Contents

V-Net Technical Report

Jing Zhang

December 2019

1 Introduction

Convolutional neural networks can help people to classify the objects. For example, U-Net is excellent in 2D medical image segmentation. In view of the modality of medical images, images are not only including 2D but also volumetric images. For example, CT, MRI, X-Ray, B-Ultrasound, etc. So it's necessary to develop 3D medical image segmentation model. In this technical report, we introduce a representative volumetric medical image segmentation architecture–V-Net. We will start from data preprocess, then model design, the selection of loss functions, and how to predict using trained model.

2 Data preprocessing

2.1 Information of data

We use a dataset from a challenge[1]. It have training set and test set. In training set, there are 40 patients of CT data and corresponding ground truth. In test set, there are 20 patients of CT data. The format of data is "nii.gz", the average size of data is around 40 MB, the shape of data is (512*512*slices), which the number of slices is different in each patient. We can see that the original data is not suitable in size and shape. Because no matter what convoluntional neural networks require the fixed shape and relatively small size data. So it's necessary to do the data preprocessing before training the model using training data.

2.2 Preprocessing steps

- **Windowing**.Windowing, also known as grey-level mapping, contrast stretching, histogram modification or contrast enhancement is the process in which the CT image greyscale component of an image is manipulated via the CT numbers; doing this will change the appearance of the picture to highlight particular structures. The brightness of the image is, adjusted via the window level. The contrast is

[1]https://competitions.codalab.org/competitions/21145

adjusted via the window width[2]. The windowing operation is based on Houndfield scale, which is a quantitative scale for describing radiodensity.

- Normalization $new_{im} = \frac{im-\mu}{\sigma}$. Generally speaking, in the unnormalized case, gradient-based optimization algorithms (i.e., neural networks) will have a very hard time to move the weight vectors towards a good solution. Moreover, normalization improves the convergence speed[3].

- Remove images without organs. The method is to remove the ground truth images that only have the number "0" (it means that this group of images don't have any organs but only background.) and the corresponding CT image. This operation can better improve the efficiency of model when it is being trained.

- Cropping. The original size of patients is 512*512 in 2D dimension. The cropping is based on the contour of patient to cut the edge of useless blank things. This step can better improve the training efficiency.

- Resizing. The destination shape of 3D image is (128*128*64), so that they can fit the designed model. In this step, we use a professional Python library called SimpleITK to resize and resample data.

3 V-Net

3.1 The architecture of V-Net

V-Net (Milletari et al. 2016) is a 3D medical image segmentation convolutional neural networks. It contains two parts like U-Net. The left part is to learn the features automatically by convolution operation. The right part is to restore the data as input. At the same time ,skip connection is used between this two parts. Figure 1 is the architecture of V-Net.

3.2 Data loader

Different person has different method for loading the input data. Because the training dataset is packed as hdf5 format that is popular nowadays[4], in which it has two keys: data and label. Therefore, in our data loading part, we have 3 steps to read the dataset. The Python code is attached below.

1. Batch reading the patients in dataset;

2. Get the name of each patient(data);

3. Then, read .h5 file into different arrays according the keys .h5 file has.

[2]The definition if from: https://radiopaedia.org/articles/windowing-ct

[3]Blog:https://jovianlin.io/why-is-normalization-important-in-neural-networks/

[4]in this blog, the author explained why hdf5 is the most popular data format. https://realpython.com/storing-images-in-python/

```
import numpy as np
import h5py
import os

def iterate_folder(folder):
    file_name = []
    for root, dirs, files in os.walk(folder):
        for file_index in files:
            file_name.append(os.path.join(root, file_index))
    return file_name


def read_h5file(path):
    file = h5py.File(path, 'r+')
    images, labels = [], []
    for _ in file.keys():
        images = np.array(file["/data"]).astype("float64")
        labels = np.array(file["/label"]).astype("int32")

    images = images.reshape(images.shape + (1,)).astype(np.float64)
    labels = labels.reshape(labels.shape + (1,))
    print("Shape of image:",images.shape)
    return images, labels


def load_data(folder):
    input_filenames = iterate_folder(folder)
    for i in range(len(input_filenames))])):
        X = np.array(([read_h5file(input_filenames[i])
    image = X[:,0,:,:,:]#(number,type,x,y,z,channel)
    label = X[:,1,:,:,:]
    print(label.shape)
    print("load data successfully!")
    return image, label
```

3.3 Implementation details

Next, I will introduce how to implement a CNN model from the scratch. This version of V-Net is based on Keras 2.19. Someone implemented before[5], but it is not suitable any more because of the problem of Keras version. My work of V-Net is clear to see and easy to understand.

[5]I'm inspired from https://gist.github.com/ravnoor/a8d26c485cd39c1d9dd21af7c27ac232

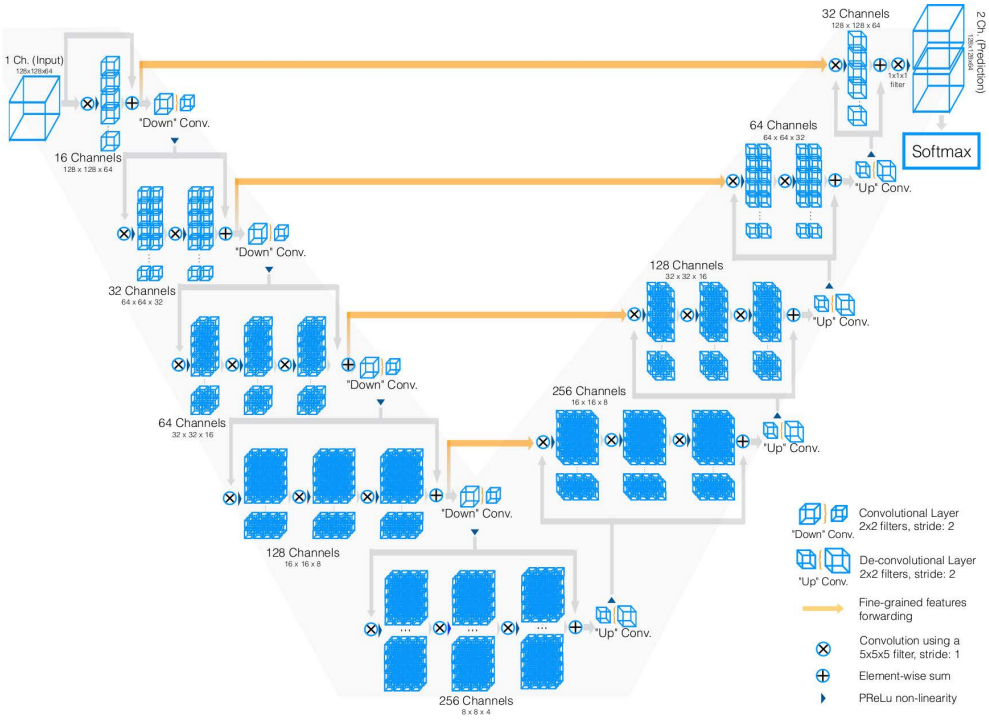

Figure 1: V-Net architecture

- Compression path: similar to ResNet, residual network. Attention, the input format of data is (batchsize, x, y, z, channels). So before we input the train data, we should check the format of input whether match the model input or not. Otherwise it will give an error. Here, we need two functions called transpose and reshape. Transpose is to change the order of dimension of data; reshape is to add one dimension in last dimension. Moreover, we should state the position of channal explicitly in the function in this way:

$$conv1 = Conv3D(filters, kernel\_size = (5,5,5), strides = 1, padding =' same',$$

$$data\_format =' channels\_last', kernel\_initializer =' he\_normal')(input)$$

  One tip is that: to check each layer's output if it is right or not, you can print the shape of each layer's tensor.

- Decompression path: Transposed convolutions instead of upsampling operations. An example:

$$up6 = Deconvolution3D(filters * 8, kernel\_size = (2,2,2), padding =' same',$$

$$data\_format =' channels\_last', strides = (2,2,2))(add6)$$

  After each layer, Batch normalization can be optional added depends on user's choice.

- Activation function: PReLu(parametric ReLu, parameters are learnable). An example:

$$up6 = PReLU()(up6)$$

- Parameters: 208,675,725. Because in decompression path, transposed convolutions are used, and PReLu activation function is used in each layer, so the amount of parameters is huge.

- Loss functions: Binary loss(Cross Entropy, Dice, Kappa, etc.) At which case to use binary loss or multi-class loss functions, this is a question. Generally, if there is only one target in the image, we definitely use binary loss in 2D or 3D image segmentation. If there is more than one target, intuitively, we should use multi-class loss function. However, there can be exceptions. The case we use binary loss or multi-class loss depends on two conditions. One is the channels that the ground truth data have, the other one is the output layer's number of feature map. That is to say, only when the number of channels of ground truth and that of output layer are exactly the same, for example, 2 or 3 or 4 or more, we can use multi-class loss function. Otherwise, we can only use binary loss. And it is not an error. It's just one of the solution. Someone would doubt, then how to distinguish different targets since the output of probability only have one channel? My suggestion here is: Given the ground truth data, each pixel have a unique number, which stands for background or targets. Then, we can re-assign these probability value into class number according to the same position of ground truth image as long as the probability value is greater than 0.5. In fact, this process is the prediction process, that is to say when we want to see the visualized segmentation results, we need to do this transform. In train stage or evaluation stage, there is no difference in binary or multi-class loss, unless you want to see each target's metric respectively. Another way is to preprocess the ground truth data that making them have multiple channels, each channels has one target, so that we can use multi-class loss function naturally.

- Metrics: Dice, Accuracy. Hausdorff distance is calculated after prediction stage. Then we can compare the predicted images and ground truth images using this metrics.

- Predict: This step is optional. If we want to see the visualized results. Change the probability value of final feature map to a integer number(from 0 to NumofClass) according to ground truth.

4 Experiments

We run the code in Colab[6].

[6]https://colab.research.google.com/notebooks/welcome.ipynb