



UNIVERSITÀ
degli STUDI
di CATANIA

Road Sign Detection CT

Progetto Machine Learning

Nome: Bruno Cognome: Montalto Matricola: 1000070013
Nome: Fabrizio Cognome: Iuvara Matricola: 1000015982

Anno Accademico 2023/2024

1. Problema

Il riconoscimento di cartelli stradali è uno dei problemi affrontati nei campi dell'elaborazione delle immagini e nel riconoscimento di pattern, il quale consiste nel rilevare e classificare correttamente segnali stradali a partire da immagini o video.

Data la natura del problema, questi applicativi sono di particolare interesse nei sistemi avanzati di assistenza alla guida (**ADAS**); questi riconoscono la segnaletica stradale attraverso l'uso di appositi dispositivi quali telecamere e sensori, con obiettivo quello di migliorare la sicurezza stradale, soprattutto nei tempi moderni, in cui le auto a guida autonoma sono sempre più diffuse. Questa tecnologia infatti utilizza anche modelli di object detection per prendere decisioni, anche in lassi di tempo brevissimi, come deviare, fermarsi, ridurre la velocità, e così via. Una decisione inaccurata può portare gravi conseguenze, potenzialmente letali per le persone coinvolte in eventuali incidenti.

In contesti come questi è fondamentale che gli algoritmi possano essere eseguiti in tempo reale e che siano estremamente accurati.

I principali problemi del riconoscimento di cartelli stradali sono legati a diversi fattori, primi tra questi le condizioni atmosferiche in cui si possono dover rilevare i cartelli; si pensi per esempio a situazioni di pioggia intensa o nebbia, che possono rendere molto difficile il task. Un altro problema da non sottovalutare è la condizione stessa dei cartelli stradali, infatti questi sono spesso soggetti a usura, sporcizia, deformazioni, danneggiamento; una parte importante dei cartelli campionati per questo progetto ricadono in queste categorie.

Altri problemi che rendono più difficile il compito sono condizioni di illuminazione eccessiva o carente che rendono il cartello più difficile da interpretare; inoltre, i cartelli possono essere inclinati e potenzialmente anche nascosti dietro ostacoli o elementi indesiderati, si pensi per esempio ad un cartello parzialmente nascosto dietro un palo o una macchina che lo oscura.

La maggior parte di questi problemi sono stati riscontrati anche nel nostro dataset, riportiamo di seguito alcune immagini che mostrano alcune delle condizioni già presentate.



Sebbene il problema possa essere affrontato con approcci classici basati sull'analisi degli istogrammi, i moderni algoritmi di machine learning si sono dimostrati più efficaci, in quanto possono adattarsi a diverse condizioni di visibilità apprendendo dai dati.

In questo progetto, vogliamo realizzare un sistema di riconoscimento di cartelli stradali specializzato in particolare per la città di Catania.

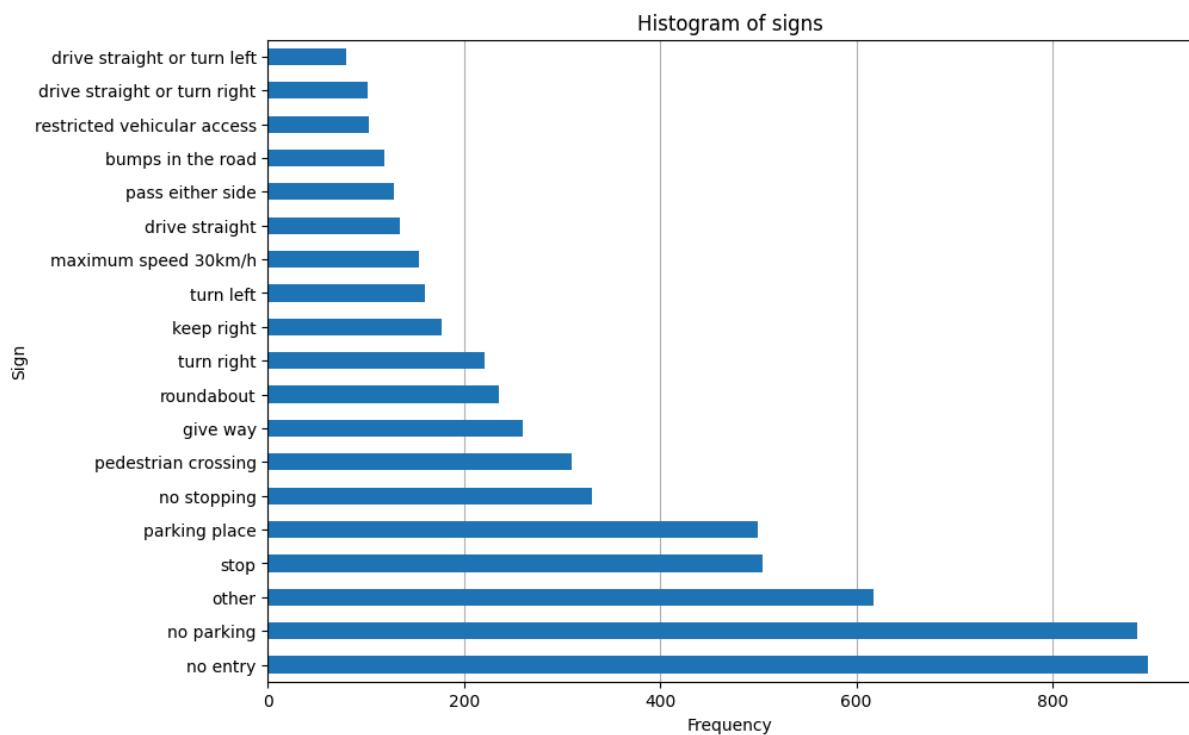
2. Dataset

La prima fase del nostro progetto è l'acquisizione del dataset che abbiamo utilizzato per allenare i nostri modelli. Questa fase è supportata tramite uno strumento da noi sviluppato descritto più dettagliatamente nella **Sezione 2.1.1**. Questa raccolta iniziale ci permette di produrre 3.175 immagini, acquisite dall'interfaccia di Google Maps, nel contesto della città di Catania e dei comuni limitrofi.

