In [1]:
```
import os
import sys
from tqdm import tqdm
import numpy as np
import torch
import torch.nn as nn
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 [4]:
```
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 [35]:
```
train_ds = PairedImageDataset(blur_dir=os.path.join(DATA_DIR, 'low_qual', 'train',
                              normal_dir=os.path.join(DATA_DIR, 'low_qual', 'train'
                              transform=train_transform)
```

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

test_ds = ImageFolder(os.path.join(DATA_DIR, 'low_qual', 'test'), transform=valid_t
```

In [36]:
```
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=
```

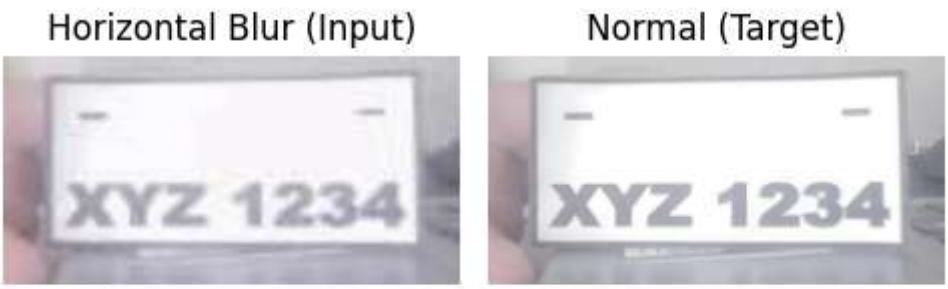
Helper functions

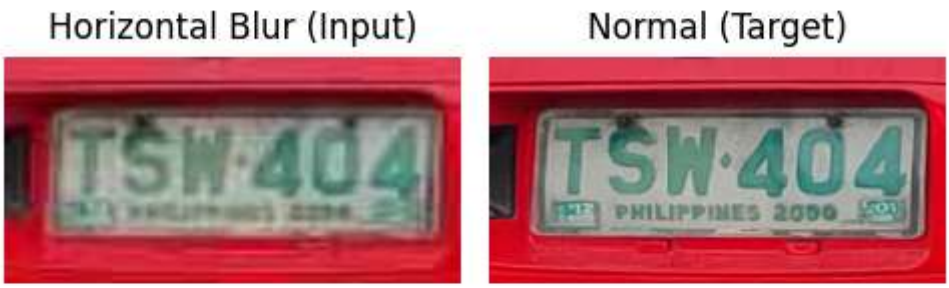
In [7]:
```
def denorm(img_tensors):
    return img_tensors * stats[1][0] + stats[0][0]

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 [8]:
```
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 [9]:
```
show_paired_samples(train_ds)
```





In [10]:
```
show_batch(valid_dl, nmax=14)
```

In [11]:
```
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 [12]:
```
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 [13]:
```
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 [14]:
```
device = get_default_device()
device
```

Out[14]:
```
device(type='cuda')
```

In [15]:
```
train_dl = DeviceDataLoader(train_dl, device)
valid_dl = DeviceDataLoader(valid_dl, device)
test_dl = DeviceDataLoader(test_dl, device)
```

Building the Model

Generator

In [16]:
```
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 [17]:
```
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 [18]:
```
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 [19]:
```
image1 = torch.rand((1, 3, 128, 256))

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

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

Model Parameters

In [20]:
```
total_params = sum(p.numel() for p in discriminator.parameters())
total_params
```

Out[20]: 2766529

In [21]:
```
discriminator = to_device(discriminator, device)
generator = to_device(generator, device)
```

Training

Loss Functions

In [22]:
```
comparison_loss = nn.BCEWithLogitsLoss()
L1_loss_fn = nn.L1Loss()
```

PaddleOCR

In [23]:
```
ocr_model = PaddleOCR(use_angle_cls=False, lang='en', det=False, rec=True, use_gpu=

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 07:00:04] ppocr DEBUG: Namespace(help='==SUPPRESS==', use_gpu=True, use_
xpu=False, use_npu=False, use_mlu=False, use_gcu=False, ir_optim=True, use_tensorrt=
False, min_subgraph_size=15, precision='fp32', gpu_mem=500, gpu_id=0, image_dir=Non
e, page_num=0, det_algorithm='DB', det_model_dir='C:\\Users\\ADMIN/.paddleocr/whl\\d
et\\en\\en_PP-OCRv3_det_infer', det_limit_side_len=960, det_limit_type='max', det_bo
x_type='quad', det_db_thresh=0.3, det_db_box_thresh=0.6, det_db_unclip_ratio=1.5, ma
x_batch_size=10, use_dilation=False, det_db_score_mode='fast', det_east_score_thresh
=0.8, det_east_cover_thresh=0.1, det_east_nms_thresh=0.2, det_sast_score_thresh=0.5,
det_sast_nms_thresh=0.2, det_pse_thresh=0, det_pse_box_thresh=0.85, det_pse_min_area
=16, det_pse_scale=1, scales=[8, 16, 32], alpha=1.0, beta=1.0, fourier_degree=5, rec
_algorithm='SVTR_LCNet', rec_model_dir='C:\\Users\\ADMIN/.paddleocr/whl\\rec\\en\\en
_PP-OCRv4_rec_infer', rec_image_inverse=True, rec_image_shape='3, 48, 320', rec_batc
h_num=6, max_text_length=25, rec_char_dict_path='C:\\Thesis\\LiPAD with Paddle\\lipa
dvenv3.9\\lib\\site-packages\\paddleocr\\ppocr\\utils\\en_dict.txt', use_space_char=
True, vis_font_path='./doc/fonts/simfang.ttf', drop_score=0.5, e2e_algorithm='PGNe
t', e2e_model_dir=None, e2e_limit_side_len=768, e2e_limit_type='max', e2e_pgnet_scor
e_thresh=0.5, e2e_char_dict_path='./ppocr/utils/ic15_dict.txt', e2e_pgnet_valid_set
='totaltext', e2e_pgnet_mode='fast', use_angle_cls=False, cls_model_dir='C:\\Users
\\ADMIN/.paddleocr/whl\\cls\\ch_ppocr_mobile_v2.0_cls_infer', cls_image_shape='3, 4
8, 192', label_list=['0', '180'], cls_batch_num=6, cls_thresh=0.9, enable_mkldnn=Fal
se, cpu_threads=10, use_pdserving=False, warmup=False, sr_model_dir=None, sr_image_s
hape='3, 32, 128', sr_batch_num=1, draw_img_save_dir='./inference_results', save_cro
p_res=False, crop_res_save_dir='./output', use_mp=False, total_process_num=1, proces
s_id=0, benchmark=False, save_log_path='./log_output/', show_log=True, use_onnx=Fals
e, onnx_providers=False, onnx_sess_options=False, return_word_box=False, output='./o
utput', table_max_len=488, table_algorithm='TableAttn', table_model_dir=None, merge_
no_span_structure=True, table_char_dict_path=None, formula_algorithm='LaTeXOCR', for
mula_model_dir=None, formula_char_dict_path=None, formula_batch_num=1, layout_model_
dir=None, layout_dict_path=None, layout_score_threshold=0.5, layout_nms_threshold=0.
5, kie_algorithm='LayoutXLM', ser_model_dir=None, re_model_dir=None, use_visual_back
bone=True, ser_dict_path='../train_data/XFUND/class_list_xfun.txt', ocr_order_method
=None, mode='structure', image_orientation=False, layout=True, table=True, formula=F
alse, ocr=True, recovery=False, recovery_to_markdown=False, use_pdf2docx_api=False,
invert=False, binarize=False, alphacolor=(255, 255, 255), lang='en', det=False, rec=
True, type='ocr', savefile=False, ocr_version='PP-OCRv4', structure_version='PP-Stru
ctureV2')
[2025/09/01 07:00:04] ppocr WARNING: The first GPU is used for inference by default,
GPU ID: 0
[2025/09/01 07:00:05] ppocr WARNING: The first GPU is used for inference by default,
GPU ID: 0
```

Discriminator Training Function

In [24]:

```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 [25]:

```
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, det=False)[0]
        real_text = ocr_model.ocr(real_np, cls=False, det=False)[0]

