Automated Left Ventricle Segmentation in Cardiac MRIs using Convolutional Neural Networks

Taman Narayan
tamann@stanford.edu

Abstract

We train a Convolutional Neural Network to perform semantic segmentation on cardiac MRI images to identify the left ventricle and leverage it to compute the volume of the ventricle throughout the course of a heartbeat. To help generalize and prevent overfitting, we propose a series of preprocessing and data augmentation steps centering around the application of image filters. We then evaluate the efficacy of a custom loss function designed to ensure consistency between predictions at nearby points in time. Finally, we demonstrate the value of post-processing the output to ensure that the final left ventricle prediction is a single filled and contiguous region.

1. Introduction

Medical imaging is a major component of modern-day health care, but analyzing and intepreting the resulting CT, MRI, and PET scans remains a challenging and time-consuming process. One prominent example is the calculation of the heart's ejection fraction, which is a barometer of cardiac health. There are a few different techniques employed, but they each by-and-large rely on manual segmentation of the left ventricle at some stage of the process. Given time-series images at several different slices, the process can take many minutes to complete.

With the success of Convolutional Neural Network (”CNN”) architectures in a wide range of settings, it is natural to focus on how effective they can be in the hospital - at improving performance, reducing errors, and saving time in the processing of medical images. We focus on the problem of left ventricle segmentation in Cardic MRI images, with the hope that CNNs can prove an effective aid in evaluating heart healthiness.

In particular, we perform CNN-based pixel-by-pixel semantic segmentation of individual Cardiac MRI images. Our output is a zero/one prediction of whether each pixel belongs to the left ventricle, which we compare to a set of ground-truth contours. As an additional task, we apply our model to a 4-Dimensional set of images without contours for each patient, with multiple slices of the heart and roughly 30 frames spanning a single heartbeat at each slice. Using some geometric approximations, we use our pixel-level predictions to compute the volume of the left-ventricle at its end-systolic (smallest) and end-diastolic (largest) phases and compare to provided volumes.

We find our segmentation model to work pretty effectively, achieving a nearly 80 percent mean intersection over union (”IOU”) on the test set with ground contours. The model performs less well when generalized to the patient level, with a Continuous Ranked Probability Score (”CRPS”) of around 0.08 when using a naive step function as the predicted Cumulative Distribution Function.

2. Related Work

There are two major pertinent strands of literature. The first is a large amount of domain-specific non-CNN approaches to left ventricle segmentation. One approach [6, 9] combines a specialized mechanism to find the center of the left ventricle with a mix of edge detection and smoothness filters moving outwards from the center. Others [10] use decision forests built on higher-level features constructed from the pixel data. Still others [15, 13] attempt to map “templates” to images using a variety of statistical tools. There are far too many techniques to individually cite, but [11] contains a thorough review of the field.

The second category of literature is from deep learning. Existing domain-specific approaches [2, 12] perform their segmentation by sliding differently sized windows over the training and test data or sampling random bounding boxes and classifying individual pixels based upon these patches. More recently, there have been considerable developments in developing fully convolutional segmentation models without explicit construction of region proposals or superpixels. Farabet et al. [3] employ a hybrid technique of performing convolutions over the entire image at different scales and concatenating the resulting features. Long et al. [8] experiment with how best to translate a course feature map generated from a series of convolutions and pooling steps back to a dense pixel space, settling on complex approach that blends fractionally strided convolutions of different sizes into a contiguous whole. Noh et al. [5] show the success of a simpler upsampling process, simply reversing a VGG 16-layer net and adding it to the original VGG net to get pixel-level predictions. It's the last of these approaches that most directly motivates our own.

3. Data

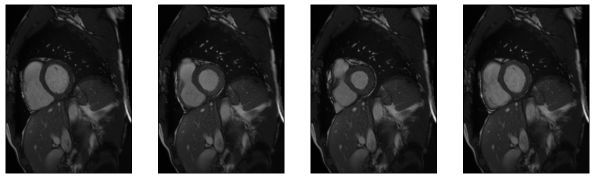

Figure 1. A series of images from a single short-axis slice.

