UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in "Ingegneria informatica e dell'automazione"

REST API per l'analisi di immagini

REST API for image analysis

Candidato:
Lorenzo PICCIONI

Relatore:
Prof. Emanuele FRONTONI

Anno Accademico 2020-2021

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in "Ingegneria informatica e dell'automazione"
Via Brecce Bianche – 60131 Ancona (AN), Italy

To

Sommario

Questo progetto nasce dalla presa di coscienza di quanto al giorno d'oggi i Social Media siano importanti nel modo dell'economia, in particolare nell'ambito del fashion marketing.
Proprio perché sono ritenuti indispensabili per far crescere un'azienda che lavora nel settore della moda si è pensato di creare un Web Service RESTful che permettesse alle aziende di analizzare delle immagini prese dai Social Network (in particolare Instagram) con lo scopo di prevedere quali saranno i futuri trend.
Nell'elaborato ci sarà quindi una spiegazione del perché è nata questa idea e del contesto da cui è scaturita; poi seguirà una breve spiegazione di come vada progettato un Web Service REST e di quali sono i tool necessari per semplificare il lavoro.
Infine sarà mostrata l'implementazione vera e propria delle API e saranno discussi i vari risultati ottenuti.

Indice

1. Introduzione **1**

1.1. Architettura REST 1
1.2. Contesto 2
1.3. Obiettivi e principali contributi 3
1.4. Struttura della tesi 4

2. Progettazione di un servizio REST **5**

2.1. Identificazione delle risorse 5
2.2. Metodi HTTP 6
2.3. Risorse autodescrittive 8
2.3.1. JSON 8
2.4. Collegamenti tra risorse 9
2.5. Comunicazione senza stato 10
2.6. Progettazione di un servizio REST 10
2.6.1. Client HTTP 11
2.6.2. Insieme delle URI 11
2.6.3. Interpretare il formato della rappresentazione 11
2.7. Test e Debugging 11
2.8. Gestione della sicurezza 12
2.9. Framework per servizi RESTful 13

3. Materiali e Metodi **14**

3.1. Principi generali di progettazione 14
3.2. Rete neurale e deep learning 15
3.3. Implementazione 15
3.3.1. API per in conteggio delle immagini con più di un oggetto di classi diverse 16
3.3.2. API per il riconoscimento dei colori dominanti di un'immagine 18
3.3.3. API per il riconoscimento degli oggetti con il modello YOLOv5 20
3.4. Soluzioni adottate 23
3.4.1. Python 23
3.4.2. Librerie Python 25
3.4.3. MongoDb 27
3.4.4. Postman 29
3.4.5. Machine learning e reti neurali 30

4. Risultati raggiunti e Discussione **33**
4.1. Conteggio delle immagini con più di un oggetto di classi diverse . . . 33
4.2. Riconoscimento dei colori dominanti di un'immagine 34
4.2.1. Possibili errori 34
4.3. Riconoscimento degli oggetti con il modello YOLOv5 36
4.3.1. Possibili errori 36

5. Conclusioni e Lavori Futuri **39**
5.1. Conclusioni 39
5.2. Lavori Futuri 39

A. Appendice **41**
A.1. API per il conteggio delle immagini con più di un oggetto di classi diverese 41
A.2. API per il riconoscimento dei colori dominanti di un'immagine . . . 42
A.3. API per il riconoscimento degli oggetti con il modello YOLOv5 . . . 43

Elenco delle figure

1.1. Tempo speso quotidianamente sui social 3
1.2. Utilizzo dei social per la ricerca di brand 3

2.1. Sintassi generica di un URI 5
2.2. Esempio di un URI 6
2.3. Simbolo JSON 8
2.4. Esempio JSON 9

3.1. Diagramma di flusso dell'API per il conteggio delle immagini con più di un oggetto di classi diverse 17
3.2. Diagramma di flusso dell'API per il riconoscimento dei colori dominanti di un'immagine 19
3.3. Diagramma di flusso dell'API per il riconoscimento degli oggetti con il modello YOLOv5 21
3.4. Logo Python 23
3.5. Logo Anaconda 24
3.6. Esempio di routing con Flask 25
3.7. Logo OpenCV 26
3.8. Logo MongoDB 27
3.9. Logo Postman 29
3.10. Barra iniziale Postman 30
3.11. Esempio di analisi con il modello YOLOv5s 32
3.12. Esempio di JSON dopo l'analisi con il modello YOLOv5s 32

4.1. Richiesta dell'API per il conteggio delle immagini con più di un oggetto di classi diverse 33
4.2. Risposta dell'API per il conteggio delle immagini con più di un oggetto di classi diverse 33
4.3. Richiesta dell'API per il riconoscimento dei colori dominanti di un'immagine 34
4.4. Risposta dell'API per il riconoscimento dei colori dominanti di un'immagine 34
4.5. Richiesta senza parametro dell'API per il riconoscimento dei colori dominanti di un'immagine 34
4.6. Risposta senza parametro dell'API per il riconoscimento dei colori dominanti di un'immagine 35

4.7. Richiesta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine . 35
4.8. Risposta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine . 35
4.9. Richiesta dell'API per il riconoscimento di oggetti con il modello YOLOv5 36
4.10. Rispsota dell'API per il riconoscimento di oggetti con il modello YOLOv5 36
4.11. Immagine che mostra gli oggetti identificati mediante il modello YOLOv5 37
4.12. Richiesta senza parametro dell'API per il riconoscimento degli oggetti con il modello YOLOv5 37
4.13. Risposta senza parametro dell'API per il riconoscimento degli oggetti con il modello YOLOv5 38
4.14. Richiesta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine . 38
4.15. Risposta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine . 38

A.1. Codice dell'API per il conteggio delle immagini con più di un oggetto di classi diverse 41
A.2. Codice dell'API per il riconoscimento dei colori dominanti di un'immagine 42
A.3. Codice dell'API per il riconoscimento degli oggetti presenti in un'immagine con il modello YOLOv5 43

Elenco delle tabelle

2.1. Mappatura uno a uno operazioni CRUD e metodi HTTP 6
2.2. Codici di stato HTTP e relativo significato 7
2.3. Codici di stato HTTP per errori lato Server 8

3.1. Implementazione API per il conteggio delle immagini con più di un oggetto di classi diverse 16
3.2. Implementazione API per il riconoscimento dei colori dominanti di un'immagine 18
3.3. Implementazione API per il riconoscimento degli oggetti con il modello YOLOv5 22
3.4. Tabella riassuntiva delle differenze tra formato JSON e BSON 28

Capitolo 1.

Introduzione

API è l'acronimo di "Application Programming Interface", un'API è un'interfaccia che consente un dialogo tra parti diverse di uno stesso software o tra parti di programmi diversi.
Esse nascono con lo scopo di permettere ad uno sviluppatore di usare lo stesso codice in contesti diversi; con le API il programmatore può evitare di riscrivere ogni volta tutte le funzioni necessarie al programma dal nulla, si può quindi dire che rientrano nel più vasto concetto del riutilizzo del codice.
Grazie alle API un servizio o un prodotto possono comunicare con altri servizi o prodotti senza sapere come questi vengono implementati, semplificandone lo sviluppo con un notevole risparmio di tempo e denaro. Esse si possono considerare delle vere e proprie "scatole nere" di cui il programmatore non deve conoscere il funzionamento ad un livello più basso ma solamente il compito generale, ciò permette di riprogettare o migliorare le funzioni all'interno dell'API senza dover cambiare il codice che si affida ad essa.
Tutte le API che utilizzano un protocollo HTTP per le richieste e per le risposte ottenute sono considerate dei "Web Service". Un Web Service non è altro che un'interazione richiesta-risposta tra un Client e un Server.
In particolare il Client richiede una risorsa e l'API Server invia una risposta utilizzando qualsiasi linguaggio di programmazione; il linguaggio utilizzato infatti non è importante in quanto la richiesta e la risposta sono effettuate tramite dei protocolli HTTP.
Tutto ciò rende i Web Service indipendenti dal linguaggio di programmazione e interoperabili tra diverse piattaforme e sistemi; ecco spiegata la grande utilità di questi servizi, non è importante con quale linguaggio di programmazione si stia lavorando, un Web Service sarà utilizzabile in qualsiasi caso.
I Web Service includono più tipi di API, tra cui le REST API.

1.1. Architettura REST

REST sta per "Respresentational State Transfer", indica la rappresentazione del trasferimento di stato di un dato, questo perché quello REST non è un protocollo

ma bensì uno stile architetturale per la progettazione di Web Service.
I principi che si trovano alla base dell'architettura REST sono i seguenti:

- **Client/Server**
  Il Server offre una o più funzionalità e ascolta le richieste di possibili Client.
- **Stateless**
  La comunicazione tra Client e Server deve essere senza stato tra le richieste.
- **Caching**
  C'è la possibilità di memorizzare nella cache una risposta per il suo riutilizzo nelle richieste successive.
- **Interfaccia uniforme**
  Identificazione delle risorse, manipolazione delle risorse attraverso rappresentazioni e risorse autodescritte.
- **Stratificazione**
  Sistema a livelli in cui gli intermediari possono intercettare il traffico Client-Server per scopi specifici.

1.2. Contesto

L'industria della moda è una delle più dinamiche, è in continuo cambiamento, e viene influenzata molto dalla comunicazione, di conseguenza non poteva certo rimanere fuori dalla "Digital Transformation".
Ormai per un'azienda avere una presenza online porta ad un vero e proprio vantaggio competitivo e i social media sono lo strumento di comunicazione che può rivelarsi fondamentale a questo scopo.
Una buona strategia di "Social Media Marketing" infatti permette di coinvolgere i clienti nella creazione di nuovi contenuti e genera in loro la volontà di essere sempre aggiornati sui prodotti che acquistano.
Una prova di tutto ciò la possiamo avere andando ad analizzare il Digital Report 2021 globale pubblicato da We Are Social, esso ci mostra quanto le persone siano sempre più connesse a internet e connesse tra loro grazie ai social.

Come si può vedere nell'immagine 1.1 in media una persona spende quasi 7 ore al giorno su internet di cui ben 2 ore e 25 minuti sui social, un tempo davvero grande. Ma il dato che ancor più fa capire quanto i social siano importanti nel mondo del fashion è il prossimo.

Dall'immagine 1.2 infatti si può notare che il 44.8% degli utenti utilizza i Social Media per ricercare informazioni riguardanti un brand, è quindi ovvio che Instagram con il suo bacino di 1.221 miliardi di profili attivi riveste un ruolo importantissimo a livello di marketing. Per un'azienda che lavora nel mondo della moda è ormai impensabile non sfruttare i social media, grazie ad essi ora il cliente può esprimere le

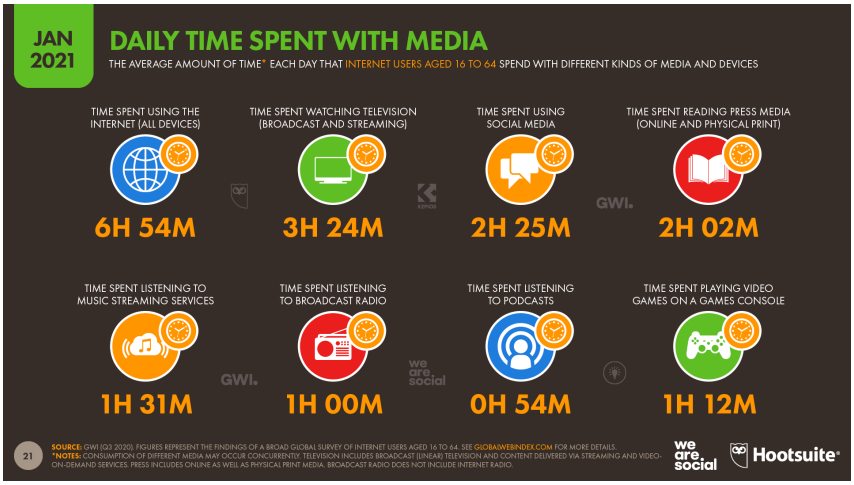

Figura 1.1.: Tempo speso quotidianamente sui social

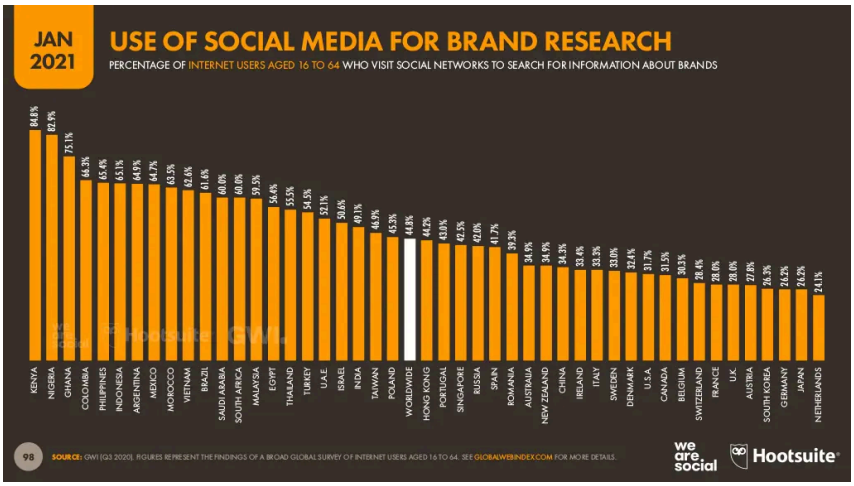

Figura 1.2.: Utilizzo dei social per la ricerca di brand

sue opinioni e partecipare attivamente alla creazione dei nuovi prodotti.
Un'altra figura fondamentale conseguente a questa digital transformation è quella dell'ambassador o dell'influencer, figure che hanno il compito di "influenzare" i propri followers e di guidarli nel processo di acquisto.
Sotto questo aspetto Instagram si è rivelato essere il social più adatto, quello su cui puntare se si vuole condurre una buona strategia di fashion social marketing. Ecco quindi che un Web Service che analizza le immagini prese da Instagram con l'idea di capire quali saranno i futuri trend potrebbe essere uno strumento fondamentale per un'impresa che lavora nel mondo del fashion.

1.3. Obiettivi e principali contributi

L'obiettivo di questo progetto era quello di creare delle API REST per l'analisi di immagini. Ognuna di queste immagini è associata anche ad un file di testo, con lo

stesso nome dell'immagine, contenente i dati che la riguardano.
I dati presenti in questi file di testo sono stati caricati in un database di tipo NoSQL in modo da renderli utilizzabili all'interno del progetto. Questo tipo di database non relazionale permette di archiviare enormi quantità di dati non strutturati dando loro molta flessibilità; i database NoSQL adottano un modello di dati semplice e sono facili da gestire, in poche parole non sono altro che una semplificazione dei database di tipo relazionale.
L'analisi di queste immagini tramite metodi di deep learning (e dei file di testo ad esse collegati) è stata svolta con lo scopo di capire quali saranno le nuove tendenze nel campo della moda e del design basandosi sui contenuti dei Social Media, in particolare Instagram.

1.4. Struttura della tesi

Ecco di cosa tratterà ogni capitolo:

- Nel **Capitolo 2** verrà spiegata la progettazione di un servizio REST.
- Nel **Capitolo 3** verrà analizzata nel dettaglio la progettazione e l'implementazione del servizio REST specifico per il progetto e verrano mostrati tutti i tool impiegati in fase di creazione.
- Nel **Capitolo 4** si vedranno in particolare le singole API e i risultati ottenuti da ognuna di esse.
- Nel **Capitolo 5** ci saranno le conclusioni tratte da questo progetto e si discuterà di eventuali implementazioni future.
- Nell'**Appendice** sarà mostrato il codice vero e proprio delle diverse API.

Capitolo 2.

Progettazione di un servizio REST

Quando si vuole progettare un web service RESTful è bene seguire le seguenti linee guida in modo da rispettare quelli che sono i principi REST.

- Identificazione delle risorse
- Utilizzo esplicito dei metodi HTTP
- Risorse autodescrittive
- Collegamenti tra risorse
- Comunicazione senza stato

2.1. Identificazione delle risorse

I Web Service RESTful si basano interamente sulle risorse, con questo termine si indica un qualsiasi elemento che sarà oggetto di elaborazione, viene da sé che ciascuna risorsa deve essere identificata univocamente.
Ecco quindi che entrano in gioco gli URI (Uniform Resource Identifier), un URI è una sequenza di caratteri che identifica universalmente ed univocamente una risorsa.

Figura 2.1.: Sintassi generica di un URI

Come si può vedere nell'immagine 2.1 la sintassi generica di un URI può essere riassunta in quattro parti :

- *<scheme>*
  Rappresenta il servizio, solitamente fa riferimento al nome di un protocollo applicativo.
- *<authority>*
  A sua volta è strutturata nella forma *<userinfo>@<host>:<port>* (non tutte le parti sono sempre presenti) dove :

  - <*userinfo*> rappresenta i dati di accreditamento di un utente.
  - <*host*> è un indirizzo IP oppure un nome di dominio (DNS).
  - <*port*> è un numero di porta, quella usata per contattare il server.

- <*path*>
  È un percorso gerarchico, con i vari livelli separati da uno slash (' / ').

- <*query*>
  È una successione di dichiarazioni di parametri valorizzati, separate dall'ampersand ' & ', ciascuna nella forma <name> = <value>.

Di queste parti solo due sono obbligatorie, cioè <*scheme*> e <*path*>.

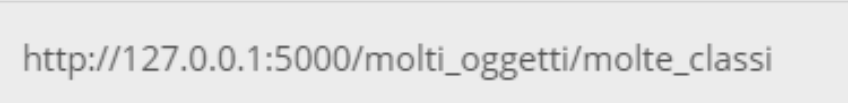

Figura 2.2.: Esempio di un URI

Nell'immagine 2.2 c'è un esempio di quello che è un URI utilizzato nel Web Service per l'analisi di immagini che è stato progettato.

2.2. Metodi HTTP

Con "Metodi HTTP" si intende lo sfruttamento dei metodi (o verbi) predefiniti del protocollo HTTP allo scopo di indicare quale operazioni effettuare su di una risorsa dopo averla individuata.
In poche parole si stabilisce una mappatura uno a uno tra le operazioni CRUD (creazione, lettura, aggiornamento ed eliminazione di una risorsa) e i metodi HTTP.

Metodo HTTP	**Operazione CRUD**	**Descrizione**
POST	Create	Crea nuova risorsa.
GET	Read	Ottiene una risorsa esistente.
PUT PATCH	Update	Aggiorna una risorsa o ne modifica lo stato. Aggiorna parzialmente una risorsa.
DELETE	Delete	Elimina una risorsa.

Tabella 2.1.: Mappatura uno a uno operazioni CRUD e metodi HTTP

- POST = Creazione:
  Crea una nuova risorsa nell'URI specificato. Il corpo del messaggio di richiesta fornisce i dettagli della nuova risorsa che si intende creare.

- GET = Lettura:
  Recupera una rappresentazione della risorsa nell'URI specificato. Il corpo del messaggio di risposta contiene i dettagli della risorsa richiesta.
- PUT = Aggiornamento:
  Modifica la risorsa nell'URI specificato. Il corpo del messaggio di richiesta specifica la risorsa da aggiornare. È utile se si vuole aggiornare una risorsa nella sua interezza. Se la risorsa non esiste allora viene creata.
- PATCH = Aggiornamento parziale:
  Esegue un aggiornamento parziale della risorsa nell'URI specificato. Il corpo della richiesta specifica i campi della risorsa che devono essere modificati.
- DELETE = Eliminazione:
  Rimuove la risorsa nell'URI specificato.

Codice	**Significato**	**Metodo HTTP**
Successo dell'operazione		
200	'OK' È il codice che viene consegnato quando una chiamata si comporta esattamente come ci si aspettava.	GET, POST, PUT
201	'Created' La nuova risorse è stata creata.	POST, PUT.
204	'No content' Il processo è stato eseguito correttamente ma non viene restituito nessun contenuto.	POST, PUT, PATCH, DELETE
Errori lato Client		
400	'Bad request' Il Server non può restituire una risposta a causa di un errore del Client.	POST, PATCH
404	'Not found' La risorsa richiesta non esiste.	GET, DELETE
409	'Conflict' La richiesta è corretta ma non è possibile elaborarla perché c'è un conflitto con lo stato attuale della risorsa.	PUT, PATCH
415	'Unsupported media type' L'enttià della richiesta è di un tipo non accettato dal Server o dalla risorsa richiesta.	PATCH

Tabella 2.2.: Codici di stato HTTP e relativo significato

Ogni volta che viene effettuata una richiesta tramite il protocollo HTTP al Server, il Client riceverà insieme alla risposta un codice di stato HTTP che indicherà se l'operazione è andata a buon fine o se ci sono stati dei problemi.
Nella tabella 2.2 sono presenti i principali codici HTTP in caso di operazione andata a buon fine o di un errore da parte del Client, il loro significato e i possibili metodi HTTP che possono generare quel codice di stato.

Codice	**Significato**
Errori lato Server	
500	'Internal Server Error' Errore generico interno al Server, al richiesta è stata formulata correttamente ma qualcosa è andato storto.
503	'Service Unavaible' Il Server non è al momento disponibile, può essere in sovraccarico oppure offline.

Tabella 2.3.: Codici di stato HTTP per errori lato Server

La tabella 2.3 rappresenta invece i principali codici HTTP in caso di errori da parte del Server e il relativo significato.

2.3. Risorse autodescrittive

Un Web Service invia al Client una rappresentazione delle risorse e non direttamente i dati presenti nel suo database, ecco quindi che assume una certa rilevanza il tipo di formato con cui si intende rappresentare una risorsa.
Il tipo di rappresentazione inviata dal Web Service al Client è indicato nella risposta HTTP tramite un MIME type.
Il modello architetturale REST non pone nessun vincolo sul formato utilizzato per lo scambio dei dati, quindi se ne potrebbe usare uno qualunque, ma di fatto è opportuno utilizzare un formato il più possibile standardardizzato e condiviso in modo da facilitare l'interazione tra Client e Web Service. Un esempio di formato utilizzato dalla maggior parte delle API è il formato JSON.

2.3.1. JSON

Figura 2.3.: Simbolo JSON

JSON (JavaScript Object Notation) è un formato adatto per lo scambio di dati fra applicazioni Client/Server. Per le persone è facile da leggere e scrivere e per le

macchine è facile generarlo e analizzarne la sintassi.
Come si può intuire dal nome si basa sul linguaggio *JavaScript*, ma è completamente indipendente dal linguaggio di programmazione, ed è proprio questo a renderlo ideale per lo scambio di dati. JSON prende origine dalla sintassi degli oggetti letterali in *JavaScript*, quindi è definito da delle coppie chiave/valore.

```
{
    "name": "Mario",
    "surname": "Rossi",
    "active": true,
    "favoriteNumber": 42,
    "birthday": {
        "day": 1,
        "month": 1,
        "year": 2000
    },
    "languages": [ "it", "en" ]
}
```

Figura 2.4.: Esempio JSON

Come si può vedere dalla figura 2.4 l'intero oggetto è racchiuso tra due parentesi graffe, ogni chiave è seguita da due punti, mentre le varie coppie chiave/valore sono separate tra di loro da una virgola.
JSON ammette i seguenti valori:

- Booleani (true e false)
- Numeri (interi, numeri in virgola mobile)
- Stringhe (racchiuse tra virgolette)
- Array
- Oggetti letterali
- Valore nullo (null)
- Liste di valori dello stesso tipo

Queste strutture possono essere annidate.

2.4. Collegamenti tra risorse

Le risorse tra loro sono messe in relazione tramite dei link ipertestuali; questo principio è noto anche come Hypermedia As The Engine Of Application State

(HATEOAS) .
In poche parole ogni richiesta HTTP GET deve restituire al Client le informazioni necessarie a trovare le risorse correlate direttamente all'oggetto richiesto tramite collegamenti ipertestuali inclusi nella risposta e deve anche contenere informazioni che descrivono le operazioni disponibili in ciascuna di queste risorse.
Il Client ha la possibilità di accedere alle risorse correlate semplicemente seguento i collegamenti presenti nella rappresentazione della risorsa corrente.
Questo fatto di utilizzare un URI per indentificare una risorsa permette al Client di accedere anche a risorse messe a disposizione da altre applicazioni.

2.5. Comunicazione senza stato

Il principio di comunicazione stateless è una delle caratteristiche principali del protocollo HTTP, con il termine stateless si va a indicare il fatto che ciascuna richiesta non ha alcuna relazione con le altre richieste, che siano esse precedenti o successive. Apllicare questo principio ad un Web Service RESTful significa avere delle interazioni tra Client e Server senza stato.

Una richiesta non deve mai richiedere al Server di recuperare un contesto o uno stato dell'applicazione, ma nelle intestazioni o nel corpo HTTP devono essere inclusi i parametri, il contesto e tutti i dati necessari al Server per generare una risposta.
Così facendo si ha una certa scalabilità, infatti l'assenza di stato offre la possibilità di scalare l'applicazione su più Server distribuendo il carico di lavoro, mentre mantenere lo stato di una sessione ha un costo in termini di risorse sul Server e quindi l'aumento del numero di Client potrebbe rendere questo costo insostenibile.
Ecco quindi che una comunicazione stateless va a migliorare le prestazioni globali di un Web Service e semplifica la progettazione e l'implementazione dei componenti lato Server.
Tutto ciò comunque non significa che un'applicazione RESTful non abbia uno stato; qualora sia necessario gestire uno stato della comunicazione infatti è compito del Client occuparsene (e non più del Server).

2.6. Progettazione di un servizio REST

Come detto in precedenza il modello architetturale REST lascia una grande libertà sulle tecnologie da utilizzare per lo sviluppo di un Web Service, e si limita a dare dei principi e delle linee guida da seguire.
Gli elementi principali di cui c'è bisogno per creare un servizio REST sono 3:

- Un Client HTTP
- Gli URI delle risorse a cui si vuole accedere

- La capacità di interpretare il formato della rappresentazione delle risorse

2.6.1. Client HTTP

Come Client HTTP si può utilizzare un semplice browser Web, ma questa è una soluzione un po' rudimentale, quindi si tende a sfruttare il Client che lo specifico linguaggio/ambiente di programmazione, che si è scelto di utilizzare, mettono a disposizione per gestire il protocollo HTTP.

2.6.2. Insieme delle URI

La seconda cosa di cui si ha bisogno è l'insieme delle URI che individuano tutte le risorse che il Servizio REST andrà a gestire.
Fatto ciò bisogna scrivere del codice che permetta di sfruttare i vari metodi/verbi HTTP, come ad esempio il codice per ottenere una risorsa corrispondente ad un determinato URI quando viene specificato il metodo GET.

2.6.3. Interpretare il formato della rappresentazione

Dopo aver stabilito il Client HTTP da utilizzare e le URI che individuano tutte le risorse è necessario scegliere con quale formato verranno rappresentate tutte le risorse richieste. Infatti sapendo come è rappresentata la risorsa sarà possibile scrivere un codice coerente con quanto viene restituito in seguito ad una richiesta al Server.

2.7. Test e Debugging

Una volta creato un Web Service è necessario testarlo per verificare che il comportamento sia effettivamente quello voluto. Gli eventuali malfunzionamenti potrebbero essere dovuti a diversi motivi; è quindi essenziale analizzare le richieste e le risposte scambiate tra Client e Web Service per comprenderne la causa.
Per testare una richiesta di tipo GET sarebbe sufficiente anche solamente un Web Browser, ma già per richieste di tipo POST o addirittura PUT e DELETE la cosa diverrebbe molto più complessa.
Ecco quindi che accorrono in aiuto i numerosi strumenti per l'esecuzione e l'analisi delle richieste HTTP che è possibile trovare in rete, tramite questi tool diventa molto più semplice effettuare il testing del Web Service.
Nel caso specifico del Web Service per l'analisi di immagini che verrà sviluppato si è scelto di utilizzare Postman, se ne parlerà meglio nella Sottosezione 3.4.4.

Indipendentemente dallo strumento utilizzato quando si analizzano i messaggi scambiati tra Client e Server bisogna controllare se:

- **Il Client riesce ad inviare la richiesta**
  Sembra scontato ma potrebbe verificarsi che il Client non riesca a comunicare

con il Web Sevice (URI errato, mancanza di rete, firewall che impedisce la comunicazione, ecc...).

- **La richiesta inviata è conforme alle specifiche del Web Service**
  Bisogna controllare se la richiesta HTTP è formulata correttamente, analizzandone sia l'eventuale contenuto del body che gli headers HTTP.
- **Il Client riesce a ricevere la risposta dal Web Service**
  Come per il caso dell'invio della richiesta anche per la ricezione potrebbero verificarsi problemi legati alla comunicazione tra Client e Server (Firewall che impedisce la comunicazione, ecc...).
- **La risposta ricevuta è conforme alle specifiche**
  Come nel caso della richiesta anche la risposta deve essere conforme alle specifiche richieste (è consigliabile controllare le intestazioni HTTP).

2.8. Gestione della sicurezza

Un aspetto fondamentale di qualsiasi Web Service è la gestione della sicurezza. La sicurezza di un sistema coinvolge i seguenti aspetti:

- **L'autenticazione**
  Ovvero la capacità di individuare le parti coinvolte nell'operazione.
- **L'autorizzazione**
  La concessione della possibilità di accedere ad una risorsa in base all'identità.
- **La fiducia**
  La capacità di poter contare sul risultato di un'operazione eseguita da terze parti.
- **La riservatezza**
  Con riservatezza si intende la capacità di mantenere privata l'informazione durante la trasmissione o la memorizzazione.
- **L'integrità**
  La capacità di impedire che l'informazione posso essere modificata da terze parti senza autorizzazione.

Un Web Service costruito secondo l'architettura REST eredita la gestione della sicurezza dal protocollo HTTP, l'architettura REST infatti si basa largamente su questo protocollo. Di conseguenza per la gestione dell'identità in un Web Service RESTful si possono utilizzare dei meccanismi propri del protocollo HTTP come l'HTTP Basic Authentication o l'HTTP Digest Authentication.
La miglior soluzione adottabile è quella di creare un canale di trasmissione sicuro

come ad esempio *HTTPS* (HTTP over SSL/TLS). Esso garantisce l'integrità e la riservatezza nella trasmissione delle informazioni, coprendo anche gli altri aspetti della sicurezza.
Qualora i meccanismi di sicurezza del protocollo HTTP non siano adeguati al tipo di Web Service che si vuole creare è possibile anche gestire la sicurezza in modo personalizzato; solitamente ciò accade per la gestione dell'autenticazione e dell'autorizzazione dei Client poiché in questi campi i criteri di sicurezza possono variare in base al dominio del problema.
C'è da ricordare in ogni caso che qulasiasi soluzione personalizzata deve rispettare i principi REST e quindi bisogna prestare attenzione a non reinserire il concetto di sessione associata al Client appena autenticato.

2.9. Framework per servizi RESTful

Nonostante l'architettura REST non preveda l'utilizzo di un framework, l'impiego di quest'ultimo nella progettazione di un Web Service può semplificare molto il lavoro, dando possibilità al programmatore di concentrarsi di più sulla logica dell'applicazione piuttosto che sui dettagli relativi alla presentazione del servizio e della comunicazione.
Le caratteristiche generali messe a disposizione da un framework per la creazione di un Web Service RESTful sono:

- **URI routing**
  Questa caratterisca consente di associare uno o più URI ad una risorsa, andando a favorire l'impiego di URI con schema posizionale.

- **Gestione della comunicazione HTTP**
  Un framework REST gestisce la comunicazione tra Client e Server, il framework richiede solamente di specificare il codice da eseguire in corrispondenza di ogni metodo HTTP. In questo modo la logica di gestione della risorsa diventa indipendente dalla comunicazione.

- **Semplificazione della rappresentazione delle risorse**
  Solitamente i framework permettono di rappresentare le risorse tramite dei formati predefiniti (JSON, Atom...) ma anche di rappresentarle in formati personalizzati.
  Ci sono alcuni framework che hanno anche la capacità di gestire la *content negotiation*, cioè sono in grado di fornire al Client la risorsa nel formato richiesto.

Capitolo 3.

Materiali e Metodi

3.1. Principi generali di progettazione

La progettazione di un Web Service RESTful si basa su alcuni punti cardine:

- **Database**
  Poiché tutti i dati necessari per la realizzazione del progetto si trovavano all'interno di diversi file di testo (che insieme alle varie immagini costituiscono il DataSet completo) si è deciso di inserirli tutti all'interno di un database in modo da poterli analizzare in maniera più agevole per soddisfare le richieste del Client, quindi il primo passo è stato quello di implementatare un database contenente tutti i dati presenti nei file di testo associati alle varie immagini. Per fare ciò è stato realizzato uno script che prendesse i dati a disposizione e li inserisse all'interno del database.
  In più oltre ai dati già presenti nei file di testo si è scelto di aggiungere un ulteriore campo nel database, relativo al numero di oggetti presenti in ciascuna immagine,in previsione delle richieste del progetto tra le quali figura un'API che deve andare a conteggiare quante immagini hanno all'interno più di un oggetto di classi diverse; per fare ciò è bastato vedere quante righe ci fossero all'interno di ogni file di testo in quanto ogni riga indicava le caratteriste relative ad un singolo oggetto.
  Il database scelto è MongoDB (trattato in dettaglio nella sezione 3.4.3), una database di tipo NoSQL.

- **URI e metodi HTTP**
  In secondo luogo sono stati definiti tutti gli URI previsti per il Web Service e i metodi HTTP eseguibili su di essi.
  Ogni URI è stato scelto in modo da essere per l'utente il più chiaro e intuitivo possibile (Es: /dominante?img={path dell'immagine}).
  Per tutte le API il metodo consentito è unicamente il metodo GET dato che l'unico obiettivo era quello di visualizzare dei dati e non di inserirne di nuovi, modificarli o eliminarli.

- **Formato dei dati**
  Come ultima cosa è stato stabilito il formato da utilizzare per le risorse restituite

al Client dal Server; il formato scelto è stato il JSON, per la sua semplicità di gestione che lo ha portato ad essere utilizzato come standard. Tutte le API quindi ottengono come risultato un oggetto JSON contenente le informazioni richieste.

3.2. Rete neurale e deep learning

Durante la progettazione di queste API è

3.3. Implementazione

Per la scrittura dei codici delle varie API, data la libertà di scelta prevista dall'archiettetura REST, si è optato per l'utilizzo del linguaggio Python (approfondito nella sottosezione 3.4.1).
Nelle prossime sottosezioni per ognuna delle API viene spiegata qual è l'idea alla base della sua implementazione.
A ciascuna di esse è associata una tabella in cui sono indicati i punti cardine che la identificano:

- **Metodo HTTP**
  Metodo HTTP associato alla chiamata (può essere: GET, POST, PUT, PATCH, DELETE).
- **URI**
  Indica l'URI specifico per la risorsa che si vuole ottenere.
- **Stato HTTP**
  Il codice di stato HTTP che viene restituito da quella particolare chiamata.
- **Formato dati**
  Il formato con cui i dati vengono restituiti nella risposta.
- **Dati restituiti**
  Il contenuto vero e proprio della risposta, ciò che si ottiene dopo aver effettuato la chiamata.

Il codice relativo ad ogni API invece è visionabile nell'Appendice (Capitolo A).

3.3.1. API per in conteggio delle immagini con più di un oggetto di classi diverse

Metodo HTTP	GET
URI	/molti_oggetti/molte_classi
Stato HTTP	200
Formato dati	JSON

Dati restituiti	{
	"Le_foto_con_piu_di_un_oggetto_di_classi_diverse_sono" : "223"
	}

Tabella 3.1.: Implementazione API per il conteggio delle immagini con più di un oggetto di classi diverse

L'API per il conteggio delle immagini con più di un oggetto di classi diverse del RESTful Web Service restituisce il numero di immagini che hanno al loro interno più di un oggetto di classi diverse.
Quest'API, come detto in precedenza, è il motivo per cui in fase di strutturazione del Database si è deciso di creare un campo relativo al numero di oggetti presenti in ciascuna foto.

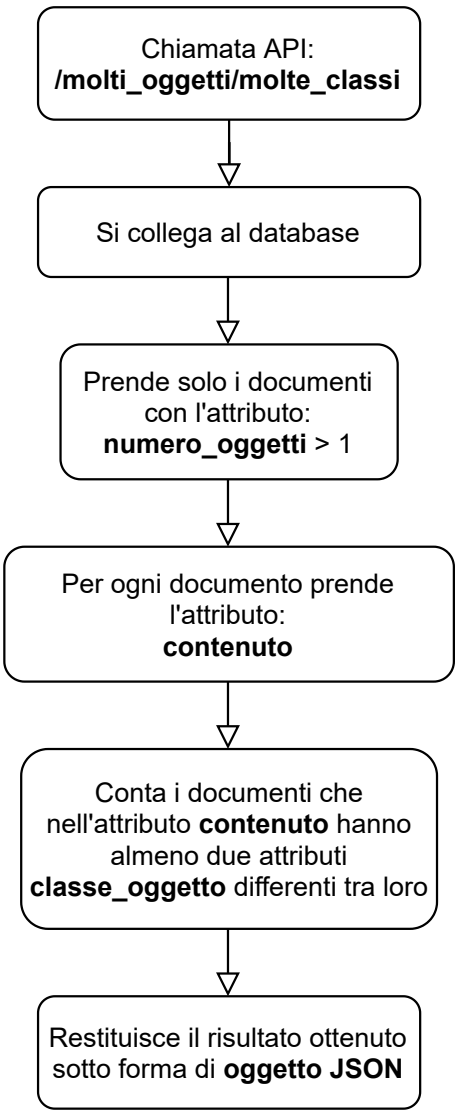

Figura 3.1.: Diagramma di flusso dell'API per il conteggio delle immagini con più di un oggetto di classi diverse

Dall'immagine 3.1 si può vedere il funzionamento di questa API; per prima cosa si vanno a estrarre dal Database tutti i documenti con l'attributo *numero_oggetti* > 1, ovvero tutti i documenti corrispondenti a delle foto che hanno al loro interno più di un oggetto.
Dopodiché tramite un confronto delle varie classi che identificano gli oggetti all'interno di ogni singola immagine (attributo *classe_oggetto*) viene fatto un conteggio del numero di immagini che contengono più di un oggetto di classi diverse.
Infine il risultato viene convertito in un oggetto JSON e restituito come risposta.

Come si può vedere dalla Tabella 3.1 l'URI di questa risorsa è */molti_oggetti/molte_classi* mentre il metodo HTTP utilizzato è il metodo GET. Il codice di stato HTTP 200 che viene restituito sta ad indicare che la richiesta è andata a buon fine, e i dati forniti sono in formato JSON.

3.3.2. API per il riconoscimento dei colori dominanti di un'immagine

Metodo HTTP	GET
URI	/dominante?img={path dell'immagine}
Stato HTTP	200 , 400, 404
Formato dati	JSON

Dati restituiti	{
	"I_colori_dominanti_per_l_immagine_{path dell'immagine}
	_sono" : "['fafaf9','502e28']"
	}

	{
	"Errore": "Non hai specificato l'immagine. Riprova specificando l'immagine"
	}

	{
	"Errore": "L'immagine specificata non esiste"
	}

Tabella 3.2.: Implementazione API per il riconoscimento dei colori dominanti di un'immagine

L'API per il riconoscimento dei colori dominanti di un'immagine restituisce il valore esadecimale dei due colori più presenti nell'immagine analizzata.
A differenza della precedente questa API non lavora con i dati contenuti nel MongoDB ma direttamente con l'immagine che le viene indicata tramite l'URI e che si trova nel DataSet dell'applicazione.

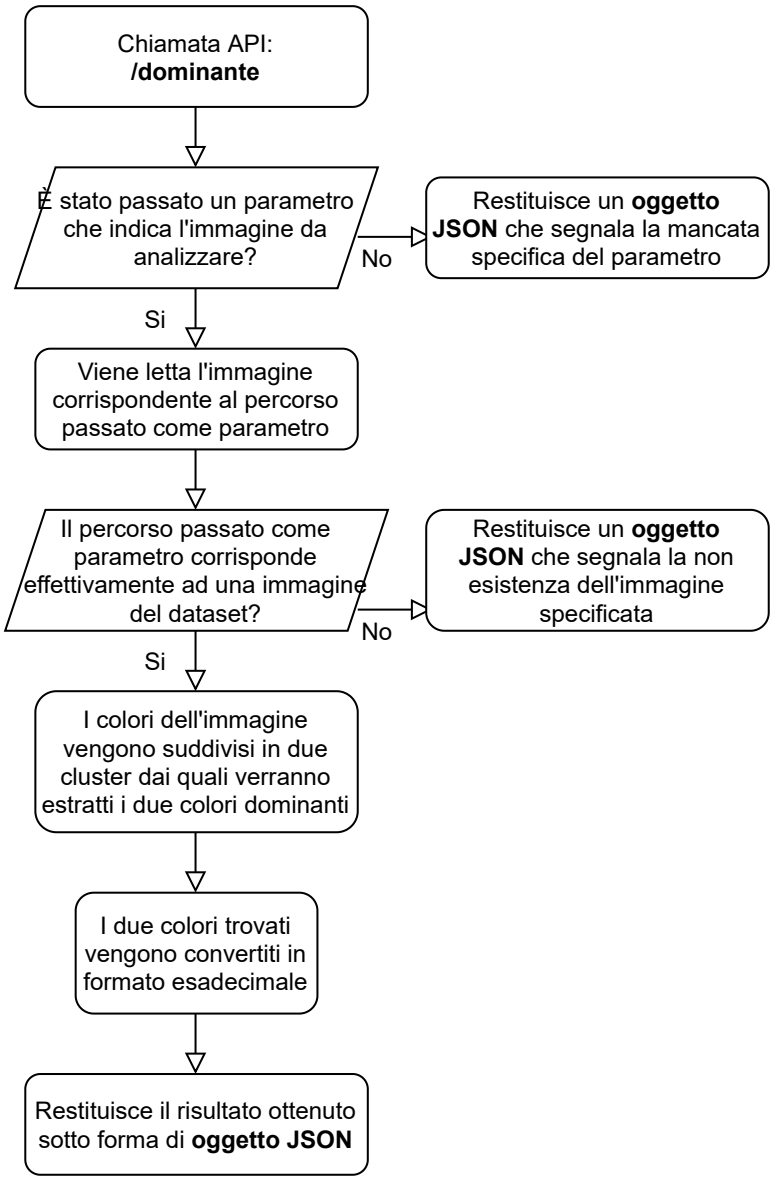

Figura 3.2.: Diagramma di flusso dell'API per il riconoscimento dei colori dominanti di un'immagine

Come si può vedere dal diagramma 3.2 si procede prima di tutto verificando se l'URI contiene il parametro *img* richiesto, questa verifica può avere due esiti:

- **Negativo**: L'API restituisce subito un oggetto JSON di nome "Errore" e con valore una stringa che esprime la necessità di specificare un'immagine da analizzare.
  Lo stato HTTP viene fissato a 400, valore che sta ad indicare una richiesta non valida.

- **Positivo**: Il valore assegnato al parametro *img* dovrebbe indicare il percorso

di un'immagine all'interno del dataset, l'immagine trovata viene letta e memorizzata in una variabile. A questo punto c'è un ulteriore controllo, se il path dell'immagine che è stato passato non è corretto e non corrisponde a nessuna immagine allora la variabile creata precedetemente sarà vuota; l'API restituisce quindi un oggetto JSON di nome "Errore" e valorizzato con una stringa che spiega che l'immagine specificata nell'url non esiste, e lo stato HTTP viene fissato a 404, valore che sta ad indicare l'inesistenza della risorsa richiesta.

Altrimenti se anche questo controllo viene superato i colori dell'immagine vengono suddivisi in due cluster (proprio perché si stanno cercando i due colori dominanti dell'immagine) e di questi cluster viene preso il centro che rappresenta esattamente il colore dominante di quel singolo gruppo.
Fatto ciò i due valori trovati vengono convertiti dal formato RGB al formato esadecimale, in modo da agevolarne la lettura da parte dell'utente, e vengono restituiti sotto forma di oggetto JSON.

Come si può vedere dalla Tabella 3.2 l'URI di questa risorsa è */dominante?img={path dell'immagine}* mentre il metodo HTTP utilizzato è il metodo GET.
Se viene restituito il codice di stato HTTP 200 allora la richiesta è andata a buon fine e i colori dominanti vengono forniti tramite un oggetto JSON.
Se viene restituito il codice di stato HTTP 400 vuol dire che la richiesta non era valida e viene restituito un JSON che fa notare l'errore all'utente.
Se invece viene restituito il codice di stato HTTP 404 vuol dire che il path inserito nell'URI non corrisponde a nessuna immagine esistente nel DataSet e quindi non è stato possibile trovare la risorsa richiesta, si avrà perciò un oggetto JSON che fa notare che l'immagine voluta non esiste.

3.3.3. API per il riconoscimento degli oggetti con il modello YOLOv5

L'API per il riconoscimento degli oggetti con il modello YOLOv5 restituisce per ogni oggetto rilevato nell'immagine le coordinate del riquadro che lo contiene, l'affidabilità del risultato, la classe corrispondente e il nome della classe.

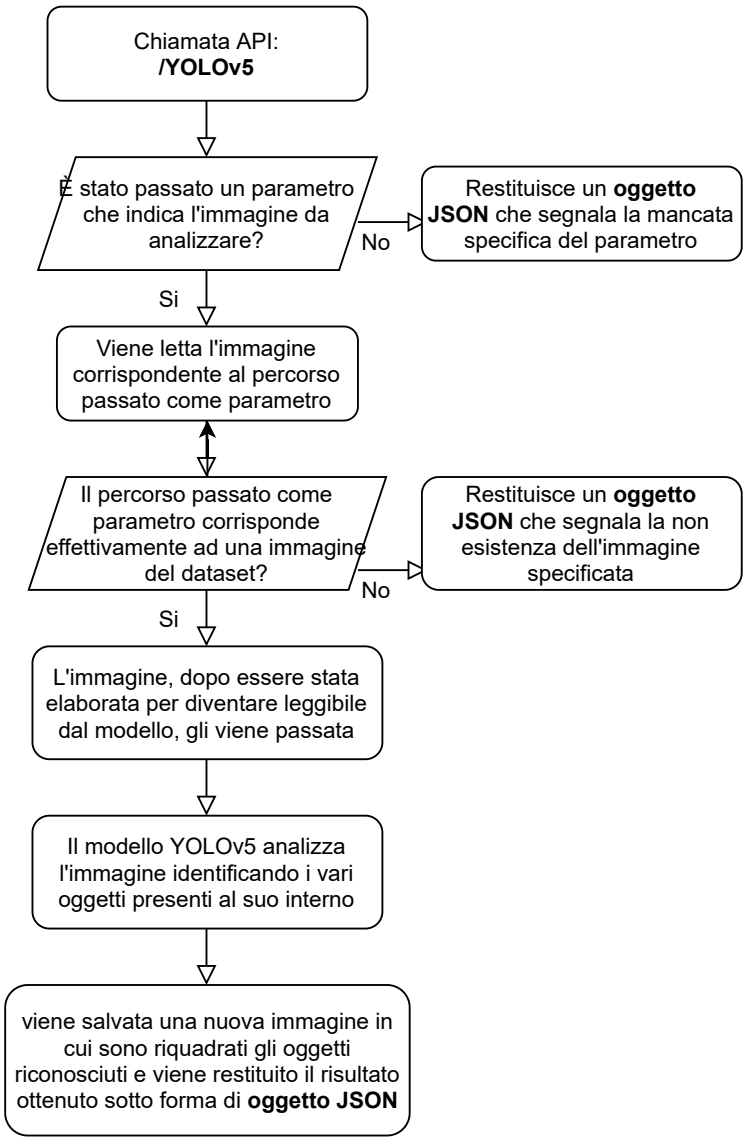

Figura 3.3.: Diagramma di flusso dell'API per il riconoscimento degli oggetti con il modello YOLOv5

Come si vede nel diagramma 3.3 per prima cosa viene effettuata una verifica sull'URI, per vedere se questo contiene il parametro *img* richiesto:

- In caso **negativo** l'API restituisce immediatamente l'oggetto JSON "Errore" valorizzato con una stringa che comunica la necessità di specificare un'immagine da analizzare.
  Lo stato HTTP viene fissato a 400, indicando una richiesta non valida.

- In caso **positivo** viene prima fatto un controllo sul parametro *img* passato, per capire se effettivamente corrisponde al path di un'immagine presente nel

DataSet oppure no.
Qualora la variabile che dovrebbe contenere l'immagine da analizzare sia vuota significa che il Client ha passato un percorso errato e l'API va a restituire un altro oggetto JSON di nome "Errore" che ha ancora una volta una stringa come valore ma diversa dalla precedente, questa indica la non esistenza dell'immagine specificata. In questo caso lo stato HTTP viene fissato a 404, valore che rappresenta l'inesistenza della risorsa passata.

Nel caso in cui anche il secondo controllo venga superato allora si procede col passare l'immagine da esaminare al modello YOLOv5; così il modello la analizza identificando i vari oggetti al suo interno e fornisce dei risultati.
I risultati appena trovati vengono restituiti all'interno di un oggetto JSON e in più viene salvata una nuova immagine identica alla precedente con la differenza che in questa ci sono in sovraimpressione le bounding box che indicano gli oggetti riconosciuti.

Metodo HTTP	GET
URI	/YOLOv5?img={path dell'immagine}
Stato HTTP	200 , 400, 404
Formato dati	JSON

Dati restituiti	{
	"Gli_oggetti_presenti_nell_immagine_{path dell'immagine}
	_sono" :{"xmin":467.7909851074,"ymin":730.4602050781,
	...
	,"class":28,"name":"suitcase"
	}

	{
	"Errore": "Non hai specificato l'immagine. Riprova specificando l'immagine"
	}

	{
	"Errore": "L'immagine specificata non esiste"
	}

Tabella 3.3.: Implementazione API per il riconoscimento degli oggetti con il modello YOLOv5

Come si può vedere dalla Tabella 3.3 l'URI di questa risorsa è */YOLOv5?img={path dell'immagine}* mentre il metodo HTTP utilizzato è il metodo GET.
Se viene restituito il codice di stato HTTP 200 allora la richiesta è andata a buon fine e i dati richiesti vengono forniti sotto forma di oggetto JSON.
Se viene restituito il codice di stato HTTP 400 vuol dire che la richiesta non era valida e viene restituito un JSON che spiega l'errore all'utente.

Se invece viene restituito il codice di stato HTTP 404 vuol dire che il percorso inserito nell'URI non corrisponde a nessuna delle immagini del DataSet e quindi non è possibile trovare la risorsa richiesta, si avrà quindi un oggetto JSON che indica all'utente che l'immagine specificata non esiste.

3.4. Soluzioni adottate

In questa sezione sono elencati e approfonditi il linguaggio, il database e i vari tool utilizzati per lo sviluppo del Web Service:

- Python
- Librerie Python
- MongoDb
- Postman
- YOLOv5

3.4.1. Python

Figura 3.4.: Logo Python

Per sviluppare queste API si è scelto di utilizzare la versione 3.7 del linguaggio Python. Python è un linguaggio di programmazione moderno, nato solamente nel 1991, i cui punti di forza sono i seguenti:

- **È facile da usare**
  Python ha una sintassi e una struttura facili da imparare rispetto ad altri linguaggi. Ad esempio grazie all'uso dell'identazione per la sintassi delle specifiche al posto delle classiche parentesi, oppure grazie all'uso di variabili non tipizzate (c'è un forte controllo dei tipi che viene eseguito a runtime: ad una variabile viene associato un nome che però durante il suo tempo di vita può essere associato a diverse variabili anche di tipo diverso).

- **È portabile**
  Python è un linguaggio interpretato, quindi lo stesso codice può essere sfruttato su qualsiasi piattaforma purché questa abbia installo il giusto interprete.
  Anche se in realtà è più efficiente di un normale linguaggio interpretato poiché il codice sorgente non viene direttamente convertito in linguaggio macchina ma si passa prima per una fase di compilazione in bytecode; e nella maggior parte dei casi questo bytecode, dopo la prima esecuzione del programma, viene riutilizzato evitando così di reinterpretare ogni volta il codice sorgente e migliorando le prestazioni.

- **È multi-paradigma**
  Python è un linguaggio multi-paradigma e supporta il paradigma della programmazione orientata agli oggetti (include funzionalità come l'ereditarietà o l'overloading degli operatori e delle funzioni), della programmazione strutturata ma anche della programmazione funzionale.

- **È gratuito**
  Python è un linguaggio completamente gratuito, non ci sono restrizioni di copyright. Ha inoltre una comunità molto attiva e per questo riceve costantemente miglioramenti che lo mantengono aggiornato e al passo coi tempi.

- **Ha numerose librerie**
  Di base Python offre una standard library che al suo interno contiene più di 200 moduli per svolgere un grandissimo numero di compiti, ma se ciò non bastasse è possibile scaricare e installare migliaia di moduli aggiuntivi creati dalla comunità.

- **Gestisce automaticamente la memoria**
  La memoria è gestita automaticamente grazie ad un meccanismo di garbage collection che si occupa autonomamente dell'allocazione e del rilascio della memoria. In questo modo il programmatore può usare le variabili liberamente senza doversi preoccupare di dichiararle e quindi di allocare/rilasciare spazi di memoria manualmente.

Anaconda

Figura 3.5.: Logo Anaconda

Anaconda è la piattaforma per la Data Science con Python più utilizzata al mondo e semplifica sensibilmente il processo di setup di un ambiente di sviluppo.
L'**Anaconda Distribution** racchiude al suo interno l'installer di Python, un package manager, un environment manager e circa 300 pacchetti installati e pronti all'uso.
Senza dimenticare che mette a disposizione anche **Anaconda Navigator**, un'interfaccia grafica che permette di gestire tutti gli aspetti della piattaforma semplificandone l'utilizzo.
Conda è il package ed environment manager e consente di installare, eseguire e aggiornare dei pacchetti con le relative dipendenze, ma anche di creare, salvare e caricare un ambiente e di passare senza difficoltà da un ambiente ad un'altro sul computer. In questo modo se si ha la necessità di installare un pacchetto che richiede una versione di Python diversa da quella installata si può creare un nuovo ambiente dove eseguirla senza andare ad intaccare la solita versione di Python presente nell'ambiente base.
C'è poi **Anaconda Cloud**, un servizio di cloud per la gestione di pacchetti che semplifica la ricerca e la conservazione di ambienti e package sia Conda che PyPl. I contenuti presenti nel cloud sono messi liberamente a disposizione di tutti e non è necessario essere registrati al servizio per farne uso.

3.4.2. Librerie Python

Flask

Per lo sviluppo di questo Web Service si è scelto di utilizzare il framework Flask. Flask è un web framework per Python che semplifica la progettazione di un Web Service dando ad esempio la possibilità di gestire le richieste HTTP.
Grazie a Flask, tramite un processo di routing, tutti gli URI che si vogliono mappare vengono associati a delle funzioni Python.
In questo modo, quando viene effettuata una chiamata HTTP, Flask verificherà se c'è una corrispondenza tra una delle rotte presenti nel codice e l'URI e il metodo HTTP richiesti. In caso di corrispondenza allora verranno eseguiti i comandi della funzione Python associata.

```
@app.route('/molti_oggetti/molte_classi', methods=['GET'])
```

Figura 3.6.: Esempio di routing con Flask

Nella figura 3.6 si può vedere come è strutturata una rotta di Flask, da una parte c'è il path da inserire nell'URI per richiamarla mentre in *methods* si trovano tutti i metodi HTTP consentiti per quella risorsa.

PyTorch

PyTorch è un framework per il deep learning che grazie alla sua semplicità ed efficienza è diventato molto popolare fra gli sviluppatori. Si può facilmente installare tramite Anaconda con il comando *conda install pytorch -c pytorch*.
L'elemento fondamentale di PyTorch sono i tensori, delle matrici multidimensionali molto simili agli array in NumPy. Entrambe le librerie, PyTorch e NumPy, forniscono metodi per elaborare dei vettori multidimensionali, con la differenza che PyTorch elabora i tensori più velocemente sfruttando l'accelerazione della GPU.
PyTorch risulta essere lo strumento migliore quando si parla di machine learning e reti neurali in quanto per queste utlime permette un comodo debugging e allo stesso tempo grazie all'utilizzo di GPU multiple tutti i processi vengono velocizzati.
Il modello preaddestrato di PyTorch scelto per analizzare le immagini per questo Web Service è il modello YOLOv5, che verrà analizzato nel dettaglio alla sottosezione 3.11.

PyMongo

PyMongo è una libreria di Python che contiene al suo interno tutti i tool necessari allo sviluppatore per lavorare con MongoDB.
Tramite PyMongo viene configurata una connessione al database MongoDB in modo da poter interagire con esso, in questo modo grazie ad alcuni comandi di PyMongo si possono andare ad eseguire le classiche operazioni CRUD con i vari documenti e collection del database.
Un'altra funzionalità messa a disposizione da PyMongo è quella di effettuare delle query sul database in modo da filtrare i dati.

OpenCV

Figura 3.7.: Logo OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria Python per la gestione della visione artificiale.
Questa libreria permette di manipolare le immagini trattandole come delle matrici di pixel alle quali è possibile accedere in maniera facile e rapida; fornisce centinaia di funzioni per l'acquisizione, l'analisi e la manipolazione dei dati visivi e può eliminare alcuni dei problemi che i programmatori devono affrontare durante lo sviluppo di

applicazioni che si basano sulla visione artificiale.
L'elaborazione di immagini attraverso OpenCV è alla base della Computer Vision e permette di modificare e migliorare le immagini o i video (ad esempio eliminando il rumore da un'immagine) in modo tale da facilitare le successive operazioni (come il riconoscimento di oggetti).

3.4.3. MongoDb

Figura 3.8.: Logo MongoDB

MongoDB è un database open-source di tipo NoSQL che memorizza i dati in documenti in stile JSON con schema dinamico (questo formato viene indicato da MongoDB come BSON).
BSON significa JSON binario (Binary JSON), gli oggetti in questo formato sono delle liste ordinate di elementi in cui ciascuno campo ha un nome, un tipo e un valore, i tipi includono:

- Stringhe
- Interi
- Long
- Decimal
- Double
- Date
- Byte array
- Booleani
- NULL
- Oggetti BSON
- Array BSON
- Codice JavaScript
- Espressioni regolari

I tipi BSON quindi non sono altro che un superset dei tipi JSON (JSON non include ad esempio i tipi byte array o date) ma con la differenza che non c'è un tipo numerico universale come in JSON.
Il formato BSON è stato progettato per essere più efficiente rispeto al JSON sia in termini di spazio richiesto dai dati sia per la velocità di ricerca.
Tuttavia in alcuni casi il formato BSON può diventare meno efficiente del JSON occupando più spazio a causa delle lunghezze fisse imposte e degli indici espliciti degli array (gli elementi di un documento BSON sono dotati di un campo lunghezza di estensione prefissata al fine di facilitare la ricerca di informazioni).
La tabella 3.4 mostra le principali differenze tra questi due formati.

	JSON	**BSON**
Codifica	UTF-8	Binaria
Acronimo	JavaScript Object Notation	Binary JavaScript Object Notation
Leggibilità	Uomo e macchina	Solo macchina
Tipi di dato supportati	String Boolean Array Null Oggetti JSON Numeri	String Boolean Array Null Oggetti BSON Int Long Decimal Double Date Espressioni regolari Codice JavaScript "ObjectId" "Timestamp" "Min Key" "Max Key"

Tabella 3.4.: Tabella riassuntiva delle differenze tra formato JSON e BSON

I database MongoDB risiedono su di un Server al quale bisogna collegarsi per poi effettuare le operazioni di lettura e scrittura volute.
I gruppi di documenti prendono il nome di collection; la caratteristica distintiva dei documenti BSON che risiedono in questo tipo di database è che non sono strutturati e non possiedono uno schema predefinito, l'unico campo obbligatorio per ogni documento e che viene indicizzato automaticamente è il campo *_id* che serve a identificarlo univocamente, per il resto ogni documento può differire dall'altro sia in termini di contenuti che di dimensioni.

Organizzando i dati in documenti schemeless MongoDB riesce a rispondere alle interrogazioni con tempi molto inferiori rispetto ai database relazionali nei quali i dati, essendo separati in tabelle multiple, necessitano di join per raggruppare i risultati. Questo tipo di organizzazione dei dati rende i database noSQL più flessibili e più facilmente scalabili rispetto a dei classici database SQL.
Ecco quindi spiegati i principali motivi della scelta di MongoDB:

- Scalabilità: l'elevata scalabilità orizzontale permette di distribuire facilmente i database su più server senza danneggiarne la funzionalità, quindi anche se il traffico aumenta le prestazione continueranno a rimanere costanti.
- Disponibilità: per garantire la disponibilità dei dati nel lungo periodo con MongoDB si possono creare delle copie dei database e metterle a disposizione su diversi Server, in modo da rendere l'applicazione sempre disponibile.
- Flessibilità: il database può adeguarsi dinamicamente al progetto in qualsiasi momento senza troppe difficoltà.
- Facilità di utilizzo: MongoDB è infatti uno dei Database che richiede la minor quantità di sforzo iniziale per l'utilizzo, gli sviluppatori che si approcciano a questo DB sprecare solo una minima quantità del loro tempo nell'imparare come scrivere del codice per interagire con un database MongoDB.
- Documenti schemeless: come detto in precedenza i documenti in un database MongoDB non sono vincolati da nessun legame se non quell di avere un id univoco, ecco quindi che ogni documento può essere modificato/cancellato/aggiunto in qualsiasi momento senza nessun problema.

3.4.4. Postman

Figura 3.9.: Logo Postman

Per eseguire dei test con le varie API del progetto si è scelto di utilizzare Postman. Postman è una piattaforma che permette di eseguire richieste HTTP ad un Server di backend. La cosa più importante di questo tool è che da la possibilità di conoscere tutti i dettagli di una richiesta HTTP, sia per quanto riguarda la *request* che la *response*.
Nell'immagine 3.10 si può vedere come si presenta la barra iniziale del programma per effettuare una richiesta HTTP, tramite questa barra è possibile:

- Specificare il tipo di richiesta da effettuare (GET,POST, PUT, PATCH, DELETE).
- Scrivere l'indirizzo (URI) a cui effettuare la richiesta.
- Inviare la richiesta tramite il tasto SEND
- Salvare la richiesta in una *collection*, in modo da poter organizzare le varie richiesta e da richiamarle facilmente in futuro.

Nella sezione sottostante alla barra invece sono messe a disposizione una serie di tab per specificare i dati per l'autorizzazione della chiamata API, il *body* e i vari *headers*.
Ecco quindi che è possibile impostare tutti i dati di una tipica chiamata API.

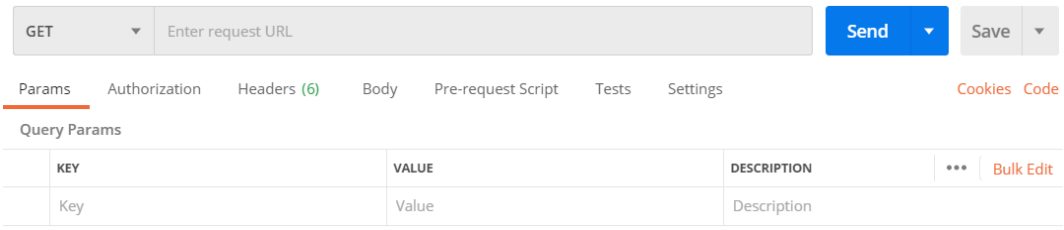

Figura 3.10.: Barra iniziale Postman

3.4.5. Machine learning e reti neurali

Una rete neurale è un modello matematico complesso che si ispira al funzionamento biologico del cervello umano e che, con meccanisimi simili ad esso, è in grado di apprendere informazioni.
Il punto di forza delle reti neurali è che non devono essere esplicitamente programmate ma basta semplicemente addestrarle con opportuni algoritmi.
Ad esempio si consideri il caso di una rete neruale che abbia lo scopo di riconoscere delle immagini; ovviamente un computer non è in grado di riconoscere che oggetto sia raffigurato in un'immagine ma lo riesce a capire andando ad analizzare le caratteristiche individuali della figura. Ed è proprio grazie ad un apposito algoritmo che il computer riesce a sapere quali sono le caratteristiche rilevanti per un determinato elemento. Questo algoritmo può essere perfezionato automaticamente dal computer stesso tramite il deep learning, ovvero un apprendimento automatico in cui il computer riesce a imparare e migliorarsi grazie alla propria esperienza senza alcun intervento umano.
Il deep learning prevede l'elaborazione di grandi quantità di dati tramite delle reti neurali, a ogni elaborazione vengono ottenute nuove informazioni e vengono ampliati i collegamenti esistenti, facendo sì che il sistema impari senza nessun intervento esterno. Per lo sviluppo dell'API per il riconoscimento degli oggetti si è scelto di adottare una rete neurale pre addestrata, ovvero l'architettura YOLOv5.

YOLOv5

YOLO (You Only Look Once) è un avanzato sistema di riconoscimento degli oggetti rilasciato l'8 giugno 2015 da Joseph Redmond, matematico e informatico dell'università di Washington. La sua quinta versione (YOLOv5) è nata il 18 maggio 2020 da Glenn Jocher (fondatore e CEO di Ultralytics) e tutto il codice si trova nella repository di Ultralytics LLC; questa versione si basa interamente sul framework PyTorch.
Il sistema YOLO lavora dividendo un'immagine in un sistema a griglia e per ogni griglia rileva gli oggetti al suo interno, al termine dell'elaborazione vengono mantenute solamente le bounding box con la confidenza più elevata e vengono scartate le altre. Questo modello può essere utilizzato anche per il rilevamento di oggetti in tempo reale.
Esistono 4 diversi modelli nella repository di Ultralytics: YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x. Il primo è il più piccolo e il meno preciso, l'ultimo è il più grande e con la massima precisione; tutti i modelli funzionano su PyTorch.
Tutti i modelli sono stati preaddestrati sul Modanet dataset, questo dataset è costituito da circa 330mila immagini suddivise in 80 categorie.
Per lo sviluppo dell'API per il riconoscimento di oggetti si è scelto di utilizzare il modello YOLOv5s, nell'immagine 3.11 si può vedere un esempio di come questo modello lavori.

Il modello YOLOv5 accetta input di tipo URL, PIL, OpenCV, Numpy e PyTorch e restituisce i rilevamenti nei formati output torch, pandas o JSON. In questo progetto si è scelto di avere in output un oggetto JSON di tipo vettoriale in cui ogni elemento dell'array corrisponde ad uno degli oggetti identificati.
Ogni elemento ha a sua volta 7 parametri:

- Le coordinate **X min** e **X max** che indicano l'inizio e la fine, relativamente all'asse delle ascisse, del riquadro che contiene l'immagine rilevata.
- Le coordinate **Y min** e **Y max** entro cui si estende il riquadro contenente l'oggetto riconosciuto.
- Un punteggio di **Affidabilità** che indica quanto sia probabile che l'oggetto si trovi all'interno del riquadro descritto in precedenza.
- La **Classe** dell'oggetto identificato, di default le classi che vengono riconosciute sono ben 80.
- Il **Nome** della classe corrispondente all'oggetto.

Nell'immagine 3.12 si può vedere un esempio di oggetto JSON restituito dopo aver effettuato l'analisi con il modello YOLOv5s.

Figura 3.11.: Esempio di analisi con il modello YOLOv5s


```
[
{"xmin":749.5,"ymin":43.5,"xmax":1148.0,"ymax":704.5,"confidence":0.8740234375,"class":0,"name":"person"},
{"xmin":433.5,"ymin":433.5,"xmax":517.5,"ymax":714.5,"confidence":0.6879882812,"class":27,"name":"tie"},
{"xmin":115.25,"ymin":195.75,"xmax":1096.0,"ymax":708.0,"confidence":0.6254882812,"class":0,"name":"person"},
{"xmin":986.0,"ymin":304.0,"xmax":1028.0,"ymax":420.0,"confidence":0.2873535156,"class":27,"name":"tie"}
]
```


Figura 3.12.: Esempio di JSON dopo l'analisi con il modello YOLOv5s

Capitolo 4.

Risultati raggiunti e Discussione

In questo capitolo sono mostrati i risultati ottenuti in seguito alla progettazione e all'implementazione delle varie API finalizzate all'analisi del Fashion DataSet.
Per ogni API sono allegate delle schermate di Postman che mostrano dei veri e propri esempi del funzionamento dell'API, mostrando i vari URI e metodi HTTP di ogni richiesta ma anche i risultati ottenuti, quindi i vari oggetti JSON che possono essere restituiti dalla chiamata, e i codici HTTP ad essi associati.

4.1. Conteggio delle immagini con più di un oggetto di classi diverse

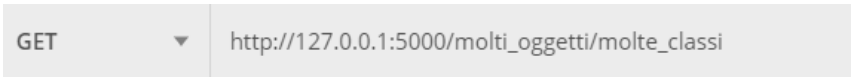

Figura 4.1.: Richiesta dell'API per il conteggio delle immagini con più di un oggetto di classi diverse

La prima API viene chiamata con il metodo HTTP **GET** e l'URI della risorsa è */molti_oggetti/molte_classi*, come si puoò vedere dall'immagine 4.1.

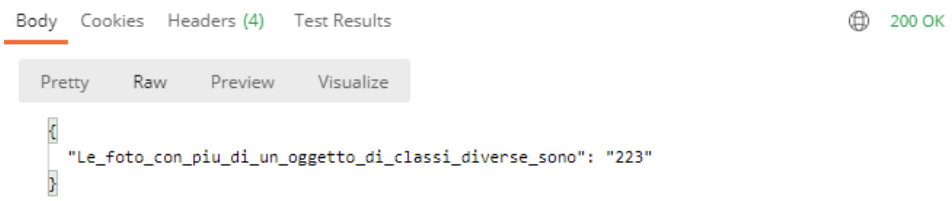

Figura 4.2.: Risposta dell'API per il conteggio delle immagini con più di un oggetto di classi diverse

Se tutto viene eseguito correttamente lo stato HTTP viene fissato a 200 ed il risultato restituito è quello visualizzabile nell'immagine 4.2.
La risposta ottenuta è un oggetto JSON, c'è quindi una coppia chiave-valore in cui la chiave è una stringa che spiega semplicemente cosa rappresenta il valore ad essa

corrispondente mentre il valore non è altro che il numero di immagini che al loro interno hanno più di un oggetto di classi diverse.

4.2. Riconoscimento dei colori dominanti di un'immagine

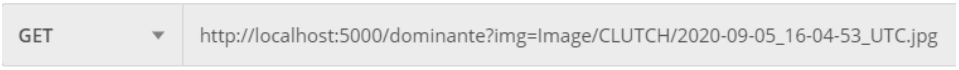

Figura 4.3.: Richiesta dell'API per il riconoscimento dei colori dominanti di un'immagine

La seconda API viene chiamata con il metodo HTTP **GET** e l'URI della risorsa è */dominante* seguito dal parametro *?img={path dell'immagine}*, come si può vedere dall'immagine 4.3.
Con *{path dell'immagine}* si intende il percorso dell'immagine, all'interno del DataSet dell'applicazione, della quale si vogliono conoscere i colori dominanti.

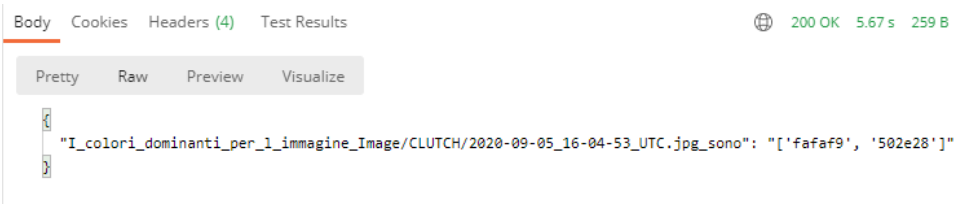

Figura 4.4.: Risposta dell'API per il riconoscimento dei colori dominanti di un'immagine

Se la chiamata viene eseguita correttamente lo stato HTTP viene valorizzato a 200 e il risultato ottenuto è quello visualizzabile nell'immagine 4.4.
Si tratta di un oggetto JSON costituito da una coppia chiave-valore dove la chiave è una stringa che descrive il risltato mentre il valore è un array formato da due elementi che rappresentano in formato esadecimale i due colori dominanti dell'immagine.

4.2.1. Possibili errori

La chiamata di questa API potrebbe concludersi anche con un errore nel caso in cui l'URI non venisse formulato correttamente.
Di seguito sono illustrati i due tipi di errori possibili.

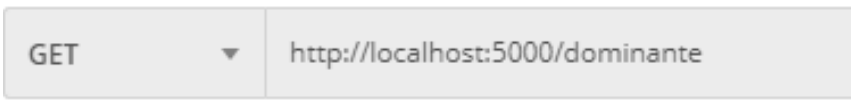

Figura 4.5.: Richiesta senza parametro dell'API per il riconoscimento dei colori dominanti di un'immagine

La figura 4.5 è un esempio di richiesta dell'API senza la specificazione del parametro immagine.
Nel caso in cui l'API venisse chiamata senza passarle il parametro necessario il codice di stato HTTP verrebbe settato a 400, ovvero BAD REQUEST, che sta a indicare che la richiesta non è stata formulata nel modo corretto.

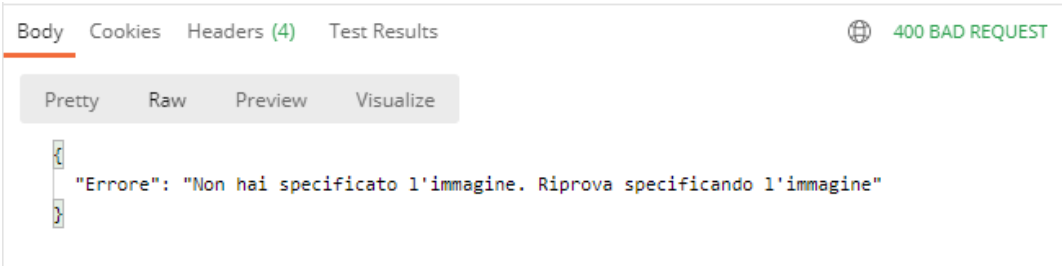

Figura 4.6.: Risposta senza parametro dell'API per il riconoscimento dei colori dominanti di un'immagine

Nella figura 4.6 vediamo che la risposta in caso di parametro mancante sarebbe un oggetto JSON di nome "Errore" che segnala all'utente qual è il motivo del mancato funzionamento dell'API.

Figura 4.7.: Richiesta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine

L'altro possibile errore che si può avere è invece dovuto all'utilizzo di un parametro errato nella richiesta, cioè non corrispondente al *path* di nessuna immagine presente nel DataSet.
L'immagine 4.7 è un esempio di richiesta con un parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine.
In questo caso lo stato HTTP viene settato a 404, ovvero NOT FOUND, questo codice infatti serve a comunicare che la risorsa richiesta non è statà trovata in quanto non esistente o non raggiungibile.

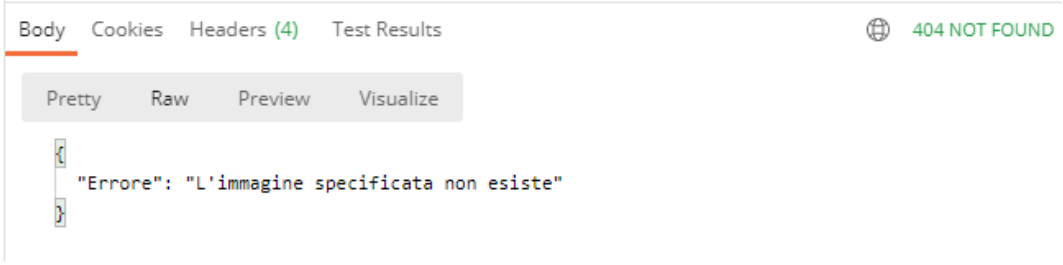

Figura 4.8.: Risposta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine

Nell'immagine 4.8 si può vedere che anche in questo caso la risposta consiste in un oggetto JSON di nome "Errore" che segnala all'utente che cosa non ha funzionato nella chiamata dell'API.

4.3. Riconoscimento degli oggetti con il modello YOLOv5

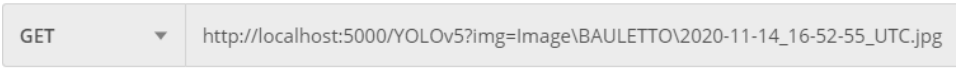

Figura 4.9.: Richiesta dell'API per il riconoscimento di oggetti con il modello YOLOv5

La terza API viene chiamata con il metodo HTTP **GET** e l'URI della risorsa è */YOLOv5* seguito dal parametro *?img={path dell'immagine}*, come si può vedere dall'immagine 4.9.
Con *{path dell'immagine}* si intende il percorso dell'immagine che si vuole analizzare con il modello YOLOv5 in modo da conoscere gli oggetti presenti in essa.

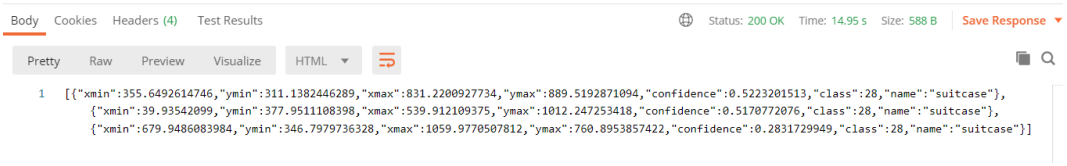

Figura 4.10.: Rispsota dell'API per il riconoscimento di oggetti con il modello YOLOv5

Se la chiamata viene eseguita correttamente lo stato HTTP viene valorizzato a 200 e nell'immagine 4.10 si può vedere qual è il risultato ottenuto.
Si tratta di un vettore composto da diversi oggetti JSON, uno per ogni oggetto che è stato riconosciuto nell'immagine.
Ogni oggetto JSON contiene i valori delle coordinate minime e massime che vanno a identificare l'oggetto, il margine di errore di questo risultato, la classe che corrisponde all'oggetto identificato e il nome della classe stessa.
In più viene salvata tra i file del progetto anche una nuova immagine identica a quella analizzata ma che ha in sovraimpressione i riquadri che identificano i vari oggetti accompagnati dal nome della classe e dal grado di affidabilità.
L'immagine 4.11 ne è un esempio.

4.3.1. Possibili errori

La chiamata di questa API potrebbe portare ad un errore qualora l'URI non venisse formulato correttamente.
Di seguito sono mostrate le due tipologie di errori possibili.

Nell'immagine 4.12 si può vedere un esempio di richiesta dell'API senza specificare il parametro immagine.

Figura 4.11.: Immagine che mostra gli oggetti identificati mediante il modello YOLOv5

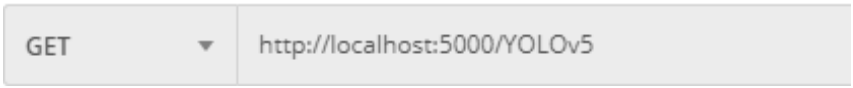

Figura 4.12.: Richiesta senza parametro dell'API per il riconoscimento degli oggetti con il modello YOLOv5

Nel caso in cui l'API venisse chiamata senza specificare il parametro necessario il codice di stato HTTP verrebbe valorizzato a 400, ovvero BAD REQUEST, indicando che la richiesta non è stata formulata nel modo corretto.

Nella figura 4.13 vediamo che in questo caso la risposta sarebbe un oggetto JSON di nome "Errore" che comunica all'utente il motivo per cui l'API non ha funzionato nel modo corretto.

L'altro errore possibile è dovuto all'utilizzo di un parametro errato nella richiesta, cioè un parametro che non corrispondente al *path* di nessuna immagine presente nel DataSet.

L'immagine 4.14 è un esempio di richiesta con un parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine. In questo caso lo stato HTTP viene fissato a 404, ovvero NOT FOUND, codice che indica che la risorsa richiesta non è statà trovata poiché non esiste o non è raggiungibile.

Nell'immagine 4.15 si può vedere che anche in questo caso la risposta è un oggetto JSON di nome "Errore" che segnala all'utente il motivo per cui la chiamata dell'API non ha funzionato.

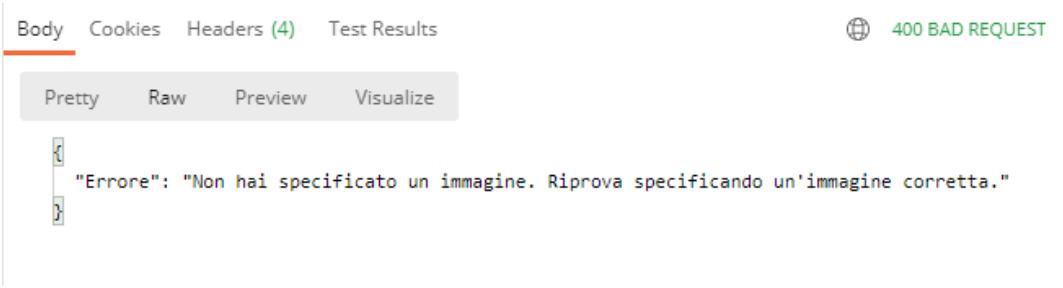

Figura 4.13.: Risposta senza parametro dell'API per il riconoscimento degli oggetti con il modello YOLOv5

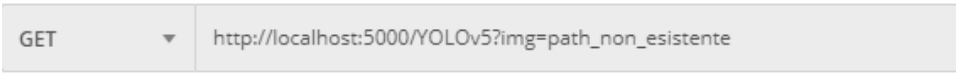

Figura 4.14.: Richiesta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine

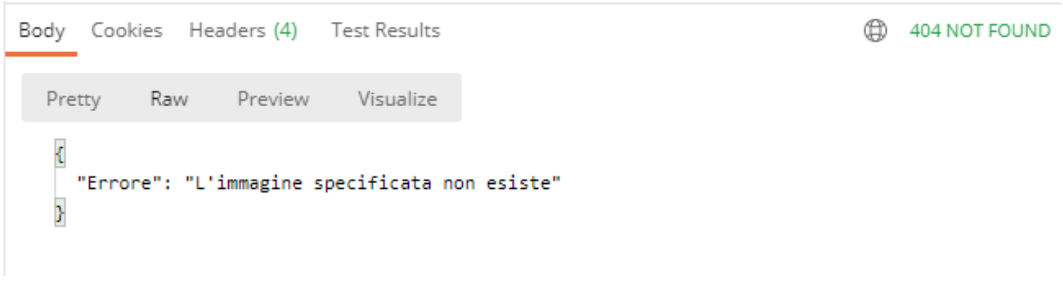

Figura 4.15.: Risposta con parametro non corrispondente a nessuna immagine dell'API per il riconoscimento dei colori dominanti di un'immagine

Capitolo 5.

Conclusioni e Lavori Futuri

5.1. Conclusioni

In questa tesi è stato mostrato quanto i Social Media siano ormai fondamentali nel mondo del Fashion Marketing, e che qualsiasi azienda deve puntare su questo settore se vuole espandersi.
A tal proposito è stato creato un Web Service basato sull'architettura RESTful con lo scopo di analizzare delle immagini provenienti dai Social Nerwork (con una particolare attenzione ad Instagram) per poi prevedere quelli che saranno i futuri trend.
Nel primo capitolo c'è un'introduzione del contesto che ha portato a quest'idea del Web Service, in cui viene spiegata l'importanza dei Social Media al giorno d'oggi. Dopodiché è stato fatto un accenno a cos'è un servizio REST e quali sono i principi su cui quest'architettura si basa.
Il secondo capitolo consiste in una vera e propria spiegazione di come vada implementato un servizio REST: a partire dall'identificazione delle risorse fino ad arrivare ai possibili framework per lo sviluppo, passando per vari aspetti come l'utilizzo dei metodi HTTP per le operazioni CRUD o la gestione dei test tramite un apposito tool.
Nel terzo capitolo è stato approfondito il processo di implementazione del Web Service creato per l'analisi dei dati, con una precisa spiegazione del funzionamento di ognuna delle API e una descrizione di tutti i pratical design impiegati (linguaggio, librerie e vari tool).
Il quarto capitolo tratta il modo in cui è possibile richiamare queste API e i vari risultati che possono essere generati; gli oggetti JSON restituiti in caso si chiamata corretta dell'API ma anche gli oggetti JSON restituiti qualora ci siano degli errori nella richiesta.

5.2. Lavori Futuri

Un modo per migliorare questo Web Service potrebbe essere quello di addestrare la rete neurale del modello YOLOv5 tramite un Dataset ancora più grande del modanet dataset utilizzato e con immagini specifiche per le varie componenti di una borsa in

modo da ottenere dei risultati ancora più precisi.
Un altro modo per migliorare il servizio potrebbe essere quello di far sì che funzioni anche con dei set di immagini attinenti ad altri capi d'abbigliamento e non solamente alle borse; in questo modo si potrebbero analizzare immagini di cappelli, abiti, scarpe, accessori, ecc...
Ecco quindi che la rete neurale potrebbe essere addestrata con un ulteriore dataset costituito da qualsiasi tipo di oggetto appartenente al mondo della moda.
Poi vista la presenza di un'API per il riconoscimento dei colori dominanti di un'immagine la si potrebbe sfruttare per capire i gusti dei clienti; infatti appongiandosi a quest'API se ne potrebbe creare un'altra che determini il colore più presente in tutte le immagini di uno specifico modello di borsa, in questo modo si troverebbe quello che è il colore preferito dai clienti per un singolo modello.
Infine un aspetto migliorabile è quello della repository in cui sono memorizzate tutte quante le immagini; attualmente si trovano all'interno di una cartella in locale e quindi l'intero dataset di immagini viene salvato ogni volta nella macchina su cui opera il programma.
Un'ottima soluzione sarebbe quella di salvare tutto il dataset online e di analizzare le immagini tramite uno stream live dalle fonti, ciò porterebbe a un enorme risparmio di spazio.

Appendice A.

Appendice

Il codice corrispondente alle API di cui si parla nella tesi è riportato in questa sezione tramite degli screenshot presi da PyCharm.

A.1. API per il conteggio delle immagini con più di un oggetto di classi diverese

```
# API n.2-2: conteggio di immagini con più di un oggetto di classi dentro

@app.route('/molti_oggetti/molte_classi', methods=['GET'])

def piu_uno_piu_classi():
    piu_classi = collection.find({'numero_oggetti': {"$gt": 1}})
    contatore = 0
    for i in piu_classi:
        array = i['contenuto']
        confronto = array[0]['classe_oggetto']
        for j in array:
            classe = j['classe_oggetto']
            if classe != confronto:
                contatore += 1
                break
    text_valore = {f'Le_foto_con_piu_di_un_oggetto_di_classi_diverse_sono': str(contatore)}
    return text_valore, 200
```

Figura A.1.: Codice dell'API per il conteggio delle immagini con più di un oggetto di classi diverse

A.2. API per il riconoscimento dei colori dominanti di un'immagine

```python
# API n.3: per una determinata classe quale è il colore predominante
@app.route('/dominante')
def colore_dominante():
    if 'img' in request.args:
        jpg = str(request.args['img'])
        img = cv2.imread(jpg)
        if img is not None:
            immagine = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            immagine = immagine.reshape((immagine.shape[0] * immagine.shape[1], 3))
            kmeans = KMeans(2)
            kmeans.fit(immagine)
            colori_dominanti = kmeans.cluster_centers_
            colori_dominanti.astype(int)

            esadecimali = []
            for i in colori_dominanti:
                j = str(i).strip("][")
                splittato = j.split()
                colori = '%02x%02x%02x' % (int(float(splittato[0])), int(float(splittato[1])), int(float(splittato[2])))
                esadecimali.append(colori)
            text_valore = {f'I_colori_dominanti_per_l_immagine_{jpg}_sono': str(esadecimali)}
            return text_valore, 200
        else:
            text_error = {"Errore": "L'immagine specificata non esiste"}
            return text_error, 404
        # return "I colori dominanti per l'immagine " + jpg + " sono " + str(esadecimali)
    else:
        text_error = {"Errore": "Non hai specificato l'immagine. Riprova specificando l'immagine"}
        return text_error, 400
```

Figura A.2.: Codice dell'API per il riconoscimento dei colori dominanti di un'immagine

A.3. API per il riconoscimento degli oggetti con il modello YOLOv5

```python
# API n.5: per il riconoscimento degli oggetti presenti in un'immagine con il modello YOLOv5s

@app.route('/YOLOv5')

def pred_yolo():
    if 'img' in request.args:
        jpg = str(request.args['img'])
        img = cv2.imread(jpg)
        if img is not None:
            model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
            img2 = cv2.imread(jpg)[:, :, ::-1]

            results = model(img2, size=640)
            results.print()
            results.save()
            predictions2 = results.pandas().xyxy[0].to_json(orient="records")

            return predictions2
        else:
            text_error = {"Errore": "L'immagine specificata non esiste"}
            return text_error, 404
    else:
        text_error = {"Errore": "Non hai specificato un immagine. Riprova specificando un'immagine corretta."}
        return text_error, 400
```

Figura A.3.: Codice dell'API per il riconoscimento degli oggetti presenti in un'immagine con il modello YOLOv5

Bibliografia

[1] Matteo Starri. Digital 2021: i dati globali. https://wearesocial.com/it/blog/2021/01/digital-2021-i-dati-globali/, 27 Gennaio 2021.

[2] Martina Foschetti. La rivoluzione dei social media nella fashion industry. https://marketing-espresso.com/la-rivoluzione-dei-social-media-nella-fashion-industry/, 26 Giugno 2020.

[3] Andrea Chiarelli. Restful web services – la guida. https://www.html.it/guide/restful-web-services-la-guida/, 23 Novembre - 6 Febbraio 2011.

[4] Progettazione dell'api web restful. https://docs.microsoft.com/it-it/azure/architecture/best-practices/api-design, 9 dicembre 2021.

[5] Andrea Barghigiani. Facciamo chiarezza su rest e restful api. https://skillsandmore.org/rest-e-restful-api-introduzione/, 23 Gennaio 2017.

[6] Introduzione a json. https://www.json.org/json-it.html.

[7] Jon Penland. Guida completa ed elenco dei codici di stato http. https://kinsta.com/it/blog/codici-di-stato-http/, 18 Agosto 2021.

[8] Wikipedia. Codici di stato http. https://it.wikipedia.org/wiki/Codici_di_stato_HTTP.

[9] Ezio Melotti. Perché usare python. https://www.html.it/pag/15608/perche-usare-python/, 7 Novembre 2016.

[10] Wikipedia. Python. https://it.wikipedia.org/wiki/Python.

[11] Wikipedia. Anaconda. https://en.wikipedia.org/wiki/Anaconda_(Python_distribution).

[12] Flask rest api tutorial. https://pythonbasics.org/flask-rest-api/.

[13] Simone Scardapane. Alle prese con pytorch. https://iaml.it/blog/alle-prese-con-pytorch-parte-1, 19 Marzo 2021.

[14] Come interrogare un database mongodb usando pymongo su python. https://datapeaker.com/big-data/procesamiento-de-imagenes-opencv-procesamiento-de-imagenes-con-opencv/.

[15] Giovanni Dezio. Introduzione ad opencv in python. https://italiancoders.it/introduzione-ad-opencv-in-c/, 23 Aprile 2019.

[16]

[17] Wikipedia. Mongodb. https://it.wikipedia.org/wiki/MongoDB.

[18] Vito Lavecchia. Cos'è e quali sono le caratteristiche del dbms mongodb. https://vitolavecchia.altervista.org/cose-e-quali-sono-le-caratteristiche-del-dbms-mongodb/.

[19] Postman per testare le api. http://www.datrevo.com/postman-per-testare-le-api/.

[20] Mohit Maithani. Guide to yolov5 for real-time object detection. https://analyticsindiamag.com/yolov5/, 19 Dicembre 2020.

[21] Glenn Jocher. Load yolov5 from pytorch hub. https://github.com/ultralytics/yolov5/issues/36, 11 Giugno 2020.

[22] D. Mukhopadhyay S. Kanoje, V. Powar. Using mongodb for social net729 working website deciphering the pros and cons. https://arxiv.org/ftp/arxiv/papers/1503/1503.06548.pdf, 2015.