

Firearm Detection - Progetto Machine Learning AA. 2023/2024

Gianmarco Basile 1000001626 - Andrea Seminara 1000070001

Contents

1	Problema	3
2	Dataset	5
2.1	Acquisizione	5
2.2	Data Augmentation	5
2.3	Esempi visuali	7
2.4	Organizzazione dei file	9
2.4.1	Struttura del dataset Faster R-CNN	10
2.4.2	Struttura del dataset YOLOv11	10
3	Metodi	12
3.1	Faster R-CNN	12
3.1.1	Resnet50	13
3.1.2	Dropout	14
3.1.3	Motivazione	15
3.2	YOLO	15
3.2.1	YOLOv11	16
3.2.2	Motivazione	17
4	Valutazioni	19
4.1	Metriche di Valutazione	19
4.1.1	Precision	19
4.1.2	Recall	19
4.1.3	F1-Score	19
4.1.4	Mean Average Precision	20
4.2	Implementazione	20
4.3	Visualizzazione dei Risultati	21

5 Esperimenti	23
5.1 Sistema	23
5.2 Modelli	23
5.2.1 Faster R-CNN	24
5.2.2 YOLO	33
6 Demo	35
7 Codice	36
7.1 Preparazione del Dataset	36
7.2 Training dei Modelli	36
7.3 Inferenza	37
7.4 Demo	38
8 Conclusione	39
Bibliografia	41

Chapter 1

Problema

Il rilevamento automatico di armi da fuoco è un settore in rapida evoluzione nell’ambito della computer vision applicata ai sistemi di sicurezza. In un contesto globale dove la videosorveglianza è sempre più diffusa, l’identificazione di potenziali minacce può costituire un elemento cruciale per la prevenzione di eventi criminali. I sistemi tradizionali di videosorveglianza richiedono un monitoraggio costante da parte di operatori umani, i quali possono facilmente imbattersi in errori dovuti a stanchezza, distrazione o alla necessità di osservare simultaneamente molteplici fonti video. L’automazione del processo di riconoscimento di armi attraverso algoritmi di deep learning offre una soluzione efficace a questi limiti, consentendo l’analisi in tempo reale di flussi video e la generazione di allerte immediate in caso di rilevamento. Le applicazioni di un sistema di firearm detection spaziano dalla sorveglianza in luoghi pubblici come scuole, banche, centri commerciali e stazioni, fino all’integrazione in sistemi di sicurezza avanzati per edifici governativi e infrastrutture critiche. Inoltre, queste tecnologie possono essere implementate in dispositivi indossabili per le forze dell’ordine che effettuano attività di pattugliamento di aree sensibili.

Il presente progetto si propone di sviluppare e confrontare sistemi di rilevamento automatico di armi da fuoco in immagini e fonti video real-time attraverso l’implementazione e l’ottimizzazione di due architetture di deep learning: Faster R-CNN e YOLOv11.

Gli obiettivi specifici includono:

- Implementare e specializzare le architetture Faster R-CNN e YOLOv11 al task specifico di rilevamento di armi da fuoco, esplorando diverse configurazioni e ottimizzazioni.

- Applicare tecniche avanzate di regolarizzazione e data augmentation per migliorare la robustezza e le performance dei modelli, affrontando in particolare il problema dell'overfitting su dataset relativamente limitati.
- Valutare e confrontare le prestazioni dei diversi approcci.
- Creare un sistema dimostrativo funzionale che possa essere facilmente testato su nuove immagini, fornendo un punto di partenza per implementazioni in contesti reali.

Lo scopo ultimo è determinare quali configurazioni e approcci offrono prestazioni migliori per il rilevamento di pistole in diverse condizioni, fornendo così una base solida per lo sviluppo di sistemi applicabili in scenari di sicurezza reali.

Chapter 2

Dataset

2.1 Acquisizione

Per il progetto sono stati utilizzati due dataset distinti: uno per le fasi di training e validation del modello, e uno specifico per la fase di testing.

Il dataset principale, reperito pubblicamente da Roboflow (<https://universe.roboflow.com/xian-douglas/weapondetection-xx3lz/dataset/5>), è composto da 17.304 immagini ripartite con una suddivisione che assegna 13.282 immagini al training set e 4.022 immagini al validation set. Il dataset non include di default nessuna data augmentation.

Per la fase di testing è stato realizzato un dataset personalizzato, raccolgendo foto che ritraggono oggetti target (pistole giocattolo e pistole reali) in diversi contesti e condizioni, garantendo così una valutazione più realistica del modello.

Dal punto di vista implementativo, la gestione dei dataset è stata automatizzata attraverso notebook jupyter dedicati: `create_dataset.ipynb` per il dataset con annotazioni COCO e `create_dataset_yolo.ipynb` per il dataset con annotazioni YOLO, adottato per la compatibilità diretta con l'omonimo framework di detection.

2.2 Data Augmentation

La pipeline di data augmentation per il modello basato su Faster R-CNN è stata implementata utilizzando la libreria `Albumentations`, applicando le seguenti trasformazioni:

```

train_transform = A.Compose([
    A.LongestMaxSize(max_size=640),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.Rotate(limit=10, p=0.5),
    A.MedianBlur(blur_limit=3, p=0.1),
    A.Normalize(mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]),
    ToTensorV2()
], bbox_params=A.BboxParams(
    format='pascal_voc',
    label_fields=['labels'],
    min_visibility=0.3,
    min_area=0
))

```

Per il fine-tuning del modello YOLO, invece, sono state utilizzate trasformazioni quali rotazione, traslazione, scaling, flipping, mosaic augmentation e variazione di tonalità/saturazione/luminosità, sfruttando le funzionalità integrate nel framework YOLO per migliorare la robustezza del modello a variazioni spaziali e contestuali.

Questa strategia di preprocessing consente di migliorare la generalizzazione del modello, evitando il rischio di sovraccarico ai dati di training e migliorando la capacità di rilevare gli oggetti target in scenari reali.

