



UNIVERSITÀ
degli STUDI
di CATANIA

CLASSIFICAZIONE DEI PUNTI DI INTERESSE DI UN SITO CULTURALE MEDIANTE OBJECT RECOGNITION

Dipartimento di Matematica e Informatica

Corso: Machine Learning (LM-18)

Docente: Giovanni Maria Farinella

Michele Ferro

Matricola: 1000037665

Luglio 2022

Sommario

| | |
|---|-----------|
| Capitolo 1: Problema | 3 |
| Capitolo 2: Dataset | 4 |
| 2.1 Raccolta dei dati e specifiche generali | 4 |
| Monastero dei Benedettini | 4 |
| Palazzo Bellomo | 4 |
| 2.2 Suddivisione in cartelle | 5 |
| 2.3 Problemi semantici | 10 |
| 2.3.1 Etichettatura non corretta | 11 |
| 2.3.2 Struttura delle cartelle non corretta | 12 |
| Capitolo 3: Metodo | 15 |
| 3.1 Object Recognition | 15 |
| 3.2 Convolutional Neural Network | 17 |
| 3.3 R-CNN | 18 |
| 3.4 YOLO | 19 |
| 3.4.1 Network Design | 20 |
| 3.4.2 Fase di training | 22 |
| FASE DI PRE-TRAINING | 22 |
| FASE DI TRAINING COMPLESSIVO | 22 |
| 3.4.3 Funzione di Loss | 23 |
| 3.4.4 Limitazioni | 23 |
| 3.4.5 Ulteriori versioni di YOLO | 23 |
| YOLOv2 | 24 |
| YOLOv3 | 26 |
| YOLOv4 | 28 |
| YOLOv5 | 30 |
| PP-YOLO | 31 |
| Capitolo 4: Valutazione | 32 |
| 4.1 Matrice di confusione e metriche di classificazione | 32 |
| 4.1.1 Accuracy | 32 |
| 4.1.2 Precision | 33 |
| 4.1.3 Recall | 33 |
| 4.1.4 F1-score | 33 |
| 4.2 Average Precision | 33 |
| 4.3 Mean Average Precision | 34 |
| 4.4 Intersection over Union | 34 |
| Capitolo 5: Esperimenti | 36 |
| 5.1 Tentativi su YOLOv5s | 36 |
| 5.1.1 Primo tentativo di training | 37 |
| Monastero dei Benedettini | 37 |

| | |
|--|-----------|
| Palazzo Bellomo | 38 |
| 5.1.2 Transfer Learning | 39 |
| Monastero dei Benedettini | 40 |
| Palazzo Bellomo | 40 |
| 5.2 Tentativi su YOLOv5m | 42 |
| 5.2.1 Primo tentativo di training | 42 |
| Monastero dei Benedettini | 42 |
| Palazzo Bellomo | 43 |
| 5.2.2 Transfer Learning | 44 |
| Monastero dei Benedettini | 44 |
| Palazzo Bellomo | 45 |
| 5.3 Tentativi su YOLOv5l | 46 |
| 5.3.1 Primo tentativo di training | 47 |
| Monastero di Benedettini | 47 |
| Palazzo Bellomo | 48 |
| 5.3.2 Transfer Learning | 49 |
| Monastero dei Benedettini | 49 |
| Palazzo Bellomo | 50 |
| 5.4 Confronto dei risultati ottenuti | 51 |
| Capitolo 6: Demo | 53 |
| Capitolo 7: Codice | 54 |
| 7.1 Gestione del dataset | 54 |
| 7.1.1 Rinomina dei file | 54 |
| 7.1.2 Riparazione delle etichette | 54 |
| 7.1.3 Creazione dello yaml di configurazione | 55 |
| 7.2 Addestramento | 55 |
| Conclusioni | 56 |

Capitolo 1: Problema

La riapertura delle strutture, e soprattutto il re-intensificarsi del turismo, ha fatto sì che nell'ultimo periodo l'affluenza di visite presso musei e siti archeologici aumentasse; un problema particolarmente sentito da questo punto di vista, è quello dell'assistenza ai visitatori, il quale non può essere risolto senza opportuni mezzi, in realtà provenienti dai diretti interessati.

Infatti, conoscere il comportamento di un individuo che sta osservando un'esposizione in un sito culturale, può aiutare molto nella sua assistenza, e soprattutto nel comprendere in che maniera è possibile migliorare il sito stesso e da quale punto di vista; tuttavia, è anche vero che – considerando la grossa mole giornaliera di visitatori – sarebbe impossibile intervenire manualmente su ognuno di essi (per esempio, sarebbe difficile indicare ad ognuno di essi come navigare in maniera agevole il sito stesso, o riconoscere le varie opere esposte). Indi per cui, oggi si potrebbe sfruttare una tecnologia di realtà aumentata (come degli **smart glass**) in grado di automatizzare questo processo di assistenza per ognuno dei visitatori: per esempio, indossando uno di questi dispositivi, si potrebbero ricevere – in tempo reale – informazioni sull'opera o sul punto di interesse, da qui in poi **Point Of Interest (POI)**, che si sta osservando per poi suggerire cosa andare a vedere successivamente.

Affinché ciò sia possibile, è necessario anzitutto poter effettuare un **riconoscimento degli oggetti**, in particolare di questi POI, così che seguitamente gli utenti possano usufruire di un siffatto servizio, e che inoltre i gestori di siti culturali possano beneficiarne, in maniera da essere in grado di registrare in tempo reale le loro reazioni a fronte di miglioramenti all'intera struttura, da un punto di vista sia culturale che tecnologico.

In particolare, in questo progetto, si vorrebbe che, a partire da dati etichettati preventivamente raccolti (la struttura di questo dataset e le modalità di raccolta verranno maggiormente approfondite nel prossimo paragrafo), un qualche algoritmo sia in grado di riconoscere il POI che si sta osservando, per poi fornire all'utente informazioni circa esso (per esempio, nome del POI, data di realizzazione o da chi è stato realizzato potrebbero essere informazioni utili). In questo modo, ogni utente presente sul sito sarà in grado di avere in tempo reale informazioni sull'opera stessa, ed il suo gestore non dovrà necessitare di una maggiore forza lavoro, cioè più addetti all'assistenza dei visitatori.

Detto in maniera più formale e precisa, si vuole realizzare un classificatore multiclasse in grado di riconoscere, mediante un processo di object recognition attuato con il framework **YOLO**, il POI raffigurato in un'immagine, e che restituisca il suo nome (per cui, ci sarà una classe per ogni identificativo dei POI).

Capitolo 2: Dataset

2.1 Raccolta dei dati e specifiche generali

Il dataset utilizzato è **EGOcentric-Cultural Heritage¹ (EGO-CH)**, il primo dataset di filmati in prima persona per la comprensione delle reazioni dei visitatori in un sito culturale; in particolare, i siti dai quali sono stati acquisiti questi dati sono il *Monastero dei Benedettini* in Catania e la *Galleria Regionale di Palazzo Bellomo* in Siracusa.

I dati che lo compongono sono stati raccolti facendo utilizzo del dispositivo indossabile **Microsoft HoloLens** su ognuno dei due siti. In generale, il dataset contiene più di 27 ore di filmati, 26 ambienti diversi, più di 200 POI e 70 visite. Di seguito la sua composizione più particolareggiata.

Monastero dei Benedettini

Secondo quanto dichiarato sulla documentazione del dataset, questo primo sito è composto da 4 ambienti e contiene 35 POI. A differenza del successivo, il *Monastero dei Benedettini* racchiude tra i suoi POI sia opere statuarie e pittoriche, sia architettoniche (pavimenti, pareti, ecc...); ciò dipende dalla differenza tra i due siti culturali: se il *Palazzo Bellomo* è “solamente” un museo, in questo caso ci troviamo in un grande complesso ecclesiastico settecentesco, per cui non tutti i POI potranno essere riconosciuti utilizzando delle strategie di object detection.

- I **video di training**, collezionati nella medesima modalità vista in precedenza, sono 48. La durata media è di 133. 06 s, mentre la massima è di 679 s.
- I **video di validazione** sono stati raccolti chiedendo a dei volontari di visitare il sito secondo lo stesso protocollo utilizzato nella raccolta attuata per il Palazzo Bellomo, per un totale di 5 filmati.

Questa categoria di filmati e i precedenti hanno entrambi una risoluzione di 1216×648 pixel ed un frame-rate pari a *24 fps*.

- Infine, sono stati raccolti 60 **video di test** chiedendo a dei reali visitatori (totalmente ignari sia della ricerca e del suo scopo, sia delle capacità di utilizzo di HoloLens) di esplorare i diversi ambienti della struttura liberamente e osservare i vari POI.

Questi video, la cui durata media è di *21 min* e la massima è di *42 min*, hanno una risoluzione di 1408×792 pixel ed un frame-rate di *30. 03 fps*.

Palazzo Bellomo

Secondo quanto dichiarato, questo sito è composto da 22 ambienti diversi e contiene 191 POI (i quali possono essere dipinti, affreschi, statue, rilievi, ecc...).

- I **video di training** sono stati collezionati da operatori istruiti a camminare per il sito, in maniera da catturare immagini di ciascun ambiente da ogni punto di vista possibile; nel caso di ambienti esterni, sono stati raccolti più filmati in diverse condizioni di luce, per un totale di 57 istanze.

¹ "EGO-CH: Dataset and Fundamental Tasks for Visitors ... - IPLab." <https://iplab.dmi.unict.it/EGO-CH/>. Ultimo accesso: 2 giu. 2022.

La loro durata media è di 81.72 min, con il più duraturo di 147 min.

- I **video di test** sono stati raccolti separatamente, chiedendo a dieci volontari di visitare il sito; si tenga in considerazione, che la maggior parte degli individui impegnati a raccogliere questa categoria di filmati, avevano limitata confidenza con l'ambiente e che nessuna istruzione era stata fornita sullo specifico ambiente da visitare o il POI da osservare: ciò ha permesso di avere dei dati realistici dell'osservazione dei POI da parte di un visitatore.
La loro durata media è di 31.27 min, con il più duraturo di 50.23 min.
- Infine, dai video di test è stato estratto in maniera casuale un filmato, utilizzato come unico elemento del **validation set**.

Tutti i filmati per questo sito hanno una risoluzione 1980×720 , con un frame-rate pari a 29.97 fps .

È importante specificare in questa disamina la presenza di un subset di fotogrammi estratti da questi da queste banche di dati, etichettati in maniera di specificare la presenza di un POI mediante una **bounding box** e, per ognuno di essi, il loro identificativo: in particolare, ogni POI è stato etichettato mediante una tupla del tipo $(class, x, y, w, h)$, la quale indica classe del singolo oggetto e altre informazioni di tipo geometrico relative alla bounding box stessa, le quali verranno approfondite in seguito.

Per la risoluzione di questo specifico task, ossia quello del riconoscimento dei POI mediante object recognition, è stato utilizzato un subset – più consono a tal scopo – estratto dal dataset originale precedentemente descritto, il quale presenta le seguenti caratteristiche:

- per quanto concerne il (sub)subset relativo al *Palazzo Bellomo*, esso presenta un totale di 56686 frame, poi suddivisi in 41111 frame di training e 15575 frame di validazione;
- relativamente al *Monastero dei Benedettini*, il suo (sub-)subset presenta un totale di 33366 frame, suddivisi in 23363 frame di training e 10003 frame di validazione.

2.2 Suddivisione in cartelle

Il dataset utilizzato per la risoluzione di questo task presenta una distribuzione in file e cartelle come presentata di seguito.

Vengono inoltre mostrati gli esempi dei frame presenti nei dataset, uno per ognuna delle classi individuate sui rispettivi siti culturali.

Monastero dei Benedettini:

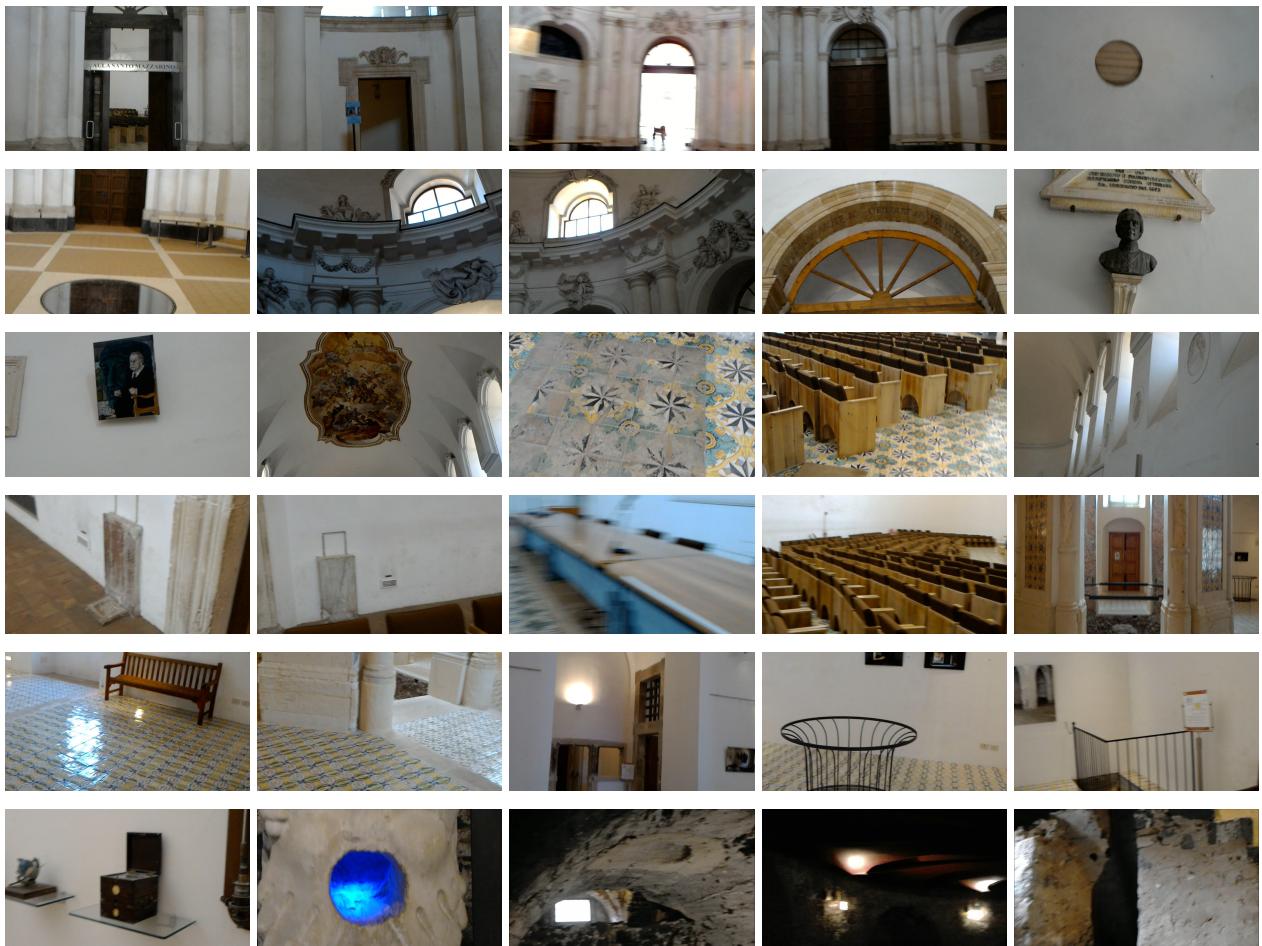
- cartella **Training** contenente per ogni POI le immagini (cartella **images**), l'etichettatura nel formato supportato da YOLO (cartella **labels**) ed un file **.csv** contenente le annotazioni relative alle bounding box; in particolare, le sottocartelle sono organizzate nel seguente modo:
 - <environment>.ID_POI:
 - **images**:
 - **frame_000abc.jpg**
 - **frame_000abc.txt**
 - **labels**:

- frame_000abc.txt
 - <environment>.ID_POI.csv
- cartella Test, contenenti le seguenti sottocartelle:
 - lab:
 - 00abcd.jpg
 - bbox_annotations:
 - lab.txt
 - YOLO_annotations:
 - lab:
 - 00abcd.txt
- cartella Validation, contenente le seguenti sottocartelle:
 - images:
 - TestN.<environment>_abcd.jpg
 - bbox_annotations:
 - TestN.<environment> abcd.txt
 - YOLO_annotations:
 - TestN.<environment>_abcd.txt

Di seguito alcuni esempi visuali dalle classi presenti:

- 5.2_PortaIngressoMuseoFabbrica
- 5.3_PortaAntirefettorio
- 5.4_PortaIngressoRef.Piccolo
- 5.5_Cupola
- 5.6_AperturaPavimento
- 5.7_S.Agata
- 5.8_S.Scolastica
- 5.9_ArcocoFirma
- 5.10_BustoVaccarini
- 6.1_QuadroSantoMazzarino
- 6.2_Affresco
- 6.3_PavimentoOriginale
- 6.4_PavimentoRestaurato
- 6.5_BassorilieviMancanti
- 6.6_LavamaniSx
- 6.7_LavamaniDx
- 6.8_TavoloRelatori
- 6.9_Poltrone
- 7.1_Edicola
- 7.2_PavimentoA
- 7.3_PavimentoB
- 7.4_Passavivande
- 7.5_AperturaPavimento

- 7.6_Scala
- 7.7_SalaMetereologica
- 8.1_Doccione
- 8.2_VanoRaccoltaCenere
- 8.3_SalaRossa
- 8.4_ScalaCucina
- 8.5_CucinaProvv
- 8.6_Ghiacciaia
- 8.7_Latrina
- 8.8_OssaeScarti
- 8.9_Pozzo
- 8.10_Cisterna
- 8.11_BustoPietroTacchini





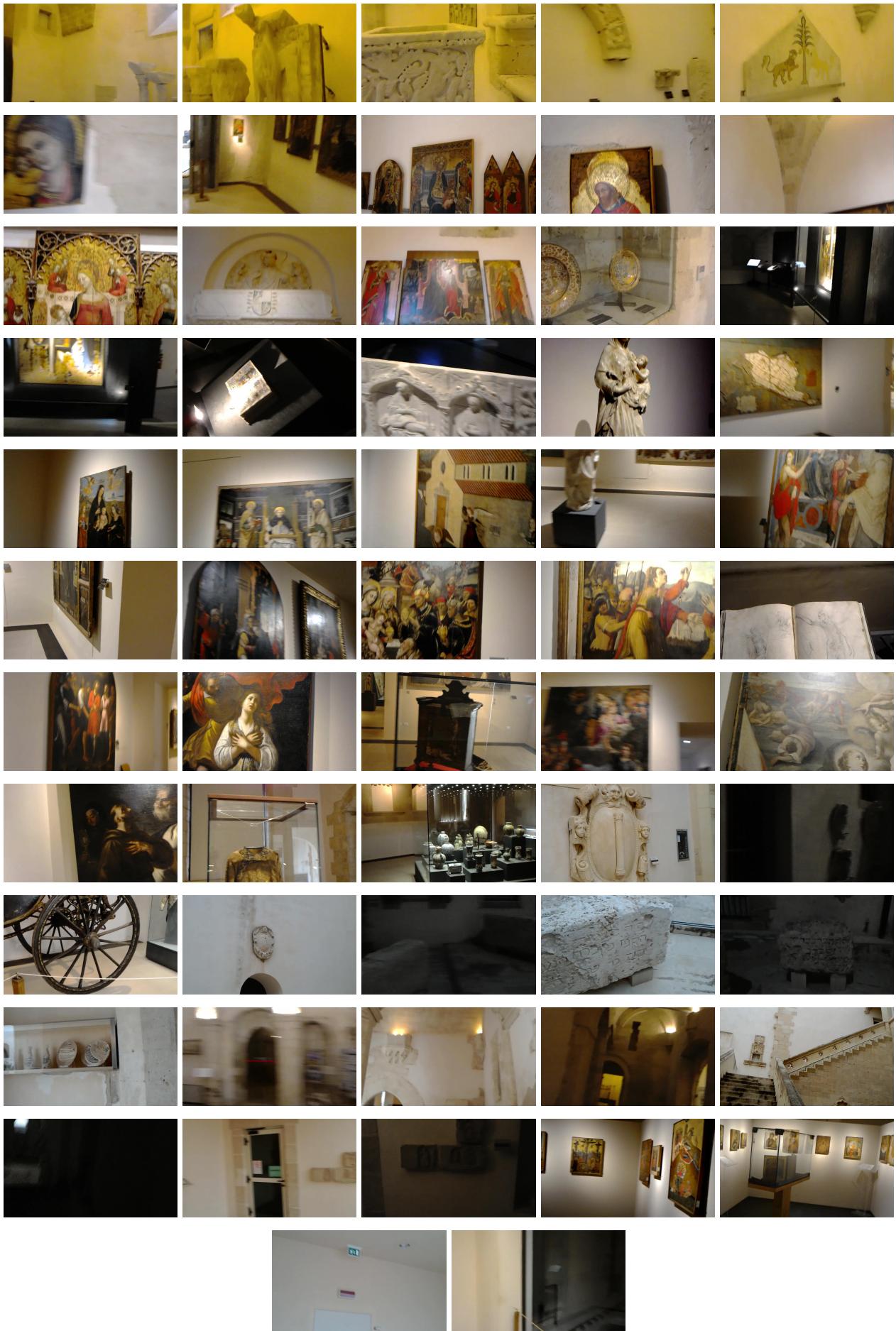
Palazzo Bellomo:

- cartelle Training e Test, contenenti tre ulteriori sottocartelle:
 - images:
 - TestN:
 - abcde.jpg
 - bbox_annotations:
 - TestN.txt
 - YOLO_annotations:
 - TestN:
 - abcde.txt

Di seguito degli esempi visuali dalle classi presenti:

- 1.0_Sala1
- 2.0_Sala2
- 2.1
- 2.2
- 2.3
- 3.0_Sala3
- 3.0_Sala3_S
- 3.1
- 3.2
- 4.0_Sala4
- 4.1
- 4.2
- 4.3
- 4.4
- 5.0_Sala5
- 5.1
- 5.2
- 5.3
- 5.4
- 6.0_Sala6
- 7.0_Sala7
- 7.1

- 7.2
- 7.3
- 8.0_Sala8
- 8.1
- 9.0_Sala9
- 9.1
- 9.2
- 9.3
- 10.0_Sala10
- 10.1
- 10.2
- 11.0_Sala11
- 11.1
- 11.2
- 12.0_Sala12
- 13.0_Sala13
- 14_CortiledegliStemmi
- 14_CortiledegliStemmi_S
- 15.0_SalaCarrozze
- 16.0_CortileParisio
- 16.0_CortileParisio_S
- 16.1
- 16.1_S
- 17.0_BIglietteria
- 17.0_Biglietteria_S
- 18.0_Portico
- 18.0_Portico_S
- 19.0_ScalaCatalana
- 19.0_ScalaCatalana_S
- 20.0_Loggetta
- 20.0_Loggetta_S
- 21.0_BoxSala8
- 21.1
- 22.0_AreaSosta
- 22.0_AreaSosta_S



N.B.:

- In <environment>.ID_POI, <environment> indica l'identificatore dell'ambiente (nel caso del *Monastero dei Benedettini* da 1 a 22, mentre nel caso di *Palazzo Bellomo* da 1 a 8), ID_POI indica identificatore e nome del POI considerato;
- in frame_000abc, abc indica ogni singola cifra del numero del fotogramma;
- in TestN.ID_abcd, N indica il numero del test, ID indica l'identificativo del, <environment> indica l'identificatore dell'ambiente (come sopra), abcd indica il numero del frame (come sopra).

