentiva di generare un ground truth per linee di testo, parole e glifi. Lo schermo dell'interfaccia era separato in due pannelli: a sinistra quello con l'immagine con disegnati sopra dati geometrici e a destra quello che mostrava i metadati in una struttura ad albero. Come nel caso di Pink Panther, TrueViz era più utile per l'annotazione di documenti con elementi regolari, piuttosto che di manoscritti. Inoltre aveva requisiti hardware specifici per l'installazione e la configurazione.

1.2.2.3 PerfectDoc, 2005

PerfectDoc [13] era simile a TrueViz, ma incorporava funzionalità avanzate come la lettura multi-pagina. Era utilizzato principalmente per la correzione del testo piuttosto che per la generazione di ground truth e, come nei casi precedenti, non è stato pensato per documenti scritti a mano.

1.2.2.4 PixLabeler, 2009

PixLabeler [14] scelse un approccio diverso dai lavori precedenti per fare le annotazioni: esse erano a livello di pixel invece che di regione, ossia tale che ogni pixel avesse la propria annotazione. Il programma distingueva fra pixel in primo piano e dello sfondo, scartava i secondi e offriva una collezione di etichette predeterminate sotto il quale registrare i

pixel selezionati, i quali venivano colorati in base alla categoria di appartenenza. I pixel erano selezionati attraverso rettangoli, lazo, poligoni o passandoci sopra (in modalità "pennello"). Nei primi due casi, le forme geometriche erano disegnate trascinando il mouse, mentre i poligoni erano creati cliccandone i vertici. Questo strumento ha ricevuto le stesse critiche di quelli già visti, ossia che funzionava bene con documenti con elementi semplici e non con documenti storici scritti a mano.

1.2.2.5 GEDI, 2010

GEDI [15] era uno strumento altamente configurabile e per questo si presentava con un'interfaccia ricca di comandi. Implementava un sistema di trascrizione del testo tale per cui per ciascuna bounding box si poteva specificare a quale altezza iniziava e finiva ogni parola, in modo da associare la trascrizione di essa non all'intero poligono ma ad una sua parte. Supportava numerose funzionalità, tra cui quelle di merging/splitting e quella di modifica dell'ordine di lettura delle zone scritte.

1.2.2.6 Aletheia, 2011

Con Aletheia [16], l'utente poteva disegnare i confini delle regioni di testo e successivamente perfezionare la selezione con comandi di divisione e riduzione dei confini del poligono; questo era l'approccio top-down. In alternativa si poteva procedere con un approccio bottom-up, in cui si selezionavano elementi base del testo che venivano aggregati sotto un'unica annotazione (per esempio, caratteri individuati separatamente per ottenere la selezione di una parola).

1.2.2.7 WebGT, 2013

È stata WebGT [17] la prima applicazione di annotazione manuale con un'interfaccia web e specificatamente pensata per documenti storici scritti a mano. Il sistema era organizzato secondo tre moduli: selezione, raggruppamento e annotazione. Esso forniva funzioni semiautomatiche per rilevare singoli elementi che si volevano selezionare e per raggrupparli in uno unico di gerarchia superiore (come lettere in una parola o parole in una linea di testo). Una volta cliccato il componente testuale che si desiderava selezionare, se il processo automatico non rilevava bene i contorni di esso, allora l'utente poteva correggerli manualmente. Successivamente, l'utente doveva passare alla modalità annotazione per indicare la trascrizione dell'elemento che si era selezionato. Gli autori riportano che la generazione del ground truth di una pagina con 300 lettere scritte a mano è stata eseguita in 40 minuti.

1.2.3 Annotatori di documenti antichi scritti a mano

Si tratta di un tipo ancora più specializzato di sistemi atti all'annotazione di immagini di documenti, con il preciso compito di segmentare e trascrivere le parole presenti sulle pagine di manoscritti. Di solito sono applicazioni sofisticate, perché devono superare ostacoli tipici come il degrado del supporto cartaceo, impaginazioni peculiari o presenza di decorazioni in mezzo al testo. Non a caso, solo negli ultimi anni sono stati pubblicati strumenti molto validi per questo tipo di annotazione, in quanto si basano tutti su potenti modelli di machine learning, la cui popolarità in questo campo è esplosa recentemente.

1.2.3.1 Transkribus, 2017

Il software Transkribus [18] rappresenta un punto di svolta nell'ambito della trascrizione automatica di doucmenti. Il progetto è iniziato nel 2013 per poi essere aggiornato e sviluppato continuativamente, tanto che oggi è uno degli strumenti di punta nel riconoscimento del testo. Il suo sistema di rilevazione del layout è automatico, per cui individua la posizione delle scritte nella pagina e le linee da cui sono composte, poi ovviamente consente di eseguire correzioni manuali. In questo caso, offre diversi comandi per cancellare, unire, espandere/ridurre le aree e le linee di testo errate. Una volta calcolata la trascrizione, questa compare in una finestra dell'applicazione, suddivisa per righe; quando una di esse viene selezionata per essere modificata, la linea di testo dell'immagine cui fa riferimento si colora, così da rendere chiaro all'utente la loro corrispondenza. L'applicazione è alquanto sofisticata e ciò si riflette nell'interfaccia, la quale fornisce una moltitudine di comandi e di informazioni, che occupano tutto il lato sinistro dello schermo. La metà destra superiore mostra l'immagine su cui si lavora, mentre quella destra inferiore contiene la finestra della trascrizione.

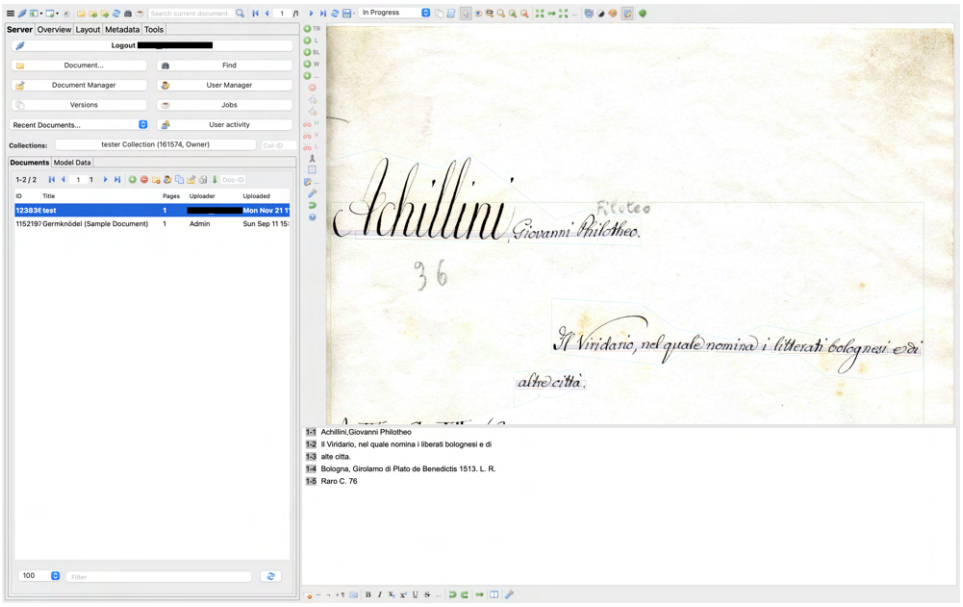


Figura 1.1: Interfaccia di Transkribus

1.2.3.2 DIVAnnotation, 2018

L'università di Friburgo da più di 15 anni finanzia un gruppo di ricerca per l'analisi automatica di documenti storici chiamato DIVA. Negli ultimi anni hanno pubblicato diversi articoli in cui aggiornano i risultati che hanno tenuto. Nel 2018 hanno presentato DIVAnnotation [19], uno strumento open-source per fare annotazione di manoscritti. La segmentazione dell'immagine è manuale: si possono costruire rettangoli cliccando e trascinando il mouse oppure dei poligoni cliccando i punti dei suoi vertici. Poi si seleziona la regione appena creata e si segmentano le righe di testo con la stessa modalità di prima oppure attraverso una funzione automatica. L'operazione manuale di annotazione delle righe è resa ancora più facile se si ha disposizione una penna stylus, che il sistema supporta. Infine è implementata una funzione di trascrizione automatica. L'interfaccia non ha un aspetto molto sofisticato, anzi appare obsoleta. È organizzata per tab, visibili nel menu di navigazione in alto, mentre sulla sinistra compare l'elenco dei comandi disponibili sotto forma di pulsanti con il testo interno che ne esplicita la funzione. Il flusso di lavoro appare un po' troppo rigido, l'estetica degli elementi è trascurata e lo spazio della finestra non viene occupato bene (quasi metà schermo, a destra, è vuoto).

1.2.3.3 eScriptorium, 2019

Nel 2019, l'Università PSL ha pubblicato il progetto eScriptorium [20] che, similmente a Transkribus, oggi rappresenta lo stato dell'arte della trascrizione automatica di documenti storici. L'iniziativa è nata perché si voleva realizzare un'applicazione per fare

analisi di documenti che fosse sia open-source sia fruibile sul web. Quando si carica l'immagine da annotare, eScriptorium la segmenta automaticamente e riconosce tutte le linee di testo presenti; in caso di falsi positivi, l'utente può facilmente eliminarli manualmente. Inoltre, il sistema fornisce la possibilità di eseguire aggiustamenti manuali delle bounding box delle regioni in maniera simile a Transkribus. Dopo di che, si clicca sulle righe di testo identificate e compare una finestra popup con l'immagine ritagliata di quella linea e una barra di input per inserire la trascrizione. Il software fornisce poi la possibilità di salvare i risultati e di allenare un modello di trascrizione automatica sul ground truth così generato.

Capitolo 2

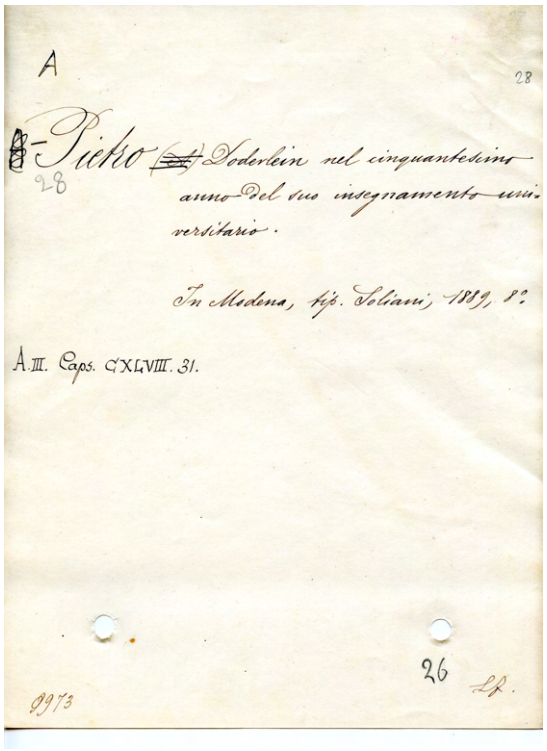
Il sistema tonicarD

È stata chiamata tonicarD l'applicazione costruita per questo progetto. Da una parte, si è trattato di sviluppare un'interfaccia web per l'annotazione delle immagini delle schede Caronti, al fine di generare il ground truth di un modello di riconoscimento automatico del testo, dall'altra si è elaborato un algoritmo in grado di svolgere parzialmente il compito dell'utente al posto suo, in modo da fornirgli un risultato di partenza su cui lavorare. Ci si è concentrati quindi sulla progettazione di un'interfaccia che fosse intuitiva, perciò veloce da imparare ad usare, e focalizzata al conseguimento di un obiettivo molto specifico. In costante considerazione degli aspetti di usabilità del software, sono state implementate solo le funzionalità più vantaggiose, che velocizzano il compito dell'utente in maniera significativa e che sono destinate ad essere sfruttate con frequenza, evitando in questo modo di arricchire in maniera superflua l'interfaccia di una moltitudine di comandi che ne aumenterebbero la complessità e avrebbero poco impatto sull'esperienza dell'utente. Si è tentato, inoltre, di inserire un sistema di annotazione automatico come elemento di intelligenza dell'interfaccia, in quanto svolge una funzione di supporto all'utente, che si ritrova nella condizione di correggere un risultato intermedio, piuttosto che in quella di costruire da zero il risultato finale.

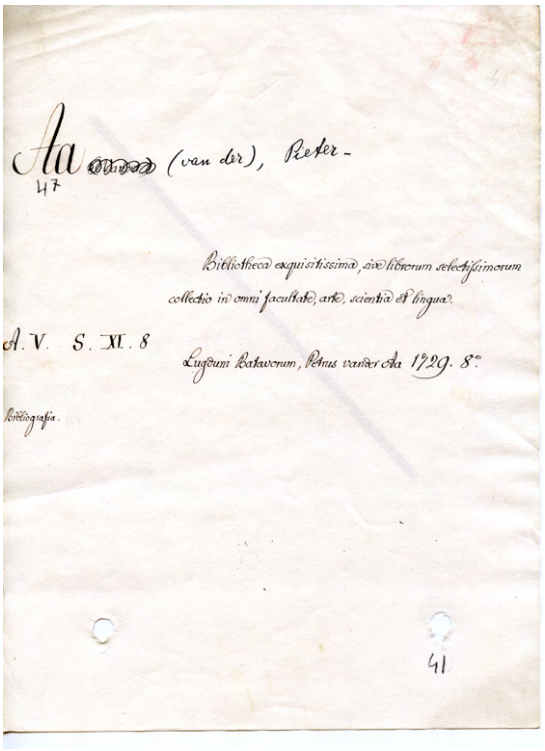
2.1 Il catalogo Caronti

Dalle circa 150 000 schede scritte a mano dal Dott. Andrea Caronti è stato prodotto un piccolo dataset di poco più di 300 immagini in alta risoluzione (600 pixel per pollice) in formato jpeg. Si tratta di schede bibliografiche che costituiscono il catalogo della Biblioteca Universitaria di Bologna e ciascuna è associata ad un libro, di cui ne indica: autore, titolo, note di catalogazione e collocazione. Non sempre sono presenti tutte e quattro queste indicazioni, che inoltre sono sparpagliate lungo la scheda. Il layout non è dunque quello classico riga per riga dal bordo sinistro a quello destro della pagina, ma ciascuna componente (autore, titolo, etc.) occupa una certa area e quando va a capo lo fa

entro i confini di essa. È riscontrabile un certo schema nella disposizione degli elementi, tale per cui l'autore occupa la posizione più in alto a sinistra, il titolo la parte destra, la collocazione sta schiacciata a sinistra a metà scheda e infine le note di collocazione si trovano al centro a destra sotto il titolo. Il testo tendenzialmente occupa solo la metà alta della scheda e tutte le scritte che ne stanno al di fuori non sono di interesse ai fini della trascrizione, tuttavia il dataset è caratterizzato da numerose eccezioni; ad esempio, la scheda in figura 2.1a non ha l'indicazione dell'autore e al suo posto prende spazio il titolo, oppure ci sono casi in cui il testo è talmente lungo da occupare anche la metà bassa della scheda.



(a) Scheda 0001028



(b) Scheda 000147

Figura 2.1: Due esempi di schede Caronti

La lingua in cui sono scritte è variabile, perché dipende dalla lingua del libro cui fanno riferimento, per cui vi sono scritte in italiano, in latino, in tedesco, ecc. Lo stile calligrafico è molto curato, rotondo, elegante e quindi diverso da una scrittura moderna e comune. Infine, le schede contengono delle correzioni: righe tracciate su scritte, aggiunte fatte a matita, scarabocchi a coprire il testo sottostante, ecc. L'insieme di questi fattori rende il dataset Caronti estremamente complesso da trascrivere automaticamente, spesso anche solo da segmentare.

2.2 Architettura di tonicarD

tonicarD è un'applicazione web single-page ed in quanto tale, consiste di due componenti fondamentali, ossia il front-end ed il back-end. Il primo si preoccupa della visualizzazione dei contenuti e dell'interazione uomo-macchina, mentre il secondo raccoglie, manipola e gestisce i dati che servono al programma per funzionare. Queste due entità sono in costante comunicazione, che prevede che il lato client invii richieste al server, il quale le riceve, esegue un qualche tipo di lavoro in base alla richiesta individuata e restituisce un valore al front-end. Nel contesto del sistema tonicarD, il client non ha mai bisogno di ricevere dei dati dal server: una risposta gli serve come mero segnale che il lavoro che ha richiesto è stato compiuto; infatti, tutte le operazioni di cui si occupa il back-end, consistono nella generazione, cancellazione e modifiche di file, che il front-end può recuperare autonomamente una volta che viene informato che essi sono stati preparati. Le tecnologie utilizzate per lo sviluppo del front-end sono state quelle tipiche del web: HTML, CSS e Javascript. In aggiunta a questi, l'unico framework che è stato usato è Bootstrap[1], che è stato sfruttato principalmente per il sistema a griglia, la gestione mostra/nascondi degli elementi e per alcuni componenti di interfaccia predefiniti (barra di navigazione, badge, toast, ecc.). Per quanto riguarda il lato server, esso è stato costruito attraverso il framework Flask[2], che permette di lavorare in un ambiente Python. Ciò ha semplificato lo sviluppo, infatti per questo progetto erano necessarie librerie che fornissero servizi di:

1. Connessione al servizio di Google Cloud Vision API
2. Funzioni di manipolazione di immagine
3. Supporto di funzioni geometriche

Python, in quanto uno dei linguaggio più popolari degli ultimi anni, ha a sua disposizione un enorme quantità di librerie, tra cui anche alcune che servono agli scopi descritti sopra. In particolare, nell'ordine, sono state usate la libreria google-cloud-vision[3], Pillow[4] e Shapely[5].

