AI for image reconstruction: workshop

Andrew J. Reader

School of Biomedical Engineering and Imaging Sciences
King's College London
UK

Workshop Overview

- **Getting the basics**
  - Simple FBP via use of iradon
- **Deep learning basics**
  - **Code** for a CNN in PyTorch to denoise an image

- **System model** ($\boldsymbol{A}$) for use in PyTorch
  - **Code** for the system model in PyTorch

- **Deep learned FBP** and **code**
- **Deep image prior** and **code**

- **Iterative reconstruction**
  - Unrolled MLEM with inter-update CNN and then **code**

References

- **I am presenting some simple but potentially fresh perspectives**
  - Open to collaboration if you are interested!

- **Consider citing**

  - Reader *et al Deep learning for PET image reconstruction* IEEE TRPMS 2020
  - Reader & Schramm *Artificial intelligence for PET image reconstruction* 2021 JNM 62 (10), 1330-1333
  - Reader AJ 2022 Self-Supervised and Supervised Deep Learning for PET Image Reconstruction AIP conference proceedings *(under review)*

Basic FBP

Brief, simple
CODING EXAMPLE
Jupyter Notebook

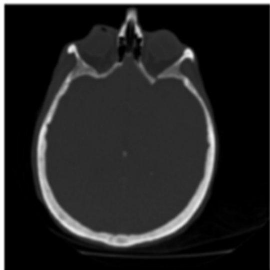
Goal: gain familiarity with notation and radon function

```python
from skimage.transform   import iradon, radon, resize   # algorithms for image processing
from skimage.data        import brain
import numpy             as np
import matplotlib.pyplot as plt                          # library for data visualisation
```

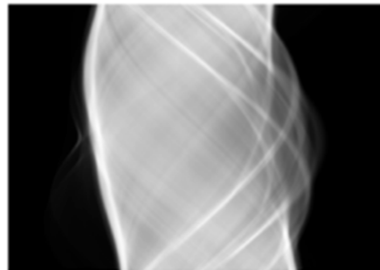
```python
plt.rcParams['figure.dpi'] = 600    # Improve figure quality
nxd    = 256                        # nxd is the number of pixels in the x dimension of the reconstructed image
nphi   = int(nxd*1.0)               # nphi is the number of view angles
azi_angles = np.linspace(0.0,180.0, nphi, endpoint=False) # nphi values between 0.0 and 180.0, excluding endpoint
```

```python
brainimage       = brain()          # a range of brain CT slices
true_object_np   = resize(brainimage[5,30:-1,:-30], (nxd,nxd), anti_aliasing=False)
true_sinogram_np = radon(true_object_np, azi_angles, circle=False)

fig1, ax = plt.subplots(1,4, figsize=(16,2)) # No. rows, cols,  figsize Width,Height (inches)
ax[0].imshow(true_object_np,     cmap='Greys_r'); ax[0].set_title('True');     ax[0].set_axis_off()
ax[1].imshow(true_sinogram_np.T, cmap='Greys_r'); ax[1].set_title('Sinogram'); ax[1].set_axis_off()

bp_recon  = iradon(true_sinogram_np,  output_size=nxd, filter_name =  None,  circle=False)    # Plain backprojection
fbp_recon = iradon(true_sinogram_np,  output_size=nxd, filter_name = 'ramp', circle=False)  # Basic ramp-filtered FBP
ax[2].imshow(bp_recon,  cmap='Greys_r'); ax[2].set_title('BP');  ax[2].set_axis_off()
ax[3].imshow(fbp_recon, cmap='Greys_r'); ax[3].set_title('FBP'); ax[3].set_axis_off()
```

True

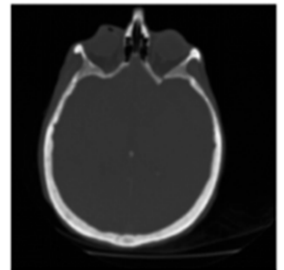


Sinogram



BP

FBP



Deep learning basics

Deep learning components

1. Training data

From no training data….

…to tens of examples pairs… to thousands

2. Architecture / inductive prior for the mapping from input to output

Trainable parameters for a code structure

E.g. fully-connected (linear) layers, convolutional neural networks (CNNs), transformers

3. Loss functions to decide how well a mapping is doing its job

Mean squared error (MSE) or L2 norm

Mean absolute error (MAE) or L1 norm

Perceptual loss

Adversarial loss

4. Optimisers

Stochastic gradient descent (SGD)

Adam

…and many more

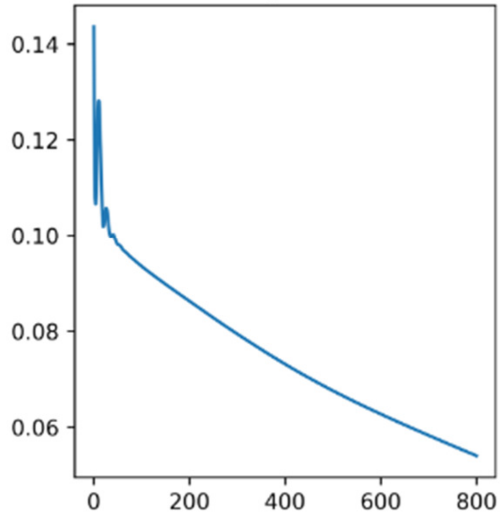
Basic CNN

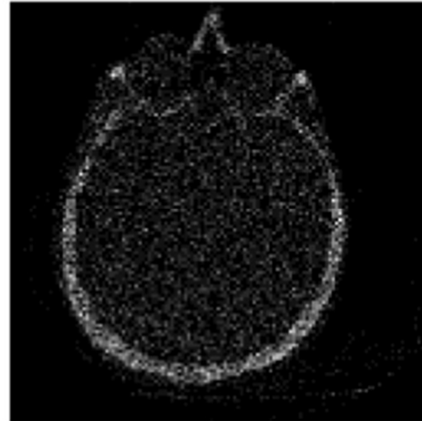
CODING EXAMPLE IN
Jupyter Notebook / Python / PyTorch

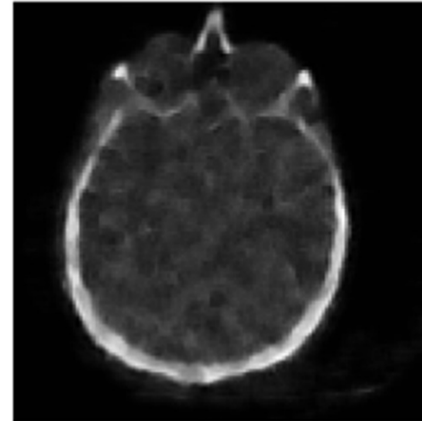
```python
class CNN(nn.Module):
    def __init__(self, num_channels):
        super(CNN, self).__init__()
        self.CNN = nn.Sequential(
            nn.Conv2d(1,            num_channels, 3, padding=1, padding_mode='reflect'), nn.PReLU(),
            nn.Conv2d(num_channels, num_channels, 3, padding=1, padding_mode='reflect'), nn.PReLU(),
            nn.Conv2d(num_channels, num_channels, 3, padding=1, padding_mode='reflect'), nn.PReLU(),
            nn.Conv2d(num_channels, num_channels, 3, padding=1, padding_mode='reflect'), nn.PReLU(),
            nn.Conv2d(num_channels, 1,            3, padding=1, padding_mode='reflect'), nn.PReLU()
        )
    def forward(self, x): return torch.squeeze(self.CNN(x.unsqueeze(0).unsqueeze(0)))

cnn = CNN(nxd).to(device)  # create a CNN object from the class
```

