Federated Simulation for Medical Imaging Appendix

Daiqing Li[1⋆], Amlan Kar[1,2,5], Nishant Ravikumar[3], Alejandro F Frangi[3,4], and Sanja Fidler[1,2,5]

[1] NVIDIA [2] University of Toronto [3] University of Leeds
[4] KU Leuven [5] Vector Institute

1 Out-of-Distribution Simulation Experiments

We simulate the case where a patient from one hospital A goes to hospital B, by running the GAN trained for hospital B on the patient's ground truth segmentation mask and downsampled CT image. An ideal segmentation network would perform well on this out-of-distribution sample. We summarize results in Tab. 1, where we see that our method consistently outperforms or performs similarly to our baseline methods on dealing with these (simulated) out-of-distribution inputs, which comes from our disentanglement of the sensor and content (shape and material), helping segmentation models trained on our simulated data generalize better. This experiment is in simulation since gaining such data (a patient with data at two different sites) is a challenge, but we hope to be able to perform this experiment on real data in the future.

Table 1. Quantitative Results of the Out-Of-Distribution simulation experiment, where we test performance on simulated data of patient from one hospital going to another hospital. Method with highest mean is in bold.

		CT20 Label Sz. 8		CT34LC Label Sz. 12		CT34MC Label Sz. 12	
		CT34LC	CT34MC	CT20	CT34MC	CT20	CT34LC
Single Site Simulation	Lower Bound	72.56±2.02	67.52±1.94	84.88±3.31	83.03±3.47	84.41±3.13	86.27±2.79
	Ours-Fix-Mat	73.96±3.12	69.79±1.49	**87.30±2.65**	**84.41±2.41**	84.14±2.62	86.94±2.75
	Ours-Pre	**76.58±3.01**	**71.84±2.21**	86.22±2.20	83.65±2.38	81.54±2.01	84.46±3.20
	Ours-Full	75.61±1.28	70.55±0.77	84.87±2.92	84.08±3.11	83.13±2.63	85.93±2.17
	Upper Bound	76.19±1.64	71.69±1.91	84.01±2.54	82.73±3.23	84.15±1.86	86.39±2.63
Fede. Sim	Direct-FL	**78.33±3.37**	73.05±3.53	85.63±2.13	84.67±2.90	83.86±2.06	**87.95±2.16**
	Ours-Sim-FL	78.07±3.53	**74.11±2.77**	**86.79±1.86**	**84.70±2.41**	**84.52±2.94**	87.76±2.83

2 Shape Preprocessing Details

Our shape model are obtained from MM-WHS challenge [5] MRI annotation. We convert the 20 cardiac volume label into mesh using Marching Cube [2]. The

⋆ Correspondence to {daiqingl,sfidler}@nvidia.com

extracted mesh model contains 4319 vertices and 8610 faces. We then estimate our SSM model using PCA to obtain the mean shape and the vectors of SSM weights. The shape class mean and covariance can be written as:

$$\tilde{s} = \frac{1}{M}\sum_{i=1}^{M} s_i \tag{1}$$

$$C = \frac{1}{M-1}\sum_{i=1}^{M}(s_i - \tilde{s})(s_i - \tilde{s})^T \tag{2}$$

The PCA of the shape produces l eigenvectors $\Phi = [\varphi_1\varphi_2...\varphi_l]$ and the corresponding eigenvalues $\mathbf{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_l])$. Then the new shape can be approximated from the following linear generative model:

$$s \approx \tilde{s} + \Phi b \tag{3}$$

where $b \in \mathcal{R}^{14}$ are shape parameters. In our experiment, we use the first 14 eigenvectors and we limite the range of b from $-1.5\sqrt{\lambda}$ to $1.5\sqrt{\lambda}$.

3 Shape/Material Parameter Generative Model Implementation Details

In all our experiments, we choose our latent vector $z \in \mathcal{R}^{32}$. Our Generative Model of Shape G_{θ_S} is parametrized as a three layers Multilayer Perceptron(MLP). Each layer is a linear layer followed by an Leaky-ReLU activation except for the last layer where the activation function is Tanh. The layer weights are of size 32×256, 256×128, and 128×21. The design of our generative model of material G_{θ_M} is based on [4]. It consists of three fully convolutional layers with $\{256, 128, 1\}$ number of channels , kernel sizes of $\{3, 3, 3\}$ with a stride of 1. Similar to [4], we add batch normalization and ReLU layers between convolutional layers, and a Tanh layer at the end. Instead of using Transposed Convolutions to increase the output spatial dimension, we use Nearest-Neighbour Upsampling before each convolution to avoid checkerboard artifacts [3]. Specifically, we upsample with a scaling factor 4 to transform input z with spatial dimension $1 \times 1 \times 1$ (and 32 channels) into $4 \times 4 \times 4$. Then we use two more upsampling layers with scaling factor 2 before each convolution layer to get the final output in $\mathcal{R}^{16\times16\times16}$.

4 Semi-Supervised Learning Training Details

At the beginning of this phase of training, we fit a multivariate normal distribution to the latent vectors optimized in the pre-training phase and sample new random latent vectors z_i for the new unlabelled data-points from this distribution. The intuition is that unlabelled data and labelled data come from the same

data distribution, thus the latent representation of unlabelled data-point should come from the same latent distribution. While training, we use two Adam [1] optimizers with two different learning rate schedules for labelled and unlabelled data respectively. When training with labelled data (data used in the pre-training phase), we use a learning rate $1e^{-4}$ for z_i, G_{θ_S} and G_{θ_M} and $1e^{-5}$ for G_{GAN}. For the unlabelled data, we use a learning rate $1e^{-3}$ for z_i, G_{θ_S} and G_{θ_M} and $1e^{-4}$ for G_{GAN}. As the unlabelled data was not used in the pre-trained stage, we use a larger learning rate these data points. These learning rates are used for the first 30 epochs, and we linearly decay the learning rate of all the models to 0 for the last 30 epochs. Note that we freeze the weights of Discriminator during the training with the assumption that it can already distinguish fake/real data well by learning from labelled data, which we found this stabilize the training process significantly.

5 Federated Learning Setup Details

In our Federated Learning experiment, we setup the gradient communication between the clients and the server synchronously with a gradient update step 1. Specifically, we assume each client holds a private dataset (in our case, CT20, CT34_LC and CT34_MC) and the same model as in the server's site. At each training step in the client site, the clients will send back the gradient with respect to the current model sequentially to the server. In the server site, the server will do gradient aggregation and update the model parameters once it receives all the gradient from the clients. After the gradient update, the server will send back the new model to each of the client. This process continues until a maximum number of iteration. Implementation wise, this is equivalent to maintain a mini-batch of each client's private data and at each training step, the model will do a forward pass sequentially for each of the mini-batch. And then update the current model's parameters using the averaged accumulated gradient from all the mini-batches. In our experiment, we choose batch size one for each of the client, which makes the overall effective batch size three since we do one gradient update step after three forward passes. To compromise it, we also use batch size three for our method in FL setup. Note that in the FL baseline implementation, we use validation set from all the clients to select the best model and adjust the learning rate, which is different from our methods where we use fix number of iteration to pick the model. This difference might give our FL baseline advantage since it can learn a more generalized model using average score from all validation set. Under this setup, our model shows comparable performance comparing to the baseline. We will further exploit different options and a more realistic FL implementation for future work.

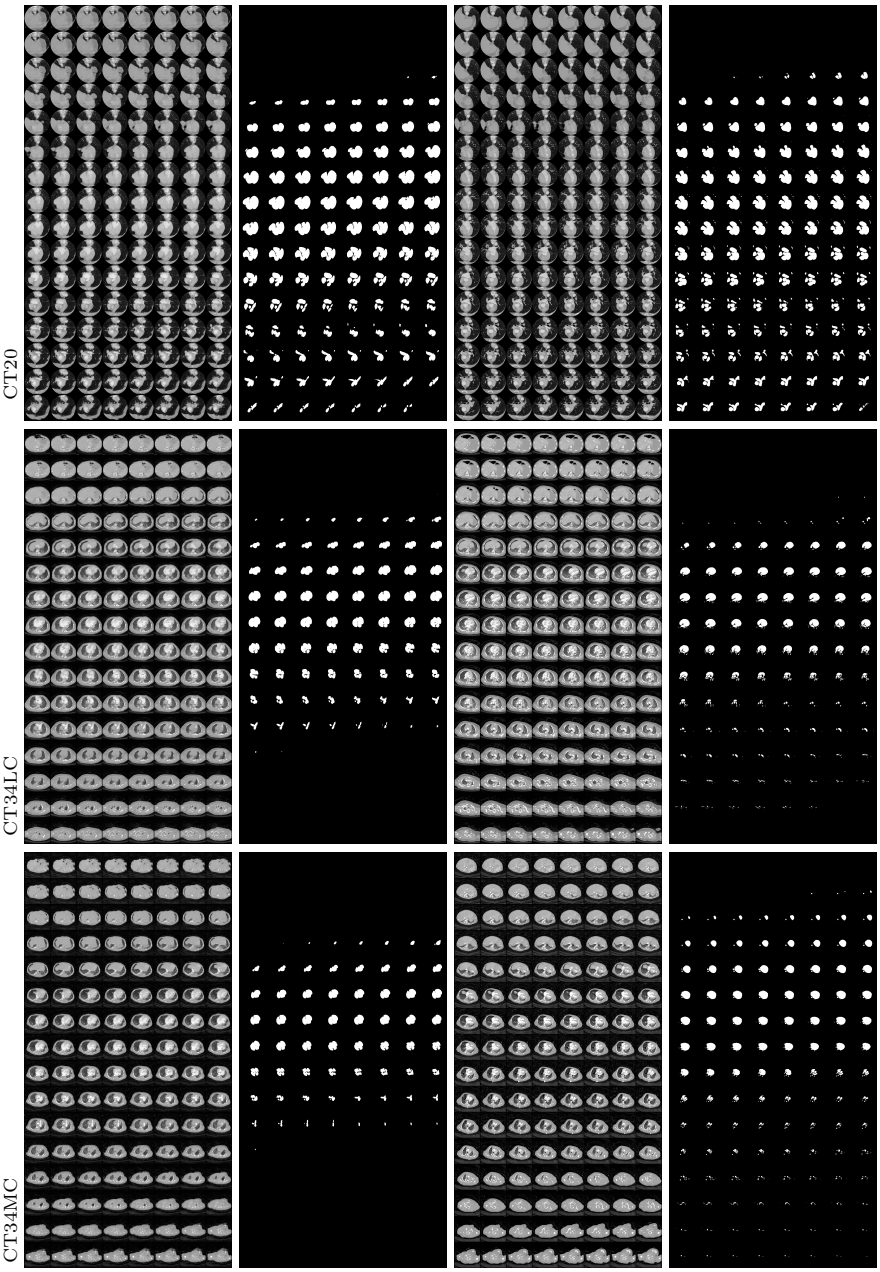

Fig. 1. Qualitative Results: First two columns show random samples (full volume) from our full model on each of the datasets. Last two columns show nearest neighbour from the training set. We see that our model can generate plausible yet novel data samples with annotations (second column).

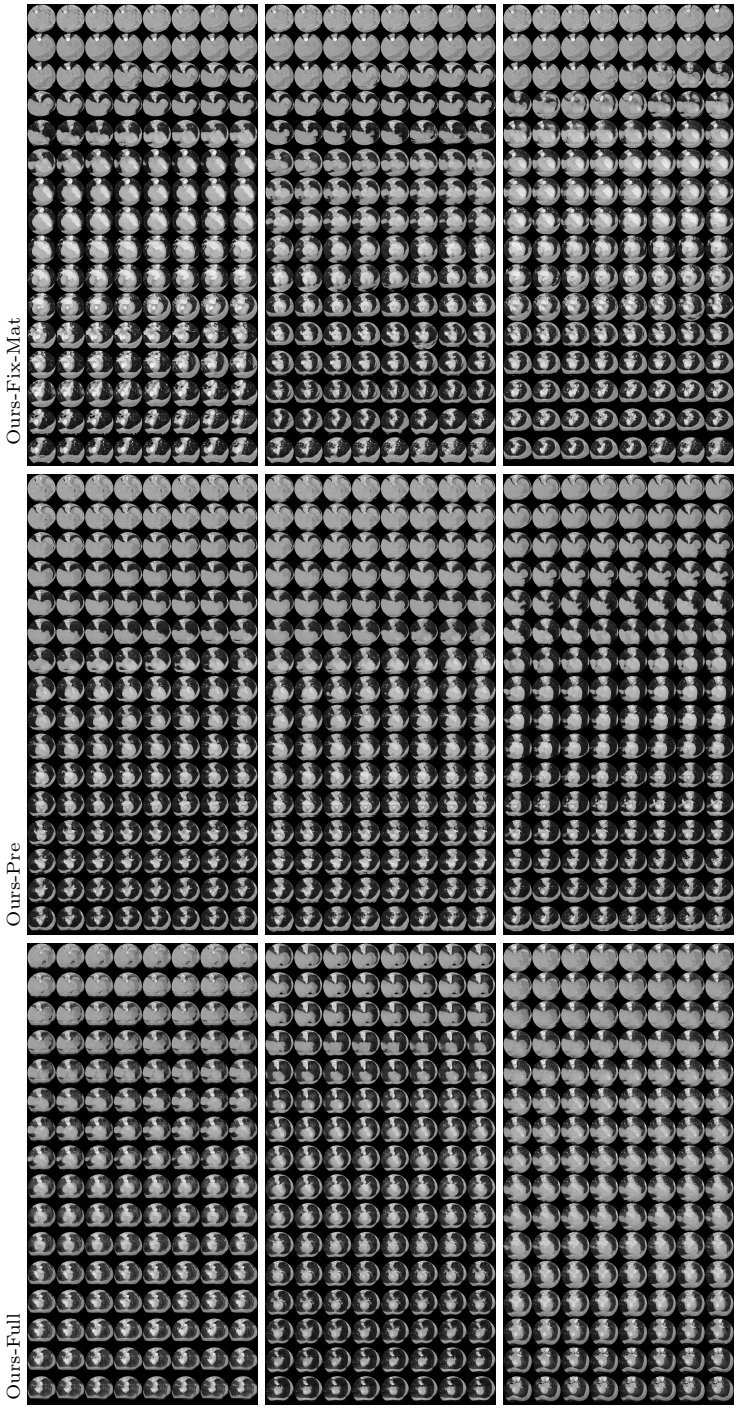

Fig. 2. Qualitative Results: Three random samples(Full Volume) from our different methods. Ours-Pre has better image quality and slice consistency comparing to Ours-Fix-Mat. Ours-Full has more diverse object shape and background comparing to the other two methods.

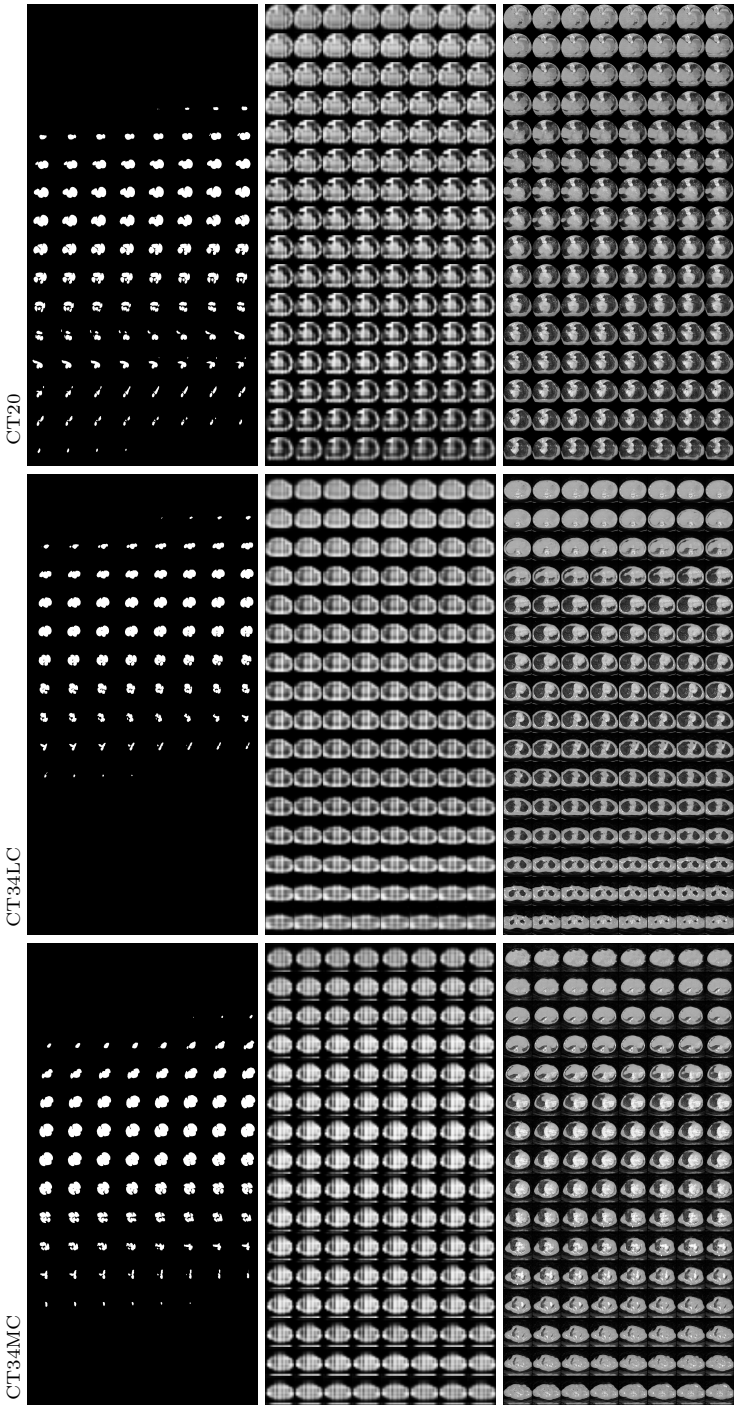

Fig. 3. Qualitative Results: Three samples generated by our method in different datasets. The first column is the generated shape, the second column is the generated material, the last column is the generated image. Note how the shape and material are consistent and correlated.

References

1. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
2. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics **21**(4), 163–169 (1987)
3. Odena, A., Dumoulin, V., Olah, C.: Deconvolution and checkerboard artifacts. Distill (2016). https://doi.org/10.23915/distill.00003, http://distill.pub/2016/deconv-checkerboard
4. Wu, J., Zhang, C., Xue, T., Freeman, B., Tenenbaum, J.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: Advances in neural information processing systems. pp. 82–90 (2016)
5. Zhuang, X., Shen, J.: Multi-scale patch and multi-modality atlases for whole heart segmentation of mri. Medical image analysis **31**, 77–87 (2016)