AOVesselCNN Guide

Gwen Musial and Hope Queener

# Overview

The AOVesselCNN Convolutional Neural Network (CNN) was designed for segmenting capillaries from perfusion images acquired by the Adaptive Optics Scanning Laser Ophthalmoscope (AOSLO) in the laboratory of Jason Porter at the University of Houston, Houston, Texas, USA. The AOSLO videos were acquired near the optic nerve to image the retinal parapapillary capillaries (RPCs). The CNN design was based on the Retina-unet CNN2,3 (github.com/orobix/retina-unet). The code and the retina-unet CNN was adapted in a number of ways as described in Musial et. al. (in press).

The AOVesselCNN project was tested using the following software prerequisites. The original retina-unet code was developed using Python 2.7. The use of Python 3.x was explored for AOVesselCNN, however, not all required features of Tensorflow/Keras were available at that time.

The table below lists two collections of versions that were tested at two time points. For the earlier date, keras and tensorflow were imported separately. For the later date, keras was imported from tensorflow, therefore the syntax of the import statements was adapted for compatibility. The code that accompanies this guide is the 2019 code. (Future revisions will support the 2020 code.)

|  |  |  |  |
| --- | --- | --- | --- |
| **TABLE 1** | | | |
| **Package** | **Version (2019)** | **Version**  **(2020)** | **URL of package** |
| Keras | 2.2.4 | 2.2.4-tf | <https://keras.io/> |
| Tensorflow (backend for Keras) |  | 2.0.0 | <https://github.com/tensorflow/tensorflow> |
| CUDA | 9.1.85 | 9.1.85 | <https://developer.nvidia.com/cuda-91-download-archive> |
| h5py | 2.8.0 | 2.8.0 | <https://pypi.org/project/h5py> |
| Python | 2.7.? | 2.7.17 | python.org |
| numpy | 1.15.4 | 1.16.5 | Numpy.org |
| scipy | 1.1.0 | 1.2.1 | SciPy.org |
| PIL Image | 5.4.1 | 6.2.1 | Pillow.readthedocs.io; Pillow fork of PIL |
| cv2 (OpenCV) | 4.0.1 | 3.4.2 |  |
| skimage | 0.14.1 | 0.14.2 | <https://scikit-image.org/docs/0.14.x/install.html> |
| sklearn | 0.20.1 | 0.20.3 | <https://github.com/scikit-learn/scikit-learn/releases> |

## High Performance Computing Resources

The AOVesselCNN modules were developed to run on the high-performance computing resources provided to University of Houston (UH) users. Users at other institutions will need to coordinate with the personnel at their respective facilities to configure Keras, Tensorflow, CUDA and Python software on suitable high-performance systems to support this CNN.

Without high-performance resources, training the CNN may be computationally unwieldy. For example, using the University of Houston resources in 2019, training and validation of the CNN using 185 images required 3 hours on an NVIDIA Volta GPU. Segmenting the 14 images in the testing set took about 2.5 minutes. All images were 768 x 768 pixels in dimension.

# Image Preparation

Preparing the AOSLO Perfusion Montages

The perfusion images are prepared in the same manner for training, validation or segmenting new data. The steps are described briefly here but are not provided in the AOVesselCNN package. The specific configuration of the AOSLO system will influence the best way to produce a perfusion image. The perfusion image is generally produced using an imaging channel that captures scattered (rather than confocal) light from the AOSLO system. Offset pinhole and split-detector imaging systems capture such light1, 5. In the Porter Lab system, the two AOSLO split detection channels are combined by dividing the difference between the channels by the sum of both channels pixel for pixel in each frame. If the sum is zero, the pixel in the combined video is set to zero. Any sinusoidal warp from the scanning system is removed based on the system calibration. Images are stabilized to compensate for eye motion. To produce the perfusion image from these stabilized videos, a procedure similar to Chui et. al.1 was used. The entire video (about 6 to 10 seconds in length at 25 Hz) is normalized to the maximum pixel value. Next, it is median filtered (with a 3 x 3-pixel kernel) to reduce noise. Breaking the video into 25-frame intervals, the standard-error for each interval is computed. Limiting the duration of the interval reduces the impact of slow changes in tissue reflectance. The resulting standard-error images for each interval are median filtered (again using a 3 x 3-pixel kernel) and averaged. The averaged result is normalized and histogram-stretched to produce the perfusion image. The stretched histogram results in the lower and upper 1% of the original histogram to be set to 0 and 255, respectively.

