DT095A – Lab 1b
Deblurring Lab: Inverse and Wiener (Chunked, Inline)

How to use. Run each code block in order. Edit only the lines marked `# EDIT` if you want to change settings. Each block produces one figure in `results/` and prints PSNR (and LPIPS if installed). You will collate and compare results yourself.

Block 0: Shared setup (run once)

```
# Block 0: imports, image, helpers (run once)
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
import lpips

lpips_fn = lpips.LPIPS(net="alex").to(device)


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())

# Noise models
def add_awgn(x, sigma=0.05):
    return (x + sigma * torch.randn_like(x)).clamp(0,1)

def add_salt_pepper(x, amount=0.02):
    prob = torch.rand_like(x)
    y = x.clone()
    y[prob < amount/2] = 0.0
    y[(prob >= amount/2) & (prob < amount)] = 1.0
```

```
    return y

# PSF and blur
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 convolve_circular(x, kernel_np):
    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

# FFT helpers and deconvolution
def padded_psf_fft(kernel_np, shape_hw, device):
    hk, wk = kernel_np.shape
    H, W = shape_hw
    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))
    return torch.fft.fft2(pad)

def inverse_filter(x, kernel_np, eps=1e-3):
    B, C, H, W = x.shape
    K = padded_psf_fft(kernel_np, (H,W), x.device)
    out = torch.zeros_like(x)
    denom = K + eps
    for c in range(C):
        Xf = torch.fft.fft2(x[:, c])
        rec = torch.fft.ifft2(Xf / denom)
        out[:, c] = torch.real(rec)
    return out.clamp(0,1)

def wiener_filter(x, kernel_np, Kreg=1e-3):
    B, C, H, W = x.shape
    K = padded_psf_fft(kernel_np, (H,W), x.device)
    Hf = torch.conj(K) / (torch.abs(K)**2 + Kreg)
    out = torch.zeros_like(x)
    for c in range(C):
        Xf = torch.fft.fft2(x[:, c])
        rec = torch.fft.ifft2(Hf * Xf)
        out[:, c] = torch.real(rec)
    return out.clamp(0,1)

x_clean = load_rgb01()
print("Image loaded on", device, "shape:", list(x_clean.shape))
```

Block 1: Build PSF and blur

```
# Block 1: choose PSF size/std and create a blurred image
```

```
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 = convolve_circular(x_clean, k_np)

print("Blur-only PSNR:", psnr(x_clean, xb), "LPIPS:", lpips_score(x_clean, xb))
plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(x_clean[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Clean")
plt.subplot(1,3,2); plt.imshow(xb[0].permute(1,2,0).cpu());       plt.axis("off"); plt.
    title("Blurred")
plt.subplot(1,3,3); plt.imshow(k_np, vmin=0, vmax=k_np.max());     plt.axis("off"); plt.
    title("PSF")
plt.tight_layout(); plt.savefig("results/blur_only.png", dpi=200); plt.close()
```

Block 2: Add AWGN to blurred image

```
# Block 2: add AWGN to blurred image
sigma = 0.01   # EDIT if desired
y = add_awgn(xb, sigma=sigma)

print("Blur+AWGN PSNR:", psnr(x_clean, y), "LPIPS:", lpips_score(x_clean, y))
plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(x_clean[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Clean")
plt.subplot(1,3,2); plt.imshow(y[0].permute(1,2,0).cpu());        plt.axis("off"); plt.
    title("Blur+AWGN")
plt.subplot(1,3,3); plt.imshow((y-xb)[0].permute(1,2,0).cpu()*0.5+0.5); plt.axis("off");
    plt.title("Added noise (scaled)")
plt.tight_layout(); plt.savefig("results/blur_awgn.png", dpi=200); plt.close()
```