Inizializzazione contenuti
Affinché l'applicazione possa essere lanciata, come prima cosa bisogna avviare il server, che si mette così all'ascolto di ogni possibile messaggio proveniente dell'interfaccia, poi

[1]Bootstrap, https://getbootstrap.com/, data di ultima consultazione 1 dicembre 2022
[2]Flask, https://flask.palletsprojects.com/en/2.2.x/, data di ultima consultazione 1 dicembre 2022
[3]Libreria Python di Google Cloud Vision, https://pypi.org/project/google-cloud-vision/, data di ultima consultazione 1 dicembre 2022
[4]Pillow, https://pillow.readthedocs.io/en/stable/, data di ultima consultazione 1 dicembre 2022
[5]Shapely, https://shapely.readthedocs.io/en/stable/manual.html, data di ultima consultazione 1 dicembre 2022

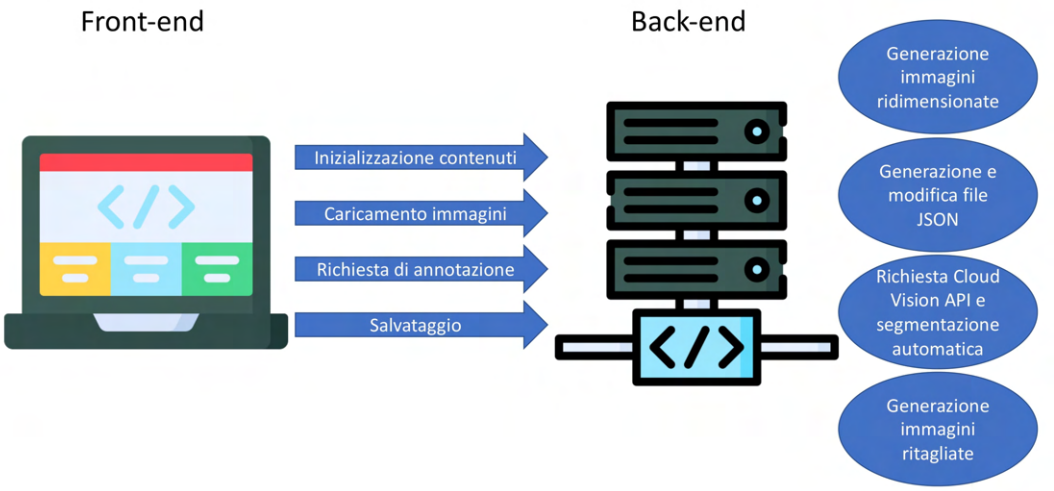


Figura 2.2: Architettura dell'applicazione tonicarD. Si noti che la comunicazione è fondamentalmente unidirezionale. In risposta ad ogni richiesta del client, il server esegue del lavoro che si concretizza nel salvataggio all'interno del file system di documenti JSON o JPEG. Una volta terminate tali operazioni, il server ne informa il client.

si attiva il lato client. Non appena viene caricata la pagina web del programma, viene svolto dal back-end del lavoro preliminare allo scopo di identificare i contenuti che devono essere mostrati. Nello specifico, dapprima si controlla quali immagini delle schede sono note al sistema in base a quelle presenti in un'apposita cartella e si confrontano con le informazioni contenute in un file JSON che serve a tenere traccia dello stato di tutte le schede. Si sincronizzano queste due entità in modo tale che:

- Se esiste l'immagine di una scheda che non è presente nel JSON, allora la si aggiunge assegnandole lo stato di scheda non annotata.
- Se nel file JSON c'è traccia di una scheda la cui immagine è irreperibile nel sistema, allora la si cancella.

In questo modo, quando un utente vuole integrare nel sistema un nuovo insieme di schede, può farlo in un colpo solo aggiungendo i file JPEG alla cartella prestabilita. Alla successiva esecuzione del programma, il server rileverà la presenza di nuove immagini e le mostrerà assieme alle altre. Prima di caricarle, però, il back-end si preoccupa anche di creare per ciascuna nuova immagine una copia ridimensionata, approssimativamente della stessa grandezza che le schede avranno sulla finestra dell'applicazione. Se così non fosse, il browser farebbe molta più fatica a calcolare le giuste proporzioni delle schede ad ogni schermata, il che ne rallenterebbe il caricamento ed il lavoro dell'utente.

Individuazione scheda e richiesta di annotazione automatica
L'utente ha almeno tre possibili modi per rintracciare l'immagine che desidera annotare:

1. Cercare lungo il corretto carosello dell'homepage.
2. Cercare nella pagina di ricerca.
3. Caricare un file presente sul proprio dispositivo attraverso l'apposito comando nella barra di navigazione.

Solo l'ultimo di questi metodi coinvolge il back-end, che si preoccupa di ridimensionare ed aggiungere il nuovo file nelle adeguate cartelle di sistema, così da poterlo sempre recuperare in futuro. Quando poi si clicca sul comando di richiesta di annotazione automatica, il client informa il server e si mette in attesa di una sua risposta. Nel frattempo, l'utente è libero di navigare nell'applicazione, eseguire annotazioni, aggiungere in coda nuove richieste o cancellare quelle pendenti. Eventuali annotazioni in attesa, verranno prese in carico una alla volta, al termine di quella precedente. Intanto, il back-end interroga Cloud Vision API, che può impiegare tra i 5 e i 50 secondi secondi a calcolare l'annotazione, a seconda della connessione internet. Una volta ottenuta, viene eseguito l'algoritmo di correzione della segmentazione e si salva un nuovo file JSON, il quale contiene le informazioni necessarie al client per poter disegnare le bounding box sull'immagine ed associarvi la trascrizione corrispondente. In caso qualcosa fosse andato storto in una qualsiasi fase della richiesta di annotazione automatica, il server lancia un'errore, che il front-end cattura ed informa l'utente che si è verificato un problema.

Correzione annotazione e salvataggio
La pagina di editing permette di visualizzare solo una scheda alla volta. Quando l'utente ne sceglie una su cui lavorare, il client si preoccupa di fornire gli strumenti di manipolazione delle bounding box e modifica delle trascrizioni. L'utente è libero di eseguire le azioni di editing senza seguire alcun ordine prestabilito. Il server viene nuovamente coinvolto solo quando l'annotazione viene salvata: esso riceve le coordinate dei vertici dei poligoni disegnati e, in base a quelli, ritaglia delle nuove immagini, che isolano una certa regione del testo della scheda, e che salva in un'apposita cartella del sistema: saranno gli elementi del nuovo ground truth. Allo stesso tempo, viene salvato/sovrascritto il file JSON che contiene le informazioni delle bounding box e delle trascrizioni della scheda.

2.3 Interfaccia di tonicarD

In questo capitolo si approfondisce il design dell'interfaccia di tonicarD. L'obiettivo era quello di avere un sistema che fosse in grado di segmentare e trascrivere le schede il più possibile in maniera autonoma e che poi fornisse ad un supervisore umano gli strumenti

per correggere i risultati e salvare le annotazioni. L'applicazione è stata progettata per essere semplice e veloce da usare, implementando solo le funzionalità necessarie per ottenere una segmentazione e trascrizione soddisfacenti, evitando quindi qualsiasi complessità ulteriore che avrebbe potuto appesantire l'interfaccia o alzare la curva di apprendimento di usabilità del software. Le operazioni fondamentali di un completo ciclo di annotazione sono:

1. Selezione del file da editare.
2. Richiesta di segmentazione e trascrizione automatica.
3. Visualizzazione e correzione manuale dell'annotazione automatica.
4. Salvataggio delle annotazioni.

L'applicazione è web, quindi funziona sul browser ed è pensata per l'uso esclusivo sul computer (non su smartphone, tablet o altri dispositivi touchscreen). D'altronde la tipologia di utente e il contesto per la quale è pensata sono molto specifici: si tratta del personale della Biblioteca Universitaria di Bologna quando si trova nel proprio ufficio, quindi su di un computer disponibile per lo staff. Non sono dunque previsti scenari in cui questo programma venga utilizzato da persone qualsiasi, in ambienti qualsiasi o con dispositivi qualsiasi. L'interfaccia è organizzata su tre pagine diverse, quali destinazioni del sistema di navigazione: homepage, pagina di editing e pagina di ricerca globale.

2.3.1 Homepage

La struttura di un'applicazione come quella che si andava a creare per questo progetto è molto semplice, né profonda né larga, come si può intuire dall'esiguo numero di operazioni che definiscono un ciclo di utilizzo. Quasi si sarebbe potuto evitare di fare una homepage separata dalla pagina di editing e piuttosto farle combaciare nella stessa, come avviene per le applicazioni desktop dei sistemi di annotazione (es: VGG, Transkribus, labelMe). Tuttavia esisteva una ragione che ha potuto giustificare l'esistenza e l'indipendenza di una pagina iniziale diversa da quella in cui avviene l'annotazione: fornire una panoramica di tutte le schede disponibili per riconoscerle ed accedervi più velocemente. La homepage si presenta dunque come una lista di caroselli che mostrano un'anteprima delle schede Caronti, ricavate da una cartella del sistema. Appare, in questo senso, simile ai siti di streaming, dove al posto delle copertine dei film compaiono le immagini delle schede e i caroselli, invece che essere distinti per genere, sono distinti per stato della scheda: non annotata, annotata automaticamente, annotata manualmente (stato conclusivo). In realtà esiste una quarta categoria che occupa un ulteriore carosello, cioè quello delle schede recenti. È infatti una soluzione comune per ogni genere di applicazione fornire delle scorciatoie per accedere più velocemente ai contenuti desiderati. Di solito le categorie che vengono implementate per questo scopo sono i contenuti recenti

ed i preferiti. Nel contesto di un annotatore non aveva senso parlare di contenuti preferiti, che quindi non sono stati concepiti, ma è stato considerato invece utile mettere in bella vista quei file che durante la sessione di lavoro sono stati aperti o di cui è stato modificato lo stato di annotazione. Ciascun carosello occupa la pagina in tutta la sua larghezza ed è intestato con il nome che ne identifica la categoria (in alto a sinistra) ed una barra di ricerca locale (in alto a destra) per filtrare le schede che ne fanno parte che soddisfano l'input di ricerca. Di default, tutti i caroselli mostrano inizialmente al più 20 schede e nascondono le altre dietro un pulsante "Carica altro" che appare come ultimo elemento della galleria. Una volta cliccato, vengono aggiunte altre 20 schede (se esistono) al carosello e se ancora avanzano ulteriori immagini, il pulsante "Carica altro" si sposta nuovamente in fondo, altrimenti scompare. Quando si inseriscono dei valori nella barra di ricerca locale, il sistema verifica quali schede contengono la stringa di input nei propri metadati (id e, per le schede annotate, titolo, autore, note e collocazione) ed esclude dal carosello tutte le altre; inoltre viene eliminato il pulsante "Carica altro", affinché tutte le schede che soddisfano le condizioni di ricerca siano immediatamente visibili. Se non vi è alcuna scheda che supera il filtro, il carosello viene svuotato completamente e popolato da una semplice scritta che avvisa del problema; se il filtro viene azzerato, ricompariranno tutte le schede (eccetto quelle che prima della ricerca erano nascoste dal pulsante "Carica altro"). Le schede si distinguono le une dalle altre, oltre che per l'anteprima, dalla didascalia dell'immagine, la quale esibisce l'id della scheda e, nel caso dei recenti, anche lo stato di annotazione, che viene rappresentato con tre stringhe diverse:

- **S:0**: non annotata.
- **S:1**: annotata automaticamente.
- **S:2**: annotata manualmente.

Posando il mouse sopra queste stringhe presenti nella didascalia appare un tooltip con una legenda che informa l'utente del significato di quei tre codici. Quando si clicca su di una scheda si viene condotti alla pagina di editing popolata con i dati della scheda selezionata, contenuti in un'apposita cartella del sistema. Esiste tuttavia un'eccezione, relativa al carosello delle schede non annotate che è peculiare rispetto agli altri. Nell'intestazione infatti è presente una checkbox che, quando è disattivata, fa sì che l'interazione con le schede sia identica a quella degli altri caroselli, ma se è attiva, allora il click su di un'immagine non porta alla pagina editing, bensì seleziona la scheda. Il sistema permette una selezione multipla delle schede non annotate, anzi fornisce un pulsante, accanto alla checkbox, che seleziona automaticamente le prime dieci schede disponibili non ancora selezionate. Il loro aspetto viene modificato visibilmente per distinguerle e un secondo click comporta la deselezione della scheda. Lo scopo è quello di poter richiedere in un colpo solo l'annotazione automatica di tutte le schede scelte. Il comando avviene tramite un click sul pulsante corrispondente, che si trova in posizione centrale sotto al carosello,

accanto ad un altro che permette di deselezionare tutte le schede. Una volta inviata la domanda di annotazione automatica, gli elementi coinvolti non sono più cliccabili e assumeranno un aspetto che ne chiarisce lo stato di sospensione da qualsiasi tipo di manipolazione. Se la richiesta verrà annullata (dalla barra di ricerca), allora tutto verrà resettato come a prima dell'invio, altrimenti le schede si sposteranno dal carosello di quelle non annotate al carosello di quelle annotate automaticamente ed un duplicato verrà appeso al carosello dei recenti. A quest'ultimo vengono aggiunte anche le schede sulle quali sono state salvate modifiche nella pagina di editing.

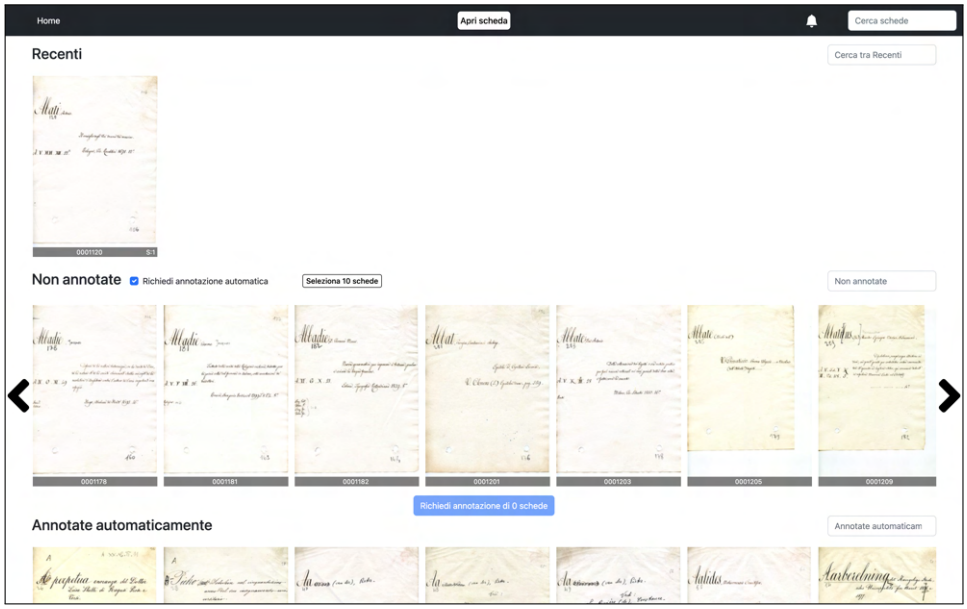


Figura 2.3: Homepage

2.3.2 Pagina di editing

La pagina dove avviene l'interazione uomo-macchina più consistente e complessa è la pagina di editing. L'obiettivo nella costruzione di quest'applicazione era evitare che l'utente dovesse costruire le segmentazioni e le trascrizioni da zero, ma che fosse invece facilitato da un'annotazione automatica. Il calcolo e la rappresentazione chiara di essa all'utente rappresenta l'elemento di intelligenza dell'interfaccia, in quanto il sistema cerca di prevedere il risultato del compito che dovrebbe svolgere l'utente e lo completa al posto suo. Ciò non significa che ci sia la pretesa di produrre un risultato perfetto, ma solo quella di migliorare l'esperienza dell'utente; l'ipotesi di base è che egli perda meno tempo a correggere un risultato intermedio piuttosto che a costruirne uno per intero. La pagina di editing ha in cima il nome del file su cui si sta lavorando, ma poi è divisa su due colonne: una per l'esposizione dell'immagine su cui disegnare le bounding box e relativi

comandi, l'altra per le trascrizioni. Entrambe le sezioni sono popolate dalle annotazioni automatiche, se sono state richieste.

