Computational MR imaging
Laboratory 10: Deep Learning-based MRI Reconstruction

Code submission is due by 12:00 before the next Thursday lab section. Please upload your code to StudOn in a described format. Late submissions will not be accepted.

Learning objectives

- Get familiar with Deep Learning (DL)-based MRI reconstruction methods.
- Train Deep Learning-based MRI reconstruction models.
- Examine effects of model architecture on reconstruction performance.
- Fine-tune the pre-trained network.

1. Deep Learning-based MRI Reconstruction

The goal is to train the Deep Learning (DL)-based MRI reconstruction model with different architectures.

1.1. Prepare training data.

- Knee data from the FastMRI public dataset

  1.1.1. Build a Pytorch dataset class, *FastMRIDataset()*.

  1.1.1.1. Arguments

  1.1.1.1.1. root: A path to the dataset. This is defined in Lab10_op().

  1.1.1.1.2. prototype: By setting this to True, only five data are loaded for pyototyping.

  1.1.1.2. Each FastMRI data is stored in h5 format and contains

  - **[Datasets]** gt: Ground truth image
  - **[Datasets]** kspace_us: Undersampled kspace
  - **[Datasets]** mask: Undersampling mask
  - **[Datasets]** sens_maps: Sensitivity maps
  - **[Attributes]** max: Maximum value of the ground truth image

  1.1.1.3. The dataset returns a named tuple, *VarNetSample*. Do not neglect the shape of each named tuple

  1.1.1.4. Transfer masked_kspace, target, mask, sens_maps, and max_value to the self.device.

  1.1.2. Analyze one data i.e., in op.data_root/"train."

  1.1.2.1. Print shapes of ["masked_kspace", "mask", "sens_maps", "target"]

  1.1.2.2. Are coefficients of the masked_kspace in complex-valued? it not, how do you deal with the complex-valued coefficients and why?

  1.1.2.3. Plot one slice of the masked_kspace and the target.

  1.1.2.4. Are the masked_ksapce and the target the same shape? if not, what is the reason for that?

1.2. Build a UNet architecture with the given *NormUnet()* model.

  1.2.1. Implement static methods, *sens_expand* and *sens_reduce* in Lab10_op.

  1.2.2. Initialization parameters

  - chans: Number of channels
  - pools: Number of encoder and decoder parts

  1.2.3. Forward

  - The model takes a single coil image

- Return: Absolute-valued reconstruction image

1.3. Define the class, *VarNetBlock(nn.Module)*.
  1.3.1. Initialization parameters
    - model: Module for "regularization" component of VN.
  1.3.2. forward
    - Achieve the update term for each cascade steps.
      - $u_t = u_{t-1} - \sum_i^N K_{i,t}^T \rho'_{i,t}\left(K_{i,t} u_{t-1}\right) - \lambda A^H (A u_{t-1} - f)$

1.4. Build a VarNet architecture with the given *NormUnet()* model.
  1.4.1. Initialization parameters
    - num_cascades: Number of cascaded for variational network
    - chans: Number of channels for cascade U-net
    - pools: Number of encoder and decoder parts
    - (Hint To define the cascades, use *nn.ModuleList()*.)
  1.4.2. Forward
    - Return: Absolute-valued reconstruction image

1.5. Implement a method, *trainer()* in Lab10_op.

1.6. Setup for training
  1.6.1. Implement a method get_model() in Lab10_op.
    1.6.1.1. Define a desired model.
    1.6.1.2. Trasnfer the model to the self.device
    1.6.1.3. Return the model
  1.6.2. Implement a method get_loss() in Lab10_op.
    1.6.2.1. Define an SSIMLoss and transfer it to the self.device
  1.6.3. Implement a method get_optimizer in Lab10_op.
    1.6.3.1. Define an Adam optimizer.
  1.6.4. Define dataloaders for train, validation, and test with *FastMRIDataset()*.
    - Batch size: 4
    - Train: shuffle / Validation & Test: not shuffle

1.7. Testing pretraining models
  1.7.1. The pre-trained models were trained using a learning rate of 0.01 for 5 epochs. The structures of both models were defined as described below. Load the pre-trained models and showcase their reconstruction results. To apply the models, utilize the *tester()* method previously defined. What can you observe? Are you happy with the results? Support your idea by providing reasons.
    - UNet
      - chans: 12
      - pools: 4
    - VarNet
      - num_cascades: 5
      - chans: 16
      - pools: 4

1.8. Fine tune the pre-trained models with a learning rate of 0.001 for 10 epochs. For reproducibility, call seed_everything() function for each training. .
    - Loss: SSIM Loss
    - Optimizer: Adam optimizer

1.8.1.1. How many parameters can be trained for each model? Is this a fair comparison?

1.8.2. Plot the train loss and the validation loss. Check for convergence.

1.8.3. Compare UNet and VarNet models in terms of metrics and image quality in different training setups (pretraining & fine-tuning). Plot ground truth, UNet, and VarNet reconstructions and discuss observations.

1.8.4. Explain the advantage of fine-tuning neural networks.

1.8.5. Discuss the trade-offs between UNet and VarNet models.

1.8.6. Analyze which model architecture performs better and provide reasons.

1.8.7. Identify generic problems in DL-based MRI reconstruction, provide examples, and explain the underlying reasons.