2.3 Problemi semantici

Duole constatare che, tuttavia, il dataset finora descritto e fornito per lo svolgimento di questo task è caratterizzato da tutta una serie di problemi e imperfezioni che, al suo stato attuale, lo rendono del tutto inutilizzabile su una qualunque versione di YOLO, cosa che per altro ha preso del tempo prezioso non indifferente, prima dell'inizio dell'effettiva sperimentazione sul modello.

Le procedure sviluppate al fine di rendere il dataset utilizzabile verranno indicate nel relativo **Capitolo 7**, dedicato al codice.

2.3.1 Etichettatura non corretta

Come già affermato, la sintassi di etichettatura supportata da una qualsiasi versione di YOLO è del seguente tipo:

```
<class x y width height>
```

dove:

- class è un intero che indica l'**identificativo** della classe individuata nella bounding box, da 0 a #class-1, dove #class è il numero di classi individuate sul dataset;
- x e y sono le **coordinate** posizionali della bounding box;
- width e height sono le **dimensioni** della bounding box.

È molto importante che le ultime due coppie includano valori **normalizzati** nell'intervallo [0, 1], onde evitare errori di non-normalizzazione o di valori “out of bounds”.

Allo stato attuale, il dataset presenta molte etichettature “out of bounds, dovute sia ad una mancanza di normalizzazione, sia ad una mancata sogliatura di valori di poco superiori a 1 o inferiori a 0.

È da segnalare, inoltre, che il subset relativo al Monastero dei Benedettini è dichiarato etichettato su 35 classi (come anche affermato prima, durante la descrizione), ma in realtà vi sono 37 diversi POI, uno per ogni cartella.

Questo indica che elementi appartenenti a **classi diverse**, vengono etichettati come appartenenti alla **stessa classe**, come nel caso che segue:

| | | |
|------------------|---|--|
| Esempio | | |
| Nome | 5.9_ArcocoNFirma: frame_00066 | 6.1_QuadroSantoMazzarino: frame_00033 |
| Etichetta | 8 0.497944078947 0.524122807018 0.994243421053 0.948830409357 | 8 0.4588815789473684 0.3070175438596491 0.28125 0.5964912280701754 |

Inoltre, nonostante il numero di classi individuate, le etichette di validazione presentano identificativi di classe **ben oltre superiori** a quelli individuati (o dichiarati).

Un esempio potrebbe essere rappresentato dall'etichetta del file `Test3.4_1771.jpg`, il quale, nonostante presenti un fotogramma del busto di S. Vaccarini (con identificativo di classe **9**), è etichettato nel seguente modo:

```
dataset > Monastero dei Benedettini > Validation > labels > Test3.4_1771.txt
      1  57 0.01 0.01 0.01 0.01
      2
```

per cui, con un identificativo di classe pari a **57**.

In altri casi, invece, sono presenti delle coordinate **non normalizzate**, causando un'**incongruenza** tra etichettature. Un esempio è rappresentato dal file `Test3.4_3307.jpg`, al quale corrisponde la seguente etichettatura:

```
I: > ML_Dataset > Points Of Interest Recognition > Monastero dei Benedettini > Validation > YOLO_annotations > Test3.4_3307.txt
      1  31 1 111 79 380
      2  31 281 245 378 416
      3  31 506 314 602 469
      4  31 664 368 741 494
      5  31 799 403 858 529
      6  31 956 406 1036 519
```

2.3.2 Struttura delle cartelle non corretta

Nel momento in cui vengono passati i file di training e di validazione, le implementazioni di YOLO si aspetterebbero una struttura ben precisa delle cartelle, nello specifico quella come segue nella pagina successiva.

Training:

- images:
 - frame_001.jpg
 - ...
 - frame_n.jpg
- labels:
 - frame_001.txt
 - ...
 - frame_n.txt

Validation:

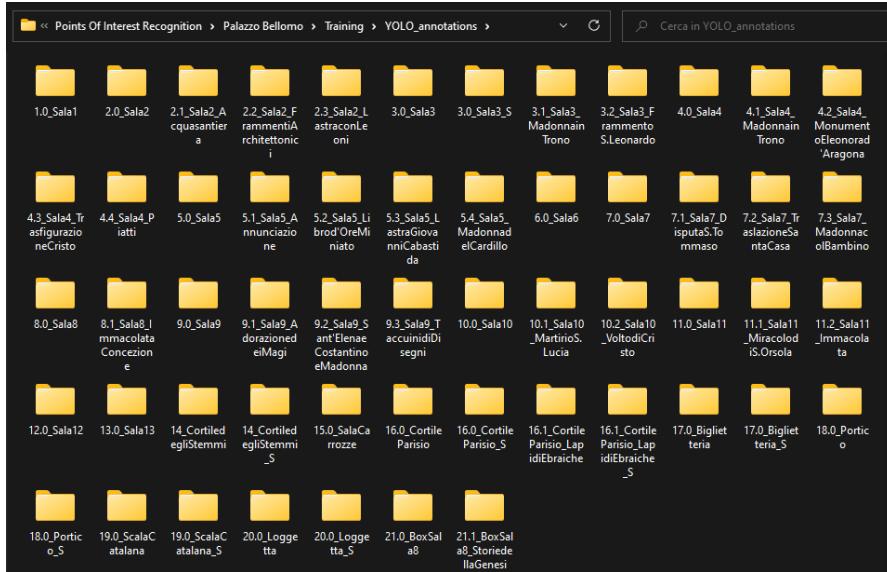
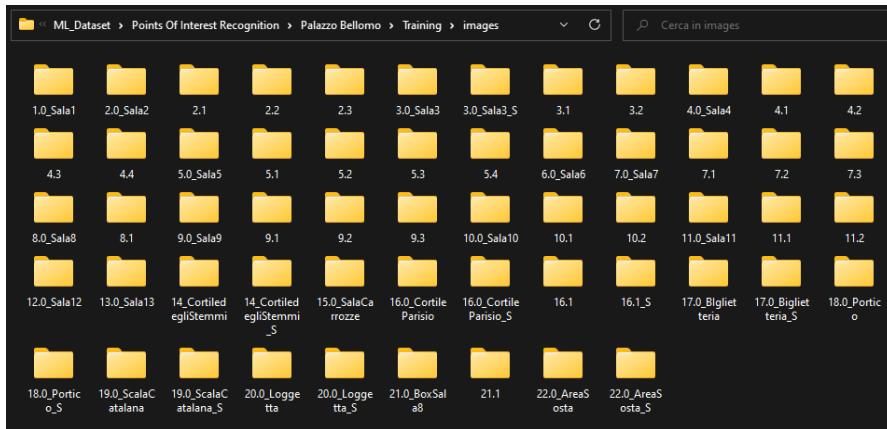
- images:
 - frame_001.jpg
 - ...
 - frame_n.jpg
- labels:
 - frame_001.txt
 - ...
 - frame_n.txt

Detto in parole povere, ad ogni file con estensione .jpg presente nelle cartella images, deve corrispondere un file del **medesimo nome** e con estensione .txt – contenente le coordinate di etichettatura descritte prima – nella cartella labels.

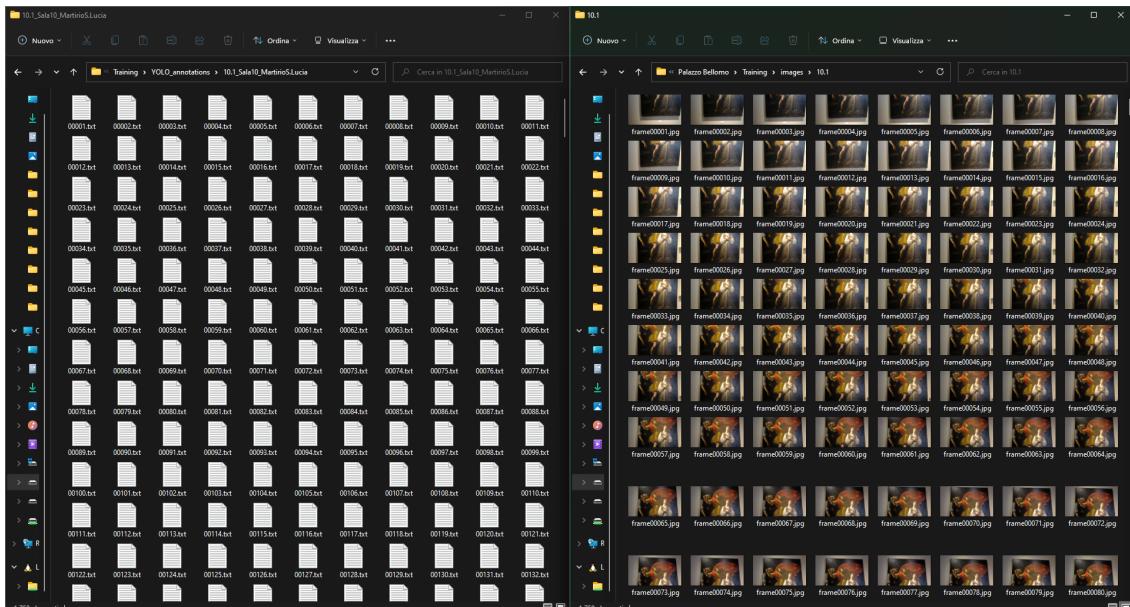
Nel dataset fornito, purtroppo, molte delle etichettature di training del subset relativo al Monastero dei Benedettini sono invece state posizionate nella cartella images. Inoltre, le etichette di validazione, sono posizionate in una cartella nominata YOLO_annotations, piuttosto che in una cartella nominata labels. Entrando in questa cartella, il nome dei file di etichettatura è del tipo “TestN.M XYZK.txt”, invece che “TestN.M_XYZK.jpg” (per cui, il nome dei file immagine di validazione non corrisponde con il nome dei file contenenti le etichette di validazione, per un carattere di **underscore**).

Problemi di questo tipo non mancano neanche nel subset relativo al Palazzo Bellomo: infatti, anch’esso presenta una cartella nominata YOLO_annotations, invece che labels.

Inoltre, osservando la cartella contenente i frame di training per questo sito, è possibile notare come non solo il numero delle sottocartelle presente in images (presumibilmente, una per ogni classe) sia **maggiore** rispetto a quello delle sottocartelle presenti in YOLO_annotations, ma che inoltre tali directory sono state organizzate in maniera da avere nomenclature **totalmente diverse**, come è possibile constatare dagli screenshot presentati di seguito.



È da segnalare, anche in questo subset, l'**incongruenza** nella nomenclatura dei rispettivi file immagine e testo, per ognuna delle sottocartelle. Infatti, sebbene i file debbano condividere lo stesso nome, come già descritto prima, anche in questo caso ciò non si verifica per tutte le cartelle. Di seguito un esempio dimostrativo.



Capitolo 3: Metodo

3.1 Object Recognition

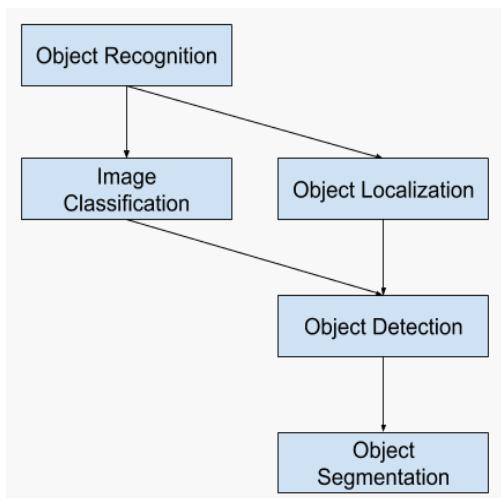
Come affermato nei precedenti paragrafi, il workflow di risoluzione di questo specifico task sarà quello tipico del **object recognition** (o **object detection**). In particolare, verrà utilizzato l'object detector YOLO.

Per “object recognition” si intende la capacità di localizzare – mediante una **bounding box** – e riconoscere – in base alla sua classe di appartenenza – un individuo, un animale, o anche un semplice oggetto ritratto in una singola immagine o in una sequenza di esse; è uno di quei problemi che risultano essere di facile risoluzione per l'uomo, ma di difficile risoluzione per un'**intelligenza artificiale (IA)**, la quale, se non opportunamente addestrata, non può riconoscere uno specifico oggetto e distinguerlo da un altro di diversa entità.

Vista la molitudine di utilizzi ai quali un algoritmo di siffatta specie potrebbe prestarsi (si pensi per esempio alla necessità di conteggio delle auto presenti in un parcheggio, la capacità di avere una copia virtuale e tridimensionalizzata dell'oggetto, o – come in questo caso – il riconoscimento di punti di interesse in un sito culturale, in maniera da prestare in tempo reale un servizio ad un visitatore), la ricerca è particolarmente concentrata su di esso, nel campo della **computer vision**.

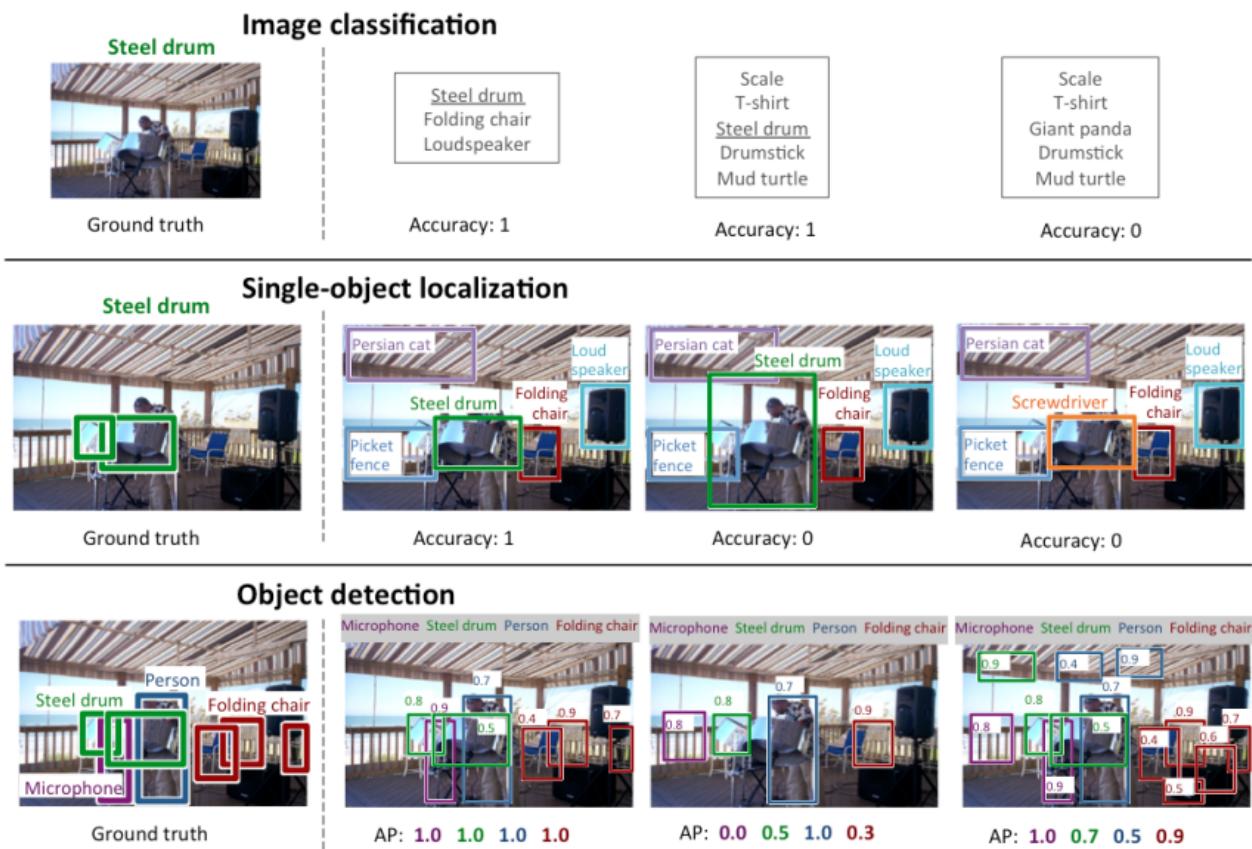
Il workflow di questa tipologia di algoritmo, è quello di estrarre una serie di caratteristiche (o **feature**) da un'immagine – che potrebbe essere una fotografia o un fotogramma proveniente da un filmato – che contraddistinguono l'oggetto in essa ritratto, così da ottenere, in maniera **gerarchica**, una sua **rappresentazione** che possa descriverlo: si parte da enti geometrici di basso livello (come gli edge estratti dell'oggetto estratti dalla fotografia), passando attraverso quelli di medio livello (i suoi contorni), e si ottiene così una rappresentazione di alto livello (la forma finale dell'oggetto) che permetterà all'algoritmo di fornire una risposta al quesito di classificazione.

Tale rappresentazione, se estratta e appresa a partire da un campione di addestramento, può poi essere utilizzata dall'algoritmo per riconoscere il medesimo oggetto, ritratto in un'immagine di test.



Naturalmente, gli algoritmi di object detection si sono evoluti negli anni, al fine di poter affrontare istanze sempre più complesse del medesimo problema di riconoscimento²:

- **classificazione dell'immagine** (2010-2014): l'algoritmo produce una lista di categorie di oggetti presenti nell'immagine;
- **localizzazione del singolo oggetto** (2011-2014): l'algoritmo produce una lista di categorie di oggetti presenti in un'immagine, in relazione ad una bounding box che indica sia la posizione, sia la scala di una singola istanza di oggetto per ciascuna categoria;
- **riconoscimento degli oggetti** (2013-2014): l'algoritmo produce una lista di categorie presenti di oggetti presenti in un'immagine, in relazione ad una bounding box che indica sia la posizione, sia la scala per tutte le istanze di oggetti presenti nell'immagine, per ciascuna categoria.



È anche vero che, in generale, fotografie che ritraggono il medesimo soggetto presentano spesso specifiche dissimili l'una dall'altra: si pensi alla risoluzione, al grado di illuminazione, alla posizione dello stesso rispetto al centro focale dell'obiettivo fotografico, disturbi e/o presenza di altri oggetti (oltre a quello di interesse) nella stessa immagine.

Un buon algoritmo object recognition deve essere in grado di superare tutte queste problematiche, applicando un metodo di riconoscimento che non sia influenzato dalle caratteristiche tecniche dell'immagine; proprio per questo motivo, i metodi maggiormente utilizzati sono quelli invarianti per traslazione, basati sulla **Scale-Invariant Feature Transform (SIFT)**.

² "ImageNet Large Scale Visual Recognition Challenge - arXiv." 1 set. 2014, <https://arxiv.org/abs/1409.0575>. Ultimo accesso: 22 mag. 2022.

Considerando quest'importante caratteristica che gli algoritmi di object detection devono avere, le tecnologie maggiormente utilizzate per la risoluzione di questo tipo di problema sono le reti neurali basate su convoluzione – la quale è un'operazione SIFT –, ossia le **Convolutional Neural Network (CNN)**.

3.2 Convolutional Neural Network

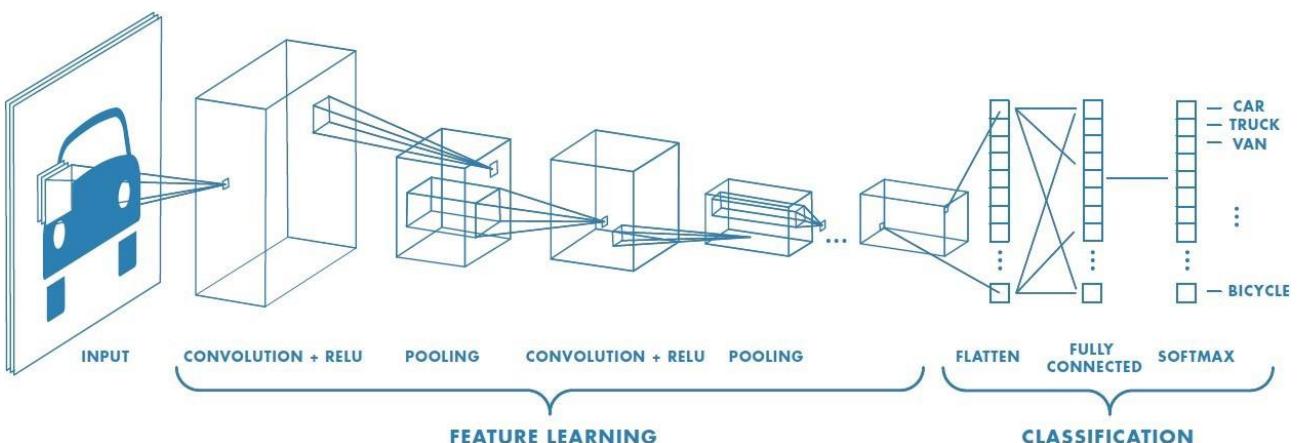
Una CNN è una rete neurale profonda (o **Deep Network**) basata su due tipologie di hidden layer: layer convoluzionali e layer fully-connected.

Le architetture di tipo CNN cercano di simulare il sistema visivo umano, ragion per cui ogni neurone è in grado di computare una limitata regione dell'immagine, denominata receptive field; essi, vengono poi sovrapposti in maniera da ricomporre l'intera immagine.

I **layer convoluzionali** fanno utilizzo dell'operazione di convoluzione per effettuare l'estrazione delle feature dell'input passato al primo layer; ciò permette di avere un processo computazionalmente meno oneroso di una tradizionale moltiplicazione matriciale e allo stesso tempo di tipo SIFT. Infatti, le tradizionali Deep Network sono caratterizzate da un'esplosione di parametri da calcolare, causata dalla moltiplicazione tra matrici: supponendo di avere un'immagine in input di dimensione 100×100 , si dovrebbero calcolare, al primo livello, esattamente 10000 parametri; ai livelli successivi questo numero aumenterà in maniera considerevole. Inoltre, la moltiplicazione matriciale non mantiene la correlazione spaziale tra le varie feature (non essendo di tipo SIFT), causando una lenta convergenza dell'algoritmo.

I pesi calcolati nelle CNN, e nello specifico, sui layer convoluzionali, sono invece dei kernel utili all'estrazione di feature presenti nelle immagini: naturalmente, è più utile fare apprendere alla rete stessa tutti i kernel convolutivi in questione, piuttosto che programmarli ad-hoc per ciascun layer; questa caratteristica, permette non solo di catturare la dipendenza spaziale tra feature, ma anche quella temporale nel caso di sequenze di immagini (come i frame che compongono un filmato).

I **layer fully-connected** sono quei layer che collegano l'output dell'ultimo layer convoluzionale al layer di output (un SoftMax), il quale potrebbe essere composto da un singolo nodo nel caso di una classificazione binaria o una serie di nodi che poi codifica un vettore one-hot-encoding nel caso di una classificazione multiclasse. A partire da quest'ultimo nodo, si avrà la risposta di classificazione del contenuto dell'immagine, o più in generale, dei dati in input.



Le più utilizzate famiglie di CNN nell'ambito della object detection sono le R-CNN e le reti YOLO, descritte di seguito.

Sulla rete R-CNN verrà fatta una breve digressione descrittiva per comprenderne le caratteristiche di base (in quanto il suo utilizzo esula da questo progetto), mentre di YOLO verrà fatta una disamina più accurata, ricalcando la sua evoluzione dalle prime versioni sino alle più recenti, poiché quella utilizzata in questo progetto.

