In [1]:
```
import os
import sys
from tqdm import tqdm
import numpy as np
import torch
import torch.nn as nn
from torch.optim.lr_scheduler import ReduceLROnPlateau
from torch import optim
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
import torchvision.utils as vutils
import torchvision.transforms as T
import torchvision.transforms.functional as TF
import torch.nn.functional as F
from torchvision.utils import save_image
from PIL import Image
import matplotlib.pyplot as plt
%matplotlib inline
from paddleocr import PaddleOCR, draw_ocr
import paddle
from Levenshtein import distance as levenshtein_distance
from pytorch_msssim import ssim as ssim_fn
from lpips import LPIPS
from torchmetrics.image.psnr import PeakSignalNoiseRatio
from torchmetrics.image.ssim import StructuralSimilarityIndexMeasure
from torchmetrics.image.fid import FrechetInceptionDistance
```

In [2]:
```
sys.path.append(os.path.abspath('../../../src/dataset'))
from paired_image_dataset import PairedImageDataset

DATA_DIR = os.path.join('..', '..', '..', 'data')
print(os.listdir(DATA_DIR))
```

```
['h_blur', 'low_light', 'low_qual', 'test', 'v_blur']
```

In [3]:
```
image_width = 256
image_height = 128
batch_size = 16
stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5)
```

In [5]:
```
train_transform = T.Compose([
    T.Resize((image_height, image_width)),
    T.ToTensor(),
    T.Normalize(*stats)])

valid_transform = T.Compose([
    T.Resize((image_height, image_width)),
    T.ToTensor(),
    T.Normalize(*stats)
])
```

In [6]:
```
train_ds = PairedImageDataset(blur_dir=os.path.join(DATA_DIR, 'low_light', 'train',
                              normal_dir=os.path.join(DATA_DIR, 'low_light', 'train
```

```
                                transform=train_transform)

valid_ds = PairedImageDataset(blur_dir=os.path.join(DATA_DIR, 'low_light', 'valid',
                                normal_dir=os.path.join(DATA_DIR, 'low_light', 'valid
                                transform=valid_transform)

test_ds = ImageFolder(os.path.join(DATA_DIR, 'low_light', 'test v2'), transform=val
```

In [7]:
```
train_dl = DataLoader(train_ds, batch_size=batch_size, shuffle=True, num_workers=4)
valid_dl = DataLoader(valid_ds, batch_size=batch_size*2, shuffle=True, num_workers=
test_dl = DataLoader(test_ds, batch_size=batch_size//2, shuffle=False, num_workers=
```

# Helper functions

In [8]:
```
def denorm(img_tensors):
    return img_tensors * stats[1][0] + stats[0][0]

def show_images(input_images, target_images, nmax=16):
    input_images = denorm(input_images.detach()[:nmax])
    target_images = denorm(target_images.detach()[:nmax])

    # Combine into a single tensor for visualization
    combined = torch.cat((input_images, target_images), 0)
    grid = make_grid(combined, nrow=nmax)

    plt.figure(figsize=(nmax, 4))
    plt.imshow(grid.permute(1, 2, 0))
    plt.axis("off")
    plt.title("Top: Horizontal Blur / Blurred | Bottom: Normal / Target")
    plt.show()

def show_batch(dl, nmax=16):
    for input_batch, target_batch in dl:
        show_images(input_batch, target_batch, nmax)
        break
```

In [9]:
```
def show_paired_samples(dataset, num_samples=4):
    fig, axes = plt.subplots(num_samples, 2, figsize=(5, 2 * num_samples))

    for i in range(num_samples):
        input_img, target_img = dataset[i]

        input_img = denorm(input_img)
        target_img = denorm(target_img)

        if isinstance(input_img, torch.Tensor):
            input_img = input_img.permute(1, 2, 0).numpy()
            target_img = target_img.permute(1, 2, 0).numpy()

        axes[i, 0].imshow(input_img)
        axes[i, 0].set_title("Horizontal Blur (Input)")
        axes[i, 0].axis("off")
```

```
        axes[i, 1].imshow(target_img)
        axes[i, 1].set_title("Normal (Target)")
        axes[i, 1].axis("off")

    plt.tight_layout()
    plt.show()

def show_generated_images(images, nrow=4):
    images = denorm(images.detach().cpu())

    # Make grid
    grid_img = vutils.make_grid(images, nrow=nrow, normalize=False)

    # Show image
    plt.figure(figsize=(nrow * 2, 2 * (len(images) // nrow)))
    plt.axis('off')
    plt.imshow(grid_img.permute(1, 2, 0))
    plt.show()
```

In [10]:
```
show_paired_samples(train_ds)
```


In [11]:
```
show_batch(valid_dl, nmax=14)
```


In [12]:
```
def print_images(image_tensor, num_images):
    images = denorm(image_tensor)
    images = images.detach().cpu()
    image_grid = make_grid(images[:num_images], nrow=5)
```

```
    plt.imshow(image_grid.permute(1, 2, 0).squeeze())
    plt.show()
```

In [13]:
```
def save_samples(generator, test_dl, epoch, sample_dir='./generated_test_outputs'):
    generator.eval()
    os.makedirs(sample_dir, exist_ok=True)

    with torch.no_grad():
        global_idx = 1  # running index for image IDs

        for inputs, _ in test_dl:
            inputs = inputs.to(device)

            fake_images = generator(inputs)
            fake_images = fake_images.clamp(-1, 1)

            batch_size = fake_images.size(0)

            for b in range(batch_size):
                img = fake_images[b]

                # create folder per image
                folder_path = os.path.join(sample_dir, str(global_idx))
                os.makedirs(folder_path, exist_ok=True)

                file_name = f"{global_idx}_{epoch}epoch.jpg"
                save_path = os.path.join(folder_path, file_name)

                save_image(denorm(img).unsqueeze(0), save_path, nrow=1)
                global_idx += 1
```

In [14]:
```
def get_default_device():
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    if isinstance(data, (list, tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        return len(self.dl)
```

In [15]:
```
device = get_default_device()
device
```

Out[15]:
```
device(type='cuda')
```

In [16]:
```
train_dl = DeviceDataLoader(train_dl, device)
valid_dl = DeviceDataLoader(valid_dl, device)
test_dl = DeviceDataLoader(test_dl, device)
```

# Building the Model

## Generator

In [17]:
```
class ResConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, dropout=0.0, batch
        super().__init__()
        padding = kernel_size // 2

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size, padding=padd
        self.bn1 = nn.BatchNorm2d(out_channels) if batch_norm else nn.Identity()

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size, padding=pad
        self.bn2 = nn.BatchNorm2d(out_channels) if batch_norm else nn.Identity()

        self.drop = nn.Dropout(p=dropout) if dropout > 0 else nn.Identity()

        self.shortcut = nn.Conv2d(in_channels, out_channels, kernel_size=1, padding
        self.bn_sc = nn.BatchNorm2d(out_channels) if batch_norm else nn.Identity()

    def forward(self, x):
        residual = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = F.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.drop(out)

        shortcut = self.shortcut(residual)
        shortcut = self.bn_sc(shortcut)

        out += shortcut
        out = F.relu(out)
        return out


class GatingSignal(nn.Module):
    def __init__(self, in_channels, out_channels, batch_norm=False):
        super().__init__()
```

```
        layers = [nn.Conv2d(in_channels, out_channels, kernel_size=1, padding=0)]
        if batch_norm:
            layers.append(nn.BatchNorm2d(out_channels))
        layers.append(nn.ReLU(inplace=True))
        self.gate = nn.Sequential(*layers)

    def forward(self, x):
        return self.gate(x)

class AttentionBlock(nn.Module):
    def __init__(self, in_channels, gating_channels, inter_channels):
        super().__init__()

        self.theta_x = nn.Conv2d(in_channels, inter_channels, kernel_size=2, stride
        self.phi_g = nn.Conv2d(gating_channels, inter_channels, kernel_size=1, padd

        self.upsample_g = nn.ConvTranspose2d(inter_channels, inter_channels, kernel
        self.combine = nn.Sequential(
            nn.ReLU(inplace=True),
            nn.Conv2d(inter_channels, 1, kernel_size=1),
            nn.Sigmoid())

        self.final_conv = nn.Sequential(
            nn.Conv2d(in_channels, in_channels, kernel_size=1),
            nn.BatchNorm2d(in_channels))

    def forward(self, x, g):
        # x: skip connection feature map
        # g: gating signal (decoder feature map)
        theta_x = self.theta_x(x)
        phi_g = self.phi_g(g)

        # Upsample gating to match theta_x
        upsample_g = self.upsample_g(phi_g)

        if upsample_g.shape != theta_x.shape:
            upsample_g = F.interpolate(upsample_g, size=theta_x.shape[2:], mode='bi

        psi = self.combine(theta_x + upsample_g)
        psi = F.interpolate(psi, size=x.shape[2:], mode='bilinear', align_corners=T
        psi = psi.expand(-1, x.shape[1], -1, -1)

        y = x * psi
        return self.final_conv(y)


class AttentionResUNetGenerator(nn.Module):
    def __init__(self, in_channels=3, out_channels=3, base_filters=64, dropout=0.0,
        super().__init__()

        # Encoder
        self.down1 = ResConvBlock(in_channels, base_filters, dropout=dropout, batch
        self.pool1 = nn.MaxPool2d(2)
        self.down2 = ResConvBlock(base_filters, base_filters * 2, dropout=dropout,
        self.pool2 = nn.MaxPool2d(2)
        self.down3 = ResConvBlock(base_filters * 2, base_filters * 4, dropout=dropo
```

```
        self.pool3 = nn.MaxPool2d(2)
        self.down4 = ResConvBlock(base_filters * 4, base_filters * 8, dropout=dropo
        self.pool4 = nn.MaxPool2d(2)

        # Bottleneck
        self.bottleneck = ResConvBlock(base_filters * 8, base_filters * 16, dropout

        # Gating and Attention
        self.gate4 = GatingSignal(base_filters * 16, base_filters * 8, batch_norm)
        self.attn4 = AttentionBlock(base_filters * 8, base_filters * 8, base_filter

        self.gate3 = GatingSignal(base_filters * 8, base_filters * 4, batch_norm) #
        self.attn3 = AttentionBlock(base_filters * 4, base_filters * 4, base_filter

        self.gate2 = GatingSignal(base_filters * 4, base_filters * 2, batch_norm) #
        self.attn2 = AttentionBlock(base_filters * 2, base_filters * 2, base_filter

        self.gate1 = GatingSignal(base_filters * 2, base_filters, batch_norm) # 128
        self.attn1 = AttentionBlock(base_filters, base_filters, base_filters) # 64

        # Decoder
        self.up4 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.dec4 = ResConvBlock(base_filters * 16 + base_filters * 8, base_filters

        self.up3 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.dec3 = ResConvBlock(base_filters * 8 + base_filters * 4, base_filters

        self.up2 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.dec2 = ResConvBlock(base_filters * 4 + base_filters * 2, base_filters

        self.up1 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        self.dec1 = ResConvBlock(base_filters * 2 + base_filters, base_filters, dro

        # Output Layer
        self.final_conv = nn.Sequential(
            nn.Conv2d(base_filters, out_channels, kernel_size=1),
            nn.Tanh())

    def forward(self, x):
        d1 = self.down1(x)
        p1 = self.pool1(d1)
        d2 = self.down2(p1)
        p2 = self.pool2(d2)
        d3 = self.down3(p2)
        p3 = self.pool3(d3)
        d4 = self.down4(p3)
        p4 = self.pool4(d4)

        bn = self.bottleneck(p4)

        g4 = self.gate4(bn)
        a4 = self.attn4(d4, g4)
        u4 = self.up4(bn)
        u4 = torch.cat([u4, a4], dim=1)
        d5 = self.dec4(u4)
```

```
        g3 = self.gate3(d5)
        a3 = self.attn3(d3, g3)
        u3 = self.up3(d5)
        u3 = torch.cat([u3, a3], dim=1)
        d6 = self.dec3(u3)

        g2 = self.gate2(d6)
        a2 = self.attn2(d2, g2)
        u2 = self.up2(d6)
        u2 = torch.cat([u2, a2], dim=1)
        d7 = self.dec2(u2)

        g1 = self.gate1(d7)
        a1 = self.attn1(d1, g1)
        u1 = self.up1(d7)
        u1 = torch.cat([u1, a1], dim=1)
        d8 = self.dec1(u1)

        return self.final_conv(d8)
```