2.3 Esempi visuali



Figure 2.1: Esempio immagine training con annotazione

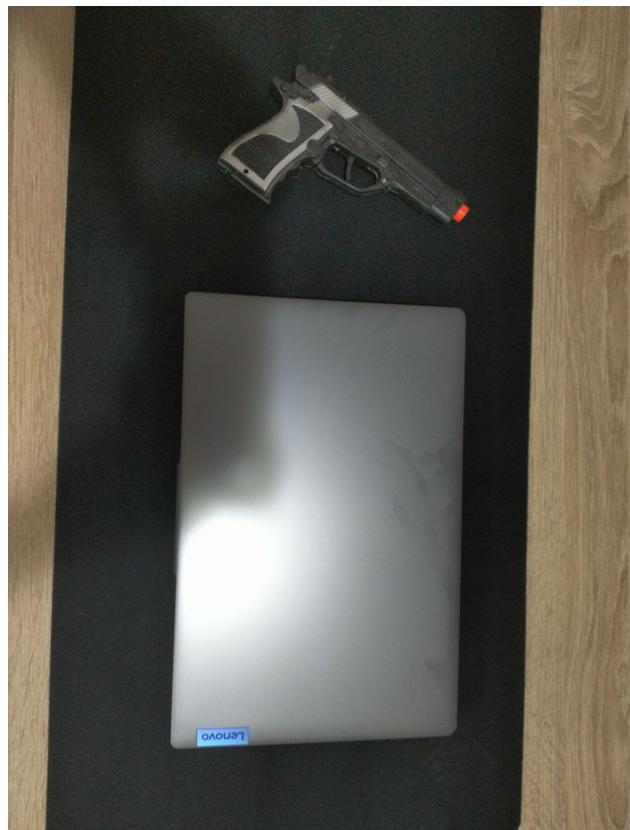
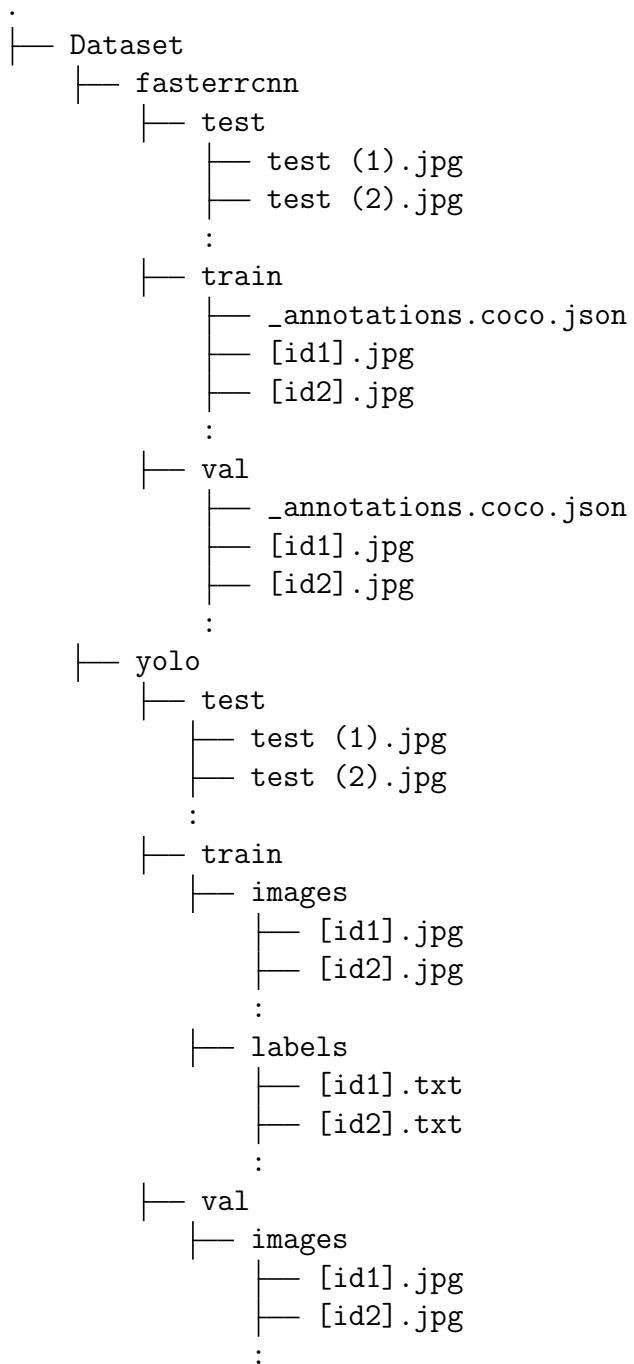
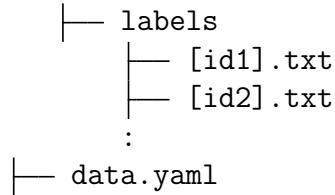


Figure 2.2: Esempio immagine di test

2.4 Organizzazione dei file





I dataset per i due approcci Faster R-CNN e YOLOv11 sono organizzati in cartelle differenti: `fasterrcnn` e `yolo`, ognuna delle quali contiene tre sottocartelle: `train`, `val` e `test`. La seguente organizzazione permette di gestire al meglio i dati per entrambi i modelli, garantendo compatibilità con i rispettivi framework.

2.4.1 Struttura del dataset Faster R-CNN

- **Train e Validation:** contengono le immagini e un file `_annotations.coco.json`, che contiene informazioni dettagliate riguardo le immagini, le categorie e le annotazioni degli oggetti in formato COCO.
- **Test:** contiene solo le immagini su cui verrà effettuata l'inferenza finale volta a valutare il modello.

2.4.2 Struttura del dataset YOLOv11

- **Train e Validation:** organizzati in due sottocartelle:
 - `images/`: contiene le immagini del dataset.
 - `labels/`: contiene i file di annotazione in formato YOLO (un file `.txt` per ogni immagine) con la seguente struttura:


```
<class_id> <x_center> <y_center> <width> <height>
```

 dove le coordinate sono normalizzate tra 0 e 1 rispetto alle dimensioni dell'immagine.
- **Test:** contiene solo immagini, analogamente alla struttura del dataset Faster R-CNN.
- **data.yaml:** file di configurazione contenente informazioni essenziali per l'addestramento di YOLOv11, tra cui:

- Percorsi delle cartelle `train`, `val` e `test`.
- Numero e nomi delle classi presenti nel dataset.

Chapter 3

Metodi

Per la realizzazione di questo progetto sono stati adottati due principali approcci: uno basato su Faster R-CNN e uno su YOLOv11. Il primo comprende tre varianti dello stesso modello, differenziate dall'uso di dropout e diverse tecniche di data augmentation, mentre il secondo rappresenta un'architettura distinta, nota per la sua velocità ed efficienza.

Di seguito verranno mostrati nel dettaglio i modelli adottati e le tecniche impiegate, evidenziando i dettagli tecnici delle soluzioni proposte e le motivazioni alla base delle scelte effettuate.

3.1 Faster R-CNN

Faster R-CNN è un'architettura di object detection composta da due moduli, che suddividono il rilevamento in due fasi principali:

- nella prima si ha una Region Proposal Network (RPN), una rete che genera delle regioni candidate, o proposte, ottimizzando così il numero di aree da analizzare
- nella seconda si ha una rete di classificazione che assegna loro una classe e una di regressione che perfeziona le coordinate delle bounding box

Faster R-CNN può usare diversi backbone, reti addestrate per l'estrazione di caratteristiche.

Il backbone utilizzato per tutti e tre i modelli analizzati è Resnet-50.

3.1.1 Resnet50

Resnet50 (*Residual Network 50 layers*) è una rete neurale convoluzionale composta da 50 livelli, sviluppata per contrastare il problema della degradazione delle prestazioni nelle reti molto profonde. Questo problema si verifica quando, aumentando il numero di strati, l'accuratezza smette di migliorare e in alcuni casi peggiora, a causa della difficoltà nel propagare il gradiente ai livelli inferiori durante l'addestramento.

