

UNIVERSITÀ TELEMATICA INTERNAZIONALE
UNINETTUNO

FACOLTÀ DI INGEGNERIA

Corso di Laurea in

Ingegneria Informatica

ELABORATO FINALE

in

Ingegneria del software e programmazione ad oggetti

EcoSentinel
Sistema CNN per rilevare emissioni in atmosfera

RELATORE
Prof. Patrizia Grifoni

CANDIDATO
Emilio Verri

CORRELATORE
Prof. Mauro Mazzei

ANNO ACCADEMICO 2022/23

Ringrazio:

*I miei genitori e mia sorella che mi hanno sostenuto in questo importante traguardo.
Luca Leporati con cui ho fatto innumerevoli riunioni su teams per supportarci negli esami.*

Giulia, la mia ragazza che mi ha aiutato a gestire i momenti difficili.

I miei amici, sempre pronti a strapparmi un sorriso.

*Tutti i colleghi del FLEET che, tra una pizza e l'altra, mi hanno appoggiato nella
conclusione degli studi.*

Fusine Energia S.r.l per i test pratici della rete CNN in centrale.

*Un ringraziamento particolare al professor Mauro Mazzei che mi ha consigliato come
realizzare un progetto completo Hardware e Software, accrescendo il mio interesse per il
linguaggio di programmazione Python.*

Indice

1	Introduzione.....	3
1.1	Cosa sono le reti CNN	5
1.2	Python.....	7
1.2.1	Storia di Python	8
1.3	Web Application.....	9
1.3.1	Protocolli di comunicazione (HTTP e WebSocket).....	11
1.3.2	Utilizzo di API.....	12
2	Librerie, Software, e Applicativi utilizzati	13
2.1	Roboflow.....	13
2.1.1	Raccolta e preparazione dei dati	14
2.1.2	Funzionamento Roboflow	16
2.2	Google Colab	17
2.2.1	Cos'è una GPU?.....	17
2.2.2	Configurazioni iniziali	21
2.2.3	Codice per addestrare la CNN	24
2.2.4	Grafici e Confusion Matrix.....	28
3	Caso di studio sviluppato.....	29
3.1	Scenario Applicativo	30
3.2	Parte Software	30
3.2.1	Struttura del Progetto	31
3.2.2	Back-End "YOLO_Video.py"	36
3.2.3	Back-End "smokeapp.py"	46
3.3	Problematiche riscontrate.....	47
3.3.1	Problemi con Importante Fumata	49
3.3.1	Augmentations	51
3.4	Test Rete CNN	52
3.4.1	Raspberry	52
3.4.2	Prove Effettuate.....	54
4	Conclusioni e sviluppi futuri.....	55
5	Bibliografia.....	58

Sommario

Nel corso del progetto EcoSentinel sono stati evidenziati numerosi vantaggi riguardanti le CNN (reti neurali convoluzionali). L'introduzione delinea l'obiettivo principale e la motivazione del lavoro svolto. Uno degli aspetti centrali della ricerca è stato il concetto di rete CNN, che rappresentano uno strumento informatico fondamentale per l'analisi e la classificazione delle immagini, infatti sono discusse le caratteristiche principali e l'utilizzo nella rilevazione di emissioni atmosferiche.

Python è un linguaggio di programmazione che ha svolto un ruolo fondamentale nell'implementazione delle CNN e nell'elaborazione dei frame analizzati. Sono stati forniti una panoramica sulla sua storia e sull'evoluzione del linguaggio nel tempo (riducendo i bug), sottolineando la sua importanza nel campo dell'apprendimento automatico.

Inoltre è spiegata l'importanza delle web-application, infatti vengono argomentati i protocolli di comunicazione tra cui http, WebSocket e l'utilizzo di API.

Durante lo sviluppo di EcoSentinel sono esaminate una serie di librerie, software e applicativi. Una delle principali piattaforme sfruttate per l'analisi, la classificazione, la raccolta e la preparazione dei dati per la rilevazione di emissioni è Roboflow. Si è dimostrato uno strumento essenziale poiché ha semplificato la gestione delle informazioni. Ha permesso di annotare e organizzare facilmente un grande database di immagini.

Successivamente viene descritto l'utilizzo di Google Colab, utilizzato soprattutto per la sua accessibilità alle unità di elaborazione grafiche (GPU) basate su cloud. Viene fornita una comprensione sulle graphic processing unit e il loro utilizzo per velocizzare l'addestramento delle CNN. Inoltre, sono spiegate le configurazioni iniziali necessarie per iniziare a lavorare con il notebook digitale, inclusa l'installazione di pacchetti e librerie.

Nel processo di addestramento delle reti neurali è presente il codice dettagliato utilizzato per l'addestramento del modello, integrato di esempi grafici e confusion matrix, per valutare le prestazioni in modo accurato.

La combinazione di Roboflow e Google Colab ha dimostrato di essere una scelta vincente per la preparazione dei dati e l'addestramento delle reti, contribuendo al raggiungimento dell'obiettivo principale.

Nella ricerca è stata spiegata l'importanza di un sistema di rilevamento delle emissioni, sottolineando che apporterebbe un contributo significativo al bene comune e all'ambiente.

È stato evidenziato come possa migliorare il monitoraggio ambientale, consentendo una risposta rapida in caso di incendi o situazioni di emergenza. Questa tecnologia potrebbe contribuire alla prevenzione della salute pubblica.

Nella seconda parte è affrontato il caso di studio sviluppato, spiegando la struttura del progetto e i software coinvolti. Viene descritta un'analisi dettagliata del back-end, soffermandosi sui 2 file principali scritti in Python “YOLO_Video.py” e “smokeapp.py”. Viene fornita una panoramica completa di come sono strutturate le funzioni, sottolineando le funzionalità e le interazioni.

Durante lo sviluppo, sono risultate diverse problematiche relative alla rilevazione delle “importanti fumate” e all’implementazione di tecniche di augmentation dei dati. Vengono illustrati le strategie utilizzate per ovviare a questi problemi.

Nell’ultima sezione sono approfonditi i test condotti sulla rete CNN, con focus il test fisico attraverso l’utilizzo di un Raspberry Pi 3 Model B, che ha permesso di validare il modello.

Durante il test il dispositivo è stato utilizzato come piattaforma di implementazione siccome è dotato di un’ottima efficienza energetica. Il tool è stato installato all’interno della centrale di fusine (Fusine Energia S.r.l) in modo strategico per monitorare le emissioni.

In sintesi, questa sezione della ricerca ha delineato il caso di studio sviluppato in modo approfondito, fornendo una comprensione chiara dell'ambito software, della struttura del progetto, delle sfide affrontate e dei test condotti. Ha dimostrato l’efficacia delle reti neurali convoluzionali nella rilevazione di emissioni in atmosfera e ha evidenziato il valore di questo progetto nel contribuire al bene comune.

Infine, si è discusso dell'importanza di una migliore gestione delle utenze attraverso l'utilizzo di un database su un server con autenticazione utente. Questa considerazione è emersa come un possibile sviluppo futuro, per garantire una gestione multiutente dell'applicativo e una maggiore accessibilità.

1 Introduzione

Il progetto contenuto in questo elaborato si concentra sull'applicazione delle CNN (reti neurali convoluzionali) nel campo della rilevazione di emissioni in atmosfera e nello sviluppo di un sito web che consenta, a chiunque ne abbia necessità, di rilevare in tempo reale la presenza di fumo attraverso l'uso di una telecamera collegata a qualsiasi dispositivo connesso online. Le CNN sono algoritmi di deep learning efficaci nel riconoscimento di modelli attraverso le immagini.

L'obiettivo di EcoSentinel è la realizzazione di un sistema CNN per la rilevazione di inquinanti atmosferici che aiuti nella prevenzione degli incendi.

La rilevazione di emissioni è fondamentale in quanto può contribuire ad individuare anticipatamente la presenza di un potenziale incendio e attivare le attività di prevenzione per la tutela di edifici e ambienti pubblici.

Le reti neurali convoluzionali sono addestrate al riconoscimento delle caratteristiche del fumo e il tool è stato progettato in modo da poter utilizzare agilmente il sistema.

Lo scopo è di realizzare uno strumento efficace nella rilevazione di fumo senza dover ricorrere ad apparecchiature complesse.

Agli utenti del tool verranno proposte due alternative per controllare la presenza di fumi: l'uso di una telecamera propria o la possibilità di caricare un video.

La prima opzione consente agli utenti di utilizzare la webcam del proprio dispositivo per inquadrare l'ambiente circostante e quindi monitorare con costanza la presenza di fumi. Attivata questa funzionalità il sistema continuerà ad acquisire immagini dalla telecamera e applicherà il modello CNN preaddestrato.

Nel caso in cui il programma individui del fumo, verrà generato un riquadro intorno alla zona interessata che varierà dimensione in base alla quantità di fumo rilevata.

La seconda opzione permette agli utenti di caricare un video preregistrato consentendo loro di rilevare la presenza di vapori tossici in momenti specifici. Questo può risultare utile per molteplici scopi, come ad esempio l'analisi di video di sorveglianza.

Nel 2022 gli incendi sono stati numerosi nei paesi dell'UE, in Italia è bruciata un'area complessiva di 1.624.381 ettari, causando seri danni economici e alla salute delle persone, danni alla vegetazione e all'ambiente.

Dato che EcoSentinel è istruito nel rilevare la presenza di fumo, può contribuire alla prevenzione incendiaria, identificare le zone colpite e facilitare l'uscita delle autorità preposte, riducendo i danni.

In sintesi un sistema CNN può fornire un mezzo efficace per la prevenzione e il monitoraggio di incendi.

Sono molti gli strumenti utilizzati nella realizzazione di EcoSentinel, tra i principali: Python, Flask, Roboflow, Google Colab, NumPy, OpenCV, Ultralytics, UIKit, Html e Css.

Python è il linguaggio principale del progetto, utilizzato nel framework Flask per la creazione di web-app. È uno dei linguaggi di programmazione più utilizzati in quanto la sintassi di Python è intuitiva e permette di importare librerie come OpenCV, NumPy e Ultralytics.

L'aspetto grafico è stato realizzato con HTML, CSS e UIKit, un framework CSS che mette a disposizione componenti per la creazione di web-app responsive.

UIKit è appunto un framework responsive, significa che il front-end del sito si adatta alla dimensione dello schermo utilizzato dall'utente e fornisce una vasta selezione di componenti utilizzabili all'interno di un'interfaccia web, tra cui bottoni, modali, form, tabelle e molto altro.

Ultralytics è una delle librerie di Python, offre strumenti per la gestione dei dataset ed essendo specializzata nella creazione di algoritmi di deep learning, mi ha fornito ottime funzionalità nell'addestramento del modello CNN per la rilevazione delle emissioni atmosferiche. Ultralytics è collegato a YOLO perché è una libreria che fornisce strumenti per il rilevamento di oggetti utilizzando il framework YOLO.

Per l'analisi delle immagini sono state utilizzate due librerie: OpenCV e NumPy.

“Open Source Computer Vision Library” è una delle librerie più popolari tra gli sviluppatori che lavorano su progetti di machine learning basati sull'utilizzo di immagini, è interfacciabile attraverso l'utilizzo di Python e fornisce tante funzionalità come la rilevazione di oggetti, il tracciamento dei movimenti, l'elaborazione video e molto altro. Viene utilizzato in svariati campi tra cui l'industria, la medicina, la sorveglianza e la robotica.

“Numerical Python” NumPy è una libreria che offre invece supporto ad array multidimensionali, e funzioni matematiche.

L'utilizzo di Google Colab è stato cruciale per il progetto perché offre un ambiente conveniente e flessibile trattandosi di una piattaforma cloud.

Inoltre fornisce l'accesso alle risorse di calcolo, compresa la GPU utilizzata in questo progetto, ed è stato adoperato per istruire il modello CNN unito al database di immagini preparato con Roboflow.

Il codice viene eseguito direttamente sui server di Google e garantisce supporto per il linguaggio di programmazione Python.

Roboflow è una piattaforma web che mette a disposizione strumenti per il processo di raccolta dei dati e l'annotazione delle immagini. L'obiettivo è quello di assegnare ad ogni immagine caricata su Roboflow una o più classi. Questo per permettere di creare un set di immagini annotate utilizzabile per addestrare modelli CNN. Nell'utilizzo dell'applicativo è comune dividere il database in tre set principali:

- L'addestramento "Train": utilizzato per addestrare il modello;
- La validazione "validation": utilizzato per la valutazione delle prestazioni del sistema durante l'addestramento;
- Il Test "test": usato per testare le capacità del modello;

EcoSentinel realizzato attraverso l'utilizzo degli strumenti, linguaggi, framework e librerie sopra descritte garantisce una totale gestione delle emissioni di fumo.

La CNN consente un monitoraggio continuativo dell'ambiente e grazie alla sua capacità di rilevare anticipatamente le emissioni in atmosfera, il sistema può contribuire a limitare i danni causati da incendi.

1.1 Cosa sono le reti CNN

La storia delle CNN (reti neurali convoluzionali) risale al 1950/1960 quando gli scienziati Hubel e Wiesel scoprirono neuroni nella corteccia visiva di gatti e scimmie, scoperte fondamentali per le reti neurali convoluzionali. Nel 1980 fu utilizzato il neocognitron un modello CNN che non richiedeva pesi addestrabili in multiple posizioni. Mentre nel 1998 venne creata la LeNet-5 una rete CNN per riconoscere dei numeri scritti a mano.

Con la creazione delle GPU (Graphics Processing Unit), un processore specializzato nell'esecuzione di più operazioni grafiche in contemporanea, adatto per il calcolo grafico, sono state create implementazioni più funzionali per addestrare le reti CNN.

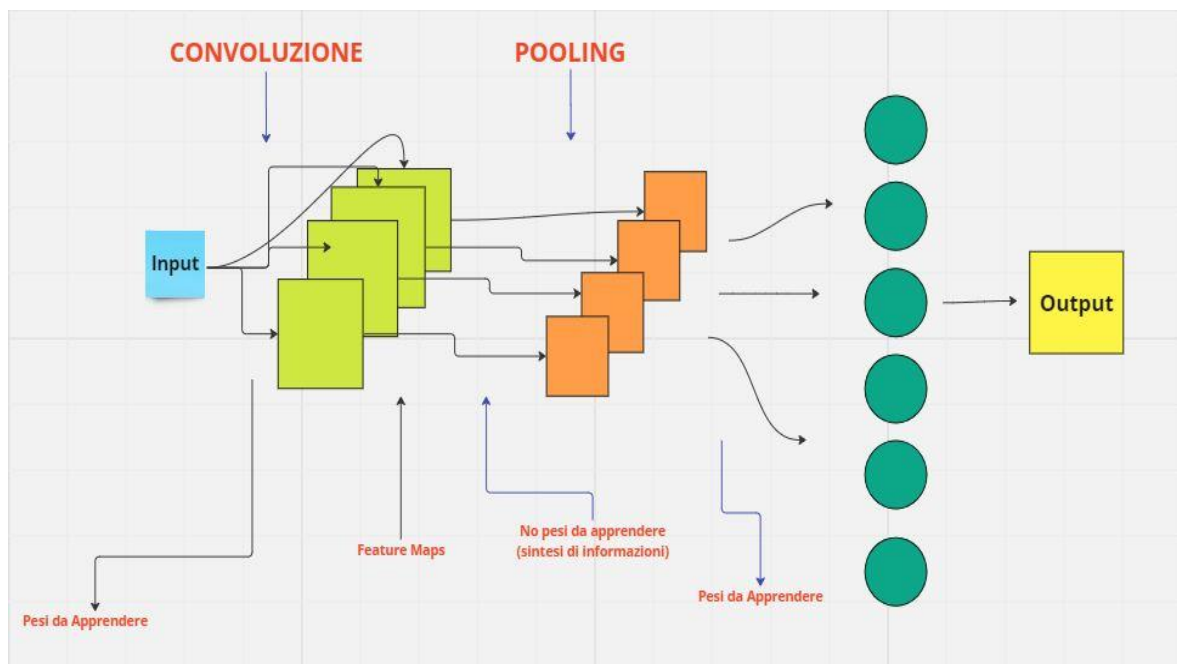


Figura 1.1 – Funzionamento reti CNN

Il processo inizia con un'immagine in Input, che subisce una convoluzione con dei filtri. I filtri sono trascinati su tutto l'input convolvendosi con esso in modo da formare diverse mappe (feature maps). I filtri sono delle piccolissime matrici di pesi che vengono traslati su tutta l'immagine per poi essere moltiplicati con i pixel corrispondenti. Vengono generate diverse "feature maps" ognuna delle quali raccoglie informazioni specifiche dall'immagine.

Ogni feature maps in seguito alla convoluzione subirà un processo di pooling, che è un modo per ridurre il numero di pesi presenti nell'architettura. In questo passaggio non ci sono nuovi pesi da imparare ma soltanto una sintesi di informazioni.

