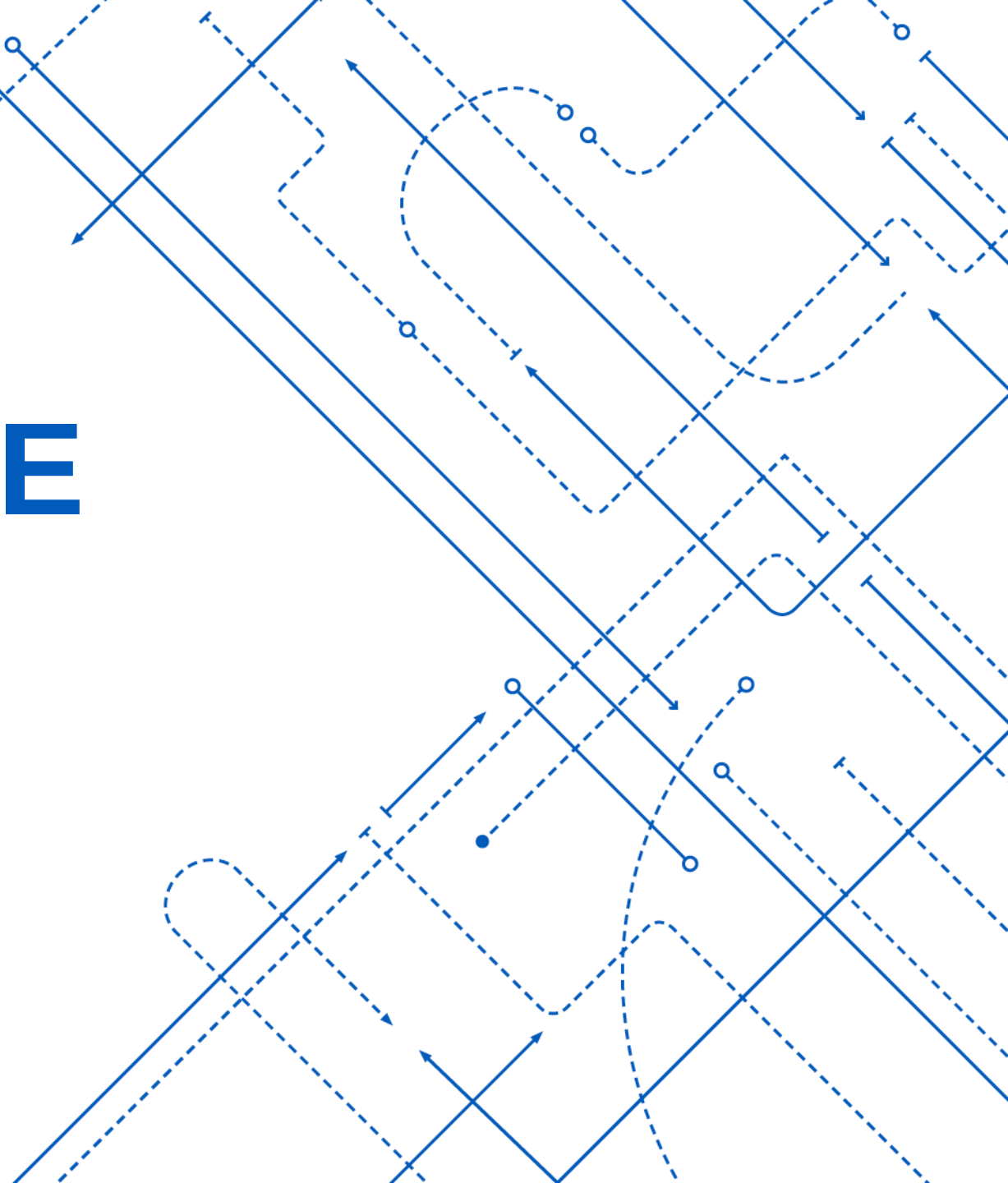
ARTISTIC STYLE TRANSFER

Group 45

CSE 676: Deep Learning, Summer 2023

05 July 2023

University at Buffalo The State University of New York



Project Description

Image Style Transfer combines the objects of a content image with the artistic style of a reference image, resulting in a visually captivating composition. It merges brush strokes, colors, and textures from the reference image to transform the content image into a unique artistic representation. This task is also known as domain adaptation or image-to-image translation.

