Automated Segmentation of 3-D Brain Imaging Data Using Convolutional Nets

Gerald Pho

Introduction

Three-dimensional imaging approaches such as X-ray Computed Topography (CT) and magnetic resonance imaging (MRI) are widely used in both medical and research settings. These approaches generate vast quantities of high-resolution volumetric data, which are subsequently assessed by a trained expert. More quantitative assessments of these datasets are impeded by the fact that manual annotation of 3-D volumes is extremely laborious and time-intensive. Automated methods for segmenting volumetric imaging data is therefore an important but challenging problem for machine learning.

In this work, I seek to apply advances in deep neural networks, specifically convolutional networks, to the problem of segmenting volumetric imaging data. The specific problem I seek to address is to analyze X-ray computed topography (CT) images of rodent brains for the purpose of neuroscience research. My goal is to develop an algorithm that can automatically label voxels corresponding to experimentally-induced lesions in a dataset of multiple whole-brain volumes, ideally using only very sparse annotations of only a few slices per volume.

Approach: U-Net

Perhaps the most successful architecture for biomedical imaging segmentation is the U-Net (Ronneberger et al 2015). U-Net is an extension of the Fully Convolutional Network (FCN) architecture, which uses zero fully-connected layers and only layers for convolution, pooling, and upsampling. This architecture uses full images and obviates the need for highly inefficient patch-based methods which use sliding windows of each image for localization. U-Net is named for its U-shaped architecture, consisting of a contracting path followed by an expanding path. The contracting path is similar to a normal CNN with interleaved steps of convolution and max-pooling which successively increase the "receptive field" of each layer to learn context. The expanding path consists of up-convolution layers which successively increase the number of pixels until layer size matches that of the input. To maintain localized spatial information, U-Net includes "skip connections" which connect from the contracting to the expanding path, combining upsampled layers with earlier layers which still have high resolution features. This architecture, combined with aggressive data augmentation using elastic deformations, enabled U-Net to perform extremely well even with sparse training data, e.g. ranking 1st on the EM segmentation challenge in March 2015 using only 30 training images. More recent work extended the U-Net approach to volumetric data by using 3-D convolutional, pooling, and upconvolution layers (Cicek et al 2016).

In this project, I implemented both a 2-D and a 3-D U-Net and evaluated the performance of each algorithm on my rat brain dataset.