In seguito allo stato di pooling, viene connesso alla rete un multiplayer perceptron che è fully connected, cioè ogni singolo neurone del livello di pooling è connesso ad ogni neurone del livello intermedio (detto anche fully connected layer). In cascata con il livello intermedio (contrassegnato con delle sfere nel disegno sopra) viene prodotto l'output. L'Output si produce in cascata con il livello intermedio. Questo stato cambia in base alla problematica che la rete CNN deve affrontare. Per la classificazione di fumi, l'output sarà composto da neuroni che rappresentano la classe del fumo da riconoscere.

Il momento più importante nella storia delle reti neurali convoluzionali che ha permesso la creazione di EcoSentinel è la nascita di “Alex Net” nel 2012, realizzato da Alex Krizhsky, Geoffrey Hinton e Ilya Sutskeve.

Questa CNN è stata una delle prime composta da un dataset di 1 milione di immagini e 1000 classi diverse (classificate in animali, visi di persone, cibo e molto altro).

Queste reti continuano e continueranno ad essere al centro dell’interesse di scienziati e ingegneri, costantemente impegnati nella ricerca di nuovi algoritmi o architetture che possano migliorare sempre di più la velocità e la qualità nella rilevazione di immagini, rendendo indispensabili le CNN in settori complessi come l’automazione industriale, la medicina e la guida autonoma delle autovetture. Con l’evoluzione tecnologica e l’accesso a enormi quantità di dati con un click, le reti neurali convoluzionali giocheranno un ruolo sempre più importante nella visione artificiale.

1.2 Python

Python è un’ottima scelta per l’implementazione di reti CNN per rilevare il fumo nell’ambiente. È un linguaggio molto popolare tra i programmatori e ciò spiega il fatto che esistono numerose librerie e framework sviluppati per l’apprendimento automatico, che riescono a semplificare lo sviluppo e la complessità del codice da scrivere. Inoltre è uno dei linguaggi classificato come il più leggibile e intuitivo, che sfrutta delle sintassi chiare e semplici.

Altre caratteristiche importanti:

- L’indentazione obbligatoria è una delle caratteristiche che distingue Python da qualsiasi altro linguaggio di programmazione e che garantisce una lettura migliore del programma realizzato;
- La compatibilità con le GPU (Graphic Processing Unit) e le TPU (Tensor Processing Unit), entrambe adatte per addestrare le reti CNN.
- Una grande community con la quale poter discutere di tematiche ed errori comuni, condividere risorse e tutorial;
- La facilità di integrazione con altri linguaggi di programmazione. È infatti possibile utilizzare librerie esterne scritte in Java, C, C++ e tanti altri;
- La documentazione ufficiale Python che fornisce una curata e specifica guida per l’utilizzo di librerie e framework;

- La possibilità di lanciare le applicazioni scritte in Python su più piattaforme senza dover modificare eccessivamente il codice;
- L'aiuto nella manipolazione dei dati, python insieme alla libreria NumPy integrata all'interno di EcoSentinel, offre la possibilità di manipolare in maniera ottimale gli array multidimensionali;
- La grande capacità di elaborare immagini, OpenCv è la libreria che consente di ridimensionare, ritagliare, caricare, salvare ecc... Molto utile per gli applicativi di object detection;
- La possibilità di utilizzare librerie come Flask per lo sviluppo web, semplice e intuitivo per gestire le richieste http;
- Condivisione di informazioni con Google Colab. È una piattaforma cloud che consente l'utilizzo di un ambiente di sviluppo che supporta Python e permette di eseguire il suo codice senza dover installare nulla localmente. Il vantaggio di questa risorsa messa a disposizione da "Google" è la possibilità di poter diffondere i notebook scritti con altre persone;
- Semplicità nel definire le funzioni utilizzando la parola chiave "def", come tutti i linguaggi di programmazione permette di specificare i parametri in input, il corpo e un ipotetico valore di ritorno;
- L'ampia varietà di funzioni disponibili, senza dover importare librerie esterne. E la possibilità di utilizzare le funzioni lambda, che sono molto utili quando bisogna passare una funzione come argomento di un'altra funzione creata;
- L'ottima gestione delle eccezioni, che consente di gestire situazioni di errore.
- Programmazione Orientata agli Oggetti (OOP);
- Un'istanza è un oggetto creato da una classe, mentre una classe è una descrizione di un oggetto. I metodi che le istanze possono utilizzare sono definiti dalle classi. Il vantaggio di Python come molti altri linguaggi di programmazione è che supportano l'ereditarietà, cioè la possibilità di creare classi che ereditano attributi e metodi da classi che esistono.
- Le variabili vengono associate ad uno specifico tipo di dato durante la dichiarazione e possono assumere diversi tipi di valori durante l'esecuzione. Le variabili vengono assegnate utilizzando il simbolo "=". Ad esempio, per assegnare il valore 5 alla variabile "x" scriveremo "x=5";

- È possibile inserire variabili che possono essere utilizzate solo all'interno di una specifica funzione "locali" o variabili che sono accessibili dentro l'intero progetto;

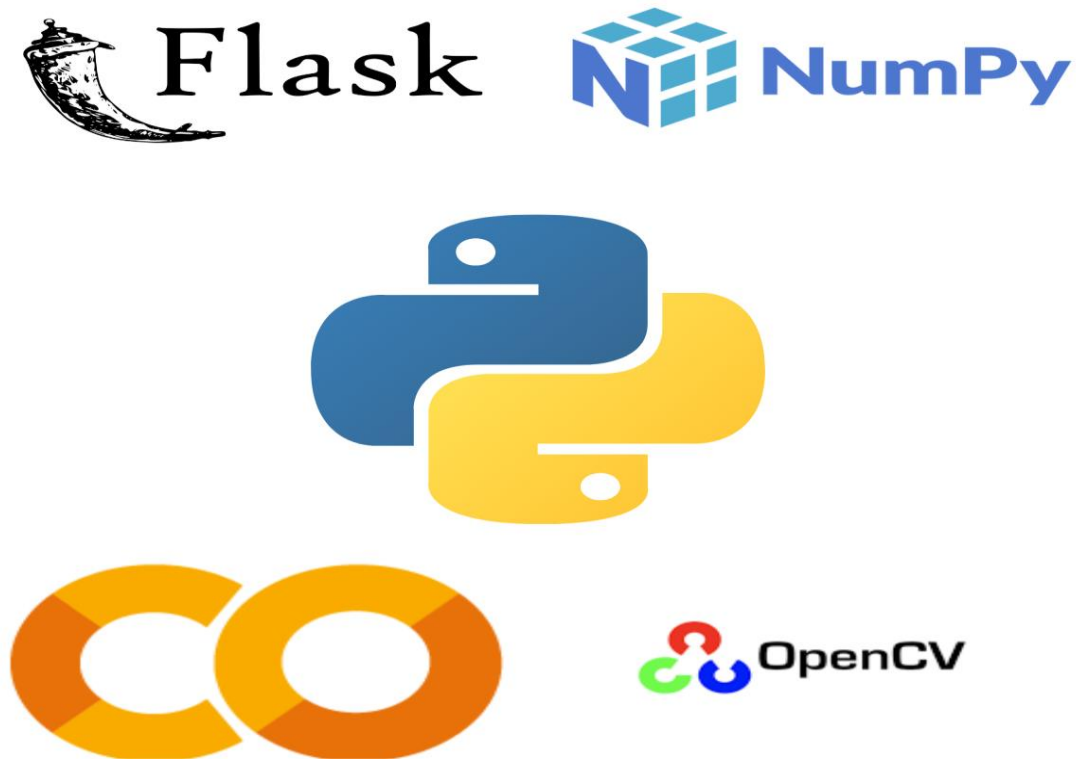


Figura 1.2 – Strumenti principali utilizzati nel progetto

1.2.1 Storia di Python

La creazione di Python risale al 1989 per mano dell'informatico olandese Guido van Rossum, esso doveva essere il successore del linguaggio ABC. Nel 2000 è stato rilasciato Python 2.0 in cui la modifica più importante è stata la trasparenza nel processo di sviluppo per la community. In questa versione è stata introdotta una gestione ottimale delle eccezioni, così da garantire agli sviluppatori un controllo migliore sugli errori nel codice.

L'ultima versione che ha segnato la storia di Python è stata nel 2008 con Python 3.0. Questa versione ha contribuito al miglioramento della sintassi rendendo il codice ancora più chiaro e leggibile.

Python ha continuato ad evolvere grazie al contributo di molti sviluppatori, ogni nuova versione ha introdotto funzionalità specifiche e corretto i bug presenti nelle vecchie release.

Queste sono le principali tappe storiche per il linguaggio. Esso continua ad evolversi e a migliorare nel campo dello sviluppo software, mantenendo la sua reputazione di essere uno strumento semplice e versatile.

1.3 Web Application

L'obiettivo del progetto è di realizzare una web-app che offra all'utente l'uso dei servizi di EcoSentinel.

Cosa è una Web-Application?

È un programma eseguito e visualizzato accedendo ad un browser. Quando una utente accede ad un sito web, il browser invia una richiesta al server in cui è stata caricata la pagina web ed il server invierà come risposta il codice HTML.

Funzionamento delle Web-Application:

- 1) Sfruttano un modello Client-Server, che va ad etichettare come client il browser dal quale l'utente accede tramite un dispositivo, e come server il computer che ospita la web-app;
- 2) Come già spiegato sopra un sito web funziona a richieste e risposte, quando un utente clicca su un sito il browser invia una richiesta http al computer ospitante (server);
- 3) Controllo da parte del server, quando il server riceve la richiesta inviata dal browser inizia ad elaborarla ed esegue il codice della web-app;
- 4) Il Back-End, è la logica dell'applicazione. Viene eseguita dal server e può essere scritto in vari linguaggi di programmazione tra cui Python, Java, Php e molti altri. Si occupa di gestire i dati e le comunicazioni con il front-end. Questa parte è nascosta all'utente che sta utilizzando un'applicazione web. Una delle principali caratteristiche del back-end è la gestione della sicurezza di un sito web;
- 5) Gestione del Database, con cui si recuperano i dati di applicazioni non statiche. Esso viene interrogato dal back-end per creare, cancellare o modificare i dati alla richiesta del client;
- 6) Invio della risposta, il server inizia a generare il contenuto e dopo averlo creato invia una risposta al browser che per la maggior parte dei casi è una pagina HTML;
- 7) Front-End, definito quella parte del programma che viene ricevuta come risposta sul browser, elaborata e resa visibile all'utente. È l'interfaccia con cui gli utenti interagiscono. Il front-end non è in grado di memorizzare i dati, è quindi indispensabile il Back-end.

Uno degli aspetti più importanti delle applicazioni web è l'accessibilità garantita all'utente da qualsiasi dispositivo dotato di browser e accesso ad internet. Ad ogni accesso l'utente avrà a disposizione la versione più aggiornata del sito web, gli aggiornamenti vengono gestiti lato server. Un altro vantaggio è che sono cross-Platform, cioè funzionano su diversi dispositivi e sistemi operativi.

I siti web sono talmente sviluppati che possono essere integrati con API e librerie esterne, come l'integrazione dei servizi di pagamento o dei social media.

EcoSentinel sfrutta come front-end HTML, con cui è stata creata la struttura della web application e UIKit un framework css che offre un'interfaccia moderna. Il Back-end è stato implementato con Python, responsabile della gestione dell'elaborazione dei video caricati o dal filmato della webcam.

Per rilevare le immagini, Python utilizza un modello preaddestrato attraverso l'utilizzo di Roboflow ed istruito con Google Colab.

1.3.1 Protocolli di comunicazione (HTTP e WebSocket)

Per consentire a uno o più dispositivi di comunicare tra loro vengono utilizzate delle regole chiamate "Protocolli di comunicazione". Nelle applicazioni web ci sono diversi protocolli di comunicazione che permettono la comunicazione tra back-end e front-end.

Il WebSocket è un protocollo di comunicazione creato per avere una connessione continuativa tra client e server. Consente un passaggio di informazioni in tempo reale, spesso utilizzata nei video giochi. Mantiene la connessione continuamente aperta consentendo la possibilità ai dati di essere scambiati in entrambe le direzioni, senza dover ad ogni scambio stabilire una nuova connessione.

Il Protocollo http (Hypertext Transfer Protocol) lo possiamo trovare all'interno di molte parti del progetto EcoSentinel. La comunicazione http avviene attraverso uno scambio di richieste e risposte tra il client e il server.

Nel codice viene utilizzato il protocollo http:

- 1) Funzione "home" e funzione "webcam" all'interno del file "smokeapp.py". Sono delle route che vengono indicate con "app.route()" e vengono chiamate attraverso richieste http al server. Quando un utente accede all'url corrispondente alle due route nel browser viene restituito il codice HTML. Quando l'utente cercherà di accedere

all'url del sito corrispondente alle route, verranno restituite corrispondentemente le pagine “index.html” e “webcam.html”.

```
@app.route('/home', methods=['GET', 'POST'])
def home():
    session.clear() # Cancella la sessione corrente
    return render_template('index.html') # Mostra il template index.html

@app.route("/webcam", methods=['GET', 'POST'])
def webcam():
    session.clear() # Cancella la sessione corrente
    return render_template('webcam.html') # Mostra il template webcam.html
```

Figura 1.3 – Funzioni “home” e webcam”

- 2) Nella funzione “front” troviamo la gestione delle richieste http POST che arrivano dal form presente nella pagina “video.html”. Quando l’utente carica un video viene lanciata una richiesta POST al server. Il server riceve la richiesta, salva il file nella cartella dei video e successivamente salva in sessione il percorso del video.

```
@app.route('/FrontPage', methods=['GET', 'POST'])
def front():
    form = UploadFileForm() # Crea un'istanza del form di caricamento file
    if form.validate_on_submit(): # Se il form viene inviato e supera la validazione
        file = form.file.data # Ottiene il file caricato dal campo del form
        # Salva il file nella cartella di destinazione con un nome sicuro
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'], secure_filename(file.filename)))
        # Salva il percorso del video nella sessione
        session['video_path'] = os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'], secure_filename(file.filename))

        # Mostra il template video.html con il form
    return render_template('video.html', form=form)
```

Figura 1.4 – Funzione “front”

- 3) Le funzioni “video” e “webapp” sono anch’esse definite con “@app.route()”. La prima restituisce i frame del video mentre la seconda quelli della webcam. Quindi quando l’utente navigherà verso l'url di queste 2 route viene generata una richiesta GET al server per catturare i frame, che vengono restituiti come risposta http al client;

```
@app.route('/video')
def video():
    # Restituisce i frame del video
    return Response(generate_frames(path_x=session.get('video_path', None)), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/webapp')
def webapp():
    # Restituisce i frame della webcam
    return Response(generate_frames_web(path_x=0), mimetype='multipart/x-mixed-replace; boundary=frame')
```

Figura 1.5 – Funzioni “video” e “webapp”

I messaggi http sono di 2 tipi: richieste e risposte. Sono scritti in formato ASCII. Le richieste si distinguono in base al metodo utilizzato:

- 1) Metodo GET, viene usato per recuperare dati dal server, quando c'è una GET il client va a specificare un indirizzo di risorsa nell' URL;
- 2) Metodo POST, è utilizzato per inviare dati al server. Quando viene lanciata una richiesta POST, i dati vengono spediti nel corpo della richiesta in formato JSON o XML;
- 3) Metodo DELETE, serve per cancellare una risposta specificata dall'URL. Al lancio di una DELETE il server elimina la risorsa corrispondente all'indirizzo indicato;
- 4) Metodo PUT, utilizzato per aggiornare un dato. I dati vengono inviati dentro alla richiesta e toccherà al server decidere se creare una nuova informazione oppure aggiornare quella presente.

Tutte le risposte che restituisce il server sono seguite da un codice di stato. Ne esistono parecchi, i più comuni sono:

- 200 Ok, il server ha elaborato correttamente la richiesta del client e ha restituito una risposta corretta;
- 201 Created, viene utilizzato quando una POST ha avuto successo e ha creato una nuova risorsa sul server;
- 202 Accepted, il server sta elaborando la risposta e il risultato finale non è ancora stato completato;
- 400 Bad Request, viene restituito se il client ha spedito dati in un formato sbagliato o non ha inserito tutte le informazioni obbligatorie;
- 401 Unauthorized, significa che il server richiede un'autenticazione prima di accedere alle risorse richieste dal client;
- 403 Forbidden, si riceve questa risposta quando il client deve autenticarsi e il server richiede un'autenticazione prima di garantire l'accesso alle informazioni;
- 404 Not Found, una delle risposte più comuni, viene restituita quando la risorsa richiesta non è presente sul server o l'url è sbagliato.

I codici di stato http vengono utilizzati come risposta dal server al client per comunicare l'esito della richiesta. Permettono di avere informazioni sulla riuscita della richiesta e di indagare se ci sono delle problematiche nella comunicazione server e client. Gestire gli errori che una web application può produrre è importante.

1.3.2 Utilizzo di API

Un API consente a 2 sistemi software di scambiarsi informazioni. Nel mondo delle applicazioni web e della programmazione, le API sono importanti perché oltre che a comunicare consentono alle applicazioni di utilizzare le funzionalità di altre tecnologie.

Le API vengono definite da file XML o JSON che stabiliscono il formato dei dati scambiati, per consentire ai 2 software, che possono utilizzare linguaggi di programmazione uguali o differenti, di comunicare con le diverse applicazioni. Per comunicare con un API ci deve essere una richiesta e una risposta. Un'app invia una richiesta ad un API con le informazioni da estrarre e quest'ultima risponde con i dati richiesti. Per garantire la sicurezza di dati sensibili un'API può richiedere un'autenticazione che può essere pubblica e privata. Quelle pubbliche sono accessibili a qualsiasi programmatore, mentre quelle private sono visibili solo da software con accessi adeguati.

Nel codice per la creazione di EcoSentinel viene utilizzato il modello YOLO che non è a tutti gli effetti un API ma è un algoritmo di rilevamento oggetti. Nella prima parte del codice sono state importate librerie per lavorare con immagini e video, tra cui OpenCv e YOLO con ultralytics per la rilevazione degli oggetti.

Con la funzione “video_detection” viene utilizzato il parametro “path_x” che rappresenta il percorso del video da analizzare per il rilevamento del fumo. Dal modulo ultralytics viene caricata la classe YOLO con un modello che è stato preaddestrato con Google Colab. La funzione prende i frame del video e li legge con la classe “cv2.VideoCapture” e itera su tutti i frame del video. Per ogni frame il modello YOLO viene lanciato usando il metodo “model(img,stream=True)”, che è un metodo per rilevare gli oggetti presenti nell'immagine.

YOLO non fa riferimento ad un API ma al modello YOLO che è stato reso disponibile dai suoi creatori.

```
# Importa la classe YOLO dal modulo ultralytics
from ultralytics import YOLO

# Importa il modulo cv2 per lavorare con immagini e video utilizzando OpenCV
import cv2
# Importa il modulo math per funzioni matematiche
import math

3 usages
def video_detection(path_x):
    # Assegna il valore del parametro path_x alla variabile video_capture
    video_capture = path_x

    # Apre il video per la cattura del frame
    cap = cv2.VideoCapture(video_capture)
    frame_width = int(cap.get(3)) #larghezza frame
    frame_height = int(cap.get(4)) # altezza frame

    model = YOLO("../YOLO-Weights/fireSmoke.pt") # Carica il modello YOLO preaddestrato per rilevare fuoco e fumo
```

Figura 1.6 – Funzione “video_detection” ed import ultralytics (modello YOLO)

2 Librerie, Software e Applicativi utilizzati

EcoSentinel è stato sviluppato creando un dataset di immagini nella web-application Roboflow per addestrare un modello CNN. Le immagini raccolte sono state utilizzate come dati di input con l'obiettivo di riconoscere le emissioni atmosferiche.

Per l'addestramento del modello è stato utilizzato Google Colab attraverso la libreria Ultralytics. È stato generato un file “.py” che sarà richiamato dal codice Python per monitorare l'inquinamento ambientale.

2.1 Roboflow

Roboflow è una piattaforma web di deep learning specifica per la gestione di immagini. La piattaforma è stata utile per semplificare attività complesse.

Roboflow ha numerosi strumenti che permettono di caricare e classificare dati di immagini e video. Garantisce la possibilità di estrarre specifici frame di un filmato ed è completa di database propri. Per lo sviluppo del progetto i dati sono stati annotati manualmente senza l'utilizzo di algoritmi di intelligenza artificiale. Supporta numerosi formati per addestrare i modelli, tra cui YOLO utilizzato in Google Colab e all'interno del codice Python. Uno degli aspetti più importanti è la possibilità di poter sfruttare dataset condivisi da altri utenti. Roboflow mette a disposizione strumenti in grado di monitorare le prestazioni del modello nel tempo e supporta l'addestramento su GPU e su TPU.

2.1.1 Raccolta e preparazione dei dati

Le emissioni atmosferiche utilizzate per addestrare il modello CNN sono state acquisite da frame di zone industriali. Le emissioni sono state classificate in tre categorie: Lieve Fumata, Evidente Fumata e Importante Fumata:

- "Lieve Fumata" indica emissioni a bassa intensità rappresentate da una sottile colonna di fumo o gas rilasciata nell'aria. Sebbene siano meno evidenti, le emissioni di questa classe sono comunque dannose per l'ambiente ed è fondamentale distinguerle. Il modello CNN è stato addestrato per classificare tali emissioni anche quando la loro intensità è minima;
- "Evidente Fumata" rappresenta emissioni d'intensità moderata ma ben visibili ad occhio nudo. Queste emissioni possono apparire come colonne di fumo dense. In

genere sono il risultato di attività industriali. L'identificazione delle sostanze inquinanti è migliore rispetto alla lieve fumata;

- "Importante Fumata" rappresenta emissioni di inquinanti in quantità elevate e molto visibili. Sono costituite da una grande quantità di fumo, o gas. Sono riconoscibili anche a lunghe distanze e indicano gravi problemi di inquinamento atmosferico. Il modello CNN è addestrato per rilevare e affrontare situazioni di emergenza ambientale.

Etichettando le immagini nelle diverse categorie di fumata, si ottiene un sistema di riconoscimento accurato che garantisce all'utente una classificazione semplice e veloce. Nella "HomePage" del sito è presente una classificazione delle emissioni e la loro localizzazione nella sezione "fotocamera" e "webcam".

2.1.2 Funzionamento Roboflow

Il primo passo è quello di creare un nuovo progetto nella web-app Roboflow. Si assegna un nome e si specifica la classificazione delle immagini, ovvero "lieve fumata", "evidente fumata" e "importante fumata". Si decide, inoltre, se il dataset di immagini raccolte sarà pubblico o privato. Una volta completato il primo passaggio, il sito reindirizza alla sezione "Unassigned" in cui si caricano i frame da classificare. È possibile caricare sia video che foto.

Dopo aver caricato il dataset, ci si sposta nella sezione "Annotating", dedicata alla classificazione delle immagini. Per ogni frame si utilizza lo strumento fornito dall'applicativo per creare una label box intorno all'area in cui è presente il fumo. Successivamente, il programma chiede per ciascun riquadro una delle tre classi precedentemente specificate: "lieve fumata", "evidente fumata" o "importante fumata". Se all'interno di un'immagine sono presenti diverse zone con emissioni atmosferiche, è possibile inserire più label box.

Più immagini sono presenti nel dataset, migliore sarà la qualità del training. Il processo di annotazione richiede tempo poiché effettuato manualmente. Una volta completato, si ottiene un dataset completo, fondamentale per addestrare il modello di rilevazione delle emissioni inquinanti.

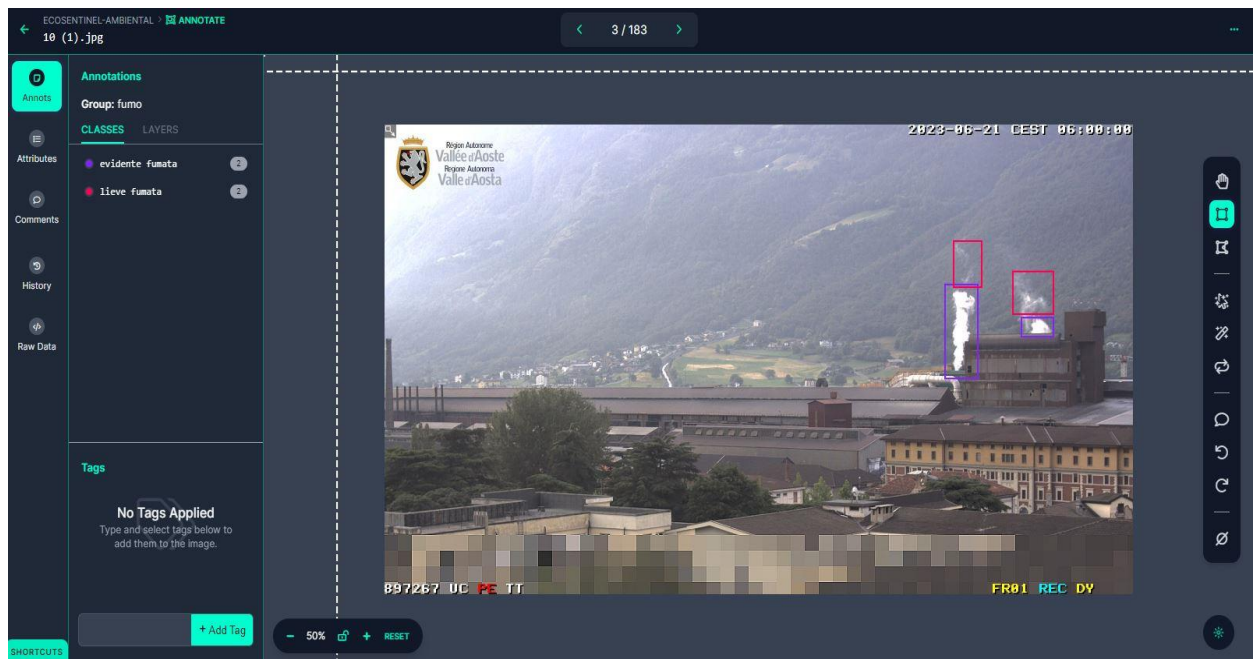


Figura 2.1 – Strumento di Roboflow per la creazione di Label Box

Nello strumento di Roboflow, è possibile inserire più label box con lo stesso nome all'interno di ogni immagine. Questa funzionalità è utile perché vengono memorizzate tutte le classi utilizzate in modo da semplificare l'inserimento dei quadrati dentro agli altri frame del dataset creato.

Una volta assegnato il nome ad una label box, lo strumento memorizza la classe e la rende disponibile per la classificazione. Nei seguenti frame le classi utilizzate precedentemente saranno disponibili per essere nuovamente selezionate. Questo evita di dover reinserire manualmente ogni volta le stesse etichette.

Le immagini inserite con le classi annotate passeranno alla sezione “dataset” in cui verranno divise tra test, valid e train.

Dopo aver aggiunto tutti i frame, si genera una versione del database appena creato, in cui roboflow andrà ad addestrare e a compilare il modello.

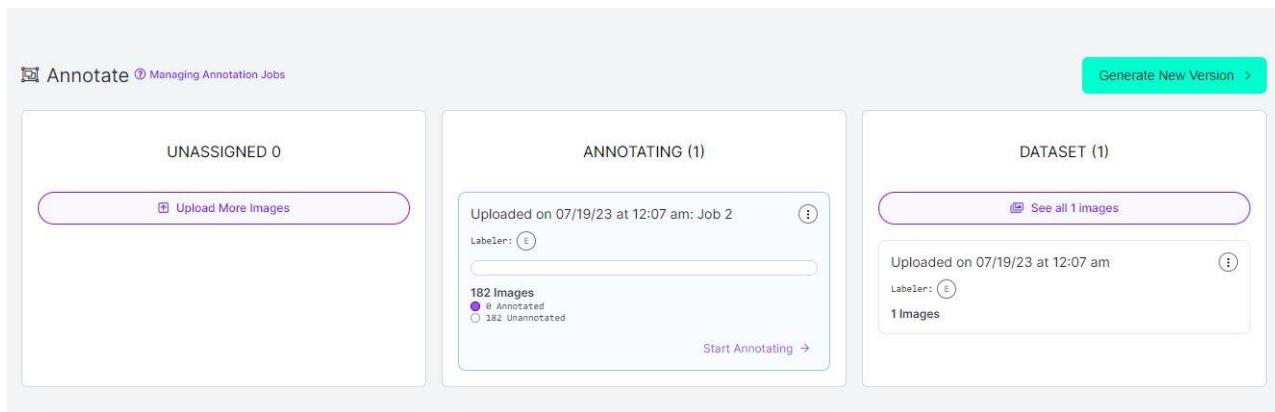


Figura 2.2 – Sezioni per addestrare il modello

L'organizzazione del database viene fatta nella sezione "dataset", dove le immagini annotate vengono suddivise in tre insiemi: test, validation e training.

- “Training”, È l'insieme di immagini utilizzato per addestrare il modello. Il modello analizza i frame e impara dai dati per identificare e riconoscere gli oggetti e le caratteristiche corrispondenti delle classi. Durante il training, il modello corregge i suoi parametri per migliorare la capacità di gestione delle predizioni;
- “Validation”, Dopo aver addestrato il modello nel Training, vanno valutate le sue prestazioni su dati che non ha mai visto, in modo da ottenere una stima obiettiva della capacità di generalizzazione. Questo è lo scopo principale del test di validazione. Il modello testa delle predizioni sulle immagini del “Validation” che vengono confrontate con le etichette inserite precedentemente. Questa valutazione permette di capire se il modello è in grado di generalizzare bene con nuovi dati.
- “Test”, una volta effettuato l'addestramento e la validazione, si utilizza il test per valutare le prestazioni. Contiene dati che il modello non ha mai visto. Si effettuano le predizioni sul test e se ne verifica la precisione per valutare l'affidabilità del modello.

Roboflow andrà a generare una nuova versione del database. Questa versione conterrà tutte le immagini con le rispettive etichette e suddivisioni tra training, validation e test.

2.2 Google Colab

Google Colab, è una piattaforma web di Google utilizzata per l'esecuzione del codice Python. La piattaforma offre l'utilizzo di risorse di calcolo, tra cui, CPU, GPU e TPU, senza l'obbligo di installare localmente software o librerie. Un vantaggio di questo applicativo è la possibilità di salvare in Google Drive i notebook di codice Python. Questo ha consentito l'archiviazione e l'accesso ai vecchi notebook in modo facile e veloce.

Perché si è utilizzato Google Colab?

Rilevare emissioni usando reti neurali convoluzionali con grandi quantità di immagini richiede tanta potenza di calcolo. Uno dei tanti vantaggi è l'accesso a librerie di “machine learning” già installate al suo interno. Google Colab è uno strumento efficace per esplorare e sviluppare modelli basati su reti CNN in modo rapido ed efficace.

2.2.1 Cos'è una GPU?

Una Graphics Processing Unit è un processore progettato per eseguire grandi operazioni di calcolo e rendering grafico. Sono state create per eseguire contemporaneamente operazioni di grafica e calcoli intensivi.

Le GPU nascono per accelerare il rendering grafico per i videogiochi o altre applicazioni grafiche. Sono molto sfruttate anche in ambito scientifico e nell'apprendimento automatico. Permettono di eseguire operazioni matematiche su larga scala, rendendo possibile l'utilizzo di applicazioni che richiedono numerosi calcoli. Le GPU nuove hanno più di mille core, rendendole il processore più potente in grado di elaborare in parallelo un numeroso flusso di dati. Google Colab una volta selezionato il runtime GPU, ci verrà assegnata una GPU NVIDIA Tesla K80 che viene utilizzata in condivisione con altre persone che la stanno sfruttando. L'unico svantaggio è che l'accesso al processore è garantito per una durata di solo 12 ore, con il vincolo che per effettuare diverse prove bisogna ricreare nuovamente il notebook.

Google Colab mette a disposizione una GPU remota per eseguire il codice lanciato. Essa è ospitata dai server di Google. Quando viene eseguito un comando il codice viene inviato al server che lo esegue restituendo i risultati sul browser. È un processo talmente veloce che l'utente utilizzatore non deve gestire direttamente il processore fisico.

2.2.2 Configurazioni iniziali

Nella sezione “Runtime” di Google Colab bisogna selezionare runtime con GPU, perché è in grado di eseguire operazioni su grandi quantità di dati e fare calcoli intensivi. Il grande vantaggio è dato dal fatto che la Graphics Processing Unit è in grado di eseguire i calcoli su più core con prestazioni superiori ad una CPU. È un passaggio fondamentale per velocizzare il processo di addestramento e creare un modello CNN completo in tempi brevi.

import os

Il modulo “os” in Python restituisce metodi che servono per interagire con il sistema operativo in cui viene lanciato il codice, permettendo la possibilità di gestire i file e navigare tra le directory. Questo modulo ha permesso una gestione migliore delle cartelle e dei file.

Import glob

L’importazione di glob è utile per la gestione dei file dentro al notebook. È utile quando si lavora con grandi quantitativi di dati o quando bisogna eseguire operazioni su più file.

from IPython.display import Image, display

La possibilità di visualizzare immagini contenenti risultati o il risultato del confusion matrix hanno evidenziato l’importanza dell’utilizzo di “Image” e “Display” dal modulo “IPython.display” permettendo la visualizzazione di video o frame all’interno del notebook. La funzione “Image” crea un oggetto immagine andando ad utilizzare il percorso del file. Con il codice Image(“esempio.jpg”) il notebook ci mostrerà l’immagine esempio salvata nella directory. La funzione “Display” viene usata per oggetti multimediali, per mostrare grafici, animazioni o molti altri contenuti. Questi 2 comandi sono stati utilizzati all’interno dell’ambiente di sviluppo di Google Colab per controllare immagini elaborate o risultati di predizioni.

!nvidia-smi

Consente di verificare se la GPU è attiva nel sistema e permette di vedere le sue caratteristiche. Questa stringa di codice (fornito per la GPU NVIDIA) permette di ottenere informazioni sulla Graphics Processing Unit.

Una volta lanciato questo comando viene riprodotto un output di questo tipo:

```
Tue Jul 25 20:17:45 2023
+-----+
| NVIDIA-SMI 525.105.17    Driver Version: 525.105.17    CUDA Version: 12.0    |
+-----+-----+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|====+=====+====+=====+=====+
|  0  NVIDIA A100-SXM...  Off     | 00000000:00:04.0 Off |             0         |
| N/A   30C    P0      42W / 400W |  0MiB / 40960MiB |      0%    Default   |
|                               |                      | Disabled  MIG M.     |
+-----+-----+-----+-----+-----+

+-----+
| Processes:                                     |
|  GPU   GI    CI          PID    Type    Process name                  GPU Memory |
|        ID    ID                                   Usage          |
|-----+-----+-----+-----+-----+
| No running processes found                    |
+-----+
```

Figura 2.3 – Risultato del comando “!nvidia-smi”

Fornisce un resoconto sulle specifiche della GPU utilizzata. L’output sopra generato permette di capire la temperatura (30°C), l’utilizzo della memoria (0MiB/40960MiB in totale), la versione del driver (525.105.17), l’utilizzo del GPU (0%) e il consumo di energia (42W su una quantità di 400W totali). Serve principalmente per monitorare le risorse utilizzate dall’unità grafica.

HOME=*os.getcwd()*

Con questo comando andiamo ad assegnare alla variabile “HOME” la directory di lavoro, per garantire un riferimento ad essa più velocemente nell’esecuzione dei prossimi comandi. Risulta molto utile per specificare percorsi di cartelle relativi alla loro posizione nel notebook.

print(HOME)

In seguito all’utilizzo del codice precedente, è stato lanciato un *print(HOME)*, che ha restituito “/content” il percorso della directory principale. Indipendentemente dal percorso della sessione la variabile “HOME” permette di riferirsi alla directory salvata.


```
!pip install ultralytics==8.0.0
```

Questo comando serve per l'installazione della versione “8.0.0” di ultralytics. È stato deciso di utilizzare questa versione perché è stabile e garantisce pochi problemi di bug e compatibilità. Essendo meno recente ma stabile la documentazione è dettagliata.

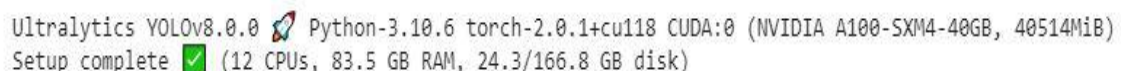
Il comando pip, quando utilizzato con “!”, esegue un'istruzione shell all'interno di Google Colab. “pip” è per definizione il package manager di “Python”, esso viene utilizzato principalmente per installare pacchetti o librerie esterne.

```
import ultralytics
```

Con questa stringa importiamo nel notebook Ultralytics. È una libreria utilizzata per l'addestramento di modelli CNN. Una delle funzionalità è il supporto per la rilevazione di oggetti da immagini, all'interno di EcoSentinel è stato utilizzato YOLO (You Only Look Once). Questa libreria fornisce un insieme completo di strumenti per testare e visionare modelli di deep-learning su immagini.

```
ultralytics.checks()
```

Inseguito all'import della libreria ultralytics bisogna verificarne la corretta importazione e il corretto funzionamento. Se la stringa produce come output “Setup complete” allora l'installazione è andata a buon fine.



```
Ultralytics YOLOv8.0.0 Python-3.10.6 torch-2.0.1+cu118 CUDA:0 (NVIDIA A100-SXM4-40GB, 40514MiB)  
Setup complete (12 CPUs, 83.5 GB RAM, 24.3/166.8 GB disk)
```

Figura 2.4 – Corretta installazione Ultralytics versione 8.0.0

```
!mkdir {HOME}/datasets
```

Il comando “!mkdir” permette la creazione di una nuova cartella chiamata “datasets” all'interno della directory principale denominata con la variabile “HOME”. Una volta eseguita se si entra in esplora file di Colab, la cartella che prima si chiamava “sample_data” cambierà nome in “datasets”.

```
%cd {HOME}/datasets
```

Una volta creata la directory sopra riportata, con questo comando navighiamo al suo interno creando e integrando le immagini presenti nel dataset di Roboflow. Per assicurarsi di essere all'interno della cartella corretta eseguiamo anche questo codice “!pwd” che ritornerà come risultato “/content/datasets”, percorso completo della cartella in cui ci si trova.

2.2.3 Codice per addestrare la CNN

```
!pip install roboflow
```

È stato utilizzato per l'installazione della libreria Roboflow all'interno del notebook. Una volta installato possiamo accedere ad una varietà di funzionalità per l'elaborazione di immagini.

Con l'installazione potremo richiamare il dataset di frame in cui sono state definite le classi per la rilevazione di emissioni

```
from roboflow import Roboflow
```

```
rf = Roboflow(api_key="xpmRVB0BhLhGDBciUa1e")
```

```
project = rf.workspace("emilio-verri-oxwqo").project("ecosentinel")
```

```
dataset = project.version(1).download("yolov5")
```

Una volta installato Roboflow andiamo ad importare dal modulo roboflow la classe Roboflow. In seguito viene creato un oggetto “rf” con la classe Roboflow che ha come parametro “api_key”. È una sottospecie di password che permette a Google Colab di accedere alle risorse protette da Roboflow, così da poter recuperare le immagini che sono state classificate. Successivamente bisogna individuare l'account una volta che si ha avuto accesso ai file protetti.

Si crea un oggetto “project” che va a richiamare 2 metodi:

- “workspace()”, individua l'utente con nome “emilio-verri-oxwqo” in Roboflow;
- “project()”, recupera il progetto contenente le immagini di EcoSentinel nell'account utilizzato per addestrare il modello.

L'ultimo oggetto che viene creato è "dataset". Andrà a scaricare la versione 1 del progetto EcoSentinel utilizzando il modello yolov5.

Il risultato ottenuto sarà una directory principale chiamata datasets contenente una folder chiamata con il nome del progetto, sotto la quale ci sono 3 cartelle, test, valid e train contenente le immagini classificate in Roboflow. Il notebook avrà al suo interno anche un file "data.yaml". È un file di configurazione all'interno del quale sono presenti la descrizione delle classi e il percorso delle 3 cartelle sopra indicate.

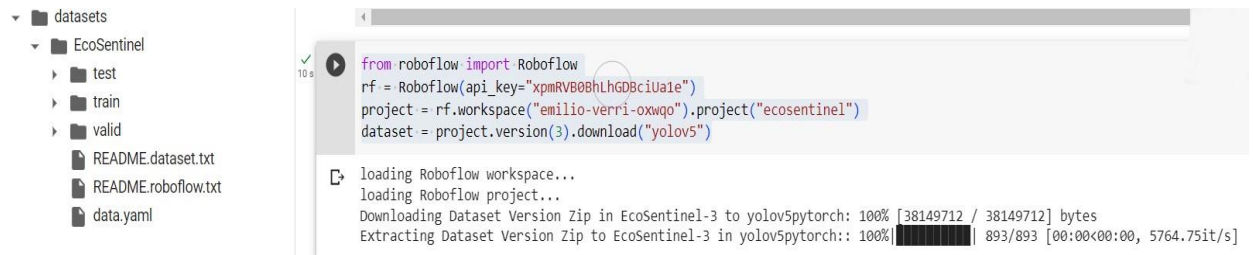


Figura 2.5 – Importazione immagini da Roboflow

Verranno importati anche 2 README file contenenti le informazioni del modello addestrato.

`%cd {HOME}`

Con questo comando torniamo alla directory principale in cui verrà eseguito il codice che addestrerà il modello.

`!yolo task=detect mode=train v5loader=True model=yolov8m.pt data=/content/datasets/EcoSentinel/data.yaml`

Lanciando questo programma viene inizializzato l'addestramento di un modello YOLO. Viene specificato il percorso del file "data.yaml" contenente le informazioni sul dataset utilizzato, sulle classi, sulle immagini e i percorsi.

Starting training for 100 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/100	6.75G	3.468	4.476	4.221	18	640: 100% 25/25 [00:10<00:00, 2.36it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 2/2 [00:02<00:00, 1.12s/it]
	all	49	89	0.00187	0.358	0.0035 0.00108

Figura 2.6 – Inizializzazione di training del modello

L'output restituito rappresenta il progresso nell'addestramento di un modello espresso in epoche su Google Colab utilizzando il framework YOLO.

Le epoche sono il numero di volte in cui il database caricato viene presentato al modello durante l'addestramento. Ad ogni epoca che passa il modello viene posto davanti ai dati di addestramento e calcola le perdite per migliorare la capacità di effettuare predizioni sugli oggetti.

Per addestrare una rete CNN bisogna obbligatoriamente utilizzare delle epoche perché il modello deve apprendere i dati in modo graduale. Nelle prime epoche, come nell'esempio sopra riportato, la precisione sarà bassa poiché il processo per il riconoscimento di emissioni ambientali è appena iniziato. Più le epoche passano più il modello migliora le prestazioni, ottimizzando i pesi e analizzando gli errori commessi nelle predizioni. Le epoche devono avere un numero adeguato, non troppe per evitare un overfitting, non poche per evitare difficoltà nella classificazione di oggetti. All'interno di EcoSentinel sono state utilizzate 100 epoche, queste rappresentano il numero di "giri" che il modello effettua durante l'addestramento.

Nell'immagine sopra riportata sono presenti dati utili per ottenere l'efficacia nell'addestramento del modello. Ecco una spiegazione più dettagliata dell'output:

- Epoch, indica il numero di epoche completate. In questo caso l'addestramento ha completato la prima epoca di cento;
- Box_loss, cls_loss, dfl_loss, sono i valori delle perdite. Le "box_loss" rappresentano la capacità del modello di predire le coordinate delle bounding box, esse circondano gli oggetti all'interno di un'immagine. Se si ottiene un valore alto indica che il modello ha difficoltà nel predire la posizione degli oggetti. Le "cls_loss" indica quanto un modello impara a classificare gli oggetti dentro alla bounding box. Più si avrà un valore alto di "cls_loss" più avrà difficoltà nel classificare gli oggetti. Le "dfl_loss" rappresenta quanto sta imparando a predire le caratteristiche degli oggetti e un valore alto, presume come nelle altre componenti di perdita una difficoltà ad individuare i dettagli delle emissioni;
- GPU_mem, rappresenta l'utilizzo del processore grafico durante l'addestramento, che in questo caso è di 6,75GB;
- Instances, indica il numero di oggetti rilevati durante l'addestramento;

- Size, indica le dimensioni delle immagini durante l'addestramento, l'altezza e larghezza;
- Class, Images, Box(P), R, mAP50, mAP50-95: Sono le valutazioni delle prestazioni del modello durante il test su un sottoinsieme del database.

Una volta che la GPU di Google Colab ha eseguito l'ultimo comando e il modello ha finito l'addestramento completando il numero di epoche, verrà generata una cartella "runs", contenente immagini di schemi con i risultati e un file "best.py" che verrà utilizzato all'interno del codice Python.

I grafici possono essere visualizzati sul web-notebook attraverso questi comandi:

```
%cd {HOME}
```

```
Image (filename=f'{HOME}/runs/detect/train1/confusion_matrix.png', width=900)
```

```
%cd {HOME}
```

```
Image (filename=f'{HOME}/runs/detect/train1/results.png', width=900)
```

Il comando `Image (filename=f'{HOME}/runs/detect/train/confusion_matrix.png', width=900)` serve per ottenere un frame creato da YOLO del confusion matrix. È a tutti gli effetti una tabella suddivisa in sezioni, in cui è possibile valutare le capacità che sono state acquisite dal modello e verificare che abbia appreso le informazioni correttamente. Permette di classificare le predizioni positive e negative.

Mentre `(filename=f'{HOME}/runs/detect/train/results.png', width=900)` permette di visualizzare i risultati dell'addestramento. Fornisce un'immagine chiara del modello durante l'addestramento. Evidenzia le perdite subite, come è stato addestrato il modello e molte altre informazioni che hanno permesso l'analisi dei risultati.

2.2.4 Grafici e Confusion Matrix

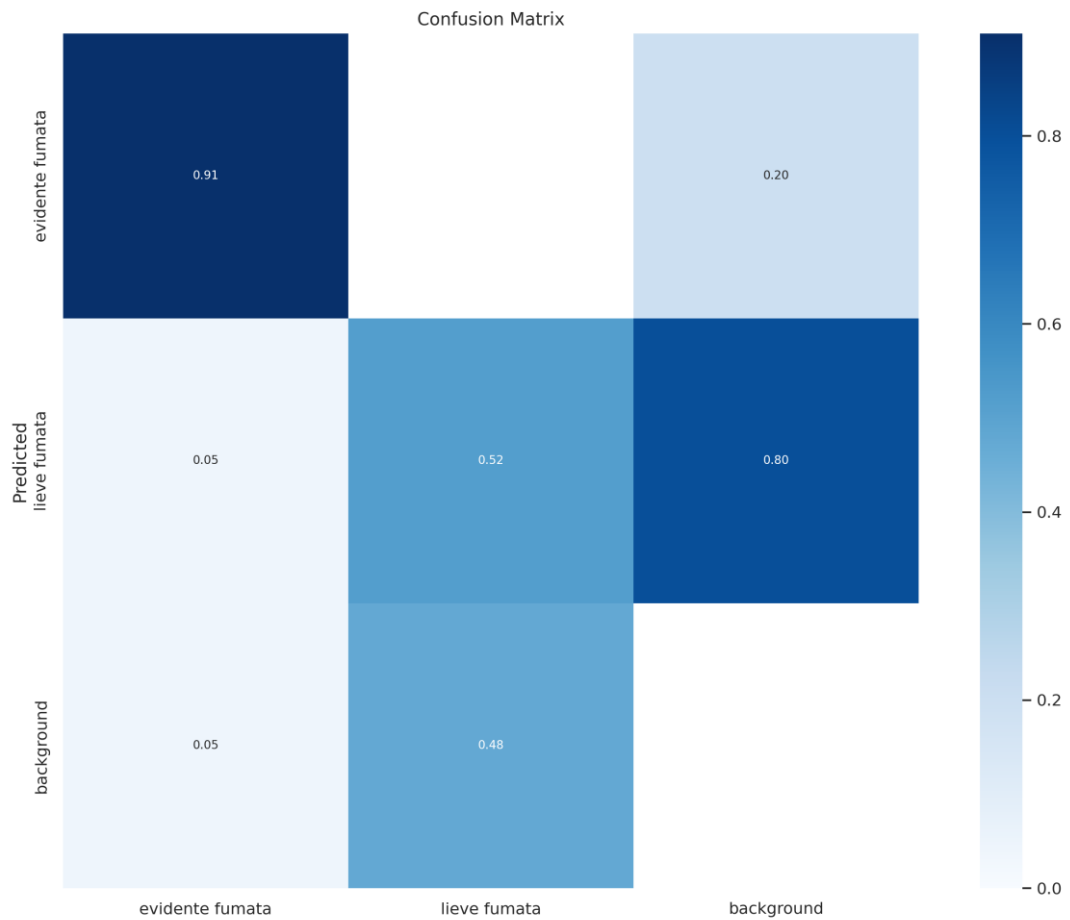


Figura 2.7 – Esempio Confusion Matrix

Questo è un esempio di confusion Matrix ottenuto con un dataset di 187 frame. Questa tabella deve essere letta partendo dall'alto verso il basso. Il modello è riuscito a riconoscere dalle immagini di test:

- 1) Evidente fumata, osservando il quadrato blu in alto a sinistra, indica che nel 91% dei casi il modello è riuscito a distinguere un'evidente fumata. Osservando il quadrato sottostante, indica che soltanto il 5% delle volte la rete CNN ha scambiato per lieve fumata un'evidente fumata e il quadrato successivo è rappresentato dal "background", indica che il 5% delle volte la rete convoluzionale ha erroneamente classificato il background come un'evidente fumata;
- 2) Lieve fumata, in questo caso il modello non ha mai confuso una lieve fumata con un'evidente fumata, però ha commesso degli errori perché il background risulta essere

del 48%, rendendo il modello impreciso perché è in grado di riconoscere una lieve fumata solo nel 52% dei casi;

- 3) Il background, è stato confuso per l'80% delle volte come lieve fumata e nel 20% per un'evidente fumata.

L'importanza di avere un dataset ben fornito di immagini, permette di ottenere un Confusion Matrix con valori alti nelle classi indicate e di avere un progetto completo in grado di rilevare gli oggetti correttamente.

Questa tabella è stata realizzata da YOLO, una volta finita l'esecuzione dell'addestramento su Google Colab.

È una matrice che restituisce i risultati ottenuti in seguito alle previsioni fatte dal modello rispetto alle classi presenti nei test. La matrice di confusione è stata molto utile perché ha permesso di aggiustare il tiro sul metodo di classificazione delle immagini, rendendolo più comprensibile all'addestramento.

Il Confusion Matrix è organizzato in questo modo:

Confusion Matrix:	<u>Classe Positiva</u>	<u>Classe Negativa</u>
<u>Previsione Positiva</u>	Positivo	Falso Positivo
<u>Previsione Negativa</u>	Falso Negativo	Negativo

Spiegazione della classificazione:

Falsi Positivi (FP): Sono il numero delle volte in cui il modello ha predetto che la classe fosse positiva quando in realtà è negativa;

Falsi Negativi (FN): Sono il numero delle volte in cui il modello ha predetto che la classe fosse negativa quando in realtà è positiva;

Positivi (TP): Rappresenta i casi in cui il modello è riuscito correttamente a prevedere la classe positiva;

Negativi (TN): Rappresenta i casi in cui il modello è riuscito correttamente a prevedere la classe negativa.

Attraverso l'utilizzo di questi parametri si possono calcolare dei valori che permettono una visione chiara del modello. È possibile calcolare:

- Accuratezza, rappresenta rispetto al totale dei casi la percentuale correttamente classificata. Quindi la somma tra Positivi e Negativi fratto la somma tra Positivi, Negativi, Falsi Positivi e Falsi Negativi.

$$\text{Accuratezza} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

- Precisione, rappresenta di tutti i test classificati come positivi la percentuale di TP (Positivi). Si calcola sottraendo ai Positivi la somma tra positivi e falsi positivi.

$$\text{Precisione} = \text{TP} / (\text{TP} + \text{FP})$$

- Sensibilità, è la percentuale di TP rispetto a tutti i casi positivi. Si ottiene sottraendo ai Positivi la somma tra Positivi e Falsi Negativi.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Nella classificazione di emissioni atmosferiche i dati forniti dal Confusion Matrix sono fondamentali per migliorare l'identificazione delle emissioni ed evitare che il programma finale rilevi erroneamente del fumo dove in realtà non è presente.

Per ottenere un modello funzionante, il numero di TP (Positivi) e TN (Negativi) deve essere maggiore al numero di FN (Falsi Negativi) e FP (Falsi Positivi)

$$\text{Modello Funzionante} = \text{TP} + \text{TN} > \text{FN} \text{ e } \text{FP}$$

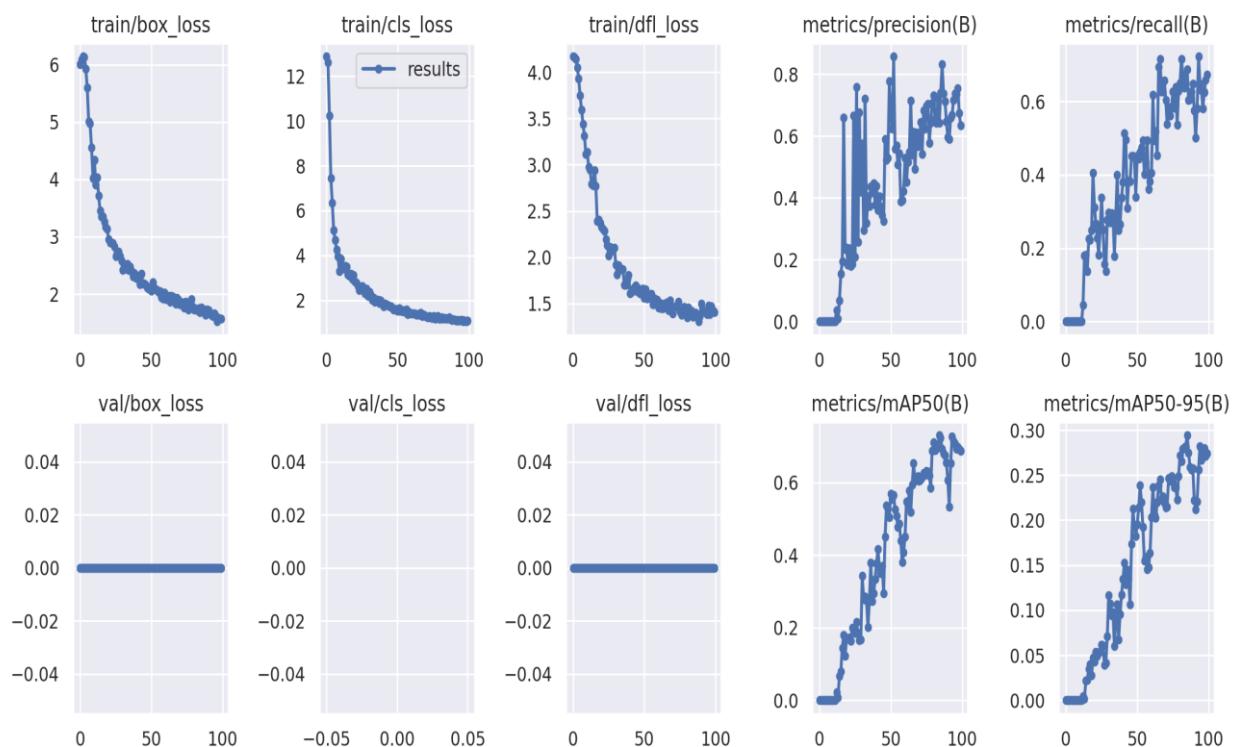


Figura 2.8 – Esempio di un Risultato

Questo è un esempio di risultato ottenuto inseguito all'addestramento di un modello con l'utilizzo di YOLO.

Le metriche che vengono specificate nella misurazione:

- Train/box_loss, rappresenta la perdita associata alla predizione delle classi.
- Train/cls_loss, indica la perdita legata alla classificazione degli oggetti.
- Train/dfl_loss, minimizzando questa perdita durante l'addestramento, si ottiene un modello in grado di individuare gli oggetti tenendo conto delle loro variazioni di forma e dimensione. È fondamentale per il rilevamento di emissioni in atmosfera che possono assumere forme diverse in breve tempo.
- Metrics/precision
- Metrics/recall(B)
- Val/box_loss
- Val/cls_loss
- Val/dfl_loss
- Metrics/mAP50(B)
- Metrics/mAP50-95(B)

Nei grafici è possibile visionare un miglioramento costante durante l'addestramento così da poter ottenere una rete CNN prestante.

Un grande insieme di frame utilizzati per il test, servono per gestire le variazioni di luce e delle posizioni, garantendo così alla CNN di valutare e individuare una grande quantità di informazioni, rendendola efficace nella rilevazione di oggetti in condizioni anche critiche.

L'istruzione del modello su un dataset corposo garantisce che la rete convoluzionale possa affrontare numerose situazioni reali.

3 Caso di studio sviluppato

3.1 Scenario applicativo

Il progetto consiste nell'addestrare una rete CNN per la rilevazione dei fumi classificati in tre categorie: importante fumata, evidente fumata e lieve fumata. Il dataset è stato integrato con Roboflow, il modello invece è stato addestrato attraverso l'utilizzo di Google Colab ed eseguito da Python. Le reti convoluzionali sono utilizzate per la rilevazione di immagini. Il vantaggio delle reti, risiede nell'apprendimento automatico, in quanto consente di rilevare specifiche zone e intensità delle emissioni.

Per la realizzazione di EcoSentinel sono stati fondamentali questi passaggi:

- La raccolta dei dati. Attraverso un database contenente immagini (1000 immagini) di emissioni classificate in base alla loro intensità;
- Suddivisione dei dati. Le immagini una volta classificate sono state suddivise. Il 70% all'addestramento, il 20% alla validazione e il 10% al test;
- Addestramento del modello. Il dataset è stato integrato all'interno del modello e attraverso il notebook di Google Colab è stato possibile iterare su 200 epoche, per far apprendere al modello le caratteristiche delle fumate;
- Controllo del modello generato. Attraverso l'utilizzo di grafici come il "confusion matrix", hanno garantito la possibilità di confrontare i risultati ottenuti e scegliere il modello migliore da utilizzare nel codice Python;
- Implementazione nel sito web. Una volta ottenuta una rete CNN ottimale è necessario integrarla nell'applicazione web utilizzando Flask.

Per ottenere risultati positivi, devono essere soddisfatti i seguenti requisiti:

- 1) Precisione;
- 2) Automazione;
- 3) Scalabilità;
- 4) Efficienza.

I vantaggi delle reti neurali convoluzionali:

- Il sistema è in grado di generalizzare nuove immagini di emissioni provenienti da fonti di terze parti;
- L'uso di reti CNN riduce la possibilità dell'errore umano, migliorando le valutazioni di un determinato evento;

- Se l'utente dispone di un computer e di una webcam di buona qualità, si possono ottenere risposte nella rilevazione in tempo reale. Il modello che utilizzato è molto pesante perché addestrato con 1000 frame e con 200 epoche;
- Un modello CNN per la rilevazione di emissioni garantisce l'utilizzo di meno risorse umane per il monitoraggio;
- La scalabilità è uno degli elementi chiave, perché se si necessita della rilevazione di ulteriori oggetti, il modello è semplicemente modificabile per garantire una maggiore capacità di elaborazione;
- Risultati in tempo reale permettono di perfezionare la rete e di aggiungere nuovi dati al DB;
- Grazie all'utilizzo di Python, il modello CNN può essere integrato con altri sistemi di sicurezza;
- Il monitoraggio ambientale non è l'unico settore a cui si applica la rilevazione di emissioni, ma può essere sfruttato anche da industrie che vogliono effettuare controlli sui fumi emessi.

3.2 Parte Software

Per la creazione di EcoSentinel si è scelto di utilizzare Python con YOLO.

Come ambiente di sviluppo si ha optato per PyCharm (sviluppato da JetBrains), la scelta è stata determinata dalla semplicità e dalle funzionalità specifiche per la programmazione Python.

È un IDE che permette l'utilizzo di strumenti di debugging interattivi per l'individuazione di errori e bug nel codice. PyCharm attraverso l'integrazione di un file chiamato "requirements.txt" contenente i valori qua sotto elencati:

ultralytics==8.0.26

numpy

matplotlib

ha installato autonomamente le librerie e i pacchetti descritti dentro al file txt.

PyCharm è compatibile con Git e GitHub che permettono il monitoraggio nelle modifiche del codice.

3.2.1 Struttura del Progetto

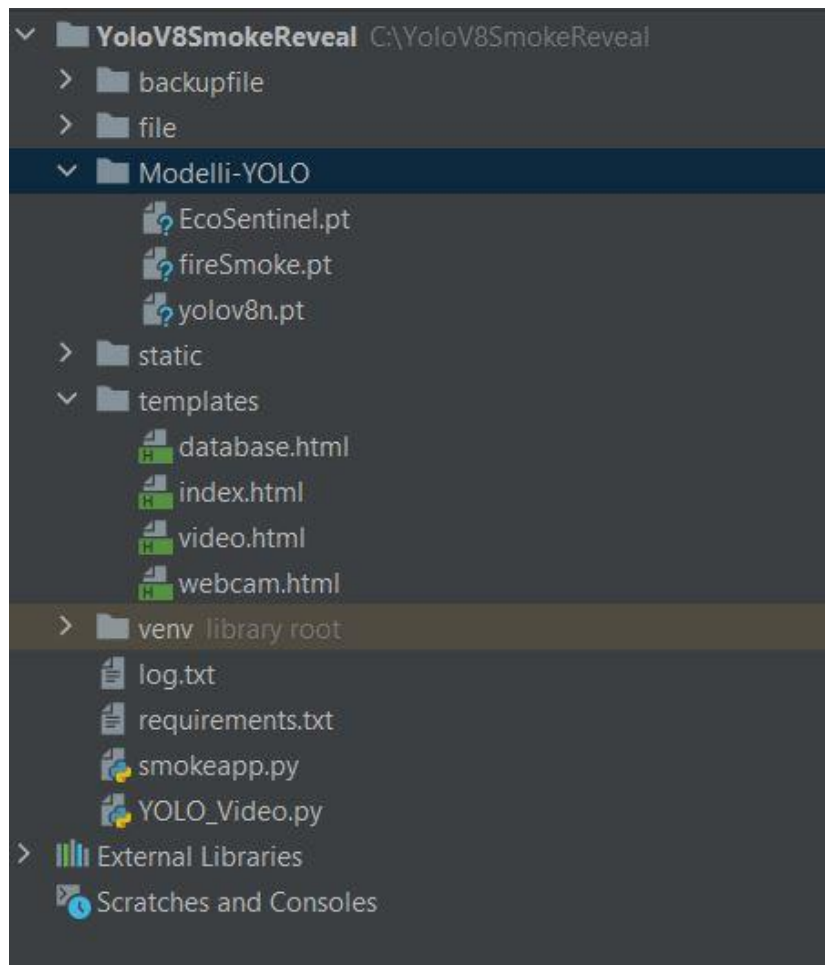


Figura 3.1 – Struttura del Progetto

Il progetto è strutturato in 4 sezioni:

- Il Front-End. Gestito nella cartella templates è composto da 4 file. “Index.html” contenente la HomePage della web-application, “Video.html”, rappresenta la sezione in cui l’utente può caricare un video pre-registrato, “Webcam.html” è la sezione dedicata alla rilevazione di fumi tramite una telecamera e “Database.html” in cui è possibile cancellare e visionare i log registrati;
- Il Back-End. Viene affidato ai 2 file “smokeapp.py” (contenente le directory del progetto) che comunica con “YOLO_Video.py” (funzioni principali);
- Il Modello. Nella folder “Modelli-YOLO” sono localizzati dei modelli in grado di rilevare emissioni in atmosfera (quello principale è soprannominato EcoSentinel.py);
- I Log. Per avere un monitoraggio completo della rilevazione di fumi. Ogni volta che il progetto catturerà delle emissioni verranno registrate nel file “log.txt”.

3.2.2 Back-End “YOLO_Video.py”

Questo file Python presenta al suo interno il codice per la rilevazione e il monitoraggio di emissioni in atmosfera, utilizzando il modello YOLO (You Only Look Once) pre-addestrato. Il codice presente in “YOLO_Video.py” ha come obiettivo di analizzare un video in input.

```
from ultralytics import YOLO
```

```
import cv2
```

```
import math
```

```
import datetime
```

Viene importata la classe YOLO dal modulo ultralytics. Si utilizza from in seguito ad import perché si è voluto importare solamente la classe YOLO. Import cv2 (OpenCV) è una libreria utilizzata per la gestione delle immagini.

È stato importato il modulo math, utilizzato per operazioni matematiche. Import datetime permette l'utilizzo di funzioni dedicate alla data e ora.

```
def video_detection(path_x):
```

```
    video_capture = path_x
```

```
    cap = cv2.VideoCapture(video_capture)
```

Viene definita la funzione “video_detection” e gli viene passato come parametro “path_x”, successivamente viene attribuito alla variabile “video_capture” il valore “path_x”.

In seguito è stato creato un oggetto “cap” per la cattura dei frame. “cv2.VideoCapture” è una classe di “OpenCV” che permette l’acquisizione delle immagini da una sorgente video.

Alla variabile “frame_width” è stato assegnato un valore intero, che rappresenta la larghezza del frame. Utilizzando il metodo “get” sull’oggetto “cap”, viene passato “3” come parametro alla funzione “get” (rappresenta la larghezza del frame).

Lo stesso funzionamento avviene per la variabile “frame_height”. In questo caso alla funzione “get” viene passato il parametro “4”, che rappresenta l’altezza dell’immagine.

```
model = YOLO("../Modelli-YOLO/EcoSentinel.pt")

classNames = ['evidente fumata', 'importante fumata', 'lieve fumata']

log_file_path="log.txt"
```

La classe “YOLO” è richiamata dal modulo “ultralytics” e attribuita all’oggetto “model”. All’interno di “YOLO” è presente il path, contenente la zona in cui è allocato il modello pre-addestrato per la rilevazione di emissioni.

Si attribuisce a “className” un elenco di classi, le stesse utilizzate durante l’addestramento con Google Colab. Mentre alla variabile “log_file_path” viene attribuito il percorso del file di log, in cui verranno inserite le informazioni sulle fumate rilevate, la data e l’ora.

```
while True:

    success, img = cap.read()

    results = model(img, stream=True)
```

“while True:” Dichiara un ciclo “while” continuo. Viene letto un frame del video attraverso “read()” dell’oggetto sopra spiegato “cap”. Il metodo restituisce due parametri, “img” che è il frame letto dal video e “success”, un valore booleano (cioè True or False), che indica se è avvenuta correttamente la lettura dell’immagine.

“stream=True” è un parametro per stampare il risultato. L’oggetto YOLO “model” viene applicato a “img” contenente l’immagine appena letta. Viene eseguita la ricerca degli oggetti presenti e i risultati di questa fase vengono salvati nella variabile “results”. In seguito si genera un ciclo infinito che legge continuamente i frame della videocamera.

```
for r in results:

    boxes = r.boxes
```

Fondamentale per il progetto è l’integrazione di questo ciclo for, che cicla ogni oggetto contenuto nella variabile “results” (contenente oggetti rilevati in un frame).

L’attributo “boxes” contiene i rettangoli degli oggetti rilevati. I rettangoli hanno quattro coordinate “x ,y angolo inferiore destro” e “x,y” angolo superiore sinistro.

for box in boxes:

```
x1, y1, x2, y2 = box.xyxy[0]
```

```
x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
```

Dentro al ciclo “for” sopra descritto, viene creato un altro ciclo che itera sui rettangoli definiti da “box” nell’oggetto “boxes”. L’attributo di “box”, “xyxy” contiene un array con quattro valori “x_max, x_min, y_max, y_min”. Questo permette di estrarre le coordinate dell’angolo in basso a destra del rettangolo “x2, y2” e dell’angolo in alto a sinistra “x1, y1”.

Le coordinate x1,y1,x2,y2 vengono successivamente convertite in valori interi, utilizzando la funzione “int()”. La conversione in valori interi è importante, così da garantire l’utilizzo del risultato trovato per operazioni successive.

```
cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 3)
```

```
conf = math.ceil((box.conf[0] * 100)) / 100
```

```
cls = int(box.cls[0])
```

```
class_name = classNames[cls]
```

Utilizzando OpenCv viene inserito un rettangolo attorno all’oggetto “img” con la funzione “rectangle()”. I parametri passati alla funzione sono “x1,y1”, “x2,y2”, “3” che indicano lo spessore del rettangolo e il colore blu rappresentato da (255,0,0).

In seguito avviene l’estrazione dell’affidabilità dalla variabile “box”. È importante sapere la sicurezza che ha il modello nella rilevazione dell’oggetto. Con la funzione “math.ceil” viene arrotondato il valore al centesimo.

“cls = int(box.cls[0])”, estrae il nome dell’oggetto rilevato dalla variabile box e lo salva nella variabile “cls”. Indica la classe che è stata rilevata. Nel progetto sono presenti 3 classi, “importante fumata”, “evidente fumata” e “lieve fumata”.

Nell’oggetto “classNames” avviene la ricerca del valore presente nella variabile “cls”, che ritorna il nome della classe rilevata. Il valore ottenuto, verrà salvato nella variabile “class_name”. L’obiettivo è quello di creare un tag sopra l’immagine.

```

label = f'{class_name}{conf}'

date_time_str = datetime.datetime.now().strftime("%d-%m-%Y %H:%M:%S")

with open(log_file_path, "a") as log_file:

    log_file.write(f'{class_name}, {date_time_str}\n')

```

Per inserire le variabili da stampare sopra il rettangolo dentro ad una stringa, utilizzo “f-string”. Viene generata una variabile stringa di nome “label” rappresentante l’etichetta che verrà mostrata. Nel tag sono presenti la confidenza dell’oggetto “conf” e il nome della classe “class_name”. Per ottenere il giorno, il mese, l’anno, l’ora, il minuto e il secondo attuale utilizzo la funzione “datetime.datetime.now().strftime("%d-%m-%Y %H:%M:%S")” e viene salvato il risultato ottenuto dentro alla variabile “date_time_str”.

Una volta acquisite le informazioni temporali, bisogna scriverle dentro al file di log in modalità “a” (consente di aggiungere nuove righe di testo al file senza andare a cancellare il contenuto già presente).

È presente un blocco “with” che permette l’apertura del file di log, contenuto nel percorso specificato dalla variabile “log_file_path”. Il file “log.txt” viene aperto come oggetto “log_file”. La riga successiva con la funzione “write()” scrive una nuova riga nel file di log, contenente il nome della classe, la data e l’ora del momento in cui l’oggetto è stato rilevato. Aggiungendo “\n”, viene mandato a capo il cursore.

```

t_size = cv2.getTextSize(label, 0, fontScale=1, thickness=2)[0]

c2 = x1 + t_size[0], y1 - t_size[1] - 3

```

Nella prima riga con la funzione “cv2.getTextSize” viene calcolata la dimensione del testo. Nella funzione viene passato il parametro “label” sopra indicato, il parametro “0” (font predefinito), “fontScale” la grandezza del font, “thickness” lo spessore del testo. Molto importante è “[0]” che estrae la larghezza del testo. Nella seconda parte di codice vengono calcolate le coordinate per disegnare un rettangolo sotto la “label”.


```

cv2.rectangle(img, (x1, y1), c2, [255, 0, 255], -1, cv2.LINE_AA)

cv2.putText(img, label, (x1, y1 - 2), 0, 1, [255, 255, 255], thickness=1,
lineType=cv2.LINE_AA)

yield img

cv2.destroyAllWindows()

```

“cv2.rectangle”, serve per disegnare un rettangolo di colore blu sotto al tag dell’immagine “img”. Le coordinate (c2_x, c2_y) rappresentano l'angolo in basso a destra, mentre (x1, y1) rappresentano l'angolo in alto a sinistra. Il parametro “-1” indica che il rettangolo deve essere riempito mentre “cv2.LINE_AA” rappresenta il tipo di linea.

Dentro a “cv2.putText” viene inserita l’etichetta (tag) nell’immagine “img”.

“yield img” restituisce il frame in seguito ad aver subito la modifica dell’etichetta e del rettangolo. Per terminare il programma si è deciso di inserire “cv2.destroyAllWindows()” che chiude le finestre di visualizzazione.



Figura 3.2 - Esempio rilevamento Evidente Fumata

3.2.3 Back-End “smokeapp.py”

```
from flask import Flask, render_template, Response, jsonify, request, session  
  
from flask_wtf import FlaskForm  
  
from wtforms import FileField, SubmitField, StringField, DecimalRangeField,  
IntegerRangeField  
  
from werkzeug.utils import secure_filename  
  
from wtforms.validators import InputRequired, NumberRange  
  
import os  
  
import cv2  
  
from YOLO_Video import video_detection
```

Come in ogni file Python vengono importati i moduli e le classi necessari per lo sviluppo del progetto. Questa parte di codice serve per creare un’applicazione web attraverso l’utilizzo del framework Flask, per analizzare e rilevare oggetti tramite il modello YOLO. In questa sezione sono definiti i vari reindirizzamenti alle funzioni dell’applicazione web.

- Flask, è un framework utilizzato per realizzare un’applicazione web utilizzando Python (framework = sono strumenti, librerie e moduli che permettono lo sviluppo di applicazioni semplificando compiti comuni);
- Render_template, consente l’utilizzo di template HTML;
- Response, per gestire le risposte http. È necessario questo modulo perchè fornisce la classe Response;
- Jsonify, consente l’utilizzo della funzione jsonify che permette di convertire in formato JSON le risposte http;
- Request, consente l’accesso a maggiori informazioni sulle richieste http, è stato utile per valutare errori;
- Session, ogni qualvolta che l’utente entra nell’applicazione si genera una sessione e questo modulo permette di gestirla;
- FlaskForm, nella sezione video è importante l’utilizzo di un form. Deve esserci una richiesta POST contenente il video caricato dall’utente perché venga analizzato dal

programma. Permette l'importazione di una classe per la creazione di form con l'utilizzo di Flask;

- `FileField`, è indispensabile per permettere all'utente di caricare i file;
- `SubmitField`, serve per inviare i dati inseriti dall'utente nel form situato nella sezione "video.html";
- `StringField`, `DecimalRangeField`, `IntegerRangeField`, sono campi che vengono utilizzati per creare gli input delle form presenti nelle applicazioni web;
- `Secure_filename`, per evitare problematiche relative alla sicurezza è necessario integrare la funzione "secure_filename" e gestire in modo sicuro i nomi dei file;
- `InputRequired`, `NumberRange`, validatori dei form di contatto, quello maggiormente utilizzato è "InputRequired", in grado di verificare se un campo è vuoto;
- `Os`, come già spiegato nella sezione di Google Colab, serve per la gestione delle path dei file;
- `Cv2`, permette di importare il modulo OpenCV;
- `Video_detection`, per utilizzare la funzione definita dentro a "YOLO_Video.py", bisogna richiamarla in questo modo "from YOLO_Video import video_detection". Funzione che permette il rilevamento degli oggetti utilizzando il modello You Only Look Once (YOLO).

```
app = Flask(__name__)

app.config['SECRET_KEY'] = 'ProvaPY!'

app.config['UPLOAD_FOLDER'] = 'file/videos'

class UploadFileForm(FlaskForm):

    file = FileField("File", validators=[InputRequired()])

    submit = SubmitField("Run")
```

Con l'oggetto `app` viene definita l'istanza dell'app Flask. Successivamente è stata creata una chiave segreta per l'applicazione, che garantisce una maggiore protezione per le sessioni. Definendo "Upload Folder" quando un utente caricherà i video nell'applicazione, verranno salvati nelle cartelle "file/video".

"UploadFileForm(FlaskForm)" rappresenta un form per caricare file

“class UploadFileForm(FlaskForm)”: Definisce una classe di FlaskForm chiamata “UploadFileForm”, che rappresenta un form per caricare file.

“file = FileField("File", validators=[InputRequired()])”, permette di garantire all’utente la possibilità di selezionare e caricare un file da qualunque dispositivo. Per questa validazione è importante definire “[InputRequired()]”.

Dentro alla classe sarà presente anche “SubmitField”, permette di creare un campo submit dentro al form. L’utilizzo di un pulsante d’invio serve per inviare i dati al server.

```
def generate_frames(path_x=""):
    yolo_output = video_detection(path_x)
    for detection_ in yolo_output:
        ref, buffer = cv2.imencode('.jpg', detection_)
        frame = buffer.tobytes()
        yield (b'--frame\r\nb'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

In questa parte di codice viene definita la funzione “generate_frames” responsabile dell’elaborazione dei frame del video.

La funzione accetta come parametro “path_x” che rappresenta il percorso del video. Salva dentro dalla variabile “yolo_output” le informazioni richiamate dalla funzione “video_detection” a cui viene passato il percorso.

“for detection_ in yolo_output” permette di eseguire un ciclo continuo che Itera sui frame restituiti dalla funzione “video_detection”.

Successivamente andando ad utilizzare il modulo cv2, viene richiamata la funzione “imencode” che permette di analizzare il frame passato in “jpg”. Uno dei passaggi fondamentali è la conversione del frame codificandolo in un array di byte, attribuendo a buffer la funzione “tobytes()”. Tutto questo viene salvato in una variabile chiamata “frame”. Come elemento conclusivo di questa funzione utilizzo l’istruzione “yield”.

Per semplificare la visione in tempo reale in una pagina web, viene utilizzata quest’istruzione per permettere di restituire il frame come flusso multipart in formato “jpg”.

```
def generate_frames_web(path_x):

    yolo_output = video_detection(path_x)

    for detection_ in yolo_output:

        ref, buffer = cv2.imencode('.jpg', detection_)

        frame = buffer.tobytes()

        yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Prima di arrivare alla spiegazione della parte di navigazione, è stato necessario definire questa funzione. “generate_frames_web” è una funzione che andrà a generare i frame della webcam e non del video come “generate_frames”. Le 2 funzioni sono speculari cambia solo il modo in cui viene passato il parametro “path_x”.

- 1) “def generate_frames(path_x=)””, è progettato per gestire, generare e restituire i frame di un video caricato all’interno dell’applicativo web;
- 2) “def generate_frames_web(path_x)”, in questo caso “path_x” non indica un percorso di un video ma rappresenta la sorgente di una webcam.

Per quanto riguarda la parte di routing del sito web, è stato gestito in seguito ad aver definito queste 2 funzioni.

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

@app.route('/home', methods=['GET', 'POST'])

def home():

    session.clear()

    return render_template('index.html')
```

Cosa sono le route di un’applicazione web?

Le route permettono di definire un URL per ogni sezione del sito web. Per ogni indirizzo il sito adotterà un funzionamento diverso. In una pagina l’utente potrà analizzare i video, nella seconda si accenderà la webcam e nella terza ci sarà la gestione del dataset. In questa parte di codice vengono definite 2 route “/” e “/home”. Accetta sia richieste “GET” e “POST”.

Si definisce una funzione “home()” associata agli “URL”. Quando un utente prova ad entrare nell’applicativo o vuole navigare verso la homepage questa funzione si attiverà. All’interno

di “home()” si trova la funzione “session.clear()”, che va a cancellare tutti i dati rimasti in sessione. Così da permettere all’utente ogni volta che naviga dentro al sito di avere una sessione vuota e quindi nessun dato memorizzato. Per restituire un file “.html” definito nella cartella “template” utilizzo “return render_template(‘index.html’)”.

return render_template('index.html'): Questa riga restituisce il contenuto del template chiamato "index.html".

```
@app.route("/webcam", methods=['GET', 'POST'])
```

```
def webcam():
```

```
    session.clear()
```

```
    return render_template('webcam.html')
```

Viene definita un’altra route esplicitata come “/webcam” e successivamente viene creata una funzione webcam() che restituirà come template “webcam.html”.

```
@app.route('/FrontPage', methods=['GET', 'POST'])
```

```
def front():
```

```
    form = UploadFileForm()
```

```
    if form.validate_on_submit():
```

```
        file = form.file.data
```

```
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)),
```

```
app.config['UPLOAD_FOLDER'], secure_filename(file.filename)))
```

```
        session['video_path'] = os.path.join(os.path.abspath(os.path.dirname(__file__)),
```

```
app.config['UPLOAD_FOLDER'], secure_filename(file.filename))
```

```
    return render_template('video.html', form=form)
```

Genera una route “FrontPage” come nei casi sopra spiegati. Se l’utente navigherà verso questo “URL”, verrà eseguita la funzione “front()”.

Definisco nella variabile “form” un’istanza della funzione “UploadFileForm”, che permetterà agli utenti di caricare un file. “if form.validate_on_submit():” se il video viene

caricato correttamente e quindi il form è valido entriamo il codice esegue il ciclo “if”. Nella variabile “file” recupero i dati del file caricato attraverso il form.

Con la funzione “save()” salvo il video nella cartella di destinazione, definita tra parentesi tonde.

Per salvare il file caricato viene utilizzata la sessione. Il percorso è salvato in “session[‘video_path’]” che verrà utilizzato per analizzare e gestire i frame del video.

“return render_template(‘video.html’, form=form)”, una volta eseguite le precedenti operazioni, restituirà il template “video.html” rendendo visibile il form per il caricamento dei file. Lato front-end:

```
{{form.hidden_tag()}}
```

```
{{form.file(class_="custom-file-input",type="file",width="100px")}}
```

All’interno dei 2 tag “<form>e </form>” sono state inserite 2 parti di codice.

Il primo rappresenta un campo nascosto, che serve per recuperare informazioni sul video caricato. Mentre il secondo pezzetto di codice rappresenta un campo input.

```

```

Utilizzata la funzione “url_for()”, che va a generare l’url verso la route “video”. Quando l’utente accederà a questa route verrà eseguita la seguente funzione:

```
@app.route('/video')
```

```
def video():
```

```
    return Response(generate_frames(path_x=session.get('video_path', None)),  
mimetype='multipart/x-mixed-replace; boundary=frame')
```

“return Response()” restituisce una risposta HTTP al client. Dentro “Response” richiamo la funzione “generate_frames” per ottenere le immagini del video. Nella funzione bisogna passare dei parametri, tra cui “path_x”, che rappresenta il percorso del video acquisto dalla sessione “video_path”.

```

```

La funzione “url_for()” si trova ulteriormente dentro al file “wecam.html”. Farà riferimento all’url “webapp”.

```
@app.route('/webapp')
```

```
def webapp():
```

```
    return Response(generate_frames_web(path_x=0),    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Come sopra indicato, viene eseguita “webapp()”, che sfrutterà la funzione “Response()”. Dentro richiama “generate_frames_web(path_x=0)” e permette di ottenere i frame della webcam live. In questo caso “path x=0” indica il percorso della webcam.

Mimetype specifica i dati restituiti dallo streaming, mentre “multipart/x-mixed-replace” permette l’invio dei frame consecutivamente e con boundry=frame indica come dividere i dati.

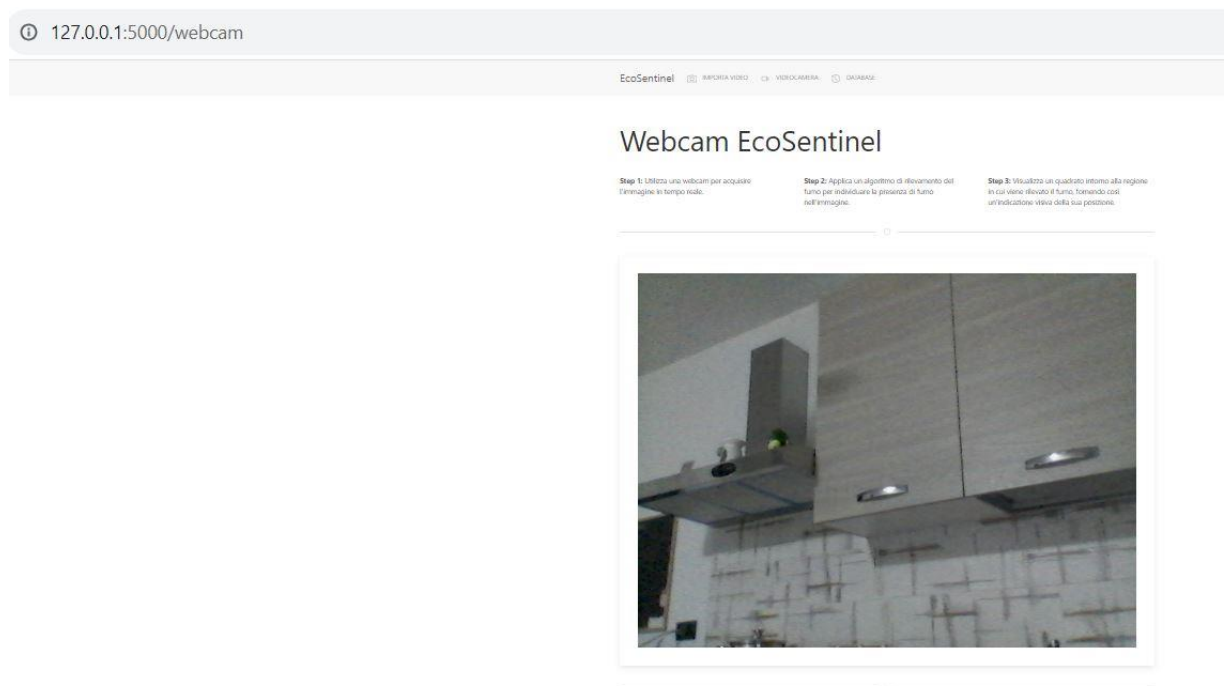


Figura 3.3 -Test live webcam su server locale

```
@app.route('/database', methods=['GET', 'POST'])
```

```
def database():
```

```
    session.clear()
```

```
    rows = []
```

```
    try:
```

```
        with open('log.txt', 'r') as file:
```



```

    for line in file:

        values = line.strip().split(' ')

        if len(values) == 2:

            rows.append(values)

    except FileNotFoundError:

        pass

    return render_template('database.html', rows=rows)

```

EcoSentinel è un progetto per il rilevamento di emissioni di fumo. Per permettere all'utente di verificare quando si è verificata un'emissione, è stato necessario introdurre una sezione dedicata al dataset, per garantire la raccolta delle informazioni. Una volta definita una route che punta all'url "/database", gestita dalla funzione "database()" (La route gestisce sia le richieste GET e POST). Viene creata una lista vuota "rows" popolata con i dati presenti nel file di log. In un blocco "try" viene eseguito "with open('log.txt','r') as file:", permette di aprire il file in modalità lettura "r".

Quando il blocco "with" sarà terminato, il file si chiuderà automaticamente. Successivamente è stato definito un ciclo che itera su ogni riga di "log.txt". Per estrarre i dati da ogni riga utilizzo "line.strip().split(' '):". Con un "if" viene verificato se dentro "values" sono presenti due valori. Nel caso in cui sia vero, sarà presente la data/ora e il nome della classe rilevata.

"rows.append(values):" se i requisiti sono soddisfatti, vengono aggiunti alla lista "rows" i valori. Per verificare la presenza del file "log.txt" sono state utilizzate delle eccezioni come "except FileNotFoundError:". In conclusione la funzione restituisce il template "database.html". Vengono passate alla pagina le informazioni presenti in "rows".

```

{% for row in rows %}

    <tr>

        <td>{{ row[0] }}</td>

        <td>{{ row[1] }}</td>

    </tr>

{% endfor %}

```

Nella pagina “database” viene ciclato su “row”, stampando un elenco con “row[0]”, il nome della classe rilevata e “row[0]”, la data/ora del rilevamento.

```
@app.route('/delete_logs', methods=['POST'])
```

```
def delete_logs():
```

```
    try:
```

```
        with open('log.txt', 'w') as file:
```

```
            file.write("")
```

```
        return "", 204
```

```
        # Risposta con successo (senza contenuto)
```

```
    except FileNotFoundError:
```

```
        return "", 404 # File non trovato
```

```
    except Exception as e:
```

```
        print('Errore durante la cancellazione dei log:', e)
```

```
        return "", 500 # Errore del server
```

Concessa all’utente la possibilità di cancellare i file di log direttamente dal template “database.html”.

Con “app.route” si genera una route “/delete_logs”, per la cancellazione delle informazioni presenti dentro al file “log.txt”.

Quando l’utente naviga verso questa sezione del sito, si attiva la funzione “delete_logs()” che apre il file in modalità scrittura “w” e con l’istruzione “file.write(‘’”, viene scritta una riga vuota, andando a cancellare tutti gli elementi che erano presenti.

La funzione viene eseguita da uno script “JavaScript” presente dentro a “database.html”:

```
<script>
```

```
    function deleteLogs() {
```

```
        fetch('/delete_logs', { method: 'POST' })
```

```
        .then(response => {
```

```
            if (response.ok) {
```

```

        // Ricarica la pagina dopo aver cancellato i log
        window.location.reload();

    } else {

        alert('Errore durante la cancellazione dei log.');
```

}

```

    })

    .catch(error => {

        console.error('Errore durante la cancellazione dei log:', error);

        alert('Errore durante la cancellazione dei log.');
```

});

```

    } </script>
```

Effettua una richiesta HTTP POST per cancellare i file di log.

In questo script sono presenti alcuni controlli per gestire la risposta della richiesta “POST” http. Se riceve una risposta in seguito alla richiesta fatta, la pagina web si aggiornerà. Altrimenti il programma stampa un “alert” contenente il messaggio di errore “Errore durante la cancellazione dei log”.

I valori nel file log possono accumulare una grande quantità di dati nel tempo. Per avere un’analisi pulita su EcoSentinel è garantita la possibilità all’utente di cancellare i dati di vecchie rilevazioni.

3.3 Problematiche Ricontrate

Nell'addestramento della CNN attraverso l'utilizzo di Google Colab sono state riscontrate problematiche nel riconoscimento dell'importante fumata.

3.3.1 Problemi con importante fumata

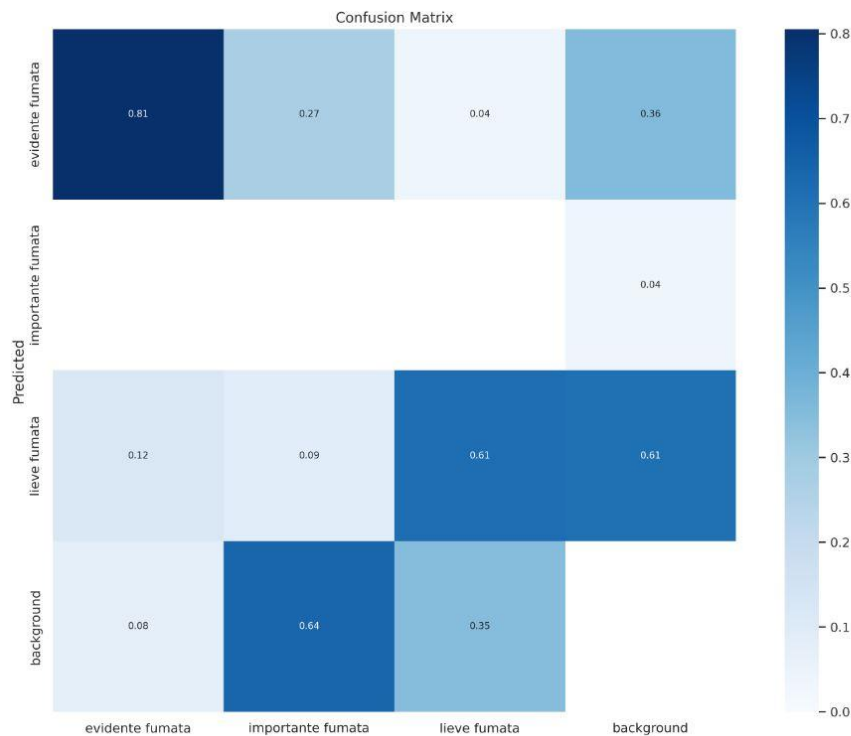


Figura 3.4 -Test 1086 immagini, con 200 epoche

Come riportato dal Confusion Matrix in figura, il test effettuato con 1086 immagini (70% al train, 20% alla validazione e 10% al test) ha riportato risultati negativi.

- Lieve Fumata, risulta riconosciuta al 61% come lieve fumata, al 35% come background e allo 0.04% come evidente fumata. Il risultato di questa classe è positivo perché la lieve fumata si confonde facilmente con il background.
- Evidente Fumata, risulta al 81% come evidente fumata, al 12% come lieve fumata e al 8% come background. Il risultato è più che positivo, il modello nell'81% dei casi è riuscito ad individuare la classe.
- Importante Fumata, risulta al 64% come background, al 9% come lieve fumata e al 27% come evidente fumata. Questa è una delle classi più importanti, perché evidenzia quando è presente un'emissione atmosferica che copre buona parte della visione della webcam (potrebbe indicare un incendio di grandi dimensioni).

Il modello in questo caso non è riuscito ad individuare neanche una volta, tra le immagini di test, un'importante fumata.

È importante sottolineare che nel modello, è necessario che tutte e 3 le classi abbiano dei risultati positivi.

Overfitting, modello troppo addestrato con le epoche. Quando viene utilizzato un numero eccessivo di epoche (come nella figura sopra, 200 epoche) la rete CNN incorre nel fenomeno dell'overfitting. Si verifica quando il modello è troppo addestrato sui dati a disposizione e perde la capacità di individuare le classi su nuovi dati forniti. Il modello impara a memoria le immagini del suo database e non è in grado di riconoscere nuovi frame.

Per verificare che non sia andato in overfit, è stata effettuata una prova con 100 epoche.

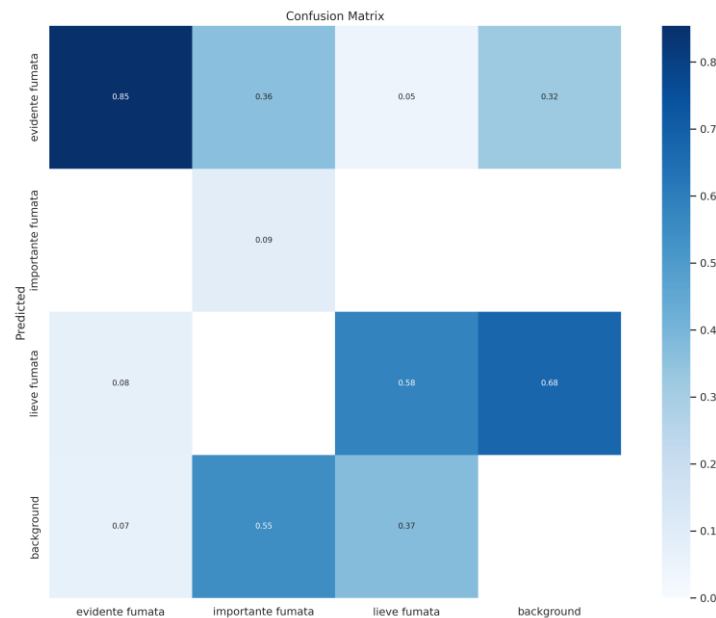


Figura 3.5 -Test 1086 immagini, con 100 epoche

Il modello presenta sempre la stessa problematica. Nonostante sia riuscito ad individuare nel 9% dei casi un importante fumata è un risultato scadente non compatibile con il risultato finale.

Vedendo un miglioramento nella rilevazione a 100 epoche, è stata effettuata una prova con 50 epoche ottenendo un risultato peggiore. Come illustrato qua sotto in figura, il modello è riuscito ad individuare anche a 50 epoche solo il 9% di importanti fumate.

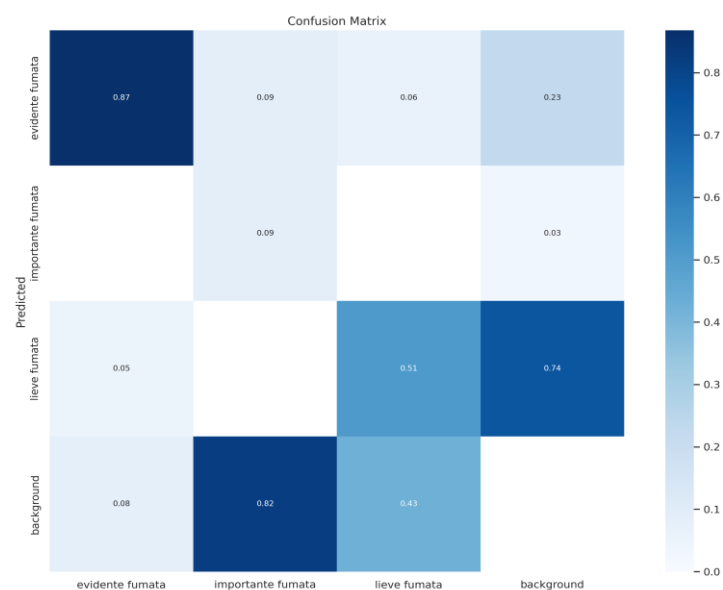


Figura 3.6 -Test 1086 immagini, con 50 epoche

3.3.2 Augmentations

Escludendo il problema dell'overfitting, l'unica soluzione è "Augmentations".

Una pratica utilizzata sui progetti di reti CNN nel caso in cui vengano riscontrati dei gravi problemi nell'addestramento. L'obiettivo è di aumentare la quantità di immagini del database, introducendo delle modifiche nei frame senza essere obbligati ad acquisire nuove foto. Le immagini possono essere ruotate di alcuni gradi, ribaltate orizzontalmente o verticalmente, ingrandite o rimpicciolite, possono subire dei cambiamenti nel contrasto e tagliate. Per scongiurare tutte le problematiche dovute all'overfit l'aumento dei dati è utile e permette di generare nuovi esempi di addestramento, senza dover manualmente acquisire e classificare nuove immagini.

Il progetto risulta più robusto e in grado di riconoscere gli oggetti in diverse condizioni. L'aumento dei dati deve essere applicato non eccessivamente. Troppe variazioni possono rendere i dati da addestrare poco realistici.

L'incremento delle immagini aumenta significativamente il tempo di addestramento del modello che genererà un file ".py" pesante. Un modello troppo grande rallenterà significativamente l'esecuzione del codice. È importante applicare questa pratica con molta attenzione e verificarne il corretto funzionamento.

Nel caso di EcoSentinel il database non è vario, per questo l'augmentation permette di apportare significativi miglioramenti nelle prestazioni.

AUGMENTATIONS

Outputs per training example: 3

Flip: Horizontal, Vertical

Crop: 0% Minimum Zoom, 30% Maximum Zoom

Shear: $\pm 15^\circ$ Horizontal, $\pm 15^\circ$ Vertical

Saturation: Between -30% and +30%

Exposure: Between -25% and +25%

Figura 3.7 – Augmentations Utilizzate con Roboflow

Roboflow fornisce uno strumento che permette di aggiungere l'Augmentations. Ogni aumento di dati ha generato 3 immagini per ogni frame del "training" e in questa versione del progetto sono stati utilizzati 5 aumenti di dati:

- 1) Flip: Horizontal, Vertical. Le immagini vengono ruotate Orizzontalmente e Verticalmente;
- 2) Crop: 0% Minimum Zoom, 30% Maximum Zoom. In questo caso le immagini vengono ingrandite di una percentuale che varia tra lo 0% e il 30%;
- 3) Shear: $\pm 15^\circ$ Horizontal, $\pm 15^\circ$ Vertical. Indica la deformazione dell'immagine ruotando i pixel lungo una specifica direzione;
- 4) Saturation: Between -30% and +30%. Indica l'alterazione dell'intensità dei colori che varierà da -30% a +30%;
- 5) Exposure: Between -25% and +25%. Con questa tecnica viene alterata l'esposizione dell'immagine, cioè la quantità di luce che la colpisce.

Con questa procedura le immagini per addestrare il modello sono aumentate notevolmente, passando da 1086 immagini a 2606, suddivise in 2280 "train", 220 "valid" e 106 "test". È importante assicurarsi che i set di validazione e test siano rappresentativi dell'emissione di fumo. Se si ottiene come risultati problemi di overfitting o underfitting durante l'addestramento, bisogna riconsiderare la dimensione dei set di dati.

La scelta delle dimensioni del database è basata sull'esperienza nell'addestramento dei modelli. Grazie all'interfaccia user friendly di Roboflow è stato possibile applicare l'augmentation e migliorare la capacità di individuazione delle 3 classi di fumate.

Per applicare questa pratica bisogna classificare le immagini e al momento della generazione della versione del database, bisogna andare a scegliere le modifiche alle immagini da applicare.

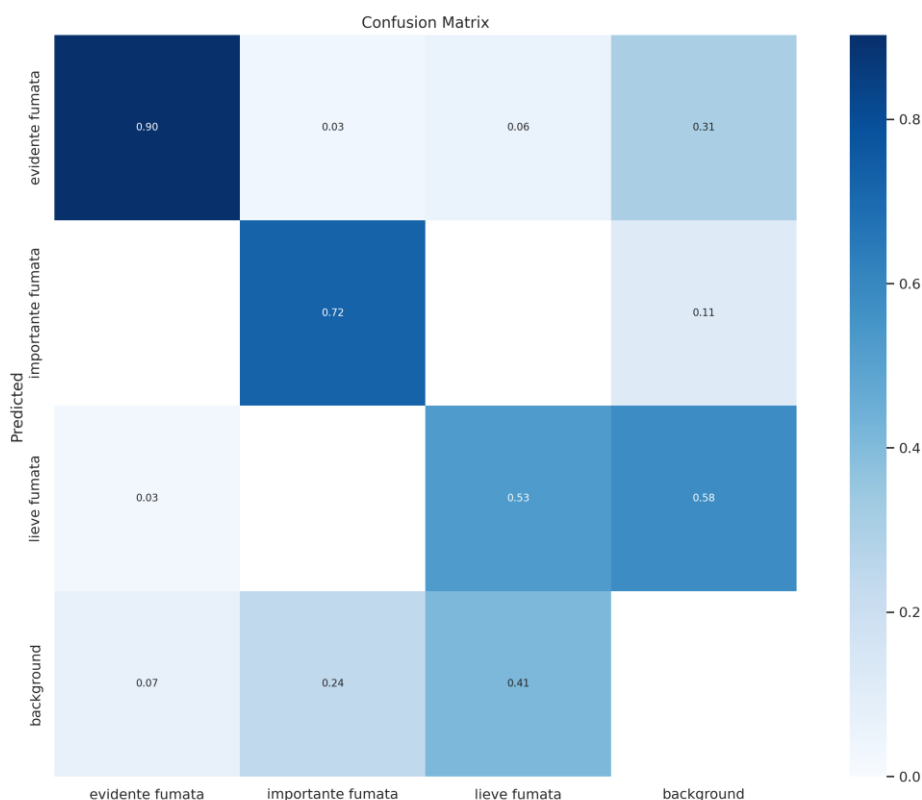


Figura 3.8 – Test 50 epoche con 2606 immagini e Augmentations

I risultati sono cambiati drasticamente in positivo. Il modello è riuscito a riconoscere:

- Lieve Fumata, nel 53% dei casi una lieve fumata. Risultato positivo, perché può esserci della confusione con il background, che in questo caso risulta del 41% dei casi.
- Evidente Fumata, nel 90% dei casi un evidente fumata. Evidenzia un modello in grado di percepire delle emissioni evidenti ad occhio. C'è stata una confusione del 7% sul background e del 3% sulla lieve fumata.
- Importante Fumata, in seguito all'utilizzo dell'augmentations, nel 72% dei casi il progetto è in grado di individuare un importante fumata. La confusione del 24% è dovuta alla presenza di nuvole in atmosfera che vengono rilevate come un'emissione.

Il risultato garantisce la possibilità di condurre un test fisico in modo efficace. Questo è possibile perché il modello è stato reso più preciso nella capacità di rilevare emissioni in atmosfera. La migliore capacità di generalizzazione e la gestione delle variazioni ambientali lo rendono efficace nella classificazione dei fumi.

L'augmentation ha contribuito a creare un database completo che ha permesso di addestrare un modello affidabile.

3.4 Test Rete CNN

3.4.1 Raspberry

Il test per verificare il funzionamento del modello è stato effettuato presso l'azienda Fusine Energia S.r.l – Centrale di Fusine, specializzata nella produzione di energia attraverso la combustione del cippato. Il test è stato effettuato utilizzando un Raspberry Pi 3 Model B.

Cos'è un Raspberry?

È a tutti gli effetti un piccolo computer utilizzato per progetti IoT. Il Raspberry Pi 3 Model B è dotato di un processore quad-core ARM Cortex-A53 con clock a 1,2 GHz e di 1 GB di RAM necessario per essere utilizzato dal programma. Importante da sottolineare che è provvisto di 4 porte USB, 1 porta Ethernet e una porta HDMI per l'uscita video. Questo dispositivo è alimentato attraverso una micro-USB.

Per il progetto EcoSentinel, il raspberry è stato dotato di un impianto per il raffreddamento composto da una ventola compresa di dissipatori GeekPi, destinata al raffreddamento della CPU e una telecamera AZDelivery da 5 Megapixel per consentire la visualizzazione delle immagini live. Per garantire un raffreddamento migliore è stata acquistata una custodia protettiva in alluminio, che permette il ricircolo dell'aria interno, così da evitare surriscaldamenti sui componenti interni.

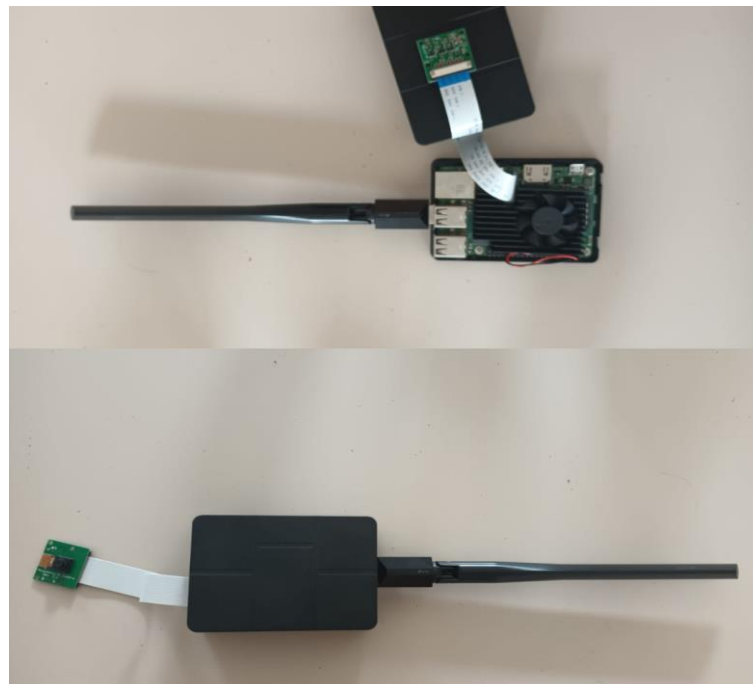



Figura 3.9 – Raspberry Pi 3 Model B

3.4.2 Prove Effettuate

Il 27/09/2023 è stato condotto con successo il test del raspberry presso la centrale Fusine Energia S.r.l. Il test è stato avviato alle 9:30 del mattino ed è durato per un totale di 4 ore, durante le quali sono stati registrati dati significativi.

Il dispositivo è stato posizionato su un tetto con vista sulle torri di raffreddamento. All'inizio delle rilevazioni, il Raspberry ha segnalato una notevole quantitativo di emissioni dalla torre di raffreddamento. Come si può notare dai dati qua sotto riportati, già da subito il tool ha sottolineato un importante fumata e un evidente fumata. Durante il test, il dispositivo ha continuato a monitorare attentamente la situazione e a raccogliere dati in tempo reale.

Storico rilevazione di fumi



ELEMENTO RILEVATO	DATA E ORA DELLA RILEVAZIONE
importante fumata	27-09-2023 09:33:49
evidente fumata	27-09-2023 09:33:50
importante fumata	27-09-2023 09:33:51
evidente fumata	27-09-2023 09:33:52
evidente fumata	27-09-2023 09:33:55
evidente fumata	27-09-2023 09:33:56
importante fumata	27-09-2023 09:33:57
importante fumata	27-09-2023 09:33:58
importante fumata	27-09-2023 09:33:59
importante fumata	27-09-2023 09:34:00
evidente fumata	27-09-2023 09:34:01
evidente fumata	27-09-2023 09:34:02
evidente fumata	27-09-2023 09:34:03
importante fumata	27-09-2023 09:34:04
importante fumata	27-09-2023 09:34:05

Figura 3.10 – Storico dati raccolti presso Fusine Energia S.r.l

I risultati del test sono positivi, sebbene sia emerso che la potenza di un Raspberry Pi 3 Model B è inferiore rispetto al modello Pi 4, perché ha comportato alcune limitazioni nelle misurazioni. Durante l'intera sessione di test, il dispositivo ha operato in modo stabile, tuttavia, alcune lacune nei dati sono emerse a causa della sua capacità di elaborazione limitata. Un vuoto significativo si è verificato alle 9:57 del mattino, quando il dispositivo ha interrotto la registrazione del filmato per un periodo di un minuto. Questo intervallo di tempo

è stato principalmente causato da un fattore principali, il tool ha mostrato segni di surriscaldamento durante questa fase, il che ha influenzato negativamente le sue prestazioni.

Per evitare il rischio di perdere dati durante le interruzioni, è stato adottato un approccio di sicurezza aggiuntivo con l'utilizzo di un telefono, che è stato posizionato nelle vicinanze per effettuare una registrazione di backup in caso di malfunzionamento o surriscaldamento del Raspberry Pi.

Dalle 11:00 in poi, il prosieguo della rilevazione è risultato vano a causa dell'aumento della temperatura ambiente. Di conseguenza, la durata effettiva del test si è limitata a un totale di 2 ore, dalle 9:30 alle 11:30 del mattino.



Figura 3.11 – Raspberry e telefono di Backup

Per eseguire il codice del tool durante il test, è stato necessario utilizzare un monitor e un mouse per consentire una gestione diretta del dispositivo.

Nonostante le sfide incontrate nel corso del test, il progetto si è concluso con successo. Le sfide, come il surriscaldamento, le interruzioni nella registrazione dei dati e le difficoltà legate alle condizioni ambientali, sono state affrontate con l'adozione di misure precauzionali, come il backup tramite il telefono.

4 Conclusione e sviluppi futuri

Nel corso di questo progetto è stato fatto uno studio approfondito sulle reti CNN per la rilevazione di emissioni in atmosfera. L'obiettivo era quello di analizzare le quantità di fumo e classificarle in base alla loro grandezza "lieve fumata", "evidente fumata" e "importante fumata". È stato necessario analizzare un ampio database di immagini e l'addestramento con Google Colab di molti modelli.

In base ai risultati ottenuti l'approccio alle reti convoluzionali è efficace nella rilevazione di emissioni, infatti il modello ha dimostrato di essere preciso nella classificazione dei frame. Inoltre l'utilizzo dell'augmentation ha migliorato notevolmente le prestazioni del progetto.

La consapevolezza dell'inquinamento ambientale è in continua crescita, le emissioni dannose rappresentano una minaccia significativa per la salute umana. L'approccio proposto in EcoSentinel permette di favorire una migliore comprensione delle sorgenti di inquinamento e di identificare potenziali problematiche nelle emissioni per prevenire situazioni di inquinamento eccessivo.

Come testato presso la centrale situata a Fusine, il progetto può essere semplicemente implementato presso impianti industriali e siti di produzione.

L'adozione di tecnologie come Raspberry Pi 3, un computer a basso costo e consumi elettrici rende questo tool accessibile a tutti.

L'importanza di una conoscenza base di Python è fondamentale per raggiungere i risultati di EcoSentinel. Senza l'utilizzo di questo linguaggio di programmazione non sarebbe stato possibile utilizzare le librerie di OpenCV, che hanno permesso una gestione completa delle immagini.

Uno degli elementi chiave è stato il test finale condotto utilizzando Raspberry. Ha contribuito a fare da ponte tra la fase di sviluppo software e l'applicazione pratica sul campo. Ha consentito di valutare l'efficacia e l'affidabilità del sistema in un ambiente reale. Durante questo test il raspberry è stato esposto a condizioni ambientali, tra cui variazioni di temperatura e umidità.

Per ulteriori sviluppi futuri, bisognerebbe considerare un'estensione del dataset, perché con l'acquisizione di più immagini, aiuterebbe a migliorare le prestazioni del modello per generalizzare meglio. Un'altra importante caratteristica implementabile, sarebbe

l'integrazione con sistemi di videosorveglianza, consentendo un monitoraggio continuo e automatizzato delle emissioni.

Per garantire una migliore gestione delle utenze bisognerebbe utilizzare un database posto su server con procedura di autenticazione utente. Così da garantire una gestione utenti dell'applicativo e non dover obbligare l'utilizzatore ad avviare il progetto localmente.

Spostando il database da un semplice file di log ad un server remoto, gli utenti possono accedere all'applicativo da qualsiasi luogo e dispositivo.

Una delle questioni più importanti sarebbe la gestione delle autenticazioni. Ogni utente avrebbe a disposizione il suo account, composto da privilegi specifici.

Inoltre un server in cui sono presenti procedure di autenticazione, offre una maggiore protezione dei dati e permetterebbe una maggiore sicurezza delle informazioni sensibili. Questo è fondamentale quando si parla di dati ambientali che possono avere importanza per la salute pubblica.

Attraverso un applicativo basato su server è possibile gestire l'intero sistema da remoto e permette una semplificazione nella manutenzione e per gli aggiornamenti software.

Con una centralizzazione i dati raccolti possono essere confrontati con ulteriori sensori ambientali o rilevatori di emissioni e unificati in un unico database. Consentirebbe anche di conservare un ampio storico dei risultati sulle emissioni.

La possibilità di accesso multiplo in contemporanea consentirebbe una condivisione delle informazioni multi parti, tra cui, aziende private, industrie e organizzazioni locali, garantendo così la collaborazione e condivisioni delle informazioni.

Il progetto EcoSentinel ha dimostrato risultati positivi nell'utilizzo di reti CNN (reti neurali convoluzionali) per la rilevazione di emissioni in atmosfera.

Con ulteriori ricerche e sviluppi potrebbe diventare un importante strumento per la protezione ambientale.

Bibliografia

- [1] Michael Nielsen, A Beginner's Guide to Convolutional Neural Networks, arXiv, 2015.
- [2] François Chollet, Deep Learning with Python, Manning Publications, 2017.
- [3] Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton, ImageNet classification with deep convolutional neural networks, Nature, 2012, 523 pagine.
- [4] Zed Shaw, Learn Python the Hard Way, No Starch Press, 2015, 304 pagine.
- [5] Mark Lutz, Python for Beginners, O'Reilly Media, 2019, 448 pagine.
- [6] Hao Wang e Yu-Hua Chen, Convolutional Neural Networks for Smoke Detection, Springer, 2020, 128 pagine.
- [7] Jon Duckett, HTML and CSS: Design and Build Websites, Wiley, 2011.
- [8] Dave Shreiner, GPU Computing for Beginners: Addison-Wesley Professional, 2012.
- [9] Jason Brownlee, Python per principianti, Machine Learning Mastery, 2018, 12 pagine.
- [10] John V. Guttang, Introduzione alla programmazione Python, MIT Press, 2017, 128 pagine.
- [11] Akhil Garg, Una guida introduttiva alla programmazione di reti CNN con Python, Towards Data Science, 2022, 3 pagine.
- [12] Jason Brownlee, Come creare una rete CNN con Python, Machine Learning Mastery, 2022, pagine 10.
- [13] Guido van Rossum, La storia di Python, Python Software Foundation, 2022, pagine 2
- [14] Jason Brownlee, Ultralytics e OpenCV: una panoramica, Machine Learning Mastery, 2022, 5 pagine.
- [15] David Beazley, Un confronto tra Ultralytics e OpenCV per la rilevazione di oggetti, O'Reilly Media, 2022, 10 pagine.
- [16] <https://www.ibm.com/it-it/topics/convolutional-neural-networks>, consultato in data 10/06/2023.
- [17] <https://github.com/abg3/Smoke-Detection-using-Tensorflow-2.2> , consultato in data 20/06/2023.

[18] <https://www.html.it/articoli/google-colab-per-il-machine-learning-cose-e-come-si-usa/>, consultato in data 14/07/2023.

[19] <https://blog.roboflow.com/getting-started-with-roboflow/>, consultato in data 18/07/2023.

[20] <https://docs.ultralytics.com/> , consultato in data 25/07/2023