UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Analisi di dati multimediali provenienti dai social media e dagli e-tailers per prevedere le tendenze nel settore della moda

Analysis of multimedia data from social media and e-tailers to predict trends in the fashion sector

Candidato:
Riccardo Mancini

Relatore:
Prof. Adriano MANCINI

Correlatore:
Dott. Rocco PIETRINI

Anno Accademico 2023-2024

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

Sommario

Il problema dell'elaborazione delle immagini, mirata all'interpretazione e alla classificazione del contenuto delle stesse, ha attirato l'attenzione dei ricercatori dai primi tempi della nascita e diffusione dei calcolatori. Con il progredire della tecnologia dei sistemi di calcolo, la categorizzazione delle immagini ha trovato applicazioni sempre più vaste, riguardando discipline di nuova generazione come l'*object recognition* e la *computer vision*. È proprio in quest'ultima disciplina che si inserisce il lavoro di tesi, il cui tema centrale è l'uso delle tecniche di deep learning per l'analisi dei dati multimediali nel settore della moda.

Il lavoro svolto si articola in diverse macro-sezioni. Nella prima parte è stato fatto un approfondimento generale sul deep learning, con un focus particolare sui vari algoritmi di apprendimento automatico. Successivamente, è stato analizzato il concetto di rete neurale artificiale, approfondendo le reti neurali convoluzionali (CNN) e l'architettura *Transformer*, che sono state il fulcro del progetto. Queste architetture hanno permesso di implementare un sistema in grado di classificare e interpretare immagini di borse, ottenute attraverso tecniche di *web scraping* da vari siti di *e-commerce*.

La seconda sezione del lavoro è stata dedicata all'acquisizione e alla preparazione dei dati. Utilizzando strumenti di *scraping* come `Selenium Webdriver`, sono stati raccolti dati dettagliati su diverse categorie di borse, compresi metadati come prezzo, marca, dimensioni, colore e stato delle offerte. Questi dati sono stati successivamente puliti, normalizzati e annotati per creare un dataset robusto e rappresentativo, utilizzato nelle fasi successive del progetto.

La fase successiva ha coinvolto l'addestramento dei modelli di rilevamento oggetti. Questi modelli sono stati addestrati utilizzando tecniche avanzate di deep learning per riconoscere e classificare correttamente le borse nelle immagini. Le metriche di performance ottenute durante le fasi di validazione e test hanno mostrato un'ottima capacità di generalizzazione, indicando che i modelli erano in grado di riconoscere accuratamente le diverse categorie di borse anche su immagini non viste precedentemente.

Infine, è stato sviluppato un applicativo basato su `AWS Lambda` per creare un'API che permette di interagire con il miglior modello addestrato. Tale API è progettata per ricevere immagini in input, processarle attraverso il modello di rilevamento oggetti e restituire informazioni rilevanti, come la categoria della borsa e la posizione all'interno dell'immagine. L'applicativo sviluppato costituisce un avanzamento importante nell'integrazione di soluzioni di intelligenza artificiale per migliorare l'esperienza

utente e automatizzare i processi di classificazione delle immagini nel settore della moda.

Questo lavoro di tesi rappresenta un contributo significativo nel campo dell'*object detection* applicata al settore della moda, dimostrando il potenziale delle tecniche di deep learning per migliorare l'efficienza e l'accuratezza nella *detection* e nella classificazione degli oggetti all'interno di un'immagine. Le soluzioni sviluppate offrono nuove opportunità per analizzare le tendenze di consumo e personalizzare l'offerta di prodotti, fornendo una base solida per futuri sviluppi e applicazioni nell'ambito della *computer vision*.

Indice

Elenco delle figure

Capitolo 1.

Introduzione

In questo capitolo viene descritto il contesto applicativo nel quale si inserisce il lavoro di tesi, gli obiettivi prefissati e una sintesi dell'approccio adottato, che ha guidato le scelte e le metodologie trattate in questo documento. Prima di addentrarsi nei dettagli tecnici e metodologici, è necessaria una doverosa e importante contestualizzazione delle possibili applicazioni pratiche delle tecniche che saranno esposte nel prosieguo di questa tesi. Questa contestualizzazione non solo fornisce una visione chiara dell'ambito di applicazione, ma evidenzia anche la rilevanza e l'impatto potenziale delle soluzioni sviluppate.

1.1. Ambito del lavoro

Negli ultimi anni, il campo dell'intelligenza artificiale ha subito una trasformazione radicale, grazie a progressi significativi nelle tecniche di deep learning, con un impatto particolare nell'ambito della *computer vision*. Tra le aree di ricerca più avanzate, l'*object detection* si distingue non solo per la sua complessità intrinseca, ma anche per la sua rilevanza strategica. Questa disciplina si concentra sull'identificazione e localizzazione precisa di oggetti all'interno di immagini e video, offrendo soluzioni che trovano applicazione in settori critici quali la sicurezza e sorveglianza, la guida autonoma, l'analisi automatica delle immagini mediche e l'automazione industriale. La crescente domanda di soluzioni basate su dati visivi, che richiedono livelli sempre più elevati di accuratezza, efficienza e scalabilità, amplifica ulteriormente l'importanza dell'*object detection*. In un mondo in cui le immagini e i video stanno diventando la fonte primaria di dati, le tecnologie capaci di estrarre informazioni significative da queste fonti stanno trasformando interi settori economici e sociali. La sfida attuale è continuare a spingere i confini di ciò che è possibile, migliorando continuamente le prestazioni degli algoritmi e adattandoli a contesti sempre più complessi e diversificati. In questo contesto di rapida evoluzione tecnologica, nasce il presente lavoro di tesi, che si propone di rispondere alle crescenti esigenze di sistemi in grado di analizzare volumi sempre maggiori di dati visivi con precisione ed efficienza. Con l'avanzamento costante delle tecnologie di deep learning, si aprono nuove opportunità per superare le limitazioni delle metodologie tradizionali, offrendo soluzioni più accurate e capaci di affrontare sfide complesse. Questo lavoro, quindi, si preoccupa di rispondere

a necessità pratiche, contribuendo allo sviluppo di strumenti all'avanguardia per l'analisi delle immagini, in un panorama in cui la capacità di elaborare informazioni visive in modo tempestivo e affidabile è sempre più importante. Parallelamente, l'emergere e la diffusione dei servizi di cloud computing stanno giocando un ruolo fondamentale nel rendere tali tecnologie accessibili su larga scala, consentendo l'elaborazione e la gestione di grandi moli di dati in modo efficiente e sostenibile. In questo scenario, l'integrazione tra tecniche avanzate di *object detection* e infrastrutture cloud rappresenta una frontiera di ricerca promettente, capace di abilitare nuove applicazioni e migliorare l'efficienza operativa dei sistemi esistenti. L'elaborazione di immagini su cloud offre infatti numerosi vantaggi, tra cui la scalabilità, la disponibilità di risorse computazionali flessibili e la possibilità di distribuire modelli complessi in maniera accessibile. Tuttavia, la realizzazione di sistemi avanzati di *object detection* richiede la disponibilità di dataset di alta qualità, che siano rappresentativi e ben etichettati. La raccolta, il *preprocessing* e la gestione dei dati costituiscono quindi una componente critica del processo, poiché la qualità e la quantità dei dati di addestramento influenzano direttamente le prestazioni e l'affidabilità dei modelli sviluppati. Il progresso tecnologico in questo settore porta con sé anche sfide considerevoli. Tra queste, spiccano la necessità di migliorare l'efficacia delle tecniche di rilevazione, la gestione efficiente di modelli complessi e la loro distribuzione su piattaforme cloud in modo scalabile e accessibile. A ciò si aggiunge la rapidità con cui il settore evolve, che impone un costante aggiornamento delle competenze e una continua valutazione delle nuove metodologie e tecnologie emergenti. Questa dinamicità richiede non solo un'attenta analisi delle tendenze attuali, ma anche una riflessione critica sulle limitazioni esistenti e sulle direzioni future di sviluppo.

1.2. Obiettivi e contributi principali

Questa tesi mira a sviluppare soluzioni innovative nell'ambito dell'*object detection*, con un focus specifico sull'identificazione automatica di borse in immagini provenienti da *e-tailer* online. L'approccio adottato ha combinato l'uso di architetture di rete neurale avanzate e una raccolta dati accurata per affrontare le sfide della rilevazione di oggetti in contesti complessi e al tempo stesso piuttosto variabili. La qualità dei dati e l'efficacia delle reti neurali sono stati i pilastri fondamentali su cui si è basata l'intera ricerca. Il lavoro ha avuto inizio con una fase di raccolta dati estremamente accurata. Utilizzando tecniche moderne di *web scraping*, è stato possibile ottenere un dataset ampio ed eterogeneo di immagini di borse provenienti da diverse fonti *e-commerce*. Questo processo ha richiesto un attento studio delle piattaforme di origine dei dati e un approfondito lavoro di *preprocessing*, necessario per garantire che i dati fossero puliti, annotati correttamente e che le classi fossero opportunamente rappresentate. La costruzione di un dataset di alta qualità ha rappresentato una base solida per l'addestramento delle reti neurali, permettendo di affrontare con maggiore efficacia le complessità intrinseche dei dati. A partire da questo dataset, la

ricerca si è concentrata sulla sperimentazione di diverse architetture di rete neurale. Sono state testate sia soluzioni tradizionali basate su reti convoluzionali, sia approcci più moderni che integrano tecnologie *Transformer*, con l'obiettivo di identificare i modelli più adatti alla rilevazione delle borse in questione. La sperimentazione non si è limitata a valutare l'accuratezza dei modelli, ma ha anche esplorato la loro capacità di generalizzare su dati mai visti prima dalle reti, affrontando sfide come l'occlusione degli oggetti e lo sbilanciamento delle classi. Questa analisi ha permesso di selezionare le architetture più promettenti, da ottimizzare ulteriormente per migliorarne le prestazioni.

L'ottimizzazione dei modelli ha richiesto un lavoro meticoloso di *tuning* dei parametri, guidato da metriche specifiche come *Precision*, *Recall*, mAP (*Mean Average Precision*) e IoU (*Intersection Over Union*). Questo processo ha permesso di bilanciare efficacemente accuratezza e velocità di inferenza, rendendo i modelli non solo precisi, ma anche adatti a essere implementati in applicazioni e contesti reali. Un fattore piuttosto importante ed essenziale per poter ottenere risultati all'altezza è inoltre l'efficienza computazionale messa a disposizione che è stata un fattore chiave per garantire che le soluzioni sviluppate potessero essere utilizzate in contesti operativi dove la rapidità di elaborazione è cruciale. Dopo l'ottimizzazione dei modelli considerati, l'attenzione si è concentrata sull'integrazione pratica delle soluzioni sviluppate. L'implementazione di un'interfaccia web ha reso i modelli accessibili e scalabili, dimostrando la loro applicabilità in contesti reali e permettendo l'interazione diretta con utenti finali. Questa implementazione ha validato la fattibilità delle soluzioni proposte, evidenziando la capacità delle tecnologie sviluppate di risolvere problemi concreti.

Nel complesso, il lavoro ha prodotto risultati significativi, spaziando dalla creazione di un dataset specializzato alla sperimentazione e ottimizzazione di modelli di rete neurale avanzati. Questo approccio ha dimostrato come la sinergia tra tecniche di raccolta dati accurate e l'adozione di architetture neurali all'avanguardia possa dar vita a soluzioni di *object detection* estremamente efficaci e scalabili. I contributi di questa ricerca non solo migliorano lo stato dell'arte nell'ambito dell'*object detection*, ma offrono anche soluzioni concrete per settori strategici come la moda e l'*e-commerce*.

Oltre ad arricchire la letteratura esistente con nuovi dati e metodologie, questa ricerca apre nuove prospettive per futuri sviluppi e applicazioni concrete. In particolare, l'analisi dei dati multimediali provenienti dai social media e dagli *e-tailers* per prevedere le tendenze nel settore del fashion rappresenta un'area di grande rilevanza, con il potenziale di rivoluzionare il modo in cui le aziende del settore anticipano e rispondono alle esigenze del mercato. Le tecniche impiegate e i risultati ottenuti rappresentano un contributo significativo, con implicazioni che vanno ben oltre il contesto specifico della moda. Questi risultati suggeriscono applicazioni in altre aree dell'*e-commerce* e in settori in cui è essenziale una gestione precisa e automatizzata di grandi quantità di dati visivi.

1.3. Organizzazione

Il presente lavoro di tesi è organizzato in cinque capitoli.

Il secondo capitolo offre una panoramica sullo stato dell'arte, esaminando le tecniche attualmente utilizzate per la raccolta e l'acquisizione dei dati, con particolare attenzione sia agli approcci tradizionali che a quelli moderni. Viene inoltre fornita una revisione dettagliata delle principali metodologie di rilevazione degli oggetti basate su deep learning, con un'analisi delle sfide attuali e delle nuove tendenze nel settore. Infine, il capitolo esplora il ruolo del cloud computing, evidenziando le sue caratteristiche principali e le sfide legate alla sua adozione.

Nel terzo capitolo, l'attenzione si sposta sui materiali e le metodologie adottate per la realizzazione del progetto. Si descrive il processo di raccolta dei dati, seguito dalle fasi di preparazione e trattamento necessari per l'addestramento. Viene successivamente approfondito lo sviluppo e la sperimentazione di modelli avanzati, nonché la loro integrazione in un'infrastruttura pensata per l'esecuzione e l'interazione con l'utente.

Il quarto capitolo si concentra sull'analisi dei risultati ottenuti, valutando le prestazioni dei modelli in base a metriche definite e discutendo i risultati delle inferenze eseguite su dati di test. Questo capitolo offre una valutazione critica del lavoro svolto, mettendo in luce sia i punti di forza che le eventuali limitazioni.

Infine, il quinto capitolo riassume i principali contributi della tesi, fornendo una riflessione sulle implicazioni del lavoro svolto e delineando possibili sviluppi futuri. Vengono inoltre suggerite direzioni per ulteriori ricerche e applicazioni che potrebbero ampliare o migliorare i risultati ottenuti.

Capitolo 2.

Stato dell'arte

2.1. Metodi di raccolta dati

La raccolta dei dati è un momento essenziale in qualsiasi processo di analisi e interpretazione delle informazioni, indipendentemente dal campo di applicazione. La necessità di rendere tale fase accurata e metodica non può essere sottovalutata, poiché su di essa poggia l'intera catena del valore dell'analisi dei dati. In effetti, la qualità delle analisi, delle previsioni e delle decisioni che derivano dall'uso dei dati dipende direttamente dall'affidabilità e dalla completezza dei dati stessi. Nell'era digitale, la disponibilità di enormi quantità di dati ha trasformato il modo in cui le organizzazioni operano e prendono decisioni. Tuttavia, la mera disponibilità di questi ultimi non è sufficiente, in quanto è necessario adottare tecniche efficaci per acquisire tali dati in modo che siano utili e pertinenti ai fini dell'analisi. La raccolta delle informazioni deve quindi essere condotta con rigore, tenendo conto delle specificità delle fonti di dati e delle tecniche appropriate per estrarli. Di fronte a questo scenario, emerge l'esigenza di comprendere come le diverse metodologie di acquisizione dei dati possano essere applicate in contesti differenti. Nel corso degli anni, sono state sviluppate numerose tecniche che spaziano dai metodi tradizionali, spesso più semplici e manuali, a quelli moderni, che sfruttano le avanzate capacità tecnologiche disponibili oggi. Questa evoluzione è stata dettata dalla necessità di affrontare volumi di dati sempre più grandi e complessi, nonché dalla varietà delle fonti da cui questi dati possono essere estratti. Pertanto, è fondamentale esplorare come queste tecniche si siano adattate e continuino ad evolversi per soddisfare le esigenze contemporanee nel processo di analisi.

L'evoluzione delle tecniche di raccolta è stata guidata dall'esigenza di gestire una quantità sempre crescente di informazioni, spesso non strutturate, provenienti da una varietà di fonti online. Tuttavia, la loro efficacia dipende non solo dalle capacità tecnologiche, ma anche dalla comprensione delle implicazioni etiche e legali legate alla raccolta dei dati. La capacità di bilanciare l'innovazione tecnologica con il rispetto delle normative vigenti è diventata una componente critica del processo di acquisizione dei dati. In questo contesto, l'analisi dello stato dell'arte delle tecniche di acquisizione dati offre una panoramica delle metodologie esistenti e delle sfide

emergenti, fornendo un quadro completo delle pratiche attuali e delle direzioni future nel campo della gestione dei dati.

2.1.1. Tecniche tradizionali

Nell'ambito appena descritto, le tecniche tradizionali rappresentano le metodologie storicamente utilizzate per ottenere informazioni prima dell'avvento delle tecnologie digitali e automatizzate. Queste tecniche, pur essendo ancora valide e largamente utilizzate in diversi contesti, si basano principalmente sull'intervento umano e su processi manuali. Nonostante i progressi tecnologici abbiano portato a metodi più efficienti e automatizzati, le tecniche tradizionali continuano a offrire un valore significativo, soprattutto quando si tratta di ottenere dati qualitativi o in situazioni in cui l'automazione non è applicabile.

Tra le principali tecniche tradizionali di raccolta dati, si possono considerare i sondaggi, le interviste, le osservazioni dirette e l'utilizzo di dataset preesistenti. Ognuna di queste modalità ha delle peculiarità che ne determinano l'adeguatezza in specifici contesti e ne influenzano l'efficacia.

- I sondaggi rappresentano una delle forme più comuni di raccolta dati nelle ricerche sociali, di mercato e accademiche. Consistono nel porre domande a individui o gruppi attraverso diversi mezzi, come moduli cartacei, chiamate telefoniche, email o questionari online. Sebbene i sondaggi possano fornire preziose informazioni, la loro efficacia è spesso limitata da fattori come i bias dei rispondenti, l'accuratezza delle risposte e i limiti legati alla dimensione del campione o alla rappresentatività del gruppo studiato. La preparazione e la somministrazione dei sondaggi richiedono un'attenta pianificazione per minimizzare questi problemi e massimizzare la qualità dei dati raccolti.

- Le interviste offrono una modalità di raccolta dati basata su interazioni dirette, che possono essere condotte individualmente o in gruppo. Questo metodo è particolarmente utile per ottenere approfondimenti qualitativi e una comprensione più profonda di fenomeni complessi. Tuttavia, le interviste richiedono un notevole impegno in termini di tempo, sia nella fase di conduzione che in quella di analisi, e possono essere soggette a vincoli come la soggettività dell'intervistatore o la difficoltà di generalizzare i risultati all'intera popolazione di interesse.

- L'osservazione diretta rappresenta un'altra tecnica tradizionale importante, basata sulla registrazione di comportamenti o eventi in tempo reale. Questo metodo può essere estremamente utile per raccogliere dati che riflettono fedelmente le dinamiche naturali di un contesto specifico, senza interferenze esterne. Tuttavia, l'osservazione è intrinsecamente limitata dal tempo e dallo sforzo necessari per effettuare il monitoraggio, nonché dalla possibilità di bias dell'osservatore, che può influenzare l'interpretazione dei dati raccolti.

- L'accesso a dataset già esistenti, come registri governativi, pubblicazioni di ricerca o database pubblici, costituisce un'altra forma di raccolta dati tradizionale. L'utilizzo di tali dataset può offrire intuizioni preziose, in quanto si tratta spesso di informazioni già verificate e strutturate. Tuttavia, l'efficacia di questo metodo è condizionata dalla disponibilità di dataset rilevanti e dalla necessità di procedere manualmente all'estrazione e all'integrazione dei dati, operazioni che possono risultare complesse e dispendiose in termini di tempo.

Sebbene queste tecniche tradizionali possano ancora offrire un contributo prezioso, è fondamentale riconoscerne i limiti. In particolare, possono risultare dispendiose in termini di tempo, richiedendo un notevole sforzo per la raccolta e l'elaborazione dei dati, soprattutto quando si opera su larga scala. La loro capacità di raccogliere grandi volumi di informazioni in tempi brevi è spesso insufficiente, creando un ostacolo significativo in un contesto sempre più orientato ai big data. Un altro aspetto critico è il rischio di introdurre bias, sia attraverso l'intervento umano che durante la selezione e l'interpretazione dei dati. Infine, queste tecniche tendono a essere poco scalabili, specialmente quando si ha a che fare con dataset complessi o con la necessità di frequenti aggiornamenti.

2.1.2. Tecniche moderne

Con l'evoluzione tecnologica e la crescente disponibilità di dati online, le moderne tecniche di raccolta dati hanno assunto un ruolo cruciale nell'acquisizione e nell'analisi delle informazioni. Sviluppate per superare i limiti dei metodi tradizionali, queste tecniche sfruttano l'automazione e la potenza computazionale per raccogliere, elaborare e gestire dati su larga scala. Tecnologie come il *data crawling* e il *data scraping* hanno rivoluzionato il modo in cui i dati vengono acquisiti da fonti online. A differenza dei metodi tradizionali, che richiedono un notevole intervento umano e tempi lunghi, le tecniche moderne possono operare in modo continuo e automatico, consentendo di navigare tra grandi volumi di dati in tempi ridotti. Questo ha reso la scalabilità, nell'adozione di queste tecniche, uno dei loro principali punti di forza.

Il *data crawling*, ad esempio, è una tecnica che simula il comportamento di un motore di ricerca, navigando attraverso le pagine web seguendo i collegamenti ipertestuali. Un *crawler*, spesso chiamato anche *spider*, è un software progettato per esplorare il web in modo sistematico, scoprendo e indicizzando le pagine per consentire un'analisi successiva. Questo processo è alla base di molte applicazioni, dai motori di ricerca che forniscono risultati pertinenti agli utenti, fino ai sistemi di monitoraggio del mercato che tengono traccia delle fluttuazioni dei prezzi in tempo reale. Il *crawling* permette di raccogliere dati provenienti da molteplici fonti, creando una base ricca di informazioni che possono essere analizzate e utilizzate per vari scopi.

Il *data scraping* rappresenta una fase avanzata e più dettagliata del processo di raccolta dati, concentrandosi nell'estrazione di informazioni specifiche da contenuti

web non strutturati, come le pagine HTML. Questa tecnica consente di trasformare dati disorganizzati e dispersi in formati strutturati e organizzati, come tabelle o database, che possono essere facilmente analizzati. Ad esempio, un'azienda può utilizzare il *data scraping* per raccogliere automaticamente i prezzi dei prodotti dai siti web dei concorrenti e integrarli in un sistema interno di analisi, adattando così le proprie strategie di determinazione del prezzo di un certo prodotto e/o servizio.

Nel panorama informatico di questi approcci di raccolta dati, una varietà di strumenti e *framework* è stata sviluppata per facilitare e ottimizzare i processi di *crawling* e *scraping*. Tra i più noti nel campo *open-source*, `Scrapy`[1] si distingue come uno dei *framework* di *crawling* più utilizzati. `Scrapy` è altamente modulare e flessibile, permettendo agli sviluppatori di costruire e personalizzare *crawler* in modo efficace. Questo strumento non solo semplifica la raccolta di dati su larga scala, ma offre anche funzionalità integrate per la pulizia e la trasformazione dei dati, rendendolo una scelta ideale per chi cerca una soluzione completa per l'acquisizione dei dati. Oltre ai *framework* di *crawling*, gli strumenti di automazione del browser come `Selenium WebDriver`[2] e `Puppeteer`[3] hanno rivoluzionato il *data scraping*, permettendo di interagire con le pagine web come farebbe un utente umano. `Selenium`, uno dei più popolari, è ampiamente utilizzato per il testing delle applicazioni web, ma è anche estremamente efficace per l'automazione del *web scraping*, soprattutto in situazioni in cui è necessario interagire con elementi dinamici delle pagine. `Puppeteer`, d'altra parte, è un'altra potente libreria che offre funzionalità simili, con un focus particolare sulla gestione di pagine web basate su JavaScript. Questi strumenti permettono di simulare il comportamento dell'utente, eseguendo azioni come il clic sui pulsanti o lo scorrimento delle pagine, e sono particolarmente utili per estrarre dati da siti complessi e dinamici. Con l'evoluzione della tecnologia e la crescente diffusione di approcci dichiarativi e intuitivi, come le interfacce "punta e clicca", sempre più utenti hanno la possibilità di sviluppare soluzioni personalizzate senza dover possedere competenze avanzate di programmazione. In risposta a questa tendenza, sono state sviluppate soluzioni gestite, ovvero piattaforme che automatizzano gran parte del processo tecnico e riducono la complessità operativa. Queste piattaforme permettono a un pubblico più ampio di avvicinarsi al *web scraping* e all'analisi dei dati, offrendo strumenti che semplificano l'intero flusso di lavoro. Servizi come `Import.io` [4] e `Octoparse`[5], ad esempio, forniscono interfacce *user-friendly* che consentono di configurare facilmente progetti di raccolta dati e connessioni dirette a database e strumenti di analisi, il tutto senza richiedere competenze di programmazione. Ciò facilita il passaggio dalla raccolta delle informazioni alla loro interpretazione e valorizzazione.

[1] `https://scrapy.org/`
[2] `https://www.selenium.dev/`
[3] `https://pptr.dev/`
[4] `https://www.import.io/`
[5] `https://www.octoparse.it/`

Infine, rimanendo nell'ambito cloud, per chi necessita di soluzioni scalabili, esistono diverse tecnologie avanzate in grado di eseguire il *crawling* ad una portata più ampia, gestendo automaticamente la rotazione dei proxy per evitare il blocco da parte dei siti web, una problematica comune nei progetti di *scraping* intensivo. Queste soluzioni *cloud-based* ottimizzano il processo di raccolta dati, migliorando l'affidabilità delle operazioni e riducendo i problemi di accesso. Tuttavia, l'adozione di questi strumenti comporta una serie di sfide che vanno oltre i vantaggi in termini di scalabilità e riduzione del bias umano. Un aspetto fondamentale riguarda le implicazioni legali ed etiche legate al *web scraping*. Estrarre dati da siti web senza il consenso dei proprietari può violare i termini di servizio o le normative sulla protezione dei dati personali. Molte piattaforme, infatti, stabiliscono restrizioni attraverso file `"robots.txt"`, che specificano quali sezioni del sito possono essere esplorate e quali devono essere evitate. Inoltre, alcune aziende implementano misure di sicurezza, come i CAPTCHA, per proteggere i loro contenuti, rendendo il processo di estrazione dati più complesso. Oltre alle considerazioni legali ed etiche, esiste anche una sfida tecnica significativa in quanto molti siti web cambiano frequentemente design e struttura, rendendo necessario un continuo adattamento degli strumenti di *scraping* per garantirne il corretto funzionamento. Questo, di fatto, richiede un monitoraggio regolare e una manutenzione costante del software, con un impegno significativo in termini di risorse tecniche e competenze specialistiche.

2.2. Object detection e deep learning

Dopo aver esplorato le tecniche di raccolta dei dati, che costituiscono il primo passo essenziale per qualsiasi analisi nel contesto della *computer vision* e dell'Intelligenza Artificiale in generale, il passo successivo naturale conduce a uno dei campi più avanzati e in rapida evoluzione: l'*object detection*. Mentre la raccolta dei dati si focalizza sull'acquisizione e sull'organizzazione delle informazioni necessarie per alimentare i modelli di analisi, l'*object detection* rappresenta il momento in cui questi dati vengono trasformati in conoscenza, estraendo significato e contesto da immagini e video. Il riconoscimento degli oggetti è considerato una delle applicazioni più potenti e versatili di questo ambito, in grado di individuare e localizzare oggetti specifici all'interno di una scena. Questo processo non solo identifica la presenza di oggetti, ma ne determina con precisione la posizione all'interno dell'immagine, un aspetto fondamentale per numerose applicazioni pratiche. Ad esempio, nella guida autonoma, è essenziale riconoscere pedoni, segnali stradali e altri veicoli[1]; nella sicurezza, si richiede la rilevazione di persone in aree riservate o pericolose[2]; e nell'analisi comportamentale nei video, il monitoraggio del flusso dei clienti può ottimizzare la disposizione degli spazi nei punti vendita[3]. Si osserva che la precisione con cui questi modelli svolgono tali compiti è strettamente legata alla qualità dei dati su cui sono stati addestrati, sottolineando l'importanza di una solida base di dati. Il rapporto tra la qualità dei dati e le prestazioni dei modelli è fondamentale:

dati ben strutturati e accurati permettono ai modelli di deep learning di raggiungere livelli di precisione e affidabilità essenziali non solo per applicazioni complesse come l'automazione della guida o la diagnostica medica, ma anche in numerosi altri settori che già traggono vantaggio da queste tecnologie.

E' importante sottolineare come l'*object detection* abbia in qualche modo rivoluzionato il campo della *computer vision* grazie all'impiego e allo sviluppo di algoritmi di deep learning, i quali hanno permesso di superare le tecniche manuali di *feature extraction*, un tempo predominanti. Questi nuovi modelli sono in grado di apprendere autonomamente le caratteristiche rilevanti direttamente dai dati grezzi, superando molte delle limitazioni dei metodi tradizionali. Proprio per questo motivo, in tale capitolo si intende approfondire i concetti fondamentali che legano questo importante compito al mondo degli algoritmi di apprendimento automatico profondo, offrendo una panoramica delle problematiche affrontate e delle soluzioni innovative emergenti. Questo permetterà di comprendere appieno le scelte metodologiche adottate nel prosieguo di questo documento, evidenziando il ruolo sostanziale dell'*object detection* nella moderna analisi delle immagini.

2.2.1. Concetti fondamentali

Per comprendere appieno la portata e la complessità del *task* in questione, è essenziale distinguere due concetti chiave: *object recognition*[4] e *object localization*[5]. Sebbene questi termini vengano talvolta usati in modo intercambiabile, rappresentano compiti differenti che affrontano vari aspetti del riconoscimento visivo.

L'*object recognition* è un processo sofisticato e molto spesso impiegato nel campo della *computer vision* volto a identificare specifici oggetti all'interno di immagini o video attraverso l'analisi di caratteristiche distintive. Questo processo va ben oltre il semplice riconoscimento della presenza di un oggetto; richiede al sistema di identificare precisamente la categoria a cui appartiene l'oggetto, come ad esempio se si tratta di un volto umano, di un'auto, di un animale o di un qualsiasi altro oggetto specifico. È una tecnologia fondamentale che alimenta una vasta gamma di applicazioni moderne, come i sistemi di guida autonoma, dove le auto devono riconoscere pedoni o altri veicoli, oppure nella medicina diagnostica, dove è essenziale identificare tessuti o cellule specifiche, così come nei sistemi di sorveglianza avanzata, per riconoscere volti o comportamenti sospetti. La complessità non deriva solamente dalla capacità di rilevare che un oggetto è presente, ma anche di comprendere che tipo di oggetto si tratta, un compito che richiede un livello di comprensione visiva molto più profondo. Ad esempio, nel riconoscimento facciale, l'algoritmo deve distinguere tra un volto umano e altri oggetti circostanti; nella guida autonoma, il sistema deve essere in grado di identificare un pedone tra molti altri elementi presenti in una scena potenzialmente complessa. In Figura 2.1 un esempio di questo concetto.

D'altra parte, anche l'*object localization* è un concetto centrale nella *computer vision* e risulta essere molto più complesso di quanto possa sembrare. Non si tratta

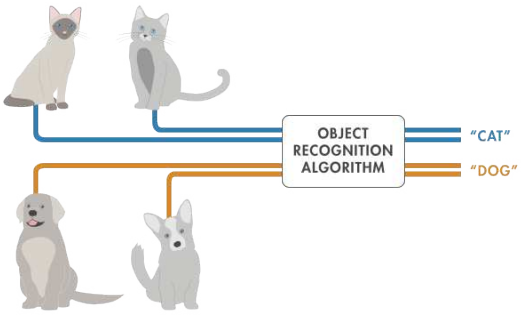

Figura 2.1.: Object recognition per identificare diverse categorie di oggetti

Immagine da: `https://it.mathworks.com/solutions/image-video-processing/object-recognition.html`

semplicemente di "individuare" un oggetto all'interno di un'immagine, ma di stabilire con precisione dove esso si trova. Questo richiede una comprensione approfondita dello spazio visivo, del contesto ambientale in cui l'oggetto è inserito e delle diverse caratteristiche che influenzano la percezione visiva di un sistema automatizzato. Un aspetto molto importante dell'*object localization*, come per il precedente *task* discusso, è che risiede alla base di molte applicazioni avanzate esistenti al giorno d'oggi. Ad esempio, nelle applicazioni di sicurezza, non basta sapere che c'è un volto umano: è necessario sapere esattamente dove si trova quel volto rispetto agli altri oggetti, allo sfondo o al campo visivo complessivo. Questa informazione è fondamentale per tracciare i movimenti e prevedere comportamenti. Lo stesso vale per le auto a guida autonoma: non è sufficiente riconoscere la presenza di un pedone, ma occorre localizzarlo esattamente rispetto al veicolo e al percorso, in tempo reale, per prevenire incidenti. Tuttavia, localizzare un oggetto non è sempre semplice: l'algoritmo deve essere in grado di gestire molteplici sfide, tra cui l'occlusione (quando l'oggetto è parzialmente nascosto), il *clutter* di sfondo (quando l'immagine contiene molti elementi disturbanti) e la variabilità nell'aspetto degli oggetti, che può dipendere da angolazioni diverse, condizioni di illuminazione o altri fattori ambientali. L'occlusione, in particolare, rappresenta una delle difficoltà maggiori. Quando un oggetto è parzialmente coperto da un altro, i sistemi di visione artificiale possono avere difficoltà a identificarlo e localizzarlo correttamente. Questo problema richiede modelli predittivi avanzati, in grado di stimare la posizione dell'oggetto anche in condizioni di visibilità ridotta. Un'altra sfida significativa è il *clutter* di sfondo (o *background clutter*). In ambienti visivamente complessi, dove molti oggetti si sovrappongono o interferiscono tra loro, il sistema deve essere in grado di distinguere l'oggetto di interesse dal "rumore" visivo circostante. Questa capacità di discriminazione è essenziale in applicazioni come la realtà aumentata, dove è necessario sovrapporre informazioni virtuali a un ambiente fisico in modo accurato e coerente. Inoltre, il fatto che lo stesso oggetto può apparire in modo completamente diverso a seconda dell'illuminazione, dell'angolo di visualizzazione o

delle condizioni atmosferiche può considerarsi un'ulteriore complicazione. Questo richiede algoritmi robusti, capaci di generalizzare su una vasta gamma di condizioni diverse, per garantire una localizzazione accurata degli oggetti in situazioni mutevoli. In Figura 2.2 un esempio pratico di questo concetto.

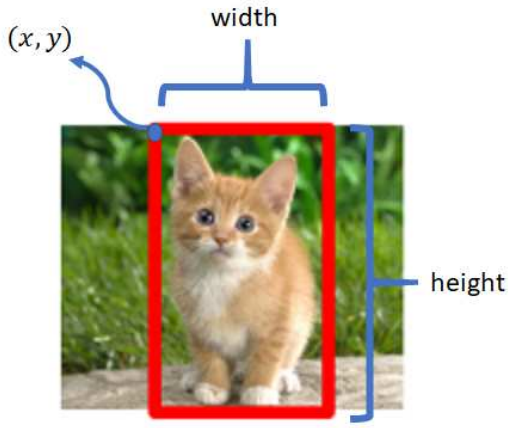

Figura 2.2.: Object localization per individuare oggetti nell'immagine

Immagine da: `https://medium.com/analytics-vidhya/object-localization-using-keras-d78d6810d0be`

È importante comprendere come l'*object detection* quindi combini entrambe queste funzioni: non solo riconosce la presenza di uno o più oggetti (*recognition*), ma ne determina anche la posizione (*localization*) all'interno dell'immagine. Questo rende il task di riconoscimento di oggetti un compito decisamente complesso e computazionalmente intensivo, poiché richiede sia la classificazione che la localizzazione simultanea degli oggetti. Per raggiungere tale obiettivo, si può ricorrere a diverse tecniche. Nel machine learning tradizionale, si fa affidamento su algoritmi che estraggono caratteristiche visive particolarmente significative, come i bordi, i gradienti, o altre caratteristiche geometriche, e queste vengono poi utilizzate per classificare gli oggetti. Esempi di tali algoritmi includono il modello HOG-SVM[6], che combina una descrizione accurata delle forme degli oggetti con un classificatore SVM[7], e il noto algoritmo Viola-Jones[8], utilizzato principalmente per il riconoscimento facciale. Questi metodi si sono dimostrati efficaci in contesti relativamente semplici, dove i dati a disposizione sono limitati e le variabili sono poche. Tuttavia, con l'adozione del deep learning, il campo dell'*object recognition* ha visto un drastico miglioramento in termini di precisione e adattabilità. Le reti neurali convoluzionali (CNN)[9], vero motore del deep learning applicato all'*object detection*, hanno rivoluzionato il modo in cui gli oggetti vengono riconosciuti. A differenza degli approcci tradizionali, che richiedono l'estrazione manuale delle caratteristiche, le CNN apprendono in modo automatico le caratteristiche rilevanti direttamente dai dati grezzi, rendendo il processo molto più efficiente e accurato. Ad esempio, una CNN può essere addestrata su milioni di immagini di cani e gatti, permettendole di riconoscere automaticamente le

differenze sottili nelle caratteristiche visive che distinguono i due animali. Questo tipo di apprendimento profondo si basa su reti con molti livelli che apprendono rappresentazioni gerarchiche degli oggetti: dai semplici bordi e *texture* nei primi strati, fino a forme più complesse negli strati successivi. Quando si utilizza il deep learning applicato a questo task, o ad altri task simili, ci sono due principali approcci. Il primo consiste nell'addestrare una rete neurale da zero, il che richiede una grande quantità di dati etichettati e una notevole potenza computazionale. Questo approccio può portare a risultati estremamente precisi, ma richiede risorse considerevoli, sia in termini di dati che di potenza di calcolo. Il secondo approccio, spesso più pratico ed efficiente, è quello del *transfer learning*, in cui si sfruttano modelli pre-addestrati su dataset vastissimi, come ImageNet[10], che poi vengono "perfezionati" per riconoscere nuovi tipi di oggetti specifici, riducendo così drasticamente il tempo e il costo di addestramento. Questa tecnica è particolarmente utile quando si ha a disposizione un numero limitato di nuove immagini per l'addestramento dei modelli, poiché permette di sfruttare la conoscenza già acquisita dai modelli preesistenti, adattandoli velocemente a nuovi contesti.

E' evidente come il deep learning abbia avuto un impatto significativo su questo task, consentendo di automatizzare e perfezionare il processo di riconoscimento e localizzazione degli oggetti. Si può inoltre comprendere come l'introduzione delle reti neurali convoluzionali ha permesso di superare molti limiti esistenti, offrendo la capacità di apprendere autonomamente rappresentazioni gerarchiche degli oggetti direttamente dai dati grezzi. Queste reti sono in grado di riconoscere pattern complessi e di identificare gli oggetti con un alto grado di precisione, anche in condizioni variabili o in presenza di rumore visivo. Questo ha reso possibile l'applicazione dell'*object detection* in una vasta gamma di settori, migliorando significativamente la capacità dei sistemi di prendere decisioni basate sull'analisi visiva in tempo reale. In Figura 2.3 è riportata una comparazione ad alto livello dei due approcci al *task* descritto.

2.2.2. Architetture di reti neurali per object detection

Come anticipato, l'evoluzione delle architetture per l'*object detection* è strettamente legata allo sviluppo delle reti neurali convoluzionali (CNN). Queste reti sono state fondamentali per l'avanzamento di molte applicazioni, grazie alla loro capacità di apprendere rappresentazioni gerarchiche dei dati visivi. Una CNN è in grado di estrarre automaticamente caratteristiche a livelli crescenti, partendo da elementi semplici come bordi e *texture* fino a rappresentazioni più complesse che includono parti di oggetti e oggetti interi. A differenza dei metodi tradizionali, che richiedevano la definizione manuale delle caratteristiche da utilizzare per l'analisi delle immagini, le CNN possono apprendere autonomamente le caratteristiche rilevanti direttamente dai dati grezzi. Questo ha rappresentato un progresso significativo, migliorando notevolmente le prestazioni dei modelli di riconoscimento e localizzazione degli oggetti. Le CNN operano attraverso una serie di strati convoluzionali, che applicano filtri per

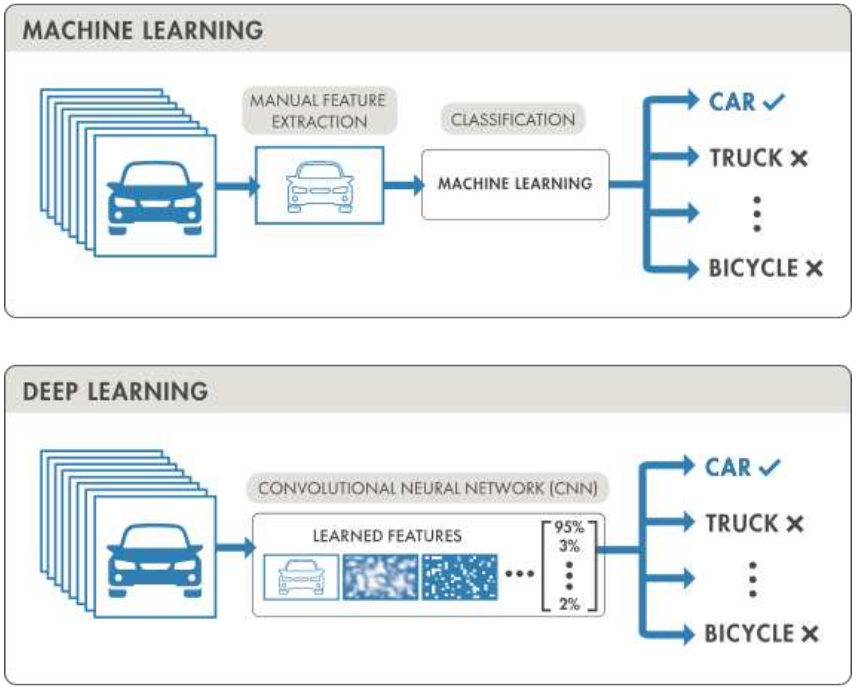

Figura 2.3.: Confronto ad alto livello tra i due approcci per il riconoscimento degli oggetti

Immagine da: `https://it.mathworks.com/solutions/image-video-processing/object-recognition.html`

rilevare caratteristiche specifiche delle immagini, ottimizzati durante l'addestramento per migliorare la capacità del modello di identificare pattern rilevanti.

