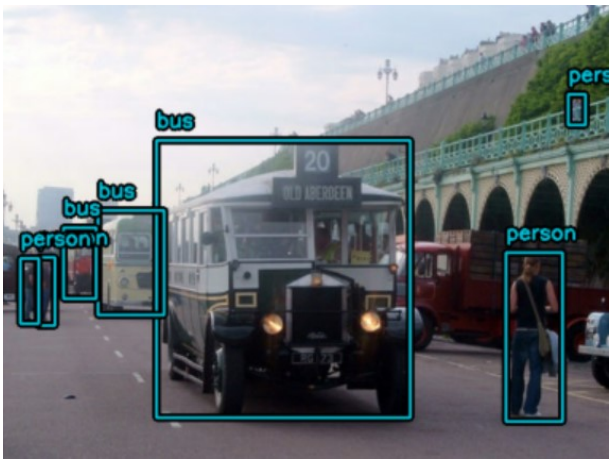
Elaborazione di Segnali Multimediali

Riconoscimento di oggetti

L.Verdoliva, D.Cozzolino

In questa esercitazione realizzeremo il riconsocimento di oggetti (Object Detection) tramite la rete RetinaNet (`https://arxiv.org/abs/1708.02002v2`). L'object detection si può vedere come un'estensione della classificazione, dove invece di classificare l'intera immagine, si identificano un numero arbitrario di classi all'interno dell'immagine. L'output è rappresentato da riquadri (box) che delimitano gli oggetti di interesse presenti nell'immagine. Per ogni riquadro viene fornito una relativa etichetta di classificazione, come nell'esempio riportato in figura.



Per questa esercitazione useremo anche le funzione delle librerie `keras_cv` e `easy_cv_dataset` che vanno installate con la seguente istruzione:

```
!pip install --upgrade git+https://github.com/davin11/easy-cv-dataset keras-cv
```

1 Preparazione dei dati

Utilizzeremo il dataset PascalVOC 2007 (`http://host.robots.ox.ac.uk/pascal/VOC/voc2007/`), dove i box sono etichettati in 20 differenti classi:

- *Persone*: person
- *Animali*: bird, cat, cow, dog, horse, sheep
- *Veicoli*: aeroplane, bicycle, boat, bus, car, motorbike, train
- *Oggetti*: bottle, chair, dining table, potted plant, sofa, tvmonitor

Su Colab potete direttamente eseguire le seguenti istruzioni per scaricare il training-set, il validation-set e il test-set di PascalVOC 2007:

```
SITE="https://raw.githubusercontent.com/davin11/easy-cv-dataset/master"
!wget -nc {SITE}/examples/object_detection/voc2007_objdet_test.csv
!wget -nc {SITE}/examples/object_detection/voc2007_objdet_train.csv
!wget -nc {SITE}/examples/object_detection/voc2007_objdet_val.csv
!wget -nc {SITE}/examples/object_detection/voc2007_download.sh
!bash voc2007_download.sh
```

A questo punto avrete due cartelle denominate "voc2007_trainval" e "voc2007_test" che contengono le immagini del dataset e tre file CSV (Comma-Separated Values): "voc2007_objdet_train.csv", "voc2007_objdet_val.csv", "voc2007_objdet_test.csv" rispettivamente per il training, validation e test set. I file CSV per l'Object Detection contengono la posizione e la classe dei box ed hanno 6 colonne: *image* con i filepath delle immagini, *xmin* con la coordinata orizzontale del pixel in alto a sinistra di ogni box, *ymin* con la coordinata verticale del pixel in alto a sinistra di ogni box, *xmax* con la coordinata orizzontale del pixel in basso a destra di ogni box, *ymax* con la coordinata verticale del pixel in basso a destra di ogni box, *class* contiene l'informazione della classe per ogni box. Di seguito un estratto del file "voc2007_objdet_test.csv":

```
image,xmin,ymin,xmax,ymax,class
./voc2007_test/VOCdevkit/VOC2007/JPEGImages/002118.jpg,1,288,189,334,car
./voc2007_test/VOCdevkit/VOC2007/JPEGImages/002118.jpg,215,109,487,211,car
./voc2007_test/VOCdevkit/VOC2007/JPEGImages/002118.jpg,24,88,102,113,car
./voc2007_test/VOCdevkit/VOC2007/JPEGImages/009083.jpg,4,160,497,318,aeroplane
./voc2007_test/VOCdevkit/VOC2007/JPEGImages/005800.jpg,1,108,249,348,bus
./voc2007_test/VOCdevkit/VOC2007/JPEGImages/005800.jpg,23,3,465,375,person
```

Nel caso un'immagine abbia più box, va indicata una riga per ogni box, come nell'esempio precedente. Adesso, utilizzando la funzione `image_objdetect_dataset_from_dataframe`, possiamo indicare come preparare le immagini per ogni set.

```
BATCH_SIZE=4
IMAGE_SIZE=640
BOX_FORMAT="xywh"

from easy_cv_dataset import image_objdetect_dataset_from_dataframe
from keras_cv.layers import Resizing, RandomRotation, RandomFlip

pre_processing = Resizing(IMAGE_SIZE, IMAGE_SIZE,
                          pad_to_aspect_ratio=True,
                          bounding_box_format=BOX_FORMAT)
augmenter = keras.Sequential(layers=[
                  RandomRotation((-20, 20), bounding_box_format=BOX_FORMAT),
                  RandomFlip("horizontal", bounding_box_format=BOX_FORMAT),
                          ])
```

```
print("test-set")
test_ds  = image_objdetect_dataset_from_dataframe("voc2007_objdet_test.csv",
                                        bounding_box_format=BOX_FORMAT,
                                        pre_batching_processing=pre_processing,
                                        shuffle=False, batch_size=BATCH_SIZE,
                                        do_normalization=False)

print("trainig-set")
train_ds = image_objdetect_dataset_from_dataframe("voc2007_objdet_train.csv",
                                        bounding_box_format=BOX_FORMAT,
                                        pre_batching_processing=pre_processing,
                                        shuffle=True, batch_size=BATCH_SIZE,
                                        post_batching_processing=augmenter,
                                        do_normalization=False)

print("validetion-set")
valid_ds = image_objdetect_dataset_from_dataframe("voc2007_objdet_val.csv",
                                        bounding_box_format=BOX_FORMAT,
                                        pre_batching_processing=pre_processing,
                                        shuffle=False, batch_size=BATCH_SIZE,
                                        do_normalization=False)
```

La funzione `image_objdetect_dataset_from_dataframe` richiede come primo parametro il file CSV. Il secondo parametro `bounding_box_format` indica il formato dei box. In questa esercitazione, useremo il formato "xywh" in quanto RetinaNet risulta compatibile solo con questo formato. Gli altri parametri della funzione `image_objdetect_dataset_from_dataframe` sono gli stessi della funzione `image_classification_dataset_from_dataframe` già visti in esercitazioni precedenti.

Notate che per tutte le immagini dei tre dataset è stata prevista un'operazione di ridimensionamento a 640×640 pixel in quanto le immagini dei PascalVOC hanno dimensioni diverse. Inoltre, per utilizzare l'architettura RetinaNet, è necessario che le immagini abbiamo una dimensione divisibile per 64. Il parametro `do_normalization` è impostato a `False` in quanto già la rete RetinaNet prevede al suo interno la normalizzazione delle immagini. Solo per il dataset di training sono previste anche operazioni di Data Augmentation. Utilizziamo le seguenti istruzioni per visualizzare alcune esempi del test set:

```
from keras_cv.visualization import plot_bounding_box_gallery
for images, boxes in test_ds.take(1): # prende il primo batch del test-set
  plot_bounding_box_gallery(     # funzione per visualizzare immagine e box
    images, y_true=boxes,
    value_range=(0,255),
    cols=BATCH_SIZE//2, rows=2,
    scale=5, font_scale=0.7,
    bounding_box_format=BOX_FORMAT,
    class_mapping=dict(enumerate(test_ds.class_names)),
  )
```

