


DNN_02_4. CNN_UNet

🕒 생성일	@2022년 6월 15일 오전 11:24
📁 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

U-Net 논문

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/89f3ff32-869d-4bc0-9ff4-9fe617b31280/UNet.pdf>

논문 리뷰:

U-Net 논문 리뷰-U-Net: Convolutional Networks for Biomedical Image Segmentation

U-Net은 Biomedical 분야에서 이미지 분할(Image Segmentation)을 목적으로 제안된 End-to-End 방식의 Fully-Convolutional Network 기반 모델이다. 네트워크 구성의 형태('U')로 인해 U-Net이라는 이름이 붙여졌다. U-Net은 이미지의 전반적인 컨텍스트 정보를 얻기 위한 네트워크와 정확한 지역화(Localization)를 위한 네트워크가 대칭 형태로 구성

 <https://medium.com/@msmapark2/u-net-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-u-net-convolutional-networks-for-biomedical-image-segmentation-456d6901b28a>





Image segmentation with a U-Net-like architecture

케라스 블로그에 게시된 U-Net 관련 프로젝트 이다.

Keras documentation: Image segmentation with a U-Net-like architecture

Keras documentation

 https://keras.io/examples/vision/oxford_pets_image_segmentation/





위 그림과 같이 특정 객체만 배경에서 분리시키는 segmentation 을 위한 딥러닝 모델을 만들려고 한다. 사용할 데이터는 oxford iiit pet dataset 이다. 코드를 따라가보자.

donwload the data

로컬 환경에서 pycharm, ananconda 같은 에디터로 작업할 계획이라면 특정 드라이브에 데이터를 저장해두어야 한다.

코랩에서 사용할 거라면 구글 드라이브 특정 폴더에 저장하면된다. 케라스 블로그에서는 input 으로 들어갈 이미지 데이터는 “images/” 를 dir path 로 설정했고,

target 데이터는 “annotations/trimaps” 으로 설정했다.

Prepare paths of input images and target segmentation masks

```
import os

input_dir = "images/"
target_dir = "annotations/trimaps/"
img_size = (160, 160)
num_classes = 3
batch_size = 32

# input image 에 대한 path를 리스트로 설정.
input_img_paths = sorted(
    [
        os.path.join(input_dir, fname) for fname in os.listdir(input_dir) if fname.endswith('.jpg')
    ]
)

# target image 에 대한 path를 리스트로 설정.
target_img_paths = sorted(
    [
        os.path.join(target_dir, fname) for fname in os.listdir(input_dir) \
        if fname.endswith('.png') and not fname.startswith(".")
    ]
)

# 결과 확인
print("Number of samples:", len(input_img_paths))
for input_path, target_path in zip(input_img_paths[:10], target_img_paths[:10]):
    print(input_path, "|", target_path)

>>> Number of samples: 7390
images/Abyssinian_1.jpg | annotations/trimaps/Abyssinian_1.png
images/Abyssinian_10.jpg | annotations/trimaps/Abyssinian_10.png
```

```
images/Abyssinian_100.jpg | annotations/trimaps/Abyssinian_100.png
images/Abyssinian_101.jpg | annotations/trimaps/Abyssinian_101.png
images/Abyssinian_102.jpg | annotations/trimaps/Abyssinian_102.png
images/Abyssinian_103.jpg | annotations/trimaps/Abyssinian_103.png
images/Abyssinian_104.jpg | annotations/trimaps/Abyssinian_104.png
images/Abyssinian_105.jpg | annotations/trimaps/Abyssinian_105.png
images/Abyssinian_106.jpg | annotations/trimaps/Abyssinian_106.png
images/Abyssinian_107.jpg | annotations/trimaps/Abyssinian_107.png
```

What does one input image and corresponding segmentation mask look like?

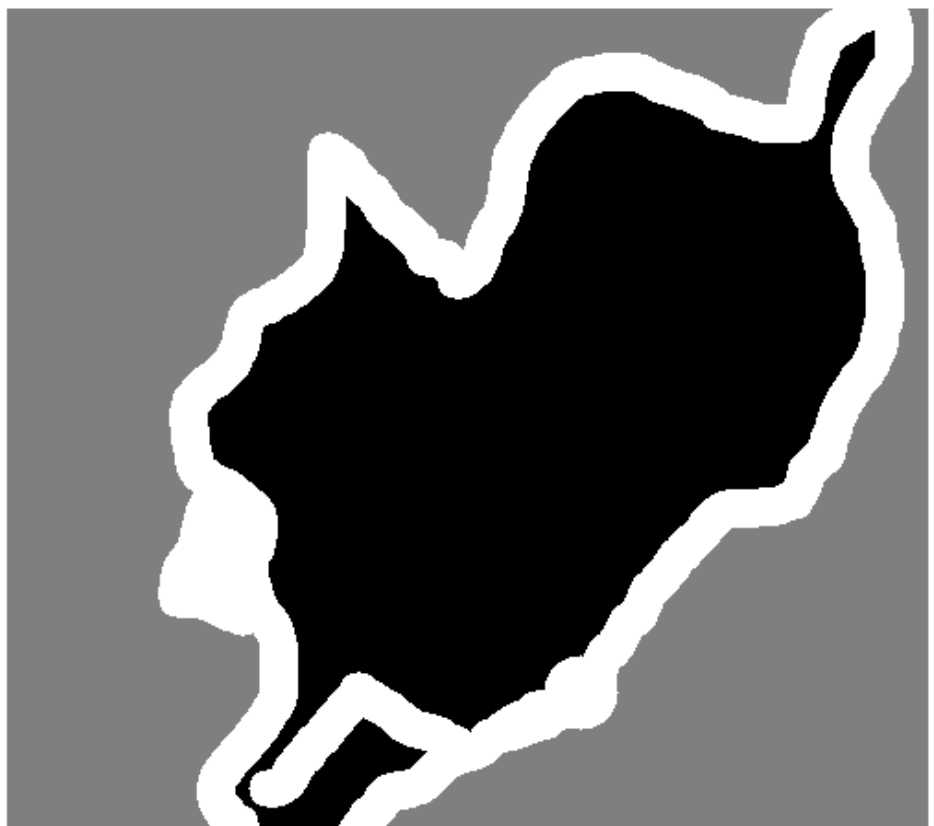
위에서 경로를 설정한 input data와 target data 는 무엇을 나타내는건지 확인하고 싶다면 다음의 코드를 실행.

```
from IPython.display import Image, display
from tensorflow.keras.preprocessing.image import load_img
import PIL
from PIL import ImageOps

# Display input image #7
display(Image(filename=input_img_paths[9]))

# Display auto-contrast version of corresponding target (per-pixel categories)
img = PIL.ImageOps.autocontrast(load_img(target_img_paths[9]))
display(img)
```

출력의 결과는 다음과 같다.



Prepare sequence class to load & vectorize batches of data

```
from tensorflow import keras
import numpy as np
from tensorflow.keras.preprocessing.image import load_img
```

```

class OxfordPets(keras.utils.Sequence):
    def __init__(self, batch_size, img_size, input_img_paths, target_img_paths):
        self.batch_size = batch_size
        self.img_size = img_size
        self.input_img_paths = input_img_paths
        self.target_img_paths = target_img_paths

    def __len__(self):
        # 전체 학습 데이터를 배치 사이즈로 나눈 몫.(소수점 이하 버림)
        return len(self.target_img_paths) // self.batch_size

# 다음의 메서드를 통해 데이터를 가져올 예정.
# dataloader 역할을 한다고 생각.
def __getitem__(self, idx):
    """
    batch #idx 에 대해 tuple (input, target) 을 반환한다.
    ex) batch 1번 -> return (input_img[1], target_img[1])
    """
    # i : 특정 idx 에 해당하는 데이터 개수
    i = idx * self.batch_size
    batch_input_img_paths = self.input_img_paths[i : i + self.batch_size]
    batch_target_img_paths = self.target_img_paths[i : i + self.batch_size]

    # list 'x' 생성
    x = np.zeros((self.batch_size,) + self.img_size + (3,)), dtype="float32")

    for j, path in enumerate(batch_input_img_paths):
        # j 는 인덱스이고 path가 정보
        img = load_img(path, target_size=self.img_size) # 여기서 target 은 그 target data 가 아니다.
        x[j] = img

    y = np.zeros((self.batch_size,) + self.img_size + (1,)), dtype="uint8")

    for j, path in enumerate(batch_target_img_paths):
        # j 는 인덱스이고 path 가 정보
        img = load_img(path, target_size=self.img_size, color_mode='grayscale')
        y[j] = np.expand_dims(img, 2)
        # 1, 2, 3 -> 0,1,2
        y[j] -= 1

    return x, y

```

Prepare U-Net Xception-style model

<https://hongl.tistory.com/45>

위 블로그에서 설명하는 Xception 은, extreme inception 의 약자로 기존 inception 모델이 채널, 공간 correlation 을 분리한 것을 depthwise separable convolution 으로 강화한 모델이다. (... 구체적으로 더 조사해봐야 이해할 것 같다. depthwise separable 이 언급된 것을 보니, 전에 나온 EfficientNet 의 특성과도 연관이 있는 것 같다.) 일단은 U-Net 의 모델을 구현하는 것에 집중한다.

다음의 코드부터는 본격적으로 U-Net 구조를 만든다.

함수형 api 로 모델을 구조화한다.

```

from tensorflow.keras import layers

def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))

    ### First half of the network: downsampling inputs ###

    # Entry block
    x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)
    x = layers.BatchNormalization(x)
    x = layers.Activation("relu")(x)

    # Set aside residual
    previous_block_activation = x

```

```

# Blocks 1, 2, 3 are identical apart from the feature depth.
for filters in [64, 128, 256]:
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    # Project residual
    residual = layers.Conv2D(filters, 1, strides=2, padding="same")(
        previous_block_activation
    )
    x = layers.add([x, residual]) # add block residual
    previous_block_activation = x # set aside next residual

#### Second half of the network: upsampling inputs ####
for filters in [256,128,64,32]:
    x = layers.Activation("relu")(x)
    x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)
    x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.UpSampling2D(2)(x)

    # Project residual
    residual = layers.UpSampling2D(2)(previous_block_activation)
    residual = layers.Conv2D(filters, 1, padding="same")(residual)
    x = layers.add([x, residual]) # add block residual
    previous_block_activation = x # set aside next residual

# Add a per-pixel classification layer
outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding="same")(x)

# 특이하게, 해당 메서드 안에 Model 클래스를 넣고, 결과로 반환을 한다.
# 모델에 대한 서브 클래싱 없이 메서드 -> 모델 output -> 결과
# Define the model
model = keras.Model(inputs, outputs)
return model

# Free up RAM in case the model definition cells were run multiple times
keras.backend.clear_session()

# Build model
model = get_model(img_size, num_classes)

```

Set aside a validation split

미리 따로 validation set 을 만들어 두지 않았기 때문에 validation set을 만드는과정 진행.

```

import random

# split our img paths into a training and a validation set
val_samples = 1000
random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_img_paths)

train_input_img_paths = input_img_paths[:-val_samples]
train_target_img_paths = target_img_paths[:-val_samples]
val_input_img_paths = input_img_paths[-val_samples:]
val_target_img_paths = target_img_paths[-val_samples:]

# Instantiate data Sequences for each split
train_gen = OxfordPets(
    batch_size, img_size, train_input_img_paths, train_target_img_paths
)
val_gen = OxfordPets(batch_size, img_size, val_input_img_paths, val_target_img_paths)

```


Train the model

```
# Configure the model for training
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')

# 체크포인트를 위한 callback 사용
check_cb = keras.callbacks.ModelCheckpoint("oxford_segmentation.h5", save_best_only=True)

epochs=15
model.fit(train_gen, epochs=epochs, validation_set=val_gen, callbacks=[check_cb])
```

Visualize predictions

```
# Generate predictions for all images in the validation set
val_gen = OxfordPets(batch_size, img_size, val_input_img_paths, val_target_img_paths)
val_preds = model.predict(val_gen)

def display_mask(i):
    """Quick utility to display a model's prediction."""
    mask = np.argmax(val_preds[i], axis=-1)
    mask = np.expand_dims(mask, axis=-1)
    img = PIL.ImageOps.autocontrast(keras.preprocessing.image.array_to_img(mask))
    display(img)

# Display results for validation image #10
i = 10

# Display input image
display(Image(filename=val_input_img_paths[i]))

# Display ground-truth target mask
img = PIL.ImageOps.autocontrast(load_img(val_target_img_paths[i]))
display(img)

# Display mask predicted by our model
display_mask(i) # Note that the model only sees inputs at 150x150.
```