Automatic Left ventricle volume calculation in cardiac MRI using Convolutional Neural Network

Name: Tencia Lee & Qi Liu

Location: Los Angeles, CA & New York, NY, USA

Email: tencia@gmail.com, liu.qi.alex@gmail.com

Date: 03/16/2016

Competition: Second Annual Data Science Bowl, 1st place solution

1. Background on you/your team

Tencia: I graduated from Caltech in 2009 with B.Sc in applied mathematics and economics. I then worked in quantitative finance, first as a researcher and then as a portfolio manager at a Los Angeles-based hedge fund, for about six years. I recently transitioned to a robotics startup as a research engineer. I became interested in deep learning almost a year ago and have been studying and learning about it since then. I decided to enter this competition as I thought it would be a great educational experience and a chance to apply the methods I have been learning. I spent approximately 160 hours working on this competition.

Qi: I got my Ph.D in theoretical physics (about Lattice Quantum chromodynamics & Charge-Parity violation of the Weak interaction) from Columbia University 4 years ago and after that I worked as a quantitative trader in a hedge fund. I'm taking a long vacation recently so I have a lot of time working on the Kaggle competitions, this one has a lot of money prize and the problem is complicated (being complicated means that the signal to noise ratio is higher on the LB) and interesting so I decide to give it a shot

We originally decided to form a team because Qi was working with a dynamic programming segmentation method and Tencia was working with neural networks, and we thought they would be complementary to each other. However, after a few weeks it became clear that the neural network approach was capable of higher precision, so we dropped the dynamic programming segmentation method completely to simplify our work and code.

2. Summary

We used as our primary model an ensemble of fully convolutional neural networks which calculated areas from DICOM images, and then used these areas to calculate heart volume at different times in the heartbeat cycle. As part of our ensemble, we also included a fully convolutional segmentation network for 4-chamber view DICOMs, a single-slice model, and an age-sex model.

Our time was spent 10% data cleaning methodology, 30% neural network design and experimentation, 30% identifying model weaknesses, 20% calculation of volume from segmentation, and 10% manually labeling data.

One of the trickiest aspects of this dataset was the number of cases with missing or poor-quality data. We developed several heuristics to evaluate segmentation performance and detect outliers, and to decide for each case which models to include.

We found that one of the most difficult problems with our approach was that the segmentation network would often fail to recognize a ventricle. Image normalization was essential to remediate this problem; however, we also used pseudo-active learning to select examples for manual segmentation. In total we segmented 130 SAX-view DICOM images by hand.

Our approach was guided by the view that since we are approximating a derived number, we should find the inputs and then calculate the end result in the same way the ground truths had been calculated. For this reason we steered away from an end-to-end prediction algorithm for

our primary model, instead opting to approximate the segmentation for each slice as closely as possible to how a doctor would, and then calculate the heart volume using those areas.

For neural network training, we used mini-batch gradient descent with the Adam optimizer (4), with use of convolutions and batch normalization in our networks. For ensembling, we optimized a linear combination of different models. Training and prediction were done using two GPUs, and all code was written in Python with the use of Theano, Lasagne, and cuDNN for neural networks.

3. Features Selection / Engineering

Fully Convolutional Neural Network for segmenting the left ventricle

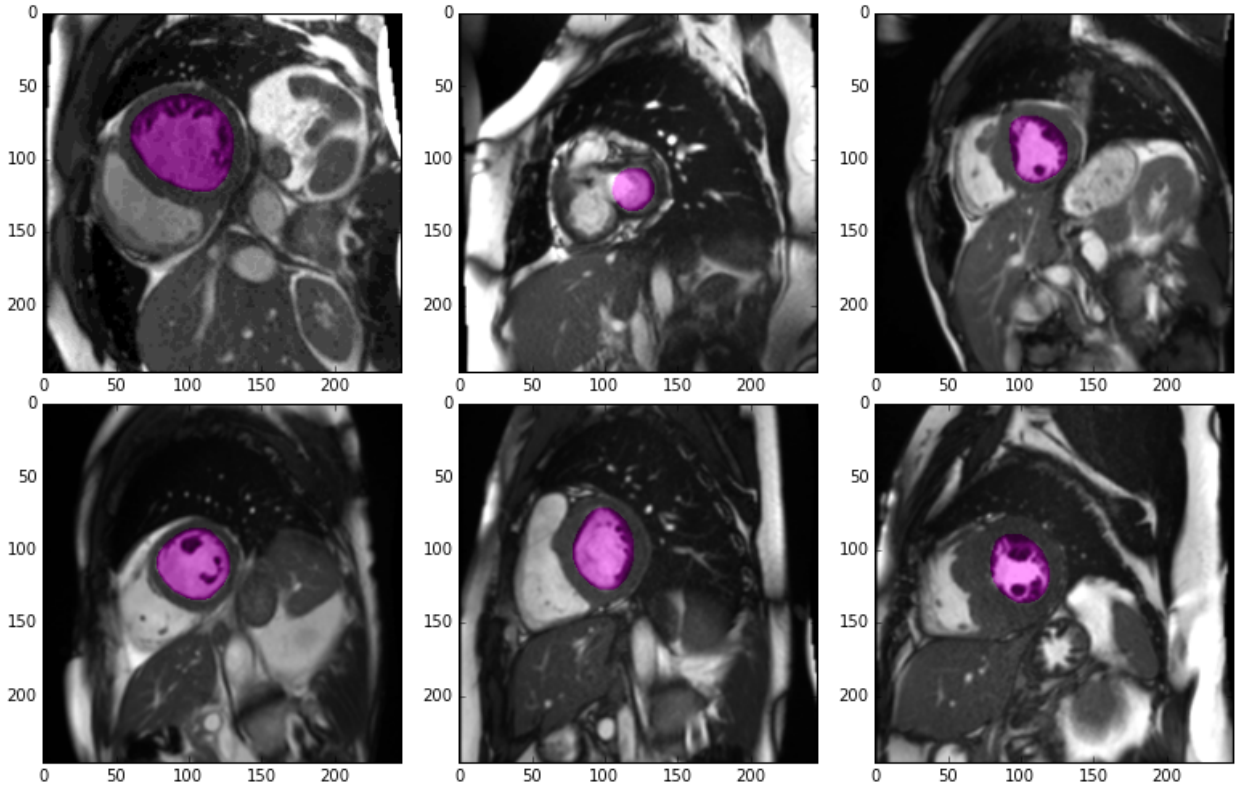
Our main model used several Fully Convolutional Neural Networks (CNN) to segment the left ventricle for each MRI image. The output of each network was an image of the same size as the input image, with pixels having values in the range [0,1]. Each pixel's output represents the probability that the corresponding pixel in the input image is part of the left ventricle.

The architecture of the networks from the CNN_B folder are as follows, where:

- b = batch size
- The four dimensions of the tensor represent batch size, channels or number of filters, and the two dimensions of the image, respectively.
- Conv = convolution with a stack of square kernels of (# filters) x (filter size) x (filter size)
- BN = batch normalization as in reference (2), implemented in Lasagne, in reference (3)
- ReLU(x) = max(0, x)
- Sigmoid(x) = 1 / (1 + exp(-x))
- Convolutional layers with valid padding output a value only for filter positions where every value in the filter has a corresponding value in the input tensor (shrinks the tensor).
- Convolutional layers with full padding output a value for all filter positions for which at least one value in the filter has a corresponding value in the input tensor (expands the tensor), with all other values zero-padded.
- MaxPool and Upscale are done across the last two dimensions of the tensor.

Layer Op / Type	# Filters / Pool / Upscale Factor	Filter Size	Padding	Output Shape
Input				(b, 1, 246, 246)
Conv + BN + ReLU	8	7	valid	(b, 8, 240, 240)
Conv + BN + ReLU	16	3	valid	(b, 16, 238, 238)
MaxPool	2			(b, 16, 119, 119)
Conv + BN + ReLU	32	3	valid	(b, 32, 117, 117)
MaxPool	2			(b, 32, 58, 58)
Conv + BN + ReLU	64	3	valid	(b, 64, 56, 56)
MaxPool	2			(b, 64, 28, 28)
Conv + BN + ReLU	64	3	valid	(b, 64, 26, 26)
Conv + BN + ReLU	64	3	full	(b, 64, 28, 28)
Upscale	2			(b, 64, 56, 56)
Conv + BN + ReLU	64	3	full	(b, 64, 58, 58)
Upscale	2			(b, 64, 116, 116)
Conv + BN + ReLU	32	7	full	(b, 32, 122, 122)
Upscale	2			(b, 32, 244, 244)
Conv + BN + ReLU	16	3	full	(b, 16, 246, 246)
Conv + BN + ReLU	8	7	valid	(b, 8, 240, 240)
Conv + sigmoid	1	7	full	(b, 1, 246, 246)

CNN_A networks are similar to the above with minor changes in number of layers and filter sizes, so for brevity we omit their exact architecture here. At test time, CNN_A evaluated an image cropped from approximate center of left ventricle, while CNN_B evaluated the full image resized to match the input size.



Example outputs from the segmentation network, with ventricle highlighted in purple.

Volume calculation from segmented images

With the segmented result from the CNN, the area $A_{i,t}$ for each slice i at time t can be calculated. We remove the slices that has area of 0. Then the volume at any time t is calculated as a sum of the pieces:

$$V_t = \sum_{i=2}^{N} (A_{i,t} + A_{i+1,t})/2 \times (z_{i+1} - z_i) + A_{1,t} \times h/2 + A_{N,t} \times h/2 \qquad (1)$$

Where z_i is the slice location, h is the slice thickness. We tried other methods to calculate the volume but this method seems to give us a slightly better result after adjustments for systematic error. It would be best if the organizer of this competition could release the actual formula so we don't need to guess one.

The systole and diastole volume is taken from the minimum of V_t and the maximum of it. We observed that for many cases, our result is larger than the "true" value while for many other cases they agree very well. We believe this is a type of systematic error that our method intends

to include all good looking slices while a doctor might decide to remove it for some reasons. So we did a correction with the formula:

$$V_{adj} = V_{CNN} - \alpha\sqrt{V_{CNN}} \qquad (2)$$

for systole and diastole volume separately. Here we have two parameters to be determined α_{sys} and α_{dias}. We also experimented with simple linear fitting but it did not perform as well.

We assume that the measured volume obeys a normal distribution with mean V_{adj} and standard deviation S_v, the normal CDF is then generated and the Continuous Ranked Probability Score (CRPS) is calculated. We fit the standard deviation S_v to be linear with V_{adj},

$$S_v = \beta V_{adj} + S_0 \qquad (3)$$

Here we have 4 parameters to be determined, 2 for Systole and 2 for Diastole volume standard deviation. These 4 parameters together with α_{sys} and α_{dias} are determined simultaneously by optimizing the final CRPS score function for the train and validation set, and then the fitted coefficients are used to calculate the volume and std of volume for test cases.

There are few cases that our method might fail. The CNN might fail to detect the Left ventricle or some cases have missing data (e.g., some dataset only have 3 SAX slices). So we developed some other simpler models to avoid getting something completely wrong.

