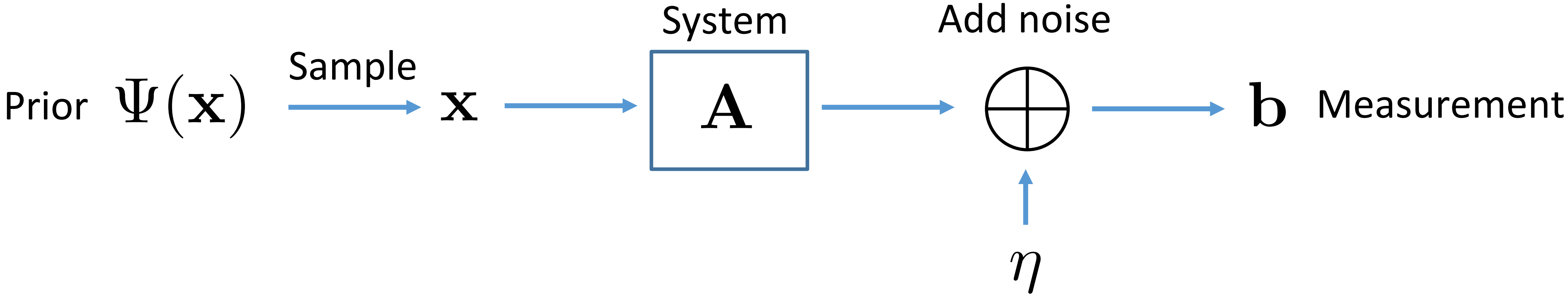
Problem Session 6

Topics

- Overview of Inverse Problems
- Iterative Methods
  - Deconvolution with Adam and Total Variation Regularization
  - Deconvolution with ADMM and DnCNN
  - Single-Pixel Imaging with ADMM
- Implementation Tips

Overview of Inverse Problems



Goal: Recover x from b!

Basic Inverse Problem

$$\underset{x}{\text{minimize}} \quad \underbrace{\frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2}_{\text{Data Term}} + \lambda \underbrace{\Psi(\mathbf{x})}_{\text{Prior}}$$

Controls strength of prior

- See notes for full explanation!

Task 1: Deconvolution with Adam and TV

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{Cx} - \mathbf{b}\|^2 + \lambda \mathrm{TV}(\mathbf{x})$$

- Algorithm: **Gradient Descent**
  - Adam: https://arxiv.org/abs/1412.6980
- Prior: **Total Variation**
  - Anisotropic: $$\|\mathbf{D_x x}\|_1 + \|\mathbf{D_y x}\|_1 = \left\| \begin{bmatrix} \mathbf{D_x x} \\ \mathbf{D_y x} \end{bmatrix} \right\|_1$$
  - Isotropic: $$\left\| \begin{bmatrix} \mathbf{D_x x} \\ \mathbf{D_y x} \end{bmatrix} \right\|_{2,1} = \sum_{i=1}^{N} \sqrt{(\mathbf{D_x x})_i^2 + (\mathbf{D_y x})_i^2}$$

$$\mathbf{D}_x\mathbf{x} \;\Leftrightarrow\; x * d_x,\; d_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{D}_y\mathbf{x} \;\Leftrightarrow\; x * d_y,\; d_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{6}$$

Task 1: Deconvolution with Adam and TV

OTF is fourier transform of PSF c

$$c * x = \mathcal{F}^{-1}\{\mathcal{F}\{x\} \cdot \mathcal{F}\{c\}\}$$

```
    # otf of blur kernel and forward image formation model
    cFT = psf2otf(c, np.shape(b))
    cFT = torch.from_numpy(cFT).to(device)
    Afun = λ x: torch.real(torch.fft.ifft2(torch.fft.fft2(x) * cFT))

    # finite differences kernels and corresponding otfs
    dx = np.array([[-1., 1.]])
    dy = np.array([[-1.], [1.]])
    dxFT = torch.from_numpy(psf2otf(dx, b.shape)).to(device)
    dyFT = torch.from_numpy(psf2otf(dy, b.shape)).to(device)
    dxyFT = torch.stack((dxFT, dyFT), axis=0)
```

$$\mathbf{D}_x\mathbf{x} \;\Leftrightarrow\; x * d_x,\; d_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{D}_y\mathbf{x} \;\Leftrightarrow\; x * d_y,\; d_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

$$\begin{bmatrix} \mathbf{D_x} \\ \mathbf{D_y} \end{bmatrix}$$ but in the Fourier domain!