We employ two datasets in this paper. Most importantly, we use the Sunnybrook Cardiac Data [11] from a 2009 Left Ventricle Segmentation Challenge. It consists of Cardiac MRI images for 45 patients collected in the Sunnybrook Health Sciences Centre in Toronto, some with healthy hearts and a majority with heart conditions of various kinds. For each patient, a subset of the images have ground-truth contours drawn by professional cardiologists. There are a total of 805 such images over the 45 patients, with 526 (30 patients) assigned to the training set, 139 (6 patients) to the validation set, and 140 (9 patients) to the test set. Each image is a 256 x 256 image cropped to focus on the 128 x 128 central portion of the image, which is where the left ventricle lies in each case.

The second dataset we use comes from the Kaggle Second Annual Data Science Bowl [1]. We focus on the training dataset, which consists of Cardiac MRI images for 500 patients and end-systolic and end-diastolic left ventricle volumes for each patient. We further limit ourselves to short-axis views of the heart in order to match the Sunnybrook data. Notably, there are no ground-truth contours for any images in the Kaggle data.

There is a great deal of the variety in the Kaggle data, above and beyond that found in the Sunnybrook data; images come from different hospitals, from patients of all ages and heart conditions, with differing levels of image quality, and with various resolutions. To solve the latter problem, we resize all of the images to either 192 x 256, 256 x 192, or 256 x 256, depending on their initial shape, and then take the central 128 x 128 portion.

4. Methods

4.1. Core Model

We employ a fully convolutional semantic segmentation model that combines pooling and upscaling layers to both start and end with the same resolution. We implement the model in Theano/Lasagne and train the model on a NVIDIA GRID K540 GPU. The model is eight layers deep, consisting of a series of three CONV-RELU-POOL layers (with 32, 32, and 64 3x3 filters), a CONV-RELU layer (with 128 3x3 filters), three UPSCALE-CONV-RELU layers (with 64, 32, and 32 3x3 filters), and a final 1x1 CONV-SIGMOID layer to output pixel-level predictions. Its structure resembles Figure 2, though with the number of pixels, filters, and levels as described here.

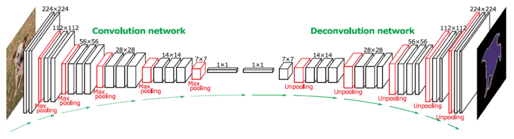

Figure 2. We use a symmetric downscaling and upscaling approach similar to Noh et al. [5], from where this image is taken.

While the back half of similar models has been implemented with fractionally strided convolutions, we employ simple upscaling layers paired with convolutions as a rough approximation, since fractionally strided convolutions are not implemented in the software package used in this paper. Both approaches seek to scale up the shrunken feature map up to the scale of the original picture and do so by aggregating nearby activations using shared weights.

The purpose of using a "downscale then upsize" model like this one is to incorporate features from a wider scale into the prediction of a given pixel's class. Without pooling layers (or strided convolutions), the effective receptive field of neurons in the net would grow very slowly with the 3x3 filters used here; after 8 layers, pixel feature maps would only contain information about a 17x17 pixel region around them. Meanwhile, a CONV taking place after just two pools is already incorporating information from an 18x18 surrounding region, and requires substantially less memory on the GPU as well. It is the combination of local and semi-local features that is key to effectively segment an image, and the number of filters increases in layers towards the middle to reflect their storing information about all of their consitutent pixels as well as regional features.

In training the model, we use a binary cross-entropy loss between the predictions for each pixel and the actual value

$$L = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{R}\sum_{r=1}^{R} -(G_r^b log(P_r^b) + (1 - G_r^b) log(1 - P_r^b))$$

where B represents the number of samples in the batch, R represents the number of pixels in an image, G_r^b represents the ground truth prediction at pixel r in image b, and P_r^b represents the prediction (between zero and one) for pixel r in image b.

The advantage of using a binary cross-entropy loss here (as opposed to, say, an L2 loss), is the classification context [4]; the predictions are generated from a sigmoid function and so always lie between 0 and 1, while the ground-truths are always 0 or 1. The cross-entropy function uses these facts to place a much stronger relative gradient on misclassified points compared to the L2 function, where the gradient increases linearly in the extent of misclassification.

