Nabla-net: a deep dag-like convolutional architecture for biomedical image segmentation

Application to high- and low-grade glioma segmentation (BRATS)

Richard McKinley[1], Roland Wiest[1], Mauricio Reyes[4]

Application to white-matter lesion segmentation in Multiple Sclerosis (MSSEG)

Richard McKinley[1], Tom Gundersen[2], Franca Wagner[1], Andrew Chan[3], Roland Wiest[1], Mauricio Reyes[4]

[1]University of Bern - Institute for Diagnostic and Interventional Neuroradiology / SCAN, Bern, Switzerland [2] Red Hat AB, Oslo, Norway
[3] University Clinic for Neurology, Inselspital, University of Bern, Switzerland [4]University of Bern –Institute for Surgical Technology & Biomechanics, Bern, Switzerland

Abstract

Biomedical image segmentation requires both voxel-level information and global context. We present a deep convolutional architecture which combines a fully-convolutional network for local features and an encoder-decoder network in which convolutional layers and maxpooling compute high-level features, which are then upsampled to the resolution of the initial image using further convolutional layers and tied unpooling. We apply the method to segmenting multiple sclerosis lesions and brain tumors.

General Method

The fundamental basis of nabla net is a deep encoder/decoder network. Such a network comprises a series of encoder layers, each of which reduces feature maps with higher spatial dimensions to feature maps with lower spatial dimension, followed by a series of decoder layers, which expand features with low spatial dimensions to features with high spatial dimensions. Concretely, the encoder-decoder pathway of the nabla net applied to a 256 * 256 image would compute in the first encoder layer 256*256 feature maps, which are then reduced to 128*128 feature maps by maxpooling. This is repeated in encoder layer two, yielding 64*64 feature maps, and then one further time, yielding 32*32 feature maps. These feature maps are then upscaled by decoder layers, yielding subsequently 64*64 feature maps, 128*128 feature maps, and finally 256*256 feature maps. The upscaling is performed by using so-called “tied unpooling”.
The output of the pure encoder-decoder network can be used to predict a good localisation of lesions, but is unable to provide crisp boundaries: for that reason, the final prediction of a nabla net is produced by combining the original 256*256 layer of features before encoding with the output of the final decoder layer. These feature maps are then processed by a final fully convolutional layer, before the prediction of the lesion map is made. This ensures that a combination of low-level and high-level features are available for the prediction.

In both applications, three networks were trained, one operating in each of the sagittal, coronal and axial planes. The segmentations in each of those planes were fused to give the final segmentation.