Part 1: 2-D U-Net

Dataset preparation

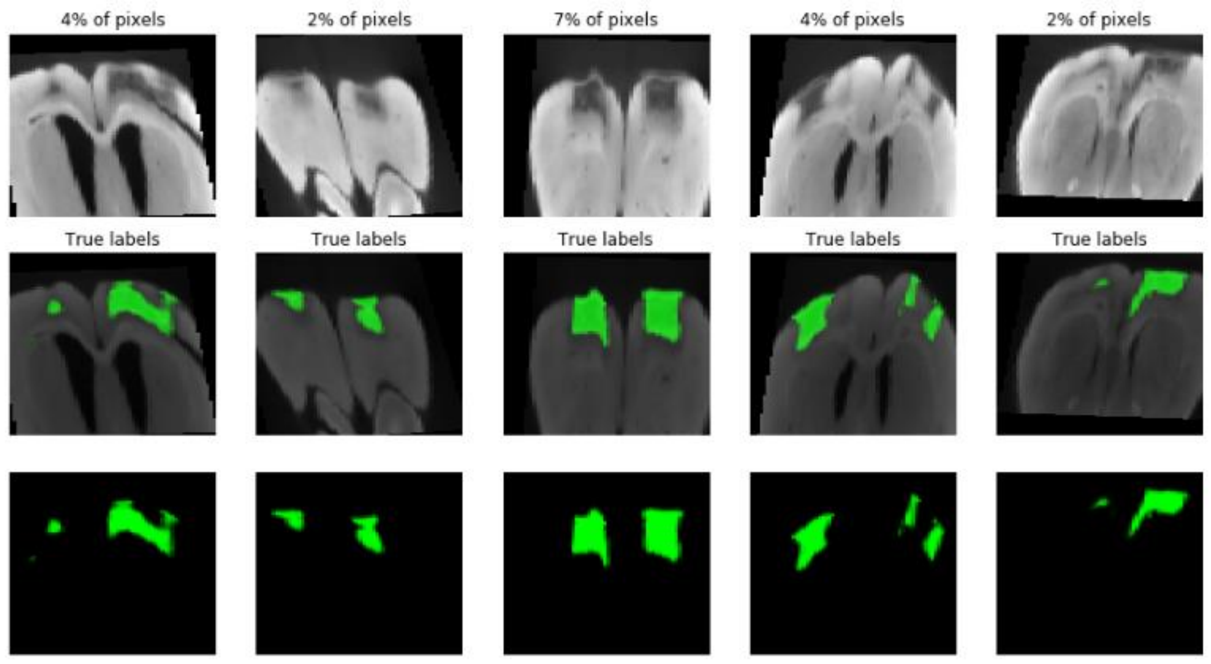
The dataset consists of three rat brain volumes, imaged with an approximate resolution of 15um X 15um X 15um per voxel. The size of each volume was about 1000 x 600 x 600 voxels. Because the lesions were quite large relative to the voxel size, I downsampled the data to just 15% in each dimension to save on memory and computational requirements. I then cropped each volume to achieve a consistent size of 128 x 64 x 64 voxels.

Labels were generated on the high-resolution images (before downsampling) with a combination of simple gray-level thresholding and manual curation. Because manual annotation was quite laborious, I focused my attention on generating high-quality labels for 10-13 coronal (XY) slices (relatively evenly spaced, each 128 x 64 pixels) for each brain. The remaining 50+ slices (with less accurately curated labels) were split in half to be used validation and test sets. Because the test sets are more poorly curated, one must take all quantitative performance metrics with a grain of salt.

Data augmentation

With such sparse training data, it is essential to perform data augmentation. I used Keras' built-in ImageDataGenerator to perform the following augmentations:

- Rotation: within 10 degrees
- Width/Height shift: within 10%
- Zoom: within 10%
- Shear: within 10%
- Horizontal flipping



Note that it is critical to perform the same exact augmentations on both the images and labels/masks.

2-D U-Net architecture

I used a U-Net architecture with a depth of 5. In the contracting path (the left side of the "U"), each "layer" consisted of two convolutional layers (size 3, stride 1), followed by a max pooling layer (size 2, stride 2). The number of channels/filters in each "layer" successively increased (16 to 32 to 64 to 128 to 256). The last layer had an output of size 8 x 4 x 256 and did not have a pooling layer.