    fake_str = fake_text[0][0] if fake_text else ''
    real_str = real_text[0][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 [26]:
```
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 [27]:
```
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 [28]:

```
@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 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 = []

     # 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'])

    # 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]generator-Datasetv2-{epoc
            torch.save(discriminator.state_dict(), f'[RAU-NET]discriminator-Dataset
            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,
                'epoch': epoch + 1
            }, f'[RAU-NET]Checkpoint-Logs-Datasetv2-{epoch + 1} epoch.pt')

        # Print progress
        try:
            print('Epoch [{}/{}], loss_g: {:.4f}, loss_d: {:.4f}, real_score: {:.4f
```

```
                    epoch+1, epochs, g_loss, d_loss, real_score, fake_score))
        except Exception as err:
            print('Error:', str(err))

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

        # 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)

        save_samples(generator, test_dl, epoch + 1, sample_dir='./visualization')

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

Hyperparameter Configurations

In [29]:
```
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 = 100
```

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

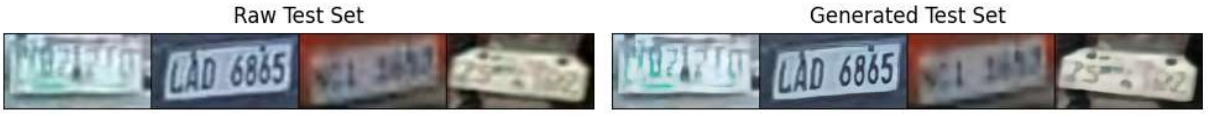
In [31]:
```
history = []
```

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

Epoch [1/100], loss_g: 13.7348, loss_d: 0.1594, real_score: 0.9313, fake_score: 0.21
12
Validation Data

Epoch [1/100] - Mean PSNR: 15.0363, Mean SSIM: 0.6222, FID: 88.5737075805664



100%|
1250/1250 [19:07<00:00, 1.09it/s]
Training Data

Epoch [2/100], loss_g: 15.3838, loss_d: 0.9125, real_score: 0.9743, fake_score: 0.8155

Validation Data

Epoch [2/100] - Mean PSNR: 15.5397, Mean SSIM: 0.6449, FID: 75.14977264404297

Raw Test Set

Generated Test Set

100%|

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

Training Data

Epoch [3/100], loss_g: 18.3615, loss_d: 0.3157, real_score: 0.9976, fake_score: 0.4491

Validation Data

Epoch [3/100] - Mean PSNR: 15.3667, Mean SSIM: 0.6459, FID: 72.73918914794922

Raw Test Set

Generated Test Set

100%|
1250/1250 [18:31<00:00, 1.12it/s]

Training Data

Epoch [4/100], loss_g: 18.2271, loss_d: 0.9220, real_score: 0.5310, fake_score: 0.67
05
Validation Data

Epoch [4/100] - Mean PSNR: 15.9021, Mean SSIM: 0.6682, FID: 68.24333190917969

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [5/100], loss_g: 11.9212, loss_d: 0.1475, real_score: 0.8396, fake_score: 0.10
92
Validation Data

Epoch [5/100] - Mean PSNR: 16.1336, Mean SSIM: 0.6742, FID: 65.99771881103516

100%|
1250/1250 [18:35<00:00, 1.12it/s]
Training Data

Epoch [6/100], loss_g: 24.8820, loss_d: 0.2865, real_score: 0.9985, fake_score: 0.42
07
Validation Data

Epoch [6/100] - Mean PSNR: 16.1009, Mean SSIM: 0.6767, FID: 66.40089416503906

100%|
1250/1250 [19:30<00:00, 1.07it/s]
Training Data

Epoch [7/100], loss_g: 13.5259, loss_d: 0.1281, real_score: 0.9729, fake_score: 0.2005
Validation Data

Epoch [7/100] - Mean PSNR: 16.3089, Mean SSIM: 0.6765, FID: 64.02198791503906

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [8/100], loss_g: 12.2336, loss_d: 0.5234, real_score: 0.9916, fake_score: 0.6238
Validation Data

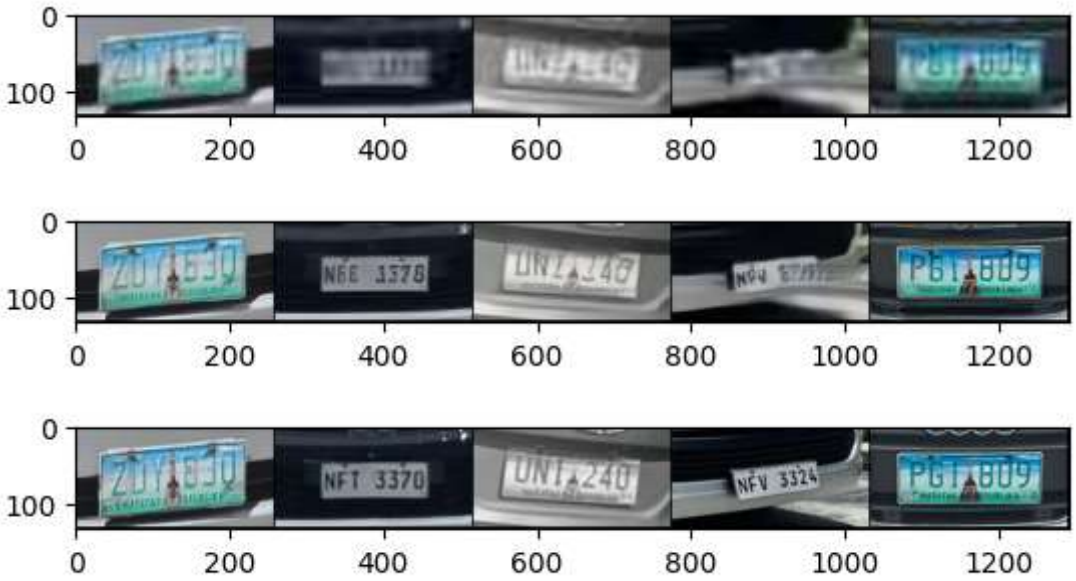
Epoch [8/100] - Mean PSNR: 16.3767, Mean SSIM: 0.6826, FID: 61.568870544433594

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [9/100], loss_g: 23.5805, loss_d: 0.3780, real_score: 0.5339, fake_score: 0.0879
Validation Data

Epoch [9/100] - Mean PSNR: 16.4338, Mean SSIM: 0.6862, FID: 62.23197555541992

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data



Epoch [10/100], loss_g: 17.9833, loss_d: 0.5120, real_score: 0.9517, fake_score: 0.5866
Validation Data

Epoch [10/100] - Mean PSNR: 16.3268, Mean SSIM: 0.6792, FID: 60.522098541259766

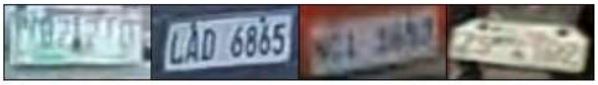
100%|
1250/1250 [18:59<00:00, 1.10it/s]
Training Data

Epoch [11/100], loss_g: 13.2169, loss_d: 0.3397, real_score: 0.8349, fake_score: 0.3814

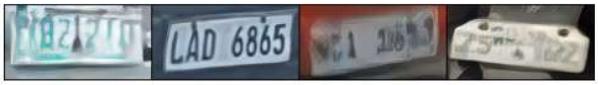
