NOVEMBER 2020

Project Report

Neural Style Transfer

PREPARED BY:
AVANEESH (2K18/CO/101)
HARDIK GUPTA (2K18/CO/136)

INDEX

Abstract

The contribution of our project is summarized as follows:

- A method for stroke based image-to-painting translation. We re-frame the stroke prediction as a parameter searching processing. Our method can be jointly optimized with neural style transfer in the same framework.
- Explored the zero-gradient problem on parameter searching and view the stroke optimization from an optimal transport perspective. We introduce a differentiable transportation loss and improves stroke convergence as well as the painting results.
- We design a new neural render-er architecture with a duapathway rendering pipeline . The proposed render-er better deals with the disentanglement of the shape and color and outperforms previous neural renderers with a large margin.

Introduction

In this project we have implemented an image-to-painting translation method that generates vivid and realistic painting art works with controllable styles. Different from previous image-to-image translation methods that formulate the translation as pixel-wise prediction, we deal with such an artistic creation process in a vectorized environment and produce a sequence of physically meaningful stroke parameters that can be further used for rendering. Since a typical vector render is not differentiable, we designed a novel neural render-er which imitates the behavior of the vector render-er and then frame the stroke prediction as a parameter searching process that maximizes the similarity between the input and the rendering output. We explored the zero-gradient problem. Experiments show that the paintings generated by our method have a high degree of fidelity in both global appearance and lo-cal textures. Our implementation can be also jointly optimized with neural style transfer that further transfers visual style from other images

Problem Statement

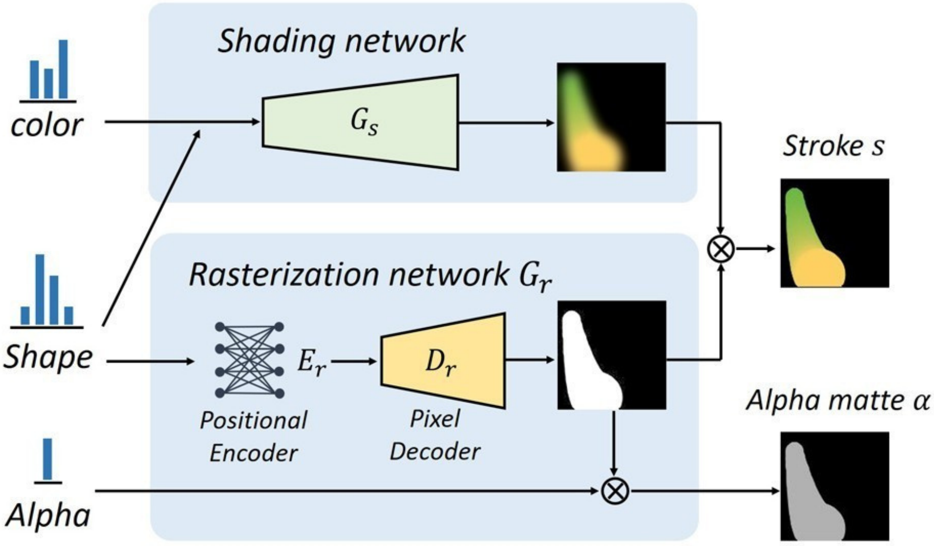
Creating artistic paintings is one of the defining characteristics of humans and other intelligent species. Previous image-to-image translation and style transfer methods typically formulate the translation either as a pixel-wise mapping or a continuous optimization process in their pixel space. However, as an artistic creation process, the paintings usually proceed as a sequentially instantiated process that creates using brushes, from abstract to concrete, and from macro to detail. This process is fundamentally different from how neural networks create artwork that produces pixel-by-pixel results. To fully master the professional painting skills, people usually need a lot of practice and learn domain expertise. Even for a skilled painter with years of practice, it could still take hours or days to create a realistic painting artwork.
In our method, different from the previous stroke-based rendering methods that utilize step-wise greed search , recurrent neural network , or reinforcement learning, we reformulate the stroke prediction as a “parameter searching” process that aims to maximize the similarity between the input and the rendering output in a self-supervised manner. Considering that a typical graphic render is not differentiable, we take advantage of the neural rendering that imitates the behavior of the graphic rendering and make all components in our method differentiable

Our method consists of three modules:

- A neural render-er that is trained to generate strokes given a set of vectorized stroke parameters.
- A stroke blender that combines multiple rendering strokes in a differentiable manner.
- A similarity measurement module that enforces the reconstruction of the input image.

Tools & Resources Used

- **Google Colab**
- **PyTorch**
- **Numpy**
- **Matplotlib**



IMPLEMENTATION CODE

style_transfer.py

```python
import argparse
import torch
torch.cuda.current_device()
import torch.optim as optim

from painter import *
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

parser = argparse.ArgumentParser(description='STYLIZED NEURAL PAINTING')
args = parser.parse_args(args=[])
args.img_path = './test_images/1.png'
args.renderer = 'oilpaintbrush'
args.canvas_color = 'black'
args.canvas_size = 512
args.max_m_strokes = 500
args.m_grid = 5
args.beta_L1 = 1.0
args.with_ot_loss = False
args.beta_ot = 0.1
args.net_G = 'zou-fusion-net'
args.renderer_checkpoint_dir = './checkpoints_G_oilpaintbrush'
args.lr = 0.005
args.output_dir = './output'
args.disable_preview = True

def optimize_x(pt):

    pt._load_checkpoint()
    pt.net_G.eval()

    pt.initialize_params()
    pt.x_ctt.requires_grad = True
    pt.x_color.requires_grad = True
    pt.x_alpha.requires_grad = True
    utils.set_requires_grad(pt.net_G, False)

    pt.optimizer_x = optim.RMSprop([pt.x_ctt, pt.x_color, pt.x_alpha], lr=pt.lr)

    print('begin to draw...')
    pt.step_id = 0
    for pt.anchor_id in range(0, pt.m_strokes_per_block):
        pt.stroke_sampler(pt.anchor_id)
        iters_per_stroke = int(500 / pt.m_strokes_per_block)
        for i in range(iters_per_stroke):

            pt.optimizer_x.zero_grad()

            pt.x_ctt.data = torch.clamp(pt.x_ctt.data, 0.1, 1 - 0.1)
            pt.x_color.data = torch.clamp(pt.x_color.data, 0, 1)
            pt.x_alpha.data = torch.clamp(pt.x_alpha.data, 0, 1)

            if args.canvas_color == 'white':
                pt.G_pred_canvas = torch.ones(
                    [args.m_grid ** 2, 3, pt.net_G.out_size, pt.net_G.out_size]).to(device)
            else:
                pt.G_pred_canvas = torch.zeros(
                    [args.m_grid ** 2, 3, pt.net_G.out_size, pt.net_G.out_size]).to(device)

            pt._forward_pass()
            pt._drawing_step_states()
            pt._backward_x()
```

IMPLEMENTATION CODE

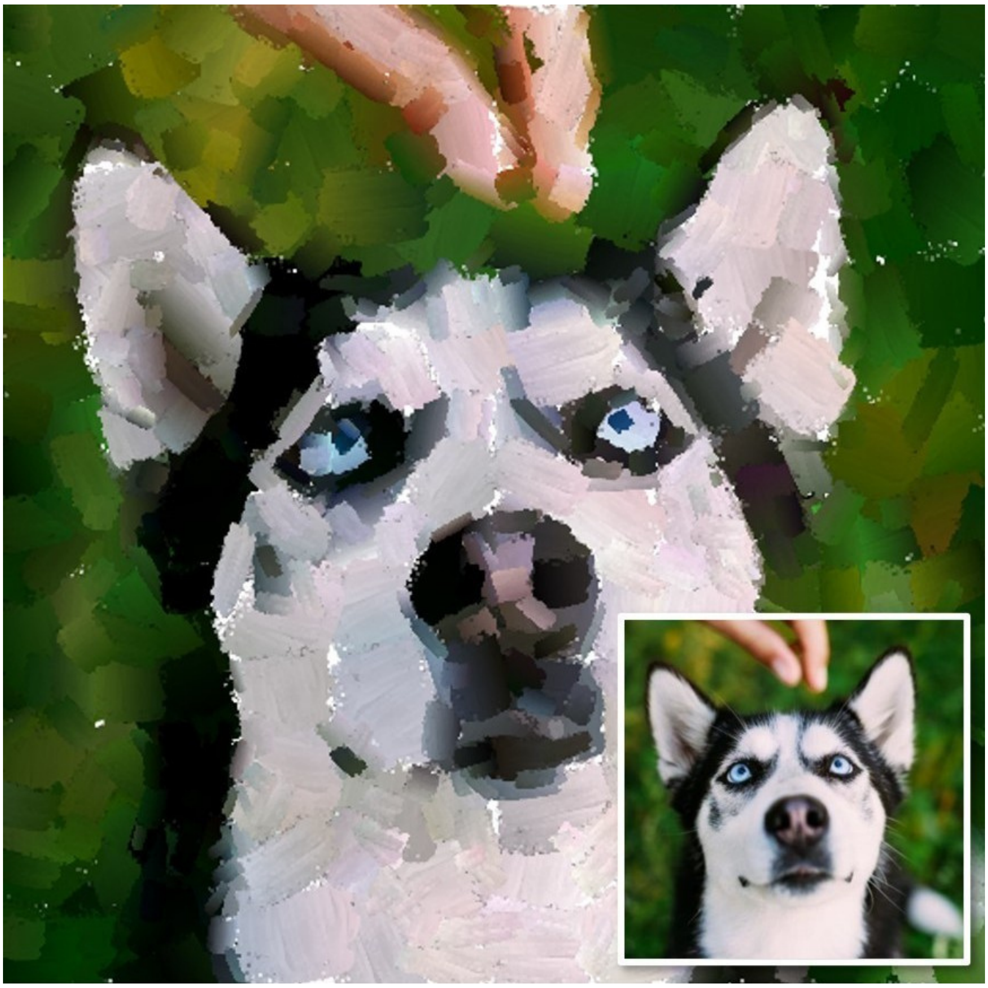
style_transfer.py

```python
            pt._backward_x()
            pt.optimizer_x.step()

            pt.x_ctt.data = torch.clamp(pt.x_ctt.data, 0.1, 1 - 0.1)
            pt.x_color.data = torch.clamp(pt.x_color.data, 0, 1)
            pt.x_alpha.data = torch.clamp(pt.x_alpha.data, 0, 1)

            pt.step_id += 1

    v = pt.x.detach().cpu().numpy()
    pt._save_stroke_params(v)
    v_n = pt._normalize_strokes(pt.x)
    v_n = pt._shuffle_strokes_and_reshape(v_n)
    final_rendered_image = pt._render(v_n, save_jpgs=False, save_video=True)

    return final_rendered_image

pt = Painter(args=args)
final_rendered_image = optimize_x(pt)


args.content_img_path = './test_images/sunflowers.jpg'
args.style_img_path = './style_images/picasso.jpg'
args.vector_file = './output/sunflowers_strokes.npz'
args.transfer_mode = 1 # style transfer mode, 0: transfer color only, 1: transfer both color and
args.beta_L1 = 1.0
args.beta_sty = 0.5
args.net_G = 'zou-fusion-net'
args.renderer_checkpoint_dir = './checkpoints_G_oilpaintbrush'
args.lr = 0.005
args.output_dir = './output'
args.disable_preview = True


def optimize_x(pt):

    pt._load_checkpoint()
    pt.net_G.eval()

    if args.transfer_mode == 0: # transfer color only
        pt.x_ctt.requires_grad = False
        pt.x_color.requires_grad = True
        pt.x_alpha.requires_grad = False
    else: # transfer both color and texture
        pt.x_ctt.requires_grad = True
        pt.x_color.requires_grad = True
        pt.x_alpha.requires_grad = True

    pt.optimizer_x_sty = optim.RMSprop([pt.x_ctt, pt.x_color, pt.x_alpha], lr=pt.lr)

    iters_per_stroke = 100
    for i in range(iters_per_stroke):
        pt.optimizer_x_sty.zero_grad()

        pt.x_ctt.data = torch.clamp(pt.x_ctt.data, 0.1, 1 - 0.1)
        pt.x_color.data = torch.clamp(pt.x_color.data, 0, 1)
        pt.x_alpha.data = torch.clamp(pt.x_alpha.data, 0, 1)

        if args.canvas_color == 'white':
            pt.G_pred_canvas = torch.ones([pt.m_grid*pt.m_grid, 3, pt.net_G.out_size, pt.net_G.ou
        else:
```

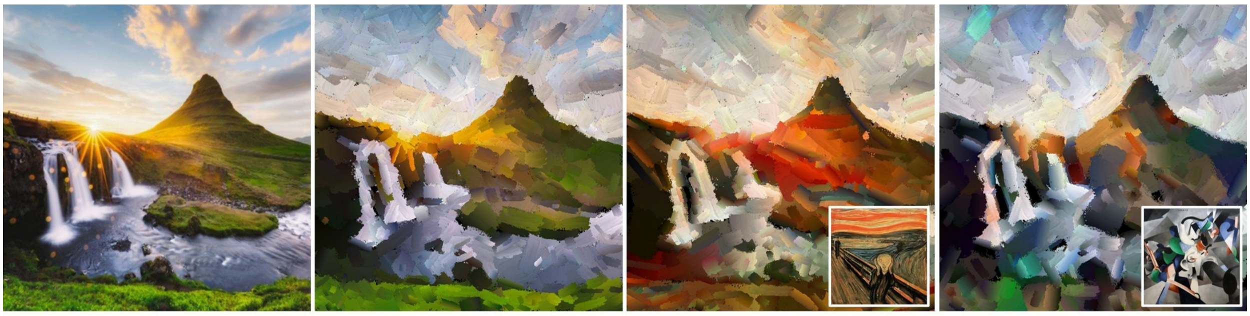
IMPLEMENTATION CODE


```python
        else:
            pt.G_pred_canvas = torch.zeros(pt.m_grid*pt.m_grid, 3, pt.net_G.out_size, pt.net_G.ou

        pt._forward_pass()
        pt._style_transfer_step_states()
        pt._backward_x_sty()
        pt.optimizer_x_sty.step()

        pt.x_ctt.data = torch.clamp(pt.x_ctt.data, 0.1, 1 - 0.1)
        pt.x_color.data = torch.clamp(pt.x_color.data, 0, 1)
        pt.x_alpha.data = torch.clamp(pt.x_alpha.data, 0, 1)

        pt.step_id += 1

    print('saving style transfer result...')
    v_n = pt._normalize_strokes(pt.x)
    v_n = pt._shuffle_strokes_and_reshape(v_n)
    final_rendered_image = pt._render(v_n, save_jpgs=False, save_video=False)
    pt._save_style_transfer_images(final_rendered_image)

    return final_rendered_image


pt = NeuralStyleTransfer(args=args)
final_rendered_image = optimize_x(pt)
```

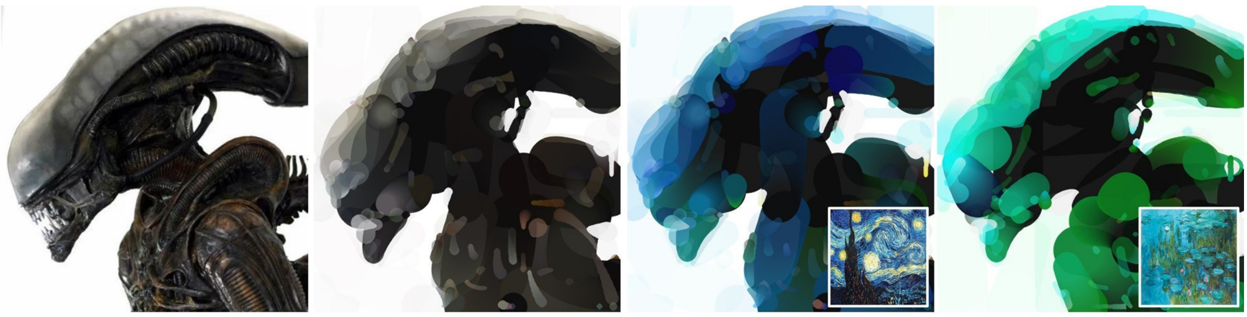
RESULT



RESULT



Input photo | Oil-painting | Stylized outputs (transfer both color & texture)



Input photo | Watercolor painting | Stylized outputs (transfer color only)