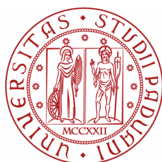

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

" Progettazione e sviluppo di un tool di annotazione distribuito con interfaccia web per immagini istologiche digitali "

Relatore: Prof. Ghidoni Stefano

Laureando: Luca Fantin (200156)

Correlatore: Dott. Barcellona Leonardo

ANNO ACCADEMICO 2022-2023

Data di laurea: 19 luglio 2023

Abstract

Le immagini istologiche rappresentano tessuti vegetali o animali analizzati al microscopio. Con l'avvento degli scanner digitali è stato possibile creare vaste raccolte e rendere molto più efficiente il loro studio e analisi. In particolare, si possono applicare modelli di Machine Learning per identificare patologie o altre proprietà dei tessuti, analizzandoli in maniera automatica. Per raggiungere questo obiettivo, le immagini usate per addestrare il modello devono essere annotate, ovvero bisogna identificare su di esse le caratteristiche che il modello dovrà successivamente individuare. Questo insieme di informazioni prende il nome di ground truth. Molti dei tool di annotazione disponibili non danno la possibilità a più utenti di lavorare sullo stesso insieme di immagini. Qualora tale opzione sia disponibile, il tool viene definito collaborativo. Quest'ultimo permette di migliorare l'efficienza del processo di annotazione e la qualità del ground truth grazie al lavoro svolto da più persone sulla stessa serie di annotazioni.

Questa tesi descrive la progettazione e lo sviluppo di un tool di annotazione collaborativo per immagini istologiche. Questo strumento permette di salvare le immagini in un database, visualizzarle e interagire con esse tramite un'interfaccia web efficace e intuitiva da usare. Il software sviluppato in questa tesi si concentra sul salvataggio e visualizzazione delle slide e si può facilmente prestare in futuro all'aggiunta di nuove feature, come l'implementazione del sistema di annotazione, l'aggiunta di label, ovvero etichette, alle annotazioni e la gestione di immagini e annotazioni da parte degli utenti.

Indice

Abstract 1

Introduzione 4

Strumenti usati per lo sviluppo del tool 5

- OpenSlide . . . 5
- Linguaggi di programmazione usati dall'applicazione . . . 6
- Altri programmi e strumenti usati nell'applicazione . . . 7

Progettazione del database 8

- Analisi dei requisiti . . . 8
  - Utenti . . . 8
  - Slide . . . 8
  - Annotazioni . . . 8
- Salvataggio delle immagini nel database . . . 8
- Scelte di progettazione concettuale per la definizione degli attributi delle entità . . . 10
- Schema ER . . . 11
- Schema relazionale . . . 14

Applicazione web 15

- Home page . . . 15
  - Caricamento della pagina . . . 16
- Viewer . . . 17
  - Caricamento della pagina . . . 18
  - Inserimento dello zoom . . . 18
  - Panoramica delle variabili globali . . . 19
  - Lettura delle immagini . . . 21
  - Inserimento dell'immagine . . . 21
  - Zoom dell'immagine . . . 22
  - Calcolo delle coordinate per la lettura delle immagini . . . 23
  - Passaggio a livello piramidale inferiore . . . 25
  - Passaggio a livello piramidale superiore . . . 29

Conclusioni 32

Ringraziamenti 33

Immagini

1 Schema ER del database . . . 12
2 Schema relazionale del database . . . 14
3 Home page dell'applicazione . . . 15
4 Viewer dell'applicazione . . . 17
5 Schema di `zoomMoveImage` . . . 23
6 Schema di `getCoords` . . . 25
7 Schema per il calcolo del valore di *ratio* da inserire in ***levelData*** per i livelli saltati nei passaggi a livelli inferiori . . . 28

Tabelle

1 Metodi di OpenSlide usati nell'applicazione 6
2 Confronto tra opzioni per salvataggio immagini in database 10
3 Entità presenti nello schema ER 12
4 Associazioni presenti nello schema ER 13
5 Variabili e strutture dati globali del codice JavaScript del viewer 20

Algoritmi

1 zoomMoveImage 22
2 getCoords 24
3 Passaggio a livello inferiore 26
4 Passaggio a livello superiore 30

Introduzione

Negli ultimi anni il campo dell'istologia, ovvero lo studio dei tessuti animali e vegetali, ha incontrato la digitalizzazione. Questo fenomeno ha portato alla creazione e diffusione di immagini istologiche digitali, anche dette *whole-slide images*; dopo essere preparati in modo tradizionale, i vetrini vengono scannerizzati per ricavarne una versione digitale [1] [2]. La caratteristica principale di queste immagini è che, in molti dei formati diffusi, sono composte da più versioni: da quella originale con risoluzione massima vengono calcolate e salvate varianti sottocampionate a risoluzione minore, per garantire un accesso più rapido all'immagine senza la necessità di caricare subito la versione più pesante [3].

Questa innovazione, oltre a migliorare sensibilmente la conservazione e archiviazione delle immagini, permette di usare questi dati in applicazioni di Intelligenza Artificiale [4] [5], in modo specifico di Machine Learning [6] [7]. Di questa macro-categoria di applicazioni, questa tesi concerne il Supervised Learning, ovvero quei casi in cui gli algoritmi addestrati, definiti agenti razionali, vengono addestrati con un dataset formato da coppie input-output, che esprimono i dati che l'agente può ricevere e le caratteristiche di rilievo che l'agente dovrà successivamente rilevare e predire, chiamate *ground truth*. Per esempio, un team di dottori specializzati può preparare un dataset che associa immagini istologiche o loro porzioni alla presenza di particolari patologie e condizioni, per poi addestrare con questo dataset un agente razionale per aiutare gli stessi dottori a riconoscere quelle stesse condizioni in nuove immagini istologiche, approccio che è già stato sperimentato su varie applicazioni di analisi e classificazione di immagini istologiche digitali [8] [9].

Per creare questi dataset, quindi, serve annotare il ground truth sulle immagini che compongono il training dataset, ovvero l'insieme di dati con cui vengono addestrati gli agenti. Questo processo viene effettuato tramite appositi tool di annotazione. Su internet sono presenti molti paper che descrivono design e implementazione di vari tool di annotazioni per whole-slide images. Questi strumenti presentano varie funzioni utili, come la capacità di trovare automaticamente centri e contorni delle aree da annotare [10] oppure la possibilità di allenare man mano che si creano annotazioni una rete neurale che assiste l'utente nella creazione delle annotazioni [11]. Tuttavia, molti di questi non offrono soluzioni collaborative. In molti casi può essere utile la collaborazione tra più utenti per creare un dataset esaustivo, in particolare in un contesto medico dove gruppi di medici potrebbero lavorare sull'annotazione dello stesso ground truth. Tuttavia, molti tool disponibili non offrono soluzioni che permettano a più persone di lavorare sullo stesso insieme di immagini.

In questa tesi, viene descritta la progettazione e l'implementazione di un tool di annotazione collaborativo sviluppato come applicazione web fornita di database dove caricare le immagini, le informazioni sugli utenti e le annotazioni create. Nella versione presentata qui, l'utente accede ad una home page dove può scegliere di visualizzare una delle immagini salvate nel database oppure caricare nuove immagini. Una volta selezionata l'immagine, viene reindirizzato ad una pagina viewer, dove viene caricata l'immagine intera nella sua versione a risoluzione minore. Questa viene posizionata in un box nel quale l'immagine può essere ingrandita o rimpicciolita, grazie allo slider presente, e scorsa, grazie alle scrollbar che appaiono e si ridimensionano man mano che lo zoom cambia. Il viewer cambia automaticamente la versione dell'immagine per mostrare all'utente risoluzioni maggiori o minori a seconda del livello di zoom applicato.

Strumenti usati per lo sviluppo del tool

OpenSlide

Le whole-slide images che devono essere trattate dal tool di annotazione presentano sfide e difficoltà totalmente assenti dai formati di immagini più comuni. Infatti, le immagini istologiche digitali hanno una struttura particolare, ovvero una "piramide" dove ogni livello, definito *livello piramidale*, rappresenta una variante dell'immagine ad una certa risoluzione; a questi livelli viene associato a un numero crescente che parte da 0, corrispondente alla versione dell'immagine a risoluzione massima, e aumenta di 1 ad ogni sottocampionamento effettuato, fino al livello massimo a risoluzione minima [3]. Nel corso della tesi, quando il numero del livello aumenterà, si dirà che si passa ad un livello inferiore, e viceversa per i livelli superiori.

Servono formati ad hoc che possano salvare tutte queste varianti, oltre a metadati e a eventuali altre piccole immagini come anteprime, in un unico file. Una conseguenza di ciò è che sul mercato esistono tanti formati diversi di whole-slide images forniti da varie aziende. Sono diversi tra loro ed è difficile trovare una documentazione esaustiva sul loro funzionamento. Sviluppare uno strumento in grado di gestire e utilizzare correttamente tutti questi formati diversi richiederebbe un'enorme mole di lavoro.

Per passare oltre questi problemi, il tool presentato si basa su OpenSlide [3], una libreria open-source in C con binding per Java e Python che offre un'API unica e uniforme per usare le whole-slide images, senza che sviluppatori e utenti debbano venire a contatto con le peculiarità di ogni diverso formato. Questa libreria è supportata da Windows, Mac OS X e molte distribuzioni di Linux, permettendole di funzionare su dispositivi diversi tra loro. I formati di immagini istologiche supportati da OpenSlide sono:

- Aperio (.svs, .tif)
- Hamamatsu (.vms, .vmu, .ndpi)
- Leica (.scn)
- MIRAX (.mrxs)
- Philips (.tiff)
- Sakura (.svslide)
- Trestle (.tif)
- Ventana (.bif, .tif)
- Generic tiled TIFF (.tif)

I metodi della API usati nell'applicazione sono riportati nella Tabella 1. Per ogni metodo vengono presentati nome, parametri e descrizione del loro scopo. I metodi usati nel codice Python dell'applicazione hanno lo stesso scopo ma nomi diversi rispetto a come vengono presentati qui e nel paper di OpenSlide [3].

`open(filename)`	dato il nome dell'immagine, la apre e restituisce un `handle` su cui applicare gli altri metodi
`get_level_count(handle)`	restituisce il numero di livelli dell'immagine
`get_level_dimensions(handle, level)`	restituisce la risoluzione di un livello
`get_level_downsample(handle, level)`	restituisce il fattore di sottocampionamento per un certo livello
`read_region(handle, x, y, level, w, h)`	legge regione di immagine istologica a certo livello e la restituisce in formato ARGB (Alpha, Red, Green, Blue) non compresso

Table 1: Lista dei metodi di OpenSlide usati nell'applicazione. Di ognuno viene riportato nome, parametri e scopo, così come vengono riportati nel paper di OpenSlide [3].

Linguaggi di programmazione usati dall'applicazione

Per l'utilizzo di OpenSlide è stato scelto di usare l'API scritta in Python, per via della semplicità di scrittura del codice e in particolare di comunicazione tra la libreria e il resto dell'applicazione web. Pertanto, tutte le operazioni sulle immagini istologiche che coinvolgono OpenSlide sono contenute in unico script Python.

Data la natura web di questa applicazione, la maggioranza del codice è stato scritto in HTML, CSS, JavaScript e PHP:

- HTML è un linguaggio di markup, ovvero un linguaggio che definisce come deve essere impaginato e rappresentato un testo, in questo caso pagine web; costituisce la struttura base delle pagine;

- CSS definisce la formattazione della pagine HTML, permettendo di personalizzare il loro aspetto e inserire animazioni;

- JavaScript è un linguaggio di programmazione usato nelle applicazioni web per gestire il lato client, ovvero il browser che apre l'applicazione web in questo caso; gestisce gli aggiornamenti in tempo reale delle pagine e i vari eventi di interazione con esse, come cliccare pulsanti e scorrere immagini;

- PHP è un linguaggio di programmazione usato nelle applicazioni web per gestire il lato server, ovvero i server su cui si regge l'applicazione; in questo caso è il punto di comunicazione tra le pagine e le altre componenti dell'applicazione, ovvero il database e lo script Python.

È da riportare l'utilizzo di due librerie importanti utilizzate nei programmi JavaScript. Sono stati tutti scritti usando jQuery [12], una libreria leggera e supportata da molti browser che semplifica lo svolgimento di molti compiti, in particolare intervenire sugli elementi HTML di una pagina e gestire i vari eventi possibili. Grazie ad essa, i codici scritti sono composti, oltre che dalle funzioni dichiarate dallo sviluppatore, da una serie di event handler: funzioni che gestiscono la risposta a specifici eventi che possono accadere a specifici elementi della pagina HTML. Queste funzioni permettono di separare in maniera molto chiara come gestire le varie interazioni tra utente e pagina web. Questi handler vengono dichiarati specificando l'elemento HTML oggetto dell'evento, operazione effettuata in una sola riga grazie alla sintassi di jQuery.