In particolare, il livello convoluzionale è il componente chiave di una CNN, dove si eseguono la maggior parte dei calcoli. L'input è tipicamente un'immagine a colori, rappresentata come una matrice tridimensionale di pixel (altezza, larghezza e profondità, corrispondenti ai canali RGB). Un filtro o *kernel* si muove attraverso l'immagine, eseguendo l'operazione di convoluzione, che genera una mappa delle caratteristiche (detta anche mappa di attivazione). Questo processo prevede:

- Applicazione di un filtro (spesso 3x3) che esegue un prodotto scalare tra i pixel dell'immagine e i valori del filtro.
- Condivisione dei parametri, in cui lo stesso filtro viene applicato su tutta l'immagine per garantire efficienza.
- Durante l'addestramento, i pesi del filtro vengono ottimizzati tramite la retro-propagazione (*backpropagation*) e la discesa del gradiente (*Gradient Descent*).

Invece, i tre iperparametri principali che influenzano l'output del livello convoluzionale, che si può osservare in Figura 2.4, sono:

- Numero di filtri: che determina la profondità dell'output (ad esempio, tre filtri creano tre mappe di caratteristiche).

- *Stride*: il passo con cui il *kernel* si sposta sull'immagine; *stride* più ampi riducono la dimensione dell'output.
- *Padding*: l'aggiunta di zero ai bordi dell'immagine, che può essere di tre tipi: *valid padding*, senza *padding*, riduce le dimensioni del risultato; *same padding*, mantiene le dimensioni dell'output uguali a quelle dell'input; *full padding*, aumenta le dimensioni del risultato aggiungendo zeri ai bordi dell'input.

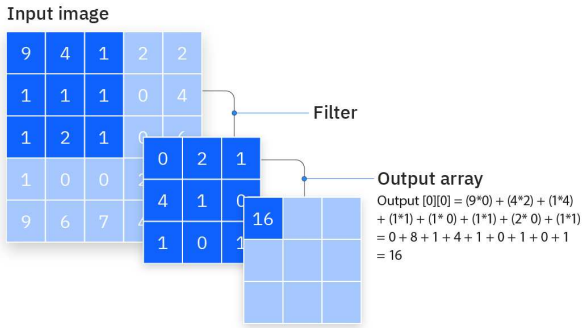

Figura 2.4.: Esempio dell'operazione di convoluzione

Immagine da: `https://www.ibm.com/topics/convolutional-neural-networks`

Dopo ogni convoluzione, una funzione di attivazione non lineare come la ReLU (Rectified Linear Unit) viene applicata per introdurre non linearità nel modello e permettere l'apprendimento di modelli complessi. Per ridurre la dimensionalità e migliorare l'efficienza, le CNN utilizzano i livelli di *pooling*, che sottocampionano l'output riducendone le dimensioni spaziali. Questo può avvenire tramite il Max pooling, che individua il valore massimo in ciascuna regione del *kernel*, oppure tramite Average pooling che invece calcola la media dei valori all'interno del campo del *kernel*. Anche se il *pooling* comporta una perdita di informazioni, è utile per ridurre la complessità computazionale e limitare il rischio di sovradattamento (*overfitting*). Alla fine del processo convoluzionale e di *pooling*, i dati vengono "appiattiti" e inviati a uno o più livelli completamente connessi (*fully connected layer*). In questo stadio, ogni neurone è connesso a ogni neurone del livello precedente, e l'obiettivo è combinare le caratteristiche apprese per eseguire la classificazione finale. Spesso, questi livelli utilizzano la funzione di attivazione Softmax per assegnare una probabilità a ciascuna classe di oggetto. Questa architettura di base consente alle CNN di assumere una struttura gerarchica. I livelli convoluzionali iniziali estraggono caratteristiche di basso livello come bordi e *texture*, mentre i livelli successivi combinano queste informazioni per riconoscere parti di oggetti. Infine, i livelli superiori riconoscono oggetti completi integrando le informazioni rilevate dai livelli precedenti. Ad esempio, per riconoscere una bicicletta, i livelli inferiori rilevano le ruote, il telaio e i pedali, mentre i livelli superiori integrano questi elementi per riconoscere l'oggetto completo, come viene

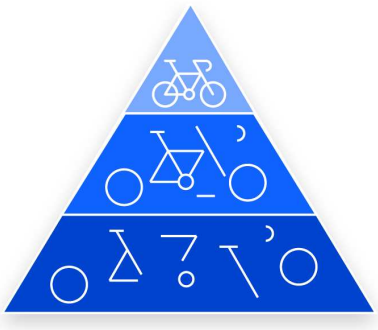

Figura 2.5.: Estrazione delle *feature* seguendo una struttura gerarchica

Immagine da: `https://www.ibm.com/topics/convolutional-neural-networks`

mostrato ad alto livello in Figura 2.5. In Figura 2.6, invece, un possibile schema di una CNN in cui sono stati inclusi gli elementi appena discussi.

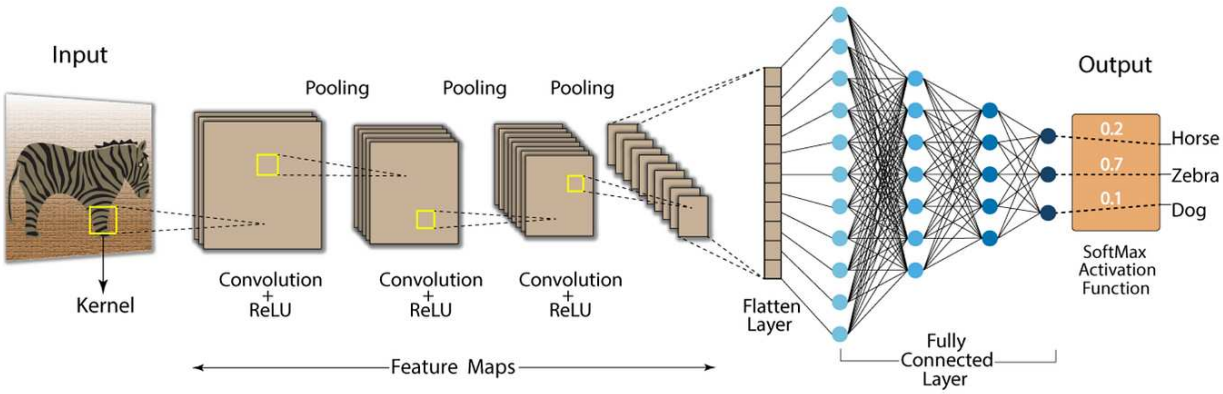

Figura 2.6.: Possibile architettura di una CNN

Immagine da: `https://developersbreach.com/convolution-neural-network-deep-learning/`

Le prime architetture CNN, pur essendo innovative, presentavano diverse inefficienze. Tra i primi modelli di rilievo si trovano le R-CNN[11] (Regions with Convolutional Neural Networks), che hanno introdotto l'idea di segmentare un'immagine in più regioni di interesse (Region Proposals), elaborate poi singolarmente con una CNN. Sebbene questo sia stato un passo importante, il processo risultava lento e oneroso in termini di risorse computazionali poiché ogni regione doveva essere analizzata separatamente. Questo approccio a due fasi, noto come *Two-Stage object detection*, ha portato allo sviluppo di versioni più ottimizzate, come Fast R-CNN[12], che ha permesso di elaborare tutte le regioni di interesse contemporaneamente, riducendo drasticamente i tempi di calcolo. Il vero progresso si è avuto con Faster R-CNN[13], che ha integrato la rete di Region Proposal direttamente nella CNN, migliorando ulteriormente l'efficienza computazionale e rendendo il processo di *object detection* più rapido e preciso. Nonostante i miglioramenti apportati dalle varie versioni delle

R-CNN, queste architetture non erano ancora sufficientemente rapide per applicazioni in tempo reale, dove la velocità è un fattore critico. A tal fine, è stato sviluppato YOLO[14] (You Only Look Once), un'architettura che ha introdotto un approccio completamente nuovo all'*object detection*. Diversamente dai metodi tradizionali, che suddividevano l'immagine in più regioni di interesse e le analizzavano singolarmente, YOLO processa l'intera immagine in un singolo passaggio, dividendo l'immagine in una griglia e predicendo, per ogni cella, le probabilità di presenza di un oggetto e le coordinate dei riquadri di delimitazione (*bounding boxes*). Questo approccio consente al sistema di riconoscere e localizzare simultaneamente più oggetti all'interno di un'immagine, offrendo prestazioni in tempo reale. Nella pratica, ogni cella della griglia è responsabile di rilevare un oggetto se il centro di tale oggetto cade all'interno della cella stessa. Per ogni cella, YOLO calcola un insieme di probabilità relative alle classi di oggetti che potrebbe contenere e le coordinate del riquadro di delimitazione, che includono la posizione (x, y), l'altezza, la larghezza e la confidenza, ossia la probabilità che un oggetto si trovi effettivamente all'interno di quel riquadro. Questo meccanismo consente di gestire l'intero processo di rilevamento con un solo passaggio attraverso la rete, riducendo drasticamente il tempo di calcolo rispetto ai modelli a due fasi, che richiedono una segmentazione preliminare delle regioni. Grazie alla sua architettura monolitica e alla sua capacità di processare l'immagine in un'unica rete neurale, YOLO è diventato uno dei modelli più utilizzati nelle applicazioni che richiedono alte prestazioni in termini di velocità, senza sacrificare la precisione. Questo approccio ha aperto la strada alla cosiddetta *One-Stage object detection*, in cui il riconoscimento e la localizzazione degli oggetti avvengono in una sola fase, al contrario dei modelli tradizionali a due fasi come Faster R-CNN. YOLO ha ispirato la creazione di altre architetture come SSD [15] (Single Shot MultiBox Detector) e RetinaNet[16], che hanno trovato un equilibrio tra velocità e accuratezza. SSD, a differenza di YOLO, utilizza più griglie di diverse dimensioni e scale per migliorare la precisione nella localizzazione degli oggetti di varie dimensioni, rendendo il modello più adattabile a oggetti grandi e piccoli. RetinaNet, invece, ha introdotto il concetto di Focal Loss, una funzione di perdita che aiuta a gestire lo squilibrio tra oggetti in primo piano e lo sfondo. Questo miglioramento è particolarmente utile nelle scene complesse in cui gli oggetti piccoli possono essere facilmente ignorati dai modelli tradizionali, e contribuisce notevolmente a migliorare le performance nella rilevazione di oggetti più piccoli e difficili da distinguere. In Figura 2.7 viene riportato un confronto tra i due approcci all'*object detection* che caratterizzano i modelli citati.

Negli ultimi anni, il campo dell'*object detection* ha visto l'introduzione di nuove architetture basate su una tecnologia innovativa: i *Transformer*[18]. Questi modelli, originariamente sviluppati per il *Natural Language Processing* (NLP), hanno rivoluzionato anche il panorama della *computer vision*, introducendo una nuova modalità di elaborazione delle immagini. A differenza delle reti neurali convoluzionali, i *Transformer* sfruttano il cosiddetto meccanismo di attenzione, che permette di gestire in modo efficiente le relazioni a lungo raggio tra diverse parti dell'immagine. Il punto

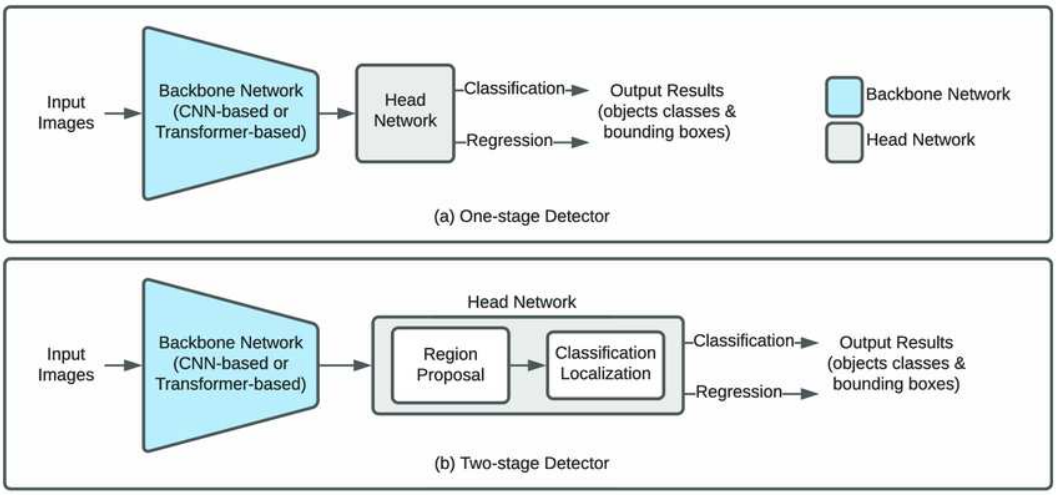

Figura 2.7.: Confronto ad alto livello tra approcci *One-Stage object detection* e *Two-Stage object detection*

Immagine da [17]

di forza dei *Transformer* risiede nella loro capacità di elaborare globalmente l'intera immagine, stabilendo connessioni tra pixel distanti e rilevando schemi complessi anche in aree remote del contesto visivo. Questo è un netto cambiamento rispetto alle CNN tradizionali, che si basano su convoluzioni locali per estrarre caratteristiche visive da porzioni limitate dell'immagine. In scenari complessi, come quelli in cui gli oggetti sono parzialmente nascosti o presenti in scene sovraffollate, i *Transformer* si dimostrano più efficaci, poiché il meccanismo di attenzione attribuisce dinamicamente maggiore importanza alle aree dell'immagine più rilevanti per il compito di rilevamento. Un esempio di questa innovazione è rappresentato dal Vision Transformer (ViT)[19], una delle prime architetture che ha applicato con successo la tecnologia *Transformer* alla *computer vision*. Il ViT tratta un'immagine come una sequenza di piccole "*patch*", ossia frammenti rettangolari che vengono interpretati come *token*, proprio come avviene nel linguaggio naturale. Questo approccio elimina la necessità di utilizzare i *kernel* convoluzionali tipici delle CNN, e al loro posto sfrutta l'attenzione per stabilire relazioni tra le varie *patch*, offrendo una visione globale dell'immagine. Le *patch* non vengono analizzate come elementi isolati, ma come parti di un insieme che contribuiscono collettivamente al riconoscimento e alla localizzazione degli oggetti. Ciò risulta particolarmente vantaggioso in applicazioni che richiedono una visione d'insieme della scena, poiché il modello è in grado di identificare oggetti anche in situazioni di occlusione o sovrapposizione. L'approccio del ViT si distingue per la sua capacità di gestire il contesto globale e locale in modo integrato, migliorando l'accuratezza del rilevamento rispetto ai modelli basati su convoluzioni locali. Nonostante i loro vantaggi, i *Transformer* richiedono grandi quantità di dati per ottenere risultati ottimali. La loro capacità di catturare relazioni globali all'interno delle immagini dipende dalla disponibilità di vasti dataset etichettati, il che li rende sensibili alla

scarsità di dati. Questo requisito è parzialmente mitigato dal *transfer learning*, che consente di utilizzare modelli pre-addestrati su dataset enormi come ImageNet per adattarli a compiti specifici con un numero ridotto di dati. Tuttavia, i *Transformer* tendono a richiedere maggiori risorse computazionali rispetto alle CNN, specialmente nelle prime fasi di addestramento. In Figura 2.8 è stato riportato un esempio di architettura *Transformer*.

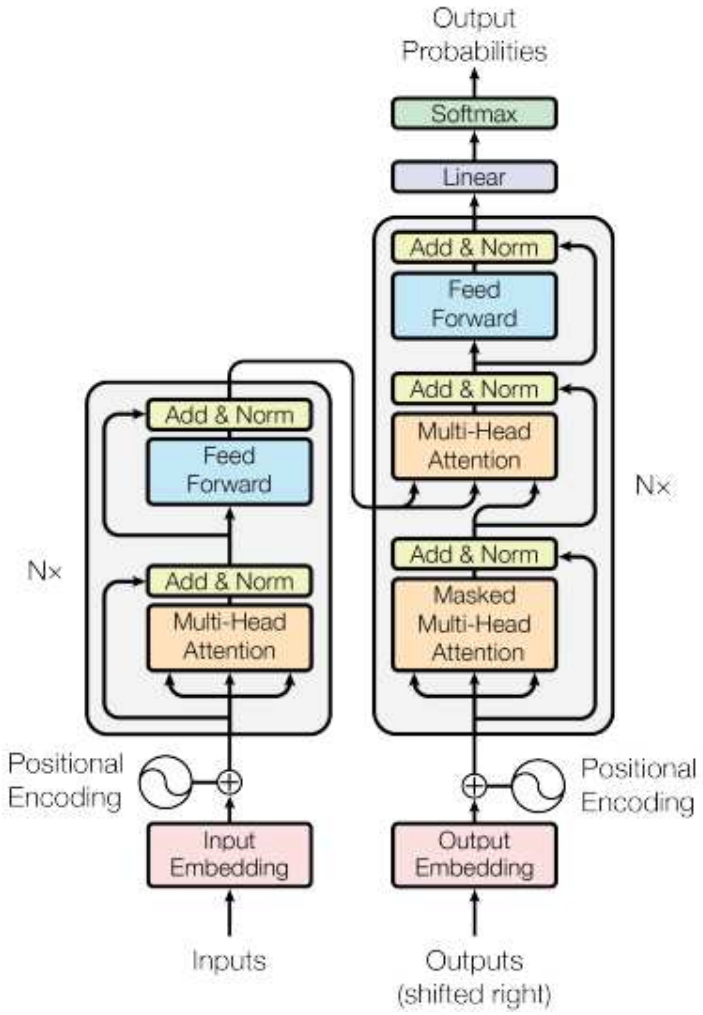

Figura 2.8.: Architettura *Transformer*

Immagine da [18]

2.2.3. Dataset di riferimento

Per l'addestramento e la valutazione dei modelli di *object detection*, l'utilizzo di dataset appropriati e metriche di valutazione rigorose è fondamentale. La qualità, la diversità e la quantità dei dati presenti nei dataset influenzano direttamente le capacità di apprendimento del modello e la sua generalizzazione in scenari reali. Pertanto, la

scelta del dataset non è solo una questione tecnica, ma anche strategica: un dataset bilanciato e realistico consente al modello di affrontare con successo le sfide visive che incontrerà nelle applicazioni pratiche. Uno dei dataset più importanti e utilizzati per l'*object detection* è MS COCO (Microsoft Common Objects in Context)[20]. MS COCO è stato progettato per rappresentare la complessità delle scene visive nel mondo reale. Contiene oltre 330.000 immagini, di cui più di 200.000 con annotazioni dettagliate, comprendendo 80 categorie di oggetti, tra cui oggetti comuni come automobili, biciclette e animali, oltre a categorie più specifiche come ombrelli, borse e attrezzature sportive. Una caratteristica distintiva di questo dataset è la sua capacità di rappresentare scene che includono più oggetti sovrapposti, in contesti diversi e con vari livelli di occlusione, variazioni di scala e pose complesse. Questo fatto rende MS COCO particolarmente utile per addestrare modelli che devono gestire scenari complessi e dinamici. Oltre ai *bounding boxes*, vengono fornite anche le maschere di segmentazione e annotazioni di *keypoints* per compiti avanzati, come la stima delle pose umane, rendendolo uno strumento molto valido per una vasta gamma di applicazioni. In Figura 2.9 sono state riportate alcune immagini estratte dal dataset in questione.

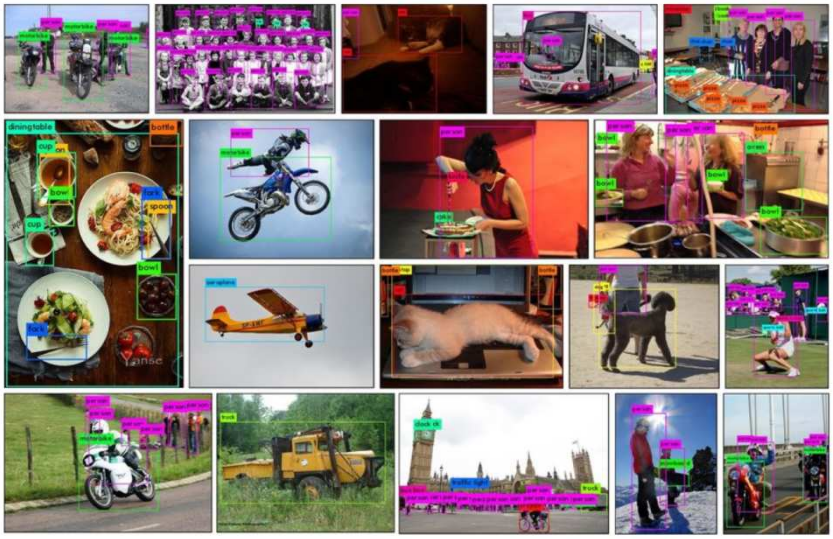

Figura 2.9.: MS COCO dataset

Un altro dataset fondamentale nel campo del rilevamento degli oggetti è Pascal VOC[21]. Pur essendo più piccolo e meno complesso rispetto a MS COCO, con circa 20 categorie di oggetti, Pascal VOC si distingue per la semplicità e la chiarezza delle sue annotazioni, che includono *bounding box* precise per ogni oggetto. Questa caratteristica lo rende ideale per chi necessita di un dataset facile da gestire, particolarmente utile nella fase di prototipazione e sperimentazione rapida di nuovi algoritmi. Inoltre, Pascal VOC ha acquisito rilevanza grazie alle sue competizioni annuali, che hanno stimolato una crescita significativa nella ricerca sull'*object detection*, fornendo un *benchmark* chiaro e accessibile per il confronto e la validazione di nuove soluzioni. In

Figura 2.9 sono state riportate alcune immagini estratte da questo secondo dataset.

Figura 2.10.: Pascal VOC dataset

2.2.4. Metriche di valutazione

La quantità di dati all'interno dei dataset è solo una parte dell'equazione. Per valutare in modo accurato e completo un modello di *object detection*, è necessario utilizzare opportune metriche di validazione. Tra le metriche fondamentali, *Precision* e *Recall* sono indispensabili per comprendere due aspetti critici del modello:

- La *Precision* misura la percentuale di rilevamenti corretti rispetto al totale delle rilevazioni effettuate, indicando quanto il modello sia preciso nel riconoscere oggetti senza generare falsi positivi. La formula per calcolare la *Precision* è:

$$\text{Precision} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} = \frac{TP}{TP + FP}$$

  dove TP (*true positives*) rappresenta i rilevamenti corretti, e FP (*false positives*) rappresenta i falsi positivi.

- La *Recall*, invece, valuta la capacità del modello di rilevare tutti gli oggetti presenti in un'immagine, misurando quanti veri positivi il modello riesce a identificare senza trascurare oggetti effettivamente presenti. La formula per il calcolo della *Recall* è:

$$\text{Recall} = \frac{\text{Correct Predictions}}{\text{Total Ground Truth}} = \frac{TP}{TP + FN}$$

  dove FN (*false negatives*) rappresenta gli oggetti effettivamente presenti ma non rilevati dal modello.

Queste due metriche, pur essendo essenziali, devono essere bilanciate per offrire una visione più completa delle prestazioni del modello.

Per visualizzare la relazione tra *Precision* e *Recall* a diverse soglie di confidenza, si utilizza la *Precision-Recall Curve* (PRC). La PRC rappresenta graficamente il bilanciamento tra *Precision* e *Recall* per vari livelli di soglia, evidenziando come il

modello mantenga l'equilibrio tra questi due aspetti. La forma della curva riflette la capacità del modello di mantenere alta *Precision* senza compromettere eccessivamente il *Recall*. Ad esempio, una curva ripida indica che il modello conserva alta *Precision* anche con incrementi di *Recall*, mentre una curva più piatta suggerisce un compromesso maggiore, con la *Precision* che cala rapidamente all'aumentare del *Recall*. In Figura 2.11 è stato riportato un esempio di curva PRC.

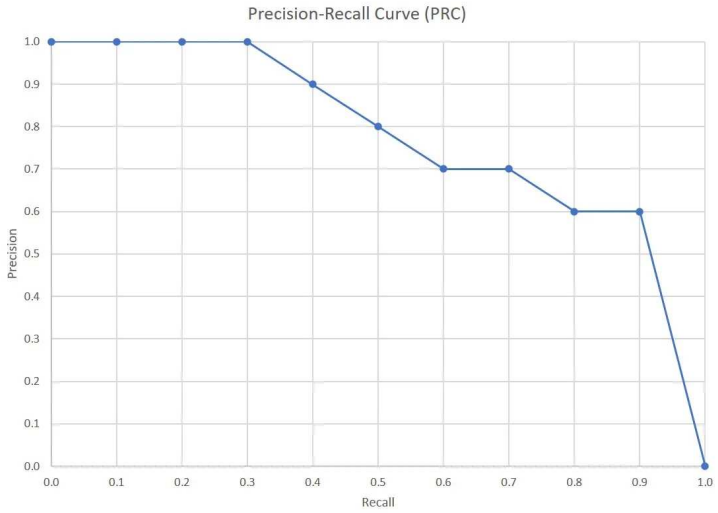

Figura 2.11.: Esempio di curva PRC considerando diversi valori di *Precision* e *Recall*

Dalla PRC, si ottiene l'*Average Precision* (AP), che rappresenta l'area sotto la curva e riassume l'efficacia complessiva del modello in termini di *Precision* e *Recall* su tutte le soglie. Quando viene espressa come un integrale, l'AP è calcolata come:

$$AP = \int_0^1 P(R)\, dR$$

dove $P(R)$ esprime la *Precision* in funzione della *Recall*, integrata sull'intervallo da $[0, 1]$. Nelle applicazioni pratiche, dove la PRC è spesso rappresentata da punti discreti, l'AP viene approssimata come una somma pesata delle variazioni di *Recall* moltiplicate per la *Precision* corrispondente:

$$AP = \sum_n (R_n - R_{n-1}) P_n$$

Un valore elevato di AP indica che il modello mantiene una buona precisione su una gamma di livelli di *Recall*, confermandone l'efficacia nel bilanciare le due metriche.

Nell'ambito dell'*object detection*, la metrica di valutazione più comunemente utilizzata è la *Mean Average Precision* (mAP), che fornisce una sintesi delle performance del modello su diverse soglie di confidenza e su diverse categorie o classi, offrendo

una valutazione completa delle sue prestazioni:

$$mAP = \frac{1}{k} \sum_{i=1}^{k} AP_i$$

dove k è il numero totale delle classi e AP_i è l'*Average Precision* per la classe i-esima. La mAP viene tipicamente calcolata in funzione della *Intersection over Union* (IoU), che misura la sovrapposizione tra il *bounding box* predetto dal modello e quello effettivo dell'oggetto. Il valore dell'IoU è fondamentale per valutare la capacità del modello di localizzare accuratamente un oggetto all'interno dell'immagine. Un IoU più alto indica una maggiore precisione nella localizzazione, mentre valori più bassi possono segnalare errori di localizzazione. In Figura 2.12 è stato riportato un esempio di IoU.

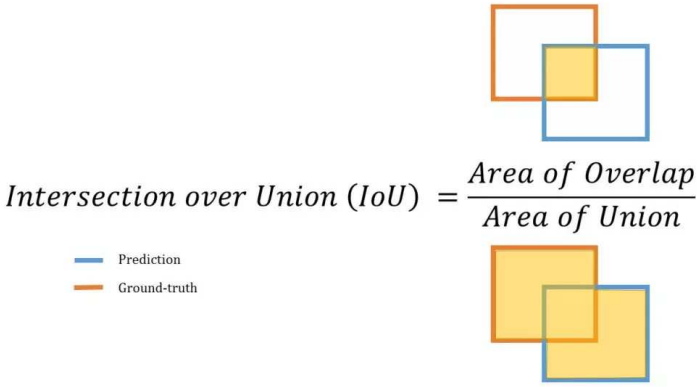

Figura 2.12.: Illustrazione del concetto di *Intersection Over Union*

Immagine da: `https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles/overview`

Quindi, di fatto, la mAP di fatto prende in considerazione diverse soglie di IoU, permettendo una valutazione dettagliata della robustezza del modello in scenari variabili. Un modello con un alto valore di mAP su una vasta gamma di soglie IoU è generalmente considerato più affidabile, poiché dimostra la capacità di localizzare oggetti con precisione anche in contesti complessi. Oltre alla mAP, altre metriche specifiche possono fornire una visione dettagliata delle prestazioni di un modello, come anche il *F1-score*, che combina *Precision* e *Recall* in un'unica metrica bilanciata, è vantaggioso in situazioni in cui è importante raggiungere un compromesso equilibrato tra entrambe le metriche.

2.2.5. Problemi noti

Nonostante i significativi progressi nel campo dell'*object detection*, permangono diverse sfide, soprattutto quando si applicano i modelli a scenari del mondo reale. Una delle problematiche principali è il sovra-adattamento (o *overfitting*). Questo

fenomeno si verifica quando un modello diventa troppo specifico rispetto ai dati di addestramento, compromettendo la sua capacità di generalizzare su dati nuovi. Il sovra-adattamento è particolarmente frequente nei modelli di deep learning, dove l'eccessiva complessità può portare il modello a catturare dettagli irrilevanti o rumore presente nei dati di addestramento, ottenendo prestazioni apparentemente ottimali su questi ultimi. Tuttavia, quando il modello viene testato su dati nuovi, non previsti, l'accuratezza tende a calare drasticamente. Per affrontare questo problema, è fondamentale disporre di dataset ampi e diversificati, anche se in pratica non è sempre possibile raccogliere un numero sufficiente di dati rappresentativi. Per mitigare l'*overfitting*, si utilizzano tecniche come la regolarizzazione, il Dropout o l'impiego di dataset sintetici, migliorando così la capacità del modello di generalizzare meglio.

All'opposto, si può riscontrare un fenomeno di sotto-adattamento (o *underfitting*), quando il modello è troppo semplice o non sufficientemente addestrato, e quindi non riesce a catturare in modo adeguato i pattern presenti nei dati di addestramento. Questo porta a prestazioni insoddisfacenti sia sui dati di addestramento che su quelli di test. L'*underfitting* può essere risolto aumentando la complessità del modello, migliorando la qualità e la varietà dei dati o ottimizzando i parametri di apprendimento.

In Figura 2.13 viene mostrato un esempio pratico di ciò che accade nei casi di *overfitting* e *underfitting*.

Un'altra sfida significativa è il *class imbalance*, che si verifica quando i dataset utilizzati per l'addestramento contengono un numero sproporzionato di esempi per alcune classi di oggetti rispetto ad altre. Questo squilibrio può influenzare negativamente le prestazioni del modello, poiché tende a favorire le classi più rappresentate. In scenari reali, è comune che classi come persone o veicoli siano molto più frequenti rispetto a classi meno comuni o più specifiche, portando il modello a ignorare o rilevare con minore accuratezza gli oggetti appartenenti a queste classi meno rappresentate.

Per affrontare questo problema, sono stati introdotti diversi approcci. Dal punto di vista dei dati, si possono utilizzare tecniche come il *re-sampling*, che consiste nell'aumentare il numero di esempi delle classi minoritarie o ridurre quello delle classi maggioritarie, oppure il *data augmentation*, che genera nuove istanze sintetiche delle classi meno rappresentate attraverso trasformazioni come rotazioni, traslazioni o cambiamenti di luminosità. In alternativa, è possibile intervenire sugli iperparametri del modello, assegnando un peso maggiore alle classi meno rappresentate durante l'addestramento, garantendo così una rilevazione più equilibrata e accurata.

2.3. Servizi di cloud computing

Dopo aver esplorato le tecniche di raccolta dati, essenziali per l'alimentazione dei modelli di rete neurale e aver analizzato più nel dettaglio le sofisticate applicazioni dell'*object detection*, è fondamentale considerare l'infrastruttura tecnologica che

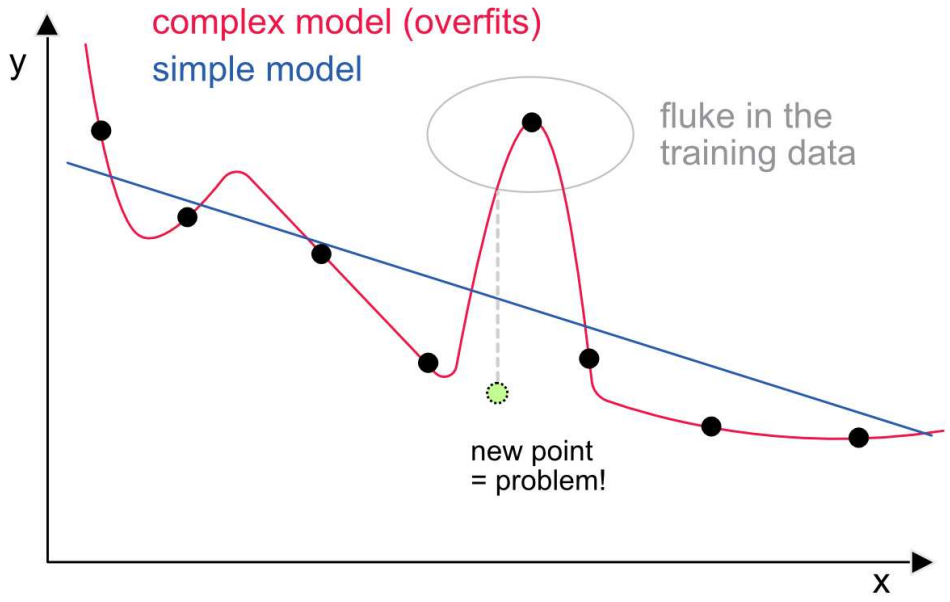

Figura 2.13.: Esempi dei fenomeni di overfitting e underfitting

Questa immagine illustra la differenza tra un modello complesso, che segue strettamente i dati di addestramento e rischia di sovradattarsi (*overfitting*), e un modello semplice, che rappresenta solo la tendenza generale (*underfitting*). Il modello complesso (in rosso) è più vulnerabile a valori anomali o errori nei dati, mentre il modello semplice (in blu) risulta più stabile quando vengono introdotti nuovi dati

rende possibile la gestione e l'elaborazione di queste grandi quantità di dati: il cloud computing. In un contesto in cui la raccolta di dati su larga scala e l'implementazione di modelli complessi di intelligenza artificiale richiedono risorse computazionali flessibili e scalabili, il *cloud computing* emerge come una soluzione indispensabile. Questa tecnologia non solo supporta l'elaborazione e l'archiviazione dei dati, ma facilita anche la distribuzione e il dispiegamento di modelli di intelligenza artificiale in ambienti produttivi. Il cloud computing rappresenta una delle evoluzioni più significative nell'ambito delle tecnologie informatiche moderne, permettendo alle organizzazioni di accedere a risorse computazionali potenti senza dover sostenere i costi e le complessità associate alla gestione di infrastrutture hardware proprie. Grazie a modelli di servizio flessibili e a una vasta gamma di opzioni di architettura, questa evoluzione è diventata il fondamento su cui molte aziende basano le loro strategie tecnologiche e di business. In Figura 2.14 è riportata un'illustrazione che mostra il ruolo centrale assunto dal cloud computing.

2.3.1. Overview e caratteristiche principali

Il cloud computing si distingue per una serie di caratteristiche uniche che lo rendono una delle tecnologie più rivoluzionarie e potenti nella gestione delle risorse informatiche. Una delle sue caratteristiche salienti è la scalabilità elastica, che con-

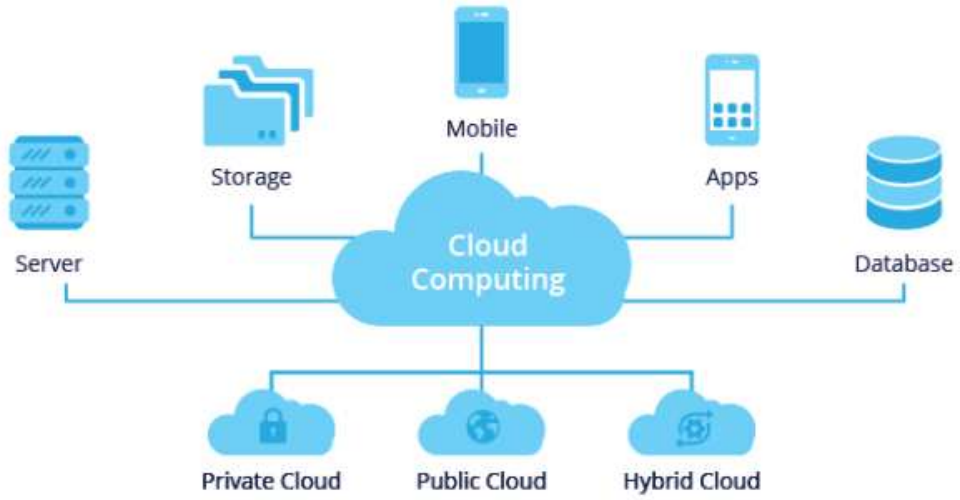

Figura 2.14.: Panoramica del cloud computing

sente alle aziende di modulare dinamicamente le risorse computazionali in base alle proprie esigenze, riducendo o aumentando rapidamente la capacità di calcolo, *storage* o connettività a seconda della domanda. In un mondo in cui le applicazioni devono spesso far fronte a picchi imprevedibili di utilizzo, la scalabilità elastica del cloud offre la flessibilità necessaria per rispondere in modo tempestivo ed efficiente. Ciò non solo migliora l'esperienza utente, ma riduce anche i costi associati al mantenimento di infrastrutture sovradimensionate. Un altro pilastro del cloud computing è il *self-service on-demand*, che permette agli utenti di accedere e configurare le risorse necessarie in modo autonomo, senza la necessità di intervento da parte dei team IT o dei fornitori di servizi. Questa caratteristica elimina i tempi di attesa tradizionalmente legati alla configurazione e al *provisioning* delle risorse, accelerando i cicli di sviluppo e favorendo un *time-to-market* molto più rapido. Gli sviluppatori possono così concentrarsi sulla realizzazione di prodotti e soluzioni, riducendo drasticamente le inefficienze legate alla gestione dell'infrastruttura. Il *multi-tenancy* è un'altra peculiarità distintiva del cloud computing, che permette a più utenti o organizzazioni di condividere le stesse risorse fisiche, mantenendo al contempo un ambiente logico sicuro e isolato per ciascun utente. Questo modello consente un utilizzo ottimale delle risorse, poiché massimizza l'efficienza di utilizzo dei *data center*, riducendo allo stesso tempo i costi operativi per gli utenti finali. Le aziende, così, possono accedere a risorse potenti senza dover sostenere gli elevati costi infrastrutturali che sarebbero altrimenti necessari per implementare soluzioni *in-house*. L'accessibilità ubiqua offerta dal cloud computing tramite reti globali rappresenta un altro vantaggio fondamentale, consentendo agli utenti di accedere alle risorse e ai servizi da qualsiasi luogo e dispositivo connesso a Internet. Questa caratteristica ha assunto una rilevanza crescente in un mondo sempre più interconnesso e orientato alla mobilità. In ambito aziendale, la possibilità

di garantire accesso continuo e sicuro alle risorse indipendentemente dalla posizione geografica degli utenti migliora l'agilità operativa e supporta nuovi modelli di lavoro, come il lavoro remoto e la collaborazione distribuita, oggi divenuti standard in molti settori. Un ulteriore aspetto che rende il cloud computing una scelta attraente per molte organizzazioni è il modello di *pricing* basato sull'uso effettivo, noto anche come *utility-based pricing*. Questo approccio permette di pagare solo per le risorse effettivamente utilizzate, un vantaggio economico notevole rispetto ai modelli tradizionali di *provisioning*, che richiedevano spesso investimenti ingenti in infrastrutture fisse, anche quando non erano utilizzate al massimo della capacità. Il cloud, invece, offre la possibilità di allineare i costi direttamente alle esigenze operative, riducendo gli sprechi e ottimizzando il budget IT. Tuttavia, questa straordinaria flessibilità comporta anche nuove sfide, soprattutto in termini di gestione e controllo dei costi. Man mano che le organizzazioni scalano le loro operazioni nel cloud, la molteplicità di servizi e risorse utilizzate può rendere complesso il monitoraggio e l'ottimizzazione dei costi. Per affrontare questo problema, sono stati sviluppati strumenti avanzati di *cloud management*, che consentono di monitorare l'utilizzo delle risorse in tempo reale, ottimizzare le prestazioni e prevedere con precisione le spese future. Questi strumenti diventano essenziali per garantire che l'adozione del cloud non comporti sorprese sgradite in termini di spese, permettendo alle organizzazioni di mantenere il pieno controllo delle proprie risorse e dei costi associati.

Grazie a queste caratteristiche, il cloud computing si configura come la base su cui si costruiscono soluzioni innovative e scalabili. Tuttavia, per comprendere appieno il suo potenziale, è fondamentale esplorare le diverse architetture e tipologie di cloud computing, che offrono modelli operativi differenti, ciascuno con vantaggi specifici in base alle esigenze organizzative e operative. In questo capitolo verranno esaminati in dettaglio tali aspetti, insieme ai principali provider che attualmente dominano questo settore.

2.3.2. Architetture e tipologie di cloud computing

L'architettura del cloud computing può essere vista come una stratificazione di servizi che forniscono diversi livelli di astrazione, dalla gestione dell'hardware fino all'esecuzione di applicazioni complesse. Questa struttura stratificata permette una gestione modulare delle risorse e consente di adattare le soluzioni tecnologiche alle esigenze specifiche di ciascuna organizzazione, migliorando la flessibilità e la scalabilità. Un modo efficace per comprendere l'architettura del cloud è suddividerla in tre componenti principali: hardware, virtualizzazione e applicazioni e servizi. Questi tre livelli non solo permettono di distribuire in modo efficiente le risorse cloud, ma offrono anche agli utenti diverse possibilità di controllo e personalizzazione, a seconda del tipo di modello di servizio scelto.

Alla base di ogni sistema cloud vi è l'infrastruttura fisica, ovvero l'hardware. Questo strato è composto da server, sistemi di archiviazione e dispositivi di rete,

che formano i *data center*. I *data center* cloud sono distribuiti su scala globale e costruiti per offrire alta disponibilità, ridondanza e tolleranza ai guasti. L'hardware sottostante è essenziale per garantire che le risorse cloud siano sempre disponibili e "performanti", indipendentemente dalla posizione geografica dell'utente o dal carico di lavoro in esecuzione. Nonostante la loro importanza, gli utenti del cloud raramente interagiscono direttamente con l'hardware. Grazie alla virtualizzazione, il livello fisico viene nascosto, permettendo agli utenti di accedere alle risorse senza preoccuparsi della manutenzione e gestione dei server fisici. In questo modo, l'hardware diventa una risorsa gestita in background, scalabile in base alle necessità operative.

Il secondo strato dell'architettura cloud è rappresentato dalla virtualizzazione, che funge da collegamento tra l'hardware fisico e le risorse computazionali utilizzabili. La virtualizzazione permette di creare macchine virtuali (VM) o *container* che possono operare in modo indipendente, isolati gli uni dagli altri. Grazie a tecnologie di virtualizzazione avanzate, è possibile far funzionare più sistemi operativi e applicazioni su un singolo server fisico, migliorando l'efficienza e riducendo i costi. La virtualizzazione rappresenta il cuore dell'agilità del cloud, consentendo una gestione flessibile delle risorse. Le risorse hardware possono essere allocate dinamicamente in base alla domanda, garantendo che le applicazioni possano scalare facilmente senza bisogno di *provisioning* manuale. Inoltre, la virtualizzazione offre maggiore sicurezza e isolamento tra gli utenti, anche quando utilizzano lo stesso hardware.

Il livello più alto dell'architettura cloud è quello delle applicazioni e servizi, che rappresenta l'interfaccia attraverso cui le organizzazioni interagiscono con il cloud. A questo livello, gli utenti possono sviluppare, eseguire e distribuire le loro applicazioni senza dover gestire l'infrastruttura sottostante. I servizi offerti a questo livello variano notevolmente a seconda del modello di distribuzione o del modello di servizio scelto, quelli principali sono stati riportanti nell'illustrazione in Figura 2.15 e sono i seguenti:

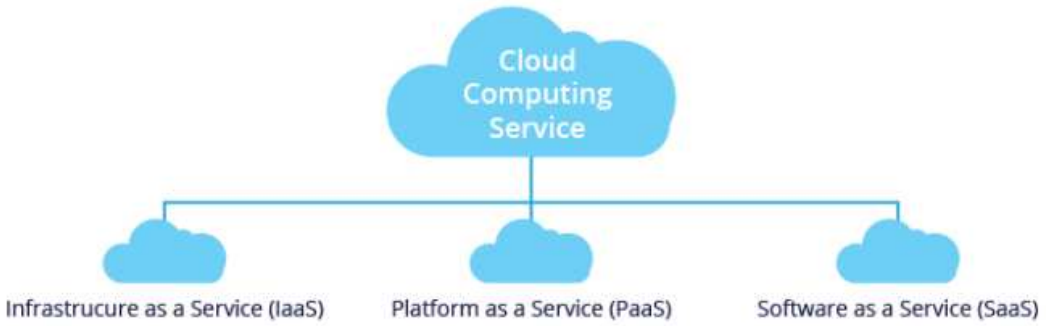

Figura 2.15.: Modelli di servizi Cloud principali

- Infrastructure as a Service (IaaS): è il modello più basilare di servizio cloud e fornisce accesso su richiesta alle risorse infrastrutturali come server virtualizzati, *storage* e *networking*. Con IaaS, le organizzazioni possono creare e gestire interi

ambienti IT virtualizzati, configurabili e scalabili, senza dover acquistare o gestire l'hardware fisico. Le aziende possono così concentrarsi sulle applicazioni e sui carichi di lavoro, delegando la gestione dell'infrastruttura al fornitore di servizi cloud. Questo modello offre la massima flessibilità in termini di configurazione e controllo, ma richiede competenze tecniche per gestire la parte infrastrutturale. È una soluzione ideale per aziende che necessitano di risorse computazionali dinamiche e scalabili, o per quelle che devono affrontare carichi di lavoro variabili senza voler investire in infrastrutture proprietarie.

- Platform as a Service (PaaS): fornisce una piattaforma completa che include non solo le risorse infrastrutturali, ma anche strumenti e ambienti di sviluppo che consentono agli utenti di creare, testare e distribuire applicazioni. Questo modello semplifica enormemente il processo di sviluppo software, poiché gli sviluppatori possono concentrarsi esclusivamente sul codice e sulle funzionalità delle loro applicazioni, mentre il provider di PaaS gestisce l'infrastruttura, la scalabilità e la sicurezza. PaaS è particolarmente utile per le aziende che desiderano accelerare il ciclo di vita dello sviluppo, riducendo i tempi e i costi di configurazione dell'ambiente. Questo modello include strumenti integrati per il monitoraggio delle prestazioni, la gestione dei database e l'integrazione con altri servizi, rendendolo una scelta ideale per progetti agili e per ambienti di sviluppo in continua evoluzione.

- Software as a Service (SaaS): è il modello di cloud computing più accessibile per gli utenti finali. In questo caso, le applicazioni sono completamente gestite e fornite dal provider di servizi cloud, accessibili via Internet senza dover installare o mantenere software localmente. Gli utenti possono accedere alle applicazioni SaaS da qualsiasi dispositivo connesso a Internet, senza preoccuparsi di aggiornamenti, sicurezza o manutenzione. Esempi comuni di SaaS includono applicazioni come `Google Workspace`[6] o `Microsoft 365`[7], che offrono strumenti di produttività direttamente via web. Il SaaS è particolarmente vantaggioso per le aziende che cercano una soluzione rapida e priva di complessità per la gestione di applicazioni e servizi, eliminando il bisogno di risorse IT dedicate.

Oltre ai modelli di servizio, il cloud computing può essere classificato in diversi tipi basati sul modello di distribuzione, ciascuno con vantaggi specifici a seconda delle esigenze aziendali.

- Il Cloud pubblico utilizza risorse di cloud computing e un'infrastruttura fisica di proprietà e gestita da un fornitore di servizi cloud di terze parti. I cloud pubblici consentono di scalare facilmente le risorse senza dover investire in

[6] `https://workspace.google.com/intl/it/`
[7] `https://microsoft365.com/`

hardware o software, ma utilizzano architetture *multi-tenant* che forniscono contemporaneamente servizi ad altri clienti.

- Il Cloud privato, al contrario, è un cloud dedicato di proprietà e gestito dalla tua organizzazione. Poiché è ospitata privatamente on-premise nel tuo *data center*, offre un maggiore controllo sulle risorse e una maggiore sicurezza per dati e infrastruttura. Tuttavia, questa architettura è notevolmente più costosa e la sua manutenzione richiede maggiori competenze IT.

- Il Cloud ibrido utilizza sia l'architettura cloud pubblica sia quella privata per fornire una combinazione flessibile di servizi cloud. Un cloud ibrido consente di eseguire la migrazione dei carichi di lavoro tra ambienti, permettendoti di utilizzare i servizi più adatti alle tue esigenze aziendali e al carico di lavoro. Le architetture cloud ibride sono spesso la soluzione preferita dalle aziende che vogliono controllare i propri dati, ma anche sfruttare i vantaggi del cloud pubblico. Negli ultimi anni, con il numero crescente di organizzazioni che utilizzano servizi cloud di più cloud provider, sta emergendo anche l'architettura multi-cloud. Gli ambienti multi-cloud si stanno diffondendo sempre più per la loro flessibilità e capacità di conciliare meglio casi d'uso e offerte specifiche, indipendentemente dal fornitore.

Ogni azienda può scegliere il modello di servizio e distribuzione che meglio risponde alle proprie esigenze, bilanciando fattori come costi, sicurezza, prestazioni e scalabilità. Per implementare efficacemente queste soluzioni, è fondamentale conoscere però i principali provider di servizi cloud. Nel capitolo successivo verranno esplorati i leader del settore, come Amazon Web Services (AWS), Microsoft Azure e Google Cloud Platform (GCP), analizzando le loro offerte e i loro punti di forza in un mercato in continua evoluzione.

2.3.3. I principali provider di servizi Cloud

Il mercato del cloud computing è dominato da alcuni grandi provider, ciascuno dei quali offre una vasta gamma di servizi e soluzioni per rispondere alle diverse esigenze delle organizzazioni. Questi provider hanno sviluppato infrastrutture globali, composte da centinaia di *data center* distribuiti in tutto il mondo, garantendo prestazioni elevate, affidabilità e disponibilità dei servizi. In Figura 2.16 è riportato il contributo dei principali provider di servizi cloud, che sono:

- Amazon Web Services (AWS), lanciato nel 2006, è il pioniere e leader del mercato cloud, detenendo circa il 32% della quota globale. La sua forza risiede nell'ampiezza e nella profondità dell'offerta: conta infatti centinaia di servizi, dai più tradizionali come il *computing* (`Amazon EC2`[8]) e lo *storage* (`Amazon`

[8] `https://aws.amazon.com/it/ec2/`

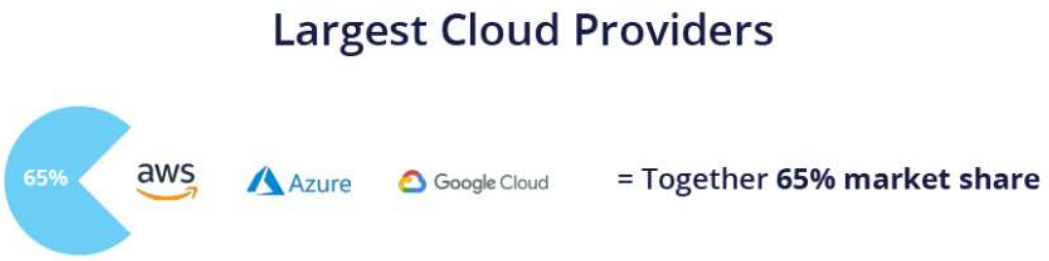

Figura 2.16.: Distribuzione dei provider di servivi Cloud

S3[9]), fino a strumenti avanzati per il machine learning (AWS SageMaker[10]), l'IoT, l'analisi dei dati e molto altro. Questa vastità fa di AWS una scelta flessibile per le aziende di ogni dimensione, che possono sfruttare la potenza della sua infrastruttura globale e l'integrazione con strumenti di automazione e gestione delle risorse. AWS si distingue anche per la sua struttura globale di *data center*, distribuiti in più regioni e zone di disponibilità. Questo garantisce non solo affidabilità e ridondanza per ridurre i tempi di inattività, ma anche una bassa latenza per gli utenti, specialmente quelli che hanno bisogno di distribuire applicazioni in più aree geografiche. Tuttavia, proprio per la sua vastità, AWS può risultare complesso da navigare per le organizzazioni meno esperte. Sebbene disponga di un'ottima interfaccia gestionale, la varietà di servizi e opzioni può essere intimidatoria per chi non è abituato alla complessità del cloud pubblico. Anche in termini di costi, AWS offre diversi modelli di pagamento, ma è facile superare il budget iniziale se non si presta attenzione a una corretta ottimizzazione e monitoraggio dei consumi.

- Microsoft Azure: Azure, nato nel 2010, ha rapidamente guadagnato terreno, grazie alla sua forte integrazione con il vasto ecosistema di Microsoft. Con una quota di mercato del 22%, Azure si distingue per la capacità di supportare ambienti ibridi, combinando le infrastrutture *on-premises* e cloud in modo fluido, grazie a strumenti come Azure Arc[11] e Azure Stack[12]. Questo rende Azure una scelta privilegiata per quelle aziende che già operano in ambienti Windows o che utilizzano prodotti Microsoft come Windows Server[13], o Office 365[14].

  Un altro aspetto che rende Azure particolarmente attraente è la facilità con cui supporta infrastrutture multi-cloud e soluzioni ibride, consentendo alle aziende di migrare carichi di lavoro in modo graduale e di mantenere il controllo su risorse sensibili, senza compromettere la scalabilità e l'agilità offerte dal cloud pubblico. Azure eccelle anche nelle soluzioni di machine learning, con

[9] https://aws.amazon.com/it/s3/
[10] https://aws.amazon.com/it/sagemaker/
[11] https://azure.microsoft.com/it-it/products/azure-arc
[12] https://azure.microsoft.com/it-it/products/azure-stack
[13] https://www.microsoft.com/it-it/windows-server
[14] https://www.microsoft.com/it-it/microsoft-365

`Azure Machine Learning`[15] che permette agli sviluppatori di creare, testare e distribuire algoritmi AI con facilità. In termini di prezzi, Azure è altamente competitivo, offrendo modelli di pagamento *pay-as-you-go* e *reserved instances*, analogamente ad AWS, ma con la particolarità di essere particolarmente economico per le organizzazioni che già utilizzano prodotti Microsoft. Tuttavia, per le aziende non legate all'ecosistema Microsoft, Azure potrebbe non essere la prima scelta a causa della sua integrazione meno immediata con strumenti e tecnologie esterne.

- Google Cloud Platform (GCP): Google Cloud, benché sia entrato più tardi nel mercato rispetto ai suoi concorrenti, si distingue per la sua eccellenza in settori specifici, in particolare per quanto riguarda il machine learning e l'analisi dei dati. Con circa l'11% di quota di mercato, GCP è particolarmente apprezzato dalle aziende *data-driven*, che necessitano di strumenti avanzati per gestire enormi volumi di dati e sfruttare algoritmi di intelligenza artificiale grazie a tecnologie come `TensorFlow`[16], sviluppata dallo stesso Google. Un altro punto di forza di GCP è la rete globale di *data center* ad alta velocità, che garantisce prestazioni elevate e bassa latenza, risultando particolarmente vantaggiosa per applicazioni ad alto carico computazionale o per progetti che richiedono una scalabilità significativa. GCP è anche noto per la sua attenzione all'usabilità: l'interfaccia utente di Google Cloud è intuitiva e semplificata, il che lo rende accessibile anche a sviluppatori meno esperti. Dal punto di vista dei costi, GCP adotta una strategia *pay-per-use* simile ai suoi concorrenti, ma si distingue per l'offerta di sconti per l'uso continuato e modelli tariffari trasparenti che aiutano a prevedere con maggiore precisione le spese.

È evidente come, al giorno d'oggi, i provider di servizi Cloud offrano soluzioni simili tra loro, con le principali differenze che emergono in termini di integrazione, facilità d'uso e funzionalità specifiche. Tra questi appena trattati, AWS domina per la vasta gamma di servizi e la presenza globale, ma può risultare più complesso da gestire senza una formazione adeguata; Azure rappresenta la scelta ideale per chi già utilizza prodotti Microsoft, grazie alle soluzioni ibride e alla facile integrazione; mentre Google Cloud si distingue per la leadership nel machine learning e nell'analisi avanzata dei dati, offrendo anche un'interfaccia utente più intuitiva.

2.3.4. Sfide e opportunità nel cloud computing

Nonostante i numerosi vantaggi che il cloud computing offre alle organizzazioni, restano ancora molte sfide che ne condizionano l'adozione e l'ottimizzazione, soprattutto man mano che le infrastrutture e i servizi cloud diventano sempre più complessi. Allo stesso tempo, queste difficoltà rappresentano opportunità per migliorare e innovare, attraverso lo sviluppo di nuove tecnologie e pratiche. Una delle sfide principali

[15] `https://azure.microsoft.com/en-us/products/machine-learning`
[16] `https://www.tensorflow.org/?hl=it`

è senza dubbio la sicurezza dei dati. Dal momento che le informazioni vengono archiviate e processate in infrastrutture di terze parti, esiste una costante preoccupazione riguardo alla protezione dei dati sensibili e alla conformità alle normative locali e internazionali. Le organizzazioni devono adottare misure rigorose di sicurezza, come la crittografia dei dati, il controllo degli accessi e la gestione delle chiavi di sicurezza, per evitare violazioni o perdite di dati. Oltre a ciò, il modello di responsabilità condivisa tra il fornitore di servizi cloud e l'utente finale richiede che entrambe le parti siano pienamente consapevoli di chi è responsabile di ciascun aspetto della sicurezza.

Un altro aspetto critico è la gestione delle risorse. Poiché le risorse cloud, soprattutto nei contesti di cloud pubblico, possono essere allocate e deallocate dinamicamente, la loro gestione efficiente diventa essenziale per evitare sprechi e ottimizzare i costi. Gli strumenti di monitoraggio in tempo reale e le pratiche di automazione giocano un ruolo chiave per garantire che l'allocazione delle risorse avvenga in maniera ottimale. In assenza di tali strumenti, le aziende potrebbero incorrere in costi elevati dovuti all'eccessiva allocazione di risorse, o rischiare una diminuzione delle prestazioni per mancanza di capacità. A questo si aggiunge il crescente fenomeno dell'eterogeneità delle risorse e delle applicazioni. Con l'aumento dell'adozione di modelli multi-cloud e *hybrid-cloud*, le organizzazioni devono gestire ambienti complessi che includono diverse piattaforme, standard e protocolli. L'integrazione dei sistemi *legacy* con le moderne infrastrutture cloud richiede spesso competenze specialistiche, e la mancanza di interoperabilità tra i vari fornitori può complicare ulteriormente la gestione. Tuttavia, questo apre anche nuove opportunità di innovazione: il mercato del *cloud management* sta sviluppando strumenti avanzati per facilitare l'interoperabilità e migliorare la gestione delle risorse tra ambienti cloud diversi.

Un'altra importante sfida è il controllo dei costi. Se da un lato il modello *pay-as-you-go* consente alle organizzazioni di pagare solo per le risorse effettivamente utilizzate, dall'altro, la proliferazione di servizi e l'utilizzo non monitorato di risorse possono portare a sorprese spiacevoli in termini di spese. Per risolvere questa problematica, sono stati introdotti strumenti di ottimizzazione dei costi che permettono di monitorare l'utilizzo delle risorse in tempo reale e prevedere con precisione le spese future.

Tra le opportunità più promettenti, vi è l'uso dell'intelligenza artificiale per la gestione del cloud. Le tecniche di machine learning possono essere impiegate per prevedere la domanda di risorse, ottimizzare l'allocazione e automatizzare la gestione della sicurezza, riducendo così la necessità di intervento umano. Questa tendenza è particolarmente rilevante in applicazioni come l'*Internet of Things* (IoT), dove i carichi di lavoro possono variare in modo significativo e improvviso[22]. L'adozione di questo approccio consente di garantire scalabilità ed elasticità automatica, un'area di ricerca in continua evoluzione, con lo sviluppo di meccanismi sempre più sofisticati che permettono alle applicazioni cloud di adattarsi automaticamente alle variazioni di carico, senza interventi manuali. Il futuro del cloud computing vedrà un'attenzione

crescente all'interoperabilità e alla portabilità tra piattaforme cloud, specialmente con la diffusione delle architetture multi-cloud. Le aziende vogliono evitare il *vendor lock-in*, ovvero la dipendenza da un singolo fornitore, per mantenere flessibilità nella gestione delle risorse e adattarsi rapidamente alle esigenze di mercato in costante evoluzione.

Capitolo 3.

Materiali e metodi

3.1. Creazione del dataset con Selenium Webdriver

Per la creazione del dataset si è scelto di utilizzare `Selenium WebDriver`, uno strumento di automazione dei browser che si è rivelato particolarmente adatto alle esigenze di raccolta dati di questo progetto di tesi. L'obiettivo era quello di estrarre informazioni da pagine web dinamiche, dove altre tecniche di *scraping* si sarebbero dimostrate insufficienti. `Selenium` è stato selezionato per la sua capacità di simulare il comportamento di un utente reale, interagendo direttamente con il *browser* e permettendo di eseguire azioni complesse come la navigazione, l'interazione con moduli e il caricamento di contenuti, rendendolo un'opzione estremamente versatile. `Selenium` è infatti un *framework open-source* che consente l'automazione dei *browser web*, nato principalmente per eseguire test automatizzati su applicazioni web, ma estremamente efficace anche per il *web scraping*, già discusso nel capitolo precedente. E' comunque importante comprendere l'architettura di tale *framework* per cogliere appieno il motivo per cui è stato scelto rispetto ad altre opzioni. In Figura 3.1 si

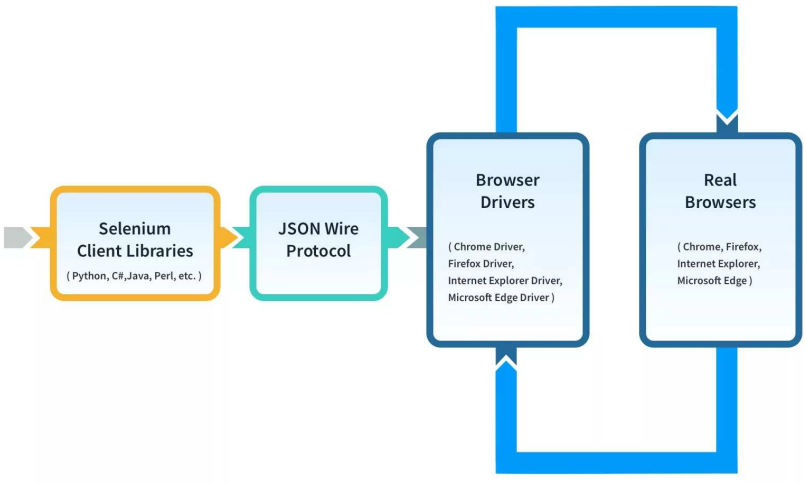

Figura 3.1.: Architettura `Selenium WebDriver`

Immagine da: `https://dev.to/selvakumar/selenium-architecture-3pm2`

può osservare il flusso di comunicazione tra il *client* e il *browser*, facilitato da un driver che traduce i comandi del codice in azioni reali eseguite dal *browser*. Questa architettura consente di eseguire operazioni come cliccare su bottoni, inserire dati nei moduli, navigare tra pagine web e gestire finestre pop-up o dialoghi. La separazione tra il *client* e il *browser*, tramite il *JSON Wire Protocol* e i *browser driver*, permette di eseguire test e raccogliere dati su diversi *browser* senza necessità di modificare il codice. Ciò rende `Selenium` uno strumento utile per garantire la compatibilità cross-*browser* e semplificare lo sviluppo di applicazioni web robuste e testabili su piattaforme diverse.

Tra i principali benefici del *framework* c'è la sua capacità di gestire pagine dinamiche. Spesso, infatti, i siti moderni non caricano tutti i contenuti all'avvio, ma utilizzano tecniche asincrone come AJAX (*Asynchronous JavaScript and XML*) per caricare dati man mano che l'utente si interfaccia con la pagina. `Selenium`, grazie alla sua capacità di interagire con il *browser* in modo simile a un utente reale, consente di attendere che gli elementi dinamici siano caricati e pronti per l'uso prima di eseguire azioni. Questo aspetto è particolarmente importante per raccogliere dati da pagine che aggiornano i loro contenuti continuamente o che richiedono input manuali per accedere a determinate sezioni. Un altro importante vantaggio riguarda la possibilità di automatizzare interazioni complesse. Molte delle pagine da cui sono stati estratti i dati richiedevano selezioni da menu a tendina, clic su pulsanti e invio di form. `Selenium` si è dimostrato estremamente utile per automatizzare tutte queste operazioni in modo fluido e senza richiedere intervento manuale. Inoltre, la scelta di utilizzare Python[1] come linguaggio di programmazione ha semplificato ulteriormente il processo, grazie alla sua leggibilità e all'ampia disponibilità di librerie per l'elaborazione dei dati. Nonostante questi evidenti vantaggi, è importante tenere in considerazione alcune limitazioni che `Selenium WebDriver` presenta. Uno dei principali limiti è la velocità. A differenza di altri strumenti di *scraping* più specializzati, come BeautifulSoup[2] o `Scrapy`, `Selenium` simula un *browser* reale, ciò significa che ogni pagina web deve essere effettivamente caricata come se fosse visualizzata da un utente. Questo processo può richiedere tempo, soprattutto su siti particolarmente pesanti o con molte risorse da caricare. L'emulazione del *browser* richiede inoltre un consumo significativo di risorse di sistema, che può diventare un problema se si ha la necessità di eseguire molteplici sessioni di *scraping* contemporaneamente o su larga scala. Un'altra difficoltà incontrata riguarda le barriere *anti-scraping* implementate da molti siti web. Alcuni siti utilizzano CAPTCHA o altre forme di protezione per impedire l'automazione delle interazioni. Anche se `Selenium` è in grado di gestire la maggior parte delle operazioni standard, affrontare queste protezioni richiede tecniche avanzate o l'integrazione con altri strumenti per superare tali ostacoli. Nonostante queste limitazioni, la scelta di utilizzare `Selenium WebDriver` si è rivelata la più appropriata per il progetto, grazie alla sua flessibilità, potenza nelle interazioni

[1] `https://www.python.org/`
[2] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

complesse con gli elementi delle pagine web e all'alta possibilità di controllo. Altri strumenti, pur essendo più veloci e meno onerosi in termini di risorse, non avrebbero offerto la stessa capacità di gestire pagine web di questo tipo o di automatizzare interazioni più articolate. Inoltre, la necessità di garantire una compatibilità *cross-browser* e di simulare con precisione le azioni di un utente ha reso `Selenium` la scelta ideale per la creazione di un dataset completo e accurato.

In Figura 3.2 è stato riportato uno schema ad alto livello dell'approccio utilizzato per la realizzazione del dataset, a partire dallo *scraping* di dati fino all'annotazione delle immagini e la suddivisione di queste in set di dati ben distinti, che verrà trattato nel seguente capitolo.

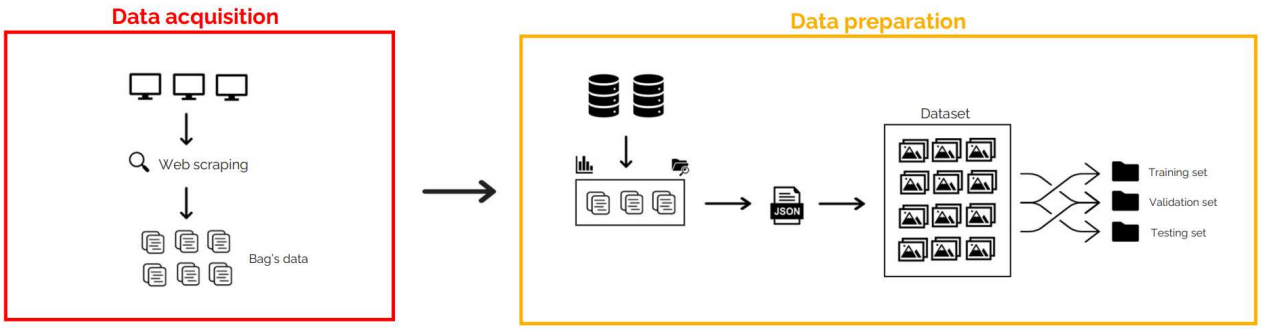

Figura 3.2.: Workflow adottato per la realizzazione del dataset di riferimento

3.2. Preprocessing e preparazione del dataset

Per quanto riguarda la preparazione del dataset è stato seguito un approccio a tre fasi ben distinte, le quali hanno richiesto l'utilizzo di `Selenium WebDriver` per automatizzare l'estrazione dei dati dai siti web dei principali rivenditori di moda di lusso. Questo processo ha permesso di raccogliere in modo efficiente e strutturato informazioni dettagliate su prodotti specifici, con particolare riferimento alle borse, per vari brand e regioni del mondo. L'obiettivo principale è stato quello di ottenere un dataset completo e chiaro, pronto per le successive fasi di analisi e addestramento dei modelli.

Nella prima fase della raccolta dati, si è proceduto all'estrazione dei link relativi ai prodotti presenti sui siti dei rivenditori. Per ogni combinazione di brand (Balenciaga, Gucci, Louis Vuitton, Saint Laurent) e regione (Italia, USA, Cina, Giappone, Corea del Sud), è stato generato un URL di base. Questo URL ha consentito di accedere alla pagina del brand specifico all'interno della regione selezionata. Da qui, `Selenium` ha permesso di automatizzare il processo di navigazione, accettazione dei *cookies*, chiusura delle finestre *pop-up* (ad esempio *newsletter* e preferenze di spedizione), e lo scorrimento della pagina per caricare tutti i prodotti visibili. Una volta caricati tutti i prodotti per ogni pagina, è stato possibile individuare i link delle borse tramite selettori HTML (usando i cosiddetti XPath) e salvarli in un file CSV. Ogni

link è stato accompagnato da informazioni aggiuntive come il brand, la regione e il genere del prodotto (uomo/donna). Questo processo è stato ripetuto per ogni pagina relativa a ciascun brand, in modo da garantire che venissero raccolti i link di tutte le borse presenti. La seconda fase ha riguardato l'estrazione dei metadati dei prodotti associati ai link raccolti nella fase precedente. In questo step, si è proceduto a leggere i file CSV contenenti i link e a filtrare i dati per regione. Una volta ottenuto l'elenco filtrato, si è avviato il processo di *scraping* dei singoli prodotti per raccogliere i dettagli specifici. `Selenium` ha consentito di aprire ciascun link, accedere alla pagina del prodotto e estrarre informazioni rilevanti come immagini, prezzo, prezzo originale (senza sconto), percentuale di sconto applicata, descrizione del prodotto, materiali, dimensioni, colore dominante (sia in formato RGB che come nome), e altre caratteristiche visive e testuali, opportunamente riportate in file JSON strutturati.

Un elemento critico di questa fase è stato il controllo delle sessioni interrotte. Prima di iniziare la raccolta dei dati per una regione, è stato verificato se fossero presenti file JSON che non contenevano ancora tutti i dati previsti. In particolare, il sistema controllava se ci fossero ancora link a borse, elencati nei file CSV di riferimento, che non erano stati elaborati. In caso positivo, il sistema avrebbe ripreso la raccolta dai link ancora da processare, evitando così la duplicazione dei dati già raccolti. Questo approccio è stato reiterato più volte nel tempo, circa due mesi, per poter estrapolare più informazioni possibili sulle borse disponibili in quel periodo. La scelta di questo approccio per la raccolta dati si è rivelata strategica per garantire efficienza e controllo. Separare l'estrazione degli URL dalla raccolta dei metadati ha permesso di ottimizzare le risorse computazionali, riducendo il rischio di errori e interruzioni. In questo modo, è stato possibile riprendere il processo da dove si era interrotto, evitando duplicazioni e perdite di dati. Perdipiù tale suddivisione ha consentito di aggiornare facilmente il dataset nel tempo, raccogliendo nuovi URL senza dover ripetere l'intero processo. La modularità del metodo ha migliorato la gestione della qualità dei dati, rendendo più facile apportare ottimizzazioni e garantendo un dataset accurato e aggiornato.

Una volta terminata la raccolta, si è passati alla manipolazione e trasformazione dei dati. Queste attività hanno previsto diversi passaggi cruciali per assicurare che il dataset fosse pulito, coerente e pronto per l'analisi. In primo luogo, si è provveduto a unire i vari file JSON generati durante la raccolta, consolidando tutte le informazioni in un unico dataset. Successivamente, è stato necessario eseguire una pulizia approfondita dei dati, che ha incluso la gestione di eventuali valori mancanti (ad esempio, prodotti senza descrizione completa o con informazioni sui materiali non specificate) e la standardizzazione dei campi in formati adeguati, come la conversione delle valute in un formato unificato per facilitare il confronto dei prezzi tra le diverse regioni. Le informazioni raccolte sono state organizzate in colonne ben definite, permettendo così di effettuare analisi comparative tra le borse in base a vari criteri, come il tipo di prodotto, il brand e la fascia di prezzo. Il dataset finale risultante da questo processo, ottenuto in formato JSON, è estremamente ricco di informazioni.

Esso include un'ampia gamma di metadati per ciascun prodotto, tra cui:

- Caratteristiche generali, come il genere (uomo/donna), brand, regione geografica, tipo e sottotipo del prodotto, nome del prodotto, URL del prodotto, data e ora della raccolta dei dati, e ID univoco del prodotto. È stato inoltre incluso lo stato del prodotto (ad esempio, nuovi arrivi, saldi, ecc.).
- Informazioni relative ai prezzi, come il prezzo corrente, il prezzo originale (senza sconto) e la percentuale di sconto. Inoltre, i prodotti sono stati classificati in cluster di prezzo per facilitarne l'analisi in base alla fascia di costo.
- Il link all'immagine del prodotto ed il colore dominante della borsa (sia in formato RGB che come nome del colore).
- Alcune caratteristiche descrittive riportate dai rivenditori, come le descrizioni dei prodotti e i materiali utilizzati nella loro produzione.
- Dimensioni: laddove disponibili, sono state incluse anche le dimensioni fisiche del prodotto, insieme ad altre informazioni sulle misure.

Tutti questi metadati forniscono una visione dettagliata dei prodotti disponibili sui principali *e-tailer* scelti, consentendo un'analisi approfondita delle tendenze di mercato, delle strategie di prezzo e della disponibilità dei prodotti nei diversi mercati geografici. Questo affinché la scrupolosità adottata nella raccolta e realizzazione del dataset permetta di impiegarlo in altre analisi e studi indipendenti da questo progetto.

Nella terza e ultima fase, si è proceduto allo *download* e all'analisi automatizzata delle immagini associate a ciascuna borsa, con l'obiettivo di estrapolare i *bounding box* relativi. Questo processo è stato realizzato mediante l'uso della libreria `OpenCV`[3] di *computer vision* in Python, che ha facilitato l'estrazione dei *bounding box* per ciascuna borsa. In particolare, l'approccio prevedeva la conversione delle immagini in scala di grigi e l'applicazione di una soglia binaria (utilizzando la tecnica di Otsu[23]), per evidenziare i contorni della borsa. Successivamente, tramite la funzione "*findContours*" di `OpenCV`, sono stati identificati i contorni principali, consentendo così di delineare automaticamente i *bounding box* attorno agli oggetti di interesse. Questa metodologia ha permesso di ridurre il bisogno di annotazione manuale, limitando l'intervento umano alla sola verifica dei risultati generati automaticamente. Il dataset finale è stato quindi convertito nel formato COCO, comunemente utilizzato per i modelli di rilevamento di oggetti. La scelta di questo formato è legata ai modelli e ai pesi pre-addestrati impiegati, che verranno approfonditi nei capitoli successivi. In Figura 3.3 è riportato un esempio di tali annotazioni in formato JSON.

Il dataset finale ottenuto attraverso il processo di raccolta e preparazione comprende 5,998 immagini di borse (ed altrettante annotazioni), organizzate nelle seguenti 9 classi, suggerite da esperti di dominio:

[3] https://opencv.org/

Figura 3.3.: Annotazioni generate in formato MS COCO

- *Backpacks* (Zaini): 172 immagini;
- *Belt bags* (Marsupi): 117 immagini;
- *Clutch bags* (Pochette): 253 immagini
- *Cross-body bags* (Borse a tracolla): 696 immagini
- *Handbags* (Borse a mano): 348 immagini
- *Mini bags* (Mini borse): 310 immagini
- *Shoulder bags* (Borse a spalla): 2.779 immagini
- *Tote bags* (Borse shopper): 1.163 immagini
- *Travel bags* (Borse da viaggio): 160 immagini

In Figura 3.4 sono mostrati alcuni estratti del dataset di immagini creato.

Dalla distribuzione del numero di *sample* per classe, è evidente uno sbilanciamento tale per cui alcune risultano rappresentate meglio di altre. Questo squilibrio complica la capacità dei modelli di generalizzare su dati non visti durante la fase di addestramento, aumentando così il rischio di *overfitting*. Per affrontare questa problematica, sono state applicate tecniche specifiche volte a mitigare l'impatto negativo dello sbilanciamento a livello di dataset, e sono stati considerati anche accorgimenti relativi agli iperparametri di addestramento, di cui si discuterà nel capitolo 3.4.

Il dataset ottenuto è stato poi suddiviso in 80% *training set*, 10% *validation set* e il restante 10% *test set*. Sulla porzione di dataset utilizzata per l'addestramento dei

(a) *Backpacks*

(b) *Belt bags*

(c) *Clutch bags*

(d) *Cross-body bags*

(e) *Handbags*

(f) *Mini bags*

(g) *Shoulder bags*

(h) *Tote bags*

(i) *Travel bags*

Figura 3.4.: Ecco alcune immagini estratte dal dataset ottenuto, distinte per classe di borsa

modelli, è stato quindi applicato un *oversampling* dei campioni delle classi minoritarie attraverso l'impiego di tecniche di *data augmentation*. In altre parole, tali tecniche sono state applicate in modo mirato per aumentare il numero di campioni delle classi meno rappresentate, anziché limitarsi a un semplice riequilibrio attraverso la duplicazione delle immagini originali. Questo ha permesso di creare una maggiore varietà all'interno delle classi minoritarie, migliorando la capacità del modello di generalizzare anche su queste categorie meno frequenti. La scelta del numero di campioni aggiuntivi per ciascuna classe minoritaria è stata effettuata sulla base di un calcolo mirato a bilanciare in modo efficace il dataset. Il numero di campioni aggiuntivi è stato determinato calcolando il numero medio di istanze per classe, così da avere un riferimento comune. Per ogni classe minoritaria, sono stati generati nuovi campioni fino a raggiungere questo valore medio, in modo da ridurre lo squilibrio con le classi più rappresentate. Questo approccio ha permesso di riequilibrare il dataset mantenendo un numero proporzionale di campioni per ogni classe, evitando sia una sovra-rappresentazione delle classi minoritarie sia una duplicazione indiscriminata dei campioni esistenti. Grazie a questa strategia, è stato possibile garantire una distribuzione più equa dei campioni, migliorando l'efficacia dell'addestramento e riducendo il rischio di *overfitting* senza sovraccaricare il modello con dati ridondanti. La *data augmentation* è una tecnica che consente di generare nuove immagini a partire da quelle esistenti attraverso trasformazioni geometriche e altre manipolazioni, aumentando così la diversità del dataset e migliorando la capacità del modello di affrontare variabilità reali. Nel caso specifico di questo lavoro, sono state utilizzate diverse trasformazioni geometriche, tra cui *Random Choice Resize* (ridimensionamenti casuali), *Random Crop* (ritagli casuali) e *Random Flip* (rotazioni casuali), oltre all'applicazione di filtri che alterano i livelli di colore, luminosità e contrasto delle immagini. Questi filtri aggiuntivi hanno aumentato la robustezza del modello, permettendogli di adattarsi meglio a condizioni di illuminazione variabili o disturbi visivi che potrebbero essere presenti in scenari reali. In questo modo, il modello è stato esposto a una maggiore varietà di scenari, il che ha migliorato la sua capacità di generalizzare su dati non visti, soprattutto per le classi che inizialmente erano sotto-rappresentate.

3.3. Modelli di object detection

In questa sezione verranno analizzati alcuni modelli di *object detection*, selezionati per il presente lavoro di ricerca, che hanno introdotto di fatto delle innovazioni significative negli ultimi anni, contribuendo all'evoluzione delle tecniche di rilevamento. I modelli scelti, ossia TOOD[24], ATSS[25], DETR[26] e Deformable DETR[27] saranno analizzati per le loro peculiarità architetturali e metodologiche. Questi modelli, pur non costituendo gli ultimi sviluppi nel campo, hanno avuto un impatto rilevante. Ciò grazie a soluzioni innovative per sfide quali la selezione adattiva dei campioni, l'uso di architetture *Transformer* e l'efficienza nell'addestramento.

Tutti i modelli utilizzati in questo lavoro sono stati impiegati in versione pre-addestrata, sfruttando i pesi forniti dalle *repository* ufficiali del *framework* `MMDetection`[4]. `MMDetection` è un *toolbox open source* per l'*object detection*, basato su `PyTorch`[5] e parte del progetto `OpenMMLab`[6]. Questo *framework* si distingue per alcune caratteristiche chiave, tra cui la sua architettura modulare, che consente di costruire e personalizzare facilmente modelli combinando diversi moduli, semplicemente modificando i file di configurazione. Un altro aspetto rilevante è l'ottimizzazione per GPU, che assicura prestazioni superiori rispetto a molte altre soluzioni, riducendo notevolmente i tempi di sviluppo. Inoltre, un altro aspetto non da sottovalutare, è che tale *tool* viene costantemente aggiornato e supportato da una vasta community, offrendo soluzioni all'avanguardia e mantenendosi sempre al passo con i più recenti sviluppi nel campo. Di fatto, questa scelta ha permesso di ottimizzare i tempi di sperimentazione, consentendo di concentrarsi sull'analisi dei risultati piuttosto che sulla fase di addestramento.

Le analisi di seguito riportate si focalizzano sui contributi chiave di ciascun modello, evidenziando le differenze rispetto ai metodi precedenti e mostrando come abbiano migliorato le prestazioni nel rilevamento degli oggetti, aprendo la strada a ulteriori progressi nell'ambito di *detection* e nel settore di applicazione.

3.3.1. Task-aligned One-stage Object Detection (TOOD)

Il modello TOOD (Task-aligned One-stage Object Detection) rappresenta un notevole passo avanti rispetto ai tradizionali modelli di *detection* a una fase, affrontando in maniera innovativa il problema del disallineamento tra classificazione e localizzazione degli oggetti. Nei modelli precedenti, questi due compiti venivano tipicamente gestiti separatamente attraverso rami paralleli, il che spesso causava discrepanze nelle previsioni, riducendo l'accuratezza complessiva. In Figura 3.5 viene illustrato il funzionamento del modello attraverso i componenti che lo compongono.

Come si può notare, TOOD risolve questa problematica attraverso l'introduzione della Task-aligned head (T-Head), una struttura che migliora l'interazione tra i due *task* principali di classificazione e localizzazione. In questa architettura, le *feature* di classificazione e localizzazione vengono apprese congiuntamente grazie a uno *stack* di *layer* convoluzionali, garantendo una comprensione simultanea e più coerente di entrambi i compiti. Questo approccio consente una maggiore sinergia tra i due *task*, migliorando l'efficacia complessiva del modello. Un elemento chiave di TOOD è la Task Alignment Learning (TAL), una tecnica che permette di allineare esplicitamente la selezione delle ancore ottimali per la classificazione e la localizzazione. TAL introduce una metrica di allineamento innovativa, che misura la coerenza tra questi due compiti, identificando le ancore più informative e di alta qualità.

Questa metrica dipende dalla seguente formula:

[4] `https://github.com/open-mmlab/mmdetection`
[5] `https://pytorch.org/`
[6] `https://github.com/open-mmlab`

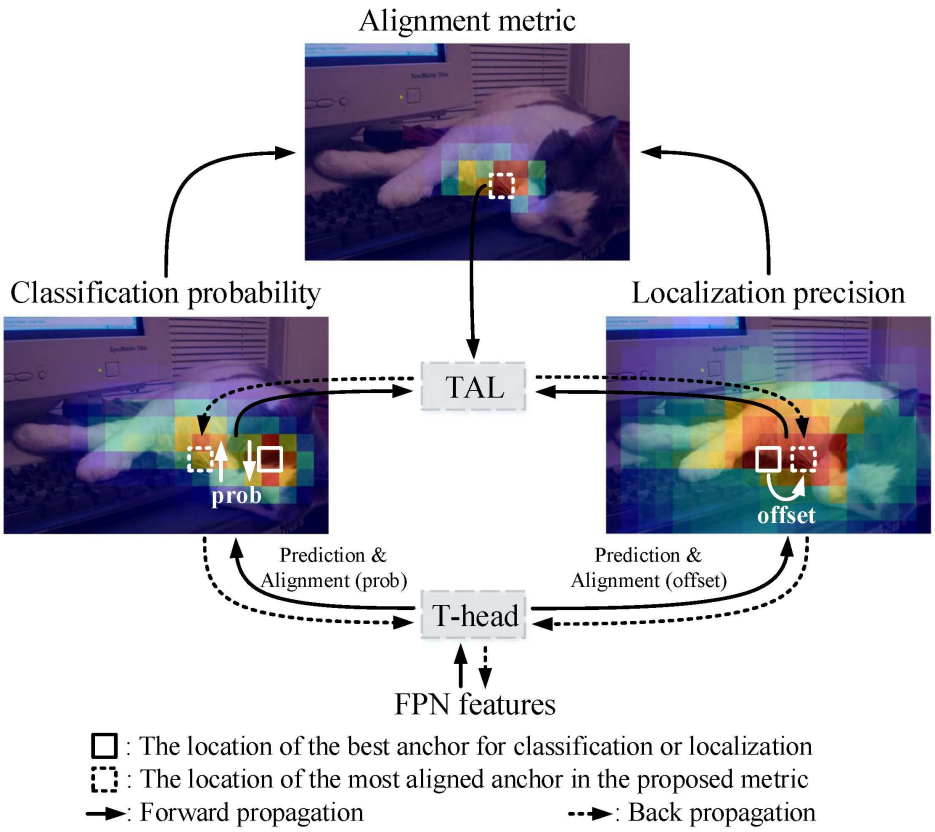

Figura 3.5.: Funzionamento del modello TOOD

Immagine da [24]

$$t = s^{\alpha} \times u^{\beta}$$

Tale formula combina quindi il punteggio di classificazione s e il valore di IoU u in un singolo parametro di allineamento t, che riflette la qualità combinata dell'ancora per entrambi i *task*. Regolando i parametri α e β, è possibile bilanciare l'influenza di ciascun compito nel calcolo del *task alignment*, consentendo al modello di dare priorità alle ancore che forniscono previsioni coerenti sia per la classificazione che per la localizzazione. Il parametro t svolge un compito essenziale nell'ottimizzazione congiunta dei due compiti, spingendo la rete a concentrarsi su ancore di alta qualità, ossia quelle che sono ben allineate dal punto di vista di entrambi i *task*. In altre parole, questa metrica aiuta il modello a concentrarsi sulle ancore che sono sia ben classificate che ben localizzate, migliorando la precisione delle previsioni globali. Il processo di assegnazione dinamica dei campioni durante il training è strettamente legato a questa metrica. TOOD adatta l'assegnazione dei campioni in modo dinamico, selezionando le ancore in base al loro valore di *task alignment* t, permettendo così di assegnare in modo più accurato i campioni positivi e negativi. Questo assicura che il modello si focalizzi sulle previsioni più promettenti durante l'addestramento.

A supportare questo processo di duplice ottimizzazione dei *task* vi è la *Task-aligned loss*. Durante il training, per le ancore positive, il valore di t (che riflette la qualità combinata di classificazione e localizzazione) sostituisce l'etichetta binaria tradizionale (0 o 1), consentendo al modello di concentrarsi sulle ancore meglio allineate. Tuttavia, per garantire un apprendimento efficace anche per le ancore difficili, t viene normalizzato in $\hat{t}$, preservandone il livello di accuratezza. Nella classificazione, il modello utilizza una *Focal Loss* per bilanciare il numero di ancore positive e negative, dando maggiore peso a quelle difficili da classificare. Per la localizzazione, il valore di t pesa il contributo di ciascuna ancora durante la regressione del *bounding box*, concentrandosi su quelle di alta qualità. Infine, la *Task-aligned loss* combina la *loss* di classificazione e quella di regressione, entrambe pesate da $\hat{t}$, garantendo un'ottimizzazione efficace e bilanciata tra i due *task*.

3.3.2. Adaptive Training Sample Selection (ATSS)

La rete ATSS (*Adaptive Training Sample Selection*) è stata sviluppata per risolvere il problema della selezione dei campioni positivi e negativi durante l'addestramento dei modelli di rilevamento di oggetti, colmando il divario tra i modelli basati su ancore (*anchor-based*) e quelli senza ancore (*anchor-free*). Tradizionalmente, i modelli di rilevamento basati su ancore, sebbene efficaci, presentano delle limitazioni legate al gran numero di ancore che devono essere utilizzate e alla scelta di soglie rigide per definire quali di queste sono considerate "positive" o "negative" durante l'addestramento. In un modello di questo genere, la selezione di campioni positivi e negativi avviene solitamente utilizzando la metrica IoU, trattata in precedenza, andando a misurare sovrapposizione tra l'*anchor box* e la *bounding box* reale. Le ancore con un IoU superiore a una soglia predefinita vengono etichettate come campioni positivi, mentre quelle con un IoU inferiore a un'altra soglia vengono considerate campioni negativi. Questo metodo, tuttavia, può non essere ottimale, poiché non tiene conto delle differenze tra oggetti di dimensioni o forme diverse, e impone iperparametri fissi che possono limitare le prestazioni del modello. Il nome di tale modello definisce un approccio più adattivo per la selezione di questi campioni. Invece di utilizzare soglie predefinite, ATSS adotta un metodo basato su statistiche per definire dinamicamente i campioni positivi e negativi, migliorando la qualità dell'addestramento e le prestazioni del modello stesso.

Per ogni oggetto presente nell'immagine, ATSS seleziona un certo numero di ancore basate sulla loro vicinanza al centro dell'oggetto e calcola la media e la deviazione standard degli IoU tra queste ancore e la *bounding box* reale. La soglia utilizzata per selezionare i campioni positivi viene quindi determinata dinamicamente aggiungendo la deviazione standard alla media. Questo meccanismo permette al modello di adattarsi meglio alle caratteristiche specifiche dell'oggetto in questione, riducendo la dipendenza da soglie fisse e rendendo il modello più robusto a diverse condizioni e variazioni. Un altro contributo importante di ATSS è la sua dimostrazione che l'uso

di molteplici ancore per ogni posizione nell'immagine potrebbe non essere necessario. Nei modelli tradizionali, come RetinaNet, si tende a "tassellare" un gran numero di ancore di diverse dimensioni e proporzioni su ogni punto della *feature map* per garantire che almeno una di queste si avvicini alla *bounding box* reale dell'oggetto. ATSS mostra che è possibile ottenere risultati altrettanto buoni o anche migliori utilizzando un numero significativamente inferiore di ancore, riducendo così il carico computazionale e rendendo il modello più efficiente.

L'algoritmo che sta alla base di questa idea, e che si è tradotta in un modello, è stato riportato in Figura 3.6.

Input:
$\mathcal{G}$ is a set of ground-truth boxes on the image
$\mathcal{L}$ is the number of feature pyramid levels
$\mathcal{A}_i$ is a set of anchor boxes from the i_{th} pyramid levels
$\mathcal{A}$ is a set of all anchor boxes
k is a quite robust hyperparameter with a default value of 9
Output:
$\mathcal{P}$ is a set of positive samples
$\mathcal{N}$ is a set of negative samples

for each ground-truth $g \in \mathcal{G}$ **do**
  build an empty set for candidate positive samples of the ground-truth g: $\mathcal{C}_g \leftarrow \varnothing$;
  for each level $i \in [1, \mathcal{L}]$ **do**
    $\mathcal{S}_i \leftarrow$ select k anchors from A_i whose center are closest to the center of ground-truth g based on L2 distance;
    $\mathcal{C}_g = \mathcal{C}_g \cup \mathcal{S}_i$;
  end for
  compute IoU between $\mathcal{C}_g$ and g: $\mathcal{D}_g = IoU(\mathcal{C}_g, g)$;
  compute mean of $\mathcal{D}_g$: $m_g = Mean(\mathcal{D}_g)$;
  compute standard deviation of $\mathcal{D}_g$: $v_g = Std(\mathcal{D}_g)$;
  compute IoU threshold for ground-truth g: $t_g = m_g + v_g$;
  for each candidate $c \in \mathcal{C}_g$ **do**
    if $IoU(c, g) \geq t_g$ and center of c in g **then**
      $\mathcal{P} = \mathcal{P} \cup c$;
    end if
  end for
end for
$\mathcal{N} = \mathcal{A} - \mathcal{P}$;
return $\mathcal{P}, \mathcal{N}$;

Figura 3.6.: Algoritmo ATSS

Immagine da [25]

3.3.3. Detection Transformer (DETR)

Il modello DETR (DEtection TRansformer) è una delle innovazioni più significative nel campo della *detection* di oggetti, grazie alla sua architettura basata sui *Transformer*. A differenza dei modelli precedenti, DETR affronta il problema del rilevamento in modo più semplice e diretto, trattandolo come un compito di predizione di insiemi

(*set prediction*). Con tale concetto si intende che il modello prevede un insieme di oggetti presenti in un'immagine senza considerare l'ordine o la sequenza degli oggetti stessi. Invece di fare affidamento su rettangoli predefiniti (ancore) o su processi di filtraggio per eliminare le previsioni ridondanti, DETR prevede direttamente gli oggetti e le loro posizioni come un gruppo. Questo approccio consente al modello di semplificare la pipeline di rilevamento, concentrandosi sulla previsione congiunta di tutti gli oggetti nell'immagine, senza passaggi intermedi tipici di altre architetture.

Questo approccio viene reso possibile grazie alla sua struttura *encoder-decoder Transformer*. L'elemento innovativo del modello, di fatto, è l'utilizzo del meccanismo di attenzione (*self-attention*), che permette al *Transformer* di modellare in modo esplicito tutte le interazioni tra gli oggetti e il contesto globale dell'immagine. Tutto questo non solo semplifica il processo di rilevamento, ma consente anche di ottenere previsioni più accurate grazie alla capacità del modello di ragionare su interazioni complesse tra gli oggetti in parallelo. In Figura 3.7 è stata riportata l'architettura del modello, dalla quale è possibile comprendere l'origine della predizione data una certa immagine in ingresso.

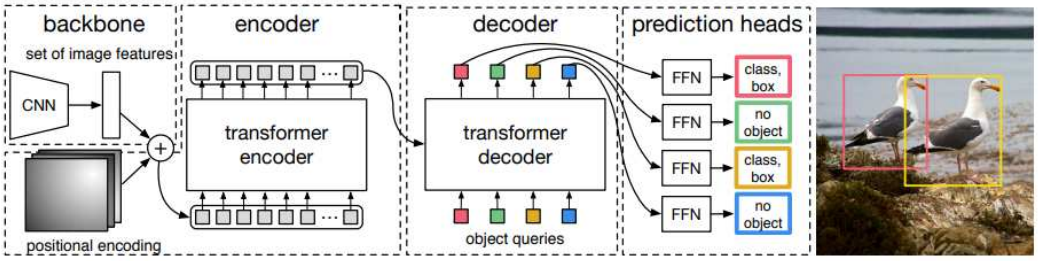

Figura 3.7.: Architettura DETR

Immagine da [26]

Ad alto livello, il processo di rilevamento in DETR è piuttosto semplice. L'immagine viene prima passata attraverso una CNN *backbone* (come ResNet-50) per estrarre una rappresentazione bidimensionale delle feature dell'immagine. Questa rappresentazione viene poi appiattita e arricchita con un *encoding* posizionale, che aggiunge informazioni sulla posizione spaziale delle *feature*. Grazie a questo passaggio, il modello riesce a comprendere la struttura spaziale dell'immagine, compensando il fatto che i *Transformer*, per loro natura, non possiedono un meccanismo intrinseco per catturare informazioni spaziali o sequenziali. Successivamente, questa rappresentazione viene passata attraverso l'*encoder* del *Transformer*, che apprende le relazioni globali tra le diverse regioni dell'immagine. L'*encoder*, grazie al meccanismo di *self-attention*, riesce a stabilire connessioni tra oggetti distanti tra loro, migliorando la capacità del modello di rilevare relazioni a lungo raggio all'interno dell'immagine. Il *decoder* del *Transformer*, invece, riceve in input un numero fisso di "*query*" apprese, conosciute come *object queries*. Queste *query* sono vettori con *embedding* posizionali appresi, ossia la rappresentazione posizionale appresa che aiuta il modello

a focalizzarsi sulle aree dell'immagine dove ci si aspetta che siano presenti gli oggetti. A questo punto, le predizioni vengono generate da una rete *feed-forward*composta da 3 *layer*, con funzione di attivazione ReLU. La FFN calcola le coordinate del centro, l'altezza e la larghezza della *bounding box* rispetto all'immagine di input, e predice la classe dell'oggetto utilizzando una funzione di classificazione. Poiché DETR prevede un numero fisso di box, che spesso è maggiore del numero reale di oggetti presenti, viene utilizzata una classe speciale, tra quelle già previste dal problema, per indicare che in quella posizione non è stato rilevato alcun oggetto. Questa classe funge da "*background*", un concetto simile a quello utilizzato in altri approcci di rilevamento degli oggetti. In questo modo, DETR rileva gli oggetti in maniera diretta ed efficiente, senza bisogno di passaggi o componenti aggiuntivi.

3.3.4. Deformable DETR

Deformable DETR è una versione estesa e migliorata di DETR (*DEtection TRansformer*), progettata per affrontare alcuni dei principali limiti dell'architettura originale. In particolare, *Deformable DETR* risolve due problemi critici del modello originale: la lenta convergenza e la difficoltà nel rilevamento di oggetti di piccole dimensioni. Uno dei contributi principali di Deformable DETR è l'introduzione del meccanismo di attenzione deformabile (*deformable attention*). Mentre l'attenzione tradizionale del *Transformer* considera tutte le posizioni spaziali dell'immagine per calcolare le interazioni tra gli oggetti, l'attenzione deformabile si concentra solo su un insieme ridotto di punti di campionamento attorno a un punto di riferimento. Questo riduce in modo significativo la complessità computazionale e permette al modello di convergere molto più velocemente. L'idea di base dell'attenzione deformabile è quella di applicare l'attenzione non sull'intera immagine, ma solo su un numero limitato di posizioni selezionate. Formalmente, l'attenzione deformabile per una query q in posizione p_q, applicata su una mappa di *feature* multi-scala F, è definita come:

$$\text{DeformAttn}(z_q, p_q, \mathbf{x}) = \sum_{m=1}^{M} \mathbf{w}_m \left[\sum_{k=1}^{K} A_{mqk} \cdot \mathbf{W}'_m \mathbf{x}(p_q + \Delta p_{mqk}) \right]$$

L'intuizione quindi è stata che, per ciascuna *query*, l'attenzione viene calcolata considerando un numero ridotto di punti di campionamento K, scelti attorno alla posizione di riferimento p_q, ciascuno con un offset Δp_{mqk}. I pesi A_{mqk} determinano quanto ciascun punto di campionamento contribuisce alla rappresentazione della *query*. In Figura 3.8 è possibile osservare il modulo appena trattato.

Questo approccio facilita di conseguenza anche la gestione delle *feature* multi-scala, senza la necessità di utilizzare strutture come le *Feature Pyramid Networks* (FPN)[28], permettendo di integrare le informazioni da più livelli di risoluzione dell'immagine. In altre parole, il modello è in grado di cogliere i dettagli sia a livello globale che locale, migliorando in modo significativo il rilevamento di oggetti di piccole dimensioni, uno

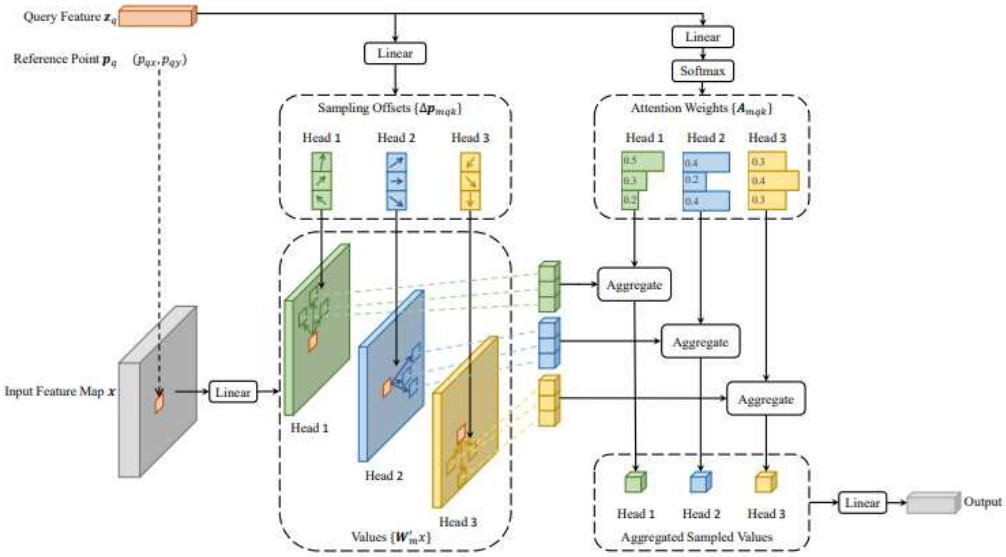

Figura 3.8.: Modulo *deformable attention*

Immagine da [27]

dei principali punti deboli dell'architettura DETR originale, come anticipato all'inizio di questo capitolo.

In Figura 3.9 invece è possibile notare l'architettura proposta del modello ad alto livello.

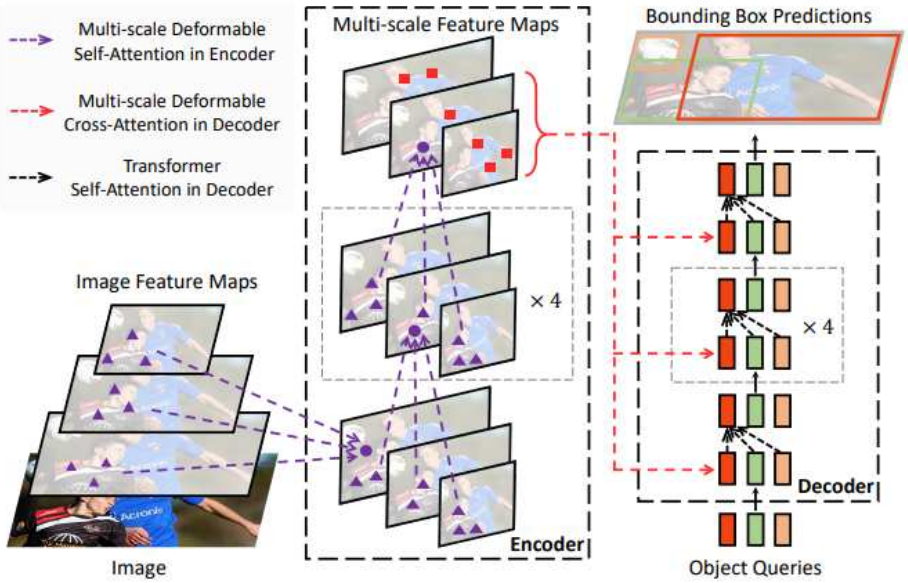

Figura 3.9.: Architettura Deformable DETR

Immagine da [27]

Un ulteriore miglioramento del modello è dato dalla rifinitura iterativa delle *bounding box*, che avviene durante il processo di *decoding*. A ogni iterazione del *decoder*, le previsioni delle *bounding box* vengono progressivamente affinate, rendendo il rilevamento più accurato. Ogni iterazione permette al modello di correggere gli errori precedenti e migliorare le coordinate delle *bounding box*, rendendo il processo di rilevamento più preciso ed efficiente.

3.4. Configurazione dei modelli e iperparametri di addestramento

Nell'implementazione dei modelli di *object detection* utilizzati in questo lavoro, si è partiti dalle configurazioni predefinite fornite dal *framework* `MMDetection`, che sono state opportunamente adattate per rispondere alle specifiche esigenze dei dati e agli obiettivi del *task*. Le configurazioni di base sono state selezionate per sfruttare al meglio i pesi pre-addestrati forniti ufficialmente, evitando così di ripetere la costosa fase di addestramento completa e permettendo di focalizzarsi maggiormente sulla fase di *fine-tuning* e ottimizzazione. Questo approccio ha consentito di ottimizzare sia i tempi di sviluppo che l'efficacia dei modelli, mantenendo comunque elevate prestazioni.

Come già sottolineato nei capitoli precedenti, i modelli scelti sono particolarmente complessi e dispendiosi dal punto di vista computazionale, richiedendo notevoli risorse hardware. La disponibilità limitata di una singola GPU RTX 2080 ha rappresentato un ulteriore vincolo che ha influenzato le scelte degli iperparametri di addestramento. Considerando la necessità di bilanciare le risorse disponibili con l'efficacia del modello, è stato essenziale adottare configurazioni che, pur non compromettendo le performance, risultassero gestibili nei tempi di addestramento. Gli iperparametri sono componenti chiave nell'addestramento dei modelli di machine learning e determinano in gran parte il comportamento del modello durante il processo di ottimizzazione. La loro corretta configurazione è essenziale per ottenere prestazioni ottimali e, soprattutto, per bilanciare l'efficienza computazionale con l'accuratezza dei risultati. La selezione di questi è stata quindi ottimizzata utilizzando un approccio progressivo ed in parte qualitativo, sperimentando inizialmente con configurazioni conservative per evitare problemi di *overfitting* o sottoutilizzo delle risorse, per poi passare a configurazioni più "aggressive" una volta verificate le capacità di convergenza dei modelli. Inoltre, vista la necessità di garantire una comparabilità diretta tra i diversi modelli, si è cercato di mantenere il più possibile delle configurazioni simili per quanto riguarda gli iperparametri chiave. Questo ha permesso di effettuare osservazioni accurate sulle differenze in termini di efficacia ed efficienza tra i vari approcci, limitando l'influenza di fattori esterni legati all'addestramento. Per esempio, il numero di iterazioni e il *learning rate* iniziale sono stati mantenuti su valori comuni

per tutti i modelli, adattandosi solo laddove fosse strettamente necessario per far fronte alle specifiche esigenze architetturali di un determinato modello.

Tra i principali iperparametri configurati vi sono il *batch size*, l'*optimizer*, il *learning rate* e la *loss function*, ciascuno dei quali gioca un ruolo fondamentale nel definire il comportamento e le prestazioni finali del modello.

- Il *batch size* rappresenta il numero di esempi che vengono elaborati dal modello prima che i pesi vengano aggiornati. Un *batch size* maggiore tende a stabilizzare gli aggiornamenti dei pesi, garantendo al modello stesso di non adattarsi troppo al singolo campione, ma mediare il contributo di più di questi per ottenere maggiora capacità di generalizzazione. Di fatto però, per un *batch size* superiore sono richieste anche maggiori risorse di memoria. Invece, un *batch size* più piccolo rende l'addestramento più rumoroso e specifico, ma consente di utilizzare meno memoria GPU. Nell'ambito di questo lavoro, il *batch size* è stato impostato a 4 per i modelli ATSS e TOOD, mentre per i modelli DETR e Deformable DETR è stato ridotto a 2. La scelta di utilizzare un *batch size* inferiore per i modelli basati su architetture *Transformer* (DETR e Deformable DETR) è stata dettata dalla loro maggiore complessità computazionale e dall'elevato consumo di memoria, che ha reso necessario ridurre il numero di campioni elaborati per volta. In generale si è comunque scelto tale parametro cercando di massimizzare l'uso della memoria della GPU senza incorrere in errori di esaurimento delle risorse.

- L'*optimizer* è l'algoritmo che regola l'aggiornamento dei pesi del modello, influenzando direttamente la velocità di apprendimento e la capacità del modello di convergere verso una soluzione ottimale. In questo lavoro, sono stati scelti due diversi ottimizzatori per gestire al meglio le specifiche dei modelli. Per i modelli ATSS e TOOD è stato utilizzato l'ottimizzatore SGD (Stochastic Gradient Descent). Questo algoritmo è noto per la sua semplicità ed efficacia, soprattutto quando si lavora con dataset di grandi dimensioni e modelli che richiedono stabilità e robustezza nell'aggiornamento dei pesi. L'SGD aggiorna i pesi basandosi su singoli campioni o piccoli batch di dati, riducendo l'*overhead* computazionale, e secondo la formula:

  $$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$$

  dove θ_t, rappresenta i pesi del modello al tempo t, η è il *learning rate* e $\nabla_\theta \mathcal{L}(\theta_t)$ è il gradiente della funzione di perdita $\mathcal{L}$, rispetto ai pesi θ_t. Questo approccio permette di aggiornare i pesi in modo iterativo, ma può soffrire di oscillazioni durante l'addestramento, specialmente in aree della funzione di perdita con pendenze variabili. Per affrontare queste oscillazioni, è stato introdotto un termine di *momentum*, con un valore di 0.9. Il *momentum* consente al modello di accumulare i gradienti passati per orientare gli aggiornamenti in una direzione

tale per cui vengano ridotte le oscillazioni e migliorata la velocità di convergenza. Il *momentum* viene applicato con le seguenti equazioni:

$$v_{t+1} = \mu v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

Qui, v_t rappresenta il termine di *momentum* accumulato fino all'iterazione t, mentre μ è il fattore di *momentum* (pari a 0.9 in questo caso). Grazie a questo approccio, il modello può mantenere una direzione costante durante l'aggiornamento dei pesi, riducendo la possibilità di rimanere intrappolato in minimi locali della funzione di perdita. Per i modelli DETR e Deformable DETR, è stato scelto l'ottimizzatore Adam (Adaptive Moment Estimation), che è particolarmente indicato per architetture complesse come quelle basate sui *Transformer*. Adam combina i vantaggi dell'SGD con il *momentum*, ma introduce anche un adattamento del *learning rate* per ogni parametro. Questo approccio permette ad Adam di aggiornare i pesi tenendo conto non solo del gradiente attuale, ma anche di una media esponenziale dei gradienti passati e delle loro varianze, attraverso le seguenti equazioni:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla_\theta \mathcal{L}(\theta_t)$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2$$

Dove, m_t rappresenta la stima esponenziale mobile del gradiente (primo momento), mentre v_t è la stima esponenziale mobile della varianza del gradiente (secondo momento). I parametri β_1 e β_2 controllano la velocità con cui queste medie esponenziali decadono, e in genere sono impostati rispettivamente a 0.9 e 0.999. Questo consente ad Adam di bilanciare il peso del gradiente corrente rispetto alla storia passata dei gradienti. L'aggiornamento finale dei pesi in Adam avviene come segue:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} m_{t+1}$$

In quest'ultima equazione, il termine ϵ è un piccolo valore costante (tipicamente pari a 10^-8) introdotto per evitare divisioni per zero. Il vantaggio principale di Adam è la sua capacità di adattare dinamicamente il *learning rate* per ogni parametro del modello, rendendolo particolarmente efficace quando i gradienti sono rumorosi o variano significativamente. Questo lo rende ideale per modelli

complessi come quelli *transformer-based*, poiché garantisce una convergenza più rapida e stabile, anche con *batch size* ridotti.

- Il *learning rate* rappresenta la velocità con cui il modello aggiorna i suoi pesi a ogni iterazione. In questo lavoro, è stato scelto un *learning rate* iniziale di 0.001 per tutti i modelli, valore considerato ottimale per avviare il processo di apprendimento senza rischiare instabilità dovute a un aggiornamento troppo rapido dei pesi. Per mantenere un processo di apprendimento stabile, è stato inoltre utilizzato un *scheduler MultiStepLR*, che ha ridotto il *learning rate* a intervalli regolari durante l'addestramento. Questo ha permesso di migliorare l'affinamento del modello verso la fine del training, riducendo la probabilità di sovra-adattamento e permettendo al modello di stabilizzarsi su soluzioni migliori.

- La funzione di perdita, o *loss function*, è un elemento essenziale nel processo di addestramento di un modello di machine learning, poiché misura la discrepanza tra le previsioni del modello e i risultati attesi. Ridurre il valore della *loss function* è l'obiettivo primario durante l'addestramento, in quanto consente al modello di migliorare progressivamente la sua capacità di effettuare previsioni accurate. Nel contesto di questo lavoro, è stata utilizzata la *Focal Loss* per affrontare una delle principali sfide legate al dataset utilizzato: lo sbilanciamento tra classi. In questo caso, lo sbilanciamento deriva dal processo di raccolta dei dati, che non ha garantito una rappresentazione bilanciata delle varie classi, trattandosi di un dataset reale. In situazioni del genere, dove alcune classi sono meno rappresentate rispetto ad altre, la *Focal Loss* si dimostra particolarmente efficace. A differenza della più tradizionale *Cross-Entropy Loss*, la *Focal Loss* è progettata per gestire meglio gli scenari in cui ci sono classi rappresentate in modo non uniforme. Questo viene fatto riducendo l'importanza degli esempi ben classificati e aumentando il peso degli esempi più difficili da classificare, come quelli appartenenti alle classi meno rappresentate. In questo modo, il modello viene incentivato a prestare maggiore attenzione alle classi meno frequenti, migliorando la capacità di rilevamento anche in condizioni di squilibrio. Matematicamente, la *Focal Loss* aggiunge un termine modulatorio alla *Cross-Entropy Loss*, che riduce il contributo degli esempi ben classificati e aumenta quello degli esempi più complessi, garantendo un apprendimento più focalizzato su ciò che il modello fatica a imparare. La formula di base è la seguente:

$$\mathrm{FL}(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

Il parametro γ è centrale in questa equazione, in quanto, quando un esempio è classificato correttamente con un'alta probabilità, il termine $(1 - p_t)^{\gamma}$ tende a zero, riducendo il contributo di questo esempio nella perdita complessiva.

Al contrario, per gli esempi difficili o mal classificati, questo termine rimane elevato, aumentando il loro impatto sull'aggiornamento dei pesi del modello.

3.5. Introduzione all'architettura del servizio

In questa sezione viene fornita una descrizione approfondita dei servizi impiegati per lo sviluppo dell'applicativo progettato per permettere l'interazione tra gli utenti e i modelli di machine learning addestrati nel corso del progetto. L'architettura di riferimento è stata realizzata, come già ampiamente discusso, con l'obiettivo di garantire prestazioni elevate, scalabilità e sicurezza, sfruttando tecnologie moderne e integrate per il *deployment* e la gestione dei modelli addestrati. Dopo un'attenta valutazione delle diverse opzioni disponibili sul mercato, è stato scelto di adottare i servizi cloud di Amazon. Questa decisione si basa su diversi fattori chiave. In primo luogo, i servizi AWS sono risultati altamente documentati, offrendo risorse tecniche dettagliate e approfondite, che hanno facilitato l'implementazione richiesta, riducendo i tempi di configurazione e semplificando la gestione dell'infrastruttura. Inoltre, l'utilizzo del piano gratuito messo a disposizione da Amazon è risultato ampiamente sufficiente per le esigenze iniziali dell'applicativo sviluppato, permettendo di avviare e testare l'intero servizio senza costi aggiuntivi, e mantenendo comunque tutte le funzionalità essenziali richieste. Nel dettaglio, i capitoli successivi offriranno una panoramica dei vari componenti utilizzati per orchestrare l'intero flusso del servizio, a partire dalla gestione delle immagini "containerizzate" fino all'esecuzione delle richieste degli utenti tramite funzioni *serverless*. L'immagine in Figura 3.10 fornisce una panoramica ad alto livello dell'architettura del servizio, illustrando i principali componenti coinvolti e le loro interazioni.

3.5.1. Amazon ECR e Docker Image

Nel contesto dello sviluppo di applicazioni moderne, il paradigma dei container è centrale, poiché permette di impacchettare il codice e le sue dipendenze in un'unità eseguibile e portatile. `Amazon ECR` (Elastic Container Registry)[7] è un servizio completamente gestito da AWS che consente agli sviluppatori di archiviare, gestire e distribuire immagini `Docker`[8] in modo sicuro e scalabile. `Amazon ECR` si integra perfettamente con altri servizi AWS per facilitare il *deployment* delle applicazioni containerizzate.

Un elemento chiave in questa architettura è l'immagine `Docker`, che merita un piccolo approfondimento. Un'immagine `Docker` è un pacchetto leggibile da macchina che contiene tutto il necessario per eseguire un'applicazione, inclusi codice sorgente, librerie di sistema, dipendenze, file di configurazione e variabili d'ambiente. Viene creata a partire da un file chiamato *Dockerfile*, che specifica una serie di istruzioni

[7] `https://aws.amazon.com/it/ecr/`
[8] `https://www.docker.com/`

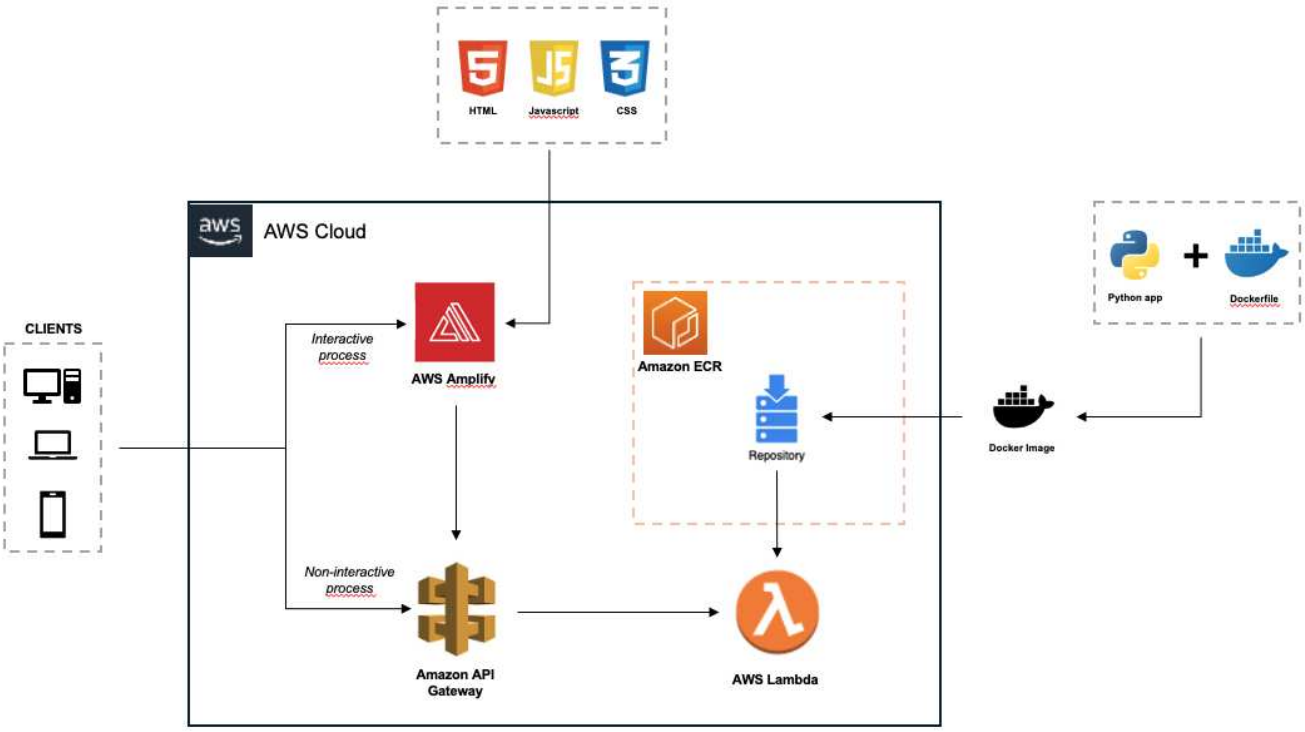

Figura 3.10.: Overview architettura

su come costruire l'ambiente di esecuzione dell'applicazione. Ad esempio, un *Dockerfile* può definire quale sistema operativo di base usare (come `Ubuntu`[9] o `Alpine Linux`[10]), quali dipendenze installare, e quali comandi eseguire quando viene avviato il container. Un aspetto distintivo delle immagini `Docker` è la loro portabilità e immutabilità. Una volta creata, un'immagine può essere eseguita in qualsiasi ambiente che supporti `Docker`, garantendo un comportamento coerente dell'applicazione indipendentemente dalla piattaforma. Questo risolve molti dei problemi legati alle differenze di configurazione tra ambienti di sviluppo, test e produzione, riducendo il rischio di bug e comportamenti imprevisti.

Dopo la creazione di un'immagine, questa può essere caricata su `Amazon ECR`, che funge da *repository* per conservare e distribuire le immagini `Docker`, offrendo anche funzionalità avanzate in termini di sicurezza, come la crittografia a riposo e in transito. La crittografia a riposo protegge i dati mentre sono memorizzati, assicurando che le immagini `Docker` siano cifrate e quindi illeggibili senza le opportune chiavi. La crittografia in transito, invece, protegge i dati mentre vengono trasferiti, ad esempio durante il caricamento o il download delle immagini, utilizzando protocolli sicuri come TLS (Transport Layer Security) per prevenire intercettazioni o attacchi. `Amazon`

[9] `https://ubuntu.com/`
[10] `https://www.alpinelinux.org/`

`ECR` è integrato con `AWS Identity and Access Management` (IAM)[11], consentendo una gestione granulare dell'accesso alle immagini. Ciò permette di controllare chi ha il permesso di visualizzare, caricare o scaricare le immagini, garantendo un elevato livello di sicurezza e conformità. Un ulteriore punto di forza di `Amazon ECR` è la sua integrazione con i flussi di lavoro DevOps, facilitando la gestione delle immagini `Docker` attraverso pipeline di CI/CD (Continuous Integration/Continuous Deployment). Grazie all'automazione fornita da queste pipeline, gli sviluppatori possono automatizzare l'intero ciclo di vita dell'immagine, dalla creazione all'esecuzione di test e verifiche di sicurezza, fino al deployment finale nel *repository*. Ogni modifica del codice può innescare la generazione automatica di una nuova immagine `Docker`, semplificando e accelerando il processo di rilascio. Una volta caricate su `ECR`, le immagini `Docker` possono essere facilmente organizzate e gestite, con supporto per il versionamento, che consente di tenere traccia delle diverse release e di ripristinare versioni precedenti in caso di necessità. Infine, non è possibile ignorare le prestazioni di `Amazon ECR`, che è progettato per scalare automaticamente in base alle esigenze dell'applicazione. Questo permette di gestire grandi volumi di immagini senza preoccuparsi delle risorse infrastrutturali sottostanti, allineandosi perfettamente ai concetti di cloud computing precedentemente discussi.

In Figura 3.11 è possibile osservare un breve riassunto ad alto livello di come avviene l'interazione tra gli strumenti appena discussi nell'ambito di questo progetto.

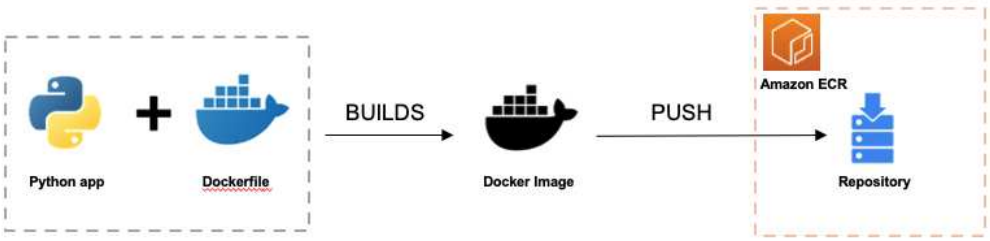

Figura 3.11.: Overview di `Amazon ECR` e `Docker`

3.5.2. AWS Lambda e Amazon API Gateway

`AWS Lambda`[12] è un servizio di calcolo *serverless* che permette di eseguire codice in risposta a eventi, senza doversi preoccupare della gestione e manutenzione del server. Uno dei principali vantaggi di `Lambda` è la sua scalabilità automatica: il codice viene eseguito solo quando viene attivato un evento, e AWS si occupa automaticamente di scalare le risorse in base al carico. Questo risulta particolarmente utile per

[11] `https://aws.amazon.com/it/iam/`
[12] `https://aws.amazon.com/it/lambda/`

applicazioni dove le richieste degli utenti possono variare notevolmente nel tempo e ove periodi di carico e scarico di richieste possono essere imprevedibili, poiché `Lambda` gestisce dinamicamente il numero di istanze necessarie per soddisfarle. Anche dal punto di vista del *pricing*, il modello di esecuzione *serverless* porta con sé diversi benefici che si traducono nella possibilità di pagare il servizio solo per il tempo effettivo di esecuzione del codice, il che lo rende estremamente conveniente rispetto ai modelli tradizionali, in cui si paga per server o istanze indipendentemente dal loro utilizzo effettivo. In Figura 3.12 una panoramica dell'interazione tra i due servizi trattati. Il funzionamento di `AWS Lambda` è semplice e altamente efficiente. Il codice

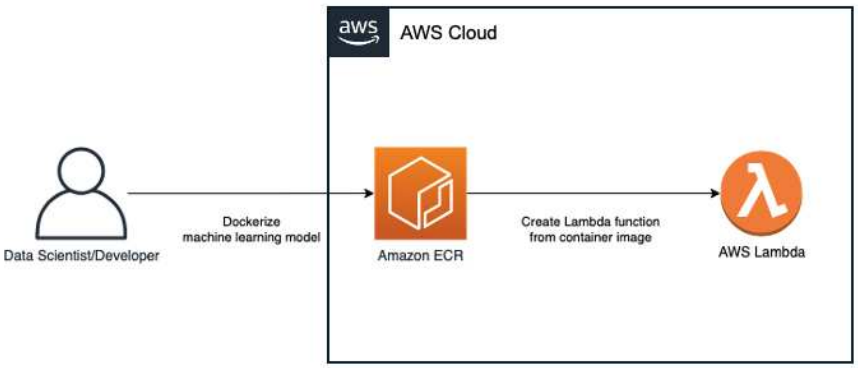

Figura 3.12.: Overview di `Amazon ECR` e `AWS Lambda`

viene caricato in `AWS Lambda` e può essere eseguito in risposta a una vasta gamma di eventi, incluse richieste HTTP, modifiche a file in `Amazon S3`, eventi di stream di dati in tempo reale e, come in questo caso, chiamate API tramite `Amazon API Gateway`[13]. `Amazon API Gateway` è un servizio completamente gestito che consente di creare, pubblicare, mantenere e proteggere API RESTful o WebSocket, rendendolo ideale per lo sviluppo di applicazioni basate su microservizi o architetture *serverless*, come quella in questione. In tale progetto, `API Gateway` funge da strato intermedio tra l'applicazione web e i modelli, permettendo agli utenti di inviare richieste (ad esempio, per eseguire inferenze con i modelli) e ottenere risposte nella maniera più rapida ed efficiente possibile.

Un servizio di gestione delle API come questo, porta con sé una serie di vantaggi dovuti a un intervento manuale minimo, tra cui la gestione automatica del *routing* delle richieste, l'autenticazione, il bilanciamento del carico, il *throttling* per prevenire l'abuso delle risorse, il *caching* delle risposte per migliorare le prestazioni, e la gestione delle versioni delle API per supportare lo sviluppo continuo. Utilizzato in sinergia con `AWS Lambda`, questo servizio permette il *triggering* delle funzioni sviluppate,

[13] `https://aws.amazon.com/it/api-gateway/`

eseguendo il codice necessario per elaborare le richieste. Questo approccio moderno allo sviluppo delle API consente di creare applicazioni completamente *serverless*, dove l'intera infrastruttura è gestita automaticamente da AWS, garantendo scalabilità e alta disponibilità.

La Figura 3.13 fornisce una panoramica dei servizi descritti all'interno della soluzione.

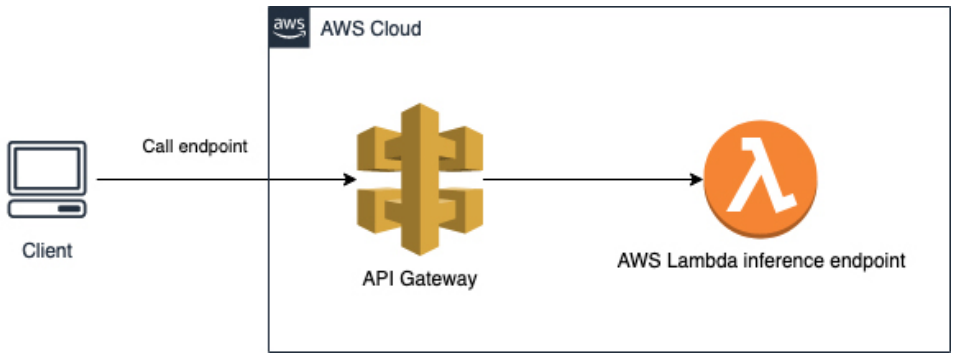

Figura 3.13.: Overview di `API Gateway` e `AWS Lambda`

3.5.3. AWS Amplify

L'ultimo servizio di Amazon scelto per questo sistema è `AWS Amplify`[14], un servizio che semplifica e accelera lo sviluppo di applicazioni web e mobile, offrendo un'ampia gamma di strumenti per la creazione, implementazione e gestione sia del *frontend* che del *backend*. `Amplify` si distingue per la capacità di rendere l'intero processo di sviluppo altamente efficiente, grazie alla sua integrazione nativa con altri servizi AWS. Questa caratteristica lo rende particolarmente adatto per applicazioni che necessitano di scalabilità, flessibilità e un'elevata interoperabilità tra i vari componenti.

Nel contesto di questo progetto, `AWS Amplify` è stato utilizzato per gestire l'intero *frontend* dell'applicazione web, fornendo una piattaforma ottimizzata per interfacciarsi agevolmente con i servizi *backend* preesistenti, come `AWS Lambda` e `Amazon API Gateway`. L'integrazione facilitata dall'ecosistema di `Amplify` ha semplificato la comunicazione tra *frontend* e *backend*, riducendo significativamente la complessità operativa spesso presente in applicazioni di questo tipo. Questo ha reso possibile un ciclo di sviluppo e aggiornamento rapido ed efficiente, migliorando al contempo l'esperienza complessiva di sviluppo e la gestione delle modifiche. In Figura 3.14 una panoramica delle interazioni del servizio di *frontend* con gli altri servizi impiegati.

[14] `https://aws.amazon.com/it/amplify/`

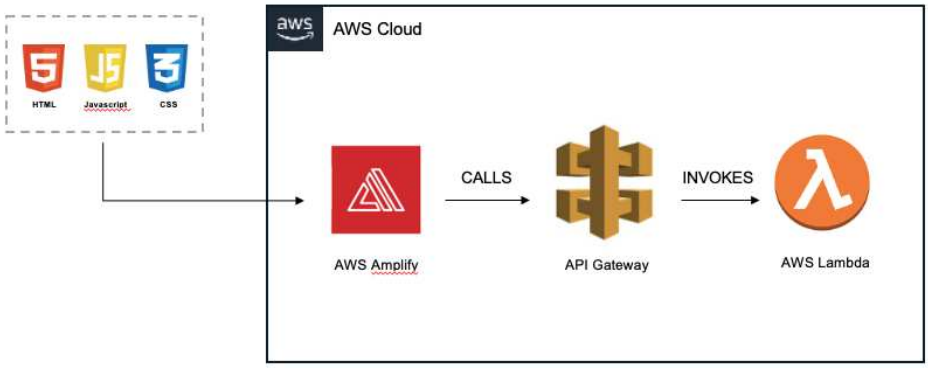

Figura 3.14.: Overview della soluzione con `AWS Amplify`

Capitolo 4.

Risultati e discussioni

4.1. Addestramento e validazione modelli

Durante l'addestramento dei modelli precedentemente descritti sul dataset di training, le due metriche monitorate sono la *loss_cls* e la *loss_bbox*. Queste metriche forniscono un feedback importante sull'andamento dell'addestramento, misurando rispettivamente la capacità del modello di classificare correttamente gli oggetti e di localizzarli con precisione. La *loss_cls* valuta la capacità del modello di identificare correttamente la classe degli oggetti. Un valore elevato indica difficoltà nella classificazione, mentre una sua riduzione riflette un miglioramento nella precisione del modello nel riconoscere le classi. D'altra parte, la *loss_bbox* misura l'accuratezza con cui il modello prevede le coordinate dei *bounding box*, che delimitano gli oggetti nelle immagini. Una riduzione di questa metrica segnala un miglioramento nella capacità del modello di localizzare correttamente gli oggetti. Di seguito, è stato riportato l'andamento delle metriche appena descritte nelle Figure 4.1, 4.2, 4.3 ed 4.4.

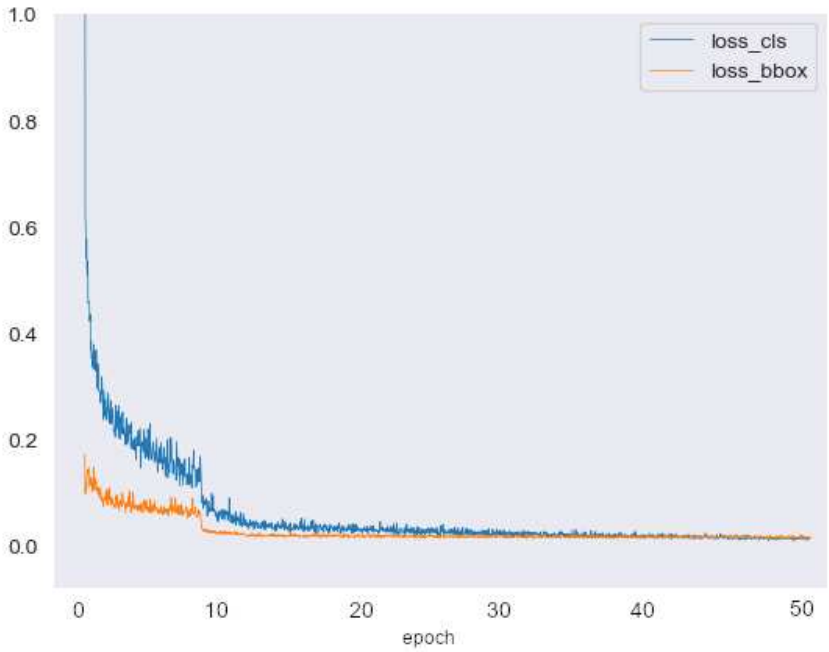

Figura 4.1.: Risultati addestramento ATSS

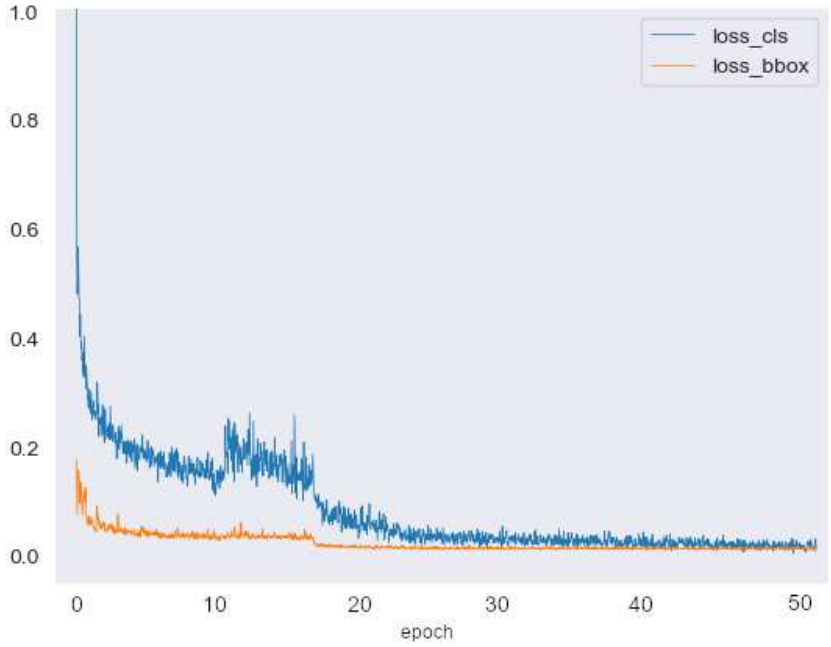

Figura 4.2.: Risultati addestramento TOOD

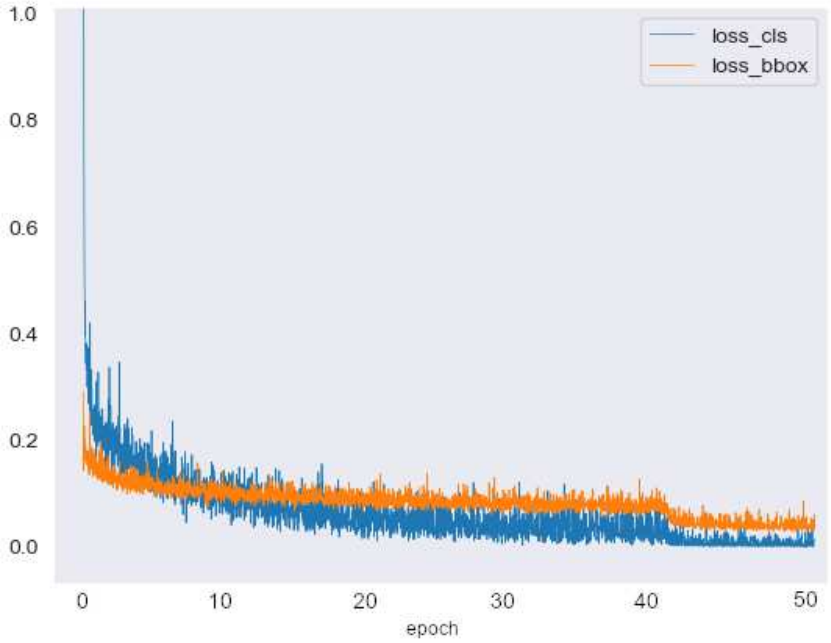

Figura 4.3.: Risultati addestramento DETR

Si può notare come l'andamento delle curve graficate, per tutti i modelli, mostri una riduzione significativa sia della *loss_cls* che della *loss_bbox* nelle prime epoche, indicando che le stesse stanno apprendendo efficacemente sia la classificazione che la localizzazione degli oggetti. Tuttavia, si riscontra un comportamento interessante con il passare delle epoche. Per i modelli ATSS e TOOD, si nota una discesa più

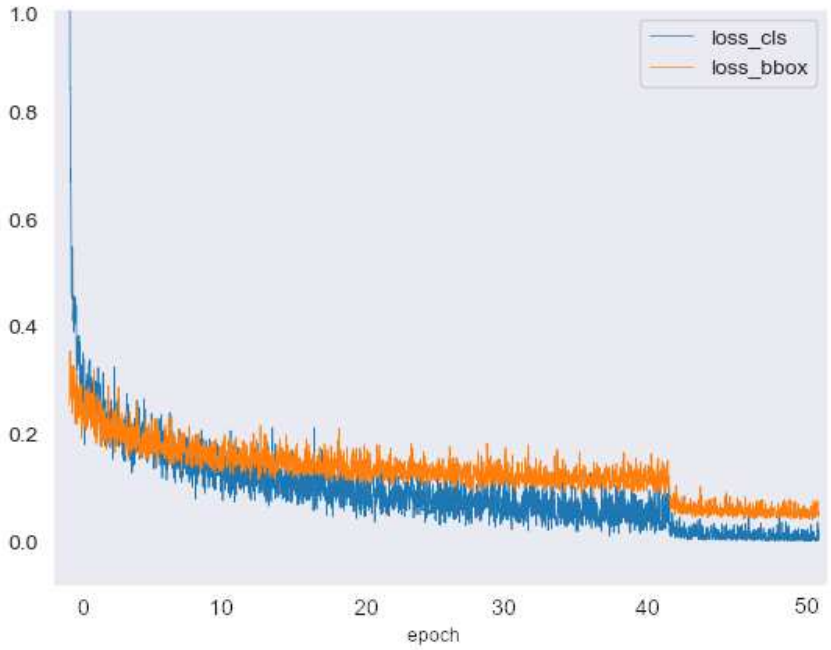

Figura 4.4.: Risultati addestramento Deformable DETR

rapida e stabile di entrambe le curve. La *loss_cls* diminuisce in modo progressivo e costante, così come la *loss_bbox*. Ciò indica che questi modelli riescono a convergere rapidamente e in modo stabile durante l'addestramento. Sebbene TOOD presenti alcune oscillazioni maggiori nella *loss_bbox* rispetto a ATSS, entrambi i modelli raggiungono valori di perdita molto bassi. Analizzando i modelli DETR e Deformable DETR, si nota una maggiore variabilità nelle curve, durante l'intera durata dell'addestramento. Tuttavia, le differenze tra i due modelli non sono così marcate come ci si potrebbe aspettare. DETR mostra un miglioramento graduale e, contrariamente alle aspettative, riesce a stabilizzarsi e a mantenere buone performance, se non superiori, rispetto a Deformable DETR nelle fasi finali dell'addestramento. Deformable DETR, invece, pur mostrando una buona capacità di ridurre entrambe le perdite, presenta fluttuazioni più pronunciate nella capacità di individuare la borsa. In generale, la maggiore variabilità osservata nei modelli *Transformer* potrebbe essere dovuta non solo alla loro complessità, ma anche al *batch size* ridotto, che può limitare la capacità di questi modelli di mantenere una rappresentazione coerente dei dati in ogni iterazione. Tuttavia, come mostrato nei grafici, DETR riesce a migliorare in modo costante e, alla lunga, sembra "performare" meglio di Deformable DETR in termini di precisione complessiva. Questo risultato potrebbe essere dovuto al fatto che i vantaggi del modello Deformable DETR non risultano così rilevanti in questo contesto, poiché le immagini analizzate presentano raramente borse di piccole dimensioni o parzialmente occluse.

Durante la fase di validazione, viene valutata la capacità dei modelli di generalizzare su un set di immagini mai viste prima dalle reti. Questo permette di calcolare le

prestazioni dei modelli man mano che i pesi vengono aggiornati per raggiungere una soluzione ottimale. Per misurare le prestazioni, si utilizzano le `CocoMetrics`[1], una serie di metriche ispirate al dataset COCO, ampiamente adottato nel campo della *computer vision*. Queste metriche comprendono, tra le altre, la mAP, che viene calcolata di fatto ad ogni epoca.

Come mostrato in Figura 4.5, si possono osservare differenze nelle prestazioni dei quattro modelli analizzati.

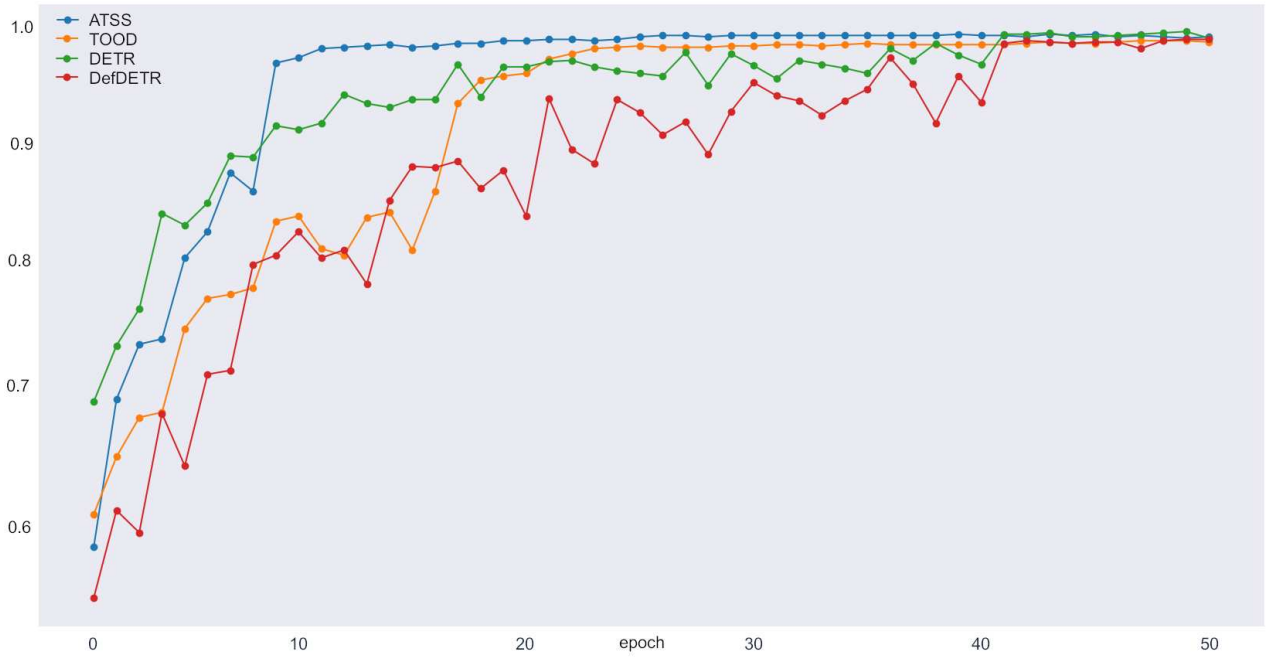

Figura 4.5.: Risultati di validazione dei modelli

Prima di passare all'analisi dettagliata delle prestazioni dei singoli modelli, è importante sottolineare che l'approccio dell'Early Stopping non è stato ritenuto utile in questo contesto. L'Early Stopping è una tecnica che interrompe l'addestramento quando il miglioramento sulle prestazioni, misurate sul set di validazione, si arresta per un numero prefissato di epoche consecutive. Tuttavia, dopo opportune valutazioni, si è scelto di non applicarlo in quanto i modelli, sebbene mostrassero oscillazioni nelle prestazioni, continuavano a migliorare anche nelle epoche successive. L'applicazione dell'Early Stopping avrebbe potuto impedire ai modelli di raggiungere il loro massimo potenziale, interrompendo il processo di apprendimento prematuramente. Si è quindi deciso di proseguire l'addestramento per il numero completo di epoche prefissate. Osservando le curve dei vari modelli:

- ATSS, rappresentato dalla curva blu, si distingue per un rapido incremento della mAP fin dalle prime epoche, con una convergenza quasi costante a partire

[1] `https://huggingface.co/spaces/rafaelpadilla/coco_metrics/blob/a08c52915eea92ac57a4b42b92a900b44611089e/README.md`

dall'epoca 20, raggiungendo valori molto elevati. Le oscillazioni sono minime, il che indica un'ottima stabilità anche durante la fase di validazione. Questo suggerisce che ATSS ha imparato a generalizzare bene sui dati non visti.

- TOOD, rappresentato dalla curva arancione, mostra un andamento simile a quello di ATSS nelle prime epoche, ma presenta maggiori fluttuazioni. Sebbene la mAP aumenti costantemente, l'andamento risulta meno regolare, indicando una variabilità nei risultati tra le epoche. Tuttavia, verso la fine del processo di addestramento, TOOD converge su valori di mAP comparabili a quelli di ATSS.

- DETR, rappresentato dalla curva verde, ha un andamento differente rispetto ai modelli precedenti. La crescita della mAP è più lenta nelle prime epoche, con difficoltà a raggiungere valori elevati in modo costante. Le fluttuazioni sono più marcate e persistono anche nelle fasi avanzate dell'addestramento, suggerendo che DETR potrebbe necessitare di più tempo per stabilizzarsi.

- Deformable DETR, rappresentato dalla curva rossa, mostra la maggiore instabilità e variabilità tra tutti i modelli fin dalle prime epoche. Nonostante ciò, si nota un miglioramento a lungo termine, anche se il comportamento iniziale è meno stabile rispetto agli altri modelli.

Un aspetto rilevante da tenere in considerazione, sia durante la fase di addestramento che in quella di validazione, è il cambiamento improvviso della pendenza delle curve di apprendimento in corrispondenza di determinate epoche. Questo fenomeno potrebbe essere attribuibile al settaggio delle *milestones* del MultiStepLR, utilizzato per ciascuno dei modelli. Le *milestones* rappresentano specifici punti temporali, misurati in epoche, nei quali il tasso di apprendimento (*learning rate*) viene ridotto in modo programmato, solitamente moltiplicandolo per un fattore prestabilito, in questi casi rappresentato dal parametro gamma, impostato pari a 0.1. Questo meccanismo consente al modello di migliorare la propria ottimizzazione nelle fasi avanzate dell'addestramento, rendendolo più capace di generalizzare sui dati e riducendo il rischio di sovra-adattamento ai dati di training. In questo caso, seguendo le indicazioni degli autori delle rispettive reti, per ATSS le *milestones* sono state fissate alle epoche 8 e 11, per TOOD alle epoche 16 e 22, mentre per i modelli basati su *Transformer* è stato adottato un approccio più semplice, riducendo il *learning rate* all'epoca 40. In corrispondenza di queste epoche si può osservare come le reti tendano ad accelerare il loro avvicinamento al minimo della funzione di perdita, idealmente il minimo globale, migliorando l'efficacia dell'ottimizzazione.

4.2. Valutazione delle prestazioni dei modelli sui dati di test

Passando alla fase di test, i modelli addestrati sono stati valutati su un test set composto da dati mai utilizzati durante l'addestramento e validazione, per misurare

la loro effettiva capacità di generalizzazione. La performance dei modelli è stata misurata tramite la mAP e le matrici di confusione, strumenti molto spesso impiegati in task di questo genere per valutare la qualità delle predizioni ottenute. La mAP è stata calcolata in concomitanza con l'IoU, ossia, per il calcolo si è utilizzato un intervallo di IoU per valutare le prestazioni su diverse soglie: IoU 0.50, IoU 0.75 e un range compreso tra 0.50 e 0.95 con incrementi di 0.05 (*mAP@[0.50:0.05:0.95]*). L'IoU 0.50 rappresenta una soglia meno severa, mentre l'IoU 0.75 è più rigoroso e consente di valutare con maggiore precisione quanto accuratamente il modello è in grado di identificare e localizzare gli oggetti, in questo caso le borsette. Il calcolo della mAP su più soglie permette di ottenere un quadro completo delle prestazioni, poiché fornisce una panoramica dell'accuratezza su diverse sensibilità di sovrapposizione (o *overlapping*). Le matrici di confusione sono state utilizzate per analizzare nel dettaglio il comportamento dei modelli nella classificazione. Queste matrici suddividono i risultati in veri positivi, falsi positivi, falsi negativi e veri negativi, e consentono di osservare come ciascuna classe viene trattata. Esse permettono di comprendere con precisione se e come i modelli stanno facendo confusione tra le classi e forniscono indicazioni utili su come migliorare la capacità di distinguere correttamente tra le diverse categorie di oggetti.

I risultati ottenuti, riassunti nella seguente tabella e accompagnati dalle matrici di confusione per ciascun modello (Figure 4.6, 4.7, 4.8 e 4.9), mostrano come le performance generali sembrino eccellenti. In particolare, si osserva una media di circa 0.95 di mAP su tutti i range di IoU calcolati. Tuttavia, tale performance, pur essendo positiva, potrebbe indicare *overfitting*, segnalando la possibilità che i modelli abbiano imparato troppo bene i pattern del dataset di addestramento, con il rischio di non generalizzare altrettanto bene su dati mai visti.

Model	**mAP**	**mAP$_{50}$**	**mAP$_{75}$**
ATSS	0,967	0,970	0,970
TOOD	0,952	0,957	0,957
DETR	**0,980**	**0,981**	**0,981**
DefDETR	0,968	0,970	0,970

Tra i modelli testati, le reti TOOD e ATSS hanno mostrato performance inferiori rispetto agli altri. In particolare, la rete TOOD, sebbene abbia ottenuto buoni risultati su alcune classi, come le *handbags* (87%) e le *tote bags* (79%), ha registrato difficoltà nel riconoscimento delle *crossbody bags*, con una precisione del 57%. Questi risultati suggeriscono che TOOD ha avuto difficoltà a distinguere tra classi visivamente simili, con un numero significativo di errori di classificazione in questa condizione. La rete ATSS, sebbene migliore rispetto a TOOD, ha comunque mostrato prestazioni inferiori rispetto alle aspettative, soprattutto se confrontate con le fasi precedenti di addestramento e validazione. Ad eccezione delle classi per le quali non sono stati commessi errori e della classe *shoulder bag*, di cui verrà discusso successivamente, per le

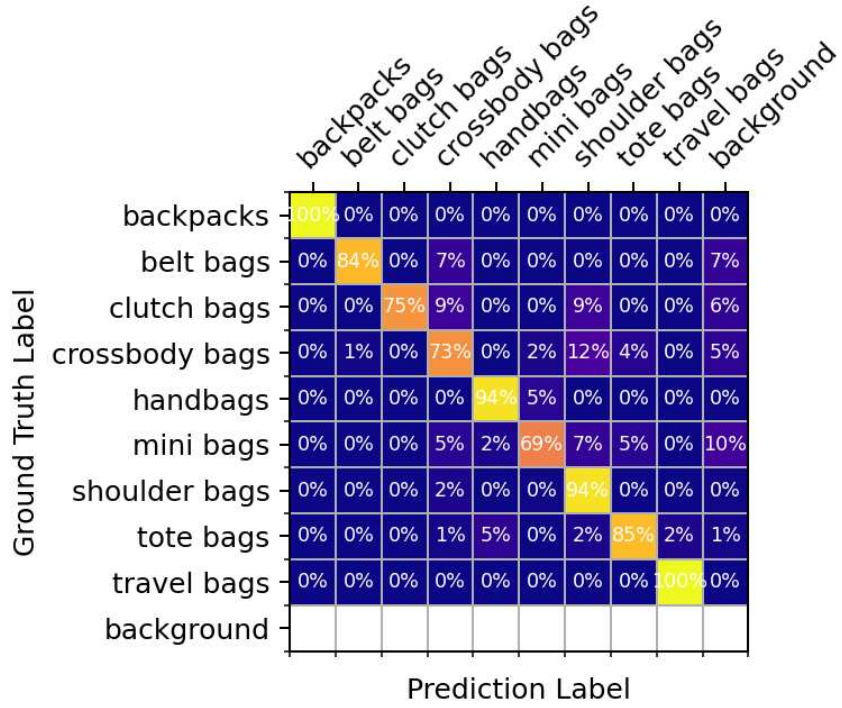

Figura 4.6.: Matrice di confusione relativa ad ATSS

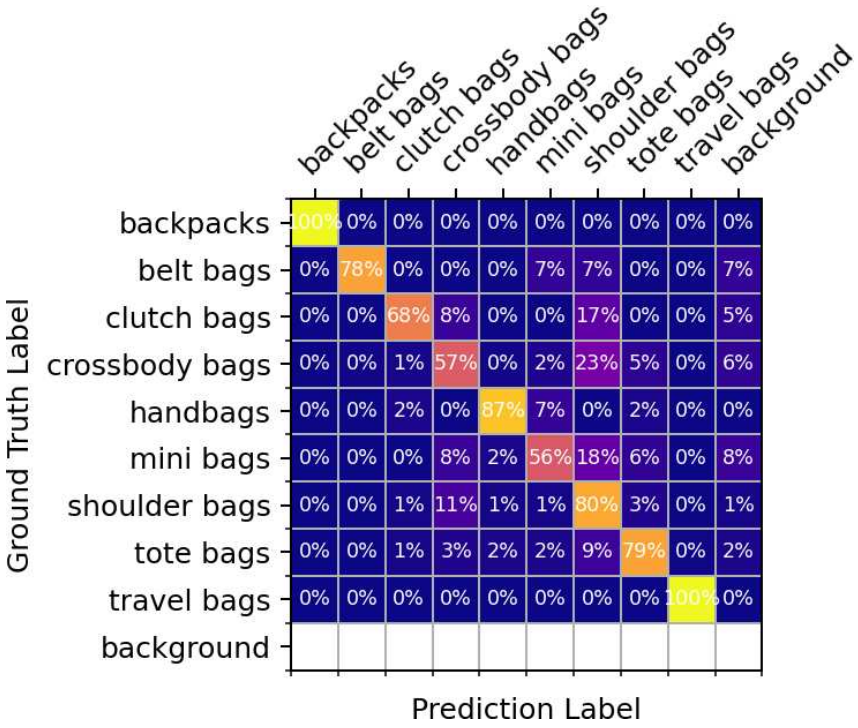

Figura 4.7.: Matrice di confusione relativa a TOOD

altre classi la percentuale media ottenuta non è sufficiente per un utilizzo in ambiente di produzione. Questo poiché non garantisce un livello di accuratezza adeguato a

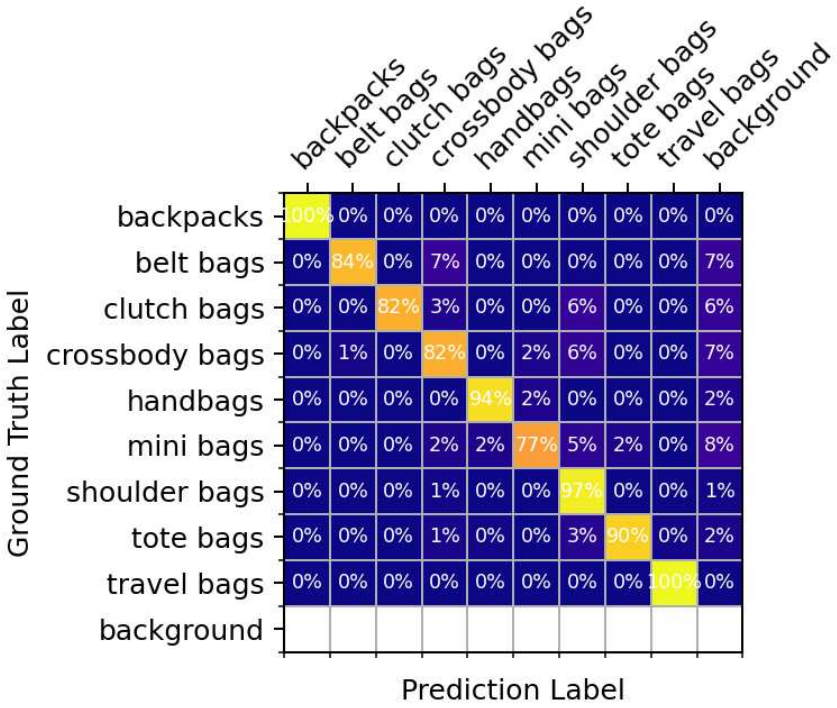

Figura 4.8.: Matrice di confusione relativa a DETR

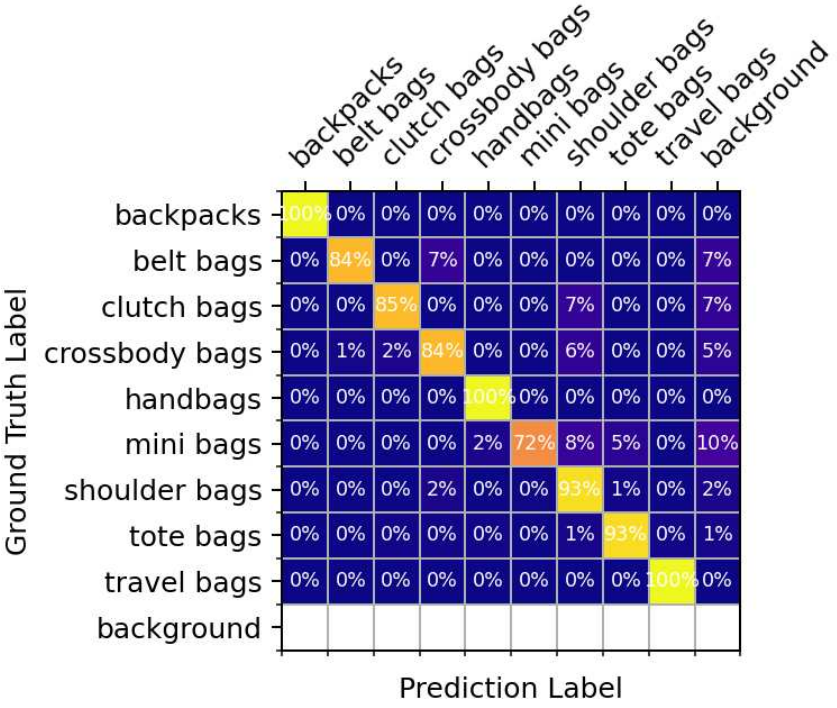

Figura 4.9.: Matrice di confusione relativa a DeformableDETR

soddisfare degli eventuali requisiti minimi. La situazione migliora invece per quanto riguarda i modelli basati su tecnologia *Transformer*. I risultati dei test confermano che

DETR e DeformableDETR si distinguono per le loro prestazioni globali, superando gli altri modelli sulla maggior parte delle classi a disposizione. Nonostante i modelli facciano confusione tra classi simili, le due reti dimostrano una notevole capacità di generalizzazione, risultando più performanti in termini di accuratezza complessiva. La forte variabilità delle *loss* misurate durante la fase di addestramento sembrerebbe non aver influito in modo rilevante nelle performance generali dei due modelli. Tuttavia, è bene sottolineare che le innovazioni introdotte con la *deformable attention* non hanno portato, almeno in questo ambito di lavoro, alcun vantaggio in termini di performance e accuratezza. Una considerazione importante è che tutti i modelli hanno dimostrato una capacità frequente di attribuire correttamente la classe *shoulder bags*. Questo potrebbe essere dovuto alla maggiore rappresentazione di questa classe nel dataset, che potrebbe aver influenzato i modelli a prediligere tale classe nelle predizioni. Nonostante gli sforzi di *data augmentation* per le classi meno rappresentate, tale classe è stata frequentemente scelta come predizione più probabile rispetto ad altre, indicando che il bilanciamento dei dati rimane un problema importante in questo lavoro.

Per completezza, di seguito sono riportati confronti diretti tra le reti (Figura 4.10 e 4.11), realizzati eseguendo inferenze su immagini campione fornite come input, al fine di valutarne le prestazioni in modo puntuale, ed osservare concretamente il significato di quanto detto fino ad ora.

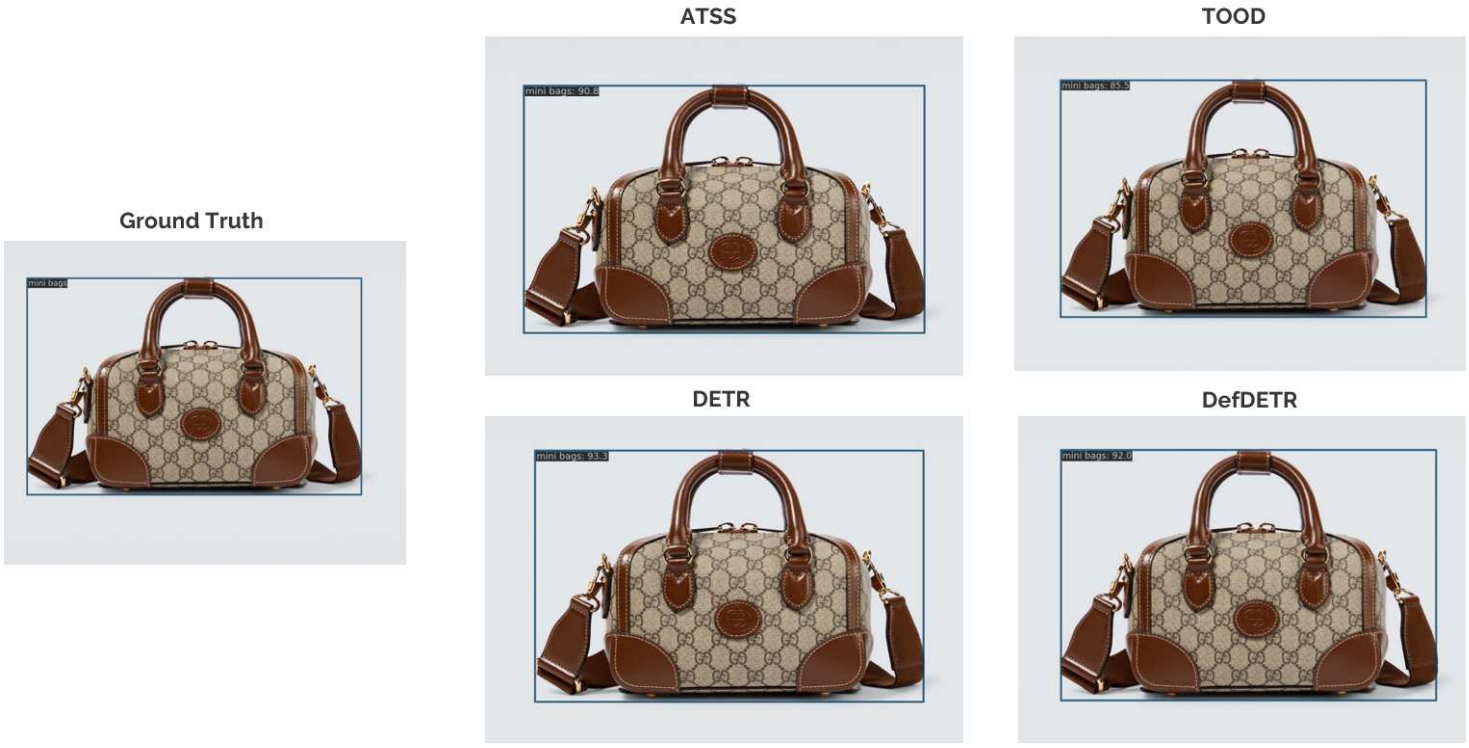

Figura 4.10.: Inferenza delle reti su una borsa *mini bags*

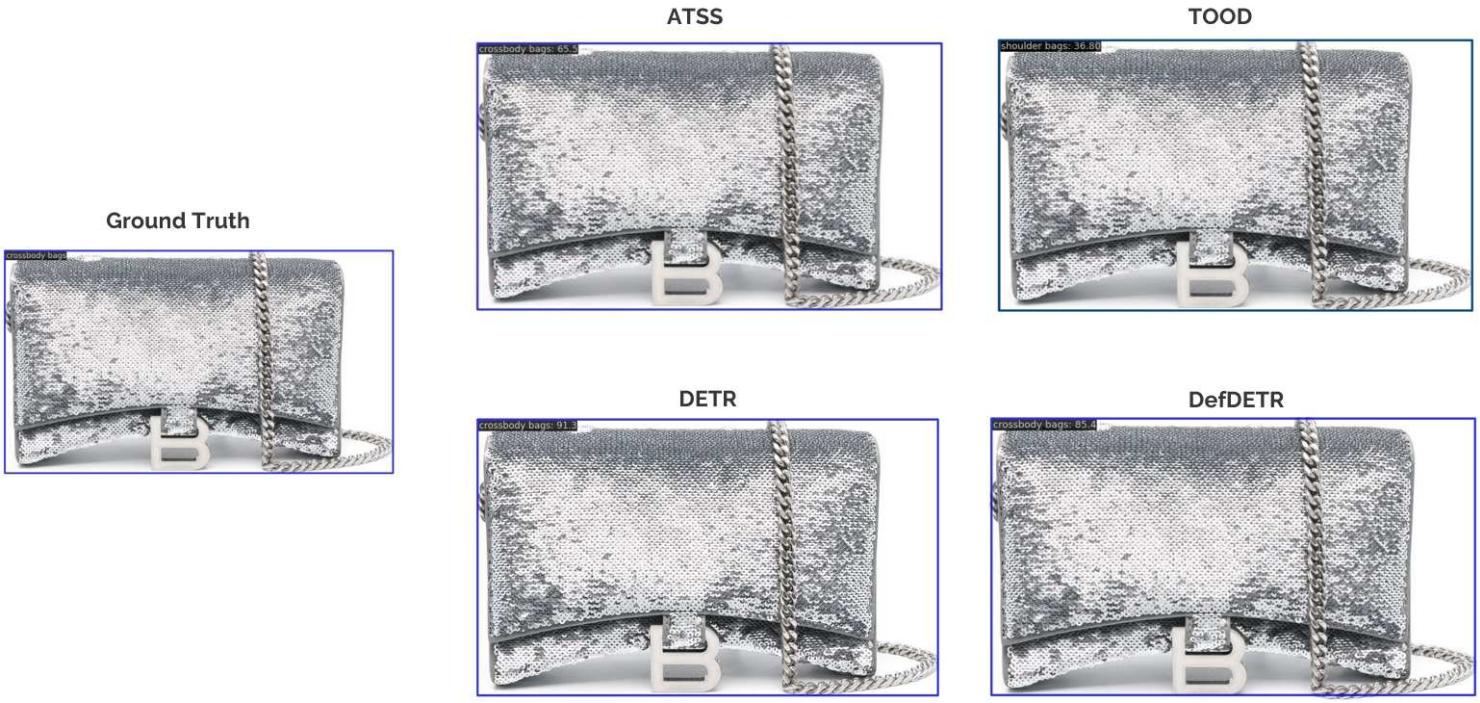

Figura 4.11.: Inferenza delle reti su una borsa *crossbody bags*

4.3. Inferenze sui dati di test

In questa sezione sono state analizzate le inferenze eseguite dai modelli sui dati di test attraverso due modalità distinte: una *non-interactive*, basata su richieste HTTP all'API REST implementata, e una *interactive*, fruibile tramite l'applicazione web sviluppata per questo progetto. Entrambi i sistemi utilizzano il modello di *object detection* DETR, selezionato per la sua capacità di generalizzare migliore rispetto alle altre reti.

Un aspetto rilevante, che è emerso durante l'interazione con l'applicativo sviluppato, riguarda il fenomeno del *cold start*, in qualche modo correlato al servizio `AWS Lambda` utilizzato. Come si può evincere dall'immagine riassuntiva riportata in Figura 4.12,

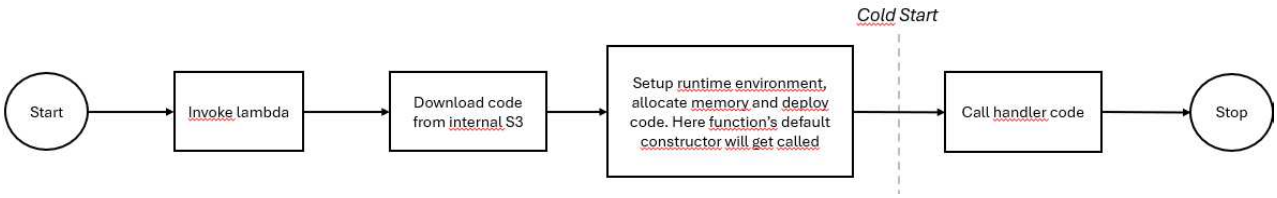

Figura 4.12.: Scenario fenomeno del *cold start*

questo problema si verifica quando una funzione `Lambda` viene invocata dopo un periodo di inattività e richiede un certo tempo per l'inizializzazione dell'ambiente di esecuzione, che include il caricamento del codice e l'installazione delle dipendenze. Il *cold start* può influire sui tempi di risposta, soprattutto in applicazioni con

requisiti di bassa latenza. La durata del *cold start* varia a seconda del linguaggio di programmazione utilizzato (linguaggi come Python e Node.js[2] hanno tempi di avvio più rapidi rispetto a Java[3] ad esempio) e della complessità del pacchetto di *deployment*, che può aggiungere ritardi significativi. Per affrontare tale problematica, esistono varie soluzioni che permettono di minimizzare o addirittura eliminare questi ritardi. Un approccio comune è il *warm start*, che mantiene attivo l'ambiente `Lambda` per un certo periodo, pronto a rispondere immediatamente alle richieste successive senza dover ripetere il processo di inizializzazione. Una delle tecniche più efficaci per implementare il *warm start* è l'uso della Provisioned Concurrency di AWS, che pre-inizializza un certo numero di container `Lambda`, eliminando i *cold start* per quelle istanze. Tuttavia, questo approccio implica un costo aggiuntivo, poiché gli ambienti sono mantenuti attivi indipendentemente dall'uso. Un'ulteriore innovazione è Lambda SnapStart, che cattura un'istantanea dell'ambiente di esecuzione inizializzato e la ripristina rapidamente alla prima invocazione della funzione, riducendo i tempi di avvio. Questa tecnologia è attualmente disponibile solo per le funzioni. Esistono anche approcci più semplici, come ridurre la dimensione del pacchetto di *deployment* o eliminare dipendenze non necessarie, per accelerare ulteriormente il processo di inizializzazione. Sebbene non sia stato implementato un sistema di *warm-up* automatico per le funzioni `Lambda`, vista la frequenza limitata di utilizzo dell'infrastruttura e il basso impatto sull'utente, l'applicativo sviluppato è stato dotato, sia lato *backend* sia lato *frontend*, di un meccanismo di gestione degli errori pensato per affrontare in modo efficace questa e altre eventuali problematiche.

Nelle sezioni successive verranno esaminate le differenze operative tra l'inferenza *non-interactive* e *interactive*. L'analisi fornirà una valutazione approfondita dei tempi di risposta, tenendo conto del limite massimo di esecuzione delle funzioni `Lambda`, pari a 15 minuti, e delle performance complessive del sistema in termini di accuratezza ed esperienza utente.

4.3.1. Processo non-interactive via API REST

Il servizio realizzato, come già anticipato, è reso possibile grazie alla creazione di un'immagine `Docker` dell'app sviluppata in Python. Questa immagine include tutte le librerie e le dipendenze necessarie per eseguire il modello scelto, insieme ai pesi opportuni ottenuti dalla fase di addestramento. In Figura 4.13 è stato riporto il *Dockerfile* utilizzato per questo scopo, che consente di integrare tutte le componenti software necessarie.

Dopo che l'immagine `Docker` è stata creata (*build*), essa rappresenta un pacchetto portabile del sistema applicativo, pronto per essere eseguito in ambienti diversi senza ulteriori configurazioni. L'immagine viene poi caricata (*push*) in una *repository* di `Amazon ECR`, come mostrato in Figura 4.14. Successivamente, utilizzando questa

[2] `https://nodejs.org/en`
[3] `https://www.java.com/it/`

```
FROM public.ecr.aws/lambda/python:3.11

RUN yum update -y && yum install -y wget ffmpeg libsm6 libxext6 ninja-build && yum clean all
RUN wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O miniconda.sh && \
    sh Miniconda3-latest-Linux-x86_64.sh -b -p /opt/miniconda

COPY . .

RUN /opt/miniconda/bin/conda install python=3.11 -y
RUN /opt/miniconda/bin/pip install torch==2.0.0+cpu torchvision==0.15.1+cpu torchaudio==2.0.1 \
    --index-url https://download.pytorch.org/whl/cpu
RUN /opt/miniconda/bin/pip install -U openmim && \
    /opt/miniconda/bin/mim install mmengine && \
    /opt/miniconda/bin/mim install "mmcv>=2.0.0"
RUN /opt/miniconda/bin/pip install awslambdaric opencv-python-headless

RUN cd mmdetection && /opt/miniconda/bin/pip install --no-cache-dir -e .
RUN /opt/miniconda/bin/conda install -c conda-forge boto3 -y

RUN mv /var/lang/bin/python3.11 /var/lang/bin/python3.11-clean && \
    ln -sf /opt/miniconda/bin/python /var/lang/bin/python3.11

ENV PYTHONPATH "/var/lang/lib/python3.11/site-packages:/"

CMD ["app.lambda_handler"]
```

Figura 4.13.: *Dockerfile* per l'applicazione `Lambda`

repository, è stato effettuato il *deploy* della `Lambda Function`, che viene attivata tramite `API Gateway` (vedi Figura 4.15 e 4.16). Questa integrazione permette l'esecuzione del modello in risposta alle richieste API, mantenendo l'infrastruttura *serverless*.

Per quanto riguarda il processo di natura *non-interactive*, le inferenze vengono eseguite tramite richieste POST all'API REST sviluppata. Sebbene questo processo sia completamente automatizzato nel *backend*, richiede una preparazione preliminare del *payload* da parte dell'utente o del sistema esterno che effettua la richiesta. In particolare, l'immagine deve essere convertita in formato Base64 e inserita nel *body* della richiesta sotto la chiave "*image*". Questa operazione aggiuntiva implica la necessità di strumenti o script per eseguire la conversione, il che potrebbe rallentare l'automazione completa del flusso.

In Figura 4.17 viene riportato un esempio di richiesta POST all'API, realizzata utilizzando Postman[4], dove l'immagine è stata codificata manualmente in Base64 e inserita nel body della richiesta stessa.

Una volta elaborata l'immagine fornita, il modello DETR restituisce una risposta in formato JSON, contenente le seguenti informazioni:

[4] `https://www.postman.com/`

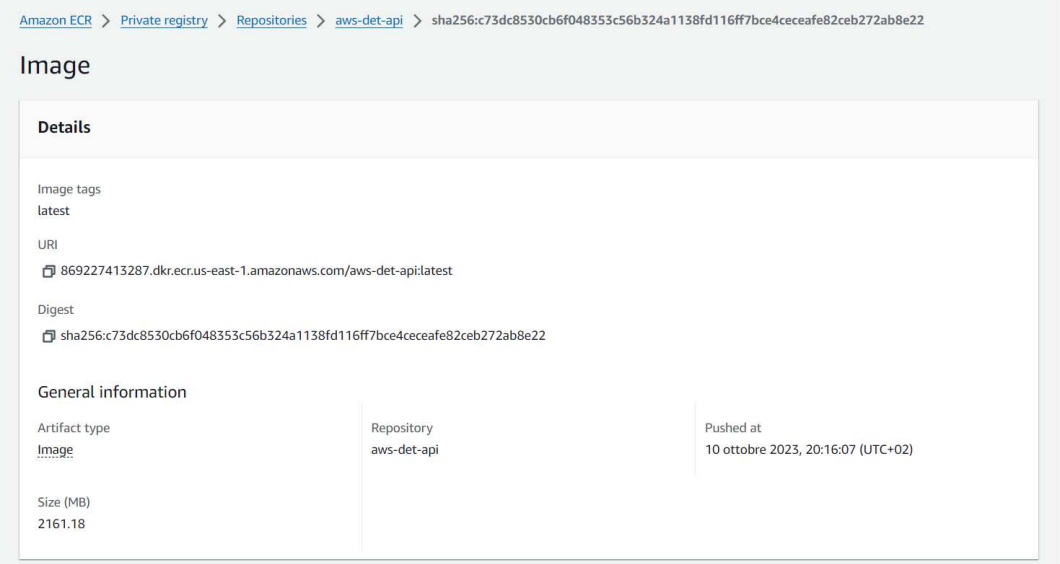

Figura 4.14.: Configurazione di `Amazon ECR` per l'applicativo

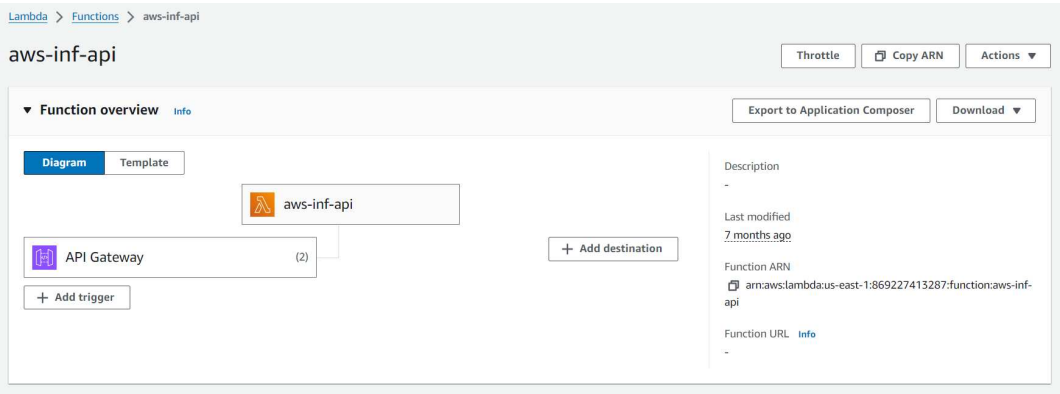

Figura 4.15.: Configurazione del servizio `Lambda` per l'applicativo

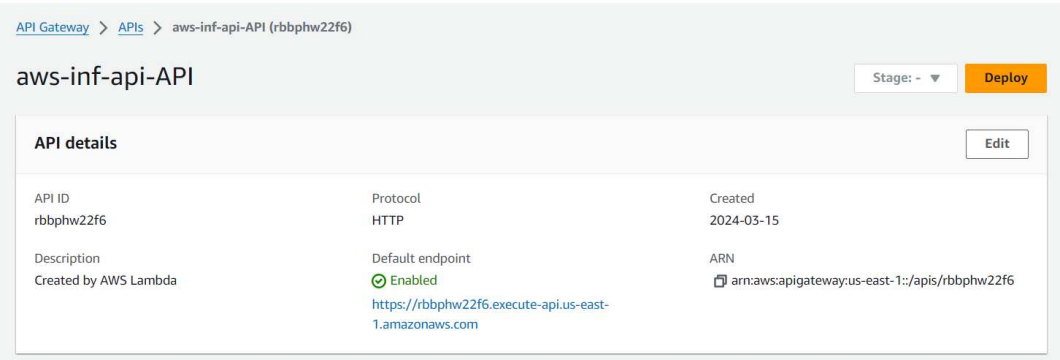

Figura 4.16.: Configurazione del servizio `API Gateway` per l'applicativo

- L'immagine elaborata dal modello con il bounding box che identifica la borsa, anch'essa codificata in Base64.

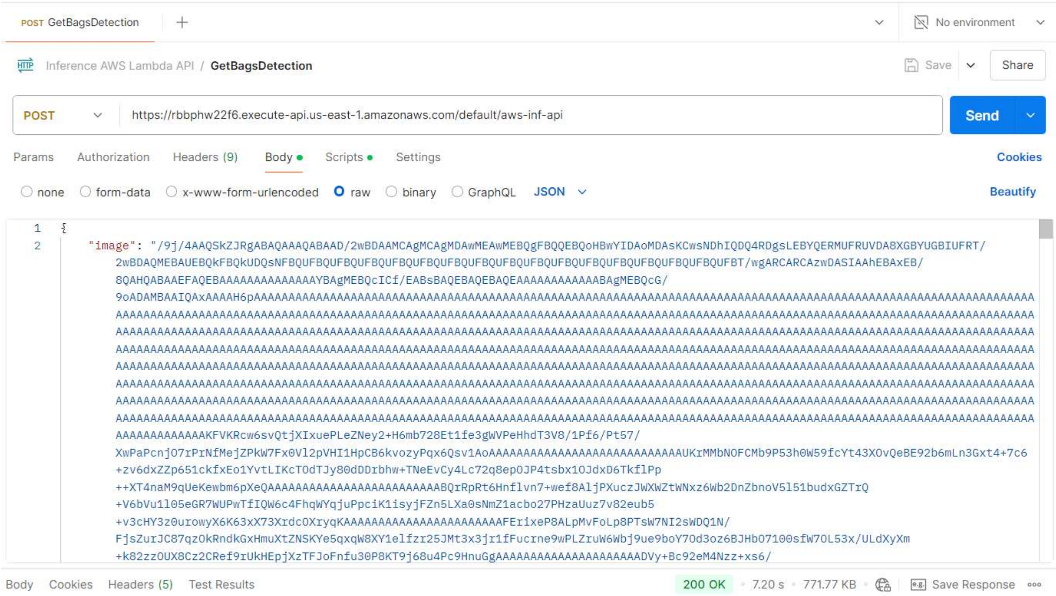

Figura 4.17.: Richiesta POST all'API realizzata via Postman

- La label della borsa individuato, ad esempio *shoulder bags*, *handbags*, ecc.
- Le coordinate Ymin, Ymax, Xmin, Xmax del rettangolo che racchiude l'oggetto individuato.
- La confidenza con cui il modello assegna l'oggetto alla specifica classe sotto forma di probabilità in percentuale
- Una caratterizzazione dei colori predominanti dell'oggetto rilevato. Questa informazione viene ottenuta applicando l'algoritmo di *clustering* K-means[29] all'oggetto identificato. Per garantire un'analisi più accurata, i colori molto vicini al bianco, che potrebbero rappresentare lo sfondo, vengono esclusi.

Di seguito viene mostrato un esempio di risposta tipo, associata alla richiesta effettuata sopra.

```
{
    "img": "**base64 code**",
    "label": "shoulder bags",
    "score": 91.3,
    "bbox": [
        92.57810974121094,
        327.4285583496094,
        750.9007568359375,
        772.4892578125
    ],
```

```
    "dominant_hex": "#a29487",
    "palette": [
        "#fefefd",
        "#2c241c",
        "#a29487",
        "#284132",
        "#524339",
        "#dbd0c4",
        "#841d27",
        "#b7aa9d",
        "#6e6055",
        "#8a7d70"
    ]
}
```


Queste informazioni sono fondamentali per studiare i trend nel contesto lavorativo in cui il sistema è stato applicato. Analizzando un numero significativo di immagini, è possibile individuare *pattern* nascosti dai quali estrarre *insight* strategici utili a supportare decisioni aziendali. Questi pattern possono, ad esempio, identificare le tendenze emergenti nella moda o nel design delle borse, offrendo un vantaggio competitivo in termini di strategia di mercato.

Come anticipato nell'introduzione di questo capitolo, il servizio sviluppato include la gestione degli errori che possono verificarsi in due situazioni principali. Da un lato, nel caso di un *payload* mal formattato, ad esempio, in assenza del *body* o con un'immagine codificata in Base64 in modo errato, viene generata una risposta di "*Bad Request*" con lo *status code* 400 e un messaggio d'errore appropriato (Figura 4.18). Dall'altro, se l'inferenza del modello non dovesse andare a buon fine, il sistema restituirebbe un "*Internal Server Error*" con *status code* 500 (Figura 4.19).

4.3.2. Processo interactive via applicazione web

Oltre all'API, che rappresenta il *backend* del sistema, è stato realizzato un semplice *frontend* per consentire l'interazione con il servizio in maniera più immediata e intuitiva. Questo permette all'utente di caricare immagini e ottenere inferenze dal modello in tempo reale, senza dover eseguire operazioni manuali complesse. Il processo *interactive* offre un'esperienza più *user-friendly* grazie all'applicazione web sviluppata. A differenza del processo *non-interactive*, qui l'utente può caricare direttamente le immagini tramite l'interfaccia web senza preoccuparsi di conversioni in Base64 o della preparazione manuale del *payload*. L'applicazione gestisce automaticamente la conversione e l'invio della richiesta all'API, rendendo l'interazione fluida e immediata. Il frontend è stato sviluppato senza l'uso di *framework* complessi, utilizzando una

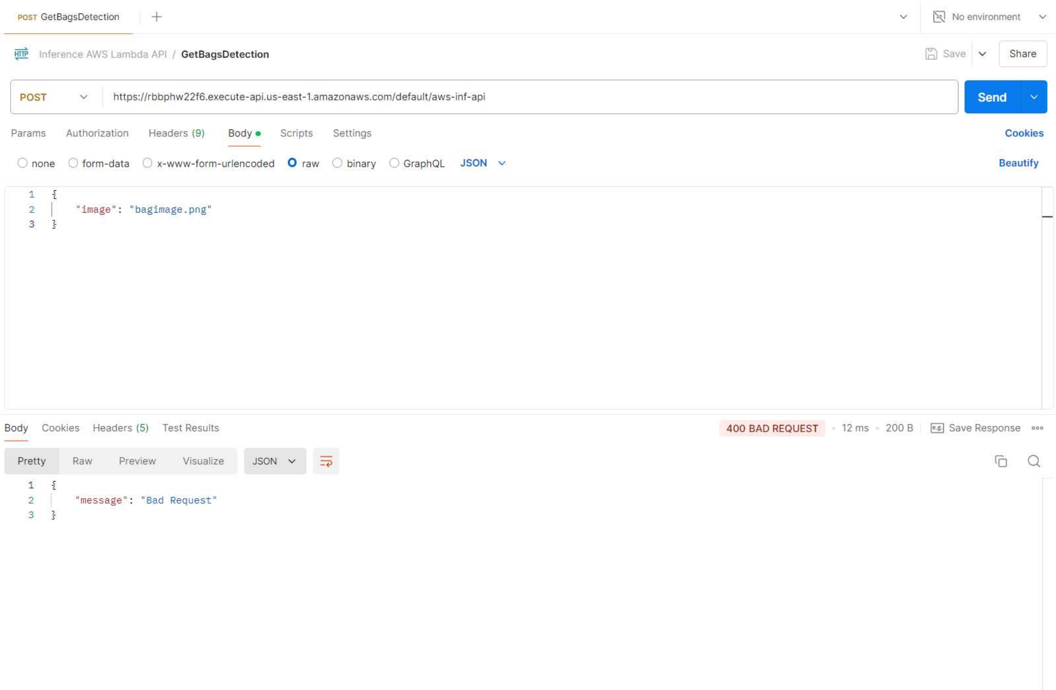

Figura 4.18.: Esempio di *"Bad Request"* via Postman

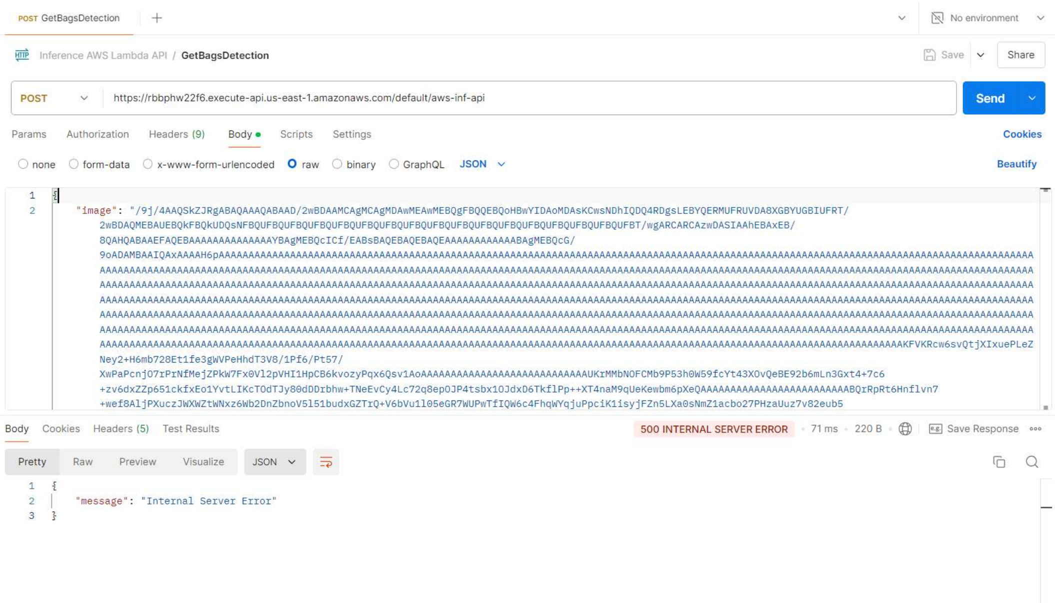

Figura 4.19.: Esempio di *"Internal Server Error"* via Postman

semplice pagina HTML con il supporto di JavaScript[5] per il caricamento delle immagini e per effettuare le richieste HTTP all'API. L'aspetto grafico è stato migliorato con l'uso di CSS[6] per creare un'interfaccia più piacevole e intuitiva.

[5] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`
[6] `https://developer.mozilla.org/en-US/docs/Web/CSS`

Il codice sorgente del *frontend* è stato compresso in un file zip e caricato su `AWS Amplify` (vedi Figura 4.20), che ha reso disponibile l'applicazione su web.

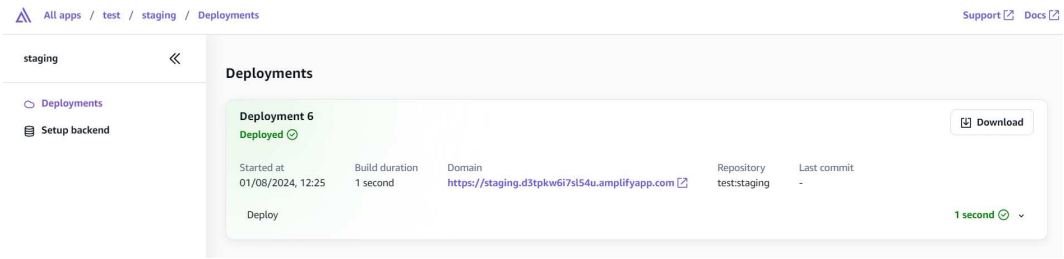

Figura 4.20.: Configurazione `AWS Amplify` per l'applicativo

In Figura 4.21 è mostrato il risultato dell'applicazione distribuita tramite `AWS Amplify`.

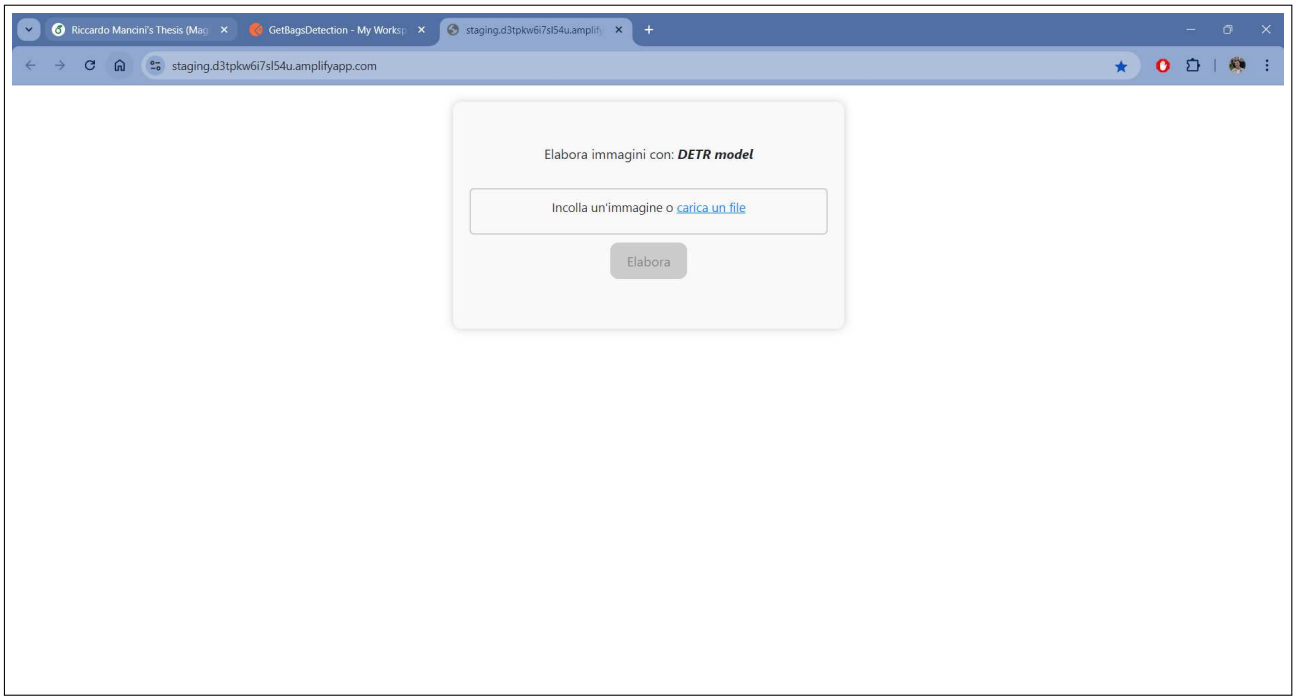

Figura 4.21.: Risultato dell'applicazione web al primo caricamento

L'utente può caricare o incollare un'immagine direttamente nell'interfaccia attraverso una form centrale visibile sulla pagina (vedi Figura 4.22).

L'interfaccia è intuitiva e progettata per rendere il caricamento delle immagini semplice e immediato, con pulsanti ben visibili per facilitare l'interazione. Sono stati implementati alcuni controlli logici in JavaScript per migliorare l'esperienza utente, impedendo, ad esempio, di caricare la stessa immagine più volte o di effettuare elaborazioni ripetute consecutive. Una volta caricata l'immagine, l'utente può inviare la richiesta con un solo click, senza bisogno di ulteriori passaggi. Dopo pochi secondi, l'applicazione visualizza i risultati dell'inferenza in tempo reale. L'immagine viene

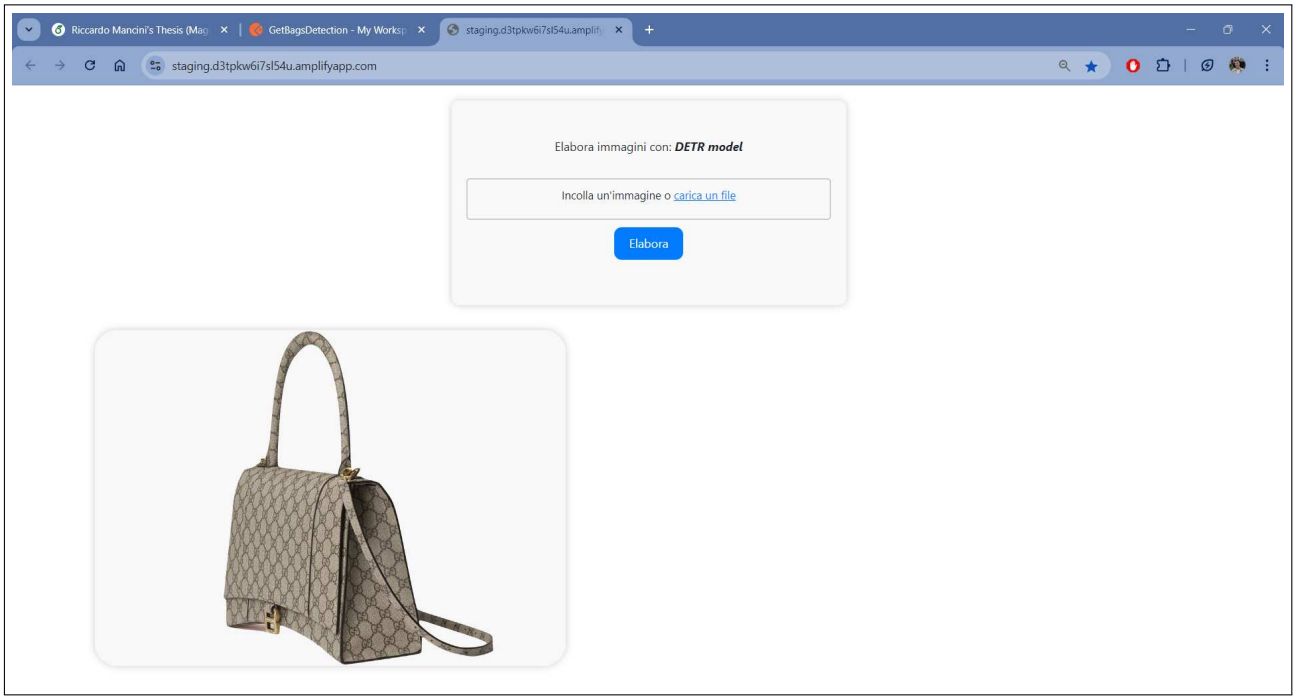

Figura 4.22.: Risultato dell'applicazione web al caricamento dell'immagine

elaborata, e il *bounding box* attorno all'oggetto rilevato è mostrato direttamente sull'immagine restituita, che viene opportunamente sostituita con quella caricata. Inoltre, i colori dominanti vengono rappresentati graficamente, fornendo un *feedback* immediato e visivo delle caratteristiche cromatiche della borsa. In Figura 4.23 è stato riportato quanto detto.

Inoltre, anche lato *frontend*, poiché viene effettuata una richiesta HTTP utilizzando JavaScript, è stata prevista una gestione degli errori in risposta al servizio chiamato, inclusa la gestione del fenomeno del *cold start*, come trattato in precedenza. In caso di errore del tipo riportato in Figura 4.24, l'utente riceverà un messaggio chiaro e dettagliato che descrive il problema e suggerisce come procedere. Questo può includere indicazioni per riprovare l'operazione o una spiegazione dell'errore che impedisce il completamento dell'inferenza.

In Figura 4.25 è riportato un estratto del codice sviluppato per gestire queste casistiche nell'istante in cui viene premuto il pulsante "Elabora", ed in Figura 4.26 il caso specifico in cui il servizio non si trovi disponibile.

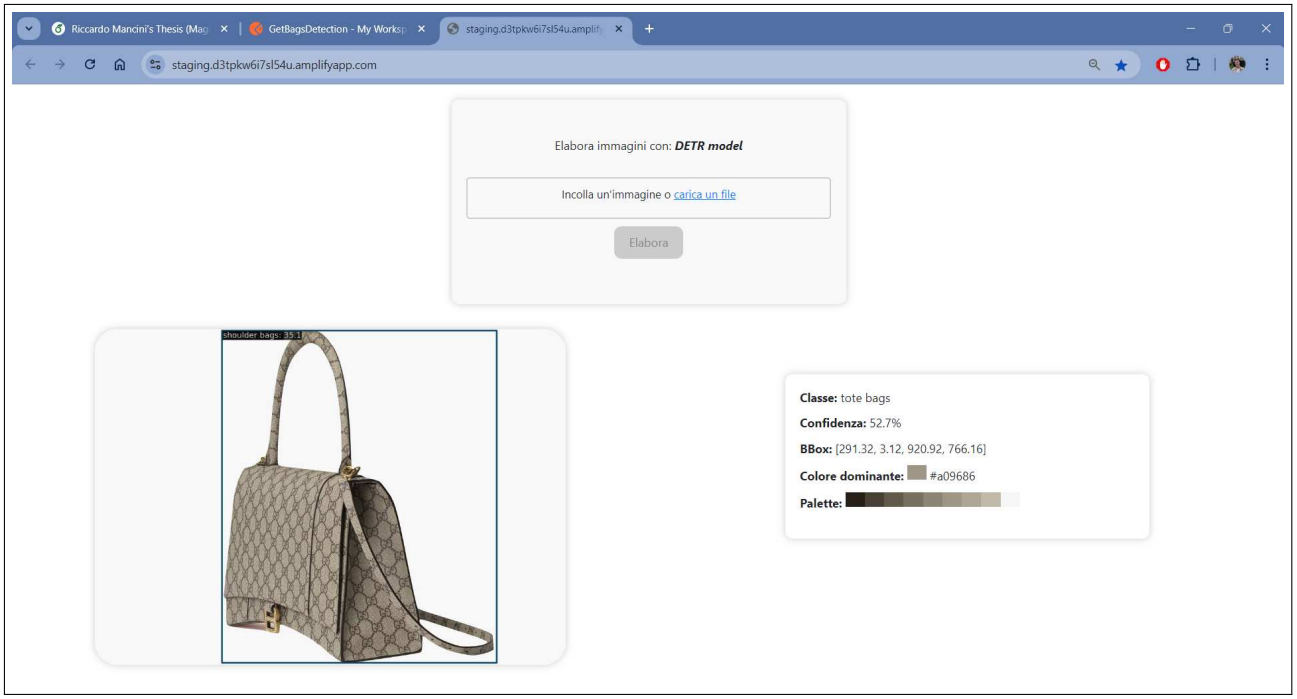

Figura 4.23.: Risultato dell'applicazione web a seguito dell'elaborazione dell'immagine

Figura 4.24.: Risposta al servizio non disponibile

```
process_btn.onclick = async () => {
    process_btn.disabled = true;
    const selectedFile = imgInput.files[0];
    if (!previousFile || !(previousFile === selectedFile)) {
        previousFile = selectedFile;
        removeAllChildren(resultList);
        document.getElementById('loaderDiv').style.display = 'block';

        let res = await do_inference(base64Img);
        document.getElementById('loaderDiv').style.display = 'none';

        if (res.status === 200) {
            res = await res.json();
            console.log(res);

            // ...
            // Handling the successful response, such as displaying the image,
            // creating elements for resultList, and showing resultDiv.
            // ...

        } else {
            res = await res.json();
            console.error("Errore:", res.error);

            let description = '';
            let solution = '';

            if (res.status === 503) {
                description = 'Si è verificato un problema di "Lambda Cold Start".
                ↪  Questo accade quando la funzione viene invocata dopo un
                ↪  periodo di inattività.';
                solution = 'Attendi qualche secondo e aggiorna nuovamente la
                ↪  pagina.';
            } else if (res.status === 400) {
                description = 'La richiesta inviata al servizio non è corretta.';
                solution = 'Verifica i dati inviati e riprova.';
            } else if (res.status === 500) {
                description = 'Il server ha riscontrato un errore durante
                ↪  l\'elaborazione della richiesta.';
                solution = 'Riprova più tardi.';
            } else {
                description = 'Si è verificato un errore imprevisto durante
                ↪  l\'elaborazione della richiesta.';
                solution = 'Riprova più tardi.';
            }

            resultList.style.display = 'block';
            resultList.appendChild(createListItem('Errore',
            ↪  `<i>${res.error}</i>`));
            resultList.appendChild(createListItem('Descrizione',
            ↪  `<i>${description}</i>`));
            resultList.appendChild(createListItem('Soluzione',
            ↪  `<i>${solution}</i>`));

            resultDiv.style.display = 'block';
        }
    }
};
```

Figura 4.25.: Gestione della risposta al click del bottone nel codice JavaScript

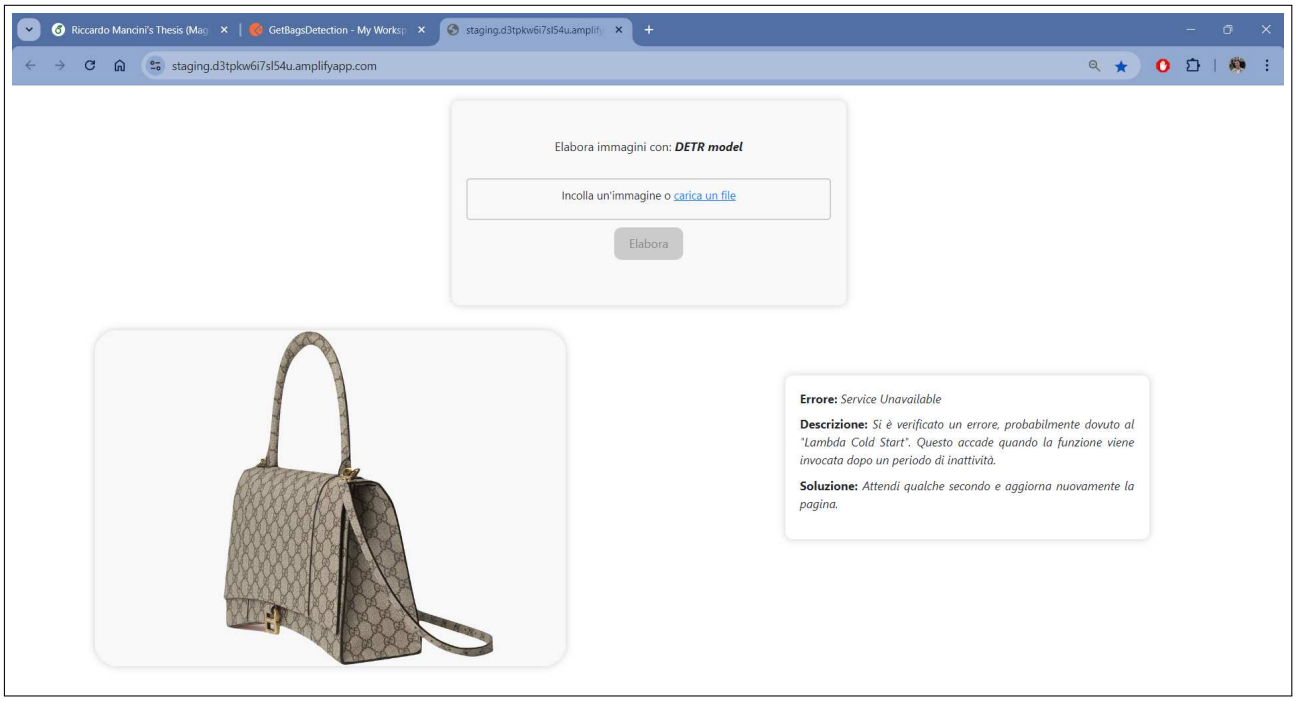

Figura 4.26.: Risultato dell'applicazione web con il servizio non disponibile

Capitolo 5.

Conclusione

Questa tesi esplora il ciclo di vita dell'analisi dei dati nel settore della moda, evidenziando come l'elaborazione di grandi quantità di informazioni visive possa supportare decisioni strategiche in vari contesti, dall'analisi delle tendenze di mercato alla progettazione di collezioni. In particolare, è stato mostrato come l'uso di tecniche avanzate di apprendimento automatico, integrate con un'infrastruttura tecnologica adeguata, consenta di raccogliere, gestire e analizzare dati in modo efficiente. Questo consente di creare un ecosistema tecnologico in grado di sfruttare le potenzialità del deep learning e dell'automazione, per rendere l'intero processo di analisi dei dati più efficace. Uno degli obiettivi primari della ricerca è stato quello di evidenziare le opportunità offerte dall'ecosistema open del web, mostrando come l'enorme quantità di dati disponibile online possa essere trasformata in valore attraverso l'utilizzo di strumenti adeguati e tecnologie di automazione moderne. Nonostante la semplificazione e il supporto ad una raccolta di dati dal web migliore, un tempo complessa e dispendiosa, il vero valore emerge solo quando tali dati vengono analizzati e trasformati in informazioni strategiche attraverso l'impiego di algoritmi altamente specializzati e mirati a compiti specifici.

La "democratizzazione" dell'automazione nella raccolta e nell'elaborazione di grandi volumi di dati ha reso accessibili a un pubblico sempre più ampio strumenti che fino a pochi anni fa erano riservati a grandi aziende con elevate capacità tecnologiche. Tuttavia, la capacità di sviluppare algoritmi efficaci non può prescindere da una riflessione critica sulla qualità dei dati utilizzati. La semplice disponibilità di grandi quantità di dati non garantisce risultati di valore: è fondamentale che i dati siano accurati, rilevanti e rappresentativi del fenomeno studiato. Il rischio di utilizzare dataset di scarsa qualità, con conseguenti risultati inaccurati o distorsioni, è ancora concreto e non deve essere sottovalutato.

Tuttavia, nonostante i risultati positivi, rimangono sfide significative da affrontare. Le limitazioni relative alla qualità e alla varietà dei dataset, così come gli elevati requisiti computazionali necessari per l'addestramento dei modelli, evidenziano aree che richiedono ulteriori sviluppi e ottimizzazioni. In questo contesto, l'adozione di soluzioni come il cloud computing, che sollevano l'utente dalla gestione diretta delle infrastrutture, si è rivelata una delle strade più promettenti per sostenere lo sviluppo e spingere i risultati verso lo stato dell'arte in diversi settori. Il continuo progresso

delle tecnologie di deep learning, combinato con l'evoluzione delle infrastrutture computazionali, promette di superare molte delle attuali limitazioni. Questo progresso non solo migliorerà la capacità di analizzare e interpretare grandi quantità di dati visivi, ma consentirà anche di applicare queste tecnologie in contesti sempre più complessi e diversificati. La moda, in particolare, sarà uno dei settori che trarrà maggiore beneficio da tali innovazioni. La possibilità di ottimizzare previsioni, anticipare tendenze e personalizzare l'offerta in modo sempre più preciso rappresenta un futuro entusiasmante per la sinergia tra tecnologia e moda.

5.1. Sviluppi futuri

Come già anticipato, di certo una delle principali aree di sviluppo futuro riguarda la necessità di ampliare e migliorare i dataset utilizzati per l'addestramento dei modelli di *object detection*. Le limitazioni temporali hanno rappresentato un ostacolo in questo lavoro, influenzando negativamente la dimensione del dataset ottenuto. Un periodo di raccolta dati più esteso, attraverso il software sviluppato, avrebbe consentito di ottenere un numero maggiore di campioni, soddisfacendo così il requisito minimo di dati per classe, spesso evidenziato nella letteratura sui modelli di rilevamento. Questa espansione quantitativa dei dati è essenziale per migliorare le prestazioni dei modelli e ridurre il rischio di sotto-campionamento, che può compromettere la capacità del sistema di generalizzare efficacemente su diverse classi di oggetti. Oltre a questo, sarebbe utile aggiornare il software di raccolta dati affinché acquisisca immagini non solo su sfondo bianco, ma anche in cui le borse siano indossate da persone o inserite in scenari reali, o addirittura con più occorrenze di borse all'interno della stessa immagine. Attualmente, tali miglioramenti non trovano applicabilità immediata, vista la definizione rigida del dominio delle immagini accettabili, come da richiesta, ma potrebbero rappresentare un'evoluzione significativa per il futuro. In concomitanza con quanto detto, un'altra area di un eventuale sviluppo riguarda la riduzione dei bias presenti nei dataset. Attualmente, i modelli di deep learning tendono a replicare e amplificare i pattern presenti nei dati di addestramento, riproducendo involontariamente le distorsioni introdotte durante le fasi di raccolta e annotazione. Per ottenere modelli più equi e affidabili, sarà necessario sviluppare dataset più bilanciati e oggettivi, minimizzando i pregiudizi impliciti.

Dal punto di vista tecnico invece, un interessante sviluppo futuro riguarda l'adozione di tecniche di Few-Shot Learning. Come si può notare dalla panoramica riportata in Figura 5.1, tale tecnica consente ai modelli di riconoscere nuovi oggetti utilizzando un numero ridotto di esempi, riducendo così la necessità di creare e annotare dataset di grandi dimensioni, o perlomeno non fin dal primo momento. Nella pratica, il processo prevede tre fasi principali: un addestramento di base sulle classi esistenti, un affiancamento con pochi esempi per nuove classi, e infine una valutazione del modello sulle classi base e nuove. Nel settore della moda, questa capacità di adattamento risulterebbe particolarmente vantaggiosa, permettendo ai modelli di generalizzare

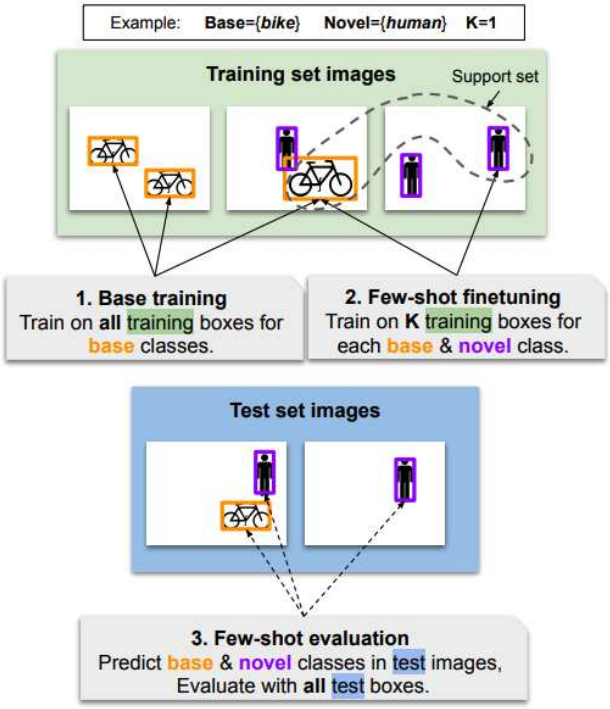

Figura 5.1.: Overview del Few-Shot Object Detection

Immagine da: `https://paperswithcode.com/task/few-shot-object-detection/latest`

meglio e di essere aggiornati facilmente con pochi dati etichettati, ed al tempo stesso riuscendo ad ottenere risultati soddisfacenti fin da subito.

Un'altra direzione promettente per dei possibili sviluppi futuri è rappresentata da un approccio differente al *task*, ossia la cosiddetta Domain Adaptation. La Domain Adaptation è un caso particolare di *transfer learning*, in cui l'obiettivo principale è adattare un modello addestrato su un dominio (dominio di origine) per funzionare efficacemente su un dominio diverso (dominio di destinazione). Questo processo si basa sull'identificazione dei fattori latenti comuni tra i due domini e sull'adattamento delle loro rappresentazioni per ridurre il disallineamento, tra gli spazi delle caratteristiche attraverso i quali vengono rappresentati i dati. In altre parole, la Domain Adaptation cerca di colmare le differenze tra i due domini in modo che un modello addestrato su uno possa generalizzare anche sull'altro. In Figura 5.2 una breve immagine riepilogativa ad alto livello di questo approccio. Si può immaginare, ad esempio, un modello di classificazione delle immagini addestrato su fotografie scattate con un cellulare (dominio di origine). Se lo stesso modello deve essere utilizzato per classificare immagini scattate con una fotocamera DSLR (dominio di destinazione), le differenze nella qualità dell'immagine, risoluzione e condizioni ambientali possono causare problemi di generalizzazione. Qui interviene la Domain Adaptation, cercando di adattare il modello alle nuove condizioni senza necessitare di un completo riaddestramento. Esistono diversi approcci alla Domain Adaptation.

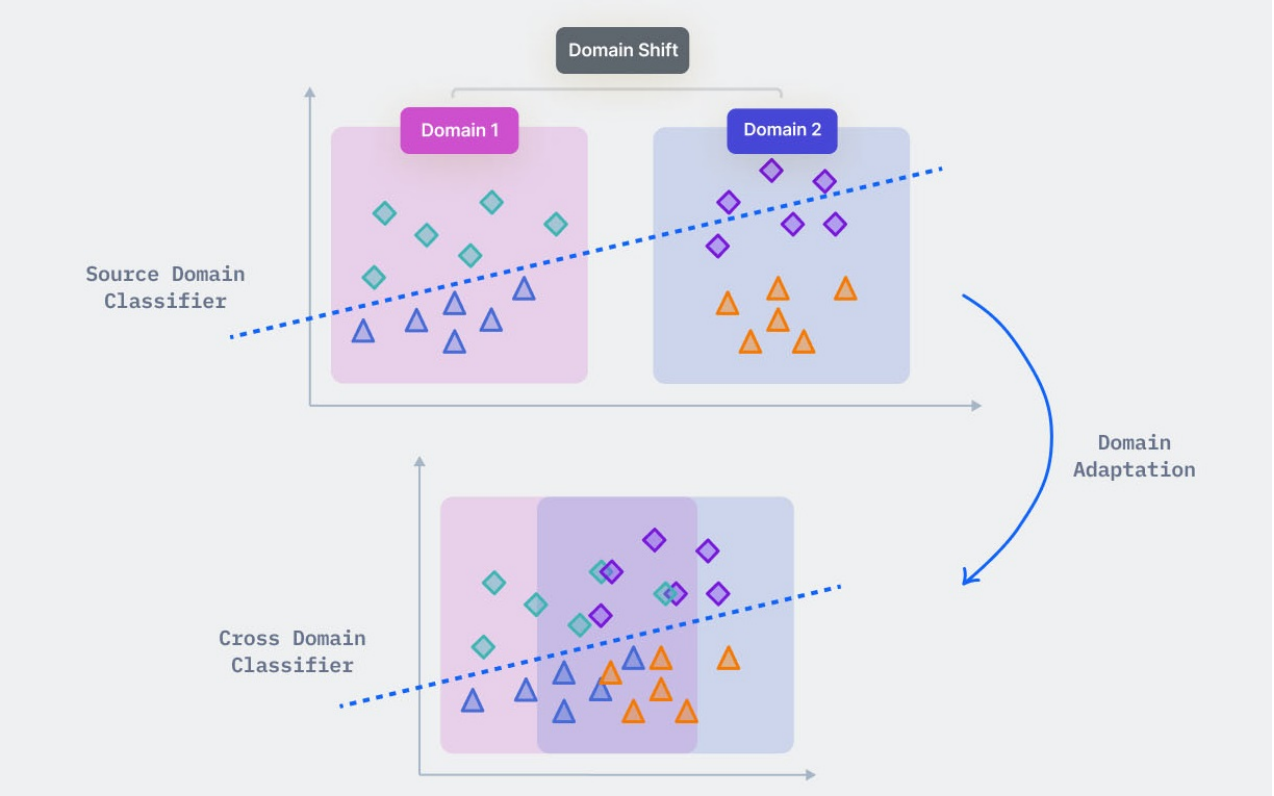

Figura 5.2.: Overview del concetto di Domain Adaption

Immagine da: `https://www.v7labs.com/blog/domain-adaptation-guide`

Uno dei più comuni è la Domain Adaptation non supervisionata (UDA), dove il modello viene addestrato in un dominio di origine con dati etichettati, ma deve essere in grado di adattarsi a un dominio di destinazione dove i dati non sono etichettati. Altri metodi includono la Domain Adaptation supervisionata e semi-supervisionata, in cui alcune etichette sono disponibili nel dominio di destinazione, ma non tutte. Questi approcci cercano di minimizzare la distanza tra le distribuzioni del dominio di origine e di destinazione, utilizzando tecniche come la *domain confusion* o reti neurali con perdite specifiche per confondere le caratteristiche dei due domini e rendere il modello più generale e applicabile. E' facile comprendere quindi, come nel contesto di questo lavoro l'adozione di questo approccio possa essere particolarmente utile per estendere il flusso di raccolta dati a una maggiore eterogeneità di fonti, garantendo che il modello mantenga la capacità di generalizzare anche quando le condizioni dell'immagine o la qualità variano notevolmente. Tale approccio permetterebbe di affrontare la sfida dell'elevata dinamicità del settore, senza richiedere un riaddestramento completo per ogni nuova sorgente informativa. L'uso dell'UDA, ad esempio, potrebbe supportare l'adattamento automatico del modello a nuove condizioni senza l'uso di dati etichettati, riducendo significativamente i tempi e i costi legati alla raccolta e all'etichettatura di nuovi dataset.

Dopo aver esaminato diverse prospettive, emerge chiaramente come queste tecnologie di analisi dei dati, grazie agli approcci innovativi illustrati, possano rispondere con crescente efficacia alle sfide e alle esigenze di settori in continua evoluzione.

Appendice A.

Funzioni principali

A.1. Web Scraping

In questa sezione sono descritte le principali funzioni dei software di *scraping* sviluppati, che hanno permesso di definire le prime due fasi di creazione del dataset di riferimento, come illustrato nella sezione 3.2. La particolarità di queste funzioni risiede nella loro capacità di essere eseguite sia in modo sequenziale, in base alle regioni e al genere (maschile o femminile) delle borse da raccogliere, sia in modo concorrente. Ovviamente tali funzioni andranno a richiamare ulteriori sottofunzioni sviluppate per svolgere *task* specifici, ma per motivi di spazio non verranno riportate in questo documento. Di seguito vengono riportati i metodi specifici utilizzati per ciascun *e-tailer* considerato.

A.1.1. Farfetch

```
def index_pages(self):
    df = pd.DataFrame(columns=['bagLink', 'regionCode', 'gender', 'brand'])
    gender_links = [config['MEN_BAGLIST_URL'], config['WOMEN_BAGLIST_URL']]
    region_link = self.get_region_link(self.region)
    brands = config['BRAND_LIST']
    try:
        for idx, g_link in enumerate(gender_links):
            for brand in brands:
                footer = config['FOOTER_WOMEN_URL'] if idx == 1 else
                ↪  config['FOOTER_MEN_URL']
                self.driver.get(region_link + g_link + brand + footer)
                self._implicit_wait()
                self._accept_cookies()
                self._close_newsletter()
                self._scroll_down_page()
                n_page = self._get_n_pages()
                print(n_page)
                current_page = 1
                while current_page <= int(n_page):
                    if current_page != 1:
                        self._close_newsletter()
                        self._scroll_down_page()
                    bag_ul = self.driver.find_elements(By.XPATH,
                    ↪  config['BAG_LIST'])
```

```
                    for li in bag_ul:
                        bag_link = li.find_element(By.TAG_NAME,
                        ↪  'a').get_attribute('href')
                        if config['BASE_URL_FARFETCH'] in bag_link:
                            df.loc[len(df)] = [bag_link, self.region, 'women' if
                            ↪  idx == 1 else 'men', brand]
                    current_page += 1
                    self._next_page()

    except BaseException as e:
        print(e)

    finally:
        if os.path.isfile(config['BAGS_LINKS_PATH']):
            df.to_csv('./backup_ex.csv', index=False)
            df1 = pd.read_csv(filepath_or_buffer=config['BAGS_LINKS_PATH'])
            df = pd.concat([df1, df])
        df.to_csv(config['BAGS_LINKS_PATH'], index=False)
        print(df.shape)
        self._close_driving()
```

```
def bags_scrape(self, gender):
    csv_df = pd.read_csv(filepath_or_buffer=config['BAGS_LINKS_PATH'])
    df_filtered = csv_df[(csv_df['regionCode'] == self.region) &
    ↪  (csv_df['gender'] == gender)]
    json_df = pd.DataFrame(columns=['gender', 'region', 'img',
    ↪  'rgb_dominant_color', 'name_dominant_color',
                                    'url', 'date_time', 'id_farfetch',
                                    ↪  'id_brand', 'brand', 'title', 'type',
                                    'subtype',
                                    'currency', 'price', 'original_price',
                                    ↪  'discount_percentage',
                                    'cluster_price', 'description', 'status',
                                    ↪  'materials',
                                    'info_materials', 'info_size'])

    # get json file
    json_name = self.region + '_' + gender
    json_path = [pos_json for pos_json in os.listdir(config['JSON_PATH_FOLDER'])
                 if pos_json.endswith('.json') and json_name in
                 ↪  Path(pos_json).stem]
    print(json_path)

    # resume scraping if json file exists
    if len(json_path) != 0:
        with open(config['JSON_PATH_FOLDER'] + json_path[0], encoding='utf-8') as
        ↪  file:
            json_file = json.load(file)
            json_df = pd.DataFrame.from_dict(json_file)
            df_filtered = df_filtered.tail(df_filtered.shape[0] -
            ↪  json_df.shape[0])
```

```
23          os.remove(config['JSON_PATH_FOLDER'] + json_path[0])
24
25      print(df_filtered.head(), df_filtered.shape)
26      json_df = self.get_bag_info(json_df, df_filtered)
27
28      # json creation
29      json_dict = json_df.to_dict(orient='records')
30      print(len(json_dict))
31      with open(config['JSON_PATH_FOLDER'] + self.region + '_' + gender + '_' +
32                datetime.now().strftime("%Y-%m-%d") + '.json', "w",
                  ↪  encoding='utf-8') as outfile:
33          json.dump(json_dict, outfile, indent=4, default=lambda o: '<not
            ↪  serializable>', ensure_ascii=False)
34
35  def get_bag_info(self, json_df: pd.DataFrame, csv_df: pd.DataFrame) ->
    ↪  pd.DataFrame:
36      try:
37          size_metric = True
38          for row in csv_df.to_dict('records'):
39              bagInfo: dict = {}
40              bag_url = row['bagLink']
41              bagInfo['gender'] = row['gender']
42              bagInfo['region'] = row['regionCode']
43              bagInfo['url'] = bag_url
44              bagInfo['date_time'] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
45
46              self.driver.get(bag_url)
47              time.sleep(2)
48              # sold out bag
49              sold_out_syntax = config['SOLD_OUT_STX'][self.region]
50              if self._check_exists_by_xpath(config['SOLD_OUT']) and \
51                      sold_out_syntax in self.driver.find_element(By.XPATH,
                        ↪  config['SOLD_OUT']).text.lower():
52                  print('SOLD OUT! - ', bag_url)
53                  self._close_newsletter()
54                  bagInfo['status'] = ['SOLD OUT']
55                  img_url = self.driver.find_element(By.XPATH,
                    ↪  '//*[@id="content"]/div/div[1]/div/div[1]/img') \
56                      .get_attribute('src')
57                  bagInfo['img'] = img_url
58                  bagInfo['brand'] = self.driver.find_element(By.XPATH,
59                                                              '//*[@id="content"]/d
                                                                ↪  iv/div[1]/div/div
                                                                ↪  [2]/h1/a').text
60                  bagInfo['title'] = self.driver.find_element(By.XPATH,
61                                                              '//*[@id="content"]/d
                                                                ↪  iv/div[1]/div/div
                                                                ↪  [2]/h1/p').text
62
63                  urllib.request.urlretrieve(img_url, 'img.jpg')
64                  img = Image.open('img.jpg')
```

```
65                  rgb_color = self._dominant_color(img)
66                  name_color = self._convert_rgb_to_names(rgb_color)
67                  bagInfo['rgb_dominant_color'] = list(rgb_color)
68                  bagInfo['name_dominant_color'] = name_color
69
70                  json_df = json_df.append(bagInfo, ignore_index=True)
71                  continue
72
73              self._close_newsletter()
74              self._scroll_down_page()
75              img_url = self.driver.find_element(By.XPATH,
                ↪  config['BAG_IMG_URL']).get_attribute('src')
76              bagInfo['img'] = img_url
77              bagInfo['brand'] = self.driver.find_element(By.XPATH,
                ↪  config['BAG_BRAND']).text.upper()
78              bagInfo['title'] = self.driver.find_element(By.XPATH,
                ↪  config['BAG_TITLE']).text
79
80              bagInfo['price'], bagInfo['currency'], bagInfo['original_price'], \
81              sale_perc, bagInfo['cluster_price'] = self._get_prices_features()
82              bagInfo['discount_percentage'] = sale_perc
83              breadcrumb = self.driver.find_elements(By.XPATH, config['BAG_TYPE'])
84              bagInfo['type'], bagInfo['subtype'] =
                ↪  breadcrumb[-2].find_element(By.TAG_NAME, 'a').text, \
85                                                  breadcrumb[-1].find_element(By.
                                                    ↪  TAG_NAME,
                                                    ↪  'a').text
86
87              bagInfo['description'] = self._get_bag_desc()
88              bagInfo['status'] = self._get_bag_status(sale_perc != 0)
89              bagInfo['id_farfetch'] = self.driver.find_elements(By.XPATH,
                ↪  config['BAG_NUMBER'])[-1] \
90                  .find_element(By.XPATH, './/p[1]/span').text
91              bagInfo['id_brand'] = self.driver.find_elements(By.XPATH,
                ↪  config['BAG_NUMBER'])[-1] \
92                  .find_element(By.XPATH, './/p[2]/span').text
93              bagInfo['materials'] = self._get_bag_materials()
94              bagInfo['info_materials'] = self._get_bag_info_mat()
95              bagInfo['info_size'], size_metric =
                ↪  self._get_bag_info_size(size_metric)
96
97              urllib.request.urlretrieve(img_url, 'img.jpg')
98              img = Image.open('img.jpg')
99              rgb_color = self._dominant_color(img)
100             name_color = self._convert_rgb_to_names(rgb_color)
101             bagInfo['rgb_dominant_color'] = list(rgb_color)
102             bagInfo['name_dominant_color'] = name_color
103
104             json_df = json_df.append(bagInfo, ignore_index=True)
105             time.sleep(3)
106
```

```
    except BaseException as e:
        print(e)

    finally:
        json_df.fillna('', inplace=True)
        return json_df
```

A.1.2. Net-a-poter

```
def index_pages(self):
    df = pd.DataFrame(columns=['bagLink', 'regionCode', 'gender', 'brand'])
    gender = 'women'
    brands = config['BRAND_LIST']
    try:
        for brand in brands:
            link = config['BASE_URL_NETAPORTER'] + self.region + '/' +
            ↪  config['MID_URL'] \
                   + brand + '/' + config['FOOTER_URL']
            self.driver.get(link)
            self._accept_cookies()
            self._close_newsletter()
            if self.region != "it":
                self._close_shipping_button()
            self._scroll_down_page()
            n_page = self._get_n_pages()
            logging.info("n_page " + str(n_page))
            current_page = 1
            while current_page <= int(n_page):
                if current_page != 1:
                    self.driver.get(link + config['PARAMETER_PAGE_NUMBER'] +
                    ↪  str(current_page))
                    self._scroll_down_page()
                bags_a = self.driver.find_elements(By.XPATH, config['BAG_LIST'])
                for bg in bags_a:
                    bag_link = bg.get_attribute('href')
                    # print(bag_link)
                    df.loc[len(df)] = [bag_link, self.region, gender, brand]
                current_page += 1

    except BaseException as e:
        logging.info(e)
    finally:
        if os.path.isfile(config['BAGS_LINKS_PATH'] + "_" + gender + "_" +
        ↪  args.region_code + ".csv"):
            df.to_csv('/csv/backup_ex.csv', index=False)
            df1 = pd.read_csv(
                filepath_or_buffer=config['BAGS_LINKS_PATH'] + "_" + gender + "_" +
                ↪  args.region_code + ".csv")
            df = pd.concat([df1, df])
        df.to_csv(config['BAGS_LINKS_PATH'] + "_" + gender + "_" +
        ↪  args.region_code + ".csv", index=False)
```

```
        logging.info("df.shape " + str(df.shape))
```

```
def bags_scrape(self):
    gender = 'women'
    csv_df = pd.read_csv(
        filepath_or_buffer=config['BAGS_LINKS_PATH'] + "_" + gender + "_" +
        ↪  args.region_code + ".csv")
    df_filtered = csv_df[(csv_df['regionCode'] == self.region)]
    json_df = pd.DataFrame(columns=['gender', 'region', 'img',
    ↪  'rgb_dominant_color', 'name_dominant_color',
                                    'url', 'date_time', 'id_product', 'brand',
                                    ↪  'title', 'type', 'subtype',
                                    'currency', 'price', 'original_price',
                                    ↪  'discount_percentage',
                                    'cluster_price', 'description', 'status',
                                    ↪  'materials',
                                    'details&care', 'info_size'])

    # get json file
    json_name = self.region + '_' + gender
    json_path = [pos_json for pos_json in os.listdir(config['JSON_PATH_FOLDER'])
                 if pos_json.endswith('.json') and json_name in
                 ↪  Path(pos_json).stem]
    logging.info(json_path)

    # resume scraping if json file exists
    if len(json_path) != 0:
        with open(config['JSON_PATH_FOLDER'] + json_path[0], encoding='utf-8') as
        ↪  file:
            json_file = json.load(file)
            json_df = pd.DataFrame.from_dict(json_file)
            df_filtered = df_filtered.tail(df_filtered.shape[0] -
            ↪  json_df.shape[0])
        if not df_filtered.empty:
            os.remove(config['JSON_PATH_FOLDER'] + json_path[0])

    logging.info("df_filtered.head(), df_filtered.shape: " +
    ↪  str(df_filtered.head()) + " " + str(df_filtered.shape))
    self.get_bag_info(json_df, df_filtered)


def get_bag_info(self, json_df: pd.DataFrame, csv_df: pd.DataFrame):
    flag = True
    for row in tqdm(csv_df.to_dict('records')):
        try:
            logging.info(row)
            self.driver.get(row['bagLink'])
            if config['LOGIN_PAGE'] in self.driver.current_url:
                logging.info('Login page. Bag skipped: ' + row['bagLink'])
                print('Login page. Bag skipped: ', row['bagLink'])
                continue
```

```
41              if flag:
42                  self._accept_cookies()
43                  self._close_newsletter()
44                  if self.region != "it":
45                      self._close_shipping_button()
46                  flag = False
47              time.sleep(2)
48
49              bagInfo = self.make_dict(row)
50
51          # Timeout-retry handling
52          except TimeoutException:
53              print('Timeout exception occured!', row['bagLink'])
54              self.driver.close()
55
56              self.driver = webdriver.Chrome(service=Service(ChromeDriverManager().
                ↪  install()),
                ↪  options=options)
57              self._implicit_wait(10)
58              flag = True
59              self.driver.get(row['bagLink'])
60              if config['LOGIN_PAGE'] in self.driver.current_url:
61                  logging.info('Login page. Bag skipped: ' + row['bagLink'])
62                  print('Login page. Bag skipped: ', row['bagLink'])
63                  continue
64              if flag:
65                  self._accept_cookies()
66                  self._close_newsletter()
67                  if self.region != "it":
68                      self._close_shipping_button()
69                  flag = False
70              time.sleep(2)
71
72              bagInfo = self.make_dict(row)
73
74          json_df = json_df.append(bagInfo, ignore_index=True)
75
76          # json creation
77          json_dict = json_df.to_dict(orient='records')
78          logging.info(len(json_dict))
79          with open(config['JSON_PATH_FOLDER'] + self.region + '_' + row['gender']
            ↪  + '_' +
80                    datetime.now().strftime("%Y-%m-%d") + '.json', "w",
                      ↪  encoding='utf-8') as outfile:
81              json.dump(json_dict, outfile, indent=4, default=lambda o: '<not
                ↪  serializable>', ensure_ascii=False)
```

A.1.3. Balenciaga

```
1   def index_pages(self):
2       if not os.path.isfile(
```

```
        config['BAGS_LINKS_PATH'] + "_" + args.gender + "_" + args.region_code +
        ↪  "_" + args.starting_date + ".csv"):

        df = pd.DataFrame(columns=['bagLink', 'imgUrl', 'regionCode', 'gender'])
        links_to_scrape = self.get_region_link(self.region)
        if args.gender == 'men':
            base_link = links_to_scrape[0]
        else:
            base_link = links_to_scrape[1]
        try:
            self.driver.get(base_link)
            print(base_link)
            self._accept_cookies()
            self._change_region()
            n_results = self._get_n_results()
            print(n_results)
            while not self._check_exists_by_xpath(

                    ↪  '//*[@id="product-search-results"]/div[2]/ul/li[contains(@data-index,
                    ↪  ' + n_results + ')]'):
                self._scroll_down_page()

            bags_elements = self.driver.find_elements(By.XPATH,
            ↪  config['BAG_LIST'])
            for c, bag in enumerate(bags_elements):
                if self._check_editorial(bag, 'class'):
                    continue
                bag_a = bag.find_element(By.TAG_NAME, 'a')
                bag_link = bag_a.get_attribute('href')
                img_url = self._get_img_url(bag_a, config['BAG_LIST'] +
                                            "[" + str(c + 1) + "]")
                df.loc[len(df)] = [bag_link, img_url, self.region, args.gender]

        except BaseException as e:
            logging.info(e)
        finally:
            if
            ↪  os.path.isfile(config['BAGS_LINKS_PATH']+"_"+args.gender+"_"+args.region_code+"_"+args.s
                df.to_csv('/csv/backup_ex.csv', index=False)
                df1 = pd.read_csv(
                    filepath_or_buffer=config['BAGS_LINKS_PATH'] + "_" +
                    ↪  args.gender + "_" + args.region_code + ".csv")
                df = pd.concat([df1, df])

            ↪  df.to_csv(config['BAGS_LINKS_PATH']+"_"+args.gender+"_"+args.region_code+"_"+args.starti
            ↪  index=False)
            logging.info("df.shape " + str(df.shape))
```

```
def bags_scrape(self, gender):
    csv_df = pd.read_csv(
```

```
3           filepath_or_buffer=config['BAGS_LINKS_PATH']+"_"+args.gender+"_"+args.reg
            ↪ ion_code+"_"+args.starting_date+".csv")
4       #logging.info(csv_df)
5       df_filtered = csv_df[(csv_df['regionCode'] == self.region) &
        ↪ (csv_df['gender'] == gender)]
6       #logging.info(df_filtered)
7       json_df = pd.DataFrame(columns=['gender', 'region', 'img',
        ↪ 'rgb_dominant_color', 'name_dominant_color',
8                                       'url', 'date_time', 'starting_date',
                                        ↪ 'id_product', 'brand', 'title', 'type',
                                        ↪ 'subtype',
9                                       'currency', 'price', 'cluster_price',
                                        ↪ 'description', 'status',
10                                      'materials', 'info_materials', 'info_size'])
11
12      # get json file
13      json_name = self.region + '_' + gender + '_' + args.starting_date
14      json_path = [pos_json for pos_json in os.listdir(config['JSON_PATH_FOLDER'])
15                   if pos_json.endswith('.json') and json_name in
                     ↪ Path(pos_json).stem]
16      logging.info(json_path)
17
18      # resume scraping if json file exists
19      if len(json_path) != 0:
20          with open(config['JSON_PATH_FOLDER'] + json_path[0], encoding='utf-8') as
            ↪ file:
21              json_file = json.load(file)
22              json_df = pd.DataFrame.from_dict(json_file)
23              df_filtered = df_filtered.tail(df_filtered.shape[0] -
                ↪ json_df.shape[0])
24          if not df_filtered.empty:
25              os.remove(config['JSON_PATH_FOLDER'] + json_path[0])
26
27      logging.info("df_filtered.head(), df_filtered.shape: " +
        ↪ str(df_filtered.head()) + " " + str(df_filtered.shape))
28      self.get_bag_info(json_df, df_filtered)
29
30  def get_bag_info(self, json_df: pd.DataFrame, csv_df: pd.DataFrame):
31      try:
32          global time_limit_reached
33          first_time = True
34          for row in tqdm(csv_df.to_dict('records')):
35              if datetime.now() - starting_time < time_limit:
36                  logging.info(row)
37                  bagInfo: dict = {}
38                  bag_url = row['bagLink']
39                  img_url = row['imgUrl']
40                  bagInfo['gender'] = row['gender']
41                  bagInfo['region'] = row['regionCode']
42                  bagInfo['url'] = bag_url
43                  bagInfo['img'] = img_url
```

```
44                  bagInfo['date_time'] = datetime.now().strftime("%Y-%m-%d
                    ↪  %H:%M:%S")
45                  logging.info("bag_url " + str(bag_url))
46
47                  self.driver.get(bag_url)
48                  if first_time:
49                      if self.region != 'it':
50                          self._change_region()
51                      self._accept_cookies()
52                      first_time = False
53                  time.sleep(2)
54                  #self._scroll_down_page()
55                  bagInfo['brand'] = 'BALENCIAGA'
56                  bagInfo['title'] = self.driver.find_element(By.XPATH,
                    ↪  config['BAG_TITLE']).text.capitalize()
57                  bagInfo['price'], bagInfo['currency'], bagInfo['cluster_price'] =
                    ↪  self._get_prices_features()
58                  bagInfo['description'] = self.driver.find_element(By.XPATH,
                    ↪  config['BAG_DESC']).text
59                  breadcrumb = self.driver.find_elements(By.XPATH, config['BC_BAG'])
60                  tp, subtp = '', ''
61                  if len(breadcrumb) > 1:
62                      tp, subtp = breadcrumb[1].find_element(By.TAG_NAME,
                        ↪  'a').text.capitalize(), \
63                          breadcrumb[2].find_element(By.TAG_NAME,
                            ↪  'a').text.capitalize()
64                  bagInfo['type'], bagInfo['subtype'] = tp, subtp
65                  bagInfo['status'] = self._get_bag_status()
66                  bagInfo['info_size'], bagInfo['info_materials'] =
                    ↪  self._get_detail_info()
67                  bagInfo['materials'] = self._get_bag_materials()
68                  bagInfo['id_product'] = self.driver.find_element(By.XPATH,
                    ↪  config['BAG_NUMBER']).text
69
70                  img_name = 'img_' + self.region + '_' + row['gender'] + '.jpg'
71                  urllib.request.urlretrieve(img_url, img_name)
72                  img = Image.open(img_name)
73                  rgb_color = self._dominant_color(img)
74                  name_color = self._convert_rgb_to_names(rgb_color)
75                  os.remove(img_name)
76                  bagInfo['rgb_dominant_color'] = list(rgb_color)
77                  bagInfo['name_dominant_color'] = name_color
78                  bagInfo['starting_date'] = args.starting_date
79
80                  json_df = json_df.append(bagInfo, ignore_index=True)
81
82                  # json creation
83                  json_dict = json_df.to_dict(orient='records')
84                  logging.info(len(json_dict))
85                  with open(config['JSON_PATH_FOLDER'] + self.region + '_' +
                    ↪  args.gender + '_' +
```

```
86                      args.starting_date + '.json', "w", encoding='utf-8') as
                        ↪  outfile:
87                  json.dump(json_dict, outfile, indent=4, default=lambda o:
                    ↪  '<not serializable>', ensure_ascii=False)
88              else:
89                  time_limit_reached = True
90                  logging.info('Time limit reached!')
91                  break
92      except BaseException as e:
93          print(e)
94
95  def cn_scrape(self, gender):
96      links_to_scrape = self.get_region_link(self.region)
97      base_link = [l for l in links_to_scrape if gender in l][0]
98      try:
99          global time_limit_reached
100         json_df = pd.DataFrame(columns=['gender', 'region', 'img',
            ↪  'rgb_dominant_color', 'name_dominant_color',
101                                         'url', 'date_time', 'starting_date',
                                            ↪  'id_product', 'brand', 'title',
                                            ↪  'type', 'subtype',
102                                         'currency', 'price', 'cluster_price',
                                            ↪  'description', 'status',
103                                         'materials', 'info_materials',
                                            ↪  'info_size'])
104
105         self.driver.get(base_link)
106         logging.info(base_link)
107         time.sleep(3)
108         self._accept_cn_cookies()
109         n_results = re.findall(r'\d+', self.driver.find_element(By.XPATH,
            ↪  config['N_RESULTS_CN']).text)[0]
110         logging.info("Bags found: " + str(n_results))
111
112         bags_elements = []
113         while len(bags_elements) < int(n_results):
114             self._scroll_down_page()
115             bags_elements = self.driver.find_elements(By.XPATH,
                ↪  config['BAG_CN_LIST'])
116             logging.info("Bags indexed: " + str(len(bags_elements)))
117
118         for c in tqdm(range(int(n_results))):
119             if datetime.now() - starting_time < time_limit_cn:
120                 act_url = self.driver.current_url
121                 bags_elements = self.driver.find_elements(By.XPATH,
                    ↪  config['BAG_CN_LIST'])
122                 bag = bags_elements[c]
123
124                 img_url = bag.find_element(By.XPATH, config['BAG_CN_LIST'] +
125                                            "[" + str(c + 1) + "]" + config['BAG_C
                                            ↪  N_IMG']).get_attribute('src')
```

```
126                 img_url = img_url.replace('Medium', 'Large')
127
128                 self.driver.execute_script("arguments[0].scrollIntoView(true);",
                    ↪  bag)
129                 bag.click()
130                 while act_url == self.driver.current_url:
131                     continue
132
133                 # self._scroll_down_page()
134                 setBag = self.bag_scrape_cn([gender, self.region, img_url,
                    ↪  self.driver.current_url])
135                 json_df = json_df.append(setBag, ignore_index=True)
136
137                 json_dict = json_df.to_dict(orient='records')
138                 logging.info(len(json_dict))
139                 with open(config['JSON_PATH_FOLDER'] + self.region + '_' +
                    ↪  args.gender + '_' +
140                       args.starting_date + '.json', "w", encoding='utf-8') as
                          ↪  outfile:
141                     json.dump(json_dict, outfile, indent=4, default=lambda o:
                        ↪  '<not serializable>', ensure_ascii=False)
142
143                 time.sleep(2)
144                 self.driver.get(act_url)
145             else:
146                 time_limit_reached = True
147                 logging.info('Time limit reached!')
148                 break
149
150     except BaseException as e:
151         logging.info(e)
152
153 def bag_scrape_cn(self, some_info: list):
154     bagInfo = {}
155     try:
156         logging.info("bag_url " + str(some_info[3]))
157         bagInfo = {'gender': some_info[0], 'region': some_info[1]}
158         img_url = some_info[2]
159         bagInfo['img'] = img_url
160         bagInfo['url'] = some_info[3]
161         bagInfo['brand'] = 'BALENCIAGA'
162         bagInfo['date_time'] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
163         bagInfo['title'] = self.driver.find_element(By.XPATH,
            ↪  config['BAG_TITLE_CN']).text.capitalize()
164         bagInfo['description'] = self.driver.find_element(By.XPATH,
            ↪  config['BAG_DESC_CN']).text
165         bagInfo['type'] = 'Bags'
166         bagInfo['price'], bagInfo['currency'], bagInfo['cluster_price'] =
            ↪  self._get_prices_features()
167         bagInfo['info_size'], bagInfo['info_materials'] = self._get_detail_info()
168         bagInfo['subtype'] = bagInfo['info_materials'][1]
```

```
169         bagInfo['id_product'] = self.driver.find_element(By.CLASS_NAME,
            ↪  'products_descWrapper__2QuZs') \
170             .find_elements(By.TAG_NAME, 'div')[2].text.split('', 1)[-1]
171         bagInfo['materials'] = self._get_bag_materials()
172         bagInfo['status'] = self._get_bag_status()
173 
174         img_name = 'img_' + self.region + '_' + some_info[0] + '.jpg'
175         urllib.request.urlretrieve(img_url, img_name)
176         img = Image.open(img_name)
177         rgb_color = self._dominant_color(img)
178         name_color = self._convert_rgb_to_names(rgb_color)
179         os.remove(img_name)
180         bagInfo['rgb_dominant_color'] = list(rgb_color)
181         bagInfo['name_dominant_color'] = name_color
182         bagInfo['starting_date'] = args.starting_date
183 
184     except BaseException as e:
185         logging.info(e)
186     finally:
187         return bagInfo
```

A.1.4. Mytheresa

```
1  def index_pages(self, csv_path, starting_page=1):
2      href_list = list()
3      for p in range(starting_page, 100):
4          logging.info("Page " + str(p))
5          try:
6              self.remove_newsletter()
7          except:
8              pass
9          try:
10             self.remove_shoeclub_invite()
11         except:
12             pass
13 
14         items = self.driver.find_elements(By.CLASS_NAME, "item")
15         for item in items:
16             a = item.find_element(By.CSS_SELECTOR, "a")
17             href_list.append(a.get_attribute("href"))
18         try:
19             nextPage = self.driver.find_element(By.CLASS_NAME, "next")
20             nextPageArrow =
               ↪  wait.until(EC.element_to_be_clickable(nextPage.find_element(By.XPATH,
               ↪  ".//a/span[2]")))
21             nextPageArrow.click()
22         except Exception as e:
23             # self.driver.get_screenshot_as_file("screen.png")
24             logging.info("No next page")
25             break
26 
```

```
    with open(csv_path, 'a', newline='') as file:
        mywriter = csv.writer(file, delimiter=',')
        for href in href_list:
            mywriter.writerow([href])
```

```
def get_elem_info(self, json_path, item_link, row_count):
    logging.info(item_link)
    html_path = os.path.join(output_date_folder, "html")
    logging.info(html_path)

    self.driver.get("file://" + html_path + "/" + region + "/" + gender + "/" +
    ↪  str(row_count) + ".html")
    error_detected = False
    recovered_from_error_detected = False
    try:
        self.remove_newsletter()
    except:
        pass
    try:
        self.remove_shoeclub_invite()
    except:
        pass

    # BRAND, TITLE, NUMBER, PRICE, DESCRIPTION
    info_dict = {}
    info_dict["gender"] = gender

    info_dict["region"] = region
    elements = ["NUMBER", "BRAND", "TITLE", "TYPE", "SUBTYPE", "PRICE",
    ↪  "DESCRIPTION"]
    # to get the image
    try:
        # NEED TO GET IMAGE URL
        image = self.driver.find_element(By.ID, config['IMG_ID_' +
        ↪  current_website.upper()])
        info_dict["img"] = image.get_attribute("src")
        info_dict["img"] = info_dict["img"].replace("file", "https", 1)
        # to get the dominant color
        # response = requests.get(info_dict["img"])
        # img = Image.open(BytesIO(response.content))
        # urllib.request.urlretrieve(image.get_attribute("src"))
        # img = Image.open("img.png") # open RGB image
        image_path = os.path.join(output_date_folder, "image")
        img = Image.open(image_path + "/" + region + "/" + gender + "/" +
        ↪  str(row_count) + ".jpg")
        rgb_color = self.dominant_color(img)
        info_dict["rgb_dominant_color"] = rgb_color
        name_color = self.convert_rgb_to_names(rgb_color)
        info_dict["name_dominant_color"] = name_color
    except Exception as e:
        # info_dict["img"] = "<not found>"
```

```
43          # info_dict["rgb_dominant_color"] = "<not found>"
44          # info_dict["name_dominant_color"] = "<not found>"
45          logging.info(gender.upper() + "/" + region.lower() + " " + "Can't find
            ↪  image. Current URL: " + self.driver.current_url)
46          #logging.info(e)
47          #self.driver.get_screenshot_as_file(gender.upper() + "-" + region.lower()
            ↪  + "-" + str(datetime.now()) + ".png")
48          error_detected = True
49      if error_detected:
50          logging.info("Can't find image. Trying to find other elements...")
51          try:
52              name_var = wait.until(EC.element_to_be_clickable((By.XPATH,
                ↪  config['NUMBER_XPATH_' % el + current_website.upper()]))).text
53              logging.info("Found other elements. Gathering info as normal...")
54              recovered_from_error_detected = True
55          except:
56              logging.info("Could not find other elements. Product skipped.")
57      if not error_detected or recovered_from_error_detected:
58          # to take the current url
59          current_url = self.driver.current_url
60          info_dict["url"] = item_link
61          # to get the current date and time
62          data_time = datetime.now()
63          info_dict["date_time"] = str(data_time)
64          for el in elements:
65              # try:
66              #     self.implicit_wait()
67              element_not_present = ""
68              try:
69                  if (region == "en-cn" and (el.lower() == "description" or
                    ↪  el.lower()=="number")):
70                      name_var = wait.until(EC.element_to_be_clickable((By.XPATH,
                        ↪  config['%s_EN-CN_XPATH_' % el +
                        ↪  current_website.upper()]))).text
71                  else:
72                      name_var = wait.until(EC.element_to_be_clickable((By.XPATH,
                        ↪  config['%s_XPATH_' % el + current_website.upper()]))).text
73                  logging.info(gender.upper() + "/" + region.lower() + " " +
                    ↪  "name_var-element: " + str(name_var))
74              except Exception as e:
75                  name_var = ""
76                  logging.info(gender.upper() + "/" + region.lower() + " " + "No "
                    ↪  + str(el.lower()))
77                  element_not_present = el.lower()
78                  #self.driver.get_screenshot_as_file("error.png")
79              if (el.lower() != element_not_present):
80                  if ("number" in el.lower()):
81                      number_list = name_var.split()
82                      info_dict[el.lower()] = number_list[-1]
83                  elif ("price" in el.lower()):
84                      try:  # successfull if sale
```

```
85                      name_var = wait.until(EC.element_to_be_clickable(
86                          (By.XPATH, config['NEW_PRICE_XPATH_' +
                            ↪  current_website.upper()]))).text
87                      original_price = wait.until(EC.element_to_be_clickable(
88                          (By.XPATH, config['OLD_PRICE_XPATH_' +
                            ↪  current_website.upper()]))).text
89                      original_price = original_price.replace(".", "")
90                      original_price = original_price.replace(",", "")
91                      original_price = float(re.findall(r'\d+',
                        ↪  original_price)[0])
92                      sales = True
93                  except:  # no sale
94                      sales = False
95                      logging.info("No sale")
96                  name_var = name_var.replace(".", "")
97                  name_var = name_var.replace(",", "")
98                  info_dict["currency"] = name_var[0]
99                  name_var = float(re.findall(r'\d+', name_var)[0])
100                 info_dict[el.lower()] = name_var
101                 if not sales:
102                     original_price = name_var
103                 info_dict["original_price"] = original_price
104                 info_dict["discount_percentage"] = round(100 - (name_var *
                    ↪  100 / original_price))
105             else:
106                 info_dict[el.lower()] = name_var
107             # except TimeoutException:
108             #     logging.info("TimeoutException handled")
109             #     continue
110
111     try:
112         info_dict["status"] = wait.until(
113             EC.element_to_be_clickable((By.XPATH, config['STATUS_XPATH_' +
                ↪  current_website.upper()]))).text
114     except:
115         if sales:
116             info_dict["status"] = "ON SALE"
117         else:
118             info_dict["status"] = "REGULAR"
119
120     # scroll down the webpage
121     self.driver.execute_script("window.scrollTo(0, 500)")
122
123     try:
124         # MATERIALS
125         if region == "en-cn":
126             el_for_scroll = self.driver.find_element(By.XPATH,
                ↪  config['MATERIALS_EN-CN_XPATH_' + current_website.upper()])
127         else:
128             el_for_scroll = self.driver.find_element(By.XPATH,
                ↪  config['MATERIALS_XPATH_' + current_website.upper()])
```

```
129             # action chain object creation
130             action = ActionChains(self.driver)
131             action.move_to_element(el_for_scroll).perform()
132         except:
133             logging.info("No scroll to do")
134         try:
135             if region == "en-cn":
136                 # materials_element = wait.until(EC.element_to_be_clickable('//*
                    ↪  [@id="mCSB_7_container"]/div/ul'))
137                 materials_element = self.driver.find_element(By.XPATH,config['MA
                    ↪  TERIALS_EN-CN_XPATH_' +
                    ↪  current_website.upper()])
138             else:
139                 materials_element =
                    ↪  self.driver.find_element(By.XPATH,config['MATERIALS_XPATH_'
                    ↪  + current_website.upper()])
140             self.driver.execute_script("arguments[0].click();", materials_element)
141             # materials_element =
                ↪  wait.until(EC.element_to_be_clickable((By.XPATH,
                ↪  config['MATERIALS_XPATH_'  + current_website.upper()])))
142             materials = materials_element.text
143             materials = materials.split('\n')
144             tuple = []
145             l = []
146             # to create a dictionary
147             for mat in materials:
148                 check = ":"
149                 if (check in mat):
150                     tuple.append(mat.split(":"))
151                 else:
152                     l.append(mat)
153             description = dict((x, y) for x, y in tuple)
154             info_dict["materials"] = description
155             info_dict["info_materials"] = l
156         except:
157             logging.info("No materials")
158
159         # scroll down the webpage
160         self.driver.execute_script("window.scrollTo(0, 500)")
161
162         # SIZE&FIT
163         # self.implicit_wait()
164         try:
165             if region == "en-cn":
166                 sizeandfit = wait.until(EC.element_to_be_clickable((By.XPATH,
                    ↪  config['SIZE&FIT_EN-CN_XPATH_' + current_website.upper()])))
167                 self.driver.execute_script("arguments[0].click();", sizeandfit)
168                 sizeandfit = wait.until(EC.element_to_be_clickable((By.XPATH,
                    ↪  config['SIZE&FITTEXT_EN-CN_XPATH_' +
                    ↪  current_website.upper()]))).text
169             else:
```

```
170                 sizeandfit = wait.until(EC.element_to_be_clickable((By.XPATH,
                    ↪  config['SIZE&FIT_XPATH_' + current_website.upper()])))
171                 self.driver.execute_script("arguments[0].click();", sizeandfit)
172                 sizeandfit = wait.until(EC.element_to_be_clickable((By.XPATH,
                    ↪  config['SIZE&FITTEXT_XPATH_' +
                    ↪  current_website.upper()]))).text
173
174             # to distinguish the categories
175             value1 = "Borse"
176             tuple1 = []
177             l = []
178             # size&fit BORSE
179             if value1 in info_dict.values():
180                 temp = sizeandfit.split('\n')
181                 for ele in temp:
182                     try:
183                         # to take only the number without cm
184                         name_val = ele.rsplit(" ", 1)
185                         name = name_val[0]
186                         val = name_val[1]
187                         number = val.replace(",", ".")
188                         val = float(re.findall(r'\d+(?:\.\d+)?', number)[0])
189                         tuple1.append([name, val])
190                     except:
191                         l.append(ele)
192                 # to create a dictionary
193                 description = dict((x, y) for x, y in tuple1)
194                 height = "Altezza"
195                 height1 = "Altezza massima"
196                 width = "Larghezza"
197                 width1 = "Larghezza massima"
198                 depth = "Profondità"
199                 depth1 = "Profondità massima"
200
201             # size&fit other items
202             else:
203                 temp = sizeandfit.split('\n')
204                 # to create a dictionary
205                 for t in temp:
206                     check = ":"
207                     if (check in t):
208                         tuple1.append(t.split(":"))
209                     else:
210                         l.append(t)
211
212                 # to create a dictionary
213                 description = dict((x, y) for x, y in tuple1)
214
215             info_dict["size&fit"] = description
216             info_dict["info_size&fit"] = l
217         except:
```

```
218             logging.info("No size and fit")
219
220         with open(json_path, "a", encoding="UTF-8") as outfile:
221             # Serializing json
222             json_object = json.dumps(info_dict, default=lambda o: '<not
                ↪  serializable>', ensure_ascii=False)
223             outfile.write(str(json_object + "\n"))
```

A.2. Preprocessing e creazione dataset

In questa sezione vengono presentate le principali funzioni a supporto della terza e ultima fase, descritta nella sezione 3.2, riguardante il preprocessing dei dati raccolti, la generazione del dataset e le relative annotazioni, successivamente utilizzate per l'addestramento dei modelli selezionati.

```
1  def get_all_bags(brand: str):
2      path_to_json = '../data/outputscraping'
3
4      # get all json file in directory and subdirectory
5      json_files = []
6      for path, subdirs, files in os.walk(path_to_json):
7          for name in files:
8              if name.endswith('.json'):
9                  json_files.append(os.path.join(path, name))
10     print("Total files: ", len(json_files))
11
12     # filter file per brand
13     json_files_f = [file for file in json_files if brand in file]
14     print("Filtered files: ", len(json_files_f))
15
16     final_json_df, cn_df = None, None
17     for n, file in enumerate(json_files_f):
18         with open(file, encoding='utf-8') as f:
19             print(f.name)
20             try:
21                 json_file = json.load(f)
22                 json_df = pd.DataFrame.from_dict(json_file)
23                 json_df = json_df[json_df['type'] != ""]
24                 if not (brand == 'balenciaga' and 'cn' in f.name):
25                     if final_json_df is None:
26                         final_json_df = json_df
27                     else:
28                         final_json_df = pd.concat([json_df, final_json_df])
29                 else:
30                     if cn_df is None:
31                         cn_df = json_df
32                     else:
33                         cn_df = pd.concat([json_df, cn_df])
34             except json.decoder.JSONDecodeError:
```

```
35                  print("FILE BAD FORMATTED: ", f.name)
36                  continue
37
38      print("Total bags: ", final_json_df.shape[0] + (cn_df.shape[0] if cn_df is
        ↪  not None else 0))
39      final_json_df = final_json_df.dropna()
40      final_json_df = final_json_df.drop_duplicates('img')
41
42      if cn_df is not None:
43          cn_df = cn_df.drop_duplicates('img')
44          for row in cn_df.to_dict('records'):
45              img = row['img'].split('/')[-1].split('.')[0] + '.jpg'
46              app = final_json_df[final_json_df['img'].str.contains(img)]
47              if not app.shape[0] != 0:
48                  final_json_df = final_json_df.append(row, ignore_index=True)
49
50      # Filtering and trasforming bags
51      final_json_df['type'] = final_json_df['type'].apply(lambda x: x.lower() if
        ↪  type(x) == str else x)
52      final_json_df['subtype'] = final_json_df['subtype'].apply(lambda x: x.lower()
        ↪  if type(x) == str else x)
53      if brand == 'mytheresa':
54          final_json_df = final_json_df[final_json_df['brand'].isin(["SAINT
            ↪  LAURENT", "GUCCI", "BALENCIAGA"])]
55          final_json_df = final_json_df[final_json_df['type'] == 'bags']
56      final_json_df['type'] = final_json_df['type'].apply(lambda x: 'bags')
57      final_json_df['subtype'] = final_json_df['subtype'].apply(lambda x:
        ↪  transform_type(brand, x))
58
59      print("Unique bags: ", final_json_df.shape[0])
60
61      with open(json_path + brand + '_bags.json', "w", encoding='utf-8') as outfile:
62          '''if brand == 'mytheresa':
63              grouped = final_json_df.groupby(['brand']).size().to_frame('count')
64              print(grouped['count'].sum())
65              grouped.to_csv(brand + '_brand.csv')'''
66          final_json_df.to_json(outfile, orient='records', force_ascii=False,
            ↪  indent=4)
67
68
69  def get_labels_brand(brand: str):
70      translator = Translator()
71
72      with open(json_path + brand + '_bags.json', encoding='utf-8') as file:
73          json_file = json.load(file)
74          final_json_df = pd.DataFrame.from_dict(json_file)
75
76      print(final_json_df.shape[0])
77      grouped = final_json_df.groupby(['type', 'subtype']).size().to_frame('count')
78      print(grouped['count'].sum())
79
```

```
    trans = []
    for row in grouped.iterrows():
        trans.append(translator.translate(row[0][1]).text.lower())

    grouped['transl'] = trans
    grouped.to_csv(csv_path + brand + '_labels.csv')


def group_bags():
    final_json_df = None
    for brand in brands:
        with open(json_path + brand + '_bags.json', encoding='utf-8') as file:
            json_file = json.load(file)
            json_df = pd.DataFrame.from_dict(json_file)
            if final_json_df is not None:
                final_json_df = pd.concat([json_df, final_json_df])
            else:
                final_json_df = json_df

    print(final_json_df.shape[0])
    grouped = final_json_df.groupby(['type', 'subtype']).size().to_frame('count')
    cats = grouped.sort_values(by=['count'],
    ↪  ascending=False).iloc[:9].index.get_level_values('subtype').tolist()
    print(cats)
    final_json_df = final_json_df[final_json_df['subtype'].isin(cats)]
    print(final_json_df.shape[0])
    with open('all_bags.json', "w", encoding='utf-8') as outfile:
        final_json_df.to_json(outfile, orient='records', force_ascii=False,
        ↪  indent=4)

    grouped = final_json_df.groupby(['type', 'subtype']).size().to_frame('count')
    print(grouped['count'].sum())
    grouped.to_csv('all_category.csv')
```

```
def img_create_and_coco_ann(df: pd.DataFrame, cat: list, set_type: str, id_count:
↪  int = 0):
    print(f'{set_type.upper()}-SET elaboration...')

    data_root = '../detectors/dataset/'

    driver = selenium_setup()

    images = []
    ann = []
    for n, item in enumerate(tqdm(list(df.to_dict('records')))):
        imD = {}
        annD = {}
        # idx = item['img_id']
        idx = n + id_count

        img_name = 'image' + str(idx) + '.jpg'
```

```
17
18          bbox, area, width, height = set_bounding_box(item['img'], data_root +
            ↪  set_type + '/' + img_name,
19                                                       'net-a-porter' in
                                                         ↪  item['url'], driver)
20          if bbox == 0:
21              continue
22
23          imD['id'] = idx
24          imD['file_name'] = img_name
25          imD['width'] = width
26          imD['height'] = height
27          imD['license']: 1
28
29          c = [d for d in cat if d['name'] == item['subtype']][0]
30          annD['id'] = idx
31          annD['image_id'] = idx
32          annD['category_id'] = c['id']
33          annD['bbox'] = bbox
34          annD['area'] = area
35          annD['iscrowd'] = 0
36
37          images.append(imD)
38          ann.append(annD)
39
40      # print(images, ann)
41
42      coco_dict = {
43          "info": {
44              "year": 2023,
45              "version": "1.0",
46              "description": "Annotations for object detection dataset"
47          },
48          "licenses": [
49              {
50                  "id": 1,
51                  "name": "License 1"
52              }
53          ],
54          "images": images,
55          "annotations": ann,
56          "categories": cat
57      }
58      with open(data_root + 'annotations/' + set_type + '_annotations.json', "w",
        ↪  encoding='utf-8') as outfile:
59          json.dump(coco_dict, outfile, indent=4, default=lambda o: '<not
            ↪  serializable>', ensure_ascii=False)
60
61      driver.quit()
62
63  def set_bounding_box(url: str, img_name: str, flag: bool, driver):
```

```
64      try:
65          # selenium_screen(url, img_name, driver)
66          if flag:
67              selenium_screen(url, img_name, driver)
68          else:
69              urllib.request.urlretrieve(url, img_name)
70      except Exception as e:
71          print('Timeout! Url: ', url)
72          return 0, 0, 0, 0
73
74      while not os.path.exists(img_name):
75          time.sleep(1)
76
77      img = cv2.imread(img_name)
78
79      gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
80      thresh_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY_INV +
        ↪  cv2.THRESH_OTSU)[1]
81
82      contours = cv2.findContours(thresh_image, cv2.RETR_EXTERNAL,
        ↪  cv2.CHAIN_APPROX_SIMPLE)
83      contours = contours[0] if len(contours) == 2 else contours[1]
84
85      xList, yList, wList, hList = [], [], [], []
86      for n, i in enumerate(contours):
87          x, y, w, h = cv2.boundingRect(i)
88          xList.append(x)
89          yList.append(y)
90          wList.append(x + w)
91          hList.append(y + h)
92      xmin = min(xList)
93      ymin = min(yList)
94      xmax = max(wList)
95      ymax = max(hList)
96      w = xmax - xmin
97      h = ymax - ymin
98
99      return [xmin, ymin, w, h], w * h, img.shape[1], img.shape[0]
```

A.3. Modelli di object detection

In questa sezione sono descritte le principali funzioni relative all'inizializzazione, configurazione, addestramento e test dei modelli scelti per eseguire *task* di *object detection*, trattati e approfonditi nelle sezioni 3.3 e 3.4, così come nel capitolo 4, dedicato ai risultati ottenuti.

```
1   def inference_det(conf_file, chkt_file, img):
2       model = init_detector(conf_file, chkt_file, device='cuda:0')
3
```

```
    # Init visualizer
    visualizer = VISUALIZERS.build(model.cfg.visualizer)
    # The dataset_meta is loaded from the checkpoint and
    # then pass to the model in init_detector
    visualizer.dataset_meta = model.dataset_meta

    # Test a single image and show the results
    result = inference_detector(model, img)

    # Show the results
    img = mmcv.imread(img)
    img = mmcv.imconvert(img, 'bgr', 'rgb')

    visualizer.add_datasample(
        'result',
        img,
        data_sample=result,
        draw_gt=False,
        show=False,
        out_file='./res.jpg')


def balance_calculation():
    with open('./dataset/annotations/annotations.json', encoding='utf-8') as file:
        json_file = json.load(file)
        df = pd.DataFrame.from_dict(json_file['annotations'])

    num_classes = len(json_file['categories'])
    # calcola la distribuzione dei campioni per classe
    class_distribution = np.array([len(df[df.category_id == i]) for i in
    ↪  range(num_classes)])
    print(class_distribution)
    class_distribution = np.array([167, 116, 221, 415, 347, 209, 500, 500, 156])
    num_classes = len(class_distribution)
    gamma = 2.0

    # Calcolo dei pesi delle classi
    p = 1.0 / num_classes
    weight_class = (1.0 - np.power(p, gamma)) / np.sqrt(class_distribution)

    # Normalizzazione dei pesi
    weights = weight_class / np.sum(weight_class)

    print(weights)


def plot_loss_cls(log_file: str):
    # Aprire il file di log in modalità lettura
    with open(log_file, 'r') as f:
        # Leggere il contenuto del file in una lista
        lines = f.readlines()
```

```
54
55      # Saltare la prima riga
56      log_data = [json.loads(line) for line in lines]
57
58      # Convertire la lista di dizionari in un dataframe
59      df = pd.DataFrame(log_data)
60      # df = df[df['mode'] == 'train']
61
62      # Creare un nuovo indice che tiene traccia dell'iterazione globale
63      global_iteration = 0
64      iteration_list = []
65
66      epoch_num = 0
67      epoch_start_iters = []
68      epoch_labels = []
69
70      for i, row in df.iterrows():
71          iteration_list.append(global_iteration + row['step'])
72          if row['step'] == max(df['step']):
73              global_iteration += row['step']
74
75              # Aggiungi un valore all'asse x all'inizio di ogni epoca
76              epoch_num += 1
77              epoch_start_iters.append(global_iteration)
78              epoch_labels.append(f'{epoch_num}')
79
80      df['iteration'] = iteration_list
81
82      # Tracciare la curva della loss_cls in funzione dell'iterazione
83      plt.plot(df['iteration'], df['loss_cls'])
84      plt.xticks(epoch_start_iters, epoch_labels)  # Imposta i valori dell'asse x
        ↪  alle epoche
85      plt.ylim(0, 2)
86      plt.xlabel('Epoche')
87      plt.ylabel('Loss_cls')
88      plt.title('loss_cls')
89
90      # Mostrare il grafico
91      plt.show()
92
93
94  def plot_mAP(log_file: str):
95      # Aprire il file di log in modalità lettura
96      with open(log_file, 'r') as f:
97          # Leggere il contenuto del file in una lista
98          lines = f.readlines()
99
100     # Saltare la prima riga
101     log_data = [json.loads(line) for line in lines if 'coco/bbox_mAP' in line]
102
103     # Convertire la lista di dizionari in un dataframe
```

```
104     df = pd.DataFrame(log_data)
105     print(df.shape)
106 
107     f = df.iloc[:, :9]
108     f.plot()
109     plt.show()
110 
111     # Crea il grafico
112     ax = df.plot(x='step', y='coco/bbox_mAP', kind='line', marker='o')
113 
114     # Aggiungi il valore in ogni punto
115     for i, row in df.iterrows():
116         ax.annotate(str(row['coco/bbox_mAP']), xy=(row['step'],
            ↪  row['coco/bbox_mAP']))
117 
118     # Mostra il grafico
119     plt.xlim(0, df['step'].max() + 1)
120     plt.show()
121 
122 
123 if __name__ == '__main__':
124     config_path = './config/'
125     chkpt_path = './chkpt/'
126     train_path = './output/'
127     img = '<FILENAME_IMMAGINE>'
128 
129     config_file = '<CONFIG_MODELLO>.py'
130     checkpoint_train = '<PESI_MODELLO>.pth'
131     inference_det(config_file, checkpoint_train, img)
132 
133     # COMANDI PRINCIPALI DA LINEA DI COMANDO
134     # Train: python mmdetection/tools/train.py config/<CONFIG_MODELLO>.py
135     # Test:  python mmdetection/tools/test.py config/<CONFIG_MODELLO>.py
        ↪  <PESI_MODELLO>.pth --out results/res.pkl --show-dir
136 
137     # Analyze-train:  python mmdetection/tools/analysis_tools/analyze_logs.py
        ↪  plot_curve <NOME_MODELLO>.json --keys loss_cls loss_bbox --legend
        ↪  loss_cls loss_bbox
138     # Analyze-val:    python mmdetection/tools/analysis_tools/analyze_logs.py
        ↪  plot_curve <NOME_MODELLO>.json --eval-interval 1 --keys bbox_mAP --legend
        ↪  bbox_mAP
139 
140     # Confusion_matrix: python
        ↪  mmdetection/tools/analysis_tools/confusion_matrix.py
        ↪  config/<CONFIG_MODELLO>.py results/res.pkl results/ --show
141     # os.system('cmd /k "python mmdetection/tools/test.py config/<CONFIG_MODELLO>
        ↪  <PESI_MODELLO>.pth')
142 
143     # plot val mAP for each class
144     # log_file = '<NOME_MODELLO>.json'
145     # plot_mAP(log_file)
```

A.4. Applicativo realizzato con servizi Amazon

In questa sezione vengono descritte le principali funzionalità dell'applicativo, sviluppato utilizzando servizi Amazon, approfonditi nei capitoli 3 e 4. Le funzionalità sono suddivise tra *backend* e *frontend*. Nel *backend*, è stata implementata la funzione *lambda_handler*, che esegue l'inferenza sull'immagine ricevuta in input, insieme a una funzione di *clustering* per estrapolare il colore dominante. Per quanto riguarda il *frontend*, è stato riportato il body della pagina web e lo script principale per la gestione dinamica della pagina in JavaScript.

A.4.1. Backend AWS Lambda

```
def lambda_handler(event, context):
    try:
        config_file = 'detr_r50_8xb2-150e_coco.py'
        checkpoint_file = './model/best.pth'
        classes = ('backpacks', 'belt bags', 'clutch bags', 'crossbody bags',
                   'handbags', 'mini bags', 'shoulder bags', 'tote bags',
                   'travel bags')

        # Decodifica l'immagine presente nel body della richiesta
        try:
            encoded = json.loads(event['body'])['image']
            img = np.frombuffer(base64.b64decode(encoded), np.uint8)
            img = cv2.imdecode(img, flags=cv2.IMREAD_COLOR)
        except (KeyError, ValueError, base64.binascii.Error) as e:
            return {
                'statusCode': 400,
                'body': json.dumps({'message': 'Bad Request'})
            }

        # Inizializza il modello, esegue inferenza e salva il risultato
        model = init_detector(config_file, checkpoint_file, device='cpu')
        visualizer = VISUALIZERS.build(model.cfg.visualizer)
        visualizer.dataset_meta = model.dataset_meta

        # Eseguire inferenza e processare il risultato
        result = inference_detector(model, img)
        triples = convert_polygon(result, with_polygons=False)
        pred = triples[0]
        label = classes[pred['label']]
        score = round(pred['score'] * 100, 1)
        bbox = pred['bbox']


        img = mmcv.imread(img)
        img = mmcv.imconvert(img, 'bgr', 'rgb')
        in_img = img
        visualizer.add_datasample(
            'result',
```

```
            img,
            data_sample=result,
            draw_gt=False,
            show=False,
            out_file='/tmp/res.png')

        # Converte l'immagine in base64
        img = cv2.imread('/tmp/res.png')
        png_img = cv2.imencode('.png', img)
        b64_string = base64.b64encode(png_img[1]).decode('utf-8')

        # Elabora il bounding box e i colori dominanti
        cropped_img = bounding_box_img(in_img, bbox)
        res = cv2.resize(cropped_img, dsize=(200, 200),
        ↪  interpolation=cv2.INTER_CUBIC)
        dom_hex, palette_hex = get_dominant_color(res)

        return {
            'statusCode': 200,
            'headers': {"Content-Type": "application/json"},
            'body': json.dumps({
                "img": b64_string,
                "label": label,
                "score": score,
                "bbox": bbox,
                "dominant_hex": dom_hex,
                "palette": palette_hex
            })
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({'message': 'Internal Server Error'})
        }
```

```
def get_dominant_color(ar):
    NUM_CLUSTERS = 10
    THRESHOLD_DISTANCE = 30

    shape = ar.shape
    ar = ar.reshape(scipy.product(shape[:2]), shape[2]).astype(float)

    print('find clus')
    codes, dist = scipy.cluster.vq.kmeans(ar, NUM_CLUSTERS)
    # print('cluster centres:\n', codes)

    vecs, dist = scipy.cluster.vq.vq(ar, codes)  # assign codes
    counts, _ = scipy.histogram(vecs, len(codes))  # count occurrences
    # print(counts)

```

```
    index_max = scipy.argmax(counts)  # find most frequent
    # print(index_max)
    peak = codes[index_max]
    dom = ''.join([hex(int(c))[2:].zfill(2) for c in peak])
    dom = f'#{dom}'

    original_dom = dom
    while color_distance(dom, "#FFFFFF") <= THRESHOLD_DISTANCE and NUM_CLUSTERS
    ↪  != 0:
        # find alternative dominant color
        counts[index_max] = 0
        index_max = scipy.argmax(counts)
        peak = codes[index_max]
        dom = ''.join([hex(int(c))[2:].zfill(2) for c in peak])
        dom = f'#{dom}'
        print(f"Il colore dominante è troppo vicino al bianco. Il colore
        ↪  successivo più frequente è {dom}")
        NUM_CLUSTERS -= 1

    if NUM_CLUSTERS == 0:
        dom = original_dom

    hex_colors = []
    for centroid in codes:
        colour_hex = ''.join([hex(int(c))[2:].zfill(2) for c in centroid])
        hex_colors.append(f'#{colour_hex}')


    return dom, hex_colors
```

A.4.2. Frontend applicazione web

```

<div class="container-fluid">
    <div class="row" style="max-height: 308px;">
        <div class="col"></div>
        <div class="col">
            <div class="container" style="background-color: #f9f9f9;">
                <div style="margin-top: 30px; margin-bottom: 30px;">
                    <label>Elabora immagini con: <b><i>DETR model</i></b></label>
                </div>
                <div>
                    <p class="loadImage">Incolla un'immagine o <label
                    ↪  for="loadImage" class="custom-file-upload">carica un
                    ↪  file</label></p>
                    <input type="file" id="loadImage" accept="image/*"
                    ↪  style="display: none">
                </div>
                <div style="margin-bottom: 40px;">
                    <button type="button" id="btn-process">Elabora</button>
                </div>
```

```
17              </div>
18          </div>
19          <div class="col"></div>
20      </div>
21      <div class="row" style="max-height: 400px">
22          <div class="col">
23                  <div id="imageContent"></div>
24          </div>
25          <div class="col">
26              <div style="padding: 60px 0;">
27                  <div id="loaderDiv" style="display: none; padding: 100px 0;">
28                          <div id="loader" ></div>
29                  </div>
30                  <div class="container", id="result-container" style="display:
                    ↪  none; width: 60%; text-align: justify;">
31                      <ul id="result-list"></ul>
32                  </div>
33              </div>
34
35          </div>
36      </div>
37  </div>
38
39  <script>
40      const do_inference = async (img) => {
41        try {
42          const response = await fetch("https://rbbphw22f6.execute-api.us-east-1.am
            ↪  azonaws.com/default/aws-inf-api",
            ↪  {
43            method: "POST",
44            body: JSON.stringify({image: img})
45          });
46
47          return await response;
48
49        } catch (error) {
50          console.error("Error:", error);
51        }
52      };
53
54      const createListItem = (label, value) => {
55          var li = document.createElement('li');
56          li.innerHTML = '<strong>' + label + ':</strong> ' + value;
57          return li
58      };
59
60      const removeAllChildren = (element) => {
61          resultDiv.style.display = 'none';
62          while (element.firstChild) {
63              element.removeChild(element.firstChild);
64          }
```

```
65  };
66
67
68
69  let base64Img = null;
70  let previousFile = null;
71  let previousPastedImg = null;
72  const imgInput = document.getElementById('loadImage');
73  const process_btn = document.getElementById("btn-process");
74  const divTarget = document.getElementById('imageContent');
75  process_btn.disabled = true;
76  const resultList = document.getElementById('result-list');
77  const resultDiv = document.getElementById('result-container');
78
79
80
81  process_btn.onclick = async () => {
82      process_btn.disabled = true;
83      const selectedFile = imgInput.files[0];
84      if (!previousFile || !(previousFile === selectedFile)) {
85          previousFile = selectedFile;
86          removeAllChildren(resultList);
87          document.getElementById('loaderDiv').style.display = 'block';
88
89          let res = await do_inference(base64Img);
90          document.getElementById('loaderDiv').style.display = 'none';
91          if (res.status === 200){
92              res = await res.json();
93              console.log(res);
94
95
96
97              const img = new Image();
98              img.src = `${res['img']}`
99              // Rimuovi eventuali immagini preesistenti
100             while (divTarget.firstChild) {
101                 divTarget.removeChild(divTarget.firstChild);
102             }
103
104             // Aggiungi l'immagine al div target
105             divTarget.appendChild(img);
106             divTarget.style.display = 'block';
107
108
109             resultList.style.display = 'block';
110             resultList.appendChild(createListItem('Classe', res['label']));
111             resultList.appendChild(createListItem('Confidenza', res['score']
                ↪  + '%'));
112             resultList.appendChild(createListItem('BBox', '[' +
                ↪  res['bbox'].map(function (value) {
113                 return value.toFixed(2);
```

```
114                 }).join(', ') + ']'));
115
116                 var colorBox = document.createElement('div');
117                 colorBox.className = 'color-box';
118                 colorBox.style.backgroundColor = res['dominant_hex'];
119                 resultList.appendChild(createListItem('Colore dominante',
                    ↪  colorBox.outerHTML + ' ' + res['dominant_hex']));
120                 const lastValue = Object.values(res).pop();
121                 lastValue.sort();
122                 resultList.appendChild(createListItem('Palette',
                    ↪  lastValue.map(function(hex) {
123                     var colorBox = document.createElement('div');
124                     colorBox.className = 'color-box';
125                     colorBox.style.backgroundColor = hex;
126                     return colorBox.outerHTML;
127                 }).join('')));
128
129                 resultDiv.style.display = 'block';
130
131             } else {
132                 res = await res.json();
133                 console.error("Errore:", res.message);
134
135                 let description = '';
136                 let solution = '';
137
138                 if (res.status === 503) {
139                     description = 'Si è verificato un errore, probabilmente
                        ↪  dovuto al "Lambda Cold Start". Questo accade quando la
                        ↪  funzione viene invocata dopo un periodo di inattività.';
140                     solution = 'Attendi qualche secondo e aggiorna nuovamente la
                        ↪  pagina.';
141                 } else if (res.status === 400) {
142                     description = 'La richiesta inviata al servizio non è
                        ↪  corretta.';
143                     solution = 'Verifica i dati inviati e riprova.';
144                 } else if (res.status === 500) {
145                     description = 'Il server ha riscontrato un errore durante
                        ↪  l\'elaborazione della richiesta.';
146                     solution = 'Riprova più tardi.';
147                 } else {
148                     description = 'Si è verificato un errore imprevisto durante
                        ↪  l\'elaborazione della richiesta.';
149                     solution = 'Riprova più tardi.';
150                 }
151
152                 resultList.style.display = 'block';
153                 resultList.appendChild(createListItem('Errore',
                    ↪  `<i>${res.message}</i>`));
154                 resultList.appendChild(createListItem('Descrizione',
                    ↪  `<i>${description}</i>`));
```

```
155                 resultList.appendChild(createListItem('Soluzione',
                    ↪  `<i>${solution}</i>`));
156
157                 resultDiv.style.display = 'block';
158
159             }
160
161          }
162      };
163
164      imgInput.addEventListener('change', (event) => {
165          process_btn.disabled = false;
166          previousPastedImg = null;
167          removeAllChildren(resultList);
168          const file = event.target.files[0];
169          if (file) {
170              const reader = new FileReader();
171              reader.onload = (e) => {
172                  const img = new Image();
173                  img.src = e.target.result;
174
175                  // Rimuovi eventuali immagini preesistenti
176                  while (divTarget.firstChild) {
177                      divTarget.removeChild(divTarget.firstChild);
178                  }
179
180                  // Aggiungi l'immagine al div target
181                  divTarget.appendChild(img);
182                  divTarget.style.display = 'block'
183
184              };
185              reader.onloadend = () => {
186                  base64Img = reader.result.split(",")[1]
187              }
188              reader.readAsDataURL(file);
189          }
190      });
191
192      document.onpaste = function(event) {
193          var items = (event.clipboardData ||
              ↪  event.originalEvent.clipboardData).items;
194          for (index in items) {
195              var item = items[index];
196              if (item.kind === 'file') {
197                  var blob = item.getAsFile();
198                  var reader = new FileReader();
199                  reader.onload = function(event){
200                      if (previousPastedImg !== event.target.result){
201                          previousPastedImg = event.target.result;
202                          removeAllChildren(resultList);
203                          const img = new Image();
```

```
204                         img.src = event.target.result;
205
206                         while (divTarget.firstChild) {
207                             divTarget.removeChild(divTarget.firstChild);
208                         }
209                         divTarget.appendChild(img);
210                         divTarget.style.display = 'block';
211                         base64Img = event.target.result.split(",")[1];
212                         process_btn.disabled = false;
213                     }
214
215                 }; // data url!
216                 reader.readAsDataURL(blob);
217
218             }
219         }
220     };
221
222 </script>
223
```

Bibliografia

[1] Razvan-Alexandru Bratulescu, Robert-Ionut Vatasoiu, George Sucic, Sorina-Andreea Mitroi, Marius-Constantin Vochin, and Mari-Anais Sachian. Object detection in autonomous vehicles. In *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 375–380, 2022.

[2] Si Van-Tien Tran, Doyeop Lee, Quy Lan Bao, Taehan Yoo, Muhammad Khan, Junhyeon Jo, and Chansik Park. A human detection approach for intrusion in hazardous areas using 4d-bim-based spatial-temporal analysis and computer vision. *Buildings*, 13(9):2313, 2023.

[3] Ganjar Alfian, Muhammad Qois Huzyan Octava, Farhan Mufti Hilmy, Rachma Aurya Nurhaliza, Yuris Mulya Saputra, Divi Galih Prasetyo Putri, Firma Syahrian, Norma Latif Fitriyani, Fransiskus Tatas Dwi Atmaji, Umar Farooq, et al. Customer shopping behavior analysis using rfid and machine learning models. *Information*, 14(10):551, 2023.

[4] Nikos K Logothetis and David L Sheinberg. Visual object recognition. *Annual review of neuroscience*, 19:577–621, 1996.

[5] Klaus-Peter Gapp. Object localization: selection of optimal reference objects. In *International Conference on Spatial Information Theory*, pages 519–536. Springer, 1995.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[7] Corinna Cortes. Support-vector networks. *Machine Learning*, 1995.

[8] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57:137–154, 2004.

[9] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[12] R Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[13] Shaoqing Ren. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[14] J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[16] T Lin. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[17] Junhyung Kang, Shahroz Tariq, Han Oh, and Simon Woo. A survey of deep learning-based object detection methods and datasets for overhead imagery. *IEEE Access*, 10:1–1, 01 2022.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[21] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[22] Yulei Wu. Cloud-edge orchestration for the internet of things: Architecture and ai-powered data processing. *IEEE Internet of Things Journal*, 8(16):12792–12805, 2020.

[23] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[24] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R. Scott, and Weilin Huang. TOOD: task-aligned one-stage object detection. *CoRR*, abs/2108.07755, 2021.

[25] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z. Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. *CoRR*, abs/1912.02424, 2019.

[26] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020.

[27] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. *CoRR*, abs/2010.04159, 2020.

[28] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[29] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

Ringraziamenti

Giunti a questo punto, è doveroso dedicare alcuni ringraziamenti a chi mi ha dato la possibilità di realizzare quanto fatto e di supportarmi in questo momento importante della mia vita.

Un ringraziamento speciale va al prof. Adriano Mancini, relatore di questa tesi, la cui passione e dedizione per il proprio lavoro è stata per me fonte di ispirazione. La sua disponibilità al confronto e i suoi consigli si sono rivelati essenziali per la buona riuscita di questo progetto. Vorrei ringraziare anche Rocco Pietrini, correlatore della tesi, ed Emanuele Balloni, che, oltre a contribuire a quanto trattato in questa tesi, sono stati fondamentali nella collaborazione su progetti che hanno arricchito in modo significativo la mia crescita professionale e personale.

Ringrazio l'intero gruppo di ricerca VRAI per avermi offerto l'opportunità di svolgere l'attività di tirocinio su cui si basa questa tesi e per tutte le altre attività correlate che mi hanno permesso di acquisire preziosa esperienza durante il mio percorso formativo. In particolare, un ringraziamento doveroso va alla prof.ssa Marina Paolanti, sempre presente e disponibile a rispondere a ogni mia richiesta, dimostrandosi un punto di riferimento costante. Le sono grato per avermi dato l'opportunità di lavorare su progetti estremamente interessanti, fondamentali per arricchire il mio bagaglio professionale.

Ringrazio i miei "compagni di viaggio", Enrico, Francesco e Arment, con cui ho condiviso questo percorso e che, insieme a me, hanno affrontato le sfide e le difficoltà con collaborazione e sostegno reciproco. La vostra amicizia è stata una risorsa preziosa e insostituibile.

Un ringraziamento speciale va alla mia famiglia, alla mia ragazza e a tutti i miei amici, che mi sono stati sempre vicini durante l'intero percorso, offrendo conforto, supporto e motivazione, elementi fondamentali per il raggiungimento di questo traguardo.

Ancona, 24 Ottobre 2024

Riccardo Mancini