MMDetection: addestrare

MMDetection: addestrare

Premessa

In queste slide saranno indicati i passi che permettono di addestrare uno specifico modelli di object detection, *RTMDet*, tramite l'utilizzo dell'ecosistema di *OpenMMLab* ed in particolare del modulo *mmdetection*.

I dati sono forniti dal processi di etichettatura con *Label Studio*.

MMDetection: addestrare

Ambienti conda

Sono assunti i seguenti aspetti:

1. Nel pc è disponibile il gestore di ambienti virtuali *conda*.
2. In *conda* è presente un ambiente dedicato al pacchetto, installato, *label-studio*.
3. In *conda* è presente un ambiente dedicato ai pacchetti, installati, dell'ecosistema *openMMLab* e, in particolare, *mmdetection*.
4. Sono stati creati dati di training e validazione tramite etichettatura con *Label Studio*.

Verifichiamo quindi gli ambienti virtuali installati:

conda env list

```
(base) C:\Users\FBAIRE>conda env list
# conda environments:
#
base                  *  C:\Users\FBAIRE\AppData\Local\miniconda3
corsoai                  C:\Users\FBAIRE\AppData\Local\miniconda3\envs\corsoai
labelstudio              C:\Users\FBAIRE\AppData\Local\miniconda3\envs\labelstudio
openmm                   C:\Users\FBAIRE\AppData\Local\miniconda3\envs\openmm
```

MMDetection: addestrare

Struttura cartelle

Accediamo con *Visual Studio Code* alla *root* di installazione dell'ecosistema *openMMLab*.

Per questa presentazione, sarà il seguente:

c:\code\myGit\openmm

All'apertura della root, saranno disponibili le cartelle contenenti i sorgenti creati in fase di installazione dell'ecosistema.

MMDetection: addestrare

Sorgente dati

Per procedere con l'addestramento, andare ad aggiungere alla struttura una nuova cartella: *data*.

In questa cartella si andranno ad inserire i dati di addestramento, validazione e test.

Nel nostro caso, si tratta della cartella *balloon* e delle sue sotto-cartelle *train* e *val*.

MMDetection: addestrare

Sorgente dati

Le due cartelle sono state ottenute dall'esportazione in formato *COCO* delle annotazioni effettuate con *Label Studio*.

All'interno:

- *images*: la cartella contenente le immagini.
- *result.json*: il file contenente le annotazioni.

MMDetection: addestrare
Configurazione

Addestrarsi per il task di *object detection* prevede i seguenti step:

1. Scegliere un modello.
2. Cercarne il file di configurazione fra i sorgenti di *mmdetection*.
3. Analizzare il file di configurazione.
4. Ottenere il checkpoint di partenza dei parametri associati al modello.

MMDetection: addestrare

Configurazione

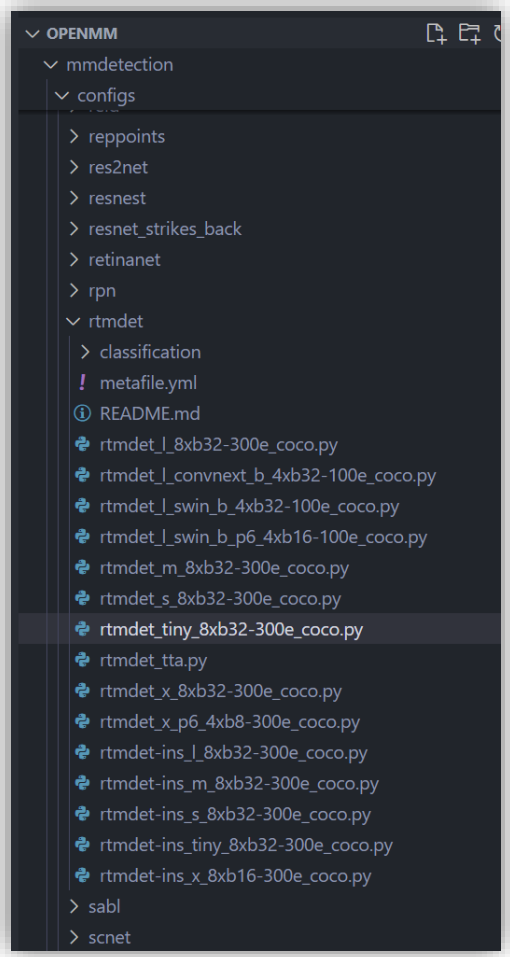
Il primo punto è deciso e, nel nostro caso, si tratta del modello *RTMDet*.