La seconda libreria usata nel codice JavaScript è la Fetch API [13], un'interfaccia per ottenere risorse tramite richieste HTTP. In questa applicazione viene usata per richiedere allo

script PHP di chiamare le varie funzioni Python che si interfacciano con OpenSlide; viene specificato l'URL dello script aggiungendo come parametri di richiesta la funzione Python da chiamare ed eventuali parametri aggiuntivi. Uno dei vantaggi dell'utilizzo di questa interfaccia è come le risposte HTTP vengono rappresentate con degli oggetti di nomi Response da cui si può ottenere il vero corpo della risposta con un metodo.

Altri strumenti e programmi usati nell'applicazione

Il progetto è stato scritto sull'IDE Visual Studio Code e testato sul browser Google Chrome, con Github usato come supporto per il versionamento dell'applicazione.

Per poter testare tutto in locale, è stato scelto un web server Apache tramite XAMPP e PostgreSQL come DBMS (DataBase Management System). Questo software open-source per la creazione e gestione dei database è stato scelto per la semplicità nella creazione di database. Una delle opzioni dispnibili per effettuare questa operazione, infatti, prevede l'utilizzo di comandi di testo da eseguire nella shell di PostgreSQL, con cui si possono anche creare database eseguendo file contenenti istruzioni SQL. Questo metodo è stato molto utile nello sviluppo dell'applicazione, permettendo di concentrare il lavoro di implementazione del database in un unico file facilmente modificabile ed eseguibile.

In tutta l'applicazione, in particolare per scambiare dati tra codici e salvarli in file separati, è stato largamente adoperato JSON [14], un formato derivato da JavaScript ma supportato da svariati linguaggi di programmazione, tra cui quelli usati in questa applicazione. JSON si basa su una raccolta ordinata di coppie chiave-valore, dove la chiave deve essere una stringa mentre il valore può appartenere a diversi tipi di dato, come una stringa tra virgolette, un numero, altre raccolte di valori. In particolare, le immagini lette da OpenSlide vengono passate tramite JSON dopo essere state codificate sotto forma di stringa tramite il sistema di codifica per dati binari Base64. Questo sistema permette di convertire dati binari in stringhe di testo. Pertanto, Base64 può essere applicato su moltissimi tipi di dati e file diversi, tra cui immagini.

Infine, è da segnalare il vasto utilizzo di richieste HTTP nell'applicazione. Le richieste, di tipo GET e POST, consentono al lato client dell'applicazione di chiedere e reperire risorse, come file oppure risultati dell'esecuzione di file, al lato server. Le richieste di tipo POST permettono di tenere nascosti all'utente i parametri della richiesta, motivo per cui sono usate nel passaggio tra le pagine dell'applicazione, quando l'utente interagisce con il browser e ha accesso agli indirizzi delle pagine. Invece, nelle richieste di tipo GET il nome del file richiesto e i parametri della richiesta sono inclusi in un'unica stringa. Dato che proprio per questo motivo sono più semplici da creare rispetto alle richieste di tipo POST, sono usate nella trasmissione di dati tra i codici JavaScript dell'applicazione e quello Python che comunica con OpenSlide; siccome questa è un'operazione nascosta all'utente, nascondere i parametri della richiesta risulta inutile.

Progettazione del database

La prima fase della creazione del database è stata l'analisi dei requisiti forniti, da cui è stata ricavata una lista delle entità da rappresentare nel database e delle associazioni tra queste entità. Dopo uno studio specifico sulle modalità possibili per il salvataggio delle immagini (caricarle direttamente nel database sotto forma di file binario oppure salvare i file in locale e caricare solo i loro nomi), si è proceduto con la creazione dello schema ER sulla base delle informazioni raccolte. Dopodichè, da questo schema si è stilato quello relazionale. Infine, si è implementato il database in un file .sql contenente tutte le istruzioni necessarie alla sua creazione, da eseguire dalla shell del DBMS.

Analisi dei requisiti

Utenti

Ogni utente è identificato dalla sua email e ha un username e una password. Un utente può appartenere ad una di due categorie: Normale o Gestore. Un utente Normale può visualizzare porzioni di slide ad una certa risoluzione, creare annotazioni e modificare o eliminare solo le annotazioni che egli stesso ha creato. In aggiunta a queste funzioni, un utente Gestore può anche caricare e rimuovere slide, scaricare e rimuovere annotazioni create da qualsiasi utente, e aggiungere o togliere label.

Slide

Ogni slide è caricata da uno e un solo utente Gestore. Ad ogni slide sono associati uno o più layer, che corrispondono ai livelli piramidali dell'immagine. Questi sono identificati dalla slide e dal livello e hanno come dato aggiuntivo la risoluzione in pixel. Inoltre, ad ogni slide possono riferirsi o meno una o più annotazioni.

Annotazioni

Ogni annotazione può essere creata da uno e un solo utente ed è associata ad una e una sola slide, alla quale possono riferirsi o meno più annotazioni. Inoltre, può contenere o meno delle label, ed è identificata dalla ROI associata.

Una label è una etichetta in cui il creatore dell'annotazione può scrivere informazioni importanti al riguardo. Ogni label è identificata da un codice numerico ed ha un testo, una data di aggiunta e un colore.

La ROI è la regione della slide su cui viene effettuata l'annotazione. Viene identificata da un codice numerico, è associata al layer della slide al quale è stata fatta l'annotazione ed è formata da un poligono e una serie di tile. Il poligono viene rappresentato come lista di vertici, identificati dalla ROI e dalla posizione nell'ordine dei vertici. Ognuno di essi possiede una posizione assoluta, relativa all'immagine al layer con livello 0, e una posizione relativa, relativa all'immagine al layer usato nell'annotazione. La ROI deve anche tenere conto del fatto che il poligono può essere chiuso o meno.

Le tile sono i rettangoli in cui viene divisa la ROI. Sono identificati da un codice numerico e possiedono posizioni assoluta e relativa, in modo analogo ai vertici, una dimensione in pixel e una risoluzione identificata dal layer della slide.

Salvataggio delle immagini nel database

Il problema principale della progettazione del database è stato la scelta del metodo con cui gestire il salvataggio delle immagini. La questione centrale consiste nella posizione in cui salvare le immagini. Questo problema prevede due soluzioni principali:

1. salvare le immagini direttamente nel database dopo averle convertite in file binario, così da poterle ricavare direttamente senza dover comunicare con un file system esterno;

2. salvare i file in un file system esterno e caricare solo i loro nomi, i quali devono essere ricavati dal database e successivamente usati per aprire i file in locale.

I vantaggi e svantaggi più rilevanti di queste soluzioni cambiano da applicazione ad applicazione, pertanto considereremo il caso specifico di questa tesi, dove tutto viene creato, salvato ed eseguito in locale e vengono usati file di dimensioni considerevoli; infatti, la maggior parte delle immagini istologiche disponibili come esempi sono dell'ordine delle centinaia di megabyte. Nella maggioranza dei parametri di valutazione considerati, riassunti nella Tabella 2, la seconda opzione presenta i vantaggi più considerevoli:

- l'implementazione del database diventa più semplice a causa del diverso tipo di dato usato;

- la dimensione dei dati salvati nel database diminuisce sensibilmente, da file dell'ordine di grandezza di centinaia di megabyte a brevi stringhe, e di conseguenza anche il tempo necessario a caricarli e scaricarli;

- la scrittura del codice necessario al caricamento dei dati nel database viene semplificato, dato che non bisogna convertire le immagini in file binario.

L'aspetto principale in cui la prima opzione si rivela invece migliore è il mantenimento dell'integrità dei dati tra il database e il file system dove vengono salvate le immagini. Infatti, se si salvano solo i nomi dei file, bisogna scegliere un'unica cartella dove salvare tutte le immagini e assicurarsi che tali file non vengano rinominati, spostati o cancellati senza che il database venga aggiornato di conseguenza. Invece, salvando direttamente le immagini nel database, tali problemi non sussistono.

	Salvare immagini sotto forma di file binario	*Salvare nomi di immagini*
Tipo di dato usato nel database per immagini	VARCHAR (stringa a dimensione variabile): dato più semplice e leggero	stringa binaria: tipo di dato varia a seconda di DBMS (e.g. BYTEA in PostgreSQL, BLOB in MySQL)
Dimensioni di dati salvati in database	ridotte, ordine di grandezza: byte	elevate, ordine di grandezza: centinaia di megabyte
Scrittura del codice per caricamento dati in database	solo caricare nome di immagine	convertire immagine in formato binario e caricare stringa risultante in database
Mantenimento integrità tra database e file system	non necessario	necessario (possibilità di modifiche a immagini o a database senza sincronizzare modifiche)

Table 2: Confronto delle due opzioni per il salvataggio delle immagini nel database rispetto a vari aspetti d'utilizzo.

Le due opzioni sono state testate usando una versione del database contenente solo la tabella corrispondente alle slide e delle pagine scritte in HTML e PHP per caricare e visualizzare le immagini. Usando delle immagini normali in formato JPEG non si notano differenze tra le due opzioni; entrambe permettono di caricare un'immagine e di visualizzarla dopo aver ricavato le informazioni su di essa dal database in poco tempo. I problemi sorgono quando si passa alle immagini istologiche da usare nell'applicazione finita. Il salvataggio del nome del file non comporta differenze, a parte la comparsa di un piccolo ritardo non trascurabile dovuto al caricamento dell'immagine nella pagina HTML. Quando invece si tenta l'altra modalità, si manifestano una serie di errori generati dal codice PHP, tutti riconducibili alla notevole dimensione del file. Questi problemi si sono rivelati risolvibili, ma al costo di modificare vari parametri del codice e delle stesse impostazioni di XAMPP, un costo considerevolmente inferiore ai benefici dell'utilizzo di questa modalità.

Questi risultati, in aggiunta all'analisi riassunta nella Tabella 2, portano alla conclusione che per questa applicazione caricare solo i nomi dei file salvati in locale è nettamente più conveniente.

Scelte di progettazione concettuale per la definizione degli attributi delle entità

Prima di procedere allo schema ER bisogna decidere come implementare gli attributi che devono esprimere certe informazioni particolari, ovvero:

1. *Un utente può appartenere ad una di due categorie: Normale o Gestore*;
2. *La ROI deve anche tenere conto del fatto che il poligono può essere chiuso o meno*;
3. *posizione relativa, relativa all'immagine al layer usato nell'annotazione* (informazione necessaria sia per i vertici delle ROI che per le tile).

Per le prime due informazioni, la scelta da fare riguarda il tipo di dato da usare. Si è optato per attributi di tipo booleano; un'alternativa consisteva in un attributo di tipo numerico con 0 e 1 come unici valori possibili.

Per la terza, invece, il problema consiste in dove salvare l'informazione, nell'entità stessa oppure nell'associazione con l'entità che rappresenta il layer. Qui l'implementazione cambia a seconda dei casi. Come si vedrà nello schema ER rappresentato nella Figura 1, esiste una relazione che collega in modo diretto il layer e la tile, mentre i vertici sono legati alla ROI che a sua volta è collegata al layer. Nel caso della tile, quindi, l'attributo "posizione relativa" è stato associato alla relazione, mentre nel caso della ROI è stato messo sull'entità che rappresenta il vertice; per ricavare il layer associato alla posizione relativa sarà necessario effettuare un join in sede di query SQL.

Schema ER

Dall'analisi dei requisiti, dall'analisi sul salvataggio dell'immagine e le varie scelte di progettazione concettuale, è stata ricavata la lista di entità, attributi e relazioni da implementare espressa nel dizionario dei dati riassunto nelle Tabelle 3 e 4. Da questi dati è stato ricavato lo schema ER (Entity-Relationship) del database, che permette di progettare la base di dati ad un livello astratto, separato dai dettagli implementativi. Lo schema ER completo viene mostrato nella Figura 1.

