Flow-based Generative Models for Learning Manifold to Manifold Mappings (Supplemental Material)

In the main paper, we showed some quantitative results of generating ODF conditioned on DTI. In this supplement, we present several additional quantitative results as well as an example of the heatmap of the p-value of the generated ODF compared with other measurements including FA, DTI, and ODF. Also, we provide some discussion regarding the motivation of the layers we choose and more details about the preliminary differential geometry concepts so that the presentation is more self-contained for the reader.

Quantitative evaluation for generating ODF from DTI

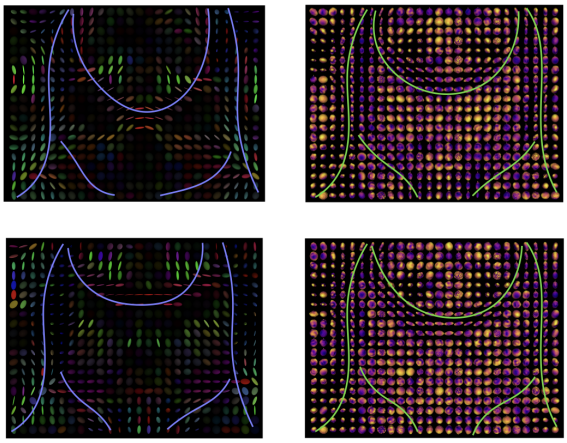

Figure 1: The samples of some fibers in the original DTI slice and in the generated ODF slice. Our model can generate the ODF structures.

In this section, we will show additional results of our model for ODF generation. As mentioned in the paper, the main focus for our paper is to generate ODF images based on DTI images. Note that in terms of acquisition time, a typical scan that may provide ODF data, will take 35 mins compared to 7–12 mins for a typical DTI acquisition. Thus, successfully using our model to generate the ODF from DTI, e.g., for statistical analysis on groups, can facilitate a nice trade-off

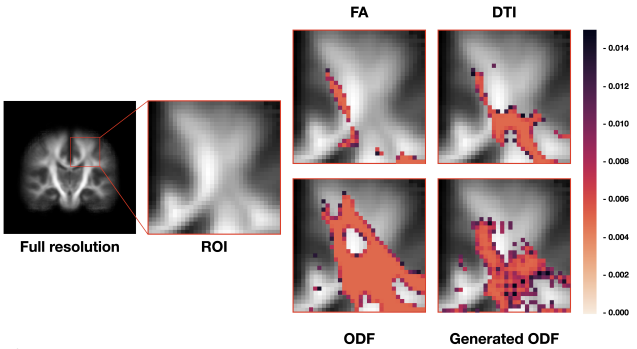

Figure 2: The p-value of one of the ROIs of the entire brain scan with full-resolution. We show that our proposed method can generate meaningful ODF with respect to the group level differences.

between the angular resolution, sensitive of statistical analysis and the actual scanning time. A few examples of our results were included in the main paper Fig. 6(a). Here, we show some additional examples of DTI to ODF generation. In Fig. 3, each "pair" includes the input DTI (first and third lines) and the generated/synthetic ODF (second and fourth lines). We can see that the structure of the generated ODF is well preserved with our model. Similar to the main paper, we only show 2D slices of the 3D images.

Also, for the reader not familiar with diffusion weighted MR images, we highlight the white matter tracts in Fig. 1. The white matter bundles are the "path" or "roads/highways" connecting different gray matter regions (processing centers) of the brain. In our generation process, the white matter bundles are the major Region of Interests (ROIs) that we seek to reconstruct, since they contain information pertinent to assessing brain connectivity, which may be different across clinically disparate groups. As an example, one might see group-level differences in brain connectivity between male and female or alternatively, Alzheimer Disease (AD) subjects and healthy controls. Since the HCP dataset that we use includes participants that are all healthy, perhaps a scientifically more interesting experiment of evaluating group-level connectivity differences between healthy and diseased subjects, cannot be performed, and we used male versus female connectivity analysis for evaluations. For statistical testing between male and female, we presented the IoU of the entire brain with low-resolution in the main paper. Here, we show a

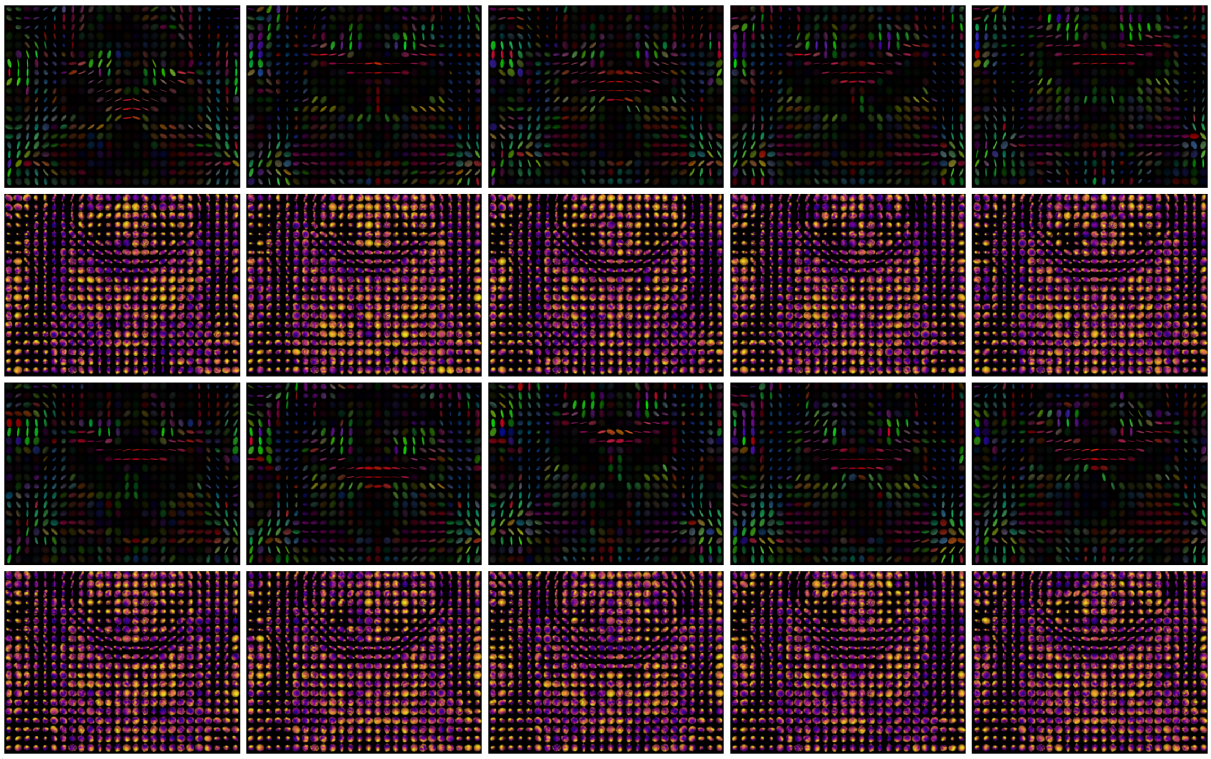

Figure 3: Generated ODF from corresponding DTI. Each pair here contains the input DTI *(top)* and the generated ODF *(bottom)*.

zoomed in example of a ROI for the full-resolution images in Fig. 2. The p-values for different ROIs are all < 0.001 in both the original ODF and our generated ODF, indicating consistency of our results, at least in terms of regions identified in downstream statistical analysis. Note that the analysis on the real ODF images serves as the ground truth.

Other potential applications for our model

Since our model basically requires two parallel GLOW models (two-stream GLOW) on two manifolds, our model can be further extended or applied to other applications. We will introduce some additional aspects of our model in this supplement as illustrative examples, noting that these secondary experiments are only to show applicability rather than claiming state of the art performance.

Generate DTI from ODF

In the main paper, we only discussed generating ODF based on DTI data, because normally this is the direction which is meaningful in the real-world acquisitions, i.e., in other words, data from shorter acquisition time (DTI) providing images comparable to those acquired from a longer acquisition time (ODF). But our model imposes no constraint that the direction can only proceed in one direction. If we reverse the direction, we can generate the corresponding DTI based on ODF data. Even though ODF contains more angular information than DTI, the standard analysis is based on functional anisotropy (FA) (scalar-valued) summary images, which is calculated from DTI directly.

Suppose $\lambda_1, \lambda_2, \lambda_3$ are three eigen values of the SPD matrix at a specific voxel.

$$\mathrm{FA} = \sqrt{\frac{1}{2}} \frac{\sqrt{(\lambda_1 - \lambda_2)^2 + (\lambda_2 - \lambda_3)^2 + (\lambda_3 - \lambda_1)^2}}{\sqrt{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}}$$

An image consisting of FA values at each voxel can be constructed based on the ODF data.

In Fig. 4, we show some results for generating DTI based on ODF. Each pair contains the input ODF and the corresponding generated DTI. The colors of DTI indicates the eigenvalues of the SPD matrix at each voxel, and the orientation indicates the eigenvectors of SPD. Thus, qualitatively, the generated DTI preserves the DTI field structure fairly well.

Model setup: generating DTI based on ODF Similar to the model described in the main paper, the generation of DTI based on ODF contains two flow-based generation model and a transformation from latent space of ODF to the latent space of DTI. Two generation models remain identical while the transformation model is from ODF to DTI.

Motivation of the three basic layers

In the main paper, we introduced three basic layers: Actnorm, 1×1 convolution, and the Affine Coupling. Here, we will provide additional motivation behind these three layers. We will also include more details about the full structure of the GLOW model and the NanoFlow (Lee, Kim, and Yoon 2020) trick.

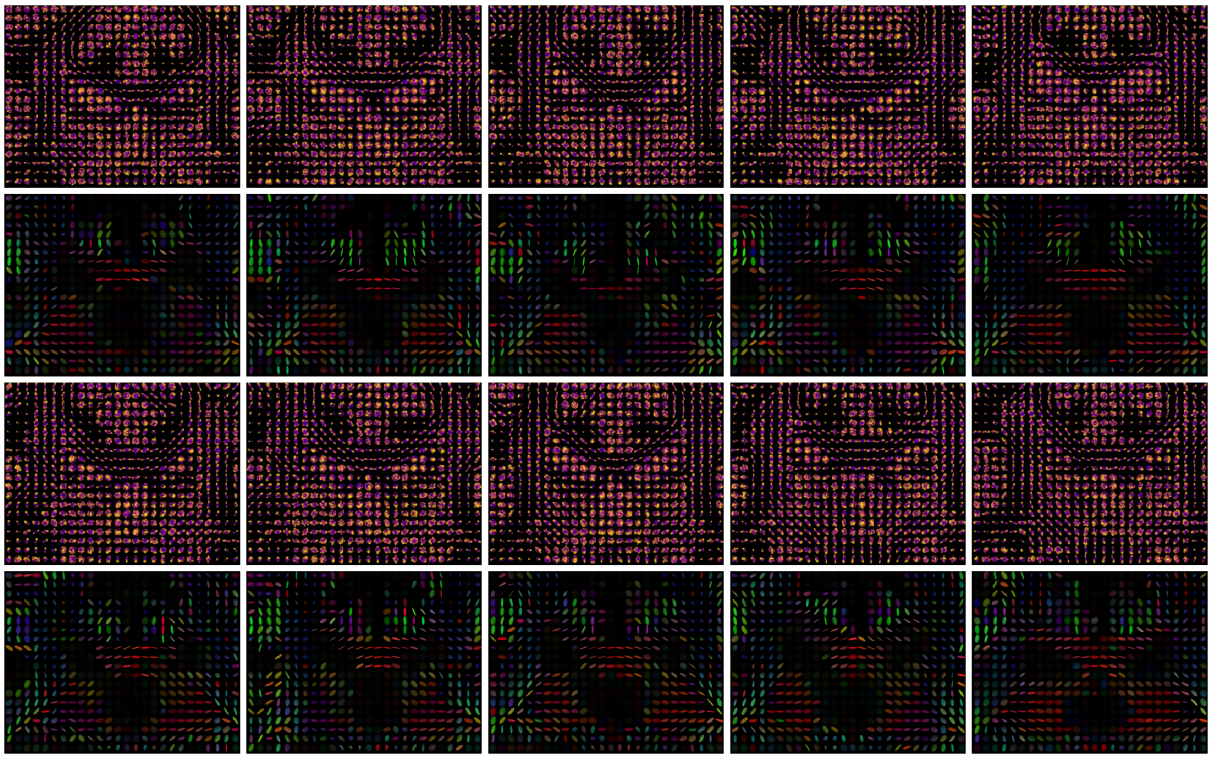

Figure 4: Generated DTI from corresponding ODF. Each pair here contains the input ODF *(top)* and the generated DTI *(bottom)*.

Actnorm

The Actnorm layer plays a similar role as Batch Normalization (Ioffe and Szegedy 2015). In a traditional neural network, e.g., Resnet (He et al. 2016), batch normalization is used to stabilize the training procedure and make the distribution of the input to be simpler. However, in a normalizing-flow based architecture, one of the biggest drawbacks of the batch normalization is that it is not invertible. Batch normalization depends on the data within one mini-batch, which is not available in the generating phase (inverse of the model). Thus, we need to modify this operation to be invertible and preserve stability in training. Thus, in (Kingma and Dhariwal 2018), the authors introduced the actnorm in Euclidean space, which is initialized with the data and then trained unconstrained. The initialization keeps the input close to the normal distribution. After the initialization, the parameters of the mean and variance are updated based on the gradient of the loss function. In our case, the target field is on the manifold. Thus, we use our modification of "adding" and "multiplication" operations on the manifold valued data to extend the Actnorm layer to the manifold setting. Mathematically, the order of the two operations "adding" and "multiplication" does not matter. But practically, applying "multiplication" first would reduce the cost to compute the gradient and thus is faster.

1×1 convolution

The 1×1 invertible convolution is one of the main contributions in GLOW (Kingma and Dhariwal 2018). The 1×1 convolution is used to permute the information, channel-wise, in the data with learnable weights. The rationale behind this layer comes from (Dinh, Sohl-Dickstein, and Bengio 2016) which used a fixed permutation to reverse the ordering of the channels to make each channel contribute equally. In the manifold-valued data, we can define the permutation after the chart map which projects the original manifold-valued data on to the Euclidean space. To make training more robust without exploding/ vanishing as well as make it easier to compute the determine of Jacobian matrix, we impose an extra constraint that the kernel of 1×1 convolution is a rotation matrix. Thus, the determinant of the Jacobian is ± 1 which is a constant number.

Affine Coupling

As the reader will notice, both operations we discussed above are linear without any activation functions. Indeed, we can build a linear neural network to be invertible easily (for example, a full-rank matrix multiplication). However, we will use the non-linearity to fit the distribution of the complicated data. In (Dinh, Krueger, and Bengio 2014), motivated by Advanced Encryption Standard (AES), the author introduced an invertible nonlinear layer called Affine Coupling. The goal of the Affine Coupling layer is to use half of the pixels of the data to estimate the mean and the variance of other half of the pixels (in the Euclidean space) and by adding and multiplication to make it more like a normal distribution. The first half of the pixels are preserved through this layer so that when inverting, the network can use the same nonlinear function to estimate the mean and variance again. In the next layer, two parts of the pixels are switched to iteratively make the final distribution of *all* pixels to be close to normal distribution. The

key functionality of this layer involves adding, multiplication, and the neural network to estimate the mean and variance. In the manifold setting, we use the chart map to project the first half of pixels into the Euclidean space to apply a simpler neural network (compared with the available neural network defined on manifold) to estimate the mean and variance of the other half. Also, we apply our modification of adding and multiplication on the manifold-valued data so that the final output will be similar as the normal distribution defined on the manifold.

The full structure of the network

In GLOW (Kingma and Dhariwal 2018), the authors designed a multi-scale architecture which includes split and squeeze functions. The basic idea of the split function is similar to Wavelets that divides a given function or continuous-time signal into different scale components. We call the combination of these three layers mentioned above as a "block". After every few blocks, the model will split the channels of the data into two pieces equally, and directly pass one of these two pieces into the latent space. So, the model will learn to first pay more attention to the overall low-frequency distribution to make it a normal distribution. After splitting, the model will try to learn the distribution of the difference (the remaining part). The squeeze function is to reduce the spatial resolution and put it to the channel-wise to enable the next splitting. For example, for the natural images case, the input of the data will be $128 \times 128 \times 3$. In the first layer, we will squeeze the spatial resolution to the channel. Thus, it will become $(128/2) \times (128/2) \times (3 \times 2^2) = 64 \times 64 \times 12$. After this block, the output will also have the same size as the input, which is $64 \times 64 \times 12$. At this stage, we split the data into two pieces and directly pass $64 \times 64 \times (12/2) = 64 \times 64 \times 6$ into the latent space. The remaining $64 \times 64 \times 6$ will again be squeezed into $(64/2) \times (64/2) \times (6 \times 2^2) = 32 \times 32 \times 24$ and pass it into the next block. In our case, we use exactly the same idea as the split and squeeze operations. The major difference is that each pixel is a point on the manifold which may have different dimensions, 361 for ODF and 6 for DTI. The channel of each manifold-valued data is always 1 in the input layer.

NanoFlow

NanoFlow (Lee, Kim, and Yoon 2020) was introduced to reduce the number of parameters for sequential data processing. The assumption of NanoFlow is that the Affine Coupling layer, if fully trained, can estimate the distribution for any parts of the input data in a fixed order. There will be some performance drop compared with training different Affine Coupling layers for different parts of the data. But the gain from reducing the parameters is significant. Thus, in our setup, due to the large 3D input, we apply the NanoFlow trick for DTI and ODF separately. For example, for the DTI path, we first split the data into $2n$ parts called $\{x_1, x_2, ..., x_{2n}\}, n \geq 1$. The input of two neural networks in Affine Coupling layer would be the $x_{2k-1}, k = 1, 2, ..., n$, while the output will be the mean and variance of $x_{2k}, k = 1, 2, ..., n$ respectively. The pixels to be fixed when passing through the Affine Coupling layer will be $1/2$ of the total pixels. But when $n > 1$ the input of the neural network will be smaller $(1/2n)$, which can in turn reduce the complexity of the neural network to save the number of parameters needed. A special case will be $n = 1$ which is the original Affine Coupling layer introduced in GLOW.

Some more discussion about the differential geometry background

Given a Riemannian manifold $\mathcal{M}$ with dimension m, it can be equipped with an (countable) indexed tuple, denoted as *chart*. Each chart, denoted by the tuple (U, Φ) where $U \subset \mathcal{M}$ and $\Phi : U \to \mathbf{R}^m$ is a diffeomorphism (an invertible differentiable mapping). Furthermore, the set of charts, $\{U_\alpha, \Phi_\alpha\}_{\alpha \in I}$ covers $\mathcal{M}$, i.e., $\mathcal{M} \subset \cup_{\alpha \in I} U_\alpha$. The collection of charts that cover $\mathcal{M}$ is called an *atlas*.

A chart (or atlas) is required to define differentiable operations on $\mathcal{M}$, as it alleviates the non-linearity of $\mathcal{M}$ by defining locally-linear structure given by ΦU. The choice of chart map for the given Riemannian manifold is not unique. As long as the chosen tuple (U, Φ) has the covering property and diffeomirphic property as mentioned before, it forms an well-defined chart map and hence an atlas.

Observe that using chart map, we can get a local coordinate system as given by $\Phi : U \to \mathbf{R}^m$. This raises an important question *does a local parametrization of $\mathcal{M}$ give a chart?* The answer is yes as long as the parameterization is a diffeomorphism as by definition of parametrization, it covers the space $\mathcal{M}$. For manifold of SPD matrices, Cholesky decomposition is a choice of parametrization and is used in our work as the choice of chart map.

Another choice of chart map is using Riemannian exponential and inverse exponential/log map. Given $p \in \mathcal{M}$, with the tangent space $T_p\mathcal{M}$, the Riemannian exponential map is defined as the diffeomorphic mapping: $\mathsf{Exp}_p : V \subset T_p\mathcal{M} \to U \subset \mathcal{M}$, here $U = \mathsf{Exp}(V)$ and hence, is also a well-defined choice for chart maps. Note that the inverse of exponential map is called Riemannian log map.

In general for a choice of atlas (an the corresponding chart maps), one may look for the *minimal atlas*, i.e., one with smaller cardinality index set, I. But in this work, we are not looking for using a minimal atlas as finding such is a computationally hard problem. Below, we provide some examples of charts used in the main paper.

For example, for the Earth without height, there are several different methods can be used to describe the location. The most acceptable one is by the Latitude and the Longitude. However, we could also use (x, y, z) where $x^2 + y^2 + z^2 = 1$ to describe the location. By splitting the Earth into two halves, we can use x, y to describe the location given the north part or the south part. All these choices are not Log-map, but can uniquely describe one point on the sphere. If we consider the tangent space at the north pole, we can always use a straight line connecting the south pole and the point on the sphere which passes through the tangent space at the north pole uniquely. Thus, we can also use the Log map and the Exp map of the north pole to transfer the points between the sphere and the tangent space uniquely.

We have shown several examples of the choice of the chart

map in the main paper in Table. 3. The rationale behind the choice is to choose a map that can uniquely map the points and be as simple as possible. Thus, for the SPD matrices, we use the Cholesky decomposition to project the original SPD matrix into the Euclidean space uniquely. As the DTI lying on $\mathsf{SPD}(3)$, the Cholesky decomposition can be computed explicitly.

$$\begin{aligned}
&\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{12} & x_{22} & x_{23} \\ x_{13} & x_{23} & x_{33} \end{bmatrix} \\
&= \begin{bmatrix} a_1 & 0 & 0 \\ b_1 & a_2 & 0 \\ b_2 & b_3 & a_3 \end{bmatrix} \times \begin{bmatrix} a_1 & b_1 & b_2 \\ 0 & a_2 & b_3 \\ 0 & 0 & a_3 \end{bmatrix} \\
&= \begin{bmatrix} a_1^2 & a_1 b_1 & a_1 b_2 \\ a_1 b_1 & b_1^2 + a_2^2 & b_1 b_2 + a_2 b_3 \\ a_1 b_2 & b_1 b_2 + a_2 b_3 & b_2^2 + b_3^2 + a_3^2 \end{bmatrix} \qquad (1)
\end{aligned}$$

Thus, one can solve all the components of the Cholesky decomposition.

$$\begin{aligned}
a_1 &= \sqrt{x_{11}} & b_1 &= x_{12}/a_1 \\
b_2 &= x_{13}/a_1 & a_2 &= \sqrt{x_{22} - b_1^2} \\
b_3 &= (x_{23} - b_1 b_2)/a_2 & a_3 &= \sqrt{x_{33} - b_2^2 - b_3^2}
\end{aligned}$$

Since the computation is simple and unique (when the matrix is full rank), we finally choose the Cholesky decomposition as the chart map for $\mathsf{SPD}(3)$. Similar argument holds for $\mathbf{S}^n$ and $\mathbf{R}_+$.

One more thing to notice is that the choice of the tangent space T_p depends on the point p on the manifold. Thus, when using the local chart map/ Log map, we will need to pass the point p along the neural network to enable invertibility. Thus, in this paper, we normally choose the global chart map that can uniquely map any point on the manifold into the Euclidean space. For those Riemmannian manifolds without global chart maps, the architecture will be far more complicated and a problem of independent interest that we do not address in this work.

Final remarks: Why making a simple Euclidean model work is extremely challenging

A natural question a reader may ask is why we cannot embed the manifold valued data into the Euclidean space and simply apply the traditional GLOW model. In other words, what is the rationale of deriving such a model based on differential geometric principles? This is a valid question and the explanation below describes why this approach allows practical implementation when it will not even be possible otherwise.

As we briefly mentioned in the main paper, the Euclidean simplification will typically require a *much larger* memory (almost infeasible) to store and compute the model. For example, in the DTI to ODF example, if we directly embed two data sources into the Euclidean space, we will have $\mathbf{R}^6$ and $\mathbf{R}^{361}$ respectively. Interestingly, notice that in the traditional GLOW model, due to the use of "squeeze" after every block, the number of channels at least doubles between layers. In the traditional GLOW paper, which only deals with the color images with channel 3, this setup already requires a huge amount of space but is at least feasible. However, things get much worse when we consider the ODF data, which has 361 channels: this is $120\times$ more than the color images. Thus, in *each* layer, the number of parameters will be $\{120\times\}^2 = 14400\times$ more than the traditional GLOW model! This requirement is not even feasible to run, even on some large computer clusters that we tried. On the other hand, if we simplify the model to contain fewer layers, the expressive power of the model drops to a level that it is not able to converge.

Further, with the Euclidean setup, there is no geometric structure on the pixel values. Thus, important hard-constraints will be violated. For example, the sum of the measurements at a ODF pixel is 1 and every value should be non-negative. Also, the pixel of DTI should be a semi-positive definite matrix. Without these constraints, the output will not be scientifically meaningful or interpretable.

We note that contemporary with our work, a paper recently proposed $\mathcal{M}_e$-flow (Brehmer and Cranmer 2020) which also argues considering the structure of the manifold. They treat the structure of the manifold as a "surface" in the Euclidean space. So the authors propose training an encoder-decoder network to project the sparse surface in the high-dimensional space into a dense low-dimensional space. Every operator in the low-dimensional space is similar to GLOW in the Euclidean space. This idea is effective for natural images and works well. But for **known** Riemannian manifolds, we already know the dimension of the manifold. Thus, there might be little space for the encoder to lower the dimension of the data unless one is willing to compromise and accept the loss of some details: this makes the final data more blurry, a phenomena we see in the Texture images showns in the main paper. Otherwise, since the $\mathcal{M}_e$-flow is even more general than GLOW, the dimension of the hidden space is even higher than the GLOW in the Euclidean space.

But in our setup, as the manifold constraint is inherent in the construction of the model, if we treat every pixel/ voxel to be manifold valued, the input channel will be 1. Thus, this will significantly reduce the number of the parameters. Since the manifold structure is "hard coded" in the neural network, we can train the model to converge better with sensible resource requirements, as we show in our main paper.

References

Brehmer, J.; and Cranmer, K. 2020. Flows for simultaneous manifold learning and density estimation. *arXiv preprint arXiv:2003.13913* .

Dinh, L.; Krueger, D.; and Bengio, Y. 2014. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516* .

Dinh, L.; Sohl-Dickstein, J.; and Bengio, S. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* .

Kingma, D. P.; and Dhariwal, P. 2018. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, 10215–10224.

Lee, S.-g.; Kim, S.; and Yoon, S. 2020. NanoFlow: Scalable Normalizing Flows with Sublinear Parameter Complexity. *arXiv preprint arXiv:2006.06280* .