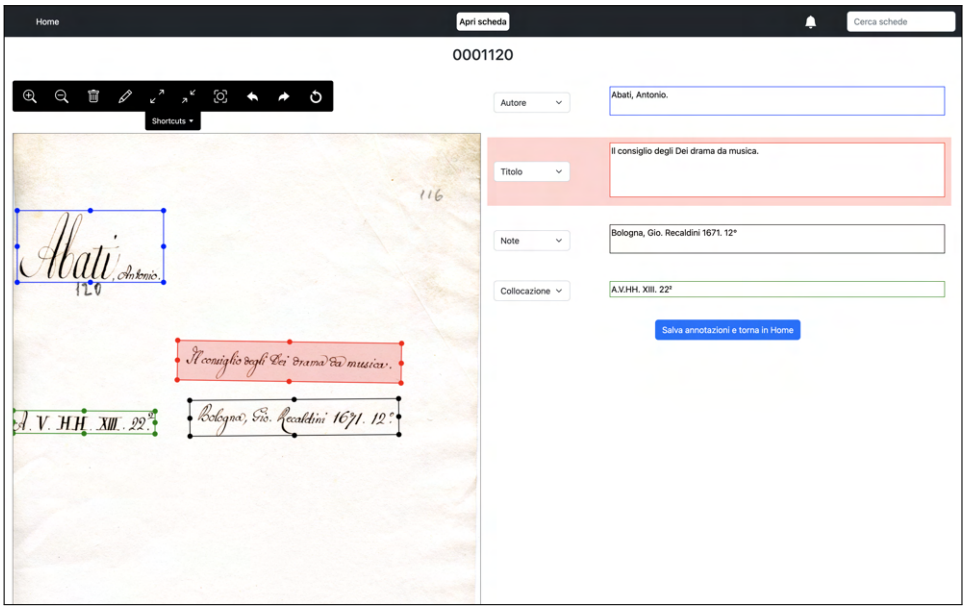


Figura 2.4: Pagina di editing

2.3.2.1 Manipolazione immagine

Questa sezione della pagina è dedicata alla segmentazione e quindi serve alla visualizzazione e correzione delle bounding box. Gli elementi che la popolano sono solo due: la barra dei comandi e l'immagine stessa. Lo scopo è quello di disegnare o correggere i poligoni attorno a ciascuna delle diverse regioni di testo presenti sulla scheda, che possono essere al più quattro: titolo, autore, note di catalogazione e collocazione. Il sistema impedisce di aggiungere ulteriori annotazioni e associa in maniera inevitabile a ciascun poligono l'etichetta cui corrisponde. La barra dei comandi compare sopra l'immagine ed è ben visibile, popolata dalle icone che corrispondono ciascuna ad una specifica azione. Le icone sono autoesplicative, ma, in ogni caso, quando il cursore del mouse posa su di esse viene mostrata anche una scritta che ne esplicita il significato (funzione standard di tutti i browser). Le azioni disponibili sono:

1. Zoom avanti
2. Zoom indietro
3. Accentramento immagine
4. Cancellazione bounding box

5. Disegno bounding box
6. Rimpicciolimento bounding box
7. Ingrandimento bounding box
8. Undo
9. Redo
10. Reset

Cliccare sulle icone tuttavia non è l'unico modo di innescare le azioni associate, bensì sono previste shortcut (scorciatoie) da tastiera, per permettere ad un utente abituale di operare più rapidamente, senza dover spostare il mouse sulla barra dei comandi. Non è necessario parlare di utente esperto, in quanto i comandi sono davvero pochi e per di più facilmente visibili. La barra infatti contiene una linguetta che la espande verticalmente e mostra sotto ciascun'icona il pulsante della tastiera associato e se l'utente lo desidera, può costantemente lasciare aperta questa guida. Quando l'utente cerca di eseguire un'azione che non è legale o impossibile in quel momento, compare un messaggio di colore rosso sopra le icone che avvisa l'utente che il comando non è avvenuto con successo e lo istruisce su cosa fare per evitare nuovamente l'errore. Sotto la barra dei comandi compare invece l'immagine della scheda. La dimensione massima di essa dipende da quella della finestra del browser: la larghezza è metà pagina, meno un po' di margine dai bordi, mentre l'altezza è determinata dal rapporto originale delle dimensioni della foto, tale per cui la proporzione viene mantenuta. L'area in cui compare l'immagine è colorata diversamente dallo sfondo del resto della pagina, in modo che i margini entro cui si può editare siano ben distinguibili; entro i suoi confini, si può muovere la foto come si preferisce. L'immagine che viene mostrata è quella originale, con però sovrapposti i disegni delle bounding box, che sono interattive e colorate in base alla categoria del testo che contengono (ogni categoria è infatti associata ad un determinato colore). È stato scelto di considerare i quadrilateri l'unica tipologia di poligono disponibile per le bounding box, allo scopo di rendere l'operazione di segmentazione la più veloce possibile. Esse si possono manipolare attraverso i comandi già descritti sopra, ma con il mouse sono disponibili ulteriori azioni:

1. Selezione della bounding box
2. Spostamento dell'intero poligono
3. Spostamento di un singolo angolo
4. Spostamento di due angoli adiacenti (spostamento di un lato)

2.3.2.2 Trascrizioni

La colonna di destra della pagina di editing serve a visualizzare e modificare le trascrizioni. Quando la pagina viene aperta, si verifica se esistono già delle annotazioni per la scheda che si è selezionata e, in caso negativo, apparirà nella sezione delle trascrizioni come primo elemento un pulsante per richiedere la segmentazione automatica (trascrizione inclusa). Questo pulsante compare anche quando, durante l'editing, vengono cancellate tutte le bounding box, ipotizzando che l'utente abbia deciso di ricostruire dal principio tutta l'annotazione. Se invece una qualche segmentazione esiste, per ciascuna bounding box sarà presente un elemento (*select*) che ne indica la categoria (autore, titolo, etc.) ed a fianco un'area di testo con la trascrizione corrispondente a quella regione; entrambi gli elementi sono modificabili. Quando una categoria viene cambiata, se già ne esiste un'altra corrispondente al nuovo valore scelto, allora le etichette si invertono, ma le aree di testo rimangono immutate. Inoltre, ogni volta che si clicca su un'area di testo o su una select è necessario rendere evidente qual è la bounding box associata a tale trascrizione. Per questa ragione il click attiva la colorazione sia dello sfondo dell'input di testo sia dell'area interna del poligono, che però in quel caso è in semi trasparenza per non coprire il testo dell'immagine. Lo stesso vale quando si clicca su di una bounding box, per evidenziare sempre il collegamento tra la regione di testo e la trascrizione. La coppia select-trascrizione viene eliminata quando la sua bounding box è cancellata. L'ultimo elemento della colonna è il pulsante per salvare le annotazioni, che reindirizza in home se l'operazione viene completata con successo. Essa consiste nella generazione di nuove immagini, una per ciascuna bounding box, e di un file JSON contenente sia le coordinate di ciascun poligono, sia le trascrizione del testo corrispondenti. Poi, lo stato della scheda annotata correttamente nella pagina di editing viene modificato da quello di partenza ad "annotata manualmente". Esistono però un paio di circostanze in cui il salvataggio non viene permesso: quando un'area di testo è stata lasciata vuota, il che significa che esiste la bounding box ma che è inutile, oppure quando non c'è alcuna annotazione. Come sempre, quando avviene un comportamento inaspettato, è necessario che compaia un feedback evidente e quindi in questi casi appare un toast di colore rosso nell'angolo in basso a destra dello schermo, vicino a dove già si trova il pulsante di salvataggio che si è appena premuto. Il toast contiene del testo che esplicita l'errore e il modo per risolverlo, poi dopo 4 secondi scompare.

2.3.3 Pagina di ricerca

Sono già state citate le barre di ricerca locale, che aggiornano il contenuto dei caroselli corrispondenti. Nella barra di navigazione esiste un'ulteriore barra di ricerca che è globale. Quando vi si scrive, si è reindirizzati alla pagina di ricerca, che è costituita da un unico carosello con tutte le schede disponibili che includono nei propri metadati la stringa inserita. Ad ogni singola modifica dell'input, il contenuto del carosello si aggiorna

immediatamente. Se non esistono schede che soddisfano il filtro, nel carosello apparirà solamente una scritta che avvisa l'utente che non ci sono risultati per quella ricerca. Come accade nel carosello dei recenti, anche in quello di questa pagina ciascuna scheda riporta nella didascalia, oltre all'id, una stringa che ne indica lo stato, fornita di un tooltip che informa l'utente del significato di tale stringa. Come l'inserimento di un input nella barra di ricerca locale elimina il pulsante "Carica altro" e mostra direttamente tutte le schede che soddisfano il filtro, allo stesso modo il carosello della ricerca globale non fa uso di tale pulsante e popola subito la galleria con tutte le schede che rispettano la condizione di ricerca. Il click su una qualsiasi scheda, a prescindere da quale sia il suo stato, reindirizzerà l'utente alla pagina di editing per la tale scheda.

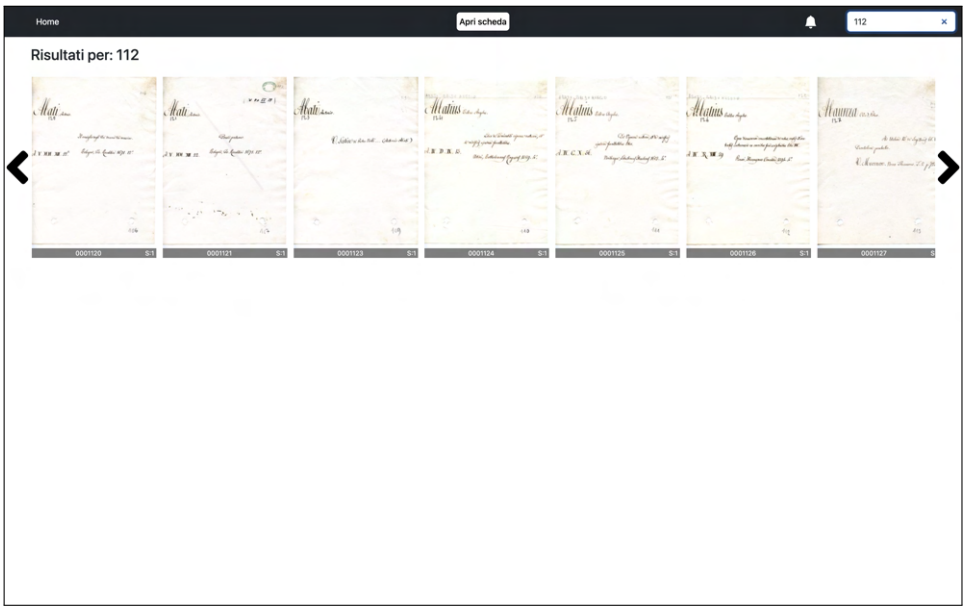


Figura 2.5: Pagina di ricerca

2.3.4 Barra di navigazione

La barra di navigazione è un elemento presente in tutte le pagine. Come di consueto, occupa la parte più alta della finestra per tutta la sua lunghezza ed è di un colore differente rispetto allo sfondo del resto della pagina. È popolata da quattro elementi (da sinistra verso destra): link alla homepage, pulsante di caricamento immagine, icona delle notifiche e barra di ricerca globale. Il link alla home è una presenza standard in ogni applicazione web, quasi imprescindibile all'interno di una barra di navigazione, quindi anche in questa compare e la sua funzione è quella di reindirizzare l'utente in homepage. In posizione centrale della barra è presente un pulsante che permette di selezionare un file dal proprio computer e caricarlo sull'applicazione, in modo da poterlo annotare. Questo pulsante è stato previsto in quanto l'interfaccia conosce e mostra solo le immagini presenti

in una certa cartella del sistema, dove si richiede agli utenti di aggiungere eventualmente quelle mancanti. Tuttavia, potrebbe capitare che ogni tanto sia più comodo e veloce caricare un'immagine estranea all'applicazione attraverso questa via. La campanella di notifica serve come sistema di feedback all'utente per aggiornarlo rispetto allo stato di una scheda di cui si è richiesta l'annotazione automatica. L'operazione di segmentazione automatica infatti ha bisogno di tempo (variabile a seconda della velocità di connessione internet) e potrebbe anche fallire. L'utente, che quando invia la richiesta potrebbe aspettarsi un risultato immediato, deve invece essere informato che l'operazione è stata presa in carico ed è in corso d'opera e non vedrà degli aggiornamenti finché non sarà stata terminata, con successo o meno. Per questo motivo, al momento della richiesta, l'icona della campanella nella barra di navigazione viene decorata con un badge rosso che le fa risaltare; l'utente che a quel punto muove il cursore del mouse su di essa, vedrà apparire una finestrella in sovrimpressione in cui vi è l'elenco di tutte le segmentazioni automatiche che sono state domandate al sistema nella sessione corrente. Ogni riga di questa lista corrisponde ad una scheda e può avere tre layout diversi:

- **Segmentazione in corso**. La riga è composta da uno spinner, il nome della scheda in grassetto, la scritta "Annotazione in corso..." e un pulsante di chiusura, il quale, se premuto, interrompe la richiesta.
- **Segmentazione fallita**. La riga è composta dall'icona di una x rossa, il nome della scheda in grassetto e la scritta "Annotazione autom. fallita".
- **Segmentazione eseguita**. La riga è composta dall'icona di un segno di spunta blu, il nome della scheda in grassetto e la scritta "Annotazione autom. fatta". Il click sul nome conduce alla pagina di editing per quella scheda.

La finestrella delle notifiche ha un'intestazione con scritto "Notifiche", la scritta "Nessuna notifica" oppure l'elenco citato sopra, poi in fondo ha un pulsante "Interrompi annotazioni" per abortire tutte le segmentazioni automatiche in corso. Una volta che si posa il mouse sopra la campanella fornita di badge, il badge scompare; ricomparirà ogni volta che una richiesta in sospeso sarà terminata o saranno fatte nuove domande di annotazione di schede. La finestrella stessa viene nascosta non appena il cursore del mouse esce dai suoi bordi o da quelli della barra di navigazione. Infine, sul margine destro è presente la barra di ricerca globale. Ad ogni caricamento di pagina il suo input viene pulito, non appena però viene digitato qualcosa al suo interno, si è reindirizzati alla pagina di ricerca, che si comporta come descritto nel paragrafo precedente. Quando invece l'input viene cancellato si ritorna alla pagina che era aperta in precedenza.

2.4 Back-end di tonicarD

Uno dei due obiettivi del progetto era costruire un'interfaccia di annotazione, l'altro invece era realizzare una linea di partenza per la trascrizione delle schede Caronti, cioè sviluppare un algoritmo che proponesse una prima, seppur approssimativa, automatica segmentazione e trascrizione del testo presente nell'immagine. Il back-end è quasi esclusivamente dedicato a questo scopo, per il quale è stato di fondamentale aiuto l'API di Google chiamata Cloud Vision. Quando l'utente richiede l'annotazione automatica di una scheda, viene chiamata una funzione lato server, che, per prima cosa, interroga Cloud Vision affinché fornisca l'annotazione dell'immagine in input (ricevuta dal client). Subito dopo, se l'operazione è andata a buon fine, scatta l'algoritmo di correzione delle segmentazioni. Si tratta di una serie di istruzioni costruite ad hoc sulla base degli errori più comuni delle annotazioni automatiche di Google, affinché la segmentazione sia ad un livello sufficientemente buono da poter essere proposta all'utente come valido tentativo di annotazione automatica. Quasi sempre, infatti, il risultato restituito da Cloud Vision è di modesta qualità e, in ogni caso, è stato deciso di modificarlo in modo tale che vi fosse una segmentazione per ciascun differente blocco di testo (autore, titolo, note, collocazione).

2.4.1 Cloud Vision API

Cloud Vision opera il rilevamento della scrittura a mano libera nelle immagini. La scelta è ricaduta su tale strumento dapprima in quanto servizio API (di cui sono sprovvisti tool come Transkribus ed eScriptorium), poi per merito della sua accessibilità e facilità d'uso. Si tratta, infatti, di un servizio parzialmente gratuito, che per le esigenze del progetto non ha richiesto alcuna spesa, e fornito di una buona documentazione. Le condizioni di utilizzo sono le seguenti [6]:

1. Creazione di un progetto Google Cloud.
2. Attivazione della fatturazione del progetto.
3. Attivazione dell'API.
4. Creazione di un account di servizio.
5. Creazione chiave dell'account di servizio.

Sul sito, sono pubblicate le istruzioni per usare l'API da svariati ambienti back-end: Go, Java, Node.js, Python, C#, PHP e Ruby. Esse consistono, nel caso di Python, di poche semplici operazioni: si installa la libreria google-cloud-vision nell'ambiente di sviluppo,

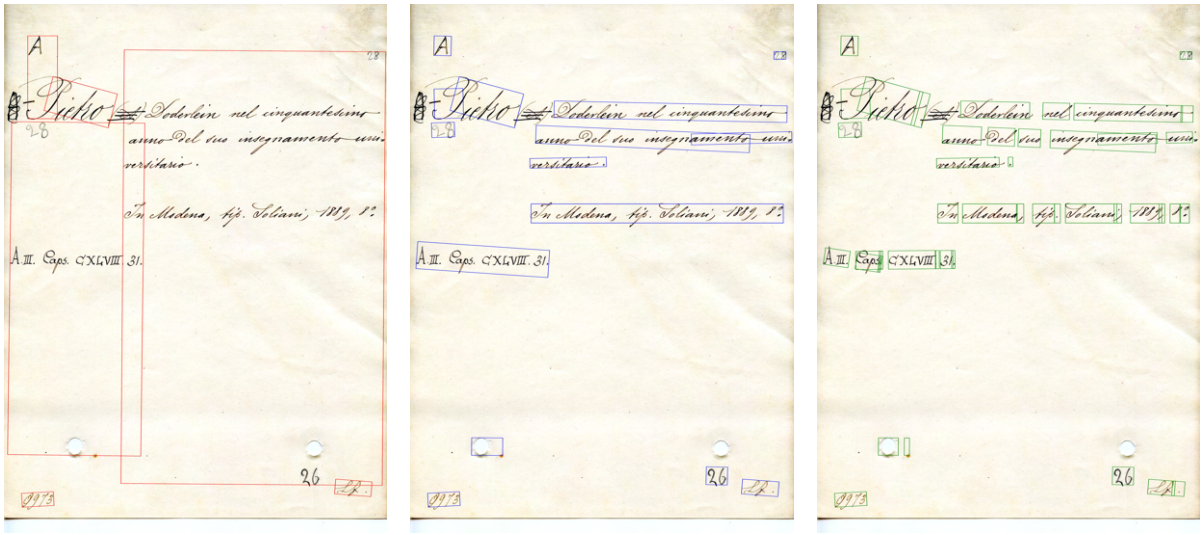
[6] Istruzioni per l'uso dei servizi Google Cloud, https://cloud.google.com/vision/docs/detect-labels-image-client-libraries, data di ultima consultazione 1 dicembre 2022

si importa tale libreria nel codice, poi si lancia la richiesta di annotazione; se questa termina con successo, viene restituito un file JSON piuttosto ricco di dati, seppur un po' intricati. Il servizio, infatti, esegue quattro livelli di annotazione diversi, che nel file sono annidati l'uno con l'altro.

```
{
  "textAnnotations": [...],
  "fullTextAnnotation": {
    "pages": [{
        "blocks": [{
            "boundingBox": {...},
            "paragraphs": [{
                "boundingBox": {...},
                "words": [{
                  "boundingBox": {...},
                  "symbols": [{
                      "boundingBox": {...},
                      "text": ""
                    }]
                }]
              }]
          }]
      }]
  }
}
```

Listing 2.1: Struttura semplificata del file JSON di Cloud Vision API con le informazioni di annotazione dell'immagine.