Nello schema, gli utenti sono rappresentati dall'entità *Utente*, che ha come attributi email, username, password e l'attributo booleano, discusso nella sezione precedente, che determina se un tale utente è Gestore oppure no. L'entità *Slide* conserva formato e percorso all'interno del file system delle immagini; in questa implementazione viene salvato il percorso relativo ad una cartella dove vengono salvate tutte le immagini utilizzabili dall'applicazione. Inoltre, le istanze di *Slide* sono collegate agli *Utenti* che hanno caricato tali file. Alle slide sono collegate le varie istanze dell'entità *Layer*, per registrare numero e risoluzione dei vari livelli piramidali. Sono presenti infine entità per i vari elementi delle annotazioni, oltre all'entità *Annotazione* stessa: *Label*, *ROI*, *Tile* e *Vertice*. *Associazione* non presenta nessun attributo, in quanto tutte le informazioni necessarie su di essa si possono ricavare dalle sue associazioni: l'*Utente* autore, la *Slide* su cui è stata creata, le *Label* e la *ROI* associate. *Associazione* è identificata dalla *ROI* associata. L'entità *Label* conserva un identificatore numerico, il testo e il colore associato ad una certa label e la data della sua creazione; le sue istanze registrano anche l'*Utente* che le ha create tramite l'associazione *Aggiunge*. *ROI* presenta un identificatore numerico e l'attributo booleano spiegato nel sottocapitolo precedente che registra se il poligono della ROI è chiuso o meno. Le informazioni su tale poligono sono ricavabili dalle varie istanze di *Vertice* collegati ad una certa istanza di *ROI*. Per i vertici vengono registrati la loro posizione nell'ordine di un certo poligono, la loro posizione assoluta rispetto alla risoluzione massima dell'immagine e la posizione relativa al livello su cui è stata creata l'annotazione; come espresso nella sezione precedente, tale livello può essere ricavato ripercorrendo le associazioni tra *Vertice* e *ROI*, ovvero *Poligono*, e tra *ROI* e *Layer*, ovvero *Livello*. Ad identificare ogni vertice sono l'attributo sull'ordine e la *ROI* associata. L'entità *Tile* presenta un identificatore numerico, la sua dimensione e la sua posizione assoluta; la posizione relativa viene salvata nell'associazione tra questa entità e *Layer*, come discusso nel sottocapitolo precedente. Ogni istanza di questa entità è associata alla relativa *ROI*.

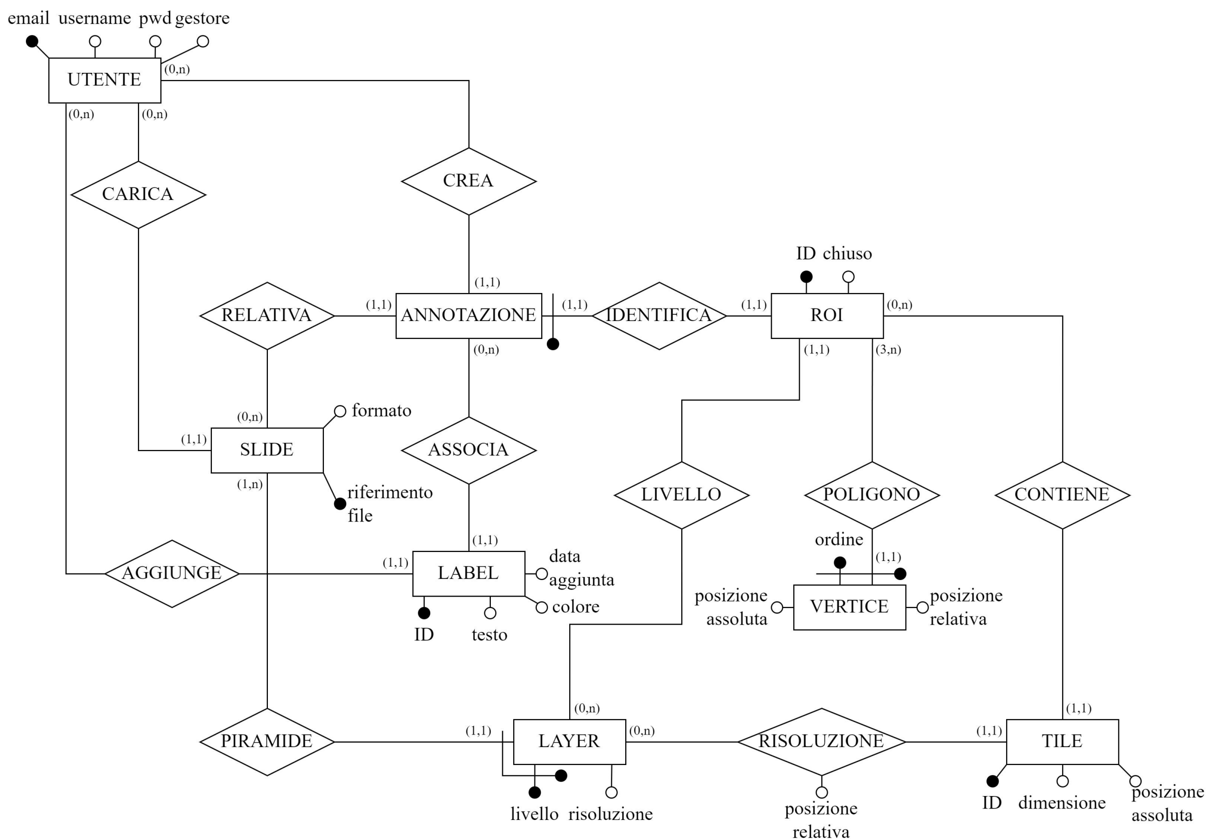

Figure 1: Schema ER del database dell'applicazione.

Entità	Descrizione	Attributi	Identificatore
Utente	Utenti che usano l'applicazione	email, user, pwd, gestore	email
Slide	File delle immagini istologiche	riferimento file, formato	riferimento file
Layer	Livelli piramidali delle immagini	livello, risoluzione	slide, livello
Annotazione	Annotazioni create sulle immagini		ROI
Label	Label associate alle annotazioni	ID, testo, colore, data aggiunta	ID
ROI	Regioni associate alle annotazioni	ID, chiuso	ID
Vertice	Vertici dei poligoni delle ROI	ordine, posizione assoluta, posizione relativa	ROI, ordine
Tile	Rettangoli in cui viene divisa la ROI	ID, dimensione, posizione assoluta	ID

Table 3: Lista delle entità presenti nello schema ER del database dell'applicazione.

Relazione	Descrizione	Entità coinvolte	Attributi
Carica	Associa una slide all'utente che l'ha caricata	Utente (0,n), Slide (1,1)	
Piramide	Associa un layer all'immagine corrispondente	Slide (1,n), Layer (1,1)	
Crea	Associa un'annotazione all'utente che l'ha creata	Utente (0,n), Annotazione (1,1)	
Relativa	Associa un'annotazione alla slide su cui è stata creata	Slide (0,n), Annotazione (1,1)	
Aggiunge	Associa una label all'utente che l'ha aggiunta	Utente (0,n), Label (1,1)	
Associata	Associa una label alla relativa annotazione	Label (1,1), Annotazione (0,n)	
Identifica	Associa un'annotazione alla ROI che la identifica	Annotazione (1,1), ROI (1,1)	
Livello	Associa una ROI al layer su cui è stata creata	ROI (1,1), Layer (0,n)	
Poligono	Associa una ROI ai vertici che costituiscono il suo poligono	ROI (3,n), Vertice (1,1)	
Contiene	Associa una tile alla ROI che la contiene	ROI (0,n), Tile (1,1)	
Risoluzione	Associa una tile al layer su cui è stata creata	Tile (1,1), Layer (0,n)	posizione relativa

Table 4: Lista delle associazioni presenti nello schema ER del database dell'applicazione.

Schema relazionale

Lo schema ER è stato successivamente tradotto nello schema relazionale mostrato nella Figura 2. Questo nuovo schema permette di visualizzare lo stesso database ad un livello più vicino all'implementazione. Infatti, in questo schema sia le entità che le associazioni vengono tradotte nelle tabelle che verranno poi dichiarate in PostgreSQL, esplicitando gli attributi che dovranno fare riferimento agli attributi identificatore di altre tabelle. Il verso delle frecce indica quale attributo deve fare riferimento all'altro in ogni associazione.

È da notare come nessuna associazione sia stata convertita in una tabella; dato che nello schema ER tutte le associazioni hanno cardinalità (1,1) almeno da un lato, al posto di una tabella vera e propria viene aggiunto un attributo alla tabella situata dal lato con cardinalità (1,1) che tiene traccia delle associazioni tra le varie istanze delle entità e che ha come nome il nome dell'entità associata. Per esempio, la tabella *Vertice* presenta l'attributo *ROI* per tenere traccia della ROI associata ad ogni vertice; tale attributo avrà come valori possibili gli identificatori delle varie istanze di *ROI*. Al di là di questa scelta, le tabelle presentate nella Figura 2 sono identiche a quelle incluse nello schema ER della Figura 1.

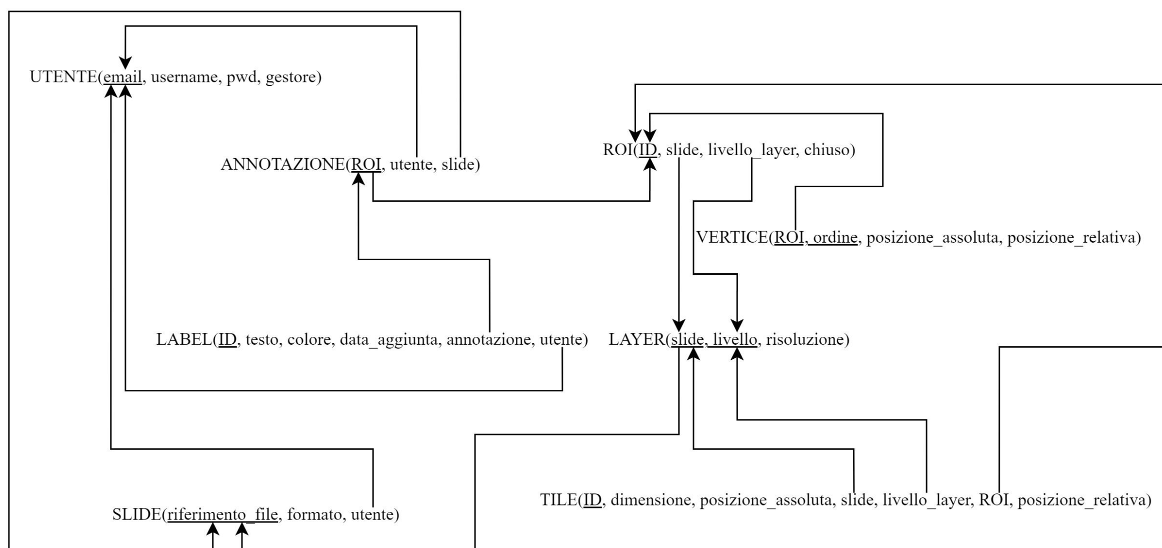

Figure 2: Schema relazionale del database dell'applicazione.

Applicazione web

L'applicazione web si divide in due pagine: una home page in cui selezionare l'immagine istologica da visualizzare oppure da caricare nel database, e una pagina viewer in cui viene aperta l'immagine che l'utente può visionare e ingrandire o rimpicciolire. Per lo scopo della tesi, queste sono le uniche feature implementate; sono state ignorate le altre funzionalità, come la creazione e il caricamento di annotazioni, la creazione di label e tile e la gestione delle immagini.

Home page

La home page è costituita da due sezioni, che corrispondono alle due funzioni principali della pagina. I vari pulsanti della pagina vengono chiamati con le lettere associate ad essi nella Figura 3.

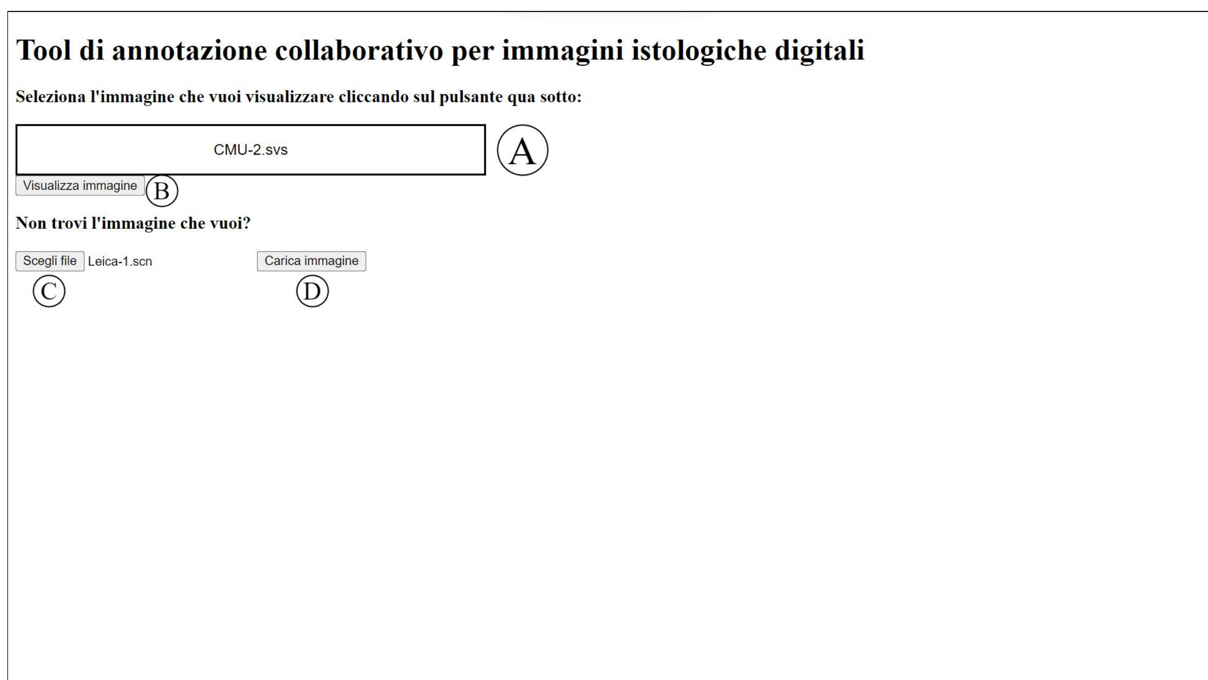

Figure 3: Home page dell'applicazione. Pulsante A: menù dropdown per la selezione dell'immagine; pulsante B: pulsante per l'apertura del viewer; pulsante C: pulsante per la selezione dei file; pulsante D: pulsante per il caricamento delle immagini. Le lettere associate ai vari pulsanti verranno usate nel corso della tesi per riferirsi a tali pulsanti.