Validation Data

Epoch [11/100] - Mean PSNR: 16.3676, Mean SSIM: 0.6774, FID: 59.345088958740234

Raw Test Set



Generated Test Set



100%|
1250/1250 [18:32<00:00, 1.12it/s]

Training Data

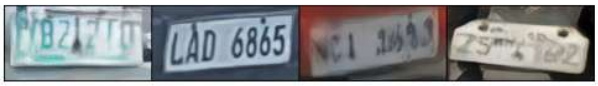
Epoch [12/100], loss_g: 13.1601, loss_d: 0.5280, real_score: 0.9245, fake_score: 0.6032

Validation Data

Epoch [12/100] - Mean PSNR: 16.6376, Mean SSIM: 0.6876, FID: 57.72936248779297

Raw Test Set

Generated Test Set



100%|

1250/1250 [18:36<00:00, 1.12it/s]

Training Data

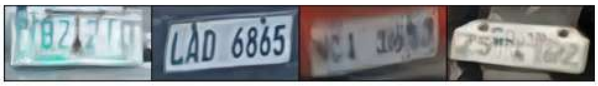
Epoch [13/100], loss_g: 10.1245, loss_d: 0.7096, real_score: 0.4820, fake_score: 0.4654

Validation Data

Epoch [13/100] - Mean PSNR: 16.4892, Mean SSIM: 0.6833, FID: 58.29381561279297

Raw Test Set

Generated Test Set



100%|
1250/1250 [18:31<00:00, 1.12it/s]

Training Data

Epoch [14/100], loss_g: 18.7539, loss_d: 0.6209, real_score: 0.9795, fake_score: 0.6467
Validation Data

Epoch [14/100] - Mean PSNR: 16.7355, Mean SSIM: 0.6924, FID: 57.52530288696289
Raw Test Set Generated Test Set

100%|
1250/1250 [18:28<00:00, 1.13it/s]
Training Data

Epoch [15/100], loss_g: 10.0308, loss_d: 0.3355, real_score: 0.7233, fake_score: 0.2732
Validation Data

Epoch [15/100] - Mean PSNR: 16.7468, Mean SSIM: 0.6896, FID: 55.776702880859375

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [16/100], loss_g: 8.8388, loss_d: 0.5859, real_score: 0.4009, fake_score: 0.1757
Validation Data

Epoch [16/100] - Mean PSNR: 16.5479, Mean SSIM: 0.6874, FID: 56.621944427490234

100%|
1250/1250 [18:33<00:00, 1.12it/s]
Training Data

Epoch [17/100], loss_g: 9.1669, loss_d: 0.3975, real_score: 0.5479, fake_score: 0.1433
Validation Data

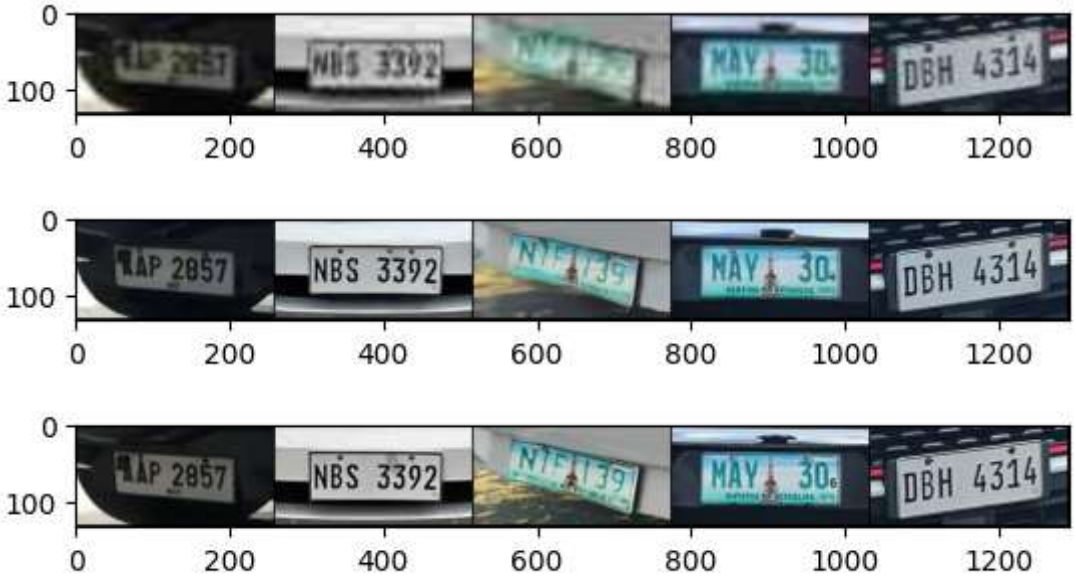
Epoch [17/100] - Mean PSNR: 16.5876, Mean SSIM: 0.6920, FID: 56.302650451660156

100%|
1250/1250 [18:29<00:00, 1.13it/s]
Training Data

Epoch [18/100], loss_g: 10.5401, loss_d: 0.8478, real_score: 0.9960, fake_score: 0.7920
Validation Data

Epoch [18/100] - Mean PSNR: 16.6729, Mean SSIM: 0.6938, FID: 58.11103439331055

100%|
1250/1250 [18:29<00:00, 1.13it/s]
Training Data



Epoch [19/100], loss_g: 11.3039, loss_d: 0.4145, real_score: 0.7972, fake_score: 0.4311
Validation Data

Epoch [19/100] - Mean PSNR: 16.5853, Mean SSIM: 0.6894, FID: 55.42158889770508

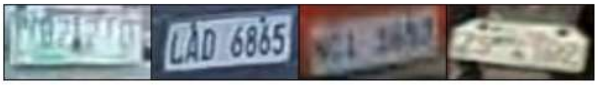
100%|
1250/1250 [18:30<00:00, 1.13it/s]
Training Data

Epoch [20/100], loss_g: 10.0287, loss_d: 0.3361, real_score: 0.6721, fake_score: 0.2093

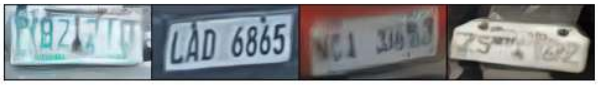
Validation Data

Epoch [20/100] - Mean PSNR: 16.6502, Mean SSIM: 0.6918, FID: 56.36728286743164

Raw Test Set



Generated Test Set

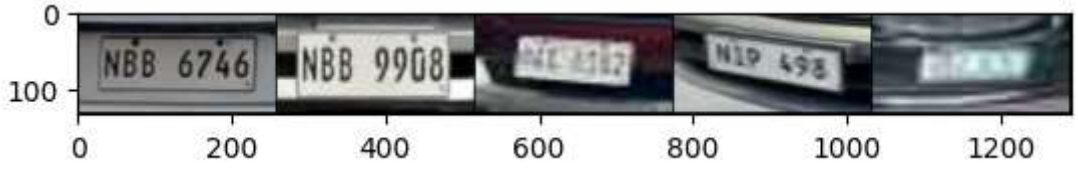


100%|
1250/1250 [18:31<00:00, 1.12it/s]

Training Data

Epoch [21/100], loss_g: 8.9730, loss_d: 1.3297, real_score: 0.0828, fake_score: 0.0494

Validation Data



Epoch [21/100] - Mean PSNR: 16.5212, Mean SSIM: 0.6864, FID: 54.23642349243164

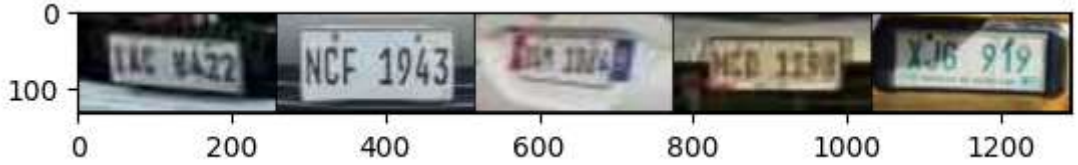
Raw Test Set

Generated Test Set

100%|

1250/1250 [18:31<00:00, 1.12it/s]

Training Data



Epoch [22/100], loss_g: 9.5040, loss_d: 0.2122, real_score: 0.9590, fake_score: 0.3090

Validation Data

Epoch [22/100] - Mean PSNR: 16.6216, Mean SSIM: 0.6889, FID: 54.598609924316406

Raw Test Set

Generated Test Set

100%|

1250/1250 [18:31<00:00, 1.12it/s]

Training Data

Epoch [23/100], loss_g: 12.8219, loss_d: 0.3694, real_score: 0.6633, fake_score: 0.2509
Validation Data

Epoch [23/100] - Mean PSNR: 16.7477, Mean SSIM: 0.6956, FID: 54.611846923828125

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [24/100], loss_g: 9.1771, loss_d: 0.4260, real_score: 0.6578, fake_score: 0.3227
Validation Data

Epoch [24/100] - Mean PSNR: 16.9083, Mean SSIM: 0.6952, FID: 53.58484649658203

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [25/100], loss_g: 8.3415, loss_d: 0.5109, real_score: 0.4099, fake_score: 0.0573
Validation Data

Epoch [25/100] - Mean PSNR: 16.7812, Mean SSIM: 0.6967, FID: 53.72076416015625

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [26/100], loss_g: 9.3117, loss_d: 1.8837, real_score: 0.0285, fake_score: 0.0254
Validation Data

Epoch [26/100] - Mean PSNR: 16.8415, Mean SSIM: 0.6935, FID: 52.91928482055664