Successivamente, abbiamo pulito il nostro dataset, eliminando 29 immagini non adatte al nostro task.

Infine, abbiamo etichettato e ricontrattato l'intero dataset (**Sezione 2.1.2**). In totale, l'intero processo di acquisizione dei dati restituisce 3.146 immagini, con 5.919 cartelli identificati e classificati in 67 classi (considerando anche classi "simmetriche" tra di loro, quali la svolta obbligatoria a destra e quella a sinistra).

Di seguito presentiamo un istogramma contenente i cartelli più frequenti del nostro dataset.



Il grafico presenta le classi con almeno 80 cartelli, le altre sono sotto l'etichetta "other"

dopo la fase di raccolta dati sono state effettuate diverse analisi statistiche del dataset al fine di prepararlo per le fasi successive del progetto;

2.1 Acquisizione

Le immagini del dataset sono state ottenute da Google Maps. In particolare gli operatori percorrono le strade con un'auto dotata di un sistema di telecamere e mediante l'utilizzo di diversi algoritmi di elaborazione di immagini rendono disponibile una panoramica a 360° dei punti del loro percorso.

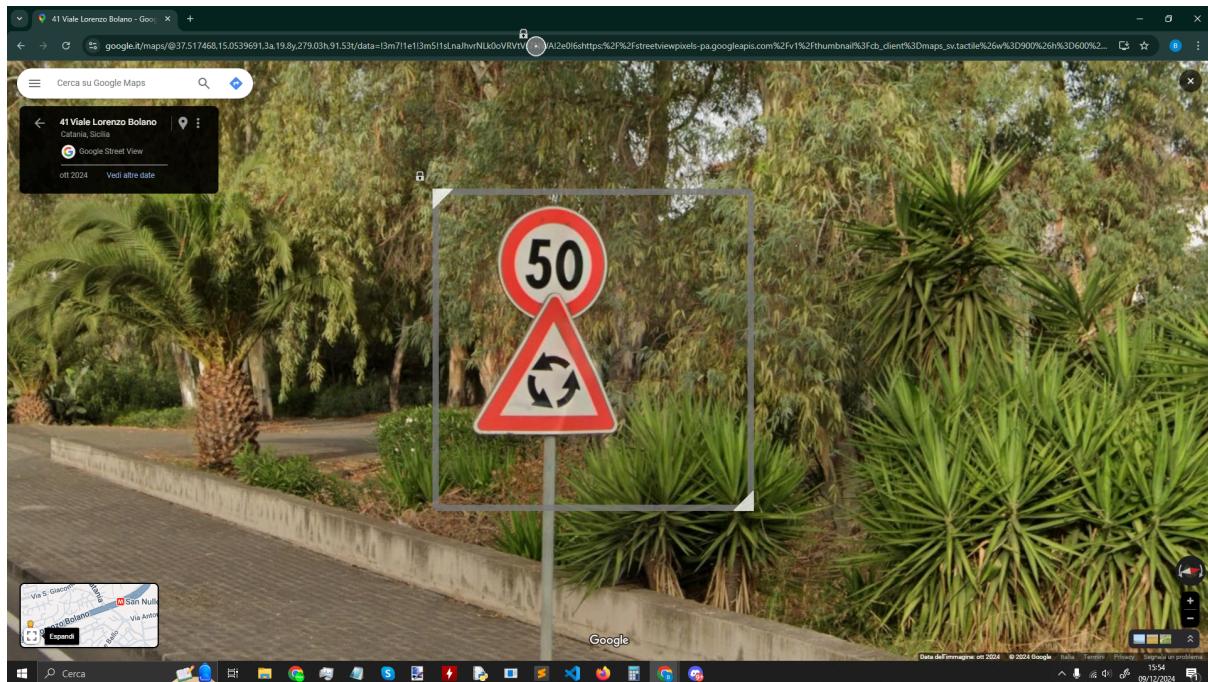
Per agevolare l'acquisizione dei dati dall'interfaccia di Google Maps, abbiamo creato un semplice tool in Python (**Sezione 2.1.1**), che ci permette di navigare sul sito per scattare le immagini ed estrarre le informazioni relative alle coordinate, contenute nei rispettivi URL.

Questi dati sono utili per tracciare le aree esplorate e per effettuare statistiche generali sul dataset che verranno presentate a breve.



2.1.1 Tool di acquisizione (Maps Image Collector)

Il tool sviluppato si presenta graficamente con un riquadro che permette di selezionare la regione di schermo da catturare, e con un cursore che va posizionato in corrispondenza della barra URL del browser.



Con il tasto L è possibile bloccare/sbloccare i widget, mentre con il tasto spazio si cattura l'immagine e si esegue una macro in corrispondenza del cursore; questa copia l'URL ed estrae informazioni aggiuntive che saranno memorizzate in data/coordinates.xml.

Il riquadro di cattura può essere ridimensionato dai widget o, alternativamente, si può fissare una dimensione modificando la terza riga del file data/data.txt.

NOTA: Il tool può essere utilizzato solo su Windows.

2.2 Etichettatura dei dati

Per etichettare le nostre immagini abbiamo utilizzato il tool **labelImg**, reperibile al link (<https://github.com/HumanSignal/labelImg>). Lo strumento presenta un'interfaccia in cui è possibile tracciare il bounding box (mediante il tasto **W**) e assegnare la corrispondente etichetta; mediante i tasti **A** e **D** è possibile scorrere nella cartella delle immagini. Le etichette vengono memorizzate in un file xml in formato PASCAL VOC (nel nostro caso i file sono raccolti nella cartella dataset/labels - PASCAL VOC).

Durante questa fase abbiamo fatto particolare attenzione ad etichettare anche cartelli molto lontani e/o occlusi da altri oggetti.



Le immagini di sopra hanno rispettivamente id 1392 e 1426.

