LAB 4

Topics: Global Average Pooling (GAP) + class activation maps

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os
import tensorflow as tf
import numpy as np

SEED = 1234
tf.random.set_seed(SEED)

# Get current working directory
cwd = os.getcwd()
```

In [3]:
```
# Run this cell only if you are using Colab with Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

apply_data_augmentation = False

# Create training ImageDataGenerator object
if apply_data_augmentation:
    train_data_gen = ImageDataGenerator(rotation_range=10,
                                        width_shift_range=10,
                                        height_shift_range=10,
                                        zoom_range=0.3,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        fill_mode='constant',
                                        cval=0,
                                        rescale=1./255)
else:
    train_data_gen = ImageDataGenerator(rescale=1./255)

# Create validation and test ImageDataGenerator objects
valid_data_gen = ImageDataGenerator(rescale=1./255)
test_data_gen = ImageDataGenerator(rescale=1./255)
```

In [5]:
```
# Run this cell only if you are using Colab with Drive
!unzip '/content/drive/My Drive/MaskDataset.zip'
```

```
Archive:  /content/drive/My Drive/MaskDataset.zip
   creating: MaskDataset/
   creating: MaskDataset/test/
   creating: MaskDataset/test/with_mask/
  inflating: MaskDataset/test/with_mask/1-with-mask.jpg
  inflating: MaskDataset/test/with_mask/106-with-mask.jpg
  inflating: ..
```

- training
- test
- val

- with-mask (1)
- without-mask (0

In [6]:
```
!ls /content
```

```
drive  MaskDataset  sample_data
```

In [7]:
```
# Create generators to read images from dataset directory
# -----------------------------------------------------------------------------
dataset_dir = os.path.join(cwd, 'MaskDataset')

# Batch size
bs = 32

# img shape
img_h = 256
img_w = 256

num_classes=2

decide_class_indices = True
if decide_class_indices:
    classes = ['without_mask',      # 0
               'with_mask',         # 1
              ]
else:
    classes=None

# Training
training_dir = os.path.join(dataset_dir, 'train')
train_gen = train_data_gen.flow_from_directory(training_dir,
                                               batch_size=bs,
                                               target_size=[img_h, img_w],
                                               classes=classes,
                                               class_mode='categorical',
                                               shuffle=True,
                                               seed=SEED)  # targets are directly converted into one-hot vectors

# Validation
validation_dir = os.path.join(dataset_dir, 'val')
valid_gen = valid_data_gen.flow_from_directory(validation_dir,
                                               batch_size=bs,
                                               target_size=[img_h, img_w],
                                               classes=classes,
                                               class_mode='categorical',
                                               shuffle=False,
                                               seed=SEED)

# Test
test_dir = os.path.join(dataset_dir, 'test')
test_gen = test_data_gen.flow_from_directory(test_dir,
```

in this way we decide the order of the classes ("without mask" will be associated to 0 and "with mask" will be associated to 1)

```
                                                batch_size=bs,
                                                target_size=[img_h, img_w],
                                                classes=classes,
                                                class_mode='categorical',
                                                shuffle=False,
                                                seed=SEED)
```

```
Found 1315 images belonging to 2 classes.
Found 142 images belonging to 2 classes.
Found 194 images belonging to 2 classes.
```

In [8]:
```
# Check how keras assigned the labels
train_gen.class_indices
```

Out[8]:
```
{'with_mask': 1, 'without_mask': 0}
```

In [9]:
```
# Create Dataset objects
# ------------------------------------------------------------------------------
# Training
train_dataset = tf.data.Dataset.from_generator(lambda: train_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
train_dataset = train_dataset.repeat()

# Validation
# ------------------------------------------------------------------------------
valid_dataset = tf.data.Dataset.from_generator(lambda: valid_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
valid_dataset = valid_dataset.repeat()

# Test
# ------------------------------------------------------------------------------
test_dataset = tf.data.Dataset.from_generator(lambda: test_gen,
                                              output_types=(tf.float32, tf.float32),
                                              output_shapes=([None, img_h, img_w, 3], [None, num_classes]))
test_dataset = valid_dataset.repeat()
```

In [10]:
```
# Architecture: Features extraction -> Classifier
start_f = 8
depth   = 5
model   = tf.keras.Sequential()
encoder = tf.keras.Sequential()

# Features extraction
for i in range(depth):

    if i == 0:
        input_shape = [img_h, img_w, 3]
    else:
        input_shape=[None]

    # Conv block: Conv2D -> Activation -> Pooling
    encoder.add(tf.keras.layers.Conv2D(filters=start_f,
                                       kernel_size=(3, 3),
                                       strides=(1, 1),
                                       padding='same',
                                       input_shape=input_shape))
    encoder.add(tf.keras.layers.ReLU())

    if i < depth -1:
      encoder.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))

    start_f *= 2

model.add(encoder)

# GAP
model.add(tf.keras.layers.GlobalAveragePooling2D())

# Classifier
model.add(tf.keras.layers.Dense(units=num_classes, activation='softmax'))
```

If we're at the last convolutional block we're not doing the maxpooling

instead of the **Flatten**

In [11]:
```
# Visualize created model as a table
model.summary()

# Visualize initialized weights
# model.weights
```

the summary it global and so we don't see what it in the first Sequential()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 16, 16, 128)       98384
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 258
=================================================================
Total params: 98,642
Trainable params: 98,642
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
model.layers[-1].weights[0].shape
```

Out[12]:
```
TensorShape([128, 2])
```

In [13]:
```
# Optimization params
# ------------------------------------------------------------------------------
# Loss
```

since it's a binary task we could also use a single neuron as classifier (and a sigmoid), in that case we would use the **Binary Crossentropy()** as loss

```
loss = tf.keras.losses.CategoricalCrossentropy()

# Learning rate
lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)


# Validation metrics
# ------------------------------------------------------------------------------
metrics = ['accuracy']

# Compile Model
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

In [14]:
```
%load_ext tensorboard
%tensorboard --logdir /content/drive/My\ Drive/Keras4/segmentation_experiments_GAP/#
```

Output hidden; open in https://colab.research.google.com to view.

In [15]:
```
import os
from datetime import datetime

cwd = os.getcwd()

exps_dir = os.path.join('/content/drive/My Drive/Keras4/', 'segmentation_experiments_GAP')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

model_name = 'CNN_GAP'

exp_dir = os.path.join(exps_dir, model_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)

callbacks = []

# Model checkpoint
# ------------------------------------------------------------------------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp_{epoch:02d}.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# ------------------------------------------------------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=1)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# ------------------------------------------------------------------------------
early_stop = True
if early_stop:
    es_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
    callbacks.append(es_callback)
```

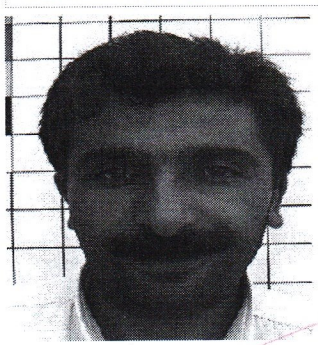
In [16]:
```
model.fit(x=train_dataset,
          epochs=100,  #### set repeat in training dataset
          steps_per_epoch=len(train_gen),
          validation_data=valid_dataset,
          validation_steps=len(valid_gen),
          callbacks=callbacks)

# How to visualize Tensorboard
# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

```
Epoch 1/100
42/42 [==============================] - 9s 216ms/step - loss: 0.6815 - accuracy: 0.5939 - val_loss: 0.6545 - val_accura
cy: 0.5845
Epoch 2/100
42/42 [==============================] - 9s 210ms/step - loss: 0.5823 - accuracy: 0.6935 - val_loss: 0.6718 - val_accura
cy: 0.5915
Epoch 3/100
42/42 [==============================] - 9s 208ms/step - loss: 0.5560 - accuracy: 0.7255 - val_loss: 0.4881 - val_accura
cy: 0.7887
Epoch 4/100
42/42 [==============================] - 9s 206ms/step - loss: 0.4507 - accuracy: 0.8000 - val_loss: 0.2880 - val_accura
cy: 0.9437
Epoch 5/100
42/42 [==============================] - 9s 206ms/step - loss: 0.3829 - accuracy: 0.8297 - val_loss: 0.2646 - val_accura
cy: 0.9296
Epoch 6/100
42/42 [==============================] - 9s 206ms/step - loss: 0.2772 - accuracy: 0.8943 - val_loss: 0.1578 - val_accura
cy: 0.9507
Epoch 7/100
42/42 [==============================] - 9s 204ms/step - loss: 0.1834 - accuracy: 0.9422 - val_loss: 0.1284 - val_accura
cy: 0.9366
Epoch 8/100
42/42 [==============================] - 8s 199ms/step - loss: 0.2393 - accuracy: 0.9095 - val_loss: 0.0877 - val_accura
cy: 0.9789
Epoch 9/100
```

```
42/42 [==============================] - 9s 208ms/step - loss: 0.1161 - accuracy: 0.9658 - val_loss: 0.0591 - val_accura
cy: 0.9718
Epoch 10/100
42/42 [==============================] - 9s 211ms/step - loss: 0.1178 - accuracy: 0.9529 - val_loss: 0.0560 - val_accura
cy: 0.9859
Epoch 11/100
42/42 [==============================] - 8s 201ms/step - loss: 0.1205 - accuracy: 0.9506 - val_loss: 0.0543 - val_accura
cy: 0.9859
Epoch 12/100
42/42 [==============================] - 9s 207ms/step - loss: 0.0845 - accuracy: 0.9757 - val_loss: 0.0351 - val_accura
cy: 0.9859
Epoch 13/100
42/42 [==============================] - 9s 210ms/step - loss: 0.0905 - accuracy: 0.9681 - val_loss: 0.0354 - val_accura
cy: 0.9859
Epoch 14/100
42/42 [==============================] - 9s 206ms/step - loss: 0.1006 - accuracy: 0.9650 - val_loss: 0.0308 - val_accura
cy: 0.9930
Epoch 15/100
42/42 [==============================] - 9s 212ms/step - loss: 0.1576 - accuracy: 0.9399 - val_loss: 0.1638 - val_accura
cy: 0.9437
Epoch 16/100
42/42 [==============================] - 9s 214ms/step - loss: 0.0983 - accuracy: 0.9681 - val_loss: 0.1070 - val_accura
cy: 0.9718
Epoch 17/100
42/42 [==============================] - 9s 207ms/step - loss: 0.0834 - accuracy: 0.9734 - val_loss: 0.0344 - val_accura
cy: 0.9859
Epoch 18/100
42/42 [==============================] - 9s 203ms/step - loss: 0.0625 - accuracy: 0.9810 - val_loss: 0.0182 - val_accura
cy: 1.0000
Epoch 19/100
42/42 [==============================] - 9s 209ms/step - loss: 0.0646 - accuracy: 0.9817 - val_loss: 0.0195 - val_accura
cy: 1.0000
Epoch 20/100
42/42 [==============================] - 9s 206ms/step - loss: 0.0561 - accuracy: 0.9840 - val_loss: 0.0188 - val_accura
cy: 1.0000
Epoch 21/100
42/42 [==============================] - 9s 207ms/step - loss: 0.0619 - accuracy: 0.9795 - val_loss: 0.0158 - val_accura
cy: 1.0000
Epoch 22/100
42/42 [==============================] - 9s 211ms/step - loss: 0.0648 - accuracy: 0.9772 - val_loss: 0.0381 - val_accura
cy: 0.9789
Epoch 23/100
42/42 [==============================] - 9s 209ms/step - loss: 0.0465 - accuracy: 0.9856 - val_loss: 0.0137 - val_accura
cy: 1.0000
Epoch 24/100
42/42 [==============================] - 9s 205ms/step - loss: 0.0593 - accuracy: 0.9795 - val_loss: 0.0129 - val_accura
cy: 1.0000
Epoch 25/100
42/42 [==============================] - 9s 208ms/step - loss: 0.0618 - accuracy: 0.9795 - val_loss: 0.0401 - val_accura
cy: 0.9859
Epoch 26/100
42/42 [==============================] - 8s 202ms/step - loss: 0.0475 - accuracy: 0.9856 - val_loss: 0.0135 - val_accura
cy: 1.0000
Epoch 27/100
42/42 [==============================] - 9s 210ms/step - loss: 0.0414 - accuracy: 0.9871 - val_loss: 0.0115 - val_accura
cy: 1.0000
Epoch 28/100
39/42 [==========================>...] - ETA: 0s - loss: 0.0540 - accuracy: 0.9808
```

In [24]:
```
# Let's visualize the activations of our network
from PIL import Image

test_iter = iter(test_dataset)

# Get a test image with a mask
test_img, target = next(test_iter)
test_img = test_img[0]

# Visualize the image
Image.fromarray(np.uint8(np.array(test_img)*255.))
```

Out[24]:



we create a new model by giving the old model inputs and as outputs some layers of the old model. We're creating a new model but the weights are the one we trained before. But now the new model allows to get 2 outputs: the model output and the features map at the end of the encoder.

In [25]:
```
np.argmax(target, 1)
```

Out[25]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

prediction

output from the last convolutional layer (model.layers[0] is the encoder, get_output_at(-1) gets the last convolutional layer)

In [26]:
```
out_model = tf.keras.Model(inputs=model.input, outputs=[model.output, model.layers[0].get_output_at(-1)])
```

Now we want to localize the object in the original input space (and so we have to re-size)

In [27]:
```
resize_feature = tf.keras.layers.experimental.preprocessing.Resizing(
    256, 256, interpolation="bilinear")
```

In [28]:
```
softmax_out, last_enc_feature = out_model(tf.expand_dims(test_img, 0))
pred                          = tf.argmax(softmax_out, 1)
resized_feature               = resize_feature(last_enc_feature)
```

→ we obtain the prediction and the encoder features

→ here we extract the predicted class

In [29]:
```
# Get the weights in input to the output neuron corresponding to the 'with_mask' class
mask_weights = model.layers[-1].weights[0][:, 1]
```

we take the weights in the fully connected layer (the classificator)

CLASS ACTIVATION MAP

```
resized_feature = tf.reshape(resized_feature, shape=[256*256, resized_feature.shape[-1]])
cam = tf.linalg.matmul(resized_feature, tf.expand_dims(mask_weights, -1))
cam = tf.reshape(cam, shape=[256, 256])
```

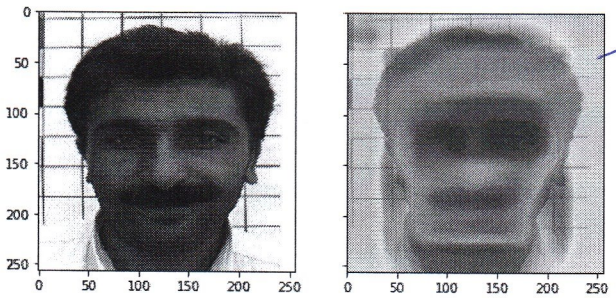
we're transforming the resized-features into a matrix
dimension of the matrix: 256 x 256 x 128

In [30]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows=1, ncols=2, sharey=True, sharex=True, figsize=plt.figaspect(0.5))
ax[0].imshow(test_img)
ax[1].imshow(test_img, alpha=0.5)
ax[1].imshow(cam, cmap='jet', alpha=0.5)
plt.show()
```

Out[30]: <matplotlib.image.AxesImage at 0x7fbc0279f7b8>

Out[30]: <matplotlib.image.AxesImage at 0x7fbc027296a0>

Out[30]: <matplotlib.image.AxesImage at 0x7fbc5007c160>



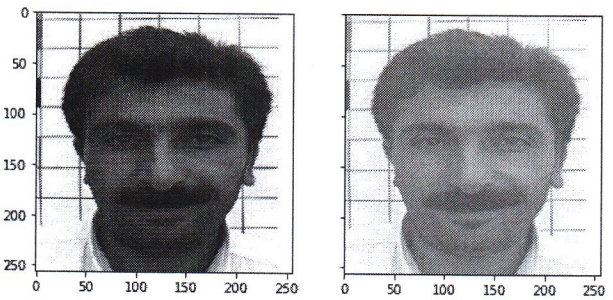
without training for object-detection or segmentation, this method is giving (by only inspecting activations in the features map) a way to localize the object we're interested in (more precisely: this method shows us what influences the most in the decision of the classification)
(However it's a very coarse localization)

In [33]:
```
# we can obtain a course segmentation of the mask by thresholding
# For example we can use thr = 0.5 * max(cam)

fig, ax = plt.subplots(nrows=1, ncols=2, sharey=True, sharex=True, figsize=plt.figaspect(0.5))
ax[0].imshow(test_img)
ax[1].imshow(test_img, alpha=0.5)
ax[1].imshow(cam > 0.5 * np.max(cam), cmap='gray', alpha=0.5)
plt.show()
```

— in this image we have no mask

Out[33]: <matplotlib.image.AxesImage at 0x7fbc50072a58>

Out[33]: <matplotlib.image.AxesImage at 0x7fbbf876d748>



Topics: Image Segmentation through Encoder-Decoder network (for **binary**-classification)
(we distinguish
- background
- foreground)

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os
import tensorflow as tf
import numpy as np

# Set the seed for random operations.
# This let our experiments to be reproducible.
SEED = 1234
tf.random.set_seed(SEED)
```

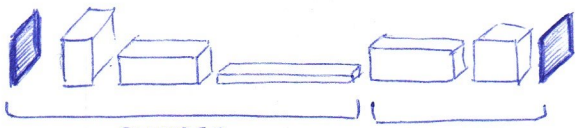
In [3]:
```
cwd = os.getcwd()
```

In [4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [5]:
```
!unzip /content/drive/My\ Drive/SegmentationDataset.zip
```

```
Archive:  /content/drive/My Drive/SegmentationDataset.zip
   creating: SegmentationDataset/
   creating: SegmentationDataset/training/
   creating: SegmentationDataset/training/images/
   creating: SegmentationDataset/training/images/img/
  inflating: SegmentationDataset/training/images/img/1408.tif
  inflating: SegmentationDataset/training/images/img/6857.tif
  inflating: ..
```



Example: Image Segmentation

Build segmentation

In [6]:
```
# ImageDataGenerator
# ------------------------------------------------------------------------------
from tensorflow.keras.preprocessing.image import ImageDataGenerator
apply_data_augmentation = True

# Create training ImageDataGenerator object
# We need two different generators for images and corresponding masks
if apply_data_augmentation:
    train_img_data_gen = ImageDataGenerator(rotation_range=30,
                                            width_shift_range=10,
                                            height_shift_range=10,
                                            zoom_range=0.3,
                                            horizontal_flip=True,
                                            vertical_flip=True,
                                            fill_mode='reflect',
                                            rescale=1./255)
    train_mask_data_gen = ImageDataGenerator(rotation_range=30,
                                             width_shift_range=10,
                                             height_shift_range=10,
                                             zoom_range=0.3,
                                             horizontal_flip=True,
                                             vertical_flip=True,
                                             fill_mode='reflect')
else:
    train_img_data_gen = ImageDataGenerator(rescale=1./255)
    train_mask_data_gen = ImageDataGenerator()

# Create validation and test ImageDataGenerator objects
valid_img_data_gen = ImageDataGenerator(rescale=1./255)
valid_mask_data_gen = ImageDataGenerator()
test_img_data_gen = ImageDataGenerator(rescale=1./255)
test_mask_data_gen = ImageDataGenerator()
```

Winning approach: **U-NET.**
The key point of this architecture is that encoder and decoder are connected at the corresponding levels.

ImageDataGenerator for the **image** (RGB image)

ImageDataGenerator for the **mask** (corresponding to the previous image)
Notice that the mask does not need to be normalized

In [7]:
```
!ls /content/SegmentationDataset/training/images
```

```
img
```

In [8]:
```
# Create generators to read images from dataset directory
# ------------------------------------------------------------------------------
dataset_dir = os.path.join(cwd, 'SegmentationDataset')

# Batch size
bs = 16

# img shape
img_h = 256
img_w = 256
num_classes=3

# Training
# Two different generators for images and masks
# ATTENTION: here the seed is important!! We have to give the same SEED to both the generator
# to apply the same transformations/shuffling to images and corresponding masks
training_dir = os.path.join(dataset_dir, 'training')
train_img_gen = train_img_data_gen.flow_from_directory(os.path.join(training_dir, 'images'),
                                                       target_size=(img_h, img_w),
                                                       batch_size=bs,
                                                       class_mode=None,
                                                       shuffle=True,
                                                       interpolation='bilinear',
                                                       seed=SEED)
train_mask_gen = train_mask_data_gen.flow_from_directory(os.path.join(training_dir, 'masks'),
                                                         target_size=(img_h, img_w),
```

```
                                                    batch_size=bs,
                                                    class_mode=None,
                                                    shuffle=True,
                                                    interpolation='bilinear',
                                                    seed=SEED)
train_gen = zip(train_img_gen, train_mask_gen)

# Validation
validation_dir = os.path.join(dataset_dir, 'validation')
valid_img_gen = valid_img_data_gen.flow_from_directory(os.path.join(validation_dir, 'images'),
                                                       target_size=(img_h, img_w),
                                                       batch_size=bs,
                                                       class_mode=None,
                                                       shuffle=False,
                                                       interpolation='bilinear',
                                                       seed=SEED)
valid_mask_gen = valid_mask_data_gen.flow_from_directory(os.path.join(validation_dir, 'masks'),
                                                         target_size=(img_h, img_w),
                                                         batch_size=bs,
                                                         class_mode=None,
                                                         shuffle=False,
                                                         interpolation='bilinear',
                                                         seed=SEED)
valid_gen = zip(valid_img_gen, valid_mask_gen)
```

```
Found 500 images belonging to 1 classes.
Found 500 images belonging to 1 classes.
Found 100 images belonging to 1 classes.
Found 100 images belonging to 1 classes.
```

In [9]:
```
# Create Dataset objects
# ------------------------------------------------------------------------------
# Training
# ------------------------------------------------------------------------------
train_dataset = tf.data.Dataset.from_generator(lambda: train_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, img_h, img_w, 3]))

# This is needed to convert targets from H x W x 3 (RGB) to label mask H x W x 1.
# Thus, 255 values become True (with y_ > 0), i.e., RGB value [255, 255, 255] -> [True, True, True],
# and then we reduce it to a single value (reduce_any), i.e., RGB value [255, 255, 255] -> [True].
# Finally, cast boolean values to float to obtain 0-1 labels.
def prepare_target(x_, y_):
    return x_, tf.cast(tf.reduce_any(y_ > 0, axis=-1, keepdims=True), tf.float32)

train_dataset = train_dataset.map(prepare_target)
train_dataset = train_dataset.repeat()

# Validation
# ------------------------------------------------------------------------------
valid_dataset = tf.data.Dataset.from_generator(lambda: valid_gen,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([None, img_h, img_w, 3], [None, img_h, img_w, 3]))
valid_dataset = valid_dataset.map(prepare_target)
valid_dataset = valid_dataset.repeat()
```

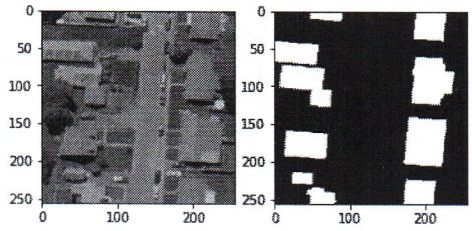
In [10]:
```
iterator = iter(train_dataset)
```

In [11]:
```
# Let's test data generator
# ------------------------------------------------------------------------------
import time
import matplotlib.pyplot as plt

%matplotlib inline

# Assign a color to each class
colors_dict = {}
colors_dict[1] = [255, 255, 255]  # foreground
colors_dict[0] = [0, 0, 0]  # background

iterator = iter(train_dataset)

fig, ax = plt.subplots(1, 2)

augmented_img, target = next(iterator)
augmented_img = augmented_img[0]   # First element
augmented_img = augmented_img * 255  # denormalize

target = np.array(target[0, ..., 0])   # First element (squeezing channel dimension)

print(np.unique(target))

# Assign colors (just for visualization)
target_img = np.zeros([target.shape[0], target.shape[1], 3])

target_img[np.where(target == 0)] = colors_dict[0]
target_img[np.where(target == 1)] = colors_dict[1]

ax[0].imshow(np.uint8(augmented_img))
ax[1].imshow(np.uint8(target_img))

plt.show()
```

```
[0. 1.]
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7fd69c1bb7b8>
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7fd66724efd0>
```



Convolutional Neural Network (CNN)

Encoder-Decoder

In [12]:
```
# Create Model
# ----------------------------------------------------------------------------------------
def create_model(depth, start_f, num_classes, dynamic_input_shape):
    model = tf.keras.Sequential()

    # Encoder (Same that we used for classification)
    # ----------------------------------------------------------------------------------------
    for i in range(depth):

        if i == 0:
            if dynamic_input_shape:
                input_shape = [None, None, 3]
            else:
                input_shape = [img_h, img_w, 3]
        else:
            input_shape=[None]

        model.add(tf.keras.layers.Conv2D(filters=start_f,
                                         kernel_size=(3, 3),
                                         strides=(1, 1),
                                         padding='same',
                                         input_shape=input_shape))
        model.add(tf.keras.layers.ReLU())
        model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))

        start_f *= 2

    # Bottleneck
    model.add(tf.keras.layers.Conv2D(filters=start_f, kernel_size=(3, 3), strides=(1, 1), padding='same'))
    model.add(tf.keras.layers.ReLU())

    start_f = start_f // 2

    # Decoder
    # ----------------------------------------------------------------------------------------
    for i in range(depth):
        model.add(tf.keras.layers.UpSampling2D(2, interpolation='bilinear'))
        model.add(tf.keras.layers.Conv2D(filters=start_f,
                                         kernel_size=(3, 3),
                                         strides=(1, 1),
                                         padding='same'))
        model.add(tf.keras.layers.ReLU())

        start_f = start_f // 2

    # Prediction Layer (not a fully connected anymore)
    # ----------------------------------------------------------------------------------------
    model.add(tf.keras.layers.Conv2D(filters=1,
                                     kernel_size=(1, 1),
                                     strides=(1, 1),
                                     padding='same',
                                     activation='sigmoid'))

    return model
```

this is the encoding (From now we have to re-construct)

In [13]:
```
model = create_model(depth=5,
                     start_f=8,
                     num_classes=1,
                     dynamic_input_shape=False)

# Visualize created model as a table
model.summary()

# Visualize initialized weights
# model.weights
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 8)	224
re_lu (ReLU)	(None, 256, 256, 8)	0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 8)	0
conv2d_1 (Conv2D)	(None, 128, 128, 16)	1168
re_lu_1 (ReLU)	(None, 128, 128, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 16)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	4640

re_lu_2 (ReLU)	(None, 64, 64, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 32)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	18496
re_lu_3 (ReLU)	(None, 32, 32, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	73856
re_lu_4 (ReLU)	(None, 16, 16, 128)	0
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 128)	0
conv2d_5 (Conv2D)	(None, 8, 8, 256)	295168
re_lu_5 (ReLU)	(None, 8, 8, 256)	0
up_sampling2d (UpSampling2D)	(None, 16, 16, 256)	0
conv2d_6 (Conv2D)	(None, 16, 16, 128)	295040
re_lu_6 (ReLU)	(None, 16, 16, 128)	0
up_sampling2d_1 (UpSampling2	(None, 32, 32, 128)	0
conv2d_7 (Conv2D)	(None, 32, 32, 64)	73792
re_lu_7 (ReLU)	(None, 32, 32, 64)	0
up_sampling2d_2 (UpSampling2	(None, 64, 64, 64)	0
conv2d_8 (Conv2D)	(None, 64, 64, 32)	18464
re_lu_8 (ReLU)	(None, 64, 64, 32)	0
up_sampling2d_3 (UpSampling2	(None, 128, 128, 32)	0
conv2d_9 (Conv2D)	(None, 128, 128, 16)	4624
re_lu_9 (ReLU)	(None, 128, 128, 16)	0
up_sampling2d_4 (UpSampling2	(None, 256, 256, 16)	0
conv2d_10 (Conv2D)	(None, 256, 256, 8)	1160
re_lu_10 (ReLU)	(None, 256, 256, 8)	0
conv2d_11 (Conv2D)	(None, 256, 256, 1)	9

Total params: 786,641
Trainable params: 786,641
Non-trainable params: 0

↓ from here we're starting the DECODING

← we've upsampling till we reach the original input space (then we combine the convolutions to obtain only 1 channel as output)

→ for each pixel we'll have just 1 value and that value is the output of a sigmoid function, that means that for each pixel we have a probability (from 0 to 1) that that pixel is a background or foreground pixel

Prepare the model for training

In [14]:
```
# Optimization params
# -------------------
# Loss
# Binary Crossentropy
loss = tf.keras.losses.BinaryCrossentropy()
# Learning rate
lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)

# Validation metrics
# ------------------
score_th = 0.5

def IoU(y_true, y_pred):
    y_pred = tf.cast(y_pred > score_th, tf.float32)

    intersection = tf.reduce_sum(y_true * y_pred)
    union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) - intersection

    return intersection / union

metrics = ['accuracy', IoU]

# Compile Model
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

(the accuracy does not work very well with segmentation task) ⇒ INTERSECTION OF A UNION

we consider the prob > score-th to be 1, otherwise 0

= percentage of the prediction overlapping the target (if the network correctly predict the class for the pixels (so, if we have highly performance) we expect the prediction to perfectly match the target, otherwise the overlapping will be small. We're **NOT** considering the background!)

that's how we define this metrics (IoU)

Training with callbacks

In [15]:
```
import os
from datetime import datetime

cwd = os.getcwd()

exps_dir = os.path.join(cwd, 'drive/My Drive/Keras4/', 'binary_segmentation_experiments')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

model_name = 'CNN'

exp_dir = os.path.join(exps_dir, model_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)
```

```
callbacks = []

# Model checkpoint
# ----------------------------------------------------------------------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp_{epoch:02d}.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# ----------------------------------------------------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=0)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# ----------------------------------------------------------------------------
early_stop = False
if early_stop:
    es_callback = tf.keras.callback.EarlyStopping(monitor='val_loss', patience=10)
    callbacks.append(es_callback)


model.fit(x=train_dataset,
          epochs=100,  #### set repeat in training dataset
          steps_per_epoch=len(train_img_gen),
          validation_data=valid_dataset,
          validation_steps=len(valid_img_gen),
          callbacks=callbacks)

# How to visualize Tensorboard
# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

```
Epoch 1/100
32/32 [==============================] - 18s 575ms/step - loss: 0.6493 - accuracy: 0.6722 - IoU: 0.0132 - val_loss: 0.62
91 - val_accuracy: 0.6962 - val_IoU: 0.0000e+00
Epoch 2/100
32/32 [==============================] - 17s 538ms/step - loss: 0.6462 - accuracy: 0.6749 - IoU: 0.0000e+00 - val_loss:
0.6157 - val_accuracy: 0.6962 - val_IoU: 0.0000e+00
Epoch 3/100
32/32 [==============================] - 17s 537ms/step - loss: 0.6296 - accuracy: 0.6791 - IoU: 0.0000e+00 - val_loss:
0.6114 - val_accuracy: 0.6962 - val_IoU: 0.0000e+00
Epoch 4/100
32/32 [==============================] - 18s 553ms/step - loss: 0.6078 - accuracy: 0.6784 - IoU: 0.0000e+00 - val_loss:
0.6163 - val_accuracy: 0.6962 - val_IoU: 0.0000e+00
Epoch 5/100
32/32 [==============================] - 18s 564ms/step - loss: 0.6189 - accuracy: 0.6768 - IoU: 0.0000e+00 - val_loss:
0.5837 - val_accuracy: 0.6962 - val_IoU: 0.0000e+00
Epoch 6/100
32/32 [==============================] - 18s 557ms/step - loss: 0.5919 - accuracy: 0.6768 - IoU: 0.0000e+00 - val_loss:
0.5876 - val_accuracy: 0.7057 - val_IoU: 0.0567
Epoch 7/100
32/32 [==============================] - 18s 551ms/step - loss: 0.5845 - accuracy: 0.7003 - IoU: 0.1890 - val_loss: 0.57
33 - val_accuracy: 0.7077 - val_IoU: 0.3031
Epoch 8/100
32/32 [==============================] - 18s 553ms/step - loss: 0.5691 - accuracy: 0.7117 - IoU: 0.2092 - val_loss: 0.55
39 - val_accuracy: 0.7250 - val_IoU: 0.3469
Epoch 9/100
32/32 [==============================] - 18s 550ms/step - loss: 0.5495 - accuracy: 0.7267 - IoU: 0.2880 - val_loss: 0.58
81 - val_accuracy: 0.6780 - val_IoU: 0.3805
Epoch 10/100
32/32 [==============================] - 18s 551ms/step - loss: 0.5453 - accuracy: 0.7338 - IoU: 0.3065 - val_loss: 0.53
02 - val_accuracy: 0.7482 - val_IoU: 0.3292
Epoch 11/100
32/32 [==============================] - 18s 556ms/step - loss: 0.5269 - accuracy: 0.7446 - IoU: 0.3412 - val_loss: 0.54
95 - val_accuracy: 0.7232 - val_IoU: 0.3932
Epoch 12/100
32/32 [==============================] - 18s 553ms/step - loss: 0.5383 - accuracy: 0.7288 - IoU: 0.2484 - val_loss: 0.53
22 - val_accuracy: 0.7179 - val_IoU: 0.4000
Epoch 13/100
32/32 [==============================] - 18s 550ms/step - loss: 0.5091 - accuracy: 0.7575 - IoU: 0.3833 - val_loss: 0.50
25 - val_accuracy: 0.7549 - val_IoU: 0.3032
Epoch 14/100
32/32 [==============================] - 18s 553ms/step - loss: 0.5161 - accuracy: 0.7514 - IoU: 0.3623 - val_loss: 0.51
64 - val_accuracy: 0.7518 - val_IoU: 0.4022
Epoch 15/100
32/32 [==============================] - 18s 551ms/step - loss: 0.5026 - accuracy: 0.7586 - IoU: 0.3967 - val_loss: 0.54
36 - val_accuracy: 0.7136 - val_IoU: 0.4075
Epoch 16/100
32/32 [==============================] - 18s 554ms/step - loss: 0.4965 - accuracy: 0.7625 - IoU: 0.3771 - val_loss: 0.50
90 - val_accuracy: 0.7626 - val_IoU: 0.3087
Epoch 17/100
32/32 [==============================] - 18s 557ms/step - loss: 0.4912 - accuracy: 0.7658 - IoU: 0.4015 - val_loss: 0.49
27 - val_accuracy: 0.7636 - val_IoU: 0.3979
Epoch 18/100
32/32 [==============================] - 18s 554ms/step - loss: 0.4831 - accuracy: 0.7758 - IoU: 0.4150 - val_loss: 0.48
26 - val_accuracy: 0.7680 - val_IoU: 0.4119
Epoch 19/100
32/32 [==============================] - 18s 552ms/step - loss: 0.4703 - accuracy: 0.7840 - IoU: 0.4494 - val_loss: 0.56
57 - val_accuracy: 0.7247 - val_IoU: 0.4539
Epoch 20/100
32/32 [==============================] - 18s 550ms/step - loss: 0.4731 - accuracy: 0.7796 - IoU: 0.4329 - val_loss: 0.47
23 - val_accuracy: 0.7766 - val_IoU: 0.4145
Epoch 21/100
32/32 [==============================] - 18s 551ms/step - loss: 0.4658 - accuracy: 0.7810 - IoU: 0.4424 - val_loss: 0.49
```

```
63 - val_accuracy: 0.7629 - val_IoU: 0.4355
Epoch 22/100
32/32 [==============================] - 18s 557ms/step - loss: 0.4640 - accuracy: 0.7838 - IoU: 0.4542 - val_loss: 0.49
01 - val_accuracy: 0.7578 - val_IoU: 0.4756
Epoch 23/100
32/32 [==============================] - 18s 549ms/step - loss: 0.4711 - accuracy: 0.7848 - IoU: 0.4606 - val_loss: 0.57
82 - val_accuracy: 0.7267 - val_IoU: 0.4760
Epoch 24/100
32/32 [==============================] - 18s 550ms/step - loss: 0.4540 - accuracy: 0.7920 - IoU: 0.4751 - val_loss: 0.51
97 - val_accuracy: 0.7532 - val_IoU: 0.4708
Epoch 25/100
32/32 [==============================] - 18s 552ms/step - loss: 0.4569 - accuracy: 0.7898 - IoU: 0.4671 - val_loss: 0.46
34 - val_accuracy: 0.7873 - val_IoU: 0.4022
Epoch 26/100
32/32 [==============================] - 17s 542ms/step - loss: 0.4409 - accuracy: 0.7979 - IoU: 0.4733 - val_loss: 0.44
50 - val_accuracy: 0.7991 - val_IoU: 0.4496
Epoch 27/100
32/32 [==============================] - 18s 550ms/step - loss: 0.4318 - accuracy: 0.8047 - IoU: 0.4989 - val_loss: 0.45
08 - val_accuracy: 0.7983 - val_IoU: 0.4669
Epoch 28/100
32/32 [==============================] - 18s 553ms/step - loss: 0.4316 - accuracy: 0.8036 - IoU: 0.4900 - val_loss: 0.45
66 - val_accuracy: 0.7879 - val_IoU: 0.5013
Epoch 29/100
32/32 [==============================] - 18s 553ms/step - loss: 0.4194 - accuracy: 0.8138 - IoU: 0.5125 - val_loss: 0.46
55 - val_accuracy: 0.7957 - val_IoU: 0.4531
Epoch 30/100
32/32 [==============================] - 18s 550ms/step - loss: 0.4260 - accuracy: 0.8070 - IoU: 0.5094 - val_loss: 0.48
47 - val_accuracy: 0.7662 - val_IoU: 0.5056
Epoch 31/100
32/32 [==============================] - 18s 550ms/step - loss: 0.4255 - accuracy: 0.8093 - IoU: 0.5077 - val_loss: 0.48
63 - val_accuracy: 0.7733 - val_IoU: 0.4485
Epoch 32/100
32/32 [==============================] - 18s 555ms/step - loss: 0.4178 - accuracy: 0.8120 - IoU: 0.5142 - val_loss: 0.44
66 - val_accuracy: 0.7923 - val_IoU: 0.4975
Epoch 33/100
32/32 [==============================] - 18s 552ms/step - loss: 0.4314 - accuracy: 0.8046 - IoU: 0.4935 - val_loss: 0.45
22 - val_accuracy: 0.7928 - val_IoU: 0.4816
Epoch 34/100
32/32 [==============================] - 18s 553ms/step - loss: 0.4147 - accuracy: 0.8160 - IoU: 0.5195 - val_loss: 0.43
87 - val_accuracy: 0.8030 - val_IoU: 0.5137
Epoch 35/100
32/32 [==============================] - 18s 551ms/step - loss: 0.4091 - accuracy: 0.8180 - IoU: 0.5305 - val_loss: 0.43
69 - val_accuracy: 0.7968 - val_IoU: 0.5182
Epoch 36/100
32/32 [==============================] - 18s 556ms/step - loss: 0.3950 - accuracy: 0.8231 - IoU: 0.5409 - val_loss: 0.44
69 - val_accuracy: 0.7938 - val_IoU: 0.5135
Epoch 37/100
32/32 [==============================] - 18s 558ms/step - loss: 0.3963 - accuracy: 0.8231 - IoU: 0.5354 - val_loss: 0.44
91 - val_accuracy: 0.7975 - val_IoU: 0.5047
Epoch 38/100
32/32 [==============================] - 17s 546ms/step - loss: 0.3979 - accuracy: 0.8222 - IoU: 0.5447 - val_loss: 0.42
38 - val_accuracy: 0.8086 - val_IoU: 0.4832
Epoch 39/100
32/32 [==============================] - 18s 550ms/step - loss: 0.3951 - accuracy: 0.8265 - IoU: 0.5442 - val_loss: 0.44
20 - val_accuracy: 0.8017 - val_IoU: 0.5279
Epoch 40/100
32/32 [==============================] - 17s 546ms/step - loss: 0.3820 - accuracy: 0.8329 - IoU: 0.5574 - val_loss: 0.41
09 - val_accuracy: 0.8171 - val_IoU: 0.5119
Epoch 41/100
32/32 [==============================] - 17s 545ms/step - loss: 0.4093 - accuracy: 0.8161 - IoU: 0.5255 - val_loss: 0.44
09 - val_accuracy: 0.7998 - val_IoU: 0.5130
Epoch 42/100
32/32 [==============================] - 17s 543ms/step - loss: 0.3932 - accuracy: 0.8240 - IoU: 0.5311 - val_loss: 0.41
83 - val_accuracy: 0.8101 - val_IoU: 0.5083
Epoch 43/100
32/32 [==============================] - 17s 537ms/step - loss: 0.3803 - accuracy: 0.8316 - IoU: 0.5633 - val_loss: 0.42
55 - val_accuracy: 0.8189 - val_IoU: 0.5117
Epoch 44/100
29/32 [==========================>...] - ETA: 1s - loss: 0.3721 - accuracy: 0.8362 - IoU: 0.5690
```

low!

Test model

Compute prediction

In [16]:
```
iterator = iter(valid_dataset)
```

In [17]:
```
fig, ax = plt.subplots(1, 3, figsize=(8, 8))
fig.show()
image, target = next(iterator)

image = image[0]
target = target[0, ..., 0]

out_sigmoid = model.predict(x=tf.expand_dims(image, 0))

# Get predicted class as the index corresponding to the maximum value in the vector probability
predicted_class = tf.cast(out_sigmoid > score_th, tf.int32)
predicted_class = predicted_class[0, ..., 0]

print(predicted_class.shape)

# Assign colors (just for visualization)
target_img = np.zeros([target.shape[0], target.shape[1], 3])
prediction_img = np.zeros([target.shape[0], target.shape[1], 3])

target_img[np.where(target == 0)] = colors_dict[0]
target_img[np.where(target == 1)] = colors_dict[1]

prediction_img[np.where(predicted_class == 0)] = colors_dict[0]
prediction_img[np.where(predicted_class == 1)] = colors_dict[1]

ax[0].imshow(np.uint8(image*255.))
ax[1].imshow(np.uint8(target_img))
```

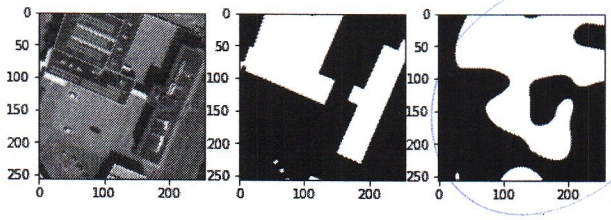
```
ax[2].imshow(np.uint8(prediction_img))

fig.canvas.draw()
time.sleep(1)
```

```
(256, 256)
```

Out[17]: <matplotlib.image.AxesImage at 0x7fd60b97a9e8>

Out[17]: <matplotlib.image.AxesImage at 0x7fd60b92a2b0>

Out[17]: <matplotlib.image.AxesImage at 0x7fd60b92a390>



result obtained on a test image (notice that we stopped the training when the model was not yet so good)

3/3

Topics : Multi-class Segmentation

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import os

# os.environ["CUDA_VISIBLE_DEVICES"]="-1"
import tensorflow as tf
import numpy as np

# Set the seed for random operations.
# This let our experiments to be reproducible.
SEED = 1234
tf.random.set_seed(SEED)
```

In [3]:
```
cwd = os.getcwd()
```

In [4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [5]:
```
!unzip /content/drive/My\ Drive/VOCDataset.zip
```

```
Archive:  /content/drive/My Drive/VOCDataset.zip
   creating: VOCDataset/
   creating: VOCDataset/Images/
  inflating: VOCDataset/Images/000032.jpg
  inflating: ..
```

In [6]:
```
!ls /content/VOCDataset/Images
```

```
000032.jpg  001724.jpg  003373.jpg  005144.jpg  006699.jpg  008403.jpg
000033.jpg  001733.jpg  ..
```

Example: Image Segmentation

Build segmentation

In [7]:
```
# ImageDataGenerator
# ------------------------------------------------------------------------------

from tensorflow.keras.preprocessing.image import ImageDataGenerator
apply_data_augmentation = True

# Create training ImageDataGenerator object
# We need two different generators for images and corresponding masks
if apply_data_augmentation:
    img_data_gen = ImageDataGenerator(rotation_range=10,
                                      width_shift_range=10,
                                      height_shift_range=10,
                                      zoom_range=0.3,
                                      horizontal_flip=True,
                                      vertical_flip=True,
                                      fill_mode='reflect')
    mask_data_gen = ImageDataGenerator(rotation_range=10,
                                       width_shift_range=10,
                                       height_shift_range=10,
                                       zoom_range=0.3,
                                       horizontal_flip=True,
                                       vertical_flip=True,
                                       fill_mode='reflect')
```

In [8]:
```
from PIL import Image
class CustomDataset(tf.keras.utils.Sequence):

  """
    CustomDataset inheriting from tf.keras.utils.Sequence.

    3 main methods:
      - __init__: save dataset params like directory, filenames..
      - __len__: return the total number of samples in the dataset
      - __getitem__: return a sample from the dataset

    Note:
      - the custom dataset return a single sample from the dataset. Then, we use
        a tf.data.Dataset object to group samples into batches.
      - in this case we have a different structure of the dataset in memory.
        We have all the images in the same folder and the training and validation splits
        are defined in text files.

  """

  def __init__(self, dataset_dir, which_subset, img_generator=None, mask_generator=None,
               preprocessing_function=None, out_shape=[256, 256]):
    if which_subset == 'training':
      subset_file = os.path.join(dataset_dir, 'Splits', 'train.txt')
    elif which_subset == 'validation':
      subset_file = os.path.join(dataset_dir, 'Splits', 'val.txt')

    with open(subset_file, 'r') as f:
      lines = f.readlines()

    subset_filenames = []
    for line in lines:
      subset_filenames.append(line.strip())

    self.which_subset = which_subset
```

```
    self.dataset_dir = dataset_dir
    self.subset_filenames = subset_filenames
    self.img_generator = img_generator
    self.mask_generator = mask_generator
    self.preprocessing_function = preprocessing_function
    self.out_shape = out_shape

  def __len__(self):
    return len(self.subset_filenames)

  def __getitem__(self, index):
    # Read Image
    curr_filename = self.subset_filenames[index]
    img = Image.open(os.path.join(self.dataset_dir, 'Images', curr_filename + '.jpg'))
    mask = Image.open(os.path.join(self.dataset_dir, 'Annotations', curr_filename + '.png'))

    # Resize image and mask
    img = img.resize(self.out_shape)
    mask = mask.resize(self.out_shape, resample=Image.NEAREST)

    img_arr = np.array(img)
    mask_arr = np.array(mask)

    # in this dataset 255 mask label is assigned to an additional class, which corresponds
    # to the contours of the objects. We remove it for simplicity.
    mask_arr[mask_arr == 255] = 0

    mask_arr = np.expand_dims(mask_arr, -1)

    if self.which_subset == 'training':
      if self.img_generator is not None and self.mask_generator is not None:
        # Perform data augmentation
        # We can get a random transformation from the ImageDataGenerator using get_random_transform
        # and we can apply it to the image using apply_transform
        img_t = self.img_generator.get_random_transform(img_arr.shape, seed=SEED)
        mask_t = self.mask_generator.get_random_transform(mask_arr.shape, seed=SEED)
        img_arr = self.img_generator.apply_transform(img_arr, img_t)
        # ImageDataGenerator use bilinear interpolation for augmenting the images.
        # Thus, when applied to the masks it will output 'interpolated classes', which
        # is an unwanted behaviour. As a trick, we can transform each class mask
        # separately and then we can cast to integer values (as in the binary segmentation notebook).
        # Finally, we merge the augmented binary masks to obtain the final segmentation mask.
        out_mask = np.zeros_like(mask_arr)
        for c in np.unique(mask_arr):
          if c > 0:
            curr_class_arr = np.float32(mask_arr == c)
            curr_class_arr = self.mask_generator.apply_transform(curr_class_arr, mask_t)
            # from [0, 1] to {0, 1}
            curr_class_arr = np.uint8(curr_class_arr)
            # recover original class
            curr_class_arr = curr_class_arr * c
            out_mask += curr_class_arr
    else:
      out_mask = mask_arr

    if self.preprocessing_function is not None:
        img_arr = self.preprocessing_function(img_arr)

    return img_arr, np.float32(out_mask)
```

In [9]:
```
from tensorflow.keras.applications.vgg16 import preprocess_input

img_h = 256
img_w = 256

dataset = CustomDataset('/content/VOCDataset', 'training',
                        img_generator=img_data_gen, mask_generator=mask_data_gen,
                        preprocessing_function=preprocess_input)
dataset_valid = CustomDataset('/content/VOCDataset', 'validation',
                              preprocessing_function=preprocess_input)
```

In [10]:
```
train_dataset = tf.data.Dataset.from_generator(lambda: dataset,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([img_h, img_w, 3], [img_h, img_w, 1]))

train_dataset = train_dataset.batch(32)
train_dataset = train_dataset.repeat()

valid_dataset = tf.data.Dataset.from_generator(lambda: dataset_valid,
                                               output_types=(tf.float32, tf.float32),
                                               output_shapes=([img_h, img_w, 3], [img_h, img_w, 1]))
valid_dataset = valid_dataset.batch(32)
valid_dataset = valid_dataset.repeat()
```

In [11]:
```
!ls /content/VOCDataset
```

```
Annotations  Images  Splits
```

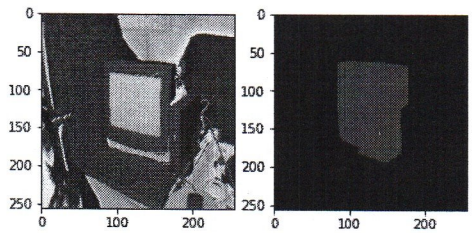
In [12]:
```
# Let's test data generator
# ----------------------------------------------------------------------------
import time
from matplotlib import cm
import matplotlib.pyplot as plt

%matplotlib inline

# Assign a color to each class
evenly_spaced_interval = np.linspace(0, 1, 20)
colors = [cm.rainbow(x) for x in evenly_spaced_interval]

iterator = iter(valid_dataset)
```

In [13]:
```
fig, ax = plt.subplots(1, 2)

augmented_img, target = next(iterator)
augmented_img = augmented_img[0]   # First element
augmented_img = augmented_img  # denormalize

target = np.array(target[0, ..., 0])   # First element (squeezing channel dimension)

print(np.unique(target))

target_img = np.zeros([target.shape[0], target.shape[1], 3])

target_img[np.where(target == 0)] = [0, 0, 0]
for i in range(1, 21):
  target_img[np.where(target == i)] = np.array(colors[i-1])[:3] * 255

ax[0].imshow(np.uint8(augmented_img))
ax[1].imshow(np.uint8(target_img))

plt.show()
```

```
[ 0. 20.]
```

Out[13]: <matplotlib.image.AxesImage at 0x7f8e102dc828>

Out[13]: <matplotlib.image.AxesImage at 0x7f8e103795c0>



In [14]:
```
vgg = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=(img_h, img_w, 3))
vgg.summary()
for layer in vgg.layers:
  layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 4s 0us/step
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 256, 256, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 256, 256, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 128, 128, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 128, 128, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 128, 128, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 64, 64, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 64, 64, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 64, 64, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 32, 32, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 32, 32, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 32, 32, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 16, 16, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 8, 8, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Convolutional Neural Network (CNN)

Encoder-Decoder

In [15]:
```
def create_model(depth, start_f, num_classes):

    model = tf.keras.Sequential()

    # Encoder
```

```
    # ----------------------------------------------------------------------------
    model.add(vgg)

    start_f = 256

    # Decoder
    # ----------------------------------------------------------------------------
    for i in range(depth):
        model.add(tf.keras.layers.UpSampling2D(2, interpolation='bilinear'))
        model.add(tf.keras.layers.Conv2D(filters=start_f,
                                         kernel_size=(3, 3),
                                         strides=(1, 1),
                                         padding='same'))
        model.add(tf.keras.layers.ReLU())

        start_f = start_f // 2

    # Prediction Layer
    # ----------------------------------------------------------------------------
    model.add(tf.keras.layers.Conv2D(filters=num_classes,
                                     kernel_size=(1, 1),
                                     strides=(1, 1),
                                     padding='same',
                                     activation='softmax'))

    return model
```

we take the encoder of the VGG and we have to add only the decoder part

In [16]:
```
model = create_model(depth=5,
                     start_f=8,
                     num_classes=21)

# Visualize created model as a table
model.summary()

# Visualize initialized weights
# model.weights
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 8, 8, 512)         14714688
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 16, 16, 512)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 16, 16, 256)       1179904
_________________________________________________________________
re_lu (ReLU)                 (None, 16, 16, 256)       0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 128)       295040
_________________________________________________________________
re_lu_1 (ReLU)               (None, 32, 32, 128)       0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        73792
_________________________________________________________________
re_lu_2 (ReLU)               (None, 64, 64, 64)        0
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 32)      18464
_________________________________________________________________
re_lu_3 (ReLU)               (None, 128, 128, 32)      0
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 256, 256, 16)      4624
_________________________________________________________________
re_lu_4 (ReLU)               (None, 256, 256, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 256, 256, 21)      357
=================================================================
Total params: 16,286,869
Trainable params: 1,572,181
Non-trainable params: 14,714,688
_________________________________________________________________
```

Prepare the model for training

In [17]:
```
# Optimization params
# ----------------------------------------------------------------------------

# Loss
# Sparse Categorical Crossentropy to use integers (mask) instead of one-hot encoded labels
loss = tf.keras.losses.SparseCategoricalCrossentropy()
# Learning rate
lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)

# Here we define the intersection over union for each class in the batch.
# Then we compute the final iou as the mean over classes
def meanIoU(y_true, y_pred):
    # get predicted class from softmax
    y_pred = tf.expand_dims(tf.argmax(y_pred, -1), -1)

    per_class_iou = []

    for i in range(1,21): # exclude the background class 0
      # Get prediction and target related to only a single class (i)
      class_pred = tf.cast(tf.where(y_pred == i, 1, 0), tf.float32)
```

```
        class_true = tf.cast(tf.where(y_true == i, 1, 0), tf.float32)
        intersection = tf.reduce_sum(class_true * class_pred)
        union = tf.reduce_sum(class_true) + tf.reduce_sum(class_pred) - intersection

        iou = (intersection + 1e-7) / (union + 1e-7)
        per_class_iou.append(iou)

    return tf.reduce_mean(per_class_iou)

# Validation metrics
# ------------------------------------------------------------------------------
metrics = ['accuracy', meanIoU]

# Compile Model
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

Training with callbacks

In [18]:
```
import os
from datetime import datetime

cwd = os.getcwd()

exps_dir = os.path.join(cwd, 'drive/My Drive/Keras4/', 'multiclass_segmentation_experiments')
if not os.path.exists(exps_dir):
    os.makedirs(exps_dir)

now = datetime.now().strftime('%b%d_%H-%M-%S')

model_name = 'CNN'

exp_dir = os.path.join(exps_dir, model_name + '_' + str(now))
if not os.path.exists(exp_dir):
    os.makedirs(exp_dir)

callbacks = []

# Model checkpoint
# ------------------------------------------------------------------------------
ckpt_dir = os.path.join(exp_dir, 'ckpts')
if not os.path.exists(ckpt_dir):
    os.makedirs(ckpt_dir)

ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(ckpt_dir, 'cp_{epoch:02d}.ckpt'),
                                                   save_weights_only=True)  # False to save the model directly
callbacks.append(ckpt_callback)

# Visualize Learning on Tensorboard
# ------------------------------------------------------------------------------
tb_dir = os.path.join(exp_dir, 'tb_logs')
if not os.path.exists(tb_dir):
    os.makedirs(tb_dir)

# By default shows losses and metrics for both training and validation
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=tb_dir,
                                             profile_batch=0,
                                             histogram_freq=0)  # if 1 shows weights histograms
callbacks.append(tb_callback)

# Early Stopping
# ------------------------------------------------------------------------------
early_stop = False
if early_stop:
    es_callback = tf.keras.callback.EarlyStopping(monitor='val_loss', patience=10)
    callbacks.append(es_callback)


model.fit(x=train_dataset,
          epochs=100,  #### set repeat in training dataset
          steps_per_epoch=len(dataset),
          validation_data=valid_dataset,
          validation_steps=len(dataset_valid),
          callbacks=callbacks)

# How to visualize Tensorboard
# 1. tensorboard --logdir EXPERIMENTS_DIR --port PORT     <- from terminal
# 2. localhost:PORT   <- in your browser
```

```
Epoch 1/100
209/209 [==============================] - 497s 2s/step - loss: 0.8915 - accuracy: 0.7971 - meanIoU: 0.1146 - val_loss:
1.7129 - val_accuracy: 0.7826 - val_meanIoU: 0.1457
Epoch 2/100
  5/209 [..............................] - ETA: 4:47 - loss: 0.4179 - accuracy: 0.8586 - meanIoU: 0.2225
```

Test model

Compute prediction

In [ ]:
```
model.load_weights('/content/drive/My Drive/Keras4/multiclass_segmentation_experiments/CNN_Nov27_08-41-36/ckpts/cp_02.ck
```

In [20]:
```
import time
import matplotlib.pyplot as plt

from PIL import Image

%matplotlib inline

iterator = iter(valid_dataset)
```

In [21]:
```
fig, ax = plt.subplots(1, 3, figsize=(8, 8))
fig.show()
image, target = next(iterator)

image = image[0]
target = target[0, ..., 0]

out_sigmoid = model.predict(x=tf.expand_dims(image, 0))

# Get predicted class as the index corresponding to the maximum value in the vector probability
# predicted_class = tf.cast(out_sigmoid > score_th, tf.int32)
# predicted_class = predicted_class[0, ..., 0]
predicted_class = tf.argmax(out_sigmoid, -1)

out_sigmoid.shape

predicted_class = predicted_class[0, ...]

# Assign colors (just for visualization)
target_img = np.zeros([target.shape[0], target.shape[1], 3])
prediction_img = np.zeros([target.shape[0], target.shape[1], 3])

target_img[np.where(target == 0)] = [0, 0, 0]
for i in range(1, 21):
  target_img[np.where(target == i)] = np.array(colors[i-1])[:3] * 255

prediction_img[np.where(predicted_class == 0)] = [0, 0, 0]
for i in range(1, 21):
  prediction_img[np.where(predicted_class == i)] = np.array(colors[i-1])[:3] * 255

ax[0].imshow(np.uint8(image))
ax[1].imshow(np.uint8(target_img))
ax[2].imshow(np.uint8(prediction_img))

fig.canvas.draw()
time.sleep(1)
```

Out[21]: (1, 256, 256, 21)

Out[21]: <matplotlib.image.AxesImage at 0x7f8db8b8e358>

Out[21]: <matplotlib.image.AxesImage at 0x7f8db8aa5eb8>

Out[21]: <matplotlib.image.AxesImage at 0x7f8db8ac1cf8>