capstone_report.md

Machine Learning Engineer Nanodegree

Capstone Project

Irina Barskaya April 16th, 2018

I. Definition

Project Overview

Neurons are the major cellular component of the central and peripheral nervous systems that transmit and receive signals to and from the rest of the body. Axons are long, slender projection of neurons, which provide the pathway for signal transmission, and therefore allows transmitting information to different neurons, muscles, and glands. To increase conduction speed along the axons in white matter, myelin surrounds the axons in a sheath-like structure. Myelin is composed of ~ 80% lipid and 20% protein and forms in a lamellar, membranous structure and plays the significant role in proper nervous system functioning. [1-3] Myelin sheath damage or loss of myelin (so-called demyelination) results in diverse symptoms, including loss of vision/hearing, weakness of arms or legs, cognitive disruption, speech impairment, memory loss, difficulty coordinating movement or balance disorder.

Also, the amount of myelin (myelin volume fraction - MVF) is the hallmark of many neurodegenerative autoimmune diseases, including multiple sclerosis, acute disseminated encephalomyelitis, transverse myelitis, Guillain–Barré syndrome, central pontine myelinosis and many others. [4-5]

All current non-invasive medical imaging techniques can only indirectly investigate myelin and estimate its amount MVF, and, therefore, they do require some gold standard showing a ground truth MVF to test the precision and sensitivity of newly developed techniques. One of a gold standards for quantitative MVF validation is histology. Being invasive and destructive method, it provides direct information on the microscopic structure of cells and tissues. Using electron microscope one can directly image myelin and estimate its amount in different areas of a nervous system. To use histology as the gold standard for validation, one needs a robust method to quantitatively analyze electron microscopy images and to estimate myelin volume fraction. To be more precise, the method should provide a binary mask, which segments myelin (pixel value = 1) from a non-myelin background (pixel value = 0) and then calculates myelin volume fraction as a sum of pixels with value equals to 1.

In this project I develop fully-automated robust approach for myelin segmentation using deep learning methods.

Problem Statement

Across different research groups, there's a variety of independently written in-house code to analyze microscopy images and to segment myelin from microscopy images. Mostly, images are segmented manually or semi-automatically [6-9]. Manual segmentation provides flexibility for variations in images but is time-consuming and user-dependent. Conversely, fully automatic segmentation requires no user-input but must be very robust to adapt to variability in image illumination, structure, etc. Semi-automatic segmentation decreases user-dependency, provides some user-control to ensure proper segmentation, but still requires a lot of efforts to perform quality control. Recently, deep learning approaches were successfully used to provide fully-automatically, robust to variability between image illumination, structure binary masks.

Metrics

Evaluating the quality of segmentation by choosing an evaluation metric is an important step in designing a deep learning model. Many evaluation metrics have been used in evaluating segmentation, [10] and there is no formal way to choose the most suitable metric(s) for a particular segmentation task and/or particular data. So most researchers choose the evaluation metrics arbitrarily or according to their popularity in similar tasks. The Dice coefficient [11] (DICE), also called the overlap index, is the most used metric in validating biomedical segmentations. DICE measures the spatial overlap between two masks, X and Y target regions, and is defined as $\frac{2\left|X\right| \cap \left|Y\right|}{\left|X\right| + \left|Y\right|}$, where ∩ is the intersection.

II. Analysis

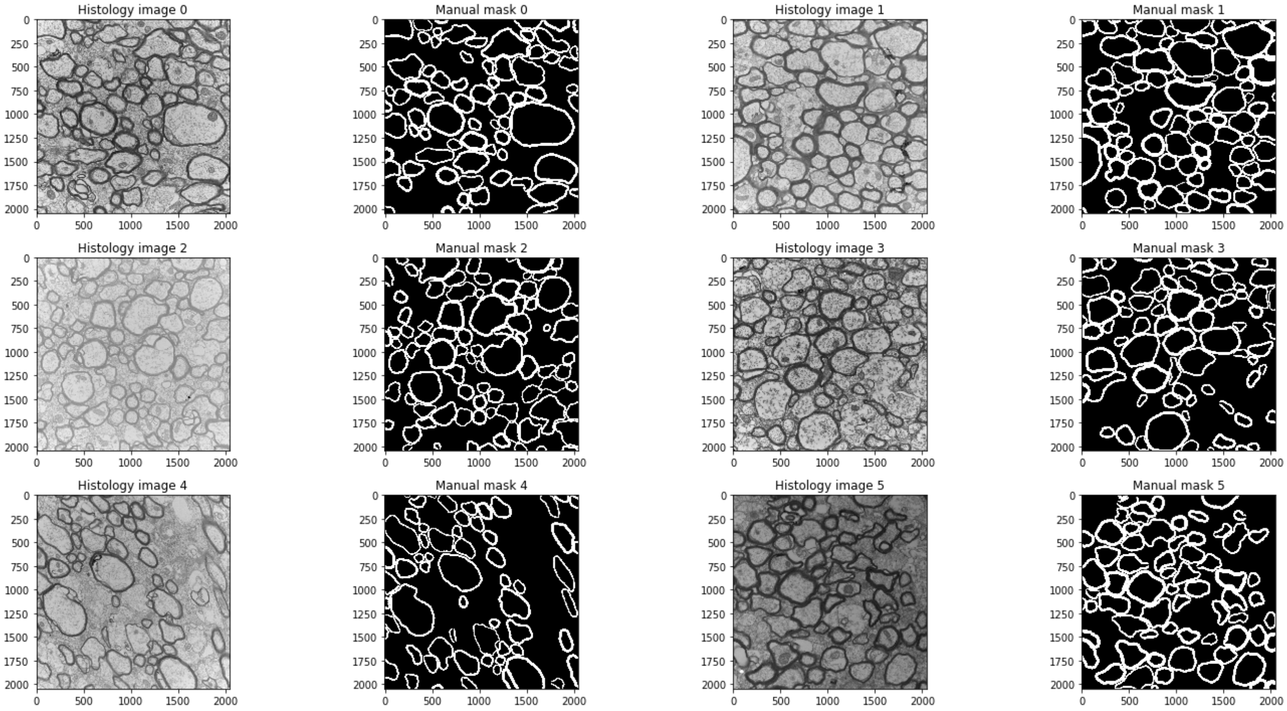
Data Exploration

In the process of REMMI project development at Vanderbilt University under supervision of Prof. Mark Does my colleagues collected electron microscopy images for 6 control rats in 4 different brain regions (the genu, mid-body, and splenium of the corpus callosum and the anterior commissure). Ultra-thin sections of brains (~ 500 x 500 x 0.07 µm) were imaged on the Philips/FEI Tecnai T12 electron microscope (FEI Company, Hillsboro, OR) at 15,000x magnification and pictures were acquired with a side-mounted AMT CCD camera. For quantification 6-12 high-resolution images were collected (~300 axons) per ROI per animal. So, in total there are 141 high resolution 2048 × 2048 raw images in greyscale. Each image was analyzed semi-automatically to derive a ground-truth binary mask and to estimate myelin volume fraction.