In [18]:
```
generator = AttentionResUNetGenerator()
x = torch.randn(4, 3, 128, 256)  # Batch of horizontal motion blurred images
y = generator(x)  # Deblurred output
print(y.shape)  # Should be (4, 3, 128, 256)
```

```
torch.Size([4, 3, 128, 256])
```

# Discriminator

In [19]:
```
class PatchDiscriminator(nn.Module):
    def __init__(self, in_channels=3, base_filters=64):
        super().__init__()
        layers = [
            nn.Conv2d(in_channels, base_filters, kernel_size=4, stride=2, padding=1
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(base_filters, base_filters * 2, kernel_size=4, stride=2, padd
            nn.BatchNorm2d(base_filters * 2),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(base_filters * 2, base_filters * 4, kernel_size=4, stride=2,
            nn.BatchNorm2d(base_filters * 4),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(base_filters * 4, base_filters * 8, kernel_size=4, stride=2,
            nn.BatchNorm2d(base_filters * 8),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(base_filters * 8, 1, kernel_size=4, stride=2, padding=1),
        ]
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

In [20]:
```
image1 = torch.rand((1, 3, 128, 256))

discriminator = PatchDiscriminator()
output = discriminator(image1)
print(output.shape)
```

```
torch.Size([1, 1, 4, 8])
```

## Model Parameters

In [21]:
```
total_params = sum(p.numel() for p in discriminator.parameters())
total_params
```

Out[21]: 2766529

In [22]:
```
discriminator = to_device(discriminator, device)
generator = to_device(generator, device)
```

# Training

## Loss Functions

In [23]:
```
comparison_loss = nn.BCEWithLogitsLoss()
L1_loss_fn = nn.L1Loss()
```

### PaddleOCR

In [24]:
```
ocr_model = PaddleOCR(use_angle_cls=True, lang='en', use_gpu=True, show_log=False)

def tensor_to_bgr_numpy(tensor_img, mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)):
    if tensor_img.dim() == 4:
        tensor_img = tensor_img[0]

    if tensor_img.shape[0] != 3:
         raise ValueError(f"Expected 3 channels, got shape {tensor_img.shape}")

    unnorm = torch.zeros_like(tensor_img)
    for c in range(3):
        unnorm[c] = tensor_img[c] * float(std[c]) + float(mean[c])
    tensor_img = unnorm.clamp(0, 1)

    pil_img = TF.to_pil_image(tensor_img.cpu())
    np_img = np.array(pil_img)[:, :, ::-1]

    return np_img
```

```
[2025/09/01 06:57:59] ppocr WARNING: The first GPU is used for inference by default,
GPU ID: 0
[2025/09/01 06:58:01] ppocr WARNING: The first GPU is used for inference by default,
GPU ID: 0
[2025/09/01 06:58:03] ppocr WARNING: The first GPU is used for inference by default,
GPU ID: 0
```

## Discriminator Training Function

In [25]:
```python
def train_discriminator(discriminator, generator, inputs, targets, d_opt):
    discriminator.train()

    real_images = targets
    fake_images = generator(inputs).detach()

    pred_real = discriminator(real_images)
    pred_fake = discriminator(fake_images)

    real_labels = torch.ones_like(pred_real)
    fake_labels = torch.zeros_like(pred_fake)

    real_score = torch.sigmoid(pred_real).mean().item()
    fake_score = torch.sigmoid(pred_fake).mean().item()

    loss_real = comparison_loss(pred_real, real_labels)
    loss_fake = comparison_loss(pred_fake, fake_labels)

    d_loss = (loss_real + loss_fake) / 2

    d_opt.zero_grad()
    d_loss.backward()
    d_opt.step()

    return d_loss.item(), real_score, fake_score
```

## Generator Training Function

In [26]:
```python
def ssim(pred, target):
    return 1 - ssim_fn(pred, target, data_range=1.0, size_average=True)

lpips_loss = LPIPS(net='vgg').to(device)
lpips_loss.eval()
for param in lpips_loss.parameters():
    param.requires_grad = False

def train_generator(generator, discriminator, inputs, targets, g_opt, adv_lambda=1.
    generator.train()

    fake_images = generator(inputs)
    pred_fake = discriminator(fake_images)

    real_labels = torch.ones_like(pred_fake)

    adv_loss = comparison_loss(pred_fake, real_labels)
```

```
    l1_loss = L1_loss_fn(fake_images, targets)
    perceptual_loss = lpips_loss(fake_images.clamp(-1, 1), targets.clamp(-1, 1)).me
    ssim_loss = ssim(fake_images, targets)

    # Text Loss
    fake_np = tensor_to_bgr_numpy(fake_images)
    real_np = tensor_to_bgr_numpy(targets)

    with torch.no_grad():
        fake_text = ocr_model.ocr(fake_np, cls=False)[0]
        real_text = ocr_model.ocr(real_np, cls=False)[0]

    fake_str = fake_text[0][1][0] if fake_text else ''
    real_str = real_text[0][1][0] if real_text else ''

    fake_str = fake_str.strip().replace(' ', '')
    real_str = real_str.strip().replace(' ', '')

    # Use normalized edit distance as loss
    edit_dist = levenshtein_distance(fake_str, real_str)
    norm_edit_dist = edit_dist / max(len(real_str), 1)
    text_loss = torch.tensor(norm_edit_dist, device=device)

    g_loss = (
        adv_loss * adv_lambda +
        l1_loss * l1_lambda +
        ssim_loss * ssim_lambda +
        perceptual_loss * perceptual_lambda +
        text_loss * text_lambda
    )

    g_opt.zero_grad()
    g_loss.backward()
    g_opt.step()

    return g_loss.item(), fake_images, {
        "total_loss": g_loss.item(),
        "adv": adv_loss.item(),
        "l1": l1_loss.item(),
        "ssim": ssim_loss.item(),
        "perceptual": perceptual_loss.item(),
        "text": text_loss.item()
    }
```

```
Setting up [LPIPS] perceptual loss: trunk [vgg], v[0.1], spatial [off]
Loading model from: C:\Thesis\LiPAD with Paddle\lipadvenv3.9\lib\site-packages\lpips
\weights\v0.1\vgg.pth
```

In [27]:

```
psnr_metric = PeakSignalNoiseRatio(data_range=1.0).to(device)
ssim_metric = StructuralSimilarityIndexMeasure(data_range=1.0).to(device)
fid_metric = FrechetInceptionDistance(feature=2048).to(device)

def to_uint8(tensor):
    # Clamp to [0, 1], scale to [0, 255], then convert to uint8
    tensor = torch.clamp(tensor, 0.0, 1.0)
    tensor = (tensor * 255.0).to(torch.uint8)
```

```
    return tensor

def evaluate_test(generator):
    raw_grid_img, gen_grid_img = create_image_grid_from_loader(generator, test_dl,

    fig, axes = plt.subplots(1, 2, figsize=(10, 8))  # 1 row, 2 columns

    # Raw Test Set
    axes[0].imshow(raw_grid_img)
    axes[0].axis("off")
    axes[0].set_title("Raw Test Set")

    # Generated Test Set
    axes[1].imshow(gen_grid_img)
    axes[1].axis("off")
    axes[1].set_title("Generated Test Set")

    plt.tight_layout()
    plt.show()
```

In [28]:

```
def create_image_grid_from_loader(generator, dataloader, device, num_images=16, ima
    generator.eval()
    generator.to(device)

    raw_images = []
    gen_images = []
    gen_count = 0
    raw_count = 0

    for batch in dataloader:
        # Assume batch is either just images or (images, labels)
        if isinstance(batch, (tuple, list)):
            inputs = batch[0]
        else:
            inputs = batch

        inputs = inputs.to(device)
        outputs = generator(inputs)

        for img in inputs:
            if denormalize:
                img = denorm(img)

            img_resized = TF.resize(img, image_size)
            raw_images.append(img_resized)

            raw_count += 1
            if raw_count >= num_images:
                break

        for out_img in outputs:
            if denormalize:
                out_img = denorm(out_img)

            out_img_resized = TF.resize(out_img, image_size)
            gen_images.append(out_img_resized)
```

```
            gen_count += 1
            if gen_count >= num_images:
                break

        if raw_count >= num_images:
            break

    raw_grid = make_grid(raw_images, nrow=4)
    gen_grid = make_grid(gen_images, nrow=4)

    raw_ndarr = raw_grid.mul(255).byte().cpu().permute(1, 2, 0).numpy()
    gen_ndarr = gen_grid.mul(255).byte().cpu().permute(1, 2, 0).numpy()

    raw_grid_image = Image.fromarray(raw_ndarr)
    gen_grid_image = Image.fromarray(gen_ndarr)

    if save_path:
        gen_grid_image.save(save_path)
        print(f"Saved image grid to {save_path}")

    return raw_grid_image, gen_grid_image
```

In [29]:

```
@torch.no_grad()
def evaluate_generator(generator, valid_dl, epoch):
    generator.eval()
    psnr_vals = []
    ssim_vals = []
    fid_metric.reset()
    shown = False

    for val_inputs, val_targets in valid_dl:
        val_inputs = val_inputs.to(device)
        val_targets = val_targets.to(device)

        val_outputs = generator(val_inputs)

        if not shown:
            print('Validation Data')
            print_images(val_inputs, 5)
            print_images(val_outputs, 5)
            print_images(val_targets, 5)
            shown = True

        # Convert to uint8 for FID
        gen_uint8 = to_uint8(val_outputs)
        target_uint8 = to_uint8(val_targets)

        fid_metric.update(gen_uint8, real=False)
        fid_metric.update(target_uint8, real=True)

        # Compute PSNR/SSIM
        psnr = psnr_metric(val_outputs, val_targets).item()
        ssim = ssim_metric(val_outputs, val_targets).item()
        psnr_vals.append(psnr)
        ssim_vals.append(ssim)
```

```
    mean_psnr = sum(psnr_vals) / len(psnr_vals)
    mean_ssim = sum(ssim_vals) / len(ssim_vals)
    fid = fid_metric.compute().item()

    print(f"Epoch [{epoch+1}/{epochs}] - Mean PSNR: {mean_psnr:.4f}, Mean SSIM: {me

    evaluate_test(generator)
    return mean_psnr, mean_ssim, fid

def get_lr(optimizer):
    return optimizer.param_groups[0]['lr']

def fit(generator, discriminator, epochs, lr, adv_lambda=1.0, l1_lambda=100.0, ssim
        perceptual_lambda=5.0, text_lambda=7.5, g_opt=None, d_opt=None, checkpoint=

    torch.cuda.empty_cache()

    losses_g = []
    losses_d = []
    real_scores = []
    fake_scores = []
    psnrs = []
    ssims = []
    fids = []
    g_losses_trace = []
    g_lrs = []
    d_lrs = []

     # Create optimizers
    if d_opt is None:
        d_opt = optim.Adam(discriminator.parameters(), lr=lr, betas=(beta1, beta2))
    if g_opt is None:
        g_opt = optim.Adam(generator.parameters(), lr=lr, betas=(beta1, beta2))

    # Load checkpoint state if resuming
    if checkpoint is not None:
        print("Resuming training from checkpoint...")
        d_opt.load_state_dict(checkpoint['opt_d'])
        g_opt.load_state_dict(checkpoint['opt_g'])

    g_scheduler = ReduceLROnPlateau(g_opt, mode='min', factor=0.5, patience=10, ver
    d_scheduler = ReduceLROnPlateau(d_opt, mode='min', factor=0.5, patience=12, ver

    # Train
    for epoch in range(epochs):
        torch.cuda.empty_cache()
        generator.train()
        discriminator.train()
        for inputs, targets in tqdm(train_dl):
            inputs = inputs.to(device)
            targets = targets.to(device)

            # Train discriminator
            d_loss, real_score, fake_score = train_discriminator(discriminator, gen
```

```
            # Train generator
            g_loss, fake_images, g_loss_trace = train_generator(generator, discrimi
                                                    l1_lambda, ssim_lambda, perce

        print('Training Data')
        print_images(inputs, 5)
        print_images(fake_images, 5)
        print_images(targets, 5)

        if (epoch + 1) % 5 == 0:
            torch.save(generator.state_dict(), f'[RAU-NET OCRDET LR]generator-Datas
            torch.save(discriminator.state_dict(), f'[RAU-NET OCRDET LR]discriminat
            torch.save({
                'generator': generator.state_dict(),
                'discriminator': discriminator.state_dict(),
                'opt_g': g_opt.state_dict(),
                'opt_d': d_opt.state_dict(),
                'losses_g': losses_g,
                'losses_d': losses_d,
                'real_score': real_scores,
                'fake_score': fake_scores,
                'psnrs': psnrs,
                'ssims': ssims,
                'fids': fids,
                'g_lrs': g_lrs,
                'd_lrs': d_lrs,
                'epoch': epoch + 1
            }, f'[RAU-NET OCRDET LR]Checkpoint-Logs-Datasetv2-{epoch + 1} epoch.pt'