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [27/100], loss_g: 8.1864, loss_d: 0.2910, real_score: 0.7740, fake_score: 0.2595
Validation Data

Epoch [27/100] - Mean PSNR: 16.6552, Mean SSIM: 0.6911, FID: 53.405426025390625

100%|
1250/1250 [18:30<00:00, 1.13it/s]
Training Data

Epoch [28/100], loss_g: 17.6641, loss_d: 0.4884, real_score: 0.5078, fake_score: 0.2155
Validation Data

Epoch [28/100] - Mean PSNR: 16.7275, Mean SSIM: 0.6922, FID: 52.65266799926758

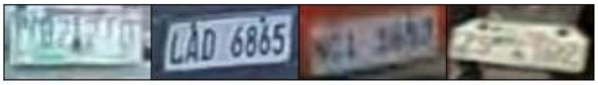
100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [29/100], loss_g: 18.3687, loss_d: 0.3072, real_score: 0.7517, fake_score: 0.2611

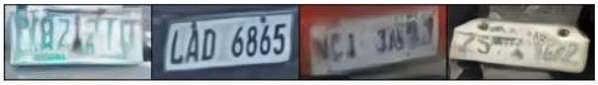
Validation Data

Epoch [29/100] - Mean PSNR: 16.8364, Mean SSIM: 0.6956, FID: 53.03778076171875

Raw Test Set



Generated Test Set



100%|██████████| 1250/1250 [18:31<00:00, 1.12it/s]

Training Data

Epoch [30/100], loss_g: 9.6385, loss_d: 0.3989, real_score: 0.7713, fake_score: 0.3899

Validation Data

Epoch [30/100] - Mean PSNR: 16.6369, Mean SSIM: 0.6930, FID: 52.135868072509766

Raw Test Set

Generated Test Set

100%|
1250/1250 [18:31<00:00, 1.12it/s]

Training Data

Epoch [31/100], loss_g: 9.6087, loss_d: 0.9785, real_score: 0.1785, fake_score: 0.0780

Validation Data

Epoch [31/100] - Mean PSNR: 16.7697, Mean SSIM: 0.6900, FID: 51.67242431640625

Raw Test Set

Generated Test Set

100%|

1250/1250 [18:30<00:00, 1.13it/s]

Training Data

Epoch [32/100], loss_g: 13.5159, loss_d: 0.2031, real_score: 0.8943, fake_score: 0.2428
Validation Data

Epoch [32/100] - Mean PSNR: 16.7448, Mean SSIM: 0.6914, FID: 52.12015151977539

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [33/100], loss_g: 15.4354, loss_d: 0.3949, real_score: 0.6614, fake_score: 0.2810
Validation Data

Epoch [33/100] - Mean PSNR: 16.5437, Mean SSIM: 0.6859, FID: 51.8897819519043

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [34/100], loss_g: 7.8392, loss_d: 0.4212, real_score: 0.7885, fake_score: 0.4213
Validation Data

Epoch [34/100] - Mean PSNR: 16.7450, Mean SSIM: 0.6930, FID: 52.21112060546875

100%|
1250/1250 [18:31<00:00, 1.12it/s]
Training Data

Epoch [35/100], loss_g: 6.5918, loss_d: 0.6630, real_score: 0.3682, fake_score: 0.2056
Validation Data

Epoch [35/100] - Mean PSNR: 16.7751, Mean SSIM: 0.6902, FID: 51.03852462768555

100%|
1250/1250 [19:08<00:00, 1.09it/s]
Training Data

Epoch [36/100], loss_g: 7.3100, loss_d: 0.3983, real_score: 0.5890, fake_score: 0.1924
Validation Data

Epoch [36/100] - Mean PSNR: 16.8161, Mean SSIM: 0.6932, FID: 52.11528778076172

100%|
1250/1250 [18:32<00:00, 1.12it/s]
Training Data

Epoch [37/100], loss_g: 8.1810, loss_d: 0.3126, real_score: 0.9423, fake_score: 0.4035
Validation Data

Epoch [37/100] - Mean PSNR: 16.6537, Mean SSIM: 0.6902, FID: 51.098419189453125

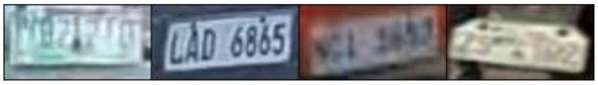
100%|
1250/1250 [18:28<00:00, 1.13it/s]
Training Data

Epoch [38/100], loss_g: 8.9853, loss_d: 0.4928, real_score: 0.5269, fake_score: 0.2444

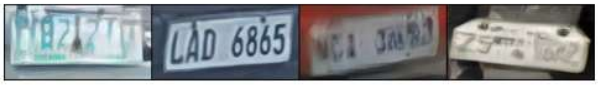
Validation Data

Epoch [38/100] - Mean PSNR: 16.8308, Mean SSIM: 0.6937, FID: 51.71621322631836

Raw Test Set



Generated Test Set



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

Training Data

Epoch [39/100], loss_g: 8.7576, loss_d: 0.4374, real_score: 0.4941, fake_score: 0.0986

Validation Data

Epoch [39/100] - Mean PSNR: 16.6123, Mean SSIM: 0.6885, FID: 51.20871353149414

Raw Test Set

Generated Test Set

100%|
1250/1250 [19:02<00:00, 1.09it/s]

Training Data

Epoch [40/100], loss_g: 9.0074, loss_d: 0.5024, real_score: 0.5204, fake_score: 0.2398

Validation Data

Epoch [40/100] - Mean PSNR: 16.7455, Mean SSIM: 0.6945, FID: 50.74522399902344

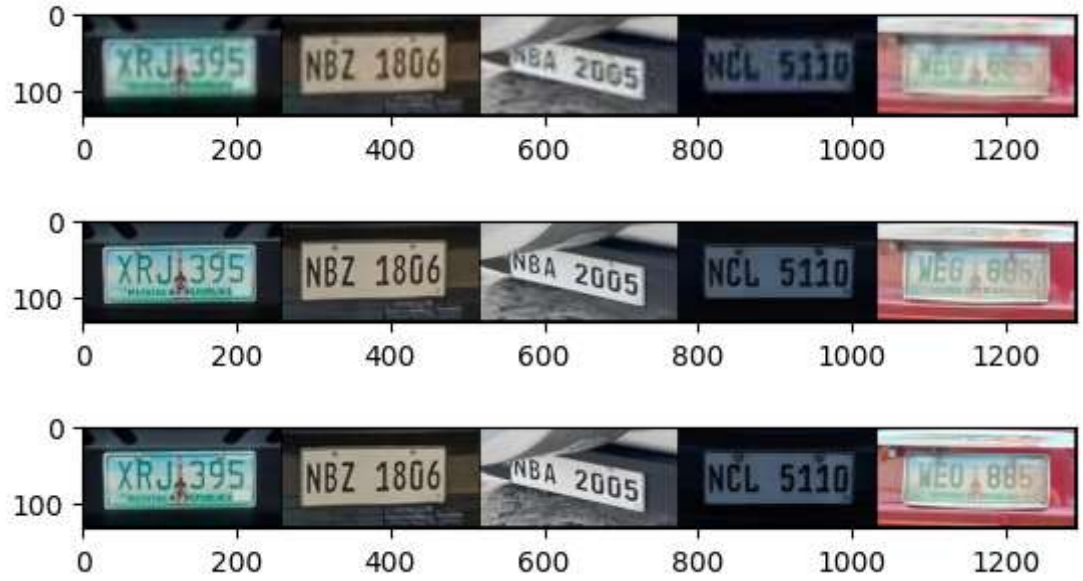
Raw Test Set

Generated Test Set

100%|██████████| 1250/1250 [19:28<00:00, 1.07it/s]

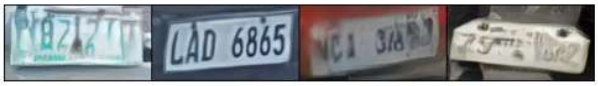
Training Data

Epoch [41/100], loss_g: 14.7173, loss_d: 0.2921, real_score: 0.6945, fake_score: 0.1692
Validation Data



Epoch [41/100] - Mean PSNR: 16.7378, Mean SSIM: 0.6915, FID: 51.01935958862305
Raw Test Set

Generated Test Set



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

Epoch [42/100], loss_g: 20.9016, loss_d: 0.5313, real_score: 0.5407, fake_score: 0.3079
Validation Data

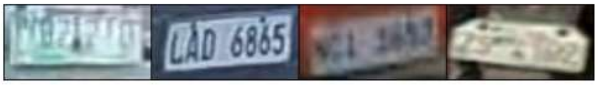
Epoch [42/100] - Mean PSNR: 16.8750, Mean SSIM: 0.6949, FID: 52.39639663696289

100%|
1250/1250 [19:26<00:00, 1.07it/s]
Training Data

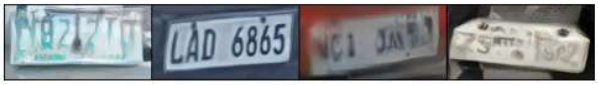
Epoch [43/100], loss_g: 13.8600, loss_d: 0.5109, real_score: 0.5384, fake_score: 0.2821
Validation Data

Epoch [43/100] - Mean PSNR: 16.8274, Mean SSIM: 0.6907, FID: 50.910499572753906

Raw Test Set



Generated Test Set



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

Epoch [44/100], loss_g: 13.3424, loss_d: 0.4471, real_score: 0.6344, fake_score: 0.3117
Validation Data

Epoch [44/100] - Mean PSNR: 16.6923, Mean SSIM: 0.6882, FID: 51.8835563659668

100%|
1250/1250 [18:32<00:00, 1.12it/s]
Training Data

Epoch [45/100], loss_g: 7.4127, loss_d: 1.0146, real_score: 0.1679, fake_score: 0.0858
Validation Data

Epoch [45/100] - Mean PSNR: 16.5438, Mean SSIM: 0.6854, FID: 51.01364517211914

100%|
1250/1250 [18:32<00:00, 1.12it/s]
Training Data

Epoch [46/100], loss_g: 7.7617, loss_d: 0.6117, real_score: 0.4447, fake_score: 0.2478
Validation Data

Epoch [46/100] - Mean PSNR: 16.6507, Mean SSIM: 0.6854, FID: 51.20905303955078

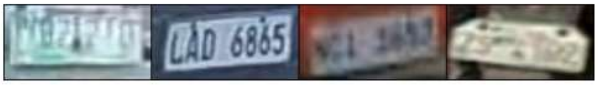
100%|
1250/1250 [18:33<00:00, 1.12it/s]
Training Data

Epoch [47/100], loss_g: 6.7227, loss_d: 0.5923, real_score: 0.8769, fake_score: 0.5978

Validation Data

Epoch [47/100] - Mean PSNR: 16.6516, Mean SSIM: 0.6883, FID: 50.21856689453125

Raw Test Set



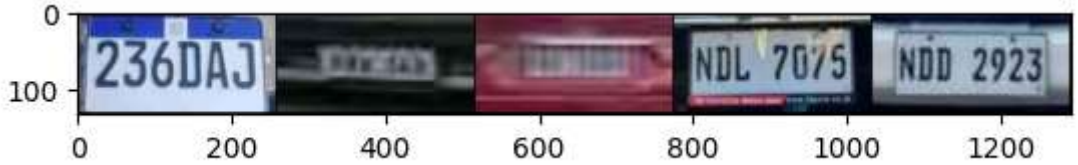
Generated Test Set

100%|
1250/1250 [18:33<00:00, 1.12it/s]

Training Data

Epoch [48/100], loss_g: 12.2718, loss_d: 0.2789, real_score: 0.7758, fake_score: 0.2349

Validation Data



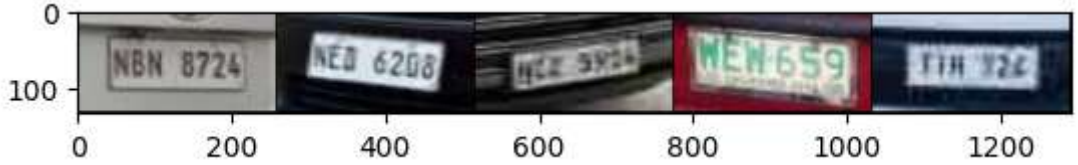
Epoch [48/100] - Mean PSNR: 16.7517, Mean SSIM: 0.6905, FID: 51.063053131103516

Raw Test Set

Generated Test Set

100%|
1250/1250 [18:32<00:00, 1.12it/s]

Training Data



Epoch [49/100], loss_g: 6.6962, loss_d: 0.4797, real_score: 0.4730, fake_score: 0.1105

Validation Data

Epoch [49/100] - Mean PSNR: 16.7504, Mean SSIM: 0.6915, FID: 51.55146789550781

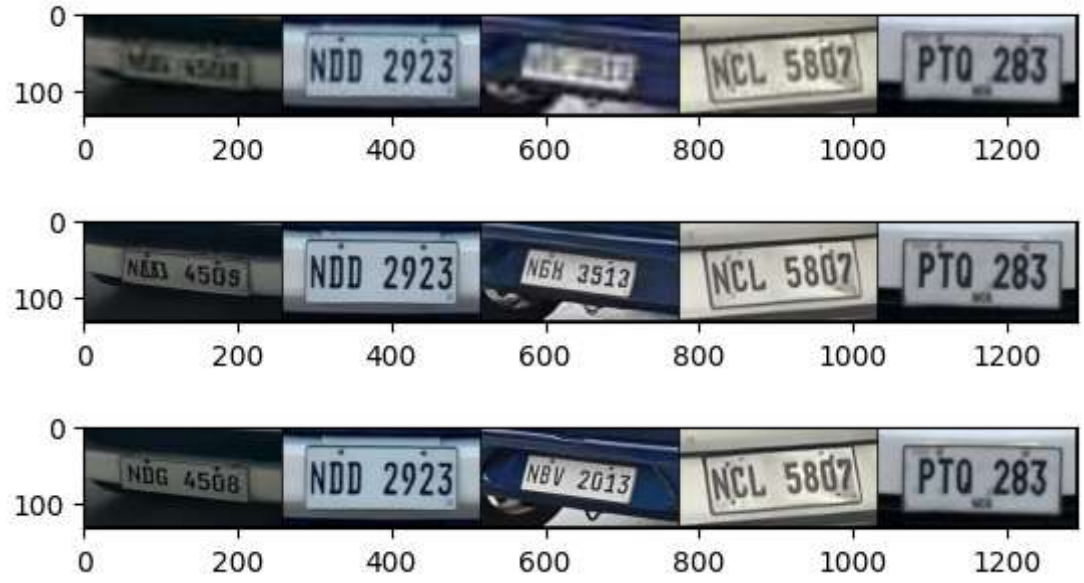
Raw Test Set

Generated Test Set

100%|
1250/1250 [18:34<00:00, 1.12it/s]

Training Data

Epoch [50/100], loss_g: 10.4136, loss_d: 0.4632, real_score: 0.7812, fake_score: 0.4484
Validation Data



Epoch [50/100] - Mean PSNR: 16.5840, Mean SSIM: 0.6856, FID: 50.91236877441406

100%|
1250/1250 [18:32<00:00, 1.12it/s]
Training Data

Epoch [51/100], loss_g: 17.5845, loss_d: 0.2439, real_score: 0.8509, fake_score: 0.2575
Validation Data

Epoch [51/100] - Mean PSNR: 16.7185, Mean SSIM: 0.6878, FID: 50.77513885498047

100%|
1250/1250 [18:43<00:00, 1.11it/s]
Training Data

Epoch [52/100], loss_g: 8.0554, loss_d: 0.4911, real_score: 0.6765, fake_score: 0.3862
Validation Data

Epoch [52/100] - Mean PSNR: 16.7433, Mean SSIM: 0.6878, FID: 50.79594039916992

100%|
1250/1250 [19:03<00:00, 1.09it/s]
Training Data

Epoch [53/100], loss_g: 16.7479, loss_d: 0.4256, real_score: 0.8745, fake_score: 0.4683
Validation Data

Epoch [53/100] - Mean PSNR: 16.9420, Mean SSIM: 0.6901, FID: 50.3238639831543

100%|
1250/1250 [18:37<00:00, 1.12it/s]
Training Data

Epoch [54/100], loss_g: 6.8816, loss_d: 0.4222, real_score: 0.7592, fake_score: 0.3785
Validation Data

Epoch [54/100] - Mean PSNR: 16.8547, Mean SSIM: 0.6906, FID: 51.44987869262695

100%|
1250/1250 [19:05<00:00, 1.09it/s]
Training Data

Epoch [55/100], loss_g: 8.0388, loss_d: 0.5639, real_score: 0.9150, fake_score: 0.5992
Validation Data

Epoch [55/100] - Mean PSNR: 16.6732, Mean SSIM: 0.6885, FID: 50.865169525146484

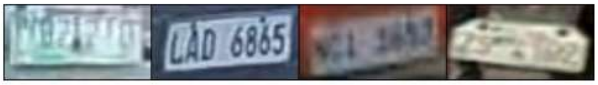
100%|
1250/1250 [19:13<00:00, 1.08it/s]
Training Data

Epoch [56/100], loss_g: 11.6357, loss_d: 0.2480, real_score: 0.9223, fake_score: 0.3124

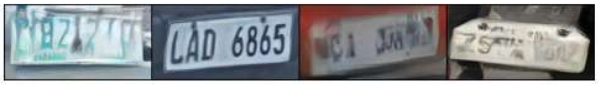
Validation Data

Epoch [56/100] - Mean PSNR: 16.7014, Mean SSIM: 0.6880, FID: 51.10187530517578

Raw Test Set



Generated Test Set



100%|
1250/1250 [18:36<00:00, 1.12it/s]

Training Data

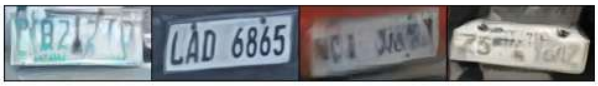
Epoch [57/100], loss_g: 16.5970, loss_d: 0.5661, real_score: 0.9274, fake_score: 0.6107

Validation Data

Epoch [57/100] - Mean PSNR: 16.7396, Mean SSIM: 0.6861, FID: 51.19881820678711

Raw Test Set

Generated Test Set



100%|
1250/1250 [18:31<00:00, 1.12it/s]

Training Data

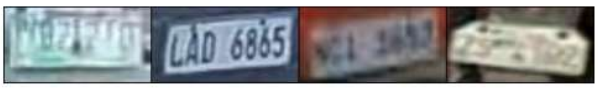

```
Epoch [58/100], loss_g: 7.0888, loss_d: 0.4762, real_score: 0.9241, fake_score: 0.53
65
Validation Data
```


```
Epoch [58/100] - Mean PSNR: 16.7776, Mean SSIM: 0.6880, FID: 51.152400970458984
```

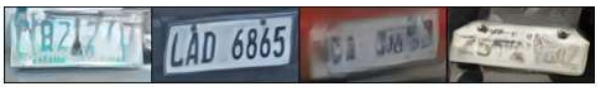

```
100%|██████████████████████████████████████████████████████████████████████████████|
1250/1250 [18:36<00:00,  1.12it/s]
Training Data
```

Epoch [59/100], loss_g: 14.6100, loss_d: 0.5146, real_score: 0.7219, fake_score: 0.4507
Validation Data

Epoch [59/100] - Mean PSNR: 16.5912, Mean SSIM: 0.6844, FID: 50.85931396484375
Raw Test Set



Generated Test Set



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

Epoch [60/100], loss_g: 6.2183, loss_d: 0.3801, real_score: 0.6578, fake_score: 0.2348
Validation Data

Epoch [60/100] - Mean PSNR: 16.6962, Mean SSIM: 0.6852, FID: 51.89915084838867

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

Epoch [61/100], loss_g: 27.1288, loss_d: 0.6676, real_score: 0.8767, fake_score: 0.6443
Validation Data

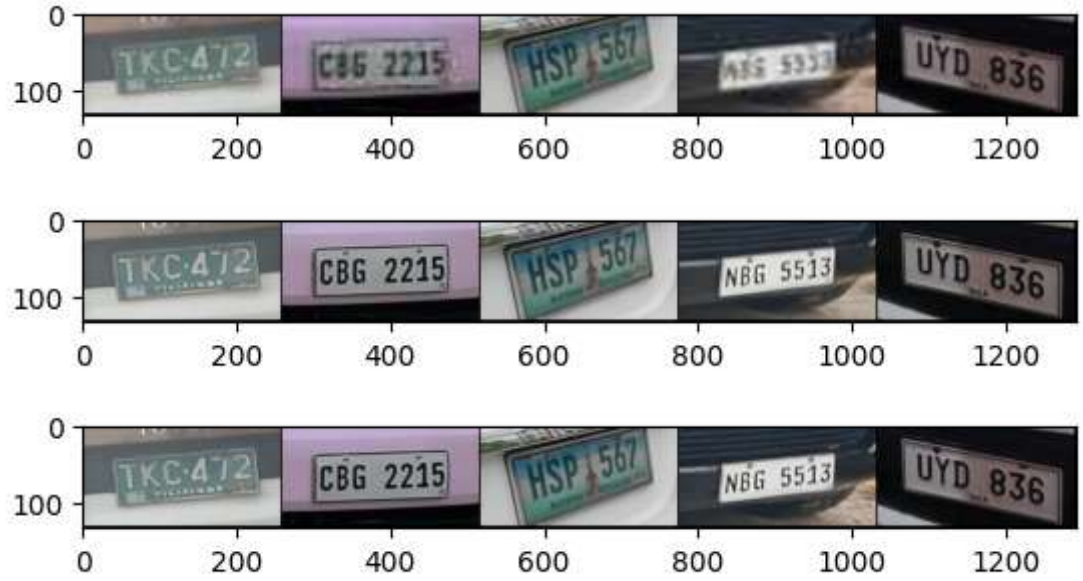
Epoch [61/100] - Mean PSNR: 16.5761, Mean SSIM: 0.6839, FID: 51.232513427734375

100%|
1250/1250 [18:35<00:00, 1.12it/s]
Training Data

Epoch [62/100], loss_g: 7.7187, loss_d: 0.6227, real_score: 0.3840, fake_score: 0.1522
Validation Data

Epoch [62/100] - Mean PSNR: 16.7867, Mean SSIM: 0.6875, FID: 52.131446838378906

100%|
1250/1250 [18:41<00:00, 1.11it/s]
Training Data



Epoch [63/100], loss_g: 9.8292, loss_d: 0.4396, real_score: 0.6724, fake_score: 0.3210
Validation Data

Epoch [63/100] - Mean PSNR: 16.7710, Mean SSIM: 0.6878, FID: 51.19508743286133

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

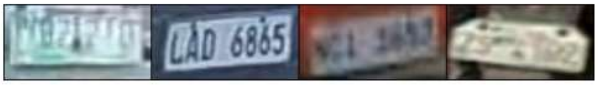
Epoch [64/100], loss_g: 7.8877, loss_d: 0.5136, real_score: 0.6005, fake_score: 0.3463
Validation Data

Epoch [64/100] - Mean PSNR: 16.8282, Mean SSIM: 0.6896, FID: 51.36892318725586

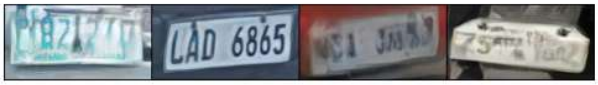
100%|
1250/1250 [19:58<00:00, 1.04it/s]
Training Data

Epoch [65/100], loss_g: 9.8715, loss_d: 0.5169, real_score: 0.4627, fake_score: 0.13
66
Validation Data

Epoch [65/100] - Mean PSNR: 16.7960, Mean SSIM: 0.6876, FID: 51.98481369018555
Raw Test Set



Generated Test Set



100%|
1250/1250 [21:03<00:00, 1.01s/it]
Training Data

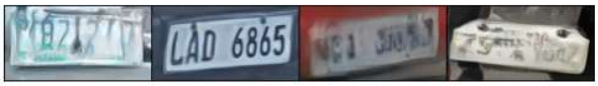
Epoch [66/100], loss_g: 11.7402, loss_d: 0.3562, real_score: 0.7679, fake_score: 0.3145

Validation Data

Epoch [66/100] - Mean PSNR: 16.8601, Mean SSIM: 0.6882, FID: 51.82242965698242

Raw Test Set

Generated Test Set



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

Training Data

Epoch [67/100], loss_g: 6.6485, loss_d: 0.4295, real_score: 0.8193, fake_score: 0.4205

Validation Data

Epoch [67/100] - Mean PSNR: 16.6254, Mean SSIM: 0.6850, FID: 50.46392822265625

Raw Test Set

Generated Test Set

100%|

1250/1250 [19:26<00:00, 1.07it/s]

Training Data

Epoch [68/100], loss_g: 14.4831, loss_d: 0.5181, real_score: 0.4964, fake_score: 0.2034
Validation Data

Epoch [68/100] - Mean PSNR: 16.5788, Mean SSIM: 0.6795, FID: 50.86103439331055

100%|
1250/1250 [18:44<00:00, 1.11it/s]
Training Data

Epoch [69/100], loss_g: 6.6487, loss_d: 0.3919, real_score: 0.8598, fake_score: 0.4204
Validation Data

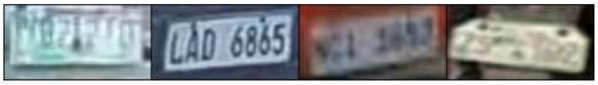
Epoch [69/100] - Mean PSNR: 16.7326, Mean SSIM: 0.6862, FID: 50.71609115600586

100%|
1250/1250 [18:45<00:00, 1.11it/s]
Training Data

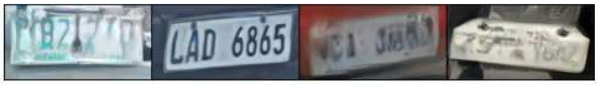
Epoch [70/100], loss_g: 8.0742, loss_d: 0.3884, real_score: 0.8281, fake_score: 0.3952
Validation Data

Epoch [70/100] - Mean PSNR: 16.7447, Mean SSIM: 0.6862, FID: 51.09672927856445

Raw Test Set



Generated Test Set



100%|
1250/1250 [18:42<00:00, 1.11it/s]
Training Data

Epoch [71/100], loss_g: 6.0035, loss_d: 0.4797, real_score: 0.6837, fake_score: 0.3714
Validation Data

Epoch [71/100] - Mean PSNR: 16.6312, Mean SSIM: 0.6865, FID: 50.84730529785156

100%|
1250/1250 [18:42<00:00, 1.11it/s]
Training Data

Epoch [72/100], loss_g: 6.3332, loss_d: 0.5572, real_score: 0.4581, fake_score: 0.1542
Validation Data

Epoch [72/100] - Mean PSNR: 16.6667, Mean SSIM: 0.6859, FID: 51.7897834777832

100%|
1250/1250 [18:43<00:00, 1.11it/s]
Training Data

Epoch [73/100], loss_g: 9.7908, loss_d: 0.3392, real_score: 0.8782, fake_score: 0.3789
Validation Data

Epoch [73/100] - Mean PSNR: 16.8446, Mean SSIM: 0.6913, FID: 51.14340591430664

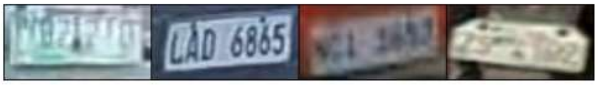
100%|
1250/1250 [19:10<00:00, 1.09it/s]
Training Data

Epoch [74/100], loss_g: 5.8312, loss_d: 0.7526, real_score: 0.2902, fake_score: 0.0525

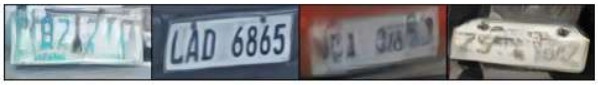
Validation Data

Epoch [74/100] - Mean PSNR: 16.7647, Mean SSIM: 0.6874, FID: 51.344581604003906

Raw Test Set



Generated Test Set



100%|

1250/1250 [19:09<00:00, 1.09it/s]

Training Data

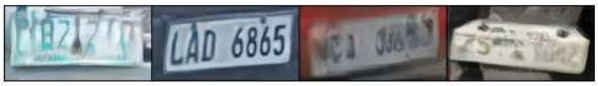
Epoch [75/100], loss_g: 6.3994, loss_d: 0.5533, real_score: 0.4785, fake_score: 0.2017

Validation Data

Epoch [75/100] - Mean PSNR: 16.8720, Mean SSIM: 0.6894, FID: 50.776729583740234

Raw Test Set

Generated Test Set



0%|
| 0/1250 [00:05<?, ?it/s]

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

Cell In[28], line 75, in fit(generator, discriminator, epochs, lr, adv_lambda, l1_la
mbda, ssim_lambda, perceptual_lambda, text_lambda, g_opt, d_opt, checkpoint)
     73 generator.train()
     74 discriminator.train()
---> 75 for inputs, targets in tqdm(train_dl):
     76     inputs = inputs.to(device)
     77     targets = targets.to(device)

File C:\Thesis\LiPAD with Paddle\lipadvenv3.9\lib\site-packages\tqdm\std.py:1181, in
tqdm.__iter__(self)
   1178 time = self._time
   1180 try:
-> 1181     for obj in iterable:
   1182         yield obj
   1183         # Update and possibly print the progressbar.
   1184         # Note: does not call self.update(1) for speed optimisation.

Cell In[13], line 18, in DeviceDataLoader.__iter__(self)
     17 def __iter__(self):
---> 18     for b in self.dl:
     19         yield to_device(b, self.device)

File C:\Thesis\LiPAD with Paddle\lipadvenv3.9\lib\site-packages\torch\utils\data\dat
aloader.py:441, in DataLoader.__iter__(self)
    439     return self._iterator
    440 else:
--> 441     return self._get_iterator()

File C:\Thesis\LiPAD with Paddle\lipadvenv3.9\lib\site-packages\torch\utils\data\dat
aloader.py:388, in DataLoader._get_iterator(self)
    386 else:
    387     self.check_worker_number_rationality()
--> 388     return _MultiProcessingDataLoaderIter(self)

File C:\Thesis\LiPAD with Paddle\lipadvenv3.9\lib\site-packages\torch\utils\data\dat
aloader.py:1042, in _MultiProcessingDataLoaderIter.__init__(self, loader)
   1035 w.daemon = True
   1036 # NB: Process.start() actually take some time as it needs to
   1037 #     start a process and pass the arguments over via a pipe.
   1038 #     Therefore, we only add a worker to self._workers list after
   1039 #     it started, so that we do not call .join() if program dies
   1040 #     before it starts, and __del__ tries to join but will get:
   1041 #     AssertionError: can only join a started process.
-> 1042 w.start()
   1043 self._index_queues.append(index_queue)
   1044 self._workers.append(w)

File ~\AppData\Local\Programs\Python\Python39\lib\multiprocessing\process.py:121, in
BaseProcess.start(self)
    118 assert not _current_process._config.get('daemon'), \
    119        'daemonic processes are not allowed to have children'
    120 _cleanup()
```

```
--> 121 self._popen = self._Popen(self)
    122 self._sentinel = self._popen.sentinel
    123 # Avoid a refcycle if the target function holds an indirect
    124 # reference to the process object (see bpo-30775)

File ~\AppData\Local\Programs\Python\Python39\lib\multiprocessing\context.py:224, in
Process._Popen(process_obj)
    222 @staticmethod
    223 def _Popen(process_obj):
--> 224     return _default_context.get_context().Process._Popen(process_obj)

File ~\AppData\Local\Programs\Python\Python39\lib\multiprocessing\context.py:327, in
SpawnProcess._Popen(process_obj)
    324 @staticmethod
    325 def _Popen(process_obj):
    326     from .popen_spawn_win32 import Popen
--> 327     return Popen(process_obj)

File ~\AppData\Local\Programs\Python\Python39\lib\multiprocessing\popen_spawn_win32.
py:93, in Popen.__init__(self, process_obj)
     91 try:
     92     reduction.dump(prep_data, to_child)
---> 93     reduction.dump(process_obj, to_child)
     94 finally:
     95     set_spawning_popen(None)

File ~\AppData\Local\Programs\Python\Python39\lib\multiprocessing\reduction.py:60, i
n dump(obj, file, protocol)
     58 def dump(obj, file, protocol=None):
     59     '''Replacement for pickle.dump() using ForkingPickler.'''
---> 60     ForkingPickler(file, protocol).dump(obj)

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 [31]:
```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 [32]:
```
deblur_image(os.path.join(DATA_DIR, 'low_qual', 'test', 'low_qual', '48.jpg'), gene
```

In [38]:
```
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 [36]:
```
torch.cuda.empty_cache()
```

Load Model

In [34]:
```
MODELS_DIR = os.path.join('..', '..', '..', 'models', 'h_blur')
```

In [37]:
```
generator = AttentionResUNetGenerator()
loaded_g_wts = torch.load('[RAU-NET]generator-Datasetv2-55 epoch.pt')
generator.load_state_dict(loaded_g_wts)
generator = to_device(generator, device)

discriminator = PatchDiscriminator()
loaded_d_wts = torch.load('[RAU-NET]discriminator-Datasetv2-55 epoch.pt')
discriminator.load_state_dict(loaded_d_wts)
discriminator = to_device(discriminator, device)
```

In [29]:
```
checkpoint = torch.load('[RAU-NET]Checkpoint-Logs-Datasetv2-75 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
```

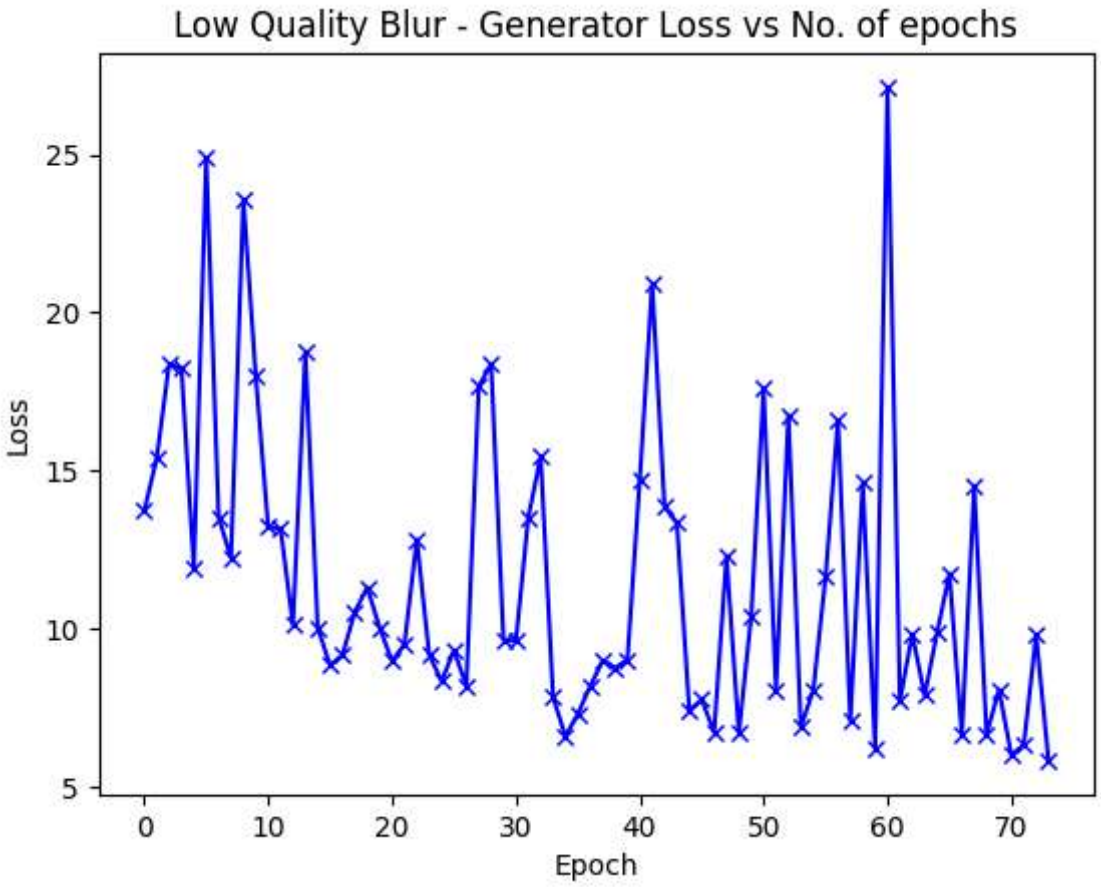
Save Reconstructed Test Set

In [36]:
```
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
```

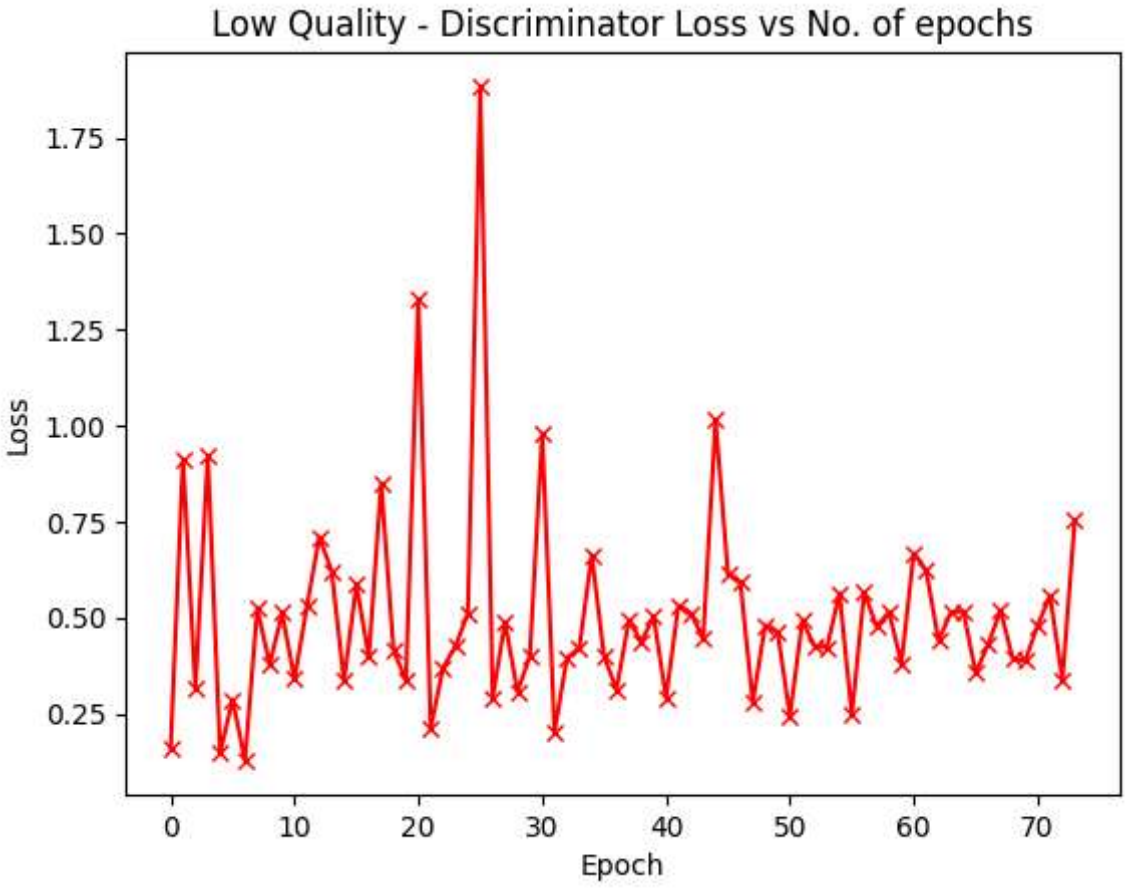
In [37]:
```
save_generated_tests(generator, test_dl)
```

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 Quality Blur - Generator Loss vs No. of epochs')
```

```
plot_g_losses(checkpoint)
```

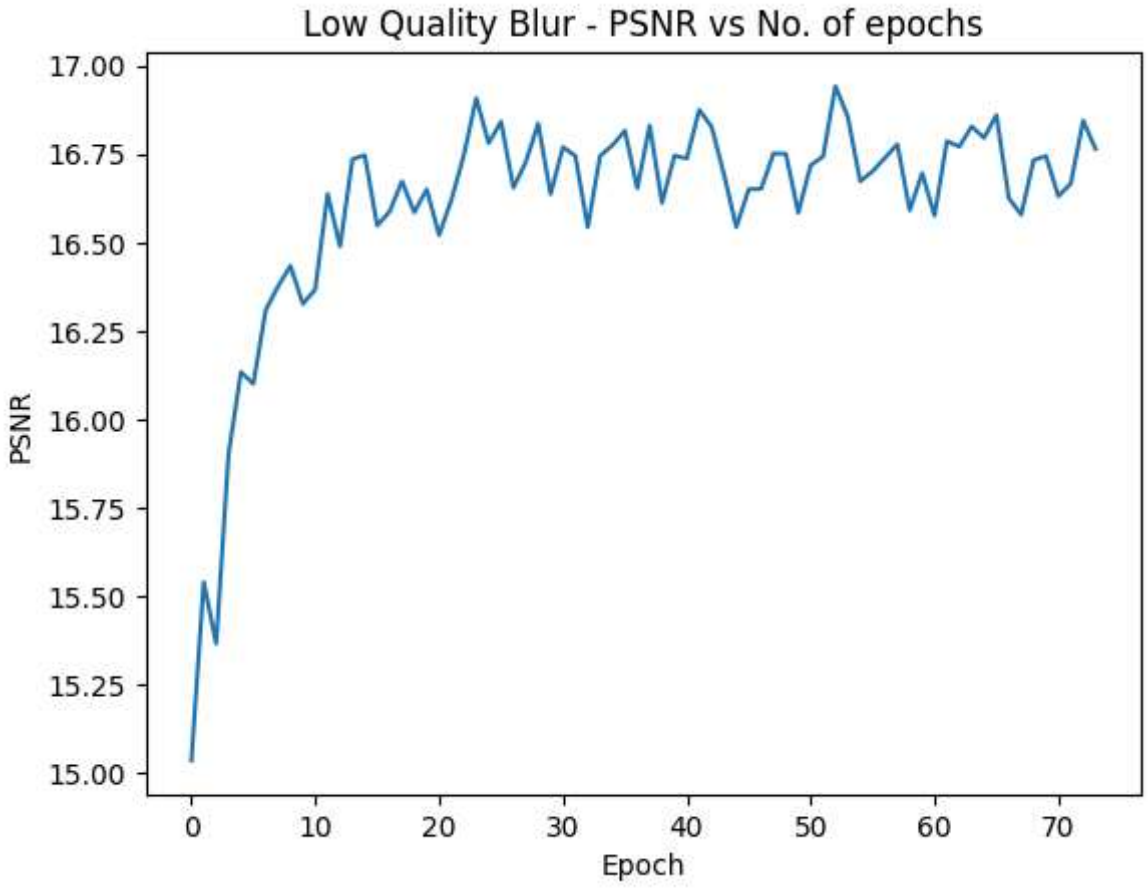


In [39]:
```
def plot_d_losses(history):
    losses_d = [x for x in history['losses_d']]
    plt.plot(losses_d, '-rx')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Low Quality - Discriminator Loss vs No. of epochs')

plot_d_losses(checkpoint)
```

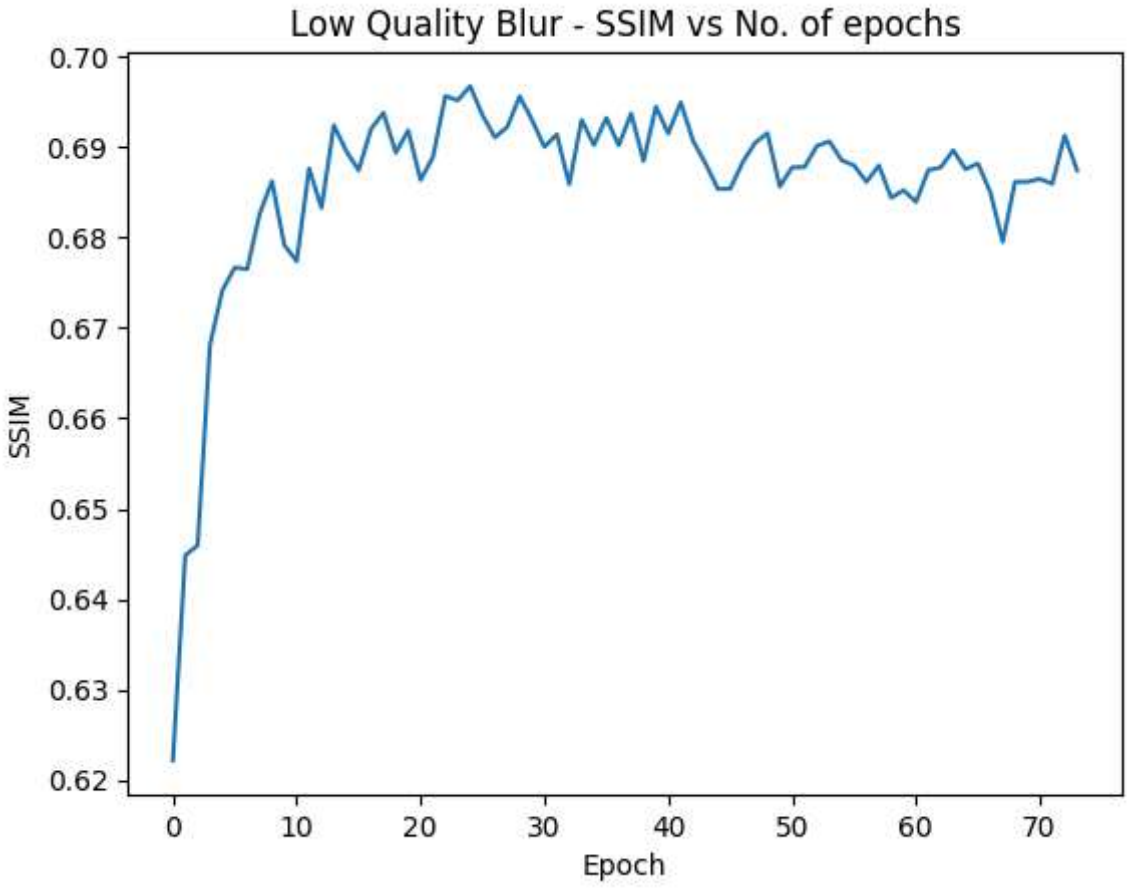


In [34]:

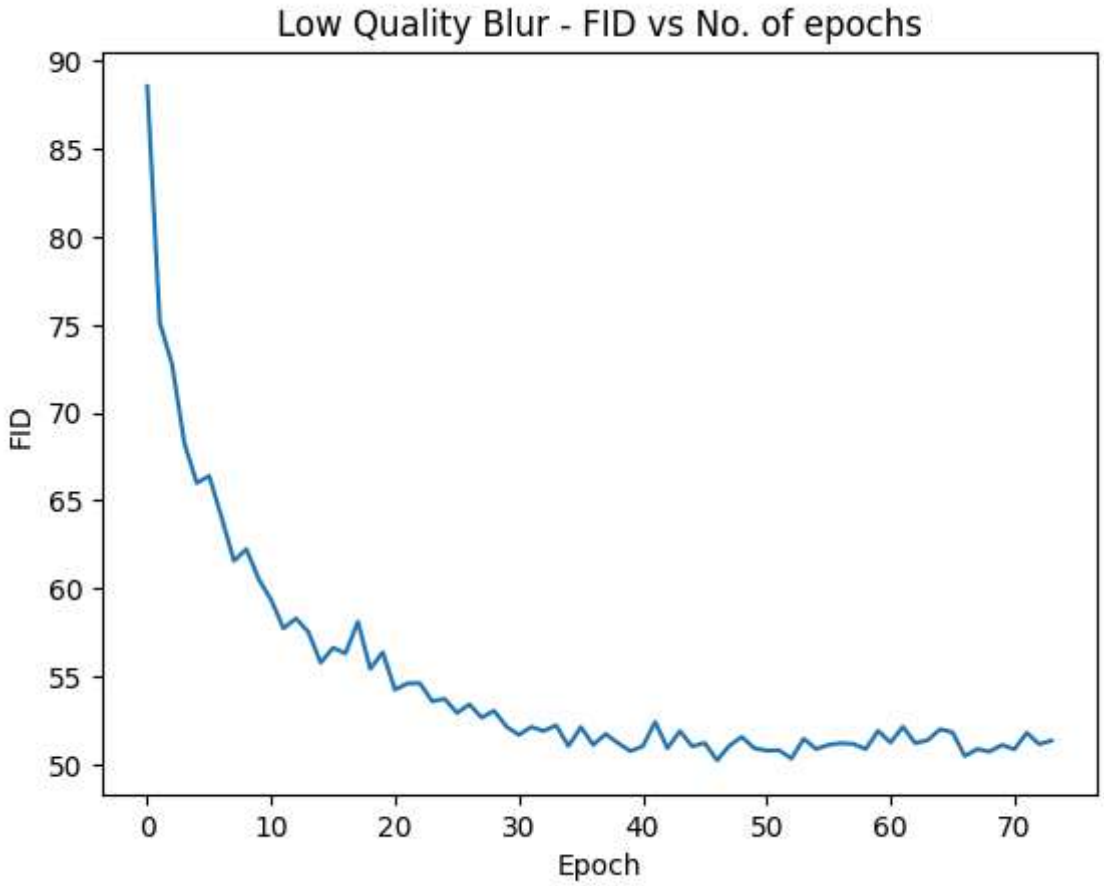
```
def plot_psnrs(history):
    psnrs = [x for x in history['psnrs']]
    plt.plot(psnrs)
    plt.xlabel('Epoch')
    plt.ylabel('PSNR')
    plt.title('Low Quality Blur - PSNR vs No. of epochs')

plot_psnrs(checkpoint)
```



In [36]:
```
def plot_ssims(history):
    psnrs = [x for x in history['ssims']]
    plt.plot(psnrs)
    plt.xlabel('Epoch')
    plt.ylabel('SSIM')
    plt.title('Low Quality Blur - SSIM vs No. of epochs')

plot_ssims(checkpoint)
```



In [37]:
```
def plot_fids(history):
    psnrs = [x for x in history['fids']]
    plt.plot(psnrs)
    plt.xlabel('Epoch')
    plt.ylabel('FID')
    plt.title('Low Quality Blur - FID vs No. of epochs')

plot_fids(checkpoint)
```


Low Quality Blur - FID vs No. of epochs
FID
Epoch
90
85
80
75
70
65
60
55
50
0
10
20
30
40
50
60
70

In [ ]: