

RELAZIONE DI PROGETTO DI "VISIONE ARTIFICIALE E
RICONOSCIMENTO" - UNIVERSITÀ DI BOLOGNA,
CAMPUS DI CESENA

Face Mask Detection

Componenti del gruppo:

- Alessandro Magnani (0001026336)
- Andrea Matteucci (0001026901)
- Simone Montanari (0001026332)

Indice

1	Introduzione	3
1.1	Object Detection	3
1.2	Face Mask Detection	4
2	Stato dell'arte	5
3	Progettazione	9
3.1	Dataset	9
3.2	Retina Net	14
3.2.1	Architettura	14
3.2.2	Backbone	14
3.2.3	Modulo RetinaHead	16
3.2.4	Loss Function	16
3.2.5	Addestramento	17
3.2.6	Fase di inferenza	18
3.3	YOLO	20
3.3.1	Architettura	20
3.3.2	Backbone	21
3.3.3	Neck	22
3.3.4	Head	22
3.3.5	Loss Function	22
3.3.6	Addestramento	23
3.3.7	Fase di inferenza	24
3.4	Faster R-CNN	24
3.4.1	Architettura	24
3.4.2	Backbone	25
3.4.3	Region Proposal Network (RPN)	27
3.4.4	ROI Pooling Layer - Region Of Interest Pooling Layer	28
3.4.5	Loss function	29
3.4.6	Addestramento	30
3.4.7	Fase di inferenza	30
3.5	Metriche di valutazione	31
3.5.1	Loss	31

3.5.2	Calcolo delle metriche di valutazione	33
3.5.3	Matrice di confusione	35
4	Analisi delle performance e considerazioni	37
4.1	Premesse	37
4.2	Considerazioni sulle prestazioni	38
5	Conclusioni	48

1 Introduzione

1.1 Object Detection

L'Object Detection è una tecnica utilizzata nell'ambito dell'intelligenza artificiale che si basa sull'individuazione e il riconoscimento di determinati oggetti di interesse all'interno di immagini o video.

In particolare, l'obiettivo dell'Object Detection è quello di individuare oggetti specifici e determinarne la posizione esatta mediante l'uso di bounding box rettangolari. Queste bounding box sono accompagnate da etichette che indicano la classe dell'oggetto (ad esempio, "persona", "automobile", etc.) e da un valore di confidenza che rappresenta il grado di sicurezza della predizione effettuata dalla rete neurale.

L'obiettivo di localizzare e determinare la posizione degli oggetti all'interno di un'immagine rappresenta un problema molto complesso che negli anni ha richiesto l'adozione di diverse metodologie. Tra queste, molte tecniche di Object Detection si basano sull'impiego di reti neurali convoluzionali (CNN). Queste reti sono in grado di estrarre autonomamente le caratteristiche più rilevanti dalle immagini e, mediante un processo di apprendimento, raggiungere l'obiettivo con la maggior precisione e accuratezza possibile.

Negli anni sono state costruite diverse architetture di object detection, che possono essere suddivise in due macro-categorie:

1. **Multi-Stage**
2. **Single-Stage**

La principale differenza tra le due metodologie risiede nel fatto che le prime eseguono l'individuazione delle regioni di interesse e successivamente processano solo quelle, mentre le soluzioni Single-Stage analizzano l'intera immagine in un'unica iterazione, esaminando posizioni prefissate. Questo rende le Single-Stage meno accurate, ma anche applicabili a scenari real-time, dove anche le tempistiche di esecuzione hanno un'importanza maggiore.

1.2 Face Mask Detection

Il progetto realizzato vuole unire i concetti dell'Object Detection all'ambito sanitario. Precisamente, l'obiettivo è quello di rilevare i volti all'interno delle immagini fornite alla rete, e stabilire se questi indossino o meno una mascherina.

Come evidenziato precedentemente esistono tanti approcci, con diverse reti convoluzionali, per portare a termine questo task. Si è quindi deciso di eseguire un confronto attraverso l'uso di queste tecniche:

1. **RetinaNet**
2. **YOLO**
3. **Faster R-CNN**

Successivamente verrà spiegato, più in dettaglio, come queste sono state strutturate.

2 Stato dell'arte

Le reti neurali per il rilevamento degli oggetti sono diventate uno dei pilastri fondamentali nell'ambito della visione artificiale. Queste reti hanno rivoluzionato l'approccio al riconoscimento degli oggetti, offrendo una combinazione di precisione e velocità senza precedenti. Negli ultimi anni, sono state sviluppate diverse architetture di reti neurali per affrontare le sfide del riconoscimento degli oggetti, ottenendo notevoli progressi scientifici e tecnologici in ambito di ricerca.

Un'area di grande rilevanza per le reti neurali per il riconoscimento degli oggetti è la rilevazione delle mascherine e la loro classificazione. Durante la pandemia di COVID-19 la necessità di riconoscere se le persone indossassero o meno una mascherina sul volto è diventata particolarmente critica per garantire la sicurezza pubblica e il rispetto delle norme sanitarie.

Le tre reti prese in esame in questo elaborato, Faster R-CNN, RetinaNet e YOLO, hanno dimostrato di essere estremamente utili in questo contesto.

Nel contesto specifico del riconoscimento delle mascherine e della classificazione, queste reti, come detto precedentemente, hanno dimostrato di essere strumenti efficaci per il monitoraggio e il controllo della pandemia di COVID-19. Tuttavia, il campo delle reti neurali per il rilevamento degli oggetti è in costante evoluzione e ulteriori progressi sono attesi per migliorare ulteriormente l'accuratezza, la velocità e l'efficienza di queste reti.

In questa ottica, il progetto è volto alla comparazione delle tre reti neurali sopra indicate, analizzando qualità e difetti di ognuna di esse.

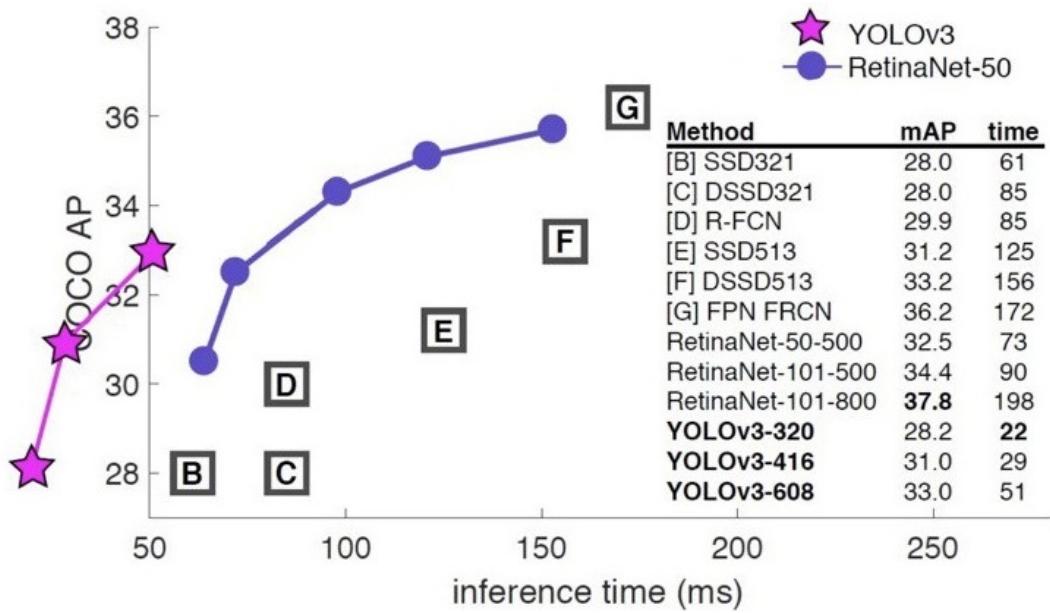


Figura 1: Comparazione dei tempi tra YOLO e RetinaNet.

Le tre reti neurali sono state messe a confronto, addestrandole sullo stesso training set offerto da COCO. Questi dati sono relativi al 2018 e, nonostante all'interno del progetto sono state utilizzate alcune varianti (come ad esempio YOLOv5 invece che YOLOv3), rimangono comunque abbastanza esemplificativi.

	backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
<i>Two-stage methods</i>							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
<i>One-stage methods</i>							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figura 2: Comparazione delle accuratezze ottenute con le varie reti su un dataset COCO nel 2018.

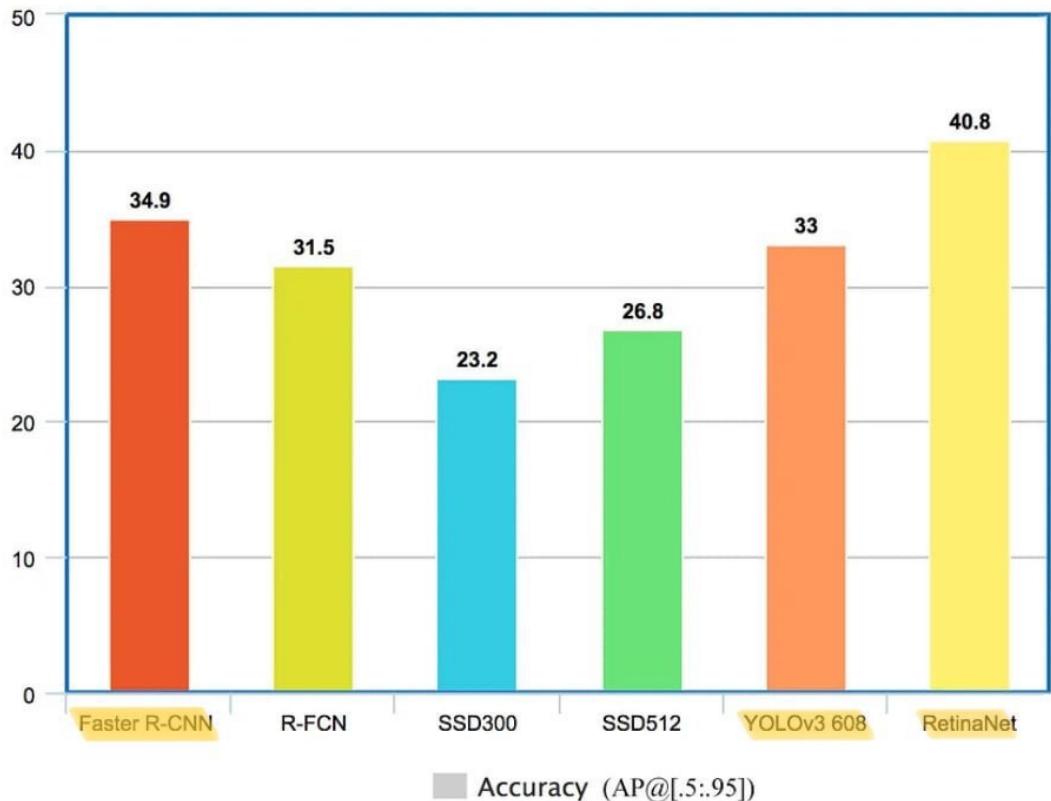


Figura 3: Comparazione delle accuratezze ottenute con le varie reti su un dataset COCO nel 2018.

Da questi valori, le aspettative maggiori per quanto riguarda l'efficacia di classificazione vertono su RetinaNet mentre come ampiamente teorizzato la rete più veloce è YOLO dato il suo utilizzo anche in ambienti real-time.