        # Print progress
        try:
            curr_glr = get_lr(g_opt)
            curr_dlr = get_lr(d_opt)

            print('Epoch [{}/{}], loss_g: {:.4f}, loss_d: {:.4f}, real_score: {:.4f
                  epoch+1, epochs, g_loss, d_loss, real_score, fake_score))
            print(f"Current LRs — Generator: {curr_glr:.6f}, Discriminator: {curr_d
        except Exception as err:
            print('Error:', str(err))

        mean_psnr, mean_ssim, fid = evaluate_generator(generator, valid_dl, epoch)

        g_scheduler.step(g_loss)
        d_scheduler.step(d_loss)

        # Record losses and scores
        losses_g.append(g_loss)
        losses_d.append(d_loss)
        real_scores.append(real_score)
        fake_scores.append(fake_score)
        psnrs.append(mean_psnr)
        ssims.append(mean_ssim)
        fids.append(fid)
        g_losses_trace.append(g_loss_trace)
        g_lrs.append(curr_glr)
        d_lrs.append(curr_dlr)
```

```
        save_samples(generator, test_dl, epoch + 1, sample_dir='./visualization-ocr

    return losses_g, losses_d, real_scores, fake_scores, psnrs, ssims, fids, g_loss
```

# Hyperparameter Configurations

In [30]:
```
adv_lambda = 0.25 # Change to 0.25
l1_lambda = 100
ssim_lambda = 5 # Change to 5
perceptual_lambda = 5.0
text_lambda = 10 # Change to 10

lr = 0.0002
beta1 = 0.5
beta2 = 0.999

epochs = 200
```

In [30]:
```
g_opt = optim.Adam(generator.parameters(), lr=lr, betas=(beta1, beta2))
d_opt = optim.Adam(discriminator.parameters(), lr=lr, betas=(beta1, beta2))
```

In [31]:
```
history = []
```

In [32]:
```
%%time
history += fit(generator,
               discriminator,
               epochs,
               lr,
               adv_lambda=adv_lambda,
               l1_lambda=l1_lambda,
               ssim_lambda=ssim_lambda,
               perceptual_lambda=perceptual_lambda,
               text_lambda=text_lambda,
               g_opt=g_opt,
               d_opt=d_opt)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
1250/1250 [19:52<00:00,  1.05it/s]
Training Data
```


Epoch [1/200], loss_g: 18.2146, loss_d: 0.6682, real_score: 0.3407, fake_score: 0.2039
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [1/200] - Mean PSNR: 16.8681, Mean SSIM: 0.7625, FID: 54.599266052246094


100%|
1250/1250 [19:37<00:00, 1.06it/s]
Training Data

Epoch [2/200], loss_g: 10.7149, loss_d: 0.4601, real_score: 0.9112, fake_score: 0.5472
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [2/200] - Mean PSNR: 17.5805, Mean SSIM: 0.7811, FID: 51.86952209472656


100%|
1250/1250 [19:46<00:00, 1.05it/s]
Training Data


Epoch [3/200], loss_g: 10.5099, loss_d: 0.5808, real_score: 0.6860, fake_score: 0.5317
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [3/200] - Mean PSNR: 17.1474, Mean SSIM: 0.7727, FID: 48.428184509277344


100%|
1250/1250 [19:16<00:00, 1.08it/s]
Training Data


Epoch [4/200], loss_g: 10.5624, loss_d: 0.4266, real_score: 0.5125, fake_score: 0.1428
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data





Epoch [4/200] - Mean PSNR: 17.0121, Mean SSIM: 0.7601, FID: 49.03854751586914

Raw Test Set


Generated Test Set


100%|
1250/1250 [19:04<00:00, 1.09it/s]
Training Data


Epoch [5/200], loss_g: 12.2944, loss_d: 0.9898, real_score: 0.9777, fake_score: 0.8444
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [5/200] - Mean PSNR: 18.3938, Mean SSIM: 0.7984, FID: 46.98902130126953


100%|
1250/1250 [19:17<00:00, 1.08it/s]
Training Data


Epoch [6/200], loss_g: 9.6396, loss_d: 1.1877, real_score: 0.1133, fake_score: 0.1102
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [6/200] - Mean PSNR: 18.2961, Mean SSIM: 0.8042, FID: 46.424007415771484


100%|
1250/1250 [19:42<00:00,  1.06it/s]
Training Data

Epoch [7/200], loss_g: 8.1887, loss_d: 0.4803, real_score: 0.6531, fake_score: 0.3864
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data

Epoch [7/200] - Mean PSNR: 18.4826, Mean SSIM: 0.8010, FID: 44.67730712890625


100%|
1250/1250 [19:40<00:00,  1.06it/s]
Training Data


Epoch [8/200], loss_g: 13.6996, loss_d: 0.7281, real_score: 0.5370, fake_score: 0.5254
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [8/200] - Mean PSNR: 18.7647, Mean SSIM: 0.8087, FID: 43.99444580078125


100%|
1250/1250 [19:20<00:00, 1.08it/s]
Training Data





Epoch [9/200], loss_g: 8.4032, loss_d: 0.4975, real_score: 0.6807, fake_score: 0.4400

Current LRs — Generator: 0.000200, Discriminator: 0.000200

Validation Data


Epoch [9/200] - Mean PSNR: 19.0534, Mean SSIM: 0.8129, FID: 42.22925567626953

Raw Test Set


Generated Test Set


100%|

1250/1250 [19:06<00:00, 1.09it/s]

Training Data


Epoch [10/200], loss_g: 9.9892, loss_d: 0.5280, real_score: 0.9569, fake_score: 0.6030
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data




Epoch [10/200] - Mean PSNR: 18.9131, Mean SSIM: 0.8105, FID: 43.680694580078125

Raw Test Set


Generated Test Set


100%|
1250/1250 [23:27<00:00, 1.13s/it]
Training Data



Epoch [11/200], loss_g: 10.6698, loss_d: 0.3221, real_score: 0.8033, fake_score: 0.3358
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [11/200] - Mean PSNR: 18.8334, Mean SSIM: 0.8119, FID: 43.634002685546875


100%|
1250/1250 [23:12<00:00, 1.11s/it]
Training Data


Epoch [12/200], loss_g: 9.5788, loss_d: 0.7207, real_score: 0.5507, fake_score: 0.5429
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [12/200] - Mean PSNR: 19.4375, Mean SSIM: 0.8178, FID: 42.24984359741211


100%|
1250/1250 [21:25<00:00, 1.03s/it]
Training Data


Epoch [13/200], loss_g: 7.0039, loss_d: 0.7567, real_score: 0.2811, fake_score: 0.1534
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data





Epoch [13/200] - Mean PSNR: 18.8323, Mean SSIM: 0.8165, FID: 41.2432861328125

Raw Test Set


Generated Test Set


100%|
1250/1250 [19:27<00:00, 1.07it/s]
Training Data





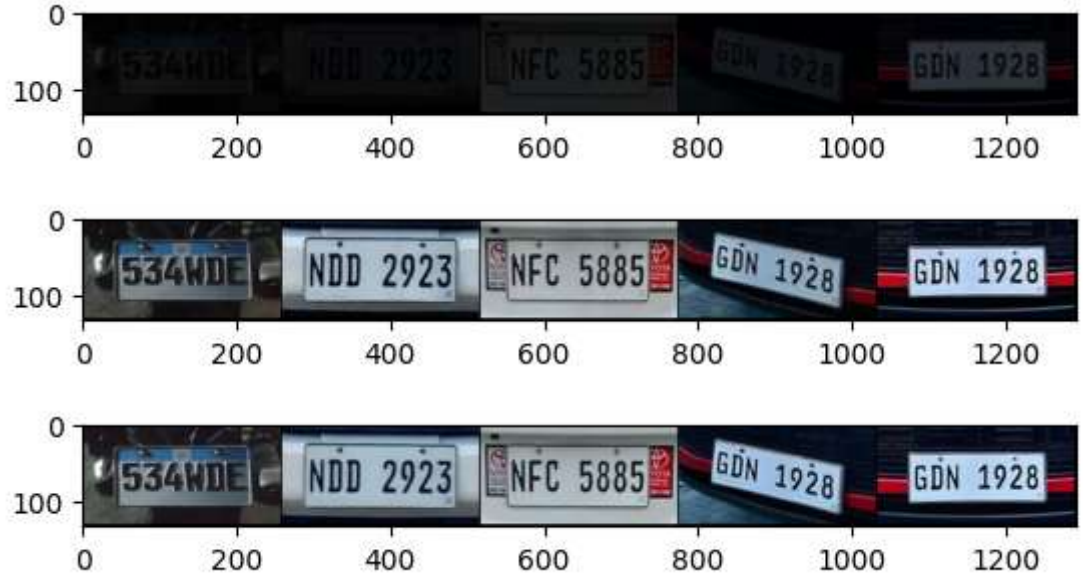



Epoch [14/200], loss_g: 8.4413, loss_d: 0.5858, real_score: 0.4340, fake_score: 0.2354
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [14/200] - Mean PSNR: 19.4585, Mean SSIM: 0.8214, FID: 41.35641860961914


100%|
1250/1250 [19:31<00:00, 1.07it/s]
Training Data


Epoch [15/200], loss_g: 7.9358, loss_d: 0.6933, real_score: 0.8897, fake_score: 0.6886
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data




Epoch [15/200] - Mean PSNR: 18.6810, Mean SSIM: 0.8018, FID: 43.930599212646484


```
100%|
1250/1250 [18:53<00:00,  1.10it/s]
Training Data
```

```
Epoch [16/200], loss_g: 9.5477, loss_d: 0.4396, real_score: 0.5152, fake_score: 0.16
06
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data
```

```
Epoch [16/200] - Mean PSNR: 18.7868, Mean SSIM: 0.8106, FID: 41.41378402709961
```