Image style transfer can be used for various reasons today, such as Visual Storytelling, Branding and Marketing, Photo editing and enhancement, Artistic exploration and experimentation, etc.

It offers a wide range of applications and opportunities to create visually appealing and engaging content.

Background

Some key background and works related to image style transfer are:

- Texture synthesis techniques in the 1990s focused on generating visually similar textures.
- Non-photorealistic rendering aimed to mimic traditional artistic styles.
- Neural style transfer, introduced in 2015, revolutionized the field by leveraging deep neural networks to separate content and style representations. This approach allows the creation of visually appealing hybrid images by optimizing an input image to match the content of one image and the style of another.
- Generative Adversarial Networks (GANs) have also been employed for style transfer tasks, learning mappings between image domains. Recent works have explored interactive style transfer, enabling user-guided transfer using sketches or brush strokes.

Background contd.

How is your approach similar or different from others?

We have focused on creating artistic image style transfer using various Generative Adversarial Networks (GANs). For our project, we are trying to take normal architectural images and paintings from various famous artists and applying Image to Image translation.

Dataset

Our dataset contains two subsets, base dataset and style dataset.

Base dataset: Google Image Scraped Dataset

Style dataset: Best Artworks of All Time

- Base dataset is scraped data from google images. Images typically belong to 4 classes: Art & culture, Architecture, Food & Drinks and Travel and Adventure.
- Style dataset is collection of paintings of the 50 most influential artists of all time.
- Both the above datasets are taken from Kaggle.

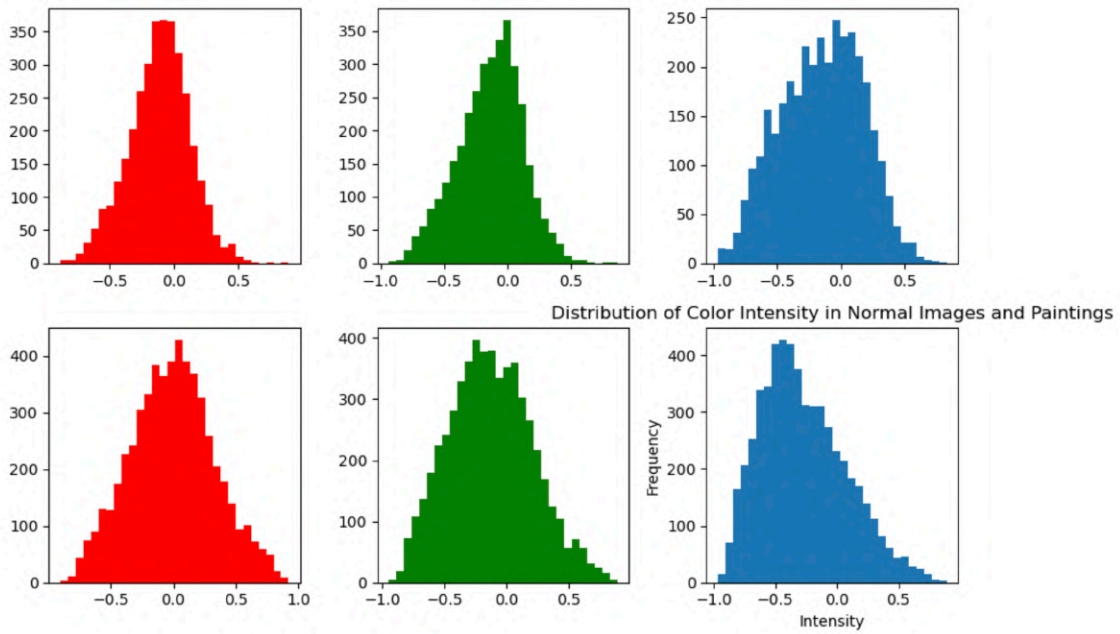
Dataset contd.

- For our project, we have used only the architectural category of base dataset and almost complete style dataset.
- Amount of data we are working with is:

```
print(f"Number of normal images: {len(normal_loader.dataset)}")
print(f"Number of art images: {len(art_loader.dataset)}")
```

```
Number of normal images: 3500

Number of art images: 5600
```

- For our project, we have resized the above images to 256*256 and normalized them.




```
Unique image sizes in normal dataset: {torch.Size([3, 256, 256])}

Unique color channels in normal dataset: {3}

Unique image sizes in art dataset: {torch.Size([3, 256, 256])}

Unique color channels in art dataset: {3}
```

Methods

We have used various GANs to achieve image-to-image translation such as:

- CycleGAN: A deep learning model and framework introduced by Jun-Yan Zhu et al. in 2017. It is designed for unpaired image-to-image translation. The key idea behind CycleGAN is to leverage the concept of cycle consistency. CycleGAN gave best results for our project.
- Pix2Pix GAN: A deep learning model and framework introduced by Isola et al. in 2016. It is specifically designed for paired image-to-image translation, where each input image is associated with a corresponding desired output image. We were not able to generate optimum results using Pix2Pix.
- StarGAN: A deep learning model and framework introduced for the task of multi-domain image-to-image translation by Choi et al. in 2018. Unlike previous GANs, which usually focus on translating images from one domain to another (like turning horses into zebras, or summer scenes into winter scenes), StarGAN can perform image translation across multiple domains using only a single model.

Methods

- UnitGAN: Unsupervised Image-to-Image Translation Networks proposed by Liu et al. in 2017 used for the purpose of image-to-image translation tasks. It takes two images, one from domain A and one from domain B, are representing the same object or scene, they can be translated into the same 'content' representation in a latent space.
- StyleGAN: A deep learning model and framework introduced by researchers at NVIDIA. The first version was introduced in 2018, with a subsequent version, StyleGAN2, coming in 2019. Unlike traditional GANs, which generate an image from random noise, StyleGAN adds another layer of complexity by introducing the style concept. StyleGAN uses a mapping network to transform the input noise into an intermediate latent space, the style space. This step helps to separate the high-level attributes (like the shape of an object in the image) and low-level attributes (like colors or textures) of the image.

Model architecture- CycleGAN

Generator

Encoder
(**Padding,**
2D-Convolutional Layers,
Normalization Layers,
ReLU activations)

Transform
(ResidualBlock)

Decoder(
ConvTranspose2d,
Normalization Layers,
ReLU and tanh activations)

ResidualBlock.

Padding,
2D-Convolutional Layers,
Normalization Layers,
ReLU activation)

Discriminator

Conv2D Layers,
InstanceNorm2d
LeakyReLU

```python
class Generator(nn.Module):
    def __init__(self, in_channels=3, out_channels=3, res_blocks=9):
        super(Generator, self).__init__()
        self.encoder = nn.Sequential(
            nn.ReflectionPad2d(3),
            nn.Conv2d(in_channels, 64, 7),
            nn.InstanceNorm2d(64),
            nn.ReLU(inplace=True),

            nn.Conv2d(64, 128, 3, stride=2, padding=1),
            nn.InstanceNorm2d(128),
            nn.ReLU(inplace=True),

            nn.Conv2d(128, 256, 3, stride=2, padding=1),
            nn.InstanceNorm2d(256),
            nn.ReLU(inplace=True)
        )
        self.transform = nn.Sequential(*[ResidualBlock(256) for _ in range(res_blocks)])
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 3, stride=2, padding=1
            nn.InstanceNorm2d(128),
            nn.ReLU(inplace=True),

            nn.ConvTranspose2d(128, 64, 3, stride=2, padding=1, output_padding=1),
            nn.InstanceNorm2d(64),
            nn.ReLU(inplace=True),

            nn.ReflectionPad2d(3),
            nn.Conv2d(64, out_channels, 7),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.transform(x)
        x = self.decoder(x)
        return x
```

(parameter) padding: _size_2

```python
class ResidualBlock(nn.Module):
    def __init__(self, in_channels):
        super(ResidualBlock, self).__init__()
        self.block = nn.Sequential(
            nn.ReflectionPad2d(1),
            nn.Conv2d(in_channels, in_channels, 3),
            nn.InstanceNorm2d(in_channels),
            nn.ReLU(inplace=True),
            nn.ReflectionPad2d(1),
            nn.Conv2d(in_channels, in_channels, 3),
            nn.InstanceNorm2d(in_channels)
        )

    def forward(self, x):
        return x + self.block(x)
```

```python
class Discriminator(nn.Module):
    def __init__(self, in_channels=3):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(in_channels, 64, 4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 4, stride=2, padding=1),
            nn.InstanceNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 4, stride=2, padding=1),
            nn.InstanceNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(256, 512, 4, padding=1),
            nn.InstanceNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(512, 1, 4, padding=1)
        )

    def forward(self, x):
        x = self.main(x)
        return x
```

Hyperparameters

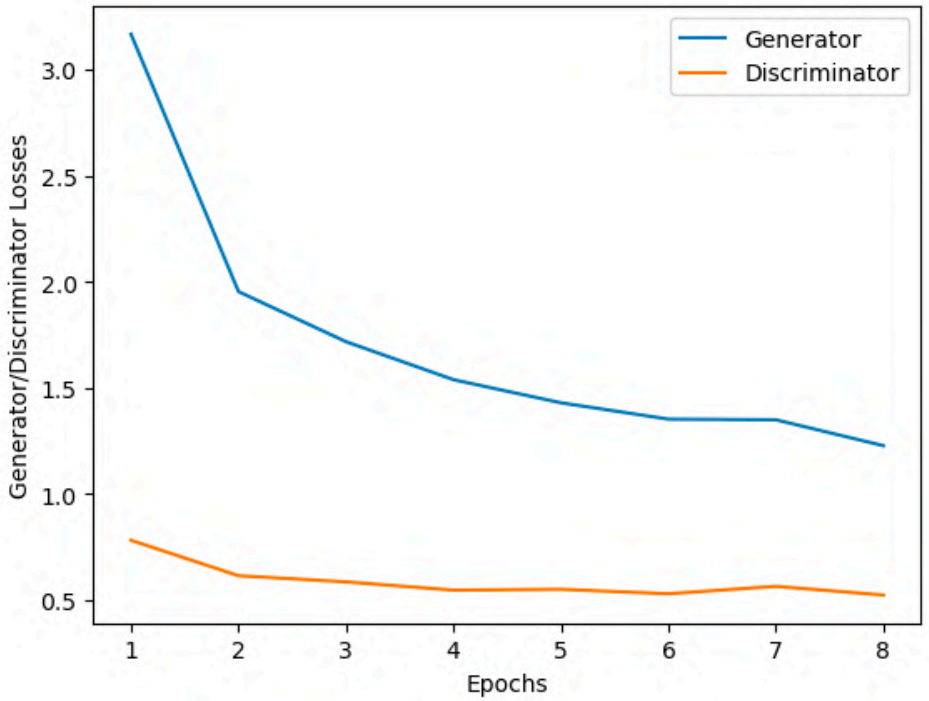
- Number of Residual Blocks in the Generator = 9
- Batch Size
- Learning Rate:0.001
- Epochs = 8
- Loss Function:- MSE
- Optimizer- Adam
- Number of Convolutional Layers
- Data Transforms

Results:

- CycleGAN gave us optimum results: Actual vs Generated

Results

- Loss – Epoch curve for Generator and Discriminator:



- The **Generator Loss** value decreases over epochs, suggesting that the generator is getting better at its job.
- The **Discriminator Loss** is also decreasing over epochs, which implies that the discriminator is also getting better at its task to not to get fooled by **Generator.**

Results

- Discriminator checking whether the image is fake or not:

```
0.8689524531364441
The generated image is classified as real artwork.
0.3370647132396698
The generated image is classified as fake normal image.
```

Model architecture- StarGAN

Generator

Initial Block(
2D-Convolutional Layers,
Normalization Layers,
ReLU activation)

Down-sampling Block(
2D-Convolutional Layers,
Normalization Layers,
ReLU activation)

Transform(**Residual Block**)

Up-Sampling Block(**ConvTranspose2d**,
Normalization Layers,
ReLU activation)

ResidualBlock.

2D-Convolutional Layers,
Normalization Layers,
ReLU activation)

Discriminator

Conv2D Layers,
InstanceNorm2d
LeakyReLU

```python
class Generator(nn.Module):
    def __init__(self, c_dim, num_res_blocks):
        super(Generator, self).__init__()

        layers = [
            nn.Conv2d(3 + c_dim, 64, kernel_size=7, stride=1, padding=3, bias=False),
            nn.InstanceNorm2d(64, affine=True, track_running_stats=True),
            nn.ReLU(inplace=True)
        ]

        # Down-sampling layers
        curr_dim = 64
        for _ in range(2):
            layers.append(nn.Conv2d(curr_dim, curr_dim * 2, kernel_size=4, stride=2, padding=1, bias=False))
            layers.append(nn.InstanceNorm2d(curr_dim * 2, affine=True, track_running_stats=True))
            layers.append(nn.ReLU(inplace=True))
            curr_dim *= 2

        # Residual blocks
        for _ in range(num_res_blocks):
            layers.append(ResidualBlock(curr_dim))

        # Up-sampling layers
        for _ in range(2):
            layers.append(nn.ConvTranspose2d(curr_dim, curr_dim // 2, kernel_size=4, stride=2, padding=1, bias=False))
            layers.append(nn.InstanceNorm2d(curr_dim // 2, affine=True, track_running_stats=True))
            layers.append(nn.ReLU(inplace=True))
            curr_dim = curr_dim // 2

        layers.append(nn.Conv2d(curr_dim, 3, kernel_size=7, stride=1, padding=3, bias=False))
        layers.append(nn.Tanh())
        self.main = nn.Sequential(*layers)

    def forward(self, x, c):
        c = c.view(c.size(0), c_dim, 1, 1).expand(-1, -1, x.size(2), x.size(3))
        x = torch.cat([x, c], dim=1)
        return self.main(x)
```

size: Any

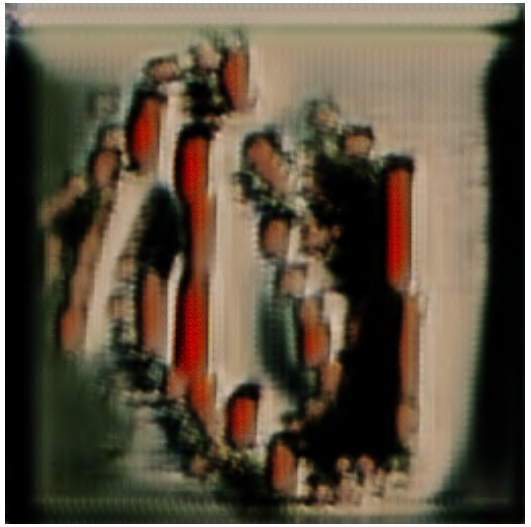
```python
class Discriminator(nn.Module):
    def __init__(self, image_size, c_dim, num_layers=5):
        super(Discriminator, self).__init__()

        layers = [
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.01)
        ]

        curr_dim = 64
        for i in range(1, num_layers):
            layers.append(nn.Conv2d(curr_dim, curr_dim * 2, kernel_size=4, stride=2, padding=1))
            layers.append(nn.LeakyReLU(0.01))
            curr_dim *= 2

        kernel_size = image_size // 2**num_layers
        self.main = nn.Sequential(*layers)
        self.conv1 = nn.Conv2d(curr_dim, 1, kernel_size=3, stride=1, padding=1, bias=False)
        self.conv2 = nn.Conv2d(curr_dim, c_dim, kernel_size=kernel_size, bias=False)

    def forward(self, x):
        h = self.main(x)
        out_src = self.conv1(h)
        out_cls = self.conv2(h)
        return out_src, out_cls.view(out_cls.size(0), out_cls.size(1))
```

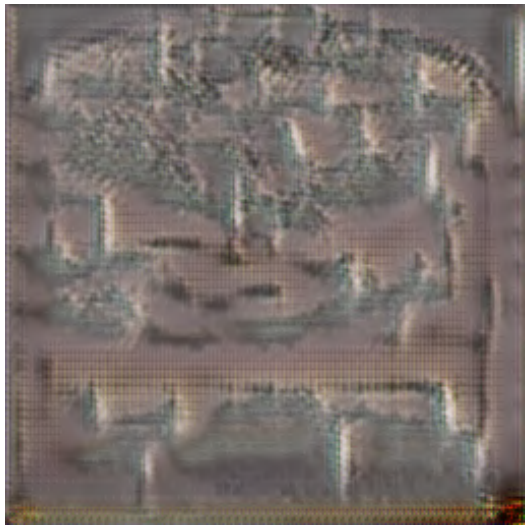
```python
class ResidualBlock(nn.Module):
    def __init__(self, in_features):
        super(ResidualBlock, self).__init__()

        self.block = nn.Sequential(
            nn.Conv2d(in_features, in_features, kernel_size=3, stride=1, padding=1, bias=False),
            nn.InstanceNorm2d(in_features, affine=True, track_running_stats=True),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_features, in_features, kernel_size=3, stride=1, padding=1, bias=False),
            nn.InstanceNorm2d(in_features, affine=True, track_running_stats=True)
        )

    def forward(self, x):
        return x + self.block(x)
```

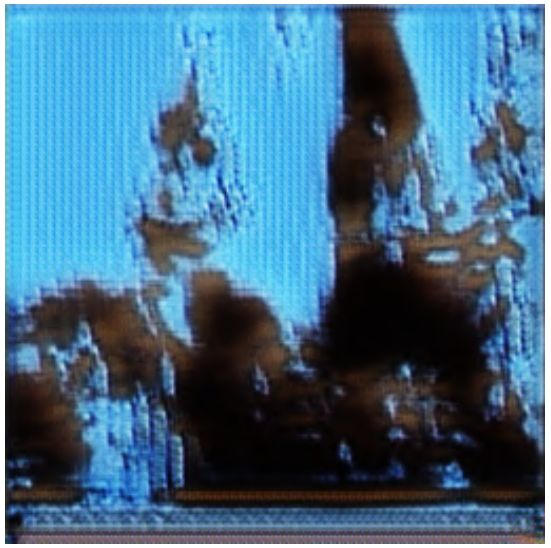
Hyperparameters

- Number of Residual Blocks in the Generator = 9
- Batch Size
- Learning Rate: 0.0002
- Epochs = 20
- Loss Function:- MSE, Cross Entropy
- Optimizer- Adam
- Number of Convolutional Layers
- Data Transforms

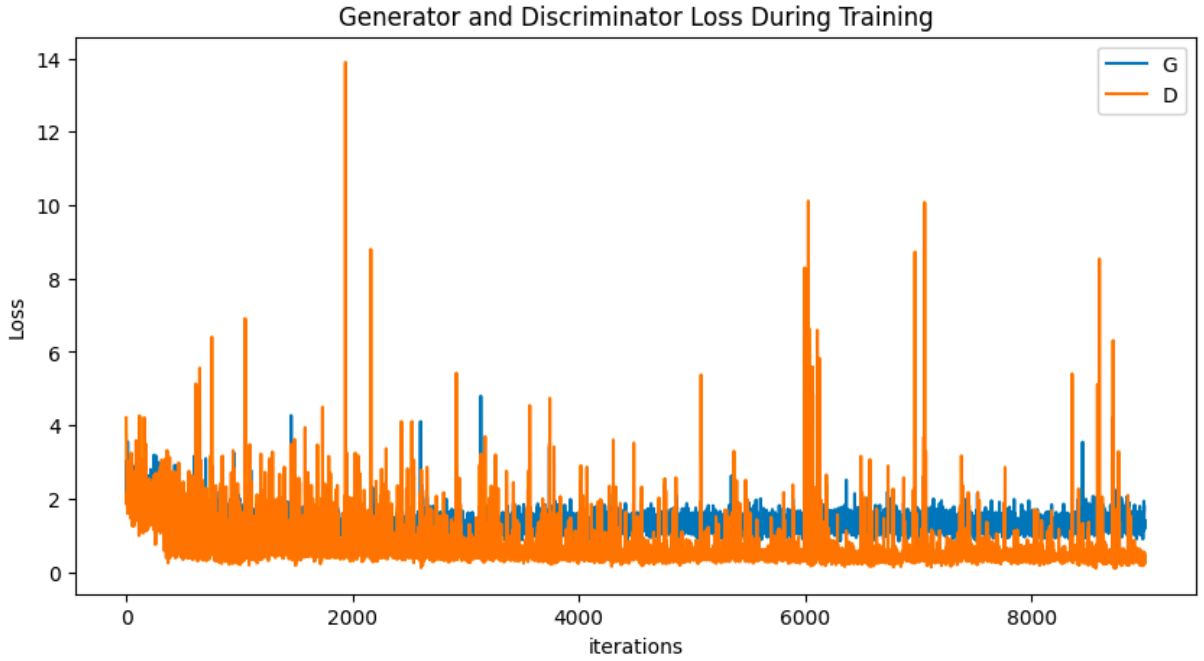
Results using StarGAN







- Generator Loss is getting better at creating fake images that the Discriminator can't distinguish from real ones In some case you can see Generator is "winning" too often.



Model architecture- UnitGAN

Generator

Initial Block(**2D-Convolutional Layers, Normalization Layers, ReLU activation**)

Down-sampling Block(**ConvBlock**)

Transform(**Residual Block**)

Up-Sampling Block(**ConvBlock**)

ConvBlock

2D-Convolutional Layers, ConvTranspose2d ,Normalization Layers,ReLU activation

Residual Block

2D-Convolutional Layers, Normalization Layers, ReLU activation)

Discriminator

Conv2D Layers, InstanceNorm2d LeakyReLU

```python
class Generator(nn.Module):
    def __init__(self, in_channels=3, num_features=32):
        super().__init__()
        self.initial = nn.Sequential(
            nn.Conv2d(in_channels, num_features, kernel_size=7, stride=1, padding=3, padding_mode="reflect"),
            nn.InstanceNorm2d(num_features),
            nn.ReLU(inplace=True),
        )
        self.down_blocks = nn.ModuleList(
            [
                ConvBlock(num_features, num_features*2, kernel_size=3, stride=2, padding=1),
            ]
        )
        self.res_blocks = nn.ModuleList(
            [
                ResidualBlock(num_features*2) for _ in range(4)
            ]
        )
        self.up_blocks = nn.ModuleList(
            [
                ConvBlock(num_features*2, num_features, down=False, kernel_size=3, stride=2, padding=1, output_padding=1),
            ]
        )

        self.last = nn.Conv2d(num_features, in_channels, kernel_size=7, stride=1, padding=3, padding_mode="reflect")


    def forward(self, x):
        x = self.initial(x)
        for layer in self.down_blocks:
            x = layer(x)
        for layer in self.res_blocks:
            x = layer(x)
        for layer in self.up_blocks:
            x = layer(x)
        return torch.tanh(self.last(x))
```

```python
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels, down=True, use_act=True, **kwargs):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, padding_mode="reflect", **kwargs)
            if down
            else nn.ConvTranspose2d(in_channels, out_channels, **kwargs),
            nn.InstanceNorm2d(out_channels),
            nn.ReLU(inplace=True) if use_act else nn.Identity()
        )

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

class ResidualBlock(nn.Module):
    def __init__(self, in_channels):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, in_channels, kernel_size=3, padding=1, padding_mode="reflect")
        self.conv2 = nn.Conv2d(in_channels, in_channels, kernel_size=3, padding=1, padding_mode="reflect")
        self.instancenorm = nn.InstanceNorm2d(in_channels)
        self.activation = nn.ReLU()

    def forward(self, x):
        initial = x
        x = self.conv1(x)
        x = self.instancenorm(x)
        x = self.activation(x)
        x = self.conv2(x)
        x = self.instancenorm(x)
```

```python
class Discriminator(nn.Module):
    def __init__(self, in_channels=3, features=[32, 64, 128]):
        super().__init__()

        self.initial = nn.Sequential(
            nn.Conv2d(
                in_channels,
                features[0],
                kernel_size=4,
                stride=2,
                padding=1,
                padding_mode="reflect",
            ),
            nn.LeakyReLU(0.2, inplace=True),
        )

        layers = []
        in_channels = features[0]
        for feature in features[1:]:
            layers.append(
                nn.Sequential(
                    nn.Conv2d(
                        in_channels, feature, kernel_size=4, stride=2, padding=1
                    ),
                    nn.InstanceNorm2d(feature),
                    nn.LeakyReLU(0.2, inplace=True),
                ),
            )
            in_channels = feature

        layers.append(
            nn.Conv2d(in_channels, 1, kernel_size=4, stride=1, padding=1, padding_mode="reflect")
        )

        self.model = nn.Sequential(*layers)

    def forward(self, x):
        x = self.initial(x)
        return torch.sigmoid(self.model(x))
```

Hyperparameters

- Number of Residual Blocks in the Generator = 4
- Batch Size = 4
- Learning Rate = 0.0002
- Epochs = 10
- Loss Function:- Binary Cross Entropy
- Optimizer- Adam
- Number of Convolutional Layers
- Data Transforms

Results using UNIT GAN

Both the generator and the discriminator are learning over time

Summary

- We observed CycleGAN to be most efficient for image-to-image translation.
- With Pix2Pix GAN, StarGAN and UnitGAN it was difficult to achieve optimum results even after hyper-parameter tuning.
- Additionally, we implemented the StarGAN + VGG19 to add the style features but the results were not good compared to all other models which we have implemented.
- Encountered issues with StyleGAN model implementation.

References

- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks - https://arxiv.org/abs/1703.10593
- Choi, Y., Choi, M., Kim, M., Ha, J., Kim, S., & Choo, J. (2018). StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). - https://arxiv.org/abs/1711.09020
- Liu, M. Y., Breuel, T., & Kautz, J. (2017). Unsupervised Image-to-Image Translation Networks. In Advances in Neural Information Processing Systems (NeurIPS) - https://arxiv.org/abs/1703.00848
- Karras, T., Laine, S., & Aila, T. (2018). A Style-Based Generator Architecture for Generative Adversarial Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).- https://arxiv.org/abs/1812.04948
- Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2019). Analyzing and Improving the Image Quality of StyleGAN - https://arxiv.org/abs/1912.04958
- https://towardsdatascience.com/
- https://medium.com/
- https://pytorch.org/tutorials/beginner/saving_loading_models.html
- https://towardsdatascience.com/how-to-deploy-machine-learning-models-601f8c13f f45
- https://seaborn.pydata.org/tutorial/introduction
- https://matplotlib.org/
- https://kaggle.com
- https://pandas.pydata.org/
- https://www.w3schools.com/python/numpy/numpy_intro.asp

Contributions Summary

- Below is the contribution summary.

Type	Member	Contribution
Dataset Preprocessing, etc	Kishan	50
Dataset Preprocessing, etc	Harinath	50
Methods	Kishan	50
Methods	Harinath	50
Results	Kishan	50
Results	Harinath	50

University at Buffalo The State University of New York

THANK YOU

Group 45:

Kishan Patel

Harinath Cingapuram