3 Progettazione

3.1 Dataset

Il dataset che si è scelto di utilizzare per l’addestramento (Face Mask Detection Dataset) si compone di 890 immagini e annotazioni su 3 classi differenti: `with_mask`, `without_mask` e `mask_weared_incorrect`.

Tutte le annotazioni sono in formato XML e contengono al loro interno tutte le informazioni relative all’immagine di cui verranno sfruttate le coordinate delle bounding box e la classe di appartenenza. Di seguito è riportato un esempio di un’annotazione e della relativa struttura.

```
<annotation>
    <folder>images</folder>
    <filename>makssksksss114.png</filename>
    <size>
        <width>400</width>
        <height>225</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>without_mask</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>4</xmin>
            <ymin>83</ymin>
            <xmax>75</xmax>
            <ymax>161</ymax>
        </bndbox>
    </object>
```

Figura 4: Esempio di annotazioni di un’immagine.

Per poter scaricare il dataset direttamente da Kaggle in tutti e tre i Notebook, è stato necessario generare il token univoco fornito dal sito come osservabile in Figura 5.

```
# Caricamento del token Kaggle
files.upload()

# Creazione cartella e copia del token con permessi di lettura e scrittura
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download del dataset
!kaggle datasets download -d andrewmvd/face-mask-detection

# Unzip del dataset
!unzip face-mask-detection.zip
```

Figura 5: Importazione del dataset da Kaggle.

Si è scelto di rinominare le immagini di modo da renderne più semplice e agevole l’uso. Nello specifico, i nomi originali delle immagini forniti nel dataset erano poco chiari e intuitivi (ad esempio l’immagine `makssksksksss1.png` viene rinominata come `00000001.png`).

```
def rename_dataset_names(directory):
    for filename in os.listdir(directory):
        replaced = filename.replace("makssksksksss", "")
        name_length = len(replaced)
        if name_length < 12:
            new_name = "0" * (12 - name_length) + replaced
            os.rename(os.path.join(directory, filename), os.path.join(directory, new_name))
```

Figura 6: Funzione per rinominare i file delle immagini.

Un’ulteriore scelta progettuale attuata è stata quella di eliminare le annotazioni di classe `mask_weared_incorrect` (Figura 7) dal momento che erano in numero insufficiente per poter poi addestrare la rete ed ottenere prestazioni ottimali. Difatti, mantenendo quella classe, si erano riscontrati diversi problemi come:

- **Difficoltà nell’apprendimento** delle caratteristiche distintive di quella classe;

- **Overfitting**, dato che il modello era incline ad adattarsi troppo bene ai pochi esempi di addestramento che si trovava, memorizzandoli invece di imparare pattern generali;
- **Disuguaglianza di rappresentanza**, essendo una classe con poche istanze rispetto alle altre, il modello aveva difficoltà nel dare lo stesso peso e importanza a tutte le classi durante l'addestramento. Ciò portava ad una discriminazione nei confronti della classe sottorappresentata e risultati non ottimali.

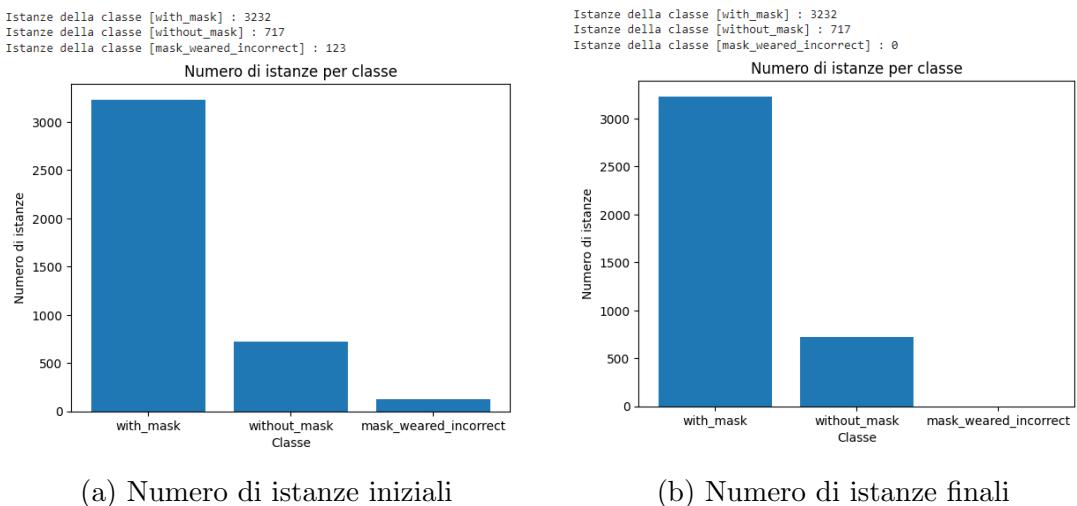


Figura 7: Numero di istanze per classe nel Dataset.

In particolare, nel Notebook riguardante YOLOv5 le annotazioni vengono normalizzate nel range [0,1] e convertite in formato TXT in quanto questa rete richiede espressamente questo formato.

Infine, il dataset viene suddiviso nei 3 set di training, validation e test con le seguenti percentuali: **56%** per il training set, **24%** per il validation set e **20%** per il test set. Per ottenere tale risultato viene utilizzata la funzione **train_test_split** di **scikit-learn**.

Nonostante avere un dataset di dimensione ridotta (solamente 890 immagini) consiglierebbe di convogliare tutti gli sforzi nella fase di addestramento passando al

training set più istanze possibili, si è optato per una suddivisione più equa del dataset.

Infatti dopo aver condotto varie prove con diverse combinazioni di percentuali, si è deciso di dare maggiore importanza alla fase di validation e test rispetto alla fase di addestramento. Questa scelta è stata fatta al fine di avere un ampio set di esempi per le predizioni, il che si traduce in **valori più realistici e significativi per le metriche di valutazione** come la *medium Average Precision (mAP)* e l'*F1 Score*.

Passando più istanze al training set e meno al validation e al test set, si è notato che queste **metriche possono variare significativamente** da un addestramento all'altro, a causa del *peso considerevole degli errori*. Ad esempio, durante le diverse iterazioni di addestramento, è stato osservato come un singolo errore possa portare a una diminuzione drastica del valore di mAP o F1-Score, quantificabile addirittura a 5 punti percentuali.

Pertanto, si è deciso di dedicare più risorse alla fase di validation e test, al fine di ottenere un insieme più rappresentativo di esempi da valutare. Questa scelta strategica permette di avere risultati più **stabili e affidabili**, poiché si è in grado di valutare le prestazioni del modello su un insieme più ampio e diversificato di esempi di test.

È importante sottolineare come l'obiettivo principale dell'analisi sia confrontare le prestazioni dei diversi modelli tra loro. Pertanto, è essenziale sottolineare che le percentuali utilizzate per valutare i risultati sono uniformi per tutti i modelli considerati. Questo approccio uniforme consente di **effettuare confronti significativi e oggettivi** tra i modelli. Tuttavia, va tenuto presente che, a causa delle diverse dinamiche di apprendimento e dei tempi di crescita variabili dei singoli modelli, l'uguaglianza delle percentuali e il numero limitato di immagini **potrebbe favorire alcuni modelli rispetto ad altri**. Questa variazione nelle prestazioni può essere attribuita alle *caratteristiche uniche di ciascun modello* e ai *differenti ritmi di apprendimento* che essi presentano.

Il dataset appare quindi suddiviso come segue:

```

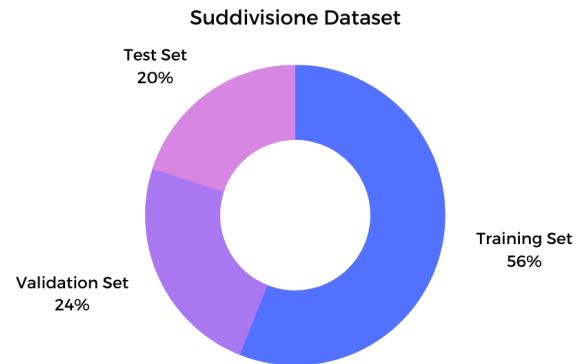
Cardinalità immagini dataset      : 832
Istanze della classe [with_mask] : 3232
Istanze della classe [without_mask] : 717

Cardinalità immagini training set : 465
Istanze della classe [with_mask] : 1774
Istanze della classe [without_mask] : 358

Cardinalità immagini validation set : 200
Istanze della classe [with_mask] : 744
Istanze della classe [without_mask] : 167

Cardinalità immagini test set     : 167
Istanze della classe [with_mask] : 714
Istanze della classe [without_mask] : 192

```



(a) Suddivisione di immagini e istanze (b) Suddivisione delle immagini in percentuale

Figura 8: Suddivisione del dataset in training, validation e test set.

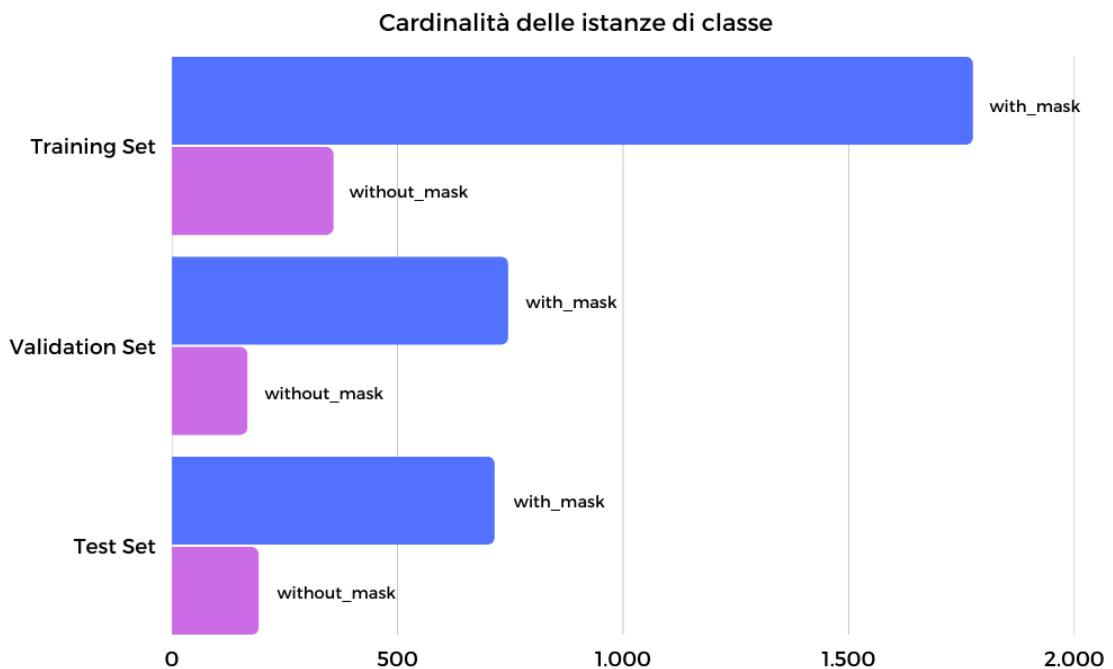


Figura 9: Suddivisione delle istanze delle classi per dataset.

3.2 Retina Net

Nel presente capitolo, verrà affrontata la progettazione del modello RetinaNet, una rete neurale profonda utilizzata per il rilevamento di oggetti in immagini. Ne verrà inoltre discussa l'architettura, il funzionamento e i principali componenti che la costituiscono.