I dati utili per l'applicazione sono tutti sotto la chiave *fullTextAnnotation*, che dapprima suddivide le annotazioni per pagina (sempre e solo una nel caso delle schede Caronti), poi per blocchi, per righe, per parole e infine per simboli/caratteri. Iterando su questi livelli di annidamento si può ricostruire l'insieme delle parole che compongono le righe ed i blocchi. Inoltre, per ciascuno sono indicati i vertici delle bounding box, in modo da poterle disegnare sull'immagine.



(a) Segmentazione per blocchi (b) Segmentazione per righe (c) Segmentazione per parole

Figura 2.6: I tre livelli principali di annotazione di Cloud Vision API

Come si può notare in figura 2.6, la qualità delle segmentazioni sulle schede Caronti è variabile a seconda del livello di granularità del blocco che identifica (blocco, linea, parola). Tra queste, quella con il risultato più consistente ed affidabile sembra essere quella della riga (chiave *paragraphs* del file JSON).
Infine, per quanto riguarda la tecnologia che si nasconde dietro il modello di annotazione automatica di Cloud Vision, si può dire ben poco, se non niente, poiché Google non fornisce alcun tipo di documentazione in merito. L'unica informazione che rivelano è che l'API, per il compito di classificazione di immagini, sfrutta modelli di machine learning quali reti neurali convoluzionali e ricorrenti. Specificatamente per la segmentazione, ci si può immaginare che usi tecnologie allo stato dell'arte, che fanno analisi dell'immagine, per le quali il machine learning non è necessario.

2.4.2 Algoritmo di miglioramento della segmentazione

Si è mostrato nella sezione precedente come Cloud Vision API venga usata per calcolare un'approssimazione dell'annotazione della scheda, che costituisce la base di partenza su cui costruire il risultato automatico che viene proposto all'utente nell'interfaccia. Dei tre tipi di regione di testo che il servizio di Google cerca di identificare, quello che funziona meglio sulle immagini di schede Caronti è il livello di riga. Per questa ragione si è deciso di sfruttare la segmentazione di tale tipo e scartare le altre. Tuttavia, i risultati sono ancora scarsi e poco affidabili; gli errori che si rilevano sono:

- Falsi positivi. Segmentazioni di aree in cui non c'è testo.

- Margini non sufficientemente alti per racchiudere tutte le lettere, soprattutto con i nomi d'autore che sono scritti in grande rispetto al resto.
- Segmentazioni sovrapposte, che si suddividono per:
  - segmentazioni annidate totalmente, o quasi, in un'altra. Di solito catturano una sola parola di una riga già contornata da una box.
  - segmentazioni sovrapposte ai margini. Il margine di una box sfocia nel margine di un'altra box e viceversa (margini destra-sinistra, alto-basso o entrambi). L'area di intersezione tra le box a volte contiene testo, altre volte no. Si possono verificare catene di bounding box a contatto l'una con l'altra.
- Segmentazione di una riga spezzata in multiple bounding box.
- Falsi negativi, ossia del testo non è catturato da alcuna bounding box. Capita molto più raramente rispetto agli altri errori.

Vi sono poi evidenti errori di trascrizioni come conseguenza diretta di una segmentazione sbagliata o della presenza di errata corrige. Le lingue che il sistema riconosce possono essere le più disparate: islandese, gallese, ilocano, indonesiano, ecc. Ma anche lingue di altri alfabeti, come quelli cirillici: russo, macedone, serbo, bulgaro, ecc. Il catalogo della Biblioteca non contiene delle scritte in simili lingue e quando capitano questi casi, le trascrizioni sono da considerarsi quasi sempre sbagliate. Al contrario, se la segmentazione è fatta sufficientemente bene e la scheda non presenta imperfezioni dovute alle correzioni, spesso la trascrizione risulta buona. L'immagine portata ad esempio (Fig. 2.6b) evidenzia che le annotazioni fornite da Google non possono essere presentate all'utente così come arrivano, ma necessitano di essere sistemate. L'algoritmo costruito a tale scopo, è stato progettato in base all'osservazione di un campione di circa 20 schede, di cui si sono studiati gli errori nelle segmentazioni automatiche e sulle quali si sono testati i risultati della procedura.

Inizializzazione bounding box
Dapprima, si creano degli oggetti *box* che corrispondono alle bounding box calcolate da Cloud Vision API. I metadati di ciascuna sono:

- Id
- Forma
- Etichetta
- Testo
- Lista di box connesse

A questo punto del codice, l'etichetta e la lista di box connesse non sono ancora state inizializzate, mentre la forma è un oggetto *Polygon* costruito sulla base dei vertici della bounding box data da Google e il testo è la concatenazione delle parole che vi appartengono.

Eliminazione bounding box
Poi, allo scopo di eliminare possibili falsi positivi, vengono cancellate tutte le box il cui testo associato è costituito al più da tre simboli di cui nessuno è un carattere. Infatti si è osservato che in questo insieme ricadono le annotazioni di fori della scheda e di numerelli aggiunti a mano negli anni dai bibliotecari che non si ha interesse a segmentare.

Identificazione blocco della collocazione
Nonostante il testo sia sparpagliato all'interno del foglio, è facile accorgersi della presenza di uno schema nella disposizione delle differenti etichette. C'è un layout comune alla maggior parte delle schede, tale per cui si possono fare delle assunzioni sulla categoria cui ciascun blocco fa parte. Dal momento che il testo della collocazione è peculiare e si distingue dal resto, è possibile avanzare assunzioni ancora più forti sulla base delle quali stabilire se una box identifica la collocazione. Il motivo per cui si cerca subito di individuare questo blocco è che ogni tanto esso viene agglomerato a quello di un altro più a destra (titolo o note) perché sembrano stare sulla stessa riga. Prima di manipolare le box a seconda delle relazioni in cui stanno tra loro, diventa cruciale separare in anticipo quella di collocazione dalle altre. Di fatto, significa modificarne una e crearne un'altra. Se si verifica questo caso, ossia di un'unica box che attraversa sia la zona della collocazione (vicino al margine sinistro) sia quella di titolo o note (vicino al margine destro), allora si spezza il blocco lungo un asse calcolato sulla base delle osservazioni del campione di studio. A questo punto si cerca di identificare il testo della collocazione, che molto spesso è riconducibile a due specifiche regex. Se una di esse corrisponde alla prima parte della trascrizione del blocco originale, allora si associa quella parte di testo alla box collocazione e la si cancella invece dalla trascrizione della box rimasta a destra.

Costruzione grafo delle bounding box connesse
Il passo successivo consiste nel determinare quali box si intersecano fra loro ed in quale ordine; il primo punto è banale, il secondo meno. Come prima cosa, si costruiscono tutte le possibili coppie di box differenti, poi, per ciascuna coppia, si esegue la seguente procedura:

1. Se una delle box è stata da considerarsi da cancellare, si passa alla coppia successiva.
2. Se le box non si incrociano, si passa alla coppia successiva.
3. Se una delle box è contenuta per il 98% della propria area entro quella dell'altra, allora la si inserisce nella lista di box da cancellare e si passa alla coppia successiva.

4. Se entrambe le box fanno parte dello stesso grafo, allora si passa alla coppia successiva.

5. Se una delle due box fa già parte di un grafo, si inserisce l'altra all'interno dello stesso grafo.

6. Se nessuna delle due box fa già parte di un grafo, allora si definisce un nuovo grafo e vi si inseriscono entrambe le box.

Per grafo si intende un insieme di box connesse l'una all'altra in base al fatto che si intersecano. Come scritto sopra, ogni box contiene fra i propri metadati la lista delle box cui, direttamente o indirettamente, è connessa. La parte più complicata di questa fase è determinare quale box viene prima secondo l'ordine di lettura. Poiché infatti le box appartenenti allo stesso grafo sono destinate a riunirsi sotto un'unica bounding box, è fondamentale comprendere in quale ordine vanno collegate, in modo da ricostruire correttamente il testo del nuovo blocco. Il metodo realizzato per decretare tale ordine si basa su un sistema di vittorie, dove ogni box viene confrontata con tutte le altre del grafo e vince quando è quella della coppia che va letta per prima. Il criterio in base al quale un blocco vince rispetto ad un altro è il seguente:

1. Si identifica il blocco più a sinistra, di cui si considera il segmento superiore (quello con le coordinate delle ordinate più basse) e il blocco più a destra, di cui si considera il segmento laterale sinistro (quello con le coordinate delle ascisse più basse). Chiamiamo il primo *topSide* e il secondo *leftSide*.

2. Si individua un valore Y che è l'ordinata del punto più alto in cui *topSide* e *leftSide* si intersecano, se si intersecano. Altrimenti Y è uguale all'ordinata del punto più alto di *topSide*.

3. Si confronta Y con l'ordinata del punto medio di *leftSide*: se Y è più piccolo (e cioè più in alto), allora vince la box di sinistra, altrimenti vince quella di destra.

Con questo approccio non c'è bisogno di preoccuparsi di eventuali pareggi tra box, perché sono impossibili, infatti, le vittorie di ciascuna box saranno sempre una in meno rispetto a quella alla propria sinistra ed una in più rispetto quella a destra. La box con più vittorie sarà la prima della catena e quella che ne ha meno sarà l'ultima.

Unione bounding box connesse

Il passo successivo dell'algoritmo consiste nel filtrare ulteriormente le box, calcolare i vertici della bounding box risultante dall'unione di ciascun grafo e determinare il testo della box appena creata. Si sottolinea che la trascrizione sarà un'approssimazione un po' grezza, che sì migliorerà il risultato rispetto alla banale concatenazione delle scritte

dei blocchi che si agglomerano, ma con certi limiti. Innanzitutto, si verifica se ci sono ancora box da eliminare. La condizione perché questo accada è che una box, una volta considerata solo l'area che non condivide con altre box con cui si sovrappone, si presuma non contenga alcun testo. Ecco come viene fatta questa verifica:

1. La box viene privata di tutti i punti che condivide con altre box, si ridefinisce il nuovo poligono risultante e si ritaglia una nuova immagine lungo i suoi bordi.

2. Si contano tutti i pixel della nuova immagine con un valore alfa uguale ad 1 e si raggruppano per colore, ottenendo una lista di coppie $\langle numero\,di\,pixel, valore\,rgba \rangle$.

3. Per ciascun colore si verifica quanto esso sia scuro e se lo è a sufficienza, si presume che i pixel di tale colore rappresentino l'inchiostro che forma lettere sulla scheda (la quale ha sempre uno sfondo chiaro, di sfumature di bianco o giallo). Si sommano quindi tutti i pixel dei colori che soddisfano questa condizione e se ne calcola la percentuale rispetto al numero di pixel totali dell'immagine.

4. Se la percentuale di pixel scuri supera una certa soglia, allora si assume che la box contenga del testo che solo lei cattura. In caso contrario, la box va eliminata in quanto, se contiene del testo, questo è già catturato da un'altra box.

Si fa notare che al punto 1, potrebbe non essere uno solo il poligono generato dall'operazione di differenza fra poligoni; in tale evenienza, si applica il resto della procedura a tutti i nuovi poligoni e se nessuno di essi contiene del testo, allora la box originale viene cancellata. Inoltre, è importante specificare che i valori soglia per definire un colore scuro o per determinare che un'immagine contenga del testo, sono arbitrari e scelti in base ad un'analisi effettuata sul campione di testing. Nello specifico, affinché un colore venga ritenuto scuro, la somma dei suoi valori RGB non deve superare 140 (quella del nero è pari a 0 e quello del bianco è 768); il numero di pixel che soddisfano questa condizione deve essere almeno il 3% dell'immagine totale perché si possa presumere che il poligono contenga del testo, che altrimenti non verrebbe catturato. Per la concatenazione delle parole tra due box direttamente connesse, si procede come segue:

1. Se entrambe le box contengono più di una parola ciascuna, si cancella la prima parola della box più a destra.

2. Se entrambe le box contengono solo una parola, si cancella la parola della box con l'area più piccola tra le due.

3. Se una box contiene più di una parola e l'altra solo una, si elimina il testo di quest'ultima.

Infine, bisogna creare la bounding box che riunisca al proprio interno tutte le parole delle box appartenenti allo stesso grafo. Tuttavia, la soluzione non può essere quella ingenua

di costruire i contorni prendendo le coordinate più estreme fra tutti gli angoli di tutte le box coinvolte, ma bisogna trova i quattro vertici che contengano tutte le parole di tutte le box e che allo tesso tempo riducano il più possibile l'area del poligono che definiscono. Altrimenti si potrebbe correre il rischio di creare delle segmentazioni molto grandi che sforano nell'area di una diversa bounding box. La soluzione implementata è questa:

1. Si calcolano i quattro vertici più ai margini fra tutti i vertici di tutte le box dello stesso grafo (il più a sinistra/destra/alto/basso).

2. Se nessuno di questo quattro vertici è contemporaneamente un estremo sia per le ascisse che per le ordinate, allora la bounding box coincide con il quadrilatero che ha essi per vertici.

3. Se uno dei vertici è un angolo del possibile nuovo quadrilatero, allora si controlla quali sono i segmenti adiacenti più convenienti da disegnare. Ad esempio: un vertice è sia quello più a sinistra sia il più alto rispetto agli altri due, quindi si può definirlo l'angolo in alto a sinistra. Nel tentativo di congiungere quell'angolo con il vertice in alto a destra (scelto fra i due vertici più a destra di tutte le bounding box), bisogna assicurarsi che une retta fra i due punti non intersechi alcun altro punto di ciascuna box del grafo; se così fosse, quel lato taglierebbe una parte di testo. Quindi, nel caso in cui ciò non accade il segmento è la linea più diretta fra l'angolo ed il vertice opposto, altrimenti viene tracciato un segmento parallelo all'asse (l'asse delle ascisse nell'esempio) che termina sul prolungamento del più lontano dei vertici opposti.

Identificazione delle etichette delle bounding box
L'ultima fase dell'algoritmo di segmentazione consiste nella classificazione delle bounding box e, nel caso che più di una appartenga alla stessa etichetta, si riapplica la procedura di unificazione tra box. Sulla base di osservazioni fatte sul campione di lavoro, si è definita l'area dell'immagine entro cui ci si aspetta appaiono le scritte che si vogliono trascrivere ed al di fuori della quale non si considera alcuna parola. Inoltre, come già riferito in precedenza, sono state fatte assunzioni sulle regioni della scheda che ciascuna etichetta occupa, che servono come base di partenza per la classificazione. In ordine, le operazioni sono:

1. Si eliminano le box che cadono al di fuori del poligono che definisce l'area dell'immagine dove si cerca il contenuto della trascrizione.

2. Per ciascun quadrilatero si calcola la sua distanza con il centroide di ogni regione associata alle varie etichette e si genera una lista di tutte le categorie ordinate per distanza crescente. La prima posizione corrisponde alla classificazione più probabile e l'ultima alla meno probabile.

3. Per ciascuna box, si verifica se la regione dell'etichetta più vicina include completamente la superficie del quadrilatero. In caso contrario, si ripete questo passaggio per l'etichetta immediatamente successiva per distanza. Se non si ha successo per nessuna categoria, allora si scarta la box, altrimenti le si assegna temporaneamente l'etichetta che per prima ha soddisfatto questa condizione. Si tiene traccia di tutte le box assegnate ad una certa categoria in un'apposita lista, una per ciascun etichetta.

4. Se nessuna box è stata classificata come *Titolo*, allora si cerca il titolo fra le note. Si suppone che una scheda abbia sempre almeno il titolo (d'altronde essa cataloga un qualche libro) e che se non è stato trovato, allora probabilmente sia stato erroneamente etichettato come *Note*, vista la vicinanza fisica tra le due regioni. Se esistono più di una box categorizzate tra le *Note*, allora si identifica qual è la prima in ordine di lettura e le si cambia etichetta da *Note* a *Titolo*. In qualsiasi altro caso, si lascia all'utente il compito di correggere la segmentazione del *Titolo*.