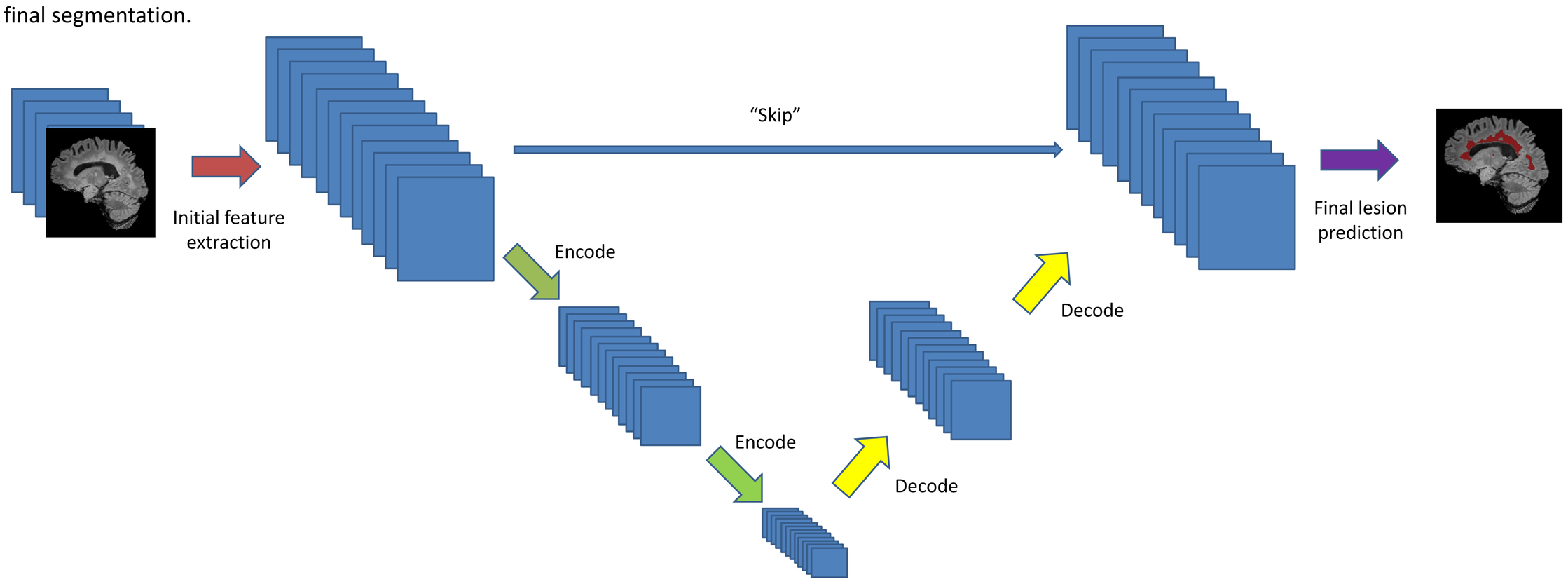

Figure 1 – Diagram of a nabla net with two encode/decode levels. The nets used in both challenges had three encode/decode levels. Input to the model is multimodal MR data: m modalities, and 2n+1 consecutive slices. The initial feature extraction comprises, for each MR modality, , 3 by 3 convolution, batch normalization, ReLU activation, 3 by 3 convolution, batch normalization, ReLu activation. If more than one MR modality is present, a further layer of the same shape is used to combine the features from the individual modalities. Encode layers have the same structure , followed by a 2 by 2 maxpool, and decode layers the same structure, but without the ReLU activations,, followed by tied unpooling. The final lesion prediction has the same structure, followed by 1 by 1 convolutions and a sigmoid activation to predict the lesion masks. All convolutions are preceded by zero-padding. The nets were built using the python package Keras [1], with Theano backend [2], and were trained using RMSprop, with reflection about the sagittal plane, and rotation in the axial and sagittal plane used a data augmentation.

Application: BRATS

A nabla net architecture was trained, on the 244 semi-manually labelled TCIA cases in the BRATS training set, with validation on the manually labelled BRATS 2012 dataset. The network predicts four lesion labels, following the manual tumor segmentation protocol: edema, gross tumor, contrast-enhancing tumor, and necrosis. Pre-processing is minimal, with only outlier suppression (restricting the range of the T2 and T1 maps to the [1^{st}, 99^{th}] percentile, the FLAIR to the [1^{st},98^{th}] percentile and the t1c to the [1^{st},96^{th}] percentile for each volume.
The final lesion masks were extracted by a random walk segmentation from heatmaps for each tumor label. Segmentation takes approx. 3 minutes on an NVIDIA GTX 980M GPU.

