In [6]:
```ipynb
# 05_second_model_edsr.ipynb
from torch.utils.data import Dataset, DataLoader

import os
from pathlib import Path

import torch
from torch import nn
from torch.utils.data import DataLoader, Subset
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt

print("PyTorch:", torch.__version__)

# 自动识别项目根目录（你现在在 notebooks/ 里）
PROJECT_ROOT = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd
DATA_DIR = PROJECT_ROOT / "data" / "celeba"
IMG_DIR = DATA_DIR / "img_align_celeba"

print("PROJECT_ROOT:", PROJECT_ROOT)
print("IMG_DIR:", IMG_DIR, "exists:", IMG_DIR.exists())

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
```

```
PyTorch: 2.6.0+cu118
PROJECT_ROOT: D:\SuperResolution_Project-main
IMG_DIR: D:\SuperResolution_Project-main\data\celeba\img_align_celeba exists: Tru
e
Device: cuda
```

In [7]:
```ipynb
# 数据集定义：HR 128x128, LR 64x64

HR_SIZE = 128
SCALE   = 2
LR_SIZE = HR_SIZE // SCALE

hr_tf = transforms.Compose([
    transforms.CenterCrop(178),
    transforms.Resize((HR_SIZE, HR_SIZE), interpolation=Image.BICUBIC),
    transforms.ToTensor(),
])

to_pil = transforms.ToPILImage()

lr_from_hr = transforms.Compose([
    transforms.Resize((LR_SIZE, LR_SIZE), interpolation=Image.BICUBIC),
    transforms.ToTensor(),
])

class CelebASR(Dataset):
    def __init__(self, img_dir: Path):
        self.paths = sorted(img_dir.glob("*.jpg"))
        if len(self.paths) == 0:
            raise RuntimeError(f"No images found in {img_dir}")
    def __len__(self):
        return len(self.paths)
    def __getitem__(self, i):
```

```
        pil = Image.open(self.paths[i]).convert("RGB")
        hr  = hr_tf(pil)           # [3, 128, 128]
        lr  = lr_from_hr(to_pil(hr))  # [3, 64, 64]
        return lr, hr

sr_ds = CelebASR(IMG_DIR)
print("Total images:", len(sr_ds))
```

```
Total images: 202599
```

In [8]:
```
from torch.utils.data import Subset, DataLoader
import torch

N = len(sr_ds)
g = torch.Generator().manual_seed(42)
perm = torch.randperm(N, generator=g)

test_ratio, val_ratio = 0.05, 0.05
test_size = int(N * test_ratio)
val_size  = int(N * val_ratio)
train_size = N - val_size - test_size

train_idx = perm[:train_size]
val_idx   = perm[train_size:train_size+val_size]
test_idx  = perm[train_size+val_size:]

train_ds = Subset(sr_ds, train_idx.tolist())
val_ds   = Subset(sr_ds, val_idx.tolist())
test_ds  = Subset(sr_ds, test_idx.tolist())

print(f"Split sizes ▶ train: {len(train_ds)}  val: {len(val_ds)}  test: {len(te

batch_size = 16  # Mac 上可以先用 8 或 16，慢一点但能跑
num_workers = 0  # 如果报错就改成 0

train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True,
                          num_workers=num_workers, pin_memory=False)
val_loader   = DataLoader(val_ds, batch_size=batch_size, shuffle=False,
                          num_workers=num_workers, pin_memory=False)
test_loader  = DataLoader(test_ds, batch_size=batch_size, shuffle=False,
                           num_workers=num_workers, pin_memory=False)

# 简单检查一个 batch
lr_batch, hr_batch = next(iter(train_loader))
print("LR batch:", lr_batch.shape, "HR batch:", hr_batch.shape)
```

```
Split sizes ▶ train: 182341  val: 10129  test: 10129
LR batch: torch.Size([16, 3, 64, 64]) HR batch: torch.Size([16, 3, 128, 128])
```

In [9]:
```
# EDSR model for 2× super-resolution

import torch
from torch import nn
import torch.nn.functional as F

class ResidualBlockNoBN(nn.Module):
    def __init__(self, n_feats=64, res_scale=0.1):
        super().__init__()
        self.res_scale = res_scale
        self.body = nn.Sequential(
            nn.Conv2d(n_feats, n_feats, kernel_size=3, padding=1),
```

```
            nn.ReLU(inplace=True),
            nn.Conv2d(n_feats, n_feats, kernel_size=3, padding=1),
        )

    def forward(self, x):
        res = self.body(x)
        return x + res * self.res_scale


class EDSR_x2(nn.Module):
    def __init__(self, in_channels=3, out_channels=3,
                 n_feats=64, n_resblocks=8, scale=2):
        super().__init__()

        # head
        self.head = nn.Conv2d(in_channels, n_feats, kernel_size=3, padding=1)

        # body: a stack of residual blocks
        blocks = [ResidualBlockNoBN(n_feats=n_feats, res_scale=0.1)
                  for _ in range(n_resblocks)]
        blocks.append(nn.Conv2d(n_feats, n_feats, kernel_size=3, padding=1))
        self.body = nn.Sequential(*blocks)

        # upsampling x2 (PixelShuffle)
        modules_upsample = [
            nn.Conv2d(n_feats, n_feats * (scale ** 2), kernel_size=3, padding=1),
            nn.PixelShuffle(scale),
            nn.Conv2d(n_feats, out_channels, kernel_size=3, padding=1),
        ]
        self.tail = nn.Sequential(*modules_upsample)

    def forward(self, x):
        x = self.head(x)
        res = self.body(x)
        x = x + res         # long skip connection
        x = self.tail(x)
        return x

edsr = EDSR_x2().to(device)
print(edsr)
```

```
EDSR_x2(
  (head): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (body): Sequential(
    (0): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (1): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (2): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (3): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (4): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (5): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (6): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (7): ResidualBlockNoBN(
      (body): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (8): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
  )
  (tail): Sequential(
    (0): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): PixelShuffle(upscale_factor=2)
    (2): Conv2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
```