Target Image
Blurry and Noisy Image
Adam + Anisotropic TV, PSNR: 26.3
Adam + Isotropic TV, PSNR: 26.3

Task 2.1: Image Deconvolution using ADMM

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{Cx}-\mathbf{b}\|^2 + \lambda \text{TV}(\mathbf{x})$$

- Algorithm: **ADMM** $\quad f(\mathbf{x}) + g(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{Dx}-\mathbf{z}+\mathbf{u}\| - \frac{\rho}{2}\|\mathbf{u}\|$
- Prior: **Total Variation**
  - x-update: $$x \leftarrow \text{prox}_{f,\rho}(\mathbf{v}) = \underset{\{\mathbf{x}\}}{\arg\min} \frac{1}{2}\|\mathbf{Cx}-\mathbf{b}\|_2^2 + \frac{\rho}{2}\|\mathbf{Dx}-\mathbf{v}\|_2^2, \quad \mathbf{v} = \mathbf{z}-\mathbf{u}$$
    $$x \leftarrow \left(\mathbf{C}^T\mathbf{C} + \rho\mathbf{D}^T\mathbf{D}\right)^{-1}\left(\mathbf{C}^T\mathbf{b} + \rho\mathbf{D}^T\mathbf{v}\right)$$
  - z-update, u-update: …
- Do x-update in **Fourier** domain.

$$\left(\mathbf{C}^T\mathbf{C} + \rho\mathbf{D}^T\mathbf{D}\right) \Leftrightarrow \mathcal{F}^{-1}\left\{\mathcal{F}\{c\}^* \cdot \mathcal{F}\{c\} + \rho\left(\mathcal{F}\{d_x\}^* \cdot \mathcal{F}\{d_x\} + \mathcal{F}\{d_y\}^* \cdot \mathcal{F}\{d_y\}\right)\right\}$$
$$\left(\mathbf{C}^T\mathbf{b} + \rho\mathbf{D}^T\mathbf{v}\right) \Leftrightarrow \mathcal{F}^{-1}\left\{\mathcal{F}\{c\}^* \cdot \mathcal{F}\{b\} + \rho\left(\mathcal{F}\{d_x\}^* \cdot \mathcal{F}\{v_1\} + \mathcal{F}\{d_y\}^* \cdot \mathcal{F}\{v_2\}\right)\right\}$$

Please look at the notes for more details.

Task 2.1: Image Deconvolution using ADMM

Conjugation in Fourier domain = time-reversal (matrix transpose) in spatial domain

```
# Blur kernel
cFT = psf2otf(c, b.shape)
cTFT = np.conj(cFT)

# finite differences kernels and corresponding otfs
dx = np.array([[-1., 1.]])
dy = np.array([[-1.], [1.]])
dxFT = psf2otf(dx, b.shape)
dyFT = psf2otf(dy, b.shape)
dxTFT = np.conj(dxFT)
dyTFT = np.conj(dyFT)
dxyFT = np.stack((dxFT, dyFT), axis=0)
dxyTFT = np.stack((dxTFT, dyTFT), axis=0)

# Fourier transform of b
bFT = fft2(b)
```

$\begin{bmatrix} \mathbf{D_x} \\ \mathbf{D_y} \end{bmatrix}$ in the Fourier domain

$\begin{bmatrix} \mathbf{D_x}^T & \mathbf{D_y}^T \end{bmatrix}$ in the Fourier domain

Task 2.2: Image Deconvolution using ADMM

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{Cx}-\mathbf{b}\|^2 + \lambda\Psi_{\text{DnCNN}}(\mathbf{x})$$

- Algorithm: **ADMM** $\quad f(\mathbf{x}) + g(\mathbf{z}) + \frac{\rho}{2}\|\mathbf{Dx}-\mathbf{z}+\mathbf{u}\| - \frac{\rho}{2}\|\mathbf{u}\|$
- Prior 2: **DnCNN**
  - x-update: $\quad x \leftarrow \text{prox}_{f,\rho}(\mathbf{v}) = \underset{\{\mathbf{x}\}}{\arg\min}\frac{1}{2}\|\mathbf{Cx}-\mathbf{b}\|_2^2 + \frac{\rho}{2}\|\mathbf{x}-\mathbf{v}\|_2^2, \quad \mathbf{v}=\mathbf{z}-\mathbf{u}$

$$x \leftarrow \left(\mathbf{C}^T\mathbf{C}+\rho\mathbf{I}\right)^{-1}\left(\mathbf{C}^T\mathbf{b}+\rho\mathbf{v}\right)$$

  - z-update, u-update: …
- Do x-update in Fourier domain.

$$\left(\mathbf{C}^T\mathbf{C}+\rho\mathbf{I}\right) \Leftrightarrow \mathcal{F}^{-1}\{\mathcal{F}\{c\}^* \cdot \mathcal{F}\{c\} + \rho\}$$
$$\left(\mathbf{C}^T\mathbf{b}+\rho\mathbf{v}\right) \Leftrightarrow \mathcal{F}^{-1}\{\mathcal{F}\{c\}^* \cdot \mathcal{F}\{b\} + \rho\mathcal{F}\{v\}\}$$

Please look at the notes for more details.

Task 2.2: Image Deconvolution using ADMM

Initialize network

Load network weights

Enable test-time behavior

Disable gradient accumulation

Move to GPU

```
# load pre-trained DnCNN model
model = net(in_nc=1, out_nc=1, nc=64, nb=17, act_mode='R')
model.load_state_dict(torch.load(denoiser_model_filename), strict=True)
model.eval()
for k, v in model.named_parameters():
    v.requires_grad = False
model = model.to(device)
```