3.2.1 Architettura

RetinaNet è un modello di rilevamento di oggetti che combina una rete convoluzionale per l'estrazione delle caratteristiche e un modulo di rilevamento a livello di rete chiamato RetinaHead.

Si può dire infatti che l'architettura può essere suddivisa in due componenti principali: la **backbone** e il modulo **RetinaHead**. La backbone è responsabile dell'estrazione delle caratteristiche dall'immagine di input, mentre il modulo RetinaHead è responsabile della generazione delle predizioni di classe (*classification*) e localizzazione per gli oggetti presenti nell'immagine (*detection*).

Inoltre RetinaNet è basata su una rete neurale convoluzionale che implementa la **Feature Pyramid Network (FPN)** per la *rilevazione multi-scala degli oggetti*. Difatti l'FPN riceve le feature map dalla backbone e *genera una piramide di mappe delle caratteristiche* con informazioni di diverse scale, consentendo a RetinaNet di **rilevare oggetti di varie dimensioni** nell'immagine.

3.2.2 Backbone

La backbone di RetinaNet utilizzata in questo progetto è una rete neurale convoluzionale pre-addestrata della **ResNet Family**, fornita dalla *versione ufficiale di PyTorch*. All'interno dell'elaborato si è scelto di usare la versione di default, ovvero la **ResNet-50-FPN**. Questo componente svolge un ruolo fondamentale nell'estrazione delle caratteristiche dell'immagine. La scelta di utilizzare una backbone pre-addestrata (Figura 10) ha permesso di beneficiare delle conoscenze apprese su un ampio set di dati durante l'addestramento del modello.

```
# retina = torchvision.models.detection.retinanet_resnet50_fpn(num_classes = 3, pretrained=False)
retina = torchvision.models.detection.retinanet_resnet50_fpn(num_classes = 3, pretrained=True)
```

Figura 10: Caricamento del modello RetinaNet preaddestrato.

Per adattare il modello alle esigenze, è stato **modificato il numero di classi**. Questo ha permesso di utilizzare la backbone pre-addestrata come punto di partenza e di adattare facilmente il modello alle specifiche senza doverne costruire uno nuovo da zero. Le altre parti della struttura del modello, come il modulo *RetinaHead* e la funzione di loss, sono rimaste invariate.

All'interno del progetto, i pesi pre-addestrati ottenuti dal **pre-addestramento** effettuato su **COCO** (dataset di immagini generali di grandi dimensioni contenente decine di categorie di oggetti), sono stati usati solamente per l'esecuzione del primo addestramento durante la fase di **fine-tuning** in quanto successivamente i *nuovi pesi ottenuti venivano salvati e ricaricati all'interno del modello*, di modo da poterli sfruttare per addestramenti futuri (i valori appresi nella fase di pre-addestramento vengono adattati per lo specifico compito di rilevazione degli oggetti). Così facendo, il modello riesce a focalizzare l'apprendimento sui dettagli specifici dei dati relativi alle mascherine (Figura 11).

```
# Pull del file coi pesi più recenti
directory = "/content/drive/MyDrive/visione-artistificiale/"

# Ottieni l'elenco dei file nella directory
files = os.listdir(directory)

# Filtra solo i file che iniziano per "retina" e hanno l'estensione ".pt"
filtered_files = [f for f in files if re.match(r"retina.*\.pt", f)]

# Ordina i file in base al numero e, in caso di numeri uguali, al secondo numero
sorted_files = sorted(filtered_files, key=lambda f: tuple(map(int, re.findall(r'\d+', f))), reverse=True)

# Se ci sono file nella lista ordinata, seleziona il primo file
if sorted_files:
    first_file = sorted_files[0]
    print(directory + first_file)

    # Carica l'ultima versione del modello salvato
    retina = torchvision.models.detection.retinanet_resnet50_fpn(pretrained=False, num_classes=3)
    retina.load_state_dict(torch.load(directory + first_file))
else:
    print("Nessun file trovato.")
```

Figura 11: Caricamento del modello addestrato precedentemente.

3.2.3 Modulo RetinaHead

Il modulo RetinaHead è responsabile della generazione delle predizioni di classe e localizzazione per gli oggetti nell’immagine. È composto da due sotto-moduli principali: un sub-network di **classificazione** e un sub-network di **regressione**.

Il sub-network di classificazione è un classificatore che **assegna una probabilità a ciascuna classe di oggetti** per ciascuna regione proposta, utilizzando le feature map della piramide di mappe delle caratteristiche generate dall’FPN per effettuare le predizioni di classe.

Il sub-network di regressione è responsabile della **generazione delle coordinate delle bounding box** per ciascuna regione proposta. Utilizza le stesse feature map della piramide di mappe delle caratteristiche per effettuare le predizioni di localizzazione.

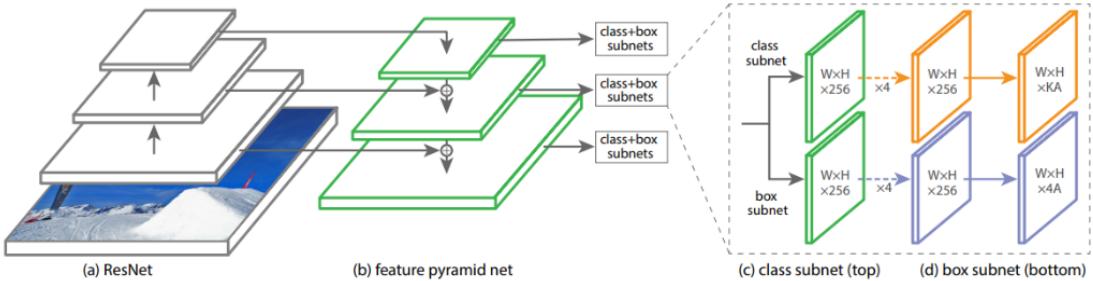


Figura 12: Rappresentazione dell’architettura del modello RetinaNet.

3.2.4 Loss Function

RetinaNet utilizza una specifica funzione di loss chiamata **Focal Loss** per addestrare il modello. La Focal Loss è una variante della **binary cross-entropy** ed è progettata per affrontare il problema dell’elevato **sbilanciamento di classi** nel rilevamento degli oggetti. In particolare, durante la fase di addestramento, gli esempi più semplici sono molto più numerosi rispetto agli esempi più complessi e questo sbilanciamento, se ogni esempio è considerato con lo stesso peso, può portare la rete a concentrarsi solamente su quelli semplici, *ignorando la minoranza di*

esempi complessi e mantenendo così valori comunque bassi di loss. Questo però non permetterà mai alla rete di imparare efficacemente a rilevare gli oggetti in quanto tenderà sempre più ad ignorare quelli complessi.

Gli esempi più intuitivi, ovvero le rilevazioni con alta probabilità, anche se producono valori di loss ridotti, possono quindi sovraccaricare collettivamente il modello. La Focal Loss affronta questa problematica introducendo un termine di riduzione di peso che dà **più enfasi agli esempi di addestramento più complessi**, riducendo l'impatto degli esempi di addestramento semplici e **aumentando l'importanza di correggere gli esempi classificati erroneamente**.

In sintesi, l'uso della Focal Loss aiuta RetinaNet a migliorare la capacità di discriminazione tra gli oggetti e **ridurre il numero di falsi positivi** durante il rilevamento.

3.2.5 Addestramento

Dopo aver eseguito le operazioni di data pre-processing, la definizione della struttura di cartelle necessaria e l'importazione del modello pre-addestrato si è pronti per la fase di addestramento.

In questa fase viene definito il numero di epoche e le variabili contenenti le loss e viene impostato il modello in fase di addestramento. Successivamente vengono ciclate le epoche sia sul training set che sul validation set, come riportato nello pseudocodice di Figura 13.

```
[ ] # Definizione del numero di epocha e delle variabili contententi le loss

# Impostazione del modello in fase di addestramento

# Ciclo di addestramento sulle epocha:

# Ciclo sul training set
# Predizione del modello
# Calcolo della loss
# Backpropagation e aggiornamento dei pesi

# Ciclo sul validation set
# Predizione del modello
# Calcolo della loss
```

Figura 13: Pseudocodice dell’addestramento sul modello RetinaNet.

3.2.6 Fase di inferenza

In questa sezione viene illustrato il processo di inferenza del modello RetinaNet. Durante questa fase, il modello applica le conoscenze apprese durante l’addestramento al fine di individuare e classificare gli oggetti di interesse nelle immagini di test.

Le immagini di test vengono passate ad una specifica funzione, osservabile in Figura 14, dove vengono processate per restituire i valori relativi alle **coordinate delle bounding box**, alla **classe di appartenenza** e alla **confidenza con la quale è stata eseguita la classificazione**. Questo processo di predizione viene attuato in *modalità di inferenza del modello*.

```

def make_prediction(model, img, thresh):
    # Imposta il modello in fase di inferenza
    model.eval()

    # Predizione
    preds = model(img)

    for id in range(len(preds)):
        idx_list = []

        for idx, score in enumerate(preds[id]['scores']):
            # Seleziona gli indici delle predizioni maggiori della soglia passata in input
            if score > thresh:
                idx_list.append(idx)

        # Salva coordinate, etichetta e score delle predizioni selezionate
        preds[id]['boxes'] = preds[id]['boxes'][idx_list]
        preds[id]['labels'] = preds[id]['labels'][idx_list]
        preds[id]['scores'] = preds[id]['scores'][idx_list]

    return preds

```

Figura 14: Funzione `make_prediction` utilizzata per effettuare le predizioni.

Dopo aver eseguito le predizioni sulle immagini di test, viene effettuato un confronto visivo tra le predizioni del modello e le annotazioni originali. Questo confronto viene effettuato sulle stesse immagini di test al fine di valutare l'accuratezza delle predizioni.

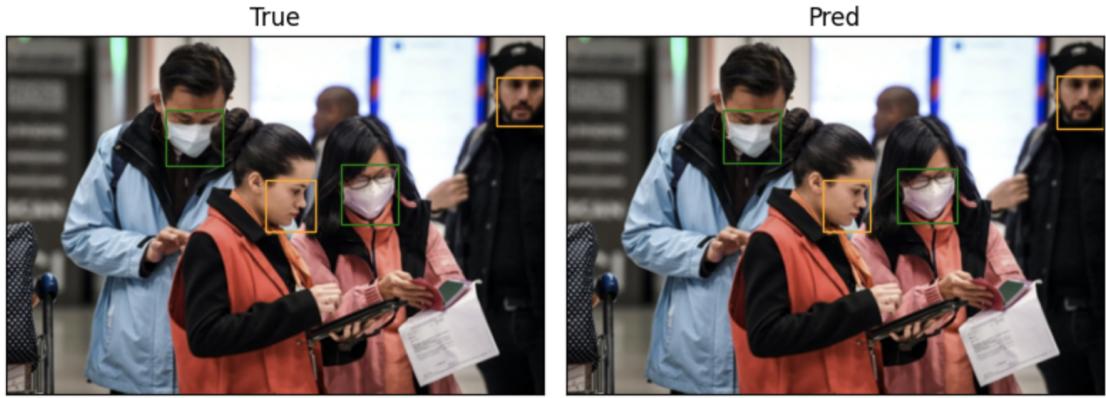


Figura 15: Esempio di confronto tra annotazioni originali e predette.

Attraverso questo confronto è possibile identificare eventuali *discrepanze tra le predizioni del modello e le annotazioni originali*, permettendo così una valutazione visiva dell'efficacia del modello.