4.2. Preprocessing

It may seem like cardiac MRI images are quite similar to one another compared to the dramatic variety in datasets like ImageNet, lessening the need for substantial data preprocessing or augmentation. However, the task at hand – pixel-by-pixel segmentation in images without obvious foregrounds and backgrounds – means that understanding the variation that is present is crucial to building a generalizable model, especially in the presence of the relatively small datasets available to us.

Through a great deal of manual visual inspection and experimentation, a few things became clear. First, it makes sense to apply a denoising filter to the input images. A number of different kinds of visual artifacts, such as light specks and odd textures, can be smoothly handled without removing any information that the model would use to distinguish left ventricular regions.

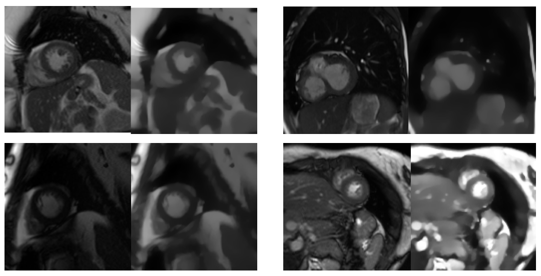
Figure 3. The effect of applying various image filters to raw images. Top left: denoising filter cleans up texture. Top right: denoising filter minimizes light specks. Bottom left: dark original and brightened counterpart. Bottom right: contrast enhancement highlights left ventricle.

It's not, however, entirely obvious what particular denoising filter would be most effective. One option would be to tune it as a hyperparameter, but given the capacity of the model to handle significantly more images, we instead throw in different denoising filters as part of a data augmentation procedure. An additional benefit of introducing multiple variable-intensity denoising filters, which effectively slightly blur the image, is that it could help address the problem that the Kaggle images have wide variety in their resolutions while the Sunnybrook images have identical resolution.

Additionally, there is a great deal of variation in lighting, both in terms of absolute brightness as well as in terms of contrast, that could easily trick a CNN attempting to distinguish boundary areas. To prepare the model to handle this variety, we throw in brightness and contrast-adjustment filters into the data augmentation procedure as well.

We proceed with 10x data augmentation using the above techniques, noting that with more time and compute power it would be possible to further ramp this up. At test time, we employ a medium-strengh denoiser to the images before passing them through the model.

4.3. Time Series Regularization

One aspect of the data that the CNN does not exploit at all thus far is the expected similarity between adjacent MRI frames. To remedy this, we build in the expected similarity across time for a given slice by adding a term to the loss function penalizing different predictions for nearby frames. This is implemented by tacking on a random sequence of images from the same slice in the Kaggle data to each minibatch in the training procedure, leaving the loss function

$$L = BCE(G) + \lambda \sum_{t=1}^{T-1} \|P^{t+1} - P^{t}\|_F$$

where $BCE(G)$ is the binary cross-entropy loss over a set of images with ground-truth contours, T is the number of continuous frames added to the end of the minibatch, and P^t is the matrix of predictions for the tth element of the time series.

We expect the time series regularization to help resolve a few problems. For one, it should help the model understand which types of image variation are relevant to locating the left ventricle. For example, there may be variations in relative brightness in the MRI over the course of a heartbeat that this would help the model ignore. In addition, there are times where part of the boundary of the left ventricle not consistently visible throughout the heartbeat due to the jostling of other parts of the heart; time series regularization can help the model understand this process. Third, the heart can look quite different at systole, with sharply shrunken ventricles that in the absence of explicit linkage with the rest of the heartbeat cycle could lead the model to not find any evidence of a left ventricle at all. Finally, since the time series images are from Kaggle, they should help the model establish consistency on a new set of images.

The other potential similarity to exploit would be spatial similarity between neighboring slices. Unfortunately, handling the similarity of MRI scans at nearby spatial locations is a nontrivial exercise. Since the different slices are captured at different points in time, there are frequently minor differences in where the ventricle is located in the different slices, meaning that metrics such as L2 difference would not be accurate comparisons and would introduce noise into the training process.

