◎기말고사 challenge 보고서 2021-2 기초인공지능

Spine Segmentation Challenge

2016145101 안장환

Summary

이번 챌린지는 '기초인공지능' 수업 기말고사로 진행되었고, spine x-ray image dataset에 대하여 7가지(background 포함)의 척추뼈로 구분 및 segmentation을 하는 task이다. 주어진 image dataset은 120장의 spine x-ray image이다. 모델 구성은 segmentation task에서 우수한 성능을 보이는 U-Net model로 구성하였다. Input되는 이미지에 대하여 output 결과는 input size와 같도록 하되 channel은 목표하는 7로 해주었다(batch * 7 * width * height). 이 경우 학습이 진행되지 않아 channel을 6으로 하여 원하는 6가지의 spine에 대해 segmentation된 결과를 합쳐 background를 만드는 전략을 택하였다.

1. Dataset

Dataset은 학습과 평가에 사용될 이미지(dataset)와 정답지(label)로 주어졌다. Dataset은 파일 형식이 DICOM 파일로서 numpy와 같은 형식으로 바로 image로 시각화하거나 학습에 바로 적용할 수 있는 데이터가 아니었다. 그리고 학습에서 정답지로 활용될 label data는 mat file이었다.

```
#absolute path of img and label
input_abs_path = os.path.join(*[self.data_dir, 'img', self.lst_input[index]])
label_abs_path = os.path.join(*[self.data_dir, 'label', self.lst_label[index]])

#np.ndarray of img and label
images = sitk.ReadImage(input_abs_path)
input_np =sitk.GetArrayFromImage(images).astype('float32')
input_np_copy = input_np.copy()

#input array를 0~1 float 로 바꿔줌
input_np_copy1 = input_np_copy - np.min(input_np_copy)
input_np_copy = input_np_copy1 / np.max(input_np_copy1)

#label numpy array
label_mat_file = scipy.io.loadmat(label_abs_path)
label_np = label_mat_file['label_separated']

label = label_np.transpose(2,0,1)[:6,:,:].copy()
# label[6] = -label[6]+1

#data['input'] = 0~1 float32 , data['label'] = 0~1 float32
data = {'input' : input_np_copy, 'label' : label.astype('float32')}

if self.transform:
    data = self.transform(data)

return data
```

Fig 1. Dataset class __getitem__ method

본 프로젝트에 사용할 framework는 Pytorch로, 학습을 위해 최종적으로 데이터를 pytorch tensor로 변환해야한다. Dataset class는 지정한 경로에 있는 dicom file과 mat file를 U-Net 모델 학습에 가능한 형태로 데이터를 변경해주는 역할을 한다. Fig 1에서 정의한 __getitem__ method는 두 종류의 데이터를 가져와 numpy type의 데이터로 변경해준다.

그리고 numpy 형식의 data를 이미지화하면 Fig 2에서 왼쪽에 있는 x-ray image와 같고, 오른쪽 두 가지(6개의 segmentation channel 중 2개)처럼 확인할 수 있다.

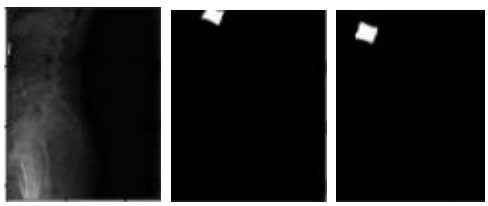

Fig 2. Input image and label image (right two)

2. Data transform

```
class ToTensor(object):
    def __call__(self,data):
        label, input = data['label'], data['input']

        input = input.astype(np.float32)
        label = label.astype(np.float32)

        data = {'label': torch.from_numpy(label), 'input': torch.from_numpy(input)}

        return data

class Normalization(object):
    def __init__(self, mean = 0.5, std = 0.5):
        self.mean = mean
        self.std = std

    def __call__(self, data):
        label, input = data['label'], data['input']

        input = (input - self.mean) / self.std
        data = {'label': label, 'input' : input}

        return data

class RandomFlip(object):
    def __call__(self, data):
        label, input = data['label'], data['input']

        if np.random.rand() > 0.5:
            label = np.fliplr(label)
            input = np.fliplr(input)

        if np.random.rand() > 0.5:
            label = np.flipud(label)
            input = np.flipud(input)

        data = {'label':label, 'input':input}

        return data
```

Fig 3. Data Transformer class

Data transform 역할을 수행하는 Data tranformer class

는 pytorch framework에서 기능을 가져와 사용가능할 수 있지만, data의 크기를 변경하는 등 customize해야하는 부분이 있어 직접 class로 정의하였다.

Fig 3을 보면 1. Dataset에서 가져온 input과 label numpy에 대해 tensor로 변형해주는 ToTensor class가 포함되어있다. RandomFlip을 통해 data augmentation으로 모델 학습과정에 발생할 수 있는 과적합(overfitting)을 방지한다.

```
class Rescale(object):
    def __call__(self, data):
        label, input = data['label'], data['input']

        # h, w = label.shape[1], label.shape[2]

        # small_val = h if h <= w else w

        # exp = 0

        # while small_val>=1:
        #     small_val = small_val / 2
        #     exp += 1

        # resize = 2**(exp-1)

        input = cv2.resize(input.transpose(1,2,0), (1024,1024))
        label = cv2.resize(label.transpose(1,2,0), (1024,1024))

        data = {'label':label.transpose(2,0,1), 'input':input}

        return data
```

Fig 4. Data transformer Rescale class

Input data의 shape을 확인해보았을 때 가장 작은 width를 넘지 않는 2의 거듭제곱 수 중 최대값은 1024였다. Fig 4처럼 2의 거듭제곱으로 data를 resize해주면 U-Net 모델을 통과한 후의 output이 1024 * 1024로 같을 것으로 기대할 수 있다.

하지만 최종 output이 (batch * channel * 1024 *1024)로 될 경우에는 원본 input이미지의 가로, 세로 사이즈와 달라지게된다. 따라서 학습할 시에는 label 이미지도 resize되기 때문에 관계없지만, 한 번도 확인하지 않은 data, 즉 test dataset으로 평가를 진행할 때는 input image의 크기를 이미지 순서대로 정렬하여 list에 저장해둔다. 그리고 output에 대해 하나씩 list에서 순서에 맞는 크기로 resize를 진행해 numpy 파일로 결과를 저장한다.

3. U-Net

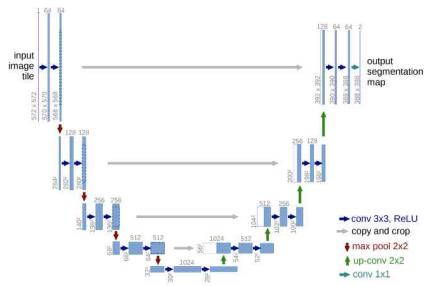

Fig 5. U-Net

본 Segmentation task에는 U-Net 모델을 사용하였다. U-Net은 Biomedical 분야에서 이미지 분할(Image Segmentation)을 목적으로 제안된 End-to-End 방식의 Fully-Convolutional Network 기반 모델이다. Fig 5를 보면 크게 왼쪽 위에서 아래로 내려가는 contracting path과 오른쪽 위로 올라가는 expanding path가 있다. Contracting path에서는 이미지 크기가 max-pooling 과정을 통해 절반씩 줄어들고, channel 수는 두 배씩 늘어난다. 그리고 expanding path에서는 feature map의 크기가 두 배씩 늘어나고 skip-connection을 하고 channel 수를 반으로 줄여준다. 마지막에는 채널 수를 조절할 때 유용한 1 * 1 convolution을 통해 64 channel의 feature map을 7 channel(background 포함) 또는 6 channel(background 미포함)으로 줄여준다.

4. 학습

```
class UNET(nn.Module):

    # U-Net에서 활용할 기본 block 등을 정의합니다.
    def __init__(
            self, in_channels=1, out_channels=7, features=[64, 128, 256, 512],
    ):
```

Fig 6. U-Net 7 output channel to 7

처음 설계했던 모델을 학습시킨 방법은 다음과 같다. (1 * 1024 * 1024)로 resize시킨 input을 넣고 (batch * 7 * 1024 * 1024)의 output을 얻은 뒤 다시 원래 이미지의 크기로 resize시킨다. Output channel은 label 채널과 같게 7 channel로 해주어 각각의 channel이 label과 같게 척추뼈 하나씩에 해당되는 segmentation image를 나타낼 것으로 기대했다.

Batch size는 4를 했을 경우에 out of memory 문제가 발생하여 2로 했다. Loss는 pytorch의 BCEwithLogitsLoss를 사용하였다. Optimizer는 보편적으로 좋은 성능을 보이는 Adam optimizer를 사용하고 learning rate를 0.001로 실험했다. Epoch은 우선 100으로 시행하여 loss를 통해 학습이 잘 되는지 보고 output을 image로 plot하여 시각적으로 확인하고자 했다.

Fig. U-Net 7 channel output

그러나 7 channel로 output을 만드는 경우에는 loss가 양수와 음수를 넘나들고, output 이미지를 확인했을 때에도 전혀 segmentation이 진행되지 않았다.

따라서 output을 6 channel로 하는 경우로 학습을 진행했다. Loss, optimizer, epoch, hyperparameter는 7 channel일

때와 동일하게 했다.

```
Train: epoch 0001 / 0100 | batch 0001 / 0050 | loss 0,6970
Train: epoch 0001 / 0100 | batch 0002 / 0050 | loss 0,6779
Train: epoch 0001 / 0100 | batch 0003 / 0050 | loss 0,6553
Train: epoch 0001 / 0100 | batch 0004 / 0050 | loss 0,6379
Train: epoch 0001 / 0100 | batch 0005 / 0050 | loss 0,6231
Train: epoch 0001 / 0100 | batch 0006 / 0050 | loss 0,6093
Train: epoch 0001 / 0100 | batch 0007 / 0050 | loss 0,5973
Train: epoch 0001 / 0100 | batch 0008 / 0050 | loss 0,5869
Train: epoch 0001 / 0100 | batch 0009 / 0050 | loss 0,5779
```

```
Train: epoch 0100 / 0100 | batch 0041 / 0050 | loss 0,0207
Train: epoch 0100 / 0100 | batch 0042 / 0050 | loss 0,0205
Train: epoch 0100 / 0100 | batch 0043 / 0050 | loss 0,0204
Train: epoch 0100 / 0100 | batch 0044 / 0050 | loss 0,0203
Train: epoch 0100 / 0100 | batch 0045 / 0050 | loss 0,0202
Train: epoch 0100 / 0100 | batch 0046 / 0050 | loss 0,0201
Train: epoch 0100 / 0100 | batch 0047 / 0050 | loss 0,0201
Train: epoch 0100 / 0100 | batch 0048 / 0050 | loss 0,0200
Train: epoch 0100 / 0100 | batch 0049 / 0050 | loss 0,0200
```

Fig 7. U-Net 6 channel output train loss

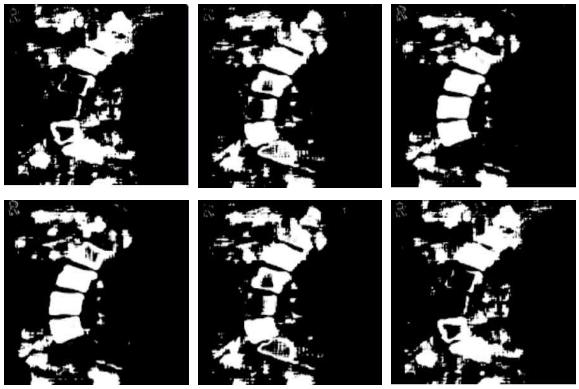

Fig 8. U-Net 6 channel output

Fig 7을 보면 loss값이 안정적으로 감소하는 것으로 보아 학습이 잘 진행되는 것으로 판단하였다. 그리고 학습이 종료된 후에 Output 이미지 Fig 8을 확인해보았을 때, 각각의 이미지들이 하나의 척추뼈에 대한 segmentation이 될 것으로 기대했지만 전체적인 척추뼈 모양을 segmentation을 한 결과가 되었다. Learning rate와 optimizer를 변경해보아도 차이가 없었다.

Test 이미지로 결과를 낸 output은 각각의 spine x-ray image에 대해 (row * column * channel)의 numpy 파일이어야한다. 위에서 6 channel로 실험한 결과의 output이 기대한 만큼 나오지 못했기 때문에 파일 형식을 맞추기 위해 마지막 6번 째 channel의 값을 7번째 channel의 값으로 복사해주었다.

실험해보지 못한 고려해볼만 사항으로는 loss를 BCEwithLogitsLoss가 아닌 DiceLoss를 사용했을 경우이다.