Architettura

Come mostrato nella figura 3.1, l'architettura di Resnet50 è così divisa:

1. Strato iniziale
 - **Zero Padding** per mantenere le dimensioni dell'input
 - **Convoluzione iniziale (CONV)** per estrarre le prime feature.
 - **Batch Normalization** e **ReLU** per stabilizzare l'addestramento.
 - **Max Pooling** per ridurre la dimensionalità e mantenere le informazioni più rilevanti.
2. Blocchi convoluzionali residuali
 - La rete è composta da blocchi convoluzionali (Conv Block) e blocchi identità (ID Block).
 - Ogni blocco include convoluzioni, normalizzazione e attivazioni, ma la chiave è la connessione diretta (skip connection), che permette di saltare alcuni strati e sommare direttamente l'input all'output del blocco. Questo consente di mantenere un flusso informativo più stabile durante l'addestramento.
3. Livelli finali
 - Dopo l'estrazione delle feature, viene applicato un **Average Pooling** per ridurre la dimensionalità dell'output.
 - Il risultato viene appiattito (**Flattening**) e passato a un **Fully Connected Layer (FC)** per la classificazione finale.

Grazie a questa architettura basata su blocchi residuali, ResNet50 è in grado di essere molto profonda senza soffrire della degradazione delle prestazioni, risultando una delle reti più efficaci per l'estrazione di feature e il trasferimento di conoscenza in molte applicazioni di computer vision.

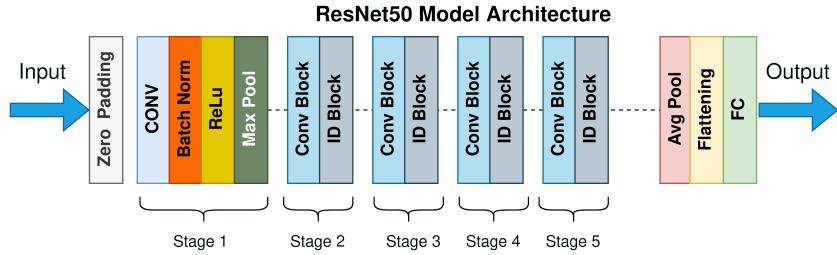


Figure 3.1: Architettura di Resnet50

3.1.2 Dropout

Il Dropout è una tecnica di regolarizzazione introdotta per ridurre l’overfitting nei modelli di deep learning. Applicata nella fase di training, consiste nel disattivare casualmente un certo numero di neuroni all’interno della rete neurale in ciascun forward pass, impedendo al modello di dipendere troppo da specifici nodi e promuovendo una rappresentazione più robusta e generalizzabile. In ogni iterazione del training, una percentuale predefinita di unità viene “spenta” (tipicamente il 20–50%). Questo significa che i pesi associati a quei neuroni non vengono aggiornati e non partecipano al calcolo dell’output. Durante l’inferenza, tutti i neuroni sono attivi, ma i pesi vengono scalati in modo da mantenere la coerenza tra training e test.

Questa strategia fornisce diversi vantaggi:

- Riduce significativamente l’overfitting in reti profonde.
- Favorisce la creazione di rappresentazioni più sparse e generalizzabili.
- Agisce come una forma隐式的 ensemble, combinando più sottoreti generate casualmente.

Tuttavia, il dropout

- può rallentare la convergenza del training, poiché la rete deve imparare a funzionare in condizioni di alta variabilità.

- non sempre è efficace in combinazione con alcune architetture, come le reti convoluzionali molto profonde o i modelli pre-addestrati, se non applicato in modo selettivo.

3.1.3 Motivazione

La scelta di utilizzare Faster R-CNN si basa su una combinazione di accuratezza, flessibilità e solidità architetturale che rende questo modello particolarmente adatto a compiti di rilevamento complessi, come la firearm detection.

- **Alta accuratezza:** Faster R-CNN è noto per offrire ottimi risultati in termini di precisione e recall, soprattutto nei contesti in cui è fondamentale localizzare con precisione oggetti di piccole o medie dimensioni.
- **Architettura modulare e personalizzabile:** la struttura del modello permette di integrare diversi backbone (come ResNet-50), facilitando il fine-tuning su dataset specifici.
- **RPN integrata (Region Proposal Network):** Faster R-CNN incorpora una rete per la generazione automatica delle proposte (region proposals), migliorando significativamente la velocità e l'efficienza del processo di rilevamento.
- **Ampia disponibilità di implementazioni e supporto:** essendo uno dei modelli più studiati e utilizzati, è ben supportato da librerie come **Torchvision**, con numerosi modelli pre-addestrati disponibili.

3.2 YOLO

YOLO (*You Only Look Once*) è una famiglia di modelli per la object detection progettata per eseguire predizioni in tempo reale, combinando velocità ed efficienza. YOLO adotta un'architettura single-stage, che permette di rilevare oggetti e stimarne le coordinate in un'unica passata della rete.

YOLO suddivide l'immagine in una griglia e, per ciascuna cella, predice direttamente le classi e le bounding boxes degli oggetti. Questo approccio consente di eseguire rilevamenti molto rapidi, rendendolo ideale per applicazioni in tempo reale, come videosorveglianza, sistemi embedded o dispositivi mobili.

3.2.1 YOLOv11

YOLOv11 rappresenta una delle versioni più recenti e avanzate della famiglia YOLO, progettata per migliorare ulteriormente il compromesso tra accuratezza, velocità e leggerezza del modello, introducendo numerose ottimizzazioni architetturali e funzionali.

Tra le principali novità, YOLOv11 adotta tecniche moderne di ottimizzazione come l'uso di blocchi convoluzionali più efficienti, migliori strategie di normalizzazione e un training potenziato da data augmentation aggressive e strategie di warmup più raffinate. Tutto ciò contribuisce a migliorare la capacità del modello di generalizzare su dataset complessi, anche in presenza di oggetti piccoli o parzialmente occlusi.

Un altro punto di forza di YOLOv11 è la sua ottimizzazione per l'inferenza: la struttura del modello è pensata per essere eseguita efficientemente anche su GPU meno potenti, mantenendo prestazioni elevate in termini di accuratezza.

Architettura

Come mostrato nella figura 3.2, l'architettura di YOLOv11 può essere suddivisa in tre componenti principali:

- **Backbone:** estrae le feature di un'immagine in input (640×640). È costituito da una sequenza di layer convoluzionali (Conv) e da blocchi **C3K2** (blocco convoluzionale ottimizzato con architettura più leggera), che permettono una combinazione efficiente di convoluzioni e connessioni shortcut. Nelle fasi finali del backbone vengono introdotti due moduli avanzati:
 - **SPPF** (Spatial Pyramid Pooling - Fast), per l'aggregazione multiscala delle informazioni spaziali;
 - **C2PSA** (Cross-Stage Partial Self-Attention), un meccanismo di attenzione che migliora la selettività delle feature.
- **Neck:** serve a combinare le feature provenienti da diversi livelli di profondità. Comprende operazioni di Upsampling (operazione di incremento della risoluzione spaziale delle feature map), concatenazione (Concat) e ulteriori blocchi C3K2. Questa parte dell'architettura funge da ponte tra l'estrazione e la predizione, rendendo la rete più sensibile sia a oggetti piccoli che grandi.

- **Head:** composta da tre rami paralleli, ciascuno destinato alla rilevazione su una scala specifica. Ogni ramo produce predizioni relative a una bounding box, classi e punteggi di confidenza.