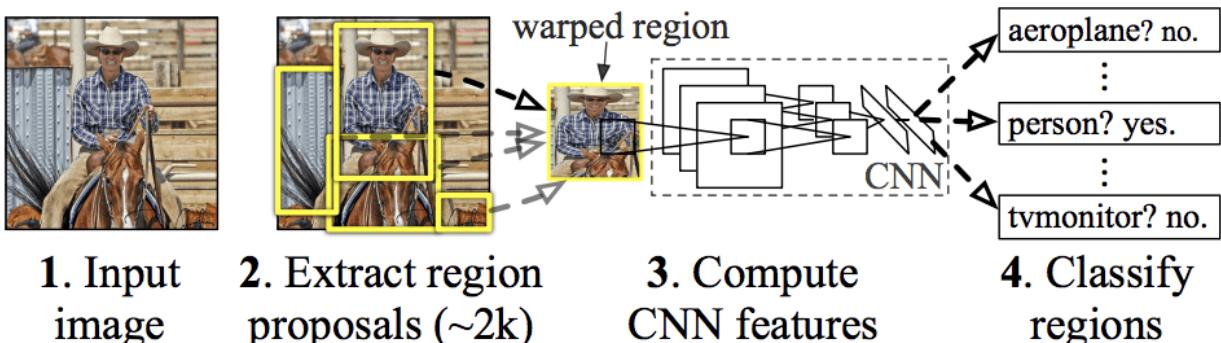
3.3 R-CNN

Le **R-CNN**³ (acronimo per “*Regions with CNN features*”) sono tra le più popolari reti utilizzate nel campo delle object recognition.

Il framework di base di una generica R-CNN si basa sui seguenti blocchi di moduli:

1. **immagine di input;**
2. **region proposal:** estrae circa 2000 regioni dall’immagine di input, in maniera da proporre delle possibili bounding box;
3. **computazione delle feature:** estrazione delle feature per ciascuna regione individuata nel passo precedente, utilizzando un processo identico a quello visto nelle CNN standard. In particolare, per ogni regione vengono estratti dei vettori di feature 4096-dimensionalni;
4. **classificazione delle regioni:** ogni regione viene classificata facendo utilizzo di un **SVM lineare** per ciascuna classe individuata sui dati di training.

R-CNN: *Regions with CNN features*



Nonostante questo framework sia concettualmente semplice, oltre che di successo nelle applicazioni pratiche di riconoscimento e localizzazione di oggetti, esso è anche caratterizzato da una lentezza di convergenza causata proprio dal suo punto di forza: l'estrazione delle feature da ben 2000 regioni per immagine (quelle proposte dall'algoritmo come possibili bounding box al passo 2), come vista in una tradizionale rete CNN.

Ovviamente, col passare degli anni sono state proposte migliorie al modello (quali la **Fast R-CNN**⁴ nel 2015 e la **Faster R-CNN**⁵ nel 2016), rendendolo più veloce rispetto alla sua prima versione e

³ "Rich feature hierarchies for accurate object detection and semantic" <https://arxiv.org/abs/1311.2524>. Ultimo accesso: 22 mag. 2022.

⁴ "[1504.08083] Fast R-CNN - arXiv." 27 set. 2015, <https://arxiv.org/abs/1504.08083>. Ultimo accesso: 22 mag. 2022.

⁵ "Towards Real-Time Object Detection with Region Proposal Networks." 6 gen. 2016, <https://arxiv.org/abs/1506.01497>. Ultimo accesso: 22 mag. 2022.

generalmente più affidabile dell'architettura YOLO, ma comunque non contraddistinto dalla medesima velocità che caratterizza quest'ultimo.

3.4 YOLO

L'architettura **YOLO**⁶ (acronimo per “*You Only Look Once*”), ossia quella utilizzata in questo progetto al fine di poter effettuare il riconoscimento dei sopracitati POI dei due siti, è basato su un'idea profondamente diversa da quella delle reti R-CNN (seppur altrettanto semplice), idea che poi ha contribuito alla sua maggiore velocità di convergenza rispetto a queste ultime.

Mentre infatti R-CNN è basato da una complessa pipeline, difficile da ottimizzare in quanto genera le bounding box a partire da una serie di proposte delle regioni estratte dall'immagine (ciò indica che ogni componente deve essere addestrata separatamente), per poi utilizzare un classificatore su ognuno esse e alla fine cancellare i risultati duplicati, YOLO cambia la prospettiva della object detection, trattandola come un singolo problema di **regressione** (piuttosto che di classificazione): utilizzando questa pipeline, l'immagine verrà osservata solamente una volta (da qui il nome di questa famiglia di modelli) in modo da definire quali oggetti sono presenti all'interno dell'immagine e dove sono localizzati.

Infatti, partendo dall'immagine di input, una singola rete convoluzionale predice simultaneamente multiple bounding box e tutte le probabilità di classe di queste box. Ciò indica che YOLO, a differenza delle R-CNN, viene addestrato a partire dalle immagini di input per intero, e non va a suddividerle in più regioni.

Esso è caratterizzato da una serie di caratteristiche che lo mettono in rilievo rispetto ad altri sistemi di object detection.

Innanzitutto, riformulando il problema in uno di regressione, non necessita di un'architettura o di una pipeline complessa, tanto da garantire un'estrema velocità dell'algoritmo stesso, in grado di processare ad una velocità di 45 frame per secondo (senza batch processing).

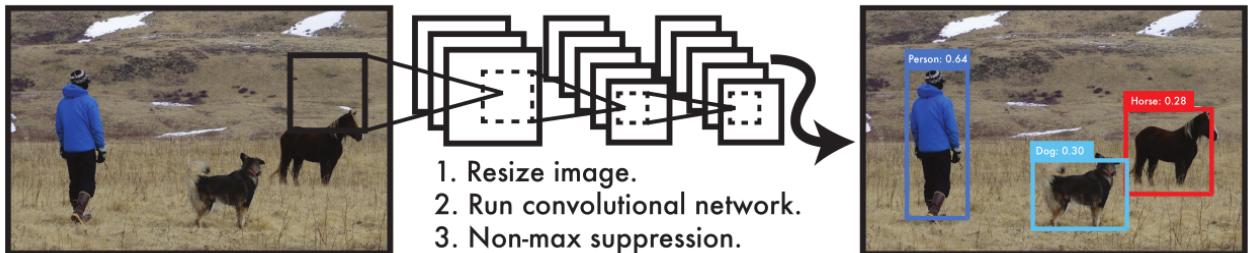
In secondo luogo, poiché osserva l'immagine per intero (e non suddivisa in diverse aree), l'algoritmo è in grado di processare sia informazioni contestuali sulle classi, sia quelle relative al loro aspetto: infatti, mentre i modelli Fast R-CNN commettono parecchi errori sulle sezioni di sfondo di un'immagine (che non possono essere contestualizzate sufficientemente a causa della suddivisione in regioni), YOLO è in grado di commettere una quantità di errori più contenuta, pari a circa la metà di quelli commessi da Fast R-CNN.

Infine, YOLO apprende delle rappresentazioni generalizzabili a partire dalle immagini di training: infatti, esso si mostra in grado di surclassare la stragrande maggioranza di framework, quando addestrato su immagini naturali e testato su opere artistiche; ciò indica che anche la problematica dello **shift di dominio** (o più semplicemente, di input inaspettato) viene facilmente risolta da YOLO, grazie alla sua grande capacità di generalizzare le rappresentazioni estratte dal campione di training.

Il comportamento di una rete di tipo YOLO, a partire di un'immagine di input, può essere sintetizzato come segue:

⁶ "You Only Look Once: Unified, Real-Time Object Detection - arXiv." 9 mag. 2016, <https://arxiv.org/abs/1506.02640>. Ultimo accesso: 22 mag. 2022.

1. l'immagine viene anzitutto scalata ad una dimensione 448×448 ;
2. l'immagine scalata viene passata ad una singola rete convoluzionale;
3. i rilevamenti eseguiti vengono poi sogliati utilizzando il parametro di confidenza del modello.



La prima versione di YOLO è comunque caratterizzata da una più bassa accuratezza rispetto ad altri sistemi di riconoscimento “state-of-the-art”: pur essendo in grado di identificare rapidamente oggetti nelle immagini, arranca tuttavia nel localizzare spazialmente con precisione alcune categorie di oggetti, in particolare quelli di piccola dimensione.

3.4.1 Network Design

Come già affermato in precedenza, YOLO estraе le feature dall'immagine, la quale viene processata per intero, al fine di predire simultaneamente ogni bounding box su tutte le classi.

Affinché ciò sia possibile, l'immagine viene suddivisa in una **griglia** di dimensione $S \times S$: se il centro di un oggetto ricade in una cella della griglia, tale cella diviene la responsabile nel rilevamento di tale oggetto.

Ciascuna cella predice B bounding box e i valori di **confidenza** per esse; queste ultime, indicano la sicurezza secondo la quale il modello afferma che l'oggetto è contenuto in una specifica box, e l'accuratezza di tale predizione: in particolare, la confidenza è definita secondo il prodotto

$$Pr(\text{Object}) * IOU_{\text{pred}}^{\text{truth}}$$

il quale è nullo se nessun oggetto è presente nella cella, ed è pari al prodotto tra la **probabilità a priori** dell'oggetto Object e l'**intersection over-union** tra la box predetta e la ground truth.

Maggiori dettagli su grandezze come l'intersection over-union (IOU) verranno date nel paragrafo successivo, relativo alle misure di valutazione considerate.

Ogni bounding box, inoltre, è caratterizzato da cinque predizioni: x, y, w, h e confidenza.

Le coordinate (x, y) rappresentano il **centro** della box relativo ai limiti spaziali della singola cella, mentre la doppia (w, h) indica **larghezza** ed **altezza** di essa predetta, relativamente all'intera immagine; infine il grado di confidenza predetto rappresenta la IOU tra la box predetta e qualsiasi box di ground truth.

Ogni cella, inoltre, predice C **probabilità a posteriori**

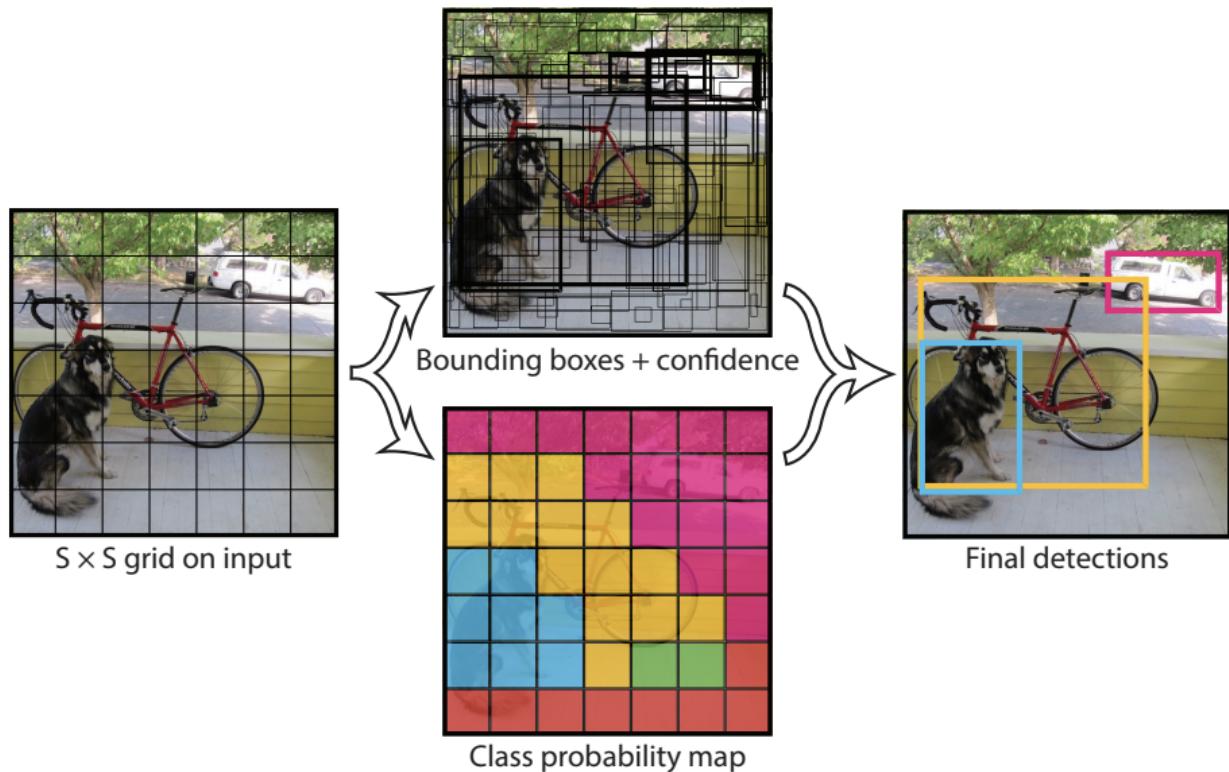
$$Pr(\text{Class}_i | \text{Object})$$

della i -esima classe rispetto all'oggetto Object . Tali probabilità sono condizionate dal se la cella contiene (o meno) l'oggetto in questione: viene esclusivamente calcolato un singolo set di probabilità di classe per ogni cella, indipendentemente dal numero B di bounding box rilevate.

Infine, durante la fase di test, viene moltiplicata la probabilità a posteriori per il grado di confidenza confidenza:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

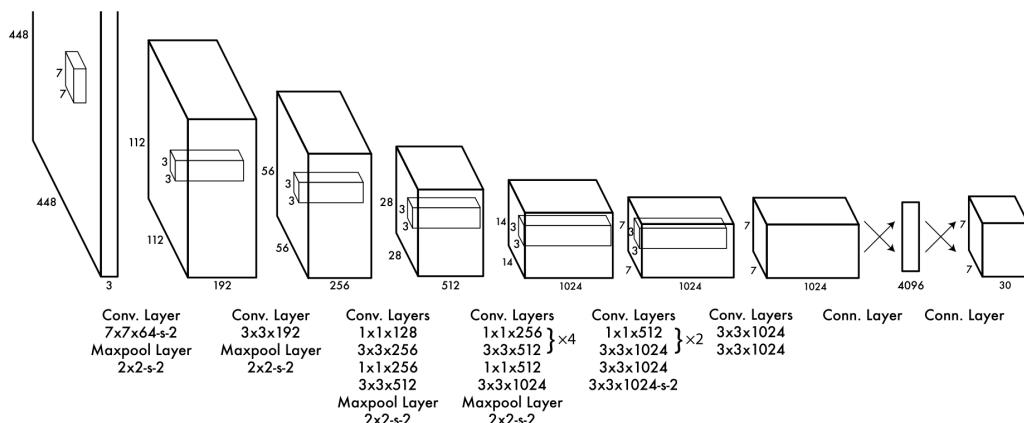
tale prodotto, non può che risultare con il grado di confidenza calcolato su ogni classe per ogni box. Tutti questi punteggi, messi insieme, codificano sia la probabilità che l' i -esima classe sia quella dell'oggetto rilevato nella box, sia quanto meglio la box in questione è in grado di contenere l'oggetto.



Entrando più nello specifico, l'**architettura** del framework YOLO è molto semplice, in quanto formata da **24 layer convoluzionali**, seguiti da due layer **fully connected**: di questi, l'ultimo layer è caratterizzato da una funzione d'attivazione lineare, mentre tutti gli altri layer utilizzano la seguente funzione d'attivazione:

$$\Phi(x) = \begin{cases} x, & \text{se } x > 0 \\ 0.1x & \text{altrimenti} \end{cases}$$

L'output finale della rete consiste in un tensore di predizioni di dimensione $7 \times 7 \times 30$.



3.4.2 Fase di training

Di seguito viene descritta la fase di training di una rete YOLO, così come concepita dai suoi autori. Essa consiste in due sotto-fasi: una di pre-training e una di training effettivo.

FASE DI PRE-TRAINING

Durante questa fase, vengono allenati solamente i **livelli più profondi** della rete: nello specifico, vengono considerati solamente i primi 20 layer convoluzionali, seguiti da un layer di average-pooling ed un layer fully-connected. Secondo quanto affermato dagli autori, applicando un pre-addestramento della durata di circa una settimana utilizzando il dataset della competizione ImageNet 2012, caratterizzato da 1000 classi e con input di dimensione 224×224 , la rete è stata in grado di ottenere un punteggio di accuratezza dell'88%.

FASE DI TRAINING COMPLESSIVO

Superata questa prima fase, gli autori hanno poi rimodellato la rete in maniera da poter effettuare la detection: in particolare, dopo aver rimosso gli ultimi due livelli della rete (aggiunti al solo fine di effettuare il pre-allenamento) sono stati aggiunti quattro layer convoluzionali e due layer fully connected con pesi inizializzati casualmente; inoltre, considerando che il rilevamento di oggetti richiede delle informazioni visive ben precise, al fine di aumentare il grado di accuratezza è stata raddoppiata la dimensione dell'input, passando quindi a immagini di dimensione 448×448 .

In questa seconda fase, la rete è stata allenata per circa 135 epoch per training e validation dataset PASCAL VOC 2007 e VOC 2012: più in particolare, quando testata sulla versione 2012, sono anche stati inclusi i dati di test della versione 2007 per l'allenamento; inoltre, sono stati utilizzati come parametri una **batch size** pari a 64, un **momentum** pari a 0.9 e un **decay rate** pari a 0.0005.

Per quanto riguarda il **learning rate**, esso è stato inizialmente impostato a un valore pari a 10^{-3} , per poi aumentare lentamente durante le prime epoch fino ad arrivare a 10^{-2} : infatti, partendo da un alto learning rate, il modello è spesso caratterizzato da una divergenza dovuta all'instabilità dei gradienti. L'attuale valore di learning rate viene così mantenuto stabile per 75 epoch, fino a riscendere a 10^{-3} per 30 epoch e giungere infine a 10^{-4} per le ultime 30 epoch.

Naturalmente, sono stati anche effettuati degli accorgimenti per evitare l'overfitting del modello: anzitutto, è stato applicato un **dropout** del 50% dopo il primo layer fully connected, per evitare il co-adattamento tra i layer; inoltre, è stata effettuata una **data augmentation**, applicando ridimensionamenti e traslazioni fino al 20% delle immagini originale, unita ad un aumento casuale dell'esposizione e della saturazione nello spazio di colore HSV, fino ad un fattore pari a 1.5.

Il layer finale è in grado di predire sia le probabilità di classe che le coordinate e caratteristiche della bounding box, come già affermato in precedenza, per cui è qui necessario normalizzare altezza e larghezza della box rispetto all'altezza e la larghezza dell'immagine stessa; inoltre, le coordinate della bounding box vengono parametrizzate in modo da essere offset di una specifica cella.

3.4.3 Funzione di Loss

Nel corso dell’addestramento, viene ottimizzata la seguente funzione di **Loss**:

$$\begin{aligned}
 & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_i^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \\
 & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{w}_i - \sqrt{\hat{w}}_i)^2 + (\sqrt{h}_i - \sqrt{\hat{h}}_i)^2] + \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \\
 & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 + \\
 & + \sum_{i=0}^{S^2} 1_{ij}^{noobj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

nella quale 1_i^{obj} indica se l’oggetto compare nella cella i , e 1_{ij}^{obj} indica se la j -esima bounding box di predizione nella cella i è “responsabile” di tale stima.

Si noti che la funzione di Loss penalizza l’errore di classificazione esclusivamente nel caso in cui un oggetto è presente in tale cella (e qui si spiega il motivo della probabilità a posteriori della classe rispetto all’oggetto, citata nel paragrafo precedente); inoltre, essa penalizza l’errore delle coordinate della bounding box solamente se tale predittore è responsabile della box di ground truth (quindi, nel caso in cui esso abbia il più alto predittore in tale cella).

3.4.4 Limitazioni

Uno dei principali problemi di YOLO, sta nell’imporre forti **vincoli spaziali** sulle predizioni delle bounding box, considerando che una ogni cella può predire esclusivamente due box e può avere una sola classe: tale vincolo, limita il numero di oggetti tra essi vicini in grado di essere rilevati dal modello, soprattutto se piccoli (un classico esempio potrebbe essere quello del rilevamento degli uccelli in uno stormo).

Inoltre, poiché il modello cerca di predire le bounding box dai dati, esso **fatica nel generalizzare** nel caso di oggetti con configurazioni o proporzioni inaspettate.

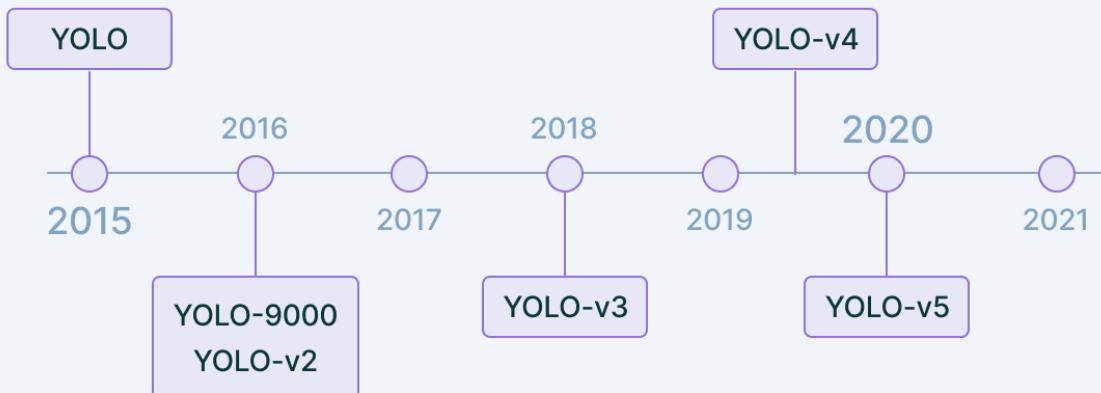
Infine, la funzione di Loss **tratta gli errori alla medesima maniera**, a prescindere che siano causati da grandi o piccole bounding box: in genere infatti, un piccolo errore in una box grande è benigno, ma un piccolo errore su una box piccola può avere effetti negativi considerevoli sulla IOU.

3.4.5 Ulteriori versioni di YOLO

Quella che è stata descritta finora è la prima versione di YOLO, la quale da qui in poi verrà nominata sotto l’appellativo di **YOLOv1**, onde evitare confusioni.

Nel corso degli anni, infatti, sono state sviluppate cinque ulteriori versioni (non tutte frutto dell’originale autore, Joseph Redmon, il quale ha pubblicato esclusivamente le prime tre) che hanno migliorato le performance dell’intero modello, rendendolo più versatile.

YOLO timeline



YOLOv2

Nota anche con l'appellativo di **YOLO9000**⁷, questa versione è stata rilasciata un anno dopo la prima, dagli stessi autori di YOLOv1. Il suo nome è dovuto alla sua capacità di riuscire a rilevare persino 9000 diverse categorie di oggetti.

Questo nuovo rilascio dipende dall'aver realizzato che, rispetto ad una Fast R-CNN, YOLOv1 commette un significativo numero di errori di localizzazione, dal quale dipende la sua minor Recall rispetto ad un modello basato su proposta di regioni. Proprio per questo motivo, l'architettura non è stata totalmente stravolta – volendo i suoi autori mantenere la velocità di classificazione, da sempre punto di vanto di questo framework –, ma è stata semplificata al fine di avere un apprendimento delle feature a sua volta più semplice: ciò ha permesso di avere un framework più evoluto che mantenesse l'accuracy di classificazione che caratterizzava YOLOv1, pur migliorando la Recall e la localizzazione degli oggetti.

Di seguito, le migliorie apportate rispetto alla prima versione.