Block 3: Add salt-and-pepper to blurred image

```
# Block 3: add salt-and-pepper noise to blurred image
amount = 0.01   # EDIT if desired
y = add_salt_pepper(xb, amount=amount)

print("Blur+SaltPepper PSNR:", psnr(x_clean, y), "LPIPS:", lpips_score(x_clean, y))
plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(x_clean[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Clean")
plt.subplot(1,3,2); plt.imshow(y[0].permute(1,2,0).cpu());        plt.axis("off"); plt.
    title("Blur+SaltPepper")
plt.subplot(1,3,3); plt.imshow((y!=xb).float()[0].mean(0).cpu(), vmin=0, vmax=1); plt.
    axis("off"); plt.title("Changed pixels (mean)")
plt.tight_layout(); plt.savefig("results/blur_saltpepper.png", dpi=200); plt.close()
```

Block 4: Inverse filter (single setting)

```
# Block 4: inverse filter for current y and k_np
eps = 1e-3   # EDIT if desired
rec = inverse_filter(y, k_np, eps=eps)
```

```
print("Inverse PSNR:", psnr(x_clean, rec), "LPIPS:", lpips_score(x_clean, rec))
plt.figure(figsize=(12,3))
plt.subplot(1,4,1); plt.imshow(x_clean[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Clean")
plt.subplot(1,4,2); plt.imshow(xb[0].permute(1,2,0).cpu());       plt.axis("off"); plt.
    title("Blur")
plt.subplot(1,4,3); plt.imshow(y[0].permute(1,2,0).cpu());        plt.axis("off"); plt.
    title("Blur+Noise")
plt.subplot(1,4,4); plt.imshow(rec[0].permute(1,2,0).cpu());      plt.axis("off"); plt.
    title("Inverse")
plt.tight_layout(); plt.savefig("results/inverse.png", dpi=200); plt.close()
```

Block 5: Wiener filter (single setting)

```
# Block 5: Wiener filter for current y and k_np
Kreg = 1e-3   # EDIT if desired
rec = wiener_filter(y, k_np, Kreg=Kreg)

print("Wiener PSNR:", psnr(x_clean, rec), "LPIPS:", lpips_score(x_clean, rec))
plt.figure(figsize=(12,3))
plt.subplot(1,4,1); plt.imshow(x_clean[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Clean")
plt.subplot(1,4,2); plt.imshow(xb[0].permute(1,2,0).cpu());       plt.axis("off"); plt.
    title("Blur")
plt.subplot(1,4,3); plt.imshow(y[0].permute(1,2,0).cpu());        plt.axis("off"); plt.
    title("Blur+Noise")
plt.subplot(1,4,4); plt.imshow(rec[0].permute(1,2,0).cpu());      plt.axis("off"); plt.
    title("Wiener")
plt.tight_layout(); plt.savefig("results/wiener.png", dpi=200); plt.close()
```

Block 6: Minimal comparison driver

```
# Block 6: quick side-by-side for your current settings
rec_inv = inverse_filter(y, k_np, eps=1e-3)   # EDIT if desired
rec_wnr = wiener_filter(y, k_np, Kreg=1e-3)   # EDIT if desired

print("Inverse PSNR:", psnr(x_clean, rec_inv), "LPIPS:", lpips_score(x_clean, rec_inv))
print("Wiener  PSNR:", psnr(x_clean, rec_wnr), "LPIPS:", lpips_score(x_clean, rec_wnr))

plt.figure(figsize=(12,3))
plt.subplot(1,4,1); plt.imshow(x_clean[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Clean")
plt.subplot(1,4,2); plt.imshow(y[0].permute(1,2,0).cpu());        plt.axis("off"); plt.
    title("Blur+Noise")
plt.subplot(1,4,3); plt.imshow(rec_inv[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Inverse")
plt.subplot(1,4,4); plt.imshow(rec_wnr[0].permute(1,2,0).cpu());  plt.axis("off"); plt.
    title("Wiener")
plt.tight_layout(); plt.savefig("results/compare_inv_wiener.png", dpi=200); plt.close()
```

Tasks

- Save figures for blur-only, blur+noise (at least one of AWGN or salt-pepper), inverse, and Wiener.

- Compare different noise and blur-levels (try noiseless as well) and compare inverse and Wiener.
- A small table you compile with PSNR (and LPIPS if enabled) for your chosen settings.
- 5–8 sentences: why inverse amplifies noise, how the Wiener parameter trades detail vs noise, and which settings worked best.