Di seguito il riferimento alla pagina di documentazione del modello fornita dal *model zoo* di *OpenMMLab*.

RTMDet

Per il secondo punto, si esplorano i sorgenti, a partire dalla *root*, che si trovano in:

- *mmdetection*
  - *configs*
    - *rtmdet*



MMDetection: addestrare

Configurazione

Il modello specifico, di *RTMDet*, scelto è la versione *tiny* che utilizza il seguente file di configurazione:

rtmdet_tiny_8xb32-300e_coco.py

Object Detection

Model	size	box AP	Params(M)	FLOPS(G)	TRT-FP16-Latency(ms) RTX3090	TRT-FP16-Latency(ms) T4	Config	Download
RTMDet-tiny	640	41.1	4.8	8.1	0.98	2.34	config	model | log

Il modo in cui *mmdetection* e, in generale, *openmm* gestiscono i sorgenti e le configurazioni è modulare. Per questo motivo, analizzando il file di configurazione è possibile capire che:

- Deriva da un ulteriore file di configurazione.
- Modifica parte dei parametri ereditati.
- Aggiunge ulteriori parametri.

MMDetection: addestrare

Configurazione

Nel caso di *rtmdet_tiny_8xb32-300e_coco.py*, deriva da:

- *rtmdet_s_8xb32-300e_coco.py* che deriva da:
  - *rtmdet_l_8xb32-300e_coco.py* che deriva da:
    - *default_runtime.py*
    - *schedule_1x.py*
    - *coco_detection.py*
    - *rtmdet_tta.py*

MMDetection: addestrare

Configurazione

Nei file di configurazione sono poi presenti molteplici parametri, ognuno dei quali va a definire cosa fare e come farlo:

- *Loop di training, validazione, test.*
- *Pre-processamenti.*
- *Modalità di logging.*
- *Visualizzazione dei dati.*
- *Come restituire i risultati.*
- …

I parametri sono innumerevoli.

Uno fra questi, di interesse, è *checkpoint*:

```
checkpoint = 'https://download.openmmlab.com/mmdetection/v3.0/rtmdet/cspnext_rsb_pretrain/cspnext-tiny_imagenet_600e.pth'
```

MMDetection: addestrare

Configurazione

Questo parametro sta ad indicare che il modello scelto ha subito un precedente addestramento e sono disponibili al download i parametri di addestramento finali raggiunti.

L'addestramento è avvenuto rispetto ad un dataset standard, in questo caso:

- *ImageNet*

Procediamo quindi al download dei parametri addestrati. In questo modo:

- L'addestramento su nuovi dati partirà da una conoscenza pregressa.
- Saranno sfruttate feature di alto livello estratte in precedenza.
- La convergenza di addestramento sarà migliore rispetto al non utilizzo dei parametri di addestramento pregressi.

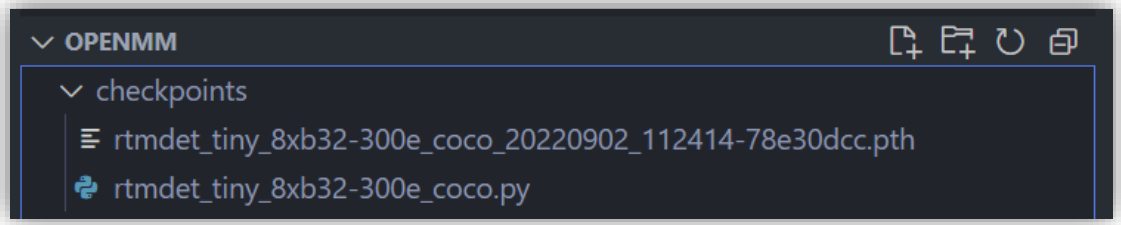
Da ambiente *openmm* attivo:

mim download mmdet --config rtmdet_tiny_8xb32-300e_coco --dest ./checkpoints

MMDetection: addestrare

Configurazione

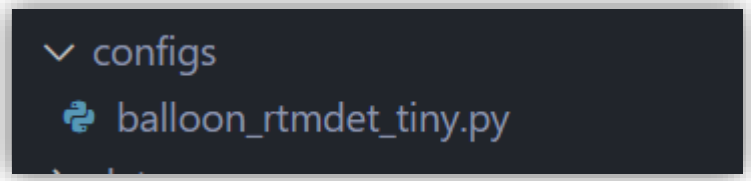
Il risultato sarà il download dei parametri in una nuova cartella, *checkpoints*:



Per procedere all'addestramento, creiamo una ulteriore cartella, *configs*, nella quale inserire un personale file di configurazione:

- *Basato su rtmdet_tiny_8xb32-300e_coco.py*
- *Dedicato al dataset sul quale ci vogliamo addestrare.*

Chiamiamo questo file *balloon_rtmdet_tiny.py*



MMDetection: addestrare

Configurazione

Del file, sono da notare alcuni parametri:

- *_base_* : il *config* è un derivato del modello rtmdet scelto.
- *max_epochs* : la durata delle epoche di addestramento.
- *base_lr* : il learning rate di partenza.
- *classes* : le classi da rilevare.
- *metainfo* : il colore da visualizzare per le classi.
- *num_classes* : il numero di classi.
- *load_from* : il file di checkpoint dal quale partire.

MMDetection: addestrare

Configurazione

Eseguire l'addestramento è estremamente semplice. Richiede l'invio di un comando a partire dalla *root* dei sorgenti e su di un terminale con ambiente *openmm* attivato:

python mmdetection/tools/train.py configs/balloon_rtmdet_tiny.py

```
- mmengine - INFO - Epoch(train) [20][55/61]  base_lr: 4.0247e-06 lr: 4.0247e-06  eta: 0:00:00  time: 0.1050  data_time: 0.0046  memory: 495  loss: 1.0135  loss_cls: 0.7085  loss_bbox: 0.3050
- mmengine - INFO - Epoch(train) [20][60/61]  base_lr: 4.0020e-06 lr: 4.0020e-06  eta: 0:00:00  time: 0.1043  data_time: 0.0046  memory: 495  loss: 0.9367  loss_cls: 0.6345  loss_bbox: 0.3022
- mmengine - INFO - Exp name: balloon_rtmdet_tiny_20240517_171107
- mmengine - INFO - Saving checkpoint at 20 epochs
- mmengine - INFO - Evaluating bbox...
eparing results...

...

age evaluation...
ation type *bbox*

valuation results...

sion  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.634
sion  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.740
sion  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.701
sion  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
sion  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.261
sion  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.774
1     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.226
1     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.700
1     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.788
1     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
1     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.708
1     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.858
- mmengine - INFO - bbox_mAP_copypaste: 0.634 0.740 0.701 0.000 0.261 0.774
- mmengine - INFO - Epoch(val) [20][3/3]    coco/bbox_mAP: 0.6340  coco/bbox_mAP_50: 0.7400  coco/bbox_mAP_75: 0.7010  coco/bbox_mAP_s: 0.0000  coco/bbox_mAP_m: 0.2610  coco/bbox_mAP_l: 0.7740  data_time: 0.0568  time: 0.1433
```

MMDetection: addestrare

Configurazione

Al termine dell'addestramento sarà possibile trovare una nuova cartella, *work_dirs*, nella quale sono presenti i risultati dell'addestramento appena effettuato:

- *Il file di configurazione.*
- *Il modello migliore.*
- *I modelli salvati ogni K epoche.*
- *L'ultimo check_point.*

Per valutare il comportamento del modello addestrato, eseguire infine lo script python *evaluate_model.py* direttamente estratto dalla risorsa X di *Virtuale*.

I risultati della detection saranno direttamente accessibili nella cartella:

eval_output

Proviamo?