Deep Convolutional Neural Network for brain segmentation: Two Pathway architecture

Matteo Causio

Academic year 2016/2017

Contents

0.1 Two-pathways architecture

Following the idea of Havaei et. al. in [1] we have implemented a so called *Two-Pathways* Convolutional network made up by a local and a global path. Both two paths get as input the four modality of a patch 33x33, centered in the pixel that we want to classify, in a slice of a RMI.

The global pathway is made up by a convolutional 2D layer with Rectified Linear activation function and 13x13 kernels that outputs 160 feature maps 21x21 followed by a drop out layer. The local pathway is made up by two blocks of convolution and max pooling layer. In the first block we have a convolutional 2D layer with Rectified Linear activation function and 7x7 kernels plus a max pooling layer with window size 4x4 followed by a drop out layer; the block outputs 64 feature maps 24x24. In the second block we have a convolutional 2D layer with Rectified Linear activation function and 3x3 kernels plus a max pooling layer with window size 2x2 followed by a drop out layer; the block outputs 64 feature maps 21x21. The two path are then merged in a concatenation layer that concatenate the feature maps of both the path and hence outputs 224 concatenated feature maps of size 21x21 that are then fed to the outputs layer. The output layer too have a convolutional shape instead of the largely used fully connected layer; this is because of the better performance in test time. It is made up by a convolutional layer with kernels 21x21 followed by a softmax and it outputs a tensor 5x1x1 expressing the probabilities of each 5 labels for that pixel.
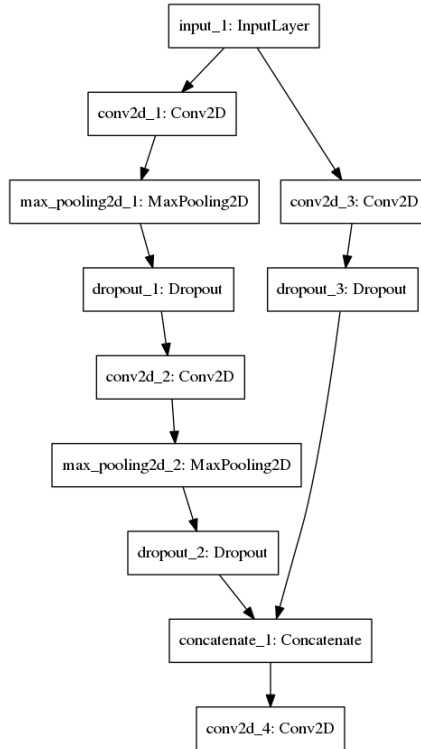

Figure 1: Two-pathways model

0.2 Cascade architecture

We have also implemented a cascade architecture model using two convolutional network of the type described above: the output of the first network is concatenated with the input of the second network. The first network gets as input the 4 modalities of a patch $65x65$ around the pixel to classify, then it outputs 5 feature maps $33x33$ that are concatenated with the 4 modalities of a smaller patch $33x33$ around the pixel. This 9 concatenated maps are then fed to the second Two-pathways CNN that outputs a tensor $5x1x1$ expressing the probabilities of each of each 5 labels.
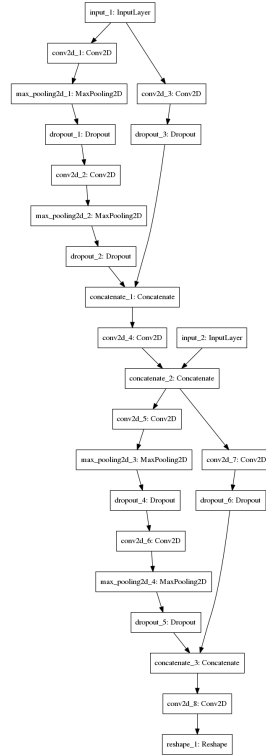

Figure 2: Cascade model

Some hyperparameters such as momentum, dropout, learning, decay, l1 and l2 rates have to be determined by cross-validation.

0.3 Two phase training

The training of a single two pathways CNN is divided in two phase to account the unbalanced nature of the data: in fact there are about 98% pixels classified as healthy, 0.18% as necrosis, 1.1% as edema, 0.12% as non-enhanced and 0.38% as enhanced. So in the first phase we train the model using patches with equiprobably labels, in other word for each class we select $\frac{T}{5}$ patches around a pixel labeled with such class; this enable the model to does not be overhelmed by the healthy pixels. In the second phase we freeze the parameters of all the layers except the output layer and then we train the parameters of the output layer with truly distributed dataset substrating

4

randomly the T patches from the training set; this permits to calibrate correctly the output probabilities.

0.3.1   Training for the cascade achitecture

To train the cascade model we apply the two phase procedure two times; once we have trained the first Two-pathways CNN, we freeze its parameters and then we train the second two pathways CNN. To train the first Two-pathways CNN we need $4x33x33$, so we can compare the segmented images and the targets during the training.

0.4   Implementation

We use *python 2.7* and Keras 2 running on top of *Tensor Flow*. We have implemented three classes called *BrainSegDCNN_2, patch_library* and *image_preprocessing*

0.4.1   image_preprocessing

The tasks of this class are to load the targets and the 4 modalities of the RMIs, both in format .mha, preprocess the latters with normalization and applying *n4 bias field correction* to correct the nonuniformity of the images. Finally it saves to specific folders, in format .png, the targets and the RMIs, compacted into a strip image containing the 4 modalities.

0.4.2   patch_library

This class is a patches extractor that selects classified pixels from the targets and exctract the relative $33x33$ and $65x65$ patches around that pixel from each modality of RMIs. It is useful to create the training set. The method *find_balanced_patches* selects $\frac{n}{5}$ pixels for each class, where n is the size of the training set, and extract the relative patches from RMIs; it is necessary for the first phase of the training where the labels have to be equiprobably. The method find_patches select randomly n pixels and extract the surrounding patches from RMIs modalities to create the training set needed to train the output layer during the second phase of training, where the labels have the true distribution.

0.4.3   BrainSegDCNN_2

This is the class designated to create and train the neural network and to make predictions of new data. The network is implemented using *Model* class in Keras. Setting the boolean field *cascade_model* we can create either a single Two-pathways DCNN or the cascaded model; during initialization the field *model* is setted using the method *comp_model*. Method *comp_model* first creates and compiles a single CNN and sets the field *cnn1* to it, then if the field *cascade_model* is true , the cascaded model is created and compiled using cnn1 as first part of the network followed by another single

Two-pathways CNN and the field *model* is setted to it, otherwise the field *model* is set to *cnn1*. The two phase training of the model is executed by the method *fit_model* that calls the helper method *freeze_model* to freeze the weights of the network and *fit_cnn1* to fit the first Two-pathways CNN; this method can fit either the single and cascade model.. To train the first two path CNN we need to change its input shape from $4x65x65$ to $4x33x33$, so in *fit_cnn1* we train a temporary Two-pathways CNN with such input shape and then we set the parameters of the CNN with input $4x65x65$ equal to the weights of the trained CNN with input $4x33x33$. The method *show_segmented_image* make prediction on new data using the *Model.predict* in Keras and show the segmented image using the library *matplotlib.pyplot*. There are also the method *save_model* and *load_model* to save and load a model and the values of their weights.

Bibliography

[1] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18 – 31, 2017.