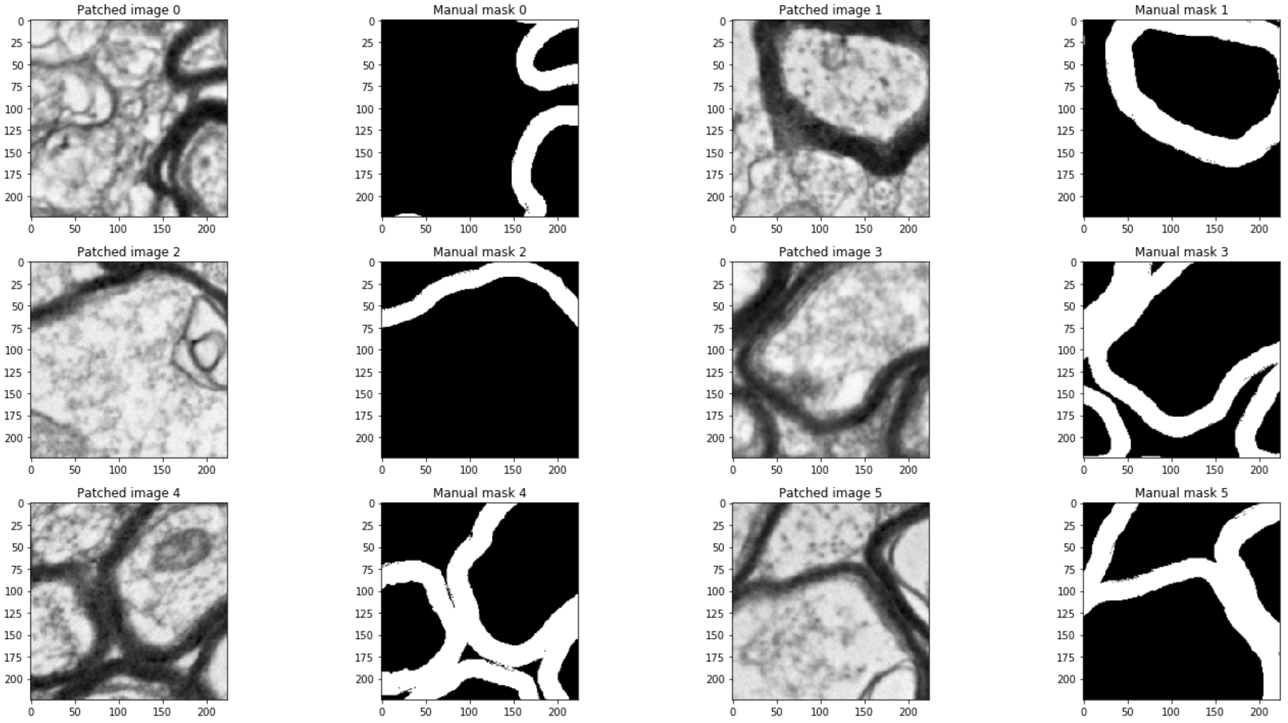
Therefore, to create deep learning neural network for electron microscopy images segmentation I had relatively big training dataset, consisting of high-resolution images of white matter and corresponding manually corrected binary masks for myelin segmentation.

Exploratory Visualization

As it was said in previous section, there are 141 high resolution 2048 × 2048 raw images and corresponding binary masks. Typically, there is a gray background with different dark-gray blotches (nucleii, proteins, etc) and almost black semi-circular shaped myelin layers. As you can see from example images, myelin layers vary in size, shape, thickness, and often overlaps with each other, making classical computer vision algorithms (watershed, thresholding) fail.



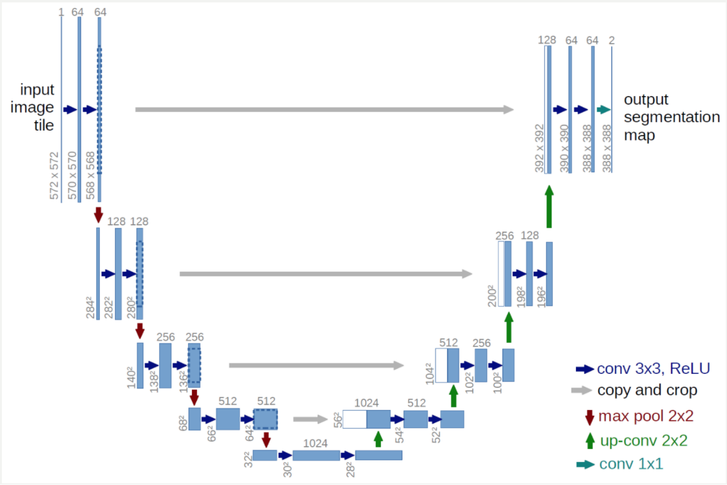
To train deep neural nets we definitely need a smaller size images to fit them into memory. Therefore, from each high-resolution image I randomly cropped 100 patches of size 224*224. So in total there are 5962 images with corresponding binary masks. In average, there are around 1-3 fragments of different axons on each image.



The whole dataset then was randomly splitted into train, validation and test datasets in 70:20:10 ration, having 4173 images in train dataset, 1192 images in validation dataset, 597 images in test dataset.

Algorithms and Techniques

The problem of building binary masks of images is actually a sub-class of semantic segmentation tasks. Semantic segmentation is understanding an image at pixel level i.e, we want to assign each pixel in the image to an object class. As with image classification, convolutional neural networks (CNN) have had enormous success on segmentation problems. One of the most successful for semantic segmentation challenge CNN approaches is encoder-decoder architecture. Encoder gradually reduces the spatial dimension by using pooling layers and decoder gradually recovers the object details and spatial dimension. To restore spatial information normally shortcut connections from encoder to decoder are used. The most popular encoder-decoder architecture for biomedical challenges is U-Net architecture [12]. As U-net up to now is the best method on the ISBI challenge for segmentation of neuronal structures in electron microscopic stacks; it has won the Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography at ISBI 2015, and it has won the Cell Tracking Challenge at ISBI 2015 on the two most challenging transmitted light microscopy categories (Phase contrast and DIC microscopy) by a large margin; has been actively used in Kaggle competitions for image segmentation tasks. [13] As U-net was originally designed for electron microscopy segmentation tasks, I've decided to implement that solution for my goals.