In [10]:

```python
from torch.optim import Adam

# 训练配置
num_epochs   = 15          # 先用 2 epoch 测试流程；觉得 OK 再改大一点
lr           = 1e-4
log_interval = 200        # 每多少个 batch 打印一次

criterion = nn.L1Loss()   # EDSR 经典用 L1 损失
optimizer = Adam(edsr.parameters(), lr=lr)

def train_one_epoch(model, loader, optimizer, criterion, device):
    model.train()
    running_loss = 0.0

    for batch_idx, (lr_img, hr_img) in enumerate(loader):
        lr_img = lr_img.to(device)
        hr_img = hr_img.to(device)

        sr = model(lr_img)
        loss = criterion(sr, hr_img)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        if (batch_idx + 1) % log_interval == 0:
            print(f"  [batch {batch_idx+1:5d}] loss = {running_loss / log_interv
            running_loss = 0.0


@torch.no_grad()
def eval_psnr(model, loader, device, max_batches=50):
    model.eval()
    mse_sum = 0.0
    n = 0

    for b_idx, (lr_img, hr_img) in enumerate(loader):
        lr_img = lr_img.to(device)
        hr_img = hr_img.to(device)

        sr = model(lr_img).clamp(0.0, 1.0)
        mse = F.mse_loss(sr, hr_img, reduction="mean").item()
        mse_sum += mse
        n += 1

        if max_batches is not None and (b_idx + 1) >= max_batches:
            break

    avg_mse = mse_sum / max(1, n)
    psnr = -10 * torch.log10(torch.tensor(avg_mse)).item()
```

```
    return psnr


# ========= 实际训练 =========
best_psnr = -1

for epoch in range(1, num_epochs + 1):
    print(f"\n===== Epoch {epoch}/{num_epochs} =====")
    train_one_epoch(edsr, train_loader, optimizer, criterion, device)
    psnr_val = eval_psnr(edsr, val_loader, device, max_batches=50)
    print(f"  → Val PSNR (first 50 batches): {psnr_val:.2f} dB")

    if psnr_val > best_psnr:
        best_psnr = psnr_val
        torch.save(edsr.state_dict(), PROJECT_ROOT / "experiments" / "edsr_x2_be
        print("  ✓ Best model updated.")
```

```
===== Epoch 1/15 =====
  [batch   200] loss = 0.0576
  [batch   400] loss = 0.0227
  [batch   600] loss = 0.0183
  [batch   800] loss = 0.0167
  [batch  1000] loss = 0.0163
  [batch  1200] loss = 0.0163
  [batch  1400] loss = 0.0156
  [batch  1600] loss = 0.0154
  [batch  1800] loss = 0.0149
  [batch  2000] loss = 0.0151
  [batch  2200] loss = 0.0149
  [batch  2400] loss = 0.0146
  [batch  2600] loss = 0.0144
  [batch  2800] loss = 0.0144
  [batch  3000] loss = 0.0142
  [batch  3200] loss = 0.0140
  [batch  3400] loss = 0.0138
  [batch  3600] loss = 0.0138
  [batch  3800] loss = 0.0139
  [batch  4000] loss = 0.0135
  [batch  4200] loss = 0.0135
  [batch  4400] loss = 0.0137
  [batch  4600] loss = 0.0131
  [batch  4800] loss = 0.0134
  [batch  5000] loss = 0.0131
  [batch  5200] loss = 0.0128
  [batch  5400] loss = 0.0129
  [batch  5600] loss = 0.0129
  [batch  5800] loss = 0.0127
  [batch  6000] loss = 0.0127
  [batch  6200] loss = 0.0126
  [batch  6400] loss = 0.0123
  [batch  6600] loss = 0.0125
  [batch  6800] loss = 0.0124
  [batch  7000] loss = 0.0123
  [batch  7200] loss = 0.0123
  [batch  7400] loss = 0.0123
  [batch  7600] loss = 0.0122
  [batch  7800] loss = 0.0121
  [batch  8000] loss = 0.0121
  [batch  8200] loss = 0.0120
  [batch  8400] loss = 0.0121
  [batch  8600] loss = 0.0121
  [batch  8800] loss = 0.0118
  [batch  9000] loss = 0.0119
  [batch  9200] loss = 0.0119
  [batch  9400] loss = 0.0117
  [batch  9600] loss = 0.0119
  [batch  9800] loss = 0.0120
  [batch 10000] loss = 0.0117
  [batch 10200] loss = 0.0116
  [batch 10400] loss = 0.0117
  [batch 10600] loss = 0.0117
  [batch 10800] loss = 0.0116
  [batch 11000] loss = 0.0116
  [batch 11200] loss = 0.0117
  → Val PSNR (first 50 batches): 33.93 dB
  ✓ Best model updated.
```

```
===== Epoch 2/15 =====
  [batch   200] loss = 0.0114
  [batch   400] loss = 0.0114
  [batch   600] loss = 0.0116
  [batch   800] loss = 0.0116
  [batch  1000] loss = 0.0115
  [batch  1200] loss = 0.0114
  [batch  1400] loss = 0.0115