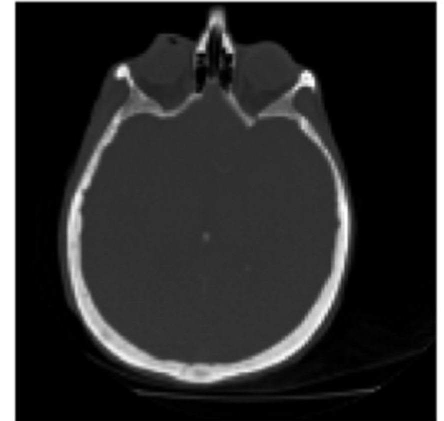
```python
from IPython.display     import display, clear_output
#================================================TRAIN THE NETWORK
loss_fun  = nn.MSELoss()
optimiser = torch.optim.Adam(cnn.parameters(), lr=1e-4)
train_loss = list()
epochs     = 5000

for ep in range(epochs):
    optimiser.zero_grad() # set the gradients to zero
    output_cnn = cnn(noisy_image_torch)

    loss = loss_fun(output_cnn, true_object_torch)

    train_loss.append(loss.item())
    loss.backward()    # Find the gradients
    optimiser.step()   # Does the update

    if ep % 20 == 0:
        fig2, ax = plt.subplots(1,4, figsize=(16,4))
        ax[0].plot(train_loss[19:-1]);                                ax[0].set_title('Loss, epoch %d' % ep)
        ax[1].imshow(torch_to_np(noisy_image_torch), cmap='Greys_r'); ax[1].set_title('Noisy Input')
        ax[2].imshow(torch_to_np(output_cnn),        cmap='Greys_r'); ax[2].set_title('CNN output')
        ax[3].imshow(torch_to_np(true_object_torch), cmap='Greys_r'); ax[3].set_title('True')
        ax[1].set_axis_off(); ax[2].set_axis_off();  ax[3].set_axis_off()
        clear_output(wait=True); plt.pause(0.001)
```

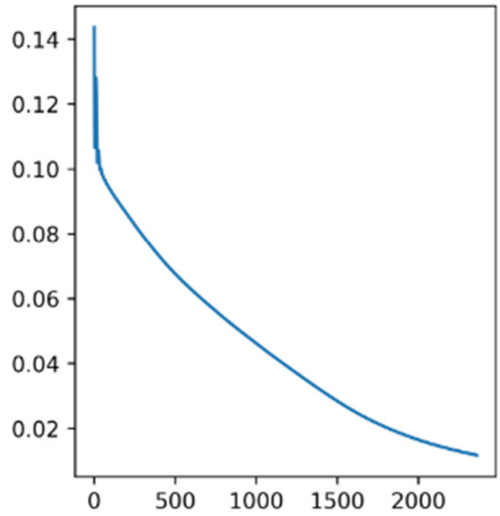
Basic CNN

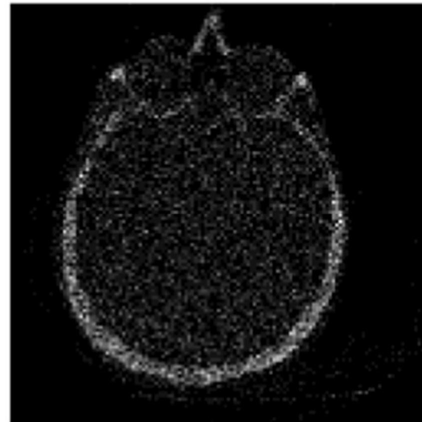


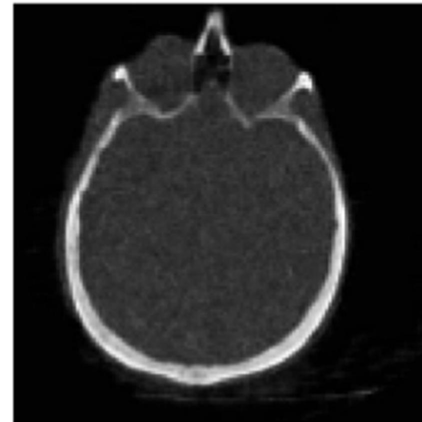


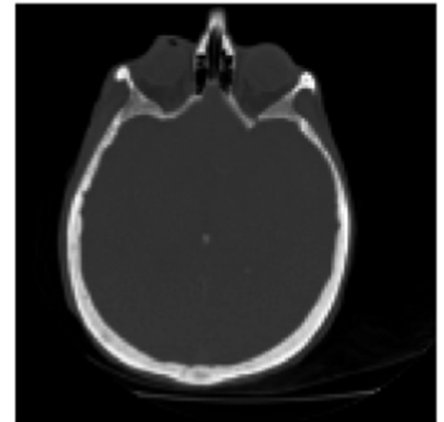








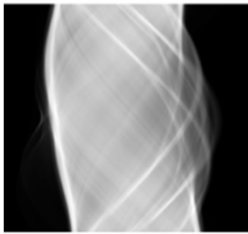


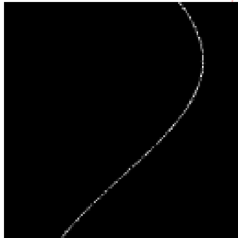


Creating a system model for use in PyTorch

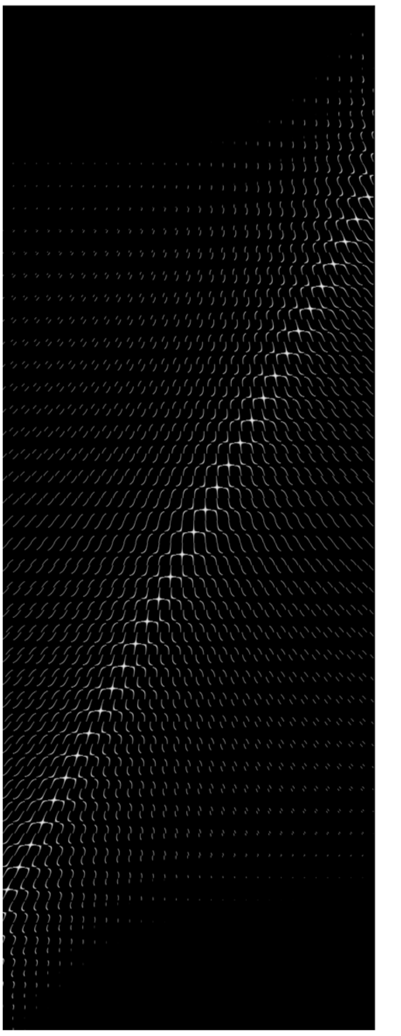
System model: matrix A

for discrete Radon or x-ray transform

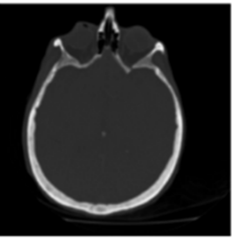




$\boldsymbol{q} = $



$\boldsymbol{x}$



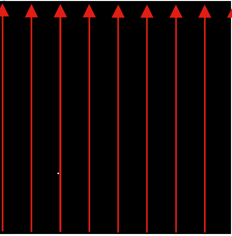
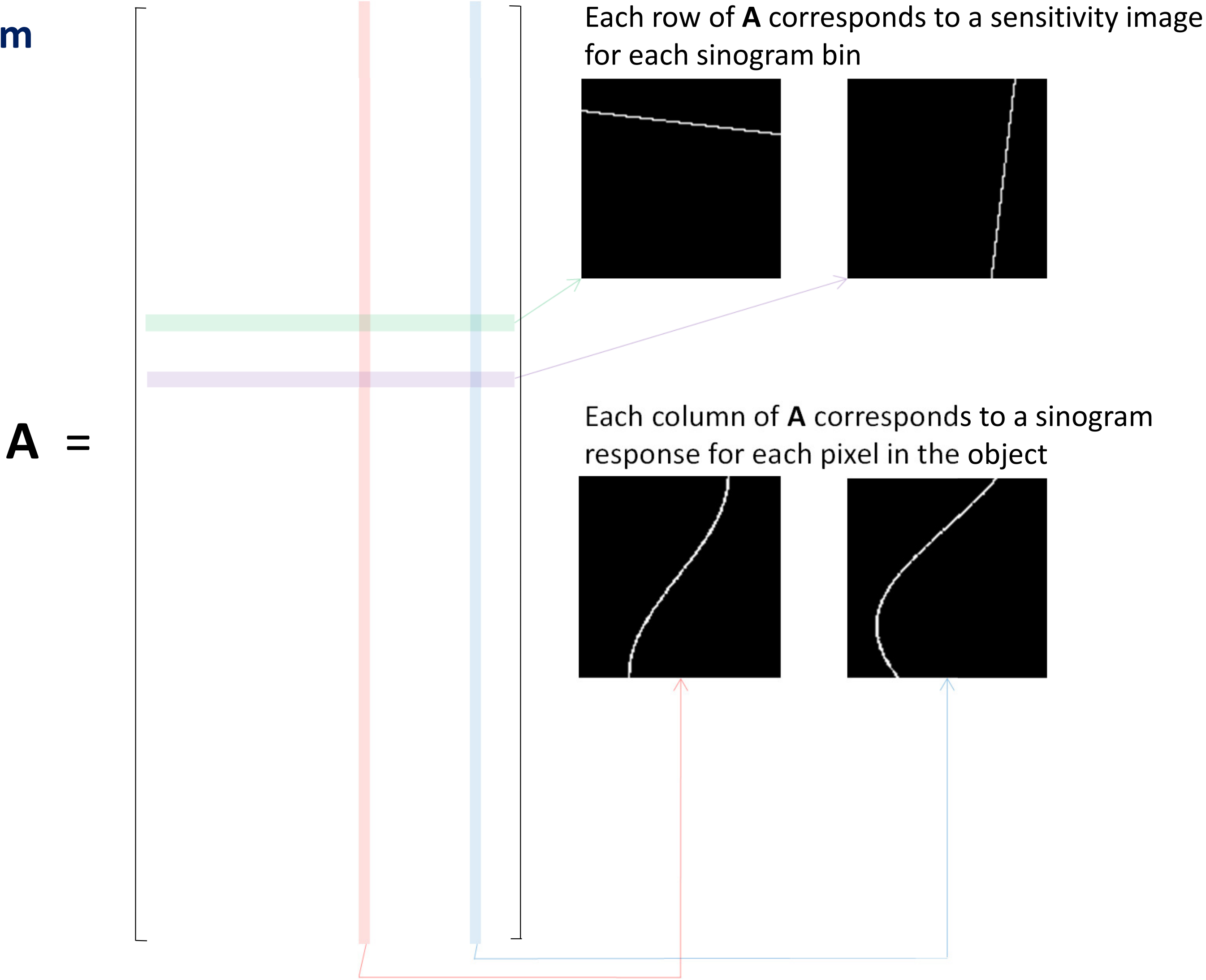


Illustration of a flattened 2D sinogram – to give just a single vector

Illustration of a flattened 2D image – to give just a vector

System model: matrix A

for discrete Radon or x-ray transform



Implementing a system matrix

Brief, simple CODING EXAMPLE Jupyter Notebook

Goal: gain familiarity with PyTorch and a system matrix

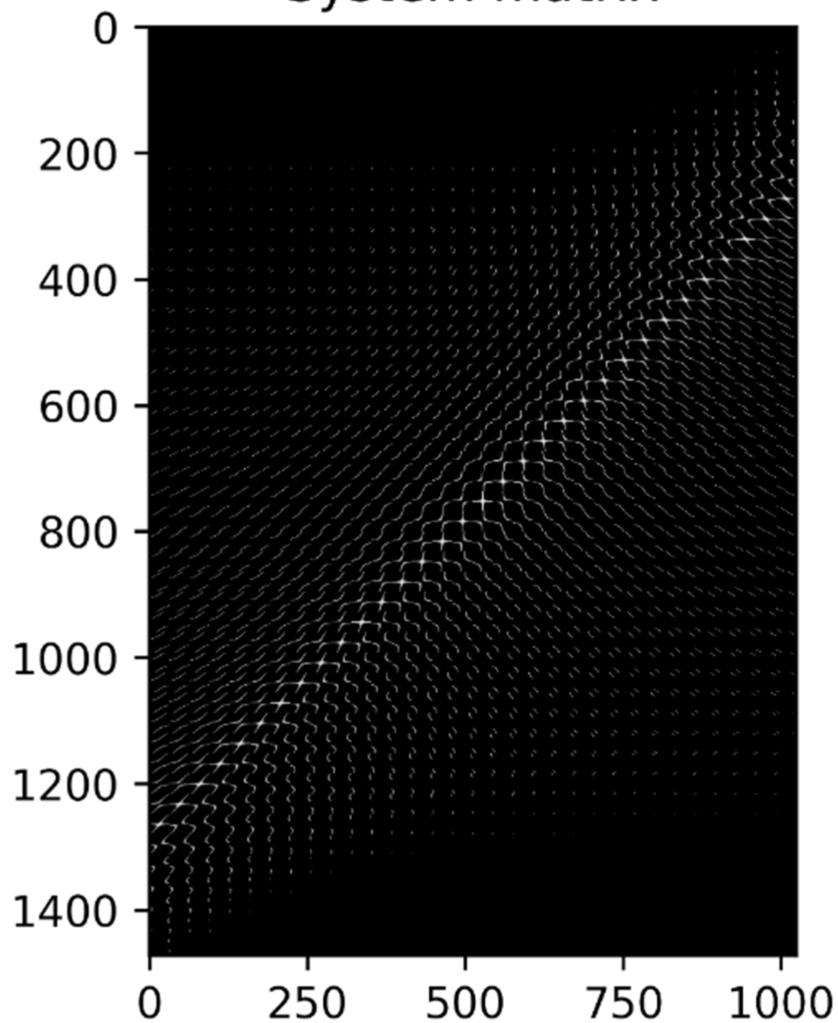
```
In [4]: import torch, torch.nn   as nn
```

```
In [5]: # To demonstrate setting up a system matrix, use smaller values for the image size for speed
nxd        = 32; nphi       = int(nxd*1.0)
#-------- Need to find out the number of projection bins in a parallel projection
empty_image   = np.zeros( (nxd,nxd) )
azi_angles    = np.linspace(0.0,180.0, nphi, endpoint=False)
sinogram_np   = radon(empty_image,azi_angles, circle=False)
nrd           = sinogram_np.shape[0]      # nrd is the no. of bins in a parallel projection
```

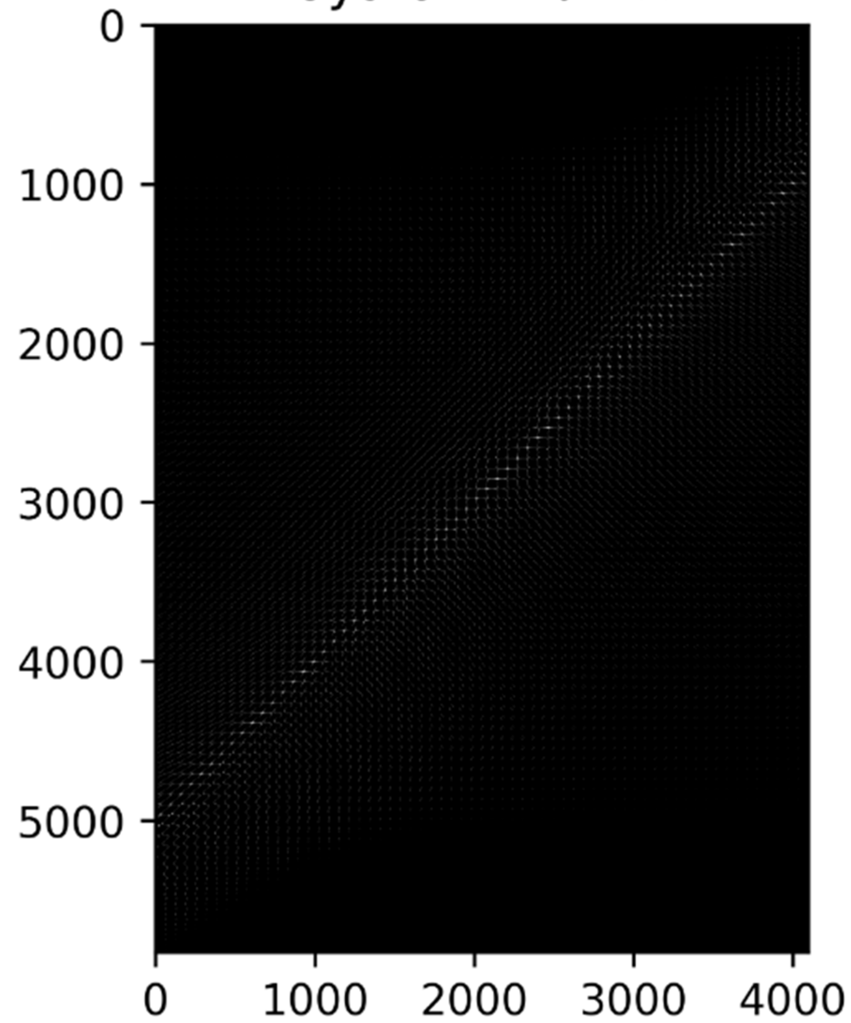
```
In [6]: #--------------------------------TORCH SYSTEM MATRIX------------------------------------------
def make_torch_system_matrix(nxd, nrd, nphi):
    point_source  = np.zeros( (nxd,nxd) )
    azi_angles    = np.linspace(0.0, 180.0,  nphi, endpoint=False)
    num_bins      = nrd * nphi
    num_pixels    = nxd * nxd
    system_matrix = torch.zeros(num_bins, num_pixels) # rows = num sino bins, cols = num image pixels
    col_index     = 0
    for xv in range(nxd):
        for yv in range(nxd):        # Now have selected pixel (xv, yv)
            point_source[:,:]   = 0.0
            point_source[xv,yv] = 1.0
            sinogram_np = radon(point_source,azi_angles, circle=False)
            system_matrix[:,col_index] = torch.reshape(np_to_torch(sinogram_np) ,(1, num_bins) )
            col_index += 1
    return system_matrix

def fp_system_torch(image, sys_mat, nxd, nrd, nphi):
    return torch.reshape(torch.mm(sys_mat,   torch.reshape(image, (nxd*nxd,1))), (nrd, nphi))
def bp_system_torch(sino,  sys_mat, nxd, nrd, nphi):
    return torch.reshape(torch.mm(sys_mat.T, torch.reshape(sino, (nrd*nphi,1))), (nxd,nxd))
```

```
In [7]: #-------------------------TORCH TO NUMPY CONVERTORS------------------------------------------
def torch_to_np(torch_array):   return np.squeeze(torch_array.detach().cpu().numpy())
def np_to_torch(np_array):      return torch.from_numpy(np_array).float()
```

```
In [8]: device      = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"); print(device)
sys_mat     = make_torch_system_matrix(nxd, nrd, nphi).to(device);            print(sys_mat.shape)
fig1, axs1 = plt.subplots(1,2, figsize=(8,4))
axs1[0].imshow(torch_to_np(sys_mat), cmap='Greys_r'); axs1[0].set_title('System matrix')
```

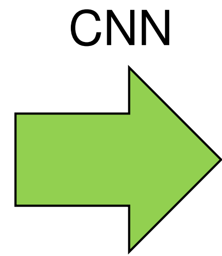


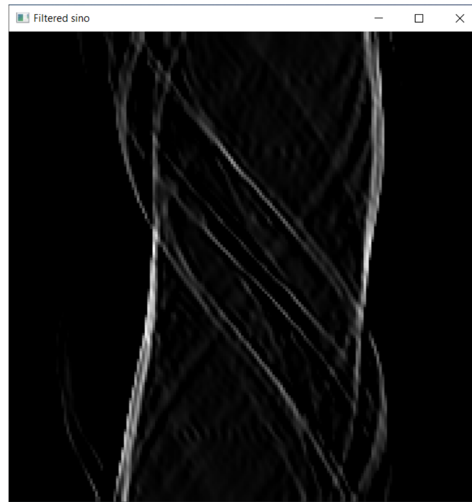
System matrix
0
200
400
600
800
1000
1200
1400
0
250
500
750
1000

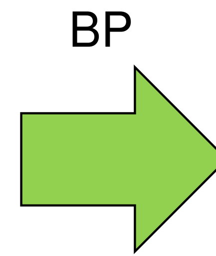


System matrix
0
1000
2000
3000
4000
5000
0
1000
2000
3000
4000

Learned FBP







Reader 2022

LEARNED FBP

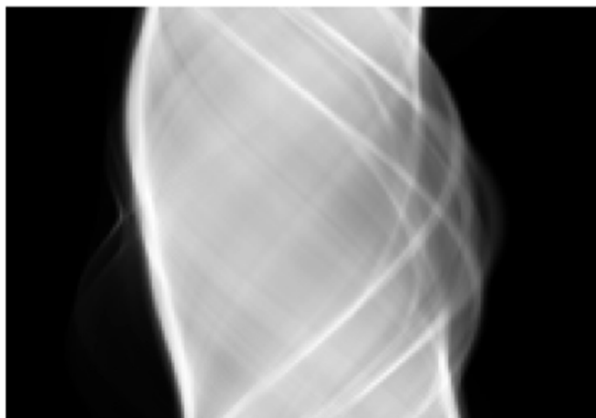
CODING EXAMPLE IN
Jupyter Notebook / Python / PyTorch

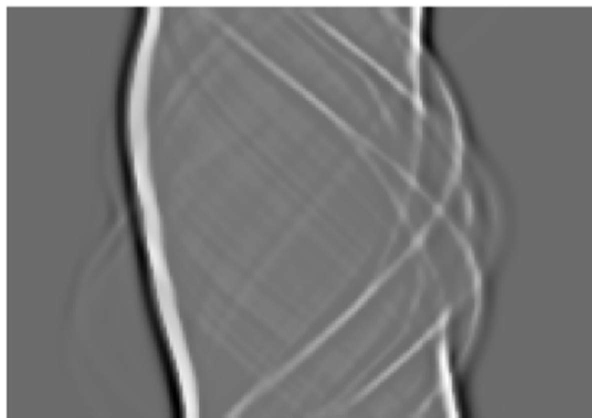
LEARNED FBP

```
class FBP_CNN_Net(nn.Module):
    def __init__(self, cnn, sino_for_reconstruction):
        super(FBP_CNN_Net, self).__init__()
        self.sino_ones  = torch.ones_like(sino_for_reconstruction)
        self.sens_image = bp_system_torch(self.sino_ones, sys_mat, nxd, nrd, nphi)
        self.cnn    = cnn
        self.prelu = nn.PReLU()
    def forward(self, sino_for_reconstruction):
        filtered_sino = self.cnn(sino_for_reconstruction)
        recon         = bp_system_torch(filtered_sino, sys_mat, nxd, nrd, nphi) / (self.sens_image+1.0e-15)
        recon         = self.prelu(recon)
        fpsino        = fp_system_torch(recon, sys_mat, nxd, nrd, nphi)
        return recon, fpsino, filtered_sino

cnn    = CNN(nxd).to(device)  # create a new CNN object from the CNN class
fbpnet = FBP_CNN_Net(cnn, true_sinogram_torch).to(device)
```

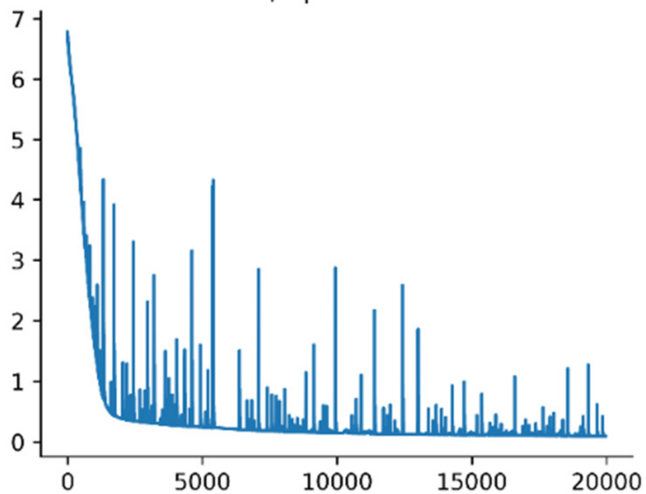
LEARNED FBP

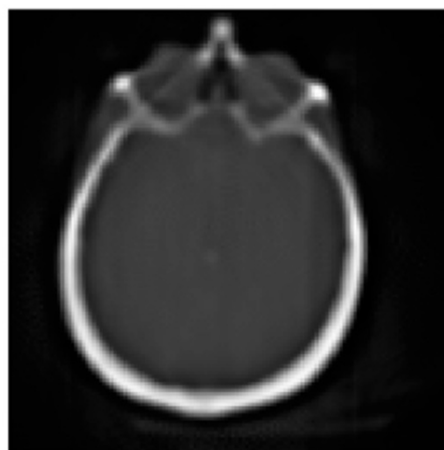
```
#==============================================TRAIN THE fbpnet NETWORK
loss_fun  = nn.MSELoss()
optimiser = torch.optim.Adam(fbpnet.parameters(), lr=1e-4)
train_loss = list()

for ep in range(5000 +1):
    optimiser.zero_grad() # set the gradients to zero
    recon, rec_fp, filtered_sino = fbpnet(true_sinogram_torch)
    # Self-supervised, data fidelity
    loss = loss_fun(rec_fp, torch.squeeze(true_sinogram_torch))
    # Ground truth supervised -> #loss = loss_fun(fbp_recon, torch.squeeze(true_object_torch))
    train_loss.append(loss.item())
    loss.backward()   # Find the gradients
    optimiser.step()  # Does the update

    if ep % 50 == 0:
        fig2, axs2 = plt.subplots(2,3, figsize=(16,8)) # No. rows, cols,  figsize Width,Height (inches)
        axs2[0,0].imshow(torch_to_np(true_sinogram_torch).T, cmap='Greys_r'); axs2[0,0].set_title('Measured data')
        axs2[0,1].imshow(torch_to_np(filtered_sino).T, cmap='Greys_r');       axs2[0,1].set_title('Filtered data')
        axs2[0,0].set_axis_off(); axs2[0,1].set_axis_off()
        axs2[0,2].set_axis_off(); axs2[1,1].set_axis_off()
        axs2[1,1].imshow(torch_to_np(recon), cmap='Greys_r'); axs2[1,1].set_title('Recon %d' % (ep))

        axs2[0,2].imshow(torch_to_np(rec_fp).T, cmap='Greys_r'); axs2[0,2].set_title('Forward projection')
        axs2[1,2].plot(train_loss[-49:-1]);  axs2[1,2].set_title('Loss, epoch %d' % ep)
        axs2[1,0].plot(train_loss[49:-1]);   axs2[1,0].set_title('Loss, epoch %d' % ep)
        axs2[1,0].spines['top'].set_visible(False); axs2[1,0].spines['right'].set_visible(False)
        clear_output(wait=True); plt.pause(0.001)
```

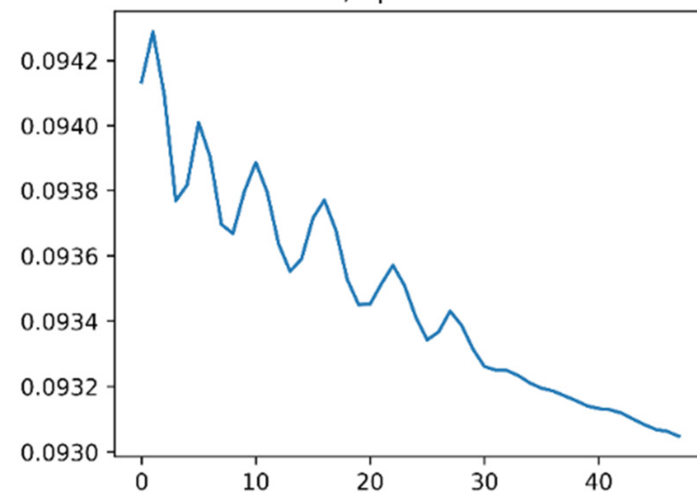

Measured data


Filtered data

Forward projection


Loss, epoch 20000
7
6
5
4
3
2
1
0
0
5000
10000
15000
20000


Recon 20000


Loss, epoch 20000
0.0942
0.0940
0.0938
0.0936
0.0934
0.0932
0.0930
0
10
20
30
40

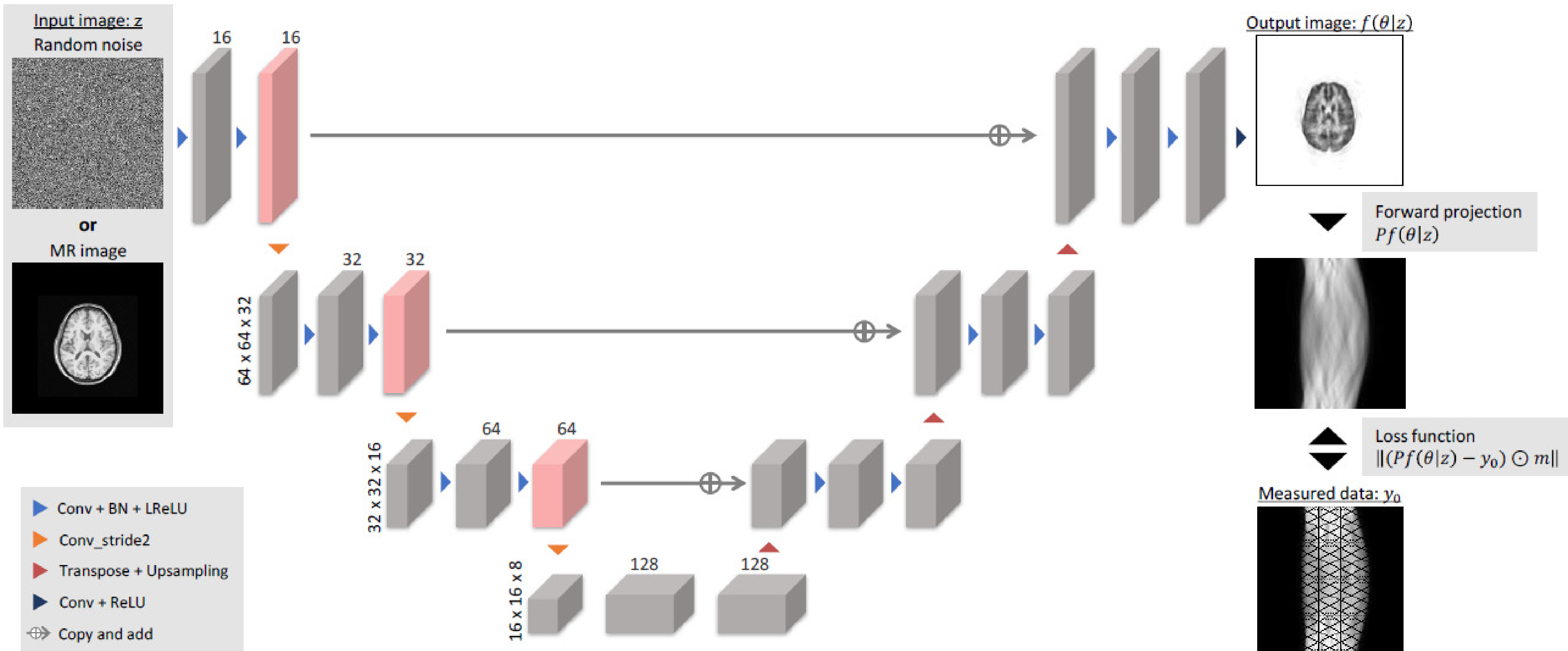
Deep Image Prior

Deep image prior
with system model

Deep Image Prior

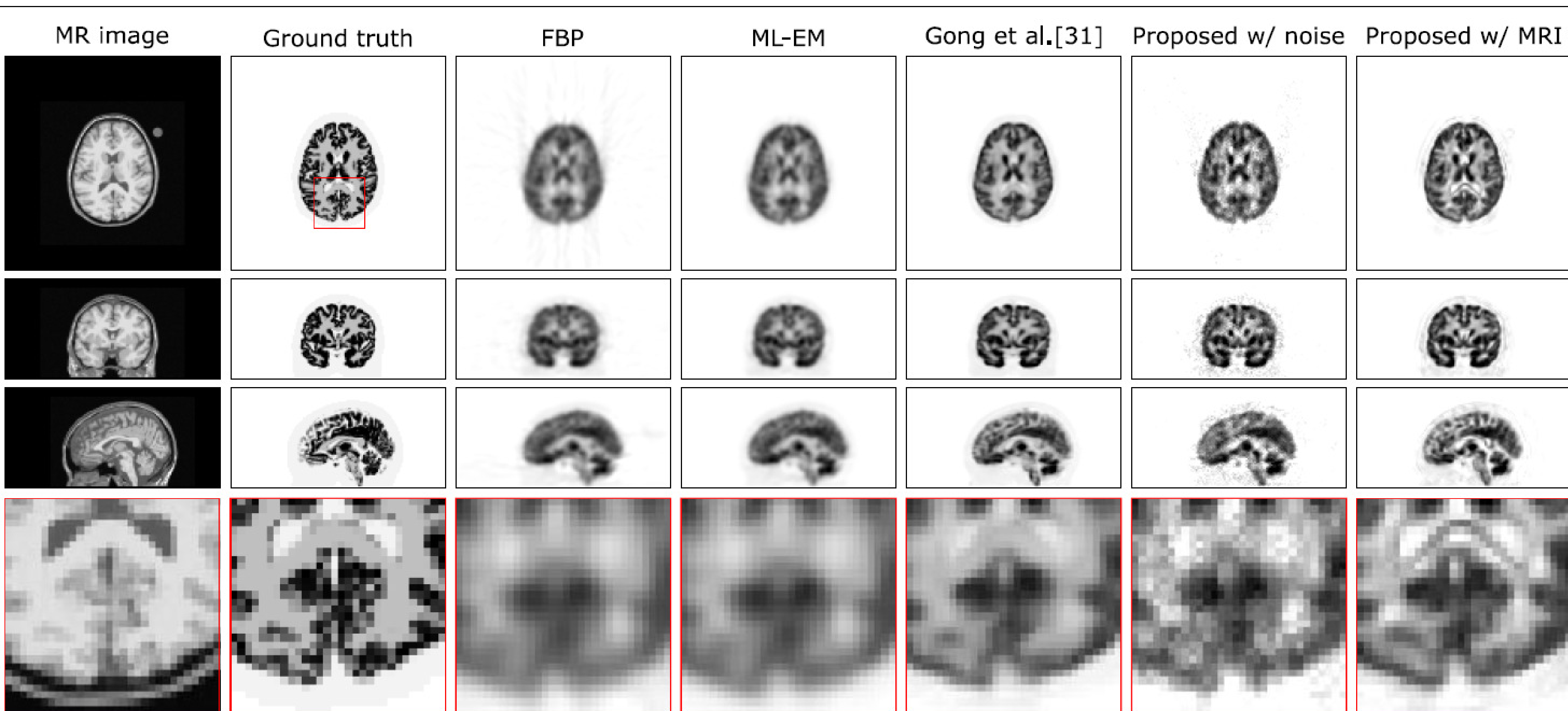
2017

Dmitry Ulyanov · Andrea Vedaldi · Victor Lempitsky



Hashimoto et al IEEE TRPMS 2022

Deep image prior with system model



Hashimoto et al IEEE TRPMS 2022

DEEP IMAGE PRIOR

CODING EXAMPLE IN
Jupyter Notebook /
Python / PyTorch

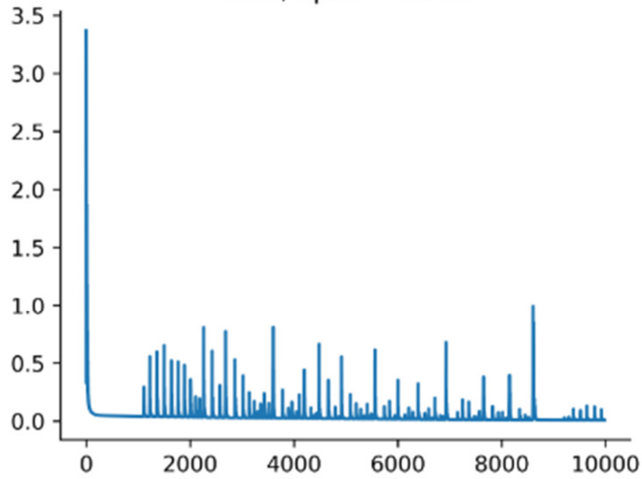
DEEP IMAGE PRIOR

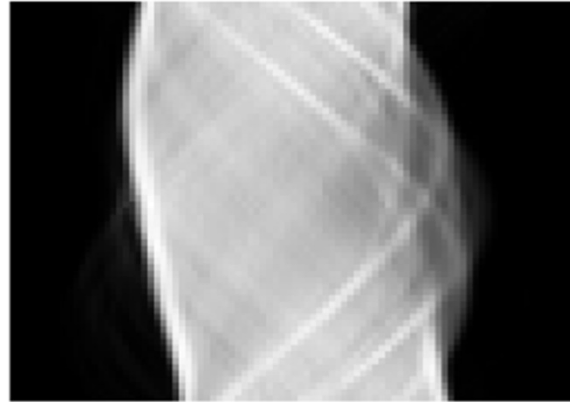
```
#=================================================================
# Now create a recon network class: process z with a CNN
#=================================================================
class Z_CNN_Net(nn.Module):
    def __init__(self, cnn, nxd, input_image):
        super(Z_CNN_Net, self).__init__()
        self.z_image    = input_image
        self.cnn        = cnn
    def forward(self):
        recon         = self.cnn(self.z_image)
        fpsino        = fp_system_torch(recon, sys_mat, nxd, nrd, nphi)
        return recon, fpsino

z_image     = torch.rand(nxd,nxd).to(device)
cnn         = CNN(nxd).to(device)  # create a new CNN object from the CNN class
znet        = Z_CNN_Net(cnn, nxd, z_image).to(device)
```

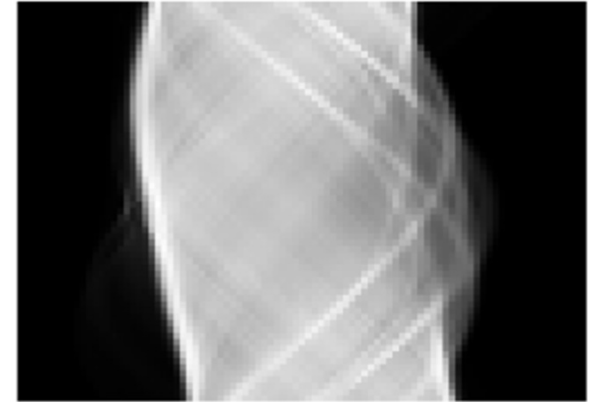
DEEP IMAGE PRIOR

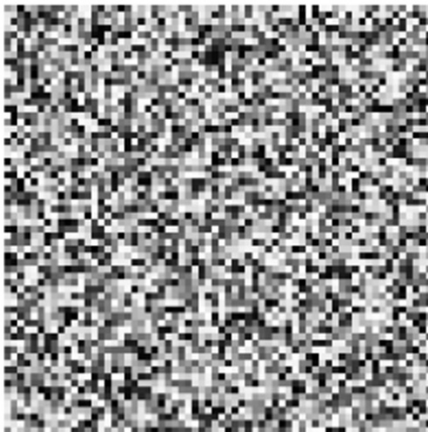
```
#================================================TRAIN THE NETWORK
loss_fun  = nn.MSELoss()
optimiser = torch.optim.Adam(znet.parameters(), lr=1e-4)
train_loss = list()

for ep in range(10000 +1):
    optimiser.zero_grad() # set the gradients to zero
    recon, rec_fp     = znet()
    loss = loss_fun(rec_fp, torch.squeeze(true_sinogram_torch))
    train_loss.append(loss.item())
    loss.backward() # Find the gradients
    optimiser.step() # Does the update

    if ep % 50 == 0:
        fig2, axs2 = plt.subplots(2,3, figsize=(16,8)) # No. rows, cols,  figsize Width,Height (inches)
        axs2[0,0].spines['top'].set_visible(False); axs2[0,0].spines['right'].set_visible(False)
        axs2[0,1].set_axis_off(); axs2[0,2].set_axis_off();
        axs2[1,0].set_axis_off(); axs2[1,1].set_axis_off();
        axs2[0,2].imshow(torch_to_np(true_sinogram_torch).T, cmap='Greys_r'); axs2[0,2].set_title('Measured data')
        axs2[0,1].imshow(torch_to_np(rec_fp).T, cmap='Greys_r'); axs2[0,1].set_title('Forward projection')
        axs2[1,0].imshow(torch_to_np(z_image), cmap='Greys_r'); axs2[1,0].set_title('z image %d x %d' % (nxd,nxd))
        axs2[1,1].imshow(torch_to_np(recon), cmap='Greys_r'); axs2[1,1].set_title('Recon %d' % (ep))
        axs2[1,2].plot(train_loss[-19:-1]);  axs2[1,2].set_title('Loss, epoch %d' % ep);
        axs2[0,0].plot(train_loss[19:-1]);  axs2[0,0].set_title('Loss, epoch %d' % ep);
        clear_output(wait=True); plt.pause(0.001)
```

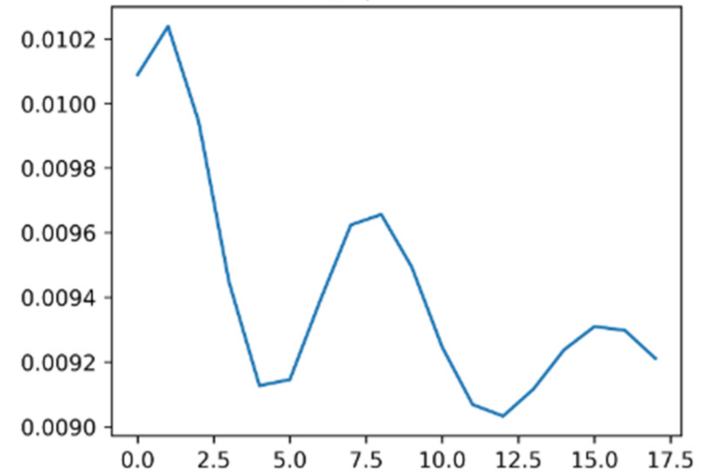
DEEP IMAGE PRIOR











MLEM, OSEM and MAPEM

Basic MLEM

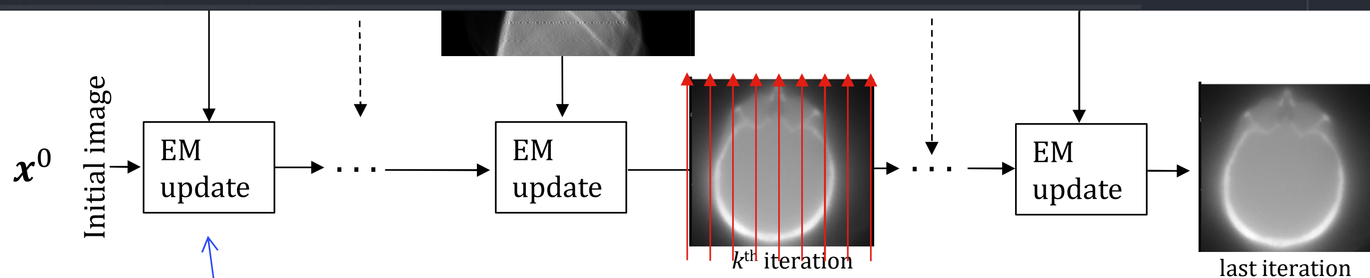
If interested, see:
https://youtu.be/lhETD4nSJec

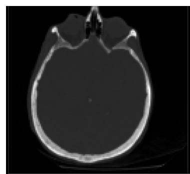
Iterative reconstruction with DL

Maximum likelihood – expectation maximisation (ML-EM)

$$x^{k+1} = \frac{x^k}{A^T 1} A^T \frac{m}{A x^k}$$

```
for it in range(self.num_its):
    fpsino = fp_system_torch(recon, sys_mat, nxd, nrd, nphi)
    ratio = sino_for_reconstruction / (fpsino +1.0e-9)
    correction = bp_system_torch(ratio, sys_mat, nxd, nrd, nphi) / (self.sens_image+1.0e-9)
    recon = recon * correction
```

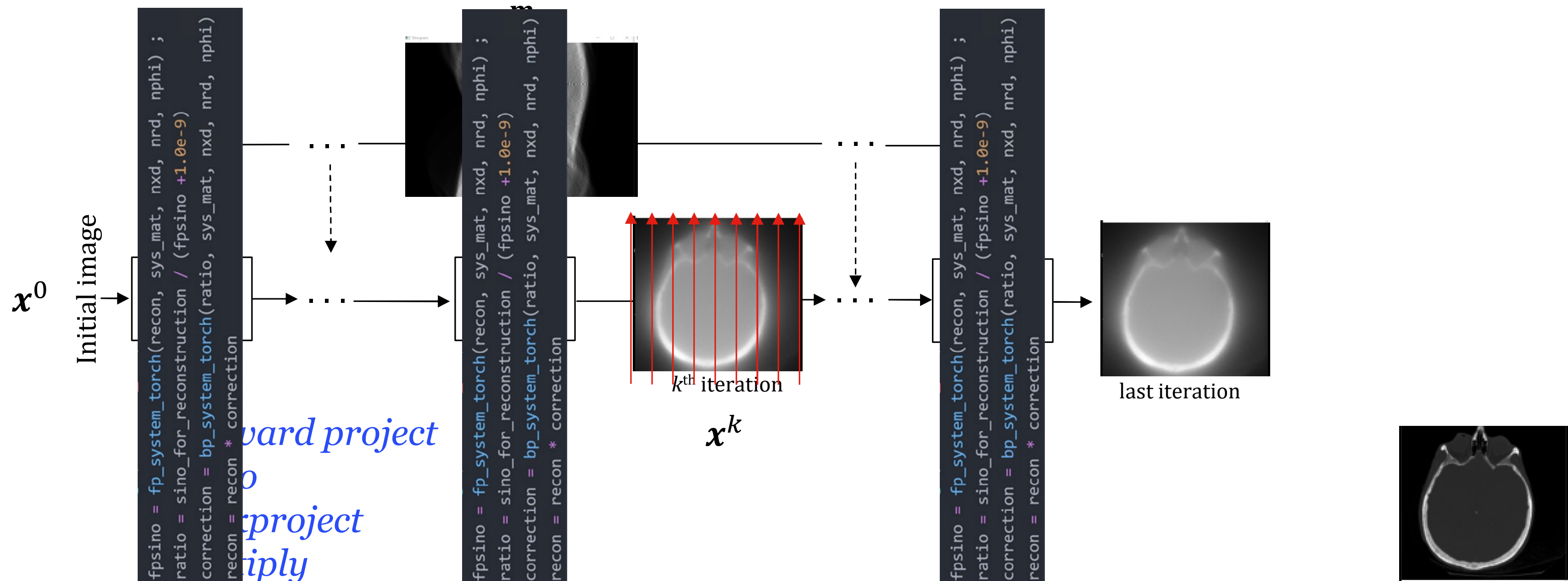




Maximum likelihood – expectation maximisation (ML-EM)

$$\boldsymbol{x}^{k+1} = \frac{\boldsymbol{x}^k}{\boldsymbol{A}^T\boldsymbol{1}}\boldsymbol{A}^T\frac{\boldsymbol{m}}{\boldsymbol{A}\boldsymbol{x}^k}$$

Unrolled into a deep network for fixed number of iterations:



Maximum likelihood – expectation maximisation (ML-EM)

$$\boldsymbol{x}^{k+1} = \frac{\boldsymbol{x}^k}{\boldsymbol{A}^T\boldsymbol{1}}\boldsymbol{A}^T\frac{\boldsymbol{m}}{\boldsymbol{A}\boldsymbol{x}^k}$$

```
class MLEM_Net(nn.Module):
    def __init__(self, sino_for_reconstruction, num_its):
        super(MLEM_Net, self).__init__()
        self.num_its  = num_its
        self.sino_ones = torch.ones_like(sino_for_reconstruction)
        self.sens_image = bp_system_torch(self.sino_ones, sys_mat, nxd, nrd, nphi)
    def forward(self, sino_for_reconstruction):
        recon  = torch.ones(nxd,nxd).to(device)
        for it in range(self.num_its):
            fpsino = fp_system_torch(recon, sys_mat, nxd, nrd, nphi)
            ratio = sino_for_reconstruction / (fpsino +1.0e-9)
            correction = bp_system_torch(ratio, sys_mat, nxd, nrd, nphi) / (self.sens_image+1.0e-9)
            recon = recon * correction
        return recon
```

Including a trainable component

```
#=================================================================
# Now set up a CNN
#=================================================================
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.CNN = nn.Sequential(
            nn.Conv2d(1, 8, 7, padding=(3, 3)), nn.PReLU(),
            nn.Conv2d(8, 8, 7, padding=(3, 3)), nn.PReLU(),
            nn.Conv2d(8, 8, 7, padding=(3, 3)), nn.PReLU(),
            nn.Conv2d(8, 8, 7, padding=(3, 3)), nn.PReLU(),
            nn.Conv2d(8, 1, 7, padding=(3, 3)), nn.PReLU()
        )
    def forward(self, x):
        x = torch.squeeze(self.CNN(x.unsqueeze(0).unsqueeze(0)))
        return x

cnn = CNN().to(device)
```

Including a trainable component

```
class MLEM_CNN_Net(nn.Module):  # torch.nn is the Base class for all PyTorch neural network modules.
    def __init__(self, cnn, sino_for_reconstruction, num_its):
        super(MLEM_CNN_Net, self).__init__()  # inherit attributes and methods from the base class, torch.nn
        self.num_its  = num_its
        self.sino_ones = torch.ones_like(sino_for_reconstruction)
        self.sens_image = bp_system_torch(self.sino_ones, sys_mat, nxd, nrd, nphi)
        self.cnn = cnn
    def forward(self, sino_for_reconstruction):
        recon  = torch.ones(nxd,nxd).to(device)
        for it in range(self.num_its):
            fpsino = fp_system_torch(recon, sys_mat, nxd, nrd, nphi)
            ratio = sino_for_reconstruction / (fpsino +1.0e-9)
            correction = bp_system_torch(ratio, sys_mat, nxd, nrd, nphi) / (self.sens_image+1.0e-9)
            recon = recon * correction
            # INTER UPDATE cnn
            recon = torch.abs(recon + self.cnn(recon))
        return recon
```

```
# Instantiate network and load onto the GPU
cnnmlem = MLEM_CNN_Net(cnn, true_sinogram_torch, core_iterations).to(device)
mlemcnn_recon = cnnmlem(true_sinogram_torch)
```

Unrolled EM reconstruction example

Brief, simple
CODING EXAMPLE
Jupyter Notebook

Goal: gain familiarity with how to unroll an iterative algorithm with trainable parameters

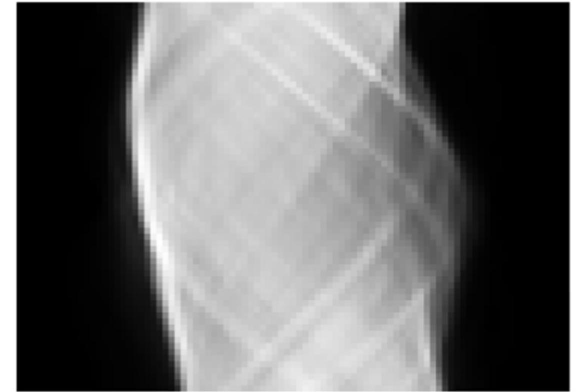
Unrolled EM

```
#=================================================================
# Now create a recon network class: MLEM
#=================================================================
class MLEM_Net(nn.Module):
    def __init__(self, num_iterations, cnn, sino_for_reconstruction):
        super(MLEM_Net, self).__init__()
        self.num_iterations = num_iterations
        self.sino_ones  = torch.ones_like(sino_for_reconstruction)
        self.sens_image = bp_system_torch(self.sino_ones, sys_mat, nxd, nrd, nphi)
        self.cnn    = cnn
        self.prelu = nn.PReLU()
    def forward(self, sino_for_reconstruction):
        recon_image = torch.ones(nxd,nxd).to(device)
        for it in range(self.num_iterations):
            fpsino        = fp_system_torch(recon_image, sys_mat, nxd, nrd, nphi)
            ratio_sino    = sino_for_reconstruction / (fpsino + 1e-10)
            bp_ratio_sino = bp_system_torch(ratio_sino, sys_mat, nxd, nrd, nphi)
            recon_image   = recon_image * bp_ratio_sino / (self.sens_image + 1e-10)
            # Inter update regularisation
            recon_image   = torch.abs(self.cnn(recon_image))
        return recon_image, fpsino, ratio_sino, bp_ratio_sino

number_MLEM_iterations = 2
sinogram_to_use_torch = true_sinogram_torch
# instantiate the class - create an object
mlemnet = MLEM_Net(number_MLEM_iterations, cnn, sinogram_to_use_torch).to(device)
```

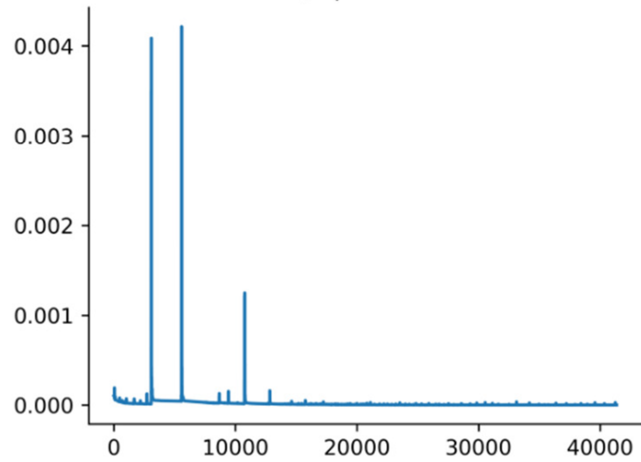
Unrolled EM

Measured data

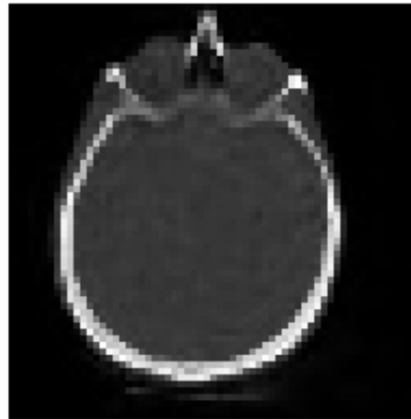
Forward projection



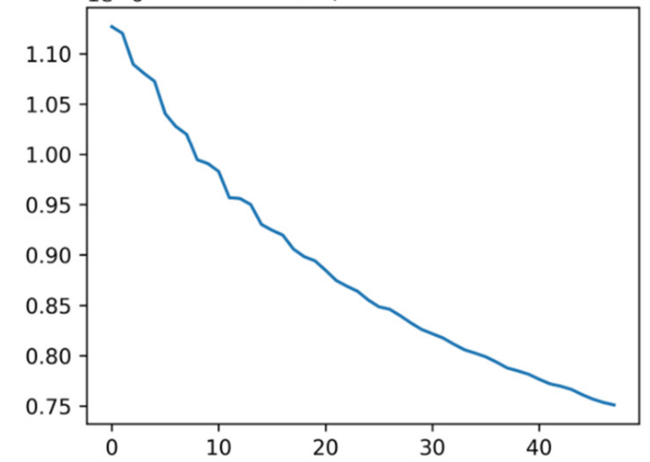
Loss, epoch 41400



Recon 41400



Loss, epoch 41400



Thank you

andrew.reader@kcl.ac.uk

IEEE TRANSACTIONS ON RADIATION AND PLASMA MEDICAL SCIENCES, VOL. 5, NO. 1, JANUARY 2021 1

Deep Learning for PET Image Reconstruction

Andrew J. Reader, Guillaume Corda, Abolfazl Mehranian, Casper da Costa-Luis, *Student Member, IEEE*, Sam Ellis, and Julia A. Schnabel, *Senior Member, IEEE*

Journal of Nuclear Medicine October 2021, 62 (10) 1330-1333; DOI: https://doi.org/10.2967/jnumed.121.262303

HOT TOPICS

Artificial Intelligence for PET Image Reconstruction

Andrew J. Reader[1] and Georg Schramm[2]

[1] *School of Biomedical Engineering and Imaging*
Department of Imaging and Pathology, KU/UZ,

Andrew Reader

1.93K subscribers

HOME VIDEOS PLAYLISTS COMMUNITY CHANNELS