Other models for failure cases

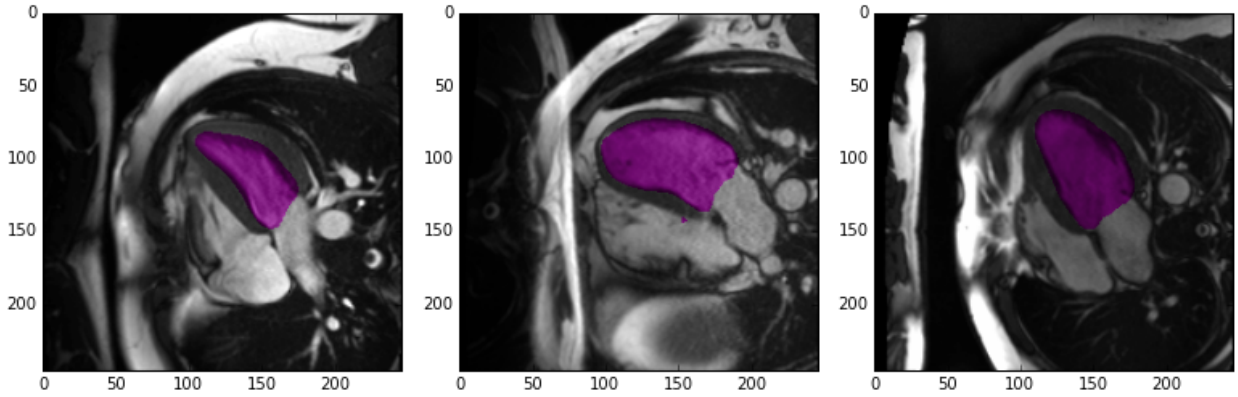
One-Slice Model: We took the 80th percentile of the area vector, A_{dias} and A_{sys} for diastole and systole respectively, and used them to fit the volumes. The reason to use 80 percentile instead of the maximum is to avoid those wrong/extreme contour readings from CNN. Because the length of the heart is more or less proportional to the diameter of the largest cross sectional area, so we fit

$$V = V_0 + \alpha \cdot A \cdot \sqrt{A} \qquad (4)$$

for diastole and systole volume separately. This model achieves a score of 0.015 in train set.

4-chamber Model: We hand-segmented 736 4-chamber view images and used them to train a fully convolutional segmentation network in identical manner to the CNN_B main model. We trained five networks, each with 20% training data held out, and used the average output of

these five as the segmentation output. The output of this network was a segmentation mask indicating whether each pixel was part of the left ventricle, from the 4-chamber view.



Examples of segmentation output of the 4-chamber view

From here we used PCA to find the main axis of the ventricle, then calculated a disk at each pixel along the main ventricle and summed the disks to arrive at a volume. This volume was then adjusted using linear fit, and standard deviation for CRPS curve was calculated as in (2).

$$V_{final} = \beta V_{seg} + V_0 \qquad (5)$$

This model achieves a score of 0.017.

Age-sex Model: It is clear that the size of the heart grows with age before it comes to more or less fixed size. And it also significantly depend on the gender of the patient. So we fit some linear models using age and sex. Our method closely follows reference (1) that was posted in the forum. This model scores 0.037 in train set.

We took a combination of the one-slice model and 4-chamber model as the first default model, and together they achieve a score of 0.0134 for the training dataset. When our original SAX view based model fails, it takes result from this default model. If it still fails then we took the result from the age-sex model which has no failure cases.

4. Training Method(s)

Training data

For training the segmentation networks, we used the Sunnybrook contour data in a similar manner as the Deep Learning Tutorial, as well as 130 hand-labeled SAX images selected from the training set.

Training procedure:

The neural networks were trained to minimize a modified version of the Sørenson-Dice Index:

$$Loss = \; - \; (2 \times \; \Sigma_{i,j} \, pred_{ij} \, target_{ij} \; + s \;) \; / \; (\; \Sigma_{i,j} \; (\; pred_{ij} \; + target_{ij} \;) + \; s) \quad (6)$$

In equation (5), *pred* for each pixel was the output of the network after the sigmoid nonlinearity, with values in the range [0,1], and *target* was whether that pixel was part of the left ventricle in the ground truth segmentation of that image, with value 0 if not, and 1 if so. We found this objective function gave much better results than binary cross-entropy. The hyper-parameter *s* was used to give a nonzero loss even when image did not contain any part of the ventricle, and was usually set to 100.

