Università degli Studi di Napoli "Parthenope"

Scuola interdipartimentale delle Scienze,
dell'Ingegneria e della Salute

Dipartimento di Scienze e Tecnologie

Corso di Laurea Triennale in Informatica

IDENTIFICAZIONE E TRACCIAMENTO DELLE REGIONI ATTIVE SUL SOLE

Relatore
Prof. Emanuel Di Nardo

Correlatore
Prof. Angelo Ciaramella

Candidata
Sara Terlizzi
0124002161

Anno Accademico 2024/2025

Abstract

Abstract italiano

Abstract (English)

Abstract inglese

A chi non respira più con me,
ma continua a vivere nei miei ricordi e nel mio cuore:
A Nonna ed Emidio, che non smetteranno mai di mancarmi.

Ai miei Genitori, all'idea che per la prima volta
sono riuscita a ripagare -per davvero- ogni vostro sforzo.
A mio fratello Francesco, ti ho desiderato con tutto il mio cuore ed
-ora che sei qui- voglio che tu sia la versione migliore di me.
Alla mia famiglia tutta, dagli incoraggiamenti prima di ogni prova
alle telefonate post-esame.

Ad ogni persona che ho incontrato sulla mia strada
-che sia rimasta o che sia stata solo di passaggio-
siete stati essenziali per srotolare il groviglio che c'è in me.

A chiunque abbia creduto in me,
nelle mie potenzialità e nel mio modo di essere,
anche quando -io stessa- non ero in grado di farlo.

A chi lotta contro il nemico invisibile dell'ansia di non essere abbastanza.
Alle notti insonni e alle lacrime.
Alla paura che ti mangia dentro e sembra non esistere via di fuga.
Al buio che mi ha fatto apprezzare la luce.
Oggi ho vinto io.

"Puoi sprecare la tua vita a tracciare confini oppure puoi decidere di vivere superandoli. Alcuni sono molto difficili da superare. Però una cosa la so: se sei pronto a correre il rischio, la vita dall'altra parte è spettacolare."

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici

Capitolo 1

Introduzione

Nel presente capitolo, si andrà a spiegare il contesto scientifico in cui si inserisce il lavoro di tesi, illustrando le motivazioni alla base della scelta del tema affrontato e definendo l'obiettivo principale della ricerca. In particolare, l'attenzione è rivolta allo studio delle regioni attive solari, in quanto sedi privilegiate dei fenomeni energetici più intensi osservati sul Sole.
Il capitolo fornisce inoltre una panoramica delle implicazioni scientifiche e applicative legate all'analisi di tali regioni e anticipa la struttura generale della tesi.

1.1 Contesto e Motivazione

Il Sole è da sempre oggetto di numerosi studi nell'ambito dell'astrofisica, in quanto sorgente di fenomeni magnetici ed energetici che influenzano l'ambiente spaziale circostante. Tra tali fenomeni, i **flare solari** rappresentano eventi improvvisi ed altamente energetici, caratterizzati dal rilascio di grandi quantità di energia sottoforma di radiazione elettromagnetica.
Numerose osservazioni hanno evidenziato come tali eventi abbiano origine principalmente nelle **regioni attive solari**, aree localizzate della fotosfera caratterizzate da intensi e complessi campi magnetici, spesso associate alla presenza di macchie solari. La complessità strutturale e l'evoluzione temporale delle regioni attive svolgono un ruolo fondamentale nell'origine dei flare solari e dei fenomeni ad essi associati, quali le espulsioni di massa coronale (*Coronal Mass Ejections, CME*) e le particelle energetiche solari (*Solar Energetic Particles, SEP*).
Quando tali eventi si propagano verso la Terra, possono dare origine a **tempeste geomagnetiche** e a una vasta gamma di effetti sull'ambiente spaziale circumterrestre, con conseguenze negative sulle infrastrutture tecnologiche [1]. Tra le manifestazioni più rilevanti vi è la generazione di **correnti geomagneticamente indotte** (*Geomagnetically Induced Currents, GIC*) nelle reti elettriche terrestri, che possono causare danni ai trasformatori e interruzioni della fornitura di energia, come dimostrato da eventi storici quali il *blackout* della rete elettrica di Hydro-Québec del marzo 1989 [2], durante il quale l'intero sistema passò da condizioni operative nominali allo spegnimento completo in circa 92

secondi, lasciando oltre sei milioni di persone senza elettricità in una giornata particolarmente fredda. Ulteriori esempi sono rappresentati dai numerosi guasti ai trasformatori che ridussero drasticamente la capacità della rete elettrica sudafricana durante le intense tempeste geomagnetiche dell'ottobre 2003 [3].
Le emissioni in raggi X e nell'ultravioletto estremo (*Extreme Ultraviolet, EUV*) prodotte dai flare solari più intensi possono modificare la struttura dell'ionosfera sul lato diurno della Terra, compromettendo la propagazione dei segnali radio e influenzando il funzionamento dei sistemi globali di navigazione satellitare (*Global Navigation Satellite Systems, GNSS*) [4]. Tali sistemi, oltre alla navigazione, forniscono servizi di temporizzazione ad alta precisione, fondamentali per settori critici come quello finanziario. Ulteriori effetti includono un aumento del *drag* atmosferico sui satelliti in orbita bassa, che può determinare una progressiva perdita di quota e, nei casi più estremi, il rientro prematuro in atmosfera. Un esempio recente di questo fenomeno è rappresentato dalla perdita di una parte della flotta di satelliti Starlink in seguito a un episodio di intensa attività geomagnetica.
Sia le espulsioni di massa coronale sia i flare solari sono inoltre associati a **tempeste di radiazione solare**, ovvero improvvisi aumenti del flusso di particelle energetiche (protoni, particelle alfa e ioni più pesanti), che possono determinare un significativo incremento dell'ambiente radiativo nello spazio circumterrestre e, in alcuni casi, anche all'interno dell'atmosfera terrestre, fino a raggiungere il livello del mare. Le tempeste di radiazione hanno effetti su una vasta gamma di sistemi elettrici ed elettronici e possono rappresentare un rischio radiativo, seppur limitato, per l'uomo, sia nello spazio sia a bordo di aeromobili. Tali effetti risultano particolarmente rilevanti per l'aviazione, soprattutto considerando il ruolo sempre più centrale dei dispositivi digitali nei sistemi di controllo del volo, nonché nei sistemi di navigazione e comunicazione.
Alla luce di ciò, l'identificazione e il tracciamento temporale delle regioni attive solari risultano di fondamentale importanza per comprendere e, potenzialmente, prevedere l'attività solare.

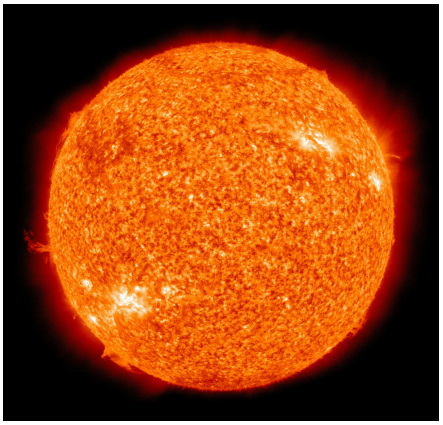


Figura 1.1: Immagine del Sole in cui sono in atto flare solari

1.2 Obiettivo della Tesi

Da quanto illustrato finora, lo studio dell'attività solare riveste un ruolo fondamentale nell'analisi dello *space weather*, ovvero l'insieme dei fenomeni astrofisici che possono influenzare le tecnologie spaziali e terrestri.
Poiché gli eventi più rilevanti in questo contesto hanno origine prevalentemente nelle regioni attive del Sole, la presente tesi si propone di:

1. Sviluppare un sistema automatico per l'identificazione delle regioni attive solari attraverso l'analisi dei magnetogrammi del disco solare, prodotti dallo strumento **Helioseismic and Magnetic Imager (HMI)** a bordo del satellite **Solar Dynamics Observatory (SDO)**.

2. Utilizzare il *data product* **HARP** (*Helioseismic and Magnetic Imager Active Region Patches*), fornito dal team scientifico del Solar Dynamics Observatory, che offre una rappresentazione strutturata delle aree magneticamente attive sulla superficie solare, permettendo l'annotazione dei magnetogrammi e lo sviluppo di un metodo di localizzazione automatica delle regioni attive, basato sull'estrazione delle *bounding box*.

3. Riadattare il modello di *object detection* **YOLOv7**, rendendolo utilizzabile in un contesto astrofisico e integrandolo con la libreria **Norfair** per il *tracking* temporale delle regioni attive.

4. Utilizzare un dataset annotato per l'addestramento del modello YOLOv7 (opportunamente modificato), con l'obiettivo di riconoscere automaticamente le regioni attive solari.

YOLOv7 è un modello di *object detection* preesistente, ideato per identificare e localizzare oggetti in immagini RGB in modo accurato ed efficiente [5]. Per adattarlo all'analisi dei magnetogrammi solari e delle regioni attive descritte nei dati HARP, sono state apportate le seguenti modifiche:

- caricamento di immagini in formato scientifico (file `.h5` contenenti magnetogrammi e *bounding box* HARP);

- estrazione delle *bounding box* a partire dai dati HARP;

- addestramento del modello per la rilevazione automatica delle regioni attive in immagini solari;

- integrazione con la libreria Norfair per il *tracking* temporale delle regioni attive rilevate.

Tali modifiche consentono di sfruttare al meglio YOLOv7 in un contesto scientifico diverso da quello per cui è stato originariamente concepito, ma mantenendo al contempo l'efficienza e la rapidità del modello originale.
I dati utilizzati in questo lavoro sono pubblicamente accessibili dal Joint Science Operations Center (JSOC), la struttura responsabile dell'elaborazione e della

distribuzione dei dati del Solar Dynamics Observatory (SDO), gestita dalla Stanford University in collaborazione con la NASA [6].
La selezione dei dati è avvenuta a cura di Elizabeth Doria Rosales, dottoranda in fisica presso l'Università di Trento.

1.3 Struttura della Tesi

La tesi è strutturata nei seguenti capitoli:

- **Capitolo 2**: Panoramica delle tecnologie e metodologie utilizzate, con focus sul modello YOLOv7 e sulla libreria Norfair.
- **Capitolo 3**: Illustrazione delle soluzioni tecniche adottate.
- **Capitolo 4**: Analisi dei risultati sperimentali, con valutazioni.
- **Capitolo 5**: Conclusioni ed eventuali sviluppi futuri.

Infine, le conclusioni racchiudono una riflessione sui risultati ottenuti, sui limiti del lavoro effettuato e le potenziali direzioni da poter intraprendere in futuro.

Capitolo 2

Modelli e Metodologie Utilizzate

Nel presente capitolo sono descritte le principali tecnologie e metodologie adottate durante lo sviluppo del progetto.

2.1 Panoramica sull'Intelligenza Artificiale, l'Object Detection e le Bounding Box

Nel contesto del progetto di tesi, queste tecnologie sono state fondamentali: l'**Intelligenza Artificiale** ha permesso l'addestramento di un modello per un compito non usuale; l'*Object Detection* è stata la metodologia per localizzare le regioni attive nei magnetogrammi; le *Bounding Box* hanno rappresentato lo strumento pratico per delimitarle visivamente.

2.1.1 Intelligenza Artificiale

L'**Intelligenza Artificiale** (IA) è un campo dell'informatica che si propone di sviluppare sistemi in grado di svolgere attività tipicamente umane, quali ragionamento, apprendimento automatico, elaborazione del linguaggio naturale e percezione visiva[7]. Negli ultimi anni, l'IA ha compiuto enormi passi in avanti, e questo lo deve all'evoluzione del *machine learning* (ML), e in particolare del *deep learning* (DL). Tutto ciò ha reso possibile affrontare con successo problemi complessi come il riconoscimento facciale, la traduzione automatica, il rilevamento di oggetti e persino la guida autonoma [8].
Nel contesto dell'IA moderna, le tecniche più rilevanti sono:

- **Machine Learning:** metodi che apprendono da dati osservati senza essere esplicitamente programmati per ogni compito.
- **Deep Learning:** architetture neurali multilivello, capaci di apprendere rappresentazioni gerarchiche e complesse dei dati.
- **Apprendimento supervisionato e non supervisionato:** rispettivamente con o senza etichette nei dati di input.

Machine Learning

Il **Machine Learning** (ML) è una branca dell'intelligenza artificiale che si occupa di progettare algoritmi in grado di apprendere automaticamente dai dati, migliorando le proprie prestazioni nel tempo senza essere esplicitamente programmati [9]. Un algoritmo di Machine Learning è addestrato su un insieme di dati *training set*, per poi essere valutato su dati non visti, detti *test set*, così da poter verificarne la sua capacità di generalizzazione.
Le principali modalità di apprendimento nel ML sono:

- **Apprendimento supervisionato:** l'algoritmo apprende da dati etichettati, ovvero ogni esempio presente dataset è associato ad un output già noto. Questa rappresenta la modalità più usata, soprattutto per quanto riguarda la classificazione di immagini o il riconoscimento vocale.
- **Apprendimento non supervisionato:** i dati non hanno etichette, per cui è l'algoritmo stesso a tentare di identificare strutture nascoste o raggruppamenti nei dati. Questa modalità è usata nel *clustering*, nella compressione o nel rilevamento di anomalie.
- **Apprendimento per rinforzo:** un agente interagisce con un ambiente e apprende tramite un sistema di ricompense e penalità, ottimizzando le proprie decisioni. Questa modalità è impiegata, ad esempio, nei videogiochi o nella robotica autonoma.

Deep Learning

Il **Deep Learning** (DL), come accennato prima, è una sottoclasse del ML che utilizza *reti neurali profonde*, composte da molti strati (*layers*) nascosti. Tali modelli sono particolarmente efficienti nel trattare dati complessi e non strutturati, come immagini, audio e testo [8]. Grazie alla grande disponibilità di dati e potenza computazionale, il Deep Learning ha rivoluzionato il campo della *computer vision*, rendendo possibili compiti prima irrisolvibili, tra cui:

- Riconoscimento facciale in tempo reale
- Traduzione automatica basata sul contesto
- Rilevamento e classificazione di oggetti in immagini ad alta risoluzione

2.1.2 Object Detection

L'**Object Detection** è un'area fondamentale della *computer vision* che ha l'obiettivo di localizzare e classificare automaticamente uno o più oggetti in un'immagine o in un video [10]. I modelli di Object Detection producono come output sia la categoria di ciascun oggetto (es. "persona", "auto", "segnale stradale"), sia una bounding box che ne racchiude l'area nell'immagine.
Le principali tecniche si dividono in due grandi famiglie:

- **Two-stage detectors:** suddividono il processo in due fasi distinte: dapprima vi è la generazione di regioni proposte (*Region Proposal Network, RPN*) e successivamente la classificazione delle regioni. Esempi noti sono R-CNN, Fast R-CNN e Faster R-CNN [11].

- **One-stage detectors:** eseguono direttamente la classificazione e la regressione delle bounding box in un'unica fase, abbreviando notevolmente i tempi. I principali rappresentanti di questa famiglia sono YOLO (*You Only Look Once*) - di cui è stato fatto uso per il presente progetto di tesi - [12] e SSD (*Single Shot MultiBox Detector*) [13].

Metriche di Valutazione

L'efficacia di un modello di Object Detection viene valutata con metriche come:

- **Intersection over Union (IoU):** misura l'intersezione tra la bounding box predetta e quella reale:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

  Un valore di IoU pari a 1 indica una corrispondenza perfetta tra predizione e *ground truth*, mentre un valore pari a 0 indica assenza totale di sovrapposizione.

- **Precision, Recall e mAP (mean Average Precision):** metriche derivate da classificazione e localizzazione, il cui scopo è confrontare le prestazioni tra modelli.

2.1.3 Bounding Box

La **bounding box** è un rettangolo utilizzato per racchiudere un oggetto rilevato all'interno di un'immagine. È solitamente rappresentata da quattro coordinate:

$$(x_{\min}, y_{\min}, x_{\max}, y_{\max})$$

dove:

- $(x_{\min}, y_{\min})$ rappresenta il vertice in alto a sinistra,
- $(x_{\max}, y_{\max})$ rappresenta il vertice in basso a destra.

Alternativamente, può essere espressa con centro e dimensioni:

$$(x_{\text{center}}, y_{\text{center}}, w, h)$$

dove w e h sono larghezza e altezza.
L'accuratezza delle bounding box è cruciale nei sistemi *real-time*, dove la minima imprecisione può compromettere l'affidabilità dell'intero sistema.

2.1.4 Applicazioni

Le possibili applicazioni dell'Object Detection, assieme all'utilizzo delle bounding box, sono molteplici. Alcuni esempi possono essere:

- **Veicoli autonomi:** per rilevare pedoni, segnali stradali e/o altri veicoli.
- **Videosorveglianza:** per il rilevamento di attività sospette o intrusioni.
- **Medicina:** per individuare anomalie in immagini radiologiche.
- **Astronomia e climatologia:** per l'identificazione automatica di eventi rari o transitori.

2.1.5 Conclusione

L'utilizzo combinato di IA, Object Detection e bounding box ha trasformato radicalmente il modo in cui le macchine percepiscono e interagiscono con il mondo visivo. I continui sviluppi in questo ambito promettono ulteriori miglioramenti in precisione, efficienza e applicabilità in scenari sempre più complessi.
Nel contesto di questa tesi, esse sono state fondamentali: l'Intelligenza Artificiale ha permesso di addestrare un modello modificato *ad hoc* per un compito differente dal solito, l'Object Detection è stata la metodologia utilizzata per la localizzazione delle regioni attive nei magnetogrammi, mentre le Bounding Box sono stato lo strumento pratico per delimitarle visivamente.

2.2 Python

Python è un linguaggio di programmazione ad alto livello, ampiamente utilizzato per la sua semplicità di sintassi, versatilità e la presenza di un ecosistema ricco di librerie. Spicca per i suoi punti di forza: adattabilità ad ogni piattaforma, interattività e dinamicità e protipizzazione rapida.
Nel contesto di questo progetto di tesi, Python è stato scelto come linguaggio principale per lo sviluppo degli script e l'esecuzione degli elaborazione dati, visione artificiale e deep learning. Si è rivelato essenziale per integrare le diverse componenti del sistema: dalla lettura dei dati scientifici HARP, alla pre-elaborazione delle immagini, fino all'addestramento ed inferenza del modello.

Figura 2.1: Logo di Python

2.2.1 Librerie ausiliarie

- **NumPy** [14]: è la libreria fondamentale per il calcolo scientifico in Python. Fornisce strutture dati per la gestione di array multidimensionali e una innumerevoli funzioni matematiche per operazioni vettoriali e matriciali. Nel progetto, è stata utilizzata per la manipolazione e preparazione dei dati numerici dei magnetogrammi, rappresentati come array multidimensionali, e per gestire le coordinate delle bounding box durante le fasi di addestramento e validazione.

- **SciPy** [15]: estende le funzionalità di NumPy, offrendo strumenti per operazioni matematiche avanzate come integrazione numerica, ottimizzazione, interpolazione e algebra lineare. Nel progetto, è stata utilizzata per supportare calcoli complessi durante l'analisi dei dati e l'elaborazione dei risultati, in particolare ha fornito strumenti di supporto per analisi statistiche preliminari sulle proprietà magnetiche delle regioni attive descritte nei dati HARP.

- **OpenCV** [16]: libreria open source per la computer vision e l'elaborazione delle immagini. Offre algoritmi ottimizzati per operazioni di filtraggio, rilevamento di caratteristiche, trasformazioni geometriche e manipolazione delle immagini. Nel progetto, è stata usata in primis perché viene utilizzata dal modello, ma anche per la pre-elaborazione e manipolazione dei magnetogrammi solari come, ad esempio, la normalizzazione dei valori e la conversione dei formati, rendendoli idonei per l'input da passare al modello.