- **Normalizzazione dei batch:** aggiungere la normalizzazione dei batch su ogni layer convoluzionale ha permesso di ottenere miglioramenti nella convergenza del modello; infatti, mediante questo stratagemma si è riusciti ad evitare l'overfitting pur rimuovendo il dropout dal modello, e riuscendo ad ottenere così un miglioramento in mAP del 2% rispetto alla prima versione.
- **Classificatore ad alta risoluzione:** ribadendo quanto detto in precedenza relativamente alla prima versione, YOLOv1 allena la rete di classificazione con input di dimensione 224×224 , per poi aumentare la risoluzione a 448×448 in virtù del riconoscimento degli oggetti presenti nell'immagine; ciò significa che la rete deve simultaneamente passare dalla fase di addestramento, a quella di aggiustamento dei parametri sugli input ad alta risoluzione. Onde evitare questo processo, si è stabilito di addestrare la rete di classificazione della versione 9000 (per dieci epoche), mediante ImageNet, direttamente sugli input di dimensione 448×448 . Questo stratagemma, dona alla rete il tempo

⁷ "[1612.08242] YOLO9000: Better, Faster, Stronger - arXiv." 25 dic. 2016, <https://arxiv.org/abs/1612.08242>. Ultimo accesso: 27 mag. 2022.

necessario ad aggiustare i filtri, in modo da processare in maniera più efficace gli input di risoluzione più alta; infine, viene ri-tarata la rete risultante in fase di localizzazione. Questa classificazione ad alta risoluzione ha permesso di ottenere un miglioramento in mAP del 4% rispetto alla versione originale.

- **Convoluzioni con anchor box:** ispirandosi a reti come Faster R-CNN, basati quindi su proposta delle regioni, YOLOv2 introduce le anchor box, ossia bounding box pre-definite che vengono ri-aggiustate nel corso dell'addestramento della rete; tali bounding box vengono prestabilite utilizzando un clustering k-means sul dataset di training. La rete viene quindi ridimensionata per operare su input di dimensione 416×416 , invece che 448×448 : questo, al fine di ottenere numero dispari di celle nella griglia, anziché pari; infatti, ciò permette di avere una singola cella centrale (gli oggetti tendono spesso a stare al centro dell'immagine). Utilizzando questa nuova strategia di definizione delle bounding box, piuttosto che di predizione, è stata ridotta la mAP, ma guadagnando in una Recall pari al 88%.
- **Fine-Grained Features:** poiché la versione originale di YOLO è caratterizzata da errori di classificazione su oggetti di piccola dimensione, si è stabilito di utilizzare mappe di feature di dimensione 13×13 , così da avere una griglia più ampia per un miglior riconoscimento degli oggetti più piccoli. L'approccio è quello di partire da una feature map di dimensione $26 \times 26 \times 512$ per poi trasformarla in un tensore di dimensione $13 \times 13 \times 2048$, il quale è un'ulteriore feature map che può essere concatenata alla feature map originale.
- **Allenamento multi-scala:** con l'aggiunta delle anchor box, YOLO prende in input immagini di dimensione 416×416 , ma essendo preferibile un'architettura multi-scala, funzionante correttamente su qualsiasi dimensione dell'immagine in ingresso, invece di ridimensionare quest'ultima nei layer convoluzionali, si è deciso di far variare la rete dopo un set variabile di iterazioni. In particolare, ogni 10 batch, la rete sceglie in maniera casuale un nuovo valore per ridimensionarsi, e poiché viene fatto un downsampling con fattore pari a 32, tale dimensione viene estratta dal seguente insieme di multipli di 32: $\{320, 352, \dots, 608\}$. Ciò porta ad avere una dimensione minima possibile pari a 320×320 , ed una dimensione massima possibile pari a 608×608 . Tale regime permette di avere una rete più versatile, in grado di predire facilmente al variare di diverse dimensioni di input.

| | YOLO | | | | | | | | YOLOv2 |
|----------------------|------|------|------|------|------|------|------|------|-------------|
| batch norm? | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| hi-res classifier? | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| convolutional? | | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| anchor boxes? | | | | ✓ | ✓ | | | | |
| new network? | | | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| dimension priors? | | | | | | ✓ | ✓ | ✓ | ✓ |
| location prediction? | | | | | | ✓ | ✓ | ✓ | ✓ |
| passthrough? | | | | | | | ✓ | ✓ | ✓ |
| multi-scale? | | | | | | | | ✓ | ✓ |
| hi-res detector? | | | | | | | | | ✓ |
| VOC2007 mAP | 63.4 | 65.8 | 69.5 | 69.2 | 69.6 | 74.4 | 75.4 | 76.8 | 78.6 |

- **Darknet-19:** questa nuova versione di YOLO introduce un nuovo modello di classificazione, chiamato Darknet-19, che diviene la sua base portante; tale classificatore, è costituito maggiormente da filtri di dimensione 3×3 , e presenta un numero di raddoppiato di canali dopo ogni fase di pooling (di tipo max), rispetto all'architettura originale. In particolare, sono presenti 19 layer convoluzionali (da cui il nome) e 5 layer di maxpool (com'è possibile vedere nella figura di seguito).

| Type | Filters | Size/Stride | Output |
|---------------|---------|----------------|------------------|
| Convolutional | 32 | 3×3 | 224×224 |
| Maxpool | | $2 \times 2/2$ | 112×112 |
| Convolutional | 64 | 3×3 | 112×112 |
| Maxpool | | $2 \times 2/2$ | 56×56 |
| Convolutional | 128 | 3×3 | 56×56 |
| Convolutional | 64 | 1×1 | 56×56 |
| Convolutional | 128 | 3×3 | 56×56 |
| Maxpool | | $2 \times 2/2$ | 28×28 |
| Convolutional | 256 | 3×3 | 28×28 |
| Convolutional | 128 | 1×1 | 28×28 |
| Convolutional | 256 | 3×3 | 28×28 |
| Maxpool | | $2 \times 2/2$ | 14×14 |
| Convolutional | 512 | 3×3 | 14×14 |
| Convolutional | 256 | 1×1 | 14×14 |
| Convolutional | 512 | 3×3 | 14×14 |
| Convolutional | 256 | 1×1 | 14×14 |
| Convolutional | 512 | 3×3 | 14×14 |
| Maxpool | | $2 \times 2/2$ | 7×7 |
| Convolutional | 1024 | 3×3 | 7×7 |
| Convolutional | 512 | 1×1 | 7×7 |
| Convolutional | 1024 | 3×3 | 7×7 |
| Convolutional | 512 | 1×1 | 7×7 |
| Convolutional | 1024 | 3×3 | 7×7 |
| Convolutional | 1000 | 1×1 | 7×7 |
| Avgpool | | Global | 1000 |
| Softmax | | | |

YOLOv3

YOLOv3⁸ vede la luce nel 2018, due anni dopo la versione 9000. In realtà, come affermato dagli stessi autori, questa versione del modello è «nulla di eccezionalmente interessante, giusto un paio di piccoli cambiamenti per migliorarlo». Principalmente, i cambiamenti apportati rispetto alla precedente versione sono frutto di «buone idee partorite da altra gente», alle quali è stata aggiunta una nuova rete di classificazione, a detta degli autori migliore delle precedenti.

Di seguito vengono riassunte le migliorie apportate rispetto a YOLO9000.

- **Predizione delle bounding box:** seguendo l'eredità di YOLOv2, questa terza versione utilizza un sistema di predizione delle bounding box basato su quattro coordinate: t_x , t_y , t_w , t_h . Se la cella ha un offset di (c_x, c_y) rispetto all'angolo in alto a sinistra dell'immagine, e la

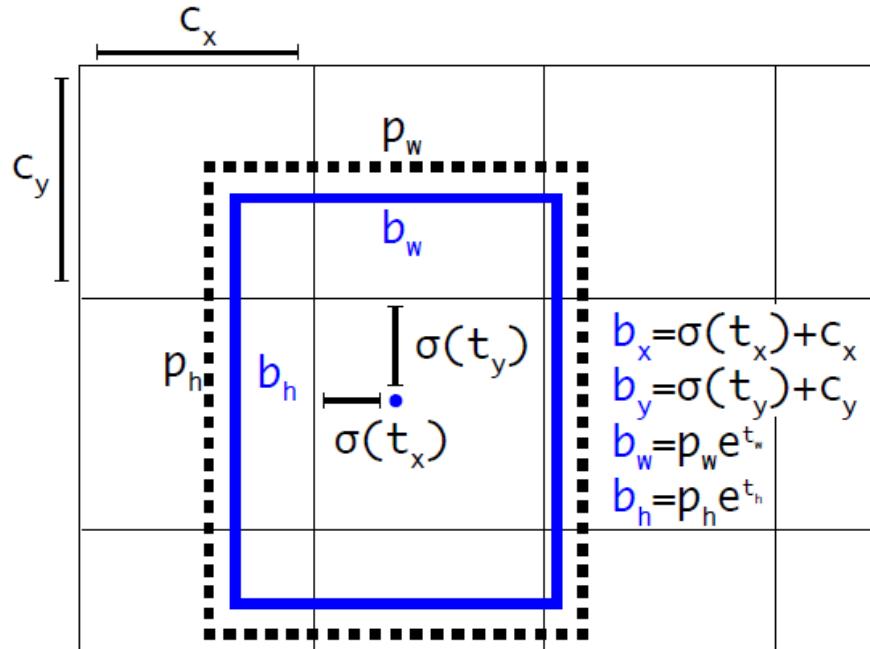
⁸ "[1804.02767] YOLOv3: An Incremental Improvement - arXiv." 8 apr. 2018, <https://arxiv.org/abs/1804.02767>. Ultimo accesso: 29 mag. 2022.

bounding box precedente ha larghezza e altezza rispettivamente pari a p_w e p_h , allora la predizione dei quattro valori corrisponde a:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x & b_w &= p_w e^{t_w} \\ b_y &= \sigma(t_y) + c_y & b_h &= p_h e^{t_h} \end{aligned}$$

Durante l'allenamento viene utilizzata la somma dell'errore quadratico: se il valore di ground truth per alcune coordinate stimate è pari a \hat{t}_* , il gradiente è dato dalla differenza della ground truth e la predizione: $\hat{t}_* - t_*$. Il valore di ground truth è facilmente calcolato invertendo le equazioni sopra mostrate.

YOLOv3 predice anche uno score, mediante la regressione logistica, che indica la “**objectness**” per ogni bounding box: tale valore è pari ad 1 se la bounding box predetta si sovrappone a quella di ground truth più di ogni altra bounding box; se invece questa sovrapposizione non è verificata, allora la predizione viene ignorata. In particolare, viene effettuata una sogliatura con threshold pari a 0.5.



- **Predizione delle classi:** ogni box predice la classe che potrebbe essere contenuta al suo interno mediante una classificazione multiesemplare. Per effettuare quest'operazione viene utilizzata una serie di semplici classificatori logistici, piuttosto che un Softmax, in quanto quest'ultimo approccio è parso essere quello computazionalmente più costoso e allo stesso tempo non necessario; la funzione di Loss utilizzata durante l'addestramento consiste in una cross-entropy binaria sulle predizioni delle classi.
- **Predizioni a diverse dimensioni:** questa nuova versione introduce la predizione di box su tre diverse dimensioni; in particolare, il modello estrae le feature prendendo ispirazione dalle reti piramidali. Partendo dal feature extractor di base di YOLO (quello esaminato nei precedenti paragrafi), sono stati aggiunti diversi layer convoluzionali, l'ultimo dei quali calcola un tensore tridimensionale di predizione, il quale codifica la bounding box in quanto tale, la “objectness” e la predizione della classe individuata al suo interno. Successivamente,

presa la feature map dai due layer precedenti, il suo numero di campioni viene raddoppiato; questa viene poi concatenata ad un’ulteriore mappa di feature presa da layer precedenti. Questo metodo permette di avere informazioni semantiche più significative. Vengono poi aggiunti due ulteriori layer convoluzionali per processare la mappa di feature ottenuta, e si calcola infine un ulteriore tensore tridimensionale, le cui dimensioni però sono adesso raddoppiate rispetto a quello esaminato precedentemente.

Per determinare le “proposte” di bounding box, viene ancora utilizzato il clustering k-means, seguendo l’eredità della precedente versione.

- **Estrattore di feature (Darknet-53):** come già accennato, questa terza versione introduce una nuova rete di classificazione. Essa si presenta come un’ibridazione tra la Darknet-19 introdotta in YOLO9000 ed una rete residuale. Poiché queste aggiunte hanno reso la rete molto più estesa rispetto alla precedente (con ben 53 layer convoluzionali), essa è stata battezzata sotto il nome di Darknet-53. Riuscendo a migliorare l’utilizzo della GPU, questa nuova rete di classificazione si è dimostrata essere non solo più veloce della precedente Darknet, ma anche delle ResNet-101/152.

| Type | Filters | Size | Output | | | | |
|---------------|---------|------------------|------------------|--|--|--|--|
| Convolutional | 32 | 3×3 | 256×256 | | | | |
| Convolutional | 64 | $3 \times 3 / 2$ | 128×128 | | | | |
| 1x | 32 | 1×1 | | | | | |
| Convolutional | 64 | 3×3 | | | | | |
| Residual | | | 128×128 | | | | |
| Convolutional | 128 | $3 \times 3 / 2$ | 64×64 | | | | |
| 2x | 64 | 1×1 | | | | | |
| Convolutional | 128 | 3×3 | | | | | |
| Residual | | | 64×64 | | | | |
| Convolutional | 256 | $3 \times 3 / 2$ | 32×32 | | | | |
| 8x | 128 | 1×1 | | | | | |
| Convolutional | 256 | 3×3 | | | | | |
| Residual | | | 32×32 | | | | |
| Convolutional | 512 | $3 \times 3 / 2$ | 16×16 | | | | |
| 8x | 256 | 1×1 | | | | | |
| Convolutional | 512 | 3×3 | | | | | |
| Residual | | | 16×16 | | | | |
| Convolutional | 1024 | $3 \times 3 / 2$ | 8×8 | | | | |
| 4x | 512 | 1×1 | | | | | |
| Convolutional | 1024 | 3×3 | | | | | |
| Residual | | | 8×8 | | | | |
| Avgpool | | Global | | | | | |
| Connected | | 1000 | | | | | |
| Softmax | | | | | | | |

| Backbone | Top-1 | Top-5 | Bn Ops | BFLOP/s | FPS |
|------------|-------------|-------------|--------|-------------|------------|
| Darknet-19 | 74.1 | 91.8 | 7.29 | 1246 | 171 |
| ResNet-101 | 77.1 | 93.7 | 19.7 | 1039 | 53 |
| ResNet-152 | 77.6 | 93.8 | 29.4 | 1090 | 37 |
| Darknet-53 | 77.2 | 93.8 | 18.7 | 1457 | 78 |

YOLOv4

Rilasciata da Alexey Bochkovskiy nel 2019, **YOLOv4**⁹ si dimostra essere un’ottimizzazione della versione precedente, in quanto facente utilizzo dello stato dell’arte “Bag of Freebies” e di alcuni “Bag of Specials”.

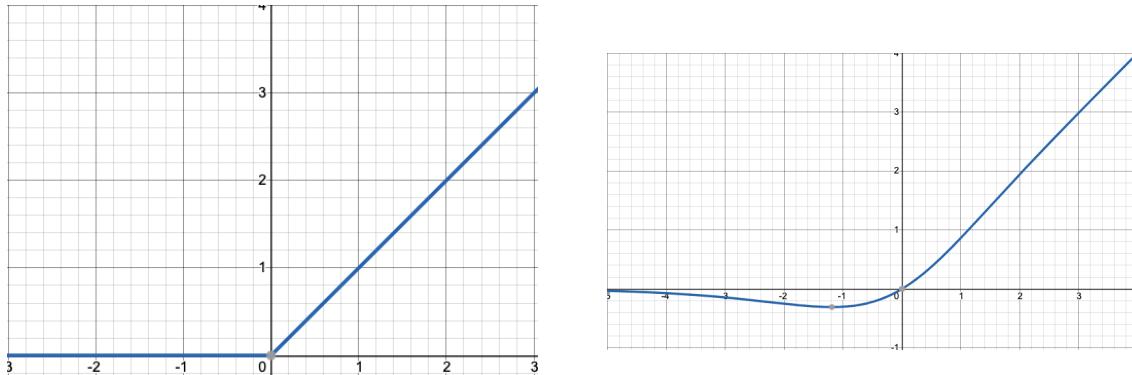
- **Bag of Freebies (BoF):** consiste nello scibile di metodi che fanno variare la strategia di addestramento (o il suo costo), al fine di migliorare l’accuratezza dello stesso, senza tuttavia modificare il tempo di inferenza.

In particolare, i metodi utilizzati in questa nuova versione sono:

⁹ "YOLOv4: Optimal Speed and Accuracy of Object Detection - arXiv." 23 apr. 2020, <https://arxiv.org/abs/2004.10934>. Ultimo accesso: 29 mag. 2022.

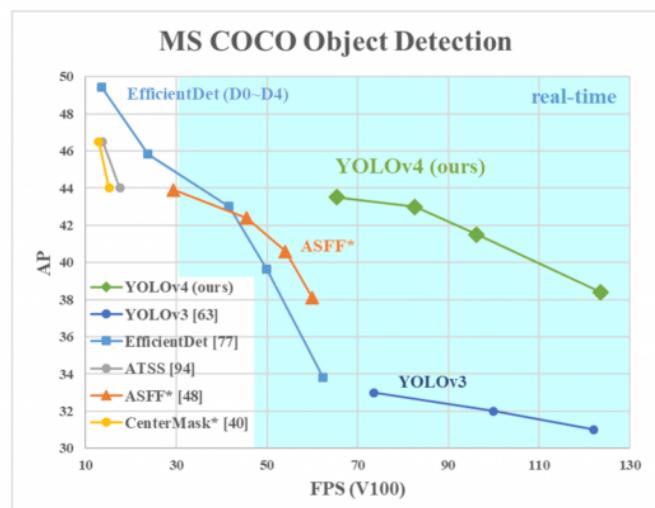
- **Data Augmentation;**
- risoluzione della **distribuzione semantica delle bias** (in particolare, utilizzando uno smoothing dell'etichettatura, nel caso di distribuzioni di classi sbilanciate);
- introduzione di una **funzione obiettivo** nella regressione delle bounding box: mentre il tradizionale rilevamento di oggetti fa utilizzo del **Mean Square Error (MSE)** al fine di effettuare una regressione al centro della box stessa, causando così un aumento della Loss basato esclusivamente sulla scala dell'oggetto (e di conseguenza, della bounding box stessa), l'approccio basato su **IOU** migliora significativamente le prestazioni in quanto viene considerata la superficie di sovrapposizione tra la box stimata e quella effettiva.
- **Bag of Specials (BoS):** consiste in una serie di plugin che aumentano lievemente il tempo di inferenza, ma garantiscono una migliore accuratezza del rilevatore.
In particolare, è stata sostituita la funzione di attivazione ReLU con una di tipo **Mish**. Mentre la derivazione della ReLU durante la discesa del gradiente, a causa della sua definizione, potrebbe portare ad un mancato aggiornamento dei neuroni (si ricorda che la ReLU restituisce 0 se presi in input dei valori negativi), la funzione Mish permette di avere valori non nulli.
Essa è definita come di seguito. È anche presentata una comparativa grafica tra la ReLU (a sinistra) e la Mish (a destra).

$$f(x) = x * \tan(\text{softplus}(x)) = x * \tanh(\ln(1 + e^x))$$



Questa nuova versione, anch'essa basata su Darknet, è stata in grado di ottenere un valore di AP pari al 43.5% sui dati del dataset COCO, con una velocità in real-time pari a 65 fps, surclassando così i più accurati e veloci rilevatori di oggetti del periodo.

Se comparata alla sua versione precedente, l'incremento è pari al 10% relativamente all'AP, e al 12% relativamente al framerate.



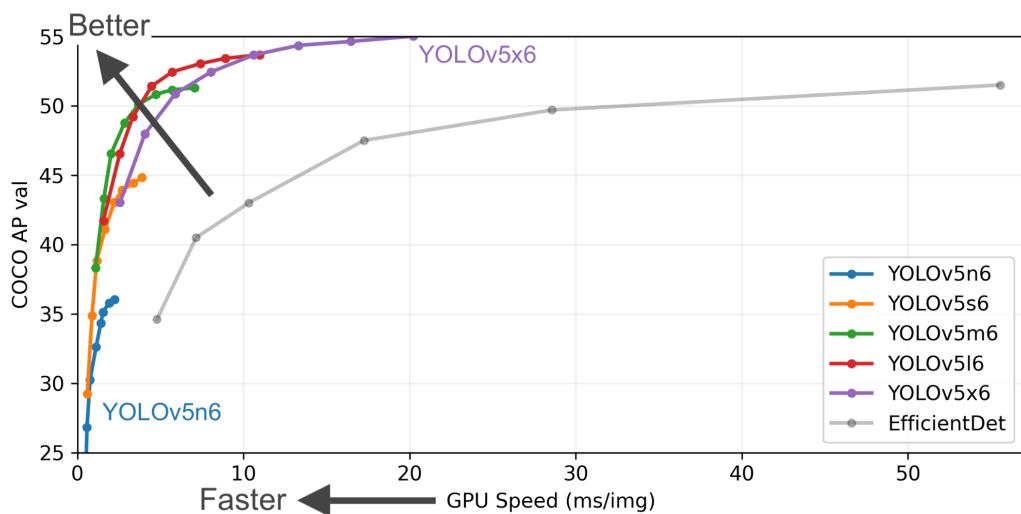
YOLOv5

Di rapida gestazione (appena due mesi dalla pubblicazione di YOLOv4), **YOLOv5¹⁰** viene pubblicata da Glenn Jocher su GitHub, il quale però non fornisce un reale paper di pubblicazione. Questa versione è profondamente differente dalle precedenti, in quanto si presenta essere un'implementazione per PyTorch, piuttosto che un fork dell'originale Darknet.

I miglioramenti apportati includono la **mosaic data augmentation** e l'**auto-apprendimento delle anchor box**.

L'autore ha rilasciato sul suo repository GitHub¹¹ una serie di modelli YOLOv5, tutti pre-addestrati sul dataset COCO, al fine di poter rendere più accessibile il framework.

| Pretrained Checkpoints | | | | | | | | | |
|------------------------|------------------|--------------------------------|---------------------------|-------------------------|--------------------------|---------------------------|---------------|-------------------|--|
| Model | size (pixels) | mAP _{val} 0.5:0.95 | mAP _{val} 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed V100 b32 (ms) | params (M) | FLOPs @640 (B) | |
| YOLOv5n | 640 | 28.0 | 45.7 | 45 | 6.3 | 0.6 | 1.9 | 4.5 | |
| YOLOv5s | 640 | 37.4 | 56.8 | 98 | 6.4 | 0.9 | 7.2 | 16.5 | |
| YOLOv5m | 640 | 45.4 | 64.1 | 224 | 8.2 | 1.7 | 21.2 | 49.0 | |
| YOLOv5l | 640 | 49.0 | 67.3 | 430 | 10.1 | 2.7 | 46.5 | 109.1 | |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | 4.8 | 86.7 | 205.7 | |
| YOLOv5n6 | 1280 | 36.0 | 54.4 | 153 | 8.1 | 2.1 | 3.2 | 4.6 | |
| YOLOv5s6 | 1280 | 44.8 | 63.7 | 385 | 8.2 | 3.6 | 12.6 | 16.8 | |
| YOLOv5m6 | 1280 | 51.3 | 69.3 | 887 | 11.1 | 6.8 | 35.7 | 50.0 | |
| YOLOv5l6 | 1280 | 53.7 | 71.3 | 1784 | 15.8 | 10.5 | 76.8 | 111.4 | |
| YOLOv5x6 + TTA | 1536 | 55.0 | 72.7 | 3136 | 26.2 | 19.4 | 140.7 | 209.8 | |



¹⁰ "YOLOv5 Documentation." <https://docs.ultralytics.com/>. Ultimo accesso: 29 mag. 2022.

