Elaborazione di Segnali Multimediali

Segmentazione semantica

L.Verdoliva, D.Cozzolino

L'obiettivo delle segmentazione è associare ogni pixel dell'immagine ad una classe ben definita fornendo una mappa di etichette. In questa esercitazione useremo U-Net per realizzare la segmentazione semantica. A tale scopo avremo bisogno di funzioni che appartengono alle 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
```

Inoltre, va scaricato il file `unet.py` tramite le seguenti istruzioni:

```
SITE="https://raw.githubusercontent.com/davin11/easy-cv-dataset/master"
!wget -nc {SITE}/examples/segmentation/unet.py
```

1 Preparazione dei dati

Utilizzeremo il dataset Oxford-IIIT Pet Dataset (`https://www.robots.ox.ac.uk/~vgg/data/pets/`), dove le immagini sono foto di cani e gatti. Per ogni immagine è disponibile una mappa di segmentazione per individuare il cane o gatto (foreground) dallo sfondo (background). Su Colab potete direttamente eseguire le seguenti istruzioni per scaricare il dataset che risulta essere già separato in training, validation e test set:

```
!wget -q -c https://www.grip.unina.it/download/guide_TF/oxford_pets_dataset.zip
!unzip -q -n oxford_pets_dataset.zip
```

A questo punto, vi troverete una cartella denominata "oxford_pets_dataset" che contiene le immagini del dataset e le corrispondenti mappe di segmentazione. Inoltre nella cartella troverete tre file CSV (Comma-Separated Values): "tab_train.csv", "tab_val.csv" e "tab_test.csv", rispettivamente per il training, validation e test set. I file con estensione CSV (Comma-Separated Values) sono file testuali che contengono una tabella e adottando il carattere virgola (,) per delimitarne le colonne. Nel nostro caso questi file contengono due colonne: *image* con i filepath delle immagini e *segmentation_mask* con quelli delle mappe di segmentazione. Adesso, utilizzando la funzione `image_segmentation_dataset_from_dataframe` prepariamo le immagini per training, validation e test set.

```
BATCH_SIZE=16
IMAGE_SIZE=160
class_names = ["Background", "Foreground",]
num_classes = 2

from easy_cv_dataset import image_segmentation_dataset_from_dataframe
from keras_cv.layers import Resizing, RandomColorDegeneration, RandomRotation

pre_processing = Resizing(IMAGE_SIZE, IMAGE_SIZE)
augmenter = keras.Sequential(layers=[
                  RandomColorDegeneration(0.5),
                  RandomRotation((-20, 20)),
                          ])

print("test-set")
test_ds  = image_segmentation_dataset_from_dataframe(
                                  dataframe="oxford_pets_dataset/tab_test.csv",
                                  class_mode='categorical',
                                  class_names=class_names,
                                  pre_batching_processing=pre_processing,
                                  shuffle=False, batch_size=BATCH_SIZE,
                                  do_normalization=False,
                                  )

print("trainig-set")
train_ds = image_segmentation_dataset_from_dataframe(
                                  dataframe="oxford_pets_dataset/tab_train.csv",
                                  class_mode='categorical',
                                  class_names=class_names,
                                  pre_batching_processing=pre_processing,
                                  shuffle=True, batch_size=BATCH_SIZE,
                                  post_batching_processing=augmenter,
                                  do_normalization=False,
                                  )

print("validetion-set")
valid_ds = image_segmentation_dataset_from_dataframe(
                                  dataframe="oxford_pets_dataset/tab_val.csv",
                                  class_mode='categorical',
                                  class_names=class_names,
                                  pre_batching_processing=pre_processing,
                                  shuffle=False, batch_size=BATCH_SIZE,
                                  do_normalization=False,
                                  )
```

La funzione `image_segmentation_dataset_from_dataframe` richiede come primo parametro il file CSV. Il secondo parametro `class_mode` indica il formato in cui convertire le mappe di segmentazione che è imposto a `'categorical'`. Infine, il terzo parametro `class_names` è una lista dei nomi delle classi in ordine coerente a come sono codificati nelle mappe di segmentazione.

Gli altri parametri della funzione `image_segmentation_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 160×160 pixel in modo che abbiano tutte la stessa dimensione. Inoltre, per utilizzare l'architettura U-Net, è necessario che le immagini abbiamo una dimensione divisibile per 16. Il parametro `do_normalization` è impostato a `False` dato che la rete che definiremo prevede già al suo interno la normalizzazione delle immagini. 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_segmentation_mask_gallery
from tensorflow.keras import backend as ops

for images, segms in test_ds.take(1): # prende il primo batch del test-set
  plot_segmentation_mask_gallery(     # per visualizzare immagini e mappe
    images, y_true=ops.argmax(segms,-1)[...,None],
    value_range=(0,255), cols=BATCH_SIZE//4, rows=4, num_classes=num_classes
)
```

2 Definizione della rete neurale

L'architettura U-Net è mostrata in figura 1 ed è formata da due parti: l'*encoder*, detto anche *contracting path*, e il *decoder*, chiamato *expansive path*. L'*encoder* segue l'architettura tipica di una rete neurale convoluzionale composta da strati convoluzionali e da strati di pooling. Gli strati di pooling riducono le dimensione spaziali delle *feature map* dimezzandole ogni volta. A valle dell'*encoder* le *feature map* avranno una risoluzione spaziale pari ad un sedicesimo di quella dell'immagine di partenza. Il *decoder*, oltre ai classici strati convoluzionali, prevede le operazioni dette *up-conv* che servono a raddoppiano ogni volta le dimensioni spaziali. L'operazione di *up-conv* è formata dalla cascata di un'interpolazione nearest-neighbor ed una convoluzione spaziale 2×2 pixel.

L'architettura U-Net prevede anche delle scorciatoie tra le due parti, dette (*skip connection*), e rappresentate delle frecce grigie nell'illustrazione di figura 1. In particolare, le feature-map calcolate ai vari livelli dell'*encoder* vengono fornite in ingresso ai veri livelli del *decoder* concatenandole con quelle del livello precedente. L'utilizzo delle *skip connection* aiuta a migliorare la precisione della mappa di segmentazione. Notate che l'architettura U-Net è una fully convolutional network, infatti non prevede strati *fully connected*. Quindi si può applicare ad immagini di diverse dimensioni. L'unico vincolo è che il numero di righe e colonne dell'immagine di ingresso siano multipli di 16.

Nel file "unet.py" è presente la funzione `Unet` per istanziare l'architettura U-Net in Keras. In questa esercitazione non useremo l'architettura classica di U-Net, ma la parte di encoder sarà formata dalla rete MobileNetV3 pre-addestrata su ImageNet.

```
from unet import Unet
from keras_cv.models import MobileNetV3Backbone

encoder = MobileNetV3Backbone.from_preset("mobilenet_v3_large_imagenet",
                          input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),
                          include_rescaling=True)
model = Unet(num_classes = 2, backbone=encoder, use_batchnorm=True)
```

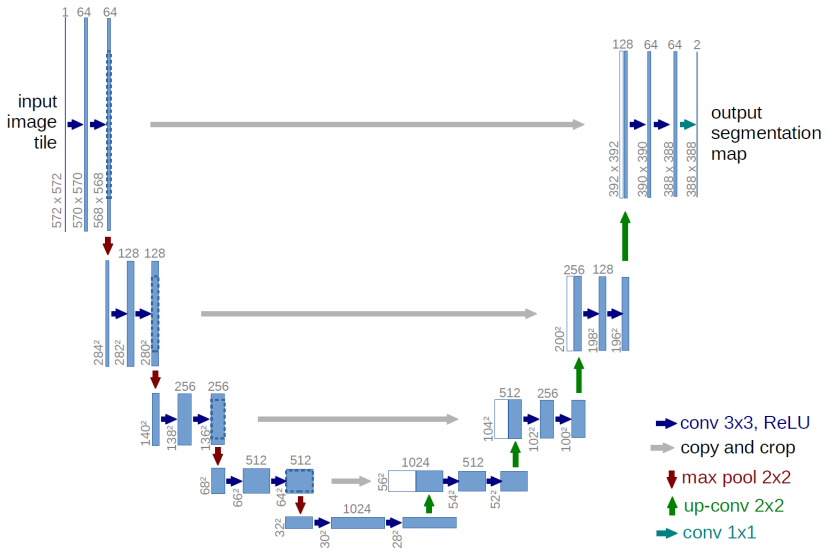

Figura 1: Architettura di U-Net. Le frecce indicano le diverse operazioni. Le caselle indicano i risultati delle varie operazioni. Il numero di canali è indicato sulla parte superiore di ogni casella.

.

Il parametro `num_classes` indica il numero di classi impostato a due coerentemente con il dataset usato. Il parametro `use_batchnorm` indica di usare o meno i livelli di normalizzazione nell'architettura del decoder. Per ridurre il rischio di over-fitting, possiamo non addestrare i primi strati della rete. Se non volete addestrare l'intero encoder utilizzate il seguente codice:

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

3 Addestramento e Testing

In questa esercitazione, utilizziamo l'ottimizzatore Nadam e come loss function la Focal Loss che è una variante della cross-entropy loss, molto utilizzata nei problemi di segmentazione.

```
from keras.optimizers impot Nadam
from keras.metrics import MeanIoU
model.compile(
  loss='categorical_focal_crossentropy',
  optimizer=Nadam(learning_rate=0.0001),
  metrics=[MeanIoU(num_classes=2, sparse_y_true=False, sparse_y_pred=False),],
)
```

Come indice prestazionale utilizzeremo la Intersection over Union (IoU) mediata sulle due classi. Non utilizziamo l'accuratezza perché per la segmentazione non è un indice affidabile in quanto è influenzato dalla dimensione della regione da segmentare rispetto a quella dell'immagine. I parametri `sparse_y_true=False` e

sparse_y_pred=False indicano alla funzione che calcolerà la IoU che rispettivamente le mappe di segmentazione e l'output della rete sono nel formato 'categorical'.
Realizziamo il training tramite la funzione fit:

```
model.fit(train_ds, epochs=2, validation_data=valid_ds)
model.save_weights('unet_weights.hdf5')
```

Dopo l'addestramento il metodo save_weights è utilizzato per salvare i parametri della rete in un file nel formato HDF5. Utilizziamo le seguenti istruzioni per vedere il risultato della rete su alcuni esempi del test set:

```
from keras_cv.visualization import plot_segmentation_mask_gallery
for images, segms in test_ds.take(1): # prende il primo batch del test-set
  pred = model(images)
  plot_segmentation_mask_gallery(     # visualizza immagini e mappe
    images,
    y_true=ops.argmax(segms,-1)[...,None],
    y_pred=ops.argmax(pred,-1)[...,None],
    value_range=(0,255), cols=BATCH_SIZE//4, rows=4, num_classes=num_classes
  )
```

Infine, utilizziamo la funzione evaluate per valutare l'intersection over union mediata su tutto il test set.

```
metrics = model.evaluate(test_ds, return_dict=True)
print(metrics)
```