4

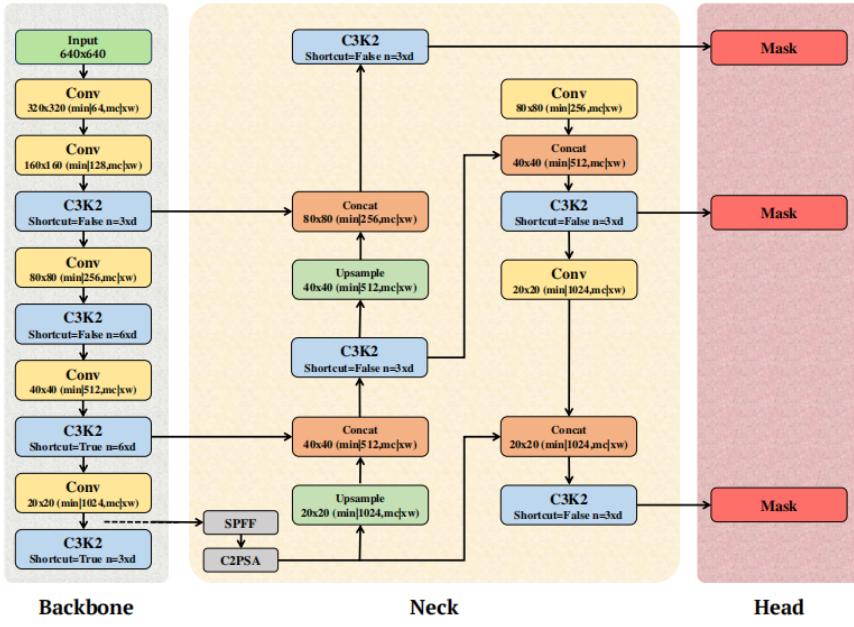


Figure 3.2: Architettura di YOLOv11

3.2.2 Motivazione

YOLOv11 è stato scelto per le sue caratteristiche architettoniche moderne, che lo rendono particolarmente adatto a compiti di object detection in ambienti con vincoli di tempo e risorse. Le principali motivazioni della scelta sono elencate di seguito:

- **Efficienza computazionale:** l'architettura è progettata per essere leggera e veloce, risultando adatta a scenari real-time e a dispositivi con risorse limitate.

- **Predizione multiscala:** grazie all'uso di tre rami paralleli nella head e alla struttura del neck, il modello riesce a rilevare oggetti di diverse dimensioni con buona sensibilità.
- **Adattabilità a contesti reali:** le ottimizzazioni introdotte in YOLOv11 lo rendono particolarmente adatto a essere integrato in sistemi applicativi concreti, dove la rapidità di risposta è un requisito fondamentale.

Per queste ragioni, YOLOv11 è stato selezionato anche come termine di confronto all'interno del progetto, offrendo una prospettiva complementare rispetto ai modelli two-stage della famiglia Faster R-CNN.

Chapter 4

Valutazioni

4.1 Metriche di Valutazione

Per valutare le prestazioni dei modelli sono state utilizzate diverse metriche standard nel campo dell'object detection:

4.1.1 Precision

La precision misura il rapporto tra i rilevamenti corretti rispetto a tutti i rilevamenti effettuati dal modello. È definita come:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (4.1)$$

4.1.2 Recall

Il recall misura il rapporto di oggetti rilevati rispetto a tutti gli oggetti presenti nelle immagini. È definito come:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (4.2)$$

4.1.3 F1-Score

L'F1-Score rappresenta la media armonica tra precisione e recall e fornisce una metrica bilanciata che tiene conto di entrambi i valori:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (4.3)$$

4.1.4 Mean Average Precision

La mAP (Mean Average Precision) è una metrica di valutazione molto usata per misurare la capacità di un modello di rilevare e classificare oggetti all'interno delle immagini.

L'Average Precision è la media della precisione calcolata a diversi livelli di recall per una certa classe. Viene generata la curva PR (Precision-Recall) e viene calcolata l'area sotto questa curva (AUC, area under the curve).

La mAP (mean Average Precision) è la media dell'Average Precision su tutte le classi.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (4.4)$$

Dove N è il numero delle classi e AP_i è l'Average Precision dell'i-esima classe.

4.2 Implementazione

L'implementazione delle metriche di valutazione si basa sulla libreria `torchmetrics`. Di seguito viene presentato il codice principale:

```
from torchmetrics.detection import MeanAveragePrecision

# Inizializzazione della metrica
metric = MeanAveragePrecision(
    box_format='xyxy',
    iou_thresholds=[
        0.5, 0.55, 0.6,
        0.65, 0.7, 0.75,
        0.8, 0.85, 0.9, 0.95
    ]
)

# Aggiornamento della metrica durante
# il training/validation
```

```

metric.update( predictions , targets)

# Calcolo delle metriche
metric_results = metric.compute()

# Estrazione dei risultati
metrics = {
    'mAP': metric_results[ 'map' ].item() ,
    'mAP_50': metric_results[ 'map_50' ].item() ,
    'mAP_75': metric_results[ 'map_75' ].item() ,
    'recall': metric_results[ 'mar_100' ].item() ,
    'f1': 2 * ( metric_results[ 'map' ].item() *
                metric_results[ 'mar_100' ].item() ) /
            ( metric_results[ 'map' ].item() +
              metric_results[ 'mar_100' ].item() + 1e-6)
}

```

Le metriche vengono calcolate sia durante il training che durante la validazione, permettendo di:

- Monitorare l'andamento dell'apprendimento
- Identificare problemi di overfitting/underfitting
- Selezionare il miglior modello (early stopping)
- Confrontare diverse configurazioni del modello

4.3 Visualizzazione dei Risultati

I risultati delle metriche vengono visualizzati attraverso grafici che mostrano l'andamento durante il training:

- Training vs Validation mAP
- Training vs Validation mAP@0.5
- Training vs Validation mAP@0.75
- Training vs Validation Recall

- Training vs Validation F1-Score
- Training vs Validation Loss

Questi grafici permettono di valutare visivamente le prestazioni del modello e identificare potenziali problemi durante l'addestramento.

Chapter 5

Esperimenti

5.1 Sistema

Tutti gli esperimenti sono stati condotti su un computer con le seguenti caratteristiche:

- **CPU:** AMD Ryzen 5 7600
- **GPU:** NVIDIA RTX 3060 12GB
- **RAM:** 32 GB DDR5
- **OS:** Windows 11 Pro 24H2 (build SO 26100.3775)

Data la VRAM disponibile, è stato adottato un **batch size** di 4.

5.2 Modelli

Per la valutazione del sistema di Firearm Detection, sono stati condotti diversi esperimenti utilizzando due principali modelli:

- **Faster R-CNN con backbone ResNet50**
- **YOLOv11**

5.2.1 Faster R-CNN

Sono state valutate due varianti dell'architettura Faster R-CNN con backbone Resnet50:

- **FPN**: Utilizza una struttura piramidale standard per l'estrazione delle feature;
- **FPNv2[2]** - Una versione che apporta alcune migliorie alla versione standard.