¹¹ "YOLOv5 in PyTorch > ONNX > CoreML > TFLite - GitHub." <https://github.com/ultralytics/yolov5>. Ultimo accesso: 29 mag. 2022.

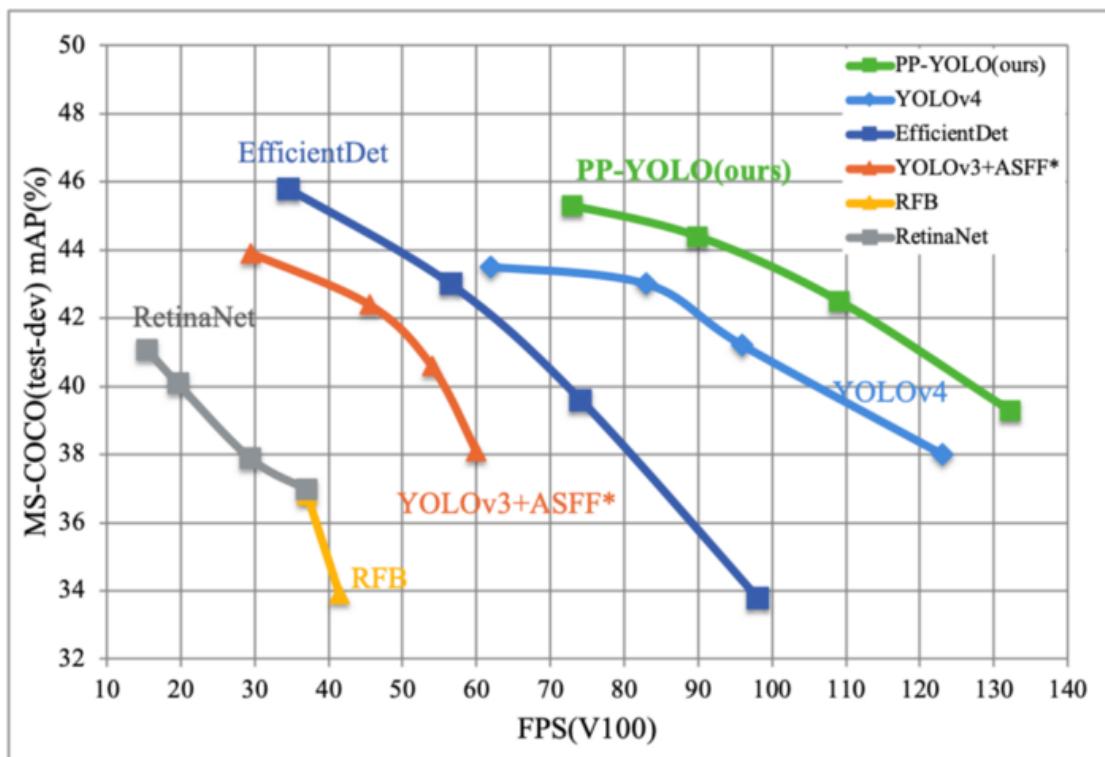
PP-YOLO

Fork della terza versione, **PP-YOLO**¹² viene rilasciato il 23 luglio del 2020 dalla Baidu Inc.

In realtà, secondo quanto affermato dagli autori, il fine ultimo di questa versione è quello di ottenere un «rilevatore di oggetti che abbia un'efficacia relativamente bilanciata e un'efficienza tale da poter essere applicato direttamente a scenari applicativi reali, piuttosto che proporre un nuovo modello di rilevamento». Infatti, le variazioni rispetto a YOLOv3 sono sì superficiali, ma comunque degne di nota a causa della maggior ottimizzazione rispetto a quest'ultimo.

In particolare, i maggiori cambiamenti apportati sono la sostituzione della rete di classificazione Darknet-53 con una **ResNet**, e l'aumento della dimensione del batch di training da 64 a 192.

PP-YOLO è in grado di ottenere un mAP del 45.2% sul dataset COCO, superando così il 43.5% ottenuto da YOLOv4. Quando testato sul dataset TeslaV100 con dimensione del batch pari a 1, PP-YOLO è stata in grado ottenere una velocità di inferenza pari a 72.9 fps, surclassando ancora YOLOv4, caratterizzata da una velocità di inferenza pari a 65 fps.



¹² "PP-YOLO: An Effective and Efficient Implementation of Object Detector." 23 lug. 2020, <https://arxiv.org/abs/2007.12099>. Ultimo accesso: 29 mag. 2022.

Capitolo 4: Valutazione

Si descrivono di seguito le metriche di valutazione utilizzate nel corso degli esperimenti, e citate nel precedente capitolo, durante la trattazione dei vari modelli.

4.1 Matrice di confusione e metriche di classificazione

Valutare un classificatore esclusivamente in base al numero di record classificati correttamente può apparire impreciso, e può fornire un'idea incompleta e inesatta sul reale comportamento del modello, rispetto ai dati di training e di test.

Al fine di studiare in maniera completa la classificazione eseguita su un set di dati, si fa utilizzo della **matrice di confusione**: essa distingue **esempi positivi (P)** ed **esempi negativi (N)** in base a due sotto-categorie, ossia valori reali e predetti.

In particolare, tale tabella di dimensione 2×2 , presenta sulle righe la quantità di esempi positivi e di esempi negativi in quanto tali, mentre sulle colonne è presente la medesima distinzione, ma effettuata dal classificatore mediante una predizione.

La tabella può essere presentata nel seguente modo.

| | Elementi stimati P | Elementi stimati N |
|-----------------------------|---------------------------|---------------------------|
| Elementi di natura P | TP | FN |
| Elementi di natura N | FP | TN |

In base alla loro posizione riga-colonna, ritroviamo i seguenti valori:

- **TP (True Positive)**: numero di elementi di natura positiva correttamente classificati come positivi;
- **FN (False Negative)**: numero di elementi di natura positiva erroneamente classificati come negativi;
- **FP (False Positive)**: numero di elementi di natura negativa erroneamente classificati come positivi;
- **TN (True Negative)**: numero di elementi di natura negativa correttamente classificati come negativi.

Questi valori potrebbero già fornire una rozza misura di valutazione statistica del classificatore, ma è sempre preferibile utilizzarle al fine di ottenere ulteriori metriche ben più robuste, delle quali si fornisce qui una descrizione.

4.1.1 Accuracy

$$ACCURACY = \frac{TP+TN}{TP+TN+FP+FN}$$

L'**Accuracy** (o **accuratezza**) misura la percentuale di elementi correttamente classificati dal modello, rispetto al totale numero di elementi presenti nel dataset utilizzato.

È sempre importante osservare questa misura tenendo in considerazione il **bilanciamento** del dataset (facilmente visibile direttamente dalla tabella di confusione): infatti, potrebbe essere possibile avere un'alta Accuracy, ma solo in quanto si stanno classificando correttamente solo gli elementi negativi.

4.1.2 Precision

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

La **Precision** (o **precisione**) misura la percentuale di elementi classificati positivi, rispetto al totale numero di elementi classificati come positivi.

In pratica, indica la percentuale di elementi stimati positivi che sono effettivamente positivi.

4.1.3 Recall

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

La **Recall** (o **sensibilità**) misura la percentuale di elementi classificati positivi, rispetto al totale numero di elementi realmente positivi individuati nel dataset.

In pratica, indica la percentuale di elementi positivi scovati dal modello.

4.1.4 F1-score

$$F1 = 2 * \frac{PRECISION*RECALL}{PRECISION+RECALL}$$

F1-score è un indicatore che riassume e bilancia le due misure di Precision e Recall precedentemente definite.

4.2 Average Precision

L'**Average Precision (AP)** è una metrica che individua il valore dell'area sottesa alla curva di Precision-Recall: essendo ambedue i valori definiti in $[0, 1]$, essa si può calcolare mediante l'integrale, definito in tale intervallo, della funzione $p(r)$ che esprime la Precision al variare della Recall.

$$AP = \int_0^1 p(r)dr$$

Poiché la funzione $p(r)$ definisce una curva a **dente di sega**, nella pratica scomoda da trattare, è possibile calcolare l'AP mediante un'interpolazione a undici punti: inizialmente, si suddivide il dominio della Recall in 10 sottointervalli, sfruttando infatti gli 11 punti: $\{0, 0.1, \dots, 0.9, 1.0\}$. Successivamente, si calcola il valore medio della Precision per ognuno di essi.

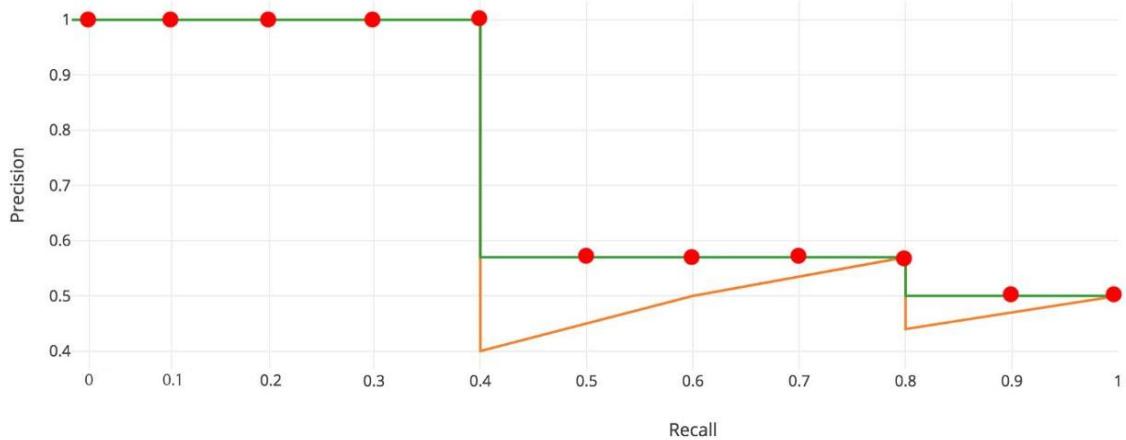
$$AP = \frac{1}{11} * (AP_r(0) + AP_r(0.1) + \dots + AP_r(1.0))$$

Più formalmente, si ha:

$$AP = \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} AP_r = \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} p_{interp}(r)$$

dove:

$$p_{interp}(r) = \max_{r \geq \bar{r}} p(\bar{r}).$$



4.3 Mean Average Precision

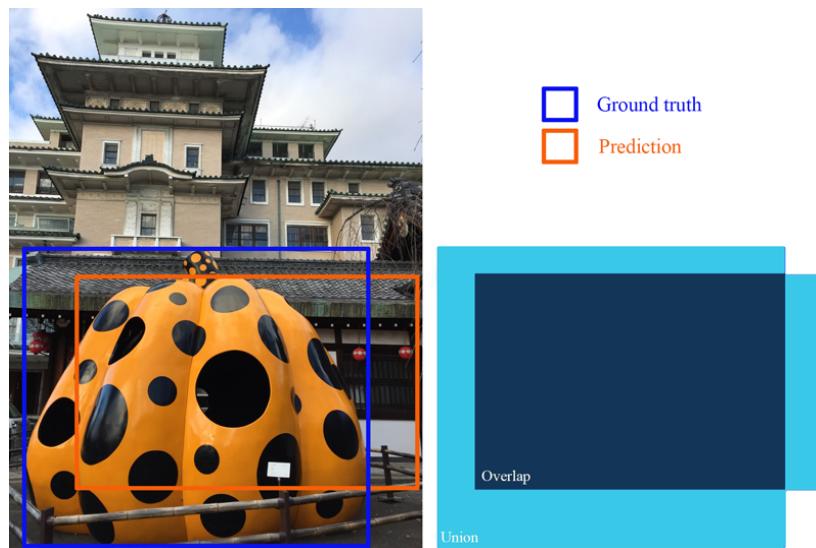
La **mean Average Precision (mAP)** è la media dell'Average Precision calcolata su ogni classe. In particolare, siano n le classi totali e sia AP_i l'Average Precision calcolata sull' i -esima classe (per $i = 1, \dots, n\}$, si ha:

$$mAP = \frac{1}{n} \sum_{i=1}^n AP_i$$

4.4 Intersection over Union

La **Intersection over Union (IOU)** è una metrica di valutazione tipica della object detection, e misura la sovrapposizione tra la bounding box predetta rispetto a quella effettiva dell'oggetto. In particolare, sia GT l'area della bounding box di ground truth (quella effettiva), e sia P quella predetta, essa si calcola nel seguente modo.

$$IOU = \frac{\text{area di sovrapposizione}}{\text{area di unione}} = \frac{\text{area}(GT \cap P)}{\text{area}(GT \cup P)}$$



La IOU è spesso utilizzata, nel caso della object detection, come valore di soglia nei confronti di Precision, Recall e mAP. In particolare, fissata una certa soglia di IOU t , se una predizione ha un'area di bounding box maggiore o uguale a tale threshold, viene classificata come TP, altrimenti viene etichettata come FP.

YOLO fa utilizzo delle seguenti due metriche:

- $mAP@0.5$: è il valore di mAP calcolato con una soglia di IOU pari a 0.5;
- $mAP@0.5:0.95$: è la media dei valori di mAP calcolata al variare di una soglia di IOU da 0.5 a 0.95, con step di 0.05.

Capitolo 5: Esperimenti

Si presentano di seguito i vari esperimenti eseguiti sul dataset. Ogni esperimento sarà suddiviso in due sottosezioni, quella relativa al subset *Monastero dei Benedettini*, e quella relativa al subset *Palazzo Bellomo*.

In mancanza di un’adeguata GPU in grado di supportare CUDA, tutti gli addestramenti sono stati effettuati mediante quella offerta dal piano free di **Google Colaboratory**, ossia una **NVIDIA Tesla T4**, con **CUDA 11.2**.

| NVIDIA-SMI 460.32.03 Driver Version: 460.32.03 CUDA Version: 11.2 | | | | | | | |
|---|----------|---------------|------------------|-----------------|----------|------------|--------|
| GPU | Name | Persistence-M | Bus-Id | Disp.A | Volatile | Uncorr. | ECC |
| Fan | Temp | Perf | Pwr:Usage/Cap | Memory-Usage | GPU-Util | Compute M. | MIG M. |
| <hr/> | | | | | | | |
| 0 | Tesla T4 | Off | 00000000:00:04.0 | Off | 0% | Default | N/A |
| N/A | 41C | P8 | 10W / 70W | 3MiB / 15109MiB | | | |

5.1 Tentativi su YOLOv5s

Seguono i primi tentativi effettuati sulla base **YOLOv5s**, versione più “piccola” e rapida di YOLOv5, già pre-allenata su COCO. Tutti i tentativi di sperimentazione sono stati effettuati per **cinque epoch** (vista la carenza di risorse e soprattutto di tempo di utilizzo della GPU offerto dal piano free, considerando che ogni epoca ha impiegato un’abbondante mezz’ora – o anche più – negli esperimenti descritti nei successivi paragrafi), e con i seguenti iperparametri:

- **Learning rate** iniziale: 0.01;
- **Momentum**: 0.937;
- **Weight decay**: 0.0005;
- **Epoche di warmup**: 3;
- **Warmup del momentum**: 0.8;
- **Treshold IOU**: 0.2.

Inoltre, per entrambi i subset, la risoluzione dei frame è stata dimezzata.

L’architettura della rete è mostrata di seguito.

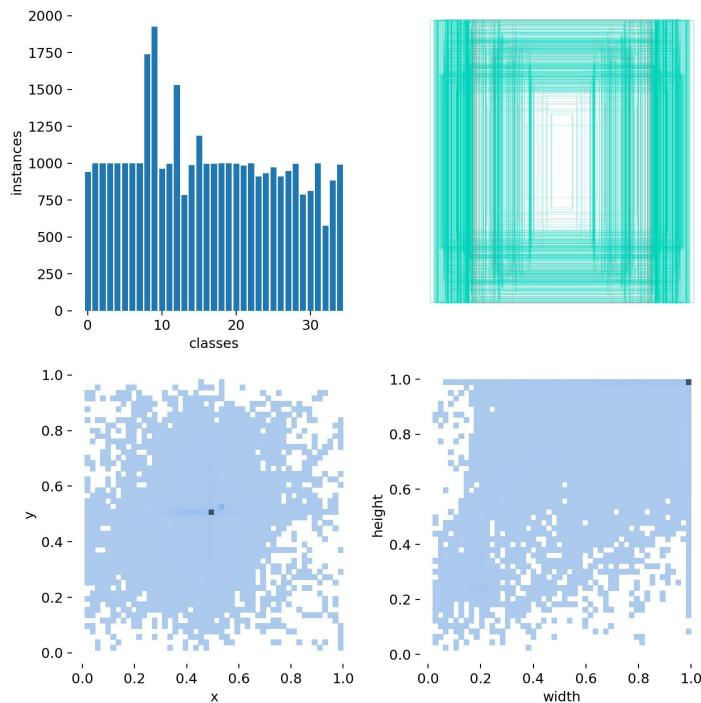
```
      from n  params module           arguments
0       -1 1    3520 models.common.Conv   [3, 32, 6, 2, 2]
1       -1 1   18560 models.common.Conv  [32, 64, 3, 2]
2       -1 1   18816 models.common.C3   [64, 64, 1]
3       -1 1   73984 models.common.Conv  [64, 128, 3, 2]
4       -1 2   115712 models.common.C3  [128, 128, 2]
5       -1 1   295424 models.common.Conv [128, 256, 3, 2]
6       -1 3   625152 models.common.C3  [256, 256, 3]
7       -1 1   1188672 models.common.Conv [256, 512, 3, 2]
8       -1 1   1182720 models.common.C3  [512, 512, 1]
9       -1 1   656896 models.common.SPPF [512, 512, 5]
10      -1 1   131584 models.common.Conv [512, 256, 1, 1]
11      -1 1       0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
12     [-1, 6] 1       0 models.common.Concat [1]
13     -1 1   361984 models.common.C3   [512, 256, 1, False]
14     -1 1   33024 models.common.Conv  [256, 128, 1, 1]
15     -1 1       0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
16    [-1, 4] 1       0 models.common.Concat [1]
17     -1 1   90880 models.common.C3   [256, 128, 1, False]
18     -1 1   147712 models.common.Conv [128, 128, 3, 2]
19     [-1, 14] 1       0 models.common.Concat [1]
20     -1 1   296448 models.common.C3   [256, 256, 1, False]
21     -1 1   590336 models.common.Conv  [256, 256, 3, 2]
22    [-1, 10] 1       0 models.common.Concat [1]
23     -1 1   1182720 models.common.C3  [512, 512, 1, False]
24    [17, 20, 23] 1   113274 models.yolo.Detect [37, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
```

5.1.1 Primo tentativo di training

Il primo tentativo è stato eseguito addestrando tutti i livelli della rete.

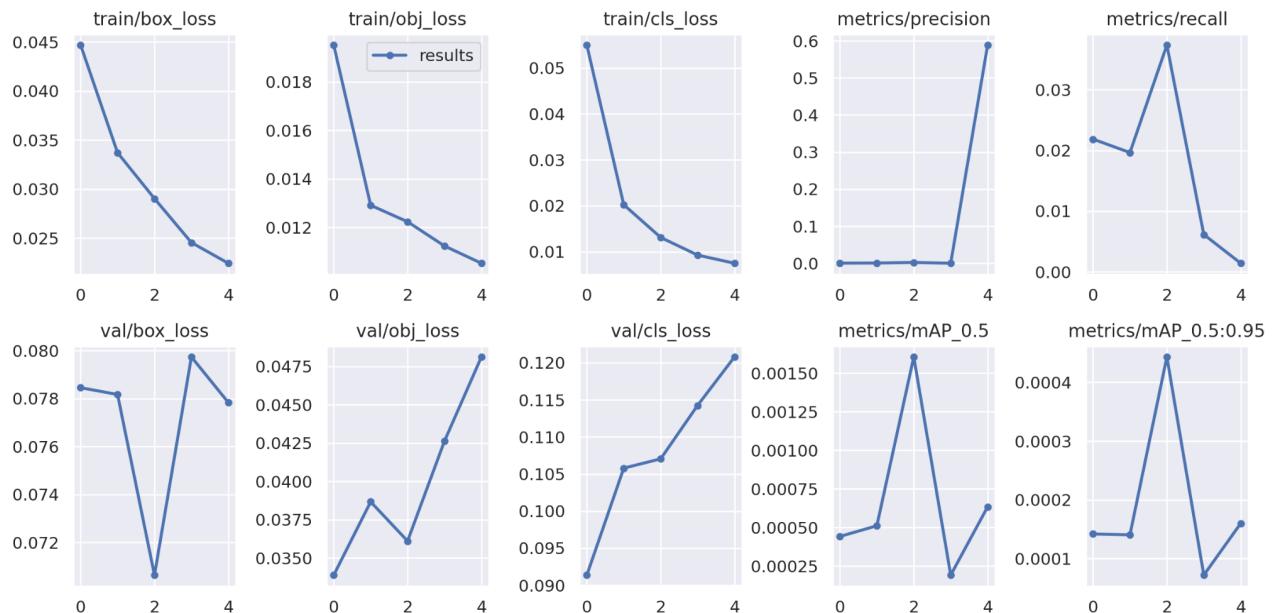
Monastero dei Benedettini

La distribuzione delle etichette è quella che segue:



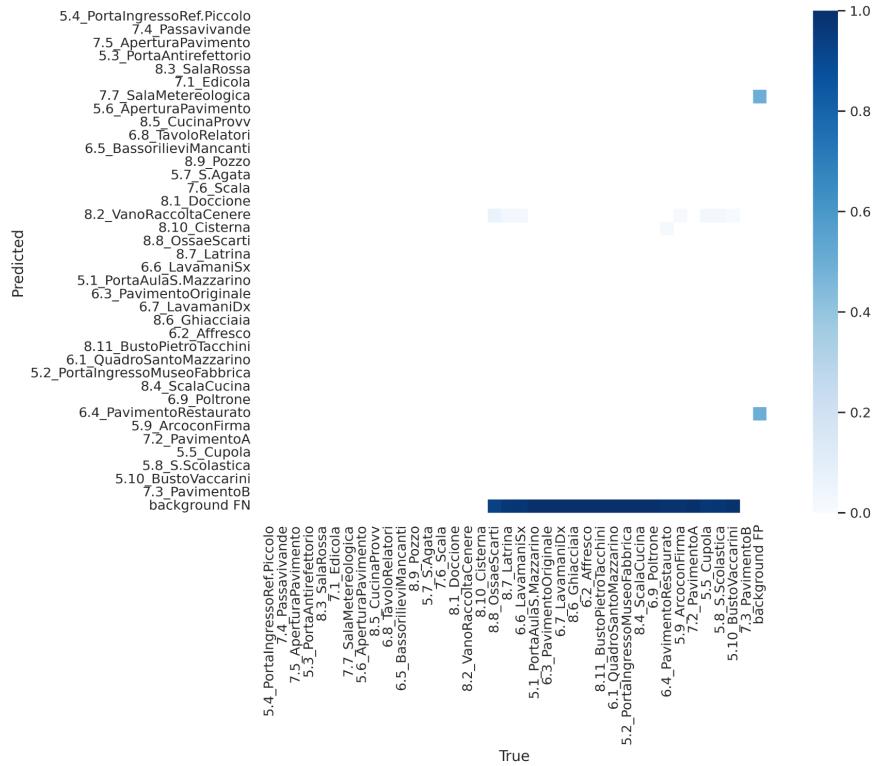
Si noti come le classi sembrano essere abbastanza bilanciate, eccetto che per quattro di esse, le quali superano in numerosità le altre.

