# Supplementary Material for "Deep Learning for Breast MRI Style Transfer with Limited Training Data"

## Appendix A Data

In this section, we will give a more detailed description of our dataset and the process of its creation, beyond that of Section 1.1.

In this study, we experimented with the Breast Cancer DCE-MR (Dynamic Contrast-Enhanced Magnetic Resonance) dataset of [20], which includes MRI scans of 922 breast cancer patients over the course of a decade. From this, we include GE Healthcare MRI scans from 628 subjects, and each scan volume consists of more than 160 physically adjacent 2D axial image slices. To focus on the salient parts of the volumes and to avoid redundancies in our dataset, for each volume we selected a single slice from the middle 50% of each patient volume used in our study to be used in our final dataset.

All images have a $512 \times 512$ pixel resolution, and were pre-processed by assigning the top 1% of pixel values in the entire dataset to a value of 255, followed by linearly scaling the remaining pixel intensities to the 0-255 range. The data was randomly divided at the patient level, and 528 datapoints were used to produce the training set. Out of the remaining images, 50 were kept as a test set, and the other 50 were used for validation. 25 images from a Siemens scanner were created the same way, to be used in Section 2.2.

This data is publicly available on the Cancer Imaging Archive (TCIA), at https://doi.org/10.7937/TCIA.e3sv-re93.

## Appendix B Image Transformation Functions/Training Styles

Here we give the explicit formulae for the parametrically-randomized image transformation functions used to simulate style in training. Note that after a randomized transformation function is applied to a training image, the image is normalized to fall within the pixel range of 0-255.

***The Linear Transformation***

The simplest intensity transfer function/transformation is linear with respect to input pixel intensity $I_{\text{in}}$ with some slope $m_{\text{lin}}$ (contrast change) and intercept $b_{\text{lin}}$ (brightness offset), with output pixel intensity $I_{\text{out}}$. We "randomize" this function during training with

$$I_{\text{out}} = T_{\text{lin}}(I_{\text{in}}) = m_{\text{lin}} I_{\text{in}} + b_{\text{lin}}, \tag{B1}$$

where given the continuous uniform distribution $\mathcal{U}(\cdot, \cdot)$, $m_{\text{lin}} = \tan\theta_{\text{lin}}$, $\theta_{\text{lin}} \sim \mathcal{U}(\pi/8, 3\pi/8)$ and $b_{\text{lin}} \sim \mathcal{U}(-20, 20)$ are sampled at each new iteration of training when the linear transformation is chosen.

***The Negative Transformation***

The negative of an image is produced by subtracting each pixel from the maximum possible intensity value $I_{\max}$, e.g. for an 8-bit image $I_{\max} = 2^8 - 1 = 255$. As such, the transformation function $T_{\text{neg}}$ used to produce an image negative can be written pixel-wise as $I_{\text{out}} = T_{\text{neg}}(I_{\text{in}}) = I_{\max} - I_{\text{in}}$, where $I_{\max}$ is the maximum pixel intensity value for the image, and $I_{\text{out}}$ and $I_{\text{in}} \geq 0$ are the output and input pixel intensity values, respectively. We "randomize" this function during training with

$$I_{\text{out}} = T_{\text{neg}}(I_{\text{in}}) = m_{\text{neg}} I_{\text{in}} + b_{\text{neg}}, \tag{B2}$$

with $m_{\text{neg}} = \tan\theta_{\text{neg}}$, $\theta_{\text{neg}} \sim \mathcal{U}(-3\pi/8, -\pi/8)$ and $b_{\text{neg}} \sim \mathcal{U}(235, 275)$.

***The Log Transformation***

The log (logarithm) transformation of an image can be expressed as a pixel-wise transformation function $T_{\log}$ as $I_{\text{out}} = T_{\log}(I_{\text{in}}) = c_{\log}\log(1 + I_{\text{in}})$, where $c_{\log}$ is a scaling constant given by $c_{\log} = I_{\max}/[\log(1 + I_{\max,\text{img}})]$, with $I_{\max,\text{img}}$ being the maximum pixel value in the given image. This factor is chosen to ensure that the range of output pixel values does not exceed $I_{\max}$. In practice the log transformation is used to map a narrow band of low-intensity input values to a wide range of output intensities. We randomize this transformation via

$$I_{\text{out}} = T_{\log}(I_{\text{in}}) = \tilde{c}_{\log}\log(1 + I_{\text{in}}), \tag{B3}$$

where $\tilde{c}_{\log} = a c_{\log}$ with $a \sim \mathcal{U}(0.7, 1.3)$.

***The Power-Law (Gamma) Transformation***

The power-law (gamma) transformation can be mathematically expressed as a pixel-wise transformation function with $I_{\text{out}} = T_{\text{pow}}(I_{\text{in}}) = c\,(I_{\text{in}}/I_{\max})^{\gamma}$, where similar to $c_{\log}$, $c_{\text{pow}} = I_{\max}$ is a scaling constant chosen to give the correct range of possible $I_{\text{out}}$ values. The exponential parameter $\gamma$ can be any chosen as any $\gamma > 0$. This transformation is used for gamma correction, a method important for the correct displaying of images on digital screens. This transformation is randomized with the scheme

$$I_{\text{out}} = T_{\text{pow}}(I_{\text{in}}) = c\left(\frac{I_{\text{in}}}{I_{\max}}\right)^{\tilde{\gamma}}, \tag{B4}$$

where $\tilde{\gamma} = 2^{\alpha}$ with $\alpha \sim \mathcal{U}(-5, 5)$.

***The Piecewise-Linear Transformation***

Piecewise-linear transformation functions are constructed by conjoining different linear functions that have disjoint intensity ranges. A common application of these transformation functions is for *contrast stretching*, a method for

expanding an image's intensity values to span the full possible range of intensities. Here, we use a three segment piecewise-linear transformation function defined "randomly" according to

$$\begin{aligned} I_{\text{out}} &= T_{\text{pw}}(I_{\text{in}}) \\ &= \begin{cases} \dfrac{s_1}{r_1} I_{\text{in}} & \text{if } I_{\text{in}} \in [0, r_1] \\ \dfrac{s_2 - s_1}{r_2 - r_1}(I_{\text{in}} - r_1) + s_1 & \text{if } I_{\text{in}} \in [r_1, r_2] \\ \dfrac{I_{\max} - s_2}{I_{\max} - r_2}(I_{\text{in}} - r_2) + s_2 & \text{if } I_{\text{in}} \in [r_2, I_{\max}], \end{cases} \end{aligned} \tag{B5}$$

where $r_1 < r_2, s_1 < s_2$ are parameters chosen by the scheme

$$\begin{aligned} r_1 &\sim \mathcal{U}(55, 95) \qquad & r_2 &\sim \mathcal{U}(130, 170) \\ s_1 &\sim \mathcal{U}(35, 75) \qquad & s_2 &\sim \mathcal{U}(205, 245). \end{aligned} \tag{B6}$$

***The Sobel X and Y Transformations***

The Sobel X and Y transformations, or operators, are edge-detection image transformations defined by applying $3 \times 3$ convolutional kernels to the input image. The X and Y kernel matrices are defined to approximate the derivatives that quantify horizontal and vertical changes of input pixel intensity values, respectively. The X and Y operators are as such defined respectively with

$$\begin{aligned} \mathbf{I}_{\text{out},x} &= \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \mathbf{I}_{\text{in}} \quad \text{and} \\ \mathbf{I}_{\text{out},y} &= \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{I}_{\text{in}}, \end{aligned} \tag{B7}$$

where $*$ indicates the linear convolution operation, $\mathbf{I}_{\text{in}}$ is the input image defined as a matrix of pixel intensity values, and $\mathbf{I}_{\text{out},x}, \mathbf{I}_{\text{out},y}$ are the output images of the X and Y transformations, respectively.

# Appendix C Full Expansion of Cross-Domain Reconstruction Triplet Loss (Equation (4))

$$
\begin{aligned}
\mathcal{L}_{\text{cross}} = & \mathbb{E}\left[\left\|G\left(E^c(X_2), E^s(X_1)\right) - X_2\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(X_1), E^s(X_2)\right) - X_1\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(X_2), E^s(T(X_1))\right) - T(X_2)\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(X_1), E^s(T(X_2))\right) - T(X_1)\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(X_2), E^s(T(X_2))\right) - T(X_2)\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(X_1), E^s(T(X_1))\right) - T(X_1)\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(T(X_2)), E^s(X_1)\right) - X_2\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(T(X_1)), E^s(X_2)\right) - X_1\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(T(X_2)), E^s(X_2)\right) - X_2\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(T(X_1)), E^s(X_1)\right) - X_1\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(T(X_2)), E^s(T(X_1))\right) - T(X_2)\right\|_1\right] \\
& + \mathbb{E}\left[\left\|G\left(E^c(T(X_1)), E^s(T(X_2))\right) - T(X_1)\right\|_1\right].
\end{aligned}
\tag{C8}
$$

# Appendix D Exploring Style Encoding Mechanics

In the previous two sections we explored two different applications of StyleMapper. In this section, we will take a different approach, and explore how style encoding can differ between styles seen and unseen in training, and between images of different styles and of the same style. To begin, in this section we will use "fixed" versions of the randomized training style transformations $T_i$ described in Section 1.2.1, for purposes of consistency and comparability of results. In particular, the randomized parameters will be fixed to the means of their respective sampling distributions: the linear function will be fixed to identity, via $m_{\text{lin}} = 1, b_{\text{lin}} = 0$. The negative transformation will be fixed to $m_{\text{neg}} = -1, b_{\text{neg}} = 255 = I_{\max}$; log: $a = 1$; power-law/gamma: $\tilde{\gamma} = 0.5$; piecewise-linear: $r_1 = 75, r_2 = 150, s_1 = 55, s_2 = 225$. The Sobel X and Y transformations are unchanged.

We will begin by comparing the style codes that the trained style encoder extracts from images of different styles. In particular, we take a test set $\{X_{\text{target}}\}$ of 25 raw DCE-MR images, which we can then apply one of the aforementioned transformations $T_i$ to, in order to obtain the corresponding set of transformed images $\{T_i(X_{\text{target}})\}$. Finally, from here we can input each of these transformed images to the trained style encoder $E^s$ to obtain the set of style codes for each these images, $\{s^{T_i}_{\text{target}} = E^s(T_i(X_{\text{target}}))\}$.

Now, we can estimate the similarity of two style code vectors $s_1, s_2 \in \mathbb{R}^8$ with a cosine-similarity/normalized inner product

$$\text{sim}(s_1, s_2) = \frac{s_1^T s_2}{s_{12} s_{22}}, \tag{D9}$$

where $\text{sim}(s_1, s_2) \in [-1, 1]\ \forall s_1, s_2$. In order to examine how the style encoder encodes various styles differently, we compare codes extracted from images of different styles but, with the same content. We do this by computing $\text{sim}(s^{T_i}_{\text{target},k}, s^{T_j}_{\text{target},k})$ for each image $X_{\text{target},k}$ in the first half of the test set ($k = 1, \ldots, 25$), where the pair of style codes $s^{T_i}_{\text{target},k}, s^{T_j}_{\text{target},k}$ are obtained from applying each pair of different transformations $T_i, T_j$ to $X_{\text{target},k}$. We then average the similarities over all of the images in the test set for each pair of transformations; the results of this are shown in Figure D1.

| | negative | log | gamma | p.w. linear | Sobel X | Sobel Y | identity |
|---|---|---|---|---|---|---|---|
| negative | 1.000 | | | | | | |
| log | -0.043 | 1.000 | | | | | |
| gamma | -0.480 | 0.884 | 1.000 | | | | |
| p.w. linear | -0.773 | 0.607 | 0.894 | 1.000 | | | |
| Sobel X | -0.039 | 0.528 | 0.512 | 0.394 | 1.000 | | |
| Sobel Y | 0.017 | 0.740 | 0.676 | 0.473 | 0.932 | 1.000 | |
| identity | -0.780 | 0.621 | 0.910 | 0.992 | 0.367 | 0.461 | 1.000 |

**Fig. D1 Comparing style codes extracted for different styles.** Entries are cosine-similarities (Equation (D9)) of style codes extracted from pairs of images of the same content but different styles, i.e. two different image transformations applied to the same test image, averaged over the entire test set. The rows and columns specify which pair of transformations/styles that the encoded images originated from. This figure is recommended to be viewed in color.

As we saw in Section 2.1, the ability of a trained style encoder to learn the style code of a given target style is mostly independent of the number of images $N_{\text{target}}$ of this target style seen by the encoder; we can see this numerically by observing the distribution of style codes obtained from applying a style encoder on a distribution of images of a certain single style, as follows. In particular, for the set of style codes extracted from images transformed with one of the transformations $T_i$, $\{s^{T_i}_{\text{target}}\}$, we can estimate how similar style codes corresponding to this transformation/style are to each other, by averaging over the similarities between all possible pairs of different style codes from $\{s^{T_i}_{\text{target}}\}$.

These results are given for each of the examined styles in Table D1. On average, style codes of different images of the same style barely differ, indicated by the average similarities all being very close to unity, and the small standard deviations thereof. In other words, the style encoder is very consistent with what code it assigns to a particular style. This is true even for styles not seen in training: for a style encoder trained on all transformations *except* for

**Table D1** Average and standard deviation of cosine similarity (Equation (D9)) of all pairs of style codes extracted from images of the same style.

| Style | Avg. similarity | std. deviation |
|---|---|---|
| Negative | 0.995 | 0.006 |
| Log | 0.974 | 0.037 |
| Gamma/power-law | 0.984 | 0.023 |
| Piecewise-linear | 0.979 | 0.028 |
| Sobel X | 0.989 | 0.020 |
| Sobel Y | 0.990 | 0.016 |
| Identity | 0.990 | 0.013 |

the power-law transformation, the style codes extracted from power-law-style test images have an average paired similarity of 0.985 with standard deviation 0.023, very close to the corresponding result for the model trained on power-law styles in Table D1.

# Appendix E StyleMapper Network Architecture

Refer to Figure. 2 for a diagram of our model; the explicit layer-by-layer architecture is given in Tables E2, E3 and E4. Here, `Conv2d(input_channels, output_channels, kernel_size, stride, padding)` denotes a standard two-dimensional convolutional layer with some number of input channels, output channels, kernel size, stride, and padding. Reflection padding and ReLU activations are used unless otherwise stated. After each convolution, the content encoder uses instance normalization (IN) layers, the decoder uses adaptive instance normalization (AdaIN) layers, and the style encoder does not use normalization layers.

**Table E2** Style Encoder $E^s$ architecture.

| | |
|---|---|
| `input image (1x256x256)` | ⇒`Conv2d(1, 64, 7, 1, 3)` |
| | ⇒`Conv2d(64, 128, 4, 2, 1)` |
| | ⇒`Conv2d(128, 256, 4, 2, 1)` |
| | ⇒ 2×`Conv2d(256, 256, 4, 2, 1)` |
| | ⇒`GlobalAvgPool()` |
| | ⇒`Conv2d(256, 8, 1, 1, 0)` |

**Table E3** Content Encoder $E^c$ architecture.

| | |
|---|---|
| `input image (1x256x256)` | ⇒`Conv2d(1, 64, 7, 1, 3)` |
| | ⇒`Conv2d(64, 128, 4, 2, 1)` |
| | ⇒`Conv2d(128, 256, 4, 2, 1)` |
| | ⇒ 4×ResidualLayer{`Conv2d(256, 256, 3, 1, 1)` |
| | ⇒`Conv2d(256, 256, 3, 1, 1)`, *no activation function*} |

**Table E4** Decoder $G$ architecture. Note that the style code is used to learn the Adaptive Instance Normalization (AdaIN) parameters that are used in the residual layers of the decoder. These parameters are outputted by a multi-layer perception (MLP) which takes the style code as input. The MLP has 8 input dimensions, one hidden layers of dimension 256, and ReLU nonlinear activations throughout (besides the output).

| | |
|---|---|
| `content code` | ⇒ 4×ResidualLayer{`Conv2d(256, 256, 3, 1, 1)` ⇒`Conv2d(256, 256, 3, 1, 1)`, *no activation function*} |
| | ⇒Upsample2x⇒`Conv2d(256, 128, 5, 1, 2)` |
| | ⇒Upsample2x⇒`Conv2d(128, 64, 5, 1, 2)` |
| | ⇒Upsample2x⇒`Conv2d(64, 32, 5, 1, 2)` |
| | ⇒Upsample2x⇒`Conv2d(32, 16, 5, 1, 2)` |
| | ⇒`Conv2d(16, 1, 7, 1, 3)`, tanh *activation* |