These small perfusion image patches are stitched together manually within a larger rectangular canvas within Adobe Photoshop to produce a perfusion montage. If montages of the same retina were collected at different time points, these montages were registered to each other using canvases of the same size. In practice, these are registered manually to an *en-face* image of the retina from a spectral-domain optical coherence tomography instrument. (Spectralis HRA+OCT, Heidelberg Engineering, Heidelberg, Germany)

## Preparing Ground Truth Images for Training and Validation

Ground truth images are required for training and validating the CNN. A pseudo-colored image is produced that represents the 4 classes: manually identified capillaries, manually traced large vessels, AOSLO image background and uniform canvas (Photoshop canvas background around the stitched AOSLO regions). In Photoshop, the capillary class was manually traced using a Wacom tablet and a single-pixel line to approximately bisect each capillary. These single pixel lines were dilated using MATLAB with a disk structuring element of 5 pixels (the average capillary size in this data set) to define pixels belonging to the capillary class. The large-vessel class was traced manually using a pencil width of 20 pixels or larger. Any other software that allows the user to create a 4-color image (of unsigned 8-bit pixels) for the 4 classes would be suitable for producing training montages, so long as the capillary class is on the order of 5 pixels wide and the large vessels are 20 pixels or more in width. The 4 class identifiers are 0 for background (non-vessel AO image), 1 for capillary, 2 for canvas and 3 for large vessels. Initial versions of the CNN only considered background and capillary. The canvas class was introduced to eliminate from the training any patches of the montage that were more than 50% canvas. The large vessel class was introduced later for the improvement of using 4-class training.

## Preparing Uniform Tiles of Classified Data (used for training and validation)

Once the 4-class images are defined the result is chopped up into tiles of dimension 768 x 768 pixels, for use by the CNN. The code produces tiles for all four classes as well as for the combined class. The output files are appended with row and column identifiers for the position of the tile within the montage. Tiles that are more than 50% canvas border are eliminated from the training.

Use **reshapeAOImages.m**  (AOVesselCNN\Preparation folder) to read in corresponding images of each class and generate the 768 x 768 tiles. The files are named in a corresponding way as: [*image-base-name*]\_[*tile-row*]\_[*tile-column*]. For example, if the original image was “ID123.tif”, the first tile image is named “ID123\_01\_01.tile”. For the canvas class, the user selects the RGB image where red pixels represent the canvas. The user selects the corresponding manually-traced capillary image and the large-vessel image and these are likewise chopped up into tile images. The outputs include the grayscale images, canvas images, capillary-only images, large-vessel only images and 4-class combination images.

Once the images are prepared, they need to be organized into the expected folder locations.

Project Folder (where code resides)

DataSet Folders\_Images

TrainingDataSet

Images

combined

TestingDataSet

Images

combined

SubjectXXX

Images

DataSet Folders\_HDF5

This folder contains the HDF5 formatted data that is saved and loaded by the python scripts. Users do not need to access this folder manually.

Training and validation datasets currently include AOSLO perfusion images acquired from human and non-human primate (NHP) and experimental glaucoma NHP. They are provided in the “Data” folder of the GitHub repository as several .zip files.

Training grayscale images are provided in three .zip files entitled Images01to17.zip, Images18to25.zip, Images28to31.zip. These must be unzipped and placed within the top-level “project” folder in one folder named “.\DataSet Folders\_Images\TrainingDataSet\Images\”. These files have a naming convention of a ##\_manual1\_##\_##, where ## is a zero padded value from 00 to 99. The first one represents an image index number which corresponds to the same subject. The second ## is the tile row index and the third ## is the tile column index. These indices are used to register the 768 x 768 grayscale and ground-truth images to one another and to rebuild these images into the original montage images.

The ground truth images are compressed in the TrainingGroundTruth.zip folder. These images contain values between 0 and 3 so that all truth images can be compressed in a single zip file. Once uncompressed, these images are expected to reside in the folder “.\DataSet Folders\_Images\TrainingDataSet\Images\combined”. The name “combined” reflects that four ground-truth classes are combined in a single image.

The validation set is smaller than the training set, so the grayscale and ground truth images are in respective .zip files named ValidationImages.zip and ValidationGroundTruth.zip. The uncompressed images are expected to be located in “.\DataSet Folders\_Images\TestingDataSet\Images\” and in “.\DataSet Folders\_Images\TestingDataSet\combined\”, respectively.

Grayscale images to be segmented can be organized in subfolders that are named in the config\_test\_only.py. The example above is “\SubjectXXX\Images”.

# CNN Code Preparation

The following modules were designed to work with the high-performance resources at UH. Note that the text (\*.txt) files and BASH scripts (\*.sh) contain specific references to the particular computing environment. The BASH scripts (\*.sh) are used to submit jobs to the specific high-performance system. The text files provide parameters and file paths required by the python code for the specific job. It is expected that these modules will be modified or replaced for the specific computing environment.

This example BASH script uses [square brackets in gray] to indicate user-defined information.

#!/bin/bash

#SBATCH -J [job name] # job name

#SBATCH –o [output-file-name].o%j # output and error file name (%j expands to jobID)

#SBATCH -N 1 -n 2 #Nodes to call and uses

#SBATCH -p gpu #can call different GPUs based on what is available on the cluster

#SBATCH --gres=gpu:1

#SBATCH -A [allocation name] #allocation

#SBATCH --mail-user=[email-address] #who to email

#SBATCH --mail-type=all # email me when the job starts and ends

#SBATCH --mem=100GB #How much memory to allocate to job

#SBATCH -t 24:00:00 #How long to let job run for (max time is 24hrs)

module load Anaconda2/python-2.7 #Loads the python module on the cluster

module add CUDA/9.1.85 #Loads the CUDA resources on the cluster

python wrapperPrepData.py 'config\_G\_Test.txt'

masterScript\_TrainandTest\_All.py 'config\_G\_Test.txt'

## Python Modules

wrapperPrepData.py

This script converts the images from their original file format into a single array of images in HDF5 file format. This script also runs checks to make sure that the images are the same size and are in grayscale format.

masterScript\_TrainandTest\_All.py

This script trains the CNN on the training dataset provided with the parameters outlined in the .txt file. It then does the validation of the trained network using the testing dataset provided.

test\_Unknown\_Only.py

This script uses weights from a previous training run to generate the CNN segmentation on AOSLO perfusion images. No ground-truth images are provided for these images.

|  |  |
| --- | --- |
| **TABLE 2** | |
| **CNN Training, Validation, and Segmentation Code** | |
| Folder: AO-VesselCNN/Project | |
|  | |
| **Module Name** | **Purpose** |
| config\_train.txt | This example text file provides input to train.sh and the input arguments to masterScript\_TrainandTest\_All.py. It specifies all of the data paths, training parameters, and testing parameters. |
| config\_test\_only.txt | This text file provides input to testUnknown.sh and input arguments to test\_Unknown\_Only.py. It contains all of the data paths and testing parameters. |
| testUnknown.sh | This BASH shell script defines the job that is managed by the cluster resources when applying the trained CNN to new images. It sets the job name, the job resources, the required import modules, requests email notifications and runs the python scripts. |
| train.sh | This BASH shell script defines the job that is managed by the cluster resources when training the CNN. It sets the job name, the job resources, the required import modules, requests email notifications and runs the python scripts. |
| wrapperPrepData.py | Python script that prepares the data from its original format into the hdf5 format. Performs checks to make sure the images are of the correct type. |
| masterScript\_TrainandTest\_All.py | Performs the training of the CNN using the parameters set in the config\_train.txt file. Also performs the validation and testing metrics of accuracy etc. |
| RPC\_CNN\_Functions.py | Python script that contains all of the helper functions called by the other three python scripts. |
| test\_Unknown\_Only.py | Uses weights from a previously trained network to segment unknown images using the parameters set in the config\_test\_only.txt file. |
| version\_print.py | Python script that prints version numbers of dependencies and imports the required modules to confirm dependencies are available on the computing resources. |
| VersionP.sh | BASH shell script to load python and CUDA, display the CUDA version and run version\_print.py. |

# Example Workflow

The workflow below describes how the remote high-performance resources were accessed at the University of Houston. They are provided as a reference for UH users and as an example to users outside of UH.

1. Transfer data to the high performance server such that the data is arranged in folders with the organization described in the section: Image Preparation.
2. Edit the configuration files (\*.txt) to match the folder names of your configuration.
3. The BASH shell script is specific to the high-performance resources and the user. The user defines the name of the submitted job and the name of the output file prefix. It is recommended that you keep the name “train” for a training run and the name “test” for a testing run.
4. If you are editing BASH scripts (\*.sh) and configuration (\*.txt) files on a local machine, transfer the updated files to the remote high-performance system (for example, by using WinSCP).
5. Arrange the code (configuration files (\*.txt), BASH scripts (\*.sh) and Python (\*.py) modules in the top-level folder (described as the “project” folder). Data files are in subfolders.
6. Use a terminal program (such as MobaXTerm and ssh) to log on to the high-performance resources. Change to the “project” directory that contains the CNNCode scripts.
7. If you are editing BASH scripts (\*.sh) and configuration (\*.txt) files on the high-performance machine, make changes prior to submitting jobs.
8. Run the command sbatch train.sh or sbatch.test\_Unknown\_only.sh to submit the job to the resource manager.
   1. To train the CNN: sbatch train.sh
   2. To apply the CNN: sbatch test\_Unknown\_only.sh
9. Wait for the job to be submitted, run and complete without errors.
10. Inspect the results and make any adjustments needed.
11. Transfer the segmentation results from the specified output folder (i.e. the experiment\_name in the configuration file) to the local machine for post-processing.

## Command Quick-Reference

|  |  |
| --- | --- |
| sbatch [bash-script].sh | submit BASH script to resource manager |
| cat [output-file].o[job number] | display output file |
| squeue –u [username] | see what node your job is on and how long your job has been running (or waiting) |
| squeue | See which other users are accessing the resources |
| scancel [job number] | Terminate job |

# Post Processing Modules

Because the original work involved longitudinal studies where the images were registered and of the same size, several test images were submitted as one large image to the CNN code. The CNN would then output all of the segmented images in a single .png image. The MATLAB modules described in table 3 can be used to cut up this combined .png image.

chopCNNOutput\_768.m

The CNN outputs the result as a single concatenated image. This module breaks the CNN .png output into regular sized images of the expected size of 768x768 (used to break up validation results).   
Input: Greyscale or Binary CNN .png File

Output: Greyscale or Binary individual image files

chopCNNOutput\_VariableSize.m

The CNN outputs the result as a single concatenated image.For longitudinal studies, montages are registered within a canvas of the same size for all time points. The resulting probability maps are the row-concatenated images of the same height. The user selects one of the original images as a template for the desired output size.

Inputs: Image of the same size that you want outputs chopped up into and the Greyscale or Binary CNN .png File

Output: Greyscale or Binary individual image files

The following MATLAB modules were used to calculated metrics of the performance of the CNN compared with the ground truth 4-class images.

dilatedDiceCoefficient\_Acc\_Sensitivity\_Density\_MSL.m

This module calculates the density, accuracy, sensitivity, Dice coefficient, dilated Dice coefficient and the mean-segment-length based on two input images.

Inputs: the binary prediction image, the binary corresponding ground truth image and the binary image that marks the canvas border in both images. The canvas border is eliminated from the density calculation.

Outputs: Excel file in the same folder as the first selected image (i.e. the prediction image) that lists all of the metrics for the two files.

dilatedDiceCoefficient\_Acc\_Sensitivity\_Density\_MSL\_Batch.m

This module calculates the density, accuracy, sensitivity, Dice coefficient, dilated Dice coefficient and the mean-segment-length based on two selected folders with images of corresponding names.

# References

1. Chui, T. Y. P., VanNasdale, D. A., & Burns, S. (2012) The use of forward scatter to improve retinal vascular imaging with an adaptive optics scanning laser ophthalmoscope. *Biomedical Optics Express,* 3 (10). DOI:10.1364/FIO.2012.FM4A.4
2. Chui, T. Y.P., Dubow, M., Pinhas, A., Shah N., Gan, A., Weitz, R., Sulai, Y. N., Dubra, A., & Rosen, R. B. (2014) Comparison of adaptive optics scanning light ophthalmoscopic fluorescein angiography and offset pinhole imaging. *Biomedical Optics Express* 5(4). DOI:10.1364/BOE.5.001173
3. Musial, G., Queener, H.M., Adhikari, S., Mirhajianmoghadam, H., Schill, A.W., Patel, N.B., Porter, J., (2020) Automatic segmentation of retinal capillaries in adaptive optics scanning laser ophthalmoscope perfusion images using a convolutional neural network, *pending publication*
4. Ronneberger O., Fischer P., Brox T. (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351. Springer, Cham. DOI: 10.1007/978-3-319-24574-4\_28
5. Scoles, D., Sulai, Y.N., Langlo C.S., Fishman, G.A., Curcio, C.A., Carroll, J. & Dubra, A. (2014) A. In vivo imaging of human cone photoreceptor inner segments. *Invest Ophthalmol Vis Sci.* 55(7): 4244-4251 DOI:10.1167/iovs.14-14542
6. Xiancheng, W. Wei, L., Bingyi, M., He, J., Jian, Z, Xu, W., Ji Z., Hong, G. (2018) Retina Blood Vessel Segmentation Using A U-Net Based Convolutional Neural Network. Int. Conf. Data Sci. 00, 1–11.