3.3 YOLO

In questa sezione si affronta l'argomento relativo alla progettazione di YOLO, una rete neurale profonda ampiamente utilizzata per il rilevamento di oggetti. Saranno esaminati l'architettura, il funzionamento e gli elementi fondamentali che costituiscono il modello YOLO.

3.3.1 Architettura

L'evoluzione di YOLO (You Only Look Once) ha segnato un importante traguardo nel campo del riconoscimento di oggetti real-time, offrendo costanti miglioramenti nel corso del tempo e conquistando una posizione di rilievo nella comunità scientifica. Le varie versioni di YOLO hanno dimostrato un costante progresso, portando significativi miglioramenti in termini di precisione, velocità di esecuzione e capacità nel rilevare oggetti di diverse dimensioni e classi. Questi sviluppi hanno reso YOLO una delle soluzioni più apprezzate per il riconoscimento di oggetti in tempo reale, suscitando grande interesse e applicazioni pratiche in diversi settori.

YOLO è un modello di rilevamento di oggetti *Single-Stage*, ciò significa che processa le immagini in un'unica iterazione, senza la necessità di fasi complesse come la ricerca di regioni di interesse o la generazione di proposte di regioni. Questo approccio consente a YOLO di raggiungere elevate velocità di esecuzione in tempo reale, rendendolo particolarmente adatto per applicazioni che richiedono una rapida elaborazione delle immagini, come la sorveglianza video o la guida autonoma.

Tuttavia, è importante notare che questo approccio può presentare alcune limitazioni come il fatto che potrebbe essere meno accurato nel rilevare oggetti di piccole dimensioni o in presenza di sovrapposizioni tra oggetti.

La variante adottata nell'elaborato è la *YOLOv5*. Nello specifico, tra le quattro versioni disponibili, l'implementazione utilizzata è la *YOLOv5s*. Il modello è composto da tre parti fondamentali:

- Model Backbone
- Model Neck

- Model Head

Le immagini vengono inizialmente processate dalla backbone per estrarre le caratteristiche più rilevanti. Questo processo produce feature map di diverse dimensioni che vengono poi inviate al neck. Questo elabora queste feature map per generare *mappe di caratteristiche* separate, ognuna dedicata al rilevamento di oggetti di piccole, medie e grandi dimensioni. Successivamente, le tre mappe vengono passate alla head, che, utilizzando le *anchor-boxes* pre-definite insieme alle mappe, si occupa di individuare gli oggetti nelle immagini producendo in output vettori multi-dimensionalni contenenti tutte le informazioni sulle predizioni.

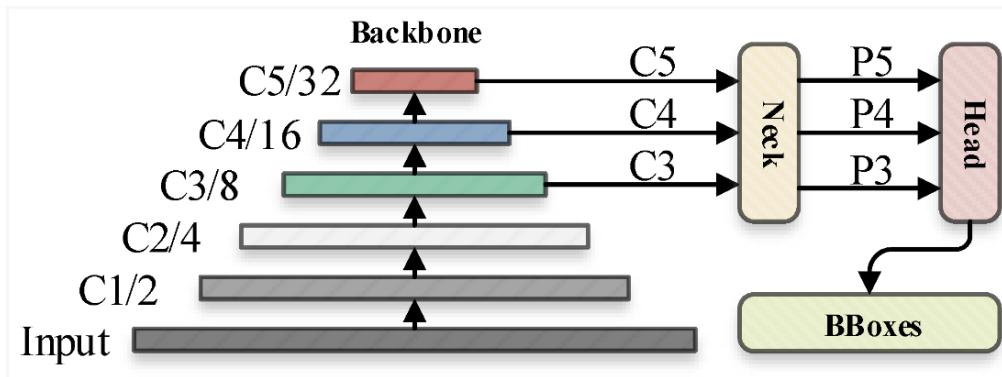


Figura 16: Rappresentazione semplificata dell’architettura di YOLOv5.

3.3.2 Backbone

La backbone utilizzata in YOLO è la *CSP-Darknet53*, una rete neurale convoluzionale con il compito di estrarre le feature più rilevanti dalle immagini processate in input. Questa CNN è una versione modificata della rete Darknet53 a cui viene aggiunto il modulo CSP.

Nello specifico, la struttura principale è composta da diverse tipologie di moduli:

- **CBS**: composto da un layer di convoluzione, un layer di *BatchNormalization* e la funzione di attivazione *SiLU* (*Sigmoid Linear Unit*);
- **C3**: utilizzato per l'estrazione di feature;
- **SPPF**: utilizzato per migliorare la capacità descrittiva delle feature estratte.

3.3.3 Neck

Questa parte dell’architettura riceve in input le feature map estratte dal backbone e si occupa di trasformarle per poter rilevare oggetti di diverse dimensioni. Nello specifico, sono utilizzate tecniche di FPN (Feature Pyramid Network) e PAN (Path Aggregation Network). L’idea alla base è quella di eseguire fasi di up-sampling delle feature map (generate da più operazioni di down-sampling applicate dal backbone) per poter rilevare oggetti di diverse scale.

3.3.4 Head

La Head utilizzata in YOLOv5 è l’unica parte rimasta invariata rispetto alle precedenti versioni YOLOv3 e v4. Questo modulo è responsabile del rilevamento finale degli oggetti. Le *anchor-boxes* pre-definite vengono applicate sulle feature map estratte per predire le bounding box. L’output risultante è un vettore multi-dimensionale contenente le coordinate delle bounding box predette, le classi associate agli oggetti rilevati e le relative confidenze.

3.3.5 Loss Function

La loss della rete YOLOv5, chiamata *YOLO loss* è composta dalla combinazione di tre loss:

- **Box Loss** (regressione delle bounding box): si occupa di valutare la discrepanza tra le coordinate delle bounding box predette e tra quelle del ground truth. Per calcolare questa loss viene utilizzato l’MSE (Mean Squared Error). L’obiettivo è quello di minimizzare la distanza tra predizioni e ground truth;
- **Objectness Loss** (confidenza delle rilevazioni): si occupa di valutare la confidenza sulla presenza di oggetti nelle bounding box. L’obiettivo è far sì che il modello predica correttamente se un oggetto è presente o meno nelle bounding box;
- **Classification Loss** (predizioni delle classi): si occupa di valutare la correttezza delle previsioni sulle classi degli oggetti presenti nelle bounding box.

Per calcolare questa loss viene utilizzata la *Binary Cross-Entropy*. L'obiettivo è far sì che il modello predica correttamente le classi degli oggetti presenti nelle bounding box.

3.3.6 Addestramento

Per poter effettuare un'addestramento del modello YOLOv5 è necessario creare un apposito file di configurazione con estensione YAML specificando il percorso della directory principale, il percorso della directory contenente le immagini di training e di validation, il numero e i nomi delle classi così come in Figura 17.

```
path: ../datasets # dataset root dir
train: train/images # train images
val: valid/images # val images

# Classes
nc: 2 # number of classes
names: ['with_mask', 'without_mask'] # class names
```

Figura 17: Contenuto del file di configurazione.

Dopo aver predisposto il file di configurazione per iniziare l'addestramento occorre eseguire il file `train.py` specificando i seguenti parametri:

- La dimensione a cui verranno ridimensionate le immagini durante l'addestramento;
- Il numero di epoch;
- Il percorso del file di configurazione (`mask_config.yaml`);
- Il percorso del file contenente i pesi di partenza del modello;
- Il numero di workers da utilizzare.

```
!python /content/yolo_utils/train.py --img 640  
--batch 15  
--epochs 200  
--data /content/yolo_utils/mask_config.yaml  
--weights /content/yolo_utils/yolov5s.pt  
--workers 0
```

Figura 18: Addestramento del modello YOLO.

3.3.7 Fase di inferenza

Per poter utilizzare il modello in fase di predizione è necessario eseguire il file `detect.py` specificando i seguenti parametri:

- Il percorso della cartella contenente le immagini di test;
- Il percorso del file contenente i pesi del modello;
- Il valore di confidenza da utilizzare;
- Lo spessore delle bounding box.

```
!python /content/yolo_utils/detect.py --source /content/datasets/test/images  
--weights /content/drive/MyDrive/exp/weights/best.pt  
--conf 0.2  
--line-thickness 2
```

Figura 19: Codice di inferenza del modello.

Similmente a quanto fatto per gli altri modelli, anche per il modello YOLOv5 è stato eseguito un confronto visivo tra le immagini predette e le immagini di test.

3.4 Faster R-CNN

3.4.1 Architettura

Faster R-CNN è un altro modello popolare utilizzato per il rilevamento di oggetti in immagini. Si tratta di un modello multi-stage ed è una versione migliorata del modello **R-CNN (Region-based Convolutional Neural Network)** che presenta diverse caratteristiche che ne aumentano l'efficienza e le prestazioni.

L’architettura di Faster R-CNN si compone di due parti principali: un **modulo di estrazione delle caratteristiche** basato su una rete neurale convoluzionale e un **modulo di proposta delle regioni** basato su una rete neurale completamente connessa (*fully connected*).

Faster R-CNN si distingue per la sua capacità di combinare l’estrazione delle caratteristiche, la generazione delle proposte di regioni e la classificazione degli oggetti in un *unico flusso di lavoro end-to-end*. Ciò consente di ottenere una maggiore efficienza computazionale rispetto a modelli precedenti come R-CNN.

Così come per RetinaNet, anche la Faster R-CNN utilizza ResNet come backbone, la quale sfrutta un approccio **Feature Pyramid Network (FPN)** per l’estrazione delle caratteristiche multi-scala dalle immagini.

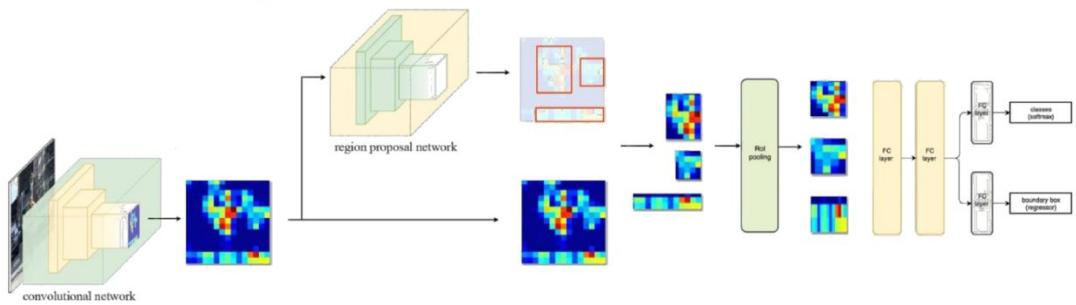


Figura 20: Rappresentazione dell’architettura del modello Faster R-CNN.

3.4.2 Backbone

Il modulo di estrazione delle caratteristiche svolge il compito di analizzare l’intera immagine e di estrarre le informazioni rilevanti per il riconoscimento degli oggetti. Questo viene realizzato attraverso l’utilizzo di una rete neurale convoluzionale pre-addestrata che è in grado di catturare le caratteristiche di alto livello dell’immagine.

Più nello specifico, la backbone di Faster R-CNN utilizzata in questo progetto, è una rete neurale convoluzionale pre-addestrata della **ResNet Family**, fornita dalla *versione ufficiale di PyTorch*. All’interno dell’elaborato si è scelto di usare

la versione di default, ovvero la ResNet-50-FPN.