2 Definizione della rete neurale

Usiamo le funzioni di KerasCV per creare un rete RetinaNet con encoder ResNet50, quest'ultimo pre-addestrato su ImageNet.

```
from keras_cv.models import RetinaNet
from keras_cv.layers import MultiClassNonMaxSuppression
model = RetinaNet.from_preset("resnet50_imagenet", num_classes=20,
                              bounding_box_format=BOX_FORMAT)
model.prediction_decoder = MultiClassNonMaxSuppression(
                              bounding_box_format=BOX_FORMAT, from_logits=True,
                              confidence_threshold=0.5, iou_threshold=0.5)
```

Il parametro `num_classes` indica il numero di classi impostato a 20 coerentemente con il dataset usato. Alla rete va anche fornita la procedura di Non-Maximum Suppression (NMS) che risolve il problema della rilevazione di più box per lo stesso oggetto. Del NMS, il parametro `confidence_threshold` indica il valore minimo di confidenza per poter accettare un box. Mentre `iou_threshold` controlla la soglia di IoU per considerare due box sovrapposti e poterne eliminare uno.

Per ridurre il rischio di over-fitting, possiamo non addestrare i primi strati della rete. Per non addestrare l'intero encoder utilizzate il seguente codice:

```
model.backbone.trainable = False
```

3 Addestramento e Testing

In questa esercitazione utilizziamo un ottimizzatore SGD con momentum e saturazione del modulo del gradiente. Per il learning-rate useremo una procedura che lo ridurrà di 10 volte dopo le prime 5 epoche e di ulteriori 10 volte dopo ulteriori 5 epoche.

```
base_lr = 0.005
from tensorflow.keras import optimizers
lr_decay = optimizers.schedules.PiecewiseConstantDecay(  # schedulatore del lr
   boundaries=[5*len(train_ds), 10*len(train_ds)],
   values=[base_lr, 0.1 * base_lr, 0.01 * base_lr],
)
optimizer = optimizers.SGD(
   learning_rate=lr_decay, momentum=0.9, global_clipnorm=10.0
)
```

Per i problemi di object detection dobbiamo definire due loss function, una relativa alle localizzazione dei box e l'altra alla classificazione dei box. Per la localizzazione dei box useremo la distanza smoothL1, mentre per la classificazione useremo la focal loss che è una variante della cross-entropy loss, che pone un peso maggiore sugli elementi difficili da classificare. Per definire le loss usiamo il metodo `compile`:

```
model.compile(
  box_loss="smoothl1",
  classification_loss="focal",
  optimizer=optimizer)
```

Notate che nel metodo `compile` non abbiamo definito nessuna metrica. La metrica ampiamente utilizzata nell'object detection è la mean Average-Precision (mAP), ma risulta estremamente onerosa valutarla sul training set, pertanto la valuteremo solo sul validation set utilizzando l'opzione `callbacks` del metodo `fit`. Avviamo il training:

```
from easy_cv_dataset.metrics import EvaluateMAPmetricsCallback
model.fit(train_ds, epochs=10,
  callbacks=[EvaluateMAPmetricsCallback(valid_ds, BOX_FORMAT),],
)
```

Ricordate di salvare i pesi dopo l'addestramento. Potete anche caricare la rete già addestrata su PascalVOC 2007 con le seguente istruzione:

```
model = RetinaNet.from_preset("retinanet_resnet50_pascalvoc", num_classes=20,
                              bounding_box_format=BOX_FORMAT)
```

Utilizziamo le seguenti istruzioni per vedere il risultato della rete su alcuni esempi del test set:

```
from keras_cv.visualization import plot_bounding_box_gallery
for images, boxes in test_ds.take(1): # prende il primo batch del test-set
  boxes_pred = model.predict(images) # esegue le rete sul batch
  plot_bounding_box_gallery(     # funzione per visualizzare immagine e box
    images, y_pred=boxes_pred,
    value_range=(0,255),
    cols=BATCH_SIZE//2, rows=2,
    scale=5, font_scale=0.7,
    bounding_box_format=BOX_FORMAT,
    class_mapping=dict(enumerate(test_ds.class_names)),
  )
```

Infine, utilizziamo la funzione `compute_mAP_metrics` per valutare la mean Average-Precision (mAP) sul test set

```
from easy_cv_dataset.metrics import compute_mAP_metrics
print(compute_mAP_metrics(model, test_ds, BOX_FORMAT))
```