

BÀI THỰC HÀNH XLA

TS. Phan Thanh Toàn

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHÂN ẢNH

Bài tập 2 — Lấy mẫu & lượng tử hoá khi scan tài liệu

Bối cảnh: Bạn số hoá đề thi A4 để lưu trữ.

Yêu cầu chi tiết:

- Mô phỏng ảnh xám 8 bit → **giảm** còn 6–4–2 bit (lượng tử hoá).
- Tính **MAE**, **MSE**, **PSNR**, **SSIM** giữa ảnh gốc và ảnh giảm bit.
- Nêu mức bit tối thiểu để chữ vẫn đọc tốt (quan sát + chỉ số).

Gợi ý: Lượng tử hoá đều: $L = 2^k$, làm tròn về các mức.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import cv2, numpy as np
from skimage.metrics import structural_similarity as ssim

def quantize_gray(img_gray, k):
    L = 2**k
    img = img_gray.astype(np.float32)
    q = np.round(img / 255.0 * (L-1))
    rec = (q / (L-1) * 255.0).astype(np.uint8)
    return rec

def mse(a, b): return np.mean((a.astype(np.float32)-b.astype(np.float32))**2)
def psnr(a, b):
    m = mse(a,b)
    return 20*np.log10(255.0) - 10*np.log10(m+1e-12)

img = cv2.imread("scan_de_thi.png", cv2.IMREAD_GRAYSCALE)
for k in [6,4,2]:
    rec = quantize_gray(img, k)
    _mse = mse(img, rec)
    _psnr = psnr(img, rec)
    _ssim = ssim(img, rec, data_range=255)
    print(f"{k} bit -> MSE={_mse:.2f}, PSNR={_psnr:.2f} dB, SSIM={_ssim:.3f}")
    cv2.imwrite(f"scan_quant_{k}bit.png", rec)
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHÂN ẢNH

Bài tập 3 — Bit-plane slicing để phát hiện mờ/đốm in

Bối cảnh: Tờ hoá đơn/phiếu thu bị mờ & lẫn nhiều muối tiêu.

Yêu cầu chi tiết:

- Tách **8 mặt phẳng bit** của ảnh xám 8 bit.
- Quan sát: bit-plane thấp (0–3) chứa nhiều hơn; bit-plane cao chứa cấu trúc chữ.
- Tạo ảnh tái dựng chỉ từ các bit-plane 4–7, đánh giá **NCC/SSIM** với ảnh gốc.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import cv2, numpy as np

img = cv2.imread("bill.png", cv2.IMREAD_GRAYSCALE)
planes = [(img >> b) & 1 for b in range(8)]
for b, p in enumerate(planes):
    cv2.imwrite(f"bitplane_{b}.png", p*255)

# tái dựng từ bit 4..7
rec = np.zeros_like(img, dtype=np.uint8)
for b in range(4,8):
    rec |= ((planes[b].astype(np.uint8)) << b)
cv2.imwrite("bill_recon_4to7.png", rec)

def ncc(a, b):
    a = a.astype(np.float32); b = b.astype(np.float32)
    a = (a - a.mean())/(a.std()+1e-6)
    b = (b - b.mean())/(b.std()+1e-6)
    return np.mean(a*b)

print("NCC:", ncc(img, rec))
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

Bài tập 4 — 4/8/m-kết nối & quãng đường robot dọc lớp học

Bối cảnh: Robot di chuyển trên lưới sàn tránh chướng ngại.

Yêu cầu chi tiết:

- Cho lưới nhị phân (0: trống, 1: vật cản). Tính đường đi ngắn nhất từ S đến T với **kết nối-4**, **kết nối-8**, và **kết nối-m** (hạn chế nối chéo xuyên khe).
 - So sánh **độ dài quãng đường** theo **Manhattan (city-block)** và **Chessboard**; đối chiếu với **Euclidean**.
- Gợi ý:** BFS/Dijkstra trên lưới; tập lân cận thay đổi theo 4/8/m.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
from collections import deque
import numpy as np
def neighbors(p, conn='4'):
    x,y = p
    N4 = [(x+1,y),(x-1,y),(x,y+1),(x,y-1)]
    N8 = N4 + [(x+1,y+1),(x+1,y-1),(x-1,y+1),(x-1,y-1)]
    return N4 if conn=='4' else N8
def shortest_path(grid, s, t, conn='4'):
    H,W = grid.shape
    INF = 10**9
    dist = np.full((H,W), INF, int)
    prev = np.full((H,W,2), -1, int)
    dq = deque([s]); dist[s]=0
    while dq:
        x,y = dq.popleft()
        if (x,y)==t: break
        for nx,ny in neighbors((x,y), conn):
            if 0<=nx<H and 0<=ny<W and grid[nx,ny]==0 and dist[nx,ny]>dist[x,y]+1:
                dist[nx,ny]=dist[x,y]+1; prev[nx,ny]=[x,y]; dq.append((nx,ny))
    # truy vết
    path = []
    if dist[t]<INF:
        cur = t
        while (cur!=(-1,-1)) and (cur!=tuple(prev[cur][0:0])):
            path.append(cur)
            px,py = prev[cur]
            if px== -1: break
            cur = (px,py)
        path.reverse()
    return path, dist[t]

grid = np.zeros((10,15), int); grid[3:7,8]=1 # vật cản
s=(0,0); t=(9,14)
for conn in ['4','8']:
    path, L = shortest_path(grid, s, t, conn)
    print(conn, "steps:", L)
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

Bài tập 5 — Chuyển hệ màu & phát hiện vùng da mặt (HSV/YCrCb)

Bối cảnh: Lọc ảnh nhạy cảm trước khi đăng lên website trường.

Yêu cầu chi tiết:

- Đọc ảnh RGB, chuyển **HSV** và **YCrCb**.
- Dùng ngưỡng kinh nghiệm (ví dụ HSV: $H \in [0,25]$, $S \in [0.2,0.7]$, $V > 0.35$; YCrCb: $Cr \in [135,180]$, $Cb \in [85,135]$) để tách vùng da.
- So sánh mặt nạ kết quả giữa hai không gian.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import cv2, numpy as np
img = cv2.imread("portrait.jpg")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
# HSV thresholds (scale H:0-179, S,V:0-255 trong OpenCV)
lower_hsv = (0, 30, 90); upper_hsv = (25, 180, 255)
mask_hsv = cv2.inRange(hsv, lower_hsv, upper_hsv)
# YCrCb thresholds
lower_ycc = (0, 135, 85); upper_ycc = (255, 180, 135)
mask_ycc = cv2.inRange(ycrcb, lower_ycc, upper_ycc)

cv2.imwrite("mask_hsv.png", mask_hsv)
cv2.imwrite("mask_ycrcb.png", mask_ycc)
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHÂN ẢNH

Lab 1 — Chuẩn hoá & lượng tử hoá động (8→6→4→2 bit) + đánh giá

Mục tiêu: Thấy rõ ảnh hưởng của **lượng tử hoá** lên chất lượng.

Bước làm:

- Đọc ảnh xám; giảm bit theo các mức; lưu ảnh.
- Tính **MAE**, **MSE**, **PSNR**, **SSIM**, **NCC** giữa gốc và từng mức.
- Vẽ bảng/kết luận mức tối thiểu chấp nhận được.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
from skimage.metrics import structural_similarity as ssim
import numpy as np, cv2

def mae(a,b): return np.mean(np.abs(a.astype(np.float32)-b.astype(np.float32)))
def mse(a,b): return np.mean((a.astype(np.float32)-b.astype(np.float32))**2)
def psnr(a,b):
    m=mse(a,b); return 20*np.log10(255.0)-10*np.log10(m+1e-12)
def ncc(a,b):
    a=a.astype(np.float32); b=b.astype(np.float32)
    a=(a-a.mean())/(a.std()+1e-6); b=(b-b.mean())/(b.std()+1e-6)
    return np.mean(a*b)

img = cv2.imread("doc.png", cv2.IMREAD_GRAYSCALE)
for k in [7,6,5,4,3,2]:
    rec = ((np.round(img/255*(2**k-1)))/(2**k-1)*255).astype(np.uint8)
    print(k, "bit:", "MAE", mae(img,rec), "MSE", mse(img,rec),
          "PSNR", psnr(img,rec), "SSIM", ssim(img,rec,data_range=255), "NCC", ncc(img,rec))
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHÂN ẢNH

Lab 2 — Zooming & Shrinking: nearest/bilinear & pixel replication

Mục tiêu: So sánh **nội suy gần nhất**, **bilinear**, **pixel replication** khi phóng/thu ảnh.

Bước làm:

- Phóng ảnh $\times 1.5$ và $\times 4$ (nearest, bilinear, pixel replication).
- Thu ảnh $/2$ và $/4$ (nearest, bilinear).
- So sánh trực quan + dùng **PSNR/SSIM** giữa ảnh gốc và ảnh sau **thu rồi phóng ngược** về kích thước ban đầu.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import cv2, numpy as np
from skimage.metrics import structural_similarity as ssim

img = cv2.imread("campus.jpg", cv2.IMREAD_GRAYSCALE)
# nearest & bilinear
up_near = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_NEAREST)
up_bili = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)

# pixel replication (k=4)
rep = np.repeat(np.repeat(img, 4, axis=0), 4, axis=1)

# shrink & back
small = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)
back = cv2.resize(small, (img.shape[1], img.shape[0]),
interpolation=cv2.INTER_NEAREST)
print("PSNR back:", cv2.PSNR(img, back), "SSIM back:", ssim(img, back,
data_range=255))
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

Lab 3 — Đo góc, cung tròn, diện tích & ước lượng hình tròn qua 3 điểm

Mục tiêu: Lấy một ảnh **vạch kẻ sơn** trên sân trường (hoặc nắp cổng tròn), đo **góc, cung, diện tích** và nội suy **đường tròn qua 3 điểm** (gắn với Bài 2–5 trên slide).

Bước làm:

- Chọn 3 điểm trên biên (click chuột) → tính tâm $O(x,y)$ và bán kính r .
- Suy ra độ dài cung giữa 2 điểm theo r và góc θ .
- Phân đoạn đối tượng, dùng contourArea để đo **diện tích (px^2)**; nếu biết kích thước thực tế d (cm) của thước tham chiếu trong ảnh, hãy quy đổi sang **đơn vị thật**.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import numpy as np

def circle_from_3pts(p1, p2, p3):
    (x1,y1), (x2,y2), (x3,y3)=map(lambda p: (float(p[0]),float(p[1])), [p1,p2,p3])
    A = np.array([[2*(x2-x1), 2*(y2-y1)],
                  [2*(x3-x1), 2*(y3-y1)]], dtype=np.float64)
    b = np.array([x2**2+y2**2 - x1**2 - y1**2,
                  x3**2+y3**2 - x1**2 - y1**2], dtype=np.float64)
    O = np.linalg.solve(A, b)
    ox, oy = O
    r = np.hypot(ox-x1, oy-y1)
    return (ox,oy), r

# ví dụ
O, r = circle_from_3pts((100,200),(200,100),(300,200))
print("O, r:", O, r)
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

Lab 4 — Lân cận & gán nhãn thành phần liên thông (4 vs 8)

Mục tiêu: Thấy ảnh hưởng của **lựa chọn lân cận** lên số lượng thành phần liên thông (ví dụ ảnh **mạch in**/đường kẻ chéo mảnh).

Bước làm:

- Nhị phân hóa ảnh (Otsu).
- Gán nhãn connectedComponents với connectivity=4 và 8.
- So sánh số thành phần, trực quan hóa bằng màu giả.

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import cv2, numpy as np

img = cv2.imread("pcb.png", cv2.IMREAD_GRAYSCALE)
_, bw = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
for conn in [4,8]:
    num, labels = cv2.connectedComponents(bw, connectivity=conn)
    print(f"Connectivity {conn}: components = {num}")
    # gán màu giả để xem
    lab_norm = (labels/(labels.max()+1e-6)*255).astype(np.uint8)
    color = cv2.applyColorMap(lab_norm, cv2.COLORMAP_JET)
    cv2.imwrite(f"labels_conn{conn}.png", color)
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

Lab 5 — Đánh giá chất lượng ảnh sau suy giảm & phục hồi

Mục tiêu: Thực nghiệm các **chỉ số chất lượng** (MAE, MSE, PSNR, SSIM, NCC) khi ảnh bị nhiễu/nén.

Bước làm:

- Nhieu Gaussian + muối tiêu với mức tăng dần.
- Giảm bit-depth ($8 \rightarrow 6 \rightarrow 4$) và/hoặc dùng cv2.imencode('.jpg', img, [IMWRITE_JPEG_QUALITY,q]).
- Tính & ghi bảng các chỉ số, rút ra chỉ số phù hợp với cảm nhận thị giác hơn (**SSIM** thường tương quan tốt hơn MSE/PSNR).

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHẬN ẢNH

```
import cv2, numpy as np
from skimage.metrics import structural_similarity as ssim

def add_gaussian_noise(img, sigma=10):
    noise = np.random.normal(0, sigma, img.shape).astype(np.float32)
    out = np.clip(img.astype(np.float32)+noise, 0, 255).astype(np.uint8)
    return out

img = cv2.imread("scene.jpg", cv2.IMREAD_GRAYSCALE)
noisy = add_gaussian_noise(img, sigma=15)
# JPEG nén
_, enc = cv2.imencode('.jpg', img, [int(cv2.IMWRITE_JPEG_QUALITY), 40])
jpeg = cv2.imdecode(enc, cv2.IMREAD_GRAYSCALE)

def mse(a,b): return np.mean((a.astype(np.float32)-b.astype(np.float32))**2)
def psnr(a,b): return 20*np.log10(255.0)-10*np.log10(mse(a,b)+1e-12)
def ncc(a,b):
    a=a.astype(np.float32); b=b.astype(np.float32)
    a=(a-a.mean())/(a.std()+1e-6); b=(b-b.mean())/(b.std()+1e-6)
    return np.mean(a*b)

for name, im in [("noisy", noisy), ("jpeg40", jpeg)]:
    print(name, "MSE", mse(img,im), "PSNR", psnr(img,im),
          "SSIM", ssim(img,im,data_range=255), "NCC", ncc(img,im))
```

THỰC HÀNH VỀ BIỂU DIỄN VÀ THU NHÂN ẢNH

Bài tập 1 — Tính dung lượng & băng thông cho hệ thống camera

Bối cảnh: Bạn thiết kế hệ thống giám sát hành lang trường học.

Yêu cầu chi tiết:

- Cho độ phân giải $M \times N$, số kênh (xám=1, RGB=3), độ sâu bit k (ví dụ 8/10/12), số khung hình/giây (fps), thời lượng lưu (ngày).
- Tính **dung lượng** (GB) và **băng thông** (Mbps) giả định lưu “thô” (chưa nén).
- So sánh kịch bản: (a) 1080p–8 bit–15 fps; (b) 720p–10 bit–25 fps; (c) 4K–8 bit–5 fps.
- Thảo luận trade-off giữa **độ phân giải không gian** (sampling) và **độ phân giải mức xám** (quantization).
- **Gợi ý:** Kích thước ảnh $\sim M \times N \times \text{channels} \times k$ (bit), quy đổi B/KB/MB/GB.