图像质量评估实现自动对焦落地流程

1：特征提取

2.训练与评估（保存可部署模型）

3.在新 z-stack 上推理与选焦（给出最佳帧）

什么是IQA

IQA（Image Quality Assessment，图像质量评估）

IQA 是量化图像“感知质量”的技术 —— 即通过算法模拟人类视觉系统，判断图像是否清晰、有无噪声 / 失真，以及整体视觉可用性，核心价值在于指导图像存储、压缩、增强等实际应用（如社交媒体的图像后处理、推荐系统的内容筛选）。

1. 核心分类（论文重点关注 NR-IQA）

● **全参考 IQA（FR-IQA）**：需同时输入“失真图像”和“无失真参考图像”（如 SSIM、LPIPS），但“野外图像”（如用户拍摄的荧光显微镜图、手机照片）通常无参考图，适用性受限。

● **无参考 IQA（NR-IQA）**：无需参考图，直接从失真图像中提取质量特征（如 BRISQUE、CONTRIQUE），是本文聚焦的方向。但传统 NR-IQA 存在短板：

○ 传统方法（如基于自然场景统计）依赖手工特征，泛化性差，对“野外复杂失真”（如混合模糊 + 噪声）评估不准；

○ 部分深度学习方法需大量“失真 – 质量分数”配对数据，标注成本高，且对未见过的失真 / 内容泛化差。

参考论文：Re-IQA: Unsupervised Learning for Image Quality Assessment in the Wild（https://arxiv.org/abs/2304.00451）

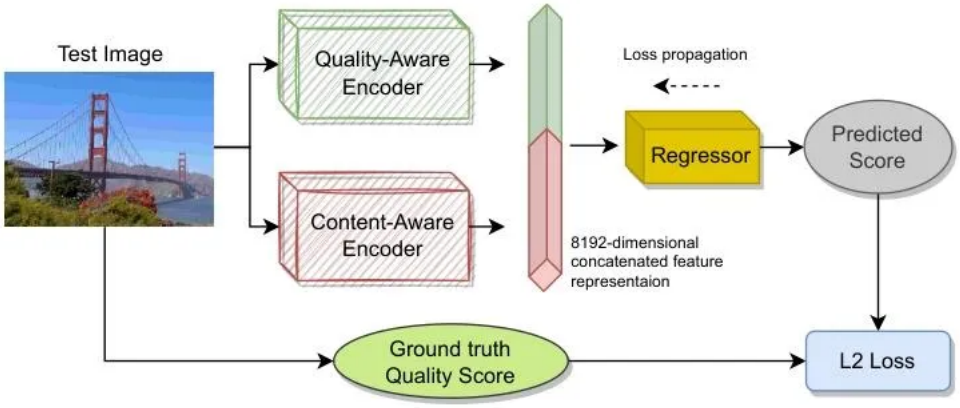
Re-IQA（Rethinking-IQA）

论文提出的无监督 NR-IQA 框架，核心是通过**“分离学习内容与质量特征”**，解决传统 IQA 泛化差、依赖监督数据的问题，实现对“野外图像”和合成失真图像的高质量评估。

1. 核心设计思路

针对“IQA 中内容与失真相互影响”的关键问题（如同一失真在不同内容图像上感知差异大），Re-IQA 采用“混合专家（Mixture of Experts）”策略：独立训练两个编码器，分别学习“高层内容特征”

和“低层质量特征”，再拼接特征训练轻量回归器预测质量分数，且全程无监督（无需人工标注的质量分数）。



图像质量评估（IQA）分数预测采用**两个编码器**，这两个编码器经训练以执行互补任务，即分别学习内容感知与质量感知图像表征。在回归器学习将图像表征映射到质量预测结果的过程中，两个编码器的权重保持冻结状态。

2. 关键结构与创新（结合论文细节）

模块	功能与实现
内容感知编码器	– 基于 MoCo–v2 无监督预训练（ImageNet 数据集），学**高层语义内容特征**（如物体 / 细胞轮廓）； – 解决“内容依赖失真感知”问题 —— 让模型先理解图像“画了什么”，再判断质量。
质量感知编码器	– 改进 MoCo–v2 框架，核心创新： ① 25 种失真 + 5 级强度的增强库（如高斯模糊、压缩失真），迫使模型学“失真规律”而非特定内容； ② OLA 裁剪（控制两裁剪区域重叠 10–30%），生成“相似质量”样本；③ 半交换策略（交换部分失真样本对），让模型聚焦“失真差异”而非内容；– 最终学**低层质量特征**（如噪声程度、边缘完整性）。

质量回归器	– 拼接两个编码器的 128 维特征（共 256 维），训练单一层感知器；– 冻结编码器权重，仅训回归器 —— 避免微调破坏无监督学到的通用特征，适配不同数据集时无需重新训练编码器。

3. 核心优势与性能

- **无监督高效**：无需“失真 – 质量分数”配对数据，仅用原始图像即可训练，降低标注成本；
- **泛化性强**：在“野外图像”（如 KonIQ–10K，SRCC=0.914）和合成失真图像（如 CSIQ，SRCC=0.947）上均达 SOTA，解决传统 IQA“对野外图像泛化差”的问题；
- **特征互补**：内容 + 质量特征结合，避免单一特征（如仅质量特征）对复杂内容图像评估不准的问题。

如何实现自动对焦+图像质量评估

传统自动对焦模型的核心实现逻辑：靠“清晰度评价函数”驱动

清晰度评价函数（Sharpness Evaluation Function, SEF）是传统自动对焦的“核心判断标准”—— 因为“清晰”是主观感知，传统方法通过数学函数量化图像的“清晰程度”，再结合“搜索策略”找到使评价函数值最大的焦距位置（清晰图像的评价函数值通常最大，失焦时最小）。

传统 AF 的完整实现流程分 3 步：

1. **初始化**：确定对焦区域（如中心区域、用户选择的目标区域），设定焦距调整范围（如镜头可移动的距离区间）；
2. **搜索与评价**：按“搜索策略”逐步调整焦距，每次调整后采集图像，用“清晰度评价函数”计算当前图像的清晰值；
3. **聚焦锁定**：找到清晰值最大的焦距位置，将镜头固定在此位置，完成对焦。

清晰度评价函数能不能替换成基于深度学习的图像质量评价模型?

把“清晰度函数→IQA”用于**荧光显微自动对焦**是可行的，但要把**“通用图像质量分”**（Re-IQA、MOS等）改造成**“对焦相关”**的分。

核心思路是：
用的显微镜 z-stack 数据，训练/微调一个“对焦质量”模型，让分数随离焦量单调变化，最好在焦面附近达到峰值。

目标定义（和传统清晰度函数的关系）

- **任务：给定一组 z-stack（不同焦面的图像），模型输出分数 $S(z)$ ，在真实焦面 z^* 处分数最大，离焦越远分数越低。**
- **与传统清晰度指标不同：不只看边缘，也要对噪声、暗场、光漂白、背景空场有鲁棒性；对浅景深/稀疏荧光等显微特性更加适配。**

数据与标注

1.1 采集策略

- 为每个视野采一段 **z-stack**：覆盖预期焦点上下（如 $\pm 5\text{–}10\mu m$ ），步进 $0.1\text{–}0.5\mu m$ 。
- 记录： `sample_id, stack_id, z_position(um), image_path` 。
- 多种条件覆盖：不同物镜/数值孔径、曝光、光强、荧光通道（GFP/RFP/…）、成分复杂度与SNR。

1.2 焦面“标签”的三种做法（任选其一或混合）

1. **人工标注**：人工标出最清晰切片 z^* 。量少但最可靠。(**暂时选择的方法**)

> 2. **教师指标生成**（弱监督）：用若干传统清晰度函数（Laplacian 方差、Tenengrad、SMD、Brenner、Tamura、FFT 高频能量等）**集成**，在每个 stack 选出峰值 z^* 。
> 3. **物理 proxy**：若已知离焦量或 PSF 模糊核 σ ，可用 $|z - z^*|$ 或 σ 作为连续标签。

训练时**不需要MOS**，而是用“**距离焦面更近**的一张 > 更远的一张”的**排序关系**即可。

1.3 ROI/掩膜

荧光常有**大面积背景**。为避免“背景主导”，请：

- 用阈值/显著性/分割得到 **前景掩膜**（细胞/结构区域）。
- **打分时只在 ROI 内**提取特征与汇聚（见 §4 里的“Top-k/分位数池化”）。

进一步增强（效果会明显更好）

- **Top-k/分位数池化**：把一张图切成 NNN 个 ROI/patch，取每张的 **前 k% 分数均值** 或 **P80–P95 分位数** 作为整图分，避免背景拖分。
- **多任务**：同时回归 $|z - z^*|$（或 PSF σ）+ 排序损失，提升单调性。
- **合成离焦**：在真实图上用光学一致的 PSF（如圈差、球差）做轻微卷积模拟离焦，扩充样本。
- **域自适应**：不同通道/物镜建**小头部**做条件化（FiLM/小 MLP），或每域微调最后层。
- **在线重校准**：设备更换或试剂变化后，用少量新 stack（几十个）做快速微调（只训头部，10–30 分钟搞定）。

结论

- **可行而且效果通常优于单一清晰度函数**，关键在于：用 **z-stack 的排序/回归监督** 让分数随离焦量单调变化；在显微域做 **ROI、分位数池化** 与 **鲁棒增强**；并用**你的设备/样本**做**轻量微调或回归头重训**。
- 以上代码骨架直接能跑通一个基线，可以先用 **冻结 ResNet18 + 排序头**起步，再替换为 **Re-IQA 特征 + 排序头**，或者走**端到端微调** 提升上限。

两个路线：模型重训和模型微调

路线 A：冻结特征 + 训练对焦头

- 特征：可用已经跑通的 **Re-IQA 双分支 8192 维特征**（或更轻的 ResNet18/ConvNeXt 特征）。
- 头部：**排序/回归头**（见 §3 的损失函数）。
- 优点：数据需求低、训练快（分钟级），和你的现有流程无缝。

路线 B：端到端微调（中等开发量）

- Backbone 用 ResNet18/MobileNetV3，**第一层改为 1 通道**（荧光通常灰度），或把灰度复制到 3

通道。

- 在你的显微数据上用**排序/回归损失**微调（最好只微调后半段/最后若干层，避免过拟合）。
- 可加入**自监督预训练**（MoCo/SimCLR）在海量无标注显微图上学更稳健的底层特征，再做排序微调。

已完成的工作—— 通用 IQA（Re-IQA 编码器 + KonIQ 回归头）”改造成“显微镜对焦质量”的模型

最省事、最稳妥的做法是：

- **不动两套编码器**（content/quality），继续当**固定特征提取器**；
- 用你的**显微 z-stack 数据**，在这些特征之上**重训一个很轻的头**，让分数在焦面附近最高、离焦越远越低；
- 先用**简单的监督信号**（例如 $-|z - z^*|$ 或高斯形状的“对焦分”）做**回归**，必要时再升级到**成对排序损失**（RankNet/BPR）；

训练步骤

步骤 1：生成 CSV 列出图像路径

1. **遍历每个文件夹**（如 `S001`，`S002`，...），并为每个图像生成一行包含该图像完整路径的条目。
2. **输出 CSV 文件**，每行表示一个图像的路径。

images_path.py Python

```python
import os
import csv
def generate_image_paths_csv(stack_root, output_csv):
    # 创建CSV文件并写入
    with open(output_csv, mode='w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(['image_path'])  # 写入表头
    # 遍历每个堆栈文件夹
    for stack_id in os.listdir(stack_root):
        stack_dir = os.path.join(stack_root, stack_id)
        if os.path.isdir(stack_dir):  # 确保是目录
            # 遍历堆栈中的所有图像
            for filename in os.listdir(stack_dir):
                if filename.endswith('.tif'):
                    # 获取图像完整路径
                    image_path = os.path.join(stack_dir, filename)
                    # 写入CSV
                    writer.writerow([image_path])
#设定路径
stack_root = r'D:\C++workspace\IQA\ReIQA\train_data'  # 根文件夹路径，存储 S001、S002、... 文件夹
output_csv = 'path_to_z-stack_images.csv'
#生成 CSV
generate_image_paths_csv(stack_root, output_csv)
```

生成的 `path_to_z-stack_images.csv` 文件内容示例：

Plain Text

```
image_path
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-1.tif
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-10.tif
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-11.tif
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-12.tif
...
D:\C++workspace\IQA\ReIQA\train_data\S011\S011_z8.tif
D:\C++workspace\IQA\ReIQA\train_data\S011\S011_z9.tif
```

生成 `focus_scores.csv`

使用 **高斯函数** 来生成离焦图像的分数，会导致离焦较远的图像分数趋近于零。特别是在大于某个距离（如 5 μm）时，分数可能变得非常小，甚至几乎为零，这会影响模型的训练效果和推理结果。

解决方案：使用 带平移和缩放的高斯函数 或者 线性衰减函数 来确保离焦图像的分数在一定范围内变化，并且避免过于极端的分数。

1. 改进高斯函数（带平移和缩放）

我们可以对高斯函数进行一些调整，使得分数不会迅速下降为零，并且保留更宽的有效范围。通过 **引入平移和缩放参数**，来控制高斯函数的衰减速度和分数的范围。

新的高斯函数公式：

$$\text{score}(z) = \frac{1}{1 + \left(\frac{|z|}{\sigma}\right)^2}$$

这个修改后的公式相比于原始的高斯函数，它会逐渐平滑地减小离焦图像的分数，但不会像普通的高斯函数那样快速趋近零。

- `z` 是图像的 `z_um` 值。
- `σ` 仍然是标准差，用来控制评分的衰减速率。
- 我们使用 **分母添加了** `1`，这将防止评分降得过快，特别是对于离焦较远的图像。

`focus_scores.csv` 文件用于存储每张图像的 **评分（Focus Score）** 和 `z_um` **值**（焦点位置）。这些分数是你在训练过程中用来标注图像的质量分数（MOS）。

文件内容：

- 每一行包含以下内容：
  - `image_path`：图像的路径（与 `path_to_z-stack_images.csv` 相同）
  - `z_um`：图像在 **z 轴上的位置**（正数表示焦点上方，负数表示焦点下方）
  - `focus_score`：该图像的 **质量评分**，一般用高斯函数生成。

focus_scores.py Python

```python
import csv
import numpy as np
import os
import re


# 改进的高斯评分函数（带平移和缩放）
def generate_focus_score(z, sigma=1.0):
    """
    改进的高斯评分函数，用于控制离焦图像的分数，避免过快接近0。
    """
    return 1.0 / (1 + (np.abs(z) / sigma) ** 2)


def extract_z_from_filename(filename):
    """
    从文件名中提取z-μm值
    例如：S001_z-10.tif 提取出 z=-10，S001_z1.tif 提取出 z=1
    """
    # 使用正则表达式从文件名中提取 z 值
    match = re.search(r'z([+-]?\d+)', filename)
    if match:
        return int(match.group(1))  # 返回提取的 z 值
    else:
        return None  # 如果无法提取 z 值，返回 None


def generate_focus_scores_csv(stack_root, output_csv, sigma=1.0):
    """
    遍历每个堆栈文件夹，生成每张图像的焦点分数，并保存到 focus_scores.csv 文件。

    stack_root: 堆栈数据的根文件夹路径
    output_csv: 输出的 CSV 文件路径
    sigma: 高斯函数的标准差（控制分数衰减的速度）
    """
    # 创建CSV文件并写入
    with open(output_csv, mode='w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(['image_path', 'z_um', 'focus_score'])  # 写入表头

        # 遍历每个堆栈文件夹
        for stack_id in os.listdir(stack_root):
            stack_dir = os.path.join(stack_root, stack_id)
            if os.path.isdir(stack_dir):  # 确保是目录
                # 遍历堆栈中的所有图像
```

```
                for filename in os.listdir(stack_dir):
                    if filename.endswith('.tif'):
                        # 从文件名中提取 z_um 值
                        z_value = extract_z_from_filename(filename)

                        if z_value is not None:
                            # 根据 `z_um` 生成改进的高斯评分
                            focus_score = generate_focus_score(z_value, si
gma)

                            # 获取图像完整路径
                            image_path = os.path.join(stack_dir, filename)
                            # 写入CSV
                            writer.writerow([image_path, z_value, focus_sc
ore])


# 设定路径
stack_root = r'D:\C++workspace\IQA\ReIQA\train_data'  # 根文件夹路径，存储 S0
01、S002、... 文件夹
output_csv = 'focus_scores.csv'

# 生成 CSV
generate_focus_scores_csv(stack_root, output_csv, sigma=2.0)
```

生成的 `focus_scores.csv` 文件内容示例：

```
Plain Text
image_path,z_um,focus_score
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-1.tif,-1,0.8
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-10.tif,-10,0.038461538461
538464
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-11.tif,-11,0.032
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-12.tif,-12,0.027027027027
02703
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-13.tif,-13,0.023121387283
236993
D:\C++workspace\IQA\ReIQA\train_data\S001\S001_z-14.tif,-14,0.02
...
D:\C++workspace\IQA\ReIQA\train_data\S011\S011_z8.tif,8,0.0588235294117647
05
D:\C++workspace\IQA\ReIQA\train_data\S011\S011_z9.tif,9,0.0470588235294117
64
```

步骤 2： 构建和测试自动对焦图像评估模型

1：特征提取

你已经上传了特征提取脚本 **`extract_mic_features_roi_patch.py`**，它会从图像中提取出 **8192 维的特征**，并将它们保存在一个 `.npz` 文件中。**extract_mic_features_roi_patch.py文件如下：**

extract_mic_features_roi_patch.py Python

```python
# extract_mic_features_roi_patch.py
# 可直接运行的无-global 版本（PowerShell/Linux/Mac均可）
import os, csv, argparse, random, math, ntpath
import numpy as np
import tifffile as tiff
import cv2
import torch
import torch.nn.functional as F
from torchvision import transforms

# ====== 复用你仓库里的加载函数（保持一致）======
LOAD_FROM_REPO = True
try:
    from extract_reiqa_features_koniq import load_encoder as _repo_load_encoder
except Exception:
    LOAD_FROM_REPO = False
    print("[WARN] 未找到 extract_reiqa_features_koniq.load_encoder；"
          "请把本脚本放在你的 Re-IQA 工程根目录或把该函数所在文件加入 PYTHONPATH。")

# ====== 常量（仅作默认值展示，不会被修改）======
DEFAULT_PATCH_SIZE      = 256
DEFAULT_PATCHES_PER_IMG = 32
DEFAULT_TOPK_FRAC       = 0.20
DEFAULT_MIN_ROI_COVER   = 0.40
DEFAULT_P_LOW           = 1.0
DEFAULT_P_HIGH          = 99.8
ROI_MIN_AREA            = 64 * 64

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# ---------- 工具函数 ----------

def read_image_float01(path, p_low, p_high):
    """读取TIF并按分位数标准化到[0,1]；兼容单/多通道，自动转灰度"""
    img = tiff.imread(path)  # (H,W) 或 (H,W,C)
    if img.ndim == 3:
        img = img.mean(axis=2)
    img = img.astype(np.float32)

    lo, hi = np.percentile(img, [p_low, p_high])
    if hi <= lo:
        lo, hi = img.min(), img.max()
        if hi <= lo:  # 极端兜底
```

```
            hi = lo + 1.0
    x = np.clip((img - lo) / (hi - lo), 0.0, 1.0)
    return x  # (H,W) in [0,1]

def make_roi_mask(img01):
    """Otsu 阈值 + 开闭运算 + 小连通域过滤；失败时放宽为全图"""
    u8 = (img01 * 255).astype(np.uint8)
    _, mask = cv2.threshold(u8, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    k = np.ones((5,5), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, k, iterations=1)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, k, iterations=1)

    num, labels, stats, _ = cv2.connectedComponentsWithStats(mask, connectivity=8)
    out = np.zeros_like(mask)
    for i in range(1, num):
        if stats[i, cv2.CC_STAT_AREA] >= ROI_MIN_AREA:
            out[labels == i] = 255
    if out.sum() == 0:
        out[:] = 255
    return (out > 0).astype(np.uint8)

def sample_patch_positions(mask, patch_size, n, min_cover=0.4, max_trials=10000):
    """在 ROI 内随机采样 patch 左上角坐标；不足时用网格补齐，再不足放宽全图"""
    H, W = mask.shape
    ps = patch_size
    cand, trials = [], 0
    ys = range(0, H-ps+1)
    xs = range(0, W-ps+1)
    while len(cand) < n and trials < max_trials:
        y = random.choice(ys); x = random.choice(xs)
        if mask[y:y+ps, x:x+ps].mean() >= min_cover:
            cand.append((y, x))
        trials += 1
    if len(cand) < n:
        step = max(1, (min(H, W) - ps) // int(math.sqrt(n) + 1))
        for y in range(0, H-ps+1, step):
            for x in range(0, W-ps+1, step):
                if mask[y:y+ps, x:x+ps].mean() >= min_cover:
                    cand.append((y, x))
                if len(cand) >= n: break
            if len(cand) >= n: break
    if len(cand) == 0:
        for _ in range(n):
            y = random.randint(0, H-ps); x = random.randint(0, W-ps)
            cand.append((y, x))
```

```
    return cand[:n]

def laplacian_var(patch01):
    """拉普拉斯方差作为纹理强度评分"""
    g = (patch01 * 255).astype(np.uint8)
    return float(cv2.Laplacian(g, cv2.CV_64F).var())

to_tensor = transforms.ToTensor()
imagenet_norm = transforms.Normalize(mean=[0.485,0.456,0.406],
                                     std =[0.229,0.224,0.225])

def prep_for_content(patch01):
    rgb = np.repeat(patch01[..., None], 3, axis=2)
    t = to_tensor(rgb)
    t = imagenet_norm(t)
    return t

def prep_for_quality(patch01):
    rgb = np.repeat(patch01[..., None], 3, axis=2)
    t = to_tensor(rgb)
    return t

def load_encoder(ckpt_path, normalize_for_content=True):
    """复用你工程里的权重加载逻辑"""
    if not LOAD_FROM_REPO:
        raise RuntimeError("未能导入仓库的 load_encoder。请确认脚本位置或 PYTHONPATH。")
    model = _repo_load_encoder(ckpt_path, device, normalize=normalize_for_content)
    model.eval()
    return model

def forward_encoder(model, x4d):
    with torch.no_grad():
        m = model.module if hasattr(model, "module") else model
        if hasattr(m, "encoder"):
            f = m.encoder(x4d)
        else:
            f = m(x4d)
        if f.ndim == 4:
            f = F.adaptive_avg_pool2d(f, 1).flatten(1)
        return f  # [B, 2048]

def extract_feat_for_one_image(path, content_model, quality_model,
                               patch_size, n_patches, topk_frac,
                               p_low, p_high, min_roi_cover):
    """单帧：ROI→随机patch→双尺度×2分支→拼接8192→按纹理Top-k均值"""
    img01 = read_image_float01(path, p_low, p_high)
```

```
    mask  = make_roi_mask(img01)
    pos   = sample_patch_positions(mask, patch_size, n_patches, min_cover=min_roi_cover)

    feats, scores = [], []
    for (y, x) in pos:
        p = img01[y:y+patch_size, x:x+patch_size]
        s = laplacian_var(p); scores.append(s)
        p_half = cv2.resize(p, (patch_size//2, patch_size//2), interpolation=cv2.INTER_AREA)

        tc1 = prep_for_content(p).unsqueeze(0).to(device)
        tc2 = prep_for_content(p_half).unsqueeze(0).to(device)
        tq1 = prep_for_quality(p).unsqueeze(0).to(device)
        tq2 = prep_for_quality(p_half).unsqueeze(0).to(device)

        f1c = forward_encoder(content_model, tc1)
        f2c = forward_encoder(content_model, tc2)
        f1q = forward_encoder(quality_model, tq1)
        f2q = forward_encoder(quality_model, tq2)

        f = torch.cat([f1c, f2c, f1q, f2q], dim=1).squeeze(0).cpu().numpy().astype(np.float32)
        feats.append(f)

    feats  = np.stack(feats, axis=0)     # [P,8192]
    scores = np.asarray(scores)          # [P]
    k = max(1, int(math.ceil(len(scores) * topk_frac)))
    top_idx = np.argsort(-scores)[:k]
    feat_img = feats[top_idx].mean(axis=0)  # [8192]
    return feat_img

# ---------- 主流程 ----------
def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--csv_images", required=True, help="CSV, 含一列 image_path")
    ap.add_argument("--content_ckpt", required=True)
    ap.add_argument("--quality_ckpt", required=True)
    ap.add_argument("--out_npz", required=True)

    ap.add_argument("--patch_size",      type=int,   default=DEFAULT_PATCH_SIZE)
    ap.add_argument("--patches_per_img", type=int,   default=DEFAULT_PATCHES_PER_IMG)
    ap.add_argument("--topk_frac",       type=float, default=DEFAULT_TOPK_FRAC)
```

```
    ap.add_argument("--min_roi_cover",   type=float, default=DEFAULT_MIN_ROI_COVER)
    ap.add_argument("--p_low",           type=float, default=DEFAULT_P_LOW)
    ap.add_argument("--p_high",          type=float, default=DEFAULT_P_HIGH)
    ap.add_argument("--seed",            type=int,   default=42)
    args = ap.parse_args()

    random.seed(args.seed); np.random.seed(args.seed); torch.manual_seed(args.seed)

    content_model = load_encoder(args.content_ckpt, normalize_for_content=True)
    quality_model = load_encoder(args.quality_ckpt, normalize_for_content=False)

    # 读取 CSV（允许列名不是 image_path，则取第一列作为路径）
    img_paths = []
    with open(args.csv_images, newline='', encoding='utf-8') as f:
        r = csv.DictReader(f)
        col = "image_path" if "image_path" in r.fieldnames else r.fieldnames[0]
        for t in r:
            p = t[col]
            if os.path.isfile(p):
                img_paths.append(p)
            else:
                print(f"[WARN] 跳过不存在的路径: {p}")

    X, names = [], []
    for i, p in enumerate(img_paths):
        try:
            feat = extract_feat_for_one_image(
                p, content_model, quality_model,
                patch_size=args.patch_size,
                n_patches=args.patches_per_img,
                topk_frac=args.topk_frac,
                p_low=args.p_low, p_high=args.p_high,
                min_roi_cover=args.min_roi_cover,
            )
            X.append(feat); names.append(ntpath.basename(p))
        except Exception as e:
            print(f"[ERR] 提特征失败 {p}: {e}")

        if (i + 1) % 20 == 0:
            print(f"[INFO] 已完成 {i+1}/{len(img_paths)}")
```

```
    if len(X) == 0:
        raise RuntimeError("没有成功提取到任何特征，请检查 CSV 路径列与图像文件是否存在。")

    X = np.stack(X, axis=0).astype(np.float32)
    names = np.asarray(names)
    np.savez(args.out_npz, X=X, names=names)
    print(f"[DONE] 保存到 {args.out_npz} | N={len(names)} | dim={X.shape[1]}")

if __name__ == "__main__":
    main()
```

1.1 准备好输入文件

确保已经准备好 `path_to_z-stack_images.csv` 和 `focus_scores.csv` 文件，这两个文件包含了图像路径、对应的 `z_um` 值和焦点分数（focus score）。

- `path_to_z-stack_images.csv`：每一行包含一个图像的路径。
- `focus_scores.csv`：每一行包含图像路径、`z_um` 值和焦点分数（`focus_score`）。

1.2 运行特征提取脚本

运行以下命令，提取图像的特征并保存为 `mic_features.npz` 文件，供后续训练使用：

PowerShell特征提取代码 Bash

```
python .\extract_mic_features_roi_patch.py `
  --csv_images "D:\C++workspace\IQA\ReIQA\path_to_z-stack_images.csv" `
  --content_ckpt "D:\C++workspace\IQA\ReIQA\reiqa_ckpts\content_aware_r50.pth" `
  --quality_ckpt "D:\C++workspace\IQA\ReIQA\reiqa_ckpts\quality_aware_r50.pth" `
  --out_npz "D:\C++workspace\IQA\ReIQA\mic_features.npz" `
  --patch_size 256 `
  --patches_per_img 32 `
  --topk_frac 0.2 `
  --min_roi_cover 0.4 `
  --p_low 1.0 `
  --p_high 99.8
```

2.训练与评估（保存可部署模型）

把下面内容保存为 `train_focus_regressor.py`。它做的事：

- 对齐 `mic_features.npz` 与 `focus_scores.csv`（通过 basename 关联）
- 生成标签（默认读取 `focus_score`；若没有，也可从 `z_um` 用改进高斯计算）

- 用 **GroupKFold(=LOSO)**（按 stack_id 分组）做评估：给出 **SRCC / PLCC**
- 训练最终模型（全量数据），保存成 `focus_regressor.pkl`
- 模型是一个 **Pipeline：StandardScaler → Ridge(alpha=1000)**，稳定好用

> ⚠️ 注意：脚本会从 `image_path` 提取 `stack_id`（例如路径中的 `\S007\` → `S007`）作为“组”，用来做留一堆栈交叉验证（LOSO），更贴近真实部署场景。

train_focus_regressor.py Python

```python
# train_focus_regressor.py
import os, re, argparse, warnings
import numpy as np
import pandas as pd
from scipy.stats import spearmanr, pearsonr
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GroupKFold
import joblib

def extract_basename(p):
    return os.path.basename(p)

def extract_stack_id(p):
    # 期望路径中含 ...\S001\... 或 .../S001/...
    m = re.search(r"[\\/](S\d{3,})[\\/]", p)
    return m.group(1) if m else "UNK"

def score_from_z(z, sigma=2.0):
    # 改进型“高斯风格”分数：1 / (1 + (|z|/sigma)^2)
    return 1.0 / (1.0 + (abs(float(z)) / float(sigma))**2)

def align_features_labels(npz_path, csv_path, target_col="focus_score", fallback_from_z=True, sigma=2.0):
    # 加载特征
    data = np.load(npz_path, allow_pickle=True)
    X = data["X"]                    # [N, 8192]
    names = data["names"]            # [N] basenames

    # 加载标签 CSV
    df = pd.read_csv(csv_path)

    # 取 basename 以便 join
    if "image_path" in df.columns:
        df["basename"] = df["image_path"].apply(extract_basename)
        df["stack_id"] = df["image_path"].apply(extract_stack_id)
    else:
        # 如果第一列就是路径
        first_col = df.columns[0]
        df["basename"] = df[first_col].astype(str).apply(extract_basename)
        df["stack_id"] = df[first_col].astype(str).apply(extract_stack_id)

```

```
    # 兜底：若没有 focus_score，则尝试从 z_um 生成
    if target_col not in df.columns:
        if fallback_from_z and ("z_um" in df.columns):
            warnings.warn(f"[WARN] 未找到列 `{target_col}`；从 z_um 生成分数（sigma={sigma}）。")
            df[target_col] = df["z_um"].apply(lambda z: score_from_z(z, sigma=sigma))
        else:
            raise ValueError(f"CSV 中既无 `{target_col}`，也无 `z_um`，无法构造监督信号。")

    # 建立索引映射：basename -> 行
    df_map = df.set_index("basename")
    rows, groups, missed = [], [], []

    for i, nm in enumerate(names):
        if nm in df_map.index:
            rows.append(df_map.loc[nm])
            groups.append(df_map.loc[nm]["stack_id"])
        else:
            missed.append(nm)

    if len(rows) == 0:
        raise RuntimeError("一个样本都没对齐上。请检查 mic_features.npz 的 names 与 CSV 的 image_path/basename 是否一致。")

    if missed:
        print(f"[WARN] 有 {len(missed)} 个特征未在 CSV 对应到标签（将被跳过）。例如：{missed[:5]}")

    Y = np.array([r[target_col] for r in rows], dtype=np.float32)
    G = np.array(groups)
    # 只保留能对齐的 X
    mask_keep = np.isin(names, list(df_map.index))
    X_keep = X[mask_keep]

    # 基本 sanity check
    if X_keep.shape[0] != Y.shape[0]:
        raise RuntimeError(f"对齐后的样本数不一致：X={X_keep.shape[0]} vs Y={Y.shape[0]}")

    return X_keep, Y, G, names[mask_keep]

def evaluate_loso(X, Y, G, alpha=1000.0, n_splits=None):
    # GroupKFold 按 stack_id (G) 分组，相当于 LOSO
    unique_groups = np.unique(G)
    n_splits = len(unique_groups) if n_splits is None else n_splits
    gkf = GroupKFold(n_splits=n_splits)
```

```
    srccs, plccs = [], []
    for tr_idx, te_idx in gkf.split(X, Y, groups=G):
        model = make_pipeline(StandardScaler(with_mean=True, with_std=True),
                              Ridge(alpha=alpha, random_state=0))
        model.fit(X[tr_idx], Y[tr_idx])
        y_hat = model.predict(X[te_idx])

        srcc = spearmanr(Y[te_idx], y_hat).correlation
        plcc = pearsonr(Y[te_idx], y_hat)[0]
        srccs.append(srcc); plccs.append(plcc)

    return float(np.nanmean(srccs)), float(np.nanmean(plccs))

def train_full_and_save(X, Y, alpha, out_path):
    model = make_pipeline(StandardScaler(with_mean=True, with_std=True),
                          Ridge(alpha=alpha, random_state=0))
    model.fit(X, Y)
    joblib.dump(model, out_path)
    return model

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--features_npz", required=True)
    ap.add_argument("--scores_csv", required=True)
    ap.add_argument("--target", default="focus_score")
    ap.add_argument("--alpha", type=float, default=1000.0)
    ap.add_argument("--sigma", type=float, default=2.0, help="当从 z_um 构造分数时使用")
    ap.add_argument("--out_model", required=True)
    args = ap.parse_args()

    X, Y, G, kept_names = align_features_labels(
        args.features_npz, args.scores_csv, target_col=args.target, fallback_from_z=True, sigma=args.sigma
    )
    print(f"[INFO] 对齐样本: {len(Y)} | 特征维度: {X.shape[1]} | stack 组数: {len(np.unique(G))}")

    srcc, plcc = evaluate_loso(X, Y, G, alpha=args.alpha)
    print(f"[EVAL][LOSO]  Ridge alpha={args.alpha:.1f} | SRCC={srcc:.4f} | PLCC={plcc:.4f}")

    model = train_full_and_save(X, Y, alpha=args.alpha, out_path=args.out_model)
    print(f"[DONE] 已保存模型到: {args.out_model}")
```

```
if __name__ == "__main__":
    main()
```

在 PowerShell 中训练与评估

PowerShell训练回归头 Bash

```
python .\train_focus_regressor.py `
  --features_npz "D:\C++workspace\IQA\ReIQA\mic_features.npz" `
  --scores_csv   "D:\C++workspace\IQA\ReIQA\focus_scores.csv" `
  --target focus_score `
  --alpha 1000 `
  --out_model "D:\C++workspace\IQA\ReIQA\focus_regressor.pkl"
```

输出会给你：

- 对齐到的样本数、特征维度、stack 数
- **LOSO 的 SRCC / PLCC**（重要，越高越好）
- 最终保存的模型文件 `focus_regressor.pkl`

> - 如果你只在 `focus_scores.csv` 写了 `z_um` 而没有 `focus_score`，脚本会自动用公式 `1 / (1 + (|z|/sigma)^2)` 生成；你也可以把 `--sigma` 改成 1.5 或 3 试试。
> - 如果你不想用 LOSO，也可以改 `evaluate_loso` 里成普通 `KFold`，不过 LOSO 更符合实际部署（跨视野泛化）。

3.在新 z-stack 上推理与选焦（给出最佳帧）

把下面保存为 `predict_focus_on_stack.py`。它做的事：

- 读取你训练好的 `focus_regressor.pkl`
- 给一串**待测图像路径**（同一 z-stack）：
  a. 调用你已有的 `extract_mic_features_roi_patch.py` 逻辑**外部先生成**一个新的 `stack_features.npz`（建议沿用同一方式）
  b. 脚本读取该 npz，跑模型，打印每一帧的分数、排序、**最佳焦距帧**

如果你已经提前把这个 z-stack 的特征提成 npz，就直接用它；如果没有，就先用你的提特征脚本跑一遍（命令示例在下面“推理步骤”里）。

predict_focus_on_stack.py Python

```python
# predict_focus_on_stack.py
import os, argparse, numpy as np, joblib, re
import pandas as pd

def load_features(npz_path):
    data = np.load(npz_path, allow_pickle=True)
    return data["X"], data["names"]

def name_to_z(name):
    m = re.search(r"z([+-]?\d+)", name)  # 兼容 z-3 / z3 / z+3
    return int(m.group(1)) if m else None

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--model_path", required=True)
    ap.add_argument("--features_npz", required=True, help="该 z-stack 的特征（由你的提特征脚本得到）")
    args = ap.parse_args()

    model = joblib.load(args.model_path)
    X, names = load_features(args.features_npz)

    yhat = model.predict(X)  # 预测分数，越大越清晰
    z_vals = [name_to_z(n) for n in names]
    df = pd.DataFrame({"name": names, "z": z_vals, "score": yhat})
    df.sort_values(by="score", ascending=False, inplace=True)

    best = df.iloc[0]
    print("\n=== 该 z-stack 评分（前10） ===")
    print(df.head(10).to_string(index=False))
    print(f"\n[RESULT] 最佳帧: {best['name']} | z={best['z']} | 预测分数={best['score']:.6f}")

if __name__ == "__main__":
    main()
```

推理步骤（PowerShell 示例）

3.1 对待测 z-stack 提特征（用你已有的提特征脚本——跟训练时一致）

先做一个包含该 z-stack 所有图片路径的 CSV（单列 `image_path`），`stack_S012.csv`，

需要的输入：

假设你的 z-stack 文件位于：

```
D:\C++workspace\IQA\ReIQA\train_data\S012\
```

并且该文件夹下有：

```
S012_z-15.tif
S012_z-14.tif
...
S012_z0.tif
...
S012_z15.tif
```

需要生成一个包含所有图像**完整路径**的 CSV 文件，比如：

```
image_path
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z-15.tif
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z-14.tif
...
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z15.tif
```

stack_S012.csv的生成代码：

```Bash
import os
import csv
import argparse

def main():
    parser = argparse.ArgumentParser(description="Generate CSV listing all image paths for one z-stack")
    parser.add_argument("--stack_dir", required=True, help="Path to the z-stack folder (e.g. D:\\IQA\\ReIQA\\train_data\\S012)")
    parser.add_argument("--out_csv", required=True, help="Output CSV path (e.g. D:\\IQA\\ReIQA\\stack_S012.csv)")
    args = parser.parse_args()

    stack_dir = args.stack_dir
    out_csv = args.out_csv

    # 查找所有 tif 文件
    images = sorted([f for f in os.listdir(stack_dir) if f.lower().endswith(".tif")])

    if not images:
        raise RuntimeError(f"❌ 没有在 {stack_dir} 找到任何 .tif 图像！")

    # 写入 CSV
    with open(out_csv, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["image_path"])
        for img in images:
            full_path = os.path.join(stack_dir, img)
            writer.writerow([full_path])

    print(f"✅ 已生成 CSV：{out_csv}")
    print(f"共 {len(images)} 张图像。示例：")
    for img in images[:5]:
        print(" ", img)

if __name__ == "__main__":
    main()

```

在 PowerShell 中运行

```
python .\make_stack_csv.py `
  --stack_dir "D:\C++workspace\IQA\ReIQA\train_data\S012" `
  --out_csv "D:\C++workspace\IQA\ReIQA\stack_S012.csv"
```

打开 `stack_S012.csv` 应该像这样：

```
image_path
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z-15.tif
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z-14.tif
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z-13.tif
...
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z0.tif
...
D:\C++workspace\IQA\ReIQA\train_data\S012\S012_z15.tif
```

然后：

```
python .\extract_mic_features_roi_patch.py `
  --csv_images "D:\C++workspace\IQA\ReIQA\stack_S012.csv" `
  --content_ckpt "D:\C++workspace\IQA\ReIQA\reiqa_ckpts\content_aware_r50.pth" `
  --quality_ckpt "D:\C++workspace\IQA\ReIQA\reiqa_ckpts\quality_aware_r50.pth" `
  --out_npz "D:\C++workspace\IQA\ReIQA\stack_S012_features.npz" `
  --patch_size 256 `
  --patches_per_img 32 `
  --topk_frac 0.2 `
  --min_roi_cover 0.4
```

3.2 用训练好的模型做预测并选焦：

```
python .\predict_focus_on_stack.py `
  --model_path "D:\C++workspace\IQA\ReIQA\focus_regressor.pkl" `
  --features_npz "D:\C++workspace\IQA\ReIQA\stack_S012_features.npz"
```

输出会给出该 z-stack 每帧分数（前 10）以及**最佳帧的文件名与 z**。

最终运行了你给的代码以及命令行，得到了下面的输出结果：

```Bash
最终运行了你给的代码以及命令行，得到了下面的输出结果：
=== 该 z-stack 评分（前10） ===
                        name    z    score
 251025Lbb_Sample5_1_AA.tif None 1.768869
 251025Lbb_Sample5_2_AA.tif None 0.980967
251025Lbb_Sample5_17_AA.tif None 0.682788
 251025Lbb_Sample5_3_AA.tif None 0.444562
251025Lbb_Sample5_18_AA.tif None 0.437337
251025Lbb_Sample5_19_AA.tif None 0.301532
 251025Lbb_Sample5_4_AA.tif None 0.290893
251025Lbb_Sample5_11_AA.tif None 0.279037
251025Lbb_Sample5_20_AA.tif None 0.260261
251025Lbb_Sample5_14_AA.tif None 0.238200

[RESULT] 最佳帧: 251025Lbb_Sample5_1_AA.tif | z=None | 预测分数=1.768869

```

效果确实还不错，后续需要提高数据量，改进代码模型。