Tutti gli esperimenti discussi di seguito hanno effettuato un massimo di 10 epoche di fine-tuning con un early stopping con soglia di pazienza pari a 3 epoche.

Esperimento 1 Inizialmente è stato utilizzato un dataset con ~ 7000 immagini e nonostante aver implementato dropout, data augmentation e regolarizzazione, il risultato presentava un evidente overfitting sui dati di training.

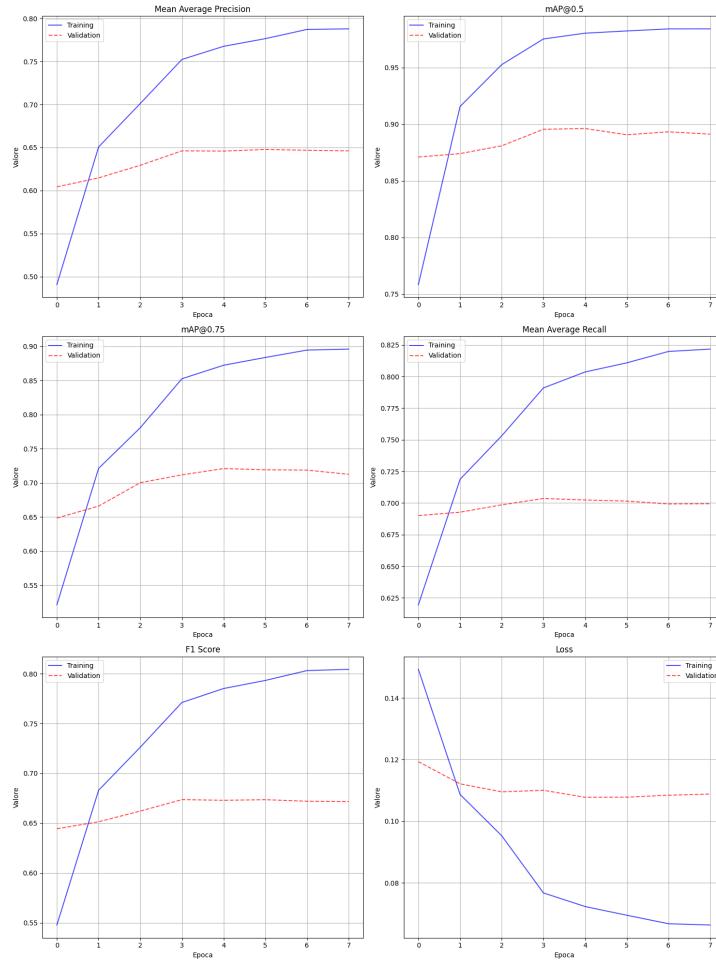


Figure 5.1: Esperimento Faster R-CNN in overfitting

A seguito di ciò, si è deciso di adottare il dataset descritto nel capitolo 2, che, grazie alla presenza di molti più esempi, ha permesso al modello di generalizzare in maniera più efficiente.

Esperimento 2 Dopo aver sostituito il dataset con uno con più esempi è stato condotto un esperimento effettuando un fine-tuning sul modello faster-rcnn_resnet50_fpn, fornito dalla libreria `torchvision`, inizializzato con i pesi pre-addestrati. Questa scelta consente di partire da un modello già in grado di estrarre feature, riducendo così i tempi di convergenza.

Si è scelto come ottimizzatore **Stochastic Gradient Descent (SDG)**, noto per la sua efficacia nel training di deep neural network, inizializzandolo con i seguenti iperparametri:

- Learning Rate: 0.001;
- Momentum: 0.9;
- Weight Decay: 0.0005;

Questi iperparametri sono stati utilizzati anche per gli altri esperimenti condotti su Faster R-CNN.

Inoltre, è stato introdotto un **learning rate scheduler** che riduce il learning rate del 30% ogni 3 epoche, al fine di migliorare la stabilità e favorire la convergenza.

Il processo di training in questo esperimento ha portato migliorie in termini di generalizzazione, pur avendo performance leggermente inferiori sui dati di training rispetto all'esperimento precedente.

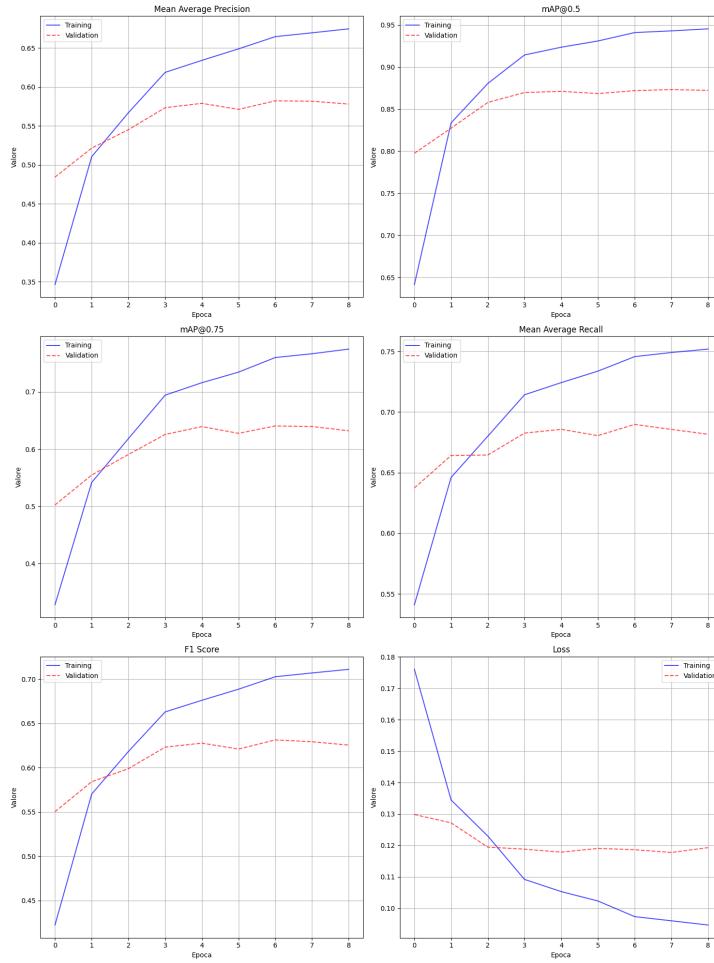


Figure 5.2: Esperimento Faster R-CNN

Il modello in questione, pur riuscendo a rilevare correttamente tutte le armi presenti nelle immagini di test, presenta qualche falso positivo.

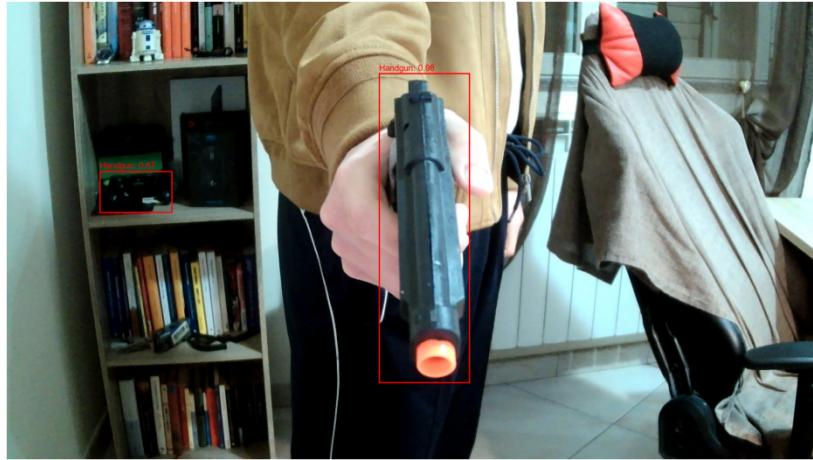


Figure 5.3: Esempio di Falso Positivo

Esperimento 3 Al fine di migliorare la generalizzazione è stato modificato il modello per implementare il dropout e data augmentation (definito nel paragrafo 2.2).

Di seguito viene presentato il codice per l'implementazione del dropout:

```
class FasterRCNNPredictorWithDropout(FastRCNNPredictor):
    def __init__(self,
                 in_channels,
                 num_classes,
                 dropout_prob=0.3):
        super(FasterRCNNPredictorWithDropout, self)
        .__init__(in_channels, num_classes)
        self.dropout = nn.Dropout(p=dropout_prob)

    def forward(self, x):
        x = self.dropout(x)
        scores = self.cls_score(x)
        bbox_deltas = self.bbox_pred(x)
        return scores, bbox_deltas
```

I risultati di questo esperimento hanno mostrato un evidente miglioramento in termini di generalizzazione, pur mantenendo delle performance generali lievemente più basse.

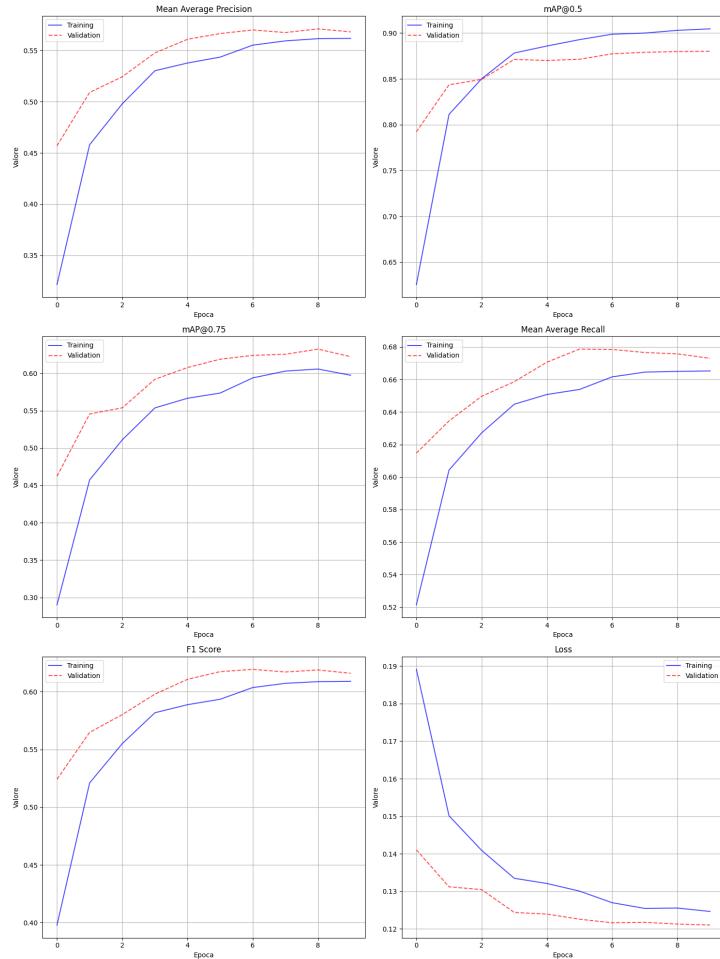


Figure 5.4: Esperimento Faster R-CNN con dropout e data augmentation

Nonostante le performance siano diminuite leggermente, il modello non rileva più falsi positivi sulle immagini di test.

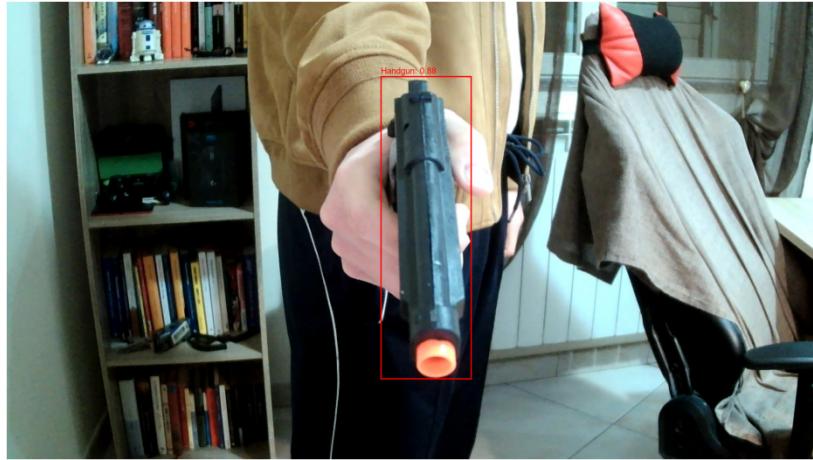


Figure 5.5: Esempio di falso positivo non più rilevato

Eperimento 4 In questo esperimento è stato adottato il modello `fasterrcnn_resnet50_fpn_v2`, sempre fornito dalla libreria `torchvision`, in sostituzione della versione standard `fasterrcnn_resnet50_fpn`. Questa scelta è stata motivata dal desiderio di migliorare ulteriormente le performance generali del sistema, grazie alle ottimizzazioni introdotte nella versione v2.

La configurazione degli iperparametri e del processo di training è rimasta invariata rispetto all'esperimento precedente (dropout, data augmentation, learning rate scheduler, ecc.).

I risultati, ottenuti al termine di un training di 7 epoch (interrotto anticipatamente per convergenza del modello), hanno evidenziato un miglioramento complessivo sia in termini di accuratezza sia di robustezza nella rilevazione, con una maggiore precisione nel localizzare le armi anche in condizioni di illuminazione o contesto non ottimali.