Per adattare il modello alle esigenze, è stato modificato il numero di classi passando il valore corretto alla backbone fornita da PyTorch. Questo ha permesso di utilizzare la backbone pre-addestrata come punto di partenza e di adattare facilmente il modello alle specifiche senza doverne costruire uno nuovo da zero.

Come di default per la Faster R-CNN, i pesi pre-addestrati sono stati ottenuti dal *pre-addestramento* su **COCO** (Figura 21). Per quanto riguarda la fase di *finetuning*, i nuovi pesi ottenuti dal primo addestramento generale vengono *salvati* e *ricaricati all'interno del modello*, di modo da poterli sfruttare per addestramenti futuri (Figura 22). Così facendo, il modello riesce a focalizzare l'apprendimento sui dettagli specifici dei dati relativi alle mascherine.

```
# Importazione modello
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

# Sostituzione del classificatore
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

Figura 21: Caricamento del modello Faster R-CNN preaddestrato.

```
# Carica il modello salvato
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False, num_classes=3)
model.load_state_dict(torch.load(model_path))
```

Figura 22: Caricamento del modello addestrato precedentemente.

Il backbone, quindi, acquisisce progressivamente informazioni dall'immagine attraverso una serie di layer convoluzionali, riducendo progressivamente le dimensioni spaziali delle feature map ma aumentando la loro profondità. Queste feature map estratte sono poi utilizzate dalla Region Proposal Network (RPN) e verranno analizzate più nel dettaglio.

3.4.3 Region Proposal Network (RPN)

La funzione principale del Region Proposal Network è generare proposte di regioni che potrebbero contenere oggetti nell'immagine di input.

L'RPN utilizza le feature map estratte dal backbone per **individuare le possibili regioni di interesse**. Queste feature map rappresentano le informazioni di alto livello estratte dall'immagine e contengono caratteristiche semantiche che possono essere associate agli oggetti. Simulando il funzionamento delle sliding windows, vengono generate un insieme di posizioni potenziali in cui potrebbero essere presenti oggetti.

Ad ogni posizione l'RPN produce le **ancore**, un *insieme di proposte di regioni candidate* (Figura 23). Le ancore sono rettangoli di dimensioni e forme predefinite che vengono collocati su diverse posizioni all'interno delle feature map. Queste regioni proposte fungono da ipotesi per la presenza di oggetti e coprono una varietà di scale e aspetti che possono essere associati a mascherine di diverse forme e dimensioni.

Più nel dettaglio vengono applicati due tipi di convoluzione:

- **classificazione**: l'RPN determina se ogni anchor box contiene o meno un oggetto di interesse.
- **regression**: l'RPN predice la posizione e le dimensioni delle bounding box relative alle anchor boxes.

Le proposte di regioni generate dall'RPN vengono successivamente utilizzate dal Region Of Interest Pooling Layer. In questo modo, l'RPN agisce come una sorta di filtro per ridurre il numero di regioni proposte da analizzare.

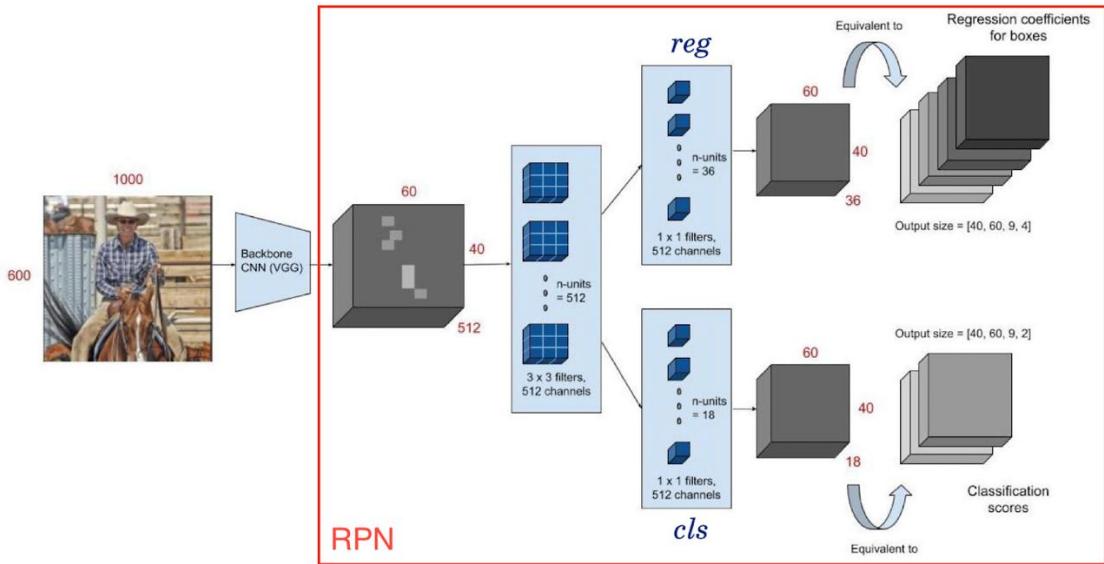


Figura 23: Rappresentazione del funzionamento del Region Proposal Network (RPN).

3.4.4 ROI Pooling Layer - Region Of Interest Pooling Layer

Il Region of Interest (RoI) pooling layer è una componente cruciale nell’architettura di Faster R-CNN, e, più nello specifico, è stato introdotto precedentemente nella FAST R-CNN. Questo strato svolge un ruolo importante nel processare le proposte di regioni generate dall’RPN e nell’ottenere **rappresentazioni di dimensioni fisse** per l’input della rete fully connected successiva.

Dopo che l’RPN ha generato le proposte di regioni, queste possono avere dimensioni e proporzioni diverse. Dato che il successivo livello fully connected **richiede feature map di dimensioni prefissate**, il RoI pooling layer si occupa di standardizzarle di modo da adattarle a un formato fisso.

Il RoI pooling layer divide ciascuna regione proposta in una griglia regolare di sottoregioni sovrapposte. Per ogni sottoregione viene effettuato un processo simile al max pooling per estrarne una rappresentazione fissa.

L'output del RoI pooling layer è quindi costituito da un insieme di sottoregioni con dimensioni fisse e rappresentazioni spaziali ridimensionate. Come esaminato precedentemente, la caratteristica che queste regioni abbiano una dimensione pre-fissata è un requisito fondamentale per i layer fully connected successivi che andranno ad eseguire la classificazione dell'oggetto e la regressione delle bounding box.

3.4.5 Loss function

La loss function di Faster R-CNN è detta **Multy-Task Loss**. Questo nome deriva dal fatto che vengono combinate la *loss di regressione* con la *loss di classificazione*, al fine di tarare al meglio i pesi della rete.

La **loss di classificazione** viene utilizzata per **addestrare il modello a classificare correttamente** gli oggetti nelle immagini. Per ogni regione di interesse generata dall'algoritmo di region proposal, il modello assegna un'etichetta di classe (ad esempio, "cane", "auto", "persona") e calcola la loss di classificazione utilizzando la funzione di *cross-entropy* tra le etichette previste e le etichette di ground truth.

La **loss di regressione**, invece, viene utilizzata per **addestrare il modello a predire con precisione le coordinate delle bounding box** che circondano ciascun oggetto. Per ogni region proposal, il modello calcola la loss di regressione utilizzando una metrica come la *L1 loss* (smoothed), restituendo una tupla di 4 valori contenente le coordinate della bounding box, la larghezza e l'altezza.

Complessivamente, la **Multi-Task Loss** combina la loss di classificazione e la loss di regressione della bounding box in un'unica loss complessiva. Durante l'addestramento, il modello **cerca di minimizzare questa loss aggregata** utilizzando tecniche di ottimizzazione come la discesa del gradiente, che aggiorna i pesi del modello in modo da ridurre la discrepanza tra le previsioni del modello e i valori di ground truth per la classificazione e la regressione delle bounding box.

3.4.6 Addestramento

Anche in questo caso, come avviene per il modello Retina, dopo aver eseguito la parte di data pre-processing, impostata la struttura delle cartelle e importato il modello pre-addestrato, si è pronti per la fase di addestramento. L'addestramento all'interno del Notebook, relativamente a Faster R-CNN riprende lo pseudocodice di RetinaNET (Figura 13).

3.4.7 Fase di inferenza

Per effettuare le predizioni con il modello di Faster R-CNN è stata utilizzata su tutte le immagini di test la funzione `single_img_predict`, osservabile in Figura 24.

```
def single_img_predict(img, nm_thrs=0.3, score_thrs=0.9):
    # Conversione a tensore e trasferimento sul dispositivo di esecuzione
    test_img = transforms.ToTensor()(img).to(device)

    # Imposta il modello in modalità di inferenza
    model.eval()

    # Predizione
    with torch.no_grad():
        predictions = model(test_img.unsqueeze(0))

    # Permutazione dimensioni dell'immagine
    test_img = test_img.permute(1, 2, 0).cpu().numpy()

    # Soppressione dei non-massimi
    keep_boxes = torchvision.ops.nms(predictions[0]['boxes'].cpu(), predictions[0]['scores'].cpu(), nm_thrs)

    # Applicazione della soglia
    score_filter = predictions[0]['scores'].cpu().numpy()[keep_boxes] > score_thrs

    # Risultati finali
    test_boxes = predictions[0]['boxes'].cpu().numpy()[keep_boxes][score_filter]
    test_labels = predictions[0]['labels'].cpu().numpy()[keep_boxes][score_filter]
    test_scores = predictions[0]['scores'].cpu().numpy()[keep_boxes][score_filter]

    return test_img, test_boxes, test_labels, test_scores
```

Figura 24: Funzione `single_img_predict` per effettuare le predizioni su una singola immagine.

Successivamente, come per gli altri modelli, è stato effettuato il confronto visivo tra le immagini predette e quelle di test.

3.5 Metriche di valutazione

In questa sezione vengono illustrate le metriche di valutazione calcolate per i tre modelli. Le metriche forniscono indicazioni chiare e quantificabili sull'accuratezza e l'efficacia. Ciò permette di trarre conclusioni sulle capacità dei modelli e di confrontarle in modo accurato.

Al termine del processo di addestramento del modello YOLOv5, viene generata automaticamente una cartella contenente i risultati dell'addestramento. Per quanto riguarda i modelli RetinaNet e Faster R-CNN, invece, le metriche di valutazione vengono calcolate manualmente sia durante (loss sul training e validation set) che dopo la fase di addestramento (precision, recall, F1-score, ecc.).

Di seguito, verrà presentato il codice relativo ai modelli RetinaNet e Faster R-CNN.

3.5.1 Loss

Le loss su training e validation set vengono calcolate manualmente durante la fase di addestramento.

Nello specifico, la loss relativa al training set viene ottenuta mediante il seguente procedimento:

- L'immagine viene passata al modello;
- L'ottimizzatore viene azzerato per prepararsi all'aggiornamento dei pesi;
- Viene eseguita la retropropagazionne dell'errore;
- I pesi del modello vengono aggiornati;
- La loss viene sommata per tenere traccia dell'andamento nelle varie epoche.

Questa sequenza di operazioni è stata implementata all'interno del codice come da Figura 25.

