(2024F) COMP-5112-Research Methodology

Group 10

Nerual Style Transfer

Junjun Hu 1264029

Zhijie Shen 1263245

Overview

Objective

Key Concepts

Main Method

Model Training

Method Implementation

Evaluation

Results

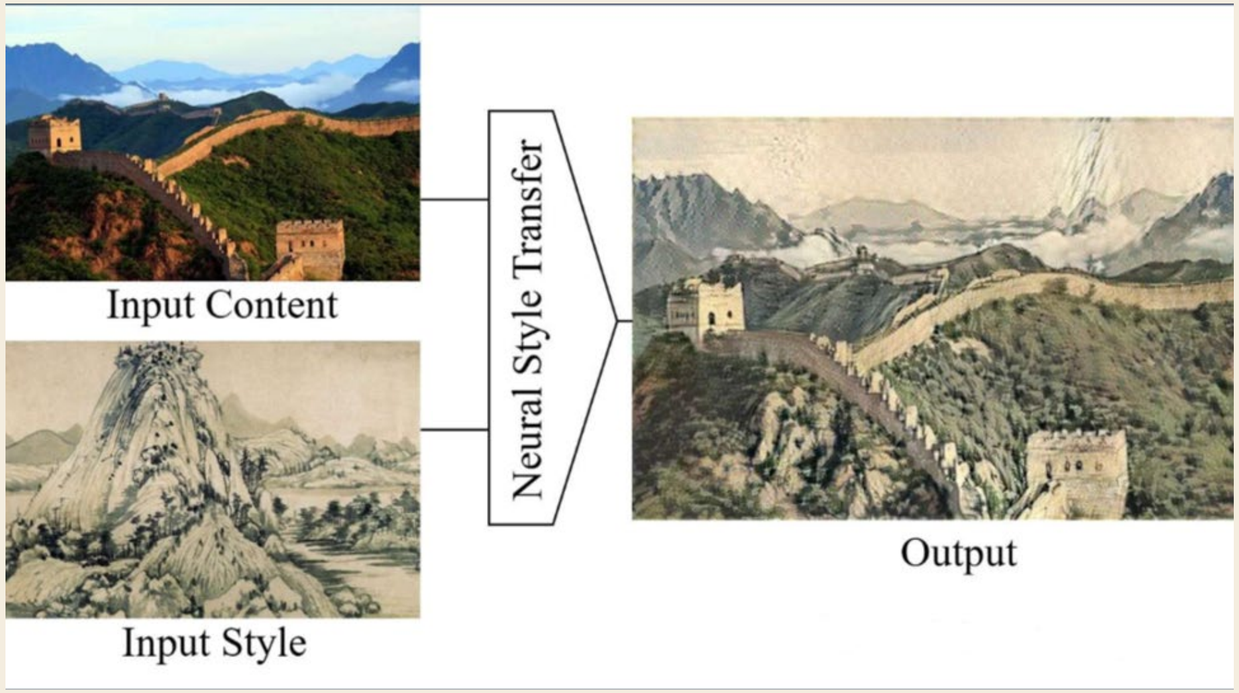
Objective

Combining the content of one image with the style of another.

A junction of the art and technique

Key concept

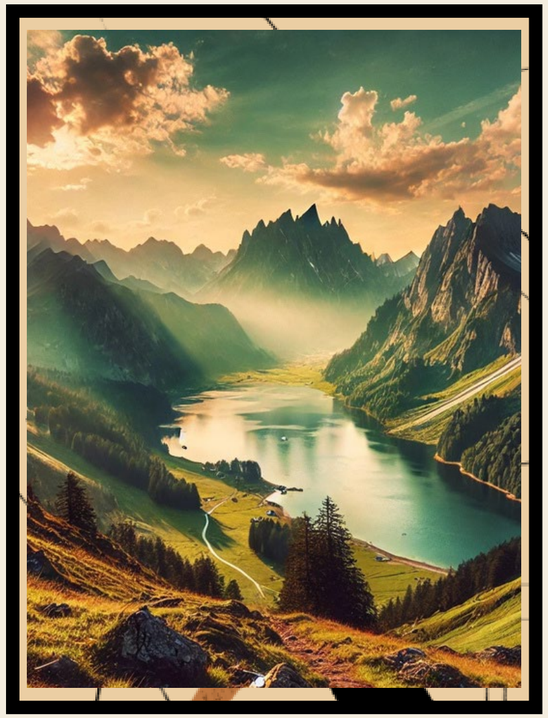
Content Image + Style Image → Generate → Combination Output

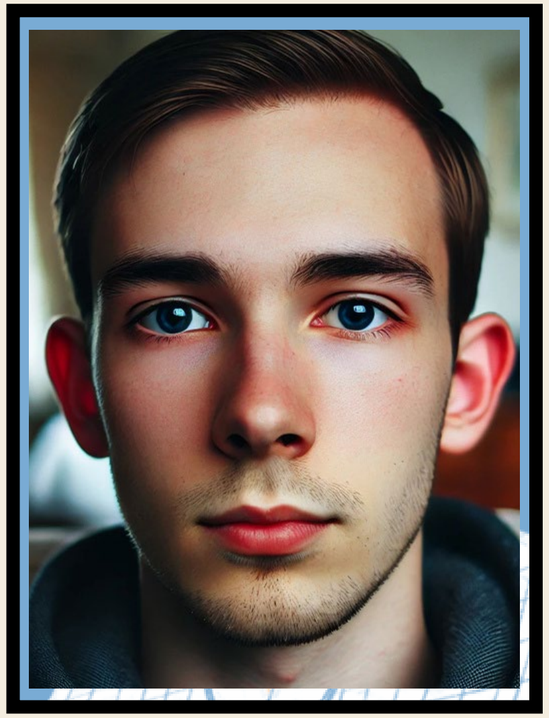


Content Image

Content refers to the content an artwork keeps.

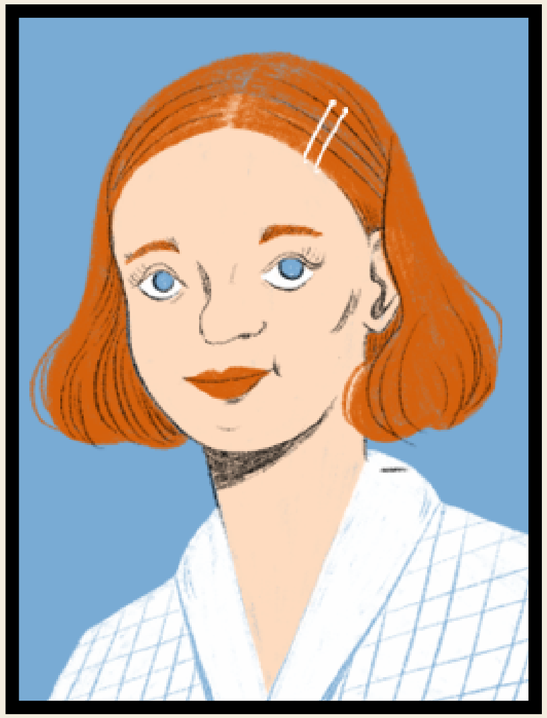
The image whose main shapes and structure you want to keep, like a building, person, or landscape.

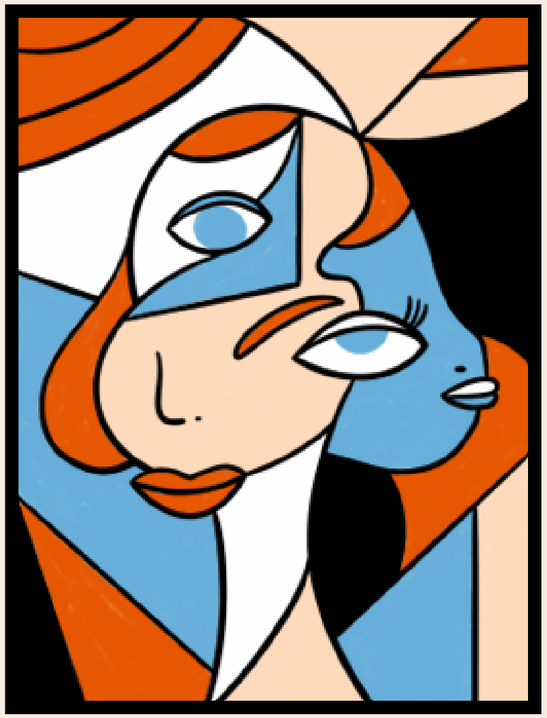




Style Image

Style Images (paintings) refers to the way an artwork looks.





Style Lost

Make sure that the high -level features of the generated image are similar to the content image.

$$L_{\text{content}} = \frac{1}{2} \sum (F_{\text{content}} - F_{\text{generated}})^2$$

Content Lost

Make sure that the low -level features of the generated image match the style image.

$$L_{\text{style}} = \sum (G_{\text{style}} - G_{\text{generated}})^2$$

Style Lost

Make sure that the high -level features of the generated image are similar to the content image.

Content Lost

Make sure that the low -level features of the generated image match the style image.

$$L_{\text{content}} = \frac{1}{2}\sum(F_{\text{content}} - F_{\text{generated}})^2$$

$$L_{\text{style}} = \sum(G_{\text{style}} - G_{\text{generated}})^2$$

Total Lost

$$\mathcal{L}_{\text{total}}(G) = \alpha\mathcal{L}_{\text{content}}(C, G) + \beta\mathcal{L}_{\text{style}}(S, G)$$

α and β control the trade-off between preserving content and adhering to the style.