5. Per ogni categoria, si stabilisce quale box fra tutte più probabilmente delle altre vi appartiene:

   (a) **Collocazione**. La box è stata identificata in una fase iniziale dell'algoritmo e non viene modificata (anche se mancante).

   (b) **Autore**. La prima box in ordine di lettura cui è stata assegnata questa etichetta.

   (c) **Note**. L'ultima box in ordine di lettura cui è stata assegnata questa etichetta.

   (d) **Titolo**. Se esiste una box in *Autore*, allora si sceglie la box con etichetta *Titolo* più vicina al centroide della regione *Titolo*. In caso contrario, si sceglie la prima box in ordine di lettura cui è stata assegnata questa etichetta.

   Tutte le altre box devono essere riassegnate.

6. Si inserisce in una lista ogni box che non è stata identificata al punto 5, ordinata dalla box che si trova più in alto a quella più in basso. Poi, per ciascuna si calcola la distanza verticale che ha dalle box che hanno già un'assegnazione e le si associa la stessa categoria del blocco più vicino.

7. Per ogni etichetta con più di una box, si costruisce un'unica bounding box secondo la stessa procedura presentata nel paragrafo precedente.

L'algoritmo di segmentazione automatico è così terminato. Come ultima operazione, il back-end genera in un'apposita cartella del sistema un file JSON con tutte le informazioni dell'annotazione, così che l'interfaccia sia in grado di disegnare le bounding box sull'immagine originale della scheda e di recuperare la trascrizione associata a ciascuna segmentazione.

2.5 Efficienza e qualità dell'annotazione automatica

Come già è stato riportato, la qualità della segmentazione automatica delle schede Caronti eseguita da Cloud Vision non è affidabile, in quanto sono frequenti sovrapposizioni di bounding box che dovrebbero essere totalmente separate o riunite in una unica, oppure perché i margini sono troppo bassi o perché alcune bounding box non dovrebbero neanche esistere poiché non identificano del testo. In aggiunta a questi errori, Cloud Vision identifica delle righe di testo (anche se non esclusivamente), mentre per lo scopo di tonicarD si desiderava una segmentazione per blocchi di testo, uno per ciascuna delle quattro possibili categorie che le schede Caronti possono manifestare: autore, titolo, note, collocazione. Si è costruito quindi un algoritmo che, oltre a risolvere i problemi dell'output dell'API di Google, fosse in grado di costruire nuove bounding box e di classificarle secondo le etichette prestabilite. Nell'immagine 2.7 si può notare come a partire dalle segmentazioni di Cloud Vision API (prima riga) una volta eseguito l'algoritmo di correzione si ottengono delle segmentazioni migliori (seconda riga), più vicine alla classificazione desiderata (terza riga).

Per ottenere dei dati su cui basare una valutazione dell'algoritmo di segmentazione automatica di tonicarD, sono state eseguite 100 annotazioni manuali di schede. Esse hanno rappresentato il termine di paragone su cui misurare la qualità dei due algoritmi automatici, i cui risultati sono stati considerati tanto più corretti quanto più coincidenti con i risultati della segmentazione manuale. È fondamentale, però, sottolineare che tonicarD implementa un sistema di segmentazione specializzato secondo una logica propria, con vincoli arbitrari, ossia che le segmentazioni sono per regioni di testo e che possono essere al più quattro. L'algoritmo di miglioramento dei risultati di Cloud Vision è stato costruito proprio al fine di ottenere un'annotazione che soddisfacesse le condizioni di segmentazione previste dal sistema, mentre il modello di Google ovviamente no. Anzi, segmentare aree di testo, come vuole fare tonicarD, è di per sé un metodo meno preciso rispetto a segmentare per righe, come invece tenta di fare Cloud Vision. Questo perché quando si disegna un poligono attorno a multiple righe, inevitabilmente la bounding box cattura più spazio vuoto (e quindi inutile) di una che si limita a circondare una singola linea di testo. Per questa ragione, se Cloud Vision eseguisse una perfetta annotazione per righe, secondo la valutazione che è stata adottata, che è contestualizzata nell'ambiente di tonicarD, molto probabilmente tale annotazione risulterebbe comunque peggiore rispetto a quella dell'algoritmo che ne usa l'output e lo corregge; tuttavia non è una circostanza verosimile, quindi ci può permette di mettere i sistemi a confronto. Una volta fatta questa importante precisazione, si può passare alla descrizione di com'è stata calcolata la qualità dei due algoritmi di segmentazione automatici. Per ogni scheda, si sono dapprima considerati tutti i poligoni disegnati dall'algoritmo automatico, di cui si è calcolata la quantità di area che non intersecava alcun poligono della segmentazione manuale; si è poi applicato il procedimento inverso, calcolando quanto delle bounding box costruite a mano non si sovrapponeva ad alcuna di quelle calcolate automaticamente. Sia x la

somma delle aree calcolate nel primo passaggio, sia y la somma di quelle calcolate nel secondo e sia $totArea$ la somma totale delle aree di tutti i poligoni di entrambe le categorie: la precisione degli algoritmi automatici rispetto ad ogni singola scheda è stata calcolata come $totArea - (x + y)$, successivamente trasformata in percentuale. Sulla base di queste misure, la qualità è stata quantificata attraverso la deviazione standard, costruita sulle 100 schede analizzate:

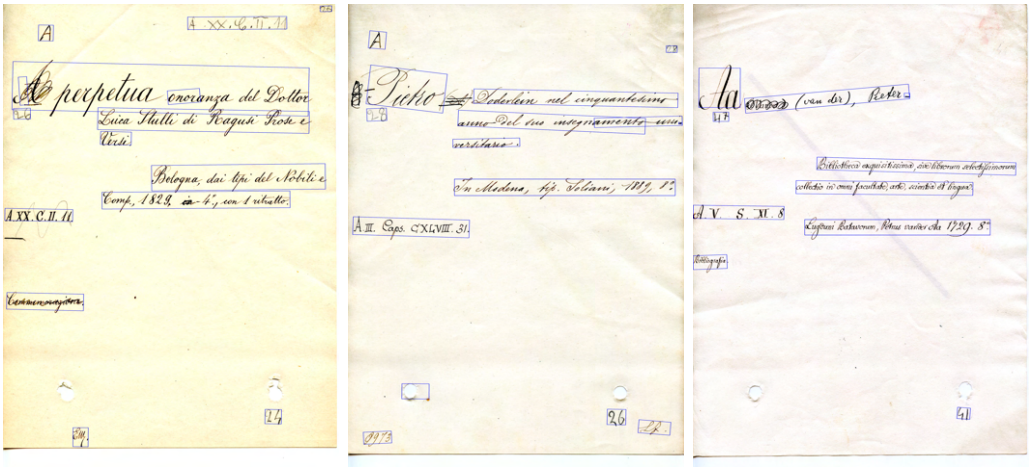
$$\sigma_{google} = 7.65\%$$

$$\sigma_{tonicarD} = 20.03\%$$

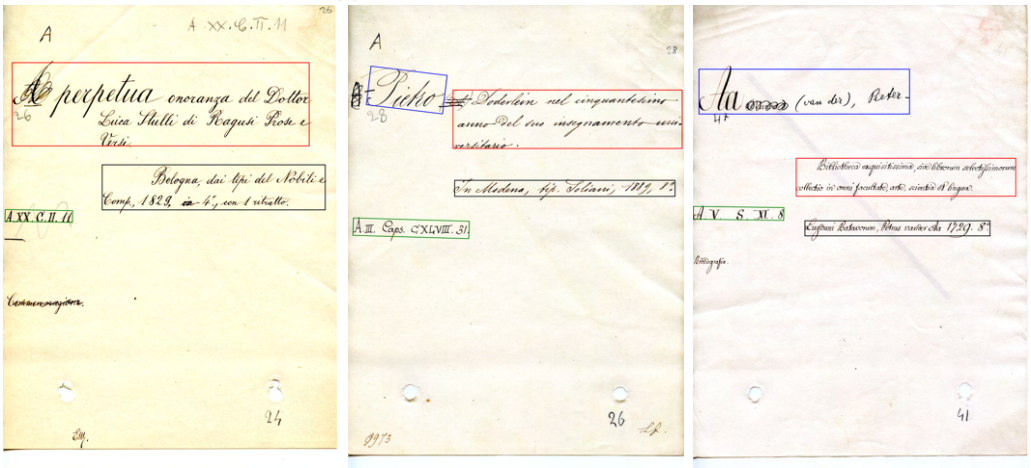
I risultati confortano sul fatto che l'algoritmo di miglioramento sia effettivamente utile, ossia si avvicina di più alla segmentazione finale che si vuole ottenere e ciò, di conseguenza, dovrebbero agevolare l'utente che è portato ad intervenire meno nella fase di editing. Difficilmente però si ottengono delle bounding box perfette: capita solo con le schede più semplici, senza errata corrige, ed in ogni caso i bordi rimangono quasi sempre da aggiustare, perché il poligono tende ad essere troppo basso per contenere le lettere che si estendono in altezza. Altri errori che possono manifestarsi al termine dell'algoritmo, sono una cattiva classificazione o falsi negativi. Problemi che invece sono stati risolti, del tutto o quasi in maniera permanente, sono l'esclusione di falsi positivi e la separazione netta tra bounding box differenti.
Per quanto riguarda la qualità delle trascrizioni, invece, i risultati ottenuti da Cloud Vision API non sono male, soprattutto se si considera che il layout e la calligrafia delle schede Caronti saranno sicuramente molto diversi rispetto alle immagini su cui il modello di Google dev'essersi allenato. A volte, a causa di una maldestra segmentazione, certe trascrizioni sono chiaramente sbagliate, come quelle che scomodano l'alfabeto cirillico, ma nel complesso sono abbastanza le parole che vengono riconosciute correttamente. Secondo un test condotto su 10 schede, su 335 parole trascritte, 63 sono parole relative ad un'errata corrige o ad una segmentazione sconclusionata (perché non ricopre bene la parola), 53 sono le parole trascritte erroneamente e 219 sono corrette. Se si considerano solo le ultime due categorie, la percentuale di parole riconosciute dal modello di Cloud Vision sono circa l'80%. Se invece si fanno rientrare nei conti anche quelle parole individuate da una segmentazione errata, che comunque spesso finiscono nella trascrizione finale mostrata all'utente, il quale deve correggerle, la percentuale di parole trascritte bene scende al 65%. Sull'aspetto delle trascrizioni non è stato sviluppato un algoritmo di correzione, ma interviene in minima parte quello di miglioramento della segmentazione, che esclude le parole che si ritiene appartengano ad una bounding box che non dovrebbe esistere o che sconfina nell'area di un'altra, che già cattura quelle stesse scritte.

Per quanto l'implementazione di un algoritmo che fosse in grado di segmentare le regioni di testo delle schede Caronti sia una delle due componenti fondanti di questo progetto, quella principale, più significativa, rimane la progettazione di un'applicazione web che

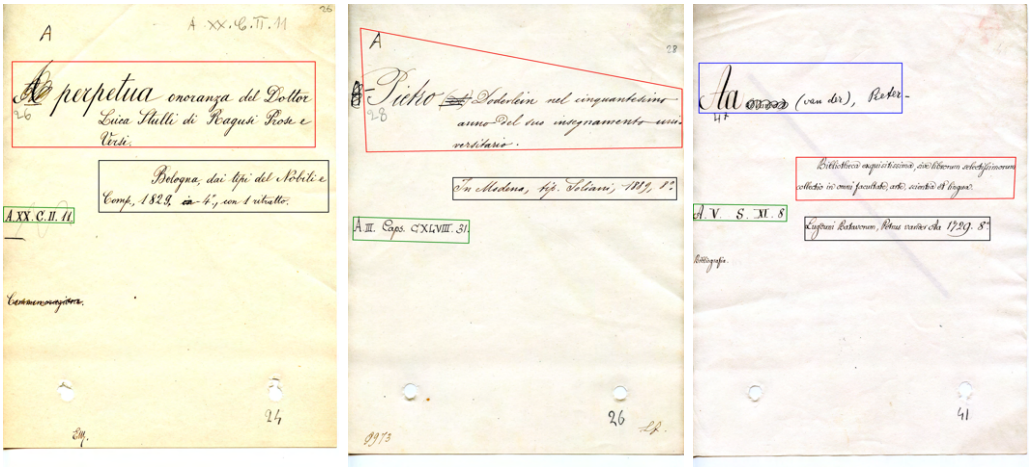
fornisce degli strumenti per fare annotazione delle immagini delle suddette schede. Di conseguenza, così come si è valutata la qualità delle segmentazioni dell'algoritmo, ancora più importante sarà la valutazione di usabilità dell'interfaccia, svolta allo scopo di comprendere se tronicarD rappresenta effettivamente un valido software per l'obiettivo che si pone.



(a) Segmentazione Cloud Vision API



(b) Segmentazione automatica tonicarD



(c) Segmentazione manuale realizzata su tonicarD

Figura 2.7: Confronto tra segmentazioni

Capitolo 3

Valutazione di usabilità

Una volta che l'applicazione è stata completata, si è predisposta una fase di testing per valutarne le proprietà di usabilità e l'esperienza dell'utente. La qualità di un'interfaccia si può infatti misurare solo attraverso l'uso che ne fanno le persone, possibilmente estranei allo sviluppo del programma, in modo da scoprire se i programmatori hanno previsto correttamente le interazioni uomo-macchina con la propria creazione. Per ragioni che sono ben facili da intuire, è stato adottato un approccio di test di tipo *discount*, quindi non volto a produrre delle statistiche significative sulle metriche dei risultati del test, bensì serve agli sviluppatori a comprendere se il modo in cui viene usato il sistema rispecchia le loro aspettative, quali sono gli errori e quanto gravi. Fornisce suggerimenti per migliorare aspetti dell'interfaccia che conducono a situazioni più o meno critiche, inoltre procura anche alcuni dati quantificabili su cui riflettere, ma a patto che se ne comprenda la fragilità e che non si sfruttino per avanzare chissà quali verità statistiche. Allo scopo di valutare l'usabilità dell'interfaccia di questo progetto, essa è stata messa a confronto con quella di un altro annotatore, che è stato individuato in Transkribus. Serviva infatti un'applicazione che fornisse gli strumenti per fare segmentazione e trascrizione automatica di un'immagine con testo e Transkribus era il candidato ideale tra i sistemi a disposizione. Un test prevede l'individuazione di tre fattori chiave:

1. Utenti
2. Task
3. Dati da raccogliere

3.1 Scelta utenti

Sarebbe importante per qualsiasi test avere a disposizione degli utente reali, che facciano parte di quella categoria di persone cui è destinato l'uso del programma che viene

analizzato. In questo caso, gli utenti cui si rivolge l'interfaccia sono certi impiegati della Biblioteca Universitaria di Bologna, i quali tuttavia non si sono potuti avere a disposizione per la fase di testing. Persone qualsiasi non si trovano mai nella situazione di dover produrre annotazioni di immagini, e cioè di segmentarle e trascriverne il contenuto. Tuttavia, si tratta di un deficit di contesto, più che di competenze: un bibliotecario non ha studiato come funzionano i modelli di riconoscimento automatico del testo, o che cosa sia un ground truth o una segmentazione; potrebbe ritrovarsi nella condizione di doverlo imparare, ma solo in una situazione particolare come quella che ha dato origine a questo progetto. È una circostanza insolita anche per un bibliotecario dover usare un sistema di annotazione automatica, che sono invece noti e sfruttati più che altro da informatici. Al posto degli impiegati della biblioteca, hanno svolto il ruolo di tester persone provenienti da diversi percorsi formativi, la maggior parte laureate, quasi tutte con un'età tra i 20 e i 30 anni. Nessuno dei tester aveva un'istruzione o un lavoro che li avesse preparati a svolgere il compito che gli sarebbe stato assegnato e nemmeno che li avesse messi a conoscenza di cosa sia un'annotazione di immagine con testo scritto.

In conclusione, così come devono essere istruiti dei tester generici su cosa sia un'annotazione, allo stesso modo lo devono essere anche dei bibliotecari. Per questa ragione, la differenza fra la tipologia di utente reale e gli utenti che invece sono stati scelti per il testing è rilevante solo parzialmente.

3.2 Scelta del task

Si tratta di individuare il compito che l'utente deve svolgere usando l'applicazione da testare. Come gli utenti del test dovrebbero essere quelli reali, così il task dovrebbe coincidere con un task reale, che l'utente dovrà compiere quando usa il sistema. Si chiederà quindi agli utenti di fare qualcosa e si terrà traccia delle loro azioni, sulla base delle quali si costruirà l'analisi di usabilità. In questo caso, si è sviluppato un'applicazione per fare annotazione di immagini di schede del catalogo della Biblioteca; è stata concepita per nient'altro che per tale compito. La scelta del task è stata quindi inevitabilmente quella di portare a terminare l'annotazione di una scheda Caronti. Task però non affatto banale, perché è frammentato e perché il suo significato è sconosciuto ai più e non scontato da far comprendere appena prima di un test, senza che si mostri prima il funzionamento di un annotatore. Per queste ragioni, è stato inevitabile fornire ai tester alcune informazioni necessarie affinché fossero in grado di completare il task, ma che rigorosamente non riguardassero l'interfaccia ed il suo funzionamento. Agli utenti è stato perciò spiegato che:

- Gli si chiedeva di eseguire l'annotazione di un'immagine con del testo.
- L'annotazione di un'immagine consiste di due operazioni: segmentazione e trascrizione.

