DSBC Challenge - Model outline

February 10, 2016

Implementation summary 1 autoencoder, 1 CNN, 1 stacked autoencoder Training/ testing time as stated by [1] (using Intel Xeon CPU 2.6 GHz, 32 RAM, MATLAB 2014):

- Training autoencoder to obtain filters: 63.3 seconds
- Training convolutional neural network: 3.4 hours
- Training stacked AE: 34.25 minutes
- Testing: ROI detection 0.25 seconds
- Testing: stacked-AE: 0.002 seconds
- Testing: final segmentation: 0.2 seconds

1 Data preprocessing

- All these preprocessing steps only refer to the Sunnybrook data
- Divide training set into large contour groups/ small contour groups. The first group contains scans close to the middle of the heart, the latter scans that belong to the apex.
- Increase training data size by image translation, rotation and changing pixel intensities based on the standard PCA technique explained in [2] ([1] ended up with a split 1350/1250).
- Resize each 256 x 256 image to a 64 x 64 image (rescale factor = 4).
- From the given ground truths create binary region of interest masks (64 x 64 images which are black (=0) where we are outside a box around the left ventricle and white inside (=1))
- Randomly extract $N = 10^4$ 11 x 11 patches from the downsampled 64 x 64 training images. They will be used in the pre-training step of the model.

2 Pretraining

The model has a pre-training step in which we learn the parameters of a very simple convolutional NN. The goal of the pre-training is to detect the region of interest in the training image, i.e. the relevant region around the left ventricle (not the contour!). The block diagram of the convolutional NN is depicted in figure 2.
Let's go through the specific layers of the CNN:

2.0.1 Input and Convolution

Feed in a 64x48 training image. Instead of using hand-crafted filters that are convolved with the input image, we learn them with an auto-encoder.

- The autoencoder is fed the small patches that were extracted during the pre-processing step. That is, the input and output layer of the autoencoder have $11^2 = 121$ units.

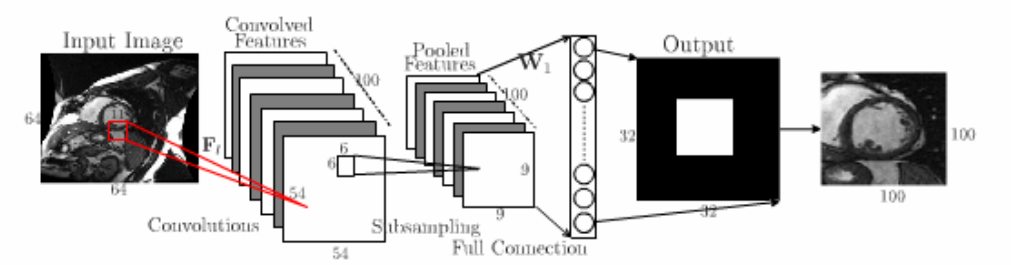


Figure 2: Block diagram of automatic detection of LV in MRI dataset.

- The input is mapped to a single hidden layer with 100 units, denoted $a_2^{(i)} \in \mathbb{R}^{100}$ for the $i-th$ mini-patch $x^{(i)}$. The mapping is described by $a_2^{(i)} = f(W_2 x^{(i)} + b_2)$, where f is a sigmoid function.
- The hidden units are mapped via $f(W_3 a_2^{(i)} + b_3)$ to the output $y^{(i)}$.
- The costfunction is given by

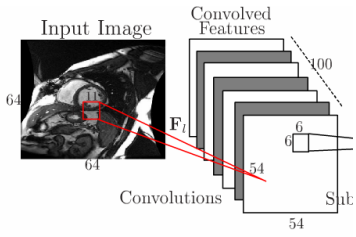
$$J(W_2, b_2) = \frac{1}{10000} \sum_{i=1}^{10000} \|y^{(i)} - x^{(i)}\|^2 + \frac{\lambda}{2}(\|W_2\|^2 + \|W_3\|^2)$$

  and is minimized with the constraint that the mean of the hidden units is equal to some rho for every mini-patch.
- Once the autoencoder is trained the rows of $W_2 \in \mathbb{R}^{100\times 121}$ are our 100 filters (they have to be resized to 11 x 11).

Now, we can obtain 100 new 54x54 images with coordinate values

$$Z_l(i,j) = \sum_{k_1}^{11} \sum_{k_2}^{11} F_l(k_1, k_2) \mathrm{Im}(i + k_1 - 1, k + k_2 - 1) + b_0(l),\ l \in \{1, \ldots, 100\}$$

where b_0 is given by the vector b_2 in the autoencoder, $i \in \{1, \ldots, 54\}$ and $j \in \{1, \ldots, 54\}$.



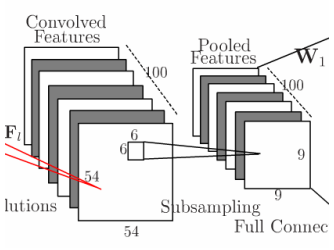
2.0.2 Pooling

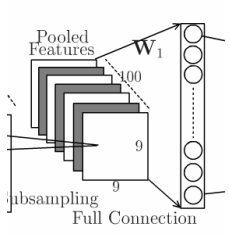
Every 54x54 image is again reduced in size by taking the average of non-overlapping 6x6 patches. This will yield a 9x9 image, and we will again have 100 of those.

- The coordinates of the pooled 9x6 image will be given by

$$P_l(i_1, j_1) = \frac{1}{6} \sum_{i=6i_1-5}^{6i_1} \sum_{i=6j_1-5}^{6j_1} Z_l(i,j)$$

- Next we resize the 9x6 images to 54x1 vectors and stack them upon each other to obtain a 5400x1 vector





2.0.3 Pre-training the output layer

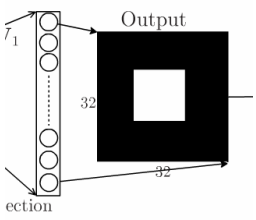
- We pass a 64x64 training image up to the last point of the pooling step.
- Then we train W_1, b_1 that map the pooled vector to the output

$$\mathbb{R}^{1024} \ni y_c = W_1 p + b_1$$

where $p \in \mathbb{R}^{8100}$ $(8100 = 100 \cdot (9 \times 9)$ minimizing the cost function

$$J(W_1, b_1) = \frac{1}{2N} \sum_{i=1}^{N} \|y_c^{(i)} - l_{roi}^{(i)}\|^2 + \frac{\lambda}{2} \|W_1\|^2$$

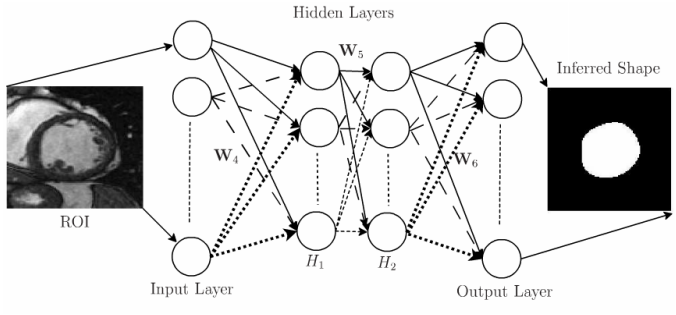
The number of training images is denoted by N and the vector $l_{roi}^{(i)} \in \{0,1\}^{1024}$ is the 32 x 32 binary mask (resize from 64 x 64) for the region of interest of the image i (obtained in the pre-processing step). The unrolled (shrinked) binary mask is a 1024x1 vector.



2.0.4 Fine-tuning

- Now, the whole model is fine tuned by minimizing the cost

$$J(F_l, b_0, W_1, b_1) = \frac{1}{2N} \sum_{i=1}^{N} \|y_c^{(i)} - l_{roi}^{(i)}\|^2 + \frac{\lambda}{2} (\|W_1\|^2 + \sum_{l=1}^{100} \|F_l\|^2)$$



3 Shape inferring

3.0.1 Setting up the stacked Autoencoder

The ouput of the convolutional NN is a 32x32 prediction of the binary mask of a training image. When we double that output in size we again have a 64x64 image (the size of the training image we started with) but note that this time the output is a prediction for the region of interest only. What goes into the stacked autoencoder shows only the left venctricle and its immediate surrounding. To infer the actual contour of the left ventricle we construct a stacked autoencoder with **two** hidden layers[1]

- The input layer x_s of the AE has 4096 units $(64 \cdot 64)$.
- The first hidden layer is computed as $h_1 = f(W_4 x_s + b_4),\ W_4 \in \mathbb{R}^{100\times 4096},\ b_4 \in \mathbb{R}^{100}$
- The second hidden layer is computed as $h_2 = f(W_5 h_1 + b_5),\ W_5 \in \mathbb{R}^{100\times 100},\ b_5 \in \mathbb{R}^{100}$
- The output layer is computed as $y = f(W_6 h_2 + b_6),\ W_6 \in \mathbb{R}^{4096\times 100},,\ b_6 \in \mathbb{R}^{4096}$ and is again a binary mask which is black everywhere except at the borders of the left ventricle.

3.0.2 Training the stacked Autoencoder

Once the stacked autoencoder (SAE) is implemented, we will train it again in a two stage fashion just as the non-stacked AE.

1. **Pretraining** In pretraining of the SAE our goal is to get good initial values for the parameters $W_4, W_5, W_6, b_4, b_5, b_6$ (see figure 3.0.2). For W_4, W_5, b_4, b_5 this is done in an unsupervised fashion. Extract two un-stacked AEs, one with layers 'input-H1-input' for W_4, b_4 and one with layers 'H1-H2-H1' for W_5, b_5, and train them using the same code as for learning the filters for convolution (see above). In contrast W_6, b_6 are learnt in a supervised fashion using the ground truth of the LV as visible units. The outputs from the last hidden unit are all passed into each unit representing a pixel in the ground truth. In these units a sigmoid function decides the predicted probability that the pixel belongs to the LV or not, so simple classification. Denoting $l_{lv}^{(i)} \in \mathbb{R}^{4096}$ the unrolled ground truth of the i-th image, the cost function that ought to be minimized is given by

$$J(W_6, b_6) = \frac{1}{2N_2}\sum_{i=1}^{N_2}\|y_s^{(i)} - l_s^{(i)}\|^2 + \frac{\lambda}{2}\|W_6\|^2$$

2. **Fine-tuning** Our pretraining results in good initial values $W_4, W_5, W_6, b_4, b_5, b_6$. Now the whole SAE is fine-tuned by minimizing the cost function

$$J(W_4, W_5, W_6, b_4, b_5, b_6) = \frac{1}{2N_2}\sum_{i=1}^{N_2}\|y_s^{(i)} - l_s^{(i)}\|^2 + \frac{\lambda}{2}(\|W_4\|^2 + \|W_5\|^2 + \|W_6\|^2$$

[1] actually we should also experiment with more hidden layers. Keep the implementation general enough to allow for such extensions.

4 Active Contour model

The last step towards the ultimate LV segmentation is running an active contour model on the LV prediction (64 x 64 binary) obtained from the stacked autoencoder.
We will evolve a contour of the LV so that the contour minimizes some energy function (high energy = bad, low energy = good). One has to imagine this procedure like gradient descent in common calculus ($x_{t+1} - x_t = -\nabla f(x_t)$) but this time we done evolve a point x_t but a function/contour $\phi^{(t)}$. We improve this function $\phi^{(t)} \longrightarrow \phi^{(t+1)}$ in direction of a gradient of a functional of the contour $\nabla_\phi \text{Energy}(\phi)$.
Contrary to intuitive imagination, $\phi : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$ is not the contour itself but defines the contour implicitly by $C = \{(x, y) |\ \phi(x, y) = 0\}$. So we rather evolve a mountain range which - sliced at the sea level - yields the contour. The energy which controls this evolution has three terms (for the maths see the paper)

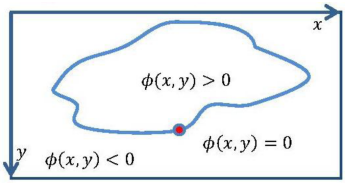


Figure 1: The regions of ϕ

- A term that makes sure the contour does not evolve too far away from our LV prediction
- A term that punishes if the contours length grows out of bounds
- A term that encourages a constant pixel intensity inside the contour which differs from the pixel intensity outside the contour

The evolution equation that we need is given by

$$\frac{\partial \phi}{\partial t} = \delta_\epsilon(\phi) \left[\alpha_1 \text{Div}\left(\frac{\nabla \phi}{\|\nabla \phi\|} \right) + \alpha_2((I_s - c_2)^2 - (I_s - c_2)^2) - 2\alpha_3(\phi - \phi_{\text{shape}}) \right] \tag{1}$$

where $I_s(x, y)$ is just the image value at (x, y) and c_1, c_2 are the average pixel values outside and inside the contour, respectively. The shape obtained from the CNN is denoted $\phi_{textshape}$.
The divergence operator is just the trace of the Jacobian of its argument,

$$\text{Div}\left(\frac{\nabla \phi}{\|\nabla \phi\|} \right) = \frac{\partial}{\partial x} \frac{\phi_x}{\|\nabla \phi\|} + \frac{\partial}{\partial y} \frac{\phi_y}{\|\nabla \phi\|}$$

The prefactor $\delta_\epsilon(\phi)$ in (1) is defined as

$$\delta_\epsilon(x) = \frac{1}{\epsilon \pi (1 + (z/\epsilon)^2)}$$

The parameters are

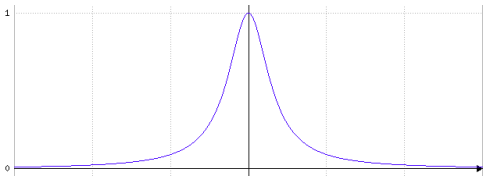


Figure 2: $\delta_\epsilon(x)$

- $\alpha_1 = 1$
- $\alpha_2 = 0.5$
- $\alpha_3 = 0.25$
- $\epsilon = \frac{1}{\pi}$

Numerical implementation

- The LV image is 64x64 binary (LV = white, background = black). We define $63 \cdot 63$ gridpoints $(x_i, y_j) = (i, j),\ 1 \leq i \leq 63,\ 1 \leq j \leq 63$.
- With $\Delta t = 0.1$ create a 2d numpy array (63 x 63) $\phi^n_{i,j} = \phi(n\Delta t, x_i, y_j)$. We initialize as $\phi^0_{i,j} = 3,\ if LV(x_i, y_j) = 1,\ \phi^0_{i,j} = -3,\ if LV(x_i, y_j) = 0$.
- DO WHILE(CONVERGENCE == FALSE)

  1. Compute all all finite differences

$$\begin{aligned}
\Delta_x \phi^n_{i,j} &:= \frac{\phi^n_{i,j+1} - \phi^n_{i,j-1}}{2} \\
\Delta_y \phi^n_{i,j} &:= \frac{\phi^n_{i+1,} - \phi^n_{i-1,j}}{2} \\
\Delta_{yy} \phi^n_{i,j} &:= \frac{\phi^n_{i+2,j} - 2\phi^n_{i,j} + \phi^n_{i-2,j}}{4} \\
\Delta_{yx} \phi^n_{i,j} &:= \frac{\phi^n_{i+1,j+1} - \phi^n_{i+1,j-1} + \phi^n_{i-1,j+1} + \phi^n_{i-1,j-1}}{4} \\
&= \Delta_{xy} \phi^n_{i,j} \\
\Delta_{xx} \phi^n_{i,j} &:= \frac{\phi^n_{i,j+2} - 2\phi^n_{i,j} + \phi^n_{i,j-2}}{4}
\end{aligned}$$

  2. Compute the averages

$$\begin{aligned}
c_1(\phi^n) &= \frac{1}{N_{\text{in}}} \sum_{i,j:\ \phi^n(x_i,y_j)>0} ROI(x_i, y_j) \\
c_2(\phi^n) &= \frac{1}{N_{\text{out}}} \sum_{i,j:\ \phi^n(x_i,y_j)<0} ROI(x_i, y_j)
\end{aligned}$$

  3. Compute the divergence

$$\text{DIV} := \frac{\Delta_{xx}\phi^n_{i,j} + \Delta_{yy}\phi^n_{i,j}}{\sqrt{(\Delta_x \phi^n_{i,j})^2 + (\Delta_y \phi^n_{i,j})^2}} - \frac{\Delta_x \phi^n_{i,j}(\Delta_{xx}\phi^n_{i,j} + \Delta_{xy}\phi^n_{i,j}) + \Delta_y \phi^n_{i,j}(\Delta_{yy}\phi^n_{i,j} + \Delta_{xy}\phi^n_{i,j})}{(\Delta_x \phi^n_{i,j})^2 + (\Delta_y \phi^n_{i,j})^2)^{3/2}}$$

  4. Compute

$$\phi^{n+1}_{i,j} = \phi^n_{i,j} + \Delta t \left[\delta_\epsilon(\phi^n_{i,j}) \left(\text{DIV} + \alpha_2 (ROI(i,j) - c_1(\phi^n))^2 - \alpha_2 (ROI(i,j) - c_2(\phi^n))^2 - 2\alpha_3(\phi^n_{i,j} - \phi^0_{i,j})\right)\right] \tag{2}$$

     Caution: In [1] $\phi > 0$ **outside** the contour. So we have to swap c_1, c_2 in (2) .

  5. CONVERGENCE == TRUE if $\|\phi^{n+1} - \phi^n\|_{\text{Frobenius}} < \epsilon$

References

[1] Avendi, D. A., et. al. (2015): A Combined Deep-Learning and Deformable-Model Approach to Fully Automatic Segmentation of the Left Vecnticle in Cardia MRI

[2] Koikkalainen, J., et. al. (2008): Methods of artificial enlargement of the training set

[3] Chan, T., Vese, L. (2001): Active Contours Without Edges