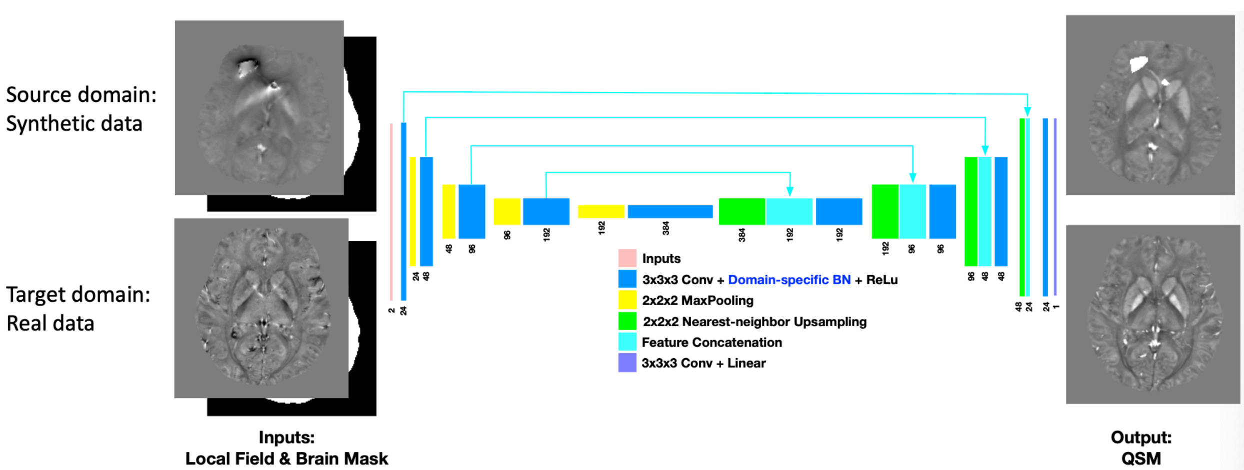
Training data: The COSMOS result of 2016 QSM challenge dataset is used to generate the training data. Using the only QSM data, the data augmentations such as elastic transform, contrast changes, and adding high susceptibility sources are used to generates synthetic QSM datasets. Then the induced field maps are calculated using dipole convolution.

Neural network training: For each dataset of 2019 QSM reconstruction challenge, which has different noise level and image contrast. We trained individual deep neural network on the synthetic training data. The neural network is a 3D convolutional neural network with encoder decoder architecture. The inputs are local tissue field and brain mask, and the output is QSM. During training, the network has four inputs (local field and brain mask from the synthetic data and real data) and two outputs (QSM outputs). Domain-specific batch normalization is used to address the domain shift problem. All other layers have shared parameters. The loss function includes the L1 loss of source domain QSM $L_{Sx} = \left\|x_s - x_{s\_Label}\right\|_1$, data consistency loss of target domain QSM $L_{Tx} = \left\|W \cdot (e^{jy_t} - e^{jd*x_t})\right\|_2$ and data regularization loss of target domain QSM $L_{Tx\_TV} = \|G_x(x_t)\|_1 + \left\|G_y(x_t)\right\|_1 + \|G_z(x_t)\|_1$, $Loss = L_{Sx} + \lambda_1 L_{Tx} + \lambda_2 L_{Tx\_TV}$. In the data consistency loss, I used the nonlinear dipole inversion, and W is the data weighting matrix which I used the magnitude image of second echo time.



Network fine-tuning: After network training, the network parameters are fine-tuned. First, load the model weights from saved pre-trained network, which the batch normalization layers are from the target domain. Then the network only takes the local field and brain mask from the target domain to do network fine-tuning based on the physical model. The loss function includes a data consistency term $L_{Tx} = \left\|W \cdot (e^{jy_t} - e^{jd*x_t})\right\|_2$ and data regularization loss of target domain QSM $L_{Tx\_TV} = \|G_x(x_t)\|_1 + \left\|G_y(x_t)\right\|_1 + \|G_z(x_t)\|_1$, $Loss = L_{Tx} + \lambda\, L_{Tx\_TV}$. After a couple of hundred of iterations, the fine-tuning is stable and stopped.

	Submitted		New method	
	SNR1	SNR2	SNR1	SNR2
rmse	50.2884	46.0791	**42.2453**	**39.3625**
rmse_detrend	49.6812	45.9711	**42.1793**	**39.4281**
rmse_detrend_Tissue	56.7727	52.4102	**47.4442**	**43.6232**
rmse_detrend_Blood	69.4347	65.9592	73.2332	71.5003
rmse_detrend_DGM	36.3525	34.0728	**25.5063**	**23.8661**
DeviationFromLinearSlope	0.0814	0.0761	**0.0451**	**0.0413**
CalcStreak	0.1148	0.1141	**0.0524**	**0.0426**
DeviationFromCalcMoment	40.1187	39.8742	**14.5971**	**10.8750**

Environments: python 2.7, tensorflow 1.12, keras 2.2.2

How to use the code:

1. use ./training_data/genChi.py to generate synthetic susceptibility maps
2. use ./training_data/qsmFmapCal_gpu.py to do dipole convolution, you can use your own MATLAB code to do dipole convolution too.
3. use ./genRDFPatchesSyn.py to get patches of the synthetic data for network training due to memory limitations of hardware required for network training
4. do network training by calling ./network_model/train_model/train.py.
   In ./network_model/train_model/train.py, change config["datasets_path"], config["targetdata_path"] to the path of the synthetic training data and real target data.
5. After network training, fine-tune the network on individual dataset by calling ./network_model/model_finetune/ train.py.