- **Matplotlib** [17]: libreria di visualizzazione dati che permette la creazione di grafici statici, animati e interattivi. Consente di rappresentare chiaramente e in modo personalizzabile i risultati ottenuti, facilitando l'analisi e la documentazione visiva dei dati elaborati. Nel progetto, è stata la libreria indispensabile per visualizzare i risultati, ad esempio per disegnare le bounding box predette direttamente sui magnetogrammi solari e per generare grafici che mostrano l'andamento delle performance del modello.

2.3 Miniconda

Miniconda è una distribuzione minimale di Conda, un sistema di gestione degli ambienti e dei pacchetti Python. È stato utilizzato per creare ambienti isolati e riproducibili, permettendo di configurare ciascuno di essi con una propria versione di Python e delle librerie di terze parti necessarie (come PyTorch, NumPy, OpenCV).
Questa gestione consente di mantenere sotto controllo le dipendenze e garantisce che le specifiche versioni delle librerie siano compatibili fra loro, evitando conflitti che avrebbero potuto compromettere l'addestramento del modello. [18]

Figura 2.2: Logo di Miniconda

2.4 PyTorch

PyTorch è un framework open-source per il deep learning sviluppato da Meta AI. Esso fornisce un'interfaccia dinamica e intuitiva per la definizione e l'ottimizzazione delle reti neurali, ed è particolarmente apprezzato per il suo supporto nativo all'accelerazione tramite GPU [19].
In questo progetto di tesi, è stato il framework su cui si basa l'intero processo di addestramento ed inferenza del modello, offrendo la flessibilità desiderata e necessaria per apportare le modifiche atte all'adattamento del compito scientifico.

Figura 2.3: Logo di Pytorch

2.5 CUDA

CUDA (*Compute Unified Device Architecture*) è una piattaforma di calcolo parallelo sviluppata da NVIDIA, che consente l'utilizzo delle GPU per compiti generici di calcolo.
Data l'enorme mole di dati e la complessità del modello utilizzato, l'accelerazione hardware è stata indispensabile per ridurre i tempi sia di addestramento che di inferenza, rendendo possibile l'esecuzione di più esperimenti in tempi alquanto ragionevoli.
Nel contesto di questa tesi, la corretta configurazione di CUDA e la sua compatibilità con la versione di PyTorch si sono rivelate essenziali per il garantire il funzionamento ottimale dei modelli su GPU.[20].

Figura 2.4: Logo di CUDA

2.6 Norfair

Norfair è una libreria open source sviluppata in Python per il *tracking* multi-oggetto in tempo reale [21]. Essa è progettata per integrarsi modularmente con qualsiasi sistema di rilevamento che fornisca coordinate spaziali (ad esempio le coordinate (x, y) dei centri delle bounding box).
La libreria si occupa esclusivamente della parte di tracking, ovvero dell'associazione temporale dei rilevamenti (*detections*) frame per frame, mantenendo un identificatore univoco stabile per ogni oggetto monitorato nel video o nel flusso di immagini. Non includendo la componente di rilevamento, essa offre la flessibilità di utilizzare qualsiasi *detector* esterno (come YOLO, Detectron2, MediaPipe).
Nel progetto sviluppato, svolge il ruolo chiave di "inseguire" le regioni attive nel tempo: una volta che YOLOv7 le ha identificate nel singolo magnetogramma, Norfair associa le rilevazioni tra frame consecutivi, dando la possibilità di studiare l'evoluzione di una specifica regione solare.

Figura 2.5: Logo di Norfair

2.6.1 Funzionamento e Caratteristiche

Il funzionamento di Norfair si basa sui seguenti concetti chiave:

- **Detections**: ogni oggetto rilevato in un fotogramma è rappresentato da coordinate spaziali, tipicamente il centro della bounding box o altri punti di interesse.
- **Tracks**: sono le tracce temporali che mantengono lo storico delle posizioni e degli identificatori degli oggetti nel tempo.
- **Funzione di distanza**: per associare i nuovi rilevamenti alle tracce esistenti, viene utilizzata una funzione di distanza personalizzabile (ad esempio euclidea o basata su caratteristiche visive).
- **Assegnazione**: ogni nuova detection viene assegnata alla traccia più vicina purché la distanza sia inferiore a una soglia predefinita (*distance threshold*); in caso contrario, viene creata una nuova traccia.

Ad ogni nuovo fotogramma, Norfair riceve l'elenco delle *detections*, aggiorna le tracce esistenti o ne crea di nuove e rimuove quelle non aggiornate per un certo numero di frame, garantendo così un tracking coerente anche in presenza di occlusioni temporanee o movimenti rapidi.

2.6.2 Applicazioni Tipiche

Grazie alla sua leggerezza e modularità, Norfair è utilizzata in molteplici ambiti, tra cui:

- **Videosorveglianza e sicurezza**: per monitorare flussi di persone o veicoli in ambienti affollati.
- **Analisi di movimento in sport e intrattenimento**: per tracciare giocatori e palla in tempo reale durante le competizioni.
- **Robotica autonoma**: per permettere ai robot di seguire oggetti o evitare ostacoli mobili.

Sebbene nato per contesti differenti da quello del lavoro di tesi, è stato dimostrato che Norfair possiede sufficiente versatilità per adattarsi efficacemente al dominio astrofisico, permettendo il monitoraggio continuo delle regioni attive sulla superficie solare.

Figura 2.6: Esempio di multi-tracking di Norfair

2.7 YOLOv7

YOLOv7 (*You Only Look Once version 7*) è uno dei modelli più recenti e avanzati per il rilevamento oggetti in tempo reale, appartenente alla famiglia di algoritmi YOLO. Sviluppato per ottenere alte prestazioni sia in termini di accuratezza che di velocità, si distingue per una serie di ottimizzazioni che lo rendono adatto ad innumerevoli applicazioni, anche su dispositivi con risorse computazionali limitate [5].
La scelta di tale modello è motivata dall'ottimo compromesso tra efficienza e precisione, una caratteristica cruciale nell'analisi di grandi moli di dati astrofisici.

2.7.1 Funzionamento e Caratteristiche

YOLOv7 è progettato per realizzare una *pipeline end-to-end*, che riceve in input un'immagine e restituisce in output le bounding box e le classi associate agli oggetti rilevati.
Il funzionamento, durante la fase di inferenza, si basa sulla suddivisione dell'immagine in griglie e sull'applicazione di convoluzioni profonde per produrre:

- Le coordinate delle bounding box.
- La classe predetta per ogni oggetto rilevato.
- La probabilità (*confidenza*) associata a ciascuna predizione.

Si tratta del cuore del sistema di rilevamento sviluppato nel presente lavoro di tesi, necessario per il successivo tracciamento.
L'architettura separa nettamente la configurazione degli iperparametri e degli *anchor* dai pesi pre-addestrati e offre diverse caratteristiche avanzate:

- **Efficienza**: elevata precisione nel rilevamento pur mantenendo alte velocità di inferenza.
- **Versatilità**: supporto per *Object Detection*, *Instance Segmentation* e *Pose Estimation*.
- **Scalabilità**: disponibilità di architetture derivate (ad esempio YOLOv7-tiny, YOLOv7-X, YOLOv7-W6) adattabili a diverse situazioni e capacità di calcolo.
- **Ottimizzazione Hardware**: supporto nativo per GPU CUDA, che consente un'inferenza efficiente su hardware NVIDIA.
- **Multitasking**: possibilità di usare *multi-head* per compiti multitask (ad esempio detection + keypoints).

In fase di addestramento, inoltre, il modello utilizza tecniche avanzate di ottimizzazione che migliorano l'apprendimento, senza aumentare la complessità dell'inferenza.

2.7.2 Applicazioni Tipiche

YOLOv7 trova impiego in numerosi ambiti, tra cui:

- **Videosorveglianza e sicurezza pubblica**.
- **Veicoli autonomi e sistemi ADAS**.
- **Controllo qualità industriale**.
- **Analisi sportiva e tracciamento di giocatori**.
- **Ricerca scientifica**.

2.7.3 Avanzamenti nei Rilevatori in Tempo Reale

YOLOv7 rappresenta un significativo passo avanti nel campo del *real-time object detection*. L'architettura introduce il concetto di **Trainable Bag-of-Freebies**: un insieme di moduli e strategie ottimizzate che migliorano significativamente la fase di addestramento senza incrementare il costo computazionale durante l'inferenza.
Tutti i modelli sono stati addestrati da zero sul dataset COCO, senza utilizzare pesi pre-addestrati o dati esterni [5].

E-ELAN: Extended Efficient Layer Aggregation Networks

Per migliorare l'apprendimento delle rappresentazioni senza interrompere il percorsi di propagazione del gradiente, YOLOv7 introduce una nuova *blackbone* denominata **E-ELAN**.
A differenza delle architetture tradizionali, E-ELAN mantiene fissa la struttura di transizione, ma agisce sui blocchi computazionali utilizzando:

- **Group Convolution**: per suddividere i canali in gruppi.
- **Shuffle & Merge Cardinality**: per mescolare e fondere le *feature map* provenienti dai diversi gruppi.

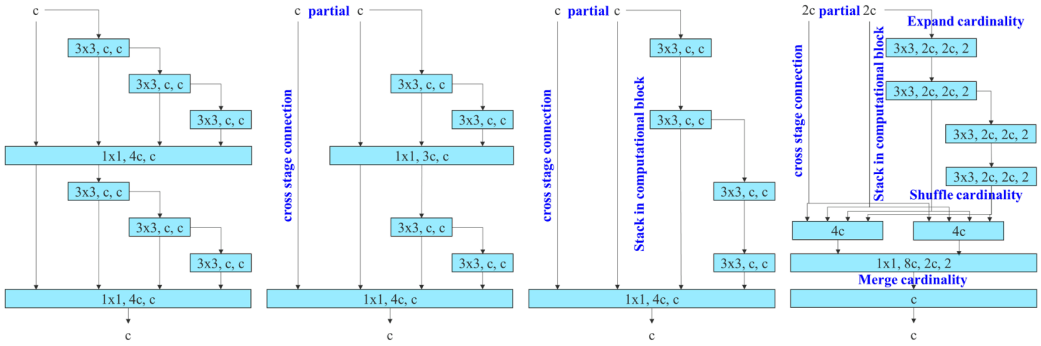


Figura 2.7: L'architettura E-ELAN mantiene invariato il percorso di propagazione del gradiente dell'architettura originale, ma utilizza convoluzioni di gruppo per aumentare la cardinalità delle feature aggiunte. Le feature provenienti da diversi gruppi vengono combinate attraverso operazioni di mescolamento e fusione, migliorando così la varietà delle rappresentazioni apprese e l'efficienza nell'uso dei parametri e del calcolo.

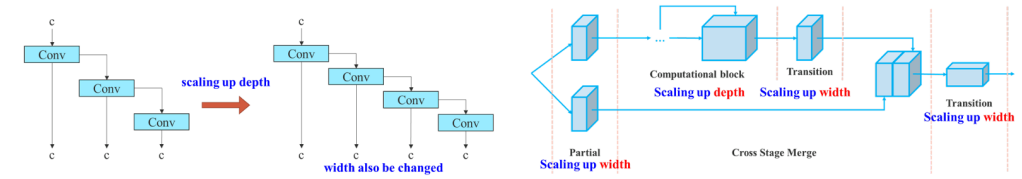
Compound Scaling

Il ridimensionamento dei modelli avviene, solitamente, modificando profondità, larghezza o risoluzione.

Tuttavia, nei modelli basati su concatenazione (come YOLO), scalare solo la profondità causa una variazione indesiderata dei canali di input nei layer successivi, rompendo l'equilibrio computazionale.
Per risolvere questo problema, YOLOv7 propone il **Compound Scaling**, una tecnica che scala profondità e larghezza in modo congiunto: quando si scala la profondità di un blocco computazionale, si scala simultaneamente la larghezza dei layer di transizione.
Le relazioni sono definite come:

$$d' = d \cdot \alpha$$
$$w' = w \cdot \beta$$

dove α e β sono fattori empiricamente scelti per mantenere l'architettura ottimale al variare della scala del modello.



(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figura 2.8: Comportamento e la soluzione proposta per lo scaling nei modelli basati su concatenazione, composta da tre sottoparti: (a): Dimostra che scalare solo la profondità (depth) in un blocco concatenativo fa aumentare la larghezza del layer di uscita; (b): Mostra che l'output più largo influenza la transizione successiva, creando problemi strutturali; (c): Presenta la soluzione proposta: uno scaling combinato (compound scaling), in cui si scala la profondità nel blocco e la larghezza nei layer di transizione per mantenere la coerenza architetturale.

Planned Re-parameterized Convolution

YOLOv7 ottimizza le convoluzioni ri-parametrizzate (*RepConv*), che combinano diverse operazioni in un unico layer per l'inferenza.
Gli autori hanno osservato che la connessione identità -tipica delle RepConv standard- degrada le prestazioni, se applicata indiscriminatamente su connessioni residue o concatenate. La soluzione adottata è, pertanto, la **Planned RepConv**, che utilizza una variante priva di identità (*RepConvN*) nei layer sensibili, garantendo la compatibilità strutturale con architetture come ResNet o DenseNet.

Label Assignment: Coarse-to-Fine Lead Guided

Una delle innovazioni più rilevanti riguarda la strategia di assegnazione delle etichette (**label assignment**), durante l'addestramento, con supervisione profonda (*deep supervision*).
Il modello utilizza due "teste" (*heads*):

- **Lead Head**: responsabile dell'output finale, genera predizioni raffinate.
- **Auxiliary Head**: assiste l'addestramento nei livelli intermedi, ricevendo etichette guidate dal lead head, non direttamente dal ground truth.

La strategia **Coarse-to-Fine** prevede che il Lead Head guidi l'apprendimento dell'Auxiliary Head:

1. Il Lead Head genera etichette di alta precisione per se stesso.
2. Dalle predizioni del Lead Head vengono derivate etichette grezze per l'Auxiliary Head. Per evitare che quest'ultima faccia *overfitting*, viene applicato un vincolo superiore (*Upper Bound Constraint*) che limita l'assegnazione delle etichette in base alla distanza dal centro dell'oggetto:

$$\text{Score}_{\text{coarse}} = \max\left(0, 1 - \frac{\text{dist}}{\text{thresh}}\right)$$

Questo permette alla testa ausiliaria di apprendere meglio le informazioni contestuali, migliorando la capacità di generalizzazione del modello (*recall*), mentre la testa principale si focalizza sulla precisione.

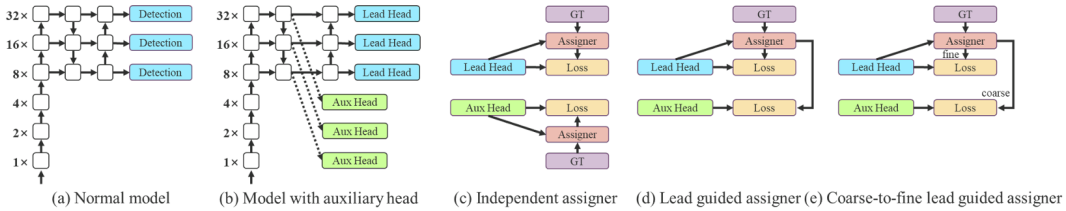


Figura 2.9: Assegnazione di etichette coarse per l'head ausiliario e fine per il lead head. Rispetto al modello standard (a), lo schema (b) include una testa ausiliaria. Diversamente dall'assegnazione indipendente delle etichette (c), sono proposte due nuove strategie: (d) assegnazione guidata dal lead head e (e) assegnazione guidata coarse-to-fine. Quest'ultima genera contemporaneamente le etichette per il training del lead head e della testa ausiliaria combinando le predizioni del lead head con il ground truth.

Altri Trainable-of-Freebies

Oltre alle innovazioni architetturali, YOLOv7 integra una serie di "trucchi" di addestramento (*Bag-of-Freebies*) ottimizzati per massimizzare le prestazioni senza costi aggiuntivi:

- **Batch Normalization fusa**: durante l'inferenza, i layer di Batch Normalization vengono fusi con i layer convoluzionali, semplificando la struttura della rete e riducendo la latenza.

- **Implicit Knowledge**: un concetto derivato da YOLOR, qui semplificato tramite l'uso di vettori statici pre-calcolati che vengono combinati con le feature map.
- **EMA Model (Exponential Moving Average)**: utilizzo di una media mobile esponenziale dei pesi del modello durante il training per ottenere il modello finale di test, tecnica che aumenta la robustezza e la stabilità delle predizioni.

Confronto delle Prestazioni

Le innovazioni introdotte permettono a YOLOv7 di superare i modelli precedenti. Come evidenziato nella Tabella 2.1, a parità di risoluzione, YOLOv7 ottiene risultati migliori in mAP, mantenendo un *framerate* (FPS) più elevato rispetto a YOLOR e YOLOv5.

Modello	FPS	mAP	Parametri	FLOPs
YOLOv5-X	83	50.7%	86.7M	205.7G
YOLOR-CSP-X	87	52.7%	96.9M	226.8G
YOLOv7-X	**114**	**52.9%**	71.3M	189.9G

Tabella 2.1: Confronto tra YOLOv7 ed altri modelli stato dell'arte (640×640).

La Tabella 2.2 dimostra, inoltre, l'efficacia del Compound Scaling: rispetto allo scaling tradizionale (solo larghezza o solo profondità), il metodo combinato ottiene il miglior risultato in accuratezza con un incremento minimo di calcoli.

Modello	mAP	Parametri	FLOPs
Base	51.7%	47.0M	125.5G
Solo larghezza	52.4%	73.4M	195.5G
Solo profondità	52.7%	69.3M	187.6G
Compound (YOLOv7-X)	**52.9%**	71.3M	189.9G

Tabella 2.2: Confronto dell'efficienza del Compound Scaling e metodi di scaling tradizionali.

2.8 Dati

2.8.1 Definizione e significato dei Dati Osservativi

Il Solar Dynamics Observatory (SDO) è una missione spaziale della NASA, lanciata nel 2010, dedicata allo studio continuo dell'attività solare e dei suoi effetti sulla Terra e sullo spazio circumterrestre [22]. Tra gli strumenti scientifici a bordo di SDO, il Helioseismic and Magnetic Imager (HMI) fornisce accurate misure del campo magnetico fotosferico sull'intero disco solare, denominate magnetogrammi, con elevata risoluzione spaziale e temporale [23]. I magnetogrammi hanno

una dimensione di 4096×4096 pixel e costituiscono una base osservativa fondamentale per lo studio delle strutture magnetiche solari e della loro evoluzione, in particolare delle regioni attive.
Le regioni attive solari rappresentano manifestazioni localizzate di intensa attività magnetica sulla fotosfera e sono caratterizzate da forti concentrazioni di flusso magnetico. Esse sono comunemente associate alla presenza di macchie solari e costituiscono le sedi principali di fenomeni energetici quali brillamenti solari ed espulsioni di massa coronale [24]. Nei magnetogrammi, le regioni attive si distinguono chiaramente dal fondo della fotosfera quieta grazie a valori elevati del campo magnetico, con polarità opposte ben definite e spazialmente correlate.
La pipeline di HMI identifica e traccia automaticamente le regioni di attività magnetica, denominate **HMI Active Region Patches (HARP)**. Ogni HARP è numerata ed è associata a una sequenza temporale di mappe (*bitmaps*) sufficientemente estese da contenere l'intera evoluzione spaziale della regione attiva durante la sua visibilità sul disco solare. Ciascuna HARP corrisponde a una singola regione attiva o a un complesso di regioni attive ed è tracciata in modo coerente nel tempo. I dati HARP forniscono principalmente informazioni di tipo geometrico sulla regione attiva, mantenendo il riferimento alla sua posizione sul disco solare.

2.8.2 Finalità e utilizzi

I magnetogrammi prodotti da HMI permettono di analizzare direttamente la distribuzione e l'intensità del campo magnetico fotosferico, che rappresenta la grandezza fisica fondamentale alla base della formazione e dell'evoluzione delle regioni attive.
La disponibilità di osservazioni *full-disk*, continue nel tempo e distribuite su lunghe scale temporali, consente non solo l'identificazione delle regioni attive sull'intero disco solare, ma anche il loro monitoraggio durante le diverse fasi del ciclo solare. Le HMI Active Region Patches (HARP) forniscono inoltre una descrizione strutturata delle regioni attive, permettendo di seguirne l'evoluzione spaziale e temporale in modo coerente e sistematico.
In questo lavoro, tali dati sono utilizzati come base osservativa per lo sviluppo di metodologie di rilevamento e tracciamento automatico delle regioni attive. L'informazione magnetica contenuta nei magnetogrammi e la segmentazione fornita dalle HARP rendono infatti questi dati particolarmente adatti ad applicazioni di object detection e tracking temporale, consentendo lo studio dell'evoluzione delle regioni attive.

2.8.3 Origine, Selezione e Organizzazione del Dataset

I dati utilizzati in questo lavoro sono disponibili pubblicamente tramite il Joint Science Operations Center (JSOC). I magnetogrammi full-disk sono forniti dalla serie `hmi.M_720s`, mentre l'informazione geometrica relativa alle regioni attive (patch tracciate dalla pipeline di HMI) è contenuta nella serie `hmi.Mharp_720s`.

Poiché l'archivio HMI copre un intervallo temporale molto esteso (oltre quindici anni), è stata definita una strategia di selezione volta a costruire un dataset rappresentativo ma computazionalmente gestibile.
La selezione temporale è stata guidata dall'andamento dell'attività solare durante il **Ciclo Solare 24**, ossia il ciclo quasi periodico di circa 11 anni che descrive la variazione dell'attività magnetica del Sole, comunemente tracciata attraverso il numero di macchie solari. Durante il massimo di un ciclo solare si osserva una maggiore attività magnetica, che si traduce in un numero più elevato di regioni attive. In particolare, il campionamento è stato limitato a un sottointervallo del ciclo (anni 2011-2019) e reso più denso nei periodi di maggiore attività: gli anni 2012-2014 sono stati intenzionalmente sovracampionati rispetto agli altri, così da aumentare la presenza di esempi contenenti regioni attive e una variabilità magnetica più marcata.
Successivamente, i dati sono stati suddivisi in training, validation e test set, evitando sovrapposizioni temporali tra i diversi insiemi. La procedura (implementata dalla dottoranda Elizabeth Doria Rosales nel Codice A.1 in Appendice A) prevede la selezione di finestre temporali di durata 90 giorni, all'interno delle quali vengono assegnati segmenti temporali consecutivi ai tre sottoinsiemi secondo le proporzioni 70% / 15% / 15%. Tra un sottoinsieme e il successivo viene inoltre introdotto un intervallo di separazione di 15 giorni, riducendo ulteriormente il rischio di *data leakage* dovuto alla persistenza e all'evoluzione delle stesse strutture magnetiche.
La distribuzione risultante dei segmenti temporali rispetto all'andamento del ciclo solare è mostrata in Figura 2.10

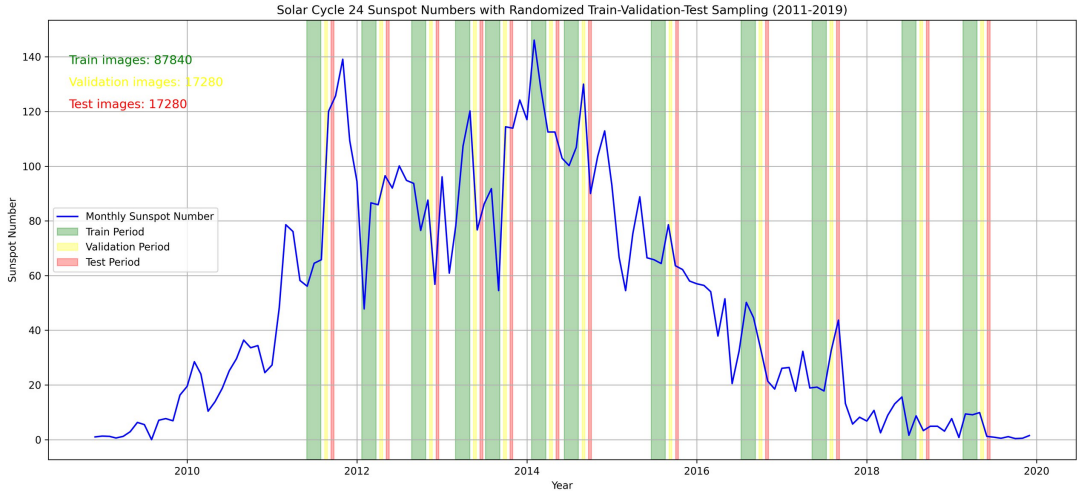


Figura 2.10: Andamento del numero di macchie solari durante il Ciclo 24 (2011-2019). Le aree colorate indicano gli intervalli temporali selezionati per il Training (verde), la Validazione (giallo) e il Test (rosso).

Il risultato di questa procedura è un insieme di periodi temporalmente disgiunti, bilanciati rispetto alle diverse fasi del ciclo solare e già strutturati nei

rispettivi split, salvati in un file di configurazione (CSV) contenente, per ciascun sottoinsieme, le date di inizio e fine dei segmenti selezionati. Per completezza, la Tabella 2.3 riporta l'elenco completo degli intervalli temporali generati.

Split	**Start**	**End**
Train	2011-05-30	2011-07-30
Validation	2011-08-15	2011-08-27
Test	2011-09-12	2011-09-24
Train	2012-01-22	2012-03-23
Validation	2012-04-08	2012-04-20
Test	2012-05-06	2012-05-18
Train	2012-08-23	2012-10-23
Validation	2012-11-08	2012-11-20
Test	2012-12-06	2012-12-18
Train	2013-02-28	2013-04-30
Validation	2013-05-16	2013-05-28
Test	2013-06-13	2013-06-25
Train	2013-07-07	2013-09-06
Validation	2013-09-22	2013-10-04
Test	2013-10-20	2013-11-01
Train	2014-01-21	2014-03-23
Validation	2014-04-08	2014-04-20
Test	2014-05-06	2014-05-18
Train	2014-06-10	2014-08-10
Validation	2014-08-26	2014-09-07
Test	2014-09-23	2014-10-05
Train	2015-06-19	2015-08-19
Validation	2015-09-04	2015-09-16
Test	2015-10-02	2015-10-14
Train	2016-07-10	2016-09-09
Validation	2016-09-25	2016-10-07
Test	2016-10-23	2016-11-04
Train	2017-05-11	2017-07-11
Validation	2017-07-27	2017-08-08
Test	2017-08-24	2017-09-05
Train	2018-06-01	2018-08-01
Validation	2018-08-17	2018-08-29
Test	2018-09-14	2018-09-26
Train	2019-02-18	2019-04-20
Validation	2019-05-06	2019-05-18
Test	2019-06-03	2019-06-15

Tabella 2.3: Elenco completo degli intervalli temporali selezionati per il Ciclo Solare 24 (2011-2019), suddivisi in Training, Validation e Test, come definiti nel file di configurazione `solar_cycle_splits.csv`.

Questa organizzazione costituisce la base per il successivo scaricamento dei dati HMI corrispondenti e delle informazioni associate alle regioni attive identificate, eseguito tramite il Codice A.3 in Appendice A.

2.8.4 Applicazioni nel presente lavoro

Nel contesto di questa tesi, i dati HARP vengono impiegati per delineare bounding box attorno alle regioni attive nelle immagini solari, consentendo l'isolamento automatico delle aree di interesse per l'analisi. Essi permettono inoltre di monitorare l'evoluzione temporale di ciascuna regione, così da individuare variazioni morfologiche e magnetiche nel tempo. Infine, i dati vengono utilizzati per studiare la correlazione tra le regioni attive e l'insorgenza di flare solari, contribuendo alla comprensione dei processi fisici che ne regolano la genesi.

2.8.5 Struttura del file

Ogni file è denominato con `hmi_magnetogram_YYYY_MM_DD_MIN_HH.h5` e contiene due gruppi:

- `harp/metadata`, al cui interno sono presenti più sottogruppi `HARP_XXXX`, ognuno dei quali corrisponde ad una regione attiva ed ospita una ricca collezione di metadati relativi alla regione stessa, ad esempio: coordinate, estensione e localizzazione della regione nel Sole, qualità dei dati, versioni del software e informazioni sul satellite..

- `magnetogram`, che include i dati scientifici sotto forma di una matrice bidimensionale rappresentativa del campo magnetico fotosferico (ossia l'immagine del disco solare che descrive la distribuzione del campo magnetico), e i metadati associati, contenenti le informazioni descrittive relative al magnetogramma stesso.

2.8.6 Metadati HARP di interesse

Poiché ogni regione attiva ha decine di attributi, si riportano nella Tabella 2.4 esclusivamente quelli più significativi ai fini del lavoro di tesi:

Campo	Significato
HARPNUM	Numero identificativo univoco della regione attiva solare (HARP)
CRPIX1, CRPIX2	Coordinate (in pixel) dell'angolo in basso a sinistra della regione nella CCD HMI
CRSIZE1, CRSIZE2	Larghezza e altezza (in pixel) del bounding box della regione attiva
AREA	Area della regione proiettata sul disco solare, espressa in micro-emisferi solari
T_REC	Informazioni temporali della serie

Tabella 2.4: Descrizione dei metadati principali utilizzati per identificare, localizzare e caratterizzare le regioni attive HARP nei magnetogrammi solari.

La selezione dei campi elencati nella Tabella 2.4 è stata guidata dalla necessità di delimitare spazialmente e ordinare temporalmente le regioni attive solari sul disco solare, al fine di sviluppare un sistema automatico di rilevamento e tracciamento delle stesse. Di seguito vengono motivate le scelte effettuate:

- **HARPNUM** è il codice univoco associato a ciascuna HARP ed è utilizzato come chiave primaria per il raggruppamento delle osservazioni successive nel tempo. Questo campo è essenziale per la costruzione di sequenze temporali coerenti, da fornire a moduli di tracking multi-frame.
- **CRPIX1, CRPIX2** e **CRSIZE1, CRSIZE2** definiscono il bounding box della HARP all'interno del magnetogramma full-disk. Tali coordinate sono utilizzate per delimitare spazialmente le regioni attive, le cui posizioni vengono successivamente convertite in coordinate normalizzate. Questi campi risultano quindi fondamentali sia per la preparazione del dataset di addestramento sia per il post-processing delle predizioni.
- **AREA** fornisce una misura dell'estensione fisica della regione attiva, espressa in micro-emisferi solari. Questo parametro può essere impiegato per filtrare regioni di dimensioni molto ridotte o affette da rumore, spesso non associate a eventi di flare significativi.
- **T_REC** rappresenta il timestamp del record ed è utilizzato per ordinare cronologicamente i dati corrispondenti a una determinata HARP.

L'insieme di questi metadati consente di costruire un dataset strutturato e annotato, in cui ogni regione attiva è localizzata, descritta magneticamente e tracciata nel tempo, con possibilità di associare ground truth fisiche (flare, classi magnetiche) per l'addestramento e la valutazione di modelli di detection e *forecasting*.

Capitolo 3

Applicazione realizzata

Nel presente capitolo, è descritta dettagliatamente l'intera architettura software del sistema sviluppato, focalizzandosi sulle strategie metodologiche adottate per trasformare un framework di Object Detection generico in uno strumento idoneo all'utilizzo di dati astrofisici.
L'analisi si concentra sull'adattamento strutturale del modello YOLOv7. In particolare, è illustrato come esso sia stato esteso per processare i magnetogrammi solari, integrando moduli specifici per la gestione del formato scientifico e per il tracciamento temporale delle predizioni.
Successivamente, viene discusso un approccio alternativo, basato sull'adattamento del dato piuttosto che del modello: un processo di pre-elaborazione che converte i dati scientifici in un formato standard, rendendoli compatibili con l'architettura originale, senza alterarne la struttura interna.

3.1 Adattamento del modello YOLOv7

L'utilizzo di un modello di Object Detection generico come YOLOv7 nel dominio dell'astrofisica solare pone una sfida architetturale notevole: l'incompatibilità del framework con i formati di dati scientifici.
In questa sezione, è illustrata la metodologia adottata per estendere le capacità di input del modello, abilitando il sistema all'interpretazione dei dati HARP. L'obiettivo è stato quello di superare i vincoli imposti dai formati di immagine tradizionali (attesi dal modello), garantendo che l'informazione fisica contenuta nei magnetogrammi venisse oreservata integralmente durante il processo di caricamento.

3.1.1 Implementazione di un Data Loader per Dati Scientifici

Come ampiamente anticipato, l'architettura di YOLOv7 è vincolata all'uso di immagini standard (es. `JPEG, PNG`) e annotazioni testuali esplicite. I dati HARP, al contrario, presentano una complessità strutturale data da:

- **Formato contenitore:** i dati sono racchiusi da un formato `HDF5` (`.h5`), ovvero un file system gerarchico pensato per dati scientifici. All'interno del singolo file, i dati sono suddivisi in gruppi distinti e non solo come una semplice immagine.

- **Annotazioni implicite:** le coordinate delle bounding box non sono fornite esplicitamente, bensì è necessaria l'estrazione, partendo dagli attributi incastonati nella gerarchia del file.

Per colmare tale divario, è stato progettato ed implementato un **modulo di caricamento dati personalizzato**, che agisce come interfaccia tra il formato scientifico e il tensore di input richiesto dal modello. Il componente implementa due strategie metodologiche per garantire efficienza e stabilità: un meccanismo di *Caching dei Metadati* per ottimizzare i tempi di avvio e un sistema di *gestione delle eccezioni* per ignorare i file corrotti in fase di addestramento.

Fase di Inizializzazione e Caching

Questa fase gestisce la pre-elaborazione di tutti i metadati del dataset. Per evitare la ridondanza di operazioni costose ad ogni avvio del training, è stata implementata una logica di *caching*. I passi sono i seguenti:

1. **Verifica della Cache**: Il sistema verifica preventivamente l'esistenza di strutture dati contenenti le etichette già elaborate e le dimensioni originali dell'immagine.

2. **Caricamento Veloce (*Cache Hit*)**: Se i file di cache sono presenti, le informazioni sono caricate immediatamente in memoria, riducendo i tempi di inizializzazione: questo rende il processo di sperimentazione molto più agile.

3. **Indicizzazione (*Cache Miss*)**: In assenza di cache, (prima esecuzione), il sistema scansiona ogni file scientifico, eseguendo le seguenti operazioni:

   (a) **Estrazione**: accesso alla struttura interna del file per recuperare le dimensioni del magnetogramma;

   (b) **Validazione dei Metadati**: viene iterato ogni sottogruppo, ciascuno dei quali rappresenta una regione attiva, sottoponendolo a tre controlli di integrità dell'etichetta:

      i. **Completezza**: verifica della presenza di tutti gli attributi necessari per la definizione delle bounding box: coordinate x e y del pixel di riferimento della regione, larghezza e altezza in pixel della regione. Se anche solo uno di questi attributi manca, la regione è scartata.

      ii. **Consistenza Dimensionale**: scarto delle annotazioni con dimensioni nulle o negative.

iii. **Limiti (*Boundary Check*)**: si esegue un controllo sulle coordinate normalizzate rispetto al centro della bounding box, escludendo dal dataset le annotazioni il cui baricentro ricade esternamente ai confini dell'immagine (valori non compresi tra 0.0 e 1.0). Questo filtraggio garantisce la validità spaziale delle regioni attive, preservando tuttavia quelle parzialmente visibili ai bordi purché il loro nucleo centrale sia all'interno del campo visivo.

(c) **Memorizzazione**: le coordinate validate sono normalizzate e salvate in memoria.

4. **Serializzazione**: Al termine del processo, i dati strutturati sono salvati su disco per essere riutilizzati nelle esecuzioni future.

Procedura di Accesso e Trasformazione del Dato

Durante la fase di addestramento, il modulo è interrogato ripetutamente dai processi paralleli per caricare i singoli magnetogrammi.
La procedura segue un flusso di trasformazione rigoroso:

1. **Lettura del Dato**: Il dato grezzo viene estratto dal file scientifico, con il supporto di meccanismi di controllo per gestire eventuali errori di Input/Output.

2. **Pre-processing**: Se la lettura ha successo, il magnetogramma subisce una catena di trasformazioni per diventare un input valido per la rete neurale:

   (a) **Trattamento delle Anomalie Numeriche**: I dati scientifici possono contenere valori non validi come `Nan` (Not A Number) o `inf` (infinito), spesso causati da errori di misurazione o di calcolo. Non essendo valori numericamente stabili, se passati ad una rete neurale, causerebbero un fallimento nel processo di training. Per questo, è eseguita una pulizia preventiva che sostituisce le eventuali occorrenze di tali valori con il valore neutro `(0.0)`.

   (b) **Clipping Dinamico**: I magnetogrammi presentano una notevole variazione di intensità tra le diverse aree, ma i valori più significativi per l'identificazione delle regioni attive si trovano in un intervallo specifico, mentre valori oltremodo alti o bassi rappresentano -nella maggior parte dei casi- rumori. Per questo motivo è necessaria un'operazione di *clipping*: si "tagliano" i valori del magnetogramma, forzando tutti i pixel al di fuori a rientrare nell'intervallo predefinito, aiutando in questo modo il modello a concentrarsi sulle caratteristiche più significative.

   (c) **Normalizzazione Min-Max**: Poiché le reti neurali apprendono più efficientemente quando i dati in input sono scalati in un intervallo piccolo, si applica una normalizzazione che riscala tutti i valori dei pixel all'interno dell'intervallo `[0,1]`, garantendo la stessa scala di valori che accelera la convergenza del training.

(d) **Ridimensionamento**: Poiché YOLOv7 richiede una dimensione di input standard (impostata a 640x640), i magnetogrammi sono ridimensionati, utilizzando un'interpolazione lineare che rappresenta un ottimo compromesso tra qualità visiva e velocità di calcolo.

(e) **Conversione a 3 Canali**: Dato che YOLOv7 accetta unicamente immagini a tre canali (RGB), è stato necessario rendere i magnetogrammi (a singolo canale) compatibili, per cui il canale unico è stato duplicato tre volte. Ciò significa che non viene aggiunta informazione, ma c'è un semplice riadattamento del dato all'input richiesto dal modello. Infine, avviene la conversione in un tensore PyTorch.

3. **Gestione delle Anomalie**: Se una qualsiasi operazione sul file fallisce (file illeggibile o corrotto), viene cattura l'eccezione. Piuttosto che interrompere l'intero training, lo script stampa a video un avviso con il nome del file problematico e restituisce un tensore di zeri (corrispondente ad un'immagine nera).

L'adozione di questa architettura rende l'intero processo efficiente in termini di tempo e robusto in caso di errori nel dataset. Il tutto è stato fondamentale per l'intero progetto: i dati scientifici sono stati utilizzati come se fossero immagini. Il dettaglio implementativo completo della classe è riportato nel Codice A.2 in Appendice A

3.1.2 Integrazione del Data Loader nella Logica di Training

La predisposizione del modulo di caricamento dati è un passo necessario, ma non sufficiente per l'adattamento completo del modello. Affinchè la nuova logica di gestione dei dati scientifici divenga operativa, è stato necessario integrarla organicamente nel flusso di lavoro.
Di conseguenza, la strategia adottata ha previsto un'estensione strutturale dei motori di addestramento e validazione. L'obiettivo è stato quello di rendere l'architettura capace di accogliere e processare i tensori magnetografici, gestendo le specificità del formato scientifico senza compromettere la stabilità delle procedure di ottimizzazioni originali.

Adattamento della Logica di Training

L'estensione del modulo di caricamento dati ha richiesto un adeguamento strutturale del motore di addestramento. L'obiettivo primario è stato quello di rendere il modello capace di selezionare dinamicamente la strategia di ingestione dei dati più appropriata - standard o scientifica - senza alterare il flusso logico del ciclo di addestramento. Per ottenere ciò, sono state apportate le seguenti strategie architetturali:

- **Integrazione Modulare delle Risorse**
  Il modulo di caricamento personalizzato è stato integrato nell'ambiente

di addestramento in modo condizionale, rendendo le classi disponibili al sistema solo al momento dell'inizializzazione del dataset, garantendo una gestione pulita delle dipendenze.
Vedi Codice A.4 in Appendice A.

- **Strategia di Assemblaggio dei Batch (*Batch Collation*)**
  Di fondamentale importanza è stata l'implementazione di una logica di aggregazione personalizzata per gestire la corretta formazione dei lotti di dati (*batch*). Quando il sistema raggruppa più campioni, è fondamentale mantenere un'associazione univoca tra tensori magnetici e relative etichette spaziali. La procedura sviluppata assegna un indice posizionale a ciascuna annotazione all'interno del batch, risolvendo le ambiguità strutturali tipiche dei dati complessi. Questo ordinamento permette al modello di correlare le predizioni con le *ground truth* per ogni singolo magnetogramma, passaggio essenziale per il calcolo della funzione di perdita (*loss function*).
  La logica di assemblaggio del batch è illustrata nel Codice A.5 in Appendice A.

- **Attivazione Contestuale tramite Configurazione**
  Per garantire una completa flessibilità del sistema, piuttosto che utilizzare un argomento da riga di comando, è stato implementato un meccanismo di *attivazione contestuale* basato sul file di configurazione del dataset. All'avvio, il sistema analizza i metadati di configurazione: la presenza di un identificatore specifico (*flag*) istruisce il framework sulla natura scientifica del dato, innescando automaticamente il meccanismo dedicato senza richiedere interventi manuali.
  Quanto descritto è mostrato nel Codice A.6 in Appendice A.

- **Selezione Dinamica del Flusso di Caricamento Dati**
  Il cambiamento più significativo riguarda la logica di inizializzazione del flusso dati. Grazie al meccanismo di riconoscimento descritto al punto precedente, è stata inserita una struttura di controllo che opera a runtime:

  - in presenza di dati scientifici, il sistema ignora la funzione di caricamento standard ed istanzia un modulo personalizzato, con la strategia di assemblaggio specifica;
  - in caso contrario, il sistema mantiene inalterato il suo comportamento originale.

  La selezione condizionale del Data Loader è riportata nel Codice A.7 in Appendice A.

- **Normalizzazione Condizionale**
  Una modifica critica ha riguardato il pre-processing interno al ciclo di addestramento. La logica originale di YOLOv7 presuppone che il data loader restituisca immagini in formato 8-bit (valori da 0 a 255) e, pertanto, applica una normalizzazione dividendo il tensore delle immagini per 255.0.

Poiché la logica personalizzata esegue già una normalizzazione, tale ulteriore divisione avrebbe reso i dati inadeguati per l'uso. Grazie all'utilizzo del medesimo flag, è stata introdotta una logica condizionale che salta tale operazione di divisione per i magnetogrammi, mantenendola attiva per i dataset standard.

La disattivazione della normalizzazione standard è visibile nel Codice A.8 in Appendice A.
L'approccio metodologico adottato segue il paradigma del *feature flagging*: una tecnica di ingegneria del software che permette di attivare funzionalità specifiche di un'applicazione senza dover apportare modifiche significative. Questo ha garantito l'estensione delle capacità di YOLOv7 in modo modulare, mantenendo la totale retrocompatibilità con l'architettura originale e assicurando che la nuova logica sia eseguita solo quando esplicitamente richiesto dal contesto dei dati.

Estensione della Logica di Validation

Per mantenere una coerenza all'interno dell'intero ciclo di apprendimento,è stato necessario estendere le funzionalità del motore di validazione. Un modello addestrato su dati scientifici deve essere necessariamente validato utilizzando le medesime procedure di pre-elaborazione e caricamento adottate durante il training. Per questo motivo, sono state implementate modifiche strutturali speculari rispetto a quelle introdotte in fase di addestramento:

- **Integrazione delle Risorse di Validazione**
  Analogamente alla fase di training, le classi necessarie per la gestione di dati scientifici sono state rese disponibili all'ambiente di validazione tramite importazioni condizionali, assicurando che il modulo di caricamento specifico sia accessibile solo quando richiesto dalla configurazione.
  Vedi Codice A.9 in Appendice A.

- **Assemblaggio Semplificato dei Batch**
  È stata definita una logica di aggregazione dei dati ottimizzata per la fase di inferenza. A differenza del training, dove l'associazione indice-etichetta è cruciale per il calcolo della *loss*, la validazione richiede un processo più snello. Pertanto, è stata implementata una procedura semplificata, con lo scopo di raggruppare i tensori caricati in un unico batch standard, massimizzando l'efficienza computazionale durante la valutazione.
  L'implementazione semplificata è nel Codice A.10 in Appendice A.

- **Flag di Controllo Interno**
  La funzione principale di test è stata estesa per accettare un nuovo parametro di controllo (*flag*) booleano. Quest'ultimo è trasmesso dal motore di addestramento al termine di ogni epoca per segnalare la natura dei dati in arrivo. Tale indicatore non attiva il caricamento dei dati scientifici, bensì governa i comportamenti successivi del sistema (ad esempio la normalizzazione delle immagini e la visualizzazione dei risultati), assicurando che i

dati scientifici vengano trattati con le procedure appropriate.
La modifica alla firma della funzione è mostrata nel Codice A.11 in Appendice A.

- **Creazione Dinamica del Data Loader**
  Per conferire flessibilità allo script di validazione, è stato implementato un blocco di istanziazione dinamica. Il sistema legge il file di configurazione del dataset e, in base alla presenza della chiave specifica per i dati scientifici:

  – in caso positivo, istanzia il modulo personalizzato con la strategia di assemblaggio semplificata;

  – in caso negativo (o in assenza della chiave), esegue la funzione di caricamento originale, mantenendo la totale compatibilità con i dataset di immagini tradizionali.

  La logica di creazione del loader è nel Codice A.12 in Appendice A.

- **Normalizzazione Condizionale**
  Per evitare alterazioni numeriche dei dati, la normalizzazione dei dei pixel è applicata in modo selettivo. Mentre le immagini standard (0-255) richiedono una divisione per essere portate nel range unitario, i magnetogrammi arrivano al modulo di validazione già normalizzati. Grazie al `flag di controllo`, il sistema inibisce la divisione standard in presenza di dati scientifici, evitando una doppia normalizzazione che comprometterebbe la corretta scala dei valori.
  Il bypass della divisione per 255 è nel Codice A.13 in Appendice A.

- **Riscalamento delle Coordinate**
  Un intervento critico ha riguardato la logica di proiezione delle bounding box (sia quelle predette dal modello che quelle reali). Il modulo personalizzato restituisce le dimensioni originali in un formato diretto (lista `[H,W]`), differente dalla struttura nidificata utilizzata dal data loader standard. Per allineare il codice a questo formato, le chiamate alla funzione di riscalamento sono state semplificate, rimuovendo gli indici non più necessari. Questo garantisce che il confronto tra le predizioni del modello ed etichette di verità avvenga correttamente nello spazio pixel originale dell'immagine, assicurando la validità delle metriche di valutazione.
  L'adattamento delle coordinate è visibile nel Codice A.14 in Appendice A.

- **Visualizzazione Scientifica**
  Infine, per garantire che i risultati della validazione siano visivamente corretti ed interpretabili, le chiamate alle funzioni di plotting sono state modificate per supportare il rendering scientifico dei magnetogrammi (mappatura dei colori e correzione geometrica).
  La chiamata modificata alla funzione di plotting è nel Codice A.15 in Appendice A.

Personalizzazione delle Utility di Visualizzazione

L'ultimo tassello nell'adattamento del framework ha riguardato la corretta rappresentazione grafica dei risultati all'interno del modulo di visualizzazione. Nella sua versione originale, il sistema è progettato per operare esclusivamente su immagini a tre canali (BGR), presupponendo che l'input sia sempre un'immagine standard.
Applicare questa logica direttamente ai magnetogrammi - che sono dati scientifici a singolo canale rappresentanti l'intensità del campo magnetico - avrebbe prodotto un output visivamente incomprensibile (falsi colori e perdita di contrasto), rendendo impossibile la distinzione delle polarità, informazione cruciale per l'analisi fisica.
L'obiettivo è stato, quindi, quello di integrare una logica di visualizzazione personalizzata che si attiva esclusivamente in presenza di dati scientifici. Tale logica include l'applicazione di mappe cromatiche specifiche, la correzione dell'orientamento geometrico e il conseguente adattamento delle coordinate delle bounding box. Allo stesso tempo, è stata mantenuta la piena compatibilità con i dataset standard.
Per raggiungere questo scopo, la procedura di visualizzazione è stata modificata attraverso i seguenti interventi:

- **Estensione della Firma della Funzione**
  La prima modifica ha riguardato l'interfaccia della procedura di plotting. È stato introdotto un parametro opzionale di controllo, il quale agisce come una *feature flag* o `indicatore di contesto`: esso permette ai motori di addestramento e validazione di segnalare al visualizzatore che il batch in arrivo contiene dati scientifici (magnetogrammi) e non immagini standard. Il valore di default impostato a falso garantisce che la funzione mantenga inalterato il suo comportamento in assenza di specifiche istruzioni.
  Vedi Codice A.16 in Appendice A.

- **Normalizzazione Condizionale dell'Input**
  Il flusso originale prevedeva una de-normalizzazione sistematica dei tensori di input (moltiplicazione per 255), assumendo che questi fossero normalizzati nel range `[0,1]` per la visualizzazione a 8-bit. Sebbene corretta per le immagini, questa operazione altera erroneamente i magnetogrammi. La modifica incapsula questa operazione in un blocco condizionale: la de-normalizzazione viene eseguita solo se il dato non è indicato come scientifico, preservando l'integrità numerica dei magnetogrammi per il successivo rendering.
  Il controllo condizionale è nel Codice A.17 in Appendice A.

- **Logica di Rendering Personalizzata per Magnetogrammi**
  Si tratta dell'intervento più sostanziale. All'interno del ciclo che processa ogni immagine del batch, è stata inserita una diramazione logica. Se il `flag scientifico` è attivo, la logica di visualizzazione standard viene sostituita da un *processo di rendering* specifico per i dati scientifici:

1. **Ripristino della Scala Fisica**: i dati sono riportati alla loro scala fisica originale (da [0,1] a [-1500,1500]), utilizzando i parametri di clipping definiti nel Data Loader.
2. **Mappatura Cromatica**: viene applicata una mappa di colori divergente (denominata `seismic`), specifica per evidenziare le polarità magnetiche con colori contrastanti (rosso per il positivo e blu per il negativo).
3. **Conversione RGB**: l'immagine colorata (che ha 4 canali RGBA) è convertita in un'immagine RGB a 8 bit, compatibile con le librerie grafiche.
4. **Correzione dell'Orientamento**: l'immagine viene capovolta verticalmente (*flip*) per allinearsi alle convenzioni di visualizzazione astrofisica standard, dove l'asse Y cresce verso l'alto (contrariamente alle immagini digitali).

Il blocco alternativo mantiene la logica originale per le immagini standard. L'algoritmo di rendering e colorazione è dettagliato nel Codice A.18 in Appendice A.

- **Adattamento delle Coordinate delle Bounding Box**
  Come conseguenza diretta dell'inversione verticale dell'immagine del magnetogramma (descritta al punto precedente), anche le ordinate spaziali delle bounding box necessitano di una trasformazione speculare. Prima che le etichette vengano disegnate sull'immagine, è stato aggiunto un blocco di calcolo che esegue un'inversione matematica. Questo assicura che le etichette (sia le predizioni del modello che la *ground truth*) rimangano geometricamente coerenti con la nuova rappresentazione visiva del magnetogramma.
  La correzione delle coordinate delle bounding box è nel Codice A.19 in Appendice A.

3.2 Approccio alternativo: Pre-conversione del Dataset

In alternativa all'adattamento architetturale di YOLOv7 (descritto ampiamente nella sezione 3.1), è stato esplorato un approccio metodologico che inverte il paradigma di adattamento: *conformare i dati al framework, piuttosto che il framework ai dati*.
Sostanzialmente, si è scelto di trattare il modello preesistente come una *black box*, lasciandone il codice sorgente inalterato. Per conseguire tale obiettivo, è stato implementato un processo di pre-conversione strutturato in due fasi, volto a trasformare l'intero dataset dal formato gerarchico HDF5 al **formato YOLO**. Quest'ultimo impone una struttura specifica:

- Per ogni magnetogramma (in formato immagine .jpg), deve esistere un file di testo corrispondente con il medesimo identificativo.
- Ogni riga del file testuale rappresenta una singola bounding box presente nella relativa immagine.
- Ogni riga è espressa nel formato `<class_id> <x_centro> <y_centro> <larghezza> <altezza>`.

Per lo scopo, è stato sviluppato un flusso di lavoro specifico.

3.2.1 Prima Fase: Transcodifica da HDF5 a YOLO

La prima fase è governata da una **routine di elaborazione batch**, progettata per scansionare l'intero archivio dati ed eseguire, su ogni campione scientifico, due flussi di trasformazione paralleli.

Estrazione e Formattazione delle Annotazioni

Analogamente a quanto implementato nella logica di caricamento dati (descritta nella sezione 3.1.1), il sistema naviga la struttura gerarchica del file per recuperare i metadati spaziali di ogni regione attiva. Vengono eseguiti i medesimi controlli di validità (verifica della completezza degli attributi e consistenza dimensionale) e il calcolo delle coordinate normalizzate. Le coordinate risultanti vengono infine serializzate come stringhe di testo e salvate nel file testuale di annotazione corrispondente.

Trasformazione e Quantizzazione del Segnale

Contestualmente, il modulo processa la matrice dei dati grezzi contenuti nel file scientifico. Questo processo converte l'informazione fisica in un'immagine standard, seguendo una serie di passi:

1. **Pulizia Dati**: rimozione di valori non validi (`Nan` e `inf`);
2. **Clipping Fisico**: saturazione dei valori di campo magnetico nell'intervallo di interesse (`[-1500,1500]`);
3. **Normalizzazione Lineare**: riscalamento dei dati nell'intervallo unitario (`[0.0,1.0]`);
4. **Ridimensionamento Spaziale**: interpolazione dell'immagine alla risoluzione target (formato 640x640 pixel).

Il *passaggio chiave* avviene al termine di questa catena: i dati in virgola mobile (`float`) vengono convertiti in interi a 8-bit. Questa operazione di *quantizzazione* riduce la vasta gamma dinamica dei dati scientifici in 256 valori discreti, rendendoli compatibili con i formati immagine standard. Il risultato viene duplicato su tre canali e codificato con compressione JPEG.

La procedura viene reiterata per le tre partizioni del dataset, producendo in output tre archivi distinti, ciascuno organizzato secondo la gerarchia standard di cartelle richiesta dal framework.
Il codice completo della procedura di conversione è riportato nel Codice A.20 in Appendice A.

3.2.2 Seconda Fase: Post-Processing e Mascheramento dello Sfondo

La procedura di conversione appena descritta introduce un *artefatto sistematico* legato alla gestione dei valori nulli dello sfondo. Durante la fase di pre-processing, i valori esterni al disco solare (`NaN`) vengono sostituiti con il valore neutro `0.0`. A seguito della normalizzazione Min-Max nell'intervallo `[-1500,1500]`, tale valore zero si colloca al centro del range dinamico `0.5` che, nella conversione finale a 8-bit, viene mappato nel valore 127.
Il risultato è un'immagine in cui lo sfondo, anziché essere nero, appare come un **grigio medio**, risultando indistinguibile dalle aree inattive del disco solare stesso (anch'esso rappresentato in scala di grigi). Questa ambiguità visiva riduce drasticamente il contrasto e introduce rumore di fondo che compromette la convergenza dell'addestramento, come evidenziato in figura 3.1.

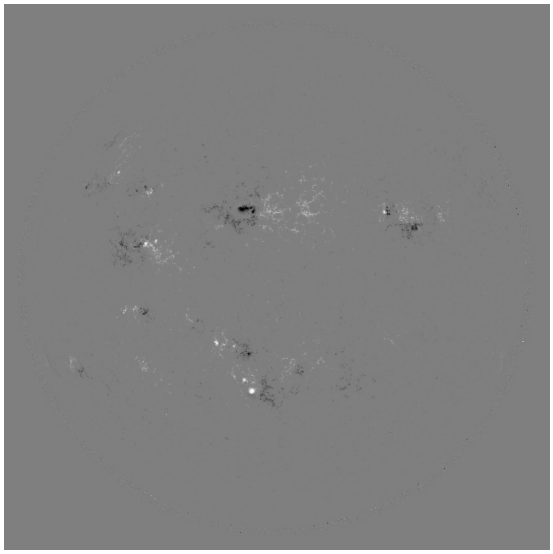


Figura 3.1: Esempio visivo dell'artefatto generato dalla routine di conversione primaria. Come si può osservare, lo sfondo è renderizzato come un grigio medio, rendendo il confine del disco solare (anch'esso in scala di grigi) visivamente ambiguo. Questa mancanza di bordo netto introduce rumore spuro nel dataset.

Per risolvere questa criticità, è stato implementato un **modulo di filtraggio ottico** (post-processing), il cui scopo è isolare geometricamente il disco solare, forzando lo sfondo esterno ad un valore di nero assoluto (0).
La procedura esegue le seguenti operazioni su ciascun campione generato:

1. **Definizione Geometrica del Raggio**: il sistema identifica il centro dell'immagine e calcola il raggio ottimale basandosi sulla dimensione minore del fotogramma. Tale raggio viene ridotto di un *fattore percentuale* (impostato al 96%) per escludere i bordi rumorosi.

2. **Generazione della Maschera**: viene inizializzata una matrice binaria (*maschera*) completamente nera, avente le medesime dimensioni dell'immagine target.

3. **Delimitazione della ROI**: sulla maschera viene disegnato un cerchio bianco pieno (valore 1), corrispondente alla Regione di Interesse (ROI) che contiene il disco solare da preservare.

4. **Applicazione del Filtro**: viene eseguita un'operazione logica bit-a-bit (*bitwise AND*) tra l'immagine originale e la maschera. Questa operazione preserva i pixel all'interno della ROI e azzera forzatamente tutti i pixel esterni, ripristinando il nero puro.

L'algoritmo per l'applicazione della maschera circolare è consultabile nel Codice A.21 in Appendice A.
L'efficacia di questo intervento è immediata: il mascheramento ripristina il corretto rapporto segnale-rumore, garantendo che la rete neurale processi esclusivamente le informazioni pertinenti, come mostrato in figura 3.2

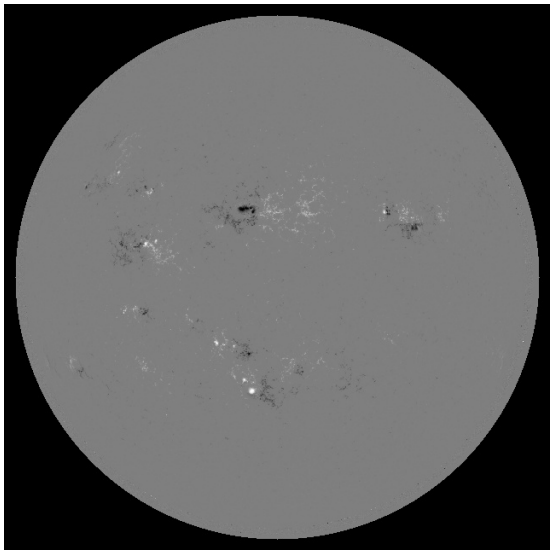


Figura 3.2: Risultato dell'applicazione del modulo di mascheramento sull'immagine precedentemente affetta da artefatti (mostrata nella figura 3.1). L'operazione ha rimosso con successo lo sfondo grigio, forzandolo a nero puro. L'immagine ora presenta un contrasto netto tra il disco solare e lo sfondo, fornendo un input di addestramento pulito e privo di ambiguità.

3.3 Tracking Multi-Oggetto con Norfair

Solo dopo aver terminato l'adattamento di YOLOv7, è stato possibile sviluppare un'applicazione in grado di utilizzare il suddetto modello addestrato per eseguire un'analisi temporale. Le regioni attive solari, infatti, non sono entità statiche, ma evolvono nel tempo, spostandosi e modificando la loro morfologia attraverso la rotazione solare.
Per tracciare tali evoluzioni, è stata integrata -come ampliamente anticipato- la libreria **Norfair**, una soluzione modulare che implementa un approccio di tipo *tracking-by-detection*. In questo paradigma, il modello neurale (YOLOv7) opera su ogni singolo fotogramma in modo indipendente, mentre il modulo di tracciamento si occupa di associare le rilevazioni tra istanti temporali consecutivi, assegnando un identificativo univoco (ID) a ciascun oggetto e mantenendolo consistente nel tempo.

3.3.1 Implementazione del Modulo di Tracciamento

Il cuore del sistema è costituito da un processo di elaborazione sequenziale progettato per processare serie temporali di magnetogrammi. Il funzionamento si articola in quattro fasi logiche:

1. **Gestione Ibrida dei Dati**
   Il modulo è stato ingegnerizzato per gestire indifferentemente sia dati scientifici (HDF5) che immagini standard. Nel primo caso, viene applicata in tempo reale la medesima procedura di pre-processing utilizzata durante il training (gestione dei valori nulli, clipping fisico e normalizzazione), garandendo la totale congruenza statistica tra i dati appresi dal modello e quelli in ingresso al tracker.

2. **Inferenza e Transcodifica**
   Per ogni frame, la rete neurale genera le bounding box delle regioni attive. Una routine di interfaccia converte i tensori di output originali in oggetti strutturati compatibili con il motore di tracciamento, preservando le coordinate spaziali ed i punteggi di confidenza.

3. **Associazione Temporale**
   Il tracker calcola la distanza tra le rilevazioni correnti e la stima della posizione degli oggetti tracciati nei frame precedenti. Data la natura fisica del problema (le macchie solari si muovono con dinamiche fluide e prevedibili), è stata adottata la metrica IoU (Inserction over Union) come criterio di associazione: se la sovrapposizione spaziale tra la predizione attuale e quella passata supera una soglia critica, l'identità dell'oggetto viene preservata.

4. **Sintesi dell'Output**
   Al termine del processo, il sistema aggrega i risultati visuali, generando un video riassuntivo che mostra l'evoluzione dinamica delle regioni attive, con i relativi colori identificativi sovraimpressi per facilitare l'analisi visiva.

L'implementazione algoritmica del tracciamento è riportata nel Codice A.22 in Appendice A.

3.3.2 Valutazione Quantitativa delle Prestazioni

Oltre all'analisi qualitativa visiva, è stato necessario quantificare la robustezza del sistema di tracciamento. A tal fine, è stato sviluppato un modulo di validazione dedicato al calcolo delle metriche standard per il Multi-Object Tracking (MOT), tra cui la *MOTA* (Accuracy) e l'*IDF1* (F1 Score sull'identificazione).
Poiché il calcolo di tali metriche richiede uno standard di annotazione specifico, il modulo implementa una routine di **standardizzazione automatica** che esegue tre compiti critici:

- **Conversione della Ground Truth**: Legge le annotazioni proprietarie e le trasforma nel formato standard internazionale (MOTChallenge), allineando coordinate e identificativi temporali.
- **Inizializzazione dell'Ambiente di Test**: Configura gli accumulatori statici necessari per registrare le associazioni corrette, le mancate rilevazioni e gli scambi di identità (*ID switches*).
- **Benchmarking Frame-by-Frame**: Confronta sequenzialmente le traiettorie generate dal tracker con i dati reali, producendo un report analitico dettagliato.

Questa infrastruttura di test permette do valutare oggettivamente la capacità del modello di mantenere stabile l'identità di una regione attiva anche in presenza di variazioni morfologiche rapide o occlusioni spaziali.
Il codice relativo al calcolo delle metriche è consultabile nel Codice A.23 in Appendice A.

3.4 Considerazioni Conclusive oppure Riepilogo delle Strategie Implementative oppure Sintesi del Capitolo ed Introduzione alla Sperimentazione

In questo capitolo, sono state presentate le strategie implementative adottate per adattare l'architettura YOLOv7 al dominio dell'astrofisica solare. È stato descritto il *duplice approccio metodologico* esplorato: da un lato, l'estensione profonda del framework per la gestione nativa dei dati scientifici, utilizzata come *proof-of-concept* per validare la fattibilità tecnica dell'apprendimento su dati grezzi; dall'altro, lo sviluppo di una catena di pre-elaborazione per la normalizzazione e quantizzazione dei magnetogrammi. Inoltre, sono stati illustrati i moduli accessori sviluppati per garantire la corretta visualizzazione dei risultati fisici e

per estendere le capacità del sistema al tracciamento temporale delle regioni attive.
Queste modifiche costituiscono l'infrastruttura tecnica su cui si basa l'intero lavoro di tesi. Nel prossimo capitolo, verranno analizzate le prestazioni quantitative dei modelli, concentrandosi in particolare sulla **strategia di pre-conversione del dataset**: tale approccio, infatti, si è rivelato il più idoneo per una sperimentazione estensiva, permettendo di valutare la robustezza del sistema tramite le metriche di rilevamento e tracciamento su larga scala.

Appendice A

Codice Sorgente

In questa appendice sono riportati i codici sorgente completi sviluppati per il progetto.

A.1 Generazione del Dataset e Splitting Temporale

```
import pandas as pd  # Libreria per la manipolazione di dati tabellari (DataFrame).
import numpy as np  # Libreria per calcoli numerici e gestione array.
import matplotlib.pyplot as plt  # Libreria per la creazione di grafici.
from datetime import datetime, timedelta  # Classi per la gestione di date e intervalli di tempo.

# --- CARICAMENTO DATI ---
# URL del dataset ufficiale SILSO (Sunspot Index and Long-term Solar Observations).
url = 'https://www.sidc.be/SILSO/DATA/SN_m_tot_V2.0.txt'
# Definizione dei nomi delle colonne per il file di testo (senza header).
columns = ['Year', 'Month', 'FracDate', 'Sunspots', 'StdDev', 'Obs', 'DefProv']
# Legge il file CSV (formato testo separato da spazi) nell'oggetto DataFrame.
df = pd.read_csv(url, delim_whitespace=True, names=columns)
# Crea una colonna 'Date' convertendo Anno e Mese in oggetti datetime (giorno impostato a 1).
df['Date'] = pd.to_datetime(df[['Year', 'Month']].assign(day=1))

# --- FILTRAGGIO TEMPORALE ---
# Filtra i dati per includere solo il Ciclo Solare 24 (Dic 2008 - Dic 2019).
cycle24 = df[(df['Date'] >= '2008-12-01') & (df['Date'] <= '2019-12-31')]

```

```
# --- CONFIGURAZIONE CAMPIONAMENTO ---
# Definisce le proporzioni desiderate per ogni sottoinsieme del dataset.
proportions = {'Train': 0.7, 'Validation': 0.15, 'Test': 0.15}

# Regola la densita' di campionamento per anno in base all'attivita' solare.
# Assegna peso doppio agli anni di massimo solare (2012-2014) per catturare piu' regioni attive.
sampling_weights = {year: 2 if 2012 <= year <= 2014 else 1 for year in range(2011, 2020)}
# Crea una lista di anni ripetuti in base al loro peso (es. 2012 appare 2 volte).
repeated_years = [year for year in range(2011, 2020) for _ in range(sampling_weights[year])]
np.random.seed(42)  # Imposta il seme random per la riproducibilita'.
np.random.shuffle(repeated_years)  # Mescola l'ordine degli anni da campionare.

# --- LOGICA DI SELEZIONE PERIODI ---
period_length = 90  # Durata in giorni di ogni blocco (Train + Val + Test).
separation_days = 15  # Giorni di "cuscinetto" tra i set per evitare data leakage (correlazione temporale).
selected_periods = {'Train': [], 'Validation': [], 'Test': []}  # Dizionario per salvare i risultati.
used_ranges = []  # Lista per tracciare i periodi gia' occupati ed evitare sovrapposizioni.

# Itera su ogni anno selezionato dalla lista pesata.
for year in repeated_years:
    year_start = datetime(year, 1, 1)  # Inizio dell'anno corrente.
    year_end = datetime(year, 12, 31)  # Fine dell'anno corrente.

    # Calcola l'ultimo giorno utile per iniziare un periodo che stia dentro l'anno.
    max_start_day = (year_end - year_start).days - period_length
    if max_start_day <= 0:
        continue  # Salta l'anno se non c'e' abbastanza spazio.

    attempts = 0  # Contatore tentativi per trovare uno spazio libero.
    while attempts < 100:
        # Sceglie un giorno di partenza casuale nell'anno.
        start_offset = np.random.randint(0, max_start_day)
        base_start = year_start + timedelta(days=start_offset)

        # Calcola le date di inizio e fine per il Training set.
        train_end = base_start + timedelta(days=int(proportions['Train'] * period_length))

```

```
57          # Calcola le date per la Validazione (dopo i giorni di
    separazione).
58          val_start = train_end + timedelta(days=separation_days)
59          val_end = val_start + timedelta(days=int(proportions['
    Validation'] * period_length))
60
61          # Calcola le date per il Test (dopo ulteriore separazione
    ).
62          test_start = val_end + timedelta(days=separation_days)
63          test_end = test_start + timedelta(days=int(proportions['
    Test'] * period_length))
64
65          # --- CONTROLLO SOVRAPPOSIZIONI ---
66          # Verifica se l'intervallo proposto [base_start, test_end
    ] tocca intervalli gia' usati.
67          overlap = any(
68              not (test_end < rng[0] or base_start > rng[1])
69              for rng in used_ranges
70          )
71
72          # Se non c'e' sovrapposizione e finiamo entro l'anno
    corrente:
73          if not overlap and test_end <= year_end:
74              # Aggiunge le tuple (start, end) alle liste
    appropriate.
75              # Nota: sottrae 1 giorno alla fine per avere estremi
    inclusivi precisi.
76              selected_periods['Train'].append((base_start,
    train_end - timedelta(days=1)))
77              selected_periods['Validation'].append((val_start,
    val_end - timedelta(days=1)))
78              selected_periods['Test'].append((test_start, test_end
     - timedelta(days=1)))
79
80              # Registra l'intero blocco come "usato".
81              used_ranges.append((base_start, test_end))
82              break  # Esce dal ciclo while (successo).
83
84          attempts += 1  # Riprova se c'era sovrapposizione.
85
86 # --- ESPORTAZIONE DATI ---
87 splits_data = []
88 # Riorganizza i dati in un formato tabellare.
89 for split, periods in selected_periods.items():
90      for start, end in periods:
91          splits_data.append({'Split': split, 'Start': start, 'End'
    : end})
92
93 # Crea un DataFrame e salva il piano di campionamento su CSV.
94 splits_df = pd.DataFrame(splits_data)
95 splits_df.to_csv('solar_cycle_splits.csv', index=False)
96
97 # --- VISUALIZZAZIONE ---
98 fig, ax = plt.subplots(figsize=(15, 7))
```

```
# Traccia l'andamento delle macchie solari nel tempo.
ax.plot(cycle24['Date'], cycle24['Sunspots'], label='Monthly Sunspot Number', color='blue')

# Aggiunge aree colorate per evidenziare i periodi selezionati.
colors = {'Train': 'green', 'Validation': 'yellow', 'Test': 'red'}
for split, periods in selected_periods.items():
    for start, end in periods:
        # axvspan disegna una banda verticale colorata.
        ax.axvspan(start, end, color=colors[split], alpha=0.3, label=f'{split} Period' if start == periods[0][0] else "")

# --- STIMA DEL VOLUME DATI ---
records_count = {}
for split, periods in selected_periods.items():
    # Calcola il numero stimato di immagini: giorni * 24h * 5 (1 immagine ogni 12 min).
    count = sum([(end - start).days * 24 * 5 for start, end in periods])
    records_count[split] = count
    print(f'{split} dataset contains {count} images.')

# Mostra i conteggi direttamente sul grafico.
for idx, (split, count) in enumerate(records_count.items()):
    ax.text(0.02, 0.9 - idx * 0.05, f'{split} images: {count}', transform=ax.transAxes, fontsize=12, color=colors[split])

# Configurazioni finali del grafico (titoli, label, griglia).
ax.set_title('Solar Cycle 24 Sunspot Numbers with Randomized Train-Validation-Test Sampling (2011-2019)')
ax.set_xlabel('Year')
ax.set_ylabel('Sunspot Number')
ax.grid(True)
ax.legend()
plt.tight_layout()
# Salva il grafico risultante su file PNG.
plt.savefig('splits.png', dpi=300)

# Stampa finale di riepilogo in console.
for idx, (split, count) in enumerate(records_count.items()):
    print(f'{split} images: {count}')
```

Codice A.1: Script dataset.py per il campionamento e la generazione degli split

A.2 Data Loader Custom

```
# --- BLOCCO IMPORTAZIONI ---
import torch  # Libreria principale per il deep learning.
from torch.utils.data import Dataset  # Classe base di PyTorch per creare dataset personalizzati.
import h5py  # Libreria specifica per leggere file in formato HDF5.
```

```
import numpy as np  # Libreria per il calcolo numerico, usata per manipolare gli array di dati.
import cv2  # Libreria OpenCV per operazioni sulle immagini, come il ridimensionamento.
import os  # Libreria per interagire con il sistema operativo.
import glob  # Libreria per trovare file che corrispondono a un pattern.
from tqdm import tqdm  # Libreria per gestire visivamente barre di avanzamento.

# --- DEFINIZIONE DELLA CLASSE DATASET ---
# Eredita da 'Dataset' di PyTorch per integrarsi con i suoi strumenti, come il DataLoader.
class DatasetH5(Dataset):
    # --- METODO COSTRUTTORE (`__init__`) ---
    # Viene eseguito una sola volta all'inizio. Prepara il dataset.
    def __init__(self, path, img_size=640, clip_range=(-1500, 1500)):
        # Salva i parametri di configurazione come attributi della classe.
        self.img_size = img_size  # Dimensione finale delle immagini.
        self.clip_min, self.clip_max = clip_range  # Intervallo per il clipping dei valori dei pixel.
        self.class_id = 0  # ID di classe fisso (0), dato che abbiamo solo una classe ("regione attiva").

        # --- LOGICA DI CACHING ---
        # Definisce una sottocartella 'cache' dove verranno salvati i dati pre-elaborati.
        cache_dir = 'cache'
        # Crea la cartella 'cache' se non esiste gia'. 'exist_ok=True' evita errori se la cartella esiste.
        os.makedirs(cache_dir, exist_ok=True)

        # Costruisce un nome univoco per i file di cache basato sul nome della cartella dei dati (es. "Train" o "Validation").
        cache_name = os.path.basename(os.path.normpath(path))
        # Crea il percorso completo per il file di cache delle etichette (es. 'cache/Train_labels.npy').
        label_cache = os.path.join(cache_dir, f'{cache_name}_labels.npy')
        # Crea il percorso completo per il file di cache delle dimensioni originali delle immagini.
        shape_cache = os.path.join(cache_dir, f'{cache_name}_shapes.npy')

        # Cerca tutti i file .h5 nel percorso dato, li ordina e ne salva la lista.
        self.h5_files = sorted(glob.glob(os.path.join(path, '*.h5')))
        # Salva il numero totale di file trovati.
```

```
38          self.n = len(self.h5_files)
39
40          # Controlla se entrambi i file di cache esistono gia'.
41          if os.path.exists(label_cache) and os.path.exists(
    shape_cache):
42              # --- CARICAMENTO VELOCE DA CACHE (AVVII SUCCESSIVI)
    ---
43              print(f"Caricamento rapido da cache per '{cache_name
    }'...")
44              # Carica l'array delle etichette dal file .npy.
45              # 'allow_pickle=True' e' necessario perche' le
    etichette sono in una lista di array.
46              self.labels = np.load(label_cache, allow_pickle=True)
47              # Carica l'array delle dimensioni dal file .npy.
48              self.shapes = np.load(shape_cache)
49              print(f"Cache caricata per {len(self.labels)} file.
    Avvio del training...")
50          else:
51              # --- CREAZIONE DELLA CACHE (PRIMO AVVIO LENTO) ---
52              print(f"Cache non trovata. Creazione della cache per
    '{cache_name}' (lento solo la prima volta)...")
53
54              # Inizializza le liste che conterranno i dati
    estratti.
55              self.labels = []
56              self.shapes = []
57              bad_labels_count = 0  # Contatore per le etichette
    scartate.
58
59              # Itera su ogni file .h5 trovato, mostrando una barra
     di avanzamento.
60              for h5_path in tqdm(self.h5_files, desc=f"Caching
    metadata from {path}"):
61                  try:  # Blocco per gestire errori di lettura dei
    singoli file.
62                      # Apre il file .h5 in modalita' lettura. '
    with' assicura la chiusura automatica.
63                      with h5py.File(h5_path, 'r') as f:
64                          # Estrae il dataset del magnetogramma.
65                          magnetogram_data = f['magnetogram/data']
66                          # Legge le dimensioni originali (altezza,
     larghezza).
67                          orig_h, orig_w = magnetogram_data.shape
68                          # Aggiunge le dimensioni alla lista 'self
    .shapes'.
69                          self.shapes.append([orig_h, orig_w])
70
71                          # Accede al gruppo dei metadati HARP.
72                          harp_group = f['harp/metadata']
73                          image_labels = []  # Lista temporanea per
     le etichette di questa immagine.
74
75                          # Itera su ogni regione attiva trovata
    nei metadati.
```

```
76                          for harp_id in harp_group:
77                              # Estrae gli attributi della regione
    attiva corrente.
78                              harp_attrs = harp_group[harp_id].
    attrs
79
80                              # Definisce le chiavi necessarie per
    calcolare una bounding box.
81                              required_keys = ['CRPIX1', 'CRPIX2',
    'CRSIZE1', 'CRSIZE2']
82                              # Controlla se tutti gli attributi
    necessari sono presenti.
83                              if not all(key in harp_attrs for key
    in required_keys):
84                                  continue  # Se ne manca uno,
    salta questa regione.
85
86                              # Converte le dimensioni in numeri
    decimali.
87                              w_abs = float(harp_attrs['CRSIZE1'])
88                              h_abs = float(harp_attrs['CRSIZE2'])
89
90                              # Controlla che le dimensioni siano
    positive.
91                              if w_abs <= 0 or h_abs <= 0:
92                                  bad_labels_count += 1
93                                  continue  # Se non lo sono,
    scarta l'etichetta.
94
95                              # Calcola le coordinate del centro e
    le dimensioni, normalizzandole rispetto alle dimensioni dell'
    immagine
96                              x_center_norm = float(harp_attrs['
    CRPIX1'] + w_abs / 2) / orig_w
97                              y_center_norm = float(harp_attrs['
    CRPIX2'] + h_abs /2) / orig_h
98                              width_norm = w_abs / orig_w
99                              height_norm = h_abs / orig_h
100
101                             # Controlla che il centro della
    bounding box sia dentro l'immagine.
102                             if not (0.0 < x_center_norm < 1.0 and
     0.0 < y_center_norm < 1.0):
103                                 bad_labels_count += 1
104                                 continue  # Se e' fuori, scarta l
    'etichetta.
105
106                             # Aggiunge l'etichetta valida (
    formato YOLO) alla lista temporanea.
107                             image_labels.append([self.class_id,
    x_center_norm, y_center_norm, width_norm, height_norm])
108
109                         # Aggiunge le etichette di questa
    immagine alla lista principale.
```

```
110                         self.labels.append(np.array(image_labels,
     dtype=np.float32) if image_labels else np.empty((0, 5),
    dtype=np.float32))
111
112                 except Exception as e:  # Se si verifica un
    errore grave durante la lettura.
113                     print(f"Errore grave durante la lettura del
    file {h5_path}: {e}")
114                     # Aggiunge placeholder per mantenere l'
    allineamento degli indici.
115                     self.labels.append(np.empty((0, 5), dtype=np.
    float32))
116                     self.shapes.append([0, 0])
117
118             # Stampa un riepilogo delle etichette scartate, se ce
     ne sono.
119             if bad_labels_count > 0:
120                 print(f"ATTENZIONE: Trovate e scartate {
    bad_labels_count} etichette corrotte.")
121
122             # Converte la lista di liste 'self.shapes' in un
    unico array NumPy.
123             self.shapes = np.array(self.shapes, dtype=np.float64)
124
125             # SALVA I DATI PROCESSATI NELLA CACHE PER USO FUTURO.
126             print(f"Salvataggio della cache in '{path}'...") #
    NOTA: Stampa il percorso dei dati, non della cache
127             np.save(label_cache, self.labels)  # Salva le
    etichette.
128             np.save(shape_cache, self.shapes)  # Salva le
    dimensioni.
129             print("Cache creata. I prossimi avvii saranno
    istantanei.")
130
131     # --- METODO `__len__` ---
132     # Restituisce il numero totale di campioni nel dataset.
133     def __len__(self):
134         return self.n  # Restituisce il numero di file contati
    all'inizio.
135
136     # --- METODO `__getitem__` ---
137     # Carica e restituisce un singolo campione (immagine +
    etichetta) dato un indice.
138     def __getitem__(self, index):
139         # Ottiene percorso e etichette pre-caricate per l'indice
    richiesto.
140         h5_path = self.h5_files[index]
141         labels_tensor = torch.from_numpy(self.labels[index])
142
143         try:  # Blocco per gestire errori di apertura file (es.
    file corrotti).
144             # Tenta di aprire il file H5 e leggere i dati dell'
    immagine.
145             with h5py.File(h5_path, 'r') as f:
```

```
                data = f['magnetogram/data'][:]  # Carica l'intero array in memoria.

            # Pulisce i dati da eventuali valori non numerici (NaN/inf).
            if np.isnan(data).any() or np.isinf(data).any():
                data = np.nan_to_num(data, nan=0.0, posinf=0.0, neginf=0.0)

            # Pre-processa l'immagine: clipping, normalizzazione e ridimensionamento.
            clipped_data = np.clip(data, self.clip_min, self.clip_max)
            normalized_data = (clipped_data - self.clip_min) / (self.clip_max - self.clip_min)
            resized_image = cv2.resize(normalized_data, (self.img_size, self.img_size), interpolation=cv2.INTER_LINEAR)

            # Converte a 3 canali (duplicando il canale unico) per compatibilita' con YOLOv7.
            image_rgb = np.stack([resized_image] * 3, axis=-1)
            # Converte l'array NumPy in un tensore PyTorch e riordina le dimensioni in [C, H, W].
            image_tensor = torch.from_numpy(image_rgb.transpose(2, 0, 1)).float()

            # Restituisce il campione completo.
            return image_tensor, labels_tensor, h5_path, self.shapes[index]

        except Exception as e:  # Se si verifica un qualsiasi errore durante il caricamento.
            # Stampa un avviso e ignora il dato corrotto.
            print(f"\nATTENZIONE: Ignorato file corrotto o illeggibile: {os.path.basename(h5_path)}")

            # Restituisce un'immagine nera per non interrompere il training.
            placeholder_image = torch.zeros((3, self.img_size, self.img_size))
            return placeholder_image, labels_tensor, h5_path, self.shapes[index]
```

Codice A.2: Classe DatasetH5 per il caricamento dei file .h5

A.3 Scaricamento e Processamento Dati Solari

```
import pandas as pd  # Libreria per leggere il file CSV con gli split temporali.
import os  # Libreria per operazioni sul file system (creazione cartelle, eliminazione file).
import drms  # Client ufficiale per accedere ai dati solari JSOC/DRMS.
```

```
import h5py  # Libreria per creare e gestire file HDF5 compressi.
import time  # Libreria per gestire pause e timeout.
import sunpy.map  # Libreria specifica per leggere e manipolare mappe solari (file FITS).
from sunpy.net import jsoc  # Modulo di SunPy per interfacciarsi con JSOC.
import multiprocessing as mp  # Libreria per il calcolo parallelo (gestione processi).
from datetime import datetime, timedelta  # Classi per la gestione temporale.

# Imposta l'email necessaria per effettuare richieste ai server DRMS.
EMAIL = 'edoria2011@gmail.com'

# Classe personalizzata che estende h5py.File per una scrittura sicura.
# Garantisce che non rimangano file corrotti su disco in caso di crash.
class SafeH5File(h5py.File):
    def __init__(self, filename, *args, **kwargs):
        super().__init__(filename, *args, **kwargs)
        self._store_filename = filename  # Memorizza il percorso del file.

    # Metodo chiamato automaticamente all'uscita dal blocco 'with'.
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close()  # Chiude il file.
        # Se si e' verificata un'eccezione (errore) durante la scrittura...
        if exc_type:
            print("An error occurred, cleaning up...")
            # ...cancella il file parziale/corrotto dal disco.
            if os.path.exists(self._store_filename):
                os.remove(self._store_filename)
        return False

# Funzione per gestire i download falliti con tentativi ripetuti (retry logic).
def export_with_retries(client, query, sleep_time=3, max_retries=100):
    retries = 0
    while retries < max_retries:
        try:
            # Tenta di esportare i dati richiedendo un URL FITS.
            response = client.export(query, method='url', protocol='fits')
            # Status 4 = Completato, Status 7 = Richiesta in coda.
            if response.status != 7:
                return response  # Successo (o errore non recuperabile).
            elif response.status == 7:
```

```
                # Se il server e' pieno, aspetta e riprova.
                print(f"[RETRY] Too many pending requests (status={response.status}). Waiting {sleep_time}s...")
            else:
                print(f"[RETRY] Unknown status={response.status}. Waiting {sleep_time}s...")
        except Exception as e:
            raise  # Se l'errore e' grave, lo lascia gestire al livello superiore.

        time.sleep(sleep_time)  # Attende prima del prossimo tentativo.
        retries += 1

    # Se supera il numero massimo di tentativi, lancia un errore.
    raise RuntimeError(f"Exceeded max retries ({max_retries}) for query: {query}")

# Funzione "worker" che processa un intero intervallo di date (es. un blocco di Training).
def process_range(args):
    split, start_str, end_str, output_dir = args  # Disimballa gli argomenti.

    print(f"[PID {mp.current_process().pid}] Starting date range {start_str} to {end_str}")

    # Inizializza i client per la connessione al database solare.
    client = drms.Client(email=EMAIL)
    jsoc_client = jsoc.JSOCClient()

    # Crea la cartella di output specifica per lo split (es. output/Train).
    split_path = os.path.join(output_dir, split)
    os.makedirs(split_path, exist_ok=True)

    # Converte le stringhe delle date in oggetti datetime.
    start = datetime.strptime(start_str, "%Y-%m-%d")
    end = datetime.strptime(end_str, "%Y-%m-%d")

    current = start
    # Ciclo principale che avanza di 12 minuti alla volta (cadenza strumento HMI).
    while current <= end:
        # Formatta la data nel formato richiesto da JSOC (TAI time).
        current_str = current.strftime('%Y.%m.%d_%H:%M:%S_TAI')
        # Definisce il nome del file di output .h5.
        filename = f'hmi_magnetogram_{current.strftime("%Y-%m-%d_%H-%M-%S")}.h5'
        output_h5 = os.path.join(split_path, filename)

        # Salta il download se il file esiste gia' (resume capability).
```

```
84          if os.path.exists(output_h5):
85              print(f"[{current}] Skipping - file already exists: {
    output_h5}")
86              current += timedelta(minutes=12)
87              continue
88
89          # --- FASE 1: DOWNLOAD E PREPARAZIONE DATI IN MEMORIA ---
90          try:
91              print(f"[{current}] Requesting HMI magnetogram data
    ...")
92              # Costruisce la query per il magnetogramma (serie hmi
    .M_720s).
93              # QUALITY<65536 filtra le immagini con errori noti.
94              hmi_query = f'hmi.M_720s[{current_str}][? (QUALITY
    <65536) ?]'
95              hmi_response = export_with_retries(client, hmi_query)
96
97              print(f"[{current}] Creating SunPy map from FITS URL
    ...")
98              # Scarica il FITS dall'URL e crea una mappa SunPy.
99              hmi_map = sunpy.map.Map(hmi_response.urls['url'][0])
100             data = hmi_map.data  # Estrae la matrice numerica (
    immagine).
101             metadata = hmi_map.meta  # Estrae l'header FITS (
    metadati immagine).
102
103             print(f"[{current}] Requesting HARP metadata...")
104             # Cerca i metadati delle regioni attive (HARP) per
    quello stesso istante.
105             harp_response = jsoc_client.search(
106                 jsoc.attrs.Time(current_str, current_str),
107                 jsoc.attrs.Series('hmi.Mharp_720s'),
108             )
109
110         except Exception as e:
111             # In caso di errore di rete o dati mancanti, stampa l
    'errore e salta al prossimo step.
112             print(f"[{current}] Error during data fetch or
    parsing: {e}")
113             current += timedelta(minutes=12)
114             continue
115
116         # --- FASE 2: SCRITTURA SICURA SU DISCO ---
117         try:
118             # Usa la classe SafeH5File per garantire l'integrita'
     del file.
119             with SafeH5File(output_h5, 'w') as f:
120                 # Salva la matrice del magnetogramma compressa
    con GZIP (livello 9).
121                 f.create_dataset('magnetogram/data', data=data,
    compression="gzip", compression_opts=9)
122
123                 # Crea un gruppo per i metadati del magnetogramma
    .
```

```
                meta_group = f.create_group('magnetogram/metadata')
                for key in metadata:
                    value = metadata.get(key)
                    try:
                        # Prova a salvare il valore direttamente.
                        meta_group.attrs[key] = value
                    except TypeError:
                        # Se il tipo non e' supportato da HDF5, lo converte in stringa.
                        meta_group.attrs[key] = str(value)

                # Crea un gruppo per i metadati HARP (Regioni Attive).
                harp_group = f.create_group('harp/metadata')
                for record in harp_response:
                    harpnum = record["HARPNUM"]
                    harp_id = f"HARP_{harpnum}"
                    # Crea un sottogruppo per ogni HARP identificata.
                    harp_entry = harp_group.create_group(harp_id)

                    # Salva tutte le colonne della risposta HARP come attributi.
                    for key in harp_response.columns:
                        value = record[key]
                        try:
                            harp_entry.attrs[key] = value
                        except TypeError:
                            harp_entry.attrs[key] = str(value)

            print(f"[{current}] File saved: {output_h5}")

        except Exception as e:
            print(f"[{current}] Error saving file: {e}")
            # Pulizia extra nel caso SafeH5File non avesse intercettato tutto.
            if os.path.exists(output_h5):
                os.remove(output_h5)

        # Avanza l'iteratore temporale di 12 minuti.
        current += timedelta(minutes=12)

    print(f"[PID {mp.current_process().pid}] Finished processing range {start_str} to {end_str}")

# Funzione principale che orchestra l'esecuzione dei job.
def run_all(csv_path, output_root="output", num_workers=4):
    print(f"Reading from: {csv_path}")
    df = pd.read_csv(csv_path)

    # Crea una lista di tuple (task) da assegnare ai worker.
    tasks = [(row["Split"], row["Start"], row["End"], output_root) for _, row in df.iterrows()]
```

```
    print(f"Launching {len(tasks)} tasks with {num_workers} worker(s)...")

    # Codice predisposto per il multiprocessing (attualmente esegue in sequenza per debug).
    # with mp.Pool(processes=num_workers) as pool:
    #     pool.map(process_range, tasks)

    # Esegue i task sequenzialmente nel processo corrente.
    for task in tasks: process_range(task)

    print("All tasks completed.")

if __name__ == '__main__':
    # Punto di ingresso dello script.
    run_all('solar_cycle_splits.csv', output_root="/projects/data/physics/data", num_workers=8)
```

Codice A.3: Script download.py per il recupero dei dati da JSOC

A.4 Modifiche a train.py

```
# Importa la classe personalizzata per la gestione dei dataset in formato HDF5.
from utils.dataset_h5 import DatasetH5
```

Codice A.4: Importazione in train.py

```
# Funzione custom per raggruppare campioni in un batch. Aggiunge l'indice del batch a ogni etichetta per l'associazione.
def h5_collate_fn(batch):
    # Separa gli elementi del batch (immagini, etichette, percorsi, dimensioni)
    imgs, labels, paths, shapes = zip(*batch)

    batched_labels = [] # Lista per accumulare le etichette indicizzate

    # Itera su ogni campione (immagine + etichette) nel batch
    # 'i' sara' l'indice dell'immagine nel batch (0, 1, 2, ...)
    for i, label in enumerate(labels):

        # Processa solo se l'immagine ha almeno un'etichetta
        if label.shape[0] > 0:

            # Crea un tensore riempito con l'indice (i) dell'immagine
            # Avra' la stessa n. di righe delle etichette di questa immagine
            batch_idx = torch.full((label.shape[0], 1), i,
                                   device=imgs[0].device)

```

```
            # Concatena l'indice (colonna 0) alle etichette [classe, x, y, w, h]
            label_with_batch_idx = torch.cat((batch_idx,
                                              label.to(imgs[0].device)), 1)

            # Aggiunge il nuovo tensore [i, classe, x, y, w, h] alla lista
            batched_labels.append(label_with_batch_idx)

    # Se sono state trovate etichette in questo batch...
    if len(batched_labels) > 0:
        # ...le unisce tutte in un unico tensore [N_tot_labels, 6]
        targets = torch.cat(batched_labels, 0)
    else:
        # ...altrimenti crea un tensore vuoto (con 6 colonne) per coerenza
        targets = torch.empty(0, 6, device=imgs[0].device)

    # Restituisce le immagini (stackate in un unico tensore batch)
    # e il tensore unico delle etichette (targets)
    return torch.stack(imgs, 0), targets, paths, shapes
```

Codice A.5: Funzione Collate per il Training

```
# Apre e legge il file di configurazione del dataset (es. harp.yaml)
with open(opt.data) as f:
    data_dict = yaml.load(f, Loader=yaml.SafeLoader)

# Controlla se la chiave 'is_h5' esiste e ha valore True; altrimenti, imposta False
is_h5_dataset = data_dict.get('is_h5', False)
```

Codice A.6: Lettura configurazione .yaml

```
# Logica condizionale per la selezione dinamica del data loader

# Controlla il flag booleano letto dal file .yaml
if is_h5_dataset:
    logger.info("Utilizzo del DataLoader custom per dataset .h5")

    # Crea un'istanza del dataset personalizzato
    dataset = DatasetH5(path=train_path, img_size=imgsz)

    # Imposta il sampler (necessario per il training distribuito)
    sampler = torch.utils.data.distributed.DistributedSampler(dataset) if rank != -1 else None

    # Crea il DataLoader di PyTorch usando la collate_fn personalizzata
    dataloader = torch.utils.data.DataLoader(dataset,
                                             batch_size=batch_size,
```

```
                                                shuffle=sampler is
    None and not opt.rect,
                                                num_workers=opt.
    workers,
                                                sampler=sampler,
                                                pin_memory=True,
                                                collate_fn=
    h5_collate_fn)
else:
    # Logica originale di YOLOv7 per dataset standard
    dataloader, dataset = create_dataloader(train_path, imgsz,
    batch_size, gs, opt,
                                                hyp=hyp, augment=True
    , cache=opt.cache_images, rect=opt.rect, rank=rank,
                                                world_size=opt.
    world_size, workers=opt.workers,
                                                image_weights=opt.
    image_weights, quad=opt.quad, prefix=colorstr('train: '))
```

Codice A.7: Selezione dinamica del DataLoader

```
if is_h5_dataset:
    # Per i dati H5, i valori sono gia' normalizzati [0,1].
    imgs = imgs.to(device, non_blocking=True).float()
else:
    # Per le immagini normali, mantengo la logica originale.
    imgs = imgs.to(device, non_blocking=True).float()/255.0
```

Codice A.8: Normalizzazione condizionale nel training

A.5 Modifiche a test.py

```
# Importa la classe personalizzata per la gestione dei dataset in
    formato HDF5.
from utils.dataset_h5 import DatasetH5

# Importa la funzione di default di PyTorch per l'assemblaggio
    dei batch.
from torch.utils.data.dataloader import default_collate
```

Codice A.9: Importazioni in test.py

```
# Funzione custom per raggruppare i dati provenienti da DatasetH5
    in un batch.
def h5_collate_fn(batch):
# Delega l'assemblaggio del batch alla funzione di default di
    PyTorch,  che impila automaticamente i campioni in un unico
    tensore.

    return default_collate(batch)
```

Codice A.10: Funzione Collate semplificata per il Test

```
def test(data,
         ...,
         is_magnetogram=False):
```

Codice A.11: Nuovo parametro nella funzione test

```
# Legge il flag 'is_h5' dal dizionario 'data' (caricato dal file .yaml).
# Se la chiave non esiste, imposta 'False' come valore di default.
is_h5_dataset = data.get('is_h5', False)

# Controlla se il dataset e' di tipo H5.
if is_h5_dataset:
    # Stampa un messaggio informativo nel log.
    print("Utilizzo del DataLoader custom per dataset .h5")

    # Istanzia la classe DatasetH5 personalizzata.
    # 'data[task]' contiene il percorso alla cartella dei dati (es. 'val').
    dataset = DatasetH5(path=data[task], img_size=imgsz)

    # Crea un DataLoader standard di PyTorch utilizzando il dataset H5.
    dataloader = torch.utils.data.DataLoader(dataset,
                                             # Imposta la dimensione del batch.
                                             batch_size=batch_size,
                                             # Disattiva lo 'shuffle' (mescolamento) per la validazione/test.
                                             shuffle=False,
                                             # Imposta il numero di processi paralleli per caricare i dati.
                                             num_workers=8,
                                             # Abilita il 'pinning' della memoria per trasferimenti piu' veloci alla GPU.
                                             pin_memory=True,
                                             # Specifica la funzione custom per assemblare i campioni in un batch.
                                             collate_fn=h5_collate_fn)
# Se il dataset non e' di tipo H5...
else:
    # ...esegue la logica originale di YOLOv7.
    # Chiama la funzione 'create_dataloader' standard del framework.
    dataloader = create_dataloader(data[task], imgsz, batch_size, gs, opt, pad=0.5, rect=True,
                                   prefix=colorstr(f'{task}: '))[0]
```

Codice A.12: Creazione Dataloader in test.py

```
# Salta la divisione se e' un magnetogramma
    if not is_magnetogram:
        img /= 255.0
```

Codice A.13: Normalizzazione condizionale nel test

```
# ... all'interno del ciclo sulle predizioni ...

# 1. Riscalare le coordinate delle PREDIZIONI
# La chiamata originale (shapes[si][0]) e' stata modificata in 'shapes[si]'
scale_coords(img[si].shape[1:], predn[:, :4], shapes[si])  # native-space pred

# ... all'interno del blocco 'if nl:' (se ci sono etichette reali) ...

# 2. Riscalare le coordinate delle ETICHETTE REALI (target)
# Anche qui, la chiamata e' stata adattata a 'shapes[si]'
tbox = xywh2xyxy(labels[:, 1:5])
scale_coords(img[si].shape[1:], tbox, shapes[si])  # native-space labels
```

Codice A.14: Adattamento scale_coords

```
# Avvia un thread separato per la funzione 'plot_images' (per non bloccare il loop principale).
# 'args' passa gli argomenti posizionali (immagine, etichette, percorso, nome file, nomi classi).
# 'kwargs' (key-word arguments) passa un dizionario:
#   'is_magnetogram' viene impostato con il valore del flag 'is_magnetogram'.
# Questo permette a 'plot_images' di sapere che tipo di immagine sta visualizzando.
Thread(target=plot_images, args=(img, targets, paths, f, names),
       kwargs={'is_magnetogram': is_magnetogram},
       daemon=True).start()
```

Codice A.15: Chiamata al thread di plotting

A.6 Modifiche a plots.py

```
# La firma della funzione originale terminava con 'max_subplots=16'.
# E' stato aggiunto il nuovo argomento 'is_magnetogram=False' alla fine.
def plot_images(images, targets, paths=None, fname='images.jpg', names=None,
                max_size=640, max_subplots=16, is_magnetogram=False):
    # Il resto del corpo della funzione...
```

Codice A.16: Firma funzione plot_images

```
# ... (codice precedente) ...
    if isinstance(targets, torch.Tensor):
        targets = targets.cpu().numpy()

    # La riga originale 'images *= 255' e' stata resa condizionale.
    # Si esegue solo se NON e' un magnetogramma E se i valori sono normalizzati [0,1].
    if not is_magnetogram and np.max(images[0]) <= 1:
        images *= 255

    tl = 3  # line thickness
    # ... (codice successivo) ...
```

Codice A.17: Check normalizzazione plot

```
# ... (dentro il ciclo for i, img in enumerate(images)) ...

        # Inizia la logica condizionale basata sul flag
        if is_magnetogram:
            # --- Blocco personalizzato per magnetogrammi ---

            # 1. De-normalizza l'immagine (da [0,1] a [-1500, 1500])
            #    (img[0] seleziona il singolo canale)
            magnetogram_data = img[0] * 3000 - 1500

            # 2. Imposta i limiti di contrasto per la visualizzazione
            contrast_min = -300
            contrast_max = 300

            # 3. Ottiene la colormap 'seismic' (Rosso-Bianco-Blu)
            cmap = plt.get_cmap('seismic')
            # 4. Normalizza i dati tra [0,1] in base al contrasto
            norm = matplotlib.colors.Normalize(vmin=contrast_min, vmax=contrast_max)

            # 5. Applica la colormap ai dati normalizzati
            colored_img = cmap(norm(magnetogram_data))
            # 6. Converte da RGBA [0,1] a RGB [0,255] (formato uint8)
            img_rgb = (colored_img[:, :, :3] * 255).astype(np.uint8)

            # 7. Capovolge l'immagine verticalmente (flip Asse X)
            img = cv2.flip(img_rgb, 0)

            # 8. Ridimensiona se necessario (come da logica originale)
            if scale_factor < 1:
                img = cv2.resize(img, (w, h))
        else:
            # --- Blocco Originale per immagini BGR/RGB ---
```

```
            img = img.transpose(1, 2, 0) # Converte da [C, H, W] a [H, W, C]
            if scale_factor < 1:
                img = cv2.resize(img, (w, h))

        # Disegna l'immagine (ora in formato RGB) sul mosaico
        mosaic[block_y:block_y + h, block_x:block_x + w, :] = img
        # ... (codice successivo)
```

Codice A.18: Logica di rendering magnetogrammi

```
# ... (codice per il calcolo dei box) ...
        if boxes.shape[1]:
            if boxes.max() <= 1.01:
                boxes[[0, 2]] *= w
                boxes[[1, 3]] *= h
            elif scale_factor < 1:
                boxes *= scale_factor

        # Aggiunta: Blocco per invertire le coordinate Y se e' un magnetogramma
        if is_magnetogram:
            # Inverte le coordinate Y (indice 1 e 3) rispetto all'altezza (h)
            # Questo compensa il 'cv2.flip(..., 0)' applicato all'immagine
            boxes[[1, 3]] = h - boxes[[1, 3]]

        # Sposta i box nella loro posizione sul mosaico (logica originale)
        boxes[[0, 2]] += block_x
        boxes[[1, 3]] += block_y

        # ... (codice per disegnare i box) ...
```

Codice A.19: Inversione coordinate Y

A.7 Script di Conversione

```
import h5py # Libreria per leggere e scrivere file HDF5
import numpy as np # Libreria per il calcolo numerico
import cv2 # Libreria OpenCV per l'elaborazione delle immagini
import os # Libreria per interagire con il sistema operativo
import glob # Libreria per trovare file sul disco tramite pattern
from tqdm import tqdm # Libreria per mostrare una barra di progresso
import argparse # Libreria per gestire gli argomenti passati da riga di comando
import sys # Libreria per interagire con il sistema
import zipfile # Libreria per creare e scrivere file .zip
import concurrent.futures # Libreria per la gestione del multithreading

```

```
# --- Impostazioni Globali ---
IMG_SIZE = 640 # Dimensione fissa (larghezza e altezza) delle immagini di output
CLIP_MIN, CLIP_MAX = -1500, 1500 # Valori minimi e massimi per il clipping dei dati scientifici
CLASS_ID = 0 # ID di classe fisso per le etichette (abbiamo solo "regioni attive")
# Numero di thread "lavoratori" da usare per processare i file H5 in parallelo
N_WORKERS = 32
# ---

def process_file_h5(h5_path):
    """
    Processa un singolo file H5.
    Questa funzione legge i dati, estrae le etichette, processa l'immagine,
    e restituisce i dati pronti per essere scritti nello zip.
    E' progettata per essere eseguita in un thread separato.
    """

    # Estrae il nome base del file (es. 'M_720s_20101210_000000_TAI_20101210_001159_TAI_01300')
    base_name = os.path.splitext(os.path.basename(h5_path))[0]
    # Definisce il percorso dell'immagine DENTRO l'archivio .zip
    img_arcname = os.path.join('images', f"{base_name}.jpg")
    # Definisce il percorso del file di etichette DENTRO l'archivio .zip
    label_arcname = os.path.join('labels', f"{base_name}.txt")

    try:
        # Apre il file H5 in modalita' lettura ('r')
        with h5py.File(h5_path, 'r') as f:

            # --- 1. Estrai Dati Immagine ---
            # Accede al dataset del magnetogramma
            magnetogram_data = f['magnetogram/data']
            # Legge le dimensioni originali (altezza, larghezza)
            orig_h, orig_w = magnetogram_data.shape
            # Carica l'intero array dei dati in memoria
            data = magnetogram_data[:]

            # --- 2. Estrai Etichette (Bounding Box) ---
            # Accede al gruppo dei metadati
            harp_group = f['harp/metadata']
            # Lista per contenere le etichette di QUESTA immagine
            image_labels = []

            # Itera su ogni regione attiva (HARP) trovata nei metadati
            for harp_id in harp_group:
                # Estrae gli attributi della regione corrente
                harp_attrs = harp_group[harp_id].attrs

```

```
58                  # Lista delle chiavi necessarie per definire un
    box
59                  required_keys = ['CRPIX1', 'CRPIX2', 'CRSIZE1', '
    CRSIZE2']
60                  # Controlla se tutte le chiavi necessarie sono
    presenti
61                  if not all(key in harp_attrs for key in
    required_keys):
62                      continue # Se ne manca una, salta questa
    etichetta
63
64                  # Converte le dimensioni (in pixel) in float
65                  w_abs = float(harp_attrs['CRSIZE1'])
66                  h_abs = float(harp_attrs['CRSIZE2'])
67
68                  # Validazione: scarta etichette con dimensioni
    non positive
69                  if w_abs <= 0 or h_abs <= 0:
70                      continue # Salta questa etichetta
71
72                  # Calcola le coordinate normalizzate in formato
    YOLO (centro_x, centro_y, w, h)
73                  x_center_norm = (float(harp_attrs['CRPIX1']) +
    w_abs / 2) / orig_w
74                  y_center_norm = (float(harp_attrs['CRPIX2']) +
    h_abs / 2) / orig_h
75                  width_norm = w_abs / orig_w
76                  height_norm = h_abs / orig_h
77
78                  # Validazione: scarta etichette il cui centro e'
    fuori dall'immagine
79                  if not (0.0 < x_center_norm < 1.0 and 0.0 <
    y_center_norm < 1.0):
80                      continue # Salta questa etichetta
81
82                  # Aggiunge l'etichetta valida alla lista
83                  image_labels.append([CLASS_ID, x_center_norm,
    y_center_norm, width_norm, height_norm])
84
85              # --- 3. Processa Immagine (come in DatasetH5) ---
86              # Pulizia: sostituisce NaN e Infinito con 0.0
87              if np.isnan(data).any() or np.isinf(data).any():
88                  data = np.nan_to_num(data, nan=0.0, posinf=0.0,
    neginf=0.0)
89
90              # Clipping: "taglia" i valori all'intervallo definito
91              clipped_data = np.clip(data, CLIP_MIN, CLIP_MAX)
92              # Normalizzazione Min-Max: porta i valori nell'
    intervallo [0.0, 1.0]
93              normalized_data = (clipped_data - CLIP_MIN) / (
    CLIP_MAX - CLIP_MIN)
94              # Ridimensionamento: porta l'immagine alla dimensione
     640x640
95              resized_image = cv2.resize(normalized_data, (IMG_SIZE
```

```
    , IMG_SIZE), interpolation=cv2.INTER_LINEAR)
96
97              # Conversione 8-bit: "appiattisce" i dati float [0,1]
     in interi [0,255]
98              image_uint8 = (resized_image * 255.0).astype(np.uint8
    )
99              # Conversione 3 Canali: duplica il canale unico per
    creare un'immagine RGB
100             image_rgb = np.stack([image_uint8] * 3, axis=-1)
101
102             # Codifica in memoria: converte l'array NumPy in un
    formato JPG (binario)
103             is_success, img_buffer = cv2.imencode('.jpg',
    image_rgb)
104             if not is_success:
105                 # Se la codifica fallisce, solleva un'eccezione
106                 raise Exception("Impossibile codificare l'
    immagine in JPG.")
107
108             # --- 4. Prepara Etichette (Formato TXT) ---
109             # Crea una lista di stringhe, una per ogni etichetta
110             label_lines = [f"{lbl[0]} {lbl[1]} {lbl[2]} {lbl[3]}
    {lbl[4]}" for lbl in image_labels]
111             # Unisce le stringhe con un "a capo", pronto per
    essere scritto su file
112             label_content = "\n".join(label_lines)
113
114             # Restituisce tutti i dati necessari al thread
    principale per la scrittura
115             return (img_arcname, img_buffer.tobytes(),
    label_arcname, label_content)
116
117     except Exception as e:
118         # Gestione degli errori (es. file H5 corrotto)
119         print(f"\nATTENZIONE: Fallimento nel processare {h5_path
    }: {e}")
120         # Crea un'immagine nera (placeholder)
121         placeholder_img = np.zeros((IMG_SIZE, IMG_SIZE, 3), dtype
    =np.uint8)
122         # Codifica l'immagine nera
123         is_success, img_buffer = cv2.imencode('.jpg',
    placeholder_img)
124         # Restituisce l'immagine nera e un file etichette vuoto
125         return (img_arcname, img_buffer.tobytes(), label_arcname,
     "")
126
127
128 def main():
129     """
130     Funzione principale che orchestra il processo.
131     Gestisce gli argomenti da riga di comando, trova i file,
132     avvia il ThreadPool ed esegue la scrittura del file .zip.
133     """
134
```

```
135     # Configura il parser per gli argomenti da riga di comando
136     parser = argparse.ArgumentParser(description="Converte il
    dataset H5 in un singolo file .zip in formato YOLO (
    multithread).")
137     parser.add_argument('--source-dir', type=str, required=True,
    help="Cartella locale contenente i file .h5")
138     parser.add_argument('--zip-file', type=str, required=True,
    help="Percorso del file .zip di output")
139     parser.add_argument('--workers', type=int, default=N_WORKERS,
     help="Numero di thread 'produttori'")
140     args = parser.parse_args() # Legge gli argomenti forniti dall
    'utente
141
142     # Trova tutti i file .h5 nella cartella sorgente (anche
    sottocartelle)
143     h5_files = sorted(glob.glob(os.path.join(args.source_dir, '**
    ', '*.h5'), recursive=True))
144     if not h5_files:
145         # Se non trova file, stampa un errore ed esce
146         print(f"Errore: Nessun file .h5 trovato in {args.
    source_dir}")
147         sys.exit(1)
148
149     print(f"Trovati {len(h5_files)} file .h5. Avvio della
    conversione in '{args.zip_file}'...")
150     print(f"Uso di {args.workers} thread lavoratori.")
151
152     # Crea e gestisce un pool di thread (max 'args.workers'
    thread attivi contemporaneamente)
153     with concurrent.futures.ThreadPoolExecutor(max_workers=args.
    workers) as executor:
154
155         # Apre il file .zip in modalita' scrittura ('w') con
    compressione
156         with zipfile.ZipFile(args.zip_file, 'w', compression=
    zipfile.ZIP_DEFLATED) as zf:
157
158             # Sottomette tutti i lavori (chiama 'process_file_h5'
     per ogni file)
159             # 'futures' e' una lista di "promesse" di risultati
    futuri
160             futures = [executor.submit(process_file_h5, h5_path)
    for h5_path in h5_files]
161
162             # Il thread principale (questo) ora itera sui
    risultati man mano che
163             # i thread "lavoratori" li completano (non in ordine
    di invio)
164             # 'tqdm' crea una barra di progresso per questa
    iterazione
165             for future in tqdm(concurrent.futures.as_completed(
    futures), total=len(h5_files), desc="Conversione in corso"):
166
167                 # Ottiene il risultato dal thread completato
```

```
                result = future.result()

                if result:
                    # Estrae i dati restituiti dalla funzione
                    img_arcname, img_buffer, label_arcname,
    label_content = result

                    # Scrive l'immagine (binaria) nel file .zip
                    zf.writestr(img_arcname, img_buffer)
                    # Scrive le etichette (testo) nel file .zip
                    zf.writestr(label_arcname, label_content)

    print("\nConversione completata.")
    print(f"Il file '{args.zip_file}' e' stato creato con
    successo.")

# Questo blocco assicura che la funzione 'main()' sia eseguita
# solo quando lo script e' avviato direttamente (non se importato
    )
if __name__ == "__main__":
    main()
```

Codice A.20: Script preprocess_to_zip.py completo

A.8 Script di Post-Processing

```
# Importa la libreria OpenCV per l'elaborazione delle immagini
import cv2
# Importa la libreria NumPy per il calcolo numerico e la gestione
    degli array
import numpy as np
# Importa la libreria per interagire con il sistema operativo
import os
# Importa la libreria per cercare file sul disco che
   corrispondono a un pattern
import glob

# --- Impostazioni ---
# Definisce il percorso da cui leggere le immagini originali (con
    sfondo grigio)
CARTELLA_INPUT = 'Validation/images_original'
# Definisce il percorso in cui salvare le immagini pulite (con
   sfondo nero)
CARTELLA_OUTPUT = 'Validation/images'

# Percentuale del raggio. 1.0 = fino al bordo.
# 0.96 taglia via i bordi rumorosi/sfumati e lo sfondo.
FATTORE_RAGGIO = 0.96
# --------------------

# Crea la cartella di output se non esiste
# Controlla se la cartella di output non esiste gia'
if not os.path.exists(CARTELLA_OUTPUT):
```

```
24      # Crea la cartella di output (e tutte le cartelle intermedie
        necessarie)
25      os.makedirs(CARTELLA_OUTPUT)
26      # Stampa un messaggio di conferma della creazione
27      print(f"Cartella '{CARTELLA_OUTPUT}' creata.")
28
29  # Cerca tutti i file immagine (anche nelle sottocartelle)
30  # Definisce una tupla di estensioni di file immagine da cercare
31  tipi_file = ('*.jpg', '*.jpeg', '*.png', '*.bmp', '*.tif')
32  # Inizializza una lista vuota per contenere i percorsi dei file
        trovati
33  file_immagini = []
34  # Avvia un ciclo per ogni estensione definita in 'tipi_file'
35  for tipo in tipi_file:
36      # Cerca (ricorsivamente, '**') i file che corrispondono al
        pattern e li aggiunge alla lista
37      file_immagini.extend(glob.glob(os.path.join(CARTELLA_INPUT, '
        **', tipo), recursive=True))
38
39  # Controlla se sono stati trovati file
40  # Controlla se la lista 'file_immagini' e' vuota (nessun file
        trovato)
41  if not file_immagini:
42      # Stampa un messaggio di avviso se non sono state trovate
        immagini
43      print(f"Nessuna immagine trovata nella cartella '{
        CARTELLA_INPUT}'.")
44      # Stampa un suggerimento per l'utente
45      print("Assicurati di aver creato una cartella 'input' e di
        averci messo le tue immagini.")
46  # Blocco eseguito se almeno un'immagine e' stata trovata
47  else:
48      # Stampa il numero di immagini trovate e avvia l'elaborazione
49      print(f"Trovate {len(file_immagini)} immagini. Inizio
        elaborazione...")
50
51  # Elabora ogni immagine
52  # Avvia un ciclo che itera su ogni percorso di file trovato
53  for percorso_immagine in file_immagini:
54      # Carica l'immagine originale
55      # Legge il file immagine dal disco e lo carica in un array
        NumPy
56      img_originale = cv2.imread(percorso_immagine)
57
58      # Controlla se il caricamento dell'immagine e' fallito (es.
        file corrotto)
59      if img_originale is None:
60          # Stampa un messaggio di errore specificando il file
61          print(f"Errore: Impossibile caricare l'immagine {
        percorso_immagine}")
62          # Interrompe questa iterazione del ciclo e passa al file
        successivo
63          continue
64
```

```
    # Ottieni le dimensioni dell'immagine
    # Estrae l'altezza ('h') e la larghezza ('w') dalle dimensioni dell'array immagine
    h, w = img_originale.shape[:2]

    # Calcola il centro e il raggio
    # Calcola la coordinata X del centro dell'immagine (divisione intera)
    centro_x = w // 2
    # Calcola la coordinata Y del centro dell'immagine (divisione intera)
    centro_y = h // 2

    # Calcola il raggio basandoti sulla dimensione piu' piccola
    # Trova il raggio massimo possibile (basato sul lato piu' corto dal centro)
    raggio_base = min(centro_x, centro_y)
    # e applica il fattore per escludere i bordi (es. 0.96) e converte in intero
    raggio = int(raggio_base * FATTORE_RAGGIO)

    # 1. Crea una maschera completamente nera
    # (della stessa dimensione e tipo dell'originale)
    # Crea un array NumPy pieno di zeri (nero) con le stesse dimensioni di 'img_originale'
    maschera = np.zeros_like(img_originale)

    # 2. Disegna un cerchio pieno bianco sulla maschera
    # Questo cerchio rappresenta l'area che vogliamo conservare
    # Disegna un cerchio bianco pieno sulla maschera
    cv2.circle(maschera, (centro_x, centro_y), raggio, (255, 255, 255), thickness=cv2.FILLED)

    # 3. Applica la maschera all'immagine originale
    # cv2.bitwise_and mantiene solo i pixel dove entrambe
    # le immagini (originale e maschera) sono non-nere.
    # Esegue un'operazione AND bit-per-bit. I pixel fuori dal cerchio (dove la maschera e' 0) diventano 0.
    risultato = cv2.bitwise_and(img_originale, maschera)

    # Costruisci il percorso di output mantenendo la struttura
    # Calcola il percorso relativo del file (es. 'sottocartella/img.jpg')
    percorso_relativo = os.path.relpath(percorso_immagine, CARTELLA_INPUT)
    # Ricostruisce il percorso di destinazione nella cartella di output
    percorso_output = os.path.join(CARTELLA_OUTPUT, percorso_relativo)

    # Estrae il percorso della cartella di destinazione (es. 'Validation/images/sottocartella')
    cartella_destinazione = os.path.dirname(percorso_output)
    # Controlla se la cartella di destinazione (per le
```

```
    sottocartelle) non esiste
    if not os.path.exists(cartella_destinazione):
        # Crea la sottocartella di destinazione se necessario
        os.makedirs(cartella_destinazione)

    # Salva l'immagine modificata
    # Salva l'array 'risultato' (l'immagine mascherata) sul disco
    nel percorso di output
    cv2.imwrite(percorso_output, risultato)

# Stampa un messaggio finale al termine di tutti i cicli
print(f"\nElaborazione completata. Immagini salvate in '{
    CARTELLA_OUTPUT}'.")
```

Codice A.21: Script convert.py per il mascheramento

A.9 Script di Tracciamento e Analisi

```
import argparse  # Gestione degli argomenti da riga di comando
import os  # Interazione con il sistema operativo
from typing import List  # Per il type hinting
import subprocess  # Per eseguire comandi esterni (es. FFmpeg)
import numpy as np  # Calcolo numerico e matriciale
import torch  # Framework PyTorch per deep learning
import cv2  # OpenCV per elaborazione immagini
import h5py  # Lettura file scientifici HDF5
import glob  # Ricerca file tramite pattern
from tqdm import tqdm  # Barra di avanzamento
import norfair  # Libreria di Object Tracking
from norfair import Detection, Tracker  # Classi principali di
    Norfair
from models.experimental import attempt_load  # Caricamento
    modello YOLOv7
from utils.general import non_max_suppression  # Algoritmo NMS

# --- FUNZIONI DI UTILITA' ---
def yolo_detections_to_norfair_detections(yolo_detections: torch.
    tensor) -> List[Detection]:
    """Converte le rilevazioni di YOLOv7 in un formato
    comprensibile da Norfair."""
    norfair_detections: List[Detection] = []
    # yolo_detections ha formato [x_min, y_min, x_max, y_max,
    conf, class]
    for detection in yolo_detections:
        # Estrae le coordinate del bounding box
        bbox = np.array(
            [
                [detection[0].item(), detection[1].item()],
                [detection[2].item(), detection[3].item()],
            ]
        )
        # Estrae i punteggi di confidenza
```

```
30          scores = np.array([detection[4].item(), detection[4].item
    ()])
31          # Crea l'oggetto Detection richiesto da Norfair
32          norfair_detections.append(
33              Detection(points=bbox, scores=scores, label=int(
    detection[5].item()))
34          )
35      return norfair_detections
36
37 # --- MAIN ---
38 # 1. DEFINIZIONE DEGLI ARGOMENTI
39 parser = argparse.ArgumentParser(description="Traccia le regioni
    attive solari.")
40 parser.add_argument("--input-dir", type=str, required=True, help=
    "Percorso file .h5 o .jpg.")
41 parser.add_argument("--model-path", type=str, required=True, help
    ="Percorso modello .pt.")
42 parser.add_argument("--img-size", type=int, default=640, help="
    Dimensione inferenza.")
43 parser.add_argument("--conf-threshold", type=float, default=0.25,
     help="Soglia confidenza.")
44 parser.add_argument("--output-dir", type=str, default="
    output_tracking", help="Cartella output.")
45 parser.add_argument("--clip-min", type=int, default=-1500, help="
    Min clipping H5.")
46 parser.add_argument("--clip-max", type=int, default=1500, help="
    Max clipping H5.")
47 args = parser.parse_args()  # Parsing degli argomenti
48
49 # 2. IMPOSTAZIONI INIZIALI
50 os.makedirs(args.output_dir, exist_ok=True)  # Crea cartella
    output se non esiste
51 device = torch.device("cuda:0" if torch.cuda.is_available() else
    "cpu")  # Selezione hardware
52
53 # 3. CARICAMENTO DEL MODELLO
54 print("Caricamento del modello...")
55 # Carica i pesi del modello sul device selezionato
56 model = attempt_load(args.model_path, map_location=device)
57 model.eval()  # Imposta il modello in modalita' valutazione (
    inferenza)
58
59 # 4. INIZIALIZZAZIONE DI NORFAIR
60 # Configura il tracker basato sulla sovrapposizione spaziale (IOU
    )
61 tracker = Tracker(
62     distance_function="iou",
63     distance_threshold=0.7,  # Soglia di distanza per associare
    oggetti
64 )
65
66 # 5. RICERCA DEI FILE
67 # Cerca prima i file scientifici H5
68 input_files = sorted(glob.glob(os.path.join(args.input_dir, '*.h5
```

```
    '')))
if not input_files: # Se non trova H5, cerca immagini JPG/JPEG
    input_files = sorted(glob.glob(os.path.join(args.input_dir, '*.jpg'))) + \
                  sorted(glob.glob(os.path.join(args.input_dir, '*.jpeg')))

if not input_files:
    print(f"ERRORE: Nessun file trovato in {args.input_dir}")
    exit()

# 6. CICLO DI ELABORAZIONE
for file_path in tqdm(input_files, desc="Tracking"):

    # --- LOGICA DI CARICAMENTO IBRIDA ---
    if file_path.endswith('.h5'): # Caso: File Scientifico
        with h5py.File(file_path, 'r') as f:
            data = f['magnetogram/data'][:]  # Legge matrice dati
        # Crea maschera per i valori mancanti (NaN)
        background_mask = np.isnan(data)
        # Sostituisce NaN con 0.0 per evitare errori numerici
        data = np.nan_to_num(data, nan=0.0, posinf=0.0, neginf=0.0)
        # Applica clipping fisico ai valori del campo magnetico
        clipped_data = np.clip(data, args.clip_min, args.clip_max)
        # Normalizza i dati nell'intervallo [0, 1]
        normalized_data = (clipped_data - args.clip_min) / (args.clip_max - args.clip_min)
        # Ridimensiona l'immagine per l'inferenza
        resized_image = cv2.resize(normalized_data, (args.img_size, args.img_size), interpolation=cv2.INTER_LINEAR)

        # Preparazione visualizzazione (Converte scala di grigi in BGR per OpenCV)
        display_image_gray = (resized_image * 255).astype(np.uint8)
        # Ridimensiona la maschera dello sfondo
        background_mask_resized = cv2.resize(background_mask.astype(np.uint8), (args.img_size, args.img_size), interpolation=cv2.INTER_NEAREST)
        # Imposta a nero i pixel dello sfondo usando la maschera
        display_image_gray[background_mask_resized == 1] = 0
        display_image = cv2.cvtColor(display_image_gray, cv2.COLOR_GRAY2BGR)
        # Crea il tensore input duplicando i canali (1 -> 3)
        image_tensor_np = np.stack([resized_image] * 3, axis=-1)

    else: # Caso: Immagine Standard
        img_raw = cv2.imread(file_path)  # Legge immagine
        if img_raw is None: continue  # Salta se errore lettura
        # Ridimensiona immagine
        display_image = cv2.resize(img_raw, (args.img_size, args.img_size), interpolation=cv2.INTER_LINEAR)
```

```
        # Normalizza pixel tra 0 e 1
        image_tensor_np = display_image.astype(np.float32) / 255.0

    # Conversione in Tensore PyTorch e spostamento su GPU/CPU
    # Transpone dimensioni da [H, W, C] a [C, H, W]
    image_tensor = torch.from_numpy(image_tensor_np.transpose(2, 0, 1)).float().to(device)
    image_tensor = image_tensor.unsqueeze(0)  # Aggiunge dimensione batch

    # --- Inferenza ---
    with torch.no_grad():  # Disabilita calcolo gradienti
        results = model(image_tensor, augment=False)[0]
    # Applica Non-Maximum Suppression per filtrare box sovrapposti
    results = non_max_suppression(results, conf_thres=args.conf_threshold)[0]

    # --- Update Tracker ---
    if results is not None and len(results) > 0:
        # Converte rilevazioni YOLO in oggetti Norfair
        detections = yolo_detections_to_norfair_detections(results)
        # Aggiorna il tracker con le nuove posizioni
        tracked_objects = tracker.update(detections=detections)
    else:
        # Aggiorna il tracker anche se non ci sono rilevazioni (per gestire oggetti persi)
        tracked_objects = tracker.update(detections=[])

    # Disegna e Salva
    # Disegna i box e gli ID tracciati sull'immagine
    norfair.draw_tracked_boxes(display_image, tracked_objects)
    # Definisce nome file output (cambia estensione in .jpg se necessario)
    output_filename = os.path.basename(file_path).replace('.h5', '.jpg')
    # Salva il frame processato su disco
    cv2.imwrite(os.path.join(args.output_dir, output_filename), display_image)

# Generazione Video con FFMPEG
print("\nGenerazione video...")
# Esegue comando esterno ffmpeg per unire i frame in un video MP4
subprocess.run([
    'ffmpeg', '-y', '-r', '10', '-pattern_type', 'glob',
    '-i', f'{args.output_dir}/*.jpg',
    '-c:v', 'libx264', '-pix_fmt', 'yuv420p',
    os.path.join(args.output_dir, "tracking_video.mp4")
])
```

Codice A.22: Script di tracciamento (track.py)

```
import argparse  # Gestione argomenti riga di comando
```

```
import os  # Gestione file system
import glob  # Ricerca file
import shutil  # Operazioni su file e cartelle (es. rimozione)
import numpy as np  # Calcolo numerico
import torch  # Deep Learning
import cv2  # OpenCV
import h5py  # Lettura file H5
from tqdm import tqdm  # Barra di progresso
import norfair  # Libreria Tracking
from norfair import Detection, Tracker  # Classi base
# Importazioni specifiche per il calcolo delle metriche MOT
from norfair.metrics import Accumulators, InformationFile
from models.experimental import attempt_load  # Loading YOLO
from utils.general import non_max_suppression  # NMS

# --- 1. UTILITY: Prepara il GT personalizzato per Norfair ---
def setup_custom_mot_structure(custom_gt_path, img_list,
    output_base, img_w, img_h):
    """
    Legge il file ground_truth proprietario e crea la struttura
    di cartelle
    standard 'MOTChallenge' richiesta dalle metriche di Norfair.
    """
    # Crea cartella 'gt' all'interno della directory temporanea
    gt_dir = os.path.join(output_base, "gt")
    os.makedirs(gt_dir, exist_ok=True)

    dest_gt_path = os.path.join(gt_dir, "gt.txt")

    print(f"Preparazione Ground Truth da: {custom_gt_path}")

    if not os.path.exists(custom_gt_path):
        print(f"ERRORE CRITICO: Il file GT {custom_gt_path} non
    esiste!")
        exit()

    # Legge il file originale e lo converte riga per riga
    with open(custom_gt_path, "r") as in_f, open(dest_gt_path, "w
    ") as out_f:
        for line in in_f:
            parts = line.strip().split(',')
            if len(parts) < 6: continue

            try:
                # Parsing dei dati: frame, id, x, y, w, h
                frame = int(parts[0])
                obj_id = int(parts[1])
                x = float(parts[2])
                y = float(parts[3])
                w = float(parts[4])
                h = float(parts[5])

                # Scrive nel formato standard MOT:
                # frame, id, left, top, width, height, conf(1),
```

```
    -1, -1, -1
52                  mot_line = f"{frame},{obj_id},{x:.2f},{y:.2f},{w
    :.2f},{h:.2f},1,-1,-1,-1\n"
53                  out_f.write(mot_line)
54              except ValueError:
55                  continue
56
57      # Creazione del file seqinfo.ini (fondamentale per le
    metriche)
58      seq_len = len(img_list)
59      seqinfo_path = os.path.join(output_base, "seqinfo.ini")
60      with open(seqinfo_path, "w") as f:
61          f.write("[Sequence]\n")
62          f.write(f"name={os.path.basename(output_base)}\n")
63          f.write(f"imDir=img1\n")
64          f.write(f"frameRate=30\n")
65          f.write(f"seqLength={seq_len}\n")
66          f.write(f"imWidth={img_w}\n")
67          f.write(f"imHeight={img_h}\n")
68          f.write(f"imExt=.jpg\n")
69
70  # --- 2. CONVERSIONE YOLO -> NORFAIR ---
71  def yolo_to_norfair(yolo_preds):
72      """Converte i tensori di output di YOLO in oggetti Detection
    di Norfair."""
73      norfair_detections = []
74      for det in yolo_preds:
75          # Estrae bbox [x_min, y_min, x_max, y_max]
76          bbox = np.array([
77              [det[0].item(), det[1].item()],
78              [det[2].item(), det[3].item()]
79          ])
80          scores = np.array([det[4].item(), det[4].item()])
81          label = int(det[5].item())
82          norfair_detections.append(Detection(points=bbox, scores=
    scores, label=label))
83      return norfair_detections
84
85  # --- MAIN ---
86  if __name__ == "__main__":
87      # Configurazione argomenti
88      parser = argparse.ArgumentParser()
89      parser.add_argument("--source", type=str, required=True, help
    ="Cartella immagini")
90      parser.add_argument("--gt-file", type=str, required=True,
    help="File GT proprietario")
91      parser.add_argument("--model-path", type=str, required=True,
    help="Modello .pt")
92      parser.add_argument("--img-size", type=int, default=640)
93      parser.add_argument("--conf-thres", type=float, default=0.25)
94      args = parser.parse_args()
95
96      device = torch.device("cuda" if torch.cuda.is_available()
    else "cpu")
```

```
97
98      # 1. Carica Modello (Gestione JIT/Torchscript)
99      print(f"Caricamento modello: {args.model_path}")
100     try:
101         model = attempt_load(args.model_path, map_location=device
    )
102     except (AttributeError, RuntimeError):
103         # Fallback per modelli esportati in TorchScript
104         print("Rilevato modello JIT/Traced. Uso torch.jit.load...
    ")
105         model = torch.jit.load(args.model_path, map_location=
    device)
106     model.eval()
107
108     # 2. Lista File
109     input_files = sorted(glob.glob(os.path.join(args.source, '*.*
    ')))
110     # Filtra solo le estensioni supportate
111     input_files = [f for f in input_files if f.endswith(('.jpg',
    '.jpeg', '.png', '.h5'))]
112     if not input_files:
113         print("ERRORE: Nessun file trovato.")
114         exit()
115
116     # Prepara Cartella temporanea per le metriche
117     TEMP_MOT_DIR = "temp_metrics_env"
118     if os.path.exists(TEMP_MOT_DIR): shutil.rmtree(TEMP_MOT_DIR)
119     os.makedirs(TEMP_MOT_DIR, exist_ok=True)
120
121     # 3. Setup Ambiente Metriche
122     # Converte il GT e crea seqinfo.ini
123     setup_custom_mot_structure(args.gt_file, input_files,
    TEMP_MOT_DIR, args.img_size, args.img_size)
124
125     # 4. Inizializza Tracker
126     tracker = Tracker(
127         distance_function="iou",
128         distance_threshold=0.7,
129         detection_threshold=args.conf_thres
130     )
131
132     # Inizializza oggetti per calcolo metriche
133     print("Inizializzazione Metriche...")
134     seqinfo_path = os.path.join(TEMP_MOT_DIR, "seqinfo.ini")
135
136     try:
137         # Crea oggetto InformationFile leggendo il file .ini
    generato
138         info_file = InformationFile(file_path=seqinfo_path)
139
140         # Crea accumulatore di statistiche
141         acc = Accumulators()
142         acc.create_accumulator(input_path=TEMP_MOT_DIR,
    information_file=info_file)
```

```
143
144     except Exception as e:
145         print(f"Errore inizializzazione Norfair: {e}")
146         exit()
147
148     print(f"\nAvvio Tracking su {len(input_files)} frames...")
149
150     # LOOP PRINCIPALE
151     for i, path in enumerate(tqdm(input_files)):
152         # Caricamento e Pre-processing Ibrido
153         if path.endswith('.h5'):
154             with h5py.File(path, 'r') as f: data = np.nan_to_num(
    f['magnetogram/data'][:])
155             # Normalizzazione scientifica [-1500, 1500] -> [0, 1]
156             norm = (np.clip(data, -1500, 1500) + 1500) / 3000
157             img0 = cv2.resize(norm, (args.img_size, args.img_size
    ))
158             img_tensor = torch.from_numpy(np.stack([img0]*3, axis
    =-1)).float()
159         else:
160             img0 = cv2.imread(path)
161             if img0 is None: continue
162             img0 = cv2.resize(img0, (args.img_size, args.img_size
    ))
163             img_tensor = torch.from_numpy(img0 / 255.0).float()
164
165         # Preparazione tensore GPU
166         img_tensor = img_tensor.permute(2, 0, 1).unsqueeze(0).to(
    device)
167
168         # Inferenza
169         with torch.no_grad():
170             try:
171                 pred = model(img_tensor, augment=False)[0]
172             except:
173                 pred = model(img_tensor)[0] # Fallback JIT
174
175         # Filtraggio NMS
176         pred = non_max_suppression(pred, args.conf_thres)[0]
177
178         # Aggiornamento Tracker
179         if pred is not None and len(pred) > 0:
180             dets = yolo_to_norfair(pred)
181             tracked_objs = tracker.update(detections=dets)
182         else:
183             tracked_objs = tracker.update(detections=[])
184
185         # Aggiornamento Metriche
186         # Confronta le predizioni del tracker con il GT per
    questo frame
187         acc.update(predictions=tracked_objs)
188
189     # 5. Calcolo Risultati Finali
190     print("\n" + "="*40)
```

```
    print(" RISULTATI FINALI ")
    print("="*40)

    try:
        # Calcola le metriche (MOTA, IDF1, ecc.)
        metrics = acc.compute_metrics()
        print(metrics)

        # Salva report su file
        os.makedirs("risultati_finali", exist_ok=True)
        acc.save_metrics(save_path="risultati_finali", file_name=
    "report_metriche_complete.txt")
        print("\nReport salvato in: risultati_finali/
    report_metriche.txt")

    except Exception as e:
        print(f"\nERRORE calcolo metriche: {e}")
```

Codice A.23: Script per il calcolo delle metriche MOT (track_metrics.py)

Bibliografia

[1] E. Camporeale, S. Wing, and J. Johnson, eds., *Machine learning techniques for space weather*. Elsevier, 2018.

[2] L. Bolduc, "Gic observations and studies in the hydro-québec power system," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 64, no. 16, pp. 1793–1802, 2002.

[3] C. T. Gaunt and G. Coetzee, "Transformer failures in regions incorrectly considered to have low gic-risk," in *2007 IEEE Lausanne Power Tech*, pp. 807–812, IEEE, 2007.

[4] M. Hapgood, "Ionospheric correction of space radar data," *Acta Geophysica*, vol. 58, no. 3, pp. 453–467, 2010.

[5] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022. `https://arxiv.org/abs/2207.02696`.

[6] W. D. Pesnell, B. J. Thompson, and P. C. Chamberlin, "The solar dynamics observatory (sdo)," *Solar Physics*, vol. 275, no. 1-2, pp. 3–15, 2012.

[7] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2010.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[9] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[10] Z. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, 2016.

[14] NumPy Developers, *NumPy Documentation*, 2023.

[15] SciPy Developers, *SciPy Documentation*, 2023.

[16] OpenCV.org, *OpenCV Documentation*, 2023.

[17] Matplotlib Development Team, *Matplotlib Documentation*, 2023.

[18] Anaconda, Inc., *Miniconda — Conda package manager*, 2023.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019.

[20] NVIDIA Corporation, *CUDA Toolkit Documentation*, 2023.

[21] "Norfair: Simple, real-time, python 3d multi-object tracker." `https://github.com/tryolabs/norfair`.

[22] W. D. Pesnell, B. J. Thompson, and P. C. Chamberlin, "The solar dynamics observatory (sdo)," *Solar Physics*, vol. 275, no. 1-2, pp. 3–15, 2012.

[23] P. H. Scherrer, J. Schou, R. I. Bush, *et al.*, "The helioseismic and magnetic imager (hmi) investigation for the solar dynamics observatory (sdo)," *Solar Physics*, vol. 275, no. 1-2, pp. 207–227, 2012.

[24] E. R. Priest, *Magnetohydrodynamics of the Sun*. Cambridge University Press, 2014.