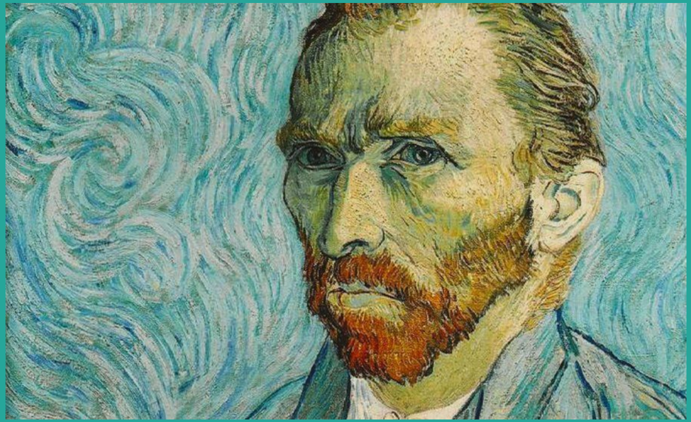
Style Transfer

By Rediet Negash

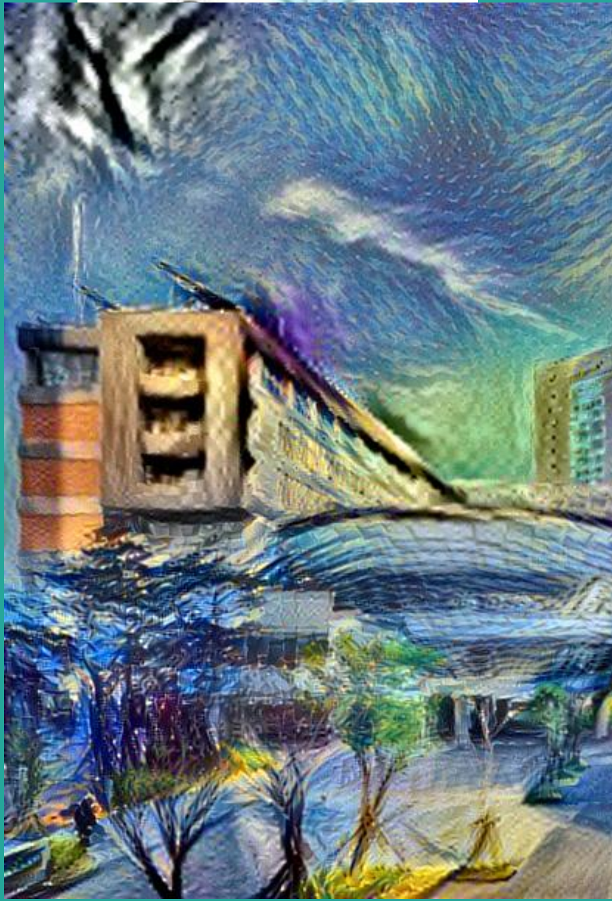
Have you ever dreamed of painting like Van Gogh?



Well, even if u don’t have the
skills to do so, don’t be sad

you still can do this with a super cool subject called Style transfer, in which the style of a piece of artwork is transferred onto a picture.

Eg: SUNY Korea + Starry Night + other style

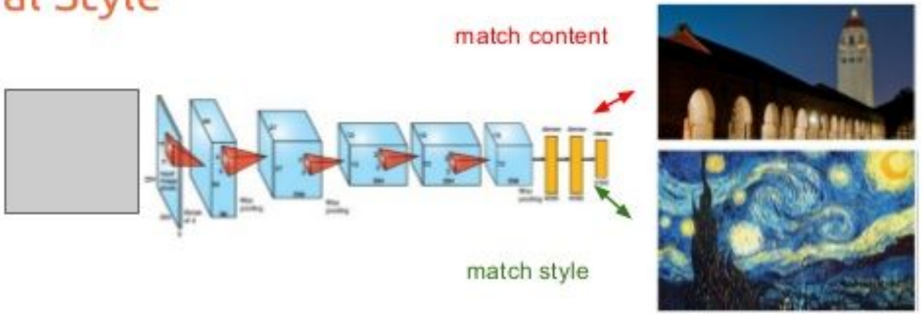


Styles used can be...

How does this work?

Style transfer is an optimization problem where the neural network is not required to do something but rather the backpropagation and optimization was used to slowly alter the image to incorporate style characteristics.

Neural Style



$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Match activations from content image

Match gram matrices from style image

Gatys et al. [Image Style Transfer Using Convolutional Neural Networks](). CVPR 2016

Content loss:

$$\mathcal{L}_{content} = \sum_{l} \sum_{i,j} (\alpha C^{l}_{i,j} - \alpha P^{l}_{i,j})^2,$$

Style Loss

Similar with content loss computation
except instead of comparing the raw outputs of the style and the target images at various layers, we compare the **Gram matrices** of the outputs.