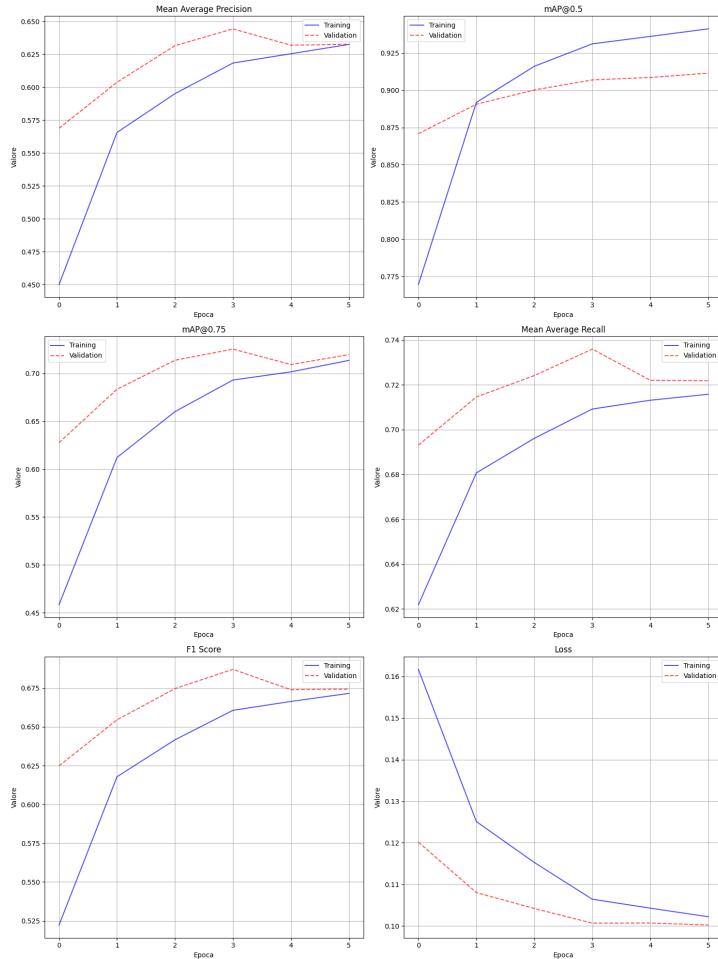


Figure 5.6: Esperimento Faster R-CNN v2 con dropout e data augmentation

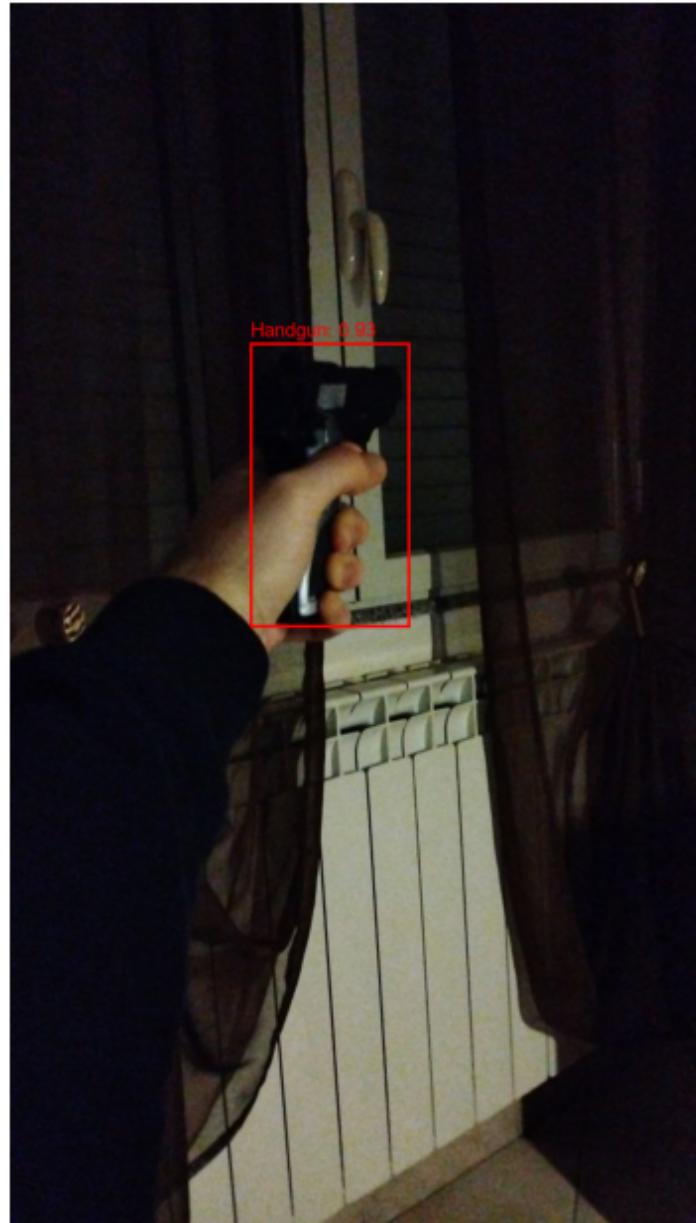


Figure 5.7: Esempio di rilevamento in condizioni di luce sfavorevole

5.2.2 YOLO

Eperimento 5 Come ultimo esperimento, è stato adottato il modello **YOLOv11** con l’obiettivo di realizzare un sistema di rilevamento in tempo reale. A differenza degli esperimenti precedenti, che si concentravano principalmente su immagini statiche, in questo caso si è lavorato su un flusso video proveniente direttamente dalla webcam, per valutare l’efficacia del modello in scenari dinamici e realistici.

Il training è stato effettuato, per 100 epochhe e con un early stopping con patience a 20 epochhe, con i seguenti iperparametri principali:

- **Scheduler:** Cosine Learning Rate Scheduler che fa decadere il learning rate seguendo una curva cosinusoidale da lr_0 fino a $lr_{final} = lr_0 * lrf$;
- **Learning Rate iniziale (lr0):** 0.01;
- **Learning Rate finale:** 0.0001;
- **Momentum:** 0.9;
- **Weight Decay:** 0.0005;
- **Warmup:** 3 epochhe con momentum iniziale 0.8 e **bias_lr** pari a 0.1;

Il modello addestrato ha mostrato ottime capacità di rilevamento in tempo reale, con una buona capacità di generalizzazione anche su scene non viste in fase di training.

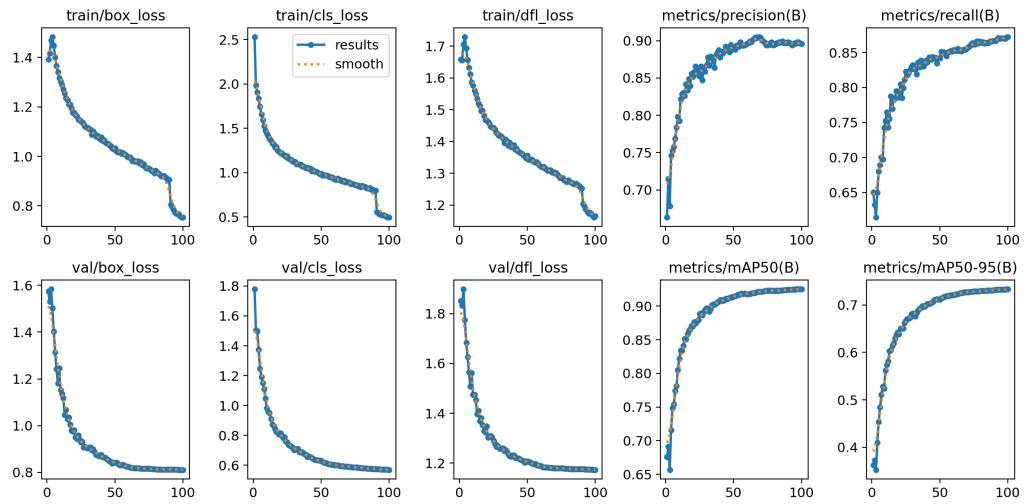


Figure 5.8: Esperimento YOLO

Chapter 6

Demo

Nel progetto sono stati sviluppati diversi notebook, ciascuno dedicato a uno specifico esperimento, con l’obiettivo di testare il funzionamento dei modelli su un sottoinsieme ristretto del dataset.

La demo consiste nel caricamento dei modelli addestrati all’interno di ciascun notebook, che eseguono inferenza sulle immagini contenute nella cartella `demo_images`. Ogni notebook è progettato per testare un modello diverso, consentendo di osservare le prestazioni di ciascuno su nuovi dati.

Inoltre, è disponibile un video dimostrativo, accessibile al seguente link: <https://www.youtube.com/watch?v=ZK4QzuFetk4> o all’interno della repository github (<https://github.com/AndreaSeminara/FirearmDetection>), in cui viene mostrato come eseguire le demo e viene presentata una dimostrazione in tempo reale dell’inferenza del modello YOLO su un flusso video proveniente dalla webcam.

Chapter 7

Codice

Il progetto è strutturato in maniera modulare per facilitare lo sviluppo, il training e l'inferenza dei modelli di rilevamento delle armi da fuoco. La directory principale `Code/` contiene i notebook jupyter necessari per le diverse fasi del workflow. Di seguito viene descritta l'organizzazione del codice.

7.1 Preparazione del Dataset

- `create_dataset.ipynb`: Questo notebook si connette a Roboflow per scaricare un dataset formattato secondo lo standard COCO. Una volta scaricato, il dataset viene organizzato in cartelle specifiche per facilitarne l'utilizzo nelle fasi successive del progetto.
- `create_dataset_yolo.ipynb`: Simile al precedente, ma pensato per il formato YOLOv11. Questo notebook scarica il dataset da Roboflow e lo suddivide automaticamente in tre cartelle principali: `train`, `valid` e `test`. Questo passaggio è fondamentale per garantire che i dati siano pronti per l'addestramento e la validazione dei modelli YOLO.

7.2 Training dei Modelli

- `fine_tuning.ipynb`: Implementa il training del modello Faster R-CNN con configurazioni di base. Utilizza `torchvision.models.detection` per caricare il modello pre-addestrato e adattarlo al dataset specifico.

- `fine_tuning_dropout_data_augmented.ipynb`: Questo notebook estende il precedente introducendo tecniche di *dropout* e *data augmentation*. Le trasformazioni applicate includono rotazioni, modifiche di luminosità e contrasto, che aiutano a rendere il modello più robusto e capace di generalizzare su dati mai visti in precedenza.
- `fine_tuning_dropout_data_augmented_v2.ipynb`: Migliora ulteriormente il training utilizzando la versione `fasterrcnn_resnet50_fpn_v2`.
- `fine_tuning_yolo.ipynb`: Questo notebook è dedicato al training del modello YOLOv11. Include una classe chiamata `YOLOTraainer`, che permette di configurare parametri come il batch size, il learning rate e le tecniche di augmentation specifiche per YOLO.

7.3 Inferenza

- `inference_fasterrcnn.ipynb`: Esegue l'inferenza utilizzando il modello base Faster R-CNN. Il notebook include funzioni per caricare immagini di test e visualizzare le predizioni, mostrando le aree rilevate e le relative classi. Il notebook include anche l'implementazione del Non-Maximum Suppression (NMS), una tecnica che elimina le predizioni ridondanti per migliorare la qualità dei risultati.
- `inference_fasterrcnn_dropout_data_augmented.ipynb`: Qui viene utilizzato il modello migliorato con dropout e data augmentation. Come nel caso precedente, vengono visualizzate le immagini con le relative predizioni.
- `inference_fasterrcnn_dropout_data_augmented_v2.ipynb`: Questo notebook utilizza il modello avanzato `fasterrcnn_resnet50_fpn_v2` per l'inferenza e, in modo analogo ai notebook precedentemente trattati, mostra le immagini con le bouding box relative alle predizioni.
- `inference_yolo.ipynb`: Esegue l'inferenza utilizzando YOLOv11. Oltre a lavorare con immagini statiche, il notebook sfrutta la libreria OpenCV per eseguire inferenze in tempo reale su flussi video, come quelli provenienti da una webcam.

7.4 Demo

All'interno della cartella `Demo/` è possibile trovare notebook utili al fine di verificare il comportamento del modello su un campione ristretto di immagini collocate all'interno della cartella `Demo/demo_images`.

- `inference_demo_fastercnn.ipynb`: Dimostrazione dell'inferenza con il modello base Faster R-CNN su un set di immagini demo.
- `inference_demo_fastercnn_dropout_data_augmented.ipynb`: Mostra l'inferenza con il modello migliorato, evidenziando i benefici delle tecniche di *dropout*.
- `inference_demo_fastercnn_dropout_data_augmented_v2.ipynb`: Mostra l'inferenza sul modello `fastercnn_resnet50_fpn_v2`.
- `inference_demo_yolo.ipynb`: Dimostrazione dell'inferenza con YOLOv11, sia su immagini che in tempo reale tramite webcam.

Chapter 8

Conclusione

Nel corso di questo progetto è stato sviluppato e confrontato un sistema di rilevamento automatico di armi da fuoco utilizzando due approcci distinti: Faster R-CNN e YOLOv11. Sono state esplorate tre varianti del primo approccio, tutte basate su ResNet-50 e potenziate con tecniche di regolarizzazione (dropout e weight decay) e strategie di data augmentation, affiancate da un modello YOLOv11 ottimizzato per l'inferenza in tempo reale.

Le sperimentazioni condotte hanno permesso di evidenziare vantaggi e limiti di ciascun metodo: mentre Faster R-CNN ha dimostrato un'elevata accuratezza su immagini statiche, YOLOv11 ha spiccato per velocità ed efficacia in scenari dinamici, risultando particolarmente adatto all'uso in contesti real-time.

Durante il lavoro sono emerse alcune importanti considerazioni:

- La qualità e varietà del dataset giocano un ruolo chiave nella capacità di generalizzazione del modello.
- Le tecniche di data augmentation e regolarizzazione, se ben calibrate, permettono di mitigare fenomeni di overfitting, come dimostrato dai test condotti con dropout.
- L'uso di metriche (mAP, Recall, F1-score) e la loro visualizzazione durante il processo di training sono strumenti preziosi per monitorare e ottimizzare le prestazioni dei modelli.

Per raffinare ulteriormente il sistema, si suggeriscono i seguenti sviluppi futuri:

- Ampliamento del dataset, includendo immagini catturate in ambienti reali, con illuminazioni e angolazioni più varie, per una valutazione ancora più rappresentativa.
- Impiego di modelli più complessi che, affiancati da dataset più eterogenei, potrebbero garantire rilevazioni più robuste e prestazioni superiori.

Bibliography

- [1] Xian Douglas. Weapondetection dataset. <https://universe.roboflow.com/xian-douglas/weapondetection-xx3lz>, apr 2022. visited on 2025-04-03.
- [2] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, and Ross Girshick. Benchmarking detection transfer learning with vision transformers. <https://arxiv.org/abs/2111.11429>, nov 2021.