4.4. Postprocessing

Ideally, we would want the predictions to be as if they resulted from drawing and filling in a single contour. Up to this point, however, nothing constrains the CNN to produce a contiguous and filled-in region. To correct this, we employ a postprocessing sequence on the resulting predictions, as illustrated in Figure 4. In particular, after rounding the predictions to zero or one, we find the largest connected component of ones in the resulting matrix and change all other ones not part of that component to zeros. This returns a contiguous prediction. Then, we fill in all pixels between the extreme ones in each row and column with ones, plugging any holes and making the resulting shape more blocky.

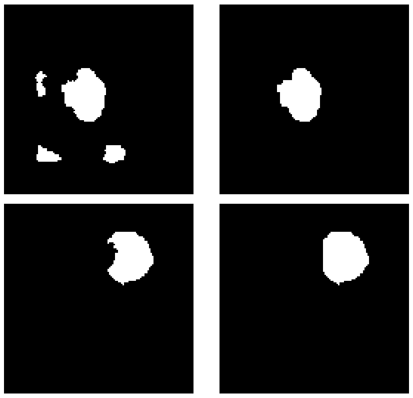

Figure 4. Example applications of postprocessing. Top: eliminates extraneous regions predicted to be the left ventricle. Bottom: fills in a predicted region.

As an important note, this postprocessing only takes place at test time. There is nothing in the loss function or training procedure that directly encourages contiguity.

5. Implementation Details

As in all CNNs, there are a lot of degrees of freedom in nailing down the particular implementation details even after laying out the structure in broad strokes. A major one is the precise architecture. The selection of the number of filters and number of layers was not too much of a choice; fewer layers and filters had substantial trouble overfitting to the data and frequently ended up with nonsensical output or zero predictions everywhere. Bounding the number of filters and layers from above were both the limited amount of training data available to learn useful parameters as well as GPU memory. The final numbers used proved a happy medium. We did not fiddle with the choice of 3x3 filters throughout, relying on the general success of 3x3 filters in the literature. We also stuck with RELU nonlinearities.

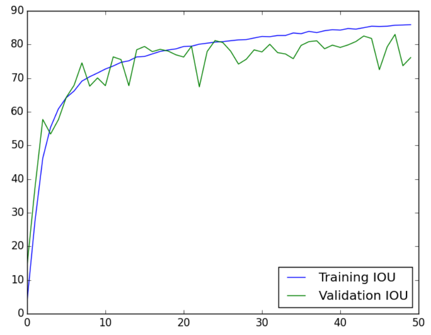

Figure 5. Training and Validation IOU by Epoch

Batch normalization proved effective at increasing the performance of the network and especially at avoiding the bad local minimum of zero predictions everywhere. We also found dropout layers on the back half of of the network to improve performance on the validation set, while dropout layers on the initial layers slightly decreased performance. We found that a dropout percentage of 0.5 worked the best after testing a few different numbers.

On the side of optimization, we used Adam with a learning rate of 0.0001. The learning rate was chosen by steadily lowering the learning rate from a starting guess of 1 until a steady decline in the loss function was achieved. After validation error appeared to reach convergence, we attempted lowering the learning rate by a factor of 10, but there was no meaningful improvement in validation error by doing so. Interestingly, validation IOU was quite “spiky” for the models trained, as seen in Figure 5, reaching high values quite early in the training process but remaining highly erratic and inconsistent between epochs. This variance likely reflects the small number of patients whose images are reflected in the validation data, meaning that small changes in interpretation would affect many the predicted output for many images.

Each model was trained for roughly 5,000 iterations over a mini-batch of 50 images, with the latter number selected with GPU constraints in mind. For models which did not use data augmentation, this translates to 500 epochs of training, while it translates to 50 epochs of training over the augmented data.

Some data processing scripts we used [14, 7] built on those made available by the Kaggle community.

6. Results

We primarily evaluate the model on two tasks. First, we run it on the test set observations with ground-truth contours from the Sunnybrook data and compare the pixel-level predictions. This is the most direct test of the model. Second, we apply it to the Kaggle data to compute left ventricle volumes for patients and compute its accuracy there. There is a lot of finetuning that could occur on the procedure that maps from image-level predictions to a cumumlative distribution function over patient-level volumes that is outside the scope of the paper, so we just present the results of a simple approach here.