In particolare, attraverso questo addestramento sono stati ottenuti i seguenti risultati:



È possibile notare come, a causa dell'aumento delle componenti di Loss in fase di validazione, il modello sia andato in overfitting dopo 5 epoche.

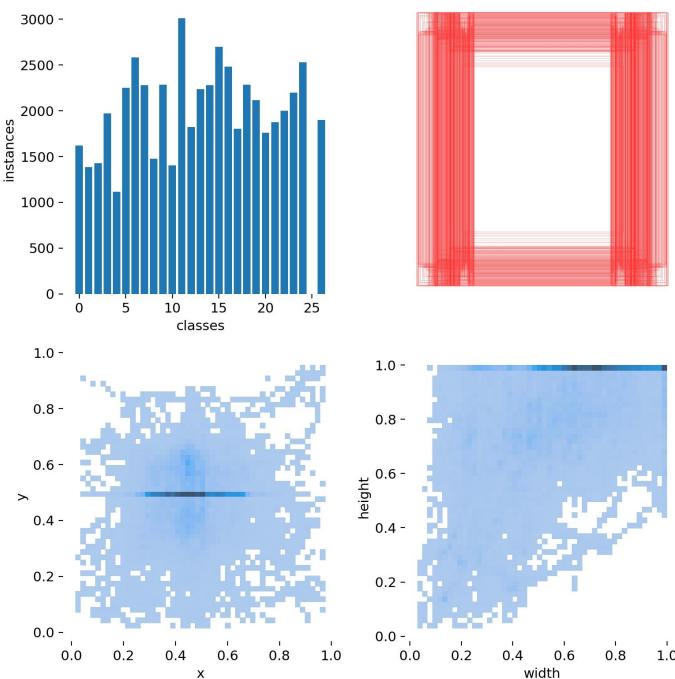
Ciò si può notare anche dalla matrice di confusione, la quale indica la presenza di molti falsi negativi, per il semplice fatto che il modello classifica male.



La mAP assume un valore pari a 0.000633 sull'ultima iterazione, il che indica che può esserci un netto miglioramento del modello addestrato su questo dataset.

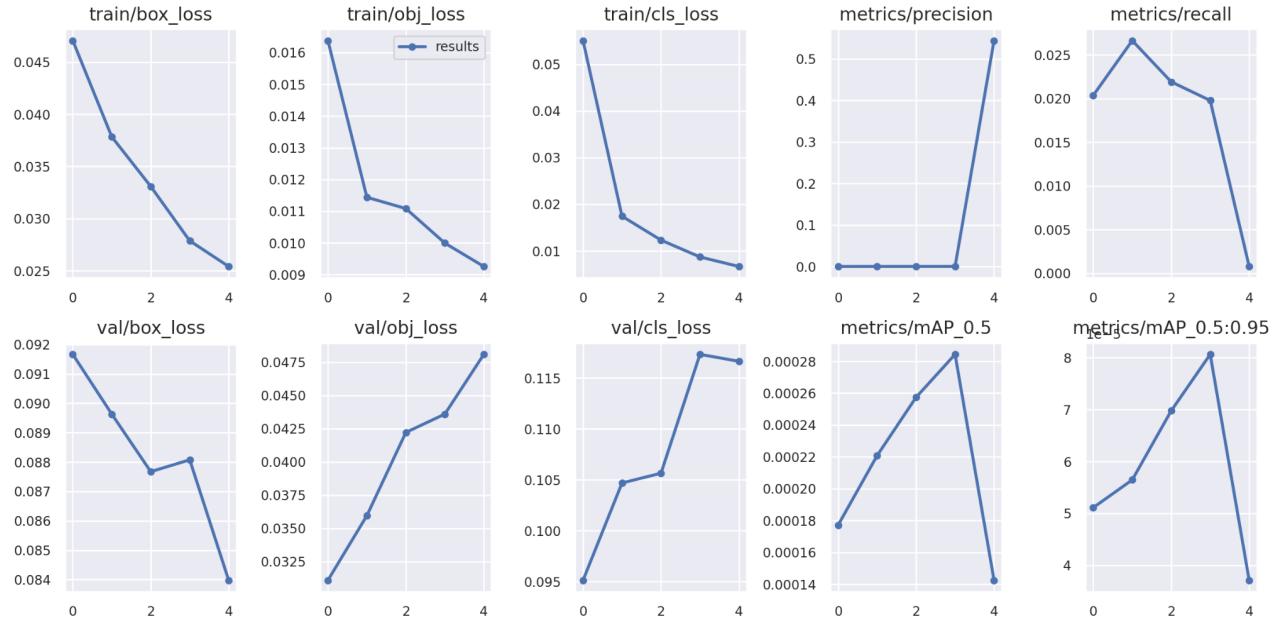
Palazzo Bellomo

Di seguito la distribuzione delle etichette di questo subset:



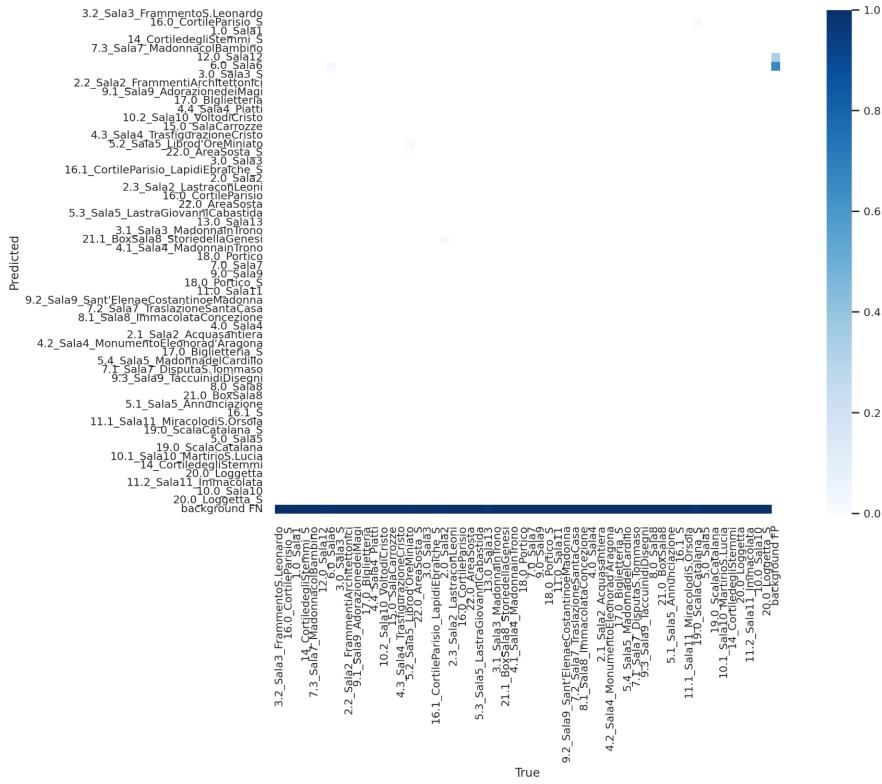
Anche qui le classi sembrano essere abbastanza bilanciate, sebbene YOLO abbia rilevato delle bounding box eccessivamente grandi.

Si mostrano i risultati ottenuti dopo un addestramento di 5 epoch:



Anche qui, possiamo notare come vi potrebbe essere un netto margine di miglioramento, considerati i bassissimi valori delle metriche.

Ciò può essere confermato dalla matrice di confusione risultante:

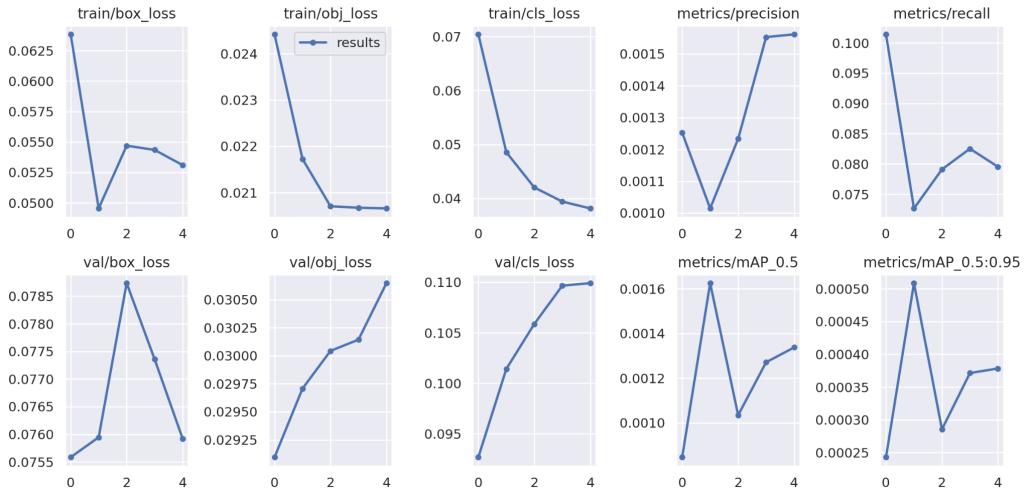


5.1.2 Transfer Learning

Visti gli scarsi risultati ottenuti allenando l'intera rete, si è voluto provare a congelare tutti i layer convolutivi di YOLOv5s (pre-addestrati su COCO) per provare ad addestrare solamente il layer fully-connected, ossia l'ultimo.

Monastero dei Benedettini

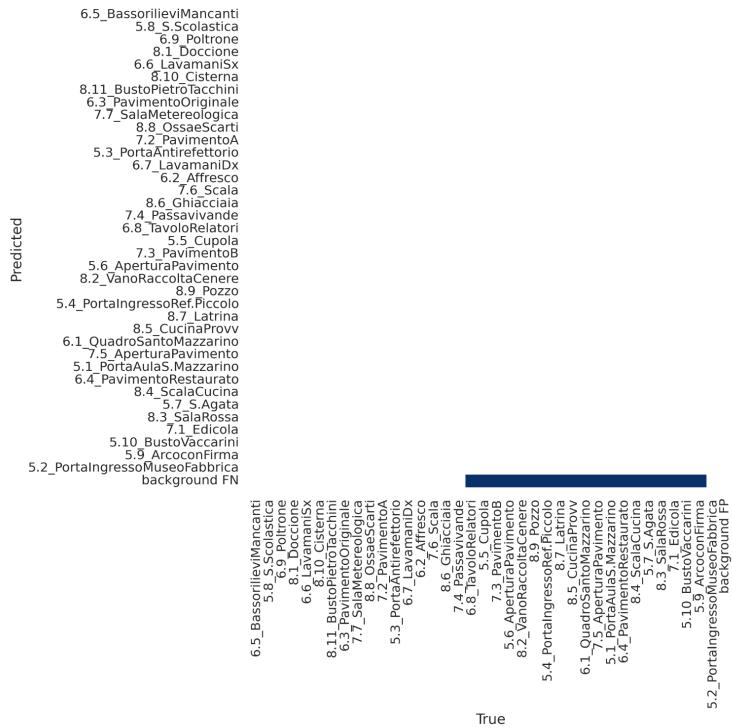
Si mostrano di seguito i risultati ottenuti al seguito di un addestramento, per cinque epoche, esclusivamente sul layer fully-connected.



È possibile notare come non ci siano notevoli differenze rispetto al precedente esperimento. Unica sensibile differenza è data dall'aumento della Precision, la quale sembra avere una crescita meno immediata rispetto a prima.

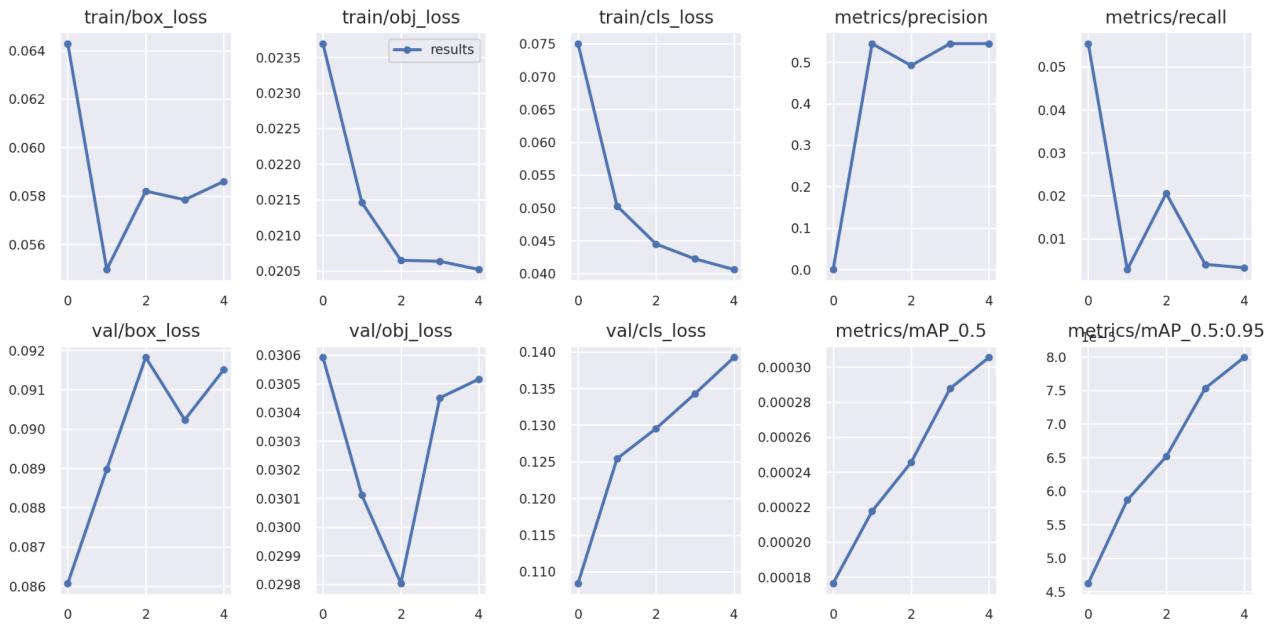
Molto probabilmente ciò è dovuto alla piccola dimensione del modello che, nel caso di un dataset complesso come questo, non è in grado di adattarsi ed apprendere.

Di seguito la matrice di confusione ottenuta.

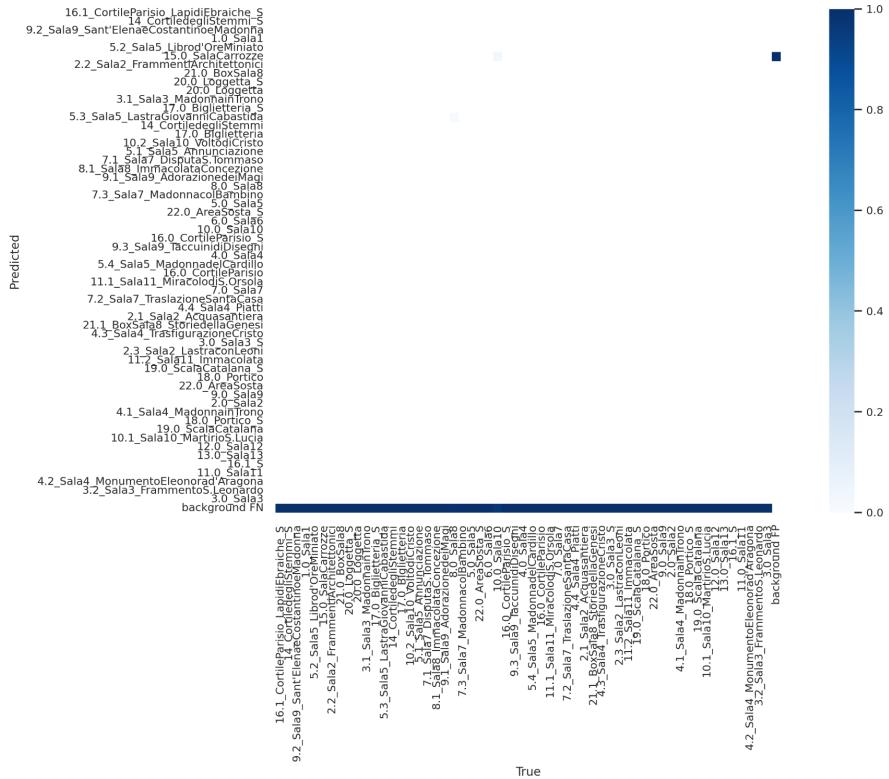


Palazzo Bellomo

Si mostrano seguitamente i risultati ottenuti allenando il classificatore sul subset di Palazzo Bellomo.



È possibile notare come, rispetto al precedente, questo modello addestrato esclusivamente sul layer fully-connected faccia meglio. Il massimo valore di mAP, seppur di poco, è più alto, e inoltre le Loss – pur continuando ad essere in aumento – presentano valori più bassi rispetto a prima. Di seguito si mostra la matrice di confusione ottenuta dalla classificazione.



5.2 Tentativi su YOLOv5m

Viste le scarse performance riscontrate con il modello “small”, si è deciso di effettuare delle sperimentazioni anche sulla versione “medium”, anch’essa pre-addestrata sul dataset COCO e nota col nome di **YOLOv5m**, che presumibilmente dovrebbe generalizzare meglio su un dataset così complesso.

Anche in questo caso, è stato effettuato un addestramento per cinque epochhe e con i medesimi iperparametri utilizzati nella precedente sperimentazione sul modello di dimensione più contenuta. L’architettura della rete è mostrata di seguito.

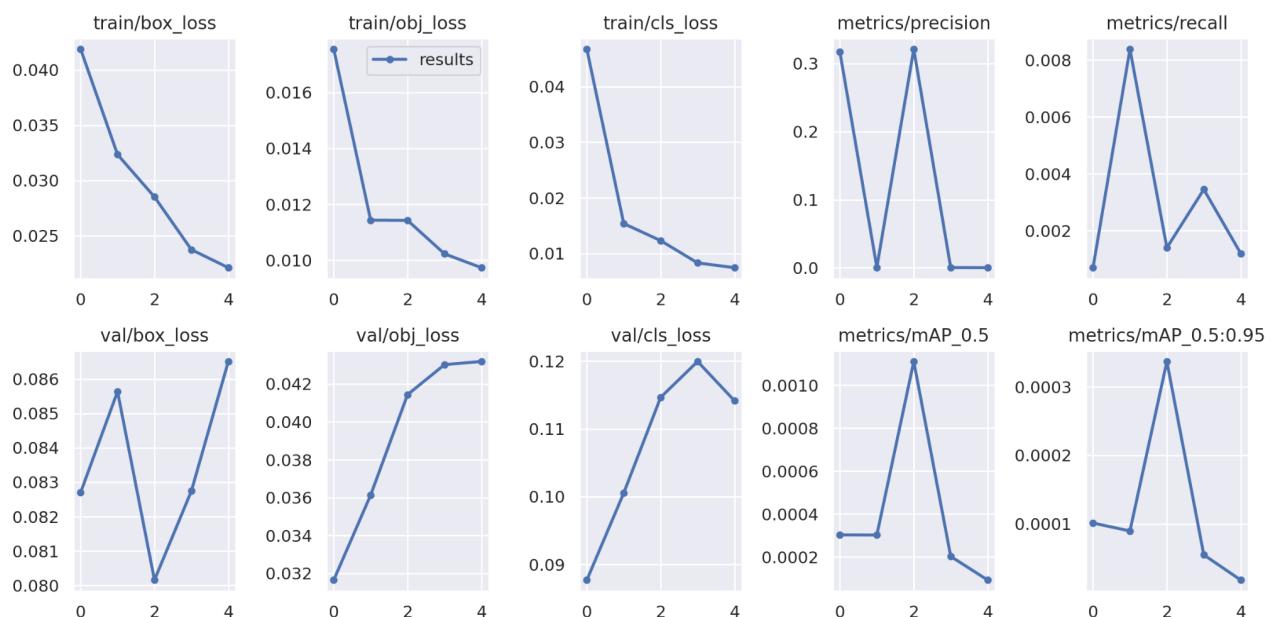
| from | n | params | module | arguments |
|------|--------------|----------|--------------------------------------|---|
| 0 | -1 | 5280 | models.common.Conv | [3, 48, 6, 2, 2] |
| 1 | -1 | 41664 | models.common.Conv | [48, 96, 3, 2] |
| 2 | -1 | 65280 | models.common.C3 | [96, 96, 2] |
| 3 | -1 | 166272 | models.common.Conv | [96, 192, 3, 2] |
| 4 | -1 | 444672 | models.common.C3 | [192, 192, 4] |
| 5 | -1 | 664320 | models.common.Conv | [192, 384, 3, 2] |
| 6 | -1 | 2512896 | models.common.C3 | [384, 384, 6] |
| 7 | -1 | 2655744 | models.common.Conv | [384, 768, 3, 2] |
| 8 | -1 | 4134912 | models.common.C3 | [768, 768, 2] |
| 9 | -1 | 1476864 | models.common.SPPF | [768, 768, 5] |
| 10 | -1 | 295680 | models.common.Conv | [768, 384, 1, 1] |
| 11 | -1 | 0 | torch.nn.modules.upsampling.Upsample | [None, 2, ‘nearest’] |
| 12 | [-1, 6] | 1 | models.common.Concat | [1] |
| 13 | -1 | 21182720 | models.common.C3 | [768, 384, 2, False] |
| 14 | -1 | 74112 | models.common.Conv | [384, 192, 1, 1] |
| 15 | -1 | 0 | torch.nn.modules.upsampling.Upsample | [None, 2, ‘nearest’] |
| 16 | [-1, 4] | 1 | models.common.Concat | [1] |
| 17 | -1 | 296448 | models.common.C3 | [384, 192, 2, False] |
| 18 | -1 | 332160 | models.common.Conv | [192, 192, 3, 2] |
| 19 | [-1, 14] | 1 | models.common.Concat | [1] |
| 20 | -1 | 1035264 | models.common.C3 | [384, 384, 2, False] |
| 21 | -1 | 1327872 | models.common.Conv | [384, 384, 3, 2] |
| 22 | [-1, 10] | 1 | models.common.Concat | [1] |
| 23 | -1 | 21434912 | models.common.C3 | [768, 768, 2, False] |
| 24 | [17, 20, 23] | 1 | models.yolo.Detect | [37, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [192, 384, 768]] |

5.2.1 Primo tentativo di training

Il primo tentativo è stato eseguito addestrando tutti i livelli della rete.

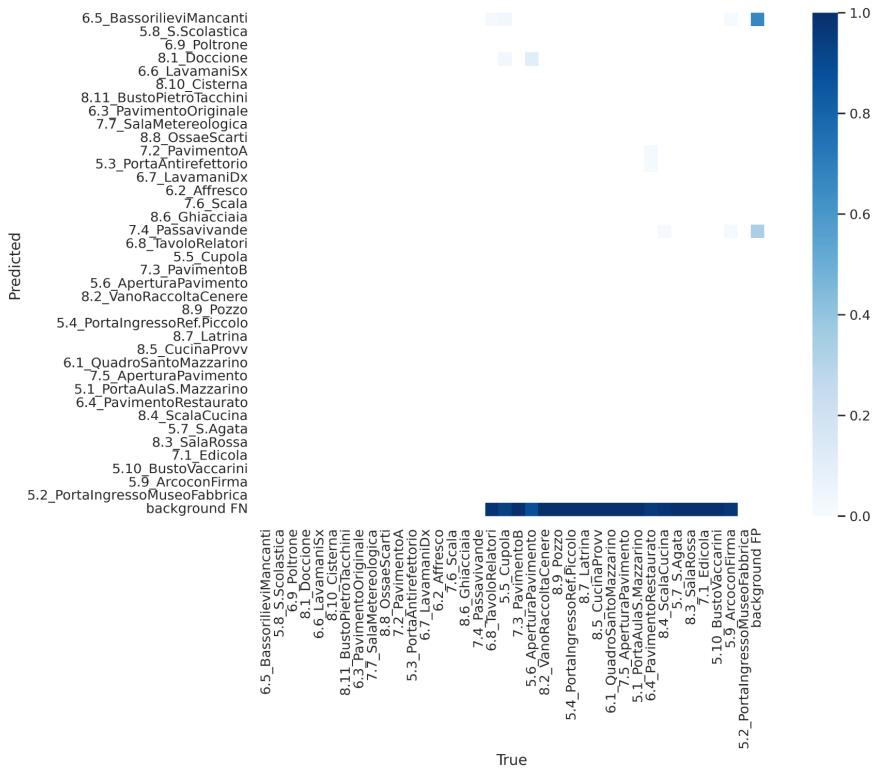
Monastero dei Benedettini

Si mostrano di seguito i risultati ottenuti su questo subset:



È possibile notare come i valori di Loss (per la validazione) siano più bassi rispetto ai tentativi precedenti, e che la mAP è scesa. Nonostante i valori siano ancora bassi (ma ciò è dovuto alla complessità del subset in quanto tale). Per cui, da un punto di vista della Loss, modelli più grandi

fanno decisamente meglio su dataset così complessi, tuttavia la mAP ne risente (seppur di poco). Di seguito si mostra la matrice di confusione ottenuta dalla classificazione.



In questo caso è possibile notare come, addestrando l'intero modello, si sia riusciti ad ottenere una mAP leggermente più alta rispetto al modello v5s allenato per intero; purtroppo, tuttavia, i valori

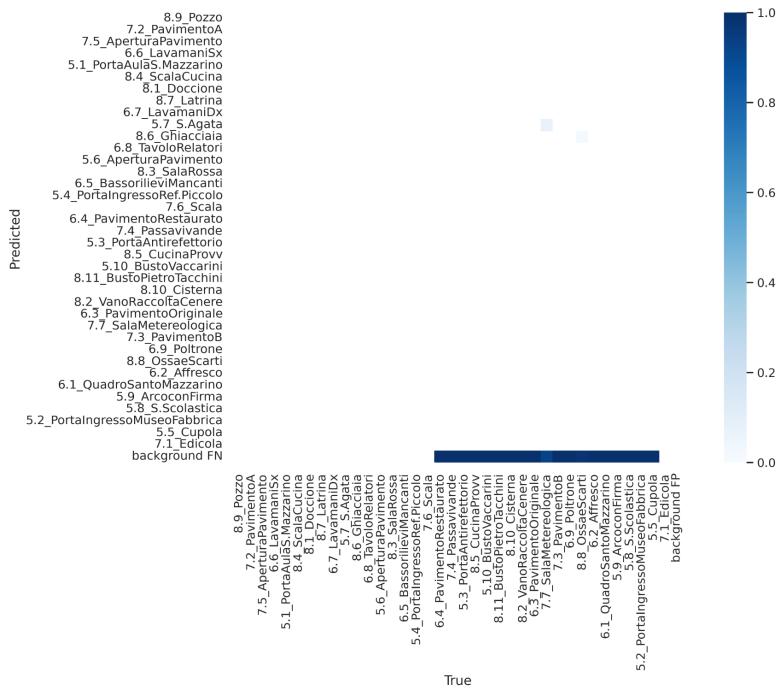
della funzione Loss sulla validazione continuano a non essere promettenti a causa della sua crescita di epoca in epoca.

Si mostra anche la matrice di confusione ottenuta dalla classificazione.



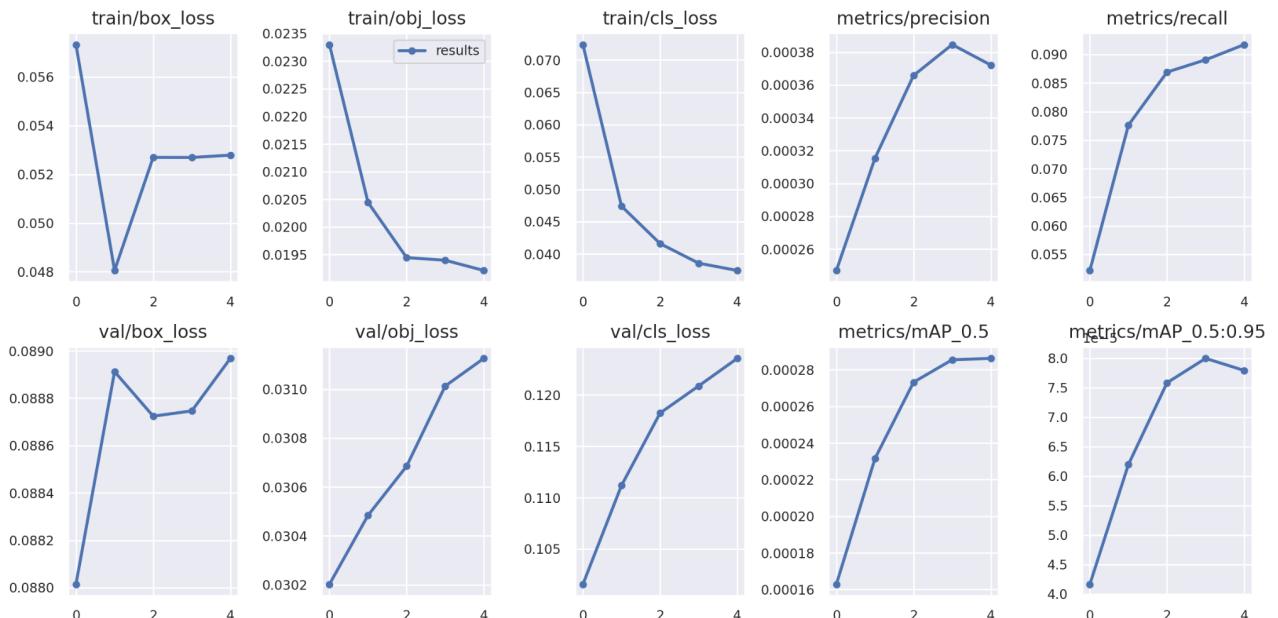
I massimi valori di Loss per questo modello, sono i minimi tra quelli ottenuti, e la mAP è rimasta pressoché invariata rispetto al precedente addestramento, eseguito su tutto il modello. Questo indica, seppur lievemente (visti i bassissimi valori), una miglior generalizzazione causata dall'addestramento esclusivo sul classificatore.

Si mostra di seguito la matrice di confusione ottenuta mediante questo classificatore.



Palazzo Bellomo

Seguono i risultati sul subset del Palazzo Bellomo:



Si noti come in questo caso si sia ottenuto un valore di mAP minore rispetto a quello del modello addestrato interamente; ciò indica come l'addestramento su tutti i layer, per un modello più grande come questo e per un subset complesso come quello di Palazzo Bellomo sia effettivamente più

indicato. Inoltre, si noti che i valori di Loss di validazione, sono rimasti pressoché invariati rispetto al modello addestrato su tutti i livelli.

Segue la matrice di confusione ottenuta mediante questa sperimentazione.



5.3 Tentativi su YOLOv5l

Vista la complessità di entrambi i subset, e considerando il grande numero di classi individuate su entrambi, si sono voluti effettuare anche dei tentativi sul modello “large”, il più grande in assoluto, ossia **YOLOv5l**.

Anche in questo modello sono stati utilizzati i medesimi iperparametri descritti nel paragrafo relativo alle sperimentazioni su **YOLOv5s**; tuttavia, rispetto a prima, vi sono state delle piccole differenze per quanto riguarda il numero di epoche di addestramento: infatti, sul subset del Monastero dei Benedettini sono stati effettuati addestramenti per cinque epoche (sia per il training completo, sia per il training sul fully-connected), mentre sul subset del Palazzo Bellomo – visti i lunghi tempi di addestramento (50 minuti per epoca) e le limitazioni tempistiche della GPU di Google Colab – sono stati effettuati per quattro epoche.

L’architettura della rete è mostrata di seguito.

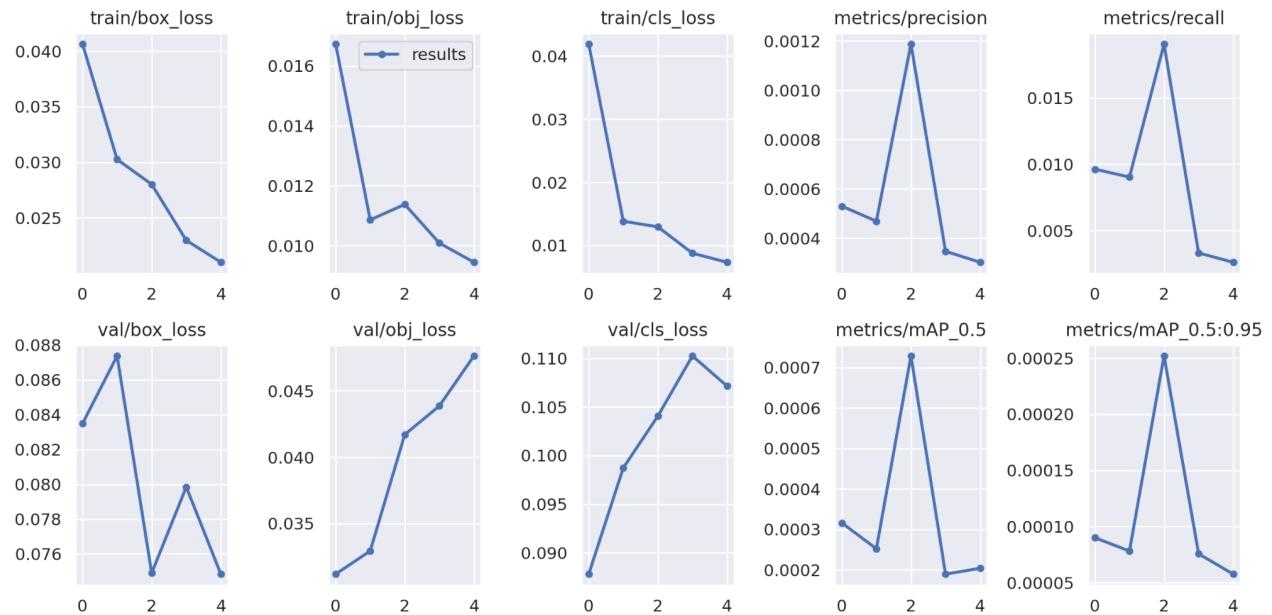
| from | n | params | module | arguments | |
|------|--------------|--------|--------------------|--------------------------------------|--|
| 0 | -1 | 7040 | models.common.Conv | [3, 64, 6, 2, 2] | |
| 1 | -1 | 73984 | models.common.Conv | [64, 128, 3, 2] | |
| 2 | -1 | 156928 | models.common.C3 | [128, 128, 3] | |
| 3 | -1 | 295424 | models.common.Conv | [128, 256, 3, 2] | |
| 4 | -1 | 6 | 1118208 | models.common.C3 | [256, 256, 6] |
| 5 | -1 | 1 | 1180672 | models.common.Conv | [256, 512, 3, 2] |
| 6 | -1 | 9 | 6433792 | models.common.C3 | [512, 512, 9] |
| 7 | -1 | 1 | 4720640 | models.common.Conv | [512, 1024, 3, 2] |
| 8 | -1 | 3 | 9971712 | models.common.C3 | [1024, 1024, 3] |
| 9 | -1 | 1 | 2624512 | models.common.SPPF | [1024, 1024, 5] |
| 10 | -1 | 1 | 525312 | models.common.Conv | [1024, 512, 1, 1] |
| 11 | -1 | 1 | 0 | torch.nn.modules.upsampling.Upsample | [None, 2, ‘nearest’] |
| 12 | [-1, 6] | 1 | 0 | models.common.Concat | [1] |
| 13 | -1 | 3 | 2757632 | models.common.C3 | [1024, 512, 3, False] |
| 14 | -1 | 1 | 131584 | models.common.Conv | [512, 256, 1, 1] |
| 15 | -1 | 1 | 0 | torch.nn.modules.upsampling.Upsample | [None, 2, ‘nearest’] |
| 16 | [-1, 4] | 1 | 0 | models.common.Concat | [1] |
| 17 | -1 | 3 | 690688 | models.common.C3 | [512, 256, 3, False] |
| 18 | -1 | 1 | 590336 | models.common.Conv | [256, 256, 3, 2] |
| 19 | [-1, 14] | 1 | 0 | models.common.Concat | [1] |
| 20 | -1 | 3 | 2495488 | models.common.C3 | [512, 512, 3, False] |
| 21 | -1 | 1 | 2360320 | models.common.Conv | [512, 512, 3, 2] |
| 22 | [-1, 10] | 1 | 0 | models.common.Concat | [1] |
| 23 | -1 | 3 | 9971712 | models.common.C3 | [1024, 1024, 3, False] |
| 24 | [17, 20, 23] | 1 | 226170 | models.yolo.Detect | [37, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [256, 512, 1024]] |

5.3.1 Primo tentativo di training

Il primo addestramento è stato – come sempre – eseguito sull’intero modello.

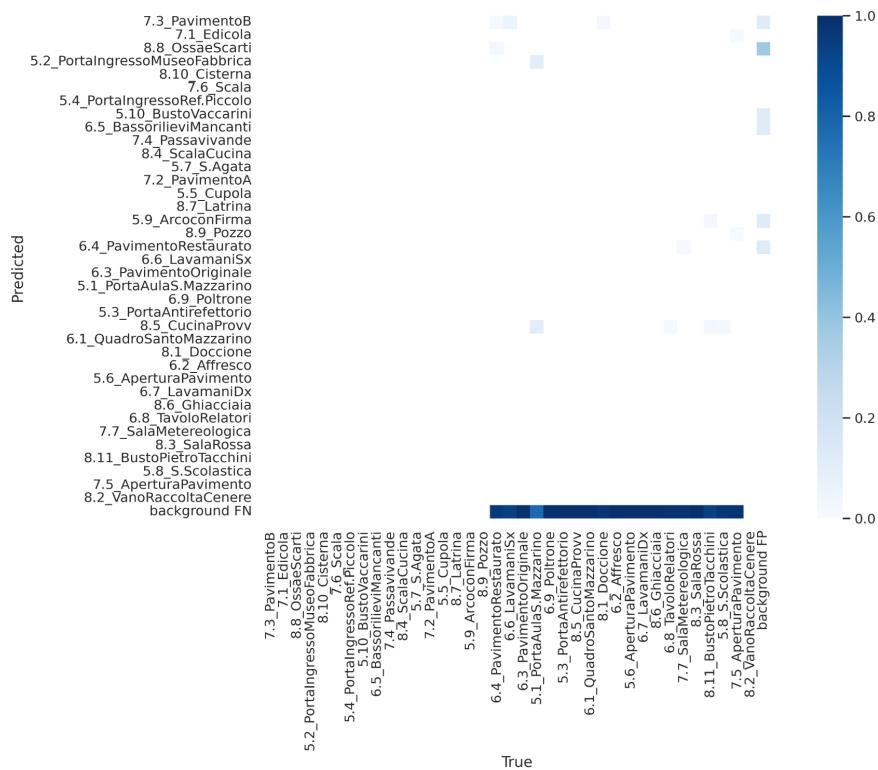
Monastero di Benedettini

Di seguito si mostrano i risultati ottenuti addestrando il modello su questo subset.



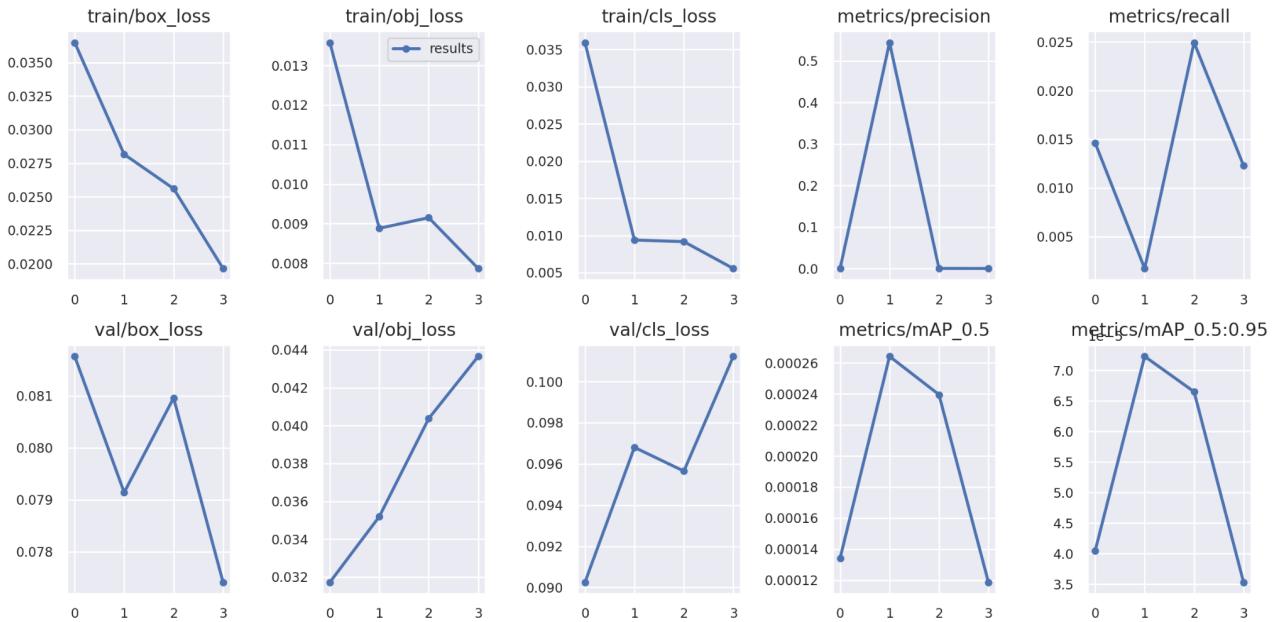
La mAP qui ottenuta, è praticamente la più bassa in assoluto sul subset del Monastero dei Benedettini. Tuttavia, anche i valori di Loss sulla validazione sono effettivamente più bassi, e la precisione è praticamente la più alta escludendo l’addestramento effettuato sull’intero YOLOv5s.

Di seguito si mostra la matrice di confusione ottenuta.



Palazzo Bellomo

I risultati ottenuti addestrando l'intero modello "large" sul subset del Palazzo Bellomo sono i seguenti.



In questo caso, la migliore mAP ottenuta è la minima in assoluto; nonostante tutto, la Precision è la più alta in assoluto e i valori sulla funzione di Loss sembrano essere abbastanza buoni sulla validazione; praticamente, questo modello sembra avere delle performance molto simili a quelle viste per lo **"small" addestrato per intero**.

Si mostra la matrice di confusione ottenuta.

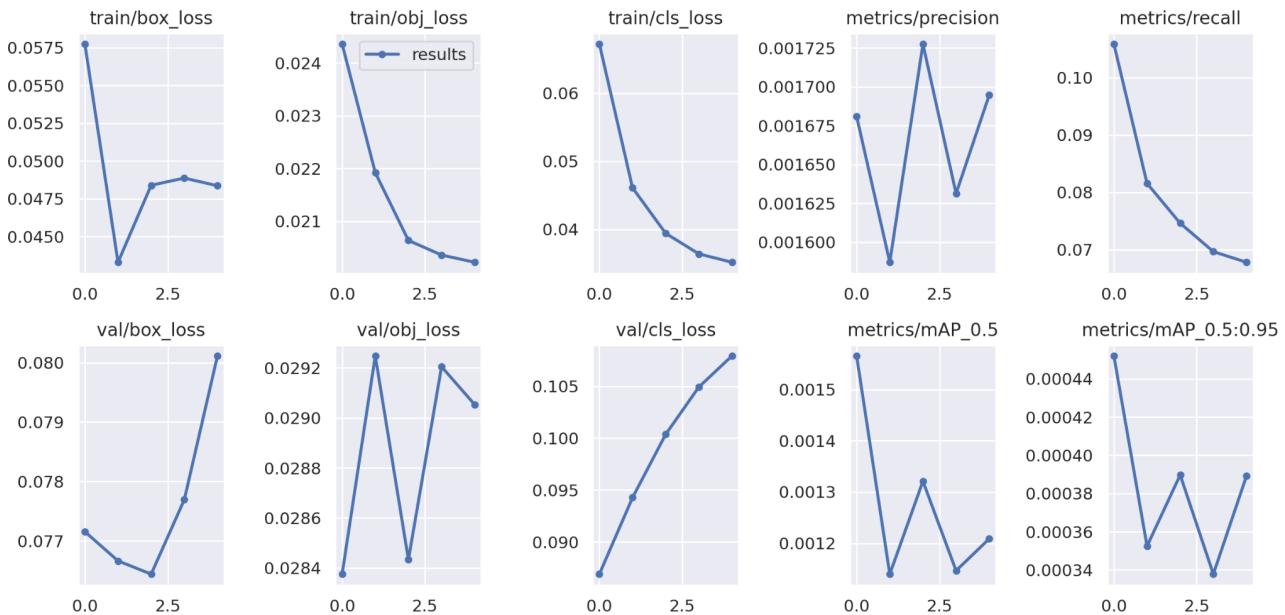


5.3.2 Transfer Learning

Anche in questo caso, l'addestramento è stato eseguito solamente sul classificatore, e quindi l'unico livello fully-connected del modello YOLOv5l.

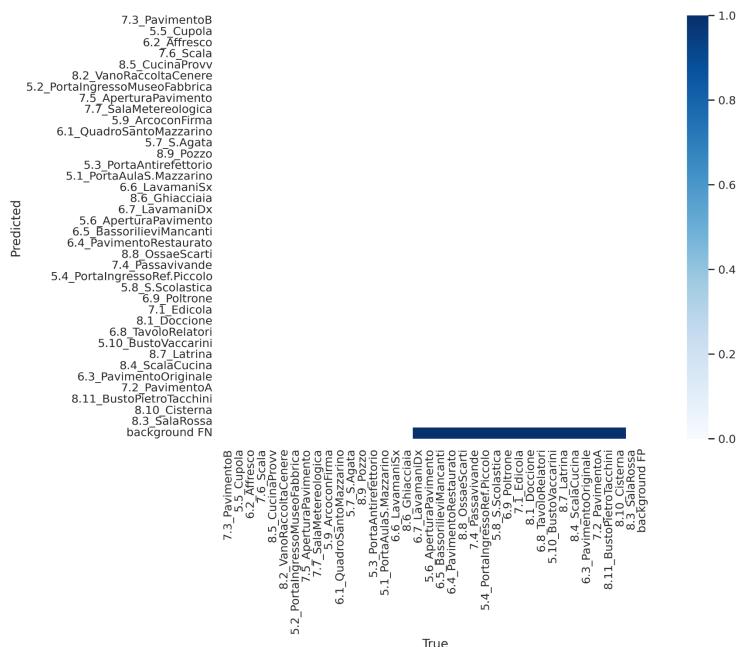
Monastero dei Benedettini

Di seguito si mostrano i risultati ottenuti addestrando il classificatore sul subset del Monastero dei Benedettini.



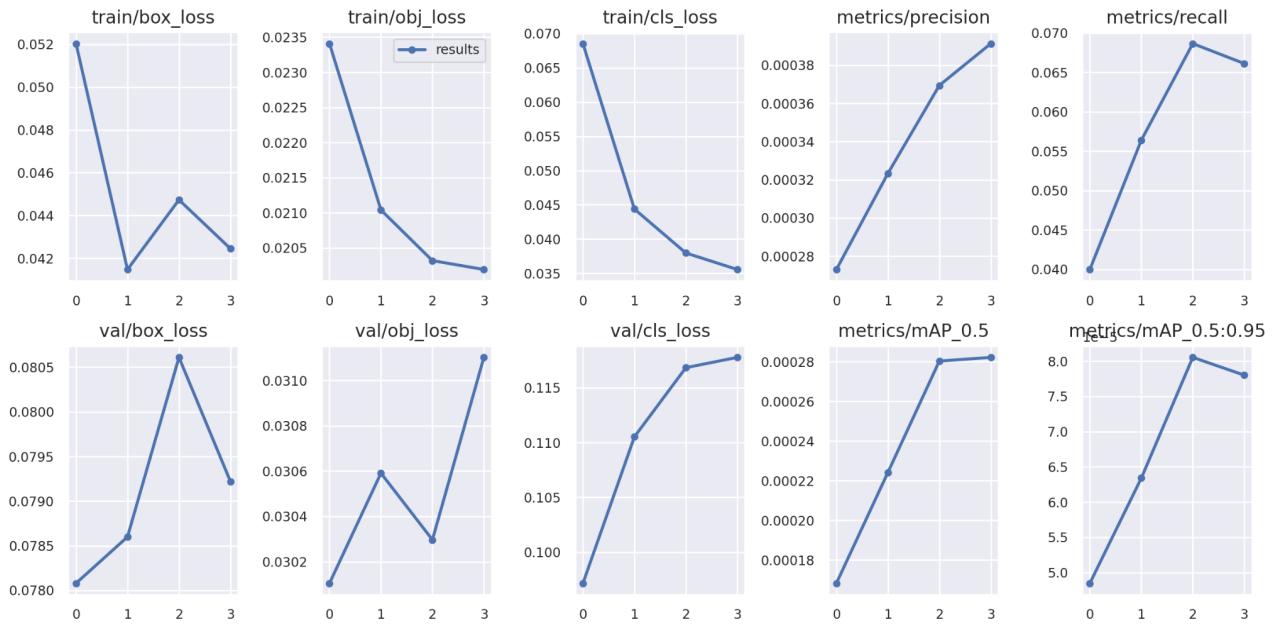
La mAP ottenuta allenando esclusivamente il layer fully connected, è più alta rispetto a quella rilevata addestrando l'intero modello, confermando quindi che allenare la parte convolutiva è poco utile nella risoluzione del problema. La massima Precision rilevata addestrando questo modello, è inoltre generalmente la più alta ottenuta addestrando il subset; tuttavia, i valori assunti dalla funzione di Loss tendono comunque ad aumentare nel corso delle cinque iterazioni.

Si mostra la matrice di confusione ottenuta in seguito alla classificazione con questo modello.



Palazzo Bellomo

Si riportano, infine, i risultati ottenuti addestrando il livello fully-connected sul dataset di Palazzo Bellomo.



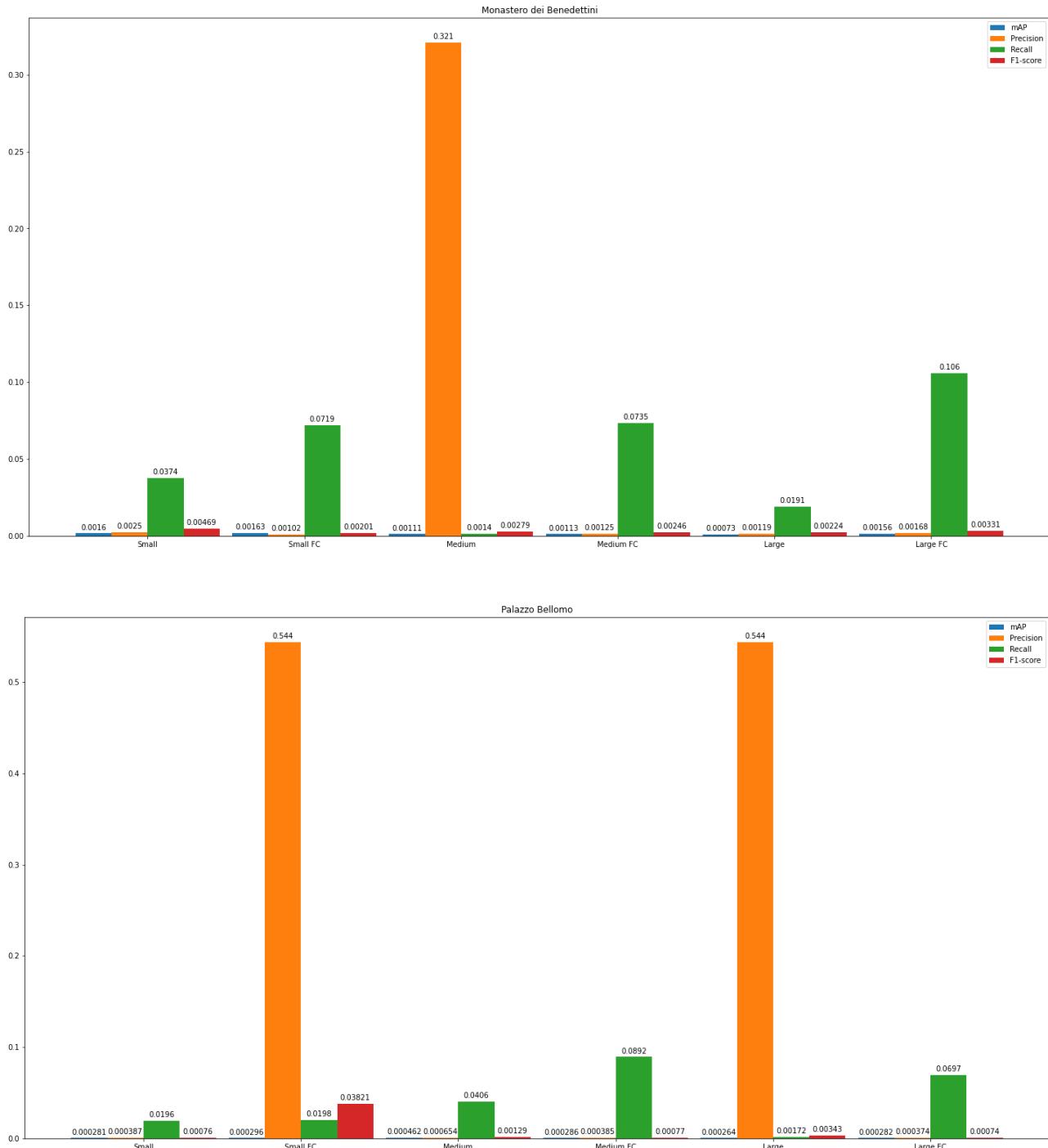
La massima mAP ottenuta nel corso di questo addestramento è generalmente la più alta, se considerata relativamente al Palazzo Bellomo. I valori di Loss, invece, sebbene in continua crescita anche in questo caso, sembrano essere in linea con quelli ottenuti con il primo addestramento in assoluto, eseguito sull'**intero YOLOv5s**, variando quindi poco rispetto al precedente addestramento eseguito sul modello “large” per intero.

La matrice di confusione ottenuta dalla classificazione è la seguente.



5.4 Confronto dei risultati ottenuti

Tirando le somme, si può facilmente constatare che i risultati raggiunti non sono così eclatanti quanto sperato: tutte e dodici le sperimentazioni (sei per ognuna dei due subset) hanno avuto alti e bassi, se relazionati tra loro, ma generalmente la mAP non ha mai raggiunto l'1%.



Osservando i grafici a barre sopra riportati, è tuttavia possibile notare che, sebbene l'utilizzo di modelli via via più grandi possa diminuire il valore di mAP, addestrando esclusivamente il layer fully-connected – per ognuno dei modelli – si riescono ad apprezzare dei leggeri aumenti di questa misura.

È anche vero che, purtroppo, essendo questi valori così piccoli da essere approssimativamente pari allo zero, è difficile effettuare un confronto più preciso.

Da un punto di vista della mAP è possibile affermare che i migliori modelli ottenuti sono:

- il medio allenato esclusivamente al layer fully connected per quanto concerne il subset del Monastero dei Benedettini;
- il medio totalmente allenato per quanto concerne il subset del Palazzo Bellomo.

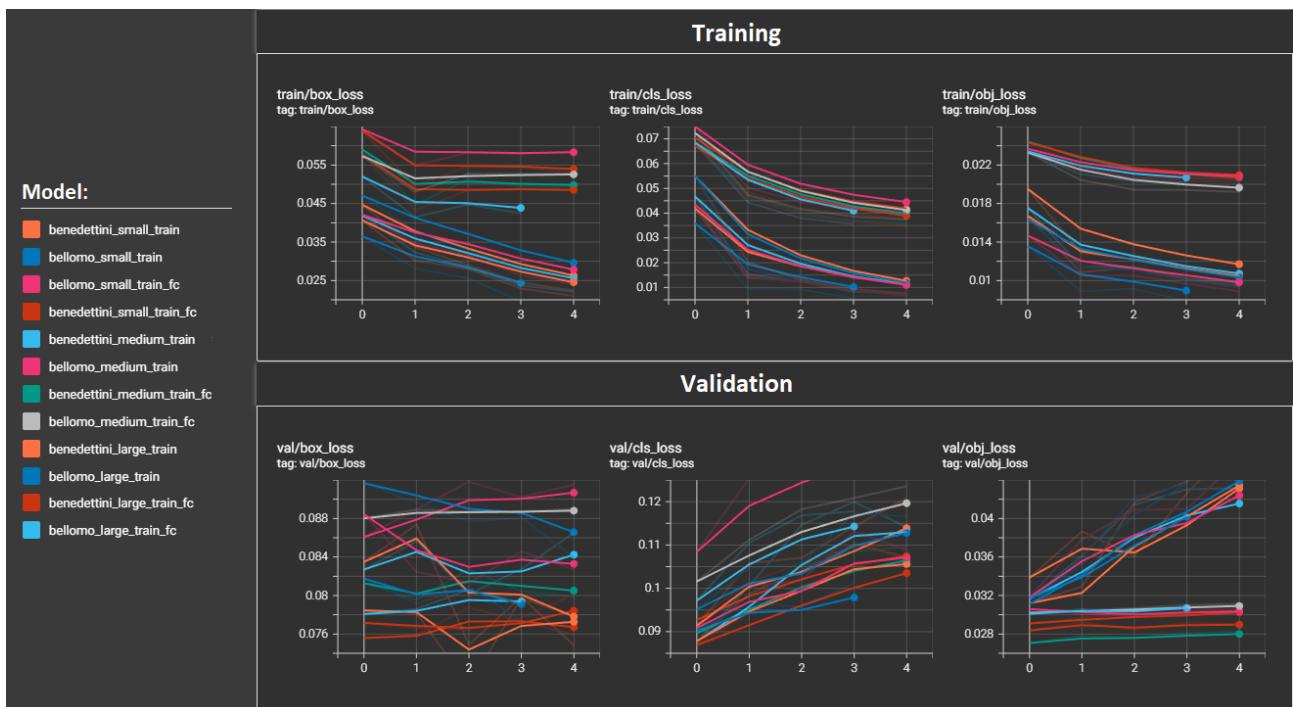
Da un punto di vista dello score F1, i migliori modelli sono:

- lo “small” per quanto concerne il subset del Monastero dei Benedettini;
- lo “small” con transfer learning per quanto concerne il Palazzo Bellomo.

Per cui, in realtà è possibile affermare che i modelli “large” sono decisamente quelli che fanno peggio, in assoluto, almeno se addestrati su questo dataset.

Anche i valori assunti dalla funzione di Loss in fase di validazione, purtroppo in continuo aumento, non promettono bene: in tutti i casi, infatti, si registra un continuo aumento, il ché è sintomo di overfitting, visti i bassi valori assunti dalla stessa in fase di training.

Tuttavia, bisogna considerare che questi valori sono stati registrati su un esiguo numero di epoche – 4/5 per la precisione –, quindi un addestramento più approfondito (inattuabile con le limitate risorse fornite da *Google Colaboratory*) potrebbe fruttare risultati migliori.



Naturalmente, questi risultati sono anche derivanti alla complessa struttura dei due subset, caratterizzati da numerosi esempi raggruppati in altrettanto numerose classi.

Capitolo 6: Demo

La demo fornita in corredo a questo progetto consiste in un Jupyter Notebook denominato **EGO-CH_YOLOv5_demo.ipynb**. Esso è composto da quattro sezioni, una per ogni operazione da effettuare in maniera consecutiva.

Mediante la **prima sezione**, è possibile scaricare, effettuando un opportuno git clone, l'ultima versione di **YOLOv5** direttamente dal repository GitHub ufficiale.

Inoltre vengono installate tutte le **dipendenze python** necessarie, mediante l'applicativo pip.

Vengono anche visualizzate le informazioni sulla **GPU**, se disponibile.

Mediante la **seconda sezione**, viene scaricato da Google Drive il file zip contenente tutti i **modelli addestrati** nel corso delle sperimentazioni, per poi estrarre tale archivio.

Mediante la **terza sezione**, l'utente può scegliere il **modello** da utilizzare e il relativo **sito** culturale di interesse.

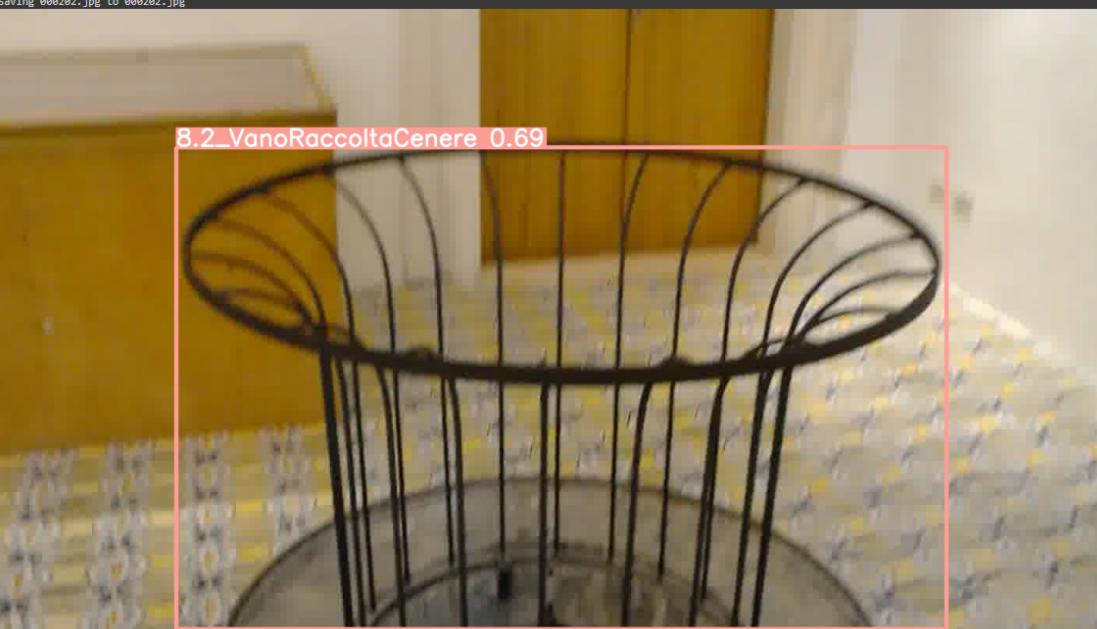
Mediante la quarta e **ultima sezione**, l'utente è in grado di effettuare un'**inferenza** su un'immagine presente sul proprio hard drive. Per l'upload dell'immagine sulla quale effettuare la detection, è stato usato il modulo `files` fornito direttamente da Google Colab.

4 - Inferenza su immagini

Eseguito la cella di seguito, è possibile caricare un file immagine ed effettuare un'inferenza su di essa.

```
1 from google.colab import files
2 from IPython.display import Image, display
3 import os
4
5 uploaded = files.upload()
6 for fn in uploaded.keys():
7     fn_p = '/content/' + fn
8     os.system("rm -rf /content/yolov5/runs/detect")
9     os.system("python /content/yolov5/detect.py --weights /content/model.pt --img 640 --conf 0.50 --source " + fn_p)
10    display(Image(filename='/content/yolov5/runs/detect/exp/' + fn))
```

Scritto da 000202.jpg
• 000202.jpg (image/jpeg) - 35838 bytes, last modified: 25/7/2018 - 100% done
Saving 000202.jpg to 000202.jpg



Capitolo 7: Codice

L'organizzazione del codice di addestramento, presente nel Jupyter Notebook **ML_model.ipynb**, prevede una sezione dedicata alla gestione delle problematiche del dataset, e l'effettivo codice di addestramento dei vari modelli.

7.1 Gestione del dataset

Nella **Sezione 2** del notebook **EGO-CH_YOLOv5_classification.ipynb** sono state scritte tutte le procedure utilizzate per la gestione del dataset e la risoluzione di tutte le problematiche affrontate nel **Capitolo 2.3** di questa relazione.

7.1.1 Rinomina dei file

Al fine di rinominare le cartelle `bbox_annotations` secondo la più appropriata nomenclatura `labels`, è stata scritta la procedura **rename_labels_folder(site)**, la quale, presa in input una variabile `site` contenente la stringa “`benedettini`” o “`bellomo`”, accede alle cartelle dell'apposito `site` (il cui indirizzamento è effettuato mediante un dizionario `folders`, con le due chiavi “`benedettini`” e “`bellomo`”, preventivamente definito), rinomina la cartella interessata.

Al fine di spostare ogni file di etichettatura presente nelle cartelle `images` dei subset, è stata definita la funzione **move_labels(site)**, la quale accede alla cartella `images` del `site` scelto e, se presenti dei file `.txt` (di etichettatura), li sposta nell'appropriata cartella `labels`.

Al fine avere, per ogni file di immagine, l'omonimo file di testo presente nella cartella `labels`, è stata definita la procedura **repair_val_names(site)**, la quale accede a tale cartella e, per ogni file in essa presente, sostituisce il carattere spaziatore con un carattere di underscore.

7.1.2 Riparazione delle etichette

Non tutte le etichette presenti nelle cartelle `labels` dei due subset presentano una sintassi corretta, per cui, la riparazione è stata eseguita definendo la funzione **threshold(site, mode)**, la quale effettua due diverse operazioni in contemporanea, per ogni file di etichettatura presente nella cartella definita da `mode` (di training, di test o di validazione), del `site` interessato:

1. normalizza eventuali valori non normalizzati;
2. soglia eventuali valori fuori dagli estremi dell'intervallo [0, 1] e ripara l'identificativo di classe, se al di fuori del range di classe massima.

La prima operazione viene eseguita mediante la funzione **norm(filename, site)**, la quale apre ogni file con nome `filename` presente nella cartella `labels` del rispettivo `site`, e normalizza i valori.

La seconda operazione, la quale, aperto ogni file con nome `filename`, limita tutti i valori al di fuori del range interessato e ripara l'identificativo di classe, viene invece eseguita mediante la funzione **repair_label_and_tresh(filename)**.

7.1.3 Creazione dello yaml di configurazione

La versione 5 di YOLO prende in input i valori di training e validazione del dataset scelto mediante un file in formato **yaml** che indica verso le suddette cartelle.

Al fine di automatizzare la creazione del file **yaml** relativo al subset interessato, è stata definita la procedura **write_yaml(site)**, la quale, a sua volta, esegue due operazioni utili al generamento di questo file del relativo **site**:

1. scrive un file di testo, per ogni modalità, che elenca ogni singolo file;
2. definisce il numero (e la lista) di classi, in base al nome delle cartelle contenute in ognuno dei set di training del subset.

La prima operazione è effettuata mediante la funzione **write_file_ls(site, mode)**, la quale, definito il sito culturale e la modalità, genera il file di testo contenente, per esempio, tutti i file di training se la **mode** scelta è definita dalla stringa “**train**”, tutti i file di validazione se la **mode** scelta è definita da “**val**” e tutti i file di test se invece è definita da “**test**”. La seconda operazione è effettuata mediante la funzione **write_names(site)**, la quale, entrando nella cartella di training del **site**, conta tutte le sottocartelle (una per classe) presenti, e salva il loro nome in una lista di stringhe.

7.2 Addestramento

Tutte le sezioni successive presenti nel notebook, contengono righe di comando eseguite per effettuare gli esperimenti secondo quanto visto nel **Capitolo 5**.

È necessario, tuttavia, fare riferimento allo script Python **train_freeze.py**, il quale è stato scritto, partendo dal file **train.py** già presente nel repository di YOLOv5, al fine di poter effettuare il congelamento – direttamente da riga di comando – dei livelli interessati sulla rete, così da poter eseguire il transfer learning. Di seguito le righe di codice aggiunte per apportare questa modifica.

```
147     # Freeze
148     freeze = ['model.%s.' % x for x in range(int(opt.freeze_level)+1)] # parameter names to freeze (full or partial)
149     for k, v in model.named_parameters():
150         v.requires_grad = True # train all layers
151         if any(x in k for x in freeze):
152             print('freezing %s' % k)
153             v.requires_grad = False
```

Naturalmente, è stata aggiunta anche una riga di codice al fine di definire il livello al di sotto del quale attivare il congelamento direttamente da riga di comando.

```
487     parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
488     parser.add_argument('--freeze_level', type=int, default=24, help='layer up to which we freeze the network') ←
489     parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
```

Conclusioni

I risultati scaturiti dalle sperimentazioni in questo progetto evidenziano come il problema di riconoscimento delle opere d'arte in un sito culturale sia ancora oggetto di ricerca, nel settore del machine learning.

La grossa mole di dati da analizzare e apprendere, infatti, rendono il problema particolarmente complesso, soprattutto se l'approccio risolutivo è caratterizzato da risorse limitate, come in questo caso.

Dopo l'osservazione di più di 30.000 immagini su un subset (*Monastero dei Benedettini*) e più di 50.000 sull'altro (*Palazzo Bellomo*), tutti i modelli di tipo YOLOv5 hanno appreso veramente poco; sono stati fatti diversi tentativi di miglioramento nell'apprendimento, mediante il transfer learning, ma anche ciò non ha fruttato i risultati sperati, sebbene in ogni caso si è registrato un sensibile aumento in mAP – rispetto ai modelli allenati per intero, si intende –, la cui cosa effettivamente conferma che sarebbe inutile addestrare i modelli per intero (relativamente a questo problema), in quanto il classificatore posto in superficie alla CNN utilizzata è il vero layer “specifico” del problema stesso.

Si può affermare, quindi, che da queste sperimentazioni le massime mAP ottenute sono le seguenti:

| Sito Culturale | mAP |
|---------------------------|-----------|
| Monastero dei Benedettini | 0.00163% |
| Palazzo Bellomo | 0.000462% |

molto più basse rispetto alle iniziali mAP ottenute dagli originali autori di EGO-CH, e qui riportate per completezza. Si noti che, tuttavia, la relazione tra le due misure ottenute è effettivamente rispettata in quanto si ha $mAP(\text{Monastero dei Benedettini}) > mAP(\text{Palazzo Bellomo})$.

| Cultural Site | mAP |
|------------------------------|--------|
| 1) Palazzo Bellomo | 10.59% |
| 2) Monastero dei Benedettini | 15.45% |

Inoltre, i migliori score F1 totalizzati tra le varie sperimentazioni sono presentati nella tabella mostrata qui di seguito:

| Sito Culturale | F1-score |
|---------------------------|----------|
| Monastero dei Benedettini | 0.00469 |
| Palazzo Bellomo | 0.03821 |

Tutte queste misure, non fanno altro che confermare come effettivamente, anche con migliori risorse, il problema non sia di semplice risoluzione.

La complessità del problema è infatti data non solo dalla numerosità dei due subset (sia intesa come numero di elementi, sia intesa come numero di classi secondo le quali gli elementi sono raggruppati), ma anche dalla scarsa presenza di etichette che caratterizza questi ultimi.