```

# Predizione del modello
loss_dict = model(images, targets)

# Somma dei valori di loss per ogni predizione
losses = sum(loss for loss in loss_dict.values())

# Azzeramento dell'ottimizzatore
optimizer.zero_grad()

# Backpropagation
losses.backward()

# Aggiornamento dei pesi
optimizer.step()

# Aggiunta delle loss alla lista
train_loss += losses.item() # Per ottenere il valore scalare

```

Figura 25: Calcolo della loss sul training set.

Per il calcolo della loss sul validation set, viene utilizzato il comando `with torch.no_grad()` per disattivare il calcolo e l'accumulo del gradiente. Successivamente la loss viene calcolata in modo simile al processo precedente, sommando i valori per ogni predizione ottenuta dal modello sul validation set.

```

# Predizione del modello
val_loss_dict = model(val_images, val_targets)

# Somma dei valori di loss per ogni predizione
val_losses = sum(val_loss for val_loss in val_loss_dict.values())

# Aggiunta delle loss alla lista
val_loss += val_losses.item()

```

Figura 26: Calcolo della loss sul validation set.

Dopo aver calcolato le due loss sul training set e sul validation set, è consueto visualizzarne l'andamento nel corso delle epoche per avere una comprensione visuale della convergenza del modello.

Monitorare l'evoluzione delle loss nel tempo consente di identificare eventuali problemi durante la fase di addestramento. Ad esempio, se la loss sul training set diminuisce ma quella sul validation set aumenta, potrebbe essere un segnale di *overfitting*.

```
# Grafico delle loss su train e validation set
plt.plot(train_loss_list, label='Train Loss')
plt.plot(val_loss_list, label='Val Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()
```

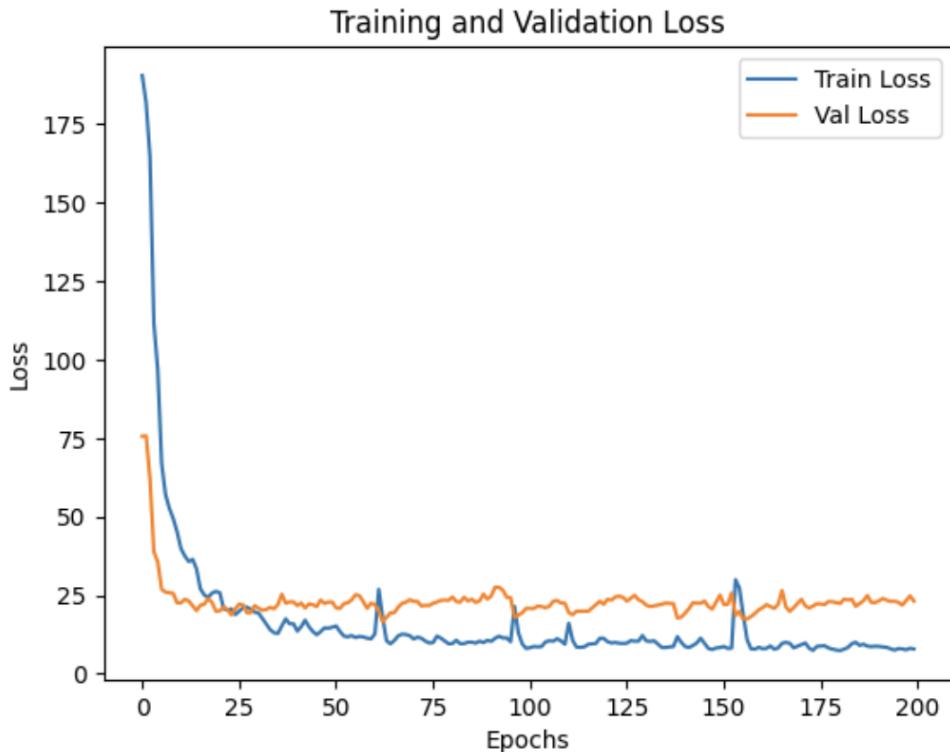


Figura 27: Visualizzazione delle due loss in funzione delle epoche di addestramento.

3.5.2 Calcolo delle metriche di valutazione

Successivamente alla fase di addestramento, sono stati calcolati diversi indici per valutare l'accuratezza e l'efficacia dei modelli. Queste metriche forniscono informazioni dettagliate sulle performance nella rilevazione e classificazione degli oggetti

di interesse.

Nello specifico, viene utilizzata la funzione `get_batch_statistics` che riceve in input le predizioni del modello, le annotazioni originali e una soglia di IoU (Intersection over Union) e restituisce un elenco di metriche di interesse.

```
# Ciclo sulle immagini di test

# Estrazione delle annotazioni per l'immagine corrente (predizioni e groundtruth)

# Ciclo sulle annotazioni

# Calcolo dell'IoU per l'annotazione corrente

# Se l'IoU è maggiore della soglia

    # Incremento del contatore relativo ai true positive (rilevazione)

    # Se la classe predetta è with_mask
        # Se corrisponde al groundtruth incrementa i true positive (classificazione)
        # Se non corrisponde al groundtruth incrementa i false positive (classificazione)

    # Se la classe predetta è without_mask
        # Se corrisponde al groundtruth incrementa i true negative (classificazione)
        # Se non corrisponde al groundtruth incrementa i false negative (classificazione)

# Conteggio dei falsi negativi (rilevazione): Num bbox groundtruth - true positive (rilevazione)
# Conteggio dei falsi positivi (rilevazione): Num bbox predette - true positive (rilevazione)
```

Figura 28: Pseudocodice della funzione `get_batch_statistics`.

Le seguenti metriche sono state calcolate utilizzando l'approccio descritto in precedenza:

- **Detection** per il quale i seguenti indici sono utili nel valutare l'accuratezza del modello nel rilevare le mascherine.
 - **True Positives** che indicano i casi in cui il modello ha individuato correttamente una bounding box che era presente nel ground truth;
 - **False Positives** che indicano i casi in cui il modello ha individuato una bounding box che non era presente nel ground truth;
 - **False Negatives** che indicano i casi in cui il modello non ha individuato una bounding box che era presente nel ground truth;
- **Classification** per il quale i seguenti indici sono utili nel valutare l'accuratezza del modello nel classificare le mascherine rilevate correttamente (veri positivi).

- **True Positives** che indicano i casi in cui il modello ha predetto correttamente la classe `with_mask`;
- **True Negatives** che indicano i casi in cui il modello ha predetto correttamente la classe `without_mask`;
- **False Positives** che indicano i casi in cui il modello ha erroneamente predetto la classe `with_mask` dove la classe originale era `without_mask`;
- **False Negatives** che indicano i casi in cui il modello ha erroneamente predetto la classe `without_mask` dove la classe originale era `with_mask`.

Successivamente, mediante l'utilizzo della funzione `ap_per_class`, sono state calcolate le seguenti metriche:

- **Precision** che misura la proporzione di previsioni corrette rispetto a tutte le previsioni positive effettuate dal modello. Misura la capacità del modello di identificare correttamente le mascherine rispetto a tutte le predizioni positive;
- **Recall** che misura la proporzione di oggetti positivi correttamente individuati dal modello rispetto al numero di oggetti presenti nel ground truth di riferimento;
- **F1-score** che è definito come la media armonica di precision e recall (penalizza i valori più bassi in modo più significativo rispetto alla media aritmetica);
- **Average Precision (AP)** che misura la precisione media del modello per ogni classe. È definito come l'area sottostante alla curva di precision-recall. L'AP valuta la capacità del modello di effettuare previsioni precise (precision) e di individuare correttamente gli oggetti delle diverse classi (recall);
- **Medium AP** che viene calcolata come la media delle varie AP ed è una misura complessiva delle prestazioni del modello che tiene conto di tutte le classi del problema.

3.5.3 Matrice di confusione

La matrice di confusione è una rappresentazione tabulare utilizzata per valutare le prestazioni di classificazione di un modello. Sulle righe della matrice sono presenti

le classi originali del dataset, mentre sulle colonne le classi predette dal modello. Nei vari elementi è presente il conteggio delle istanze che appartengono ad una specifica classe reale e vengono predette come appartenenti ad una specifica classe predetta.

La matrice di confusione viene creata, utilizzando gli indici di classificazione calcolati precedentemente, in questo modo:

```
confusion_matrix = np.array([[conf_total_tp, conf_total_fp], [conf_total_fn, conf_total_tn]])
```

Figura 29: Creazione della matrice di confusione.

Successivamente, viene normalizzata e visualizzata a video grazie alle funzioni `normalize_confusion_matrix` e `show_confusion_matrix`.

```
def normalize_confusion_matrix(conf_matrix):
    # Somma degli elementi per ogni riga della matrice
    row_sums = conf_matrix.sum(axis=1)

    # Normalizzazione
    normalized_conf_matrix = conf_matrix / row_sums[:, np.newaxis]

    return normalized_conf_matrix

def show_confusion_matrix(conf_matrix, class_names, figsize=(10, 10)):
    # Normalizzazione
    normalized_conf_matrix = normalize_confusion_matrix(conf_matrix)

    # Creazione della griglia
    fig, ax = plt.subplots(figsize=figsize)
    img = ax.matshow(normalized_conf_matrix)
    tick_marks = np.arange(len(class_names))
    _ = plt.xticks(tick_marks, class_names, rotation=45)
    _ = plt.yticks(tick_marks, class_names)
    _ = plt.ylabel('Real')
    _ = plt.xlabel('Predicted')

    # Aggiunta dei valori percentuali
    for i in range(len(class_names)):
        for j in range(len(class_names)):
            text = ax.text(j, i, '{0:.1%}'.format(normalized_conf_matrix[i, j]),
                           ha='center', va='center', color='w')
```

Figura 30: Visualizzazione della matrice di confusione.

4 Analisi delle performance e considerazioni

Prima di procedere con l'effettiva analisi delle performance è importante prendere in considerazione alcune riflessioni da comprendere e contestualizzare, al fine di attribuire un significato all'analisi stessa.

4.1 Premesse

Anzitutto, a livello teorico, per una corretta valutazione, occorrerebbe **addestrare le reti su più epoche** rispetto alle 200 eseguite all'interno del progetto e si dovrebbe prendere il punto ottimale di ognuna di esse. Facendo così, le metriche *rappresenterebbero al massimo le potenzialità di ogni rete* e risulterebbero molto più significative.

Per quanto riguarda RetinaNet e Faster R-CNN, dalle informazioni in possesso, questo non è risultato pratico e fattibile. Per questo motivo si è scelto di considerare i valori delle prestazioni dell'ultima epoca di addestramento anche per YOLO di modo che *non risultasse sbilanciato il confronto* e che fosse *mantenuta una certa equità* con le altre due reti, nonostante YOLO salvasse i valori delle metriche ad ogni epoca dell'addestramento e che quindi sarebbe bastato recuperare i valori dell'epoca ottimale. Oltretutto è importante sottolineare come l'epoca ottimale di YOLO potesse non esserlo per tutte le metriche (per esempio una determinata epoca può essere ottimale per la mAP ma non per la precision o la recall).

Sicuramente, avendo scelto di addestrare solamente su 200 epoche, la **differenza** tra l'ultima epoca e quella ottimale risulta essere comunque **minima**.

Come spiegato in precedenza, anche il numero di immagini limitato del dataset e come questo è stato suddiviso in percentuale, incide fortemente su ogni rete in modo diverso. Infatti a causa delle diverse dinamiche di apprendimento e dei tempi di crescita variabili dei singoli modelli, l'uguaglianza delle percentuali e il numero limitato di immagini **potrebbe favorire alcuni modelli rispetto ad altri**. Questa variazione nelle prestazioni può essere attribuita alle *caratteristiche uniche di ciascun modello* e ai *differenti ritmi di apprendimento* che essi presentano.

Infine, per una valutazione esaustiva dei tre modelli, è importante prendere in considerazione il fatto che **YOLO** richiede tempi di addestramento significativamente inferiori rispetto a RetinaNet e Faster R-CNN (risulta infatti circa 5/6 volte più veloce). Questa disparità è attribuibile al fatto che YOLO è una rete single-stage, a differenza di Faster R-CNN che è composta da più fasi di elaborazione.

4.2 Considerazioni sulle prestazioni

Dopo aver chiarito tutte le premesse, è ora possibile procedere all'analisi precisa delle metriche ottenute dalle predizioni di ciascuna rete. Inizialmente, prendendo in considerazione le loss relative al training e al validation set, si osserva in Figura 31a che nel caso di RetinaNet le loss mantengono una curva costante durante tutte le epoche. Nello specifico, la loss nel training mostra una *tendenza discendente*, suggerendo la **possibilità di ulteriori miglioramenti** anche oltre la duecentesima epoca. Al contrario, la loss di validazione rimane relativamente stabile nel tempo suggerendo un principio di **overfitting**.

Un discorso molto simile può essere fatto con le loss di YOLO, visibili in Figura 32, che tendono a diminuire con uno sviluppo pressochè costante. Faster R-CNN presenta qualche picco nella train loss e un andamento a "dente di sega" per la validation loss (Figura 31b). Questo potrebbe essere dovuto a diversi fattori quali:

- La **dimensione del mini-batch** che potrebbe portare la rete a imparare in modo non uniforme dai campioni. In tal caso, è consigliabile esaminare e sperimentare con diverse dimensioni del mini-batch per ottenere una convergenza più stabile;
- Il **learning rate troppo alto** che potrebbe causare oscillazioni nella loss durante l'addestramento. Si potrebbe pensare di ridurlo gradualmente nel corso delle epoche, per facilitare una convergenza più fluida, nonostante il valore di per sé sia già contenuto (0.005);
- La presenza di un **dataset sbilanciato** che potrebbe rendere difficile per il modello imparare correttamente le classi meno rappresentate. In questo caso, è utile considerare strategie come la *riponderazione delle classi* o l'uso

di tecniche di generazione di dati sintetici (**data augmentation**) per bilanciare la distribuzione delle classi e migliorare le performance complessive del modello.

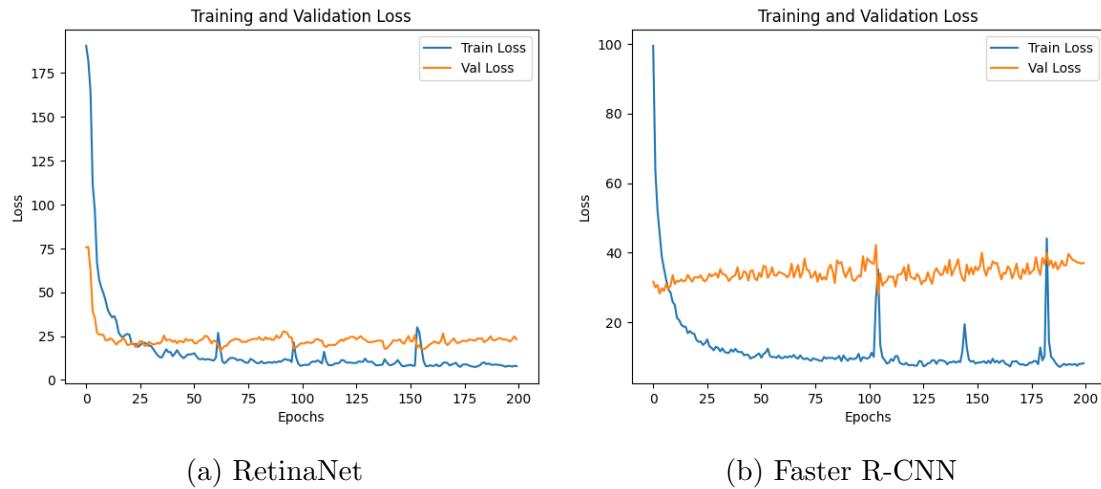


Figura 31: Training loss e validation loss su un addestramento di RetinaNet e Faster R-CNN da 200 epoche.

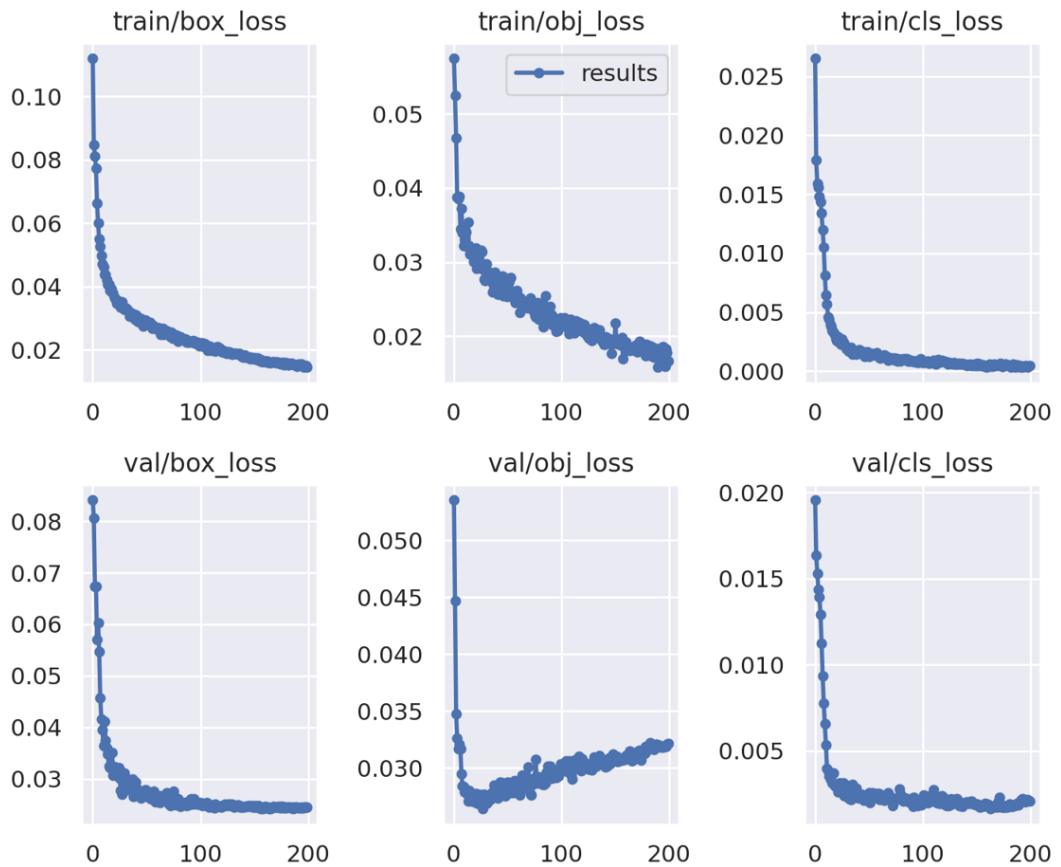


Figura 32: Loss di regressione, di objectness e di classificazione su training e validation set per un addestramento di YOLO da 200 epoche.

Per quanto riguarda le successive metriche, sono necessarie alcune precisazioni. In tutte le metriche di valutazione e per tutti i modelli è evidente una differenza significativa tra i valori ottenuti per la classe `with_mask` e quelli per la classe `without_mask` con quest'ultima che **registra risultati nettamente inferiori**.

Queste differenze sono principalmente attribuibili al fatto che il numero di bounding box etichettate come `without_mask` sia inferiore rispetto alla classe `with_mask`. Di conseguenza, la **rete dispone di un numero minore di esempi** di addestramento per apprendere correttamente a riconoscere questa classe, con un impatto negativo sulle metriche di valutazione.

È importante considerare questa disparità nella quantità di dati disponibili per le diverse classi poiché può influire sulle prestazioni della rete nella classificazione di quelle meno rappresentate. Per affrontare questa problematica, potrebbe essere utile **raccogliere ulteriori dati per la classe without_mask** o adottare **strategie di bilanciamento dei dati** per garantire una *rappresentazione più equilibrata* delle classi durante l'addestramento.

Inoltre è comunque interessante riflettere sul fatto che potrebbe esserci anche una *diversa difficoltà nel riconoscimento delle due classi*. Ciò potrebbe essere dovuto a vari fattori, come la presenza di diverse variazioni nell'aspetto delle persone senza mascherina (es. diversi tipi di espressioni facciali, posizioni della testa, etc.), che rendono più complesso il compito di rilevamento. Se la classe **without_mask** presenta sfide maggiori rispetto alla classe **with_mask** potrebbe influire negativamente sulle metriche di valutazione per quella specifica classe.

Infine questo potrebbe essere aggravato dal fatto che le bounding box predette come **without_mask** corrispondano effettivamente a *esempi in cui inizialmente erano presenti istanze mask_weared_incorrect* nel ground truth, ma che sono state eliminate durante la fase di pre-elaborazione dei dati. Ciò può causare una **discrepanza** tra le etichette effettive e le etichette predette.

Se si approfondisce l'analisi, in Figura 33 e in Figura 34 emerge che la Faster R-CNN ha una maggiore difficoltà nel riconoscimento delle bounding box rispetto agli altri modelli poiché presenta una *recall più bassa*. Tuttavia, quando riesce a individuare queste bounding box, dimostra un'*ottima precisione* nella loro classificazione, superiore alle altre reti. Questo aspetto si può constatare anche nella confusion matrix prodotta (Figura 38a), che mostra valori perfetti nella classificazione.

Precision

Precisione del modello nel riconoscere correttamente le mascherine e le loro assenze

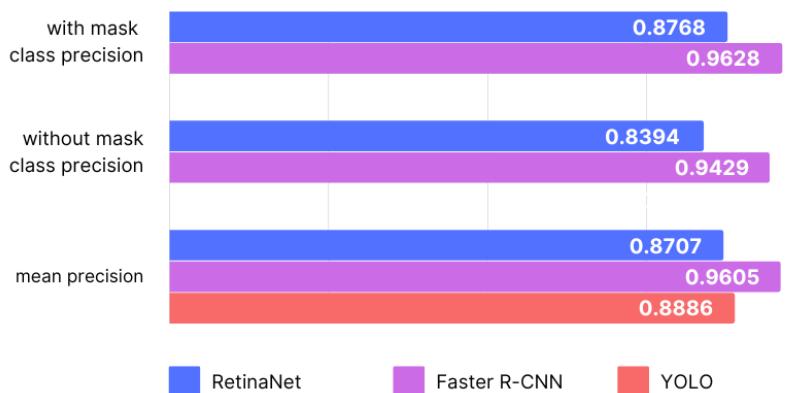


Figura 33: Comparazione valori di precision tra le reti.

Recall

Capacità del modello di individuare tutti gli oggetti positivi in modo completo e accurato



Figura 34: Comparazione valori di recall tra le reti.

In questo senso, RetinaNet e YOLO sono nuovamente più simili tra loro, *evidenziando un'ottima recall e una precision leggermente peggiore*. In tutti i casi si parla comunque di valori buoni se non ottimali, soprattutto considerando il dataset limitato.

Tuttavia, è importante considerare che i valori inferiori di recall per la classe `without_mask` nella Faster R-CNN potrebbero non essere necessariamente negativi o quantomeno potrebbero essere giustificati dal fatto che, come spiegato in precedenza, **sono state eliminate le bounding box di classe mask_weared_incorrect e non sono state etichettate in modo perfetto alcune istanze** nel dataset di partenza. Di conseguenza, è possibile che la Faster riconosca delle bounding box che nel ground truth non erano presenti (per la poca attenzione nell'etichettatura e/o l'eliminazione della classe), risultando in una recall più bassa nonostante la possibile correttezza nelle predizioni. Questo è facilmente apprezzabile in Figura 35a e in Figura 35b, dove nella prima si può notare la disattenzione nell'etichettare le bounding box sullo sfondo mentre nella seconda è chiara la mancanza di bounding box per le persone che indossano in modo non propriamente corretto la mascherina. Questa considerazione è *valida per tutte le reti* dal momento che sono stati registrati casi simili in ognuna di esse, sebbene Faster R-CNN potrebbe averne sofferto maggiormente rispetto alle altre.



(a) Esempio di disattenzione nell'etichettatura (b) Esempio di eliminazione della classe

Figura 35: Differenze tra ground truth e bounding box predette dalla Faster.

Principalmente per la mAP, nella quale è più impattante ed evidente anche se osservabile pure nelle altre metriche, **YOLO dimostra prestazioni superiori a tutti** (Figura 37). Questo risultato non è solo *sorprendente* ma è anche *totalmente inaspettato* dal momento che YOLO è un modello single-stage e questo dovrebbe renderlo meno preciso se comparato a modelli a due fasi.

Medium Average Precision (mAP)

Stima generale dell'accuratezza e della capacità del modello di riconoscere correttamente gli oggetti di diverse classi

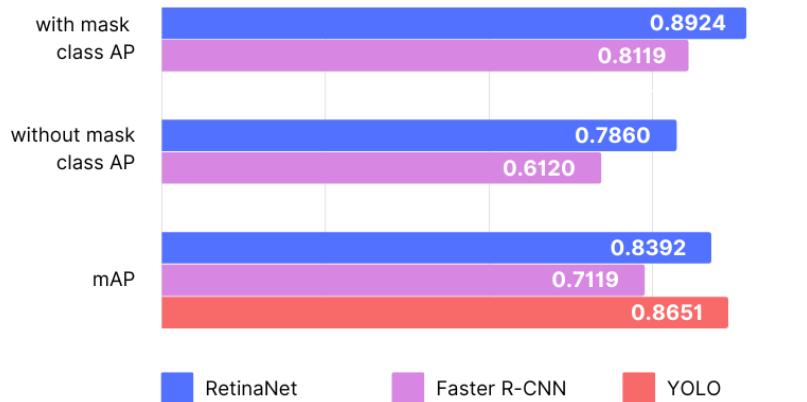


Figura 36: Comparazione valori di mAP tra le reti.

F1-score

Misura complessiva delle prestazioni di un modello che tiene conto sia della precision che della recall

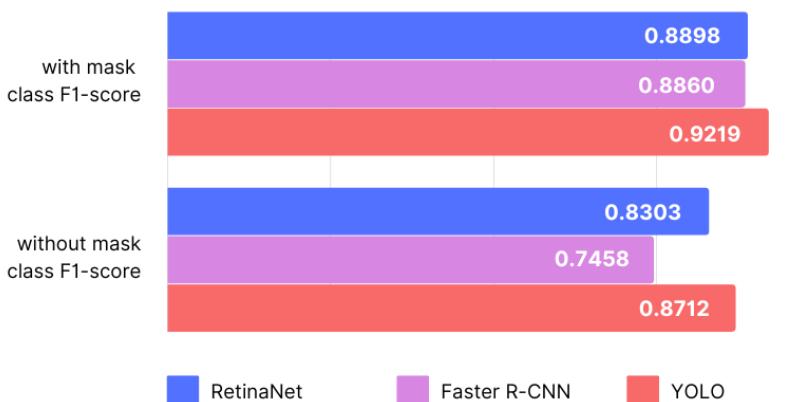
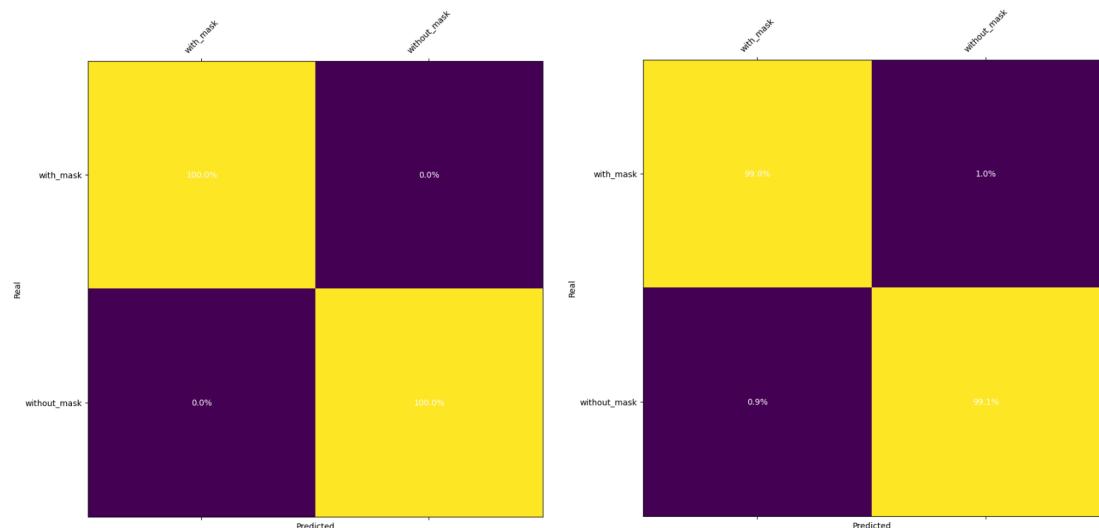


Figura 37: Comparazione valori f1-score tra le reti.

Sempre considerando la mAP va comunicato che per tutte le reti è stata scelta una threshold di 0.5, nonostante per YOLO fosse possibile recuperare anche le performance con soglia tra 0.5 e 0.95; questa scelta è sempre riconducibile ad un senso di equità tra le reti.

Non si è manifestato *nessun problema con YOLO per quanto riguarda l'uso delle celle di dimensioni fisse* al posto di vere e proprie bounding box. Questo **avrebbe potuto precludere la rilevazione di oggetti molto vicini tra loro**. Tuttavia, si suppone che le celle utilizzate possano essere sufficientemente piccole, in modo tale che questo problema non influisca significativamente, o, come in questo caso, non si verifichi affatto.

Infine, le riflessioni attuate sui risultati consigliano anche come sviluppo futuro quello di **eseguire operazioni di data augmentation** sul dataset di modo da poter disporre di più dati in fase di addestramento, oltre che la possibilità di introdurre operazioni di **pre-processing** come la normalizzazione dei valori di intensità (di modo da uniformare le luminosità che potrebbero differire da un'immagine all'altra) o la possibilità di convertire le coordinate delle bounding box portandole da fisse a un valore normalizzato tra 0 e 1.



(a) Confusion matrix di Faster R-CNN.

(b) Confusion matrix di RetinaNet.

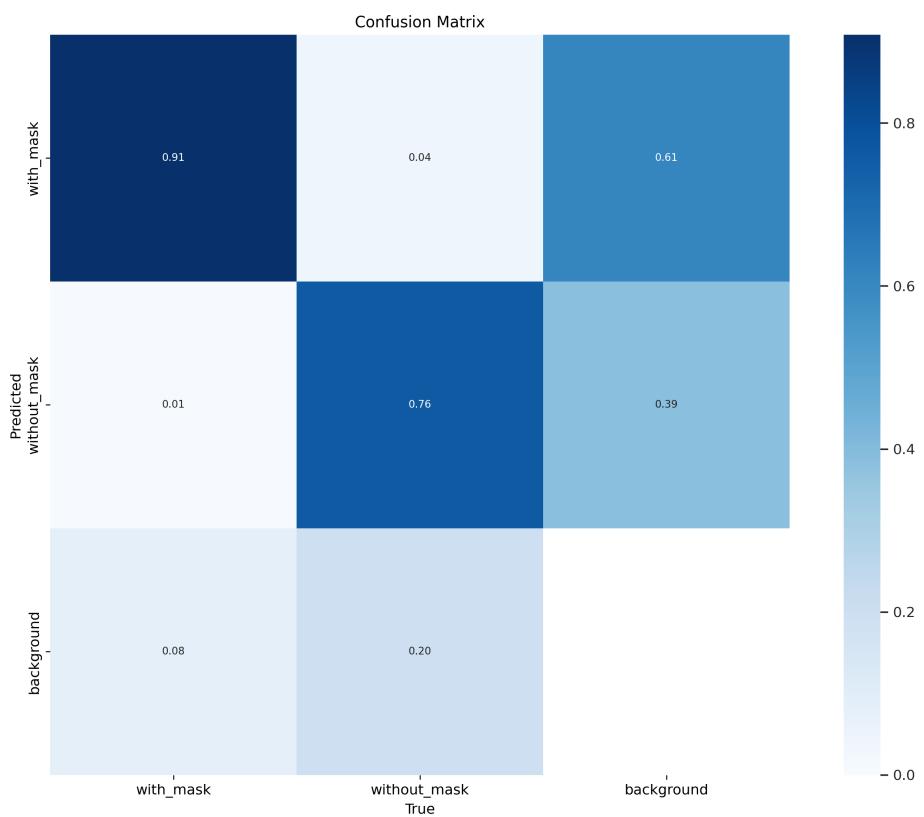


Figura 39: Confusion matrix di YOLO.

Vengono di seguito riportate le tabelle riepilogative con tutti i valori delle tre reti (Figura 40).

Modello	Retina Net	Faster R-CNN	YOLOv5
Precision	0.8707	0.9605	0.8886
Recall	0.8896	0.7910	0.8067
mAP	0.8392	0.7119	0.8651
F1 (with_mask)	0.8898	0.8860	0.9219
F1 (without_mask)	0.8303	0.7458	0.8712

Figura 40: Comparazione delle performance.

5 Conclusioni

In conclusione di quanto realizzato in merito a questo elaborato, riteniamo che sia stata una tipologia di lavoro molto utile, sia per affinare i concetti teorici elaborati in aula, sia per assimilare meglio quelli pratici.

Ci siamo trovati a mettere in pratica nozioni, a nostro parere, non semplici ma che con l'intesa di gruppo siamo riusciti a gestire in modo efficace. Nello specifico, è risultato particolarmente interessante riuscire a creare reti di diverso tipo, calcolarne le prestazioni e confrontarle tra loro. Comprendiamo che questi argomenti sono e saranno sempre più importanti nel mondo del lavoro e per questo abbiamo cercato, in tutte le parti di questo progetto, di utilizzare al meglio le nostre conoscenze al fine di ottenere il risultato migliore possibile.

In particolare, abbiamo trovato molto interessante quanto le potenzialità che tutto il ramo dell'informatica e non possano evolversi grazie all'utilizzo di queste tecnologie. Ci auguriamo che gli svariati ambiti di applicazione e i risultati che si possono raggiungere, ci forniscano con le esperienze future spunti di approfondimento sui diversi temi.

In conclusione, dato l'interesse del gruppo verso questi argomenti, ci auguriamo di svolgere progetti simili anche in futuro.