CNN Tutorial

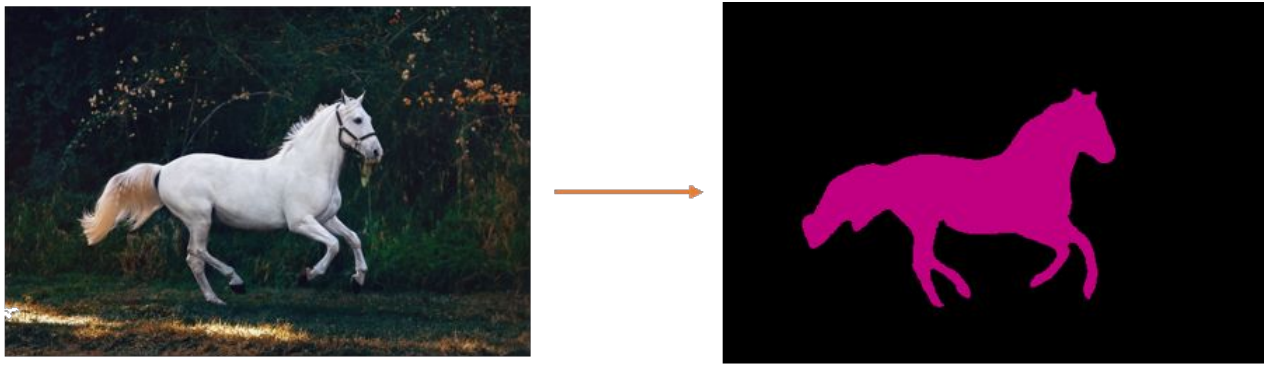
Segmentation

목차

What is segmentation?

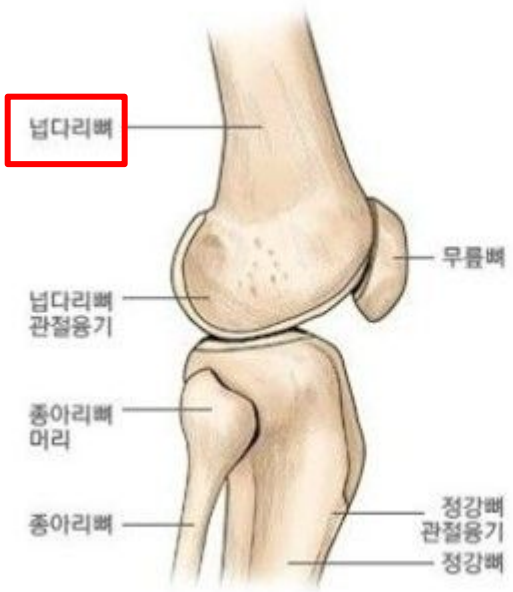
이미지의 각 픽셀이 어느 클래스에 속하는지 예측하는것

아래 이미지의 경우, 배경과 말 2개의 클래스로 예측



Intro

무릎 X-ray 이미지에서 대퇴골(femur)을 segmentation



Data

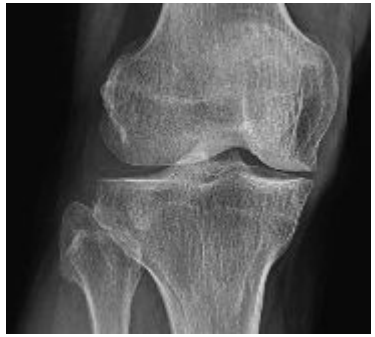
Input: 무릎 X-ray 이미지(.png)

Output: label 이미지(.png)

각각 왼쪽 무릎 315개, 오른쪽 무릎 315개, 총 630개

train: validation = 8 : 2

데이터 평균 size: 240 x 220



무릎 X-ray 이미지 (.png)

630개

label 이미지(.png)

630개

Model

U-Net

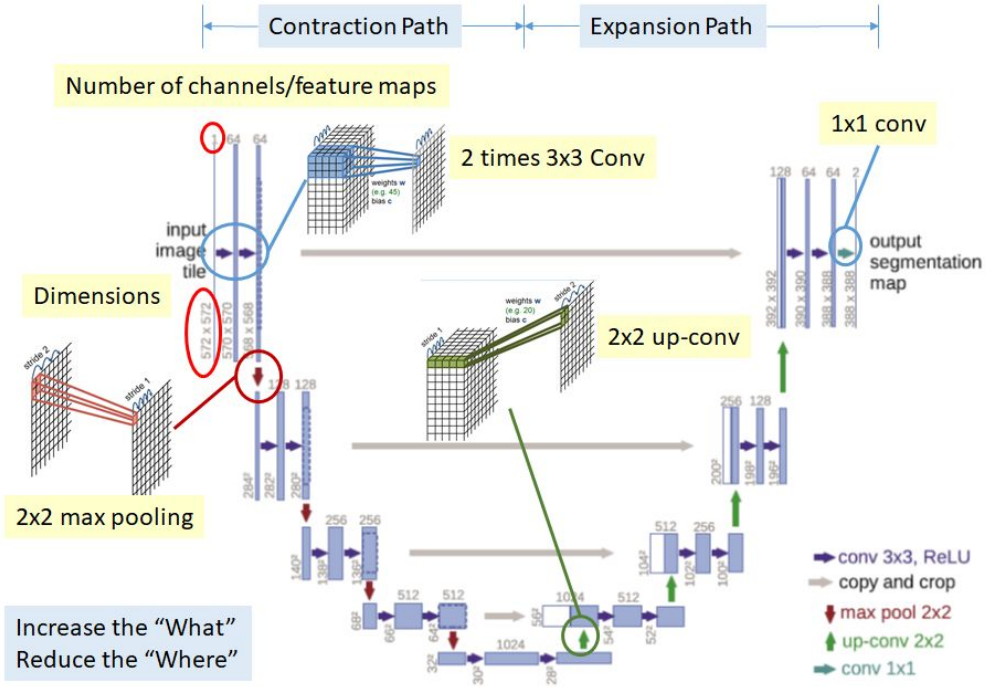
Contraction Path: 일반적인 CNN

채널의 수가 2배씩 증가

VGG base의 구조

Expansion Path: Localization 부분

contraction에서 pooling 하기 전 feature map을 결합(concat)하여 해상도를 높였다.



Model

batch_size_train = 8

batch_size_val = 2

num_epoch = 100

lr = 0.0001

momentum = 0.9

val loss: 0.075

Result

예측 결과가 예상했던 label 데이터와 너무 다르다.

Input

Label

predict

예상 이유

label 데이터가 흑백영상(gray scale image)이 아니라 이진영상(binary image)

Result

수정한 부분

Image.open(mask_path).convert('L') → Image.open(mask_path).convert('1')

이미지를 불러올때 .convert()의 옵션을 gray scale에서 binary로 변경

```
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., 0., ..., 0., 0., 0.]])
```

```
mask.max(), mask.min(), mask.mean(), mask.std()
```

```
(255, 0, 46.17821023597916, 98.19886205845971)
```

```
mask.max(), mask.min(), mask.mean(), mask.std()
```

```
(1.0, 0.0, 0.1810910205332516, 0.3850935766998419)
```

Result

예측 결과가 예상했던 label 데이터와 너무 다르다.

Input

Label

predict

Result

수정 전

흑백 영상

mean: 135.53

```
array([[142, 125, 130, ..., 162, 162, 161],
       [ 93,  71,  72, ..., 152, 155, 144],
       [ 99,  73,  72, ..., 140, 163, 146],
       ...,
       [103,  78,  79, ...,  70,  91, 123],
       [110,  99,  85, ...,  78,  94, 136],
       [139, 125, 131, ..., 113, 138, 149]], dtype=uint8)
```

수정 후

이진 영상

mean: 94.56

```
array([[ 95,  78,  73, ...,  89,  89, 100],
       [ 60,  43,  40, ...,  81,  85,  91],
       [ 60,  43,  34, ...,  78,  76,  93],
       ...,
       [ 70,  39,  44, ...,  35,  59,  67],
       [ 80,  64,  69, ...,  62,  73,  73],
       [ 89,  77,  70, ...,  62,  81,  88]], dtype=uint8)
```

Plan

데이터에 대한 전처리 부분을 더 추가

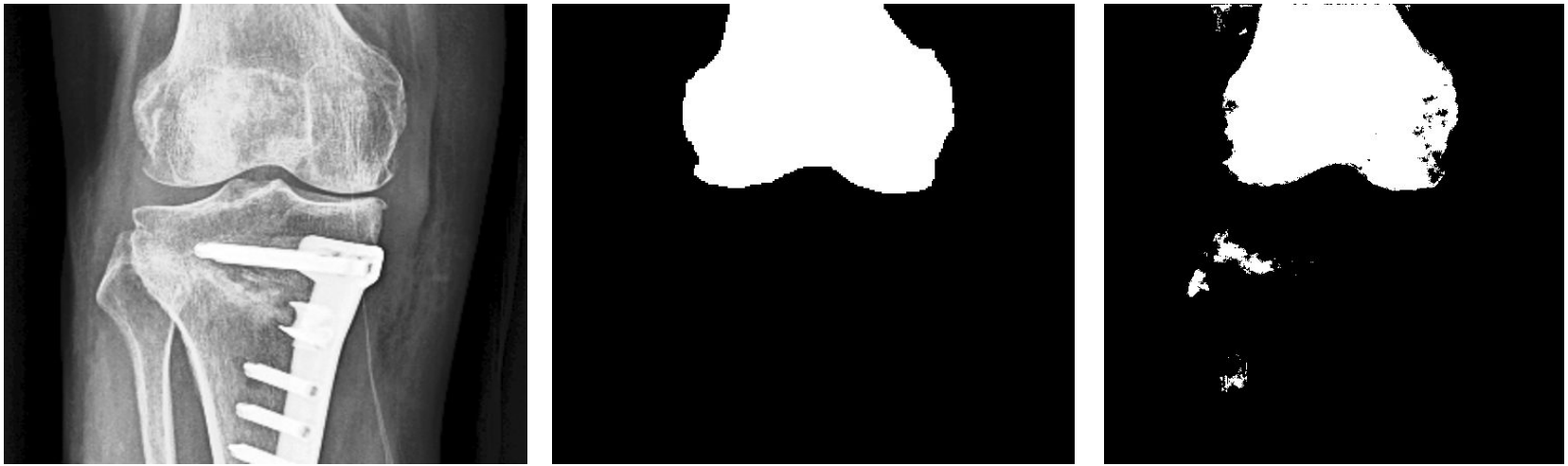
Data Augmentation 진행

두 이미지를 비교해서 성능을 측정하는 Dice score 추가

END

피드백으로 인한 변경사항

- inference 결과에 Threshold를 적용하여 기준보다 작으면 0, 크면 1로 변경



Good!

Dice score

- 이미지등의 Segmentation 에서 쓰이는 지표

- 영상 이미지등에서 정답과 예측값간의 차이를 알기위해 사용

- 라벨링된 영역과 예측한 영역이

  정확히 같다면, 1

  그렇지 않을 경우에는 0

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

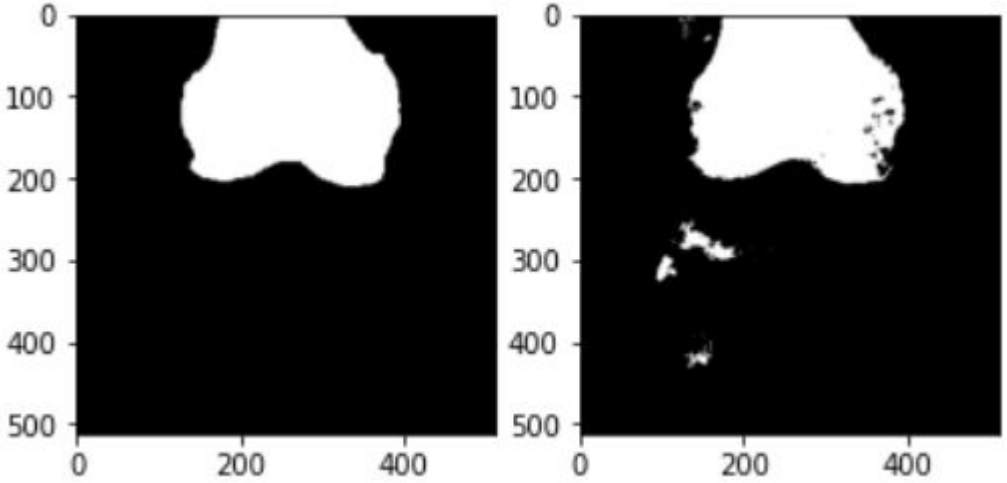
Dice score

```python
def dice(im1, im2, empty_score=1.0):
    im1 = np.asarray(im1).astype(np.bool)
    im2 = np.asarray(im2).astype(np.bool)

    if im1.shape != im2.shape:
        raise ValueError("Shape mismatch: im1 and im2 must have the same shape.")

    im_sum = im1.sum() + im2.sum()
    if im_sum == 0:
        return empty_score

    # Compute Dice coefficient
    intersection = np.logical_and(im1, im2)

    return 2. * intersection.sum() / im_sum
```

출처: https://gist.github.com/brunodoamaral/e130b4e97aa4ebc468225b7ce39b3137

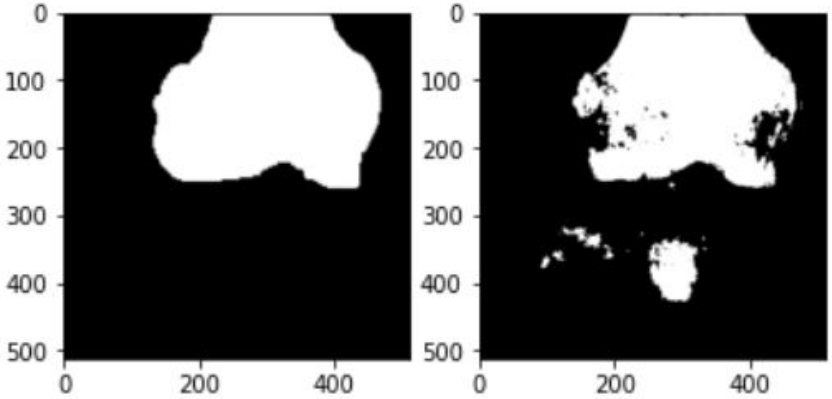
Dice score

결과값: 0.9486171291087918

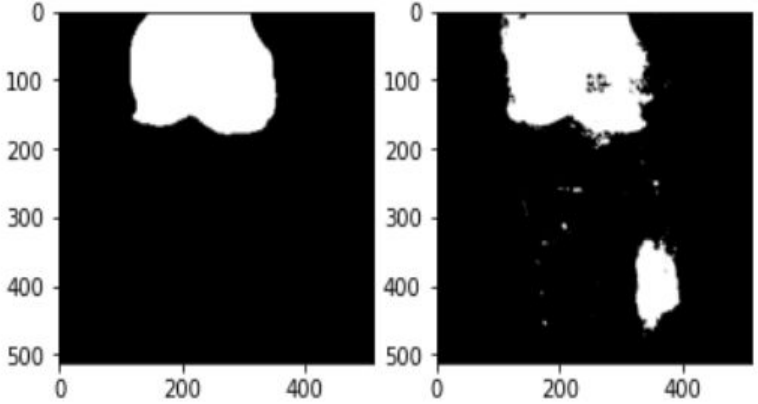


Dice score

결과값: 0.8803202034938091



결과값: 0.8615755990847143

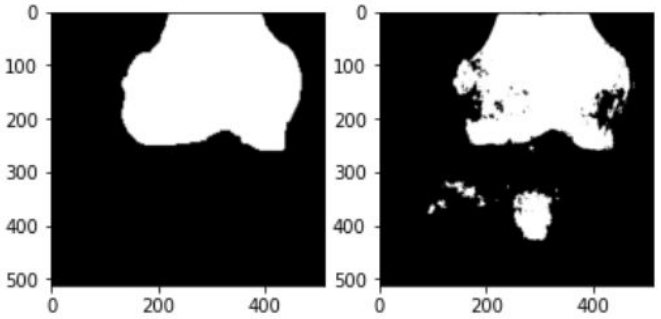


epoch and learning rate 변경

추가로 epoch와 learning rate를 변경해봤다.

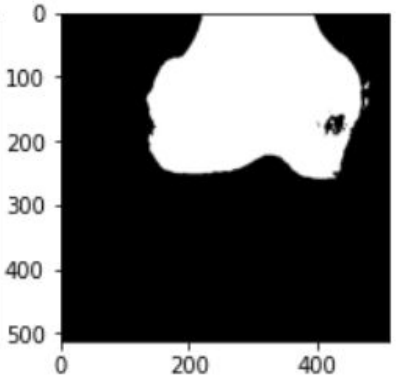
	before	after
epoch	100	30
learning rate	0.0001	0.001

- 성능이 더 좋아졌다.
- 적절한 파라미터 설정이 간단하면서도 중요한지 알 수 있다.



원본

before
dice score: 0.88



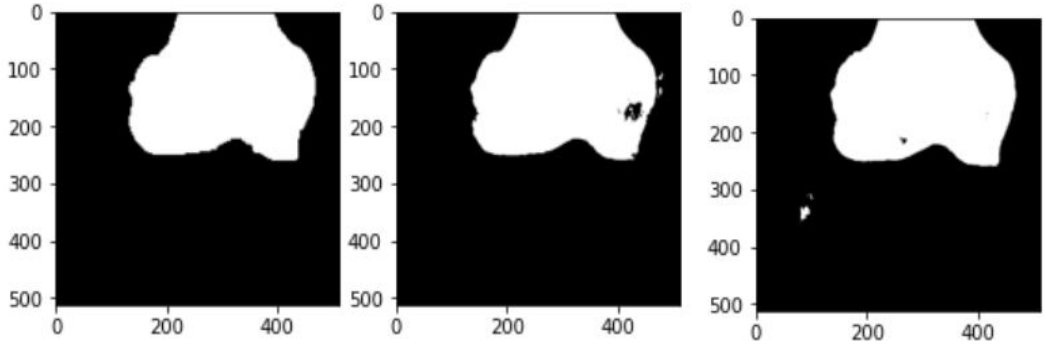
after
dice score: 0.98

input image normalization

- 추가로 input image에 normalization을 적용해봤다.

inputs = (inputs - mean) / std

- 약간이지만 더 성능이 좋아졌다.



원본

lr & epoch 변경 후

dice score: 0.98325

normalization 적용 후

dice score: 0.98334