2.3 Analisi del dataset

Dopo la fase di etichettatura e una generale pulizia del dataset, ci siamo occupati di effettuare delle analisi per agevolarne l'utilizzo nelle fasi successive del progetto.

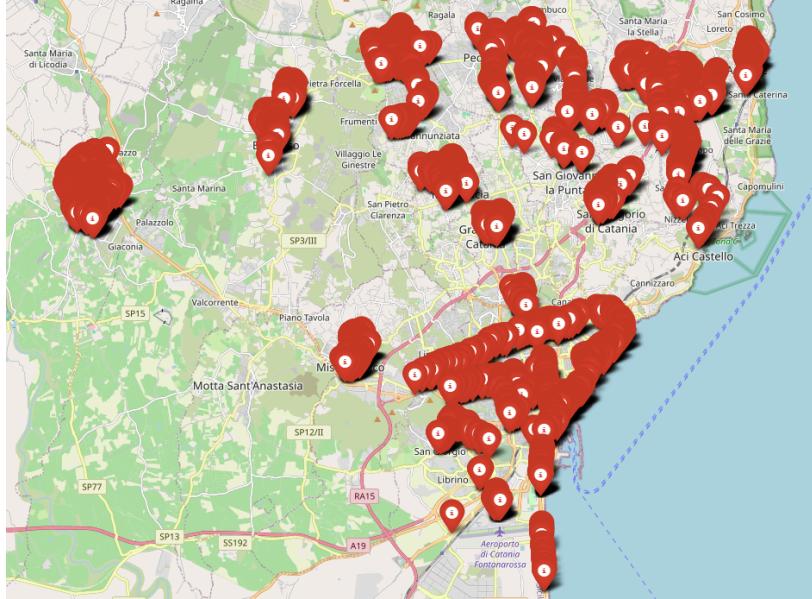
Il dataset ripulito si compone di **3146 immagini** (inizialmente erano 3175), contenenti **5919 cartelli** appartenenti a 67 classi diverse; come si evince dall'istogramma prima presentato il dataset è fortemente sbilanciato per diverse classi; la classe più numerosa è quella dei divieti di accesso con 898 entry; se consideriamo con questi anche i divieti di sosta e i segnali di stop arriviamo al 38.6% del dataset.

Durante la fase di acquisizione delle immagini abbiamo notato che possono essere presenti cartelli simili nelle stesse zone, di conseguenza abbiamo cercato di diversificare il dataset scattando foto in posti diversi.



Le immagini di sopra hanno rispettivamente id 1217 e 2842.

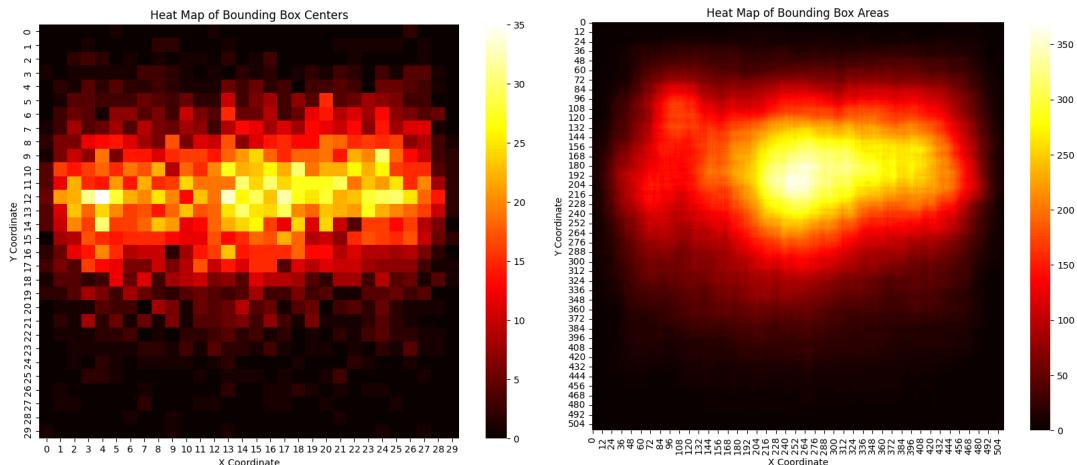
Con i dati raccolti siamo in grado di generare la seguente mappa contenente i luoghi in cui sono scattate le foto; questa mappa si è rivelata molto utile in quanto ci ha permesso di effettuare clustering (**Sezione 2.3.3**):



La mappa è stata creata dal file “create interactive map.py”

2.3.1 Mappe di densità

Abbiamo realizzato i seguenti grafici per analizzare la distribuzione dei boundingbox nel dataset. In particolare, il primo grafico mostra una mappa di densità dei punti centrali di ogni bounding box, calcolata con 30 bin per entrambe le dimensioni. Il secondo grafico, invece, è una mappa di densità pixel per pixel delle aree coperte dai bounding box.



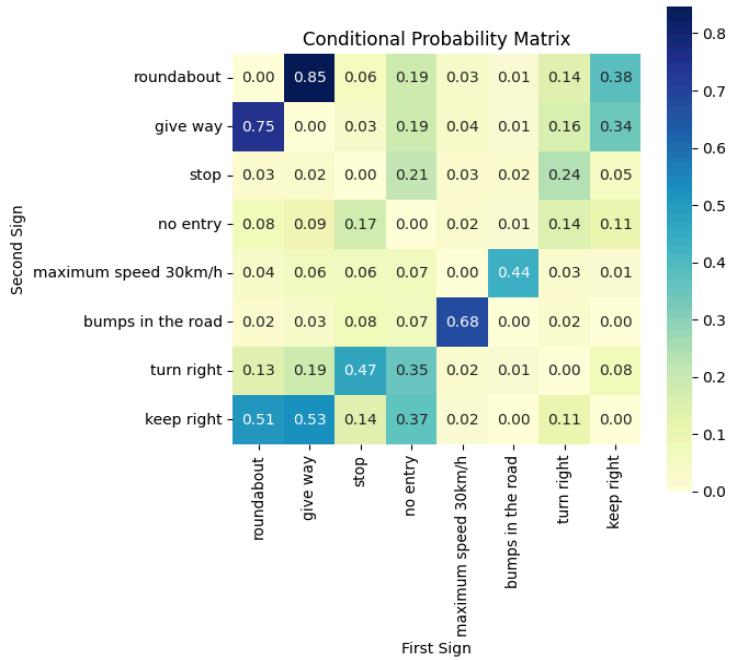
Queste immagini sono state calcolate nel notebook “statistics.ipynb”