```
100%|
1250/1250 [19:01<00:00,  1.10it/s]
Training Data
```


Epoch [17/200], loss_g: 9.0014, loss_d: 0.3868, real_score: 0.6185, fake_score: 0.2206
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [17/200] - Mean PSNR: 16.7515, Mean SSIM: 0.7782, FID: 43.40441131591797


100%|
1250/1250 [22:56<00:00, 1.10s/it]
Training Data





Epoch [18/200], loss_g: 7.5288, loss_d: 0.4232, real_score: 0.7459, fake_score: 0.3966
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data




Epoch [18/200] - Mean PSNR: 19.5137, Mean SSIM: 0.8267, FID: 40.65325164794922

Raw Test Set


Generated Test Set

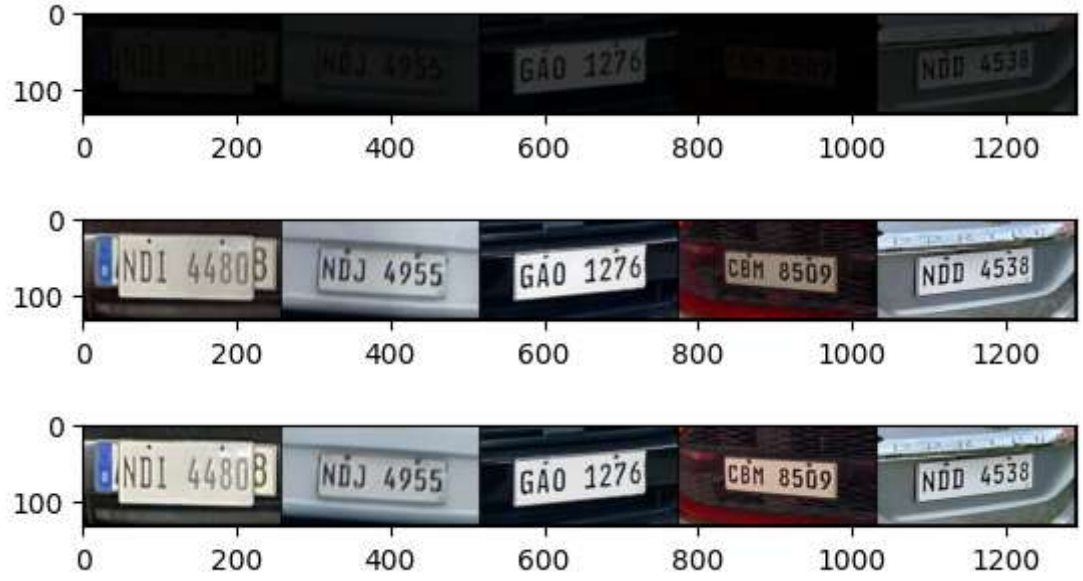

100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [19/200], loss_g: 9.3909, loss_d: 0.3903, real_score: 0.6573, fake_score: 0.27
24
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [19/200] - Mean PSNR: 18.5926, Mean SSIM: 0.8067, FID: 41.4069938659668


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data

Epoch [20/200], loss_g: 18.4405, loss_d: 0.5489, real_score: 0.6026, fake_score: 0.4075
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data




Epoch [20/200] - Mean PSNR: 19.0711, Mean SSIM: 0.8102, FID: 43.037315368652344

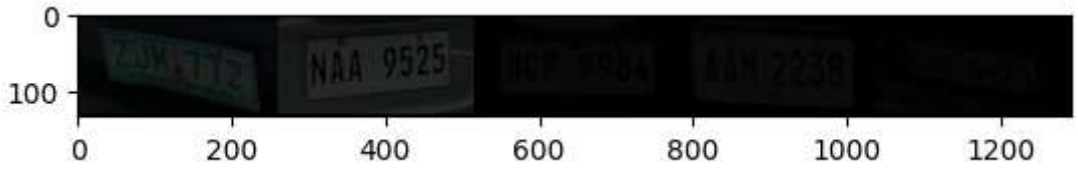

100%|
1250/1250 [19:31<00:00, 1.07it/s]
Training Data


Epoch [21/200], loss_g: 8.5958, loss_d: 0.5092, real_score: 0.5487, fake_score: 0.3047
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [21/200] - Mean PSNR: 18.8582, Mean SSIM: 0.8164, FID: 38.688232421875


100%|
1250/1250 [18:58<00:00, 1.10it/s]
Training Data


Epoch [22/200], loss_g: 6.5751, loss_d: 0.6358, real_score: 0.5311, fake_score: 0.4137
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data







Epoch [22/200] - Mean PSNR: 19.3957, Mean SSIM: 0.8226, FID: 38.86845016479492

Raw Test Set

Generated Test Set

100%|

1250/1250 [18:52<00:00, 1.10it/s]

Training Data




Epoch [23/200], loss_g: 9.2164, loss_d: 0.4331, real_score: 0.5851, fake_score: 0.2346

Current LRs — Generator: 0.000200, Discriminator: 0.000200

Validation Data



Epoch [23/200] - Mean PSNR: 19.2563, Mean SSIM: 0.8223, FID: 39.82571792602539


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [24/200], loss_g: 6.6367, loss_d: 0.4406, real_score: 0.6854, fake_score: 0.3596
Current LRs — Generator: 0.000200, Discriminator: 0.000200
Validation Data


Epoch [24/200] - Mean PSNR: 19.4466, Mean SSIM: 0.8276, FID: 38.446407318115234


```
Epoch 00024: reducing learning rate of group 0 to 1.0000e-04.
100%|
1250/1250 [18:51<00:00,  1.10it/s]
Training Data
```

```
Epoch [25/200], loss_g: 6.8812, loss_d: 0.3451, real_score: 0.9111, fake_score: 0.42
12
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data
```

```
Epoch [25/200] - Mean PSNR: 18.6253, Mean SSIM: 0.8169, FID: 40.5386848449707
```

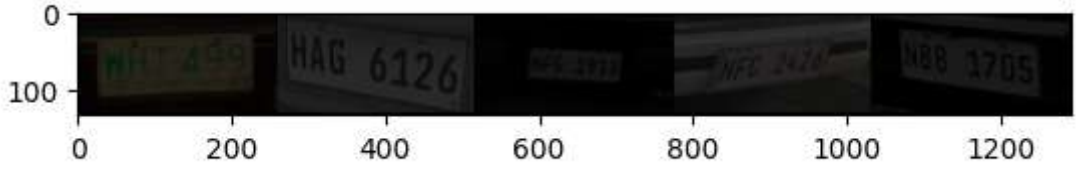

```
100%|
1250/1250 [18:52<00:00,  1.10it/s]
```

Training Data


Epoch [26/200], loss_g: 7.4148, loss_d: 0.5868, real_score: 0.6321, fake_score: 0.4575
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [26/200] - Mean PSNR: 19.1911, Mean SSIM: 0.8207, FID: 39.65391159057617


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data







Epoch [27/200], loss_g: 7.5348, loss_d: 0.5608, real_score: 0.8693, fake_score: 0.59
35
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data




Epoch [27/200] - Mean PSNR: 19.5470, Mean SSIM: 0.8248, FID: 40.53878402709961
Raw Test Set


Generated Test Set


100%|
1250/1250 [18:52<00:00,  1.10it/s]
Training Data



Epoch [28/200], loss_g: 8.2881, loss_d: 0.3993, real_score: 0.7811, fake_score: 0.3907
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [28/200] - Mean PSNR: 19.2068, Mean SSIM: 0.8188, FID: 37.874725341796875
Raw Test Set
Generated Test Set

100%|
1250/1250 [18:51<00:00, 1.10it/s]
Training Data

Epoch [29/200], loss_g: 6.1950, loss_d: 0.3684, real_score: 0.6133, fake_score: 0.1816
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [29/200] - Mean PSNR: 19.7472, Mean SSIM: 0.8326, FID: 38.3443489074707


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [30/200], loss_g: 8.0763, loss_d: 0.4904, real_score: 0.6919, fake_score: 0.4179
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [30/200] - Mean PSNR: 18.9249, Mean SSIM: 0.8193, FID: 37.139339447021484


100%|
1250/1250 [18:51<00:00, 1.10it/s]
Training Data


Epoch [31/200], loss_g: 6.3003, loss_d: 0.3672, real_score: 0.7716, fake_score: 0.3429
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data





Epoch [31/200] - Mean PSNR: 19.5960, Mean SSIM: 0.8316, FID: 37.91668701171875


100%|
1250/1250 [18:51<00:00, 1.10it/s]
Training Data





Epoch [32/200], loss_g: 7.3745, loss_d: 0.7945, real_score: 0.2545, fake_score: 0.0933
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [32/200] - Mean PSNR: 18.4271, Mean SSIM: 0.8119, FID: 37.850799560546875


100%|
1250/1250 [18:50<00:00, 1.11it/s]
Training Data


Epoch [33/200], loss_g: 5.9734, loss_d: 0.5630, real_score: 0.4399, fake_score: 0.1864
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [33/200] - Mean PSNR: 19.3256, Mean SSIM: 0.8245, FID: 36.93244552612305


100%|
1250/1250 [18:49<00:00,  1.11it/s]
Training Data

Epoch [34/200], loss_g: 6.5473, loss_d: 0.3547, real_score: 0.7066, fake_score: 0.26
22
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data

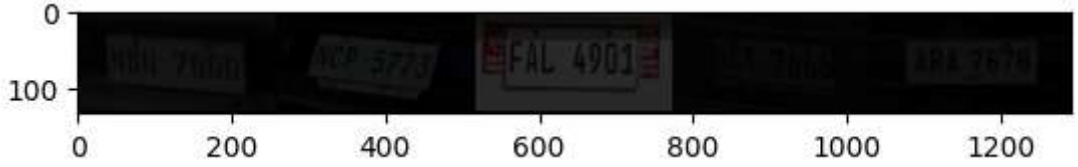
Epoch [34/200] - Mean PSNR: 19.2308, Mean SSIM: 0.8264, FID: 36.49135208129883


100%|
1250/1250 [18:49<00:00,  1.11it/s]
Training Data


Epoch [35/200], loss_g: 7.0147, loss_d: 0.4041, real_score: 0.7281, fake_score: 0.3457
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [35/200] - Mean PSNR: 19.2603, Mean SSIM: 0.8260, FID: 35.86093521118164


100%|
1250/1250 [18:51<00:00, 1.10it/s]
Training Data







Epoch [36/200], loss_g: 6.3441, loss_d: 0.2667, real_score: 0.6959, fake_score: 0.13
15
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [36/200] - Mean PSNR: 19.6823, Mean SSIM: 0.8325, FID: 36.370670318603516


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [37/200], loss_g: 5.7839, loss_d: 0.4171, real_score: 0.8979, fake_score: 0.4830
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data



Epoch [37/200] - Mean PSNR: 19.3616, Mean SSIM: 0.8301, FID: 35.28145217895508

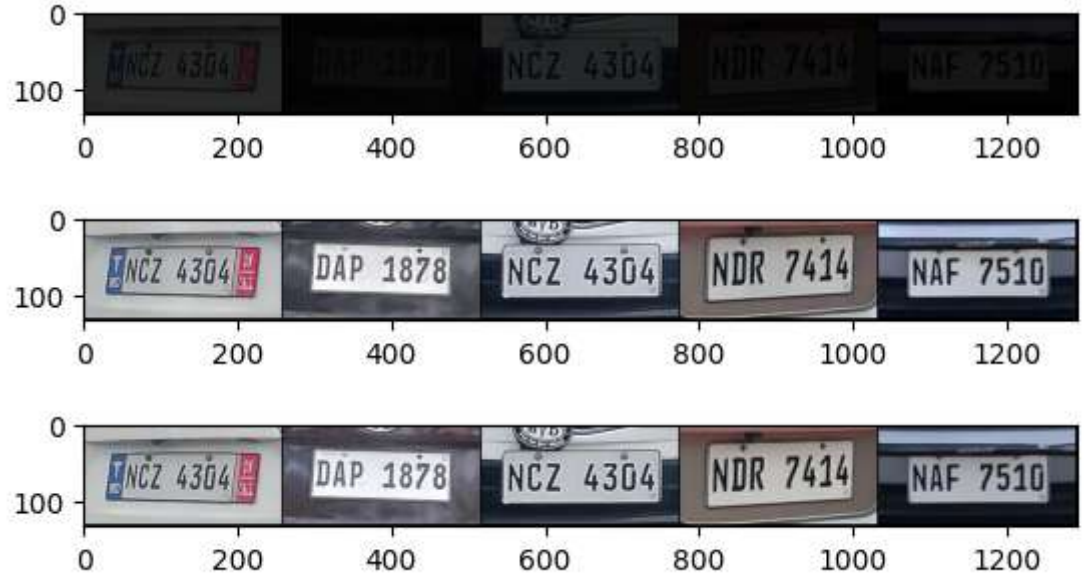
Raw Test Set


Generated Test Set


100%|
1250/1250 [18:51<00:00, 1.10it/s]
Training Data



Epoch [38/200], loss_g: 5.5461, loss_d: 0.3104, real_score: 0.8567, fake_score: 0.3453
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data




Epoch [38/200] - Mean PSNR: 19.6853, Mean SSIM: 0.8278, FID: 36.04048156738281


100%|
1250/1250 [18:51<00:00, 1.10it/s]
Training Data


Epoch [39/200], loss_g: 5.6925, loss_d: 0.5041, real_score: 0.9068, fake_score: 0.5611
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [39/200] - Mean PSNR: 19.6486, Mean SSIM: 0.8346, FID: 37.27739334106445


100%|
1250/1250 [19:43<00:00, 1.06it/s]
Training Data

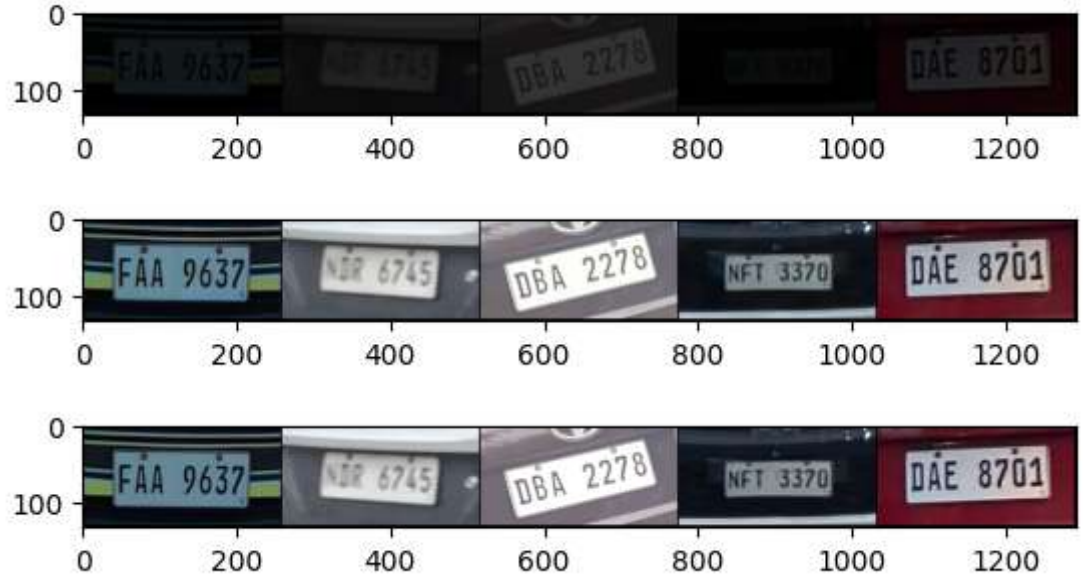

Epoch [40/200], loss_g: 5.1701, loss_d: 0.4868, real_score: 0.6584, fake_score: 0.3881
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data





Epoch [40/200] - Mean PSNR: 19.7682, Mean SSIM: 0.8306, FID: 36.91020965576172


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data










Epoch [41/200], loss_g: 5.1171, loss_d: 0.6099, real_score: 0.7514, fake_score: 0.5397
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [41/200] - Mean PSNR: 18.8694, Mean SSIM: 0.8104, FID: 41.353050231933594


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [42/200], loss_g: 5.8439, loss_d: 0.4059, real_score: 0.5288, fake_score: 0.1113
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [42/200] - Mean PSNR: 19.6525, Mean SSIM: 0.8296, FID: 35.89630889892578


100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data

Epoch [43/200], loss_g: 6.4200, loss_d: 0.6108, real_score: 0.7863, fake_score: 0.5813
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data

Epoch [43/200] - Mean PSNR: 19.7165, Mean SSIM: 0.8333, FID: 35.90210723876953


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data


Epoch [44/200], loss_g: 6.4869, loss_d: 0.5800, real_score: 0.4240, fake_score: 0.1801
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [44/200] - Mean PSNR: 19.3867, Mean SSIM: 0.8283, FID: 35.04948806762695


100%|
1250/1250 [19:11<00:00, 1.09it/s]
Training Data





Epoch [45/200], loss_g: 8.3734, loss_d: 0.3464, real_score: 0.9302, fake_score: 0.42
98
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [45/200] - Mean PSNR: 19.8657, Mean SSIM: 0.8340, FID: 35.597843170166016


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data

Epoch [46/200], loss_g: 7.0787, loss_d: 0.4467, real_score: 0.6669, fake_score: 0.3433
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data

Epoch [46/200] - Mean PSNR: 19.8077, Mean SSIM: 0.8324, FID: 36.3724250793457

Raw Test Set

Generated Test Set

100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data

Epoch [47/200], loss_g: 4.9231, loss_d: 0.7286, real_score: 0.3368, fake_score: 0.2044
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [47/200] - Mean PSNR: 19.9029, Mean SSIM: 0.8369, FID: 35.2142219543457

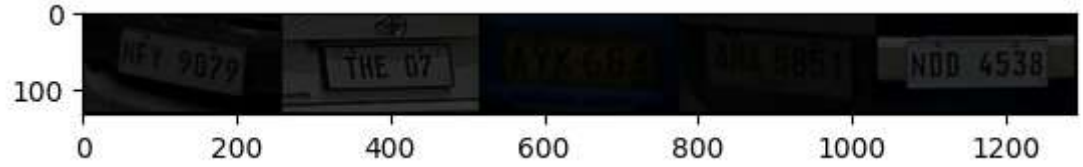

100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [48/200], loss_g: 4.8662, loss_d: 0.4509, real_score: 0.6252, fake_score: 0.2898
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data


Epoch [48/200] - Mean PSNR: 19.1885, Mean SSIM: 0.8145, FID: 39.32527160644531


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data


Epoch [49/200], loss_g: 4.9082, loss_d: 0.4044, real_score: 0.7409, fake_score: 0.3698
Current LRs — Generator: 0.000200, Discriminator: 0.000100
Validation Data







Epoch [49/200] - Mean PSNR: 19.6040, Mean SSIM: 0.8319, FID: 34.38228225708008

Raw Test Set

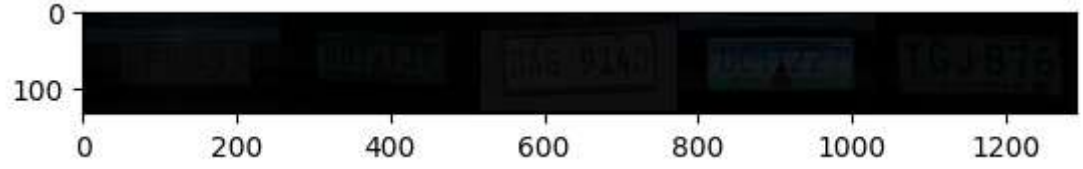

Generated Test Set


Epoch 00049: reducing learning rate of group 0 to 5.0000e-05.

100%|
1250/1250 [18:52<00:00,  1.10it/s]
Training Data




Epoch [50/200], loss_g: 5.6646, loss_d: 0.4466, real_score: 0.5714, fake_score: 0.2401
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data



Epoch [50/200] - Mean PSNR: 19.4780, Mean SSIM: 0.8273, FID: 35.92219543457031


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data

Epoch [51/200], loss_g: 5.9107, loss_d: 0.4719, real_score: 0.5146, fake_score: 0.1900
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data

Epoch [51/200] - Mean PSNR: 19.3230, Mean SSIM: 0.8229, FID: 39.22008514404297


100%|
1250/1250 [18:54<00:00,  1.10it/s]
Training Data

Epoch [52/200], loss_g: 5.7576, loss_d: 0.4770, real_score: 0.7740, fake_score: 0.4648
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data

Epoch [52/200] - Mean PSNR: 19.5790, Mean SSIM: 0.8318, FID: 35.6588134765625


100%|
1250/1250 [18:54<00:00,  1.10it/s]
Training Data


Epoch [53/200], loss_g: 5.3253, loss_d: 0.5075, real_score: 0.5630, fake_score: 0.3145
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data

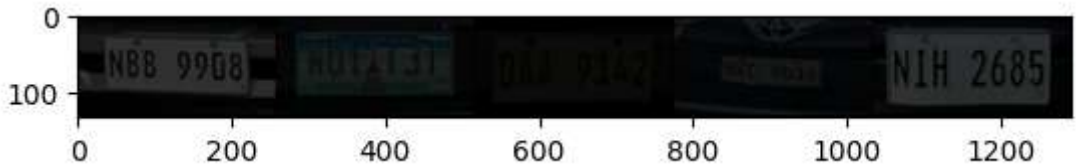

Epoch [53/200] - Mean PSNR: 19.1471, Mean SSIM: 0.8299, FID: 33.63098907470703


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data





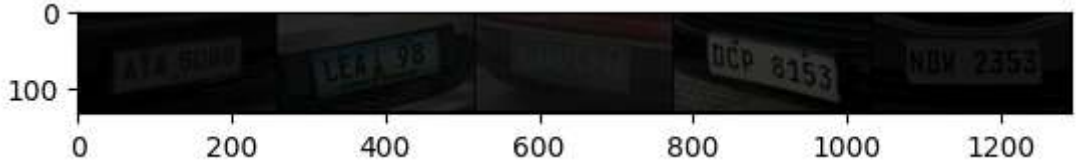
Epoch [54/200], loss_g: 5.8746, loss_d: 0.6818, real_score: 0.3782, fake_score: 0.2284
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data






Epoch [54/200] - Mean PSNR: 18.9682, Mean SSIM: 0.8186, FID: 40.606109619140625


100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data





Epoch [55/200], loss_g: 5.4337, loss_d: 0.2782, real_score: 0.6853, fake_score: 0.1366
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data


Epoch [55/200] - Mean PSNR: 19.5003, Mean SSIM: 0.8295, FID: 36.19438552856445


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data

Epoch [56/200], loss_g: 5.4361, loss_d: 0.3521, real_score: 0.7003, fake_score: 0.26
34
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data


Epoch [56/200] - Mean PSNR: 19.9919, Mean SSIM: 0.8382, FID: 33.99836730957031


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data


Epoch [57/200], loss_g: 3.9459, loss_d: 0.6200, real_score: 0.6484, fake_score: 0.50
01
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data


Epoch [57/200] - Mean PSNR: 19.5011, Mean SSIM: 0.8229, FID: 37.963592529296875


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data


Epoch [58/200], loss_g: 4.8085, loss_d: 0.3983, real_score: 0.6430, fake_score: 0.2594
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data





Epoch [58/200] - Mean PSNR: 19.8338, Mean SSIM: 0.8371, FID: 33.86127853393555


100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data


Epoch [59/200], loss_g: 5.5164, loss_d: 0.4907, real_score: 0.5346, fake_score: 0.2369
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data


Epoch [59/200] - Mean PSNR: 19.6117, Mean SSIM: 0.8320, FID: 36.521175384521484

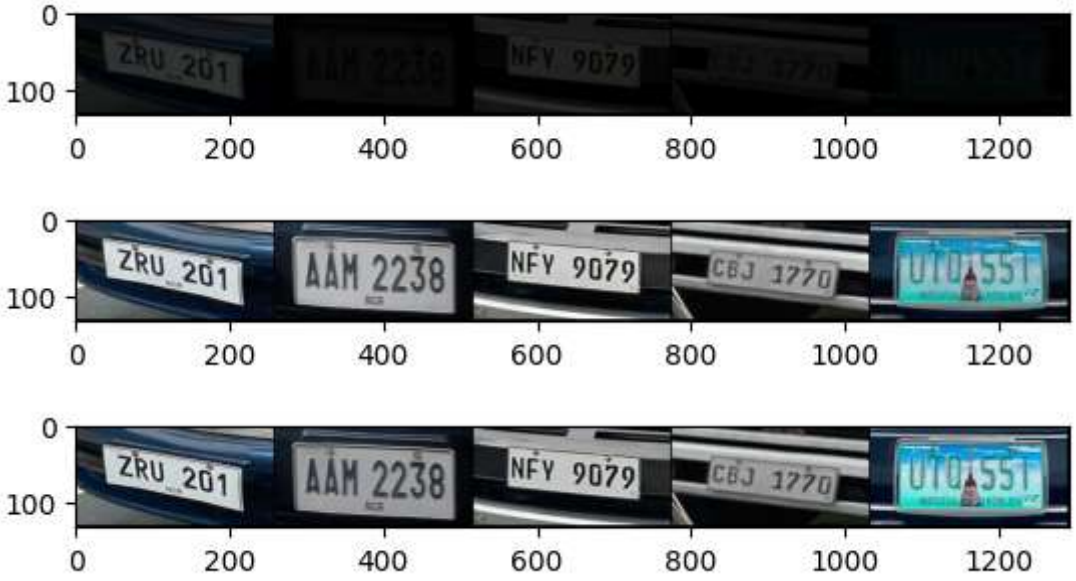

100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data


Epoch [60/200], loss_g: 6.4611, loss_d: 0.6633, real_score: 0.9854, fake_score: 0.6857
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data


Epoch [60/200] - Mean PSNR: 18.7329, Mean SSIM: 0.8182, FID: 37.373294830322266


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data

Epoch [61/200], loss_g: 5.5181, loss_d: 0.3960, real_score: 0.7753, fake_score: 0.3819
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data



Epoch [61/200] - Mean PSNR: 19.4703, Mean SSIM: 0.8283, FID: 34.49689483642578


100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data


Epoch [62/200], loss_g: 7.1770, loss_d: 0.3835, real_score: 0.5847, fake_score: 0.1590
Current LRs — Generator: 0.000200, Discriminator: 0.000050
Validation Data


Epoch [62/200] - Mean PSNR: 19.6412, Mean SSIM: 0.8327, FID: 33.97242736816406

Raw Test Set


Generated Test Set

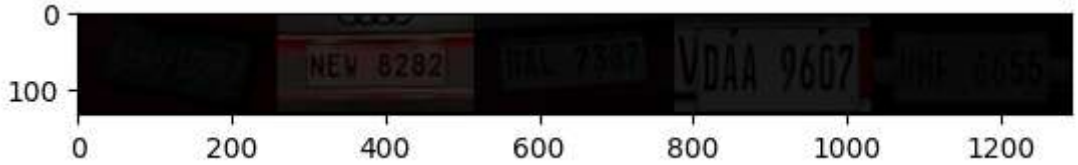

Epoch 00062: reducing learning rate of group 0 to 2.5000e-05.

100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data





Epoch [63/200], loss_g: 4.7057, loss_d: 0.5082, real_score: 0.5016, fake_score: 0.2238
Current LRs — Generator: 0.000200, Discriminator: 0.000025
Validation Data






Epoch [63/200] - Mean PSNR: 20.0471, Mean SSIM: 0.8383, FID: 33.27448654174805


100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data


Epoch [64/200], loss_g: 4.4399, loss_d: 0.5381, real_score: 0.5633, fake_score: 0.3456
Current LRs — Generator: 0.000200, Discriminator: 0.000025
Validation Data


Epoch [64/200] - Mean PSNR: 19.8891, Mean SSIM: 0.8381, FID: 32.707763671875

Raw Test Set


Generated Test Set


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data

Epoch [65/200], loss_g: 4.9765, loss_d: 0.3520, real_score: 0.8566, fake_score: 0.3965
Current LRs — Generator: 0.000200, Discriminator: 0.000025
Validation Data


Epoch [65/200] - Mean PSNR: 19.6465, Mean SSIM: 0.8330, FID: 33.009307861328125

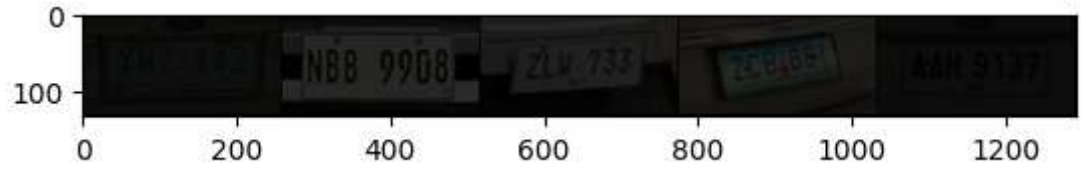

100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data


Epoch [66/200], loss_g: 4.0273, loss_d: 0.5859, real_score: 0.5196, fake_score: 0.3389
Current LRs — Generator: 0.000200, Discriminator: 0.000025
Validation Data


Epoch [66/200] - Mean PSNR: 19.9328, Mean SSIM: 0.8391, FID: 32.64397048950195


100%|
1250/1250 [18:55<00:00, 1.10it/s]
Training Data


Epoch [67/200], loss_g: 5.0199, loss_d: 0.3486, real_score: 0.6669, fake_score: 0.2154
Current LRs — Generator: 0.000200, Discriminator: 0.000025
Validation Data







Epoch [67/200] - Mean PSNR: 19.6101, Mean SSIM: 0.8333, FID: 34.50214767456055

Raw Test Set


Generated Test Set


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data




Epoch [68/200], loss_g: 4.4843, loss_d: 0.5318, real_score: 0.7383, fake_score: 0.4897
Current LRs — Generator: 0.000200, Discriminator: 0.000025
Validation Data



Epoch [68/200] - Mean PSNR: 19.5516, Mean SSIM: 0.8278, FID: 36.05794143676758


Epoch 00068: reducing learning rate of group 0 to 1.0000e-04.

100%|
1250/1250 [18:53<00:00, 1.10it/s]

Training Data


Epoch [69/200], loss_g: 5.2764, loss_d: 0.7444, real_score: 0.6191, fake_score: 0.5829

Current LRs — Generator: 0.000100, Discriminator: 0.000025

Validation Data

Epoch [69/200] - Mean PSNR: 19.9033, Mean SSIM: 0.8383, FID: 32.36399841308594


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data


Epoch [70/200], loss_g: 5.2822, loss_d: 0.3369, real_score: 0.6463, fake_score: 0.1805
Current LRs — Generator: 0.000100, Discriminator: 0.000025
Validation Data


Epoch [70/200] - Mean PSNR: 19.9513, Mean SSIM: 0.8362, FID: 33.87950134277344


100%|
1250/1250 [18:53<00:00, 1.10it/s]

Training Data


Epoch [71/200], loss_g: 4.5609, loss_d: 0.4064, real_score: 0.6947, fake_score: 0.3235
Current LRs — Generator: 0.000100, Discriminator: 0.000025
Validation Data


Epoch [71/200] - Mean PSNR: 20.0985, Mean SSIM: 0.8401, FID: 31.13563346862793


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data





Epoch [72/200], loss_g: 4.3425, loss_d: 0.4502, real_score: 0.6470, fake_score: 0.3281
Current LRs — Generator: 0.000100, Discriminator: 0.000025
Validation Data


Epoch [72/200] - Mean PSNR: 19.9807, Mean SSIM: 0.8428, FID: 32.067867279052734


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data


Epoch [73/200], loss_g: 4.1055, loss_d: 0.5455, real_score: 0.5956, fake_score: 0.3929
Current LRs — Generator: 0.000100, Discriminator: 0.000025
Validation Data

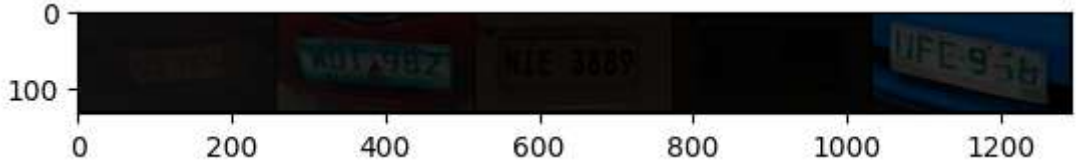



Epoch [73/200] - Mean PSNR: 19.6563, Mean SSIM: 0.8357, FID: 32.406192779541016

Raw Test Set


Generated Test Set


100%|
1250/1250 [18:52<00:00, 1.10it/s]
Training Data




Epoch [74/200], loss_g: 4.4238, loss_d: 0.6718, real_score: 0.7015, fake_score: 0.5808
Current LRs — Generator: 0.000100, Discriminator: 0.000025
Validation Data


Epoch [74/200] - Mean PSNR: 19.6914, Mean SSIM: 0.8361, FID: 33.11274719238281


100%|
1250/1250 [18:53<00:00, 1.10it/s]
Training Data


Epoch [75/200], loss_g: 4.3721, loss_d: 0.6064, real_score: 0.4503, fake_score: 0.2776
Current LRs — Generator: 0.000100, Discriminator: 0.000025
Validation Data


Epoch [75/200] - Mean PSNR: 19.7120, Mean SSIM: 0.8353, FID: 34.76650619506836


Epoch 00075: reducing learning rate of group 0 to 1.2500e-05.

100%| 1250/1250 [18:52<00:00, 1.10it/s]

Training Data


Epoch [76/200], loss_g: 5.2036, loss_d: 0.6257, real_score: 0.4503, fake_score: 0.2957

Current LRs — Generator: 0.000100, Discriminator: 0.000013

Validation Data





Epoch [76/200] - Mean PSNR: 19.8029, Mean SSIM: 0.8357, FID: 32.24737548828125


100%|
1250/1250 [18:54<00:00, 1.10it/s]
Training Data


Epoch [77/200], loss_g: 3.9547, loss_d: 0.6082, real_score: 0.5193, fake_score: 0.3750
Current LRs — Generator: 0.000100, Discriminator: 0.000013
Validation Data


Epoch [77/200] - Mean PSNR: 20.3417, Mean SSIM: 0.8443, FID: 32.42433166503906


100%|
1250/1250 [19:12<00:00, 1.08it/s]
Training Data


Epoch [78/200], loss_g: 3.8956, loss_d: 0.4747, real_score: 0.6029, fake_score: 0.3149
Current LRs — Generator: 0.000100, Discriminator: 0.000013
Validation Data


Epoch [78/200] - Mean PSNR: 20.1911, Mean SSIM: 0.8436, FID: 31.16459846496582


100%|
1250/1250 [18:56<00:00,  1.10it/s]
Training Data

Epoch [79/200], loss_g: 4.6817, loss_d: 0.3201, real_score: 0.7232, fake_score: 0.24
55
Current LRs — Generator: 0.000100, Discriminator: 0.000013
Validation Data

Epoch [79/200] - Mean PSNR: 19.8689, Mean SSIM: 0.8388, FID: 32.296390533447266


100%|
1250/1250 [19:43<00:00,  1.06it/s]
Training Data


Epoch [80/200], loss_g: 4.3185, loss_d: 0.5358, real_score: 0.6889, fake_score: 0.46
16
Current LRs — Generator: 0.000100, Discriminator: 0.000013
Validation Data


Epoch [80/200] - Mean PSNR: 19.7793, Mean SSIM: 0.8401, FID: 31.546554565429688


0%|
| 3/1250 [00:11<1:18:03, 3.76s/it]

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
File <timed exec>:1

Cell In[28], line 91, in fit(generator, discriminator, epochs, lr, adv_lambda, l1_la
mbda, ssim_lambda, perceptual_lambda, text_lambda, g_opt, d_opt, checkpoint)
     88     d_loss, real_score, fake_score = train_discriminator(discriminator, gene
rator, inputs, targets, d_opt)
     90     # Train generator
---> 91     g_loss, fake_images, g_loss_trace = train_generator(generator, discrimin
ator, inputs, targets, g_opt, adv_lambda,
     92                                                         l1_lambda, ssim_lambda, percep
tual_lambda, text_lambda)
     94 print('Training Data')
     95 print_images(inputs, 5)

Cell In[25], line 54, in train_generator(generator, discriminator, inputs, targets,
g_opt, adv_lambda, l1_lambda, ssim_lambda, perceptual_lambda, text_lambda)
     50 g_loss.backward()
     51 g_opt.step()
     53 return g_loss.item(), fake_images, {
---> 54     "total_loss": g_loss.item(),
     55     "adv": adv_loss.item(),
     56     "l1": l1_loss.item(),
     57     "ssim": ssim_loss.item(),
     58     "perceptual": perceptual_loss.item(),
     59     "text": text_loss.item()
     60 }

KeyboardInterrupt:
```