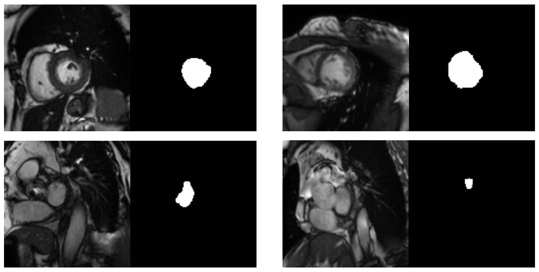
Figure 6. Sample model predictions of varying quality. Top row is largely accurate. Bottom left gets most of the left ventricle but might be confused by a small dark portion in the upper portion of the ventricle. Bottom right misses entirely, confused by the high brightness and lack of boundaries between the left ventricle and other heart organs.

6.1. Sunnybrook

The primary metric we use to evaluate the pixel-level predictions against the ground-truth contours is the Sorenson-Dice index, due to the prevalence of that metric among other papers in the segmentation field:

$$S = \frac{2|A \cap B|}{|A| + |B|}$$

where A is the set of pixels which are actually left ventricle pixels and B is the set of pixels which are predicted to be left ventricle pixels.

We also mention the Jaccard index, which we find to be a more easily interpretable number:

$$J = \frac{|A \cap B|}{|A \cup B|}$$

Both equations are very similar (in fact, they merely differ by a term $|A \cap B|$ in the numerator and denominator), but the Jaccard index has a clean interpretation as the mean IOU between the predicted and actual left ventricle pixels.

An important feature of both of these metrics, which run from zero to one, is that they do not overly reward the model for accurately predicting the background pixels accurately. Since around 90 percent of the pixels in the sample of MRI images are not left ventricle pixels, a model graded on pixel-by-pixel accuracy could score 90 percent merely by predicting that every pixel is a background pixel. As a result, differentiating between the performance of various models would become an exercise of parsing very small differences in accuracy. With the Sorenson-Dice and Jaccard metrics, meanwhile, all-background, all-ventricle, and random predictions will all score very poorly.

Model	Test Err. no PP	Test Err. w/ PP
Local CNN	30.4	51.2
Raw Pixels	74.9	81.0
Denoising	81.0	85.0
Augmentation	85.5	85.7
Time Reg	82.7	78.8
Challenge	89.0	89.0

Table 1. Sorenson-Dice scores for segmentation on Sunnybrook test set. PP stands for post-processing on the predictions. Local CNN refers to a CNN without pooling or upsampling layers. Raw Pixels refers to a model where the input is the unaltered image. Denoising refers to a model where the input is denoised images. Augmentation refers to a model with 10x data augmentation from image filters. Time Reg refers to a model with time series regularization. Challenge refers to the best results from the 2009 Left Ventricle Segmentation Challenge.

Moving on to the actual performance from Table 1, we find the best model to be the one which utilizes preprocessing and data augmentation, as well as postprocessing, but not the time series regularization. It scores 0.871 on the Sorenson-Dice index, with corresponds to 0.773 on the Jaccard index. Its performance is comparable to the best results from the 2009 Left Ventricle Segmentation Challenge from which the Sunnybrook data arises. As described in the literature review, those models were substantially more complex and domain-specific and generally were built to predict contours. It's quite remarkable that a seemingly naive approach of pixel-by-pixel prediction with some blunt tools to force the prediction into a contiguous filled region performs similarly to those models.

The "local" CNN mentioned in the results table to serve as a comparison is nearly identical to the front-half of the baseline model used in this paper, with the exception that it does not use pooling layers. Thus, the prediction for each pixel is the result of repeated shape-preserving convolutions gathering local features only. It also uses 5x5 filters instead of 3x3 filters to slightly expand the scope of its local features (3x3 filters performed quite poorly given the 4-layer structure employed, though it is likely they would perform better if more layers were added). Getting this local model to avoid the local minimum of zero predictions everywhere was actually quite challenging and ultimately required using a different loss function entirely - one which took the equally-weighted average of cross-entropy loss over background and ventricle pixels instead of lumping them together, thereby dramatically increasing the penalty for missing the less-populous ventricle pixels. This weighted loss performed worse than a standard unweighted cross-entropy loss for all non-local CNNs.

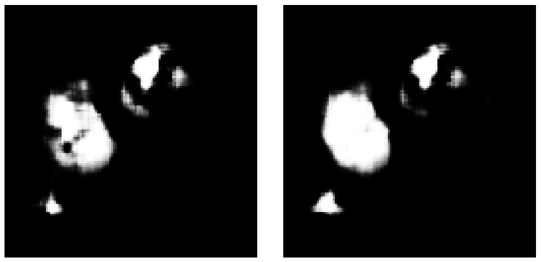
Figure 7. Example predictions on neighboring frames from a model without time series regularization.

Interestingly, the time series regularization model performs worse with post-processing, the only model to do so. Anecdotally, as illustrated in Figure 7, the reason for this seems to be that the most prominent disagreements between predictions over nearby frames tended to be over the classification of extraneous pixels away from the left ventricle and pixels deep within the interior of the left ventricle. These issues would both be resolved by postprocessing. Whatever insight the model gained over predicting those regions accurately would have to be weighed against the slightly higher uncertainty over the prediction of border pixels, which would generally "legitimately" differ between nearby frames. In this case, the calculus came out slightly against the inclusion of frame-by-frame regularization.

6.2. Kaggle

The most natural metric to evaluate performance on the Kaggle data would probably be Root Mean Squared Error ("RMSE"), since the final output is two real numbers for each patient. Since the CRPS metric is used in the competition, we also present that. Normally, CRPS is used to judge predicted cumulative distribution functions. However, since we simply output a point estimate for each value in this paper (though a lot more clever things could be done), it reduces to

$$CRPS = \frac{1}{1200N}\sum_{m=1}^{N}|V_D^m - P_D^m| + |V_S^m - P_S^m|$$

where N is the number of patients evaluated, V_D^m is the actual end-diastolic left ventricle volume, V_S^m is the actual end-systolic volume, and P_D^m and P_S^m are the predicted volumes.

To get to a point-estimate of end-systolic and end-diastolic volumes, we use the following procedure:

1. Apply the CNN and postprocessing to compute final 0/1 pixel-level predictions for each image.

Model	RMSE	CRPS
Raw Pixels	72.5	0.094
Denoising	70.4	0.091
Augmentation	65.6	0.084
Time Reg	66.9	0.085
Median	67.6	0.093

Table 2. Model performance on Kaggle training data. First hundred patients used due to computational constraints. Post-processing used for all models. Raw Pixels refers to a model where the input is the unaltered image. Denoising refers to a model where the input is denoised images. Augmentation refers to a model with 10x data augmentation from image filters. Time Reg refers to a model with time series regularization. Median refers to a prediction of the median value for the dataset for each observation.

2. Count the number of left ventricle pixels and scale to account for the image resolution and any resizing to get an area.

3. Choose the smallest and largest areas in each slice.

4. Stitch together the areas for each slice by approximating the volume of the region between neighboring slices as a frustum (cone with its point chopped off)

As we can see in Table 2, the best performer is again the model with data augmentation but without time series regularization, though the time series model is a lot closer here than in the testing on the Sunnybrook data. The highest CRPS achieved is 0.084. This beats a model predicting the median volume for each patient by about 10 percent. This might not seem like much, but it's worth noting that Kaggle's Deep Learning Tutorial [14] ends up with a segmentation model that performs about twice as poorly as a model that predicts the median each time.

There are a number of things that could be done, even holding fixed the pixel-level predictions, to score better on the CRPS. One strand is better approximating the volume. This could be achieved by analyzing whether the frustum is the best approximation to use and, if so, calibrating the resulting volumes against a portion of the Kaggle data so that their means match. We could also explore whether or not taking the absolute minimum and maximum area in each slice is the appropriate way to go, as opposed to trusting the Kaggle data to be arranged in order of time and so picking the overall minimum and maximum volume time slice (the answer appears to be that it usually is arranged in such a way).

Another way to improve CRPS performance would be making the final predictions more error-resistant. One way would be to set a floor and a ceiling for individual frame-level predictions to avoid nonsensical outcomes like zero area in a given slice. Since only the extreme values are used in the current calculation, the final output is very susceptible to these errors. Another technique would be fitting a smoother to the predicted areas in each slice and taking the maximum and minimum of those smooth functions instead of the raw output.

A third improvement would be outputting an actual CDF instead of just a point estimate. The best way to do this would probably be using the Kaggle training data to calibrate the parameters of a smooth CDF function. Additionally, the variance in image-level predictions for a given patient might turn out to be a proxy for uncertainty in the predicted CDF and this could be tested.

Again, though, the focus of this paper is on the problem of semantic segmentation rather than Kaggle optimization, so we merely suggest these improvements instead of implementing them.

7. Conclusion

We explored a number of techniques to improve CNN performance on medical images. Attempting to leverage the unique aspects of the problem, we explored introducing more variety in noisiness, lighting, and contrast to the images and found substantial improvements in performance at test time. Adding a time-series regularization component to the loss function sounded promising but ended up giving slightly worse performance overall. And postprocessing techniques of taking the largest connected component and filling it in meaningfully improved performance on both the Sunnybrook and Kaggle data.

Aside from the Kaggle-specific optimizations that could improve competition performance, there are a number of broader questions that would benefit from further research. A big one is how best to leverage spatiotemporal similarity in MRI images. The type of regularizer tried here turned out to not improve performance, but there may be a number of other ways to avoid treating every image as separate and distinct. It is also an open question how best to output a filled-in contour using CNNs; the postprocessing approach here, while effective, seems a bit too crude to be the best general approach.

More generally, given the prominence of medical imaging, there are great gains to understanding along which precise axes images differ from one another in order to maximize performance and generalizability with tools like data augmentation. CNNs are certainly a fantastic general purpose tool, but the uniqueness of medical images compared to most other tasks they are employed for may call for domain-specific best practices. With continued and determined research efforts, software can make a meaningful improvement in the delivery of health care.

References

[1] Data science bowl cardiac challenge data.

[2] O. Emad, I. Yassine, and A. Fahmy. Automatic localization of the left ventricle in cardiac mri images using deep learning. In *Engineering in Medicine and Biology Society, 2015 37th Annual Conference of the IEEE*, 2015.

[3] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[4] P. Golik, P. Doetsch, and H. Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, pages 1756–1760, 2013.

[5] H. N. S. Hong and B. Han. Learning deconvolution network for semantic segmentation, 2015.

[6] S. Huang, J. Liu, et al. Segmentation of the left ventricle from cine mr images using a comprehensive approach. *The MIDAS Journal*, 2009.

[7] M. Jocic. Keras deep learning tutorial for Kaggle 2nd annual data science bowl, 2015.

[8] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR (to appear)*, Nov. 2015.

[9] Y. Lu, P. Radau, K. Connelly, A. Dick, and G. Wright. Automatic image-driven segmentation of left ventricle in cardiac cine MRI. *The MIDAS Journal*, 2009.

[10] J. Margeta. *Machine Learning for Simplifying the Use of Cardiac Image Databases*. PhD thesis, MINES Paris Tech, 2015.

[11] P. Radau, Y. Lu, et al. Evaluation framework for algorithms segmenting short axis cardiac MRI. *The MIDAS Journal*, 2009.

[12] H. R. Roth, L. Lu, A. Farag, H.-C. Shin, J. Liu, E. Turkbey, and R. M. Summers. Deeporgan: Multi-level deep convolutional networks for automated pancreas segmentation, 2015.

[13] W. Shi, X. Zhuang, et al. Automatic segmentation of different pathologies from cardiac cine MRI using registration and multiple component EM estimation, 2011.

[14] V. Tran. A fully convolutional network for left ventricle segmentation, 2015.

[15] Y. Znehg, A. Barbu, et al. Four-chamber heart modeling and automatic segmentation for 3-d cardiac CT volumes using marginal space learning and steerable features. In *IEEE Transactions on Medical Imaging*, pages 1668–1681, 2008.