- La segmentazione consiste nell'evidenziare le porzioni dell'immagine in cui si riconosce la presenza di testo.
- La trascrizione consiste nel trascrivere per ciascuna porzione evidenziata nella segmentazione il testo che essa identifica.
- Con esplicito riferimento alla scheda che avrebbero dovuto annotare, gli è stato indicato quale fosse il testo corrispondente ad autore, titolo, note e collocazione.
- Con esplicito riferimento alla scheda che avrebbero dovuto annotare, gli è stato chiesto di evitare, se possibile, di segmentare e trascrivere parti di testo cancellate o estranee alle categorie titolo, autore, note e collocazione.
- Gli è stato mostrato in quale cartella del computer usato per il test risiedeva il file della scheda che avrebbero dovuto annotare.

Queste sono le informazioni di base che sono state riferite a tutti gli utenti per tutti i test eseguiti. Dopo di che, la scalette ufficiali del task erano leggermente personalizzate in base all'applicazione che si testava. Questa è stata quella per Transkribus:

1. Visualizzare la scheda del file X sull'interfaccia.
2. Usare la funzione di segmentazione automatica fornita dal sistema.
3. Correggere i risultati della segmentazione.
4. Usare la funzione di trascrizione automatica fornita dal sistema.
5. Correggere i risultati della trascrizione.
6. Salvare.

Mentre la scaletta del task di tonicarD era:

1. Individuare la scheda del file X sull'interfaccia.
2. Usare la funzione di annotazione automatica fornita dal sistema.
3. Correggere i risultati sia della segmentazione sia della trascrizione.
4. Salvare.

L'esistenza di due elenchi diversi per la presentazione del task è conseguenza del fatto che su Transkribus la segmentazione e la trascrizione sono due operazioni separate, mentre su tonicarD sono racchiuse in un unico comando, quello dell'annotazione. Non si voleva dare l'idea al tester che ci fosse un ordine giusto o sbagliato da rispettare nella correzione

di bounding box e di trascrizioni; tuttavia, per paragonare meglio alcuni aspetti dei test, in certi casi il task di correzione su tonicarD è stato spezzato in due. Esiste però un elemento che manifesta ancora di più la differenza fra i due test, ossia che per quello su Transkribus è stato messo a disposizione dell'utente una pagina del tutorial ufficiale di alcune delle operazioni fondamentali del task; il tutorial poteva essere liberamente consultato. Tale condizione si è ritenuta indispensabile al fine di permettere all'utente di completare il task su Transkribus senza l'aiuto di un'altra persona. L'applicazione Transkribus è infatti estremamente sofisticata, con centinaia di funzionalità di carattere tecnico, incomprensibili a chiunque non sia un esperto. Perdersi in un tale ambiente così ricco di comandi e rivolto ad un target specializzato sarebbe naturale per chiunque lo affronti per la prima volta. Il tutorial è stata considerata una guida fondamentale e tra l'altro è la stessa produzione del software che, consapevole della difficoltà di approccio l'applicazione, la mette a disposizione del pubblico. Purtroppo non includeva le istruzioni per avviare la trascrizione automatica del testo, per trovare la quale gli utenti si sono dovuti arrangiare. Al contrario, nessun tipo di aiuto esterno è stato dato per eseguire il task su tonicarD, sulla cui scaletta però bisogna fare una precisazione: è possibile sia visualizzare la scheda e poi richiederne l'annotazione, sia il contrario. Nel resoconto dei test si manterrà l'elenco così come definito in questo paragrafo, a prescindere dal flusso degli eventi, prendendo i giusti accorgimenti per evitare ambiguità.

3.3 Thinking aloud test

Il primo test è stato eseguito su cinque persone, che hanno lavorato ciascuna su schede differenti, in modo da diversificare le situazioni. La tipologia del test è stata *Thinking aloud*. Essa prevede che l'utente rimanga in costante comunicazione con l'osservatore, il quale deve intervenire al solo scopo di incoraggiare il tester a parlare quando non lo fa più. L'osservatore non dice mai cose al fine di indirizzare l'attenzione dell'utente all'uno o all'altro elemento dell'interfaccia, per avvisarlo che ha sbagliato o per mettergli il dubbio che abbia sbagliato. Lo scopo è solo quello di sapere a cosa pensa il tester mentre lavora al completamento del task. L'osservatore fornisce effettivamente un aiuto all'utente solo quando egli dimostra l'intenzione di arrendersi ed abbandonare il test. Lo scopo del test non è giudicare l'interfaccia di Transkribus, che viene usata solo per avere un termine di paragone, bensì quella di tonicarD e comprenderne le debolezze. Metriche dei risultati verranno collezionate e presentate nella prossima sezione, che riguarda un test fatto appositamente per tale scopo; qui, si raccoglie qualche prima impressione in base ai test il cui svolgimento può essere consultato nell'appendice A.

3.3.1 Efficienza

Innanzitutto, l'aspetto più evidente è che il task viene eseguito molto più velocemente su tonicarD rispetto che su Transkribus e le ragioni che si sono individuate sono fondamentalmente tre:

1. tonicarD è un'applicazione specifica per queste schede, che sono direttamente integrate nel sistema, quindi è inevitabilmente più veloce l'accesso ai file desiderati rispetto che su un'applicazione general purpose.

2. tonicarD è progettato per un'unica funzionalità, il che riduce il numero di comandi e di operazioni con cui l'utente deve interagire rispetto ad un'applicazione come Transkribus.

3. tonicarD offre poca elasticità sulla quantità delle segmentazioni, il che riduce il numero di segmentazioni da modificare.

È fondamentale però sottolineare che il punto 3 è direttamente correlato ad una tradeoff che il sistema fa. Si rinuncia alla precisione di una segmentazione esatta, linea per linea o parola per parola che permetterebbe di escludere sempre falsi positivi, con la consapevolezza che ciò potrebbe generare un risultato sporco, non perfetto. Ciò che si ottiene da questa mancanza è un'annotazione più rapida, che chiede all'utente meno sforzi per completare il task. Egli, infatti, ha meno correzioni da fare e meno precise di quelle che potrebbe fare, ad esempio, su Transkribus.

3.3.2 Efficacia

Il rispetto del task di correzione della segmentazione è fallito allo stesso modo su entrambi i sistemi. Le barre dei comandi di editing delle segmentazioni sono state quasi completamente ignorate, infatti nessuno ha mai cercato di comprendere il significato delle loro icone e delle loro funzionalità. Ciò ha comportato delle segmentazioni tagliate male su tonicarD e la presenza di falsi positivi su Transkribus. Più che un problema di interfacce, si può riconoscere in questo comportamento dell'utente un'incomprensione del task e della sua importanza. Il tester non istruito non sa che tale lavoro serve a ritagliare l'immagine in punti specifici e che deve essere fatto solo dove c'è del testo. Per quanto riguarda la trascrizione, non è di interesse la misura in cui essa fosse corretta, dal momento che dipende dalla sola capacità della persona di riconoscere la calligrafia del dott. Caronti. In conclusione, la qualità delle annotazioni, relativamente al miglior risultato possibile per ciascuna applicazione, è stata modesta in entrambi i casi. A proposito, invece, dell'accuratezza con cui le varie sottofasi del task sono state svolte, si può notare che essa è stata molto più alta su tonicarD. Su Transkribus, infatti, a volte non è stato portato a termine (senza l'aiuto dell'osservatore) il compito di visualizzare la

scheda, oltre al fatto che sono stati spesso cliccati comandi sbagliati alla ricerca di quelli desiderati.

3.3.3 Soddisfacibilità

Gli utenti, dopo aver terminato entrambi i test, hanno dimostrato preferenza per tonicarD, che hanno riconosciuto essere un sistema più semplice. Tutti, infatti, hanno speso molto più tempo su Transkribus, sul quale hanno affrontato diverse difficoltà e hanno dovuto risolvere problemi. Al contrario, su tonicarD, il flusso di lavoro è stato decisamente più fluido. Ciò non significa che avessero sempre tutto sotto controllo e che sapessero se stavano agendo bene o male, ma quantomeno non hanno mai affrontato errori bloccanti che gli hanno richiesto di esplorare a lungo l'interfaccia alla ricerca di una soluzione o di chiedere aiuto all'osservatore.

3.4 Bottom line-data test

Quando si vogliono raccogliere dei dati quantitavi, misurabili, che riflettano in qualche modo la qualità di un'interfaccia, bisogna eseguire un test bottom line-data. Di solito gli elementi che si tengono in considerazione sono il tempo che un utente impiega per completare un task e il numero e la gravità degli errori che commette. Pur trattandosi di un test di tipo *discount*, si differenzia dal test *thinking aloud*. Quest'ultimo infatti non è adatto quando si vogliono misurare dei dati, in quanto parlare rende l'utente più lento o anche, a volte, più consapevole durante l'esecuzione del task, il che influisce direttamente (negativamente nel primo caso e positivamente nel secondo) sul tempo e sugli errori. Calcolare la quantità di tempo è, di fatto, un'operazione banale, mentre misurare il numero e soprattutto la gravità degli errori è un compito sicuramente più complicato. Bisogna semplicemente accettare il fatto che non esiste un protocollo standard, non esistono criteri esatti e quindi l'analisi è a discrezione di chi progetta il test. Si è deciso di svolgere questo tipo di test per avere oltre che idee, suggerimenti ed impressioni, anche dei valori numerici su cui basare il confronto tra le due interfacce, Transkribus e tonicarD. Gli approcci in questo caso sono due:

- **Within-groups experiment**. Si usa un solo gruppo di tester, i quali svolgeranno il task su entrambi i sistemi. Questa soluzione è leggermente problematica perché potrebbe arrecare qualche svantaggio o vantaggio eseguire un compito per la seconda volta, dopo che già lo si è portato a termine con il primo sistema. Bisognerebbe pensare a task diversi per le due interfacce. Tuttavia, è problematico soprattutto quando si testano due diverse versioni della stessa applicazione. Eseguire l'annotazione su due sistemi diversi come quelli che si paragonano qui, non dovrebbe essere particolarmente dannoso ai fini del test.

- **Between-groups experiment**. Si usano due gruppi di tester che svolgeranno lo stesso task, uno sul primo sistema e l'altro sul secondo. In questo modo si possono confrontare senza problemi i valori tipici dello stesso task quando eseguiti su di una o sull'altra interfaccia.

Per questo lavoro, si è deciso di adottare l'approccio *between-groups experiment* che fornisce dati un po' più consistenti. Si sono creati due gruppi composti da quattro persone ciascuno. Il gruppo A ha lavorato su Transkribus, il gruppo B su tonicarD. Le indicazioni sul task sono rimaste le stesse di quelle presentate nella sezione 3.2.

3.4.1 Efficienza

Per misurare l'efficienza, si è cronometrato il tempo di esecuzione del task di annotazione per ciascun utente, considerando anche i parziali per ogni sottofase.

	Visualizzazione immagine	**Segmentazione automatica**	**Correzione segmentazioni**	**Trascrizione automatica**	**Correzione trascrizioni**	**Salvataggio**	**Task completo**
A1	9:14	3:58	/	12:59	2:34	1:05	30:50
A2	4:45	0:30	/	5:39	5:51	0:25	17:10
A3	12:53*	0:59	7:16*	/	5:16	2:00	28:24
A4	11:20	6:40	/	1:00*	3:35	1:41	24:16

Tabella 3.1: Tempo di esecuzione del task di annotazione di una scheda Caronti su Transkribus (minuti). L'asterisco (*) indica che il task non è stato completato o che è stato terminato con l'aiuto dell'osservatore.

	Visualizzazione immagine	**Segmentazione automatica**	**Correzione segmentazioni**	**Correzione trascrizioni**	**Salvataggio**	**Task completo**
B1	2:00	0:10	/	1:50	0:03	4:03
B2	1:19	0:21	1:00*	1:48	0:03	4:31
B3	0:30	0:40	0:46*	2:04	0:01	4:01
B4	1:55	0:07	/	3:57	0:05	6:04

Tabella 3.2: Tempo di esecuzione del task di annotazione di una scheda Caronti su tonicarD (minuti). L'asterisco (*) indica che il task non è stato completato o che è stato terminato con l'aiuto dell'osservatore.

Si possono esprimere alcune considerazioni superficiali da una prima visione dei risultati, prima di analizzare i valori di calcoli più significativi. Per prima cosa, è evidente la differenza di tempo per il completamento dello stesso task sulle due interfacce, lungo su Transkribus e molto più breve su tonicarD. Su di quest'ultimo, i tempi per accedere all'immagine su cui lavorare sembrano essere stati ridotti di netto grazie al fatto che i file vengono inclusi nel sistema e quindi visualizzati in homepage e rintracciabili attraverso una barra di ricerca. La fase di correzione delle bounding box sembra che intimorisca l'utente inesperto tanto su un'applicazione quanto sull'altra, dal momento che anche chi

ci ha speso tempo, ha comunque lasciato perdere ad un certo punto. Poi, è doveroso riportare che i tempi di correzione delle trascrizioni sono legittimamente più lunghi su Transkribus, poiché lì viene fatta una segmentazione riga per riga, che è più precisa, ma richiede più click all'utente, quindi più minuti. Il salvataggio sul sistema Caronti è stato sempre estremamente basso, poiché il comando è in bella vista e nessun tester ha mai dovuto cercarlo nell'interfaccia, al contrario di Transkribus, dove spesso è stato usato il tutorial per ritrovarlo.
Si vuole a questo punto fare una stima di quelli che sono i tempi tipici per il completamento del task di annotazione per ciascun sistema testato e poi valutarne la differenza. L'incertezza di questi valori è data dal basso numero di test e dalla variabilità dei risultati, tuttavia ha senso calcolarli.
Per prima cosa si stabiliscono i valori di input:

	$t_1(s)$	$t_2(s)$	$t_3(s)$	$t_4(s)$
Transkribus	1850	1030	1704	1456
tonicarD	243	271	241	364

Tabella 3.3: Tempi di esecuzione del task di annotazione di una scheda Caronti per ciascun utente (secondi).

Date le formule necessarie a calcolare l'intervallo entro cui è presumibile che risieda il valore tipico di completamento del task [21], consultabili in B, i risultati calcolati sono stati:

- Transkribus:

$$\mu_T = 1510 \tag{3.1a}$$
$$\sigma_T = 358.95 \tag{3.1b}$$
$$SE_T = 179.47 \tag{3.1c}$$
$$range_T = [1\,330.53s, 1\,689.47s] \tag{3.1d}$$

- tonicarD:

$$\mu_C = 279.75 \tag{3.2a}$$
$$\sigma_C = 57.81 \tag{3.2b}$$
$$SE_C = 28.9 \tag{3.2c}$$
$$range_C = [221.94s, 337.56s] \tag{3.2d}$$

Rispetto ai valori ottenuti, si può assumere che il tempo richiesto per completare un'annotazione su Transkribus da un utente generico si aggiri tra i 22 e i 28 minuti, mentre su tonicarD tra i 3 minuti e mezzo ed i 5 minuti e mezzo.

3.4.2 Efficacia

Per misurare l'efficacia si è cercato di tenere conto del tipo degli errori commessi e del rapporto tra task completati con successo e task totali. Gli errori si distinguono per: fase del task, frequenza (1 minimo, 4 massimo) e gravità. I valori della gravità hanno i seguenti significati:

- 1: rallenta di poco il completamento del task.
- 2: fa perdere una quantità di tempo significativa al completamento del task.
- 3: fa perdere un'esagerata quantità di tempo al completamento del task.
- 4: impedisce il completamento del task.

Si è cercato di scegliere un valore medio rispetto all'insieme dei test. Ora si considerino

Fase del task	**Errore**	**Frequenza**	**Gravità**
Visualizzazione scheda	Click su comandi sbagliati	3	1
	Selezione file non selezionabile	3	2
	Non aggiorna la lista dei file caricati	3	3
	Ripetizione task	2	2
Segmentazione automatica	Click su comandi sbagliati	1	1
	Ripetizione task	1	1
Trascrizione automatica	Click su comandi sbagliati	4	4
	Non seleziona il modello di riconoscimento	2	4
Correzione trascrizioni	Click su comandi sbagliati	1	1
	Click errato (misclick)	1	2

Tabella 3.4: Errori su Transkribus

Fase del task	**Errore**	**Frequenza**	**Gravità**
Visualizzazione scheda	Ricerca nel menù sbagliato	2	1
	Click su immagine sbagliata	1	1
Annotazione automatica	Azioni della fase successiva	1	1
Correzione trascrizioni	Inserimento testo nella locazione sbagliata	1	1
	Click inutili	4	1

Tabella 3.5: Errori su tonicarD

come task falliti quelli che nelle tabelle 3.1 e 3.2 non hanno valore, che indica proprio il fatto che non sono stati portati a termine, e quelli con l'asterisco, cioè quelli che non sarebbero stati completati senza l'intervento dell'osservatore. Il calcolo del rapporto fra task terminati con successo e task totali su Transkribus è uguale a:

$$Efficacia = \frac{13}{20} \times 100 \rightarrow 65\%$$

Mentre quello su tonicarD è:

$$Efficacia = \frac{16}{20} \times 100 \rightarrow 80\%$$

Per rendere più equo il paragone, si è considerata la fase 3 del task di tonicarD come se fosse due task separati, come per Transkribus, ossia uno di correzione delle segmentazioni ed uno delle trascrizioni. Infatti la correzione delle segmentazioni è stata un'operazione ignorata o a cui gli utenti hanno rinunciato dopo le prime interazioni con l'interfaccia di editing, su entrambi i sistemi. Valgono quindi le stesse considerazioni avanzate alla sezione 3.3.2 per quanto riguarda l'efficacia, la cui conclusione è che la qualità delle segmentazioni sono piuttosto scarse.

3.4.3 Soddisfacibilità

Per quanto sia effettivamente impossibile misurare la soddisfazione delle persone, sono stati creati svariati metodi per tentare di calcolare quanto un utente ha apprezzato l'interazione con un programma. Tra le varie tecniche, il progetto ha adottato il *System Usability Scale (SUS)* [22]: un test molto generico, veloce ed approssimativo inventato nel 1986 per misurare l'usabilità di un prodotto. È ancora considerato un sistema valido, in quanto è facilmente scalabile ed è in grado di differenziare effettivamente sistemi inservibili da sistemi buoni. Consiste in una lista di 10 affermazioni standard a cui bisogna rispondere con i valori di una scala Likert, e cioè con numeri da 1 a 5 dove 1 corrisponde a "Forte disaccordo" e 5 a "Fortemente d'accordo". Le frasi del test sono:

1. Penso che userei questo sistema frequentemente.
2. Ho trovato il sistema inutilmente complesso.
3. Penso che il sistema fosse facile da usare.
4. Penso che avrei bisogno dell'aiuto di un tecnico per essere capace di usare il sistema.
5. Ho trovato le varie funzioni del sistema ben integrate.
6. Ho pensato che ci fosse troppa incoerenza nel sistema.
7. Immagino che la maggior parte delle persone imparerebbe ad usare questo sistema molto velocemente
8. Ho trovato il sistema piuttosto caotico da usare.
9. Mi sentivo molto sicuro mentre usavo il sistema.
10. Avevo bisogno di imparare molte cose prima di poter andare avanti con il sistema.

In base alle risposte, si calcola un valore che quantifica il risultato del test, secondo il seguente algoritmo:

1. Si calcola il contributo (da 0 a 4) di ciascuna risposta:
    - Quello delle numero 1-3-5-7-9 è uguale al valore scelto della scala Likert meno 1.
    - Quello delle numero 2-4-6-8-10 è uguale a 5 meno il valore scelto della scala Likert.
2. Si moltiplica la somma dei contributi per 2.5.

Si ottiene un valore compreso tra 0 e 100, dove 68 indica il valore medio. Questi sono i risultati del test SUS ottenuti dagli 8 tester:

	Frase 1	**Frase 2**	**Frase 3**	**Frase 4**	**Frase 5**	**Frase 6**	**Frase 7**	**Frase 8**	**Frase 9**	**Frase 10**	**Contributo**	**SUS**
A1	3	4	3	4	3	2	1	4	2	5	14	35
A2	1	5	2	3	1	4	2	3	2	4	12	30
A3	2	3	2	4	2	2	3	2	1	2	21	52, 5
A4	5	3	2	4	4	4	3	3	2	3	24	60
B1	3	1	3	2	4	1	5	1	4	2	36	90
B2	5	2	5	5	1	1	5	1	1	5	27	67,5
B3	4	4	4	2	4	1	5	1	1	4	30	75
B4	4	3	5	3	4	3	4	4	4	2	31	77,5

Tabella 3.6: Valutazione SUS

La media dei valori SUS che si ottengono per Transkribus e tonicarD sono rispettivamente 44,37 e 77,5. Si può immaginare che una così ampia differenza tra i due risultati sia giustificata dalla grande differenza di tempo che i tester hanno speso per completare i task. Su Transkribus gli utenti ci hanno messo tanto a terminare il test, il che potrebbe essere indicatore della complessità dell'interfaccia. Al contrario, su tonicarD tutti gli utenti hanno terminato in breve il loro compito, segno del fatto che hanno trovato sempre in fretta i comandi che gli servivano; ciò si riflette nei valori del test di usabilità.

Capitolo 4

Conclusione

Algoritmo di segmentazione
La scelta di progettazione forse più limitante dell'intero lavoro è stata quella di prevedere una segmentazione esclusivamente per blocchi, una per ciascuna categoria di testo (autore, titolo, note e collocazione), piuttosto che una per riga, come avviene di solito. In questo modo si rende più veloce il lavoro dell'utente che annota manualmente le schede, ma c'è maggior probabilità di produrre risultati sporchi se la scheda contiene delle correzioni. Inoltre, se si segmentasse per righe, si otterrebbero più immagini (più pulite) e di conseguenza un dataset più grande, che favorirebbe l'efficacia di un modello di identificazione del testo scritto. Se si volesse rimediare a tale condizione, si potrebbe includere nell'algoritmo di segmentazione il riconoscimento delle righe, mantenendo quello del blocco: si dovrebbe individuare quando due o più annotazioni, separate o sovrapposte, fanno parte della stessa linea e generare la bounding box corrispondente, esattamente come già succede per la ricostruzione dei blocchi. Questa modifica dovrebbe poi riflettersi nell'interfaccia, in cui sarebbe permesso disegnare più di un poligono per categoria, fornire strumenti per unire e separare quelli con la stessa etichetta e creare un input di testo per ciascuno. In questo modo, si riuscirebbe, per esempio, ad escludere dall'annotazione un'errata corrige che si trova a metà di una riga di testo, ma all'utente sarebbe richiesto più lavoro e una maggior consapevolezza degli strumenti di manipolazione dell'immagine.

Un altro limite dell'algoritmo di segmentazione consiste nel ridotto campione su cui si sono costruiti i valori usati per il filtraggio e la classificazione delle bounding box. In particolare, si tratta dei vertici dei poligoni che definiscono l'area di un'etichetta, entro cui una bounding box è obbligata a rientrare affinché possa essere classificata come tale. Inoltre, si tratta anche dei centroidi di ciascun etichetta, ossia dei punti dell'immagine calcolati come la media delle coordinate dei centroidi di tutte le bounding box del campione di testing suddivise per etichetta. Ogni etichetta ha il proprio centroide, sulla base dei quali viene fatta la prima classificazione delle box. Il dataset Caronti è formato da tante eccezioni ed è prevedibile che alcune di queste abbiano caratteristiche tali

che le coordinate delle proprie box si discostino molto dalle costanti calcolate per fare la classificazione, il che potrebbe comportare un filtraggio e/o etichettatura errati. In merito a questo problema, come sviluppo futuro, si potrebbe considerare l'implementazione di una funzione che, ogni volta che viene prodotta una nuova annotazione manuale sull'applicazione, aggiorni le costanti del programma in base alle segmentazioni appena realizzate.

Infine, è giusto specificare che alla base di tutta la fase della segmentazione risiede il lavoro eseguito da Cloud Vision e quindi, in un certo senso, si può dire che il limite originale del sistema consiste nell'inconsistenza e nell'imprecisione delle segmentazioni prodotte da Google. L'algoritmo è stato costruito per rimediare agli errori dei risultati dell'API, tra quelli che sono stati osservati sul campione di studio. Se in futuro dovesse manifestarsi una nuova tipologia di errori o una diversa frequenza con cui essi compaiono, l'algoritmo potrebbe diventare meno affidabile e richiedere nuove modifiche.

Algoritmo di trascrizione

Allo stato attuale del programma, non viene eseguita alcun'analisi delle parole trascritte da Cloud Vision, che rimangono nello stesso stato in cui vengono ricevute. Al più, nella ricostruzione delle bounding box e quindi delle frasi, alcune parole potrebbero essere scartate e non essere incluse nella trascrizione. Come sviluppo futuro, viene lasciato quello di esaminare le parole in modo da modificarle o cancellarle se si riconosce che esse sono costituite da simboli che è impossibile che si trovino su di una scheda Caronti, come lettere dell'alfabeto cirillico. Il compito non è banale, in quanto ogni lingua ha il proprio alfabeto, che di solito è variante di uno comune a molti, come l'italiano deriva da quello latino, e Python non possiede funzioni di sistema per riconoscere a quale alfabeto corrispondono certi simboli o che distinguano, ad esempio, i caratteri "occidentali" da tutti gli altri.

Interfaccia

Dai test è risultato che le persone che devono essere istruite su cosa sia un'annotazione, non sono in grado poi di farla correttamente senza una spiegazione specifica delle azioni che devono compiere. Dimostrano inconsapevolezza o incertezze sul modo in cui devono agire e commettono errori, come la mancata correzione dei bordi delle bounding box o una trascrizione nel posto sbagliato. Quindi, affinché un utente usi correttamente l'applicazione è necessario che egli abbia ben chiaro come funziona una segmentazione, sia in generale sia nel contesto dello strumento che usano, in modo che non abbia dubbi e sappia sempre cosa deve fare in qualsiasi situazione. Per questo motivo, si potrebbe dotare l'interfaccia di una guida o di un qualche tipo di tutorial. Potrebbero essere implementati dei suggerimenti a schermo in base a quello l'utente che sta facendo oppure includere un elenco di istruzioni consultabile attraverso il click su di un pulsante apposito. L'obiettivo sarebbe quello di rendere capace qualsiasi persona di eseguire un'annotazione perfetta

fin dal primo utilizzo dell'applicazione, senza vedere prima una dimostrazione del suo funzionamento e senza ricevere insegnamenti da utenti più esperti. Altri accorgimenti minori che potrebbero essere presi riguardano la pagina di manipolazione dell'immagine. Ad esempio, in una finestra del browser non tanto alta, la scheda appare in tutta la propria larghezza ma solo in circa metà della propria altezza. Per natura delle schede Caronti, il testo non compare nella parte inferiore e, se lo fa, è direttamente collegato con le righe superiori, quindi nessuna informazione dovrebbe essere isolata e nascosta in fondo; tuttavia, per ogni evenienza, si potrebbe considerare di mostrare l'immagine in modo che sia visibile per intero, a prescindere dalla dimensione della finestra, così che l'utente veda immediatamente tutte le scritte della scheda, senza che necessiti di zoomare indietro o scrollare la pagina del browser per avere la conferma di non essersi perso del contenuto da trascrivere. Un altro possibile intervento potrebbe essere fatto nei confronti dei poligoni: al momento sono concepiti unicamente dei quadrilateri. Sebbene in generale non servano dei poligoni perfettamente ritagliati attorno alle parole per ottenere una buona precisione durante il riconoscimento del testo, ma siano sufficienti dei quadrilateri [23], nel caso specifico delle schede Caronti, c'è la probabilità di includere in una segmentazione una parte di testo che non si vorrebbe evidenziare (perché di un'errata corrige o per altre circostanze particolari); pertanto in tali situazioni sarebbero desiderabili dei poligoni con più di quattro vertici in modo da circondare in maniera più accurata le parole che si vogliono catturare. L'implementazione di una funzionalità che permette di disegnare queste forme merita considerazione, ma ancora una volta si rischia di complicare leggermente la comprensibilità dell'interfaccia per l'utente.

Il modello di riconoscimento automatico del testo elaborato per trascrivere le schede Caronti, che costituiscono il catalogo storico della Biblioteca Universitaria di Bologna, non è stato in grado di raggiungere risultati soddisfacenti. I limiti rappresentati dal dataset, troppo ridotto ed impreciso, non lasciavano margini di miglioramento, per cui ci si è proposti di intervenire su di esso, espandendolo e raffinandolo. Tale obiettivo si può raggiungere attraverso l'annotazione, un'operazione che consiste nell'individuare le diverse porzioni dell'immagine che contengono le scritte e nell'associare a ciascuna di esse la trascrizione corrispondente. Trattandosi di un lavoro manuale tedioso, si è costruita un'applicazione web che fornisse gli strumenti per eseguirlo in maniera veloce ed intuitiva: tonicarD. Il sistema è stato integrato con un algoritmo su misura per calcolare automaticamente le segmentazioni delle immagini, in modo da presentare all'utente dei risultati intermedi da correggere, piuttosto che costringerlo a realizzarli da zero. La base di partenza per le annotazioni automatiche è costituita da quelle fornite dal servizio Cloud Vision API di Google, che si preoccupa sia di segmentare sia di trascrivere una qualsiasi immagine in cui è presente del testo scritto a mano. I risultati non sono sufficientemente soddisfacenti per essere mostrati all'utente, ma attraverso l'analisi del layout

delle schede e del colore dei singoli pixel dell'immagine si riesce a manipolarli in modo da ottenere delle segmentazioni utili. Sulle trascrizioni, invece, non si effettua alcun'analisi e le parole identificate vengono presentate così come ricevute dall'API di Google. È stato poi condotto uno studio di usabilità dell'applicazione. tonicarD, a differenza di altri programmi simili, si concentra esclusivamente (oltre che sulle schede Caronti) sulla funzione di annotazione e non prende in considerazione la successiva fase di progettazione del modello di riconoscimento automatico. Questa scelta ha permesso la realizzazione di un'interfaccia semplice, con cui un utente non esperto è in grado di lavorare con discreto successo fin dal primo utilizzo, come ha dimostrato la fase di testing del software. Attraverso di essa sono stati messi a confronto tonicarD e Transkribus, l'applicazione forse migliore per l'analisi di documenti storici scritti a mano. Secondo un approccio *discount usability test*, sono stati condotti due separati e differenti test: il *thinking aloud* ed il *bottom line-data*. Il primo è servito per comprendere il processo mentale degli utenti nell'interazione con i programmi, il secondo ha permesso di raccogliere dei valori con cui misurare efficienza ed efficacia di completamento del task, oltre che la soddisfazione degli utenti. I risultati ottenuti suggeriscono che, seppur rimane la necessità che gli utenti siano istruiti sul significato di annotazione d'immagine, tonicarD abbia una curva di apprendimento bassa e bassi tempi di esecuzione del task di annotazione. Il prezzo che il sistema paga per una tale facilità d'uso è la precisione delle segmentazioni, che non sono per riga ma per regioni di testo e ciò comporta che in certi casi venga impedita l'esclusione di falsi positivi dal risultato finale.

Appendice A

Svolgimento thinking aloud test

A.1 Utente 1

Transkribus

1. **Visualizzare la scheda del file X sull'interfaccia**
   Il tester ha impiegato un po' di tempo prima di trovare il giusto comando per caricare il file ed una volta trovato è stato incapace di usarlo correttamente (cercava di aprire il file JPEG dell'immagine ma l'app prevede l'apertura di una cartella, non di un file). È stato aiutato una prima volta per permettergli di andare avanti e caricare il documento sull'app. Tuttavia l'app non mostra i documenti appena caricati senza un refresh esplicito della finestra apposita; ciò ha fatto sì che l'utente abbia cercato a lungo un modo per visualizzare l'immagine. Dopo aver esplorato numerosi menù non idonei a tale scopo, è stato aiutato una seconda volta affinché potesse terminare il primo punto.

2. **Usare la funzione di segmentazione automatica fornita dal sistema.**
   È stato in grado di eseguire l'operazione velocemente senza errori grazie alla consultazione del tutorial.

3. **Correggere i risultati della segmentazione**
   L'utente ha ignorato questo passaggio.

4. **Usare la funzione di trascrizione automatica fornita dal sistema.**
   Ha cercato la funzionalità ovunque nel sistema, perdendo molto tempo, fino a che non si è arreso. A questo punto gli è stato mostrato come fare.

5. **Correggere i risultati della trascrizione**
   Durante la correzione delle trascrizioni ha cancellato per errore una riga e tentato di annullare l'operazione con *Cmd+Z* ma senza successo.

6. **Salvare**
   Ha impiegato alcuni secondi per trovare il pulsante, ma poi ha salvato.

L'utente ha manifestato molta frustrazione durante tutto lo svolgimento del test, che ha richiesto circa 40 minuti. Ha rivelato grande difficoltà in quasi tutte la fasi del task.

tonicarD

1. **Individuare la scheda del file X sull'interfaccia**
   Ha individuato il file corretto scorrendo lungo il carosello delle schede non annotate con velocità e senza perdite di tempo.
2. **Usare la funzione di annotazione automatica fornita dal sistema**
   Ha richiesto l'annotazione direttamente dalla homepage non appena ha selezionato la scheda.
3. **Correggere i risultati sia della segmentazione sia della trascrizione**
   Ha ignorato le segmentazioni, le quali erano tutte presenti nella posizione giusta ma avevano bisogno di aggiustamento dei bordi. Invece, ha corretto le trascrizioni senza problemi.
4. **Salvare**
   Ha salvato immediatamente cliccando sull'apposito pulsante.

Il test è stato breve, meno di 5 minuti.

A.2 Utente 2

tonicarD

1. **Visualizzare la scheda del file X sull'interfaccia**
   Ha cercato la scheda nel carosello sbagliato delle annotate automaticamente, poi si è spostata nel carosello delle schede non annotate e ha trovato il file corretto.
2. **Usare la funzione di annotazione automatica fornita dal sistema**
   Ha richiesto l'annotazione direttamente dalla homepage non appena ha selezionato la scheda. Durante l'attesa dell'annotazione automatica, ha pensato di non aver ancora completato la fase quindi ha esplorato la funzione "Apri scheda" nella barra di navigazione, per poi rinunciare. In ritardo, ha visto dalla campanella delle notifiche che doveva solo attendere, poi quando è termina l'annotazione automatica, ha aperto la scheda dal carosello dei recenti.

3. **Correggere i risultati sia della segmentazione sia della trascrizione**
   Ha aggiustato senza problemi i contorni dei poligoni della segmentazione. La correzione della trascrizione è risultata invece un po' faticosa, poco fluida, ma è stata portata a termine anch'essa.

4. **Salvare**
   Ha salvato immediatamente cliccando sull'apposito pulsante.

Il test è durato circa 6 minuti.

Transkribus

1. **Visualizzare la scheda del file X sull'interfaccia**
   Con un po' di ricerca ha trovato il comando giusto per caricare il file sull'app, ma si è bloccato nel tentativo di selezionare il JPEG in quanto operazione impossibile; dopo un po' ha capito che doveva caricare la cartella e lo ha fatto. Ha tentato poi di eseguire la segmentazione automatica, ma non ci è riuscito perché doveva prima aggiornare la finestra dei documenti. Dopo aver perso molto tempo, è stato richiesto l'intervento dell'osservatore per completare questa fase del task.

2. **Usare la funzione di segmentazione automatica fornita dal sistema.**
   Nonostante il tutorial, ha impiegato un paio di minuti prima di trovare la finestra *Tools* che dà accesso alla funzione di segmentazione automatica. Inizialmente è andato incontro all'errore di mancato caricamento del file sul server anticipato nel punto precedente, poi una volta risolto ha eseguito il compito.

3. **Correggere i risultati della segmentazione**
   L'utente ha ignorato questo passaggio.

4. **Usare la funzione di trascrizione automatica fornita dal sistema.**
   Ha trovato dopo un po' di tempo il comando per avviare la trascrizione automatica, ma il sistema non ha preselezionato il modello da usare e l'utente non sapeva cosa dovesse fare a quel punto. È stato necessario intervenire per selezionare un modello di riconoscimento, dopo di che il tester ha avviato il comando di trascrizione.

5. **Correggere i risultati della trascrizione**
   Le trascrizioni sono state corrette senza particolari difficoltà.

6. **Salvare**
   Per alcuni secondi ha cercato il dischetto di salvataggio, che una volta cliccato, ha portato a termine il task.

Il test è durato circa 25 minuti nonostante ci siano stati degli aiuti esterni ed è stato stancante per l'utente. Mentre l'utente completava il task, gli è venuto spontaneo paragonare le due interfacce e perciò ha espresso un paio di commenti: su Transkribus ha apprezzato che quando si clicca la riga da trascrivere, la finestra dell'immagine si focalizzi sulla segmentazione associata (al contrario di quanto succede sull'altra applicazione), ma ha avuto rimostranza per la scarsa qualità della trascrizione automatica.

A.3 Utente 3

tonicarD

1. **Individuare la scheda del file X sull'interfaccia**
   Ha dapprima scrollato un po' lungo il carosello delle schede non annotate per poi usare la barra di ricerca locale. Ha selezionato una scheda, ma dall'anteprima si è reso conto che si trattava della scheda sbagliata, quindi ha filtrato meglio e ha trovato la scheda giusta.

2. **Usare la funzione di annotazione automatica fornita dal sistema**
   Al momento di richiedere l'annotazione automatica da homepage, ha invece annullato la selezione degli elementi in quanto ne aveva presi due e non voleva richiedere l'annotazione di entrambi. Ha poi selezionato solo quello corretto, ha richiesto l'annotazione automatica e visualizzato la scheda scegliendola tra le recenti.

3. **Correggere i risultati sia della segmentazione sia della trascrizione**
   Si è preoccupato di escludere l'errata corrige della prima segmentazione: ha cancellato e ricreato per tre volte la bounding box con la speranza di disegnarne una che potesse asservire al suo scopo; non riuscendoci, ci ha rinunciato e ha aggiustato la trascrizione. Per le segmentazioni successive ha intuitivamente usato correttamente i puntini dei poligoni di metà segmento per aggiustare i contorni e ha speso molto tempo per fornire una trascrizione quanto più accurata.

4. **Salvare**
   Ha salvato immediatamente cliccando sull'apposito pulsante.

Il test ha richiesto circa 8 minuti, la maggior parte dei quali è stata spesa per riconoscere alcuni caratteri del testo di difficile interpretazione.

Transkribus

1. **Visualizzare la scheda del file X sull'interfaccia**
   Ha trovato subito il comando giusto, ma nel tentativo di aprire il file JPEG, e non la cartella come vuole il sistema, non è riuscito a completare l'operazione e ha cercato un altro modo per farla. Ha perso del tempo cliccando sui menù sbagliati. Con l'intervento dell'osservatore ha finalmente caricato il file corretto, ma poi è servito altro tempo per riuscire ad aggiornare la finestra dei documenti ed a visualizzare l'immagine.

2. **Usare la funzione di segmentazione automatica fornita dal sistema.**
   Con l'aiuto del tutorial ed un po' di pazienza, ha trovato il comando giusto e ha lanciato la segmentazione automatica.

3. **Correggere i risultati della segmentazione**
   Subito ha cancellato un falso positivo, poi ha cercato di aggiustare le baseline della segmentazione (operazione non necessaria), ma ha faticato con l'editing: muoveva o cancellava bounding box erroneamente e poi non sapeva come rimediare, il che lo ha spinto a ripetere la segmentazione automatica altre due volte, per resettarla. In tutto questo tempo, non si è mai accorto della barra dei comandi di modifica immagine e ha usato solo il trackpad.

4. **Usare la funzione di trascrizione automatica fornita dal sistema.**
   Ha cercato in tutte le barre dei comandi dell'interfaccia il pulsante per lanciare la trascrizione automatica, finché non lo ha trovato nella finestrella giusta.

5. **Correggere i risultati della trascrizione**
   Ha corretto con molta attenzione tutte le righe della trascrizione senza problemi.

6. **Salvare**
   Ha trovato immediatamente il pulsante per salvare, senza bisogno del tutorial.

Il tester ha impiegato circa 30 minuti per completare il task e ha avuto bisogno per una volta di aiuto per completare una sottofase. Ha espresso il parere che con il mouse invece del trackpad avrebbe potuto modificare le segmentazione in maniera migliore.

A.4 Utente 4

Transkribus

1. **Visualizzare la scheda del file X sull'interfaccia**
   Come primo tentativo, è entrato in un menù errato ed ha usato una funzione di ricerca che non aveva a che fare con quello che gli serviva. Una volta compreso di essersi sbagliato, ha aperto il menù corretto, ma è rimasto confuso quando non è riuscito ad aprire il file JPEG ed ha immaginato di essere nuovamente in torto. Ha cercato quindi un'altra strada per completare il task, finché non ha chiesto un aiuto.

2. **Usare la funzione di segmentazione automatica fornita dal sistema.**
   Verificando il tutorial, ha lanciato il comando giusto ma ha ricevuto un errore, quello per cui si deve prima caricare il file sul server, invece che semplicemente aprire quello che risiede sulla propria macchina. Non ha compreso l'errore e non sapeva come proseguire. Dopo aver peso un po' di tempo, ha chiesto nuovamente aiuto.

3. **Correggere i risultati della segmentazione**
   Ha ignorato completamente questo passaggio.

4. **Usare la funzione di trascrizione automatica fornita dal sistema.**
   Ha trovato quasi immediatamente dove stava il comando che gli serviva e ha richiesto la trascrizione automatica. Per ragioni ignote, l'applicazione durante questo test era lenta nell'esecuzione di quest'operazione e l'utente, dopo una decina secondi, pensava che fosse fallita e che dovesse cercare un nuovo comando per portare a termine questa fase. In questo caso, è stato avvisato di essere solo paziente e quindi ha aspettato un paio di minuti, dopo i quali gli è stato chiesto di procedere con l'annotazione manuale, in quanto Transkribus sembrava non riuscire a completare il riconoscimento automatico del testo.

5. **Correggere i risultati della trascrizione**
   Non si è accorto che esisteva una precisa associazione tra riga della trascrizione e blocco che si trascrive e quindi ha trascritto tutte le righe nell'ordine sbagliato, eccetto la prima.

6. **Salvare**
   Dopo più di una decina di secondi cercando nella barra dei comandi, ha trovato l'icona di salvataggio e ha terminato il task.

Il test ha richiesto circa 25 minuti e l'utente ha affermato che fosse molto complicato.

tonicarD

1. **Individuare la scheda del file X sull'interfaccia**
   Scorrendo lungo il carosello delle schede non annotate, ha premuto il pulsante "Carica altro" e ha trovato la scheda che cercava.
2. **Usare la funzione di annotazione automatica fornita dal sistema**
   Da homepage ha lanciato il comando di richiesta di annotazione della scheda e, una volta eseguita, ha aperto la scheda dal carosello delle schede recenti.
3. **Correggere i risultati sia della segmentazione sia della trascrizione**
   Non ha neanche tentato di correggere i bordi delle bounding box, mentre ha proceduto a modificare le trascrizioni.
4. **Salvare**
   Ha salvato immediatamente cliccando sull'apposito pulsante.

Il test ha richiesto solo 3 minuti e mezzo e sarebbe stato eseguito perfettamente senza errori, se non fosse che ancora una volta è stata ignorata la fase di correzione delle segmentazioni.

A.5 Utente 5

tonicarD

1. **Individuare la scheda del file X sull'interfaccia**
   Muovendosi lungo il carosello delle schede non annotate, è arrivato al pulsante "Carica altro", l'ha cliccato e ha trovato la scheda stava cercando, che ha selezionato.
2. **Usare la funzione di annotazione automatica fornita dal sistema**
   Ha subito richiesto l'annotazione dal pulsante apposito in homepage e aperto il file dal carosello delle schede recenti.
3. **Correggere i risultati sia della segmentazione sia della trascrizione**
   Ha cominciato la correzione delle trascrizioni, poi ha cercato di aggiustare una bounding box che a metà riga aveva una cancellazione: ha escluso l'errore assieme ad una parte del contenuto da trascrivere.
4. **Salvare**
   Ha salvato immediatamente cliccando sull'apposito pulsante.

Il test è stato molto veloce, solo 3 minuti e mezzo, infatti non ci sono stati problemi durante il suo svolgimento.

Transkribus

1. **Visualizzare la scheda del file X sull'interfaccia**
   Ha iniziato leggendo il tutorial, ma ha avuto difficoltà quando ha cercato di aprire il file JPEG, poi grazie ad un doppio click fortuito, ha aperto la cartella e visualizzato l'immagine della scheda.

2. **Usare la funzione di segmentazione automatica fornita dal sistema.**
   Con il tutorial ha trovato velocemente il comando corretto, ma si è verificato un problema: non aveva ancora importato l'immagine sul server di Transkribus. L'errore non è stato compreso dall'utente, che non sapendo bene cosa fare, ha perso tempo cercando comandi inutili e non ha considerato quello giusto quando ci capitava sopra. L'utente è stato aiutato per completare l'importazione del file e concludere così la fase di segmentazione.

3. **Correggere i risultati della segmentazione**
   Ha iniziato la correzione alzando le baseline delle segmentazioni tanto quanto l'altezza dei caratteri (operazione inutile), poi ha spostato e cancellato per errore il blocco di annotazione dell'intera scheda, il che ha eliminato tutte le annotazioni. Grazie al tasto UNDO ha ripristinato lo stato corretto dell'immagine e ha terminato questa fase. L'editing è risultato complicato con il trackpad.

4. **Usare la funzione di trascrizione automatica fornita dal sistema.**
   Ha trovato e lanciato il comando di trascrizione automatica velocemente, senza perdere tempo, ma avendo precedentemente modificato il layout dell'applicazione, la finestra delle trascrizioni è rimasta nascosta, dando l'impressione all'utente che non fosse successo nulla. Essendo egli confuso e non sapendo più che fare, gli è stato mostrato come visualizzare le trascrizioni appena eseguite.

5. **Correggere i risultati della trascrizione**
   Ha corretto tutte le righe della trascrizione senza problemi e cancellato un falso positivo che non aveva individuato durante la correzione delle segmentazioni.

6. **Salvare**
   Ha trovato immediatamente il pulsante per salvare, senza bisogno del tutorial.

Il test è durato circa 16 minuti, ma l'utente è stato aiutato in un paio di occasioni in cui sembrava bloccato.

Appendice B

Formule matematiche

- Media aritmetica: $\mu_t = \frac{\sum_{i=1}^{n}}{n}$
- Deviazione standard: $\sigma_t = \sqrt{\frac{\sum_{i=1}^{n} t_i^2 - \frac{(\sum_{i=1}^{n} t_i)^2}{n}}{n-1}}$
- Errore della deviazione standard: $SE = \frac{\sigma_t}{\sqrt{n}}$
- Range del valore tipico: $range_t = [\mu_t - 2 \times SE, \mu_t + 2 \times SE]$

Bibliografia

[1] *Transformer OCR*. 2020. URL: https://github.com/him4318/Transformer-ocr.

[2] Tobias Mathias Hodel et al. "General models for handwritten text recognition: feasibility and state-of-the art. German kurrent as an example". In: *Journal of open humanities data* 7.13 (2021), pp. 1–10.

[3] José Carlos Aradillas Jaramillo, Juan José Murillo-Fuentes e Pablo M Olmos. "Boosting handwriting text recognition in small databases with transfer learning". In: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE. 2018, pp. 429–434.

[4] *Cloud Vision API*. URL: https://cloud.google.com/vision/docs/handwriting.

[5] Sébastien Eskenazi, Petra Gomez-Krämer e Jean-Marc Ogier. "A comprehensive survey of mostly textual document segmentation algorithms since 2008". In: *Pattern Recognition* 64 (2017), pp. 1–14.

[6] Jamshed Memon et al. "Handwritten optical character recognition (OCR): A comprehensive systematic literature review (SLR)". In: *IEEE Access* 8 (2020), pp. 142642–142668.

[7] Bryan C Russell et al. "LabelMe: a database and web-based tool for image annotation". In: *International journal of computer vision* 77.1 (2008), pp. 157–173.

[8] *labelMe*. 2011. URL: https://github.com/wkentaro/labelme.

[9] Abhishek Dutta e Andrew Zisserman. "The VIA annotation software for images, audio and video". In: *Proceedings of the 27th ACM international conference on multimedia*. 2019, pp. 2276–2279.

[10] Mykhaylo Andriluka, Jasper RR Uijlings e Vittorio Ferrari. "Fluid annotation: a human-machine collaboration interface for full image annotation". In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 1957–1966.

[11] Berrin A Yanikoglu e Luc Vincent. "Pink Panther: a complete environment for ground-truthing and benchmarking document page segmentation". In: *Pattern Recognition* 31.9 (1998), pp. 1191–1204.

[12] Chang Ha Lee e Tapas Kanungo. “The architecture of trueviz: A groundtruth/metadata editing and visualizing toolkit”. In: *Pattern recognition* 36.3 (2003), pp. 811–825.

[13] Sherif Yacoub, Vinay Saxena e Sayeed Nusrulla Sami. “Perfectdoc: A ground truthing environment for complex documents”. In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE. 2005, pp. 452–456.

[14] Eric Saund, Jing Lin e Prateek Sarkar. “Pixlabeler: User interface for pixel-level labeling of elements in document images”. In: *2009 10th International Conference on Document Analysis and Recognition*. IEEE. 2009, pp. 646–650.

[15] David Doermann, Elena Zotkina e Huiping Li. “GEDI-a groundtruthing environment for document images”. In: *Ninth IAPR International Workshop on Document Analysis Systems (DAS 2010)*. Citeseer. 2010.

[16] Christian Clausner, Stefan Pletschacher e Apostolos Antonacopoulos. “Aletheia-an advanced document layout and text ground-truthing system for production environments”. In: *2011 International Conference on Document Analysis and Recognition*. IEEE. 2011, pp. 48–52.

[17] Ofer Biller et al. “Webgt: An interactive web-based system for historical document ground truth generation”. In: *2013 12th International Conference on Document Analysis and Recognition*. IEEE. 2013, pp. 305–308.

[18] Philip Kahle et al. “Transkribus-a service platform for transcription, recognition and retrieval of historical documents”. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 4. IEEE. 2017, pp. 19–24.

[19] Mathias Seuret et al. “A semi-automatized modular annotation tool for ancient manuscript annotation”. In: *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE. 2018, pp. 340–344.

[20] Benjamin Kiessling et al. “eScriptorium: an open source platform for historical document analysis”. In: *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*. Vol. 2. IEEE. 2019, pp. 19–19.

[21] Clayton Lewis e John Rieman. “Task-centered user interface design”. In: *A practical introduction* (1993), pp. 77–92.

[22] John Brooke et al. “SUS-A quick and dirty usability scale”. In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[23] Veronica Romero et al. “Influence of text line segmentation in handwritten text recognition”. In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. 2015, pp. 536–540.

Ringraziamenti

Desidero ringraziare il prof. Fabio Vitali che è stato il relatore sia della mia tesi di laurea triennale che della presente tesi di laurea magistrale, quindi indubbiamente una figura fondamentale per la mia formazione universitaria. Ringrazio il correlatore Francesco Sovrano, che è stato molto disponibile e mi ha seguito durante tutto il processo di svolgimento della tesi, indicandomi ogni volta cosa andava fatto e cosa no. Ringrazio Giacomo Nerozzi, responsabile della Biblioteca Universitaria di Bologna, per aver dato origine al progetto di digitalizzazione delle schede Caronti, che ha determinato poi il mio lavoro. Ringrazio, infine, Irene Schena, responsabile della struttura informatica della Biblioteca universitaria, la quale è stata l'intermediaria tra il dipartimento di Informatica e la Biblioteca.