  [batch  1600] loss = 0.0117
  [batch  1800] loss = 0.0114
  [batch  2000] loss = 0.0113
  [batch  2200] loss = 0.0115
  [batch  2400] loss = 0.0114
  [batch  2600] loss = 0.0112
  [batch  2800] loss = 0.0113
  [batch  3000] loss = 0.0113
  [batch  3200] loss = 0.0114
  [batch  3400] loss = 0.0114
  [batch  3600] loss = 0.0114
  [batch  3800] loss = 0.0112
  [batch  4000] loss = 0.0113
  [batch  4200] loss = 0.0113
  [batch  4400] loss = 0.0113
  [batch  4600] loss = 0.0112
  [batch  4800] loss = 0.0112
  [batch  5000] loss = 0.0113
  [batch  5200] loss = 0.0112
  [batch  5400] loss = 0.0111
  [batch  5600] loss = 0.0112
  [batch  5800] loss = 0.0110
  [batch  6000] loss = 0.0112
  [batch  6200] loss = 0.0113
  [batch  6400] loss = 0.0112
  [batch  6600] loss = 0.0112
  [batch  6800] loss = 0.0112
  [batch  7000] loss = 0.0111
  [batch  7200] loss = 0.0111
  [batch  7400] loss = 0.0111
  [batch  7600] loss = 0.0113
  [batch  7800] loss = 0.0111
  [batch  8000] loss = 0.0109
  [batch  8200] loss = 0.0111
  [batch  8400] loss = 0.0112
  [batch  8600] loss = 0.0110
  [batch  8800] loss = 0.0109
  [batch  9000] loss = 0.0110
  [batch  9200] loss = 0.0111
  [batch  9400] loss = 0.0109
  [batch  9600] loss = 0.0110
  [batch  9800] loss = 0.0111
  [batch 10000] loss = 0.0109
  [batch 10200] loss = 0.0110
  [batch 10400] loss = 0.0108
  [batch 10600] loss = 0.0110
  [batch 10800] loss = 0.0111
  [batch 11000] loss = 0.0109
  [batch 11200] loss = 0.0111
  → Val PSNR (first 50 batches): 34.39 dB
  ✓ Best model updated.
```

```
===== Epoch 3/15 =====
  [batch   200] loss = 0.0109
  [batch   400] loss = 0.0109
  [batch   600] loss = 0.0110
  [batch   800] loss = 0.0108
  [batch  1000] loss = 0.0109
  [batch  1200] loss = 0.0108
  [batch  1400] loss = 0.0107
  [batch  1600] loss = 0.0111
  [batch  1800] loss = 0.0108
  [batch  2000] loss = 0.0108
  [batch  2200] loss = 0.0109
  [batch  2400] loss = 0.0110
  [batch  2600] loss = 0.0108
  [batch  2800] loss = 0.0109
  [batch  3000] loss = 0.0107
  [batch  3200] loss = 0.0108
  [batch  3400] loss = 0.0109
  [batch  3600] loss = 0.0108
  [batch  3800] loss = 0.0109
  [batch  4000] loss = 0.0109
  [batch  4200] loss = 0.0108
  [batch  4400] loss = 0.0108
  [batch  4600] loss = 0.0108
  [batch  4800] loss = 0.0108
  [batch  5000] loss = 0.0108
  [batch  5200] loss = 0.0107
  [batch  5400] loss = 0.0107
  [batch  5600] loss = 0.0107
  [batch  5800] loss = 0.0108
  [batch  6000] loss = 0.0107
  [batch  6200] loss = 0.0107
  [batch  6400] loss = 0.0109
  [batch  6600] loss = 0.0108
  [batch  6800] loss = 0.0106
  [batch  7000] loss = 0.0108
  [batch  7200] loss = 0.0108
  [batch  7400] loss = 0.0108
  [batch  7600] loss = 0.0107
  [batch  7800] loss = 0.0107
  [batch  8000] loss = 0.0109
  [batch  8200] loss = 0.0107
  [batch  8400] loss = 0.0106
  [batch  8600] loss = 0.0107
  [batch  8800] loss = 0.0107
  [batch  9000] loss = 0.0107
  [batch  9200] loss = 0.0107
  [batch  9400] loss = 0.0107
  [batch  9600] loss = 0.0106
  [batch  9800] loss = 0.0107
  [batch 10000] loss = 0.0109
  [batch 10200] loss = 0.0106
  [batch 10400] loss = 0.0107
  [batch 10600] loss = 0.0107
  [batch 10800] loss = 0.0107
  [batch 11000] loss = 0.0106
  [batch 11200] loss = 0.0106
  → Val PSNR (first 50 batches): 34.62 dB
  ✓ Best model updated.
```

```
===== Epoch 4/15 =====
  [batch   200] loss = 0.0106
  [batch   400] loss = 0.0106
  [batch   600] loss = 0.0107
  [batch   800] loss = 0.0105
  [batch  1000] loss = 0.0105
  [batch  1200] loss = 0.0107
  [batch  1400] loss = 0.0107
  [batch  1600] loss = 0.0106
  [batch  1800] loss = 0.0106
  [batch  2000] loss = 0.0106
  [batch  2200] loss = 0.0104
  [batch  2400] loss = 0.0106
  [batch  2600] loss = 0.0105
  [batch  2800] loss = 0.0106
  [batch  3000] loss = 0.0105
  [batch  3200] loss = 0.0106
  [batch  3400] loss = 0.0106
  [batch  3600] loss = 0.0106
