REPORT DI FINE PROGETTO

Corso di Comunicazioni Multimediali

A.A. 2020/2021

“The Finder”

Alberto Casagrande Alessio Belli Chiara Calore

Davide Zordan Mattia Perin

TV team

Sommario

1 Introduzione

La fotografia è una tecnologia che permette di registrare eventi appartenenti al mondo che ci circonda e di poter catturare un istante preciso.
Con il passare degli anni, grazie anche al progresso tecnologico, la fotografia ha cambiato forma, usi e fruibilità, rendendo possibile immortalare aspetti del quotidiano, ovunque ci si trovi e con un vasto numero di strumenti da poter utilizzare.
Con l'avvento della tecnologia digitale, si è sempre più persa l'abitudine di stampare le proprie fotografie, ma non quella di conservarle in raccolte. La maggior parte delle volte, però, queste vengono aggiornate nel tempo, rendendo difficile estrapolare le immagini ricercate.
Da questa esigenza, nasce la necessità di sviluppare dei sistemi per la ricerca delle foto, agevolando così il processo di estrapolazione delle immagini desiderate.

1.1 L'idea

Dalle necessità precedentemente riportate, nasce l'idea di implementare una libreria di immagini e di sviluppare un algoritmo di ricerca e matching, capace di restituire le foto più similari a quella di un'immagine campione, scelta dall'utente.
Questa tecnica di confronto è stata scelta poiché, seppur i metodi di ricerca basati sui metadati interni alle fotografie siano efficaci in alcune situazioni, in altre risultano insufficienti ai fini della ricerca, non permettendo di avere, alla fine del processo, dei risultati pienamente soddisfacenti.

1.2 Le fasi

Il progetto si struttura principalmente in 10 fasi:

- definizione del linguaggio di programmazione e delle relative librerie;
- creazione del dataset di immagini campione;
- ricerca e definizione dei criteri di matching per le immagini;
- specifica delle caratteristiche dell'interfaccia grafica;
- ricerca e prima stesura del codice degli algoritmi di matching;
- sviluppo software della GUI;
- sviluppo software degli algoritmi;
- integrazione e ottimizzazione del codice;
- testing del progetto realizzato;
- stesura del report.

Per il raggiungimento degli obiettivi preposti, si è partiti dall'iniziale ricerca del linguaggio di programmazione e delle librerie necessarie per lo sviluppo degli algoritmi da utilizzare.
Una volta definiti questi criteri, il focus si è spostato sulla ricerca e definizione degli algoritmi di matching per le immagini, che andranno ad essere il fulcro del progetto.
In parallelo, è stata portata avanti la prima definizione dell'interfaccia grafica e delle sue caratteristiche, oltre alla creazione del dataset di immagini.
Ciò è stato possibile grazie ad una precisa suddivisione dei ruoli e alla grande collaborazione avuta nelle varie fasi operative del progetto.
Successivamente, si è passati ad una prima sommaria stesura degli algoritmi e allo sviluppo software della GUI.
Gli algoritmi precedentemente trovati sono stati poi ottimizzati e resi compatibili per l'architettura scelta, operazione ripetuta, in fase finale, anche per il codice dell'interfaccia grafica.
Per concludere, sono stati effettuati i controlli tramite le fasi di testing, per verificare il corretto funzionamento dell'architettura e la stesura finale del suddetto report.

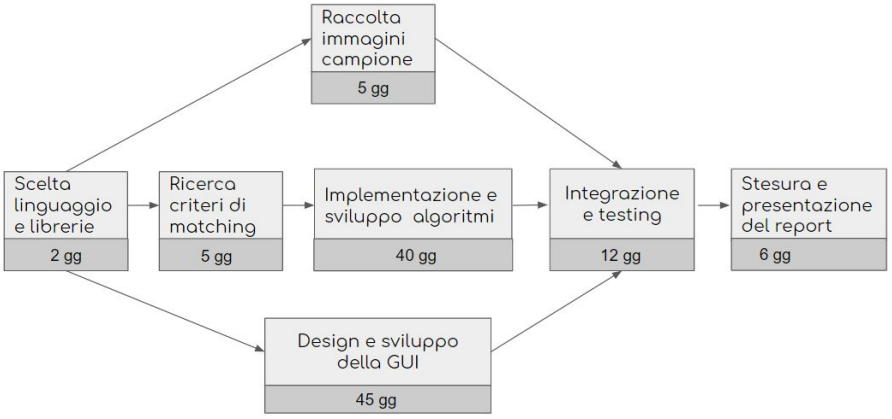

Figura 1: diagramma PERT

1.3 Le fonti

Tutto il materiale reperito per la realizzazione del progetto sarà citato all'interno del report, tramite delle note a piè di pagina, così da garantire una correlazione immediata tra le fonti e la loro attuazione pratica. Le librerie utilizzate sono licenziate come software libero:

OpenCV - BSD license
NumPy - BSD license
Flask - BSD license
TensorFlow - Apache 2.0
Keras - MIT license
ResNet50 [1] - MIT license

[1] K. He et al. (2015). "Deep Residual Learning for Image Recognition" http://arxiv.org/abs/1512.03385

2 Linguaggio e Librerie

Inizialmente, prima di procedere con le fasi "operative", si è pensato di risolvere le questioni pratiche riguardanti la scelta del linguaggio di programmazione e delle librerie da utilizzare per l'implementazione degli algoritmi. Senza questo passaggio fondamentale, sarebbero sicuramente sorte in fase di realizzazione delle problematiche legate alla poca chiarezza delle direttive: ciò ci ha permesso di avere sempre un quadro chiaro della situazione, evitando di sprecare tempo per la ridefinizione degli obiettivi primari del progetto.

2.1 Python

Dapprima si è discusso su quale linguaggio di programmazione utilizzare per l'implementazione del progetto. La scelta finale è ricaduta su Python per la quantità di librerie supportate, per il suo essere multi-paradigma, per la sua portabilità e le sue prestazioni elevate[2]. La versione utilizzata per il progetto è Python 3.8

2.2 Numpy

Numpy è una libreria open source di Python che presenta funzioni matematiche per la manipolazione di grandi matrici e array multidimensionali[3]. Nel progetto ci si è trovati più volte a utilizzare funzioni, facendo riferimento a questa libreria, per la gestione e il salvataggio su file di array multidimensionali, soprattutto nella fase di estrazione delle features con Deep Learning.

2.3 OpenCV

Ulteriore libreria utilizzata è stata OpenCV, una libreria software multipiattaforma nell'ambito della visione artificiale in tempo reale. Se integrata con altre librerie, come Numpy, rende possibile l'elaborazione della struttura degli array per l'analisi, identificando il pattern dell'immagine e le sue varie caratteristiche tramite lo spazio vettoriale, rendendo perciò possibile anche operare matematicamente su queste caratteristiche[4]. In particolare, sono state usate principalmente le funzioni per il calcolo e il confronto di istogrammi e la classe FileStorage per il loro salvataggio in formato YAML.

[2] Advantages of Using Python for Computer Vision https://fullscale.io/blog/advantages-using-python-computer-vision/
[3] Numpy: the fundamental package for scientific computing with Python https://numpy.org/
[4] OpenCV: Open Source Computer Vision https://docs.opencv.org/3.4/index.html

2.4 Flask

Flask è una struttura Web scritta in Python, basata sui tool-kit WSGI Werkzeug, che permette di implementare richieste, oggetti di risposta e funzioni di utilità per la costruzione di un framework web. Inoltre utilizza Jinja2 per combinare i template con le fonti di dati per rendere le pagine web dinamiche[5]. Abbiamo deciso di utilizzare Flask in quanto è un framework semplice da usare, soprattutto per iniziare a prendere dimestichezza con Python. Inoltre, è adatto per creare web app semplici che non richiedono infrastrutture complesse per funzionare.
La struttura tipica di un progetto in flask è così formata:

```
app.py
      /templates
            index.html
            …
      /static
            logo.png
            myStyle.css
            …
```

Nella cartella templates vengono inseriti tutti i file `*.HTML` che verranno inviati dal Server verso il Client (visualizzati quindi nel browser).
Nella directory static, invece, vengono inseriti tutti i file statici come: file CSS, JS e anche immagini che si vogliono visualizzare all'interno dell'architettura.

Vi è una funzione principale utilizzata da Flask per inizializzare il Server su una determinata porta: `app.run(port=4555)`

All'interno del file Python che gestisce la nostra web app, si utilizzano dei decoratori (es. `@app.route("/")`) che Flask fornisce al fine di assegnare facilmente gli URL nella nostra app alle funzioni[6]. Tramite questi oggetti, è possibile anche indicare alcuni parametri aggiuntivi (come per esempio le tipologie di richieste HTTP come POST e GET).

Al fine di visualizzare una determinata pagina HTML, è possibile utilizzare la funzione `render_templete("index.html")`, alla quale è possibile passare dei parametri che saranno poi gestiti lato client tramite Jinja2.

Siccome nel nostro progetto è richiesto l'invio di immagini dal Server al Client in maniera dinamica, abbiamo deciso di utilizzare un'altra tipologia di funzioni che poteva esserci utile. Tale metodo è: `send_from_directory(directory,filename)`

[5] Flask: web development, one drop at a time https://flask.palletsprojects.com/en/1.1.x/
[6] The Art of Routing in Flask: https://hackersandslackers.com/flask-routes/

2.5 TensorFlow & Keras

TensorFlow è una piattaforma open source end-to-end per il machine learning. È stata creata inizialmente dal gruppo di sviluppatori di Google Brain, rilasciata poi con licenza Apache 2.0 nel 2015.

TensorFlow utilizza dataflow graphs, ossia strutture che descrivono come i dati si muovono lungo un grafo, che è composto da una serie di processing nodes. Ciascun nodo nel grafo rappresenta un'operazione matematica, e ciascuna relazione o connessione tra nodi è un array multidimensionale, anche detto tensore[7].

Nel nostro progetto tale piattaforma è utilizzata come framework di basso livello per l'interfaccia ad alto livello di Keras.

Keras è una libreria scritta in Python (rilasciata sotto licenza MIT), per l'apprendimento automatico e per le reti neurali e supporta come back-end TensorFlow (dal 2017). Keras offre moduli utili per organizzare differenti livelli. Il tipo principale di modello è quello sequenziale.

Il deep learning è un sottocampo specifico dell'apprendimento automatico: si basa in particolare sull'apprendimento di livelli successivi di rappresentazioni significative. Il termine "deep" rappresenta l'idea di strati successivi di rappresentazioni. La quantità di livelli che contribuisce a un modello di dati è chiamata profondità del modello[8].

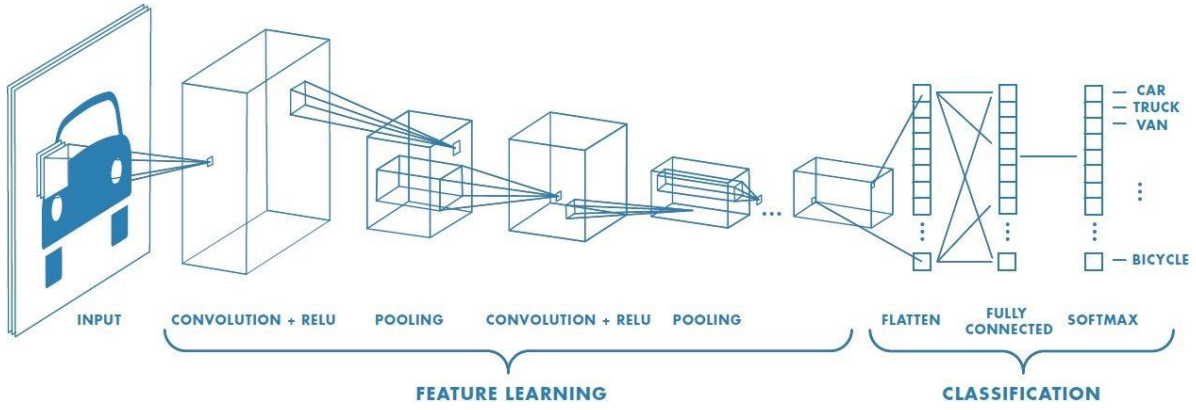

Figura 2: esempio di rete neurale convoluzionale [9]

[7] What is TensorFlow? The machine learning library explained
https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[8] Keras, cos'è, a cosa serve, come si usa nel deep learning
https://www.ai4business.it/intelligenza-artificiale/keras-cose-a-cosa-serve-come-si-usa-nel-deep-learning/

[9] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Nel nostro caso, sono stati utilizzati modelli resi disponibili già "addestrati", a costo di una dimensione di download iniziale sicuramente considerevole ma riducendo notevolmente la complessità computazionale del modello in quanto non è necessaria la fase di training. Alla prima istanziazione di un modello, i pesi sono automaticamente scaricati dagli archivi online e salvati in locale.
Sono disponibili decine di modelli, caratterizzati da profondità, numero di parametri e dimensione diversi (vedi fig. 3).

Come compromesso fra tutte queste caratteristiche è stato scelto ResNet50, anche se inizialmente era stato preso in considerazione VGG16. Quest'ultimo ha ottime prestazioni ma necessita al primo avvio dell'applicazione di un download di ben 500 MB di weights pre-calcolati, mentre ResNet si ferma a 90 MB.

Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

Figura 3: modelli disponibili su Keras [10]

[10] Keras Applications https://keras.io/api/applications/

2.6 ResNet50

ResNet è un modello di rete neurale convoluzionale sviluppato nel 2015 da ricercatori di Microsoft e vincitore del ILSVRC di quell'anno (competizione nel campo di riconoscimento di oggetti e scene nelle immagini)[11].
E' detto convoluzionale poiché utilizza dei filtri di convoluzione e di pooling. La dimensione e i valori dei filtri vengono appresi in maniera automatica nella fase di training, tramite l'utilizzo di gradienti. Il processo si ripete finché la rete non individua quei valori dei filtri tali per cui il matching tra l'immagine in input e il valore predetto produca meno errori possibile: il processo è noto come backpropagation[12].

Normalmente, in una rete convoluzionale si presenta il problema detto "Vanishing Gradient". Ciò limita il numero massimo di livelli di profondità poiché, durante la fase training, scendendo lungo la struttura della rete, il gradiente tende a 0 (dato che si tratta di una catena di prodotti fra gradienti che hanno valore compreso tra 0 e 1)[13]. Si arriva perciò a un punto per cui i livelli non contribuiscono più all'apprendimento.

La principale novità apportata dal ResNet è la "skip connection". Una rete residuale, introducendo degli shortcuts (vedi figure sotto), può saltare blocchi di livelli (e non considerarli nel calcolo del gradiente). Tale processo, oltre a fornire al livello successivo l'output convoluto F(x), passa anche la funzione identità (quindi x stesso) e la somma a F(x) per ottenere infine il residuale y = f(x) + x.

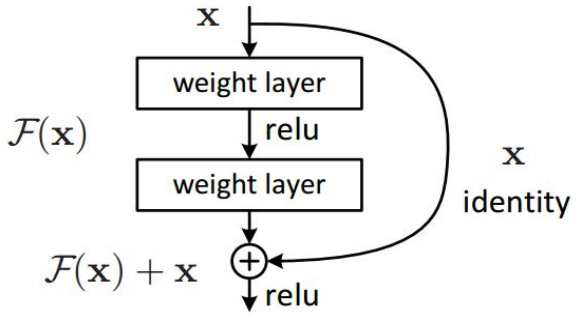

Figura 4: Residual block [14]

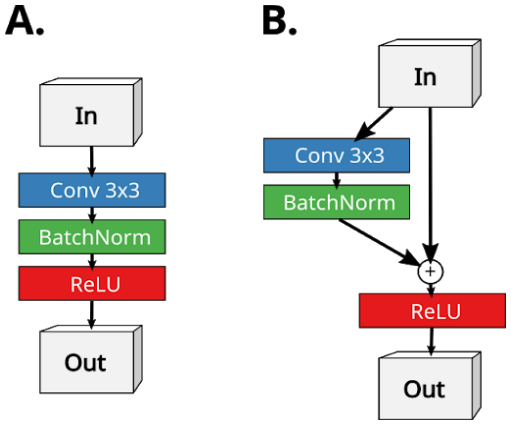

Figura 5: CNN (A) without and (B) with shortcut connections [15]

[11] Deep Residual Learning for Image Recognition(Kaiming He, et al.) https://arxiv.org/abs/1512.03385
[12] Convolutional Neural Network https://andreaprovino.it/convolutional-neuralnetwork/
[13] A Brief and Simple Introduction to ResNet
https://medium.com/@shdangwal/a-brief-and-simple-introduction-to-resnet-47432eff95b8
[14] Undestand and implement ResNets https://cv-tricks.com/keras/understand-implement-resnets/
[15] Review of He, Zhang, Ren, Sun (2015) Deep Residual Learning for Image Recognition
http://neural.vision/blog/article-reviews/deep-learning/he-resnet-2015/

Durante la fase di training, il modello imparerà quali livelli contribuiscono efficacemente alla backpropagation e quali è conveniente saltare per evitare il Vanishing Gradient.

Infine, il blocco ReLU è la funzione di attivazione della rete Neurale, che innesca i nodi solo se in presenza di input positivo. Al contrario, l'output sarà 0. Corrisponde perciò a una funzione rampa (vedi fig. 6).

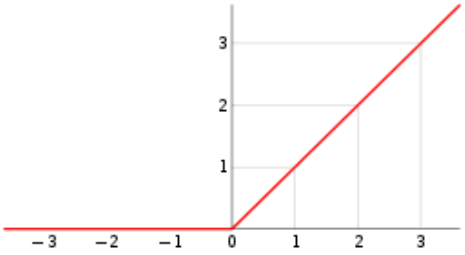

Figura 6: Funzione ReLU

In conclusione, sarà quindi possibile raggiungere un numero di livelli molto elevato (anche più di 100), ponendo una nuova base per i modelli di deep learning, considerato che il modello più profondo creato fino ad allora era di 22 strati (VGG-19, vincitore dell' ILSVRC del 2014).

Nel progetto viene utilizzata un'architettura a 50 livelli, basata su dei weights precalcolati sui dataset ImageNet. Viene data in input un'immagine RGB 224x224, viene effettuato un primo livello di convoluzione e successivamente ci sono 4 blocchi principali composti da 3 livelli di convoluzione ciascuno (vedi fig. 7) [16].

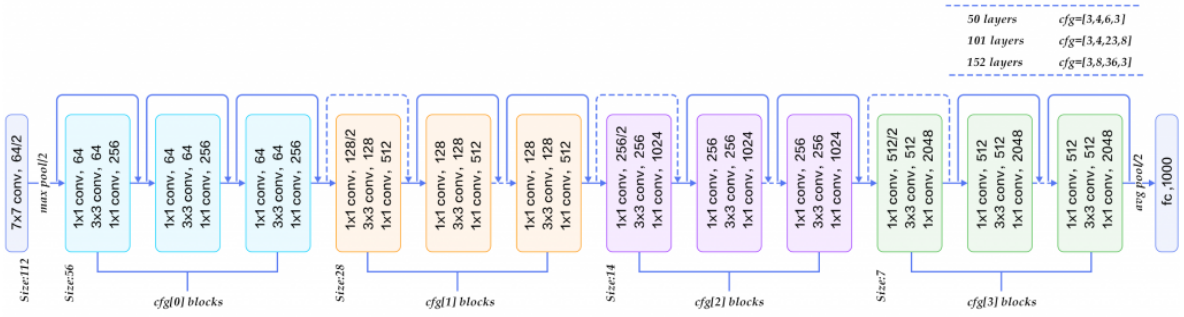

Figura 7: architettura ResNet

[16] Understanding and Coding a ResNet in Keras
https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

3 Il progetto

"The Finder" nasce con l'idea di creare un'interfaccia grafica client-server accattivante e di facile utilizzo, tramite cui inserire un'immagine presente sul proprio dispositivo e ricercare, all'interno di un dataset di immagini presenti sulla piattaforma, le migliori corrispondenze. Come risultato, verranno restituite le immagini più somiglianti a quella inserita, con le relative percentuali di similarità associate. L'utente, inoltre, può selezionare un numero (da 1 a 10) di immagini correlate da visualizzare.
"The Finder" è un progetto ambizioso, pensato per operare su server esterno e per essere accessibile da più utenti simultaneamente.

Questo diagramma rappresenta lo schema logico del progetto:

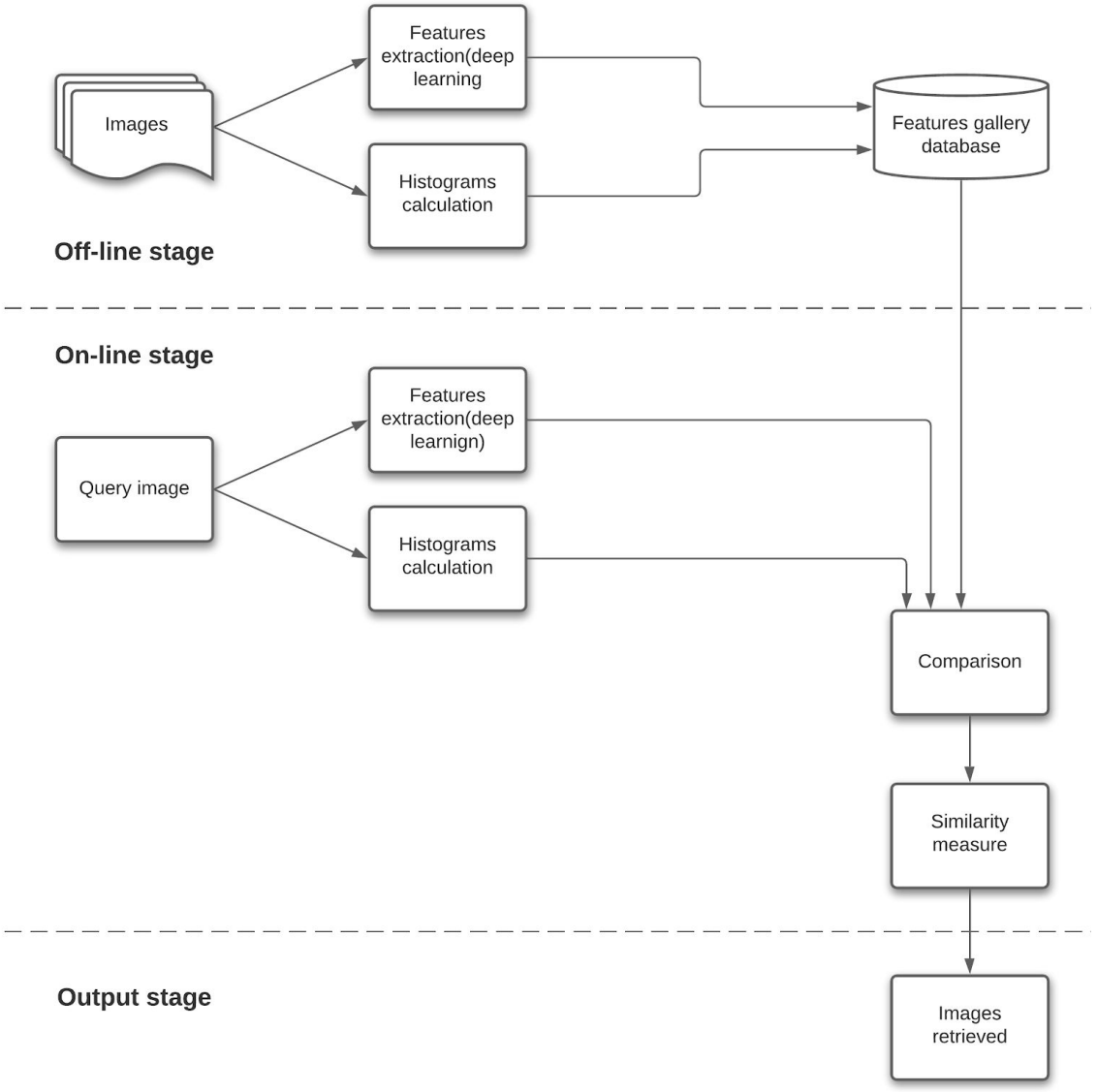

Figura 8: Schema logico del progetto

3.1 Dataset

Il dataset è la raccolta di immagini campione presenti sulla piattaforma.
Inizialmente, la nostra idea è stata quella di creare un dataset di immagini personalizzato, tramite l'utilizzo del sito Unsplash, che permetteva l'utilizzo delle immagini presenti per usi commerciali e non[17] [18].
Dopo la presentazione intermedia, sotto suggerimento del professore, abbiamo optato per un dataset già predefinito.
Nonostante ciò, il dataset trovato non soddisfaceva a pieno le nostre esigenze.
Per questo motivo abbiamo integrato il dataset trovato, eliminando il 50% delle immagini, considerate come ridondanti ai nostri occhi, con delle immagini aggiuntive da ulteriori dataset[19] [20] [21] [22].
La nostra libreria di immagini finale consiste quindi in 1150 immagini in formato .jpg.

3.2 Criteri di matching

I criteri di matching inizialmente valutati sono stati il colore (tramite la comparazione di istogrammi) e il calcolo di keypoints e descriptors (tramite algoritmi come ORB[23] e SURF). Successivamente è stato implementato un miglioramento della comparazione degli istogrammi attraverso la divisione dell'immagine in 4 finestre e il relativo calcolo dell'istogramma per ciascuna parte.

Gli istogrammi hanno prodotto i risultati preventivati mentre la ricerca tramite keypoints ha sollevato delle problematiche. Queste sono state particolarmente evidenti durante la comparazione di due immagini, apparentemente molto simili, ma con forme leggermente differenti. Abbiamo quindi verificato come gli algoritmi come ORB e SURF siano efficaci solo quando si ha la stessa figura, anche di dimensione e colore diversi ma con la stessa forma per quanto ruotata o rimpicciolita. Per questi motivi la ricerca tramite keypoints è stata accantonata in quanto difficilmente l'immagine di query avrà

[17] Unsplash: the internet's source of freely-usable images https://unsplash.com/
[18] Unsplash: License https://unsplash.com/license
[19] Content Based Image Retrieval/ Image Database Search Engine http://wang.ist.psu.edu/docs/related/
[20] UTK Face: Large Scale Face Dataset https://susanqq.github.io/UTKFace/
[21] Home Object Dataset http://www.vision.caltech.edu/pmoreels/Datasets/Home_Objects_06/
[22] Stanford Dogs Dataset http://vision.stanford.edu/aditya86/ImageNetDogs/
[23] ORB (Oriented FAST and Rotated BRIEF) https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html

esattamente gli stessi oggetti presenti in una delle immagini del dataset.

Per questo motivo, dopo una ricerca approfondita, si è ritenuto opportuno utilizzare i metodi di Deep Learning, nonostante non fossero stati affrontati nel corso. Questi metodi, tramite dei processi di convoluzione e filtraggio a successivi livelli di profondità, riescono a estrarre efficacemente features molto rilevanti, a differenza di altri metodi testati precedentemente. Queste feature (salvate sotto forma di array numerici multidimensionali) vengono poi comparate tramite distanza euclidea.

Come ulteriore affinamento degli istogrammi, sotto anche suggerimento del professore, abbiamo deciso di rendere più accurata la similarità dei colori delle immagini attraverso la segmentazione di quest'ultime in quattro finestre. Questa tecnica permette di ovviare agli inconvenienti relativi al posizionamento. Per estendere questo concetto è utile fornire un esempio: assumiamo di avere un'immagine contenente montagne con un cielo azzurro ed un'altra con il mare dello stesso colore. Senza utilizzare la tecnica di segmentazione, la similarità risulterebbe elevata sebbene le immagini abbiano contenuti differenti poiché effettuata sull'intera immagine, mentre andando a valutare 4 finestre emergerebbero differenze poiché cielo delle montagne e mare sono situati in finestre differenti.

3.3 Architettura e interfaccia web

Come accennato precedentemente, per realizzare l'architettura alla base della nostra applicazione, abbiamo utilizzato il framework Flask. Questo ci ha permesso di sviluppare la nostra interfaccia grafica mediante delle semplici pagine HTML.
L'architettura realizzata permette l'accesso a più utenti contemporaneamente tramite l'adozione e la gestione delle sessioni.
Inizialmente l'applicazione era concepita per funzionare con un solo client ma, tramite lo studio di soluzioni alternative, siamo riusciti a realizzare un progetto multi-client, in linea con la filosofia del web.

La struttura del nostro progetto è:

```
TheFinder
      theFinder.py
      requirements.txt
      /templates
            index.html
            404.html
            gallery.html
            results.html
            upload.html
      /static
            error404.png
            Image upload-bro.png
            Images-amico.png
            Search-pana.png
            Team page-amico.png
            undraw_online_gallery_dmv3.png
      /features
            1.npy
            …
      /gallery
            1.jpg
            …
      /images
            immagineUtente1.jpg
            …
      /utils
            saveHisto.py
            save_feature.py
            feature_extractor.py
            query.py
            comparazione.py
            histograms.yml
```

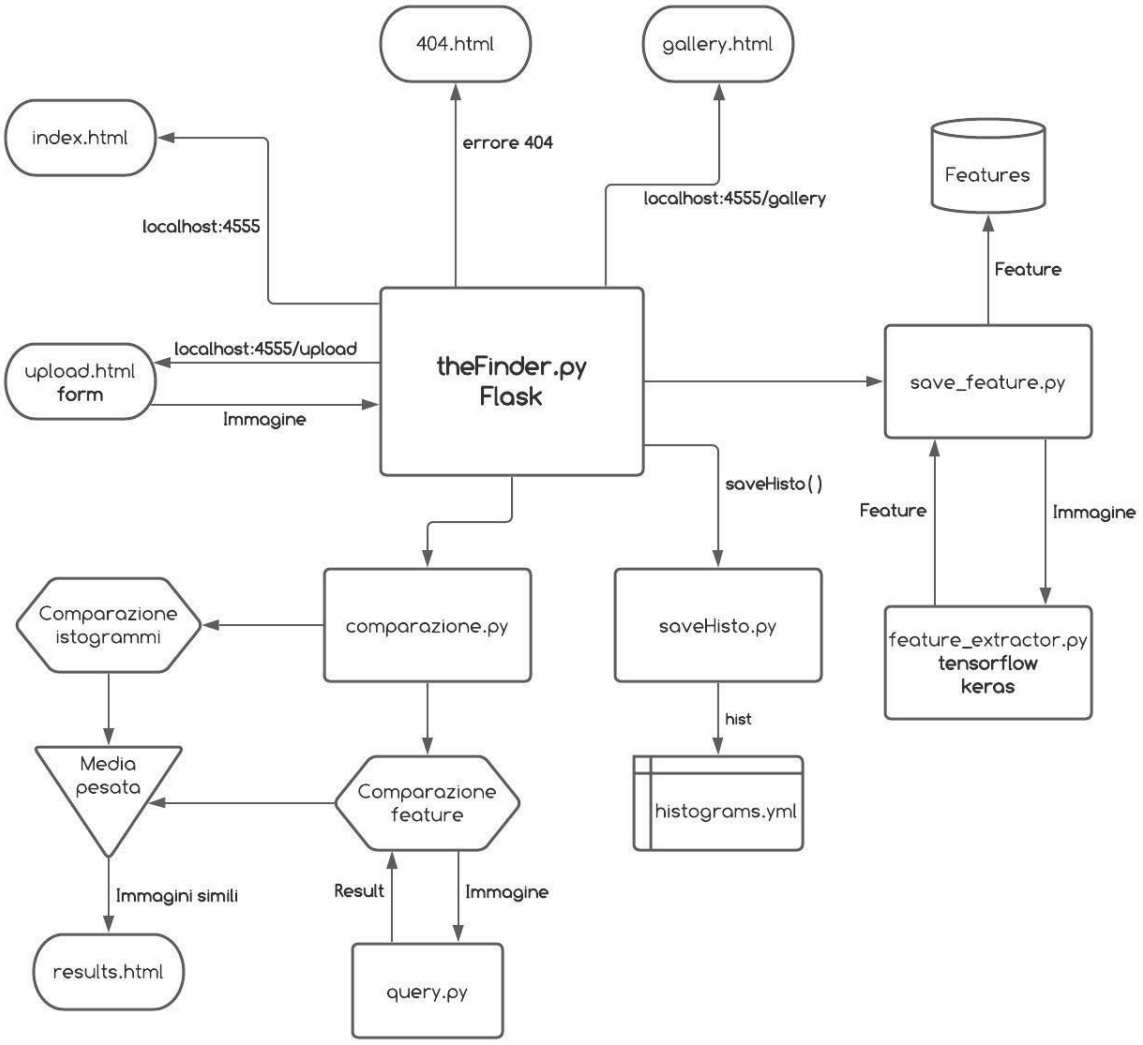

Figura 9: diagramma a blocchi dell'architettura

3.3.1 File Python

3.3.1.1 theFinder.py

Il file principale della nostra applicazione è `'theFinder.py'`, il quale presenta tutte le funzioni che permettono di gestire il server web e l'interazione con i client.

3.3.1.2 comparazione.py

File in cui avviene il confronto dell'istogramma dell'immagine caricata dall'utente con gli istogrammi delle immagini della galleria precedentemente calcolati (vedi capitolo 3.4.3).
Inoltre, una volta richiamato il metodo `compara()` presente all'interno del file `'query.py'` e aver ricevuto il vettore contenente le informazioni sulla similarità delle immagini calcolata tramite la comparazione delle features, viene calcolata la media pesata della similarità.

Abbiamo deciso di dare maggior peso all'algoritmo di deep learning (75%) e un peso minore all'algoritmo di comparazione mediante istogrammi (25%). Il risultato della comparazione viene ritornato (insieme alle due percentuali singole) sotto forma di stringa in formato JSON[24] il quale risulta leggero per l'archiviazione e il trasporto dei dati. La stringa ritornata sarà del tipo:

```
"[{'name':'nomeImmagine.jpg','percentage':percentuale,
    'percentage_histo':percentuale_istogramma,
    'percentage_features':percentuale_features},
... ]"
```

3.3.1.3 save_feature.py

Estrazione delle feature con metodi di Deep Learning dalle immagini della galleria e salvataggio nella cartella 'features'. Verrà creato un file con estensione `.npy` (array di NumPy) per ogni immagine.

3.3.1.4 query.py

In questo file vi è una funzione denominata `compara(input)`, che viene utilizzata per comparare l'immagine di input dell'utente con tutte quelle del dataset tramite la comparazione delle features estratte tramite ResNet50. Come risultato, tale funzione restituisce un oggetto contenente il nome di tutte le immagini del dataset con le relative percentuali di somiglianza rispetto all'immagine di input.
È stato sviluppato tramite algoritmo al fine di produrre tali percentuali nella maniera migliore rispetto alla nostra esigenza. (vedi capitolo 3.4.1)

3.3.1.5 feature_extractor.py

All'interno di questo file è definita la classe `FeatureExtractor` che viene richiamata da `save_feature.py` e `query.py` e che contiene effettivamente il codice dell'analisi dell'input su modello ResNet50 e l'estrazione delle features.

3.3.1.6 save_Histo.py

Tale file Python viene utilizzato al fine di creare 4 istogrammi per ogni immagine presente all'interno del nostro dataset. Essi saranno salvati all'interno di un file denominato `'histograms.yml'`.
La creazione degli istogrammi, e quindi la chiamata alla funzione '`saveHisto()`'all'interno del file '`save_Histo.py`', viene fatta ogni qual volta si decida di inizializzare il server.

Per più dettagli sul funzionamento dei singoli algoritmi, si rimanda al capitolo 3.4.

[24] What is JSON https://www.w3schools.com/whatis/whatis_json.asp

3.3.2 File HTML

3.3.2.1 index.html

Il file `index.html`, è la Homepage della nostra applicazione web. Viene caricato ogni qual volta si cerca il nostro sito tramite la barra degli indirizzi (nel nostro caso `localhost:4555`). In questa pagina vi sono elencate delle informazioni generali sul progetto.

3.3.2.2 upload.html

`Upload.html` è la pagina web che viene visualizzata dal browser per permettere all'utente di caricare una sua immagine sul Server attraverso un file picker. Inoltre l'utilizzatore del sito potrà anche scegliere quante immagini simili (da un minimo di 1 ad un massimo di 10) vorrà visualizzare.

3.3.2.3 results.html

In questa pagina saranno visualizzate le immagini simili, in accordo con il numero scelto dall'utente. Ad ogni fotografia visualizzata sarà visibile una percentuale di somiglianza rispetto all'immagine data in input dall'utente. Inoltre, passando con il mouse sopra il valore percentuale, sarà possibile visualizzare singolarmente le percentuali del confronto fra istogrammi e del confronto fra features

3.3.2.4 gallery.html

Attraverso questa pagina, è possibile visualizzare un estratto di 100 immagini del dataset. Tale pagina è stata progettata in modo tale da poter restituire immagini diverse ad ogni caricamento. Abbiamo scelto tale strategia per poter trasmettere un senso di dinamicità all'utente e anche per poter alleggerire il "lavoro" del Server.

3.3.2.5 404.html

Abbiamo realizzato questa pagina per poter creare un template personalizzato in caso di errore. Tale pagina viene anche richiamata se l'utente prima ancora di caricare la sua immagine, visualizza la pagina `results.html` (quella che dovrebbe contenere le immagini simili).

Per realizzare le pagine `upload.html`, `results.html`, `404.html` e `gallery.html` ci siamo serviti di un template predefinito fornito da w3schools[25], che abbiamo utilizzato come base di partenza, adattandolo poi alle nostre esigenze. Inoltre, per realizzare alcuni elementi grafici, abbiamo utilizzato Bootstrap, ovvero "una raccolta di strumenti grafici, stilistici e di impaginazione che permettono di avere a disposizione una gran quantità di funzionalità e di stili modificabili e adattabili a seconda delle nostre esigenze" [26]. Per utilizzare Bootstrap è sufficiente importare nella sezione `'head'` della pagina html la libreria attraverso CDN.
Nelle pagine `results.html` e `gallery.html` abbiamo utilizzato JINJA2, un motore di template attrezzato per il linguaggio Python[27]. Viene impiegato soprattutto per visualizzare le immagini all'interno delle due pagine sopra citate. È uno strumento ottimo e molto efficace che ci consente di usufruire dei parametri provenienti dal Server all'interno del browser e di utilizzare cicli for nelle pagine HTML.

3.3.3 Funzionamento dell' applicazione

In questa sezione, cercheremo di spiegare il funzionamento del Server seguendo la normale sequenza di azioni da parte di un utente.
Per prima cosa, (dopo aver iniziato l'esecuzione dell'applicativo "`theFinder.py`") ci si collega al sito digitando sulla barra degli indirizzi del browser '`localhost:4555`'. Così facendo, verrà visualizzata la pagina index.html con le informazioni del progetto. L'utente, dopo averle lette, premendo sul pulsante 'upload' verrà reindirizzato alla pagina `upload.html`. A questo punto, una volta cliccato sul file picker visualizzato al centro della pagina, il visitatore sceglierà l'immagine da caricare sul Server e selezionerà il numero di immagini simili da visualizzare. Per far avviare la comparazione delle immagini, l'utente dovrà premere sul pulsante upload. Una volta premuto, il Server creerà una sessione associata all'utente (se non esistente) e salverà l'immagine all'interno della cartella `/images`. Dopo qualche secondo si verrà indirizzati nella pagina di `results.html` dove vengono visualizzate le immagini simili, prelevate dalla cartella `/gallery` contenente tutte le foto del nostro dataset, con le relative percentuali.
L'utente potrà anche recarsi sulla pagina `gallery.html` per poter visualizzare alcune immagini disponibili del nostro dataset. Per la corretta visualizzazione delle pagine html è necessario disporre di connessione a internet, in quanto le librerie grafiche, come Bootstrap, sono importate mediante CDN (Content Delivery Network).

[25] W3.CSS Templates https://www.w3schools.com/w3css/tryw3css_templates_portfolio.htm
[26] Bootstrap: cos'è e perchè usarlo
https://verytech.smartworld.it/bootstrap-cos-e-e-perche-usarlo-353740.html
[27] Jinja - https://jinja.palletsprojects.com/en/2.11.x/

3.4 Sviluppo algoritmi

Sono stati sviluppati principalmente due algoritmi in maniera parallela: il principale, per l'estrazione delle feature rilevanti tramite ResNet50, e il secondario, per il confronto dei colori tramite degli istogrammi calcolati su 4 finestre. Inoltre, era stato sviluppato, senza aver portato a risultati soddisfacenti, un codice per l'estrazione di keypoints e descriptors tramite algoritmo ORB.

3.4.1 Estrazione feature tramite ResNet50

Keras permette di creare nuovi modelli oppure utilizzarne di già esistenti, nel nostro caso ResNet50. In particolare, non è stato utilizzato il modello nella sua interezza perché non siamo interessati alla funzione di predizione e classificazione (che viene effettuata dagli ultimi livelli del modello) ma solo a quella di estrazione features. Per questo motivo, l'ultimo livello utilizzato è quello denominato `"AVG_POOL"`.

Il codice, dopo aver istanziato un modello, passa un immagine (ridimensionata a 224x224 per corrispondere alle specifiche richieste da ResNet) e la rende una matrice Numpy.
Il modello tramite funzioni predefinite passa perciò a estrarre le features[28] .
Si procede poi al salvataggio di tali array multidimensionali contenenti le features in file .npy diversi per ciascuna immagine tramite funzioni della libreria Numpy.[29]
Ciò avviene al primo avvio del server di Flask; una volta salvati tali file .npy e mantenendo lo stesso dataset sarà possibile saltare questo processo, che è il più oneroso a livello computazionale. Ad ogni modifica del dataset all'interno del server, sarà invece necessario richiamare la funzione e ricalcolare tutte le features.

All'inserimento di un'immagine di query, vengono richiamate tutte le features dai singoli file e vengono appese a una lista unica.
Vengono poi estratte le feature dell'immagine di query e a quel punto viene calcolata la distanza euclidea (tramite funzione matematica di Numpy)[30] fra tutte le features salvate e quelle di query. Tale caratteristica rappresenta quanto "vicine" sono le rappresentazioni delle immagini. Se la distanza è 0 le immagini sono uguali.

[28] Extract features from an arbitrary intermediate layer with VGG19
https://keras.io/api/applications/#extract-features-from-an-arbitrary-intermediate-layer-with-vgg19
[29] Build An Image Search Engine Using Python
https://towardsdatascience.com/build-an-image-search-engine-using-python-ad181e76441b
[30] How can the Euclidean distance be calculated with NumPy?
https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy

Il range di distanze, dopo diversi test, è stato notato come andasse da circa 0.8 in caso di somiglianza evidente (anche meno di 0.6 in caso di stesse figure in frame differenti o somiglianze al limite dell'uguaglianza) fino a circa 1.5/1.6.

Tale range è stato perciò adattato[31] e invertito, in maniera che si passasse a un range 0-100 rappresentabile come percentuale.
In particolare, il valore massimo del vecchio range (quindi l'immagine più distante) è stato preso come nuovo minimo (corrispondente a 0), mentre come valore minimo del vecchio range è stato preso 0.5 in maniera empirica (oppure 0 nel raro caso ci dovesse essere una distanza inferiore a 0.5) per ottenere percentuali realistiche.
I punteggi così ottenuti vengono infine associati in ordine decrescente in un array all'indice dell'immagine da cui sono stati calcolati.

Gli svantaggi riscontrati sono:

-velocità di analisi e salvataggio features che dipende pesantemente dalle prestazioni della CPU e RAM:
dai nostri test impiega per più di 1000 immagini più di 3 minuti. Tale attesa è comunque accettabile poiché avviene solo al primo avvio dell'applicazione nel server, quando ancora l'utente non può accedere alla piattaforma web; o meglio ancora basterebbe calcolare le feature una sola volta in assoluto (a meno che non si aggiungano nuove immagini nel dataset) poiché esse vengono salvate fisicamente nel server sotto forma di file .npy all'interno della cartella "features".

-dimensione della libreria:
TensorFlow pesa approssimativamente 400 MB, mentre la dimensione dei weights precalcolati di ResNet50 è circa 90 MB. Anche questo svantaggio può essere giustificato in quanto l'applicazione è stata idealmente pensata per funzionare in un server dedicato, con dimensionamento sufficiente per contenere le immagini, le librerie e le ulteriori componenti, senza andare a gravare in alcun modo sull'utente finale.

[31] Convert a number range to another range, maintaining ratio
https://stackoverflow.com/questions/929103/convert-a-number-range-to-another-range-maintaining-ratio

3.4.2 Keypoints e descriptors tramite ORB

Inizialmente, per cercare una corrispondenza di forme era stato valutato di utilizzare l'algoritmo ORB, contenuto nella libreria OpenCV.
Nel codice si estraevano i descriptor di tutte le immagini del dataset e li si salvava sotto forma di file JSON[32].
All'inserimento di un'immagine di query, si recuperavano tutti i descriptors e li si andavano a confrontare con quelli della nuova immagine tramite FLANN[33]. Il punteggio di similarità era ottenuto calcolando il numero di "matched points", ossia di punti confrontati con esito positivo, su numero totale di punti contenuti nei descriptor dell'immagine.

Questo metodo è risultato inefficace, come già spiegato nel capitolo 3.2, perciò il codice prodotto è stato scartato.

3.4.3 Istogrammi a 4 zone

Inizialmente gli istogrammi venivano calcolati sull'intera immagine. Il procedimento è di seguito spiegato.
La prima parte consisteva nel calcolo dell'istogramma per l'intera immagine tramite 'calcHist' (con i tre canali RGB, ogni canale con 8 bins e range 0-255)[34].
Successivamente la funzione flatten e la normalizzazione davano luogo a un vettore di dimensione 512 (8*8*8) che veniva salvato tramite classe apposita di OpenCV FileStorage[35] (in formato YAML) in un file chiamato `histograms.yml`.

Nella fase di query, veniva a sua volta calcolato l'istogramma complessivo dell'immagine di input e venivano estratti singolarmente tutti gli istogrammi presenti nel file su disco (tramite il metodo 'getNode' di FileStorage), e ciclicamente si procedeva infine al confronto tramite la funzione 'compHist' di OpenCV, con un range di risultati numerici con un massimo di 1 se sono identici (il minimo può assumere anche un valore negativo)

Successivamente, è stato deciso (vedi capitolo 3.2) di migliorare il metodo introducendo una suddivisione in diverse finestre

[32] Saving KeyPoint as String and converting back to KeyPoint
https://stackoverflow.com/questions/56267675/saving-keypoint-as-string-and-converting-back-to-keypoint
[33] Feature Matching with FLANN
https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html
[34] How-To: 3 Ways to Compare Histograms using OpenCV and Python
https://www.pyimagesearch.com/2014/07/14/3-ways-compare-histograms-using-opencv-python/
[35] File Input and Output using XML and YAML files
https://docs.opencv.org/master/dd/d74/tutorial_file_input_output_with_xml_yml.htm

dell'immagine e procedendo poi al calcolo e al confronto per ogni parte.[36]
La suddivisione è stata ottenuta prendendo le dimensioni dell'immagine e dividendole per 2 (arrotondando per eccesso con math.ceil per evitare di creare ulteriori finestre ai bordi se la dimensione è dispari).

A quel punto, considerando l'immagine come matrice, è possibile considerare intervalli di matrice corrispondenti a metà delle dimensioni e numerarli tramite identificativi (n_riga, n_colonna): ad esempio la prima finestra in alto a sx sarà (0,0), la seconda finestra in alto a dx (0,1) e così via (vedi figure sotto).

Figura 10: immagine di esempio [37]

finestra [0,0]

finestra [0,1]

finestra [1,0]

finestra [1,1]

Figura 11: immagine suddivisa in 4 finestre

[36] Divide an image into 5x5 blocks in python and compute histogram for each block https://stackoverflow.com/questions/22685274/divide-an-image-into-5x5-blocks-in-python-and-compute-histogram-for-each-block
[37] Source: https://www.wallpaperup.com/1141348/Nature_Convict_Lake_Autumn.html

Creando un ciclo all'interno dell'immagine per ogni finestra così ottenuta è possibile calcolare l'istogramma specifico per quella finestra (con gli stessi metodi visti sopra) e salvarlo come nodo `"histogram_{numriga}_{numcol}_{nomeFile}"` all'interno del file histograms.yml. Si calcolano quindi 4 istogrammi per file, nel nostro caso per un totale di circa 4400 vettori. Tali operazioni hanno durata complessiva di pochi secondi.

Nella fase di query si utilizza lo stesso procedimento per estrarre 4 istogrammi dall'immagine di input (tale informazione non viene però salvata su file ma mantenuta in un dictionary (array associativo) locale, dove gli indici sono `"histogram_{numriga}_{numcol}"` e i valori sono i vettori normalizzati rappresentanti l'istogramma).
Come ultima fase, viene effettuato un ciclo per il numero di immagini del dataset e all'interno un ulteriore ciclo per poter accedere alle singole finestre: per ogni finestra, viene estratto dal file histograms.yml il nodo corrispondente e viene effettuato il confronto fra il nodo e l'istogramma della query tramite il metodo di correlazione (vedi fig. 12) [38].

Correlation

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

where

$$\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$$

and N is a total number of histogram bins.

Figura 12: formula per correlazione istogrammi

Il risultato verrà poi sommato ai risultati delle altre finestre e diviso per il numero di finestre (4) per ottenere la media complessiva dell'intera immagine. La media viene poi appesa ad un dictionary, usando come indice il nome del file di immagine appena analizzato. Le operazioni appena descritte vengono ripetute per ogni file del dataset.
Il range viene adattato nello stesso modo spiegato in 3.4.1 per avere come valore minimo 0 e non valori negativi (il massimo rimane uguale).
L'array associativo viene infine ordinato per valore in ordine decrescente[39] per ottenere ai primi posti le immagini con correlazione più alta (dove 1 è un istogramma identico; per il calcolo percentuale verrà poi moltiplicato per 100).

[38] OpenCV documentation
https://docs.opencv.org/3.4/d6/dc7/group__imgproc__hist.html#gga994f53817d621e2e4228fc646342d386a38c6d2cb4ae572e00aebd5642fae2add
[39] Sorting Associative Array in Python
https://www.codespeedy.com/sorting-associative-array-in-python/

3.5 Integrazione e testing

La fase di integrazione, effettuata sia sugli algoritmi che sulla GUI, è stata una delle fasi più laboriose del progetto. In questo frangente abbiamo dovuto adattare gli algoritmi individuati nelle fasi precedenti e il loro codice per permettere il loro corretto funzionamento in una struttura complessa, in un'ottica di "cooperazione" tra gli algoritmi stessi, per ottenere in fase di testing, un risultato il più accurato possibile.
Stessa situazione si è verificata con la GUI, che da semplice architettura web-based è stata ottimizzata per interagire col codice e gli algoritmi da noi implementati.

L'ultima fase di verifica del progetto è stata la fase di testing.
La fase di testing è stata fondamentale per valutare il comportamento del nostro programma e per individuare problematiche altrimenti invisibili con la sola scrittura del codice.
Inizialmente il testing degli algoritmi di comparazione è avvenuto "offline", da terminale python, per poter correggere man mano gli errori prima di condividere il codice con il team dell'interfaccia web.
Successivamente una volta integrate le due strutture, si è proceduti a un testing complessivo con vari casi di situazioni.
Per iniziare, abbiamo svolto tutte le procedure di avvio del progetto descritte in precedenza (vedi capitolo 3.3.3 - Funzionamento dell' applicazione). Abbiamo testato e appurato il corretto funzionamento della pagina di galleria, dove vi sono mostrate 100 immagini campione, prese in maniera randomica dal dataset, e della pagina di upload, dove è possibile cliccare su un file picker e scegliere un file di formato .png o .jpg, altrimenti viene visualizzato un errore.
Appurato il corretto funzionamento delle fasi iniziali, abbiamo testato la parte principale e più significativa della nostra architettura.

La fase di testing ha portato alla luce delle pecche.
Avendo optato per un dataset di limitate dimensioni, per non rendere l'architettura eccessivamente pesante, non sempre è stato possibile avere in output una buona corrispondenza fra l'immagine caricata dall'utente e le immagini presenti all'interno del dataset.
In particolare, testando l'architettura, abbiamo riscontrato un'affidabilità dei risultati maggiore quando venivano caricate dall'utente immagini di paesaggi in generale (montagne, spiagge, tramonti, città ecc.) o animali/oggetti già presenti nel dataset. Se invece venivano caricate immagini contenenti animali estranei al dataset oppure oggetti specifici, i risultati ottenuti erano meno accurati ma comunque correlati (es. un leopardo, non presente nel dataset, dà come risultati altri tipi di animali ma non ad esempio paesaggi).

In generale, tutti i malfunzionamenti e i bug sono stati segnalati nella pagina `issues` del progetto in GitHub, per tenerne traccia e notificare una volta che si erano risolti.
Ad esempio, altre problematiche e mancanze riscontrate nel corso dello sviluppo sono state dei bug presenti nella visualizzazione dell'index e della griglia delle immagini, difficoltà nell'implementare le funzionalità riguardanti la divisione in 4 finestre durante il calcolo degli istogrammi, il passaggio da VGG16 a ResNet50, l'aggiunta del file `requirements.txt`, l'aggiunta del tooltip per la visualizzazione delle singole percentuali e problematiche legate alla documentazione, quali la stesura del file README.md (dov'è contenuta una breve introduzione del progetto) e l'aggiunta dei commenti interni al codice, per permettere anche agli altri componenti del gruppo di poter avere, a colpo d'occhio, un breve resoconto del funzionamento del codice scritto.

Tutto il progetto è stato sviluppato e testato rendendo la propria macchina un server locale e accedendo tramite localhost:4555 alla pagina del browser.
Ad ogni modo, per simulare effettivamente il funzionamento di tale progetto se fosse situato in un server esterno, tramite una piccola modifica al codice[40] è stato possibile accedervi tramite un altro dispositivo (collegato alla stessa rete). Tale test ha avuto successo e ha confermato come all'utente finale non sia necessario alcun download e che la velocità di esecuzione delle query dipenda dall' HW della macchina server. Tuttavia, l'attuale struttura delle pagine web è ottimizzata per dispositivi desktop, da mobile sono stati riscontrati dei problemi (non si è ritenuto di risolverli perché sarebbe andato ben oltre agli scopi principali del progetto).

[40] Configure Flask dev server to be visible across the network
https://stackoverflow.com/questions/7023052/configure-flask-dev-server-to-be-visible-across-the-network

4 Conclusioni

4.1 Risultato finale

Il risultato finale del progetto è complessivamente soddisfacente e rispetta le richieste della task iniziale.
Caricando un'immagine sul Server, vengono mostrate in output le immagini del dataset più simili a quella caricata, in numero equivalente a quello scelto dall'utente. Ogni immagine del dataset presenta una percentuale che va ad indicare la similitudine con l'immagine caricata e, se il cursore viene posizionato sulla percentuale, viene mostrata la percentuale di somiglianza data dalla singola comparazione dell'istogramma e dalla comparazione delle features.

Chiaramente, ci sono dei limiti dovuti principalmente alla dimensione del dataset memorizzato (circa 1.150 immagini).
Le difficoltà e i principali punti deboli sono descritti in seguito (cap. 4.2)

Le seguenti tre figure mostrano come si presenta all'utente il prodotto finale:

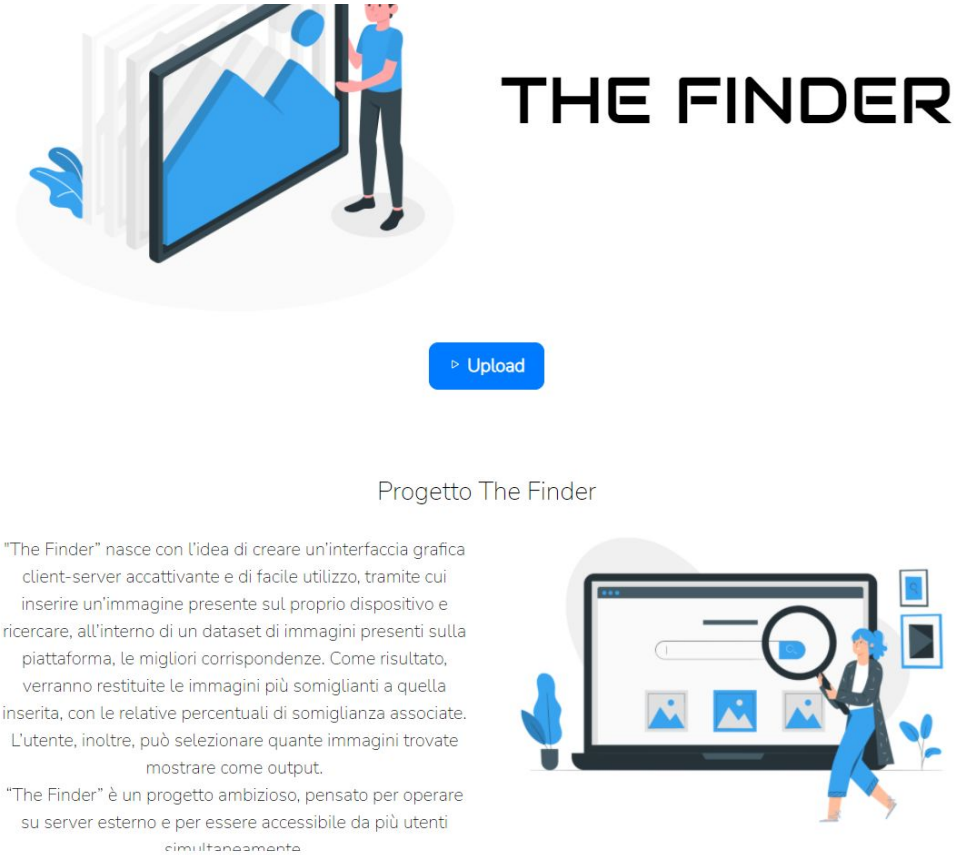

Figura 13: screenshot della pagina home

Figura 14: screenshot della pagina di upload

Figura 15: screenshot della pagina con i risultati

4.2 Difficoltà riscontrate nella realizzazione

In primo luogo, non avendo mai utilizzato Python, ci siamo dovuti documentare sul linguaggio e, prima di iniziare a programmare per il nostro progetto, abbiamo eseguito dei test individuali.
All'inizio avevamo pensato di sviluppare "The Finder" come un'applicazione desktop, attraverso una libreria messa a disposizione da Python denominata "Tkinter"[41]. Abbiamo deciso però di abbandonare questa strada, per permettere all'utente di utilizzare la nostra web app in qualsiasi posto e da qualsiasi dispositivo.
Fin da subito ci siamo accorti della difficoltà elevata nell'utilizzo di diversi linguaggi e la loro interazione tra il Server e il Client. In questo frangente, Flask è stato fondamentale poiché ci ha permesso di scambiare i dati in modo semplice ed efficace.
Durante la fase di realizzazione dell'architettura e dell'interfaccia grafica, ci siamo resi conto che la soluzione che stavamo adottando era molto miope, perché non permetteva la connessione simultanea di più utenti. Dopo aver realizzato l'esistenza di questa problematica, abbiamo analizzato il problema e si è giunti alla decisione di riprogettare il sistema per ovviare a questa carenza.

Per quanto riguarda lo sviluppo degli algoritmi, come già riportato nel capitolo 3.2 e 3.4, sono state seguite differenti linee di pensiero rispetto a quanto ipotizzato inizialmente, poiché gli strumenti "standard" non permettevano l'ottenimento di risultati soddisfacenti: dopo un iniziale approccio di documentazione sulle reti convoluzionali neurali, è stato deciso di approfondire tale metodo e cercare di comprenderne a grandi linee il funzionamento, riuscendo infine ad implementarlo pur scendendo a compromessi riguardo a velocità di esecuzione e dimensione delle librerie richieste come prerequisito. Abbiamo, pertanto, preferito ottenere una buona resa complessiva ma con metodi macchinosi rispetto a un semplice sviluppo lineare .

Abbiamo deciso di mantenere anche il risultato della comparazione degli istogrammi (successivamente migliorato con la suddivisione in 4 zone) perché volevamo tenere traccia della similitudine delle immagini in base al loro colore. Abbiamo assegnato empiricamente una percentuale del 25% al valore di comparazione degli istogrammi, rispetto al 75% delle features, perché non volevamo influenzare in maniera eccessiva i risultati già apprezzabili dell'algoritmo principale.

[41] Tkinter - https://docs.python.org/3/library/tkinter.html

Inoltre, l'attuale situazione sanitaria ha reso più difficoltoso il lavoro di gruppo, non essendo possibile incontrarci di persona. Fortunatamente, attraverso l'utilizzo di strumenti quali Zoom, GitHub e Google Drive, siamo riusciti comunque a collaborare in modo ottimale e ordinato.
Infatti, la suddivisione in ruoli e conseguente parallelizzazione fra sviluppo interfaccia web e sviluppo algoritmi ha funzionato piuttosto bene.

4.2 Sviluppi futuri

Se il progetto dovesse proseguire, una delle questioni più importanti da risolvere sarebbe sicuramente la riduzione dei tempi di elaborazione delle immagini ed estrazione delle features. Una possibile soluzione potrebbe essere quella di ospitare la nostra applicazione su un Server 'reale', il quale avrebbe chiaramente una maggiore potenza di calcolo e di conseguenza ridurrebbe il tempo necessario all'elaborazione delle immagini. Ciò permetterebbe anche di aumentare il numero di immagini presenti nel dataset.

Gli svantaggi dell'utilizzo del Deep Learning (dimensione e complessità computazionale) vengono inoltre compensati dall'efficacia complessiva di questi metodi, e inoltre è da considerare che il deep learning è una scienza in evoluzione e ogni anno vengono studiati nuovi tipi di rete con miglioramenti sostanziali (vedi fig. 16).[42]

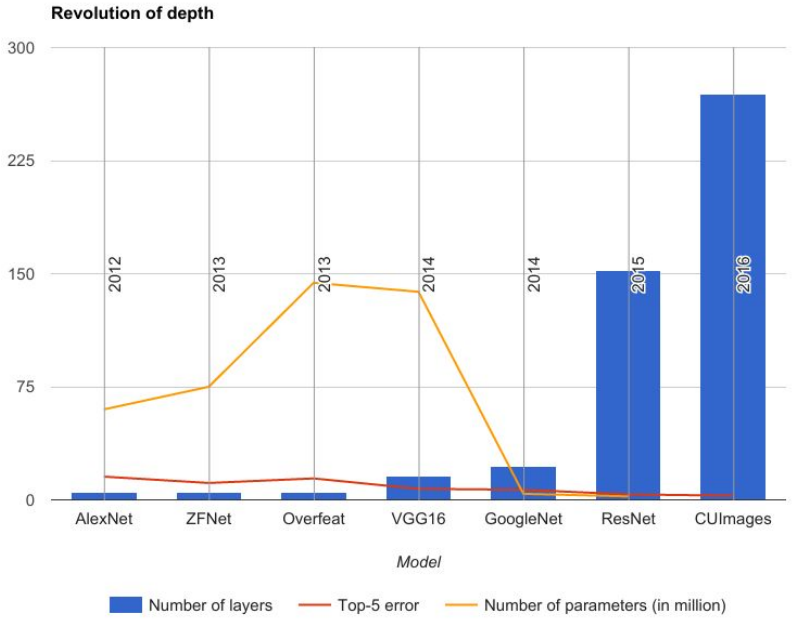

Figura 16: evoluzione nel tempo di Convolutional Neural Networks

[42] CNN — Do we need to go deeper?
https://medium.com/finc-engineering/cnn-do-we-need-to-go-deeper-afe1041e263e

Successivamente si potrebbe integrare all'interno della nostra applicazione un database in grado di salvare diverse informazioni degli utenti e delle immagini prese in questione.

Un'ulteriore sviluppo potrebbe essere quello di espandere il nostro parco algoritmi implementandone di nuovi ed efficaci. Il tutto può essere anche integrato con una scelta, da parte dell'utente, dei criteri con cui effettuare la comparazione delle immagini (per esempio solo attraverso i colori, oppure con i metadati ecc...). Infine, si potrebbero apportare delle migliorie all'interfaccia grafica, perfezionando degli elementi grafici e rendendola maggiormente user-friendly.