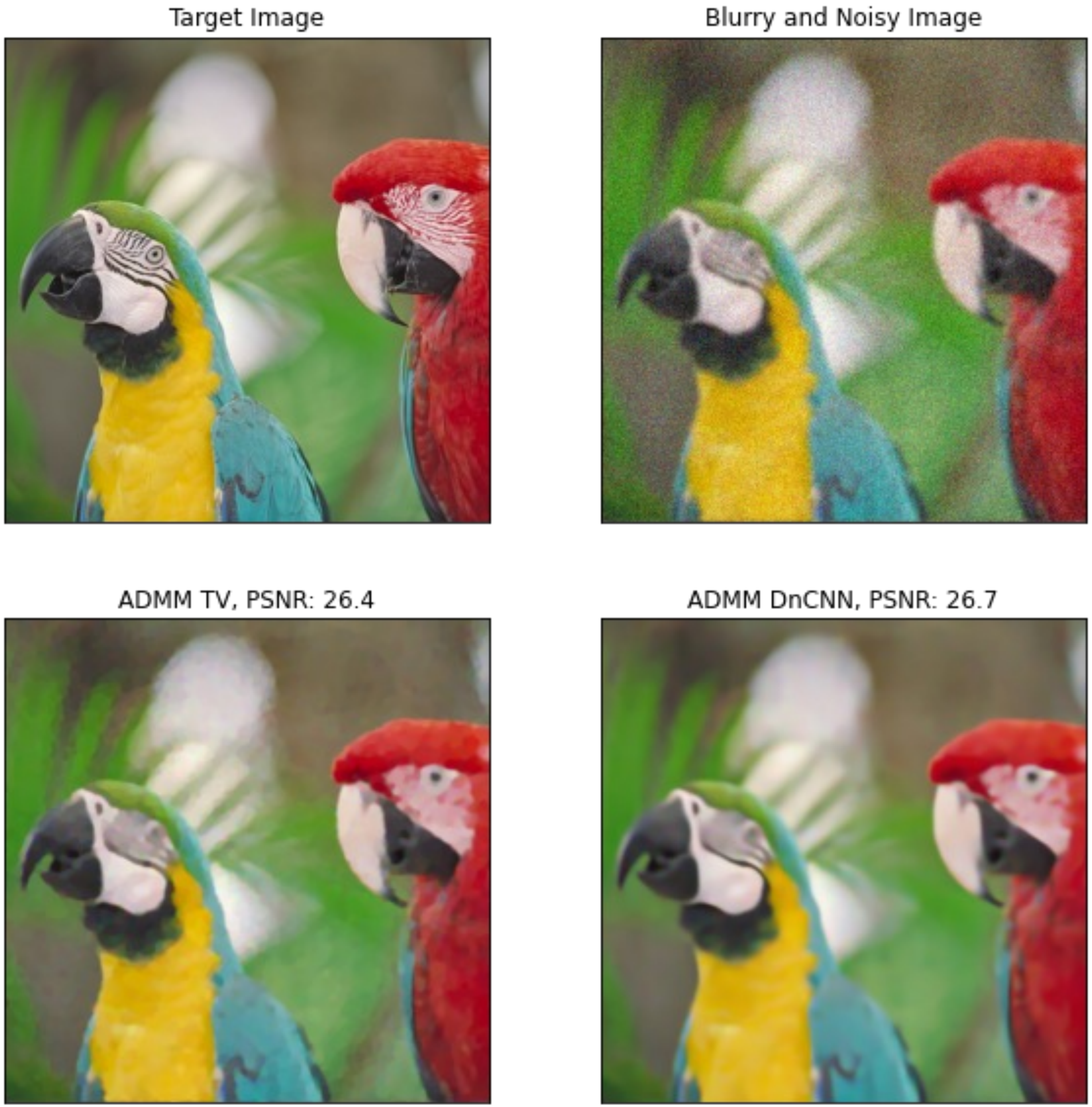
Convert numpy array to torch tensor

Run model

```
# run DnCNN denoiser
x_tensor = torch.reshape(torch.from_numpy(x).float().to(device), (1,1,x.shape[0],x.shape[1]))
x_tensor_denoised = model(x_tensor)

z = torch.squeeze(x_tensor_denoised).cpu().numpy()
```

Convert output back to numpy


Target Image
Blurry and Noisy Image
ADMM TV, PSNR: 26.4
ADMM DnCNN, PSNR: 26.7

Task 3.1: Single-Pixel Imaging using CG

$$\begin{aligned} &\underset{x}{\text{minimize}} && \|\mathbf{x}\|_2 \\ &\text{subject to} && \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned}$$

- Algorithm: **Conjugate Gradient (CG)**
  - Least Norm Analytic Solution:

$$\mathbf{x} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b}$$

  - Solve in 2 stages:
    - Find u: $\mathbf{u} = \left(\mathbf{A}\mathbf{A}^T\right)^{-1}\mathbf{b} \Leftrightarrow \left(\mathbf{A}\mathbf{A}^T\right)\mathbf{u} = \mathbf{b}$
    - Obtain $x = A^T u$

Task 3.1: Single-Pixel Imaging using CG

```
LinearOperator(size, matvec=method)
```

1. Size = (number of measurements, number of measurements)
2. Method = a function that implements the operation AA^T

   You are given Afun and Atfun already, but be careful about input and output shapes.

```
cg(A=LinearOperator, b=b, maxiter=num_iters, tol=cg_tolerance)
```

Use the linear operator in the cg call.

Don't forget to apply the final A^T!

Task 3.2: Single-Pixel Imaging using ADMM + TV

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{Ax}-\mathbf{b}\|^2 + \lambda \mathrm{TV}(\mathbf{x})$$

- Algorithm: **ADMM**
- Prior 1: **Total Variation**
  - x-update: $\mathbf{x} \leftarrow \mathrm{prox}_{\|\cdot\|_2,\rho}(\mathbf{v}) = \underset{\{\mathbf{x}\}}{\arg\min}\, \frac{1}{2}\|\mathbf{Ax}-\mathbf{b}\|_2^2 + \frac{\rho}{2}\|\mathbf{Dx}-\mathbf{v}\|_2^2, \quad \mathbf{v} = \mathbf{z} - \mathbf{u}$

$$\mathbf{x} \leftarrow \boxed{\Big(\underbrace{\mathbf{A}^T\mathbf{A} + \rho\mathbf{D}^T\mathbf{D}}_{\tilde{\mathbf{A}}}\Big)^{-1}} \Big(\underbrace{\mathbf{A}^T\mathbf{b} + \rho\mathbf{D}^T\mathbf{v}}_{\tilde{b}}\Big)$$

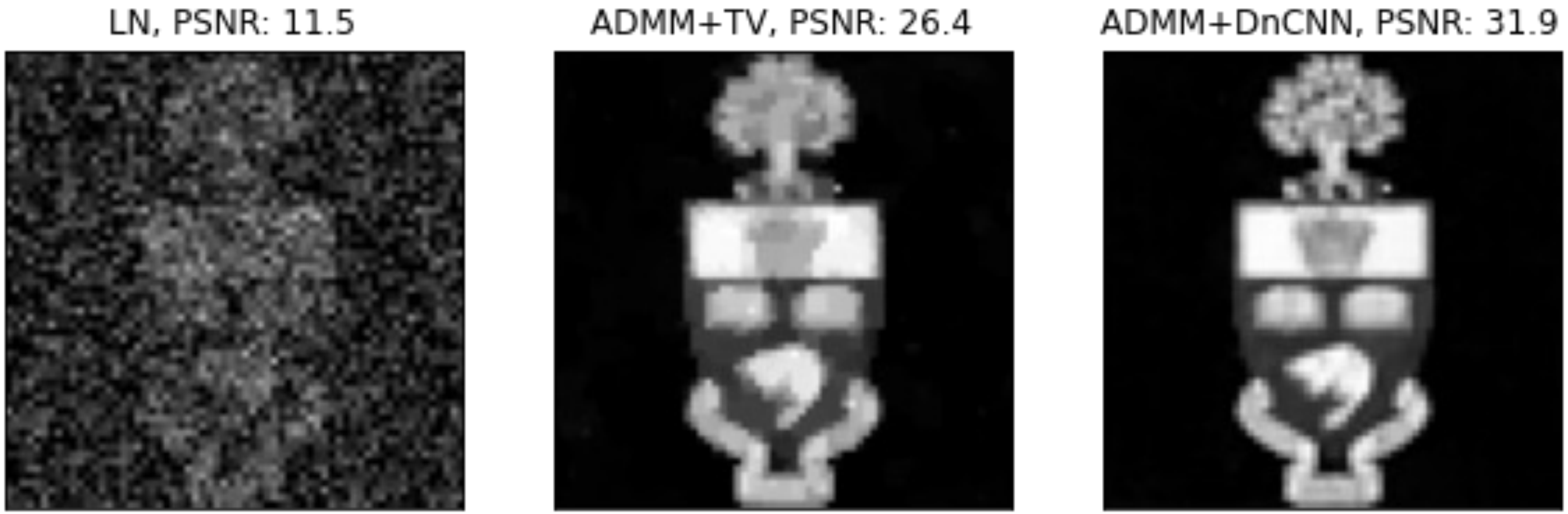
Unlike Deconvolution, the red part doesn't have Fourier counterpart.

  - z-update, u-update: …
- Use Conjugate Gradients to do the x-update. Combine the least squares objectives first.

Task 3.3: Single-Pixel Imaging using ADMM + DnCNN

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{Ax}-\mathbf{b}\|^2 + \lambda\Psi_{\text{DnCNN}}(\mathbf{x})$$

- Algorithm: **ADMM**
- Prior 2: **DnCNN**
  - x-update: $$\mathbf{x} \leftarrow \text{prox}_{\|\cdot\|_2,\rho}(\mathbf{v}) = \underset{\{\mathbf{x}\}}{\arg\min} \frac{1}{2}\|\mathbf{Ax}-\mathbf{b}\|_2^2 + \frac{\rho}{2}\|\mathbf{x}-\mathbf{v}\|_2^2, \quad \mathbf{v}=\mathbf{z}-\mathbf{u}$$
    $$\mathbf{x} \leftarrow \boxed{\Big(\underbrace{\mathbf{A}^T\mathbf{A}+\rho\mathbf{I}}_{\tilde{\mathbf{A}}}\Big)^{-1}}\Big(\underbrace{\mathbf{A}^T\mathbf{b}+\rho\mathbf{v}}_{\tilde{b}}\Big)$$
  - z-update, u-update:…
- Use Conjugate Gradients to do the x-update. Combine the least squares objectives first.


LN, PSNR: 11.5
ADMM+TV, PSNR: 26.4
ADMM+DnCNN, PSNR: 31.9

Good luck with the homework!