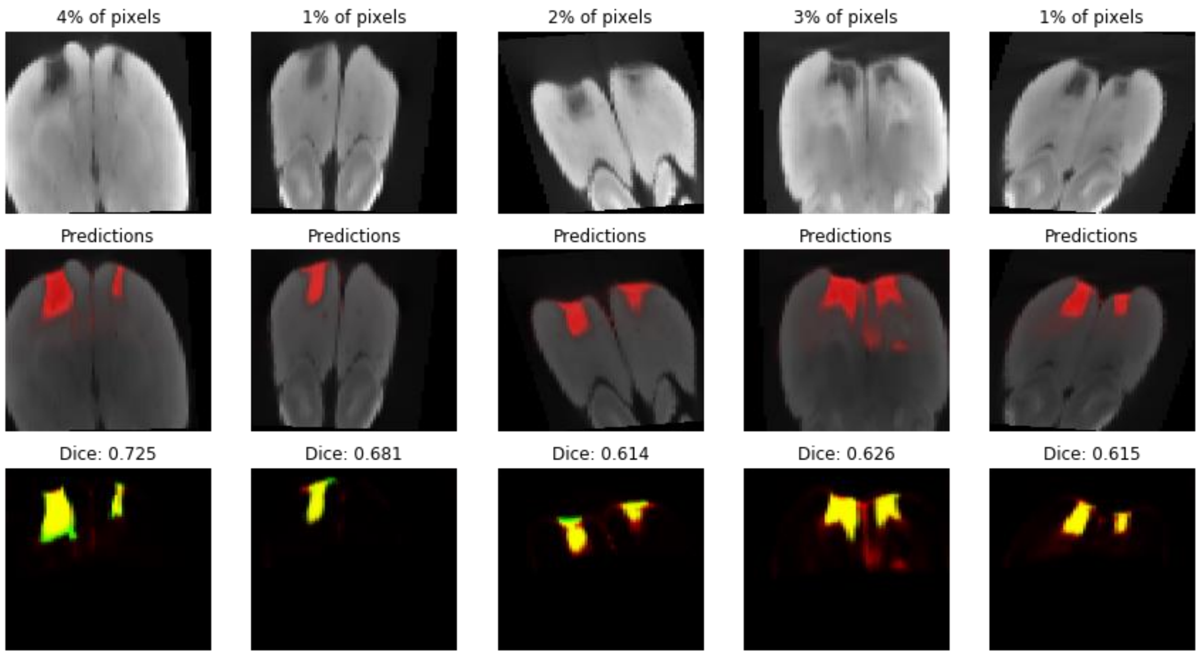
The expanding path utilized transposed convolutions (size 2, stride 2) for upconvolution, which allows learning of the upsampling parameters. Each upconvolution was followed by a skip connection, which concatenated the left side and right side of the U at each depth, and then two convolutional layers which mirrored the contracting path in channel count (from 128 up to 16).

The final output was a convolution with size 1 and a sigmoid activation which generates a probability for a binary output. In total, the architecture consisted of 1,940,817 parameters – all convolutional filters.

I did not experiment much with the architecture, except to optionally include dropout layers as a Boolean flag on the model generation function.

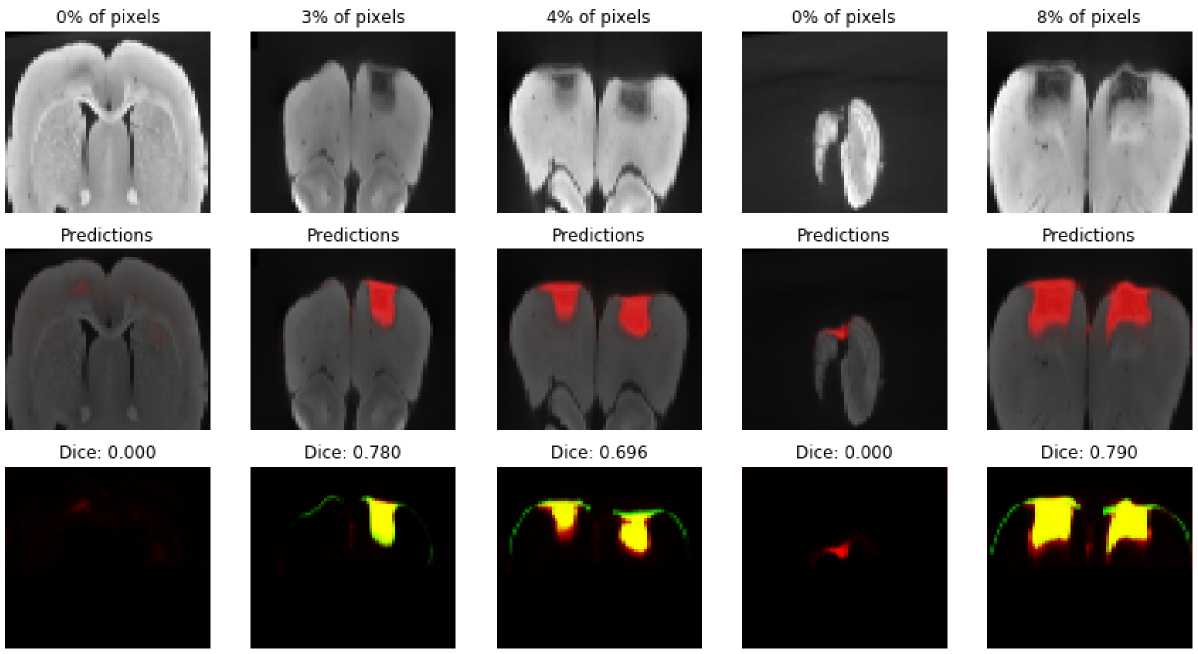
Evaluation metrics/losses

It turned out after experimentation that the choice of loss function was critical for good performance. I started out by using the dice coefficient, which is a direct measure of the performance of a segmentation algorithm. However, learning with this loss was not reliable, and I learned afterwards that the gradient of this loss can be unstable. I therefore ultimately used a weighted cross-entropy (log) loss. Weighting is necessary because of class imbalance – there are far more non-lesion pixels than lesion. (I used 0.95 for lesion and 0.05 for non-lesion). The choice of this weight strongly affected performance – with more time I would try to optimize this hyperparameter. See dice metric (range 0-1) as measure of performance on training data below:



Results

I trained the model for 20 epochs of 100 batches (batch size 32), using validation loss and the dice coefficient as metrics of performance. Qualitatively, the performance was quite good on the test data. It’s undoubtedly helpful that the test data comes from the same volumes, more work needs to be done to see how well it generalizes to out-of-sample brains.



Part 2: 3-D U-Net

The previous algorithm showed that a U-Net can successfully identify lesions to some accuracy using individual 2-D slices as input. However this approach throws away the 3rd dimension of depth which could significantly constrain the problem. I therefore sought to implement a 3-D U-Net (Cicek et al 2016) that could "interpolate" the segmentations between labeled training slices in a given rat volume.

Dataset preparation

I imitated the approach of the 3-D U-Net paper (Cicek et al 2016) by labelling additional slices in the orthogonal directions. Each volume therefore had information from 10-13 coronal slices (XY, 128 x 64), 8-11 horizontal slices (XZ, 128 x 64), and 17-22 sagittal slices (YZ, 64 x 64). All non-labeled voxels were given a different class label (0 for unlabeled vs. 1 for non-lesion and 2 for lesion) that had a weight of 0 in training. This approach, used by Cicek et al, allows the algorithm to make predictions on unlabeled voxels but only using labeled training data for learning.

The test and validation sets are different subsamples of the unlabeled voxels. As mentioned before, the "ground-truth" labels for the test sets were not carefully curated, so all quantitative metrics must be assessed with some skepticism.

Data augmentation

Data augmentation is even more essential in this regime, my entire dataset consists of just three samples. Unfortunately, Keras does not have a 3D ImageDataGenerator, so I had to implement my own using scipy.ndimage operations. The VolumeDataGenerator class I implemented performs the same rotation, shift, zoom, brightness, and flip augmentations, but in any of the three-dimensions. However, it's rather inefficient (1.8s per volume), which ended up being somewhat of a bottleneck in training the 3-D algorithm.

3-D U-Net architecture

I implemented the 3-D U-Net in exactly the same way as the 2-D U-Net, but with 3D convolution, max-pooling, and transposed convolution layers replacing their 2-D counterparts. I chose to use the same filter sizes, strides, and channel counts (but in 3D) to enable direct comparison with the 2D network. One minor change is that instead of using a binary sigmoid activation on a single output, the final output was a softmax on multiple outputs. This (in theory) would allow me to have more than 2 classes (for example, lesion, background, and brain) – though I didn't have time to pursue this in depth.

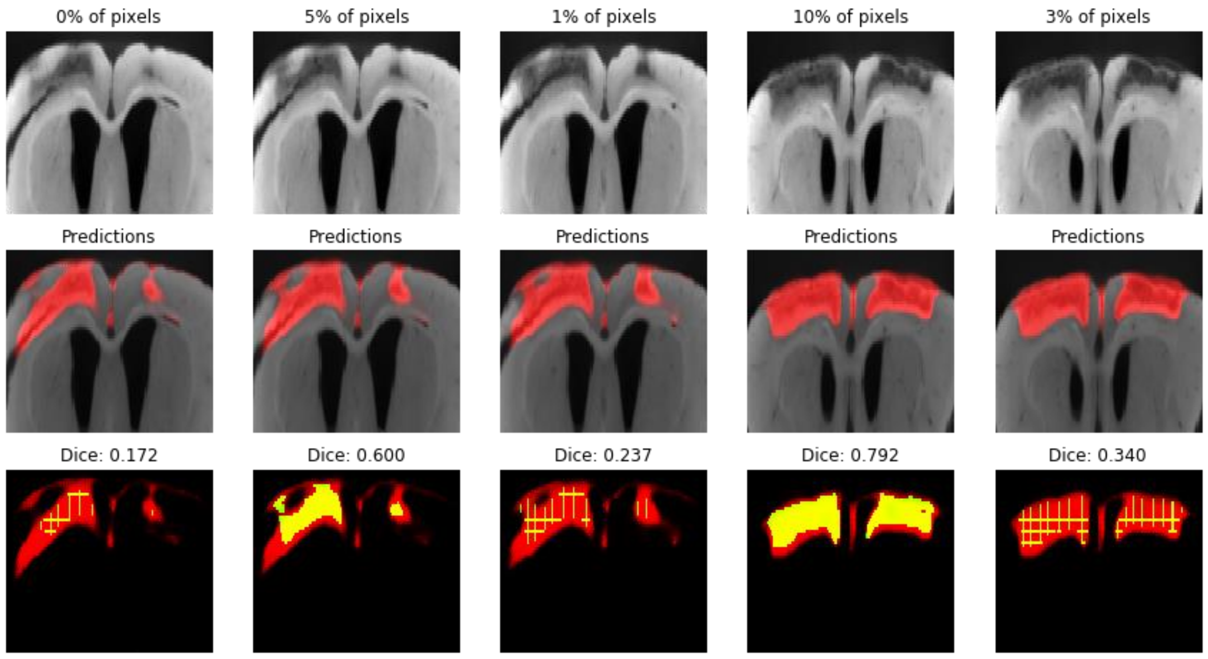
Altogether the 3D architecture has a measly 5 million parameters, being trained on just 3 training samples… could anything possibly go wrong?

Evaluation metrics/losses

As mentioned above, the weighted cross-entropy loss was only applied to labeled voxels. Nonetheless I forced the network to make predictions on unlabeled voxels (it must decide between class 1 or 2 -- class 0 is not an option). Therefore, the network output was one of two classes, while the labels themselves had 3 classes. I wrote a custom Keras loss function to handle this and the weights.

Results

Results with 3D U-Net were significantly worse. The algorithm does fairly ok at predicting when close to a ground-truth slice, but is much worse for slices distant from ground truth. The below plot shows 3 “held out” coronal slices interleaved by 2 training slices.



Conclusions and future work

Overall I am very impressed with how easy and effective U-Net was to implement for this problem. The 2D U-net in particular performed really well, but more work needs to be done to test its performance out-of-sample. I’m not entirely sure how to improve the 3D U-net. Perhaps better augmentation or more training would solve the problem, or optimization of the class weight parameter.

Lastly, it would be wonderful to develop an algorithm that could predict lesions just using image labels or bounding box labels, as even generating the ground truth pixelwise labels for this project was very laborious.