# Post Training

In [34]:
```
torch.cuda.empty_cache()
```

## Save Model

In [34]:
```
torch.save(generator.state_dict(), "[RAU-NET]generator-Datasetv2-5epoch.pt")
torch.save(discriminator.state_dict(), "[RAU-NET]discriminator-Datasetv2-5epoch.pt"
```

In [35]:
```
torch.save({
    'generator_state_dict': generator.state_dict(),
    'discriminator_state_dict': discriminator.state_dict(),
    'opt_g_state_dict': g_opt.state_dict(),
    'opt_d_state_dict': d_opt.state_dict(),
    'history': history
}, f'[RAU-NET]Checkpoint-Datasetv2-30epoch.pt')
```

# Model Evaluation

In [ ]:

```
psnr_vals = []
ssim_vals = []

fid_metric = FrechetInceptionDistance(feature=2048).to(device)
fid_metric.reset()

generator.eval()
for val_input, val_target in valid_dl:
    val_input = val_input.to(device)
    val_target = val_target.to(device)

    with torch.no_grad():
        gen_output = generator(val_input)

    # Convert to uint8 for FID
    gen_uint8 = to_uint8(gen_output)
    target_uint8 = to_uint8(val_target)

    fid_metric.update(gen_uint8, real=False)
    fid_metric.update(target_uint8, real=True)

    show_generated_images(gen_output)

    # Compute PSNR/SSIM
    psnr = psnr_metric(gen_output, val_target).item()
    ssim = ssim_metric(gen_output, val_target).item()
    psnr_vals.append(psnr)
    ssim_vals.append(ssim)

print("Mean PSNR:", sum(psnr_vals) / len(psnr_vals))
print("Mean SSIM:", sum(ssim_vals) / len(ssim_vals))
print("FID:", fid_metric.compute().item())
```


NCL 5807
NBD 8944
NDD 8687
CAZ 5059
NBB 6746
CBM 8509
NIP 1693
NDH 9369
AAC 3674
NAY 5738
CAU 5254
NBD 6823
DAJ 9657
UOA 813
CJC 86
NIP 1693
AIA 3167
ABP 5735
NBB 9908
XFH 671
ZRU 201
DAM 4386
WGJ 901
WPC·242
DAJ 9657
NIH 2685
XPN 483
BDC 888
NAC 6182
NHF 6655
NBP 3657
NIP 498


