DT095A – Lab 1a
Noise & Spatial Filters — Modify & Compare

Instructions (read first): Run each chunk in order, *one configuration at a time*. Your job is to **change the hyperparameters** in each chunk, re-run, save images, and **record PSNR (and LPIPS if available)**. You will **collate and compare** results yourself.

Deliverables (concise)

- A few saved figures per chunk showing your chosen settings.
- A small table you create (in your notes) with PSNR (and LPIPS if used) for chosen settings.
- 5–8 sentences: which noise–kernel matches worked best and why.

1 Chunk 0 — Minimal setup (run once)

```
# --- CHUNK 0: imports, image, metrics (run once per session) ---
import os; os.makedirs("results", exist_ok=True)
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 toggle
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 unavailable. Set USE_LPIPS=True and `pip install lpips` to enable.")

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

def psnr(ref, out, eps=1e-12):
    mse = F.mse_loss(out, ref)
    return (10 * torch.log10(1.0 / (mse + eps))).item()

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

```
x_clean = load_rgb01()
H, W = x_clean.shape[-2:]
print("Loaded image:", (H, W), "device:", device)
```

2 Chunk 1 — Additive noise: AWGN (modify `sigma`)

Task: Change σ (e.g., 0.03, 0.05, 0.10, 0.20). Save image; record PSNR/LPIPS.

```
# --- CHUNK 1: AWGN ---
def add_awgn(x, sigma=0.10):
    y = x + sigma * torch.randn_like(x)
    return y.clamp(0,1)

sigma = 0.10   # <-- EDIT THIS
y = add_awgn(x_clean, sigma=sigma)
print("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(f"AWGN σ={sigma}")
plt.subplot(1,3,3); plt.imshow((y-x_clean)[0].permute(1,2,0).cpu()*0.5+0.5); plt.axis("
    off"); plt.title("Noise (scaled)")
plt.tight_layout(); plt.savefig(f"results/noise_awgn_sigma{sigma}.png", dpi=200); plt.
    close()
```

3 Chunk 2 — Poisson noise (modify `b`)

Task: Run and observe. How does the scale variable impact the noise? Why is it set to 255 per default?

```
# --- CHUNK 2: Poisson noise ---
def add_poisson(x, scale = 255):
    vals = float(scale)
    noisy = torch.poisson(x * vals) / vals
    return noisy.clamp(0, 1)
y = add_poisson(x_clean, scale = 255)
print("Poisson 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(f"
    Poisson")
plt.subplot(1,3,3); plt.imshow((y-x_clean)[0].permute(1,2,0).cpu()*0.5+0.5); plt.axis("
    off"); plt.title("Noise (scaled)")
plt.tight_layout(); plt.savefig(f"results/noise_poisson.png",); plt.close()
```

4 Chunk 3 — Multiplicative: Speckle (modify `sigma`)

Task: Change σ (e.g., 0.10, 0.20, 0.30). Save; record PSNR/LPIPS. Note multiplicative behavior.

```
# --- CHUNK 3: Speckle noise (multiplicative) ---
def add_speckle(x, sigma=0.20):
    m = 1.0 + sigma * torch.randn_like(x)
```

```
    return (x * m).clamp(0,1)

sigma = 0.20   # <-- EDIT THIS
y = add_speckle(x_clean, sigma=sigma)
print("Speckle 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(f"Speckle σ={sigma}")
plt.subplot(1,3,3); plt.imshow((y-x_clean)[0].permute(1,2,0).cpu()*0.5+0.5); plt.axis("
    off"); plt.title("Noise (scaled)")
plt.tight_layout(); plt.savefig(f"results/noise_speckle_sigma{sigma}.png", dpi=200); plt.
    close()
```

5 Chunk 4 — Impulse: Salt–Pepper (modify `amount`)

Task: Change `amount` (e.g., 0.02, 0.05, 0.10). Save; record PSNR/LPIPS.

```
# --- CHUNK 4: Salt & Pepper (impulse) ---
def add_salt_pepper(x, amount=0.05):
    # independent per-pixel, per-channel
    prob = torch.rand_like(x)
    y = x.clone()
    y[prob < amount/2] = 0.0
    y[(prob >= amount/2) & (prob < amount)] = 1.0
    return y

amount = 0.05   # <-- EDIT THIS
y = add_salt_pepper(x_clean, amount=amount)
print("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(f"SaltPepper amt={amount}")
plt.subplot(1,3,3); plt.imshow((y!=x_clean).float()[0].mean(0).cpu(), vmin=0, vmax=1);
    plt.axis("off"); plt.title("Changed pixels (mean)")
plt.tight_layout(); plt.savefig(f"results/noise_saltpepper_amt{amount}.png", dpi=200);
    plt.close()
```

Short Question (Noise comparison): How do AWGN, Poisson, Speckle, and Salt–Pepper *visually* differ at "similar" PSNR? Note at least one mismatch between PSNR and perceptual impression.

6 Chunk 5 — Mean (Box) filter (modify `k`)

Task: Choose one noisy image from Chunks 1–4; set `k` (e.g., 3, 5, 7, 11). Save; record PSNR/LPIPS.

```
# --- CHUNK 5: Mean filter ---
def mean_filter(x, k=5):
    pad = k//2
    kernel = torch.ones((1,1,k,k), device=x.device) / (k*k)
    kernel = kernel.repeat(x.shape[1], 1, 1, 1)  # per-channel
    xp = F.pad(x, (pad,pad,pad,pad), mode="reflect")
```

```
    return F.conv2d(xp, kernel, groups=x.shape[1])

# PICK YOUR NOISY INPUT (uncomment one or create new)
# noisy = add_awgn(x_clean, sigma=0.10)
# noisy = add_poisson(x_clean, scale=255)
# noisy = add_speckle(x_clean, sigma=0.20)
noisy = add_salt_pepper(x_clean, amount=0.05)

k = 5   # <-- EDIT THIS
den = mean_filter(noisy, k=k)
print(f"Mean(k={k}) PSNR:", psnr(x_clean, den), "LPIPS:", lpips_score(x_clean, den))

plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(noisy[0].permute(1,2,0).cpu()); plt.axis("off"); plt.title
    ("Noisy")
plt.subplot(1,3,2); plt.imshow(den[0].permute(1,2,0).cpu());   plt.axis("off"); plt.title
    (f"Mean k={k}")
plt.subplot(1,3,3); plt.imshow(x_clean[0].permute(1,2,0).cpu()); plt.axis("off"); plt.
    title("Clean")
plt.tight_layout(); plt.savefig(f"results/mean_k{k}.png", dpi=200); plt.close()
```

7 Chunk 6 — Median filter (modify k)

Task: Same workflow as Mean. Expect robustness to impulse noise.

```
# --- CHUNK 6: Median filter ---
def median_filter(x, k=5):
    pad = k//2
    xp = F.pad(x, (pad,pad,pad,pad), mode="reflect")
    patches = xp.unfold(2, k, 1).unfold(3, k, 1)  # [B,C,H,W,k,k]
    patches = patches.contiguous().view(x.shape[0], x.shape[1], x.shape[2], x.shape[3],
    -1)
    return patches.median(dim=-1).values

# PICK YOUR NOISY INPUT
# noisy = add_awgn(x_clean, sigma=0.10)
# noisy = add_poisson(x_clean, b=0.05)
# noisy = add_speckle(x_clean, sigma=0.20)
noisy = add_salt_pepper(x_clean, amount=0.05)

k = 5   # <-- EDIT THIS
den = median_filter(noisy, k=k)
print(f"Median(k={k}) PSNR:", psnr(x_clean, den), "LPIPS:", lpips_score(x_clean, den))

plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(noisy[0].permute(1,2,0).cpu()); plt.axis("off"); plt.title
    ("Noisy")
plt.subplot(1,3,2); plt.imshow(den[0].permute(1,2,0).cpu());   plt.axis("off"); plt.title
    (f"Median k={k}")
plt.subplot(1,3,3); plt.imshow(x_clean[0].permute(1,2,0).cpu()); plt.axis("off"); plt.
    title("Clean")
plt.tight_layout(); plt.savefig(f"results/median_k{k}.png", dpi=200); plt.close()
```

8 Chunk 7 — Bicubic kernel (modify a)

Task: Change the cubic parameter a (e.g., $-0.5, -0.75, -1.0$). Note smoothing vs. ringing.

```
# --- CHUNK 7: Bicubic-like 4x4 kernel ---
def cubic_weight(x, a=-0.5):
    ax = abs(x)
    if ax < 1:  return (a+2)*ax**3 - (a+3)*ax**2 + 1
    if ax < 2:  return a*ax**3 - 5*a*ax**2 + 8*a*ax - 4*a
    return 0.0

def bicubic_filter(x, a=-0.5):
    xs = [-1.5, -0.5, 0.5, 1.5]
    w1 = torch.tensor([cubic_weight(t, a=a) for t in xs], device=x.device, dtype=x.dtype)
    w1 = w1 / (w1.sum() + 1e-12)
    k2d = (w1[:,None] @ w1[None,:]).unsqueeze(0).unsqueeze(0)  # [1,1,4,4]
    k2d = k2d.repeat(x.shape[1], 1, 1, 1)
    pad_l, pad_r = 1, 2
    xp = F.pad(x, (pad_l,pad_r,pad_l,pad_r), mode="reflect")
    return F.conv2d(xp, k2d, groups=x.shape[1])

# PICK YOUR NOISY INPUT
noisy = add_awgn(x_clean, sigma=0.10)
# noisy = add_poisson(x_clean, b=0.05)
# noisy = add_speckle(x_clean, sigma=0.20)
# noisy = add_salt_pepper(x_clean, amount=0.05)

a = -0.5   # <-- EDIT THIS
den = bicubic_filter(noisy, a=a)
print(f"Bicubic(a={a}) PSNR:", psnr(x_clean, den), "LPIPS:", lpips_score(x_clean, den))

plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(noisy[0].permute(1,2,0).cpu()); plt.axis("off"); plt.title
    ("Noisy")
plt.subplot(1,3,2); plt.imshow(den[0].permute(1,2,0).cpu());   plt.axis("off"); plt.title
    (f"Bicubic a={a}")
plt.subplot(1,3,3); plt.imshow(x_clean[0].permute(1,2,0).cpu()); plt.axis("off"); plt.
    title("Clean")
plt.tight_layout(); plt.savefig(f"results/bicubic_a{a}.png", dpi=200); plt.close()
```

9 Chunk 8 — Bilateral (independent hypers)

Task: Choose ksize, σ_{space}, σ_{color} independently. Expect slow runtime (naive, illustrative). What kind of kernels are used here? What is the impact of the different kernels and their parameters?

```
# --- CHUNK 8: Bilateral (naive) ---
def bilateral_filter(x, ksize=7, sigma_space=2.0, sigma_color=0.10):
    assert x.shape[0] == 1, "Use batch=1 for this demo."
    pad = ksize // 2
    xp = F.pad(x, (pad,pad,pad,pad), mode="reflect")
    _, C, H, W = x.shape
    coords = torch.arange(ksize, device=x.device) - pad
    X, Y = torch.meshgrid(coords, coords, indexing="ij")
    spatial = torch.exp(-(X**2 + Y**2) / (2*sigma_space**2))
    out = torch.zeros_like(x)
    for i in range(H):
        for j in range(W):
            patch = xp[0, :, i:i+ksize, j:j+ksize]            # CxKxK
            center = x[0, :, i, j].view(C,1,1)               # Cx1x1
            # Range kernel over RGB; squared L2 across channels
            range_w = torch.exp(-((patch - center).pow(2).sum(dim=0)) / (2*sigma_color
    **2))
```

```
            w = spatial * range_w
            w = w / (w.sum() + 1e-12)
            out[0, :, i, j] = (patch * w).sum(dim=(1,2))
    return out

# PICK YOUR NOISY INPUT
# noisy = add_awgn(x_clean, sigma=0.10)
# noisy = add_poisson(x_clean, b=0.05)
noisy = add_speckle(x_clean, sigma=0.20)
# noisy = add_salt_pepper(x_clean, amount=0.05)

ksize = 7        # <-- EDIT THIS (e.g., 5, 7, 9)
s_space = 2.0    # <-- EDIT THIS
s_color = 0.10   # <-- EDIT THIS
den = bilateral_filter(noisy, ksize=ksize, sigma_space=s_space, sigma_color=s_color)
print(f"Bilateral(k={ksize}, ss={s_space}, sc={s_color}) PSNR:", psnr(x_clean, den), "
    LPIPS:", lpips_score(x_clean, den))

plt.figure(figsize=(9,3))
plt.subplot(1,3,1); plt.imshow(noisy[0].permute(1,2,0).cpu()); plt.axis("off"); plt.title
    ("Noisy")
plt.subplot(1,3,2); plt.imshow(den[0].permute(1,2,0).cpu());   plt.axis("off"); plt.title
    ("Bilateral")
plt.subplot(1,3,3); plt.imshow(x_clean[0].permute(1,2,0).cpu()); plt.axis("off"); plt.
    title("Clean")
plt.tight_layout(); plt.savefig(f"results/bilateral_k{ksize}_ss{s_space}_sc{s_color}.png"
    , dpi=200); plt.close()
```

Final prompts (you compile/compare)

- For each noise type (AWGN, Poisson, Speckle, Salt–Pepper), which single filter + hyperparameter setting gave the best *visual* result? Did PSNR/LPIPS agree?
- When increasing mean/median kernel size, where did benefits saturate?
- Bicubic: how does a trade smoothing vs. ringing? Give one concrete example.
- Bilateral: show a pair of settings where you (i) oversmooth and (ii) preserve edges with residual noise. Explain the difference between σ_{space} and σ_{color}.
- Report your chosen *noise–kernel matches* and justify briefly.

Caveats (be critical): Median excels for impulse noise but can destroy fine textures; mean blurs edges; bicubic here is a fixed 4×4 kernel (illustrative, not a full resampler); bilateral depends sensitively on σ_{color} and is slow in this naive form.