Each neural network was trained for between 150 and 300 epochs using mini-batch gradient descent, with batch sizes either 8 or 16 images, using the Adam optimizer (4) with learning rate 3e-3. Hyperparameters were hand-tuned for effectiveness, but due to time constraints were not automatically optimized.

For some models in the ensemble, training was done with some part of the data held-out as a validation set. In these cases, the parameters that were saved as the result of that run were set that yielded the lowest validation loss. In others, training was done with all available data, and in these cases the parameters at the end were saved.

Data preprocessing and augmentation:

Image normalization was necessary as a pre-processing step to homogenize the dataset and facilitate neural network training. Two normalization methods were used:

- Mean and standard-deviation: the mean and standard deviation of the pixel values calculated over the middle 60% of the image, and the mean was then subtracted from the image and the result divided by the standard deviation. This was done because extreme values tended to occur at the edges, so this provided a more stable normalization.
- Percentile: the image intensity was rescaled, with a set percentile range being stretched to the entire range, as in Ref. (6).

Due to the small size of the segmentation training set, data augmentation during training was essential for successful results. Both CNN_A and CNN_B used random rotations and translations at training. CNN_B used a fixed normalization method for each training run, whereas CNN_A used randomized (on the percentile values) image normalization as an augmentation method. CNN_A also used random scaling of the images.

An approximate center and the bounding box of the Left ventricle can be estimated from the time variance of the images. We can also rotate the images based on the Dicom information to align all the cases roughly into the same orientation. To diversify our list of models, we only let CNN_A crop from the approximate center of the images and do the rotation alignment, while CNN_B uses the full image and only rotate those few cases that have approximate 90 degree rotations.

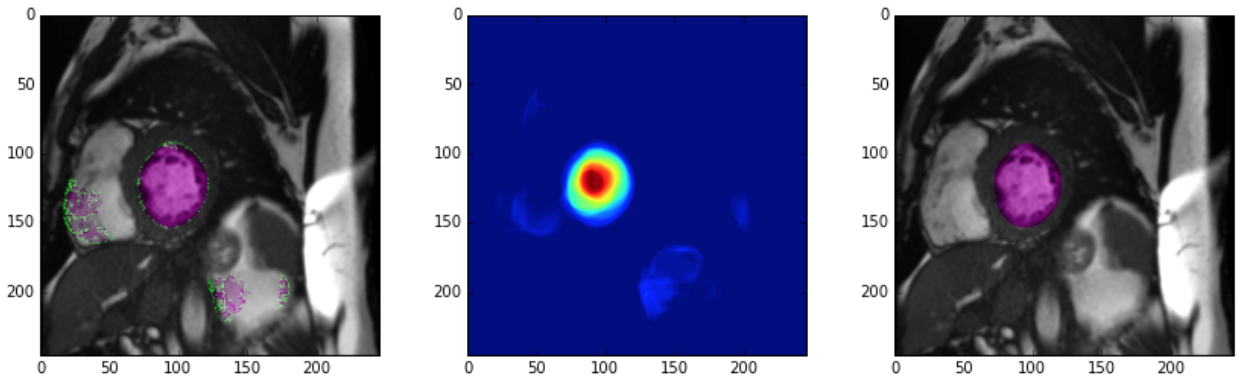
5. Interesting findings

Most of the most useful tricks we used related to detecting and fixing bad data. In many cases a patient's folder would contain DICOM images from multiple scans, only some of which belonged in the volume calculation, so we incorporated information from the DICOM metadata to filter which slices should be kept or dropped. Several fairly non-intuitive methods were used at this step, and this may have been one of the factors that separated us most effectively from the other competitors.

Post-processing results from the FCN output was also essential. We came up with two main heuristics to clean the segmentation results.

Firstly, we noticed that the network would often output false positives, in which it mistakenly thought pixels outside the ventricle were part of it. To eliminate these, for each case, we took the segmentation mask outputs for every image from that patient and calculated the average mask value for each pixel. This creates a “mean mask image” (center frame in below figure).

Then we fit a 2-D isotropic Gaussian kernel centered at the maximum value. For every contiguous patch returned by the segmenter, we calculated the average likelihood across all of its pixels under this Gaussian distribution, and eliminated every patch that had an average less than a set threshold. If more than one patch remained, we eliminated all but the one with the highest cumulative likelihood. In the below figure the extraneous patches visible in the first image have been removed using this method, yielding the clean mask in the third image.

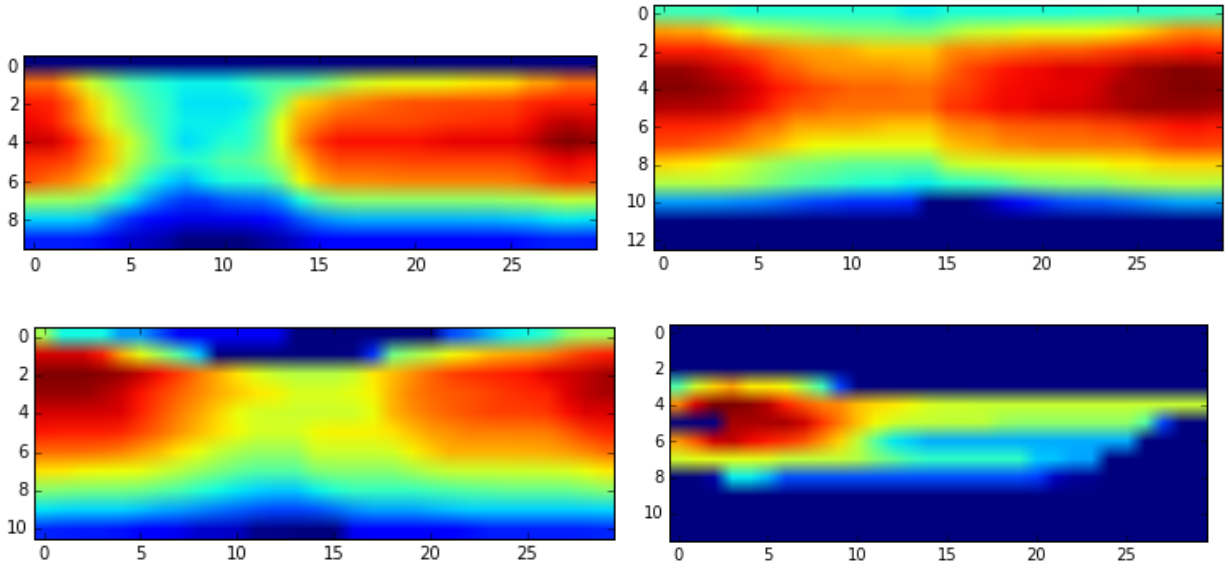


Secondly, once the SAX view images for a patient are segmented, we compose the areas found after applying the above filter into a matrix, with the x-axis corresponding to time and the y-axis to slice ordering. These matrices can then be compared to each other, to infer a “quality of read” measurement for each patient. To facilitate comparisons, the matrices were resized using bilinear sampling to 10 x 30 and normalized to values in [0,1], and for each of the 300 “pixels” we calculated a mean and standard deviation across all patients in the training set. Then each patient’s resized matrix can be given a log-likelihood score of:

$$LL = \Sigma_{t,s} log(\, p(a_{ts} \mid \mu_{ts}, \sigma_{ts}\,)) \quad (6)$$

When the map contains unusual values, LL as calculated above would be low, and this usually corresponded to the segmentation net being unable to segment some or all of the images

correctly. Therefore, in some of our models we used LL as a filter to determine when not to use the SAX model. In retrospect, this filter was fairly imprecise because the pattern of relative areas is not fixed across patients, and because LL considers each pixel separately without considering the joint distribution of pixels across a map. This filter could likely be improved by adding a term representing variation, because usually bad reads would also correspond to highly noisy or bumpy area values in the map.



Examples of a heatmaps with low (bottom right) and high (other positions) LL scores

One interesting aspect of this dataset is that it is much more highly structured even than our method reflects. The true shape of the ventricle is usually smooth, with no outlying values, and the area profile described by the SAX view and the 4-chamber view have to agree. Nevertheless, we did not use any collaborative method for detecting failures between these two views, or enforce any smoothness constraints. We did notice that in some cases outlier values would make their way into the volume calculation; however, we did not have time to build a more complete model that included all available information.

6. Simple Methods

We took an ensemble of 10 CNNs, but it only slightly improves the result of the average of two configurations under CNN_A/config_v3/6.py. The training time for a single CNN takes about 3

hours for input image size 256x256, and 2 hours for 196x196. The prediction time is about 20 seconds for each case.

To further simplify the model, we can directly take the volume calculated from eq. (1) that uses the segmentation output of the CNN. The adjustment equation (2) is used to match the ground truth provided by human annotation of the contours which has lots of uncertainty by itself from whether to include a slice or not at the base of the heart. As we examined many cases, it seems that the ground truth results are not fully consistent in keeping the end slices or not.

Appendix

A1. Model Execution Time

For the 8 versions of CNN (2 different architectures with different input image size 256 and 196, and 4 different kind of parameters) in CNN_A, each CNN takes about 3 hours to train on a GPU GTX 970. It takes about 10-20 seconds to forecast for each case (depending on how many SAX slices it has). For all 700 test cases, the prediction takes less than 3.5 hours for each CNN.

For the 2 version of CNN in CNN_B, since we train 5 folds of each of the two models (each with 20% of data held-out) in addition to one full run, it takes about 30 hours to train and 8 hours to evaluate on a GTX 980Ti with cuDNN. The 4-chamber model trains only 5 folds (no full run), and training and evaluation together take around 9 hours.

A2. Dependencies

For all the code except those in CNN_B, it can run with Python 2.7.6 on Ubuntu 14.04 with a GTX 970 GPU, and it imports the following packages: cv2 3.1.0, theano 0.8.0.dev0.dev-RELEASE, lasagne 0.2.dev1, pandas 0.14.1, numpy 1.12.0.dev0+a2f5392. cuda 7.5 is used and cudnn is enabled.

To run code in CNN_B, you need to use cv2 version 2.4.12.

A3. How To Generate the Solution (aka README file)

A. Download and prepare data

Change the directories in SETTINGS.py to your settings, and download the sunny-brook data set, the train, valid, and test data set. Append the rows from validate.csv to train.csv and rename it as train_valid.csv

The directory manual_data/ includes all the hand labeled images and the contours, they are combined with the sunnybrook data to train the CNN networks.

B. Train CNNs to predict the contours of the LV

Part A

1. run >> bash CNN_A/run_train.sh
2. a) it preprocesses the image data for the CNN net to train.
3. b) it trains many version of the CNN models with different parameters. To save time, you can simply just run versions 3 and 6 and get a slightly worse result but 1/4 of the total amount of time. For each version of CNN, it takes about 3 hours to train on a GPU GTX 970, and 20 second to predict for each case.
4. c) it loads the trained CNN models and predicts the contours for all cases.
5. d) it extracts the sex-age inforamtion for later use to build sex-age based default model.
6. If there are additional cases that you need to make predictions, just run the run_test.sh script:
7. run >> bash CNN_A/run_test.sh
8. a) predicts the contours for test cases.
9. b) extracts the sex-age information for test cases.

Part B

run >>python train.py

C. Calculate the volumes

Combine (average) all the processed results that contains the area of the contours, calculate the volumes for each case, and fit simple models based on the train dataset to correct systematic errors, and predict for the unknowns.

run >> ./train_pred.py

A4. References

1) https://www.kaggle.com/c/second-annual-data-science-bowl/forums/t/18375/0-036023-score-without-looking-at-the-images
2) http://arxiv.org/abs/1502.03167
3) https://github.com/Lasagne/Lasagne/blob/master/lasagne/layers/normalization.py
4) http://arxiv.org/abs/1412.6980
5) https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient
6) http://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.rescale_intensity