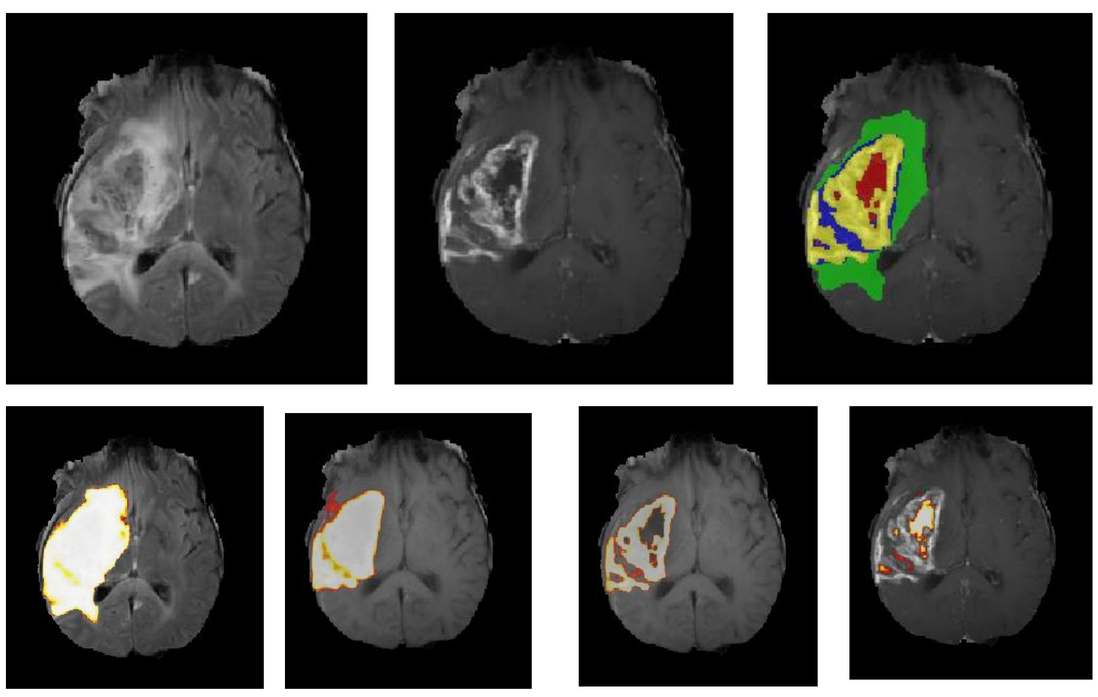

Figure 2 – Above: The BRATS nabla net segments brain tumors from FLAIR (left), T1c (middle), T1- and T2- (not shown) weighted imaging, yielding a lesion mask (right) of the tumor compartments.
Below: The four heatmaps (edema, gross tumor, enhancing, core) which comprise the raw output of the nabla net.

Application: MSSEG

A nabla net architecture was trained, on the 15 fused label sets of the MSSEG challenge dataset. Segmentation was performed using the skull-stripped, pre-processed FLAIR only. For each of the three directions {axial, sagittal, coronal}, 12 cases were used to train a classifier (four from each scanner type), with the remaining three being used to validate. The finial lesion masks were extracted by a random walk segmentation. Segmentation takes approx. 3 minutes on an NVIDIA GTX 980M GPU.

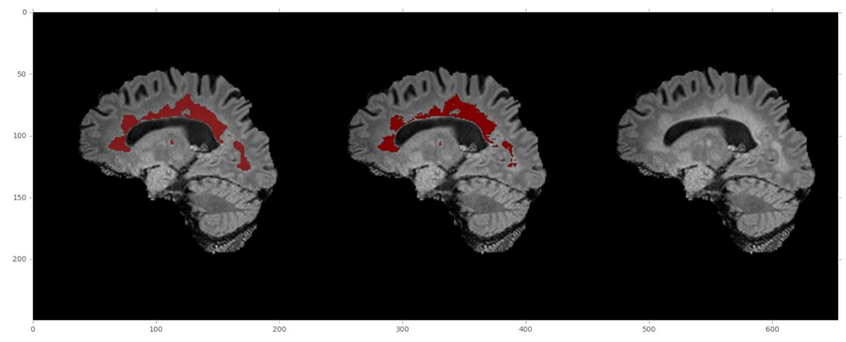

Figure 2 – A case from the Inselspital: Right: FLAIR image. Middle: Segmentation performed by a student, in the axial direction. Left: Segmentation performed by nabla net.

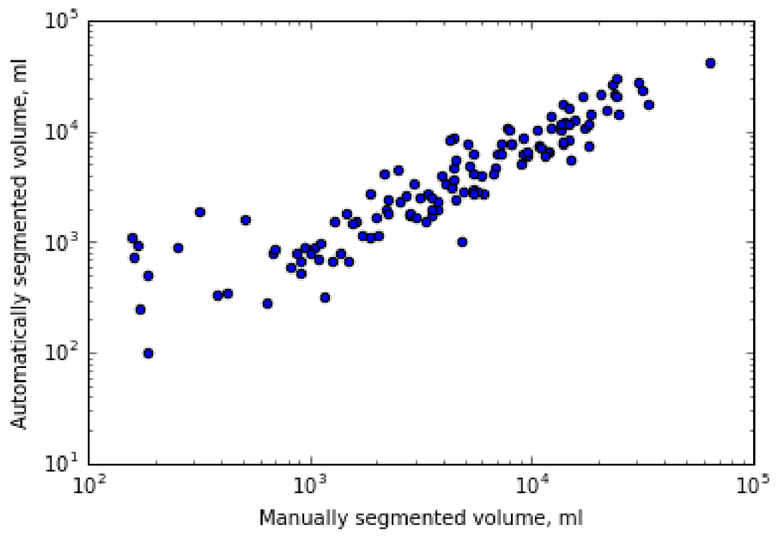

Figure 3 – Volumetric comparison of manual (masters student) and automated (nabla net) segmentation on 129 cases from the Inselspital, Bern, Switzerland. Log scale on both axes.

References

[1] Francois Chollet, Keras https://github.com/fchollet/keras
[2] Theano Development Team. “Theano: A Python framework for fast computation of mathematical expressions” http://arxiv.org/abs/1605.02688