def regroup\_reds\_dataset(train\_path, val\_path):  
 val\_folders = glob.glob(os.path.join(val\_path, '\*'))  
 for folder in val\_folders:  
 new\_folder\_idx = int(folder.split('\\')[-1]) + 240  
 print(new\_folder\_idx)  
 os.system(f'cp -r {folder}   
 {os.path.join(train\_path, str(new\_folder\_idx))}')

获得帧序列：

clip\_name, frame\_name = key.split('/')  
# ensure not exceeding the borders  
start\_frame\_idx = center\_frame\_idx - self.num\_half\_frames  
end\_frame\_idx = center\_frame\_idx + self.num\_half\_frames  
# each clip has 100 frames starting from 0 to 99  
while (start\_frame\_idx < 0) or (end\_frame\_idx > 99):  
 center\_frame\_idx = random.randint(0, 99)  
 start\_frame\_idx = (  
 center\_frame\_idx - self.num\_half\_frames)  
 end\_frame\_idx = center\_frame\_idx + self.num\_half\_frames  
 frame\_name = f'{center\_frame\_idx:08d}'  
 neighbor\_list = list(  
 range(center\_frame\_idx - self.num\_half\_frames,  
 center\_frame\_idx + self.num\_half\_frames + 1))  
 # random reverse  
 if self.random\_reverse and random.random() < 0.5:  
 neighbor\_list.reverse()  
#get GT frame  
img\_gt\_path = self.gt\_root / clip\_name / f'{frame\_name}.png'  
img\_bytes = self.file\_client.get(img\_gt\_path, 'gt')  
img\_gt = imfrombytes(img\_bytes, float32=True)  
#get neighboring LQ frames  
img\_lqs = []  
for neighbor in neighbor\_list:  
 img\_lq\_path = self.lq\_root / clip\_name / f'{neighbor:08d}.png'  
img\_bytes = self.file\_client.get(img\_lq\_path, 'lq')  
img\_lq = imfrombytes(img\_bytes, float32=True)  
img\_lqs.append(img\_lq)

裁剪LQ帧和GT帧：

h\_lq, w\_lq, \_ = img\_lqs[0].shape  
h\_gt, w\_gt, \_ = img\_gts[0].shape  
lq\_patch\_size = gt\_patch\_size // scale  
# randomly choose top and left coordinates for lq patch  
top = random.randint(0, h\_lq - lq\_patch\_size)  
left = random.randint(0, w\_lq - lq\_patch\_size)  
# crop lq patch  
img\_lqs = [  
 v[top:top + lq\_patch\_size, left:left + lq\_patch\_size, ...]  
 for v in img\_lqs]  
# crop corresponding gt patch  
top\_gt, left\_gt = int(top \* scale), int(left \* scale)  
img\_gts = [  
 v[top\_gt:top\_gt + gt\_patch\_size, left\_gt:left\_gt + gt\_patch\_size, ...]  
 for v in img\_gts]  
return img\_gts, img\_lqs

进行随机翻转和旋转：

def augment(imgs):  
 hflip = random.random() < 0.5  
 vflip = random.random() < 0.5  
 rot90 = random.random() < 0.5  
  
 def \_augment(img):  
 if hflip: # horizontal  
 cv2.flip(img, 1, img)  
 if vflip: # vertical  
 cv2.flip(img, 0, img)  
 if rot90:  
 img = img.transpose(1, 0, 2)  
 return img  
  
 imgs = [\_augment(img) for img in imgs]  
 return imgs

残差块：

class ResidualBlockNoBN(nn.Module):  
 def \_\_init\_\_(self, num\_feat=64, res\_scale=1):  
 super(ResidualBlockNoBN, self).\_\_init\_\_()  
 self.res\_scale = res\_scale  
 self.conv1 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1, bias=True)  
 self.conv2 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1, bias=True)  
 self.relu = nn.ReLU(inplace=True)  
  
 def forward(self, x):  
 identity = x  
 out = self.conv2(self.relu(self.conv1(x)))  
 return identity + out \* self.res\_scale

特征提取模块

self.feature\_extraction = make\_layer(ResidualBlockNoBN, num\_extract\_block, num\_feat=num\_feat)  
self.conv\_l2\_1 = nn.Conv2d(num\_feat, num\_feat, 3, 2, 1)  
self.conv\_l2\_2 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)  
self.conv\_l3\_1 = nn.Conv2d(num\_feat, num\_feat, 3, 2, 1)  
self.conv\_l3\_2 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)

对齐模块网络组成

self.offset\_conv1 = nn.ModuleDict()  
self.offset\_conv2 = nn.ModuleDict()  
self.offset\_conv3 = nn.ModuleDict()  
self.dcn\_pack = nn.ModuleDict()  
self.feat\_conv = nn.ModuleDict()  
  
for i in range(3, 0, -1):  
 level = f'l{i}'  
 self.offset\_conv1[level] = nn.Conv2d(num\_feat \* 2, num\_feat, 3, 1, 1)  
 if i == 3:  
 self.offset\_conv2[level] = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)  
 else:  
 self.offset\_conv2[level] = nn.Conv2d(num\_feat \* 2, num\_feat, 3, 1, 1)  
 self.offset\_conv3[level] = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)  
 self.dcn\_pack[level] = DCNv2Pack(num\_feat,num\_feat,3,padding=1,  
 deformable\_groups=deformable\_groups)  
 if i < 3:  
 self.feat\_conv[level] = nn.Conv2d(num\_feat \* 2, num\_feat, 3, 1, 1)  
self.cas\_offset\_conv1 = nn.Conv2d(num\_feat \* 2, num\_feat, 3, 1, 1)  
self.cas\_offset\_conv2 = nn.Conv2d(num\_feat, num\_feat, 3, 1, 1)  
self.cas\_dcnpack = DCNv2Pack(num\_feat,num\_feat,3,padding=1,  
 deformable\_groups=deformable\_groups)  
self.upsample = nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=False)  
self.lrelu = nn.LeakyReLU(negative\_slope=0.1, inplace=True)

融合模块正向传播

feat = self.lrelu(self.feat\_fusion(aligned\_feat))
  
  
attn = self.lrelu(self.spatial\_attn1(aligned\_feat))
  
attn\_max = self.max\_pool(attn)
  
attn\_avg = self.avg\_pool(attn)
  
attn = self.lrelu(
  
 self.spatial\_attn2(torch.cat([attn\_max, attn\_avg], dim=1)))
  
# levels
  
attn\_level = self.lrelu(self.spatial\_attn\_l1(attn))
  
attn\_max = self.max\_pool(attn\_level)
  
attn\_avg = self.avg\_pool(attn\_level)
  
attn\_level = self.lrelu(
  
 self.spatial\_attn\_l2(torch.cat([attn\_max, attn\_avg], dim=1)))
  
attn\_level = self.lrelu(self.spatial\_attn\_l3(attn\_level))
  
attn\_level = self.upsample(attn\_level)
  
  
attn = self.lrelu(self.spatial\_attn3(attn)) + attn\_level
  
attn = self.lrelu(self.spatial\_attn4(attn))
  
attn = self.upsample(attn)
  
attn = self.spatial\_attn5(attn)
  
attn\_add = self.spatial\_attn\_add2(
  
 self.lrelu(self.spatial\_attn\_add1(attn)))
  
attn = torch.sigmoid(attn)
  
  
# \* 2 makes (attn \* 2) to be close to 1.
  
feat = feat \* attn \* 2 + attn\_add
  
return feat