Si può notare la presenza di un bias: la maggior parte dei cartelli che abbiamo campionato si trovano principalmente tra la zona superiore e centrale dell'immagine, con uno sbilanciamento verso destra. Questo è spiegabile dal fatto che, essendo in un paese con guida a destra, la maggior parte dei cartelli si trova nel lato destro della strada e cartelli presenti nella parte inferiore dell'immagine sono molto rari, tra questi infatti ci sono solo eccezioni come il cartello “keep right” che è presente spesso sugli spartitraffico e nelle zone in cui è presente un cantiere, quindi è piazzato sulla carreggiata.

2.3.2 Coppie frequenti

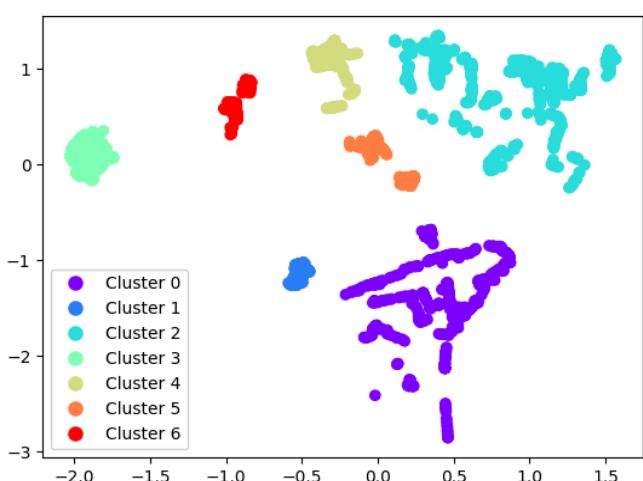
Un altro aspetto che bisogna tenere in considerazione è che diversi cartelli si trovano spesso insieme nella stessa immagine, per mostrare ciò riportiamo una matrice che rappresenta la probabilità condizionale di trovare un cartello in un'immagine ("Second sign") dato che nella stessa immagine è già presente un dato cartello ("First sign").

Dal grafico si vede per esempio come la probabilità di vedere un segnale di Dare precedenza dato che è presente un segnale di rotatoria è dell'85%; un'altra correlazione da notare è tra i cartelli di presenza di dossi sulla strada e dei limiti di velocità, primo tra tutti il limite di 30km/h, cartello più presente tra questi limiti.



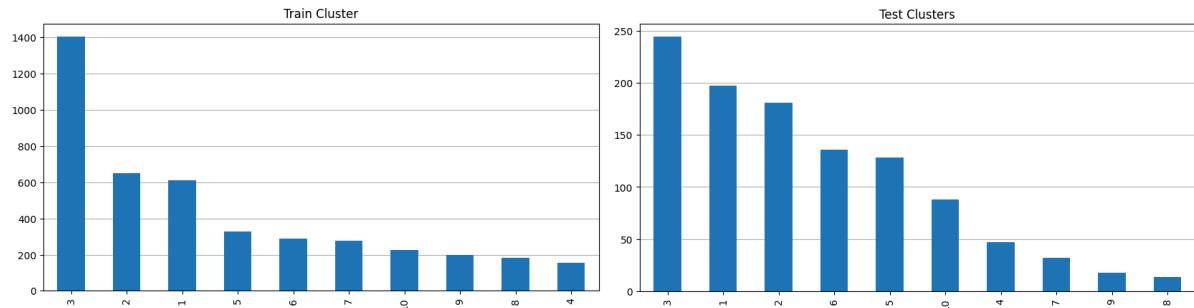
2.3.3 Clustering

Le informazioni relative alle coordinate di acquisizione ci hanno inoltre permesso di realizzare uno split train/test tale che le immagini di training siano state acquisite in aree separate da quelle di test, in modo da rendere gli esperimenti più significativi. In particolare, abbiamo applicato l'algoritmo di clustering **DBSCAN** ai punti della mappa (con parametro $\text{epsilon} = 0.2$), ottenendo 7 clusters. È stato considerato anche l'algoritmo K-means ma non ha dato buoni risultati in quanto i cluster ottenuti non rappresentano bene i nostri dati; infatti si può vedere che abbiamo cluster molto numerosi con forme diverse e il K-means non è adatto a rappresentare questa struttura dei dati in quanto presuppone dei cluster di forma sferica; inoltre, avendo dati sbilanciati, i cluster più piccoli vengono "assorbiti" dai più grandi e questo ci ha portato a utilizzare l'approccio DBSCAN, più appropriato nella nostra situazione.



Sono riportati a sinistra i cluster ottenuti con DBSCAN, numerati da 0 a 6, i quali contengono rispettivamente 747, 158, 1030, 498, 286, 279 e 148 punti.

Dopo svariate prove, abbiamo scelto di utilizzare i cluster 0, 2, 3 e 4, come training set e i rimanenti come test set. Lo split ottenuto è circa 81-19. Questo split genera inoltre due distribuzioni delle classi non troppo divergenti nei due set, sono riportati di seguito gli istogrammi delle frequenze:



Notiamo che le sequenze delle etichette, ordinate per frequenza, sono abbastanza simili, a meno della classe other (identificata da “3” nei grafici).

2.4 Considerazioni

Le immagini di google maps vengono scattate in condizioni ottimali di traffico e meteo. Per questo motivo, non è detto che approcci basati sul dataset che abbiamo raccolto possano generalizzare e avere buone performance anche in condizioni diverse.

Inoltre, le immagini possono presentare elementi di interfaccia grafica, come nomi di strade o piccoli marker, soprattutto in zone urbane.

2.5 Organizzazione della cartella

Dato che le dimensioni del dataset superano il limite consentito da GitHub, lo abbiamo reso reperibile al seguente link: [dataset](#).

La directory dataset è organizzata come segue:

- **images**: contiene le immagini in formato .png.
- **labels - PASCAL VOC**: contiene i file .xml delle etichette, in formato Pascal VOC.
- **labels**: contiene le etichette in formato YOLO.
- **coordinates.xml**: file che associa a ogni immagine le coordinate di acquisizione.

Allo stesso livello della cartella dataset, abbiamo incluso:

- **splits**: cartella contenente i file .txt che definiscono i due split random e cluster-based.
- **dataset (random).yaml e dataset (cluster).yaml**: file di configurazione per il fine tuning di YOLO.
- **statistics.ipynb**: notebook jupyter contenente il codice utilizzato per calcolare alcune delle statistiche e dei grafici presentati prima.
- **map.html**: una mappa delle coordinate.
- **map (interactive).html**: mappa che permette anche di visualizzare l'id di ogni mark (per via dell'elevato numero di marks la navigazione potrebbe risultare molto lenta).
- **utility scripts**: cartella contenente script Python utilizzati durante la fase di pulizia del dataset, per estrarre semplici statistiche e per effettuare controlli di integrità. Contiene anche gli script per generare le mappe.
- **Maps Image Collector**: cartella del tool di acquisizione.
- altri file e cartelle riguardanti l'allenamento dei modelli (**Sezione 7. Codice**).

3. Metodi

3.1 Faster R-CNN

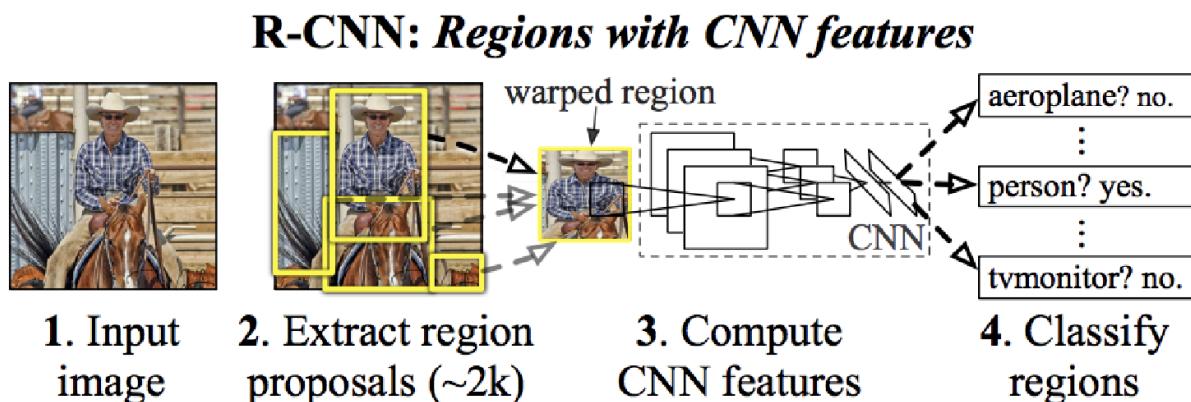
Prima di introdurre l'architettura di Faster R-CNN, descriviamo le due architetture sottostanti.

3.1.1 R-CNN

Il primo modello Region Based Convolutional Neural Network (R-CNN) risale al 2014 e fu sviluppato da J.R.R. Uijlings et al.

L'architettura divide il processo di detection in tre fasi:

- **Generazione di region proposals:** utilizzando un algoritmo chiamato Selective Search, R-CNN genera fino a 2000 proposte di regioni all'interno delle immagini che potrebbero contenere i nostri oggetti.
- **Feature extraction:** ogni regione viene ritagliata e ridimensionata per adattarsi a una dimensione fissa. Successivamente, queste vengono elaborate da una rete convoluzionale pre addestrata per estrarre le caratteristiche.
- **Classificazione:** le caratteristiche così ottenute vengono passate a un classificatore SVM per determinare la categoria dell'oggetto.

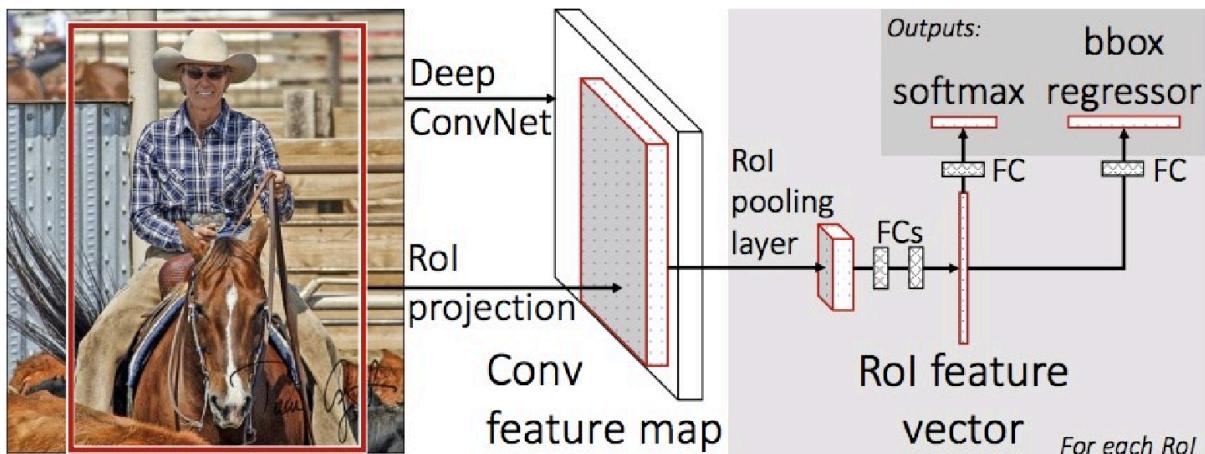


Architettura modello R-CNN (link <https://arxiv.labs.arxiv.org/html/1311.2524>)

3.1.2 Fast R-CNN

È un approccio che migliora significativamente R-CNN in termini di velocità e semplicità di addestramento.

Si distingue principalmente grazie all'aggiunta di un nuovo layer chiamato **ROI** (Region of Interest) **pooling** che ridimensiona ogni regione proposta a una dimensione fissa; inoltre un'unica rete convoluzionale svolge sia la classificazione che la regressione dei bounding box, rendendo l'addestramento più semplice.

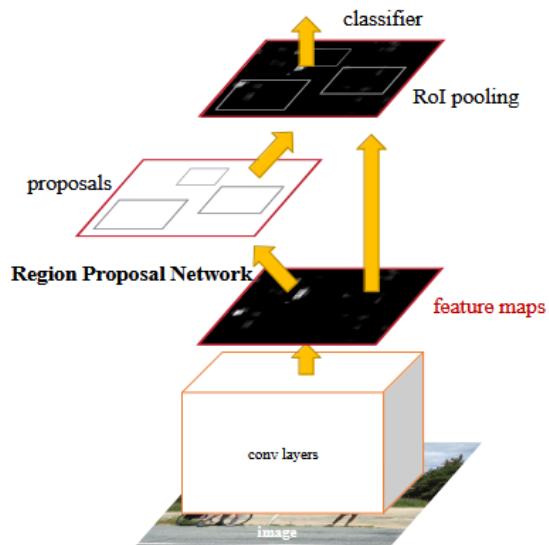


Architettura modello fast R-CNN (link <https://arxiv.org/abs/1504.08083>)

3.1.3 Faster R-CNN

Il modello Faster R-CNN rappresenta l'evoluzione dei due metodi citati sopra.

In particolare, Faster R-CNN è un metodo **two stage** che fa uso di un **Region Proposal Network** (RPN), una rete convoluzionale che si occupa di generare Region Proposal partendo dalla mappa delle feature; successivamente, per ogni posizione nella mappa, vengono generati una serie di **anchor boxes** di dimensioni diverse che vengono poi raffinati per prevedere regioni precise. Il modello Faster R-CNN finale risulta essere una combinazione tra il modello RPN e il Fast R-CNN e presenta un miglioramento in velocità rispetto ai due modelli precedenti; questo è dovuto al fatto che il RPN sostituisce il Selective Search e permette di risparmiare molto tempo di computazione.

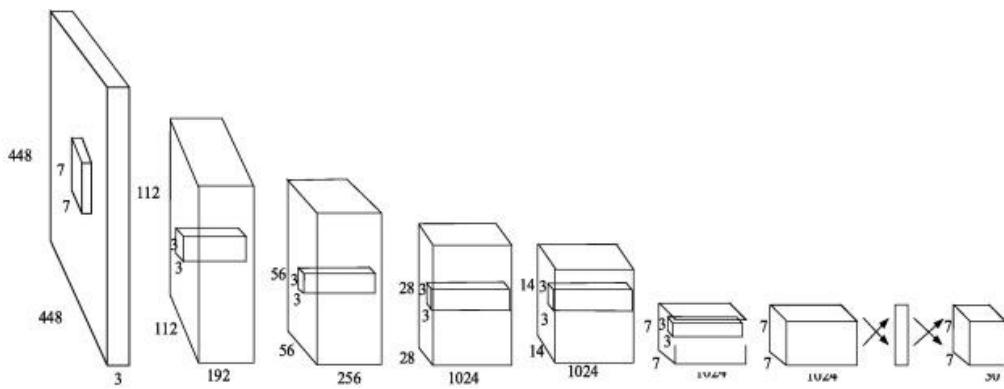


Architettura modello Faster R-CNN :(link <https://arxiv.org/abs/1506.01497>)

3.2 You Only Look Once (YOLO)

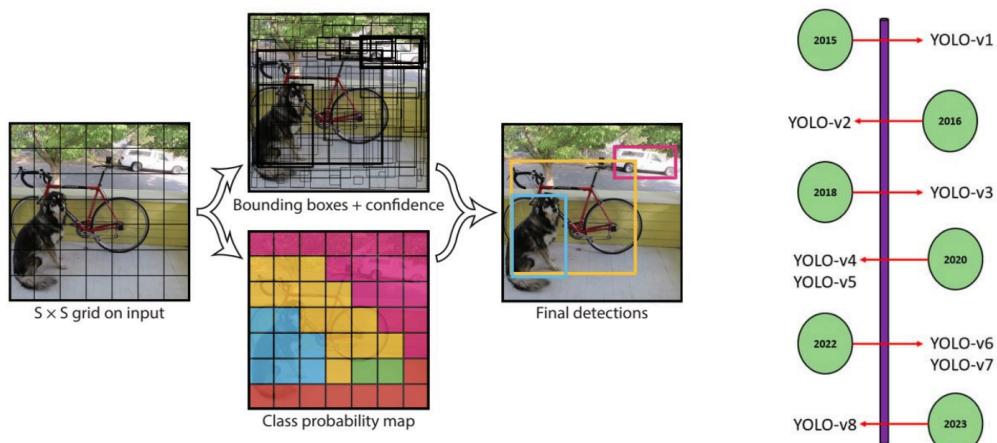
YOLO è un modello **one stage**, ovvero predice direttamente i bounding boxes e le class probabilities con una singola rete ed in un singolo passo. Questa sua caratteristica, assieme alla grande quantità di ottimizzazioni sviluppate dagli autori attraverso le varie versioni, permette di ottenere predizioni real-time su hardware consumer.

Inizialmente il modello prende in input un'immagine e la divide in una griglia SxS. Ogni cella di questa griglia predice **B bounding boxes** con un punteggio di confidenza (la probabilità di rilevare l'oggetto moltiplicato per la IoU tra la bounding box predetta e di ground truth). La rete utilizzata consta di 24 layer convoluzionali seguiti da 2 fully connected layers. Il layer finale manda in output un tensore di dimensione $S \times S \times (C + B \times 5)$ che corrisponde alle predizioni per ogni cella della griglia, C è il numero di probabilità stimate per ogni classe e B è il numero di anchor boxes per cella.



architettura modello YOLO : (link <https://arxiv.org/pdf/1506.02640v5>)

YOLOv8 in particolare presenta un'architettura più leggera ed efficiente rispetto alle versioni precedenti essendo progettata per ridurre il consumo di risorse. Inoltre fa uso di una vasta gamma di tecniche di data augmentation come la **“Mosaic augmentation”** e il **“MixUp”**. Queste permettono di migliorare le performance del modello, nonché di generalizzare meglio su dataset variegati. Gli autori provvedono inoltre un'implementazione dell'architettura per i moderni framework di deep learning, PyTorch e TensorFlow.



4. Valutazione

Una prima metrica di valutazione per modelli di object detection da considerare è la **Intersection over Union**, calcolata a partire delle aree dei due bounding box (l'originale, detta di ground truth, e quella predetto dal modello) e si ottiene dividendo l'intersezione delle due per l'unione, in formula si scrive come:

$$IoU = \frac{A \cap B}{A \cup B}$$

È un valore compreso tra 0 e 1 e rappresenta l'area che si sovrappone tra il bounding box predetto dal modello e quello di ground truth, più è alto questo valore migliore è la predizione del modello. Si può considerare di filtrare i bounding box predetti, mantenendo solo quelli con IoU superiore a una data threshold.

Prima di procedere con altre metriche è importante riportare i valori tipici delle matrici di confusione in ambito di object detection:

True Positive (TP): il modello riconosce e localizza correttamente un oggetto, con IoU che raggiunge o supera una determinata threshold.

False Positive (FP): un rilevamento inaccurato, il modello identifica un oggetto non presente in ground truth o il bounding box predetto non raggiunge la specifica IoU.

False Negative (FN): il modello non riesce a identificare un oggetto presente in ground truth.

True Negative (TN): nel caso di object detection non è molto usata, conferma l'assenza di oggetti; il nostro obiettivo è più quello di identificare oggetti che verificarne l'assenza.

Date queste possiamo definire le nostre metriche:

Precision: calcola la percentuale di True Positive tra tutte le previsioni positive, valutando quindi la capacità del modello di evitare i falsi positivi; di conseguenza è una metrica molto importante quando la priorità è ridurre al minimo i falsi rilevamenti.

$$Precision = \frac{TP}{TP + FP}$$

Recall: calcola la percentuale di veri positivi tra tutti i positivi effettivi, misurando la capacità del modello di rilevare tutte le istanze di una classe; metrica molto importante nel momento in cui è necessario rilevare ogni istanza di un oggetto.

$$Recall = \frac{TP}{TP + FN}$$

F1 score: è la media armonica di Precision e Recall, e rappresenta il **trade-off tra le due metriche**; fornisce una valutazione equilibrata delle prestazioni di un modello, considerando sia i falsi positivi che i falsi negativi.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Average Precision: denotata AP, questa metrica calcola l'area sotto la curva precision-recall, fornendo un unico valore che racchiude le prestazioni di Precision e Recall del modello. Si può rappresentare graficamente riportando la precisione $p(r)$ in funzione della recall r , calcolando quindi il valore medio di $p(r)$ nell'intervallo tra 0 e 1:

$$AP = \int_{r=0}^1 p(r) dr$$

Mean Average Precision: denotata mAP estende il concetto di Average Precision calcolando il valore medio di questa su più categorie di oggetti. È una metrica molto utile in scenari di rilevamento di oggetti multiclass per fornire una valutazione completa delle prestazioni del modello.

$$mAP = \frac{1}{n} \sum_{k=1}^n AP_k$$

la formula è da intendersi con n numero di classi e AP_k come Average Precision di classe k

5. Esperimenti

Abbiamo applicato i due metodi (**YOLOv8** e **Faster R-CNN**) sia sullo split basato su cluster, che su uno split randomico, calcolato con il medesimo rapporto train / test (81% - 19%). Ci aspettiamo che lo split randomico porti a risultati migliori, dato che gli esempi di train sono spazialmente vicini a quelli di test.

Per entrambi i modelli, abbiamo anche esplorato tecniche di data augmentation, in modo da migliorare ulteriormente le performance.

5.1 Faster R-CNN

Per questo metodo abbiamo allenato tutti i modelli per **10 epoch**, con una batch size pari a 4. Siamo partiti dal modello pre-allenato sul dataset COCO (versione 2017). Anche in questo caso sono state allenate 3 versioni, rispettivamente sugli split random e cluster (con e senza data augmentation). Per una migliore comparazione, implementiamo le tecniche mosaic e mixup manualmente (tecniche di cui YOLO fa uso, ma che il framework utilizzato non mette a disposizione nativamente). Nelle ultime due epoch, seguendo la metodologia di training di YOLO, la mosaic augmentation viene disattivata.

L'algoritmo di ottimizzazione usato è SGD, con learning rate 0.005, momentum 0.9, weight decay 0.0005. È stato utilizzato lo scheduler del learning rate StepLR, con step size 0.3 (epoch) e gamma 0.1.

5.2 YOLOv8

Tutti i modelli sono stati allenati per **50 epoch**, con una batch size pari a 64, mantenendo tutti i parametri allenabili, a meno del layer “model.22.dfl.conv.weight”. Le prime due versioni sono state allenate rispettivamente negli split random e cluster, senza utilizzare data

augmentation. In una terza versione, allenata sullo split cluster, abbiamo integrato anche trasformazioni di data augmentation; In particolare, viene applicato uno sfasamento nello spazio HSV, rispettivamente di 0.015 per hue, 0.7 per saturation e 0.4 per value, insieme alla trasformazione mixup, che avviene con probabilità 0.5. Inoltre, per tutte le epoches ad eccezione delle ultime 10, viene applicata la trasformazione mosaic. Le trasformazioni mosaic e mixup sono particolarmente utili per mitigare il problema di bias di cui abbiamo parlato in (**Sezione 2.3.1**).

L'algoritmo di ottimizzazione (AdamW), il learning rate (0.000714) e il momentum (0.9), sono stati determinati automaticamente dalla procedura di train di ultralytics.

5.3 Tabella completa dei risultati

Modello	Split	Augmentation	Precision	Recall	mAP50	mAp50-95	F1 score
YOLOv8	random	no	0.93561	0.80637	0.89617	0.82672	0.86619
YOLOv8	cluster	no	0.88267	0.8188	0.87437	0.73437	0.84953
YOLOv8	cluster	si	0.96523	0.8509	0.92777	0.77914	0.90446

Modello	Split	Augmentation	Precision	Recall	mAP50	mAp50-95	F1 score
Faster R-CNN	random	no	0.854	0.783	0.921	0.770	0.817
Faster R-CNN	cluster	no	0.864	0.783	0.938	0.771	0.821
Faster R-CNN	cluster	si	0.882	0.796	0.956	0.789	0.837

I risultati dei modelli YOLO sembrano confermare la nostra ipotesi iniziale, ovvero che il task risulta più semplice su uno split randomico, tuttavia questo non è accaduto con i modelli Faster R-CNN. Sarebbe quindi opportuno sperimentare con altri split randomici.

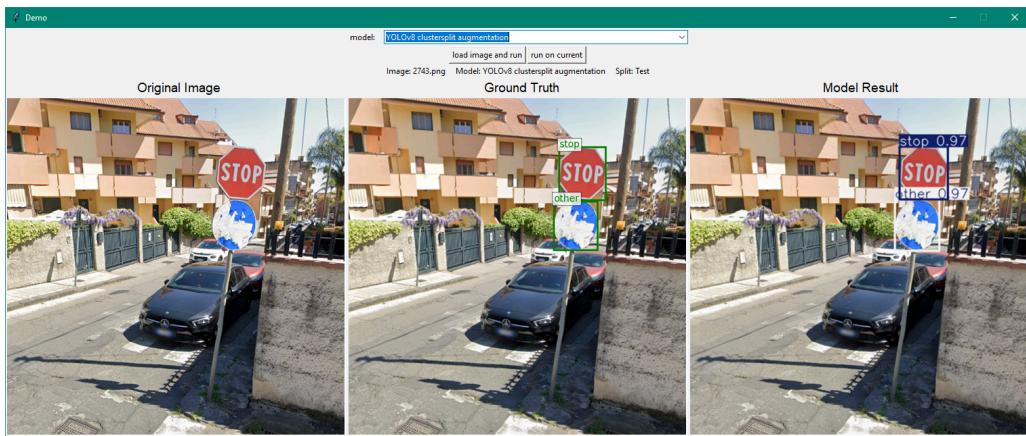
5.4 Considerazioni

Ai fini di questo progetto, abbiamo deciso di non utilizzare le trasformazioni di flip orizzontale e verticale. Alcuni dei cartelli del nostro dataset, infatti, non sono simmetrici. Se classi come “turn right” una volta flippatte orizzontalmente sono ancora rilevanti (basta invertire l'etichetta), altre producono risultati che non vengono mai riscontrati in inferenza (esempio: “no parking”, “stop”, “parking place”). Cartelli invece come “no stopping” possono essere flippati sia orizzontalmente che verticalmente. Bisognerebbe quindi applicare le trasformazioni di flip solo nelle immagini che lo permettono, creando un ulteriore sbilanciamento tra le classi. Ulteriori sviluppi potrebbero comunque esplorarne l'utilizzo insieme a tecniche di bilanciamento, come la funzione di perdita **Focal Loss**.

6. Demo

Per utilizzare la demo è necessario scaricare i modelli allenati: [models](#).

La demo realizzata fornisce una semplice interfaccia grafica che permette di selezionare uno tra i modelli allenati e di mostrare i suoi output su immagini caricate dall'utente.



Dopo aver caricato l'immagine (load image and run), il programma legge i file .txt relativi allo split su cui è allenato il modello selezionato, per verificare se si tratta di un'immagine di train o di test, e visualizza tale informazione. Successivamente vengono mostrati, in tre riquadri appositi, l'immagine caricata, le etichette di ground truth relative all'immagine, e le predizioni del modello selezionato. Se si desidera provare più modelli su una stessa immagine, sarà sufficiente scegliere un altro modello dal menù a tendina e premere “run on current”.

7. Codice

Il codice relativo all'allenamento dei modelli è contenuto nel notebook `train.ipynb`.

Altre statistiche, mostrate nelle sezioni precedenti, sono state ottenute con il codice del notebook `statistics.ipynb`.

Per replicare tutte le procedure di allenamento sarà sufficiente installare le dipendenze contenute in `requirements.txt` ed eseguire in ordine le celle del notebook `train.ipynb`.

Conclusione

In questo progetto abbiamo voluto sviluppare tutti i passaggi chiave della produzione di un modello di machine learning moderno.

Per prima cosa, abbiamo creato, pulito ed etichettato un dataset contenente oltre 3000 immagini e 5900 cartelli stradali annotati mediante bounding boxes 2D ottenute da Google Maps. Questa operazione è stata agevolata grazie ad un tool di nostra realizzazione.

Dopo aver effettuato un'analisi sui nostri dati (con particolare attenzione al clustering delle posizioni), abbiamo effettuato il fine tuning di architetture pre-allenate per il task di object detection e abbiamo mostrato come un piccolo dataset come il nostro sia comunque sufficiente a ottenere discrete performance sul task, e come queste possano essere migliorate con l'utilizzo di tecniche di data augmentation.

Per eventuali sviluppi futuri, sarebbe probabilmente utile considerare tecniche di ribilanciamento (magari basate su mappa di densità delle posizioni), inclusione ulteriore di nuovi dati, tecniche e modelli.