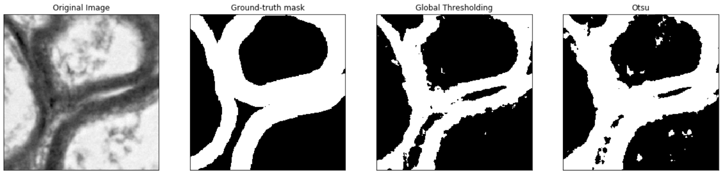


I have used the original architecture with some modification: 6 convolutional layers in both encoder and decoder, with 32, 64, 128, 256, 512 and 1024 filter. The detailed architecture with all the parameters could be found in Methodogy section.

Benchmark

As a benchmark model, I used two approaches which are often employed as a good start for manual and semi-automatic methods - global thresholding and Otsu's thresholding. [14] Global thresholding method is really simple and straighforward: all pixels falling below the threshold = 1 and are considered myelin, while all pixels above the threshold = 0 and are considered non-myelin. While this technique was a good start, it does not work consistently, produces not smooth, sharp masks with many artifacts. Global threshold was chosen manually to satisfy most of the images and was equal to 90.

Otsu's thresholding assumes that the image contains two classes of pixels following bi-modal histogram (myelin pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal.



I used OpenCV implementation for both methods. [15] As you can see on the image, both methods're having troubles with proper segmentation, as there are impurities on the images of the same color as myelin, the brightness and contrast of images significantly varies from image to image. Calculated dice coefficients (averaged across the whole dataset) for masks produced using 1) global thresholding method = 0.589; 2)Otsu's thresholding = 0.687.

III. Methodology

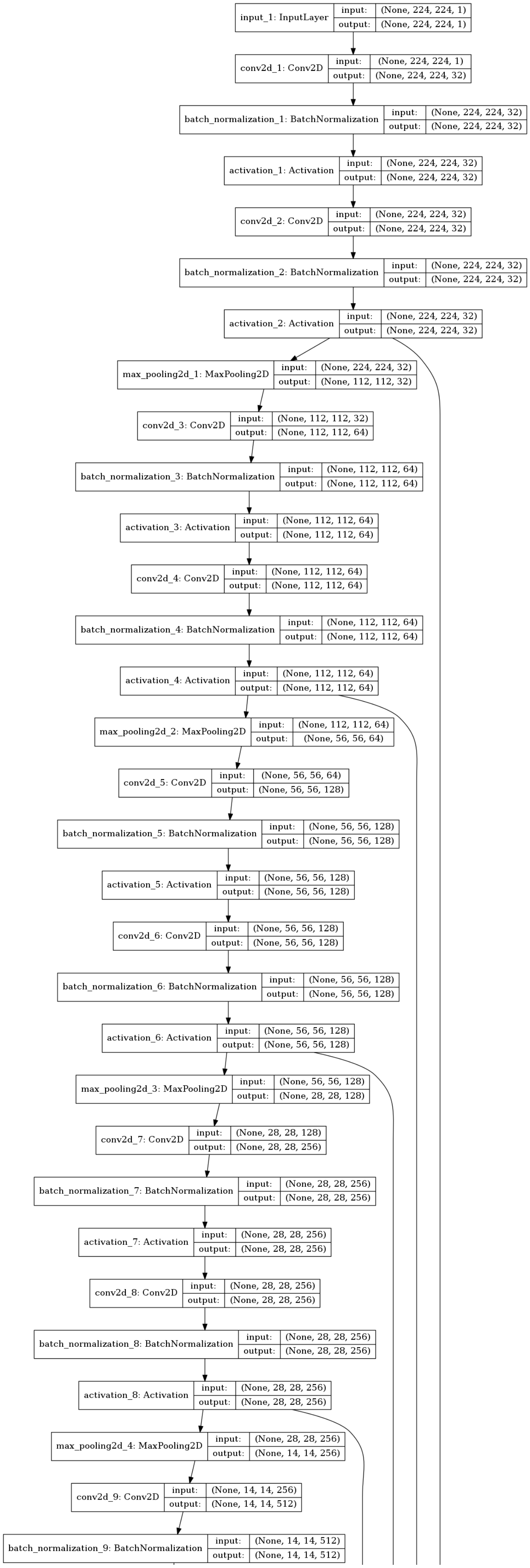
Data Preprocessing

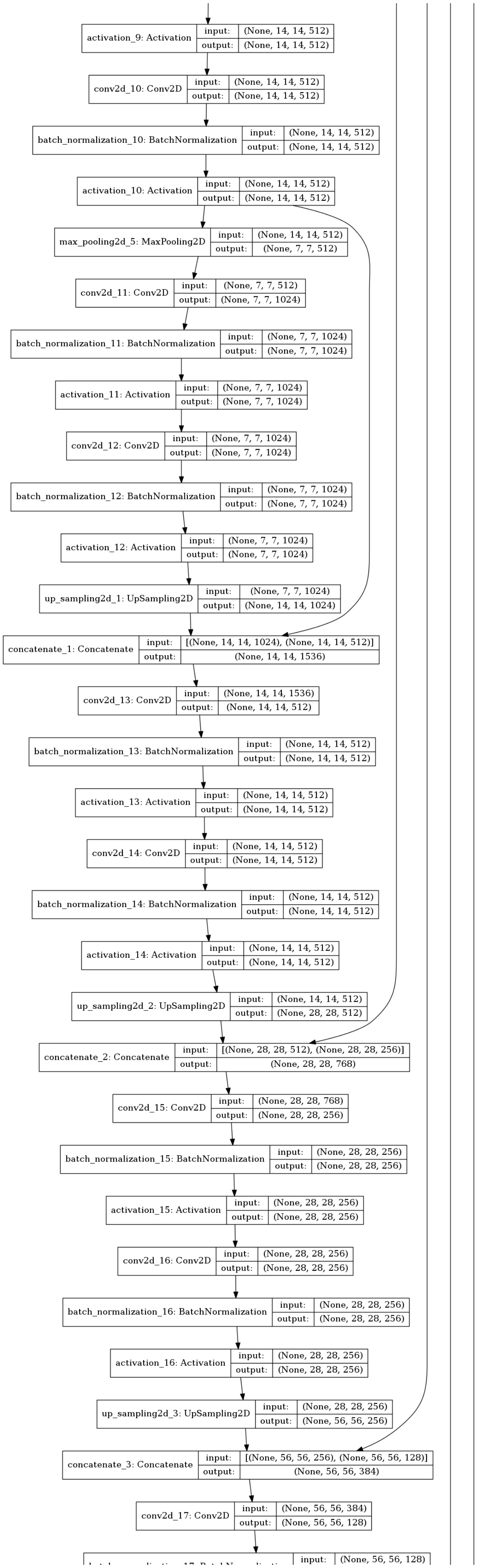
There was no specific image preprocessing. The only one: all input images were normalized to maximum (255) and substracted center value (0.5), all masks were normalized to maximum (255) as well. I wrote custom data generator for mini-batch training (data_generator.py), which read corresponding number of images and masks, preprocesses them and forms a batch.

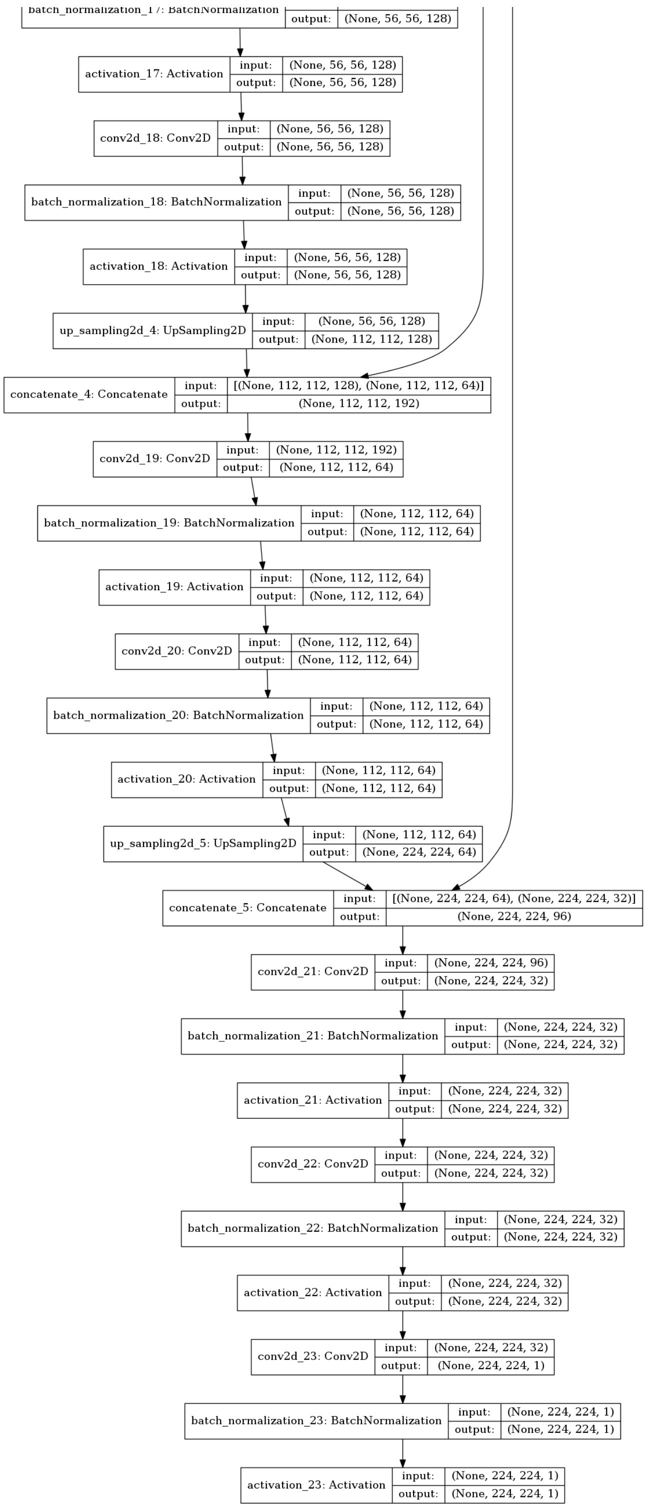
Implementation

All code was writen in Keras 2.1.3+ Tensorflow 1.7.0 with GPU support. All the required packages and their versions could be found in requirements.txt. For training I used GeForce GTX TITAN GPU 12 Gb.

As it was said earlier, I used U-net-like architechure with 6 convolutional layers in both encoder and decoder, with 32, 64, 128, 256, 512 and 1024 filter. The model is written using Keras core layers and placed in model.py file. The detailed architecture is the following:


input_1: InputLayer
input: (None, 224, 224, 1)
output: (None, 224, 224, 1)
conv2d_1: Conv2D
input: (None, 224, 224, 1)
output: (None, 224, 224, 32)
batch_normalization_1: BatchNormalization
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
activation_1: Activation
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
conv2d_2: Conv2D
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
batch_normalization_2: BatchNormalization
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
activation_2: Activation
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 224, 224, 32)
output: (None, 112, 112, 32)
conv2d_3: Conv2D
input: (None, 112, 112, 32)
output: (None, 112, 112, 64)
batch_normalization_3: BatchNormalization
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
activation_3: Activation
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
conv2d_4: Conv2D
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
batch_normalization_4: BatchNormalization
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
activation_4: Activation
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 112, 112, 64)
output: (None, 56, 56, 64)
conv2d_5: Conv2D
input: (None, 56, 56, 64)
output: (None, 56, 56, 128)
batch_normalization_5: BatchNormalization
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
activation_5: Activation
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
conv2d_6: Conv2D
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
batch_normalization_6: BatchNormalization
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
activation_6: Activation
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
max_pooling2d_3: MaxPooling2D
input: (None, 56, 56, 128)
output: (None, 28, 28, 128)
conv2d_7: Conv2D
input: (None, 28, 28, 128)
output: (None, 28, 28, 256)
batch_normalization_7: BatchNormalization
input: (None, 28, 28, 256)
output: (None, 28, 28, 256)
activation_7: Activation
input: (None, 28, 28, 256)
output: (None, 28, 28, 256)
conv2d_8: Conv2D
input: (None, 28, 28, 256)
output: (None, 28, 28, 256)
batch_normalization_8: BatchNormalization
input: (None, 28, 28, 256)
output: (None, 28, 28, 256)
activation_8: Activation
input: (None, 28, 28, 256)
output: (None, 28, 28, 256)
max_pooling2d_4: MaxPooling2D
input: (None, 28, 28, 256)
output: (None, 14, 14, 256)
conv2d_9: Conv2D
input: (None, 14, 14, 256)
output: (None, 14, 14, 512)
batch_normalization_9: BatchNormalization
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)

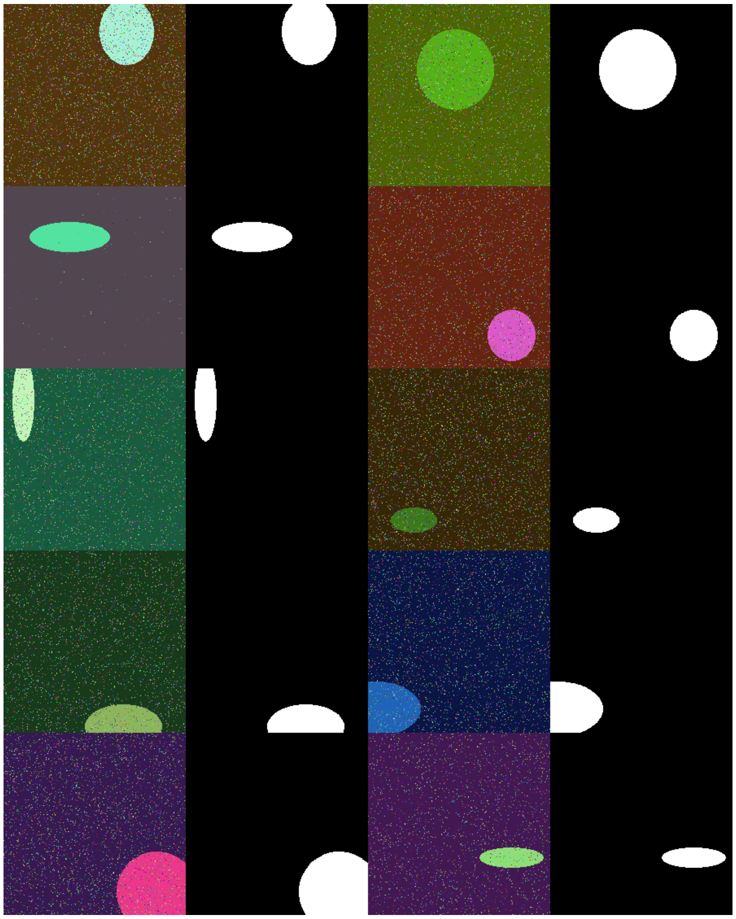

activation_9: Activation | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
conv2d_10: Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
batch_normalization_10: BatchNormalization | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
activation_10: Activation | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
max_pooling2d_5: MaxPooling2D | input: (None, 14, 14, 512) | output: (None, 7, 7, 512)
conv2d_11: Conv2D | input: (None, 7, 7, 512) | output: (None, 7, 7, 1024)
batch_normalization_11: BatchNormalization | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
activation_11: Activation | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
conv2d_12: Conv2D | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
batch_normalization_12: BatchNormalization | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
activation_12: Activation | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
up_sampling2d_1: UpSampling2D | input: (None, 7, 7, 1024) | output: (None, 14, 14, 1024)
concatenate_1: Concatenate | input: [(None, 14, 14, 1024), (None, 14, 14, 512)] | output: (None, 14, 14, 1536)
conv2d_13: Conv2D | input: (None, 14, 14, 1536) | output: (None, 14, 14, 512)
batch_normalization_13: BatchNormalization | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
activation_13: Activation | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
conv2d_14: Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
batch_normalization_14: BatchNormalization | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
activation_14: Activation | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
up_sampling2d_2: UpSampling2D | input: (None, 14, 14, 512) | output: (None, 28, 28, 512)
concatenate_2: Concatenate | input: [(None, 28, 28, 512), (None, 28, 28, 256)] | output: (None, 28, 28, 768)
conv2d_15: Conv2D | input: (None, 28, 28, 768) | output: (None, 28, 28, 256)
batch_normalization_15: BatchNormalization | input: (None, 28, 28, 256) | output: (None, 28, 28, 256)
activation_15: Activation | input: (None, 28, 28, 256) | output: (None, 28, 28, 256)
conv2d_16: Conv2D | input: (None, 28, 28, 256) | output: (None, 28, 28, 256)
batch_normalization_16: BatchNormalization | input: (None, 28, 28, 256) | output: (None, 28, 28, 256)
activation_16: Activation | input: (None, 28, 28, 256) | output: (None, 28, 28, 256)
up_sampling2d_3: UpSampling2D | input: (None, 28, 28, 256) | output: (None, 56, 56, 256)
concatenate_3: Concatenate | input: [(None, 56, 56, 256), (None, 56, 56, 128)] | output: (None, 56, 56, 384)
conv2d_17: Conv2D | input: (None, 56, 56, 384) | output: (None, 56, 56, 128)
input: (None, 56, 56, 128)


batch_normalization_17: BatchNormalization
output: (None, 56, 56, 128)
activation_17: Activation
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
conv2d_18: Conv2D
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
batch_normalization_18: BatchNormalization
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
activation_18: Activation
input: (None, 56, 56, 128)
output: (None, 56, 56, 128)
up_sampling2d_4: UpSampling2D
input: (None, 56, 56, 128)
output: (None, 112, 112, 128)
concatenate_4: Concatenate
input: [(None, 112, 112, 128), (None, 112, 112, 64)]
output: (None, 112, 112, 192)
conv2d_19: Conv2D
input: (None, 112, 112, 192)
output: (None, 112, 112, 64)
batch_normalization_19: BatchNormalization
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
activation_19: Activation
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
conv2d_20: Conv2D
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
batch_normalization_20: BatchNormalization
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
activation_20: Activation
input: (None, 112, 112, 64)
output: (None, 112, 112, 64)
up_sampling2d_5: UpSampling2D
input: (None, 112, 112, 64)
output: (None, 224, 224, 64)
concatenate_5: Concatenate
input: [(None, 224, 224, 64), (None, 224, 224, 32)]
output: (None, 224, 224, 96)
conv2d_21: Conv2D
input: (None, 224, 224, 96)
output: (None, 224, 224, 32)
batch_normalization_21: BatchNormalization
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
activation_21: Activation
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
conv2d_22: Conv2D
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
batch_normalization_22: BatchNormalization
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
activation_22: Activation
input: (None, 224, 224, 32)
output: (None, 224, 224, 32)
conv2d_23: Conv2D
input: (None, 224, 224, 32)
output: (None, 224, 224, 1)
batch_normalization_23: BatchNormalization
input: (None, 224, 224, 1)
output: (None, 224, 224, 1)
activation_23: Activation
input: (None, 224, 224, 1)
output: (None, 224, 224, 1)

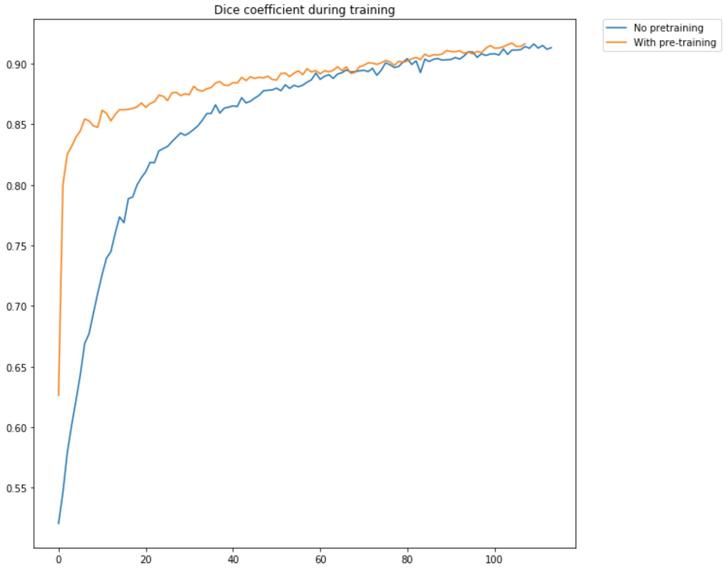
For training I used custom metric: dice coefficient and custom loss function, which is simply equal to -dice_coeff. Both, loss function and metrics are stored in losses.py file. The model was trained for 400 epochs with batch size equal to 12, Adam optimizer with learning rate 0.001. Training for one epoch takes approximately 100 seconds. All hyperparameters were optimized. All logs are save in unet_224_train.csv file and the final model is save to unet_224.h5 file. The best results are the following: dice coefficient for train dataset is equal to 0.9065 and 0.8962 for validation dataset.

Refinement

When searching on different improvement options, I found that pre-training on synthetic data is extremely helpful. Therefore, I used an approach suggested by Kaggle user ZFTurbo [16] to train the model first on synthetic dat. During pre-training, random images are synthesized as a combination of ellipse-shape figures of random color + background color + gaussian noise (generator code available in train_infinite_generator.py).



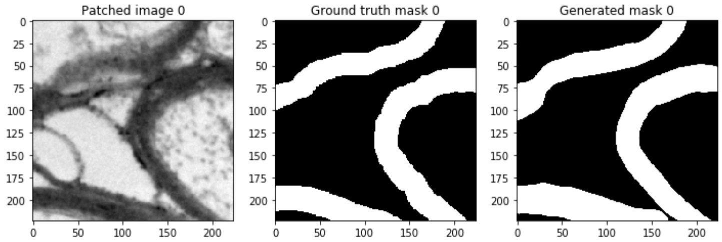
The idea is that it's easy for neural net first to catch basic simple shape figures, and then, from the warm start to segment more complicated electron microscopy images. I've adopted the approach and slightly modified the net for grayscale images. The pre-training took around 90 minutes and achieved 0.9828 dice coefficient for synthetic data. After that, using the weights from pre-training, I've started training on electron microscopy data. As you can see from training logs, using pre-trained model help to achieve higher accuracy with shorter training time.

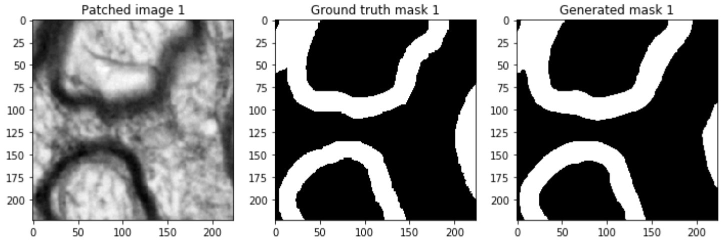


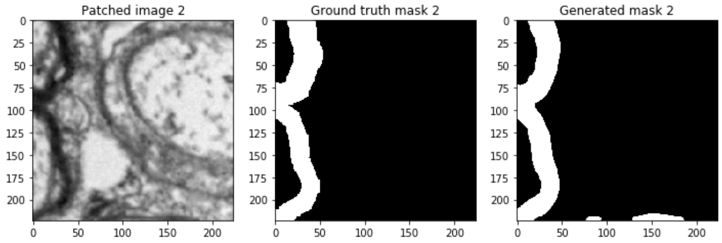
IV. Results

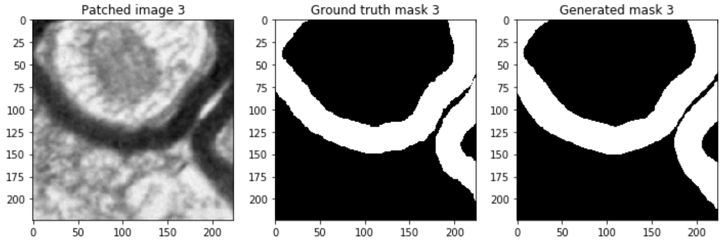
Model Evaluation and Validation

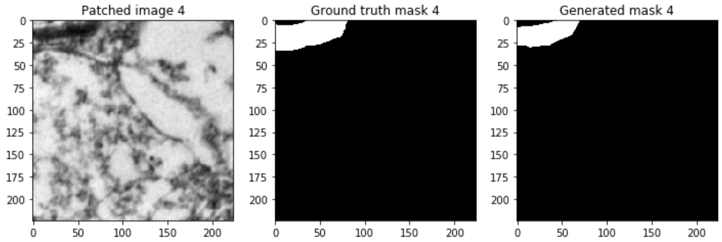
The final model with optimized hyperparameters and with pre-training on synthetic data produced really good, high-quality segmentation masks with dice coefficient on train/validation/test dataset: 0.9553 / 0.8905 / 0.8594. If you look at the examples of predicted masks, you can see that they have nice, smooth boarder with lack of artefacts.

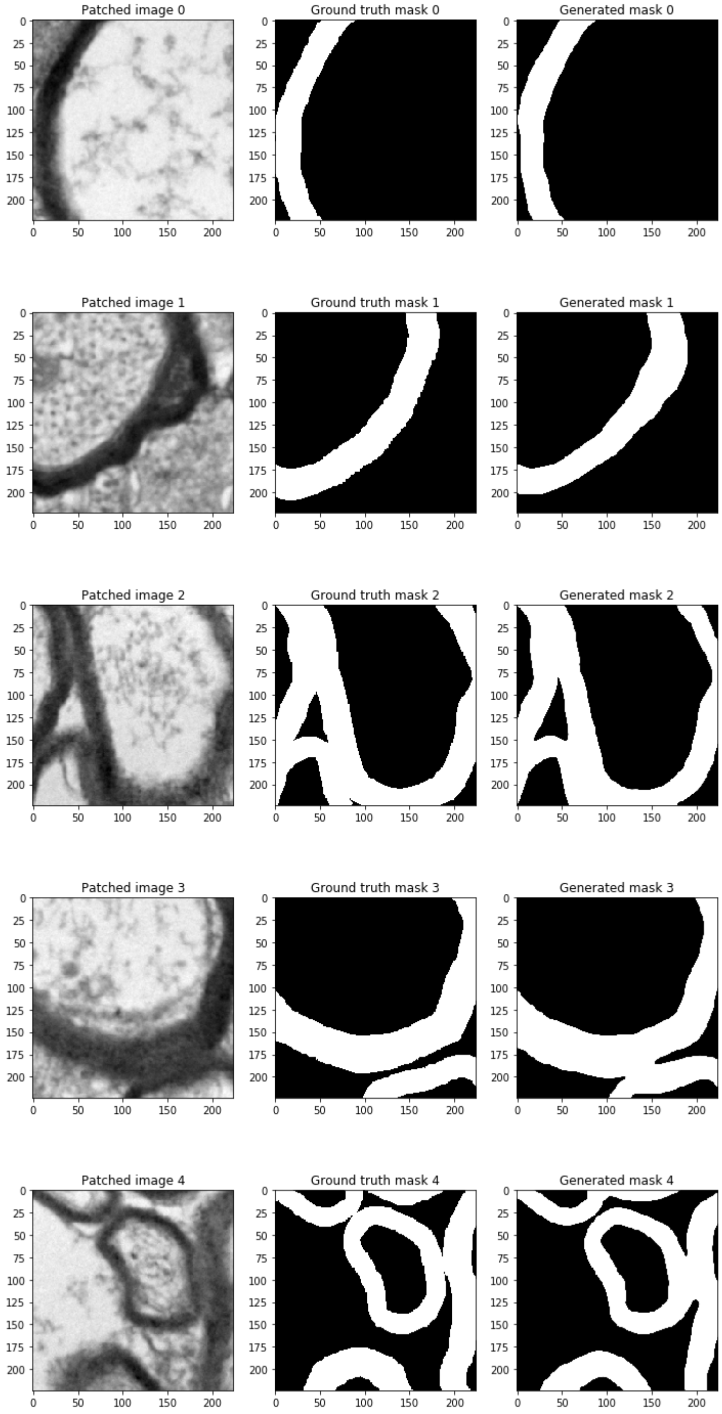


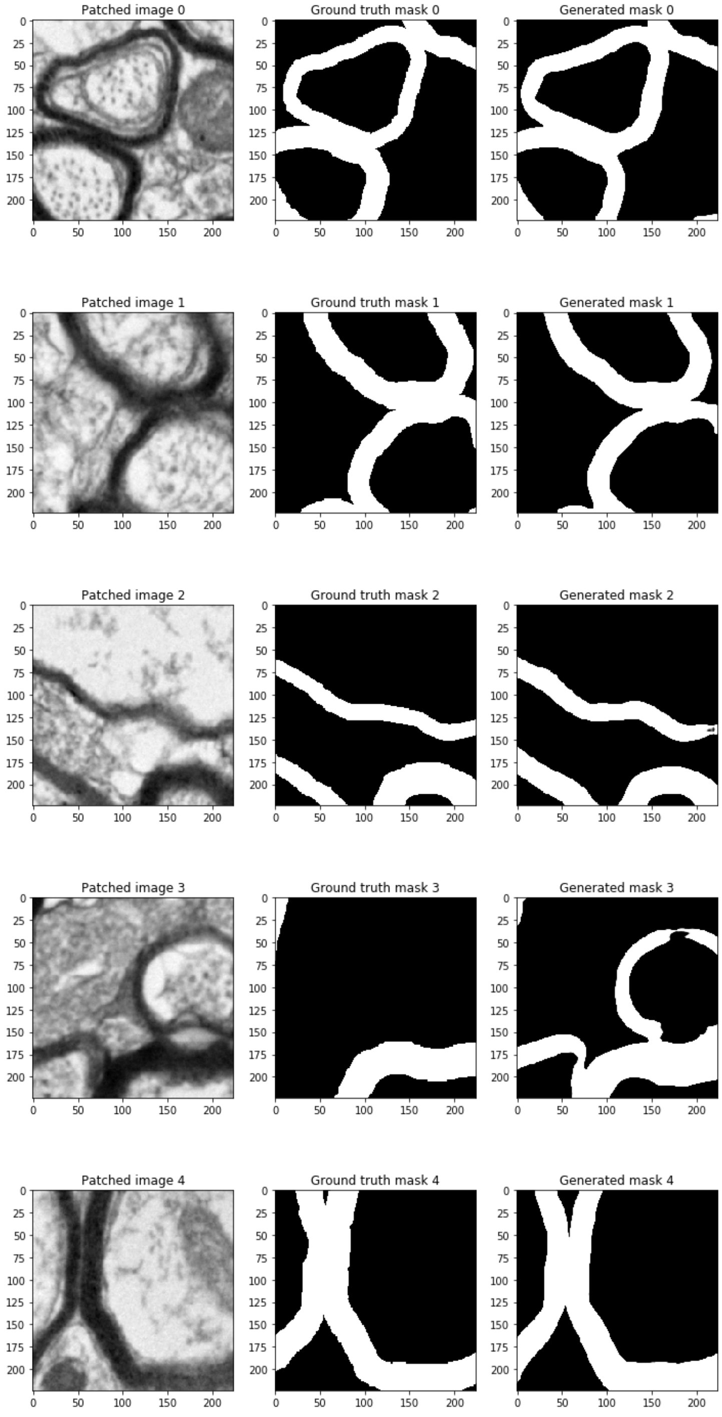









Patched image 0
Ground truth mask 0
Generated mask 0
Patched image 1
Ground truth mask 1
Generated mask 1
Patched image 2
Ground truth mask 2
Generated mask 2
Patched image 3
Ground truth mask 3
Generated mask 3
Patched image 4
Ground truth mask 4
Generated mask 4
0
25
50
75
100
125
150
175
200
0
50
100
150
200



Justification

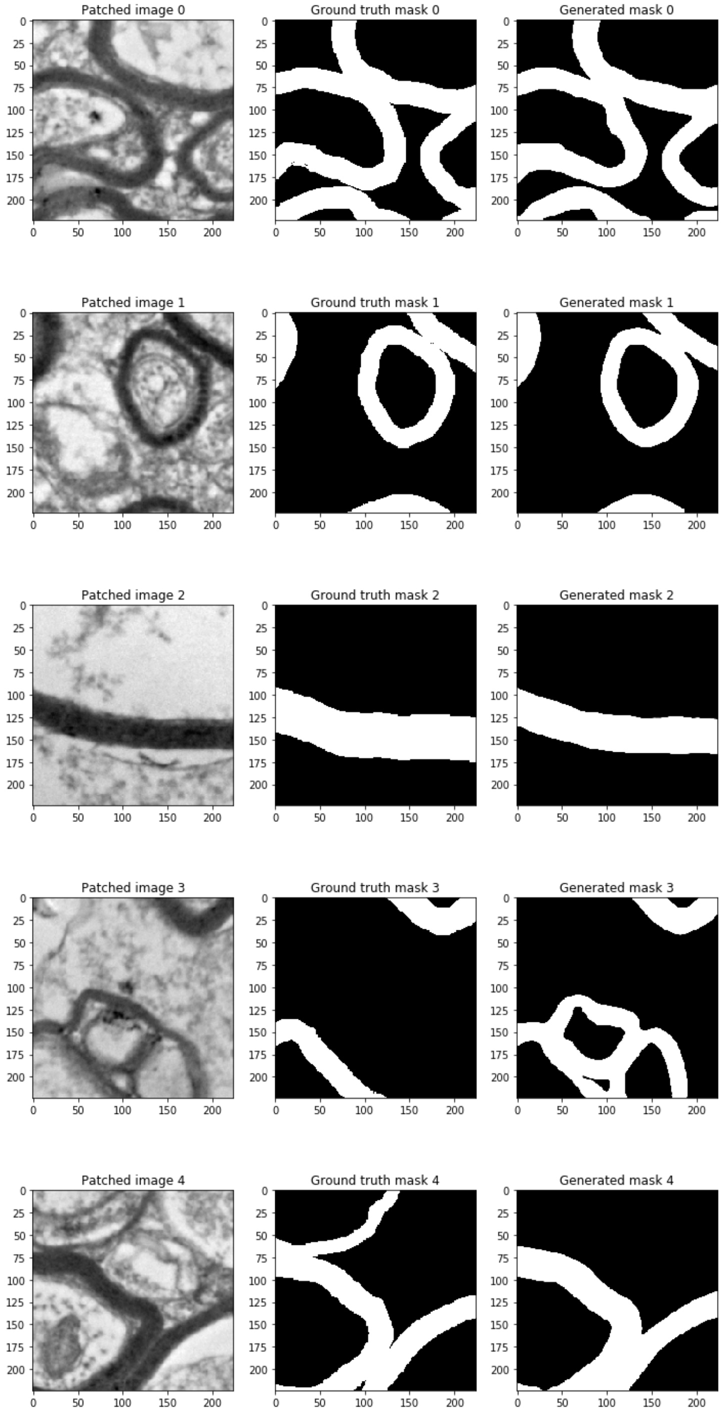
As you can simply see from the image analysis, the quality of the masks predicted using deep learning approach is significantly better than benchmark model: they have nice smooth boarders, do not include artefacts from dark background, do not fail in case of overlapping a few myelin layers. Dice coefficient is also significantly higher: 0.8397 vs 0.687.

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

The overall results of predicted segmentation masks is good. Also the model generalizes well and is not overfitted.



For example, if you look at the patched image 3 from the image above, you can notice that there is a mistake in ground truth label and two myelin sheath at the bottom are not presented in this particular image, but the model does see them and segments them properly.

Reflection

The final goal of this project was to solve binary segmentation challenge of electron microscopy images in order to segment myelin from the background. I used deep learning approach with U-net--like encoder-decoder architecture, which is extremely successful neural net architecture for segmentation tasks, especially in biomedical applications. After tuning the hyperparameters and adding pre-training on simpler synthetic data, the model achieved really good quality with dice coefficient on test set equals to 0.8397. From image analysis one can see that the quality of masks is good, they correspond well to ground truth labels, almost do not have any artifacts, even in case of challenging images, where dark similar to myelin shape different impurities were presented.

Improvement

One of the significant limitations to achieve a higher quality of the model is lack of labeled data. Typically for segmentation tasks one need starting from 10k images or so. But getting biological data is always expensive and time-consuming problem. Another issue I had, that manually (using semi-automated methods) labeled masks are far away from perfection, so it's hard to expect from the model to have really high quality predicted masks if it was trained on not perfect labels. One of the main improvements besides getting more data and correcting labels, would be to use data augumentation such as affine transformations, zooming, flipping and shifting, as there's a significant variation in myelin shape and size, it would be helpful for the neural net robustness if it trains on more diverse training data. Also if I had more computational power I would definitely try to increase the number of convolutional layers in the architecture to get finer features.

References

1. Morell P, Quarles R, Norton W. Formation, structure, and biochemistry of myelin. In: 4th ed. New York: Raven Press Ltd; 1989. pp. 109–136.
2. Trapp BD, Kidd G. Structure of the myelinated axon. In: London: Elsevier Academic Press; 2004. pp. 3–27.
3. Van De Graff K. Nervous tissue and the central nervous system. In: New York: McGraw-Hill; 2002. p. 351.
4. Trapp BD, Ransohoff R, Rudick R. Axonal pathology in multiple sclerosis: relationship to neurologic disability. Current opinion in neurology 1999;12:295–302.
5. Simao G, Raybaud C, Chuang S, Go C, Snead O, Widjaja E. Diffusion Tensor Imaging of Commissural and Projection White Matter in Tuberous Sclerosis Complex and Correlation with Tuber Load. American Journal of Neuroradiology 2010;31:1273–77.
6. Jelescu I, Zurek M, Winters K, et al. In vivo quantification of demyelination and recovery using compartment-specific diffusion MRI metrics validated by electron microscopy. Neuroimage 2016;132:104–14.
7. Stikov N, Campbell JS, Stroh T, Lavelée M, Frey S, Novek J, Nuara S, Ho M-K, Bedell BJ, Dougherty RF. In vivo histology of the myelin g-ratio with magnetic resonance imaging. NeuroImage 2015;118:397–405.
8. Dula AN, Gochberg DF, Valentine HL, Valentine WM, Does MD. Multiexponential T2, magnetization transfer, and quantitative histology in white matter tracts of rat spinal cord. Magnetic Resonance in Medicine 2010;63:902–9. doi: 10.1002/mrm.22267.
9. West K, Kelm N, Carson R, Does M. A revised model for estimating g-ratio from MRI. NeuroImage 2016;125:1155–8.
10. Taha AA, Hanbury A. Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. BMC Medical Imaging. 2015
11. Dice LR. Measures of the amount of ecologic association between species. Ecology. 1945;26(3):297–302. doi: 10.2307/19324094
12. Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation
13. University of Freiburg. U-Net: Convolutional Networks for Biomedical Image Segmentation
14. Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66. doi:10.1109/TSMC.1979.4310076.
15. [OpenCV Image Thresholding] (https://docs.opencv.org/3.3.0/d7/d4d/tutorial_py_thresholding.html)
16. [Pre-training on synthetic data for segmentation tasks] (https://github.com/ZFTurbo/ZF_UNET_224_Pretrained_Model)