Lab 2: ISTA Deblurring
Pixel vs. Multi-Level Wavelet Sparsity

Advanced Image Processing

Goal. Compare ISTA with an L1 penalty in pixel space vs. in a multi-level Haar wavelet domain. Run blocks in order. Edit only lines marked `# EDIT`. You collate and compare results yourself.

Dependency note. Wavelet blocks use `pytorch_wavelets`. Install once: `pip install pytorch_wavelets`.

Block 0: Setup and helpers (run once)

```
# Block 0: imports, image I/O, blur operators, metrics, Lipschitz
import os; os.makedirs("results", exist_ok=True)
import math
import torch, torch.nn.functional as F
import torchvision.transforms as T
import matplotlib.pyplot as plt
import numpy as np
from skimage import data
from PIL import Image

device = "cuda" if torch.cuda.is_available() else "cpu"

# Optional LPIPS
USE_LPIPS = True
try:
    import lpips
    lpips_fn = lpips.LPIPS(net='alex').to(device) if USE_LPIPS else None
except Exception:
    lpips_fn = None
    print("LPIPS disabled or missing. pip install lpips to enable.")

def load_rgb01():
    img = Image.fromarray(data.astronaut())
    x = T.ToTensor()(img).unsqueeze(0)   # [1,3,H,W] in [0,1]
    return x.to(device)

def psnr(ref, out, eps=1e-12):
    mse = F.mse_loss(out, ref)
    return float(10.0 * torch.log10(1.0 / (mse + eps)))

def lpips_score(ref, out):
    if lpips_fn is None: return None
    with torch.no_grad():
        return float(lpips_fn(ref*2-1, out*2-1).mean().item())

def gaussian_kernel_2d(size=15, std=3.0):
    ax = np.arange(size) - (size-1)/2.0
    xx, yy = np.meshgrid(ax, ax)
    k = np.exp(-(xx**2 + yy**2) / (2.0*std**2))
    k = k / k.sum()
    return k.astype(np.float32)

def H_blur(x, kernel_np):
    # Circular convolution per-channel
    k = torch.from_numpy(kernel_np).to(x.device, dtype=x.dtype).unsqueeze(0).unsqueeze(0)
    pad = kernel_np.shape[0] // 2
    out = torch.zeros_like(x)
    for c in range(x.shape[1]):
        ch = x[:, c:c+1]
        chp = F.pad(ch, (pad,pad,pad,pad), mode='circular')
        out[:, c:c+1] = F.conv2d(chp, k, padding=0)
    return out

def HT_blur(x, kernel_np):
    # For symmetric Gaussian PSF with circular bc, H^T = H
    return H_blur(x, kernel_np)

def lipschitz_HtH(kernel_np, shape_hw):
    H, W = shape_hw
    hk, wk = kernel_np.shape
    if hk > H or wk > W:
        raise ValueError("Kernel larger than image.")
    pad = torch.zeros((H, W), dtype=torch.float32, device=device)
    pad[:hk, :wk] = torch.from_numpy(kernel_np).to(device)
    pad = torch.roll(pad, shifts=(-hk//2, -wk//2), dims=(0,1))
    K = torch.fft.fft2(pad)
    return float((torch.abs(K)**2).max().item())
```

Block 1: Generate blurred+noisy data

```
# Block 1: PSF, blur, AWGN
x_clean = load_rgb01()

k_size = 15   # EDIT if desired
k_std  = 3.0  # EDIT if desired
k_np = gaussian_kernel_2d(size=k_size, std=k_std)
xb = H_blur(x_clean, k_np)

sigma = 0.01  # EDIT if desired
y = (xb + sigma * torch.randn_like(xb)).clamp(0,1)

# Enforce even H,W for wavelets; use same crop for both methods
H_img, W_img = x_clean.shape[-2:]
H_even = H_img if H_img % 2 == 0 else H_img - 1
W_even = W_img if W_img % 2 == 0 else W_img - 1
x_clean_c = x_clean[:, :, :H_even, :W_even]
y_c       = y[:,       :, :H_even, :W_even]

print("Blur-only  PSNR:", psnr(x_clean, xb), "LPIPS:", lpips_score(x_clean, xb))
print("Observed   PSNR:", psnr(x_clean, y),  "LPIPS:", lpips_score(x_clean, y))
plt.figure(figsize=(12,3))
plt.subplot(1,4,1); plt.imshow(x_clean[0].permute(1,2,0).cpu().clip(0,1));  plt.axis('off
    '); plt.title("Clean")
plt.subplot(1,4,2); plt.imshow(xb[0].permute(1,2,0).cpu().clip(0,1));       plt.axis('off
    '); plt.title("Blurred")
```

```
plt.subplot(1,4,3); plt.imshow(y[0].permute(1,2,0).cpu().clip(0,1));        plt.axis('off
    '); plt.title("Blur+AWGN")
plt.subplot(1,4,4); plt.imshow(k_np, vmin=0, vmax=k_np.max());               plt.axis('
    off'); plt.title("PSF")
plt.tight_layout(); plt.savefig("results/lab2_data.png", dpi=200); plt.close()
```

Block 2P: Pixel-domain ISTA

```
# Block 2P: ISTA with L1 on pixels
L   = lipschitz_HtH(k_np, (H_even, W_even))
tau = 0.9 / L         # EDIT if desired (step size)
lam = 0.01            # EDIT if desired (pixel-domain L1)
iters = 200           # EDIT if desired

x_pix = y_c.clone()
for _ in range(iters):
    grad  = HT_blur(H_blur(x_pix, k_np) - y_c, k_np)
    x_pix = torch.sign(x_pix - tau*grad) * torch.clamp(torch.abs(x_pix - tau*grad) - lam*
    tau, min=0.0)

print("Pixel-ISTA PSNR:", psnr(x_clean_c, x_pix), "LPIPS:", lpips_score(x_clean_c, x_pix)
    )
plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(y_c[0].permute(1,2,0).cpu().clip(0,1));  plt.axis('off');
    plt.title("Input y (crop)")
plt.subplot(1,3,2); plt.imshow(x_pix[0].permute(1,2,0).cpu().clip(0,1)); plt.axis('off');
     plt.title("Pixel ISTA")
plt.subplot(1,3,3); plt.imshow(x_clean_c[0].permute(1,2,0).cpu().clip(0,1)); plt.axis('
    off'); plt.title("Clean (crop)")
plt.tight_layout(); plt.savefig("results/lab2_pixel_ista.png", dpi=200); plt.close()
```

Block 2W: Multi-level wavelet ISTA (Haar, pytorch_wavelets)

```
# Block 2W: ISTA with L1 on multi-level Haar details (LH, HL, HH only)
# Requires: pip install pytorch_wavelets
from pytorch_wavelets import DWTForward, DWTInverse

J = 3                   # EDIT if desired (levels)
lam_wavelet = 0.01      # EDIT if desired (wavelet-domain L1)
tau = tau               # reuse same tau as pixel ISTA
iters = iters           # reuse same iterations

xfm = DWTForward(J=J, wave='haar').to(device)
ifm = DWTInverse(wave='haar').to(device)

x_wav = y_c.clone()
for _ in range(iters):
    grad = HT_blur(H_blur(x_wav, k_np) - y_c, k_np)
    v    = x_wav - tau * grad
    LL, details = xfm(v)   # details is a list over scales; each is (B,C,3,H,W) or (B,3,C
    ,H,W)

    new_details = []
    for yh in details:
        # Handle orientation dimension (3) whether it is axis=2 or axis=1
        if yh.ndim != 5:
            raise RuntimeError(f"Unexpected DWT shape: {yh.shape}")
        if yh.shape[2] == 3:   # (B,C,3,H,W)
            LH, HL, HH = yh[:,:,0], yh[:,:,1], yh[:,:,2]
            t = lam_wavelet * tau
            LH = torch.sign(LH) * torch.clamp(LH.abs() - t, min=0.0)
            HL = torch.sign(HL) * torch.clamp(HL.abs() - t, min=0.0)
            HH = torch.sign(HH) * torch.clamp(HH.abs() - t, min=0.0)
            yh = torch.stack([LH, HL, HH], dim=2)
        elif yh.shape[1] == 3: # (B,3,C,H,W)
            LH, HL, HH = yh[:,0], yh[:,1], yh[:,2]
            t = lam_wavelet * tau
            LH = torch.sign(LH) * torch.clamp(LH.abs() - t, min=0.0)
            HL = torch.sign(HL) * torch.clamp(HL.abs() - t, min=0.0)
            HH = torch.sign(HH) * torch.clamp(HH.abs() - t, min=0.0)
            yh = torch.stack([LH, HL, HH], dim=1)
        else:
            raise RuntimeError(f"Cannot locate orientation dim in {yh.shape}")
        new_details.append(yh)

    x_wav = ifm((LL, new_details))   # no clamp inside loop

print("Wavelet-ISTA PSNR:", psnr(x_clean_c, x_wav), "LPIPS:", lpips_score(x_clean_c,
    x_wav))
plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(y_c[0].permute(1,2,0).cpu().clip(0,1));   plt.axis('off');
     plt.title("Input y (crop)")
plt.subplot(1,3,2); plt.imshow(x_wav[0].permute(1,2,0).cpu().clip(0,1));  plt.axis('off')
    ; plt.title("Wavelet ISTA")
plt.subplot(1,3,3); plt.imshow(x_clean_c[0].permute(1,2,0).cpu().clip(0,1)); plt.axis('
    off'); plt.title("Clean (crop)")
plt.tight_layout(); plt.savefig("results/lab2_wavelet_ista.png", dpi=200); plt.close()

# Optional equivalence check when lam_wavelet = 0
if lam_wavelet == 0:
    diff = (x_pix - x_wav).abs().max().item()
    print("Max |pixel - wavelet| (lam_wavelet=0):", diff)
```

Tasks

- Compare ISTA in pixel-space with ISTA enforcing wavelet sparsity for different blur (kernel size and width), noise-levels and regularization weights. What do you notice?
- Create a small table you compile with PSNR (and LPIPS if enabled) for your chosen settings.
- 5–8 sentences: compare pixel vs wavelet sparsity; effect of J and lam_wavelet; what changes when also shrinking LL.