Artificial neural network filters for enhancing 3D optical microscopy images of neurites

We applied artificial neural networks on 3D neuron image processing, in order to assist neuron tracing algorithms. Python codes for 3 artificial neural networks are provided here along with 3 trained networks.

Overview

Data

The data folder contains 2 folders: L1 and training_result. L1 folder contains 6 training images and their labels in tiff format obtained from Neocortical Layer 1 Axons in the Diadem Challenge. Training_result is the default path to save training results, we also provide 3 trained networks inside this folder.

Codes

The python folder contains 3 training codes, 3 test codes, and a code (common_functions.py) that contains functions used by different training or testing codes.

Models

Three network architectures (Figure 1) are available: Shallow feedforward network (CC), Multilayer feedforward network (CTC) and 3D UNet (3DUNet). For each network there is a code _train for training the model and a code _test for loading a trained model. A trained network is provided for each architectures.

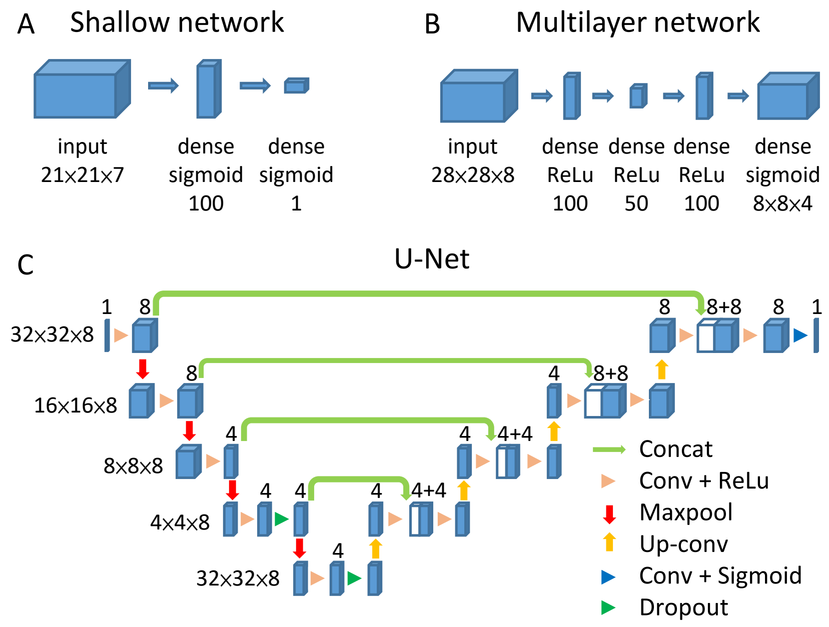

Figure 1: Architectures of the 3D NN filters used in this study. A. (CC) Shallow feedforward network with dense connectivity and sigmoidal neurons. Blue boxes represent neuron layers with neuron numbers shown. The network receives a 21×21×7 voxel sub-image as an input and generates a scalar output. B. (CTC) Multilayer feedforward network with dense connectivity and ReLu neurons. This network has a bottleneck. It receives a 28×28×8 voxel sub-image as an input and generates 8×8×4 voxel output. C. (3DUNet) Blue boxes represent feature maps with the number of channels denoted above each box. The network receives a 32×32×8 voxel input and generates an output of the same size.
mask pixels are in [0, 1] range.

How to use

Dependencies

This tutorial depends on the following libraries:

- Tensorflow
- Keras >= 1.0

Also, this code is compatible with Python versions 3.6.

Inputs and outputs

The input and label images must be in 8-bit tiff format. The output is a numpy array with the same size as the input image, output range from 0 to 1.

Training codes

For training, user should specify the path of images and labels used for training and validating, as well as learning rate, dropout ratio, batch size, maximum number of training steps, and plotting step. Input size, output sizes and label size can also be adjusted. For CC and CTC the training loss is visualized with Tensorboard, for 3DUNet, its visualized with pyplot.

Testing codes

For testing, user should specify the path of which model to restore and using which image for testing. The original image, label and the enhanced image will be showed for testing.

Trained networks

3 trained networks (inside CC_0, CTC_0, and 3DUNet_0 folders) are provided. User can use testing codes to restore the model within the folders. The var.npz file contains parameters of the models.