  [batch  3800] loss = 0.0104
  [batch  4000] loss = 0.0106
  [batch  4200] loss = 0.0105
  [batch  4400] loss = 0.0105
  [batch  4600] loss = 0.0106
  [batch  4800] loss = 0.0106
  [batch  5000] loss = 0.0106
  [batch  5200] loss = 0.0105
  [batch  5400] loss = 0.0106
  [batch  5600] loss = 0.0107
  [batch  5800] loss = 0.0107
  [batch  6000] loss = 0.0103
  [batch  6200] loss = 0.0105
  [batch  6400] loss = 0.0104
  [batch  6600] loss = 0.0105
  [batch  6800] loss = 0.0105
  [batch  7000] loss = 0.0105
  [batch  7200] loss = 0.0107
  [batch  7400] loss = 0.0105
  [batch  7600] loss = 0.0104
  [batch  7800] loss = 0.0104
  [batch  8000] loss = 0.0105
  [batch  8200] loss = 0.0106
  [batch  8400] loss = 0.0105
  [batch  8600] loss = 0.0105
  [batch  8800] loss = 0.0106
  [batch  9000] loss = 0.0105
  [batch  9200] loss = 0.0105
  [batch  9400] loss = 0.0105
  [batch  9600] loss = 0.0104
  [batch  9800] loss = 0.0104
  [batch 10000] loss = 0.0105
  [batch 10200] loss = 0.0105
  [batch 10400] loss = 0.0105
  [batch 10600] loss = 0.0106
  [batch 10800] loss = 0.0104
  [batch 11000] loss = 0.0103
  [batch 11200] loss = 0.0106
  → Val PSNR (first 50 batches): 34.77 dB
  ✓ Best model updated.
```

```
===== Epoch 5/15 =====
  [batch   200] loss = 0.0104
  [batch   400] loss = 0.0105
  [batch   600] loss = 0.0106
  [batch   800] loss = 0.0106
  [batch  1000] loss = 0.0104
  [batch  1200] loss = 0.0104
  [batch  1400] loss = 0.0104
  [batch  1600] loss = 0.0104
  [batch  1800] loss = 0.0103
  [batch  2000] loss = 0.0104
  [batch  2200] loss = 0.0106
  [batch  2400] loss = 0.0104
  [batch  2600] loss = 0.0103
  [batch  2800] loss = 0.0103
  [batch  3000] loss = 0.0105
  [batch  3200] loss = 0.0103
  [batch  3400] loss = 0.0104
  [batch  3600] loss = 0.0104
  [batch  3800] loss = 0.0104
  [batch  4000] loss = 0.0104
  [batch  4200] loss = 0.0104
  [batch  4400] loss = 0.0105
  [batch  4600] loss = 0.0104
  [batch  4800] loss = 0.0103
  [batch  5000] loss = 0.0105
  [batch  5200] loss = 0.0104
  [batch  5400] loss = 0.0104
  [batch  5600] loss = 0.0104
  [batch  5800] loss = 0.0103
  [batch  6000] loss = 0.0103
  [batch  6200] loss = 0.0104
  [batch  6400] loss = 0.0104
  [batch  6600] loss = 0.0103
  [batch  6800] loss = 0.0103
  [batch  7000] loss = 0.0103
  [batch  7200] loss = 0.0103
  [batch  7400] loss = 0.0103
  [batch  7600] loss = 0.0104
  [batch  7800] loss = 0.0104
  [batch  8000] loss = 0.0104
  [batch  8200] loss = 0.0103
  [batch  8400] loss = 0.0105
  [batch  8600] loss = 0.0105
  [batch  8800] loss = 0.0103
  [batch  9000] loss = 0.0104
  [batch  9200] loss = 0.0104
  [batch  9400] loss = 0.0103
  [batch  9600] loss = 0.0104
  [batch  9800] loss = 0.0102
  [batch 10000] loss = 0.0102
  [batch 10200] loss = 0.0102
  [batch 10400] loss = 0.0104
  [batch 10600] loss = 0.0103
  [batch 10800] loss = 0.0104
  [batch 11000] loss = 0.0104
  [batch 11200] loss = 0.0102
  → Val PSNR (first 50 batches): 34.87 dB
  ✓ Best model updated.
```

```
===== Epoch 6/15 =====
  [batch   200] loss = 0.0103
  [batch   400] loss = 0.0103
  [batch   600] loss = 0.0102
  [batch   800] loss = 0.0104
  [batch  1000] loss = 0.0104
  [batch  1200] loss = 0.0104
  [batch  1400] loss = 0.0102
  [batch  1600] loss = 0.0103
  [batch  1800] loss = 0.0103
  [batch  2000] loss = 0.0103
  [batch  2200] loss = 0.0103
  [batch  2400] loss = 0.0104
  [batch  2600] loss = 0.0103
  [batch  2800] loss = 0.0102
  [batch  3000] loss = 0.0103
  [batch  3200] loss = 0.0104
  [batch  3400] loss = 0.0103
  [batch  3600] loss = 0.0101
  [batch  3800] loss = 0.0104
  [batch  4000] loss = 0.0102
  [batch  4200] loss = 0.0102
  [batch  4400] loss = 0.0103
  [batch  4600] loss = 0.0102
  [batch  4800] loss = 0.0103
  [batch  5000] loss = 0.0104
  [batch  5200] loss = 0.0103
  [batch  5400] loss = 0.0102
  [batch  5600] loss = 0.0102
  [batch  5800] loss = 0.0102
  [batch  6000] loss = 0.0103
  [batch  6200] loss = 0.0102
  [batch  6400] loss = 0.0101
  [batch  6600] loss = 0.0102
  [batch  6800] loss = 0.0102
  [batch  7000] loss = 0.0104
  [batch  7200] loss = 0.0102
  [batch  7400] loss = 0.0103
  [batch  7600] loss = 0.0102
  [batch  7800] loss = 0.0103
  [batch  8000] loss = 0.0103
  [batch  8200] loss = 0.0102
  [batch  8400] loss = 0.0102
  [batch  8600] loss = 0.0103
  [batch  8800] loss = 0.0102
  [batch  9000] loss = 0.0103
  [batch  9200] loss = 0.0101
  [batch  9400] loss = 0.0101
  [batch  9600] loss = 0.0102
  [batch  9800] loss = 0.0103
  [batch 10000] loss = 0.0101
  [batch 10200] loss = 0.0103
  [batch 10400] loss = 0.0102
  [batch 10600] loss = 0.0102
  [batch 10800] loss = 0.0101
  [batch 11000] loss = 0.0102
  [batch 11200] loss = 0.0103
  → Val PSNR (first 50 batches): 34.93 dB
  ✓ Best model updated.
```

```
===== Epoch 7/15 =====
  [batch   200] loss = 0.0102
  [batch   400] loss = 0.0101
  [batch   600] loss = 0.0102
  [batch   800] loss = 0.0101
  [batch  1000] loss = 0.0102
  [batch  1200] loss = 0.0102
  [batch  1400] loss = 0.0102
  [batch  1600] loss = 0.0102
  [batch  1800] loss = 0.0103
  [batch  2000] loss = 0.0101
  [batch  2200] loss = 0.0102
  [batch  2400] loss = 0.0101
  [batch  2600] loss = 0.0101
  [batch  2800] loss = 0.0102
  [batch  3000] loss = 0.0102
  [batch  3200] loss = 0.0101
  [batch  3400] loss = 0.0102
  [batch  3600] loss = 0.0102
  [batch  3800] loss = 0.0103
  [batch  4000] loss = 0.0101
  [batch  4200] loss = 0.0104
  [batch  4400] loss = 0.0101
  [batch  4600] loss = 0.0102
  [batch  4800] loss = 0.0102
  [batch  5000] loss = 0.0101
  [batch  5200] loss = 0.0101
  [batch  5400] loss = 0.0101
  [batch  5600] loss = 0.0101
  [batch  5800] loss = 0.0102
  [batch  6000] loss = 0.0101
  [batch  6200] loss = 0.0102
  [batch  6400] loss = 0.0101
  [batch  6600] loss = 0.0102
  [batch  6800] loss = 0.0102
  [batch  7000] loss = 0.0101
  [batch  7200] loss = 0.0103
  [batch  7400] loss = 0.0102
  [batch  7600] loss = 0.0101
  [batch  7800] loss = 0.0103
  [batch  8000] loss = 0.0102
  [batch  8200] loss = 0.0100
  [batch  8400] loss = 0.0101
  [batch  8600] loss = 0.0102
  [batch  8800] loss = 0.0103
  [batch  9000] loss = 0.0102
  [batch  9200] loss = 0.0101
  [batch  9400] loss = 0.0101
  [batch  9600] loss = 0.0101
  [batch  9800] loss = 0.0102
  [batch 10000] loss = 0.0102
  [batch 10200] loss = 0.0102
  [batch 10400] loss = 0.0102
  [batch 10600] loss = 0.0101
  [batch 10800] loss = 0.0101
  [batch 11000] loss = 0.0100
  [batch 11200] loss = 0.0101
  → Val PSNR (first 50 batches): 35.01 dB
  ✓ Best model updated.
```

```
===== Epoch 8/15 =====
  [batch   200] loss = 0.0102
  [batch   400] loss = 0.0101
  [batch   600] loss = 0.0101
  [batch   800] loss = 0.0102
  [batch  1000] loss = 0.0101
  [batch  1200] loss = 0.0102
  [batch  1400] loss = 0.0101
  [batch  1600] loss = 0.0101
  [batch  1800] loss = 0.0101
  [batch  2000] loss = 0.0100
  [batch  2200] loss = 0.0102
  [batch  2400] loss = 0.0102
  [batch  2600] loss = 0.0101
  [batch  2800] loss = 0.0102
  [batch  3000] loss = 0.0101
  [batch  3200] loss = 0.0101
  [batch  3400] loss = 0.0103
  [batch  3600] loss = 0.0101
  [batch  3800] loss = 0.0100
  [batch  4000] loss = 0.0101
  [batch  4200] loss = 0.0102
  [batch  4400] loss = 0.0100
  [batch  4600] loss = 0.0101
  [batch  4800] loss = 0.0101
  [batch  5000] loss = 0.0100
  [batch  5200] loss = 0.0100
  [batch  5400] loss = 0.0099
  [batch  5600] loss = 0.0101
  [batch  5800] loss = 0.0101
  [batch  6000] loss = 0.0101
  [batch  6200] loss = 0.0102
  [batch  6400] loss = 0.0101
  [batch  6600] loss = 0.0100
  [batch  6800] loss = 0.0100
  [batch  7000] loss = 0.0102
  [batch  7200] loss = 0.0102
  [batch  7400] loss = 0.0102
  [batch  7600] loss = 0.0101
  [batch  7800] loss = 0.0101
  [batch  8000] loss = 0.0102
  [batch  8200] loss = 0.0102
  [batch  8400] loss = 0.0101
  [batch  8600] loss = 0.0100
  [batch  8800] loss = 0.0101
  [batch  9000] loss = 0.0101
  [batch  9200] loss = 0.0101
  [batch  9400] loss = 0.0101
  [batch  9600] loss = 0.0101
  [batch  9800] loss = 0.0101
  [batch 10000] loss = 0.0101
  [batch 10200] loss = 0.0100
  [batch 10400] loss = 0.0102
  [batch 10600] loss = 0.0101
  [batch 10800] loss = 0.0100
  [batch 11000] loss = 0.0101
  [batch 11200] loss = 0.0100
  → Val PSNR (first 50 batches): 35.06 dB
  ✓ Best model updated.
```

```
===== Epoch 9/15 =====
  [batch   200] loss = 0.0101
  [batch   400] loss = 0.0101
  [batch   600] loss = 0.0101
  [batch   800] loss = 0.0101
  [batch  1000] loss = 0.0100
  [batch  1200] loss = 0.0100
  [batch  1400] loss = 0.0100
  [batch  1600] loss = 0.0100
  [batch  1800] loss = 0.0100
  [batch  2000] loss = 0.0101
  [batch  2200] loss = 0.0100
  [batch  2400] loss = 0.0101
  [batch  2600] loss = 0.0102
  [batch  2800] loss = 0.0100
  [batch  3000] loss = 0.0101
  [batch  3200] loss = 0.0101
  [batch  3400] loss = 0.0099
  [batch  3600] loss = 0.0101
  [batch  3800] loss = 0.0100
  [batch  4000] loss = 0.0100
  [batch  4200] loss = 0.0100
  [batch  4400] loss = 0.0100
  [batch  4600] loss = 0.0101
  [batch  4800] loss = 0.0101
  [batch  5000] loss = 0.0100
  [batch  5200] loss = 0.0101
  [batch  5400] loss = 0.0101
  [batch  5600] loss = 0.0101
  [batch  5800] loss = 0.0100
  [batch  6000] loss = 0.0100
  [batch  6200] loss = 0.0101
  [batch  6400] loss = 0.0101
  [batch  6600] loss = 0.0100
  [batch  6800] loss = 0.0101
  [batch  7000] loss = 0.0100
  [batch  7200] loss = 0.0100
  [batch  7400] loss = 0.0099
  [batch  7600] loss = 0.0101
  [batch  7800] loss = 0.0101
  [batch  8000] loss = 0.0100
  [batch  8200] loss = 0.0101
  [batch  8400] loss = 0.0100
  [batch  8600] loss = 0.0100
  [batch  8800] loss = 0.0102
  [batch  9000] loss = 0.0100
  [batch  9200] loss = 0.0099
  [batch  9400] loss = 0.0100
  [batch  9600] loss = 0.0099
  [batch  9800] loss = 0.0099
  [batch 10000] loss = 0.0101
  [batch 10200] loss = 0.0100
  [batch 10400] loss = 0.0100
  [batch 10600] loss = 0.0099
  [batch 10800] loss = 0.0102
  [batch 11000] loss = 0.0100
  [batch 11200] loss = 0.0101
  → Val PSNR (first 50 batches): 35.08 dB
  ✓ Best model updated.
```

```
===== Epoch 10/15 =====
  [batch   200] loss = 0.0100
  [batch   400] loss = 0.0100
  [batch   600] loss = 0.0100
  [batch   800] loss = 0.0099
  [batch  1000] loss = 0.0101
  [batch  1200] loss = 0.0100
  [batch  1400] loss = 0.0101
  [batch  1600] loss = 0.0101
  [batch  1800] loss = 0.0101
  [batch  2000] loss = 0.0099
  [batch  2200] loss = 0.0099
  [batch  2400] loss = 0.0100
  [batch  2600] loss = 0.0101
  [batch  2800] loss = 0.0100
  [batch  3000] loss = 0.0099
  [batch  3200] loss = 0.0100
  [batch  3400] loss = 0.0100
  [batch  3600] loss = 0.0101
  [batch  3800] loss = 0.0099
  [batch  4000] loss = 0.0101
  [batch  4200] loss = 0.0100
  [batch  4400] loss = 0.0100
  [batch  4600] loss = 0.0102
  [batch  4800] loss = 0.0101
  [batch  5000] loss = 0.0100
  [batch  5200] loss = 0.0099
  [batch  5400] loss = 0.0100
  [batch  5600] loss = 0.0100
  [batch  5800] loss = 0.0100
  [batch  6000] loss = 0.0100
  [batch  6200] loss = 0.0100
  [batch  6400] loss = 0.0100
  [batch  6600] loss = 0.0099
  [batch  6800] loss = 0.0100
  [batch  7000] loss = 0.0101
  [batch  7200] loss = 0.0099
  [batch  7400] loss = 0.0100
  [batch  7600] loss = 0.0100
  [batch  7800] loss = 0.0100
  [batch  8000] loss = 0.0099
  [batch  8200] loss = 0.0099
  [batch  8400] loss = 0.0099
  [batch  8600] loss = 0.0099
  [batch  8800] loss = 0.0100
  [batch  9000] loss = 0.0100
  [batch  9200] loss = 0.0099
  [batch  9400] loss = 0.0100
  [batch  9600] loss = 0.0100
  [batch  9800] loss = 0.0099
  [batch 10000] loss = 0.0100
  [batch 10200] loss = 0.0100
  [batch 10400] loss = 0.0100
```

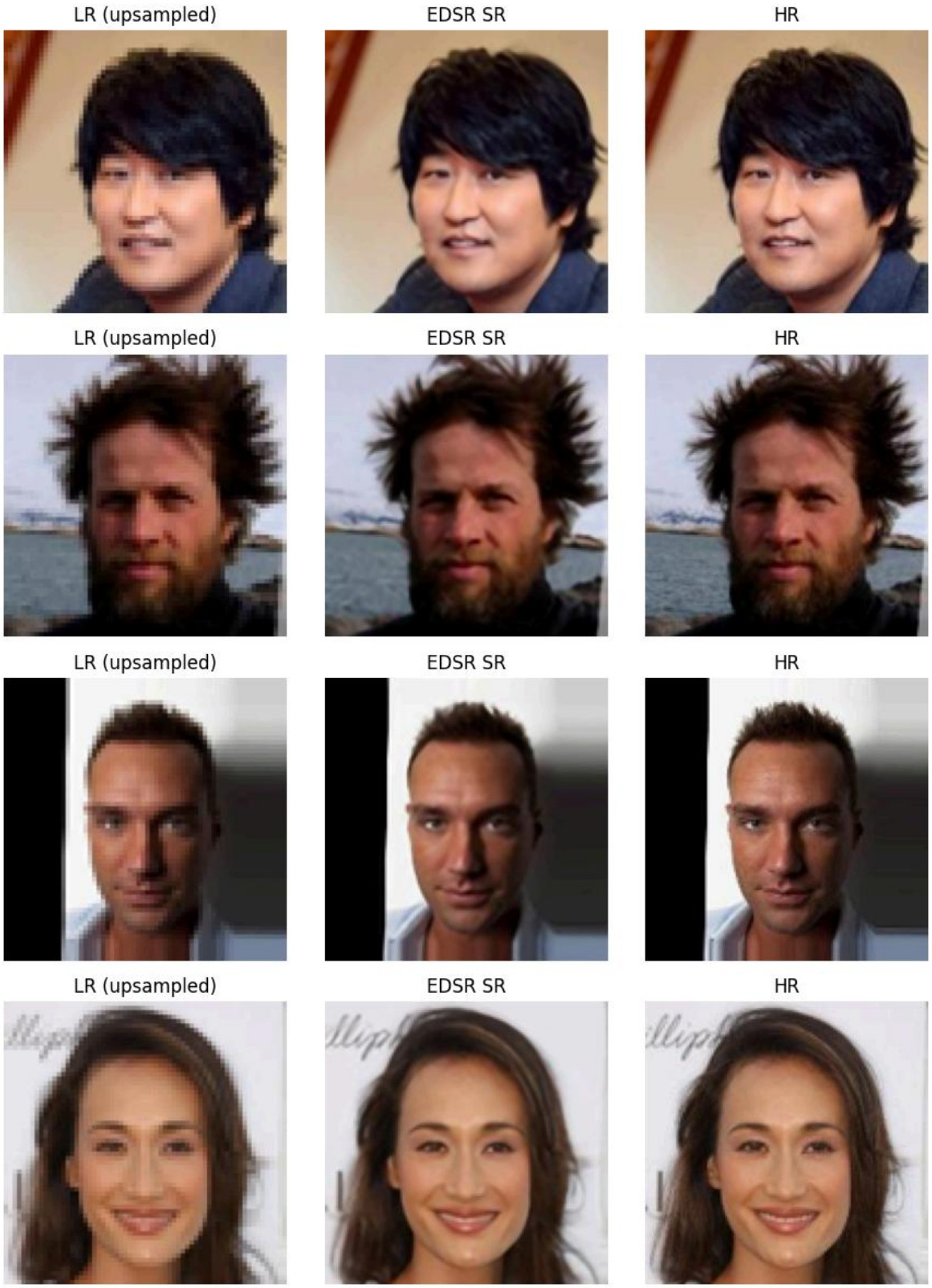
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[10], line 61
     59 for epoch in range(1, num_epochs + 1):
     60     print(f"\n===== Epoch {epoch}/{num_epochs} =====")
---> 61     train_one_epoch(edsr, train_loader, optimizer, criterion, device)
     62     psnr_val = eval_psnr(edsr, val_loader, device, max_batches=50)
     63     print(f"  → Val PSNR (first 50 batches): {psnr_val:.2f} dB")

Cell In[10], line 26, in train_one_epoch(model, loader, optimizer, criterion, dev
ice)
     23 loss.backward()
     24 optimizer.step()
---> 26 running_loss += loss.item()
     28 if (batch_idx + 1) % log_interval == 0:
     29     print(f"  [batch {batch_idx+1:5d}] loss = {running_loss / log_interva
l:.4f}")

KeyboardInterrupt:
```

In [11]:
```python
import torch
from pathlib import Path

ckpt_path = Path("outputs/checkpoints/edsr_quicksave.pth")
ckpt_path.parent.mkdir(parents=True, exist_ok=True)

torch.save(edsr.state_dict(), ckpt_path)
print("✅ EDSR checkpoint saved to:", ckpt_path)
```

```
✅ EDSR checkpoint saved to: outputs\checkpoints\edsr_quicksave.pth
```

In [14]:
```python
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt

@torch.no_grad()
def show_sr_examples(model, loader, device, num_images=4):
    model.eval()

    # 取一个 batch
    lr_batch, hr_batch = next(iter(loader))
    lr_batch = lr_batch.to(device)
    hr_batch = hr_batch.to(device)

    # 前向得到 SR，并先搬回 CPU
    sr_batch = model(lr_batch).clamp(0.0, 1.0).cpu()

    # HR 也搬回 CPU，后面画图用
    hr_batch = hr_batch.cpu()

    fig, axes = plt.subplots(num_images, 3, figsize=(9, 3 * num_images))
    if num_images == 1:
        axes = [axes]

    for i in range(num_images):
        # LR 这里最终也要在 CPU 上画图
        lr_up = F.interpolate(
            lr_batch[i].unsqueeze(0),
            size=hr_batch.shape[-2:],
            mode="nearest"
```

```
        ).cpu().squeeze(0)

        hr_img = hr_batch[i]      # 已经在 CPU
        sr_img = sr_batch[i]      # 已经在 CPU

        axes[i][0].imshow(lr_up.permute(1, 2, 0))
        axes[i][0].set_title("LR (upsampled)")
        axes[i][0].axis("off")

        axes[i][1].imshow(sr_img.permute(1, 2, 0))
        axes[i][1].set_title("EDSR SR")
        axes[i][1].axis("off")

        axes[i][2].imshow(hr_img.permute(1, 2, 0))
        axes[i][2].set_title("HR")
        axes[i][2].axis("off")

    plt.tight_layout()
    plt.show()

# 调用
show_sr_examples(edsr, test_loader, device, num_images=4)
```


LR (upsampled)
EDSR SR
HR
LR (upsampled)
EDSR SR
HR
LR (upsampled)
EDSR SR
HR
LR (upsampled)
EDSR SR
HR

In [ ]: