In [1]:
```
!pip install gdown
import gdown
import zipfile
file_id = "1znqUeFzYz-DeAE3dYXD17qoMPK82Whji"
output = "dataset.zip"
gdown.download(f"https://drive.google.com/uc?id={file_id}", output, quiet =

with zipfile.ZipFile("dataset.zip", "r") as zip_ref:
  zip_ref.extractall("/content/dataset")
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.11/dist-packa
ges (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/d
ist-packages (from gdown) (4.13.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pa
ckages (from gdown) (3.17.0)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.11/
dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packag
es (from gdown) (4.67.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/di
st-packages (from beautifulsoup4->gdown) (2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/py
thon3.11/dist-packages (from beautifulsoup4->gdown) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.11/dist-packages (from requests[socks]->gdown) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dis
t-packages (from requests[socks]->gdown) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
11/dist-packages (from requests[socks]->gdown) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
11/dist-packages (from requests[socks]->gdown) (2025.1.31)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/pyth
on3.11/dist-packages (from requests[socks]->gdown) (1.7.1)
Downloading...
From (original): https://drive.google.com/uc?id=1znqUeFzYz-DeAE3dYXD17qoMPK8
2Whji
From (redirected): https://drive.google.com/uc?id=1znqUeFzYz-DeAE3dYXD17qoMP
K82Whji&confirm=t&uuid=920e2e13-edb0-44d5-8d5f-1f2d7f66783c
To: /content/dataset.zip
100%|██████████| 2.89G/2.89G [00:58<00:00, 49.3MB/s]
```

In [2]:
```
file_id = "1uJmDZw649XS-r-dYs9WD-OPwF_TIroVw"
output = "datasetResolution.zip"

gdown.download(f"https://drive.google.com/uc?id={file_id}", output, quiet=Fa

with zipfile.ZipFile(output, "r") as zip_ref:
    zip_ref.extractall("/content/datasetResolution")
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1uJmDZw649XS-r-dYs9WD-OPwF_TIroVw
From (redirected): https://drive.google.com/uc?id=1uJmDZw649XS-r-dYs9WD-OPwF_TIroVw&confirm=t&uuid=c7024567-ef35-4a64-ad74-8882499d1db3
To: /content/datasetResolution.zip
100%|██████████| 533M/533M [00:12<00:00, 41.6MB/s]
```

In [3]:
```python
import os
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms
from torch.cuda.amp import GradScaler, autocast
import torch.nn.functional as F
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score
from skimage.metrics import structural_similarity as ssim
import math
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from tqdm import tqdm
```

In [4]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

In [5]:
```python
class MAEDataset(Dataset):
    def __init__(self, file_paths, mask_ratio=0.85, patch_size=8, transform=
        self.file_paths = file_paths
        self.mask_ratio = mask_ratio
        self.patch_size = patch_size
        self.transform = transform

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        img = np.load(self.file_paths[idx])
        img = torch.tensor(img, dtype=torch.float32)
        # Adding channel dimension
        if img.ndim == 2:
            img = img.unsqueeze(0)
        if self.transform:
            img = self.transform(img)
        # Creating a masked version of the image.
        masked_img, mask = self.apply_mask(img)
        return masked_img, img, mask

    def apply_mask(self, img):
        """
        ivides the image into non-overlapping patches and randomly masks a
```

```
        """
        C, H, W = img.shape
        patch_size = self.patch_size
        n_patches_h = H // patch_size
        n_patches_w = W // patch_size
        total_patches = n_patches_h * n_patches_w
        num_mask = int(total_patches * self.mask_ratio)
        # Randomly choosing patch indices to mask
        mask_indices = np.random.choice(total_patches, num_mask, replace=Fal
        mask = torch.zeros((n_patches_h, n_patches_w))
        mask.view(-1)[mask_indices] = 1.0  # 1 indicates the patch is masked

        # Creating a copy of the image and zero-out masked patches.
        masked_img = img.clone()
        for i in range(n_patches_h):
            for j in range(n_patches_w):
                if mask[i, j] == 1:
                    masked_img[:, i*patch_size:(i+1)*patch_size, j*patch_siz
        return masked_img, mask
```

In [6]:
```
class ClassificationDataset(Dataset):
  def __init__(self, data_path, classes, transform = None):
    self.samples = []
    for i, cls in enumerate(classes):
      for path in data_path[cls]:
        self.samples.append((path, i))
      self.transform = transform

  def __len__(self):
    return len(self.samples)

  def __getitem__(self, idx):
    path, label = self.samples[idx]
    img = np.load(path)
    img = torch.tensor(img, dtype=torch.float32)
    if img.ndim == 2:
      img = img.unsqueeze(0)
    if self.transform:
      img = self.transform(img)
    return img, label
```

In [7]:
```
class SRDataset(Dataset):
  def __init__(self, lr_paths, hr_paths, transform = None):
    self.lr_paths = lr_paths
    self.hr_paths = hr_paths
    self.transform = transform

  def __len__(self):
    return len(self.lr_paths)

  def __getitem__(self, idx):
    lr = np.load(self.lr_paths[idx])
    hr = np.load(self.hr_paths[idx])
    lr = torch.tensor(lr, dtype=torch.float32)
    hr = torch.tensor(hr, dtype = torch.float32)
```

```
    if lr.ndim == 2:
      lr = lr.unsqueeze(0)
    if hr.ndim == 2:
      hr = hr.unsqueeze(0)
    if self.transform:
      lr = self.transform(lr)
      hr = self.transform(hr)
    return lr, hr
```

In [8]:
```
class SEBlock(nn.Module):
  def __init__(self, channel, reduction = 16):
    super(SEBlock, self).__init__()
    self.avg_pool = nn.AdaptiveAvgPool2d(1)
    self.fc = nn.Sequential(
        nn.Linear(channel, channel // reduction, bias = False),
        nn.ReLU(inplace = True),
        nn.Linear(channel // reduction, channel, bias = False),
        nn.Sigmoid()
    )

  def forward(self, x):
    b, c, _, _ = x.size()
    y = self.avg_pool(x).view(b, c)
    y = self.fc(y).view(b, c, 1, 1)
    return x * y
```

In [9]:
```
class ResidualBlock(nn.Module):
  def __init__(self, in_channels, out_channels, stride = 1, use_se = False):
    super(ResidualBlock, self).__init__()
    self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size = 3, stric
    self.bn1 = nn.BatchNorm2d(out_channels)
    self.act1 = nn.LeakyReLU(inplace = True)
    self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size = 3, stri
    self.bn2 = nn.BatchNorm2d(out_channels)
    self.act2 = nn.LeakyReLU(inplace = True)
    self.se = SEBlock(out_channels) if use_se else None

    self.downsample = None
    if stride != 1 or in_channels != out_channels:
      self.downsample = nn.Sequential(
          nn.Conv2d(in_channels, out_channels, kernel_size = 1, stride = str
          nn.BatchNorm2d(out_channels)
      )
      self.act2 = nn.LeakyReLU(inplace = True)

  def forward(self, x):
    identity = x
    out = self.conv1(x)
    out = self.bn1(out)
    out = self.act1(out)
    out = self.conv2(out)
    out = self.bn2(out)
    if self.se is not None:
      out = self.se(out)
    if self.downsample is not None:
```

```
      identity = self.downsample(x)
    out += identity
    out = self.act2(out)
    return out
```

In [10]:
```
class MAE(nn.Module):
  def __init__(self):
    super(MAE, self).__init__()
    self.encoder = nn.Sequential(
        ResidualBlock(1, 32, stride = 2, use_se = True),
        ResidualBlock(32, 64, stride = 2, use_se = True),
        ResidualBlock(64, 128, stride = 2, use_se = True),
        ResidualBlock(128, 256, stride = 2, use_se = True)
    )


    self.dc1_up = nn.ConvTranspose2d(256, 128, kernel_size = 3, stride = 2,
    self.dc1 = ResidualBlock(128 + 128, 128, stride = 1,use_se = True)
    self.dc2_up = nn.ConvTranspose2d(128, 64, kernel_size = 3, stride = 2, p
    self.dc2 = ResidualBlock(64 + 64, 64, stride=1, use_se=True)
    self.dc3_up = nn.ConvTranspose2d(64, 32, kernel_size = 3, stride = 2, pa
    self.dc3 = ResidualBlock(32 + 32, 32, stride = 1, use_se = True)
    self.dc4_up = nn.ConvTranspose2d(32, 16, kernel_size = 3, stride = 2, pa
    self.dc4_conv = nn.Sequential(
        nn.Conv2d(16, 1, kernel_size = 3, stride = 1, padding = 1),
        nn.Sigmoid()
    )

  def forward(self, x):
    e1 = self.encoder[0](x)
    e2 = self.encoder[1](e1)
    e3 = self.encoder[2](e2)
    e4 = self.encoder[3](e3)

    d1_up = self.dc1_up(e4)
    d1_concat = torch.cat([d1_up, e3], dim = 1)
    d1 = self.dc1(d1_concat)

    d2_up = self.dc2_up(d1)
    d2_concat = torch.cat([d2_up, e2], dim = 1)
    d2 = self.dc2(d2_concat)

    d3_up = self.dc3_up(d2)
    d3_concat = torch.cat([d3_up, e1], dim = 1)
    d3 = self.dc3(d3_concat)

    d4_up = self.dc4_up(d3)
    out = self.dc4_conv(d4_up)
    return out, e4
```

In [11]:
```
class Classifier(nn.Module):
  def __init__(self, encoder, num_classes = 2, latent_dim = 256):
    super(Classifier, self).__init__()
    self.encoder = encoder
    self.pool = nn.AdaptiveAvgPool2d((1, 1))
```

```
    self.fc = nn.Linear(latent_dim, 2)

  def forward(self, x):
    latent = self.encoder(x)
    x = self.pool(latent)
    x = torch.flatten(x, 1)
    logits = self.fc(x)
    return logits
```

In [12]:
```
class CustomEncoder(nn.Module):
    def __init__(self, pretrained_mae):
        super(CustomEncoder, self).__init__()
        self.encoder = nn.Sequential(
            pretrained_mae.encoder[0],
            pretrained_mae.encoder[1]
        )
    def forward(self, x):
        return self.encoder(x)
```

In [ ]:
```
class SRDecoderModified(nn.Module):
    def __init__(self, in_channels=64, target_size=(150, 150)):
        super(SRDecoderModified, self).__init__()
        self.target_size = target_size
        self.up1 = nn.ConvTranspose2d(in_channels, 32, kernel_size=4, stride
        self.conv1 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
        self.act = nn.LeakyReLU(inplace=True)
        self.up2 = nn.ConvTranspose2d(32, 16, kernel_size=4, stride=2, paddi
        self.conv2 = nn.Conv2d(16, 16, kernel_size=3, padding=1)
        self.up3 = nn.ConvTranspose2d(16, 8, kernel_size=4, stride=2, paddir
        self.conv3 = nn.Conv2d(8, 1, kernel_size=3, padding=1)
        self.final_act = nn.Sigmoid()

    def forward(self, x):
        x = self.up1(x)
        x = self.conv1(x)
        x = self.act(x)
        x = self.up2(x)
        x = self.conv2(x)
        x = self.act(x)
        x = self.up3(x)
        x = self.conv3(x)
        if x.shape[2:] != self.target_size:
            diff_h = x.shape[2] - self.target_size[0]
            diff_w = x.shape[3] - self.target_size[1]
            x = x[:, :, diff_h//2:diff_h//2+self.target_size[0], diff_w//2:d
        x = self.final_act(x)
        return x
```

In [14]:
```
class SRModel(nn.Module):
    def __init__(self, pretrained_encoder, target_size=(150, 150)):
        super(SRModel, self).__init__()
        self.encoder = pretrained_encoder
        self.decoder = SRDecoderModified(in_channels=64, target_size=target_

    def forward(self, x):
```

```
        latent = self.encoder(x)
        out = self.decoder(latent)
        return out
```

In [15]:
```
def train_mae(model, dataloader, optimizer, scheduler, device, scaler, epoch
  model.train()
  criterion = nn.MSELoss()
  for epoch in range(epochs):
    epoch_loss = 0.0
    pbar = tqdm(dataloader, desc = f"[MAE Pre-training] Epoch {epoch+1}/{epo
    for masked_img, img, _ in pbar:
      masked_img = masked_img.to(device)
      img = img.to(device)
      optimizer.zero_grad()
      with autocast():
        recon, _ = model(masked_img)
        loss = criterion(recon, img)
      scaler.scale(loss).backward()
      scaler.step(optimizer)
      scaler.update()
      epoch_loss += loss.item() * masked_img.size(0)
      pbar.set_postfix(loss=f"{loss.item():.4f}")
    scheduler.step()
    print(f"[MAE Pre-training] Epoch [{epoch+1}/{epochs}] Loss: {epoch_loss/
```

In [16]:
```
def train_classifier(model, dataloader, optimizer, scheduler, device, scaler
  model.train()
  criterion = nn.CrossEntropyLoss()
  for epoch in range(epochs):
    epoch_loss = 0.0
    pbar = tqdm(dataloader, desc=f"[Classifier Fine-tuning] Epoch {epoch+1}/
    for imgs, labels in pbar:
      imgs = imgs.to(device)
      labels = labels.to(device)
      optimizer.zero_grad()
      with autocast():
        logits = model(imgs)
        loss = criterion(logits, labels)
      scaler.scale(loss).backward()
      scaler.step(optimizer)
      scaler.update()
      epoch_loss += loss.item() * imgs.size(0)
      pbar.set_postfix(loss=f"{loss.item():.4f}")
    scheduler.step()
    print(f"[Classifier Fine-tuning] Epoch [{epoch+1}/{epochs}] Loss: {epoch
```

In [17]:
```
def train_sr(model, dataloader, optimizer, device, scaler, epochs=10):
    model.train()
    criterion = nn.MSELoss()
    for epoch in range(epochs):
        epoch_loss = 0.0
        pbar = tqdm(dataloader, desc=f"SR Training Epoch {epoch+1}/{epochs}"
        for lr, hr in pbar:
            lr = lr.to(device)
            hr = hr.to(device)
```

```
            optimizer.zero_grad()
            with torch.amp.autocast("cuda"):
                sr = model(lr)
                loss = criterion(sr, hr)
            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()
            epoch_loss += loss.item() * lr.size(0)
            pbar.set_postfix(loss=f"{loss.item():.4f}")
        print(f"Epoch {epoch+1}/{epochs}, SR Loss: {epoch_loss/len(dataloade
```

In [18]:

```
def evaluate_and_plot_roc(model, dataloader, device, num_classes=2):
    model.eval()
    all_labels = []
    all_probs = []

    with torch.no_grad():
        for imgs, labels in tqdm(dataloader, desc="Evaluating", leave=False)
            imgs = imgs.to(device)
            logits = model(imgs)
            if num_classes > 1:
                probs = torch.softmax(logits, dim=1)
            else:
                probs = torch.sigmoid(logits)
            all_probs.append(probs.cpu().numpy())
            all_labels.append(labels.cpu().numpy())

    # Concatenate lists into single arrays.
    all_probs = np.concatenate(all_probs, axis=0)
    all_labels = np.concatenate(all_labels, axis=0)

    # Ensure all_probs is 2D. If 1D, reshape to [n_samples, 1].
    if all_probs.ndim == 1:
        all_probs = all_probs[:, np.newaxis]
    if num_classes == 2 and all_probs.shape[1] == 1:
        all_probs = np.hstack([1 - all_probs, all_probs])
    if num_classes == 2:
        y_true = np.zeros((all_labels.shape[0], num_classes))
        y_true[np.arange(all_labels.shape[0]), all_labels] = 1
    else:
        y_true = label_binarize(all_labels, classes=list(range(num_classes))

    # Compute ROC curve and AUC for each class.
    fpr = {}
    tpr = {}
    roc_auc = {}
    for i in range(num_classes):
        fpr[i], tpr[i], _ = roc_curve(y_true[:, i], all_probs[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot ROC curves for all classes.
    plt.figure(figsize=(8, 6))
    for i in range(num_classes):
        plt.plot(fpr[i], tpr[i], lw=2, label=f"Class {i} (AUC = {roc_auc[i]:
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.label("False Positive Rate")
```

```
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.legend(loc="lower right")
    plt.show()

    return all_labels, all_probs, roc_auc
```

In [19]:
```
def evaluate_sr(model, dataloader, device):
    model.eval()
    mse_loss = 0.0
    psnr_total = 0.0
    ssim_total = 0.0
    criterion = nn.MSELoss(reduction='mean')
    n = 0
    with torch.no_grad():
        for lr, hr in tqdm(dataloader, desc="Evaluating SR", leave=False):
            lr = lr.to(device)
            hr = hr.to(device)
            sr = model(lr)
            mse = criterion(sr, hr).item()
            mse_loss += mse * lr.size(0)
            # Compute PSNR (assuming pixel values in [0,1])
            psnr_val = 10 * math.log10(1.0 / mse) if mse > 0 else 100
            psnr_total += psnr_val * lr.size(0)
            # Compute SSIM for each image (convert tensors to numpy arrays)
            sr_np = sr.cpu().numpy()
            hr_np = hr.cpu().numpy()
            for i in range(sr_np.shape[0]):
                # Assuming single channel images
                ssim_val = ssim(hr_np[i, 0], sr_np[i, 0], data_range=hr_np[i
                ssim_total += ssim_val
                n += 1
    mse_avg = mse_loss / len(dataloader.dataset)
    psnr_avg = psnr_total / len(dataloader.dataset)
    ssim_avg = ssim_total / n
    print(f"SR Evaluation - MSE: {mse_avg:.4f}, PSNR: {psnr_avg:.2f}, SSIM:
```

In [20]:
```
base_path_for_6A = "/content/dataset/Dataset"
classes = ['no_sub', 'cdm']
data_paths = {
    cls: [os.path.join(base_path_for_6A, cls, f)
    for f in os.listdir(os.path.join(base_path_for_6A, cls)) if f.endswith('
    for cls in classes
}
```

In [21]:
```
batch_size = 32
pretrain_epochs = 10
finetune_epochs = 10
lr = 1e-3
transform = None
```

In [22]:
```
no_sub_paths = data_paths['no_sub']
pretrain_dataset = MAEDataset(no_sub_paths, mask_ratio=0.75, patch_size=8, t
pretrain_loader = DataLoader(pretrain_dataset, batch_size=batch_size, shuffl
```

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
UserWarning: This DataLoader will create 4 worker processes in total. Our su
ggested max number of worker in current system is 2, which is smaller than w
hat this DataLoader is going to create. Please be aware that excessive worke
r creation might get DataLoader running slow or even freeze, lower the worke
r number to avoid potential slowness/freeze if necessary.
  warnings.warn(
```

In [23]:
```
modelMAE = MAE().to(device)
optimizer_mae = optim.Adam(modelMAE.parameters(), lr=lr)
scheduler_mae = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer_mae, T_
scaler = GradScaler()
```

```
<ipython-input-23-6e79cb26ed3e>:4: FutureWarning: `torch.cuda.amp.GradScaler
(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)`
instead.
  scaler = GradScaler()
```

In [24]:
```
print("Starting MAE pre-training...")
train_mae(modelMAE, pretrain_loader, optimizer_mae, scheduler_mae, device, s
```

```
Starting MAE pre-training...
[MAE Pre-training] Epoch 1/10:   0%|          | 0/921 [00:00<?, ?it/s]<ipyth
on-input-15-e6c8494355bf>:11: FutureWarning: `torch.cuda.amp.autocast(arg
s...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` inste
ad.
  with autocast():
[MAE Pre-training] Epoch [1/10] Loss: 0.0619
[MAE Pre-training] Epoch [2/10] Loss: 0.0577
[MAE Pre-training] Epoch [3/10] Loss: 0.0576
[MAE Pre-training] Epoch [4/10] Loss: 0.0575
[MAE Pre-training] Epoch [5/10] Loss: 0.0575
[MAE Pre-training] Epoch [6/10] Loss: 0.0574
[MAE Pre-training] Epoch [7/10] Loss: 0.0574
[MAE Pre-training] Epoch [8/10] Loss: 0.0574
[MAE Pre-training] Epoch [9/10] Loss: 0.0574

[MAE Pre-training] Epoch [10/10] Loss: 0.0574
```

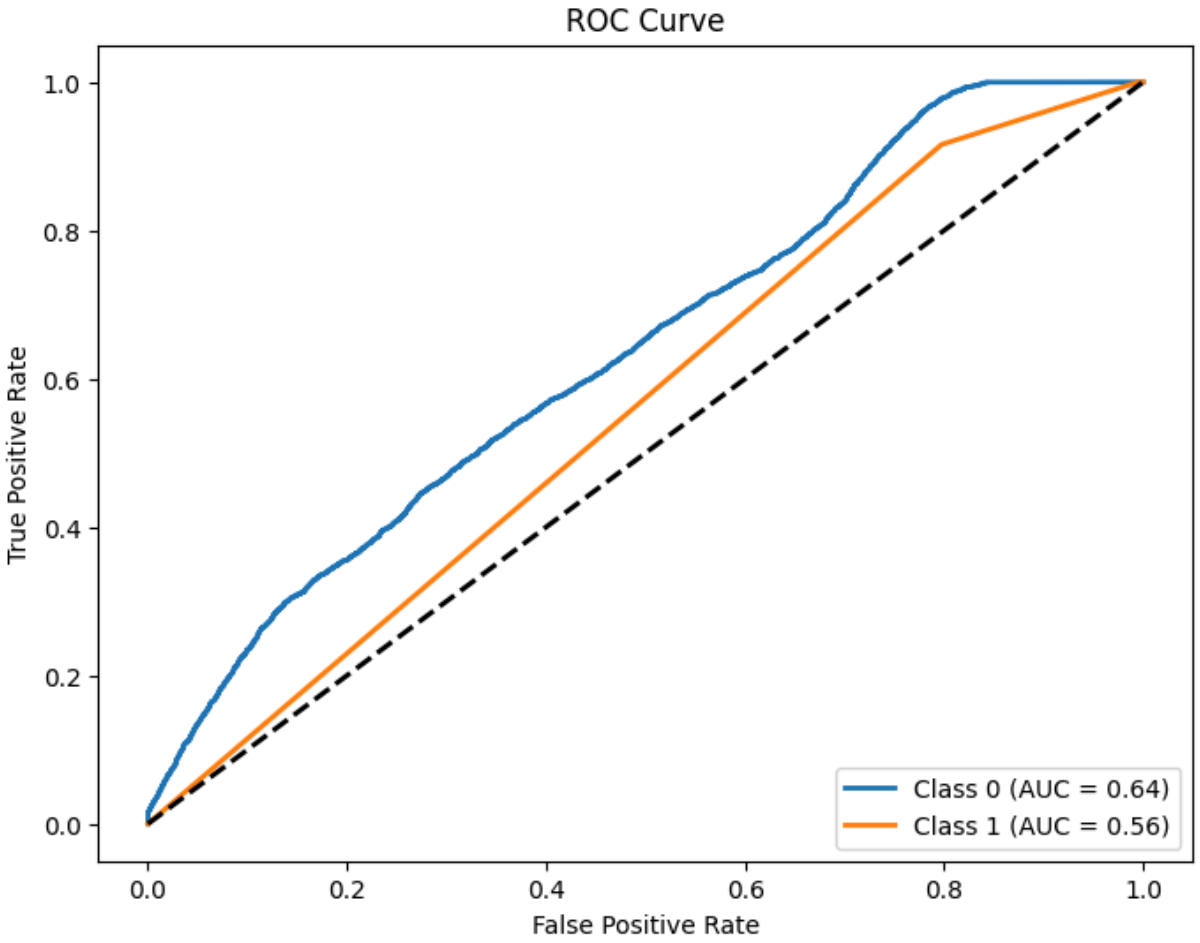
In [25]:
```
classifier_model = Classifier(encoder=modelMAE.encoder, num_classes=len(clas
optimizer_cls = optim.Adam(classifier_model.parameters(), lr=lr)
scheduler_cls = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer_cls, T_
```

In [26]:
```
class_dataset = ClassificationDataset(data_paths, classes, transform=transfo
train_size = int(0.8 * len(class_dataset))
val_size = len(class_dataset) - train_size
train_dataset, val_dataset = random_split(class_dataset, [train_size, val_si
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, r
```

In [27]:
```
train_classifier(classifier_model, train_loader, optimizer_cls, scheduler_cl
```

```
[Classifier Fine-tuning] Epoch 1/10:   0%|          | 0/1481 [00:00<?, ?it/
s]<ipython-input-16-d471aa5bcf1d>:11: FutureWarning: `torch.cuda.amp.autocas
t(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)`
instead.
  with autocast():
[Classifier Fine-tuning] Epoch [1/10] Loss: 0.0986
[Classifier Fine-tuning] Epoch [2/10] Loss: 0.0096
[Classifier Fine-tuning] Epoch [3/10] Loss: 0.0063
[Classifier Fine-tuning] Epoch [4/10] Loss: 0.0043
[Classifier Fine-tuning] Epoch [5/10] Loss: 0.0027
[Classifier Fine-tuning] Epoch [6/10] Loss: 0.0021
[Classifier Fine-tuning] Epoch [7/10] Loss: 0.0013
[Classifier Fine-tuning] Epoch [8/10] Loss: 0.0006
[Classifier Fine-tuning] Epoch [9/10] Loss: 0.0001

[Classifier Fine-tuning] Epoch [10/10] Loss: 0.0000
```

In [28]:
```
all_labels, all_probs, roc_auc = evaluate_and_plot_roc(classifier_model, val
```



In [29]:
```
torch.save(modelMAE.state_dict(), 'pretrained_mae_weights.pth')
```

In [30]:
```
base_path = "/content/datasetResolution/Dataset"
os.path.join(base_path, "LR")
```

```
hr_dir = os.path.join(base_path, "HR")
lr_paths = [os.path.join(lr_dir, f) for f in os.listdir(lr_dir) if f.endswit
hr_paths = [os.path.join(hr_dir, f) for f in os.listdir(hr_dir) if f.endswit
lr_paths.sort()
hr_paths.sort()
```

In [31]:
```
sr_dataset = SRDataset(lr_paths, hr_paths, transform=None)
```

In [32]:
```
train_size = int(0.8 * len(sr_dataset))
val_size = len(sr_dataset) - train_size
train_dataset, val_dataset = random_split(sr_dataset, [train_size, val_size]
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, num_wo
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False, num_worke
```

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
UserWarning: This DataLoader will create 4 worker processes in total. Our su
ggested max number of worker in current system is 2, which is smaller than w
hat this DataLoader is going to create. Please be aware that excessive worke
r creation might get DataLoader running slow or even freeze, lower the worke
r number to avoid potential slowness/freeze if necessary.
  warnings.warn(
```

In [35]:
```
pretrained_MAE = modelMAE
pretrained_MAE = pretrained_MAE.to(device)
pretrained_MAE.load_state_dict(torch.load('pretrained_mae_weights.pth'))
custom_encoder = CustomEncoder(pretrained_MAE)
sr_model = SRModel(pretrained_encoder=custom_encoder, target_size=(150,150))
```

In [36]:
```
optimizer = optim.Adam(sr_model.parameters(), lr=1e-4)
scaler = torch.amp.GradScaler('cuda')
print("Starting Super-Resolution Fine-Tuning...")
train_sr(sr_model, train_loader, optimizer, device, scaler, epochs=10)
```

```
Starting Super-Resolution Fine-Tuning...
SR Training Epoch 1/10:   0%|          | 0/500 [00:00<?, ?it/s]/usr/local/li
b/python3.11/dist-packages/torch/utils/data/dataloader.py:624: UserWarning:
This DataLoader will create 4 worker processes in total. Our suggested max n
umber of worker in current system is 2, which is smaller than what this Data
Loader is going to create. Please be aware that excessive worker creation mi
ght get DataLoader running slow or even freeze, lower the worker number to a
void potential slowness/freeze if necessary.
  warnings.warn(
Epoch 1/10, SR Loss: 0.0290
Epoch 2/10, SR Loss: 0.0005
Epoch 3/10, SR Loss: 0.0003
Epoch 4/10, SR Loss: 0.0003
Epoch 5/10, SR Loss: 0.0002
Epoch 6/10, SR Loss: 0.0002
Epoch 7/10, SR Loss: 0.0002
Epoch 8/10, SR Loss: 0.0002
Epoch 9/10, SR Loss: 0.0002

Epoch 10/10, SR Loss: 0.0001
```

In [37]:
```
evaluate_sr(sr_model, val_loader, device)
```

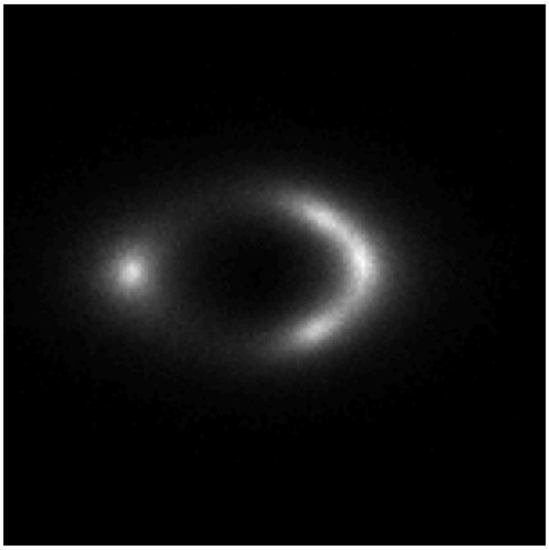
```
SR Evaluation - MSE: 0.0001, PSNR: 38.28, SSIM: 0.9439
```

In [40]:
```
def compare_hr_sr(model, dataloader, device):
  model.eval()
  with torch.no_grad():
    for lr, hr in dataloader:
      lr = lr.to(device)
      hr = hr.to(device)
      sr = model(lr)

      # Iterate through each image in the batch
      for i in range(hr.shape[0]):  # hr.shape[0] is the batch size
          hr_np = hr[i].cpu().numpy().squeeze(0)  # Select image i from the
          sr_np = sr[i].cpu().numpy().squeeze(0)  # Select image i from the

          ssim_score = ssim(hr_np, sr_np, data_range=hr_np.max()-hr_np.min()

          fig, axes = plt.subplots(1, 2, figsize = (10, 5))
          axes[0].imshow(hr_np, cmap='gray')
          axes[0].set_title('Original HR')
          axes[0].axis('off')
          axes[1].imshow(sr_np, cmap='gray')
          axes[1].set_title(f'Super-Resolved (SSIM: {ssim_score:.4f})')
          axes[1].axis('off')
          plt.show()
      break
```

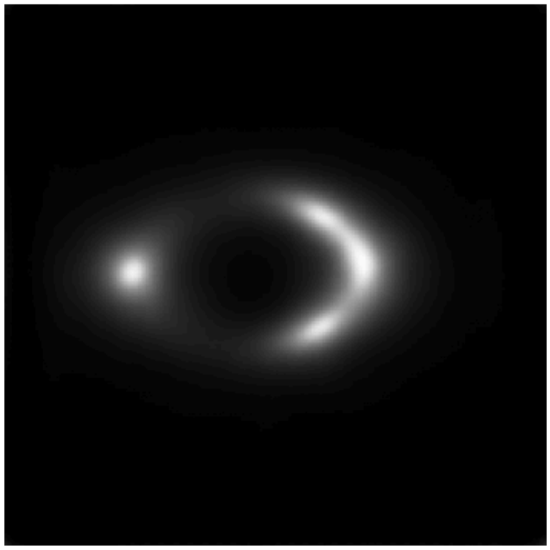
In [41]:
```
compare_hr_sr(sr_model, val_loader, device)
```

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
UserWarning: This DataLoader will create 4 worker processes in total. Our su
ggested max number of worker in current system is 2, which is smaller than w
hat this DataLoader is going to create. Please be aware that excessive worke
r creation might get DataLoader running slow or even freeze, lower the worke
r number to avoid potential slowness/freeze if necessary.
  warnings.warn(
```

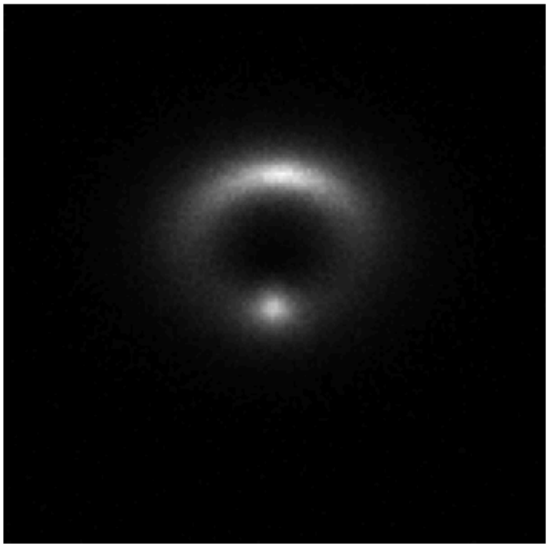
Original HR



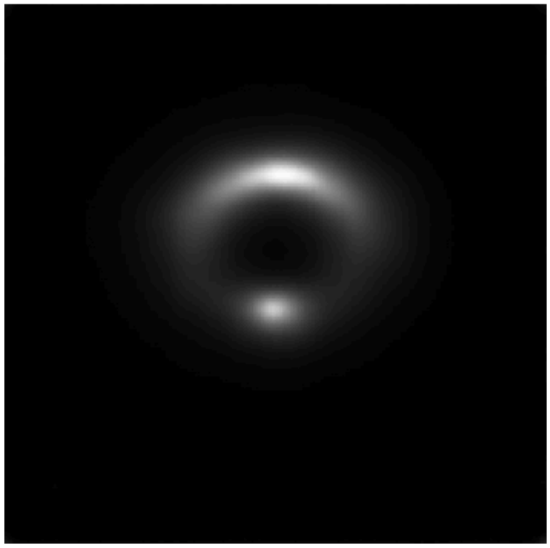
Super-Resolved (SSIM: 0.9477)



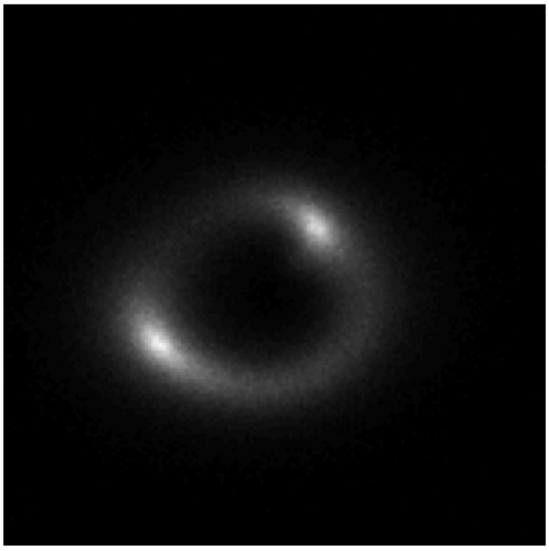
Original HR



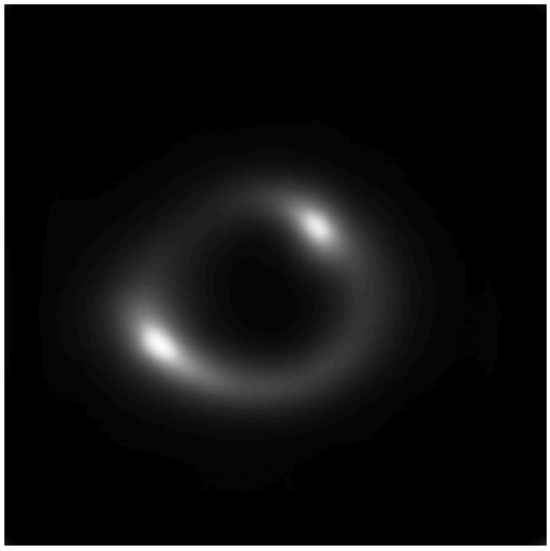
Super-Resolved (SSIM: 0.9337)

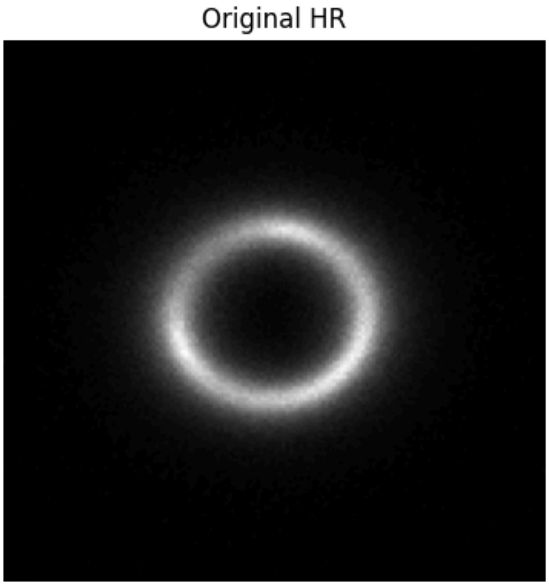


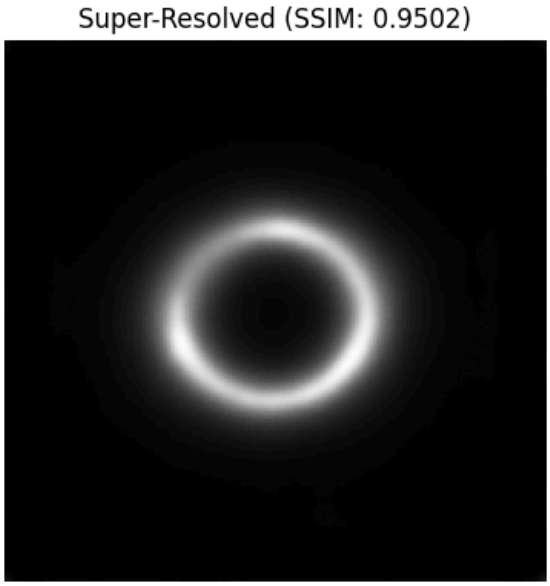
Original HR

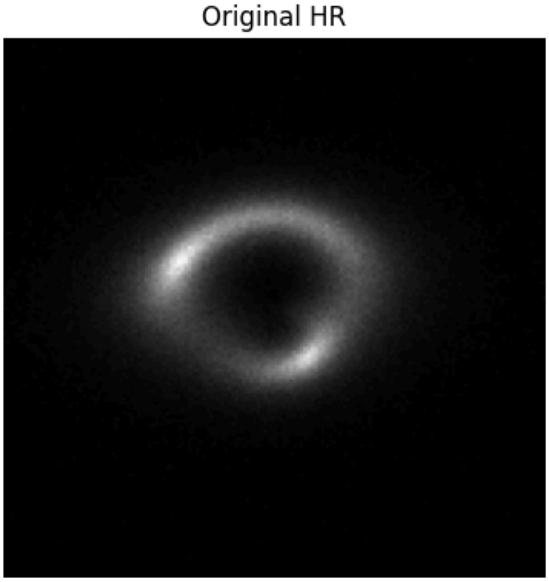


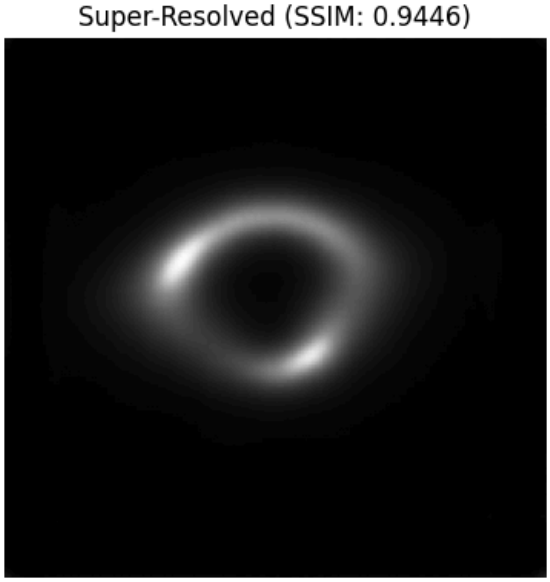
Super-Resolved (SSIM: 0.9429)

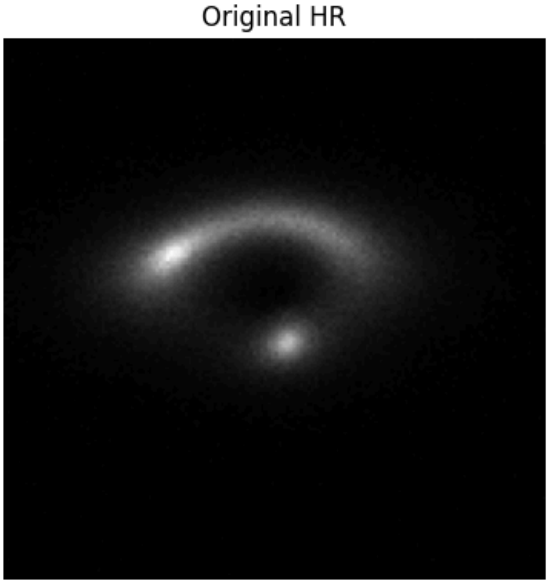


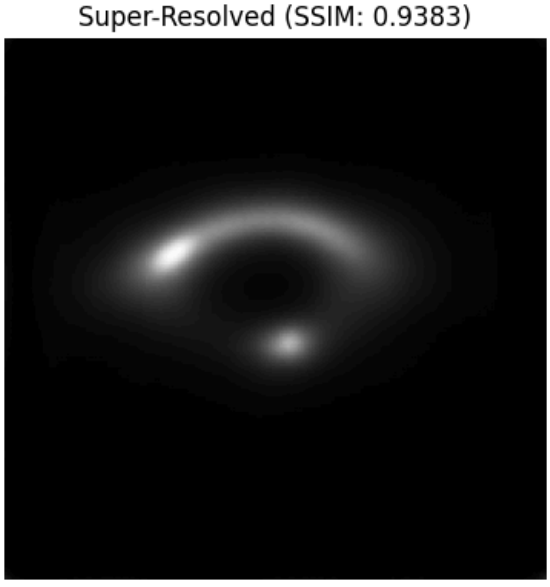




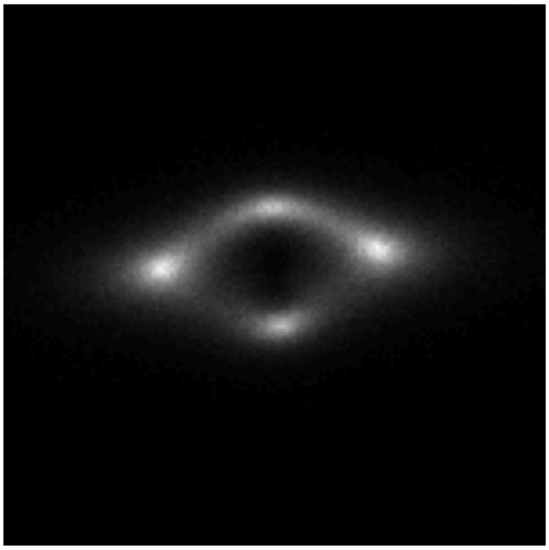




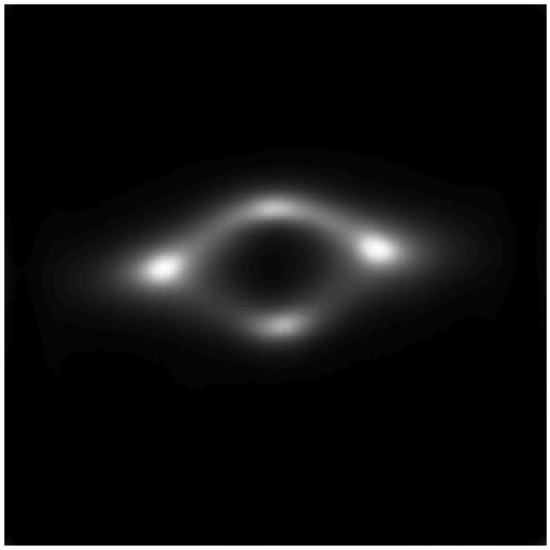




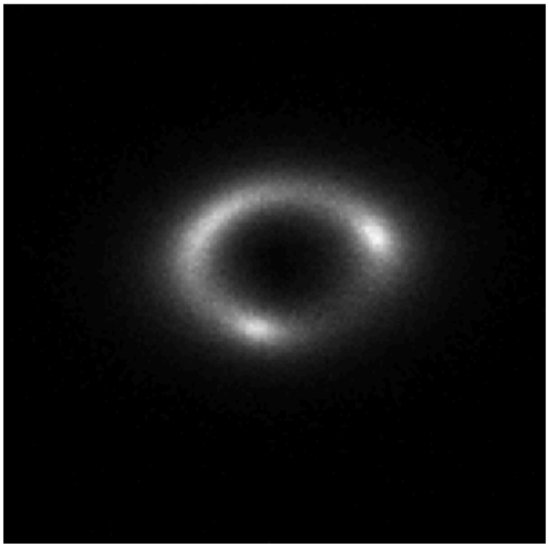
Original HR



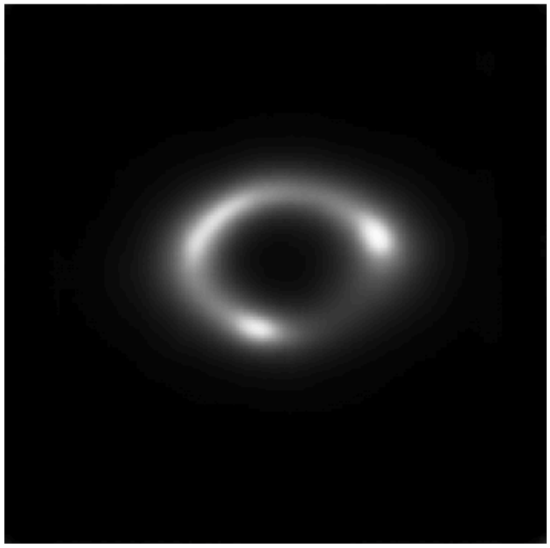
Super-Resolved (SSIM: 0.9441)



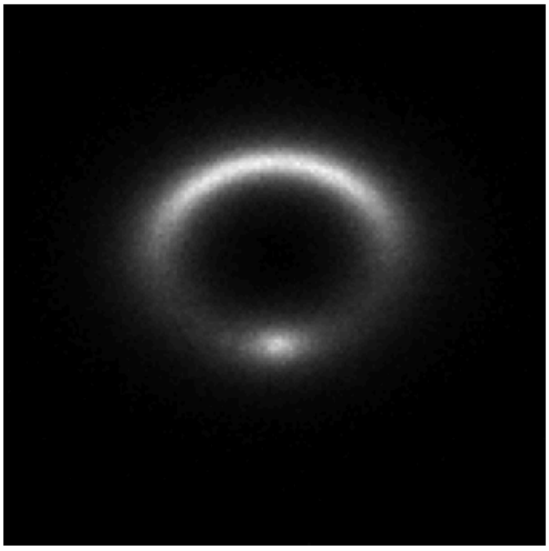
Original HR



Super-Resolved (SSIM: 0.9438)



Original HR



Super-Resolved (SSIM: 0.9455)