9 - a 이미지 분할

```
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz
!tar -xf images.tar.gz
!tar -xf annotations.tar.gz
```

“!” 문자는 뒤에 나오는 텍스트가 쉘 명령임을 나타낸다.
!tar -xf 압축 파일: 현재 작업 디렉토리에 압축 파일을 해제한다.

```
import os
# 총 7390개 이미지
# 입력 이미지가 있는 디렉토리 경로
input_dir = "images/"
# 타겟 알파마스크 파일이 있는 디렉토리 경로
target_dir = "annotations/trimaps/"

# 입력 이미지 파일 경로 리스트 생성
input_img_paths = sorted(
    [os.path.join(input_dir, fname)  # 입력 이미지 파일 경로 생성
     for fname in os.listdir(input_dir)  # 입력 이미지가 있는 디렉토리 내 파일명 리스트를 가져옴
     if fname.endswith(".jpg")])  # 파일 확장자가 ".jpg"인 파일만 리스트에 포함시킴

# 타겟 알파마스크 파일 경로 리스트 생성
target_paths = sorted(
    [os.path.join(target_dir, fname)  # 알파마스크 파일 경로 생성
     for fname in os.listdir(target_dir)  # 알파마스크 파일이 있는 디렉토리 내 파일명 리스트를 가져옴
     if fname.endswith(".png") and not fname.startswith(".")])  # 파일 확장자가 ".png"이면서 파일명이 "."으로
시작하지 않는 파일만 리스트에 포함시킴
```

Sorted() : 압축을 풀 때 순서가 바뀌는 경우을 대비하여, 파일의 정렬된 순서를 보장한다.

if fname.endswith(”.png”)and not fname.startwith(”.”) - .png로 끝나거나, 온점으로 시작하지 않는 파일만 리스트에 포함시킨다. → . 으로 시작하는 이상한 이미지는 제외

```
import matplotlib.pyplot as plt
from tensorflow.keras.utils import load_img, img_to_array

plt.axis("off")
plt.imshow(load_img(input_img_paths[9])) # 인덱스 9에 해당하는 이미지를 출력한다.
```

img_to_array - 이미지를 numpy 배열로 바꿔준다.

```
def display_target(target_array):
    normalized_array = (target_array.astype("uint8") - 1) * 127 # 1.
    plt.axis("off")
    plt.imshow(normalized_array[:, :, 0]) # 2

img = img_to_array(load_img(target_paths[9], color_mode="grayscale")) # 3.
display_target(img)
```

1. 0,1,2 -> 0,127,254로 표현하기 위함.
2. 원래 (448,500,3) 의 RGB 컬러를 지닌 이미지인데 이중에서 GRAYSCALE(448,500,1)만 보기 위함 → 이미지를 효과적으로 표시한다.

3. color_mode=”grayscale”로 지정하여 로드한 이미지를 하나의 컬러 채널이 있는것처럼 다룬다.

GRAYSCALE인데 회색이 아니다
이유는 잘 보이게 viridis가 디폴트로 적용중이기 때문
imshow의 기본 색상 매핑이 viridis이고, 낮은 건 파랑, 높은 건 노랑으로 매핑 한다.

```
import numpy as np
import random

img_size = (200, 200) #입력과 타깃을 모두 200x200 크기로 변경
num_imgs = len(input_img_paths) #데이터에 있는 전체 샘플 개수 7

# 1.
random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_paths)

#입력 이미지의 경로를 입력받아 해당 이미지를 Numpy 배열 변환
def path_to_input_image(path):
    return img_to_array(load_img(path, target_size=img_size)) # 사이즈는 200x200
    #load_img() 함수로 이미지를 로드 -> img_to_array() 함수로 이미지를 Numpy 배열로 변환

#타겟 이미지의 경로를 입력받아 해당 타겟 이미지를 Numpy 배열로 바꾸는 함수
def path_to_target(path):
    img = img_to_array(
        load_img(path, target_size=img_size, color_mode="grayscale")
    #load_img() 함수로 이미지를 로드 -> img_to_array() 함수로 이미지를 Numpy 배열로 변환
    )
    img = img.astype("uint8") - 1 # 2.
    return img

# 3.
input_imgs = np.zeros((num_imgs, ) + img_size + (3, ), dtype="float32")
targets = np.zeros((num_imgs, ) + img_size + (1, ), dtype="uint8")
#입력은 3개의 채널(RGB값)을 가지고 타깃은 (정수 레이블을 담은) 하나의 채널을 가짐

#전체 이미지, 타깃 마스크 로드해와서 넘파이 배열로 바꾸기
for i in range(num_imgs): # 7390
    input_imgs[i] = path_to_input_image(input_img_paths[i])
    targets[i] = path_to_target(target_paths[i])


num_val_samples = 1000 # 검증에 1000개의 샘플 사용
train_input_imgs = input_imgs[:-num_val_samples]
train_targets = targets[:-num_val_samples]
val_input_imgs = input_imgs[-num_val_samples:]
val_targets = targets[-num_val_samples:]
#맨 끝 1000개의 데이터가 검증 세트로 분리됨
```

1. 입력 경로와 타깃 경로가 동일한 순서를 유지하도록 두 명령의 시드값(1337)을 같게 한다.
2. 정수 레이블을 0부터 시작해 0,1,2가 되도록 1을 뺀다.
3. 입력 이미지는 3개의 채널(RGB)를 가지고, 타깃은 (정수 레이블을 지닌)하나의 채널을 가진다.

학습에 사용할 모델 정의

```
from tensorflow import keras
from tensorflow.keras import layers

def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))
    x = layers.Rescaling(1./255)(inputs) # 1.

    x = layers.Conv2D(64, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(128, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2D(128, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(256, 3, strides=2, padding="same", activation="relu")(x)
    x = layers.Conv2D(256, 3, activation="relu", padding="same")(x)

    x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same", strides=2)(x)
    x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same", strides=2)(x)
    x = layers.Conv2DTranspose(64, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(64, 3, activation="relu", padding="same", strides=2)(x)

    outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding="same")(x)

    model = keras.Model(inputs, outputs)
    return model

model = get_model(img_size=img_size, num_classes=3)
model.summary()
```

이 모델에서는 Input Image의 Segmentation Map을 학습한

1. 입력 이미지를 [0~1]의 범위로 만든다.

Conv2D로 다운샘플링 한다. strides =2 옵션으로 3번의 다운 샘플링을 거친 (25,25,256) 크기의 특성맵이 생성됨. 이렇게 출력된 특성 맵을 다시 Conv2DTansepose로 업샘플링하여 (200,200,3) 크기로 맞춰준다.

위치 정보를 유지하면서 특성 맵을 다운 샘플링 하기 위해 Conv2D를 사용한다. (MaxPooling2D는 위치 정보가 사라짐)

model.summary() 실행 결과

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 200, 200, 3)]	0
rescaling (Rescaling)	(None, 200, 200, 3)	0
conv2d (Conv2D)	(None, 100, 100, 64)	1792
conv2d_1 (Conv2D)	(None, 100, 100, 64)	36928
conv2d_2 (Conv2D)	(None, 50, 50, 128)	73856
conv2d_3 (Conv2D)	(None, 50, 50, 128)	147584
conv2d_4 (Conv2D)	(None, 25, 25, 256)	295168
conv2d_5 (Conv2D)	(None, 25, 25, 256)	590080
conv2d_transpose (Conv2DTran	(None, 25, 25, 256)	590080
conv2d_transpose_1 (Conv2DTr	(None, 50, 50, 256)	590080
conv2d_transpose_2 (Conv2DTr	(None, 50, 50, 128)	295040
conv2d_transpose_3 (Conv2DTr	(None, 100, 100, 128)	147584
conv2d_transpose_4 (Conv2DTr	(None, 100, 100, 64)	73792

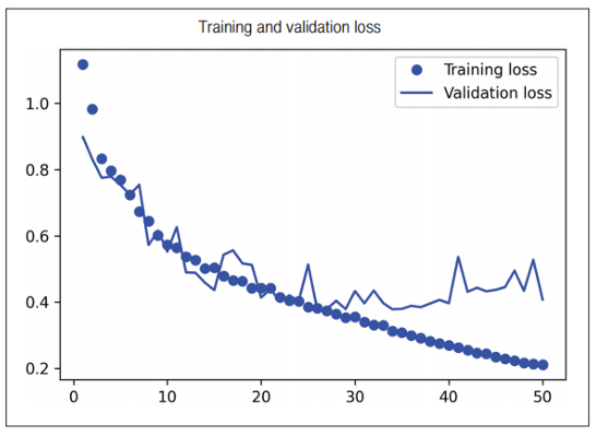
모델을 컴파일하고 훈련

```
model.compile(optimzer="rmsprop", loss="sparse_categorial_crossentropy")
callbacks = [
        keras.callbacks.ModelCheckpoint("oxford_segmentation.keras", save_best_only = True"]

histroy = model.fit(train_input_imgs, train_targets,
                                            epochs = 50,
                                            callbacks = callbacks,
                                            batch_size = 64,
                                            validation_data = (val_input_imgs, val_targets))
```

훈련과 검증 손실 그리기

```
epochs = range(1, len(history.history["loss"]) + 1)
loss = history.history["loss"]
val_loss = history.history["val_loss"]
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
```



검증 손실을 기준으로 최상의 모델을 다시 로드하여 분할 마스크를 예측한다.

```
from tensorflow.keras.utils import array_to_img

model = keras.models.load_model("oxford_segmentation.keras") # 최상의 모델불러오기

i = 4
test_images = val_input_imgs[i]
plt.axis("off")
plt.imshow(array_to_img(test_images))

# 1.
mask = model.predict(np.expand_dims(test_images, 0))[0]

def display_mask(pred):
    mask = np.argmax(pred, axis=-1) # 2.
    mask *= 127
    plt.axis("off")
    plt.imshow(mask)

display_mask(mask)
```

1. test_image 배열의 차원을 확장한다.
   넘파이는 배치를 사용하므로 이미지가 1개가 들어와도 배치가 있다고 생각해 이미지 덩어리로 가져와진다. 따라서 이미지 1개의 predict 값을 보기위해 첫번째[0] 사진을 가져온다.
2. 마지막 축을 기준으로 했을 때 pred의 최댓값 인덱스를 저장하고 127을 곱해 0,127,254로 만든다