DAZ 9945
NBK 7762
AIA 3157
TGJ·876
DCP 9851
NDG 4508
0401-0
DCP 9984
DAG 9140
NAV 70
534NDE
TGJ·876
TGJ·876
NAP 9734
NBP 3657
JIX 125
AYK·684
WLI 301
WPC·242
DAL 7387
ZDA 908
NIP 498
NDS 5916
UCI 227
ZRU 201
CAD·968
VDAA 9607
UIB 652
NLQ 302
DAG 9140
NDR 7414
DAA 9142
359DPK
AAA 6660
NBZ 4064
DAG 9140
WPC·242

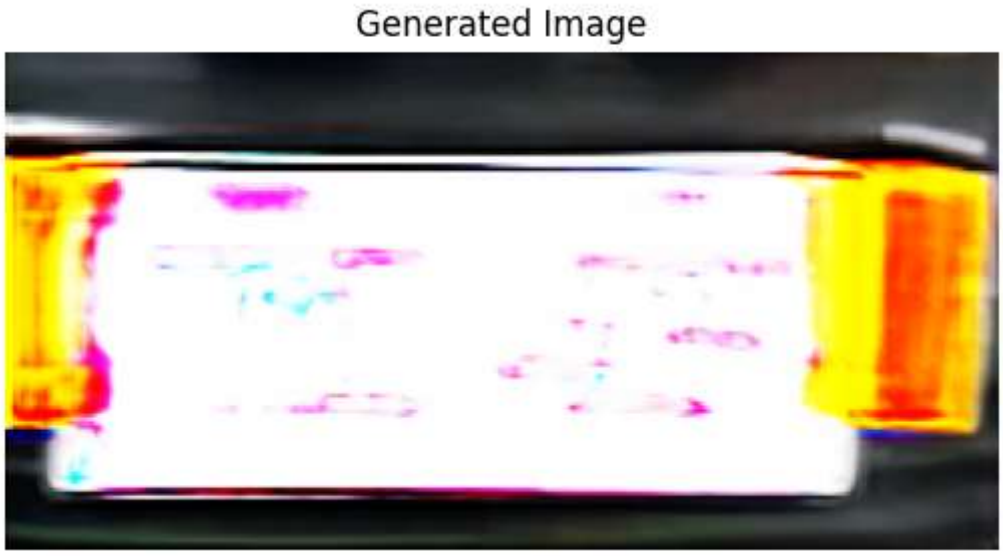

NCC 1984
NBS 9297
NCR 6979
NAA 9525
XRJ 395
UHJ 230
DAG 9140
DCP 9851
DAP 1878
E1 D960
NDD 4538
PRW 348
NDD 2923
DAA 9142
CAM 5524
WQR 579
ZDH 705
THE 07
NEO 8835
NBA 2005
NAF 7510
DAL 7387
DAZ 4722
XKZ 642
AHH 758
AHA 5851
NBQ 5239
NCJ 6149
NCZ 4304
ATA 5018
GEB 4887
NBU 4937
AIA 3157
NBZ 4064
DCP 9851
PNB 439
236DAJ
TGJ 876


ATA 5018
NDL 7075
NAF 7510
ABO 1958
ZDA 908
NFC 5885
NDD 8687
CAM 5524
NBZ 1806
NAC 6182
BEP 961
WPC·242
UUO 605
AAA 5409
NFY 9079
ZJM 772
WUT 131
NIP 498
ZLW 733
NDJ 4955
NFY 9079
WPC·242
ABO 1958
GAO 1276
DAG 9140
NDS 5916
TGJ·876
WLT 301
NGK 6104
CAU 5254
359DPK
DAG 9140
ARA 2976
359DPK
NEF 3315
WKV 161
NEG 5445
359DPK


In [32]:
```python
def deblur_image(image_path, generator, device):
    image = Image.open(image_path).convert('RGB')
    input_tensor = valid_transform(image).unsqueeze(0).to(device)

    generator.eval()
    with torch.no_grad():
        gen_image = generator(input_tensor)

    output = denorm(gen_image.squeeze(0).cpu())
```

```
    plt.imshow(output.permute(1, 2, 0))  # CHW → HWC
    plt.axis('off')
    plt.title("Generated Image")
    plt.show()
```

In [31]:
```
deblur_image(os.path.join(DATA_DIR, 'h_blur', 'v2', 'test', 'h_blur', '129.jpg'), g
```




In [33]:
```
raw_grid_img, gen_grid_img = create_image_grid_from_loader(generator, test_dl, devi

fig, axes = plt.subplots(1, 2, figsize=(15, 15))  # 1 row, 2 columns

# Raw Test Set
axes[0].imshow(raw_grid_img)
axes[0].axis("off")
axes[0].set_title("Raw Test Set")

# Generated Test Set
axes[1].imshow(gen_grid_img)
axes[1].axis("off")
axes[1].set_title("Generated Test Set")

plt.tight_layout()
plt.show()
```


In [33]:
```
torch.cuda.empty_cache()
```

# Load Model

In [31]:
```
MODELS_DIR = os.path.join('..', '..', '..', 'models', 'h_blur')
```

In [31]:
```
generator = AttentionResUNetGenerator()
loaded_g_wts = torch.load('[RAU-NET OCRDET LR]generator-Datasetv2-80 epoch.pt')
generator.load_state_dict(loaded_g_wts)
generator = to_device(generator, device)

discriminator = PatchDiscriminator()
loaded_d_wts = torch.load('[RAU-NET OCRDET LR]discriminator-Datasetv2-80 epoch.pt')
discriminator.load_state_dict(loaded_d_wts)
discriminator = to_device(discriminator, device)
```

In [29]:
```
checkpoint = torch.load('[RAU-NET OCRDET LR]Checkpoint-Logs-Datasetv2-80 epoch.pt')
#g_opt.load_state_dict(checkpoint['opt_g_state_dict'])
#d_opt.load_state_dict(checkpoint['opt_d_state_dict'])
#history = checkpoint['losses_g']
```

In [39]:
```
def are_models_identical(model1, model2):
    # Check if state dictionaries have the same keys
    if model1.state_dict().keys() != model2.state_dict().keys():
        return False

    # Compare each parameter's values
    for key in model1.state_dict():
        if not torch.equal(model1.state_dict()[key], model2.state_dict()[key]):
            return False
    return True
```

In [45]:
```
if are_models_identical(generator1, generator2):
    print("Models have identical weights and biases")
else:
    print("Models have different weights or biases")
```

```
Models have different weights or biases
```

In [9]:
```
len(history)
```

Out[9]: 79

# Save Reconstructed Test Set

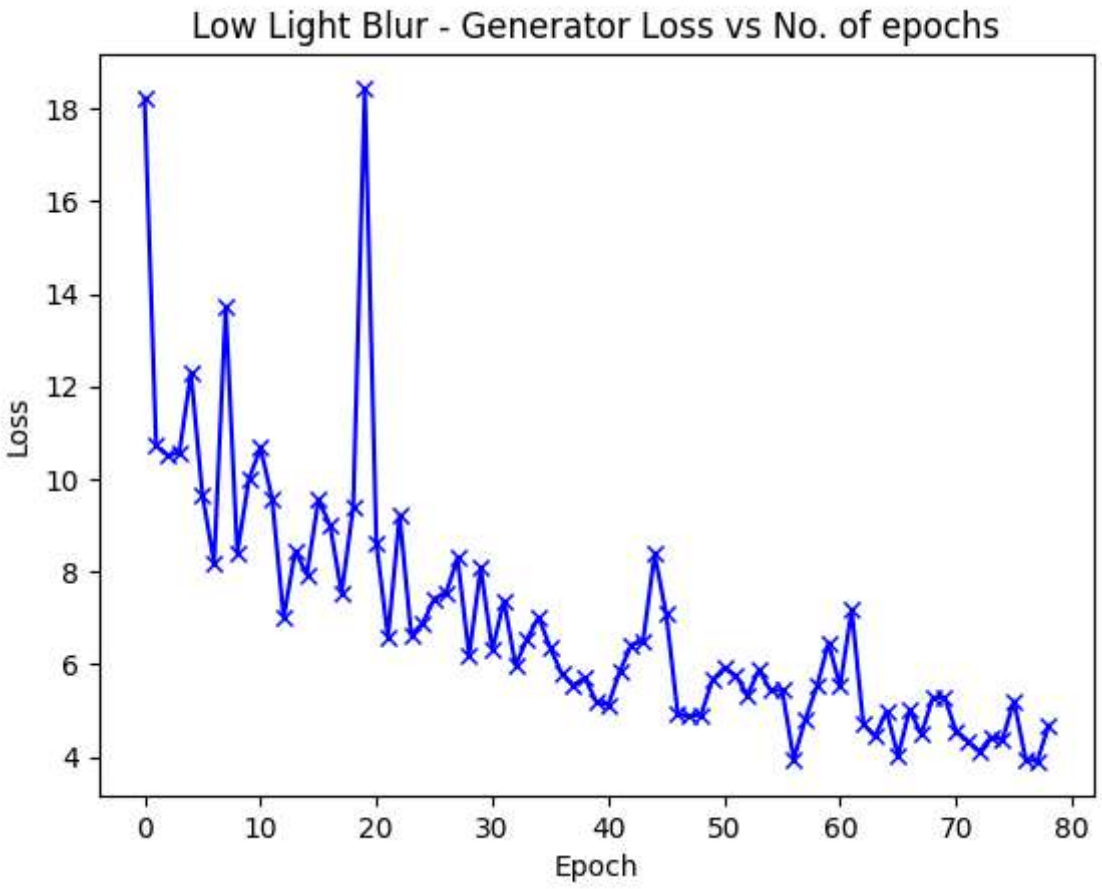
In [39]:
```
def save_generated_tests(generator, test_dl, sample_dir='./results'):
    generator.eval()
    os.makedirs(sample_dir, exist_ok=True)

    with torch.no_grad():
        global_idx = 1  # running index for image IDs

        for inputs, _ in test_dl:
            inputs = inputs.to(device)

            fake_images = generator(inputs)
            fake_images = fake_images.clamp(-1, 1)

            batch_size = fake_images.size(0)
```

```
            for b in range(batch_size):
                img = fake_images[b]

                file_name = f"{global_idx}.jpg"
                save_path = os.path.join(sample_dir, file_name)

                save_image(denorm(img).unsqueeze(0), save_path, nrow=1)
                global_idx += 1
```

In [40]:
```
save_generated_tests(generator, test_dl)
```

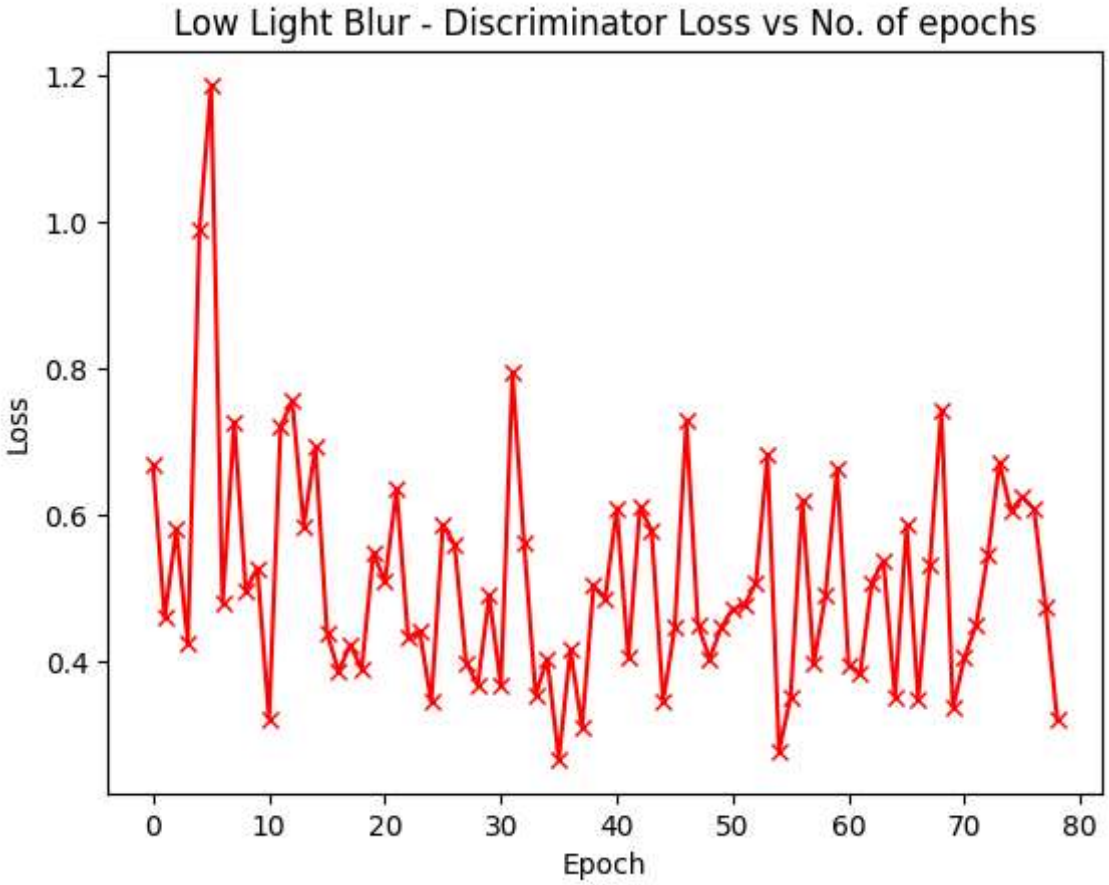
In [ ]:
```
```

In [38]:
```
def plot_g_losses(history):
    losses_g = [x for x in history['losses_g']]
    plt.plot(losses_g, '-bx')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Low Light Blur - Generator Loss vs No. of epochs')

plot_g_losses(checkpoint)
```

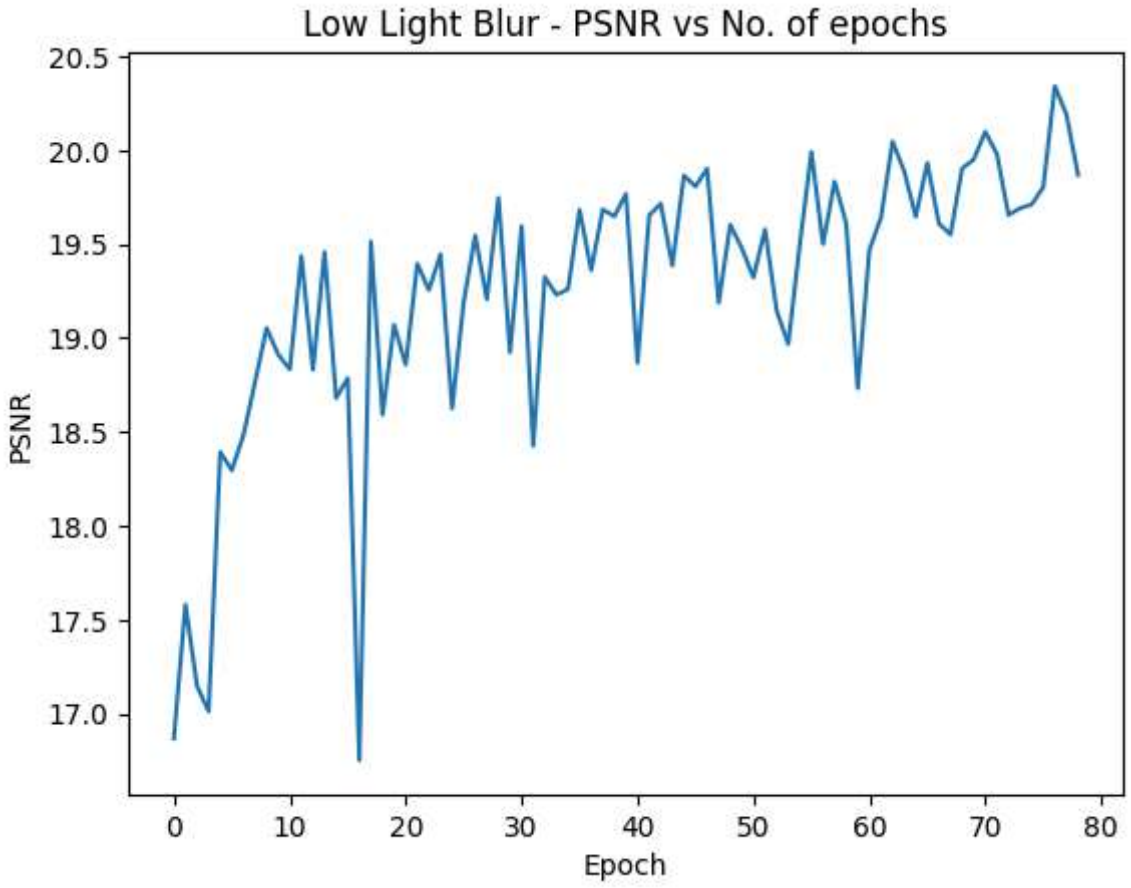



In [39]:
```
def plot_d_losses(history):
    losses_d = [x for x in history['losses_d']]
    plt.plot(losses_d, '-rx')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
```

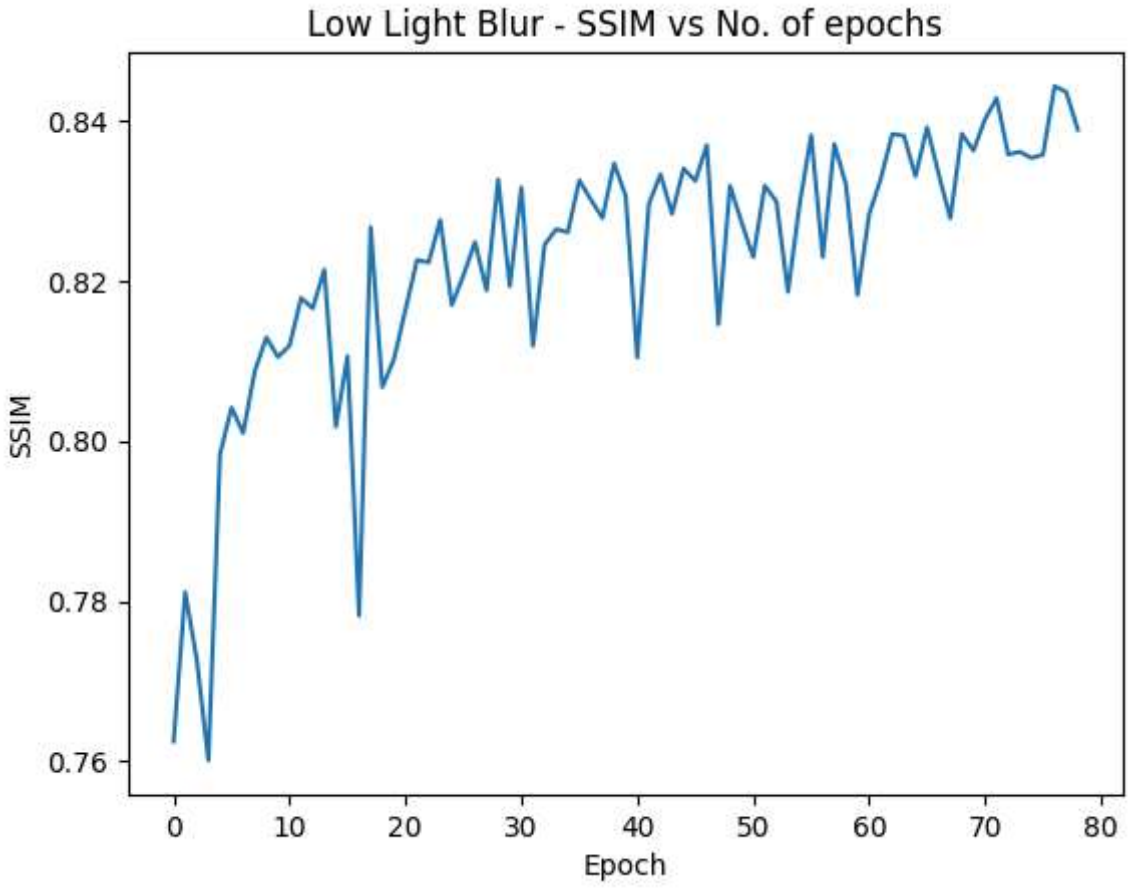
```
    plt.title('Low Light Blur - Discriminator Loss vs No. of epochs')

plot_d_losses(checkpoint)
```



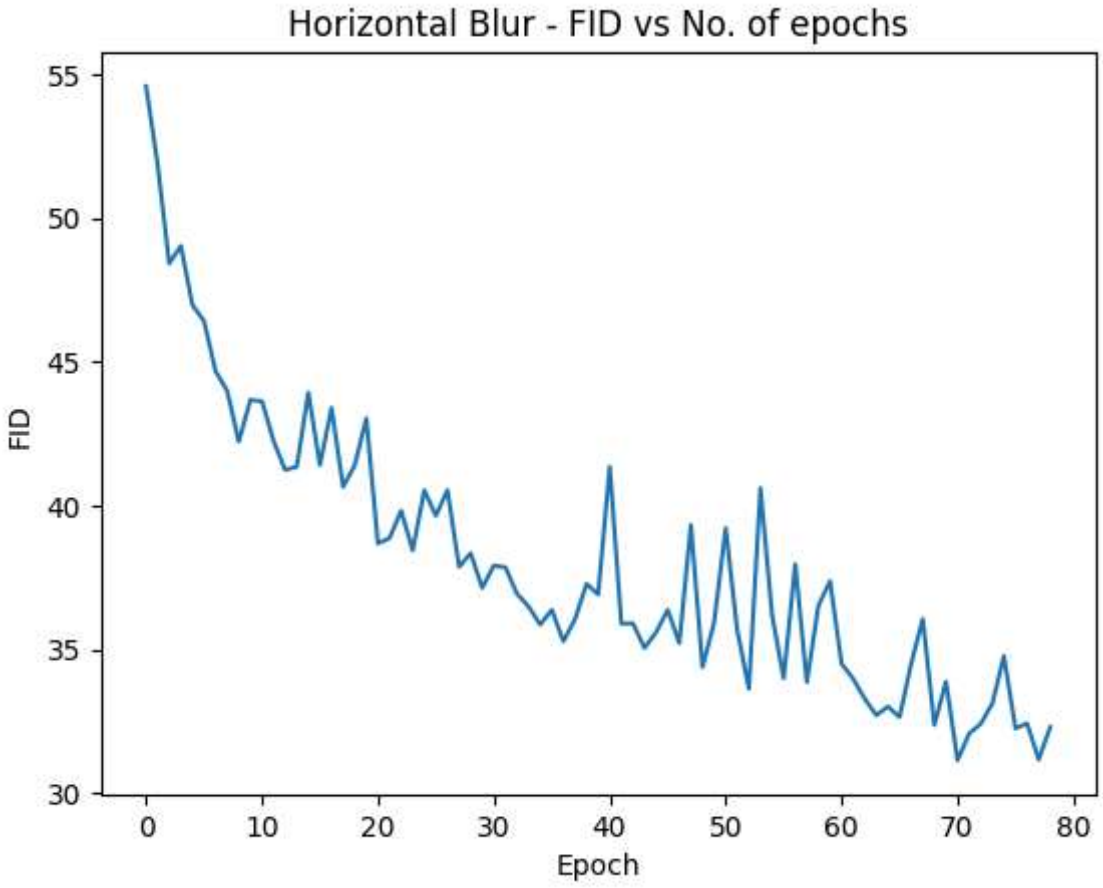

In [40]:
```
def plot_psnrs(history):
    psnrs = [x for x in history['psnrs']]
    plt.plot(psnrs)
    plt.xlabel('Epoch')
    plt.ylabel('PSNR')
    plt.title('Low Light Blur - PSNR vs No. of epochs')

plot_psnrs(checkpoint)
```




In [41]:
```
def plot_ssims(history):
    psnrs = [x for x in history['ssims']]
    plt.plot(psnrs)
    plt.xlabel('Epoch')
    plt.ylabel('SSIM')
    plt.title('Low Light Blur - SSIM vs No. of epochs')

plot_ssims(checkpoint)
```




In [30]:
```
def plot_fids(history):
    psnrs = [x for x in history['fids']]
    plt.plot(psnrs)
    plt.xlabel('Epoch')
    plt.ylabel('FID')
    plt.title('Horizontal Blur - FID vs No. of epochs')

plot_fids(checkpoint)
```


Horizontal Blur - FID vs No. of epochs
FID
Epoch
55
50
45
40
35
30
0
10
20
30
40
50
60
70
80

In [ ]: