DeepInPy: Deep Inverse Problems in Python

Jonathan I Tamir,[1,2] Stella X Yu[1,3], Michael Lustig[1]

[1]*Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA, United States*

[2]*Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, United States*

[3]*International Computer Science Institute, University of California, Berkeley, Berkeley, CA, United States*

Target Audience: Image reconstruction and deep learning researchers and developers.

Introduction: Motivated by the great success of recognition tasks in computer vision, deep learning has recently flooded the MRI reconstruction space. In particular, deep learning in tandem with model-based iterative optimization has shown great promise at solving imaging-based inverse problems beyond the capabilities of conventional parallel imaging and compressed sensing [1-5]. To develop a new deep learning reconstruction, practitioners must build training and testing code from scratch, including models, data loaders, training loops, visualizers, hyperparameter search, loggers, and more. We present **`DeepInPy`**, a Python software package that streamlines the above processes, freeing the user to focus on the innovative components while still easily following best practices for training.

Availability: `DeepInPy` is currently in alpha stage and is freely available under the MIT license (link in box below).

Functionality: `DeepInPy` combines PyTorch [6], PyTorch Lightning [7], and Test Tube [8] to support rapid prototyping of deep inverse problems. Among its features, `DeepInPy` incorporates the following functionality:

- Modular reconstruction block design,
- Multi-GPU training across multiple machines,
- TensorBoard training visualization,
- Hyperparameter logging and search.

`http://github.com/jtamir/deepinpy`

At its core, `DeepInPy` provides an abstract interface for a **`Recon`** object, which transforms a data variable $\mathbf{y}$ to a reconstruction variable $\hat{\mathbf{x}}$ with forward model $\mathbf{A}$ and reconstruction parameters $\boldsymbol{\theta}$ (Fig. 1). `Recon` objects are composed of a sequence of modular blocks, including `System` blocks (e.g. multi-channel MRI), `Model` blocks (e.g. ResNet [9], U-Net [10]), and `Optimization` blocks (e.g. Conjugate Gradient). Using this interface, new reconstructions can be built that incorporate multiple types of blocks, including data consistency, loops (unrolls), and neural networks. `DeepInPy` can leverage packages such as SigPy [11] to create the (non-Cartesian) forward models and other signal processing functions.

Demonstration: As a proof of concept, we implement the MoDL [2] reconstruction built with a ResNet backbone. The Stanford fully sampled knees dataset downloaded from http://mridata.org was used and retrospectively under-sampled with variable density Poisson disc sampling masks, and sensitivity maps were computed using ESPIRiT [12]. Fig. 2 shows the TensorBoard training visualization and hyperparameter configuration, which also lists the learnable network parameters. Fig 3. Shows the reconstruction result on an image from the validation set during training.

Summary: `DeepInPy` is a Python software package for developing deep learning-based inverse problems in Python. `DeepInPy` can be used to quickly compare and develop new deep learning reconstructions while inheriting training best practices without the need to redevelop boilerplate training code.

References: 1. Lustig, MRM 2007. **2.** Aggarwal, IEEE TMI 2018. **3.** Hammernik, MRM 2017. **4.** Schlemper, IEEE TMI 2017. **5.** Tamir, ISMRM 2019. **6.** Paszke, NIPS 2017. **7.** Falcon, https://github.com/williamFalcon/pytorch-lightning. **8.** Falcon, https://github.com/williamFalcon/test-tube. **9.** He, CVPR 2016. **10.** Ronneberger, MICCAI 2014. **11.** Ong, ISMRM 2019. **12.** Uecker, MRM 2014.

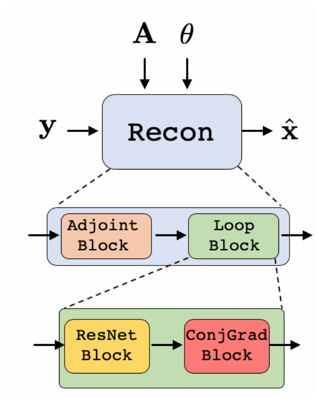
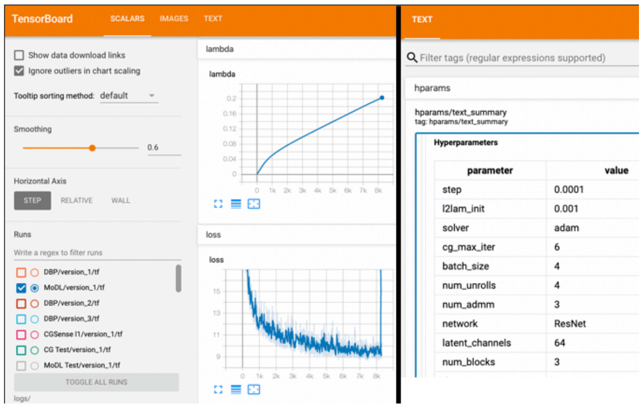


Fig 1. Composition of multiple blocks comprising a `Recon`.



Hyperparameters

parameter	value
step	0.0001
l2lam_init	0.001
solver	adam
cg_max_iter	6
batch_size	4
num_unrolls	4
num_admm	3
network	ResNet
latent_channels	64
num_blocks	3

Fig 2. TensorBoard visualization of MoDL training loss curves and hyperparameters of the reconstruction that are automatically logged.

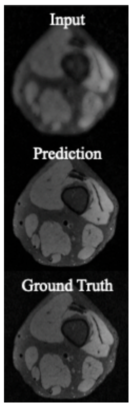


Fig 3. Visualization of MoDL reconstruction during training.