Main Method
Model Training

Main Method

1.preprocessing images: PIL and Transforms

2.Content and Style Loss Definitions:
ContentLoss--MSE StyleLoss---Gram

3.Feature Extraction:VGG19/Vision Transformer/VGG19+VIT

4.Optimizer and Loss Optimization:Adam optimizer

Model Training

Vision Transformer

1.Introduction and background:

ViT applies the Transformer architecture to image classification by dividing images into smaller patches and using self-attention to capture global relationships, offering better context understanding compared to traditional CNNs.

2.Dataset and Model Architecture:

- Dataset: CIFAR-100, consisting of 100 classes with images of size 32x32.
- Model Architecture: The ViT model splits images into small patches and uses a self-attention mechanism to capture global relationships between patches.
- Key model parameters: depth=6, dimension =512,number of heads=8, patch size=4,learning_rate = 3e-4,batch_size=64

```python
# Vision Transformer model
model = ViT(
    image_size=32,
    patch_size=4,
    num_classes=100,
    dim=512,
    depth=6,
    heads=8,
    mlp_dim=1024,
    dropout=0.1,
    emb_dropout=0.1
).to(device)
```

Model Training

Vision Transformer

Data Loading:

```python
# Data preprocessing
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
# CIFAR-100 dataset
train_dataset = torchvision.datasets.CIFAR100(root='./data',
 train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.CIFAR100(root='./data',
 train=False, download=True, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

Training Loop&Saving the model

```python
# Training loop
def train_model(model, train_loader, criterion, optimizer, epochs):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        for i, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)
            # Zero the parameter gradients
            optimizer.zero_grad()
            # Forward pass
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            # Backward pass and optimization
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % 100 == 99:  # Print every 100 mini-batches
                print(f'Epoch [{epoch + 1}/{epochs}],
                     Step [{i + 1}/{len(train_loader)}], Loss: {running_loss / 100:.4f}')
                running_loss = 0.0
        # Save the model after each epoch
        torch.save(model.state_dict(), f'vit_cifar100_epoch_{epoch + 1}.pth')
```

Model Training

Vision Transformer

Training Epoches

Model Training

VGG-19

1.Introduction and background:

VGG-19 is a deep convolutional neural network by the Visual Geometry Group at Oxford. It uses small 3x3 convolutions to increase depth, improving performance. VGG-19 has 19 layers and achieved strong results in the 2014 ImageNet competition. Its simplicity makes it widely used in computer vision.

2.Dataset and Model Architecture:

- Dataset: CIFAR-100, consisting of 100 classes with images of size 32x32.
- Model Architecture: The ViT model splits images into small patches and uses a self-attention mechanism to capture global relationships between patches.
- Key model parameters: epoches=10,lr=0.01,batch_size=64

Model Training

VGG19

Data Loading:

```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5071, 0.4865, 0.4409), (0.2673, 0.2564, 0.2
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5071, 0.4865, 0.4409), (0.2673, 0.2564, 0.2
])
trainset = torchvision.datasets.CIFAR100(root='./data',
 train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR100(root='./data', train=False,
 download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
 shuffle=False, num_workers=2)
```

Training Loop&Saving the model

```
# Training loop
def train(epoch):
    vgg19.train()
    running_loss = 0.0
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = vgg19(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if batch_idx % 100 == 99:
            print(f'Epoch [{epoch+1}], Batch [{batch_idx+1}], Loss: {running_loss / 100:.3f}')
            running_loss = 0.0
```

Method
Implementation

Method Implementation

Code Snippets

Define loss of content and loss of style

```
class ContentLoss:
    function forward(input):
        Compute MSE between input and target

function gram_matrix(input):
    Flatten and compute Gram matrix

class StyleLoss:
    function forward(input):
        Compute MSE between Gram matrices of input and target
```

Load and preprocess images

```
function load_image(image_name):
    Load and resize image
    Convert image to tensor with batch dimension
    Return image tensor on device
```

Method Implementation

Code Snippets

Define loss of content and loss of style

```python
# Custom content loss
class ContentLoss(nn.Module):
    def __init__(self, target):
        super(ContentLoss, self).__init__()
        self.target = target.detach()

    def forward(self, input):
        self.loss = F.mse_loss(input, self.target)
        return input

# This is for the style loss
def gram_matrix(input):
    a, b, c, d = input.size()  # a=batch size(=1)
    features = input.view(a * b, c * d)  # resize F_XL into \hat F_XL
    G = torch.mm(features, features.t())  # compute the gram product
    return G.div(a * b * c * d)
```

Load and preprocess images

```python
# Detect if cuda is available for GPU training otherwise it will use CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# desired size of the output image
imsize = 512 if torch.cuda.is_available() else 128  # use small size if no gpu

print(imsize)
# scale imported image
# transform it into a torch tensor
loader = transforms.Compose([transforms.Resize(imsize),  transforms.ToTensor()])
```

Method Implementation

Vision Transformer model

```
1. Define ViT architecture:
   - Split image into patches
   - Flatten patches, add positional encodings
   - Create Transformer encoder layers with multi-head self-attention and feed-forward net
   - Add classification token and final fully connected layer

2. Load and preprocess dataset:
   - Load ImageNet or other large-scale dataset
   - Resize, normalize, and split into training, validation, and test sets

3. Set up training:
   - Define cross entropy loss for classification
   - Use Adam optimizer with learning rate schedule

4. Train model:
   - For each epoch and batch:
     - Extract image patches, add positional embeddings
     - Forward pass through Transformer encoder and fully connected layer
     - Compute loss, backpropagate, update weights

5. Evaluate and save:
   - Evaluate on validation set to monitor accuracy
   - Save trained weights for later use
```

VGG 19 model

```
1. Define VGG19 architecture:
   - Create sequential model
   - Add convolutional, pooling, and fully connected layers
   - Initialize weights

2. Load and preprocess dataset:
   - Load ImageNet
   - Split into training, validation, and test sets

3. Set up training:
   - Define cross entropy loss
   - Use SGD optimizer

4. Train model:
   - For each epoch and batch:
     - Forward pass, compute loss
     - Backpropagate, update weights using optimizer

5. Evaluate and save:
   - Evaluate on validation set
   - Save trained weights
```

Method Implementation

Vision Transformer model

```python
# Vision Transformer (ViT) integration for content extraction
class ViTContentExtractor(nn.Module):
    def __init__(self, image_size=256, patch_size=32,
                 num_classes=1000, dim=1024, depth=6,
                 heads=16, mlp_dim=2048, dropout=0.1, emb_dropout=0.1):
        super(ViTContentExtractor, self).__init__()
        self.vit = ViT(
            image_size=image_size,
            patch_size=patch_size,
            num_classes=num_classes,
            dim=dim,
            depth=depth,
            heads=heads,
            mlp_dim=mlp_dim,
            dropout=dropout,
            emb_dropout=emb_dropout
        ).to(device)
```

VGG 19 model

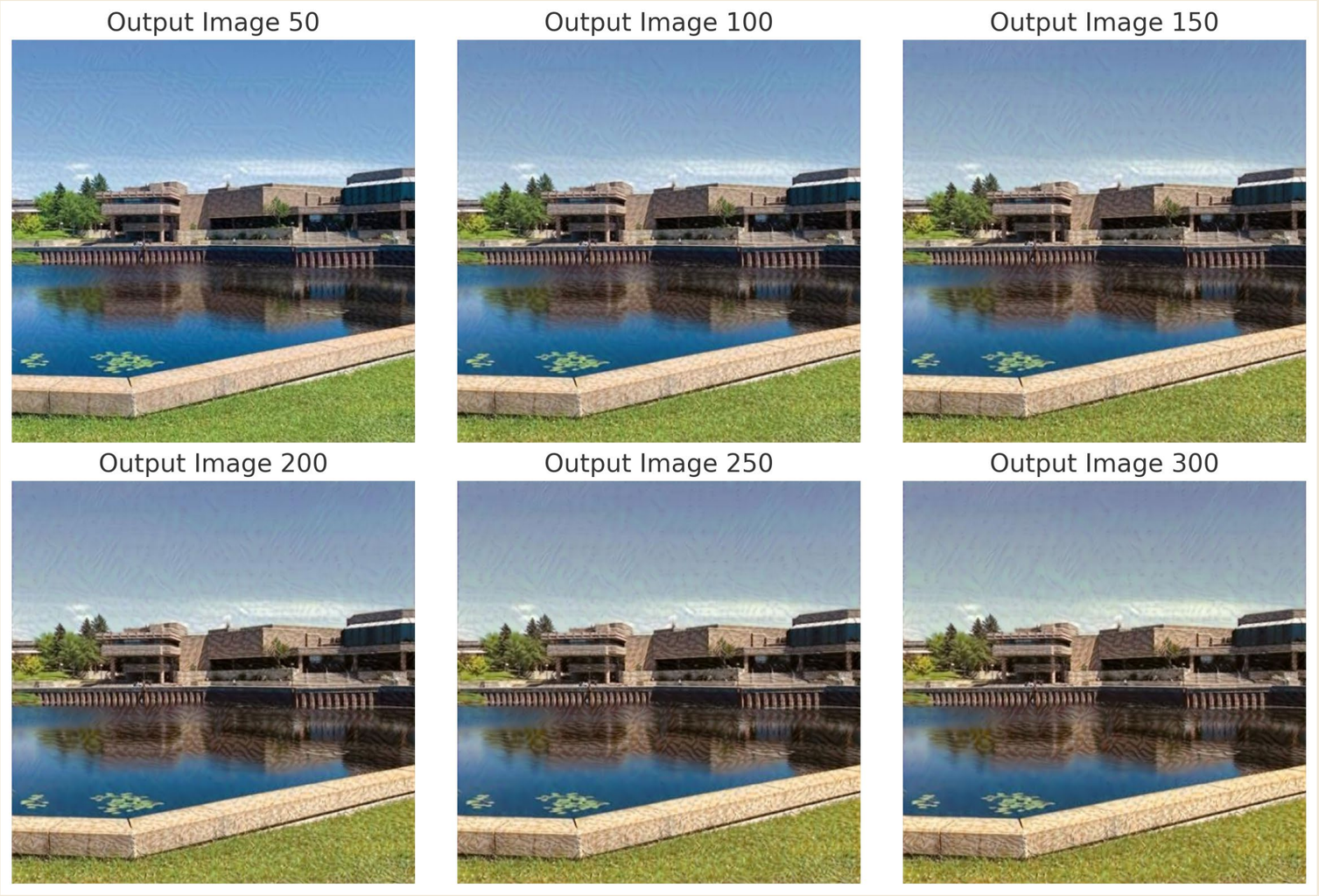
```python
# Importing the VGG 19 model like in the paper (here we set it to evaluation mode)
cnn = models.vgg19(pretrained=True).features.to(device).eval()

# VGG network are normalized with special values for the mean and std
cnn_normalization_mean = torch.tensor([0.485, 0.456, 0.406]).to(device)
cnn_normalization_std = torch.tensor([0.229, 0.224, 0.225]).to(device)

# create a module to normalize input image so we can easily put it in a nn.Sequential
class Normalization(nn.Module):
    def __init__(self, mean, std):
        super(Normalization, self).__init__()
        # .view the mean and std to make them [C x 1 x 1] so that they can
        # directly work with image Tensor of shape [B x C x H x W].
        # B is batch size. C is number of channels. H is height and W is width.
        self.mean = torch.tensor(mean).view(-1, 1, 1)
        self.std = torch.tensor(std).view(-1, 1, 1)

    def forward(self, img):
        # normalize img
        return (img - self.mean) / self.std
```

Results

Thunder Bay Art Gallery

An unconventional exhibition featuring four artists at different career stages, exploring new expressions and experiments with natural inks, dyes, and pigments in Northern Ontario. The works include painting, textile, installation, video, and photography, cultivating new experiences of color, landscape, gentleness, and community.

petal, clay, berry, leaf: natural inks, dyes, and pigments of Northern Ontario(after Millet), by Vincent Van Gogh 1890

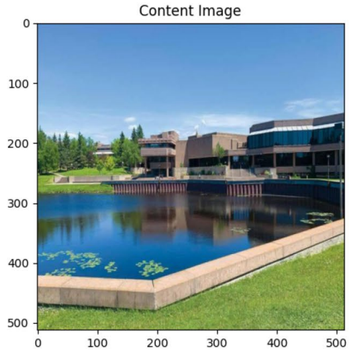


VGG19

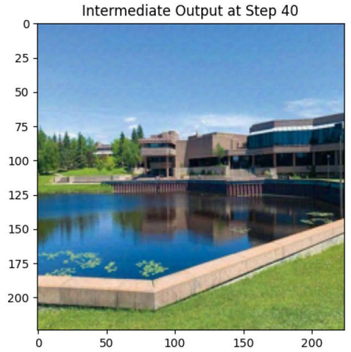
Loss of style and
loss of content per iteration

```
Iteration Results:
- Run [20]: Style Loss: 2.1645, Content Loss: 5.6274
- Run [40]: Style Loss: 0.8229, Content Loss: 5.6576
- Run [60]: Style Loss: 0.5561, Content Loss: 5.6541
- Run [80]: Style Loss: 0.4987, Content Loss: 5.6443
- Run [100]: Style Loss: 0.4711, Content Loss: 5.6334
- Run [120]: Style Loss: 0.4542, Content Loss: 5.6222
- Run [140]: Style Loss: 0.4422, Content Loss: 5.6112
- Run [160]: Style Loss: 0.4332, Content Loss: 5.6006
- Run [180]: Style Loss: 0.4261, Content Loss: 5.5908
- Run [200]: Style Loss: 0.4205, Content Loss: 5.5816
- Run [220]: Style Loss: 0.4160, Content Loss: 5.5732
- Run [240]: Style Loss: 0.4123, Content Loss: 5.5655
- Run [260]: Style Loss: 0.4093, Content Loss: 5.5585
- Run [280]: Style Loss: 0.4067, Content Loss: 5.5522
- Run [300]: Style Loss: 0.4045, Content Loss: 5.5465
```

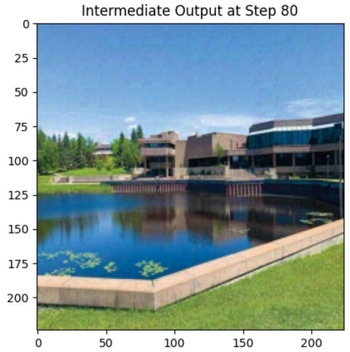
Content Image



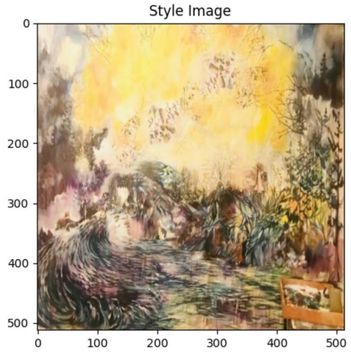
Intermediate Output at Step 40



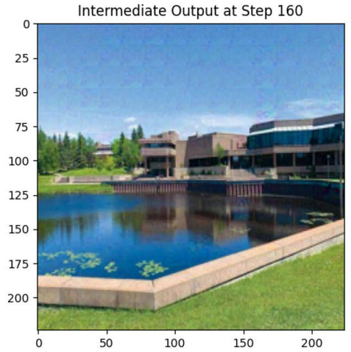
Intermediate Output at Step 80



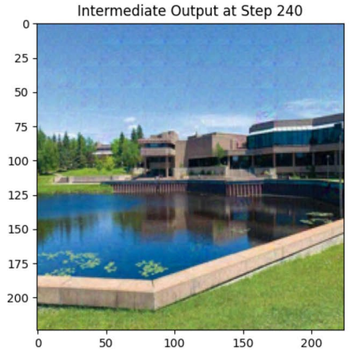
Style Image



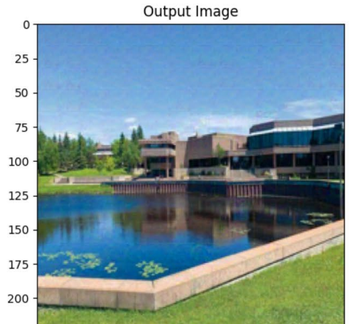
Intermediate Output at Step 160



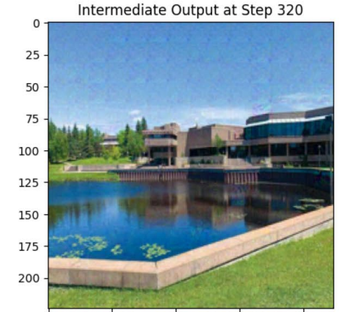
Intermediate Output at Step 240



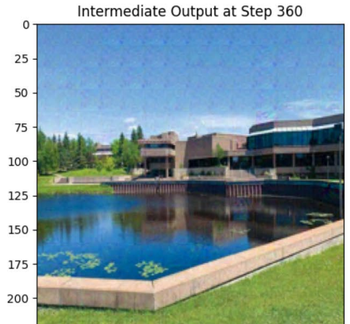
Output Image



Intermediate Output at Step 320



Intermediate Output at Step 360

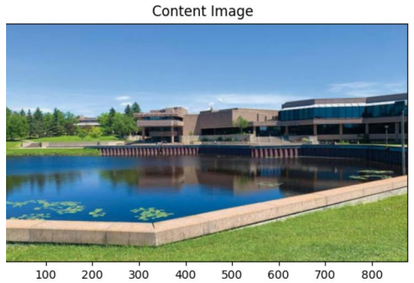


Vision Transformer

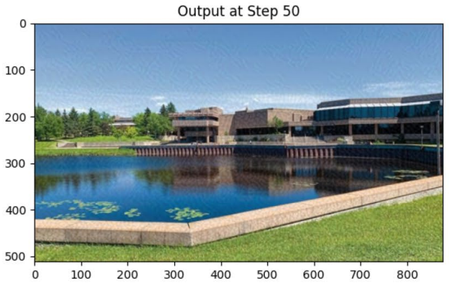
Loss of style and loss of content per iteration

```
run [10]: Style Loss: 4.02 Content Loss: 5.59
run [40]: Style Loss: 0.82 Content Loss: 5.66
run [80]: Style Loss: 0.50 Content Loss: 5.64
run [120]: Style Loss: 0.45 Content Loss: 5.62
run [160]: Style Loss: 0.43 Content Loss: 5.60
run [200]: Style Loss: 0.42 Content Loss: 5.58
run [240]: Style Loss: 0.41 Content Loss: 5.57
run [280]: Style Loss: 0.41 Content Loss: 5.55
run [290]: Style Loss: 1.41 Content Loss: 5.55
run [300]: Style Loss: 1.40 Content Loss: 5.55
```

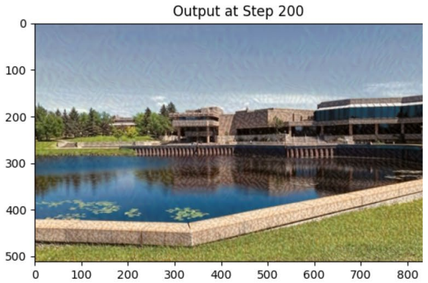
Content Image



Output at Step 50

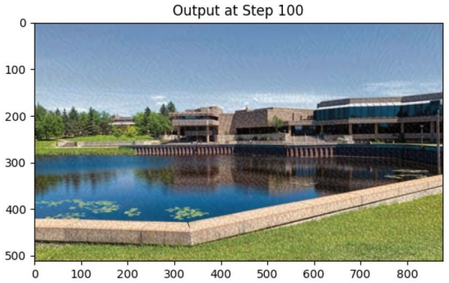


Output at Step 200

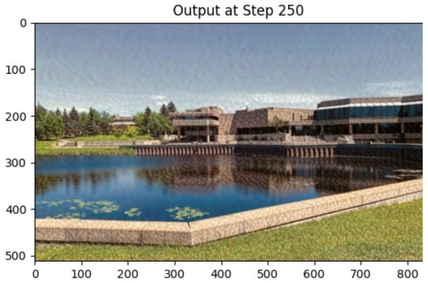


Style Image

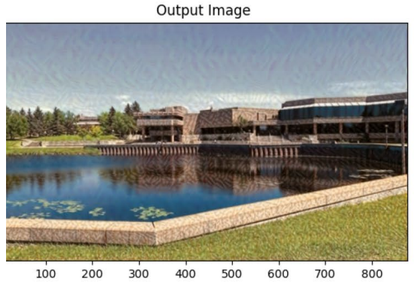
Output at Step 100



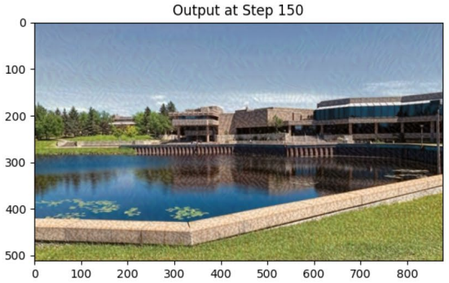
Output at Step 250



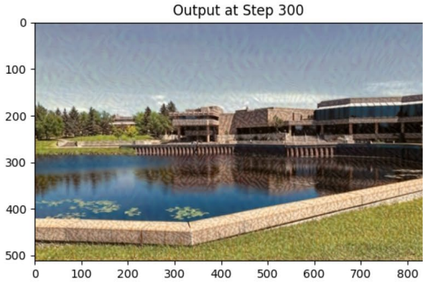
Output Image



Output at Step 150



Output at Step 300



VGG19+VIT

Loss of style and
loss of content per iteration

```
run [10]: Style Loss: 0.502160 Content Loss: 5.687279
run [40]: Style Loss: 0.182289 Content Loss: 5.757600
run [70]: Style Loss: 0.152199 Content Loss: 5.749733
run [100]: Style Loss: 0.147113 Content Loss: 5.733432
run [130]: Style Loss: 0.144777 Content Loss: 5.716612
run [160]: Style Loss: 0.143315 Content Loss: 5.700634
run [190]: Style Loss: 0.142317 Content Loss: 5.686095
run [220]: Style Loss: 0.141602 Content Loss: 5.673198
run [250]: Style Loss: 0.141072 Content Loss: 5.661938
run [300]: Style Loss: 0.144976 Content Loss: 5.101154
```

Evaluation

Evaluation

1.Content Preservation(Structural similarity index)

VGG19:SSIM: 0.72 VIT:SSIM 0.88 VIT+VGG SSIM:0.85

2.Image Quality - Noise and ArtifactsColor Consistency

VGG19:PSNR: 20 dB ViT:PSNR: 25 dBVIT + VGG19:PSNR: 23 dB

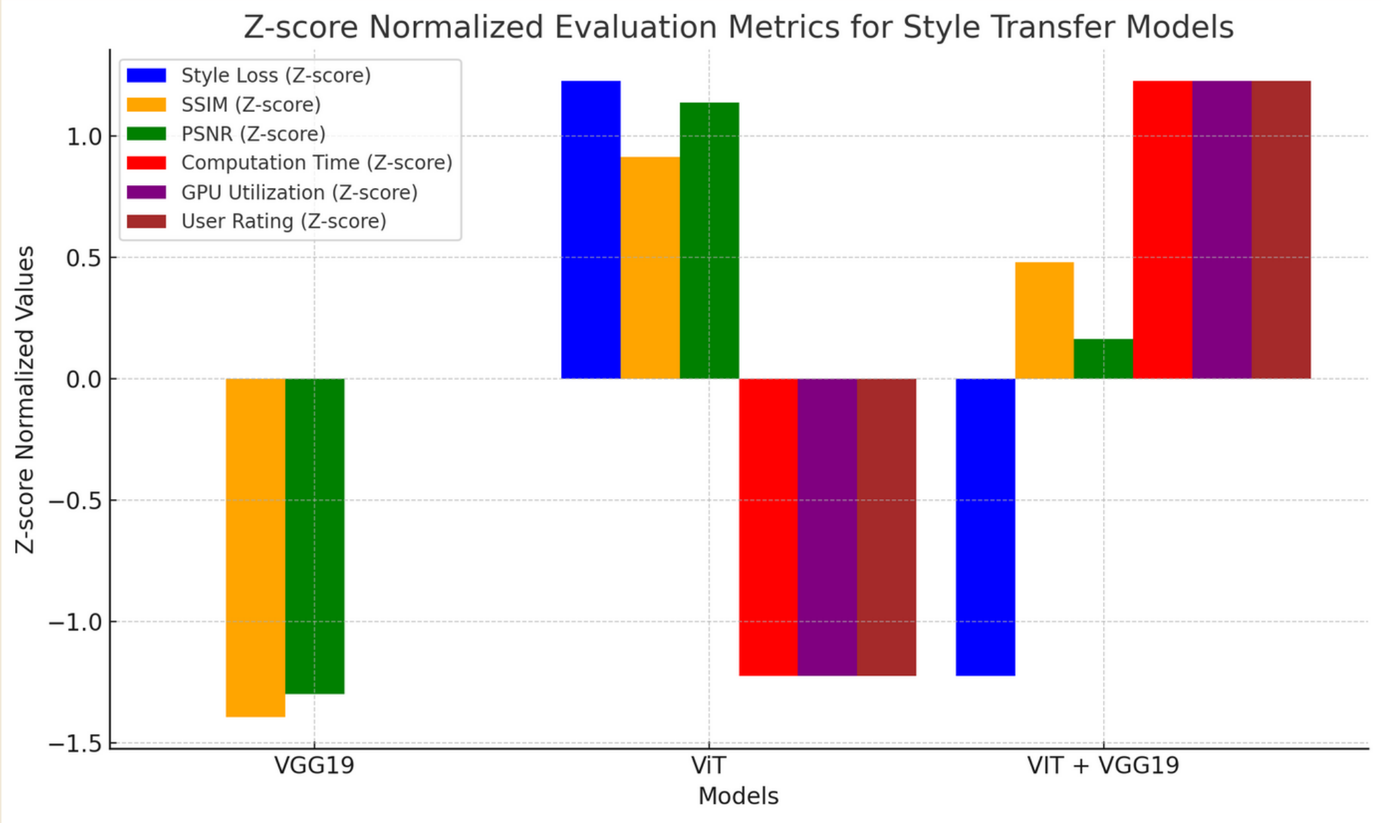
3.Computation Time and Resource Consumption

VGG19: Reasoning time: 3.5 seconds GPU utilization: 75% ViT: Reasoning time: 2.8 seconds GPU utilization: 65% VIT + VGG19: Reasoning time: 4.2 seconds GPU utilization: 85

4. Color Consistency

VGG19： 15.5

Evaluation


Z-score Normalized Evaluation Metrics for Style Transfer Models
Style Loss (Z-score)
SSIM (Z-score)
PSNR (Z-score)
Computation Time (Z-score)
GPU Utilization (Z-score)
User Rating (Z-score)
Z-score Normalized Values
1.0
0.5
0.0
−0.5
−1.0
−1.5
VGG19
ViT
VIT + VGG19
Models

Reference

•[1] Gatys, L. A., Ecker, A. S., & Bethge, M. (2015, September 2). A neural algorithm of artistic style. arXiv.org.

•[2] Huang, X., Liu, M. -Y., Belongie, S., & Kautz, J. (2018, August 14). Multimodal unsupervised image -to -image translation. arXiv.org.

•[3] Wikimedia Foundation. (2024, September 9). Neural style transfer. Wikipedia.

•[4] Guardian News and Media. (2022, November 12). Is generative AI really a threat to creative professionals?. The Guardian.

•[5] Rhodes, N. (2021, March 19). YouTube.

•[6] Johnson , J., Alahi, A., & Li, F. (2016, March 27). *1603.08155* . Perceptual Losses for Real-Time Style Transfer and Super-Resolution | Cool Papers - Immersive Paper Discovery.

•[7] Yang, L., Yang, L., Zhao, M., & Zheng, Y. (2018, October). Controlling stroke size in fast style transfer with recurrent convolutional neural network. In Computer Graphics Forum (Vol. 37, No. 7, pp. 97-107).

Reference

•[8] Li, P., Tang, C., Huang, Q., & Li, Z. (2024, May 17). *ART3D: 3d gaussian splatting for text - guided artistic scenes generation* . arXiv.org.

•[9] Subsub. (2016, November 22). Painting like van gogh with Convolutional Neural Networks - sub-subroutine. sub.

•[10] Persson, A. (2020a, July 2). Pytorch Neural Style Transfer Tutorial. YouTube. https://www.youtube.com/watch?v=imX4kSKDY7s&t=343s

•[11] Shorten, C. (2019, February 13). Towards fast neural style transfer. Medium. https://towardsdatascience.com/towards-fast-neural-style-transfer-191012b86284

•[12]Togo, R., Kotera, M., & Ogawa, T. (2022). Text-guided style transfer-based image manipulation using multimodal Generative Models | IEEE Journals & Magazine | IEEE Xplore.

•[13]Albanie, S. (2023a, November 25). Vision Transformer Basics. YouTube. https://www.youtube.com/watch?v=vsqKGZT8Qn8

(2024F) COMP-5112-Research Methodology

Group 10

THANK YOU

Junjun Hu 1264029

Zhijie Shen 1263245