



POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Relazione di Software Architecture and Pattern Design

Web app per il rilevamento di strade accidentate

Studenti :

Antonio Colacicco, Giuseppe Farano, Vito Guida

Anno Accademico 2023 - 2024

Abstract

Questo progetto mira a sviluppare una webapp innovativa per il rilevamento automatico di danni stradali lungo percorsi specifici, utilizzando modelli avanzati di machine learning e tecnologie cloud. L'applicazione integra le API di Google Maps e OpenStreetMap per acquisire immagini stradali aggiornate e utilizza YOLOv7 ottimizzato per rilevare e classificare danni come crepe e buche. L'architettura del sistema, basata sul pattern MVC e sviluppata con il framework Flask, garantisce modularità e facilità di manutenzione. Test effettuati su percorsi urbani dimostrano l'efficacia del sistema nel fornire informazioni dettagliate e visivamente accessibili sui danni rilevati, con ampie prospettive per l'ottimizzazione e l'espansione futura

Indice

1	Introduzione	1
1.1	Motivazione del progetto e Background	1
1.2	Stato dell'arte	3
2	Requisiti funzionali e non funzionali	4
2.1	Requisiti funzionali	4
2.2	Requisiti Non funzionali URPS+	5
3	Workflow	7
3.1	Fase iniziale	7
3.1.1	Analisi del problema e delle soluzioni proposte	7
3.1.2	Approccio proposto	7
3.2	Fase di determinazione dei requisiti	8
3.3	Fase di ricerca del modello	8
3.4	Scelta di realizzare una webapp con Flask e scelta dei servizi esterni	9
3.5	Fase di scelta dell'architettura	10
3.6	Fase di sviluppo con metodologia DevOps	11
3.7	Fase di test	11
4	Tecnologie utilizzate	12
4.1	Modello YOLOv7x_640	12
4.1.1	Yolo	12
4.1.2	Transfer learning	14
4.2	Google Cloud Platform	14
4.2.1	Interazione con Google Cloud Platform	14
4.3	Open Street Map API	15
4.4	Flask	15
5	Architettura e struttura del progetto	17
5.1	Pattern MVC	17
5.1.1	Model	18
5.1.2	View	18

5.1.3	Controller	18
5.2	Struttura della WebApp	19
5.2.1	BackEnd	20
5.2.2	FrontEnd	21
6	Analisi dei risultati	23
6.1	Fase di test	23
6.1.1	Test N.1	24
6.1.2	Test N.2	25
6.1.3	Test N.3	26
	Bibliografia	28

Capitolo 1

Introduzione

1.1 Motivazione del progetto e Background

Le condizioni delle strade sono fondamentali per la sicurezza dei trasporti e la qualità della vita urbana. Danni come buche (1.1), crepe e deformazioni della pavimentazione non solo riducono il comfort di guida, ma contribuiscono significativamente a incidenti e rallentamenti, aumentando i costi di manutenzione per le autorità locali. Tuttavia, rilevare e gestire questi danni su larga scala è complesso,



Figura 1.1

poiché i metodi tradizionali richiedono manodopera e tecnologie costose come sensori ad alta precisione, poco sostenibili economicamente per molte amministrazioni [2].

Con l'aumento delle capacità di visione artificiale e apprendimento automatico, emerge la possibilità di automatizzare queste ispezioni tramite rilevamento dei danni su immagini stradali raccolte da veicoli comuni. Il progetto in esame si basa su questa idea e ha come obiettivo la realizzazione di un'applicazione web che identifica i tratti stradali danneggiati lungo un percorso specifico, come un'app di navigazione che segnala buche e crepe lungo il tragitto. La soluzione qui presentata sfrutta uno dei modelli di rilevamento dei danni stradali proposti nell'ambito della challenge CRDDC2022 organizzata dall'IEEE nel 2022 [1].

Background

Il contesto del progetto è legato alla sfida CRDDC (Crowdsensing-based Road Damage Detection Challenge), un'iniziativa dell'IEEE per lo sviluppo di metodi di rilevamento danni in immagini stradali raccolte in diverse condizioni geografiche. La challenge CRDDC2022 prevedeva un dataset composto da immagini stradali da sei paesi (Giappone, India, Repubblica Ceca, Norvegia, Stati Uniti e Cina), che comprendono circa 47.000 immagini e oltre 55.000 annotazioni di danni, suddivisi in categorie principali: crepe longitudinali, crepe trasversali, crepe a pelle di coccodrillo e buche [1]. Questa base di dati ha favorito lo sviluppo di modelli di visione artificiale in grado di generalizzare su diverse tipologie di strade e condizioni ambientali, puntando su una soluzione economica e accessibile per il monitoraggio stradale.

La CRDDC2022 ha incentivato l'uso di modelli di deep learning avanzati per il rilevamento e la classificazione automatica dei danni stradali. Le soluzioni proposte si sono basate principalmente su modelli di rilevamento oggetti a uno stadio (YOLO) e a due stadi (Faster-RCNN), poiché queste architetture offrono un ottimo compromesso tra velocità e precisione. La soluzione vincente ha utilizzato un approccio di ensemble learning che combina YOLO e Faster-RCNN: il modello YOLO è stato scelto per la sua capacità di rilevare rapidamente i danni, mentre il modello Faster-RCNN con backbone Swin Transformer ha migliorato la precisione, grazie a una maggiore capacità di estrazione delle caratteristiche semantiche. Questa combinazione di modelli, con tecniche di data augmentation e trasformazioni per l'ottimizzazione delle predizioni, ha permesso di ottenere elevati F1-score nelle valutazioni della challenge [2].

La combinazione dei modelli permette una gestione ottimale sia dei dettagli locali (per il rilevamento di danni minori) sia dei danni più evidenti, garantendo una copertura ampia ed efficiente delle diverse tipologie di danni stradali lungo il percorso dell'utente. Tuttavia, per il progetto è stato adottato un modello di rilevamento ispirato all'approccio one-stage, utilizzando una versione di YOLOv7 adattata ai dati specifici dell'applicazione e ottimizzata tramite tecniche di data augmentation. Il sistema, integrato nell'applicazione web, elabora le immagini e segnala i danni, visualizzando le immagini di strade che presentano tratti danneggiati.

1.2 Stato dell'arte

Negli ultimi anni, la rilevazione automatica dei danni stradali tramite visione artificiale è emersa come una soluzione efficace e scalabile rispetto ai metodi tradizionali, che richiedono ampie risorse umane e costi elevati. Questa tecnica si è consolidata grazie all'uso di modelli di deep learning come YOLO e Faster R-CNN, che sono in grado di individuare e classificare diverse tipologie di danni stradali (buche, crepe longitudinali, crepe trasversali, crepe a pelle di coccodrillo). Entrambi questi modelli si sono rivelati fondamentali nel contesto della Crowdensing-based Road Damage Detection Challenge (CRDDC), organizzata dall'IEEE [4], la quale ha sviluppato e reso disponibili dataset come RDD2022 per il miglioramento di algoritmi di rilevamento robusti e versatili.

Lo stato dell'arte si è evoluto ulteriormente grazie al contributo di progetti open-source, come quello descritto nel repository GitHub sopra menzionato, che sfrutta YOLOv7 per un rilevamento accurato dei danni. In particolare, questi progetti integrano pipeline end-to-end per l'elaborazione in tempo reale di immagini e video, offrendo al contempo soluzioni per l'implementazione di server locali e applicazioni web. Il progetto utilizza il modello YOLOv7 fine-tunato sul dataset RDD2022 per il rilevamento di quattro tipologie di danni, dimostrando come modelli pre-addestrati possano essere personalizzati per scenari locali e testati con GPU modeste (ad esempio, su GPU RTX2060) per ottenere buoni risultati di precisione e richiamo.

Nel complesso, l'avanzamento dei modelli basati su YOLO ha consentito una maggiore efficienza e accuratezza nella rilevazione di danni stradali, permettendo il monitoraggio su larga scala e a basso costo. Questi approcci non solo rappresentano una soluzione innovativa per la manutenzione delle infrastrutture, ma abilitano anche applicazioni in contesti di smart city e trasporto intelligente, rendendo pratico e immediato il monitoraggio stradale.

Capitolo 2

Requisiti funzionali e non funzionali

In questo capitolo vengono analizzati i requisiti del sistema utilizzando il modello FURPS+, un framework che ci permette di suddividere i requisiti in diverse categorie per una trattazione completa e organizzata. Il modello FURPS+ classifica i requisiti in Funzionalità, Usabilità, Affidabilità, Prestazioni e Supportabilità, con l'aggiunta di altre caratteristiche come la manutenibilità e l'estensibilità. Que-

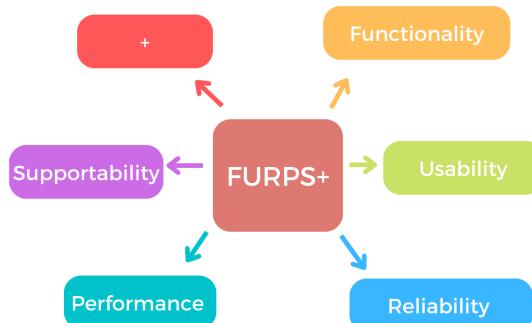


Figura 2.1: FURPS+

sta struttura consente di descrivere i requisiti in modo dettagliato, facilitando la comprensione e la futura implementazione del sistema. Nei paragrafi successivi, i requisiti saranno elencati e descritti secondo le categorie di questo modello.

2.1 Requisiti funzionali

- Operare sul percorso inserito dall'utente

Il sistema deve essere in grado di prendere in input le località di partenza e destinazione di interesse per l'utente, e fornire informazioni riguardo lo stato delle strade appartenenti al percorso consigliato da Google Maps.

- Ottenimento automatico delle immagini

L'applicazione deve essere in grado di reperire automaticamente le immagini delle strade lungo il percorso a partire dalle località di partenza e destinazione.

- Rilevamento danni da immagini

Il sistema deve essere in grado di analizzare le immagini ottenute per rilevare e classificare automaticamente danni stradali, come crepe e buche. Utilizzando algoritmi di deep learning, ad esempio YOLOv7, l'applicazione identifica e classifica in tempo reale i danni, supportando diversi tipi di input e formati multimediali.

- Classificazione per tipo di danno

I danni rilevati vengono classificati in base alla tipologia (crepe longitudinali, buche, ecc.), migliorando la navigabilità e l'analisi dei dati da parte dell'utente. Questo permette di focalizzarsi su danni specifici, ad esempio per analisi di sicurezza o gestione della manutenzione.

- Integrazione con API di Navigazione

Interagendo con le API di Google Maps e Google Street View, il sistema può sfruttare informazioni di navigazione e immagini utili al raggiungimento dello scopo.

- Specifica del luogo e del periodo di ciascuna immagine

Ai fini di un'ottimale interpretazione e usabilità dei risultati ottenuti, è importante che sia chiaro ottenere informazioni sul luogo esatto in cui è stato rilevato il danno e al periodo in cui risale la fotografia; quest'ultimo punto è importante per dare la possibilità ad un ente che si occupa di manutenzione di verificare se vi sono stati lavori o meno in seguito alla data ottenuta.

- Reportistica e Esportazione Dati

Il sistema genera report sui danni rilevati, con possibilità di esportazione in formati standard (CSV) per documentazione o analisi.

2.2 Requisiti Non funzionali URPS+

- Usabilità (Usability)

L'esperienza utente deve essere intuitiva, con un'interfaccia user-friendly. Tutorial e guide integrate devono assistere gli utenti nelle operazioni principali, favorendo l'accessibilità anche per utenti meno esperti.

- **Affidabilità (Reliability)**

Il sistema deve essere disponibile e affidabile, con un uptime elevato e meccanismi di backup regolari. Questo garantisce che gli utenti possano accedere all'applicazione in ogni momento.

- **Prestazioni (Performance)**

Il sistema deve rispondere rapidamente, con tempi di risposta ottimali per il caricamento, rilevamento e visualizzazione dei danni. Il tempo medio di rilevamento per immagini e video deve essere ridotto per mantenere un'esperienza utente fluida.

- **Supportabilità (Supportability)**

Manutenibilità: l'architettura dell'applicazione deve essere modulare per facilitare la manutenzione e gli aggiornamenti futuri; portabilità: il sistema deve poter essere facilmente trasferito o replicato in diversi ambienti server o piattaforme, permettendo flessibilità e facilitando future migrazioni.

- **Altri Fattori (+):**

- Scalabilità: L'architettura deve essere scalabile, in grado di supportare un numero crescente di utenti e operazioni senza degradare le performance. Questo è essenziale per gestire carichi variabili, con il supporto di infrastrutture cloud per l'espansione elastica.
- Logging e Monitoraggio: L'applicazione deve includere un sistema di logging dettagliato e strumenti di monitoraggio delle performance, per identificare e risolvere velocemente eventuali anomalie. Ciò migliora la gestione dell'infrastruttura e l'efficienza operativa complessiva.
- Compatibilità Multi-Dispositivo: L'interfaccia dell'applicazione deve essere completamente responsiva, supportando una varietà di dispositivi (desktop, tablet, smartphone) senza comprometterne la funzionalità.

Capitolo 3

Workflow

3.1 Fase iniziale

3.1.1 Analisi del problema e delle soluzioni proposte

L’analisi dello stato delle strade rappresenta un elemento fondamentale per garantire la sicurezza, l’efficienza e la sostenibilità della mobilità sia urbana che extraurbana.

Nella prima fase sono stati esaminati diversi approcci per il monitoraggio delle condizioni stradali [1]. Inizialmente, è stata analizzata l’efficacia delle ispezioni manuali, evidenziandone l’elevato dispendio di tempo e risorse, nonché la loro limitata capacità di garantire una copertura costante. Successivamente, si è passati alla valutazione di sistemi basati su sensori installati su veicoli specializzati, che, pur offrendo un monitoraggio più automatizzato, richiedono una complessa rete logistica e presentano difficoltà nel raggiungere tutte le aree con capillarità. Infine, è stato analizzato l’unico servizio web attualmente disponibile [10] per l’individuazione dei danni stradali, constatando le sue limitazioni, in quanto l’utente deve caricare manualmente le immagini da analizzare.

3.1.2 Approccio proposto

Si propone un approccio innovativo per il monitoraggio e la classificazione dei danni stradali in maniera rapida e automatizzata. Nel progetto in esame, l’utente è tenuto unicamente a inserire le località di partenza e di destinazione, mentre la webapp provvede a ottenere in automatico le immagini da Google Street View, che vengono successivamente analizzate e classificate mediante un modello di machine learning. Oltre a ottimizzare le attività di manutenzione stradale, tale tecnologia ha il potenziale di migliorare i servizi offerti da Google Maps, introducendo la possibilità per gli utenti di evitare percorsi danneggiati, migliorando così l’esperienza di guida e riducendo il rischio di danni al proprio veicolo.

3.2 Fase di determinazione dei requisiti

Una volta individuato l’obiettivo del progetto, e avendo analizzato i punti di forza e i punti di debolezza delle soluzioni già fornite, si è passati alla fase di determinazione dei requisiti. Per quanto riguarda i requisiti funzionali, l’applicazione deve permettere all’utente di inserire le località di partenza e di destinazione attraverso un’interfaccia intuitiva. Il sistema calcolerà automaticamente il percorso migliore tra le località indicate e scaricherà un certo numero di immagini raffiguranti porzioni del percorso in questione. Di seguito, analizzerà le immagini per rilevare eventuali danni infrastrutturali. In output devono essere fornite informazioni sulla tipologia, gravità, posizione e data dei danni, con l’inclusione di immagini dei punti danneggiati rilevati.

Per quanto riguarda i requisiti non funzionali, è stato deciso di focalizzarsi su un’interfaccia utente che garantisca massima usabilità, mantenendola semplice e accessibile per tutti gli utenti. L’integrazione con le API di Google Maps assicura che le immagini fornite siano allo stato più aggiornato possibile fra le varie opzioni attraverso le quali è possibile ottenere fotografie delle strade. Per ottimizzare le prestazioni del sistema, è stato scelto un modello che garantisse un buon equilibrio tra accuratezza e velocità di esecuzione, evitando l’adozione di modelli più lenti, anche se maggiormente performanti. La supportabilità e la manutenibilità del sistema sono state assicurate attraverso un’architettura modulare, che consente aggiornamenti e interventi di manutenzione con facilità.

3.3 Fase di ricerca del modello

Il modello utilizzato nell’applicazione è stato selezionato tra i migliori modelli classificati alla Crowdsensing-based Road Damage Detection Challenge (CRDDC2022) organizzata dall’IEEE. Questo concorso ha rappresentato un punto di riferimento internazionale per la ricerca di soluzioni innovative nel rilevamento dei danni stradali, stimolando la realizzazione di modelli di machine learning avanzati e dimostrando l’efficacia delle tecniche più all’avanguardia nel campo della visione artificiale.

Tratti comuni individuati nei modelli classificati ai primi posti sono l’utilizzo di reti neurali convoluzionali (CNN) avanzate, con architetture ottimizzate per catturare dettagli nelle immagini, e adozione di metodi di preprocessing e data augmentation per migliorare le prestazioni e la capacità di generalizzare.

La ricerca e l’utilizzo del modello più adeguato per il task di rilevamento dei danni stradali si sono rivelate particolarmente complesse, in quanto molti modelli disponibili online non fornivano una chiara descrizione delle operazioni di preprocessing necessarie per l’inferenza. Questa lacuna ha reso difficile replicare accuratamente i risultati, anche quando i pesi erano disponibili, a causa dell’incertezza sulle

trasformazioni applicate ai dati di input e altri dettagli tecnici. Ulteriori complicazioni si sono presentate con le prestazioni dei modelli: alcuni si distinguevano per un'elevata accuratezza, ma i tempi di inferenza erano troppo lenti per applicazioni pratiche, mentre altri modelli, pur essendo più rapidi, non garantivano risultati soddisfacenti in termini di precisione. Dopo un'attenta analisi comparativa, è stato selezionato il modello sviluppato dal team "ShiYu_SeaView", che ha dimostrato un equilibrio ottimale tra accuratezza e velocità di esecuzione, soddisfacendo i requisiti del progetto.

Rank	Team Name	LeaderBoard - 1 (overall 6 countries)	LeaderBoard - 2 (India)	LeaderBoard - 3 (Japan)	LeaderBoard - 4 (Norway)	LeaderBoard - 5 (United States)	Average Score
1	ShiYu_SeaView	0.770	0.583	0.789	0.595	0.844	0.716
2	DongjunJeong	0.743	0.540	0.750	0.538	0.801	0.674
3	MDPT	0.741	0.516	0.735	0.504	0.817	0.663
4	SGG-RS-Group	0.727	0.545	0.727	0.481	0.779	0.652
5	IRCV-URV	0.726	0.518	0.735	0.498	0.779	0.651
6	IMSC	0.728	0.542	0.725	0.478	0.774	0.649
7	NJUPT	0.718	0.559	0.720	0.458	0.743	0.640
8	TUT	0.694	0.519	0.773	0.464	0.727	0.636
9	MILA	0.697	0.494	0.716	0.462	0.775	0.629
10	SIAI	0.612	0.417	0.608	0.424	0.663	0.545
11	kubapok	0.603	0.421	0.594	0.383	0.649	0.530

Figura 3.1: Top 11 teams for CRDCC'2022

3.4 Scelta di realizzare una webapp con Flask e scelta dei servizi esterni

Per raggiungere l'obiettivo, è stato scelto di realizzare una webapp utilizzando il framework Flask, che offre una soluzione semplice e flessibile per sviluppare applicazioni web. Flask consente di gestire facilmente le richieste HTTP e di integrare diverse funzionalità tramite API esterne. Si è scelto di adottare le API di Google Street View in quanto offrono un modo efficace per ottenere immagini aggiornate e personalizzabili di mappe statiche o dinamiche, permettendo di integrare facilmente visualizzazioni geografiche dettagliate nelle applicazioni, migliorando l'esperienza utente. Tuttavia, sono state riscontrate difficoltà nella configurazione dei parametri richiesti dalle Static API di Google Street View, in quanto non vi è un parametro che permetta di ottenere un'immagine frontale rispetto al P.O.V., quindi è stato necessario settare i parametri in modo che funzionino bene la maggior parte delle volte. Inoltre, è stato utilizzato OpenStreetMap (OSM) per ottenere tutti i punti di un percorso. La combinazione di Flask, Google Maps e OSM ha permesso di sviluppare una webapp robusta e versatile, in grado di rispondere alle esigenze del progetto.

3.5 Fase di scelta dell'architettura

Nel processo di sviluppo della webapp, la scelta di adottare l'architettura MVC (Model-View-Controller) è stata determinante per gestire l'interazione tra le API di Street View e OSM, il modello di machine learning per la detection e l'interfaccia utente.

Dopo un'analisi dei requisiti, si è reso evidente che un'architettura modulare era necessaria per coordinare in modo efficiente la raccolta di immagini attraverso le API, l'elaborazione tramite il modello e la visualizzazione dei risultati.

MVC si è dimostrato ideale per separare queste componenti chiave: il modello gestisce le chiamate alle API esterne e al sistema di inferenza, il controller coordina il flusso di dati e richieste, e la vista presenta i risultati in modo intuitivo e user-friendly.

Questo approccio non solo ha semplificato la gestione del codice e la possibilità di aggiornamenti, ma ha anche garantito la flessibilità necessaria per integrare nuove funzionalità e ottimizzare le prestazioni.

La scelta di adottare il pattern MVC (Model-View-Controller) nella webapp per il rilevamento dei danni stradali è stata ulteriormente facilitata dall'uso di Flask, un framework noto per la sua compatibilità con tale architettura, ideale per suddividere le responsabilità tra l'acquisizione delle immagini tramite le API di Street View, l'inferenza del modello di machine learning e la presentazione dei risultati.

Grazie a questa configurazione, il controller di Flask gestisce il flusso di richieste e risposte, coordinando le operazioni tra il modello e la vista in modo efficace.

3.6 Fase di sviluppo con metodologia DevOps

Il processo di sviluppo della webapp è stato affrontato seguendo una metodologia DevOps, mirata a garantire un ciclo di vita del software continuo, efficiente e collaborativo. Durante la fase di sviluppo, sono state integrate pratiche di integrazione e distribuzione continua (CI/CD) per automatizzare i processi di testing e allineare tutti i membri del gruppo alla versione più aggiornata del codice. È stato utilizzato Git per il controllo di versione e la gestione del codice, garantendo una collaborazione efficace e una tracciabilità dettagliata delle modifiche. La webapp è stata costantemente testata per garantire stabilità e prestazioni; il feedback raccolto da queste iterazioni rapide ha permesso una collaborazione sinergica tra i componenti del team, riducendo i tempi di avanzamento e migliorando la qualità complessiva del prodotto finale.

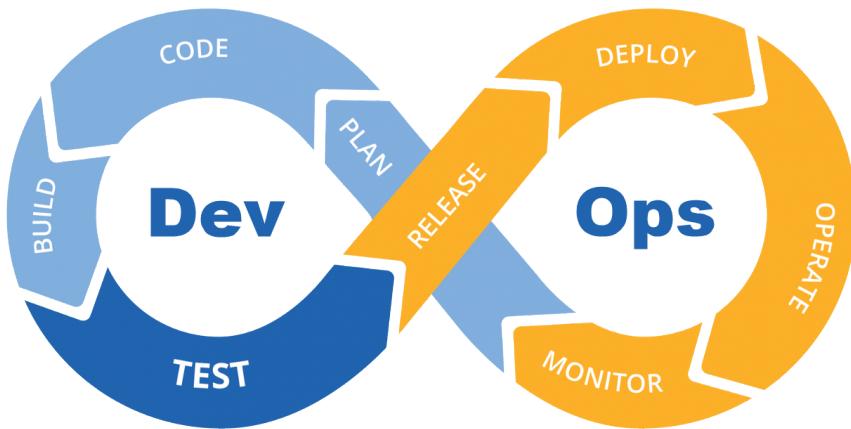


Figura 3.2: Processo di sviluppo DevOps

3.7 Fase di test

Il Test Driven Development (TDD) ha costituito la base dell'approccio al testing, garantendo che ogni componente della webapp fosse sviluppato e testato in modo iterativo e affidabile. Questa metodologia ha permesso al team di affrontare la complessità della soluzione, attraverso un processo multilivello che include test automatici, test manuali e monitoraggio continuo in fase di produzione, assicurando che il sistema fosse efficiente e conforme ai requisiti del progetto. Infine, l'applicazione è stata testata su 3 casi di test su alcuni percorsi della città di Bari.

Capitolo 4

Tecnologie utilizzate

Per il raggiungimento dell'obiettivo si è scelto di implementare un'applicazione web, la quale prende come input un indirizzo di partenza e uno di destinazione, e restituisce informazioni sullo stato delle strade interessate dal percorso che congiunge i due punti. Si illustrano di seguito le tecnologie adottate per la realizzazione dell'applicazione.

4.1 Modello YOLOv7x_640

Il modello di machine learning selezionato [9] per effettuare l'operazione di individuazione e classificazione dei danni stradali di un'immagine è "YOLOv7x_640" una variante di YOLO versione 7 fine-tunata sul dataset fornito dagli organizzatori della challenge mediante il transfer learning.

4.1.1 Yolo

YOLO (You Only Look Once) [5] è un modello di deep learning utilizzato per object detection, ovvero l'identificazione e la localizzazione di oggetti in un'immagine o in un video. Questo modello appartiene alla famiglia degli "One-stage" detectors, infatti a differenza di approcci che utilizzano pipeline a più stadi per la rilevazione degli oggetti, YOLO affronta il problema in un solo passaggio, rendendolo estremamente veloce e adatto per applicazioni che richiedono una bassa latenza.

Architettura di YOLO

Yolo si basa su una rete neurale convoluzionale (CNN) che divide l'immagine di input in una griglia, dove ogni cella della griglia è responsabile della previsione di un numero fisso di bounding box e delle probabilità di classe per gli oggetti che possono trovarsi in quelle box. Le caratteristiche principali della sua architettura sono:

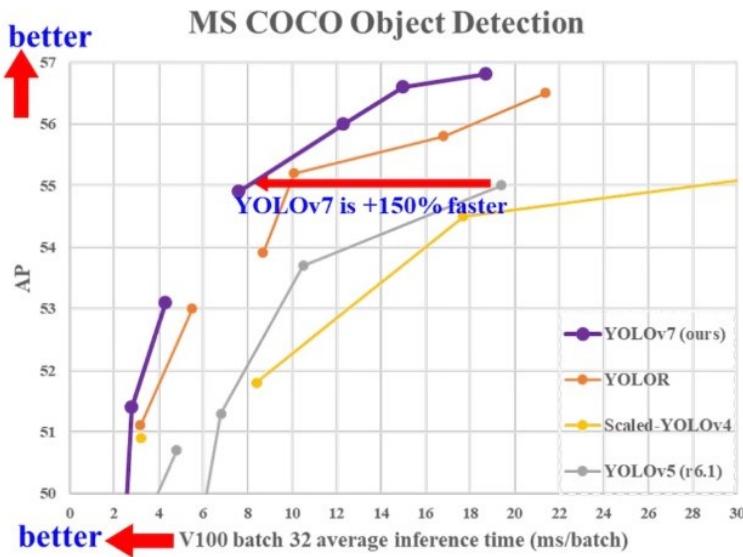


Figura 4.1: Performance di Yolov7

1. Convoluzioni iniziali per estrarre caratteristiche dalle immagini, seguite da diversi livelli di convezione e pooling.
2. Output Layer. Ogni cella nella griglia genera un'uscita contenente:
 - Coordinate del box dell'oggetto individuato (x, y, ossia altezza e larghezza).
 - Punteggio di confidenza, che indica la probabilità che il box contenga un oggetto e la precisione del box stesso.
 - Previsione della classe dell'oggetto contenuto (ad esempio, persona, auto, bicicletta).
3. Loss Function che combina tre componenti:
 - Errore delle coordinate
 - Errore di classificazione
 - Errore del punteggio di confidenza

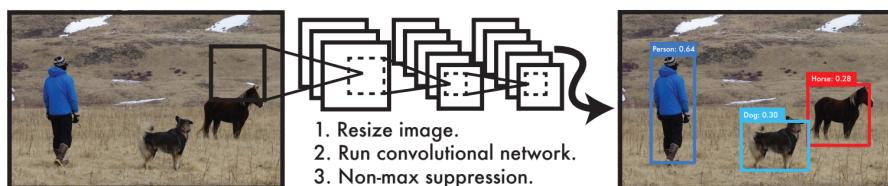


Figura 4.2: Esempio di individuazione e classificazione con Yolo

4.1.2 Transfer learning

Il transfer learning è una tecnica del machine learning in cui un modello pre-addestrato su un certo compito viene riutilizzato come base per un altro compito correlato. L'idea centrale è quella di sfruttare le conoscenze apprese in un contesto per trasferirle e applicarle a un altro, solitamente utilizzando una quantità limitata di nuovi dati di addestramento. Il task che questa applicazione affronta è la localizzazione e classificazione di danni stradali, pertanto il modello pre-addestrato è stato fine-tunato per delle epoche ulteriori sul dataset fornito.

4.2 Google Cloud Platform

Google Cloud Platform (GCP) [3] è una suite di servizi cloud offerti da Google per aiutare gli sviluppatori a costruire, implementare e gestire applicazioni attraverso l'infrastruttura di Google. Ciascuno dei servizi offerti espone delle API per potervi interagire. Le interfacce dei servizi utilizzati sono:

- Geocoding API: permette di ottenere informazioni dettagliate su un indirizzo o un punto specifico, come latitudine, longitudine, città, partendo da un indirizzo fisico o viceversa.
- Maps JavaScript API: consente di visualizzare mappe interattive su una pagina web e fornisce suggerimenti per l'autocompletamento nell'inserimento dei nomi delle località.
- Street View Static API: permette di richiedere immagini panoramiche statiche (immagini fisse) di un luogo specifico, utilizzando le coordinate geografiche (latitudine e longitudine). È possibile personalizzare la richiesta per includere dettagli come l'orientamento della fotocamera, il campo visivo e la risoluzione dell'immagine.

4.2.1 Interazione con Google Cloud Platform

Le API citate sono esempi di REST API (Representational State Transfer API), un tipo di architettura di servizi web che consente di accedere a risorse tramite protocollo HTTP, utilizzando metodi standard come GET, POST, PUT e DELETE. Di seguito sono elencati i tratti caratteristici di queste API.

- HTTP come protocollo di comunicazione: le risorse (ad esempio, dati su luoghi, mappe o immagini di Street View) sono accessibili tramite richieste HTTP. Ogni richiesta inviata al server contiene un URL che definisce la risorsa a cui si desidera accedere, insieme a parametri specifici del servizio richiesto. In particolare si utilizza il metodo GET per recuperare dati dalle API.

- Formato dei dati: le risposte di tipo testuale sono generalmente fornite in formato JSON, che è un formato di dati leggero e facile da interpretare. Nel caso della Street View Static API, invece, la risposta può essere un’immagine in un formato come JPEG o PNG.
- Autenticazione e chiavi API: per poter interagire con le API dei servizi di GCP è necessario avere una chiave API, che funge da sistema di autenticazione per identificare l’applicazione che sta facendo la richiesta. La chiave viene inviata fra i parametri nell’URL della richiesta.
- EndPoint: ciascun servizio espone uno o più endpoint a cui inviare le richieste. Ad esempio, per ottenere informazioni su un indirizzo specifico, si invia una richiesta all’ endpoint <https://maps.googleapis.com/maps/api/geocode/json> della Geocoding API, mentre per ottenere un’immagine di Street View, si invia una richiesta all’endpoint <https://maps.googleapis.com/maps/api/streetview> della Street View Static API.

4.3 Open Street Map API

OpenStreetMap (OSM) [7] è una piattaforma di mappatura collaborativa che fornisce dati geografici liberi e accessibili. Le API di OpenStreetMap permettono agli utenti di interagire con questi dati in vari modi, come la creazione, la modifica, la ricerca e la visualizzazione di informazioni geografiche. OpenStreetMap non fornisce direttamente una API di routing, ma esistono implementazioni di terze parti che utilizzano i dati di OSM per il calcolo dei percorsi, come

- OSRM (Open Source Routing Machine): un motore di routing che permette di calcolare percorsi tra due o più punti, ritornando la lista dei punti che è necessario attraversare per giungere dal punto A al punto B. Un grande punto a favore di questo servizio è la semplicità di utilizzo e la bassa latenza, che lo ha reso preferibile agli analoghi servizi offerti da Google Cloud Platform.

4.4 Flask

Flask [8] è un framework per Python che permette di creare applicazioni web in modo semplice e modulare. È un framework WSGI (Web Server Gateway Interface) che fornisce solo le basi per costruire un’applicazione web, senza includere un set di funzionalità predefinite come altri framework più grandi (ad esempio, Django). Questo approccio consente agli sviluppatori di scegliere liberamente le librerie o gli strumenti da integrare nel progetto, mantenendo alta la flessibilità e la personalizzazione. Flask si occupa dell’interazione tra il backend e il frontend dell’applicazione: in particolare, gestisce le richieste in arrivo dal client, lanciando

azioni lato server al verificarsi di eventi sul lato client, come l'invio di un modulo o il clic su un pulsante. Successivamente, Flask invia una risposta dal server al client, che può consistere in una nuova pagina HTML, un aggiornamento dinamico tramite AJAX o altre informazioni necessarie per interagire con l'utente. Questo modello di funzionamento rende Flask ideale per progetti di piccole e medie dimensioni, dove la semplicità e la personalizzazione sono prioritari. Di seguito si illustrano le principali caratteristiche di Flask.

- Utilizzo di Werkzeug, una libreria WSGI che fornisce un set di utility per la gestione delle richieste HTTP, come il parsing degli URL, il routing, la gestione delle sessioni, la gestione delle risposte e la gestione dei cookie.
- Flask fornisce una gestione degli URL tramite il sistema di routing, uno dei servizi principali di Flask. Questa funzionalità permette di mappare gli URL a funzioni Python eseguite lato server, gestendo come l'applicazione risponde a richieste HTTP su determinati percorsi. In un'applicazione Flask, ogni volta che un client invia una richiesta, il framework esegue una funzione associata alla route corrispondente. Tuttavia, quando un'applicazione cresce, il routing può diventare complesso e difficile da gestire se tutto è definito in un unico file o modulo. Flask Blueprint è uno degli strumenti che il framework offre per organizzare e modularizzare il codice di routing. Un Blueprint è una sorta di "piano" per una sezione dell'applicazione, che definisce le route, le funzioni e persino la logica di gestione dei template, ma che non è legato direttamente all'oggetto Flask fino a quando non viene registrato.
- Jinja2 è il motore di template di Flask; esso permette di generare dinamicamente l'HTML che verrà restituito al client. Con Jinja2, è possibile inserire variabili Python all'interno dei template HTML, implementare logica condizionale (come loop e if), e separare la logica del programma dal design dell'interfaccia utente.



Figura 4.3: Flask

Capitolo 5

Architettura e struttura del progetto

In questo capitolo si descrivono l'architettura e la struttura del progetto, con un focus sull'applicazione del pattern Model-View-Controller (MVC) [6] e sulle scelte progettuali adottate. Verrà inoltre illustrata la suddivisione della WebApp tra FrontEnd e BackEnd, analizzando le tecnologie impiegate per garantire efficienza e manutenibilità.

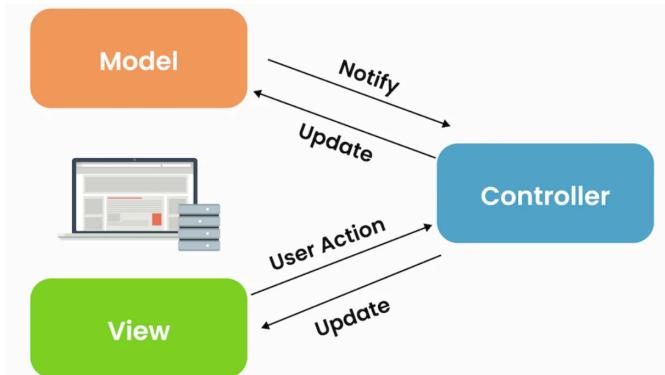


Figura 5.1: MVC pattern

5.1 Pattern MVC

Il pattern Model-View-Controller (MVC) è un'architettura software ampiamente utilizzata per organizzare il codice di applicazioni complesse, suddividendole in tre componenti distinti. Model rappresenta la logica di gestione dei dati, occupandosi della loro creazione, manipolazione e memorizzazione. View è responsabile della presentazione dei dati, gestendo l'interfaccia utente e il rendering delle informazioni. Infine, il Controller funge da intermediario tra Model e View, ricevendo le richieste degli utenti, elaborandole. Questa suddivisione consente una gestione più

chiara del codice, facilitando la manutenibilità, l'espandibilità e la collaborazione tra sviluppatori, grazie alla netta separazione delle responsabilità.

5.1.1 Model

Il Model rappresenta il livello dell'applicazione dedicato alla gestione dei dati e della logica. È responsabile della comunicazione con il database o altri servizi per memorizzare e recuperare le informazioni necessarie. Nel progetto, il Model è stato progettato per essere indipendente dalla View, garantendo una chiara separazione delle responsabilità e facilitando la gestione e la validazione dei dati. Nel caso specifico del nostro progetto, la logica dei dati è suddivisa in due file Python principali: obtain_images.py e inference.py. Il file obtain_images.py gestisce il recupero di coordinate geografiche utilizzando l'API di Google Geocoding e il download di immagini da Google Street View per analisi successive. Utilizza inoltre OSRM per ottenere percorsi dettagliati, ottimizzando la gestione delle immagini da processare. Il file inference.py si occupa invece dell'inferenza delle immagini tramite il modello YOLOv7 per il rilevamento di danni. Questa architettura assicura una gestione efficiente dei dati, suddividendo chiaramente le responsabilità tra acquisizione e analisi delle immagini.

5.1.2 View

La View si occupa della presentazione dei dati agli utenti finali, gestendo l'interfaccia grafica e la visualizzazione delle informazioni. È progettata per ricevere i dati dal Model tramite il Controller e visualizzarli in modo coerente e intuitivo. Nel nostro progetto, la View è stata realizzata con un focus sull'usabilità e la reattività, utilizzando tecnologie moderne per un'interazione fluida e dinamica. La View in Flask si occupa della presentazione dei dati e della gestione delle risposte HTML verso il client. Viene gestita attraverso l'utilizzo di pagine html definite templates, le quali vengono generate dinamicamente utilizzando il motore di template Jinja2, inserendo i dati forniti dal Model. Le View sono progettate per essere semplici e intuitive, rendendo facile la navigazione dell'applicazione e migliorando l'esperienza utente. Flask rende possibile una gestione flessibile delle route, permettendo di collegare le View alle azioni definite nel Controller.

5.1.3 Controller

Il Controller agisce come intermediario tra il Model e la View, gestendo le richieste dell'utente, elaborandole e determinando quale azione intraprendere. Coordina la logica dell'applicazione, invocando il Model per l'elaborazione dei dati e aggiornando la View con le informazioni corrette. Nel progetto, il Controller è stato implementato per garantire un flusso di dati efficiente e per mantenere una logica

chiara e ben strutturata, migliorando così la manutenibilità e l'estensibilità del sist. Il Controller in Flask è implementato attraverso le route che gestiscono le richieste HTTP. Queste route sono definite nel codice dell'applicazione per elaborare le richieste degli utenti, invocare i metodi del Model per ottenere o manipolare i dati, e quindi restituire la risposta appropriata tramite le View. Flask facilita l'organizzazione del codice del Controller, offrendo una gestione chiara delle route e un controllo completo sul flusso delle richieste. Inoltre flask permette una gestione modulare dei controller attraverso l'utilizzo dei blueprint, questo rende più semplice la gestione di applicazioni grandi o complesse.

5.2 Struttura della WebApp

La WebApp fornisce un'interfaccia interattiva, la quale consente all'utente di analizzare lo stato di una strada di un qualsiasi percorso. Per quanto riguarda lo sviluppo della webapp sia frontend che backend sono stati realizzati utilizzando flask. In questa sezione, esploreremo la struttura di entrambe le componenti.

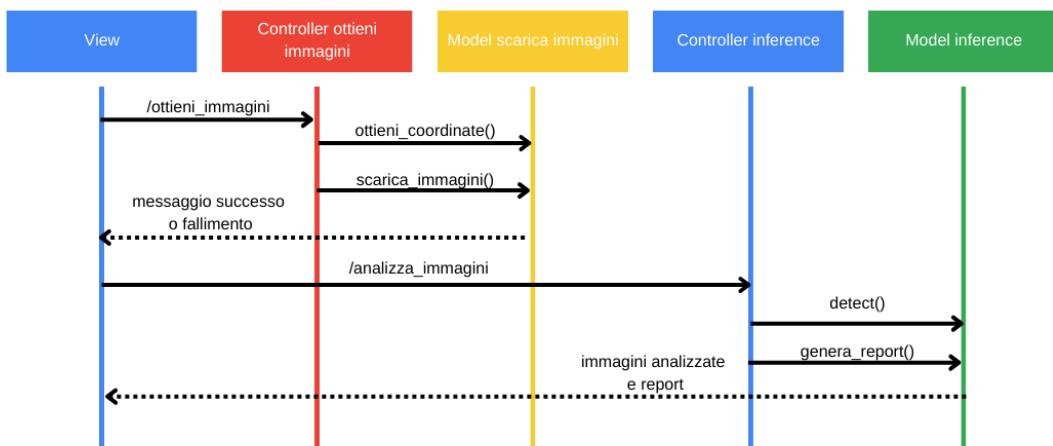


Figura 5.2: Sequence diagram UML

5.2.1 BackEnd

La struttura del backend è organizzata per garantire una separazione chiara tra i vari componenti, facilitando la manutenzione e l'espansione del sistema. In questo caso Flask risulta essere particolarmente utile in quanto permette di scompattare il controller in file separati chiamati blueprint ognuno dei quali dedicato ad un endpoint specifico; di seguito analizzeremo le funzionalità di ogni singolo componente.

Endpoint esposti:

- “/“ (Route principale): L'endpoint di base dell'applicazione, che gestisce la pagina principale. Fornisce un'interfaccia per l'inserimento delle informazioni necessarie e permette all'utente di interagire con le funzionalità del sistema, come l'invio di richieste per ottenere immagini o per analizzare i danni.
- ‘/ottieni_immagini’: questo endpoint è gestito da un controller dedicato che riceve due parametri: "partenza" e "destinazione". Il controller utilizza un model specifico per recuperare le coordinate utilizzando le API di Google Maps. Una volta ottenute le coordinate, per determinare il percorso, viene sfruttato OpenStreetMap, grazie al quale, attraverso una funzione specifica ci siamo calcolati i punti intermedi sul percorso. Successivamente, riutilizziamo le API di Google Street View per ottenere un numero totale di immagini pari al numero di punti intermedi. Le immagini ottenute vengono memorizzate localmente per essere analizzate successivamente.
- ‘/analizza_immagini’: anche questo endpoint dispone di un controller e un model dedicato. Il suo compito è analizzare le immagini ottenute dall'endpoint precedente utilizzando un modello di machine learning, identificando così eventuali danni stradali presenti. Il funzionamento del modello e la sua architettura sono stati analizzati nel capitolo 4.3. Una volta completata l'analisi, i risultati vengono restituiti come risposta alla chiamata. Nello specifico in risposta saranno inviati le immagini sulle quali sono stati rilevati danni stradali e un'analisi con tutti i danni rilevati.

5.2.2 FrontEnd

Il frontend dell'applicazione è stato sviluppato utilizzando la struttura dei template di Flask. Questo approccio consente di organizzare il codice HTML in modo modulare e riutilizzabile.

Il sistema di template di Flask utilizza file HTML con blocchi dinamici, che vengono riempiti con dati provenienti dal backend. I template principali sono strutturati in modo gerarchico, sfruttando l'estensione di altri template per evitare la duplicazione del codice.

L'approccio utilizzato è stato quello di avere tutta l'interfaccia su di una single page, in seguito verrà analizzata la struttura della pagina.

- ‘index.html’: questo è il template principale, che definisce l’interfaccia iniziale dell’applicazione. La struttura è semplice, con una singola colonna sulla sinistra dove l’utente può inserire il percorso desiderato specificando un punto di partenza e uno di destinazione. L’uso dei template consente di inserire blocchi vuoti (ad esempio, ‘`{% block obtain_images %}{% endblock %}`’) che possono essere popolati successivamente con contenuti specifici da altri template.

The screenshot shows a web form titled "Inserire partenza e destinazione". It has two input fields: "Partenza" (Departure) and "Destinazione" (Destination), both labeled "Inserire una città di partenza" (Insert a city of departure) or "Inserire una città di destinazione" (Insert a city of destination). Below the fields is a blue button labeled "Ottieni immagini" (Get images).

Figura 5.3: index.html

- ‘obtain_images.html’: questo template estende ‘index.html’ e il suo compito è quello di popolare il blocco omonimo, fornendo all’utente un messaggio relativo all’ottenimento delle immagini, come conferme di successo o errore. Oltre al messaggio nel blocco viene inserito anche un pulsante per avviare l’analisi delle immagini, aggiungendo funzionalità interattive all’interfaccia. Se il recupero delle immagini non è andato a buon fine il pulsante sarà nascosto.

Inserire partenza e destinazione

Partenza:

Destinazione:

Ottieni immagini

Coordinate e immagini salvate con successo!

Avvia Analisi

Figura 5.4: obtain_images.html

- ‘inference.html’: questo template è responsabile della visualizzazione dei risultati ottenuti. Nello specifico, genera un report sotto forma di istogramma, posizionato nella colonna di sinistra, che mostra quanti danni sono stati rilevati per ciascuna categoria. Oltre al report, viene popolato tutto lo spazio sulla destra, responsabile della visualizzazione delle immagini del percorso scelto. E’ presente un’intestazione che indica le località di partenza e destinazione del percorso a cui appartengono i risultati, e un carosello che consente all’utente di visualizzare le immagini analizzate, nelle quali sono stati rilevati danni corrispondenti a una o più categorie, arricchite con informazioni sul luogo e la data in cui è stata scattata la fotografia; le immagini sono ordinate in ordine decrescente per gravità dello stato stradale. Per chiarezza e per una maggior velocità del sistema le immagini senza nessun danno rilevato non vengono visualizzate all’interno del carosello.

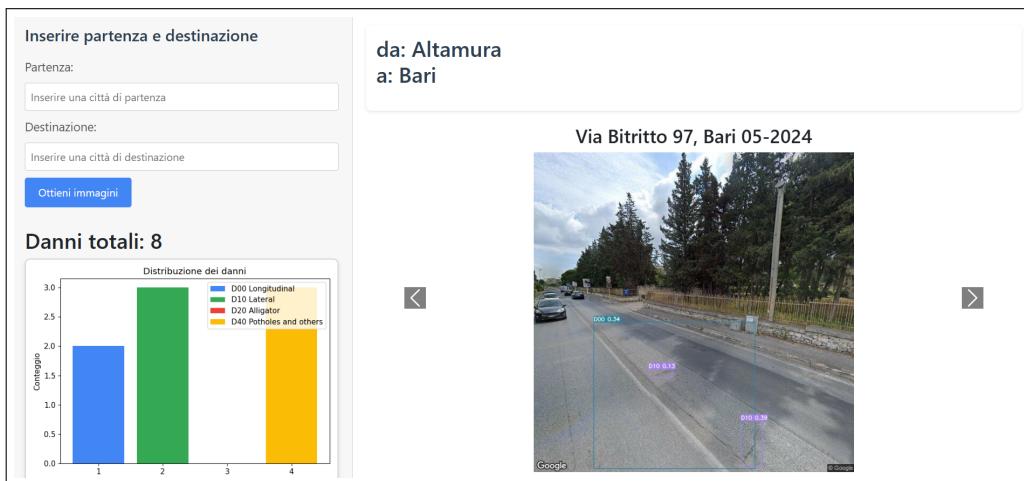


Figura 5.5: obtain_images.html

Capitolo 6

Analisi dei risultati

In questo capitolo verranno analizzati vari casi di test, effettuati su alcuni percorsi della città di Bari, in particolare è stata analizzata la zona circostante il campus del Politecnico.

6.1 Fase di test

Il progetto è stato sviluppato seguendo un approccio Test Driven (TDD), al fine di assicurarsi di rispettare, in ogni fase di sviluppo, i requisiti specifici. Al termine dello sviluppo sono stati eseguiti vari testi finali per assicurarsi il corretto funzionamento dell'app in tutte le sue parti.

In questa sezione verranno analizzati 3 casi di test, caratterizzati da percorsi differenti. Ogni test è stato effettuato analizzando le immagini di 100 punti fra quelli ottenuti per ogni percorso.

6.1.1 Test N.1

Per il primo caso di test è stata presa in considerazione una delle strade adiacenti al Politecnico di Bari, Via Re David. Il test è stato eseguito su di un percorso lineare con una lunghezza di circa 500m.

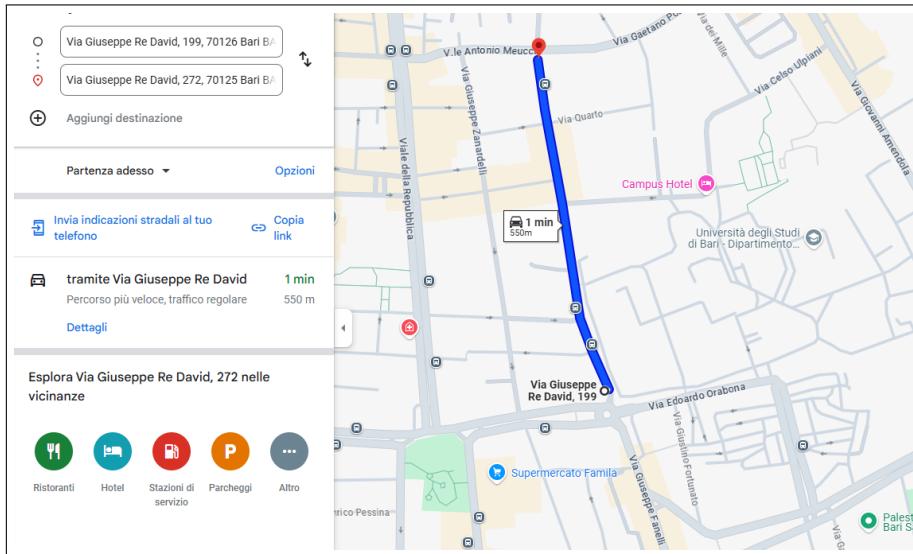


Figura 6.1: Via Re David, Bari



Figura 6.2: Risultati Test N.1

Com'è possibile vedere in figura 6.2 sul percorso sono stati rilevati 24 danni totali, di cui la maggior parte di categoria D10 (danni laterali). Una volta ottenute le immagini analizzate è stato effettuato un controllo sui danni rilevati e il risultato è stato coerente con gli score del modello. Nell'immagine presente in figura 6.2 possiamo notare un falso positivo: un danno di categoria D00 rilevato che in realtà non è un danno stradale ma è un danno del marciapiede.

6.1.2 Test N.2

Questo test, a differenza del primo, è stato effettuato su di un percorso non lineare con una lunghezza di circa 1.5 Km, comprendente strade diverse con molti incroci. Essendo un percorso più lungo del precedente il tempo necessario all'analisi delle immagini è stato maggiore.

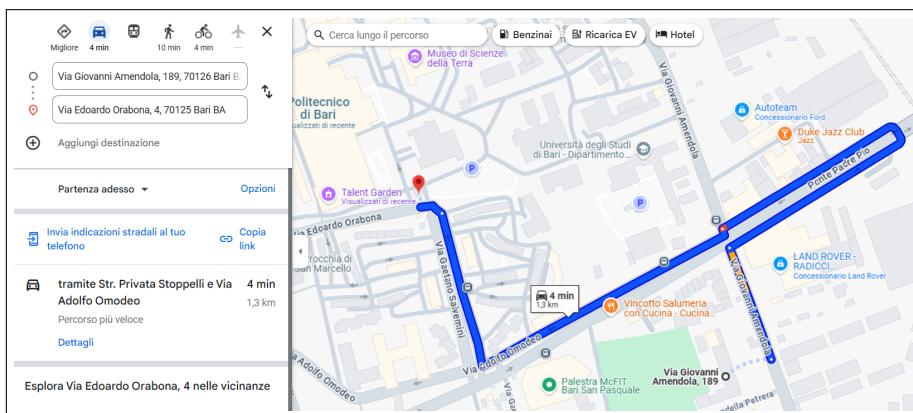


Figura 6.3: Via Amendola, Bari



Figura 6.4: Risultati test N.2

In questo caso, la categoria col maggior numero di danni è stata la categoria D00 (danni longitudinali). Essendo un percorso comprendente anche strade secondarie, quello che si è potuto constatare è stato un maggior numero di danni proprio sulle immagini relative a queste strade.

6.1.3 Test N.3

Per l'ultimo test è stato scelto un percorso molto più lungo di circa 4 Km per visualizzare, oltre ai danni, il tempo necessario all'ottenimento dell'analisi stessa. In quanto a un percorso più lungo corrispondono un numero maggiore di immagini da analizzare. Il percorso scelto è stato il lungomare di Bari dal teatro Margherita fino alla spiaggia Pane e Pomodoro.

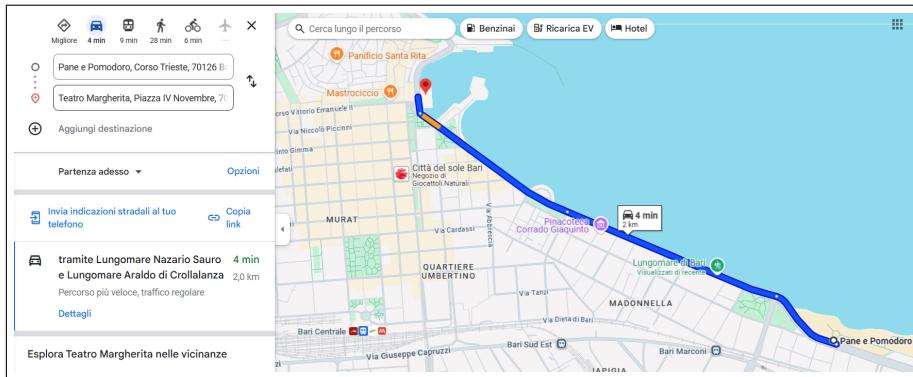


Figura 6.5: Lungomare

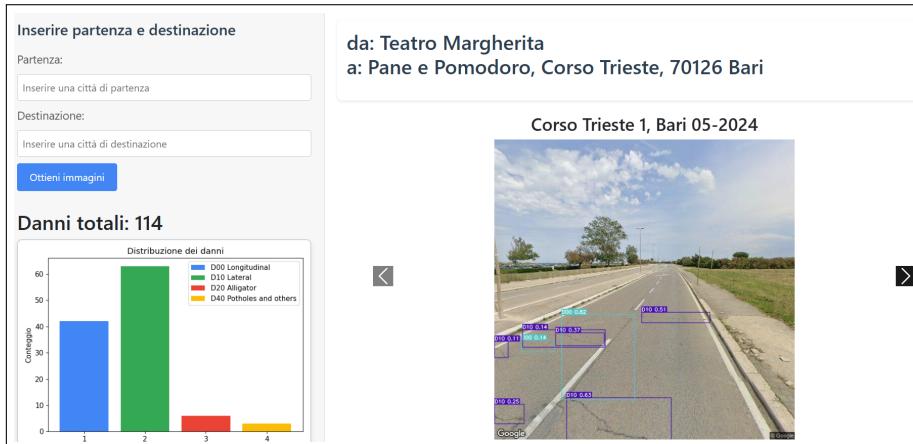


Figura 6.6: Risultati test N.3

Per quanto riguarda i danni, in questo caso la categoria con un numero di danni maggiori è la categoria D10 (danni laterali). Il tempo necessario all'analisi delle immagini è stato di circa 2 min. Questo è dovuto ovviamente ad un numero di immagini da analizzare superiore, ma questo tempo elevato è influenzato principalmente al contesto di run dell'applicazione, in quanto l'app è stata sviluppata e testata in locale su un PC portatile.

Sviluppi futuri e Conclusioni

- **Miglioramento della Precisione del Modello:** integrare versioni aggiornate o più avanzate dei modelli YOLO per aumentare la precisione nell'identificazione dei danni.
- **Espansione della Copertura Geografica:** estendere il supporto a ulteriori aree geografiche, migliorando l'affidabilità del servizio in regioni con dati limitati.
- **Ottimizzazione delle Prestazioni:** ridurre i tempi di elaborazione delle immagini implementando tecniche di parallelizzazione e migliorando l'efficienza del codice.
- **Integrazione con Sistemi di Allerta:** sviluppare un sistema di notifica che avvisi gli utenti in caso di rilevamento di danni gravi, offrendo informazioni in tempo reale.
- **Analisi Predittiva:** implementare algoritmi di machine learning per prevedere potenziali danni futuri in base ai dati storici e ambientali.
- **Miglioramenti all'Interfaccia Utente:** aggiungere funzionalità interattive, come filtri personalizzabili per visualizzare specifiche categorie di danni o statistiche avanzate.

Il progetto ha soddisfatto pienamente gli obiettivi preposti, offrendo un sistema efficiente per l'analisi e la visualizzazione dei danni rilevati su immagini geolocalizzate. Grazie all'integrazione delle API di Google e del servizio di routing OSRM, è stato possibile ottenere con precisione le immagini necessarie per l'analisi. I modelli YOLO impiegati hanno garantito un'accurata identificazione dei danni, raggiungendo alti standard di precisione richiesti dal progetto. Inoltre, l'architettura backend sviluppata con Flask si è dimostrata robusta e flessibile, supportando efficacemente la gestione delle richieste e l'elaborazione dei dati. Il front-end intuitivo ha permesso agli utenti di consultare facilmente i risultati dell'analisi, completando così con successo tutte le funzionalità pianificate per migliorare l'esperienza utente.

Bibliografia

- [1] Deeksha Arya, Hiroya Maeda, Sanjay Kumar Ghosh, Durga Toshniwal, Hiroshi Omata, Takehiro Kashiyama, and Yoshihide Sekimoto. Crowdensing-based Road Damage Detection Challenge (CRDDC'2022) . In *2022 IEEE International Conference on Big Data (Big Data)*, pages 6378–6386, Los Alamitos, CA, USA, December 2022. IEEE Computer Society.
- [2] Wenchao Ding, Xu Zhao, Bingke Zhu, Yinglong Du, Guibo Zhu, Tao Yu, Lei Li, and Jinqiao Wang. An ensemble of one-stage and two-stage detectors approach for road damage detection. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 6395–6400, 2022.
- [3] Google. Cloud computing services, 2024.
- [4] Japan IEEE Big Data 2022 Osaka. Crowdensing-based road damage detection challenge (crddc2022), 2022.
- [5] Ross Girshick Ali Farhadi University of Washington Allen Institute for AI Facebook AI Research Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection. 2016.
- [6] Ankit Kumar, Saroj Kumar Pandey, Sunny Prakash, Kamred Udham Singh, Teekam Singh, and Gaurav Kumar. Enhancing web application efficiency: Exploring modern design patterns within the mvc framework. In *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, pages 43–48, 2023.
- [7] Open Street Map. Open source routing machine by osm, 2024.
- [8] Pallet. Flask, 2010.
- [9] Bingke Zhu Yinglong Du Guibo Zhu Tao Yu Lei Li Jinqiao Wang School of Artificial Intelligence University of Chinese Academy of Sciences Beijing China National Laboratory of Pattern Recognition Institute of Automation Chinese Academy of Sciences Beijing China ObjectEye Inc. Beijing China Wenchao Ding, Xu Zhao. An ensemble of one-stage and two-stage detectors approach for road damage detection. 2022.
- [10] Mahdi Yusuf. Road damage detection application, 2023.