UNIVERSITÀ DI TRENTO

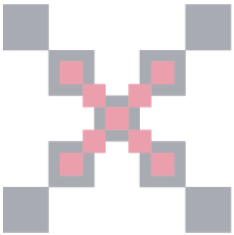
PIXELS

Corso di Comunicazioni Multimediali
A.A. 2020/2021

Report finale di progetto

Ferronato Alessandro
Giro Giorgia
Parpinello Davide
Peruzzo Martina

SOMMARIO



1 Introduzione

1.1 Idea

Pixels è una web app che nasce dall'esigenza di organizzare una libreria di immagini partendo dal confronto con un'immagine in input fornita dall'utente.

A differenza di altri sistemi con simili funzionalità, la nostra web app non utilizza informazioni ottenute dai metadati che, il più delle volte, non sono efficienti. Per evitare che ciò avvenga, la nostra web app estrae direttamente le informazioni necessarie dalle immagini presenti.

Grazie alle tecniche di computer vision utilizzate, ogni elemento della ricerca è sotto il diretto controllo del nostro sistema e i criteri considerati sono più coerenti possibili con il nostro obiettivo ovvero rendere la piattaforma oltre che bella esteticamente anche semplice e intuitiva per l'utente finale.

1.2 Sviluppo

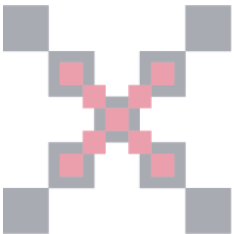
Per riuscire a concretizzare la nostra idea abbiamo deciso di dividere il suo sviluppo in cinque diverse fasi:

- *Definizione dei requisiti*
  Identificazione degli elementi necessari allo sviluppo della piattaforma e del sistema che la supporta al fine di fornire basi solide alla nostra idea.

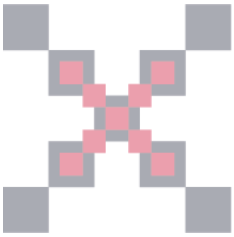
- *Sviluppo dello scheletro della web app e design*
  Creazione dell'interfaccia grafica per permettere all'utente di visualizzare e interagire con la piattaforma nel modo più intuitivo e semplice possibile.

- *Sviluppo algoritmi di comparison in Python*
  Realizzazione degli algoritmi che il sistema utilizza per analizzare e confrontare le varie immagini: Confronto fra istogrammi, SSIM (Structural Similarity Index Measure) e YOLO.

- *Interfacciamento dell'algoritmo con la web app*
  Unione delle diverse componenti già sviluppate con l'aggiunta di un dataset di 150 immagini ed eventuali adattamenti alle singole parti in modo da garantire la migliore cooperazione possibile durante la fase di estrazione delle immagini e ritorno dei risultati del confronto.



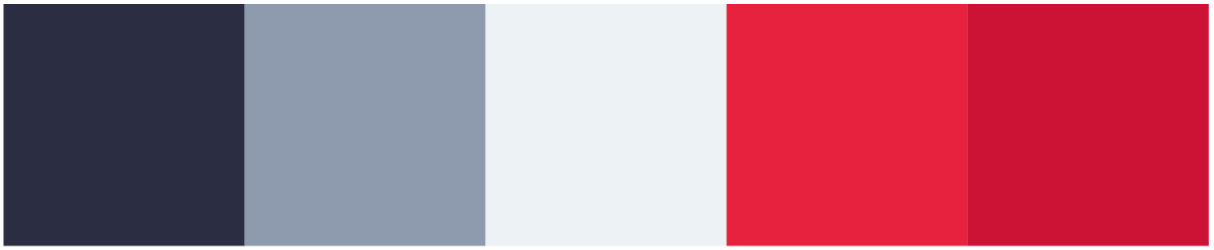
- *Testing e valutazione*
  Una volta ultimata l'implementazione principale del progetto se ne verifica il corretto funzionamento attraverso una serie di prove ed aggiustamenti dove necessario.
  Il risultato ottenuto viene quindi valutato controllando che si siano raggiunte le aspettative iniziali.



2 Stile

È stata prestata particolare attenzione alla parte estetica e visiva del sistema, partendo dalle immagini scelte per popolare il nostro dataset fino alla struttura della pagina web, in quanto riteniamo che questa componente possa incentivare ancor più gli utenti a scegliere la nostra piattaforma.

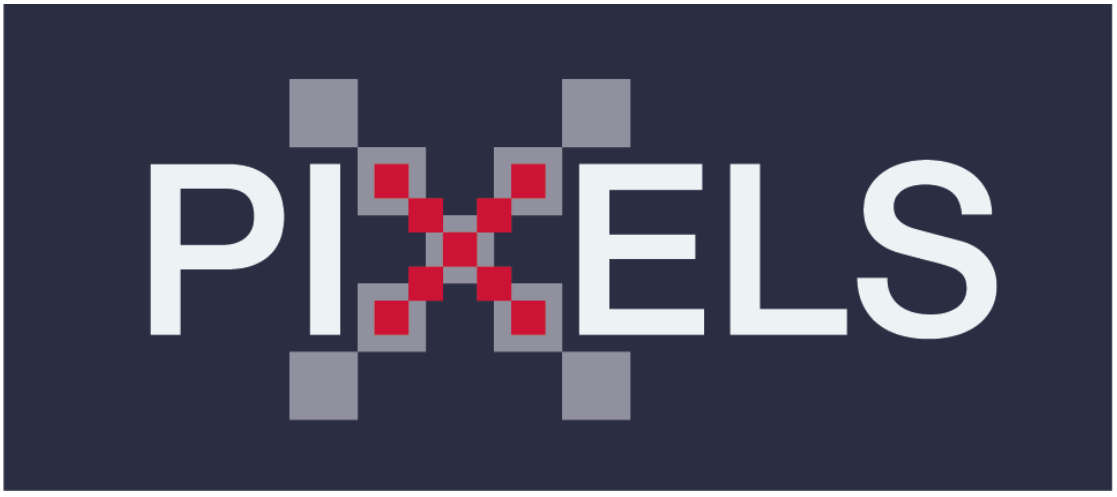
2.1 Colori



Come si può notare sono stati scelti colori che fossero in armonia tra loro, ma d'impatto.

Per rendere la grafica del sito il più gradevole possibile alla vista, la palette è stata creata partendo da sfumature del blu, simbolo di armonia ed equilibrio.
A queste sono state associate a completamento tonalità fredde del rosso che si legassero bene con le sfumature di blu scelte, ma senza perdere il senso di energia e vitalità che il rosso rispecchia.

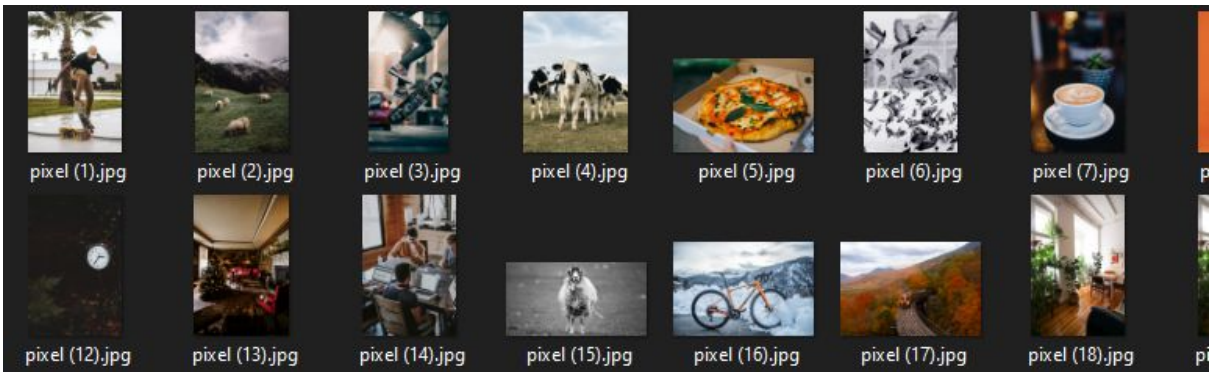
2.2 Logo



Usando i colori scelti come base cromatica per il sito, è stato creato il logo. Il logo è la carta d'identità del nostro team. L'obiettivo è far riconoscere il gruppo: per questo, il nostro logo compare sia nelle presentazioni che nel sito web.

Il nome Pixels è nato inizialmente dal fatto che il progetto fosse relativo a una libreria di immagini. Poi, cercando di dare una definizione un pò più filosofica, abbiamo osservato come ogni membro del gruppo possa essere visto come un singolo pixel, che individualmente non dice molto, ma una volta unito agli altri membri, è parte fondante dell'immagine della nostra squadra.

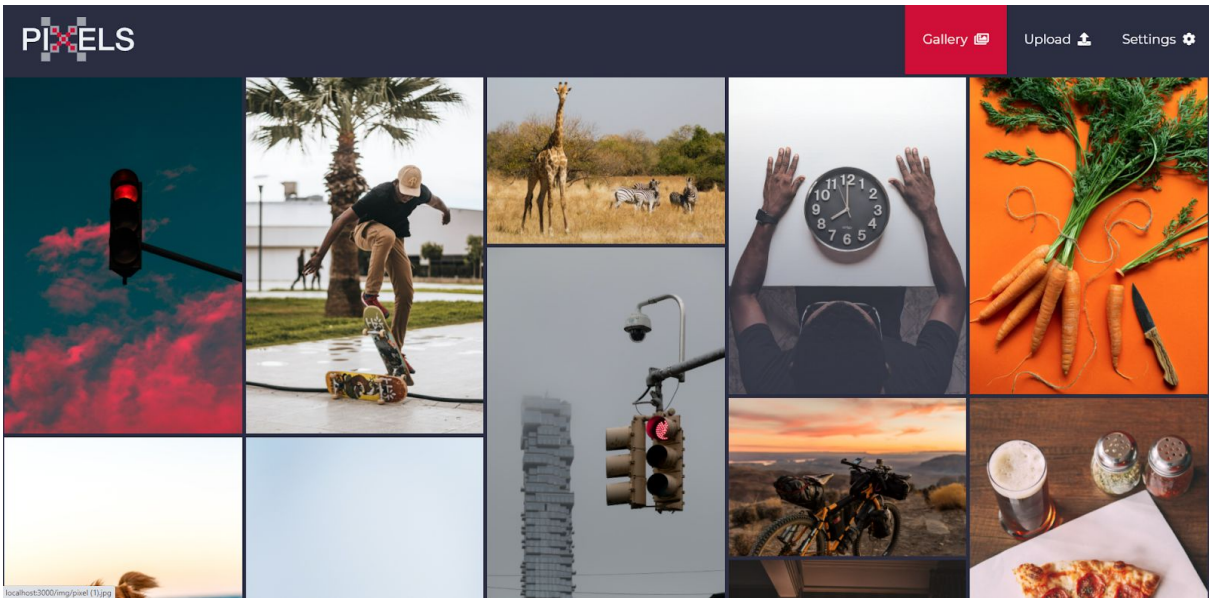
2.3 Scelta del dataset



Il dataset è costituito un totale di 150 immagini JPG con dimensioni medie di 600KB. In linea con quanto detto precedentemente, riguardo all'importanza della componente estetica che vogliamo dare alla web app, anche le varie immagini che popolano il dataset sono state selezionate seguendo quest'ottica.

Essendo la prima cosa che l'utente vede quando entra nella web app Pixels, le immagini, per entrare a far parte del nostro dataset, dovevano soddisfare degli standard di qualità e ricercatezza.

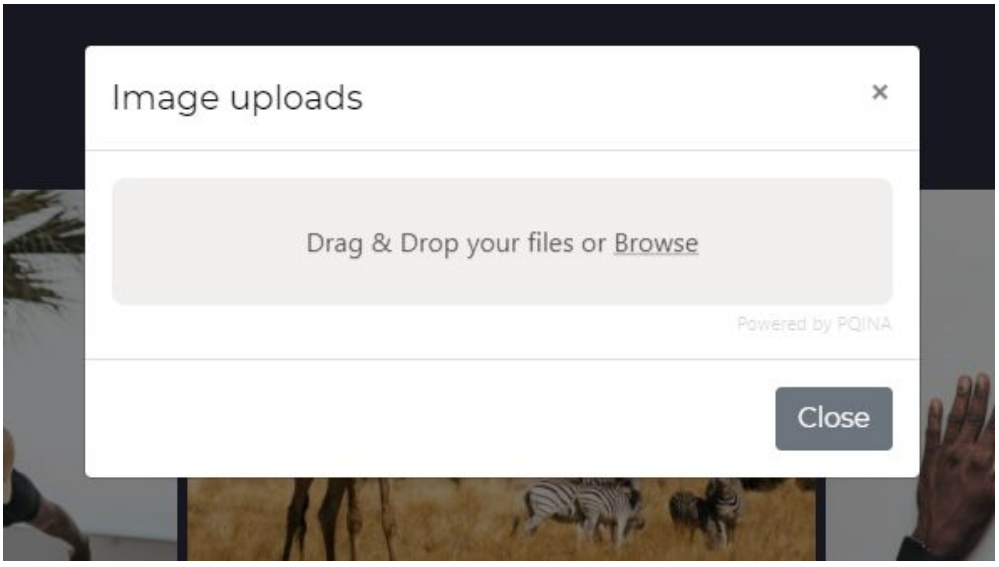
2.4 Grafica sito

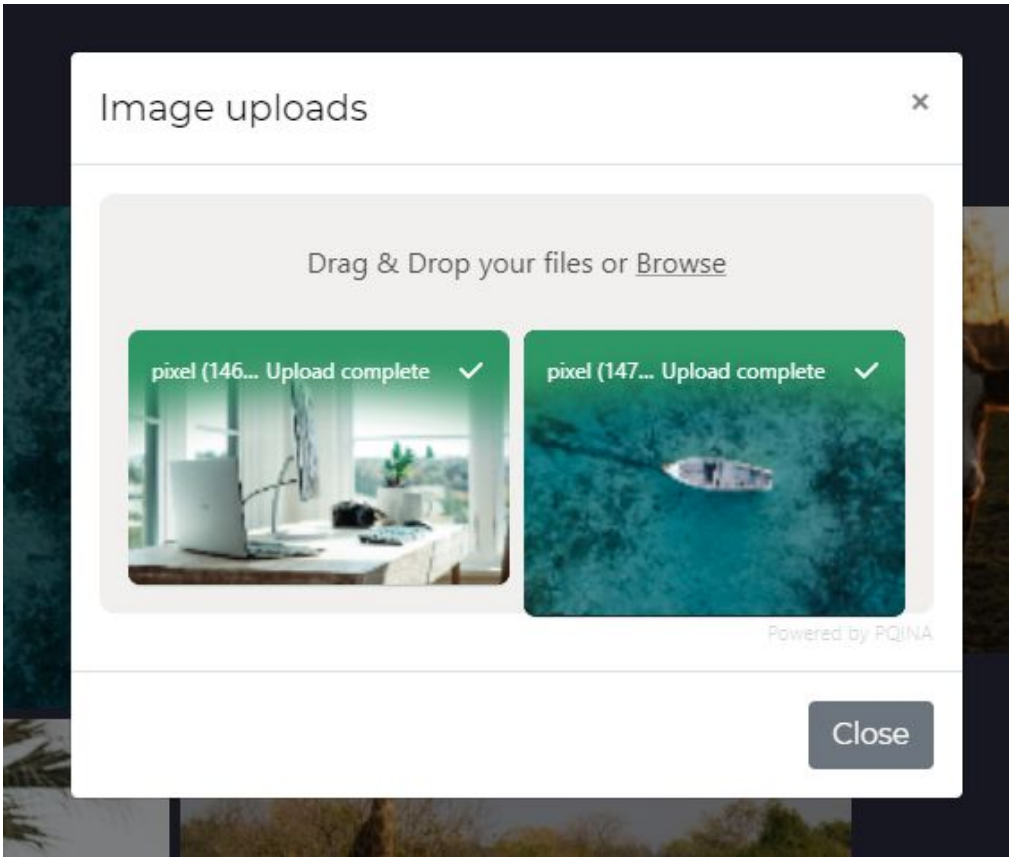


Per quanto riguarda la grafica del sito, è stata scelta una disposizione che valorizzasse il più possibile le immagini scelte.
Si sono quindi disposte le immagini in modo da evitare eventuali buchi derivanti da dimensioni diverse.

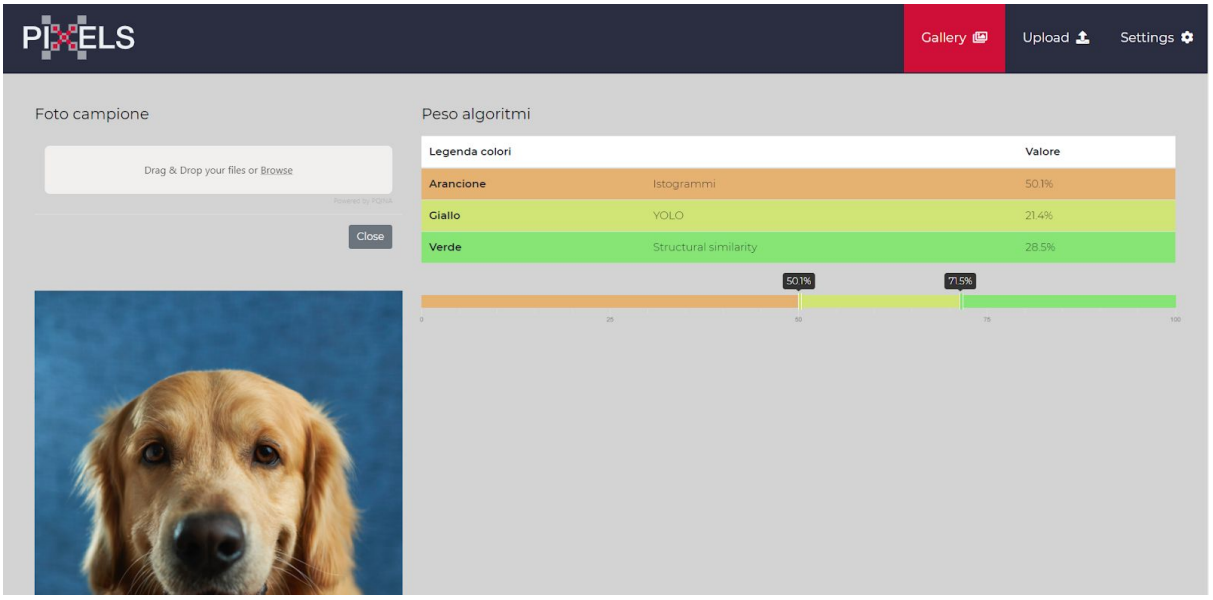
La web app permette una volta cliccata un'immagine di visualizzarla a pieno schermo e inoltre di scorrere la libreria in modo fluido.

Cliccando in alto a destra su "Upload" il sito permette di aggiungere una o più immagini, come mostrato nelle seguenti figure:



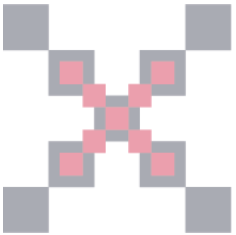


Una volta caricate le immagini compariranno nel sito insieme a quelle già presenti.



Attraverso il pulsante “Settings” è possibile accedere ad una pagina di configurazione, dove è possibile caricare l’immagine campione da utilizzare come base per i confronti.

In questa finestra si può anche personalizzare il peso che i tre algoritmi avranno sul risultato finale calcolato da Python attraverso uno slider. Lo script in Python controlla automaticamente se il campione è stato cambiato e, se è così, le percentuali vengono aggiornate.



3 Linguaggio e Librerie

Per questo progetto abbiamo scelto di lavorare con due diversi linguaggi uno per la parte che riguarda l'architettura della web app e uno per quella che riguarda gli algoritmi.

Per la parte dell'interfaccia grafica e architettura della web app è stato utilizzato Node.js mentre per la parte degli algoritmi di estrazione e confronto è stato usato il linguaggio Python.

All'interno degli algoritmi di comparison sono state importate le seguenti librerie:

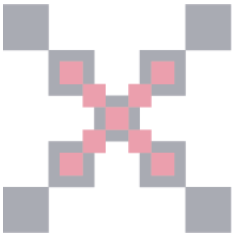
- *OpenCV e NumPy*
  per le principali funzionalità di elaborazione immagini e per l'estrazione e confronto degli istogrammi;

- *Darknet*
  per creare la rete neurale necessaria alla parte di algoritmo dedicata all'Object Detection (dipendenze: Torch e Matplotlib);

- *Json*
  per riuscire a scrivere e leggere file di tipo *.json* in cui salvare e successivamente estrarre i dati delle varie operazioni, ciò permette di non dover creare un database vero e proprio;

- *Request*
  per gestire le richieste GET da effettuare al server web utilizzate per caricare i risultati dei confronti;

- *Scikit-image*
  per la funzione che esegue il calcolo di SSIM.



4 Progetto

4.1 Criteri di matching

I criteri di matching che abbiamo scelto di applicare all'interno della nostra web app sono tre:

- il confronto fra gli **istogrammi**, per avere una rappresentazione della distribuzione dei valori nei vari pixels rispetto allo spettro cromatico RGB e anche dell'immagine sulla scala di grigi;

- il secondo criterio invece, è il confronto fra le liste di oggetti riconosciuti nelle immagini prese in considerazione, tramite una tecnica di **Object Detection** chiamata Yolo.

- l'ultimo criterio implementato è **SSIM**, questo ci permette di verificare la relazione fra i pixels adiacenti, così da evitare il focus solamente sulla distribuzione dei valori all'interno dell'immagine.

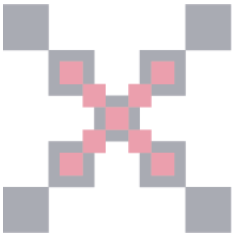
4.2 Script in Python

Per quanto riguarda l'implementazione vera e propria della web app, partiamo con l'analizzare lo script in Python che permette di estrarre dalle immagini le features scelte e il loro successivo confronto, fino alla restituzione della percentuale di somiglianza al server Node che gestisce la web app.

Il programma *pixels.py* si occupa dell'estrazione delle features dalle varie immagini e del loro confronto. Adesso analizziamo più nello specifico alcune parti del codice per capire meglio ciò che effettivamente fa l'algoritmo Python.

4.2.1 Estrazione e confronto fra istogrammi di colore

Il primo criterio che abbiamo scelto per comparare le varie immagini è il confronto fra gli istogrammi di colore, che rappresentano la distribuzione dei valori assunti dai pixels all'interno delle nostre immagini. Abbiamo scelto di aggiungere anche l'istogramma della scala di grigi, a seguito di una conversione dell'immagine in bianco e nero, per ridurre la dipendenza dell'analisi dal colore.



```
istogrammi = {'red': [], 'green': [], 'blue': [], 'gray': []}

img1 = cv2.cvtColor(campione, cv2.COLOR_BGR2GRAY)
istogrammi['gray'] = cv2.calcHist([img1], [0], None, [256], [0, 256])

img1 = cv2.cvtColor(campione, cv2.COLOR_BGR2RGB)

istogrammi['red'] = cv2.calcHist([img1], [0], None, [256], [0, 256])
istogrammi['green'] = cv2.calcHist([img1], [1], None, [256], [0, 256])
istogrammi['blue'] = cv2.calcHist([img1], [2], None, [256], [0, 256])
```

File: pixels.py

In questo primo frammento di codice avviene l'estrazione degli istogrammi di colore, attraverso la funzione `calcHist()` della libreria OpenCV, i quali vengono salvati all'interno di un dizionario per riuscire ad avere sempre a portata di mano i valori. Questo passaggio viene fatto sia sull'immagine campione che sulle altre immagini che compongono il nostro dataset.

```
percHistGray = (
    1+cv2.compareHist(istogrammi['gray'], nuovaImmagine["istogrammi"]['gray'], cv2.HISTCMP_CORREL))*50
percHistRed = (
    1+cv2.compareHist(istogrammi['red'], nuovaImmagine["istogrammi"]['red'], cv2.HISTCMP_CORREL))*50
percHistGreen = (
    1+cv2.compareHist(istogrammi['green'], nuovaImmagine["istogrammi"]['green'], cv2.HISTCMP_CORREL))*50
percHistBlue = (
    1+cv2.compareHist(istogrammi['blue'], nuovaImmagine["istogrammi"]['blue'], cv2.HISTCMP_CORREL))*50

percHist = (percHistGray+percHistRed +
            percHistGreen+percHistBlue)/4
```

File: pixels.py

Una volta estratti gli istogrammi confrontiamo i valori attraverso il metodo `compareHist()` sempre della libreria OpenCV, che abbiamo scelto per confrontare la correlazione fra gli istogrammi come riportato dalla formula in seguito.

Correlation

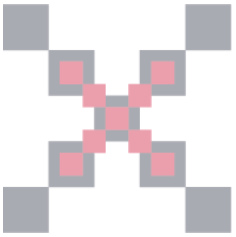
$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

where

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

and N is a total number of histogram bins.

Questa funzione ritorna un valore tra -1 e 1, dove 1 corrisponde alla massima correlazione possibile e -1 alla minima.



4.2.2 Algoritmo di Object Detection

Per quanto riguarda invece la parte di riconoscimento degli oggetti all'interno della nostra immagine abbiamo dovuto innanzitutto pre-configurare l'ambiente necessario all'algoritmo per lavorare. Per questa parte dello script abbiamo deciso di utilizzare YOLO (You Only Look Once), un algoritmo di object detection apprezzato per la sua velocità e precisione.

Per avere un'idea generale di come funziona YOLO partiamo col dire che il suo obiettivo principale è quello di identificare e posizionare, attraverso dei boxes, oggetti all'interno di un'immagine quasi in real time. Per fare ciò utilizza una rete neurale convoluzionale (CNN), che nel nostro caso è già passata per un processo di pre-training che ci permette di riconoscere correttamente 80 oggetti diversi. La lista degli oggetti è presente all'interno del file *coco.names*.
Il processo avviene sempre con l'ottica di trovare i bounding boxes, ossia le porzioni di immagine che contengono i vari oggetti. Non tutte queste porzioni arrivano alla fine del processo, infatti vengono fatti due processi a posteriori per ritornare solamente i boxes più significativi.
Il primo è chiamato "Non-Maximal Suppression" (NMS), dove vengono scartati i boxes in cui la probabilità con cui sono stati identificati oggetti è minore di una soglia data (generalmente 0.6).
Il secondo processo invece è "Intersection Over Union" (IOU), in questo caso fra i boxes che hanno valori di probabilità più alti di quello prefissato (di solito 0.4) vengono eliminati.
I bounding boxes rimanenti vengono infine salvati in una matrice multidimensionale, dove per ogni immagine vengono salvate informazioni sia riguardo all'oggetto identificato che al box che lo contiene.
Per capire meglio com'è stato implementato questo processo si guardi il programma *utils.py*.

```
resized_image = cv2.resize(img1, (m.width, m.height))

boxes = detect_objects(m, resized_image, iou_thresh, nms_thresh)

oggetti = {}

for i in range(len(boxes)):
    box = boxes[i]
    if len(box) >= 7 and class_names:
        cls_conf = box[5]
        cls_id = box[6]
        oggetti[cls_id] = cls_conf
```

File: pixels.py

Dopo la preconfigurazione per ogni immagine viene chiamata la funzione `detect_objects()`, presente nel file *utils.py*, che salva all'interno della matrice `boxes` varie informazioni riguardo agli oggetti identificati e alla porzione di

immagine che li contiene. Di queste informazioni salviamo solamente l'identificativo dell'oggetto e la percentuale di accuratezza con cui l'algoritmo è stato in grado di riconoscere l'oggetto.

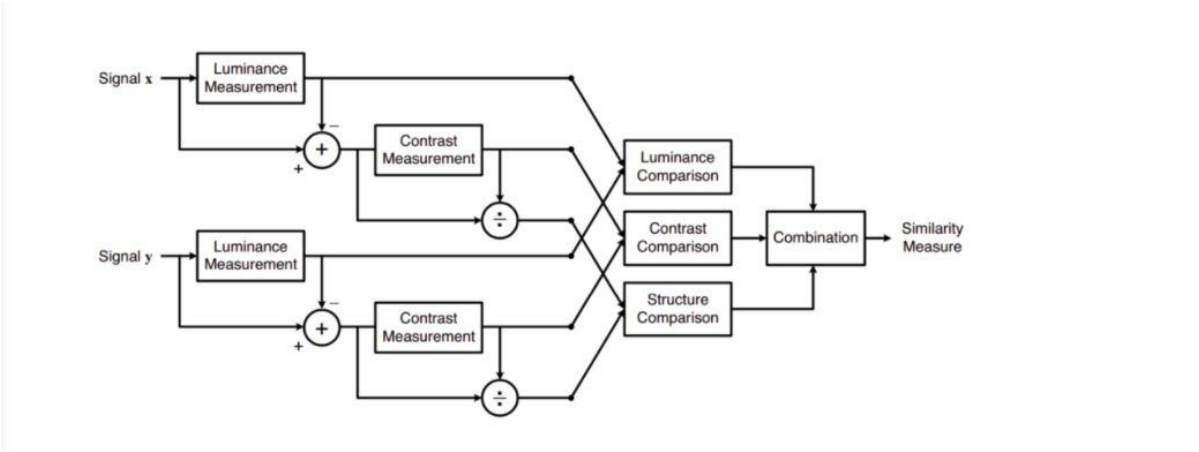
```
oggetti = item['oggetti']

tmp = 0
percOggetti = 0.0
for (oggKey, oggValue) in nuovaImmagine["oggetti"].items():
    tmp += 1
    for (key, value) in oggetti.items():
        if oggKey == key:
            percOggetti += (value.item() * oggValue.item())

percOggetti = (percOggetti/tmp)*100
if percOggetti > 100:
    percOggetti = 100
```

File: pixels.py

Passando infine al confronto fra gli oggetti identificati nelle due immagini, si ha un ciclo che ci permette di controllare se lo stesso oggetto è stato trovato in entrambe le immagini, se questo è vero si moltiplicano i valori di accuratezza di entrambe le immagini per una maggior precisione.

4.2.3 Structural Similarity Index Measure

L'ultimo elemento del confronto è il calcolo del SSIM. A differenza di altre tecniche di analisi delle immagini che si basano sulla quantificazione della differenza fra un campione e un'altra immagine, per esempio il Mean Squared Error.
SSIM prende spunto dal sistema di percezione visiva umana, che si concentra sulle differenze delle informazioni estratte dalle immagini in input. Questo indice tiene conto di tre principali elementi, la luminanza, il contrasto e la struttura, collegati come spiegato nell'immagine seguente.
Per un maggiore approfondimento sull'effettivo funzionamento di questo lasciamo i riferimenti nelle fonti.



Il risultato di questa comparison è sempre un valore compreso fra -1 e 1, dove si ottiene 1 quando le due immagini hanno una differenza minima o sono uguali mentre -1 quando la differenza è massima. In questa versione della funzione però si ottiene un risultato già normalizzato, i valori oscillano quindi fra 0 e 1.

```
img1ssim = cv2.resize(img, (1000, 1000))
img2ssim = cv2.resize(campione, (1000, 1000))
percSSIM = (ssim(img1ssim, img2ssim, multichannel=True))*100
```

File: pixels.py

La funzione `ssim()` era già definita all'interno della libreria *scikit-image*, per riuscire però ad applicarla alle nostre immagini abbiamo dovuto fare un resize di queste, poiché l'algoritmo opera solo su immagini di dimensioni uguali.

4.2.4 Interfacciamento con la web app

Infine illustriamo i principali elementi che permettono allo script in Python di interfacciarsi e lavorare correttamente con la web app.

```
with open('./images.json') as json_data:
    images = json.load(json_data)
```

File: pixels.py

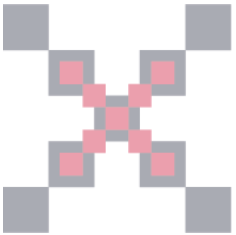
Grazie ad un ciclo while lo script viene eseguito in loop, in modo che all'aggiunta di altre immagini nella web app, il programma sarà in grado di estrarne le features e fare le operazioni di comparison senza comandi aggiuntivi o senza far ripartire il programma ogni volta.

Lo script Python è in grado di reperire le immagini attraverso un file JSON dove, per ognuna, sono salvati il percorso per raggiungerla e un identificatore.

```
request = requests.get(
    'http://localhost:3000/util/pyupdate/' + str(ID) + '/' + str(comparison))
```

File: pixels.py

Infine, una volta calcolata la percentuale di somiglianza fra un'immagine del dataset e l'immagine campione, viene fatta una richiesta HTTP di tipo GET per inviare al server Node.js il risultato della comparison che passerà direttamente all'interfaccia grafica.



4.3 Web App

Passando allo sviluppo del server web, abbiamo deciso di utilizzare un sistema runtime orientato agli eventi: Node.js. Come suo completamento, abbiamo implementato Express.js: un micro-framework per Node.js, progettato per creare web application; e API ed ormai definito il server framework standard de facto per Node.js.

Poiché Express parte dal basso livello di Node.js, mentre PHP o Python sono linguaggi di alto livello, Express ci permette comunque di restare ad un livello abbastanza basso ma di avere dei vantaggi, ad esempio nel generare percorsi (URL) per l'applicazione più facilmente o anche per importare le librerie.

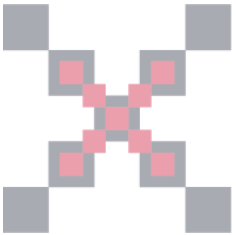
Successivamente, riportiamo alcuni frammenti di codice del server Node.js significativi per lo sviluppo del nostro progetto.

4.3.1 Popolamento file JSON

```
19  function populateDB(callback) {
20      var imagesDB = [];
21      var i = 1;
22      fs.readdir(imagesFolder, function (err, files) {
23          //handling error
24          if (err) {
25              return console.log('Unable to scan directory: ' + err);
26          }
27          //listing all files using forEach
28          files.forEach(function (file) {
29              var image = {}
30              image.ID = i++;
31              image.path = "img/" + file
32              imagesDB.push(image);
33          });
34
35          fs.writeFileSync('images.json', JSON.stringify(imagesDB), 'utf-8');
36          callback();
37      });
38  }
```

File: index.js

In questa porzione di codice è presente la funzione che, analizzando tutti i file nella cartella `img` ,crea un oggetto Javascript contenente i tag ID (numero incrementale) e il path della foto. Questo oggetto verrà poi salvato in un file JSON, utilizzato da Python per accedere alle immagini e modificato dal server Node ogni qual volta ci sarà un aggiornamento (come una nuova foto analizzata, oppure modifica della foto campione).



4.3.2 Aggiornamento percentuali mediante richiesta GET

```
app.get('/util/pyupdate/:ID/:PERC', function (req, res) {
    res.send("Ok");
    imagesParsed.forEach(element => {
        if (element.ID == req.params.ID) {
            element.perc = req.params.PERC;
            fs.writeFileSync('images.json', JSON.stringify(imagesParsed), 'utf-8');
            return;
        }
    });

    var update = {
        'ID': req.params.ID,
        'perc': req.params.PERC
    }

    io.emit('imageUpdate', JSON.stringify(update));

})
```

File: index.js

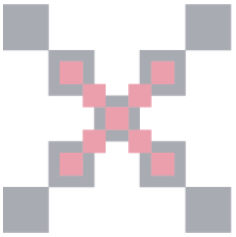
In seguito all'arrivo della richiesta GET da parte di Python per l'aggiornamento delle percentuali, il server Node.js cerca l'ID fornito dalla richiesta tra tutte le immagini presenti nell'archivio. Una volta trovato l'ID corretto, viene aggiunto all'oggetto il parametro della percentuale di somiglianza tra la foto corrispondente e il campione.
Completato questo, il file JSON viene aggiornato, in modo da consentire a Python di riconoscere le immagini già analizzate.
Infine, viene generato un nuovo oggetto `update` da inviare al client mediante WebSocket, per consentirgli di aggiornare la galleria con la nuova percentuale trovata.

4.3.3 Attivazione del WebSocket lato server

```
var io = require('socket.io')(serverSocket, {
    cors: {
        origin: "http://localhost:3000",
        methods: ["GET", "POST"]
    }
});
```

File: index.js

Si è deciso di creare un WebSocket perchè ci permette di aggiornare dinamicamente le percentuali della galleria in tempo reale, senza ricaricare la pagina.
Con il protocollo WebSocket, è sufficiente che il client apra la connessione a un server web tramite un handshake. Dopo l'handshake, il canale di comunicazione rimane sempre aperto. Il server quando necessario si attiva da solo e fornisce al client tutte le informazioni senza attendere le richieste. Se ci sono nuove



informazioni lato server, le comunica al client senza dover emettere una richiesta extra lato client.

4.3.4 Aggiornamento pesi algoritmi

```
62      app.get('/util/weightsUpdate/:hist/:yolo/:ssim', function (req, res) {
63          var newHist = req.params.hist;
64          var newYolo = req.params.yolo;
65          var newSsim = req.params.ssim;
66
67          var newWeights = {};
68          newWeights.hist = newHist;
69          newWeights.yolo = newYolo;
70          newWeights.ssim = newSsim;
71          fs.writeFileSync('algWeights.json', JSON.stringify(newWeights), 'utf-8');
72          imagesParsed.forEach(element => {
73              delete element["perc"]
74          });
75          fs.writeFileSync('images.json', JSON.stringify(imagesParsed), 'utf-8');
76          res.send("Ok");
77      })
```

File: index.js

In questo frammento di codice il server riceve grazie ad una richiesta GET l'aggiornamento dei pesi e modifica il file JSON nel quale sono salvati.
L'utente attraverso uno slider nella sezione "Settings" può decidere il peso che ogni algoritmo di comparison ha nel risultato finale, successivamente il browser manda un evento tramite una richiesta GET al server, passando i nuovi valori, il server web Node aggiorna il file JSON dei pesi e resetta le percentuali delle foto, infine Python rilegge i valori e confronta nuovamente le diverse immagini.

4.3.5 Generazione singolo elemento

```
1  <div class="grid-item" data-perc="<%if(photo['ID'] == 0) { %>110<% } else if (photo['perc'] != null){%><%=photo.perc%><% } %>">
2      <figure class="effect-sadie">
3          <img src="<%=photo.path%>" data-id="<%=photo.ID%>" alt="Image" class="img-fluid tm-img">
4          <figcaption>
5              <h2 class="tm-figure-title"> <span><strong></strong></span></h2>
6              <p class="tm-figure-description">
7                  <%if(photo["ID"] == 0) { %>
8                      Campione
9                  <% } else if (photo["perc"] != null){%>
10                     <%=photo.perc%>%
11                 <% } %>
12             </p>
13             <a href="<%=photo.path%>">View more</a>
14         </figcaption>
15     </figure>
16 </div>
```

File: views/partials/singlePhoto.ejs

L'immagine riportata sopra mostra il codice implementato per la generazione di un singolo elemento nella griglia delle foto nella web app. Per consentire la generazione del blocco, vengono passati:

- Parametro ID (per riconoscere univocamente le foto)
- Path (per accedere e mostrare la foto)

- Percentuale (per l'ordinamento e per mostrare sopra ogni foto il risultato ottenuto)

Come si può vedere, i file `.ejs` consentono di inserire dati dinamici elaborati dal server e effettuare cicli o blocchi condizionali insieme a classici tag HTML. Ad esempio, per inserire il path della foto si utilizza la direttiva `<%= photo.path %>`

4.3.6 Griglia immagini

```
<div class="tm-img-gallery gallery-one grid">
    <%- include('../partials/singlePhoto', {"photo": {
        "ID" : 0,
        "path" : "campione.jpg"
    }}); %>
    <% for(var i=0; i<imagesParsed.length; i++) {%>
        <%- include('../partials/singlePhoto', {"photo": imagesParsed[i]}); %>
            <% } %>


</div>
```

File: views/pages/index.ejs

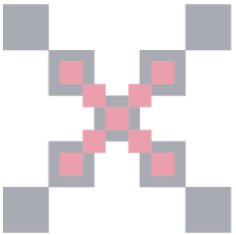
Questo frammento di codice mostra l'inclusione della foto campione inserita dall'utente e un ciclo for sugli elementi dell'oggetto JSON `imagesParsed` implementato per generare automaticamente la griglia delle immagini, che apparirà successivamente nella pagina principale della web app.

4.3.7 Attivazione WebSocket lato client

```
const socket = io("ws://localhost:3001");

socket.on('imageUpdate', (data) => {
    console.log("Message received from socket");
    var update = JSON.parse(data);
    $('.tm-img-gallery .grid-item img[data-id="' + update.ID + '"]').parent().find('.tm-figure-description').text(update.perc + "%");
    $('.tm-img-gallery .grid-item img[data-id="' + update.ID + '"]').parent().parent().attr("data-perc", update.perc)

    $('.tm-img-gallery').find('.grid-item').sort(function(a, b){
        return b.dataset.perc - a.dataset.perc;
    }).appendTo($('.tm-img-gallery'))
});
```

File: views/pages/index.ejs

Si passa quindi all'attivazione del WebSocket dal lato del client, grazie alla libreria `socket.io`, in modo da permettere di ricevere i messaggi di aggiornamento dal server. Quando viene ricevuto un aggiornamento, la nuova percentuale viene inserita nel blocco HTML contenente l'immagine corrispondente (riconosciuta grazie all'ID). Dopodiché, la galleria viene riordinata in base alle percentuali, per consentire una visualizzazione dalle immagini più simili al campione alle meno simili.



5 Conclusioni

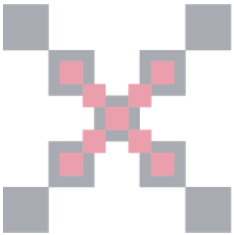
Confrontando gli obiettivi che ci eravamo posti e la web app creata, si può dire che siamo soddisfatti dei risultati ottenuti.

Sebbene ci siano state diverse difficoltà causate dalla situazione COVID-19, siamo comunque riusciti a lavorare in modo armonioso ed efficiente tra di noi.

Le video chiamate hanno permesso di confrontarci e lavorare insieme, anche se abbiamo avuto dei rallentamenti causati dal non poter essere fisicamente nella stessa stanza. Il lavoro infatti è stato suddiviso in modo da essere meno limitati possibile dalla situazione.
Abbiamo lavorato insieme per la parte di definizione dei requisiti, per poi dividerci in due coppie: una che si occupasse della parte di sviluppo degli algoritmi di comparison (Alessandro Ferronato e Martina Peruzzo) e una coppia che si occupasse dello scheletro e del design della web app (Giorgia Giro e Davide Parpinello). Una volta completata la parte di sviluppo da parte delle due coppie, ci siamo trovati tutti e quattro in videochiamata, prima per interfacciare l'algoritmo con la web app, e poi per fare test, aggiustamenti e valutare il risultato finale ottenuto.

Un elemento fondamentale che ha aiutato la nostra collaborazione è stato il fatto che fossimo tutti "sulla stessa linea d'onda". Quindi riuscivamo a lavorare fluidamente e senza troppe interruzioni dovute a contrasti di idee.

In conclusione, pensiamo tutti che sia stata una bella esperienza che ci ha permesso di mettere in pratica le conoscenze nostre conoscenze teoriche; cosa che riteniamo molto importante per la nostra crescita e formazione professionale.



6 Fonti e librerie

- https://github.com/gilbitron/flexmasonry
- https://www.pyimagesearch.com/
- https://github.com/Garima13a/YOLO-Object-Detection
- https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/
- https://unsplash.com/
- https://codepen.io/yamini/pen/eGbpqR
- https://pqina.nl/filepond/
- https://colorhunt.co/
- https://garimanishad.medium.com/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2
- https://medium.com/srm-mic/all-about-structural-similarity-index-ssim-theory-code-in-pytorch-6551b455541e