La prima permette di selezionare l'immagine da visualizzare tra quelle presenti nel database. È composta dal menù dropdown per la selezione del file e dal pulsante per il caricamento dell'immagine nel viewer. Il menù è composto dal pulsante A, che ha come testo di default *Scegli l'immagine*, e dalla lista di file, che viene generata al caricamento della pagina ed è poi mostrata o nascosta a seconda delle azioni dell'utente. Il pulsante B (*Visualizza immagine*), quello per aprire il viewer, è disabilitato di default ed è racchiuso all'interno di un form: un elemento della pagina HTML che include vari elementi attraverso i quali inserire dei dati di input e che punta ad un'altra pagina HTML. Quando il pulsante incaricato di aprire quest'ultima pagina viene cliccato, vengono presi i dati di input e inseriti come parametri in una richiesta HTTP con la quale viene aperta la nuova pagina. In questo caso la pagina puntata è la pagina viewer, il nome del file da aprire viene preso da un elemento della pagina e la richiesta effettuata è di tipo

POST. Dato che l'elemento HTML che conserva il nome del file non deve essere modificato manualmente dall'utente, non viene mostrato nella pagina.

La seconda permette di caricare nuove immagini nel database. Anche questa contiene due pulsanti. È composta da un unico form che punta alla pagina stessa. Contiene il pulsante C per selezionare il file (*Scegli file*) con a fianco *Nessun file selezionato* come testo di default, un elemento nascosto per salvare separatamente il formato del file scelto e il pulsante D per inviare la richiesta POST (*Carica immagine*). Quest'ultimo pulsante viene non può essere cliccato finchè non viene selezionato un'immagine salvata in uno dei formati accettati da OpenSlide.

A questa pagina HTML sono affiancati un codice JavaScript che gestisce le operazioni da effettuare in risposta alle varie interazioni con la pagina e un codice PHP che si interfaccia col database per ottenere la lista di immagini disponibili e caricarne di nuove. Queste ultime funzioni vengono chiamate da uno script PHP inserito all'interno della pagina HTML ed eseguito al suo caricamento.

Caricamento della pagina

La prima operazione che viene effettuata al caricamento della pagina è l'esecuzione dello script PHP incluso nella pagina che chiama le due funzioni del file PHP: `getAvailableImages` e `uploadNewImage`. La prima si occupa di ottenere la lista di immagini presenti nel database e viene chiamata sempre. La seconda carica una nuova immagine nel database, della quale vengono forniti nome e formato come parametri della funzione (*filename* e *format*), e viene chiamata se tra i parametri della richiesta POST con cui viene aperta la pagina è presente il nome di un file.

Entrambe le funzioni seguono una procedura comune: si connettono al database, eseguono una o più query ed elaborano i dati restituiti. La funzione `getAvailableImages` esegue una query per ottenere i nomi delle immagini presenti nel database, consistente in una proiezione della tabella *SLIDE* sull'attributo *riferimento_file*:

$$\Pi_{riferimento_file}(SLIDE) \Longrightarrow \texttt{SELECT riferimento_file FROM SLIDE}$$

Con i risultati della query, la funzione costruisce e ritorna il codice HTML corrispondente alla lista di file.

La funzione `uploadNewImage` esegue due query. La prima verifica la presenza nel database dell'immagine; viene effettuata una selezione sulla tabella *SLIDE*, usando come condizione il confronto tra i vari nomi di file salvati nel database e *filename*:

$$\sigma_{riferimento_file=filename}(SLIDE) \Longrightarrow$$
$$\Longrightarrow \texttt{SELECT * FROM SLIDE WHERE riferimento_file=filename}$$

È da notare come, sebbene il codice JavaScript si occupi già di impedire il caricamento di file già presenti, la funzione faccia un controllo ulteriore sulla stessa condizione. Questa scelta è stata effettuata poichè, se si carica un file per poi ricaricare la pagina senza uscire o selezionare un nuovo file, i parametri della richiesta POST rimangono; pertanto si verifica nuovamente la presenza del file nel database per evitare di ripetere la query di inserimento, operazione che provocherebbe un errore a causa della scelta di *riferimento_file* come identificatore della tabella *SLIDE*.

La seconda query di `uploadNewImage` serve a caricare effettivamente la nuova immagine nel database. L'attributo *utente* è stato impostato ad un valore di default per questa versione dell'applicazione:

```
INSERT INTO SLIDE VALUES(filename, format, 'email')
```

Dopo il codice PHP viene eseguito il codice JavaScript, che definisce i vari event handler usati, che gestiscono il caricamento della pagina, gli eventi di click e la selezione di file.

Viewer

La pagina viewer costituisce il centro dell'applicazione: è la pagina che ospita la maggior parte delle funzionalità, che ha richiesto la quantità maggiore di lavoro e che può ospitare la netta maggioranza delle funzioni implementabili in futuro. Permette di visualizzare, ingrandire, rimpicciolire e muovere l'immagine, mentre la pagina regola automaticamente la risoluzione mostrata per mantenere quella ottimale, né troppo dettagliata né troppo sgranata.

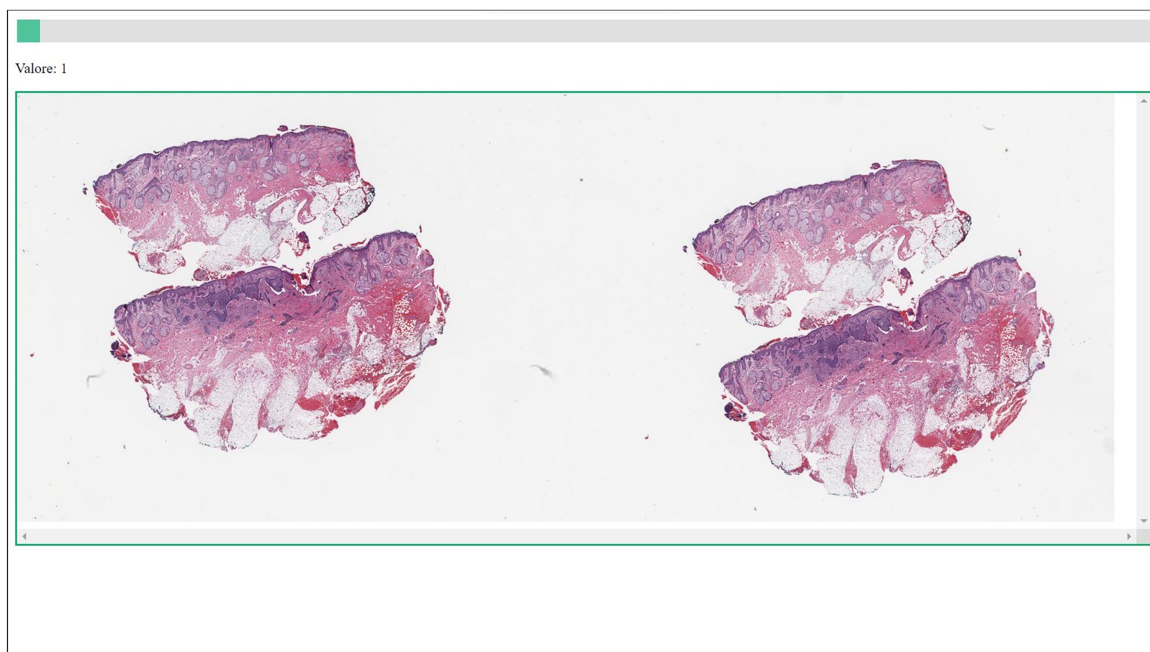

Figure 4: Pagina viewer dell'applicazione.

La pagina viewer è composta da uno slider per regolare lo zoom, una riga di testo per mostrare il livello attuale di zoom e un box dove viene visualizzata l'immagine. L'utente può muovere lo slider per ingrandire o rimpicciolire l'immagine e muoversi su di essa tramite le scrollbar che appaiono ai lati. Un elemento HTML nascosto conserva il nome dell'immagine da aprire, in modo da poter essere recuperato facilmente senza che l'utente possa tentare di modificare tale elemento.

Questa pagina si appoggia a un codice JavaScript che gestisce le interazioni dell'utente con la pagina e a uno script Python che esegue le operazioni su OpenSlide. Questi due codici comunicano tramite un codice PHP.

Durante l'utilizzo del viewer, il codice JavaScript prepara delle richieste GET per il codice PHP, passando come parametri la funzione Python che vuole usare ed eventuali dati da fornire in input a tale funzione. Il codice PHP costruisce, a partire dai dati della richiesta HTTP, un comando per eseguire il codice Python da riga di comando. Dato che l'applicazione è stata sviluppata su Windows, il comando rispetta la seguente struttura:

`python` ⟨filepath del codice Python⟩ ⟨parametri da fornire a codice⟩

Normalmente questi comandi vengono digitati nella shell di Windows per eseguire il codice Python desiderato e stampare l'output a schermo. Tuttavia, PHP dispone di una funzione, `shell_exec`, per eseguire comandi da riga di comando e salvarne l'output in una variabile. Questo è il metodo usato in questa applicazione: viene indicato il filepath assoluto del codice Python, mentre i parametri per il codice sono la funzione Python desiderata e i dati necessari alla funzione; queste informazioni vengono ricavate dai parametri della richiesta GET inviata dal codice JavaScript. Il codice Python viene eseguito e questo restituisce il suo risultato sotto

forma di stringa JSON. Questa stringa viene ricevuta dal codice PHP, il quale restituisce la stessa stringa al codice JavaScript come risposta alla richiesta iniziale.

È stato scelto di impiegare un codice PHP come interfaccia tra il codice JavaScript e il codice Python, anziché far chiamare direttamente le funzioni Python dal codice JavaScript. Entrambe le scelte richiedono la creazione di richieste HTTP da parte del codice JavaScript con cui chiedere l'esecuzione di altri file. Tuttavia, la comunicazione JavaScript-PHP risulta più facile da realizzare grazie alla Fetch API e al fatto che non è necessario modificare il codice Python con librerie speciali. Realizzare la comunicazione diretta JavaScript-Python, invece, richiede sia di usare librerie per realizzare richieste HTTP via JavaScript che di modificare il codice Python per includere e utilizzare Flask: un framework, ovvero un'architettura di supporto, per realizzare applicazioni web con Python.

Questo ulteriore grado di complessità nell'implementazione ha portato alla scelta di PHP come interfaccia tra JavaScript e Python. L'opzione è poi stata testata, rivelandosi di semplice utilizzo.

Vengono inoltre usati un file di testo dove lo script Python registra le caratteristiche delle porzioni di immagine prodotte e di un file JSON dove viene salvato il nome dell'immagine da visualizzare.

Caricamento della pagina

Al caricamento della pagina, il nome dell'immagine da aprire viene ricavato dai parametri della richiesta POST con cui è stata aperta la pagina e salvato nell'elemento HTML nascosto. Dopodiché, si attiva l'event handler del codice JavaScript che gestisce il caricamento della pagina per preparare il viewer.

Il nome del file viene racchiuso in un JSON passato come parametro della prima richiesta GET per il codice PHP. Viene chiamata la funzione Python per generare la prima immagine da visualizzare, ovvero quella completa presa al livello piramidale massimo, a risoluzione minima. Il codice Python restituisce l'immagine, la sua risoluzione, il suo livello e la lista dei fattori di sottocampionamento di tutti i livelli. L'immagine viene inserita nel box e vengono inizializzati tutti i dati necessari al corretto funzionamento dell'applicazione. Tali dati sono elencati e approfonditi nel paragrafo sulle variabili globali.

Infine, lo slider viene impostato al valore minimo, ovvero 100. Il viewer è pronto per l'utilizzo.

Inserimento dello zoom

L'utente può modificare lo zoom dell'immagine trascinando il cursore dello slider avanti e indietro. Ogni volta che tale cursore viene spostato si attiva un event handler del codice JavaScript, che gestisce la regolazione della grandezza dell'immagine, garantisce che il punto dell'immagine che si trova al centro del box rimanga lo stesso mentre viene spostato il cursore ed effettua gli eventuali passaggi a livelli superiori o inferiori. Quando l'immagine diventa più grande del box, appaiono ai lati di quest'ultimo delle scrollbar che permettono di spostare manualmente l'immagine. Se si regola lo zoom dopo aver spostato l'immagine, il punto da mantenere al centro del box viene ricalcolato.

Il valore numerico associato alla posizione del cursore all'interno della slider viene salvato come livello di zoom assoluto, rispetto all'immagine iniziale. Da questo si ricava il livello relativo, calcolato rispetto alla porzione di immagine corrente. Dopodiché, si confronta il livello assoluto con i diversi valori di zoom soglia associati ai vari livelli dell'immagine per determinare quale livello piramidale deve essere visualizzato. Infine, si calcolano le coordinate del punto dell'immagine visualizzato nell'angolo in alto a sinistra del box e si ricalcola il punto da mantenere al centro del box se necessario.

Dopo queste fasi si gestiscono i passaggi di livello. Se il livello da visualizzare è uguale a quello attuale bisogna solo ridimensionare e spostare l'immagine. Altrimenti si effettua una serie di operazioni valide per tutti i passaggi di livello, a risoluzioni sia maggiori che minori, sia di livello singoli che di livelli multipli. Se la nuova porzione da visualizzare non è già salvata in locale, vengono calcolati i vari dati necessari alla sua creazione per poi mandare la richiesta GET al codice Python per leggere l'immagine. La funzione richiesta fornisce la nuova immagine a partire dai dati forniti in input, racchiusi in un JSON:

- le coordinate del vertice in alto a sinistra della nuova porzione, calcolate relativamente all'immagine iniziale, a risoluzione minima;
- i livelli attualmente visualizzato e nuovo;
- le dimensioni della porzione, relative al livello attuale.

Il codice Python restituisce la nuova immagine, insieme alla sua risoluzione e al suo livello. La porzione viene inserita nel box e poi è ridimensionata e spostata a seconda del caso. Dopo queste operazione l'utente può riprendere a usare lo slider e regolare lo zoom come prima.

Panoramica delle variabili globali

Il codice JavaScript si basa su una vasta serie di variabili e strutture dati globali, che possono essere lette e scritte da tutte le funzioni del codice. Tutti questi dati vengono inizializzate durante il caricamento della pagina e aggiornate in modo appropriato durante l'utilizzo del viewer. Possono essere divise in due gruppi principali in base alle informazioni che registrano: quelle relative alla porzione attualmente visualizzata e quelle relative ai vari livelli dell'immagine.

Per il primo gruppo, un'importante proprietà delle immagini visualizzate consiste nella differenza tra le loro dimensioni originali e quelle del box. Pertanto, vengono salvate entrambe le coppie di valori: quella originale viene conservata da ***ogDim***, mentre quella che assume dopo essere stata ridimensionata da ***stdDim***; il rapporto tra queste due dimensioni è salvato in ***dimFactor***. Viene inoltre conservato il punto dell'immagine da mantenere al centro del box, in ***currentCenter***; queste coordinate vengono calcolate rispetto alle dimensioni di ***stdDim***. Quando l'utente interagisce con le scrollbar, alla modifica successiva del valore dello slider tale punto andrà ricalcolato: la variabile ***centerToChange*** tiene conto di questa possibilità. Infine, vengono usate tre variabili per occuparsi dei livelli di zoom: ***baseZoom*** conserva il limite inferiore per il livello attuale, ***currentZoom*** l'ultimo valore di zoom inserito e ***relativeZoom*** il nuovo valore appena impostato tramite lo slider.

Il secondo gruppo registra informazioni analoghe a quelle appena citate, ma dalla prospettiva dell'intera serie di livelli piramidali. Il livello massimo viene salvato in ***maxLevel*** e quello attuale in ***curLevel***. I vari fattori di sottocampionamento vengono registrati in ***downsamples***. Per gli intervalli di zoom associati ad ogni livello viene registrato il limite inferiore, sia il suo valore assoluto rispetto all'immagine iniziale che quello relativo al livello precedente, in ***levelZoomLimits***. Questi valori vengono generati a partire dai fattori di sottocampionamento: i fattori vengono tradotti nei limiti assoluti, visto che rappresentano il rapporto tra la risoluzione massima e quella di un certo livello, mentre quelli relativi vengono calcolati tramite rapporti tra il fattore di un certo livello e quello del livello immediatamente superiore. Man mano che vengono generate nuove porzioni, le immagini e le informazioni relative ad esse ritornate dal codice Python vengono mantenute in ***imageData***, per essere usate nei passaggi a livelli più alti, e le informazioni sulle coordinate e dimensioni delle nuove porzioni rispetto a quelle precedenti vengono calcolate e salvate in ***levelData***.

Nella Tabella 5 viene proposto un riassunto delle variabili esposte in questo paragrafo, con informazioni aggiuntive sui loro tipi.

Nome	Informazioni	Tipo
baseZoom	limite inferiore di zoom per il livello attuale	numero frazionario positivo
centerToChange	determina se punto dell'immagine da mantenere al centro del box va ricalcolato o meno	valore booleano (`true` se centro deve essere ricalcolato, `false` altrimenti)
curLevel	livello attuale dell'immagine	numero naturale
currentCenter	coordinate di punto dell'immagine da mantenere al centro del box, calcolate rispetto a ***stdDim***	coppia di numeri frazionari positivi
currentZoom	zoom attuale (non viene aggiornata subito con nuovi valori di slider)	numero frazionario positivo
dimFactor	rapporto tra i valori di ***stdDim*** e ***ogDim***	numero frazionario positivo
downsamples	fattori di sottocampionamento dei vari livelli dell'immagine	numeri frazionari positivi, uno per ogni livello
imageData	informazioni ritornate dal codice Python per l'ultima porzione visualizzata di ogni livello	JSON, uno per ogni livello tranne quello 0
levelData	dato livello *x*: dimensione della porzione visualizzata per *x* rispetto a livello *x-1*, ***ogDim*** per *x*, rapporto tra coordinate di vertice in alto a sinistra di porzione visualizata per *x* rispetto a livello *x* e ***ogDim*** per il livello *x*	tre coppie di numeri frazionari positivi per ogni livello: $(size, ogDim, ratio)$
levelZoomLimits	limite inferiore di zoom per ogni livello, calcolato sia rispetto all'immagine iniziale (*absolute*) che rispetto a quella visualizzata per il livello precedente (*relative*)	coppia di numeri frazionari positivi per ogni livello *(absolute, relative)*
maxLevel	livello massimo dell'immagine	numero naturale
ogDim	dimensione originale dell'immagine attualmente mostrata, ritornata dal codice Python	coppia di numeri frazionari positivi
relativeZoom	zoom attuale (viene aggiornato subito con nuovi valori di slider)	numero naturale
stdDim	dimensione dell'immagine attualmente mostrata dopo essere stata ridimensionata per il box	coppia di numeri frazionari positivi
N.B: tutte le coordinate salvate sono intese come coppie di coordinate *(x,y)*; analogamente, tutte le dimensioni sono intese come coppie di valori *(larghezza, altezza)*		

Table 5: Panoramica delle variabili e strutture dati globali principali usati dal codice JavaScript del viewer, con nome, descrizione e tipo di dato.

Lettura delle immagini

Essendo l'unico codice con accesso ad OpenSlide, il codice Python si occupa interamente della lettura delle porzioni di immagine da visualizzare a partire dai file originali. Il metodo che effettua questa operazione è get_image.

Il compito principale del metodo, oltre a leggere l'immagine, consiste nell'adattare le coordinate e le dimensioni ricevute in input dal codice JavaScript per il metodo di OpenSlide che genera la porzione, ovvero read_region. Infatti, coordinate e dimensioni vengono calcolati dal codice JavaScript rispetto all'immagine iniziale e al livello attuale rispettivamente, mentre il metodo di OpenSlide richiede che gli stessi dati siano calcolati rispetto al livello 0 (a risoluzione massima) dell'immagine e al livello nuovo da raggiungere rispettivamente. Questa operazione di scala viene effettuata sfruttando altri metodi di OpenSlide e la coppia di livelli ricevuta in input.

Dato $i \in [0, MAX]$ il livello attualmente visualizzato e $j \in [0, MAX]$ il livello nuovo da visualizzare, chiamando d_i e s_i la risoluzione e il fattore di sottocampionamento per i rispettivamente e riferendosi alle coordinate e alle dimensioni inviate al codice Python con C e D rispettivamente per i valori assunti prima dell'operazione di scala e con C' e D' per i valori assunti dopo, vengono effettuati i seguenti calcoli:

$$C' = C \cdot \frac{d_0}{d_{MAX}} \tag{1}$$

$$D' = D \cdot \frac{s_i}{s_j} \tag{2}$$

Come già mostrato nella Tabella 1, le risoluzioni vengono ottenute usando il metodo di OpenSlide get_level_dimensions, mentre i fattori di sottocampionamento vengono ricavati da get_level_downsamples.

Una volta ottenuti questi dati, vengono dati come input a read_region, che restituisce la nuova porzione. Questa viene codificata in una stringa tramite Base64 e racchiusa in un JSON insieme alla dimensione della porzione e al suo livello. Questo JSON è il risultato di get_image.

Un caso particolare è la creazione della prima immagine da visualizzare, gestita dal metodo get_starter_image. Dato che alla sua apertura il viewer conosce solo il nome dell'immagine, questo è l'unico dato che invia. Il codice Python ottiene il numero e le dimensioni del livello dai metodi di OpenSlide get_level_count e get_level_dimensions. Al JSON ricevuto in output da get_image viene poi aggiunta la lista dei fattori di sottocampionamento. Il JSON viene infine restituito dal codice Python.

Inserimento dell'immagine

Con i dati ritornati dal codice Python, il codice JavaScript può sostituire l'immagine mostrata nel box. Questo avviene tramite il metodo setSlide, che inserisce l'immagine e imposta il valore di diverse variabili globali grazie ai valori ritornati dal codice Python.

Anziché un'immagine salvata in locale di cui inserire il filepath nella pagina HTML, abbiamo un'immagine espressa in una stringa creata con codifica Base64. Il codice Python, infatti, converte l'immagine in questa stringa per poterla trasmettere al codice JavaScript. È necessario quindi fornire informazioni aggiuntive alla pagina su come interpretare la stringa rappresentante l'immagine. Questa viene impostata con un URL che segue lo schema "data", definito nel Request For Comments (RFC) 2397 [15] e che permette di includere piccole quantità di dati in modalità in-line, come se fossero in realtà risorse esterne. Il formato generale è data:[<mediatype>][;base64],<data>:

- la dicitura data indica lo schema seguito;

- il parametro opzionale `<mediatype>` indica il formato dei dati riportati sotto forma di uno degli Internet media type definiti dall'Internet Assigned Numbers Authority (IANA) [16], in questo caso `image/png`;
- la dicitura `;base64` indica che i dati riportati sono codificati con Base64;
- il parametro `<data>` indica i dati da visualizzare.

Zoom dell'immagine

Una delle operazioni più importanti del viewer è l'ingrandimento e spostamento dell'immagine, da effettuare in modo tale che il punto dell'immagine che si trova al centro del box rimanga lo stesso durante l'intero spostamento del cursore dello slider. Queste operazioni vengono effettuate dal metodo JavaScript `zoomMoveImage`, che riceve in input il livello di zoom inserito tramite lo slider e sfrutta le coordinate del punto da mantenere al centro salvate in ***currentCenter***.

L'immagine viene ingrandita moltiplicando le dimensioni specificate in ***stdDim*** per il livello di zoom inserito in input, valore che viene salvato in ***currentZoom***. Dopodiché, viene spostata l'immagine, ma anziché intervenire direttamente su di essa si agisce sulle scrollbar laterali del box. Ad ognuna di queste è associato un valore numerico rappresentante la posizione attuale del cursore in pixel.

Algorithm 1 zoomMoveImage

Require: *zoom*: livello relativo di zoom inserito tramite slider

1: $\boldsymbol{currentZoom} \leftarrow zoom$
2: $image.width \leftarrow \boldsymbol{stdDim}.left * zoom$
3: $center \leftarrow currentCenter * currentZoom$
4: $imageRange \leftarrow image - box$
5: $shiftedCenter \leftarrow center - box/2$
6: $percent \leftarrow shiftedCenter/imageRange$
7: $maxScroll \leftarrow$ valori massimi assumibili dalle scrollbar del box nelle due dimensioni (x,y)
8: valori (x,y) delle scrollbar del box $\leftarrow maxScroll * percent$
9: **if** $\boldsymbol{curLevel} = \boldsymbol{maxLevel}$ **then**
10:     **if** $maxScroll.x = 0$ **then**
11:         $\boldsymbol{currentCenter}.x \leftarrow \boldsymbol{stdDim}.x/2$
12:     **end if**
13:     **if** $maxScroll.y = 0$ **then**
14:         $\boldsymbol{currentCenter}.y \leftarrow \boldsymbol{stdDim}.y/2$
15:     **end if**
16: **end if**

Il calcolo della posizione da impostare su cui è fondato l'Algoritmo 1, effettuato su entrambi gli assi di movimento, si basa sull'analogia tra la posizione massima assumibile dalle scrollbar (*maxScroll*) e le coordinate massime assumibili dal punto dell'immagine che si trova al centro del box, calcolate mettendo a 0 le coordinate minime (*imageRange*). Infatti, come mostra la Figura 5, man mano che si sposta l'immagine e cambia la porzione mostrata nel box, le coordinate del punto al centro del box rispetto all'immagine completa (incrocio delle linee rosse) e la posizione delle scrollbar (rettangoli blu) assumono la stessa percentuale rispetto ai loro valori massimi. Su questo fenomeno si basa `zoomMoveImage`.

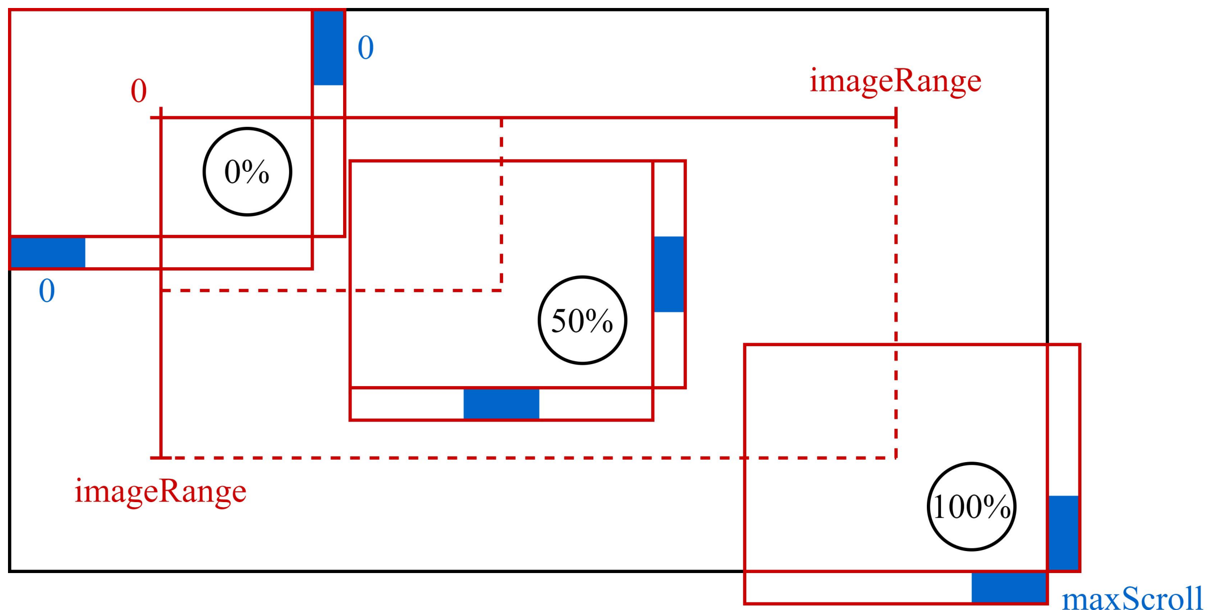

Figure 5: Schema che rappresenta il calcolo alla base del metodo `zoomMoveImage`. Vengono rappresentati tre casi di una stessa immagine (rettangolo nero) della quale vengono mostrate porzioni diverse nel box (rettangoli rossi grandi), per uno stesso livello di zoom. Le linee rosse continue rappresenta il valore di *imageRange*, mentre quelle tratteggiate rappresentano i valori scalati di ***currentCenter*** nei tre casi. I rettangoli blu rappresentano le posizioni delle scrollbar rispetto alla posizione massima *maxScroll* (lato corto a destra e in basso dei rettangoli rossi adiacenti ai box). I numeri nei cerchi sono le percentuali (***currentCenter*** scalato)/*imageRange*=(posizione scrollbar)/*maxScroll* dei tre casi.

La prima operazione effettuata dall'Algoritmo 1 dopo aver ingrandito l'immagine consiste nell'adattare ***currentCenter*** in modo tale che rappresenti un valore relativo all'intervallo $[0, imageRange]$. Questo viene realizzato scalando ***currentCenter*** al livello di zoom dato in input e sottraendo a entrambe le coordinate valori pari alla metà delle dimensioni del box (righe 3,5). Si calcola la percentuale di queste coordinate adattate rispetto a *imageRange* (riga 6); questa percentuale viene poi moltiplicata per *maxScroll* per ottenere il valore da impostare per le scrollbar (riga 8). Infine, se una o entrambe le scrollbar scompaiono, ovvero se i loro valori massimi vanno a 0, viene reimpostata la componente corrispondente di ***currentCenter*** a metà componente di ***stdDim***, ovvero a metà immagine (righe 9-16).

Calcolo delle coordinate per la lettura delle immagini

I passaggi tra livelli pongono un problema fondamentale per questa e per le altre operazioni che coinvolgono coordinate di certi punti dell'immagine, come il salvataggio delle annotazioni. Le coordinate che si possono ottenere dagli elementi HTML della pagina sono relative alla porzione attualmente visualizzata, che non comprende l'immagine intera ed appartiene ad un livello diverso da quello a risoluzione minima. Invece, il codice JavaScript deve mandare le coordinate rispetto all'immagine iniziale, ovvero l'immagine completa a risoluzone minima. Il metodo `getCoords` si occupa di adattare le coordinate da un sistema di riferimento all'altro.

Questo metodo si basa sulle informazioni salvate in ***levelData***. Per ogni livello x attraversato durante l'utilizzo del viewer, vengono salvati:

- *size*: la dimensione della porzione visualizzata per x, calcolata rispetto alla porzione visualizzata per $x+1$;

- *ogDim*: la dimensione originale (***ogDim***) della porzione visualizzata per *x*;
- *ratio*: il rapporto, per entrambe le componenti, tra le coordinate del vertice in alto a sinistra della porzione visualizzata per *x-1*, calcolate rispetto alla porzione visualizzata per *x*, e la dimensione ***ogDim*** della porzione visualizzata per *x*.

Queste informazioni vengono usate da un algoritmo ricorsivo che permette di adattare le coordinate non solo al livello massimo, necessario per salti a livelli inferiori, ma anche a livelli intermedi, utile per salti a livelli superiori.

Algorithm 2 getCoords

Require: *liv*: livello; *ratio*1: rapporto tra coordinate di vertice in alto a sinistra di nuova porzione e ***ogDim*** per livello *liv*; *cap*: livello da raggiungere, valore di default ***maxLevel***

1: **if** $liv = cap$ **then**
2: $\quad curOgDim \leftarrow \boldsymbol{levelData}[liv].ogDim$
3: $\quad$ **return** $curOgDim * ratio1$
4: **end if**
5: $dim \leftarrow \boldsymbol{levelData}[liv].size$
6: $aboveLevel \leftarrow \boldsymbol{levelData}[liv+1]$
7: $aboveOgDim \leftarrow aboveLevel.dim$
8: $aboveRatio \leftarrow aboveLevel.ratio$
9: $ratio2 \leftarrow dim/aboveOgDim$
10: $shiftedRatio \leftarrow ratio1 * ratio2$
11: $summedRatio \leftarrow aboveRatio + shiftedRatio$
12: **return** $getCoords(liv+1, summedRatio, cap)$

L'algoritmo riceve in input il livello di partenza *liv*, il valore di *ratio* per *liv*, chiamato *ratio1*, e il livello *cap* rispetto al quale adattare le coordinate, che rappresenterà il caso base dell'algoritmo. I valori di *ratio1* vengono calcolati prima della chiamata del metodo, calcolando il rapporto tra le coordinate da adattare e il valore attuale di ***ogDim***.

I calcoli di `getCoords` si basano su quattro valori. Come già anticipato, le coordinate da adattare (*coord*, segmento 1 della Figura 6) sono calcolate rispetto alla dimensione attuale della porzione (***ogDim***, segmento 2 della Figura 6). Il fattore per il quale scalare le coordinate equivale al rapporto tra la dimensione della porzione attuale rispetto a quella visualizzata per il livello immediatamente superiore (*dim*, segmento 3 della Figura 6) e la dimensione di quest'ultima porzione (*aboveOgDim*, segmento 4 della Figura 6). La discrepanza tra questi due sistemi di riferimento deriva dal fatto che il metodo `get_image` regola le dimensioni che riceve in input per fare in modo che prima e dopo il passaggio di livello venga visualizzata la stessa parte di immagine nonostante le diverse risoluzioni dei livelli; a causa di questo, ***ogDim*** e *dim* sono valori numericamente diversi che rappresentano la stessa grandezza.

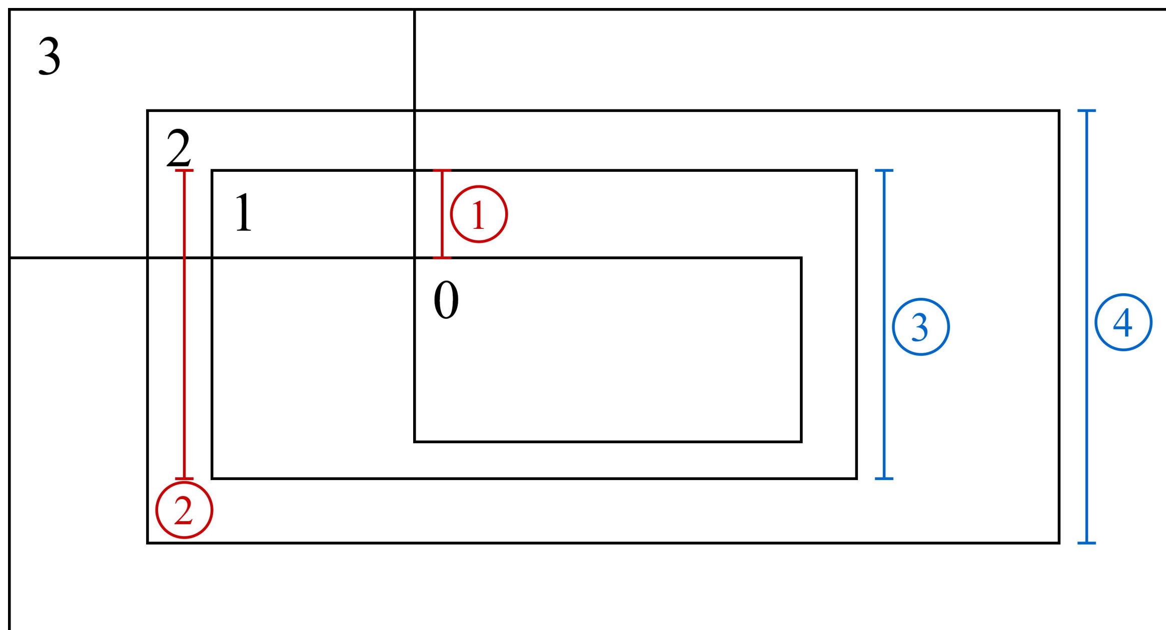

Figure 6: Schema che rappresenta il calcolo alla base del metodo `getCoords`. Viene illustrato il caso di un'immagine istologica a 4 livelli, per la quale si vogliono adattare le coordinate calcolate al livello 1 per leggere la porzione da visualizzare per il livello 0. Le grandezze indicate in rosso sono calcolate rispetto a ***ogDim*** del livello 1: il segmento 1 rappresenta le coordinate da adattare (*coord*), mentre il segmento 2 raffigura ***ogDim***. Le grandezze blu, invece, sono calcolate rispetto al valore ***ogDim*** del livello 2 (*aboveOgDim*): il segmento 3 mostra la dimensione della porzione per il livello 1 calcolata rispetto al livello 2 (*dim*), mentre il segmento 4 rappresenta *aboveOgDim*.

Nel metodo (Algoritmo 2, righe 5-12), si ricavano i valori di *dim* e *aboveOgDim*, oltre al valore di *ratio* salvato per il livello immediatamente superiore (*aboveRatio*), da ***levelData***. Si scala *ratio1* per il livello superiore, moltiplicandolo per il rapporto tra *dim* e *aboveOgDim* (*ratio2*). Questo rapporto scalato (*shiftedRatio*) viene sommato ad *aboveRatio* per ottenere il valore del parametro *ratio1* da passare alla chiamata ricorsiva (*summedRatio*). Questa riceverà come parametri il valore del livello immediatamente superiore (*liv+1*), *summedRatio* e *cap*.

Al raggiungimento del caso base (Algoritmo 2, righe 1-4), si moltiplica *ratio1* per il valore di ***ogDim*** per il livello raggiunto, così da ottenere le coordinate adattate al livello *cap* da restituire.

Passaggio a livello piramidale inferiore

L'algoritmo per gestire i passaggi a livelli inferiori, eseguito all'interno dell'event handler associato agli input inseriti tramite movimento del cursore dello slider, è il più grande dell'intero codice JavaScript e gestisce tutti i possibili casi di questo fenomeno. Possiamo infatti identificare i salti di livello singoli, nei quali si passa al livello immediatamente inferiore, e quelli multipli, dove alcuni livelli vengono saltati. Teoricamente la loro gestione è molto simile, ma il fattore che rende questa distinzione significativa è il riempimento di ***levelData***. Infatti, ad ogni salto di livello singolo è possibile salvare i dati su coordinate e dimensioni calcolate per la nuova porzione e salvarle nella struttura dati, così che i dati siano presenti per tutti i livelli precedenti a quello attuale e `getCoords` possa funzionare correttamente in tutti i casi. Sotto questo aspetto i salti multipli sono invece casi critici, perché non potendo sapere i dati sulle porzioni dei livelli saltati bisogna individuare un altro modo per inserire tutti i dati necessari in ***levelData***.

Il procedimento ideato per governare tutte queste evenienze prevede diversi passaggi, svolti dai diversi blocchi identificati nell'Algoritmo 3:

1. calcolare tutti i livelli di zoom necessari ai passaggi successivi;
2. ingrandire l'immagine fino al limite superiore di zoom del livello attuale per poter ottenere i dati del livello successivo per ***levelData***;
3. ottenere i dati dei livelli saltati per ***levelData***;
4. ingrandire l'immagine fino al limite inferiore di zoom del nuovo livello per poter ottenere i dati della nuova porzione da visualizzare e mandarli al codice Python;
5. ingrandire l'immagine con il livello di zoom relativo inserito per anticipare all'utente la nuova porzione in arrivo;
6. ottenere, impostare e ingrandire la nuova immagine con il nuovo zoom relativo, calcolato rispetto allo zoom base del nuovo livello.

Algorithm 3 Passaggio a livello inferiore

```
Require: absoluteZoom: livello di zoom rispetto a immagine iniziale; newLevel: nuovo livello
         da visualizzare
 1: slider disabilitato
    ▷ blocco 1: ottenere livelli di zoom
 2: absolute ← levelZoomLimits[newLevel].absolute
 3: firstRelative ← levelZoomLimits[curLevel − 1].relative
 4: allRelatives ← 1
 5: for i in [newLevel, curLevel − 1] do
 6:     allRelatives ← allRelatives ∗ levelZoomLimits[i].relative
 7: end for
 8: newZoom ← absoluteZoom/absolute
    ▷ blocco 2: ottenere dati di livello successivo per levelData
 9: zoomMoveImage(firstRelative)
10: hidden ← coordinate di punto di immagine situato nel vertice in alto a sinistra del box
11: coord ← hidden/(currentZoom ∗ dimFactor)
12: ratio ← coord/ogDim
13: size ← ⌈box/(currentZoom ∗ dimFactor)⌉
14: levelData[curLevel].ogDim ← ogDim
15: levelData[curLevel].ratio ← ratio
16: levelData[curLevel − 1].size ← size
    ▷ blocco 3: ottenere dati di livelli saltati per levelData
17: tmpDimFactor ← dimFactor
18: tmpSize ← size
19: for l in [curLevel − 1, newLevel + 1] do
20:     dsRatioAbove ← downsamples[l + 1]/downsamples[l]
21:     tmpOgDim ← ⌈tmpSize ∗ dsRatioAbove⌉
22:     if tmpOgDim.width ≥ tmpOgDim.left then
23:         tmpDimFactor ← box.height/tmpOgDim.height
24:     else
25:         tmpDimFactor ← box.width/tmpOgDim.width
26:     end if
27:     r ← levelZoomLimits[l].relative
28:     tmpSize ← ⌈box/(r ∗ tmpDimFactor)⌉
29:     levelData[l].ratio ← ((r − 1)/(2 ∗ r), (r − 1)/(2 ∗ r))
```

30: $\boldsymbol{levelData}[l].ogDim \leftarrow tmpOgDim$
31: $\boldsymbol{levelData}[l-1].size \leftarrow tmpSize$
32: $\boldsymbol{imageData}[l-1] \leftarrow$ stringa vuota
33: **end for**
▷ blocco 4: ottenere dati su nuova porzione da visualizzare
34: $zoomMoveImage(allRelatives)$
35: $baseImageHidden \leftarrow$ posizione dell'immagine rispetto al box
36: $baseImageCoord \leftarrow baseImageHidden/(\boldsymbol{currentZoom} * \boldsymbol{dimFactor})$
37: $baseImageRatio \leftarrow baseImageCoord/\boldsymbol{ogDim}$
38: $baseImageSize \leftarrow \lceil box/(\boldsymbol{currentZoom} * \boldsymbol{dimFactor}) \rceil$
39: $baseImageCoord \leftarrow getCoords(\boldsymbol{curLevel}, baseImageRatio)$
▷ blocco 5: mostrare a utente zona che apparirà
40: $zoomMoveImage(\boldsymbol{relativeZoom})$
▷ blocco 6: ottenere e impostare nuova immagine
41: $level \leftarrow$ coppia $(\boldsymbol{curLevel}, newLevel)$
42: $jsonData \leftarrow$ JSON contenente $baseImageCoord$, $level$, $baseImageSize$
43: $newImage \leftarrow$ risposta di richiesta GET al codice PHP del viewer, funzione richiesta get_image, dati forniti $jsonData$
44: $newImageData \leftarrow newImage$ convertito da stringa a oggetto JSON
45: $setSlide(newImageData)$
46: **if** $newLevel > 0$ **then**
47: $\boldsymbol{imageData}[newLevel-1] \leftarrow newImageData$
48: **end if**
49: $zoomMoveImage(newZoom)$
50: slider abilitato

Il blocco 1 (righe 2-8) calcola:

- *firstRelative*: valore necessario al blocco 2, rappresenta il livello di zoom relativo rispetto al livello attuale oltre il quale si passa a quello successivo;
- *allRelatives*: valore necessario al blocco 4, rappresenta il livello di zoom relativo rispetto al livello attuale oltre il quale si passa al livello nuovo, ricavato dal prodotto di tutti gli zoom relativi soglia compresi tra il livello nuovo e quello successivo all'attuale;
- *newZoom*: valore necessario al blocco 5, rappresenta il livello di zoom relativo rispetto al nuovo livello con il quale bisognerà ingrandire la nuova porzione, ricavato dal rapporto tra lo zoom relativo inserito e quello base del nuovo livello.

I blocchi 2 (righe 9-16) e 4 (righe 34-39) seguono procedimenti analoghi. Il primo calcola coordinate e dimensioni della (ipotetica se vengono saltati dei livelli) nuova porzione per il livello successivo a quello attuale, mentre il secondo determina gli stessi dati per la nuova porzione da visualizzare dopo il passaggio di livello. Nonostante queste differenze i due blocchi seguono gli stessi passsaggi. Vengono calcolate le coordinate del punto dell'immagine che si trova nel vertice in alto a sinistra del box (*hidden*/*baseImageHidden*). Queste coordinate vengono scalate di un fattore ***currentZoom****∗***dimFactor***, per fare in modo che siano relative non a ***stdDim*** ma a ***ogDim***, ovvero le dimensioni originali della porzione attualmente visualizzata. Si ricavano poi il rapporto tra le coordinate e ***ogDim*** (*ratio*/*baseImageRatio*) e la dimensione della porzione (*size*/*baseImageSize*), ricavata dividendo le dimensioni del box per ***currentZoom****∗***dimFactor*** per adattarle al sistema di riferimento di ***ogDim***, analogamente alle coordinate. Nel caso del blocco 2 questi dati vengono inseriti in ***levelData***, mentre il blocco 4 li manda al codice Python.

In virtù di questo, questo blocco effettua l'operazione aggiuntiva di adattare *baseImageRatio* al livello massimo usando `getCoords`.

Il blocco 3 (righe 17-32) si basa su un'approssimazione dei dati da inserire in corrispondenza dei livelli saltati. Le misure precise si possono ricavare solamente calcolandole in modo diretto sulla pagina, ma possiamo ottenere stime di *size*, *ogDim* e *ratio* molto vicine ai valori reali. Per *ratio* supponiamo che, per le porzioni corrispondenti ai livelli salvati, le aree che mostrano si trovino al centro dell'immagine del livello precedente. In questa situazione possiamo calcolare il valore di *ratio* con il metodo illustrato nella Figura 7.

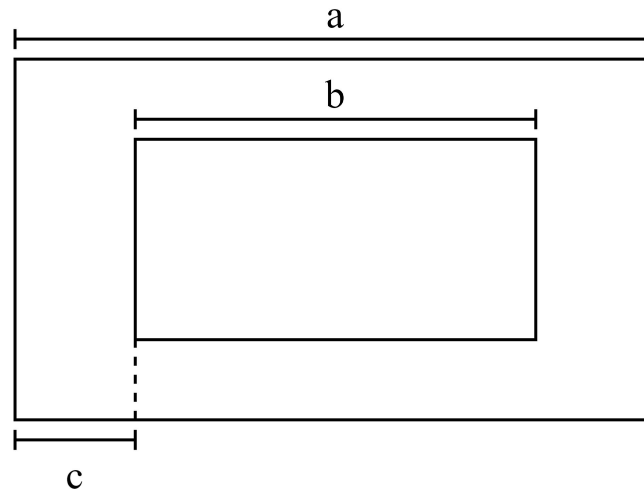

Figure 7: Schema che rappresenta il calcolo del valore di *ratio* da inserire in ***levelData*** per un certo livello x saltato durante un passaggio a un livello inferiore. a: larghezza dell'immagine visualizzata per x; b: larghezza della porzione da visualizzare per *x-1* rispetto alla porzione visualizzata per x; c: coordinata orizzontale del vertice in alto a sinistra della porzione da visualizzare per *x-1* rispetto alla porzione visualizzata per x.

$$b = \frac{a}{x} \tag{3}$$

$$c = \frac{a-b}{2} = \frac{1}{2}(a - \frac{a}{x}) = \frac{a}{2}(1 - \frac{1}{x}) \tag{4}$$

$$ratio = \frac{c}{a} = \frac{1}{2}(1 - \frac{1}{x}) = \frac{x-1}{2x} \tag{5}$$

Il procedimento è analogo per le altezze. La formula ricavata nell'Equazione 5 vale per tutti i livelli ed è esattamente quella usata nell'Algoritmo 3, alla riga 29.

Per *size* e *ogDim* si segue un procedimento diverso. Qui è fondamentale capire in che modo, per ogni livello, la prima variabile viene trasformata nella seconda dal codice Python. Consideriamo un passaggio di livello da *x+1* a x e chiamiamo d_{x+1} e d_x i loro fattori di sottocampionamento:

$$size \xrightarrow{\texttt{get_image}} dim = size \cdot \frac{d_{x+1}}{d_x} \xrightarrow{\text{ritorno al codice JavaScript}} ogDim = dim$$

Pertanto, data la dimensione della nuova porzione per un certo livello x, moltiplicando tale dato per il rapporto tra il suo fattore di sottocampionamento e quello del livello precedente si ottiene una buona approssimazione del valore che avrà ***ogDim*** dopo l'impostazione dell'immagine. Con questo risultato è possibile ricavare tutti gli altri dati necessari. Si ricava il valore che ***dimFactor*** avrebbe per x, si prende da ***levelZoomLimits*** il limite superiore

di zoom per x e lo si usa per calcolare la dimensione che la nuova porzione di livello $x-1$ avrebbe rispetto all'immagine di livello x. Si inseriscono i valori ipotetici appena calcolati di *size*, *ogDim* e *ratio* in ***levelData*** e si cancellano eventuali immagini salvate per x. Questo procedimento viene ripetuto per tutti i livelli saltati e garantisce che per ogni passaggio di livello ***levelData*** sia riempito con i valori corretti oppure altri molto vicini a questi ultimi. Funziona anche per salti singoli; il ciclo non parte, in quanto non vengono saltati livelli.

Il blocco 5 (riga 40) è costituito da una sola riga di codice che ingrandisce e riposiziona l'immagine in base al valore di ***relativeZoom*** per offrire all'utente un'anteprima della porzione che il codice Python sta generando.

Infine, il blocco 6 (righe 41-49) ottiene la nuova porzione da visualizzare. Vengono preparati i dati richiesti da `get_image` e racchiusi in un JSON. Dopodiché il codice Python viene chiamato con una richiesta GET. Il JSON restituito viene passato come parametro a `setSlide`, per impostare l'immagine, e conservato in ***imageData*** se il livello attuale non è 0. Infine, si ingrandisce la porzione appena impostata con il nuovo zoom, *newZoom*.

Passaggio a livello piramidale superiore

Anche nel caso dei passaggi a livelli superiori, vengono gestiti salti sia singoli che multipli con un unico algoritmo. Tuttavia, questa distinzione non è il problema principale in questo caso; la problematica implicata da questa classificazione, riguardante l'aggiornamento di ***levelData*** per i livelli saltati, viene già gestita durante i passaggi a livelli inferiori. La sfida principale diventa quindi la possibilità di non avere a disposizione un'immagine salvata per il nuovo livello. Questa evenienza viene gestita generando la porzione mancante al momento.

Algorithm 4 Passaggio a livello superiore

Require: *absoluteZoom*: livello di zoom rispetto a immagine iniziale; *newLevel*: nuovo livello da visualizzare; *hidden*: coordinate di punto di immagine situato nel vertice in alto a sinistra del box

1: $tmpCenter \leftarrow (hidden + box/2)/(\boldsymbol{currentZoom} * \boldsymbol{dimFactor})$
2: $centerRatio \leftarrow tmpCenter/\boldsymbol{ogDim}$
3: $center \leftarrow getCoords(\boldsymbol{curLevel}, centerRatio, newLevel)$
4: $previousImage \leftarrow \boldsymbol{imageData}[\boldsymbol{newLevel} - 1]$
5: **if** $previousImage =$stringa vuota **then**
6:     slider disabilitato
7:     $newRatio \leftarrow (0,0)$
8:     $newCoord \leftarrow getCoords(newLevel, newRatio)$
9:     $level \leftarrow (newLevel + 1, newLevel)$
10:     $newSize \leftarrow \boldsymbol{levelData}[newLevel].size$
11:     $jsonData \leftarrow$ JSON contenente *newCoord*, *level*, *newSize*
12:     $newImage \leftarrow$ risposta di richiesta GET al codice PHP del viewer, funzione richiesta *get_image*, dati forniti *jsonData*
13:     $newImageData \leftarrow newImage$ convertito da stringa a oggetto JSON
14:     **if** $newLevel > 0$ **then**
15:         $\boldsymbol{imageData}[newLevel - 1] \leftarrow newImageData$
16:     **end if**
17:     $previousImage \leftarrow newImageData$
18:     slider abilitato
19: **end if**
20: $setSlide(previousImage)$
21: $newZoom \leftarrow absoluteZoom/\boldsymbol{baseZoom}$
22: $\boldsymbol{currentCenter} \leftarrow center * \boldsymbol{dimFactor}$
23: $\boldsymbol{centerToChange} \leftarrow true$
24: $zoomMoveImage(newZoom)$

All'inizio dell'Algoritmo 4 (righe 1-3) viene ricalcolato il punto dell'immagine da mantenere al centro del box e scalato per adattarlo al sistema di riferimento relativo alle dimensioni di ***ogDim***. Dopodiché, vengono adattate al livello di destinazione dando il rapporto tra coordinate e ***ogDim*** come parametro a `getCoords`. È da notare come in questo caso il parametro *cap* venga impostato manualmente, dato che il nuovo livello può essere diverso da quello iniziale.

Successivamente, si estrae il contenuto di ***imageData*** per il livello attuale (riga 4). Se è presente un JSON con un'immagine lo si usa, altrimenti viene generata (righe 5-19). Per questa operazione possiamo usare i dati contenuti in ***levelData***, dato che l'algoritmo che gestisce i passaggi a livelli inferiori garantisce la loro correttezza. Per ottenere la porzione giusta per il nuovo livello, si considera il suo vertice in alto a sinistra e si adattano le sue coordinate, ovvero $(0,0)$, al livello più alto. Come dato sui livelli vengono passati il nuovo livello e quello immediatamente precedente. Infine, come dimensione della porzione viene usata quella salvata in ***levelData***. Il passaggio dei dati tramite richiesta GET, l'elaborazione della sua risposta e il salvataggio della nuova immagine avvengono nello stesso modo illustrato dalle righe 42-44, 45-48 dell'Algoritmo 3.

A questo punto, abbiamo a disposizione la nuova immagine, indipendentemente dal fatto che questa fosse già presente in ***levelData***; questa viene inserita nel box (righe 20-24). Viene calcolato il nuovo livello di zoom tramite il rapporto di quello indicato dallo slider e il limite inferiore. Successivamente, si modifica ***currentCenter*** moltiplicando il valore calcolato per ***dimFactor***, in modo da rendere il suo valore relativo alle dimensioni di ***stdDim***, e si imposta ***centerToChange*** a `true` per avere coordinate precise del punto da mantenere al centro del box al prossimo input tramite slider. Infine, si ingrandisce la nuova immagine.

Conclusioni

Questa tesi ha presentato la progettazione e sviluppo di un tool di annotazione collaborativo con interfaccia web per immagini istologiche digitali, con il quale medici e specialisti possono salvare le immagini in un database, visualizzarle e annotarle per segnare su di esse patologie e altre particolare condizioni. Questi dati possono poi essere utilizzati nell'apprendimento di algoritmi di Machine Learning per la rilevazione automatica delle stesse caratteristiche. Data la loro struttura particolare, con vari livelli piramidali corrispondenti a diverse risoluzioni della stessa immagine, è stata realizzata usando l'API Python di una libreria, OpenSlide, che fornisce un'interfaccia uniforme e intuitiva per aprire e interagire con tali immagini.

L'applicazione sviluppata si basa sull'interazione tra una vasta gamma di linguaggi e strumenti. È stato progettato un database per conservare le informazioni sulle immagini caricate, sugli utenti che usano l'applicazione e sulle annotazioni create. Questo database è stato poi implementato tramite il DBMS PostgreSQL. Le pagine web sviluppate includono una home page, da cui scegliere quale immagine visualizzare e caricare nuovi file nel database, e un viewer nel quale l'immagine scelta viene visualizzata e dove l'utente può ingrandirla, rimpicciolirla e spostarla mentre la pagina gestisce automaticamente i passaggi tra i vari livelli piramidali dell'immagine, per assicurare che la risoluzione vista dall'utente sia sempre soddisfacente. Queste pagine sono state sviluppate usando una vasta gamma di linguaggi, mentre le interazioni con OpenSlide sono state implementate tramite codice Python. Per la comunicazione tra le pagine web e il codice Python è stato creato un codice PHP che riceve le richieste HTTP in arrivo dal codice JavaScript e avvia il codice Python fornendo come dati di input i parametri della richiesta, per poi ricevere l'output dello script Python e inoltrarlo al codice JavaScript.

La versione descritta in questa tesi si può prestare facilmente a futuri sviluppi e aggiunte di nuove funzionalità. Le aggiunte principali riguardano l'implementazione del sistema di annotazioni, tile e label, nella pagina del viewer, e del sistema di utenti normali e gestori, accessibile dalla home page. Altri miglioramenti possibili riguardano vari aspetti dell'esperienza utente del viewer, come animazioni per mostrare in modo più chiaro i momenti di attesa dovuti alla creazione delle nuove porzioni oppure riempire l'intera pagina viewer con il box dell'immagine, ponendo al suo interno lo slider e le opzioni per aggiungere annotazioni, tile e label.

Ringraziamenti

Al termine di questa tesi, desidero fortemente ringraziare tutte le persone che hanno reso questi tre anni un'esperienza da ricordare con affetto e gioia.

In primo luogo, vorrei ringraziare il mio relatore Stefano Ghidoni e il mio correlatore Leonardo Barcellona. La loro disponibilità e i loro preziosi consigli sono stati fondamentali nella stesura dell'elaborato e ne farò tesoro nei miei progetti futuri.

Ringrazio la mia famiglia, Evelina, Paolo e Marco: una certezza nella vita e in questi tre anni, sempre disponibili per aiutarmi e supportarmi. Grazie infinite!

Infine, un grazie di cuore a tutti gli amici che hanno reso questi tre anni unici: Francesco, Ippolito, Kabir, Marius, Samuele, Riccardo, Francesco, Matteo, Alessandro, Elisa, Giulia, Fabio, Rachele, Giulia, Giulio, Tommaso, Damiano, Alessandro, Riccardo, per citarne alcuni. Sono stati il vero tesoro di questo percorso universitario: tanti bei momenti insieme, tanti ricordi, tante occasioni di crescita e confronto, tanti aiuti ricevuti negli studi. Vi voglio un mondo di bene, che i nostri rapporti possano crescere nel tempo!

Bibliografia

[1] Farzad Ghaznavi et al. "Digital Imaging in Pathology: Whole-Slide Imaging and Beyond". In: *Annual Review of Pathology: Mechanisms of Disease* 8.1 (2013). PMID: 23157334, pp. 331–359. DOI: 10.1146/annurev-pathol-011811-120902. eprint: https://doi.org/10.1146/annurev-pathol-011811-120902. URL: https://doi.org/10.1146/annurev-pathol-011811-120902.

[2] Matthew G. Hanna, Anil Parwani, and Sahussapont Joseph Sirintrapun. "Whole Slide Imaging: Technology and Applications". In: *Advances In Anatomic Pathology* 27.4 (July 12, 2020), pp. 251–259. ISSN: 1072-4109. DOI: doi:10.1097/PAP.0000000000000273. URL: https://www.ingentaconnect.com/content/wk/adapa/2020/00000027/00000004/art00005.

[3] Adam Goode et al. "OpenSlide: A vendor-neutral software foundation for digital pathology". In: *Journal of Pathology Informatics* 4.1 (2013), p. 27. ISSN: 2153-3539. DOI: https://doi.org/10.4103/2153-3539.119005. URL: https://www.sciencedirect.com/science/article/pii/S2153353922006484.

[4] James H. Fetzer. "What is Artificial Intelligence?" In: *Artificial Intelligence: Its Scope and Limits*. Dordrecht: Springer Netherlands, 1990, pp. 3–27. ISBN: 978-94-009-1900-6. DOI: 10.1007/978-94-009-1900-6_1. URL: https://doi.org/10.1007/978-94-009-1900-6_1.

[5] Pavel Hamet and Johanne Tremblay. "Artificial intelligence in medicine". In: *Metabolism* 69 (2017). Insights Into the Future of Medicine: Technologies, Concepts, and Integration, S36–S40. ISSN: 0026-0495. DOI: https://doi.org/10.1016/j.metabol.2017.01.011. URL: https://www.sciencedirect.com/science/article/pii/S002604951730015X.

[6] Pramila P. Shinde and Seema Shah. "A Review of Machine Learning and Deep Learning Applications". In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018, pp. 1–6. DOI: 10.1109/ICCUBEA.2018.8697857.

[7] Yalin Baştanlar and Mustafa Özuysal. "Introduction to Machine Learning". In: *miRNomics: MicroRNA Biology and Computational Analysis*. Ed. by Malik Yousef and Jens Allmer. Totowa, NJ: Humana Press, 2014, pp. 105–128. ISBN: 978-1-62703-748-8. DOI: 10.1007/978-1-62703-748-8_7. URL: https://doi.org/10.1007/978-1-62703-748-8_7.

[8] Jiahui Li et al. "Hybrid Supervision Learning for Pathology Whole Slide Image Classification". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*. Ed. by Marleen de Bruijne et al. Cham: Springer International Publishing, 2021, pp. 309–318. ISBN: 978-3-030-87237-3.

[9] Ziwang Huang et al. "Integration of Patch Features Through Self-supervised Learning and Transformer for Survival Analysis on Whole Slide Images". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*. Ed. by Marleen de Bruijne et al. Cham: Springer International Publishing, 2021, pp. 561–570. ISBN: 978-3-030-87237-3.

[10] Marc Aubreville et al. "SlideRunner". In: *Bildverarbeitung für die Medizin 2018*. Ed. by Andreas Maier et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 309–314. ISBN: 978-3-662-56537-7.

[11] Runtian Miao et al. *Quick Annotator: an open-source digital pathology based rapid image annotation tool*. 2021. arXiv: 2101.02183 [eess.IV].

[12] *jQuery*. URL: https://jquery.com/.

[13] MDN contributors. *Fetch API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

[14] *ECMA 404 The JSON Data Interchange Syntax*. 2017.

[15] Larry M Masinter. *The "data" URL scheme*. RFC 2397. Aug. 1998. DOI: 10.17487/RFC2397. URL: https://www.rfc-editor.org/info/rfc2397.

[16] Alexey Melnikov, Darrel Miller, and Murray Kucherawy. *Media types*. URL: https://www.iana.org/assignments/media-types/media-types.xhtml.