A Gram matrix results from multiplying a matrix with the transpose of itself:

$$G^l_{i,k} = \sum_k F^l_{i,k} F^l_{j,k}$$

Why Gram Matrix

Since every column is multiplied with every row in the matrix, the spatial information that was contained in the original representations could be considered to have been "distributed". The Gram matrix instead contains non-localized information about the image, such as texture, shapes, and weights - which is what we want for style transfer

$$\mathcal{L}_{style} = \sum_{l} \sum_{i,j} (\beta G_{i,j}^{s,l} - \beta G_{i,j}^{p,l})^2$$

Experiments

Objective

- Improve aesthetics of a target Image
- Multiple styles
- Photo style transfer
- Improve Denoising resulted

Approaches used

Hyper parameter tuning

1, Doing different number of Iterations that results in a desired image

2, Changing the convolutional layers used

- For the style layers
- For content layers

3, Changing the weights of the style and content losses (alpha and beta)

4, changing the optimizers hyper parameters- learning_rate

Frameworks used

Torch

VGG19

Torchvision:-

Python Imaging Library(PIL).

For content comparison I used activation/feature map at some deeper layer conv4_2 and conv4_2 layers prior to the output(softmax) layer.

Adam Optimizer

```
In [36]: import torch
         from PIL import Image
         from torchvision import transforms, models
         import numpy as np
         import matplotlib.pyplot as plt
```

In [37]:
```
model= models.vgg19(pretrained=True).features


# freeze the parameter that we don't use to minimize computation and memeory
for parameter in model.parameters():
    parameter.require_grad = False

#print(model)

# extract features
def extract_features(image, model):
    features = {}
    layers = {
        '0' : 'conv1_1',
        '5' : 'conv2_1',
        '10': 'conv3_1',
        '19': 'conv4_1',
        '21': 'conv4_2', # for measuring content loss
        '28': 'conv5_1'
    }
    feature_image = image
    # apply singleton dimention image
    feature_image = feature_image.unsqueeze(0)
    # pass the image to layers in the model
    for key, layer in model._modules.items():
        feature_image = layer(feature_image)
        # if the current layer in the model is part of the layers
        # planned to use add activated feature to the features dictionary
        if key in layers:
            features[layers[key]] = feature_image
            #print(feature_image)

    return features
```

VGG model

Mode layers

Sequential(

(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU(inplace=True)

(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU(inplace=True)

(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(6): ReLU(inplace=True)

(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(8): ReLU(inplace=True)...

model._modules.items()

[('0', Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))), ('1', ReLU(inplace=True)), ('2', Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))), ('3', ReLU(inplace=True)), ('4', MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)), ('5', Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))), ('6', ReLU(inplace=True)), ('7', Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))), ('8', ReLU(inplace=True)), ('9', MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)), ('10', Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))), ('11', ReLU(inplace=True)),...

Helper functions used

```
transform = transforms.Compose([transforms.Resize(300),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5,0.5, 0.5),(0.5,0.5,0.5))])



def tensor_to_Img(tensor):
    img = tensor.clone().detach().numpy().squeeze()
    img = img.transpose(1,2,0)
    img = img*np.array((0.5,0.5,0.5)) + np.array((0.5,0.5,0.5))
    return img
def img_show(content, style):
    fig, (ax1,ax2) = plt.subplots(1,2)
    ax1.imshow(tensor_to_Img(content), label = "Content")
    ax2.imshow(tensor_to_Img(style), label = "Style")
    plt.show()
```

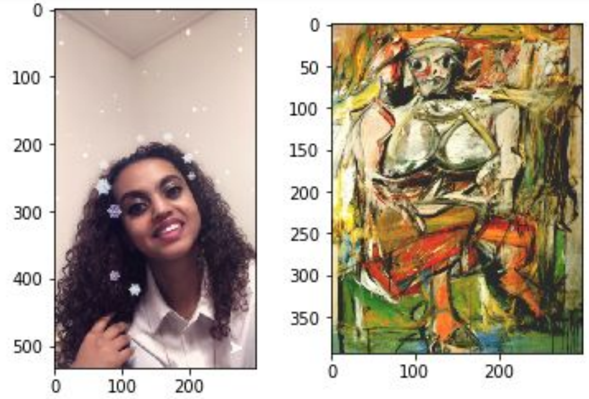
```python
def create_style_loss(style, output_features):

    style_features = extract_features(style,model)
    style_weight = {"conv1_1" : 0.4,
                    "conv2_1" : 0.3,
                    "conv3_1" : 0.2,
                    "conv4_1" : 0.2,
                    "conv5_1" : 0.2



#                     "conv1_1" : 1.0,
#                     "conv2_1" : 0.8,
#                     "conv3_1" : 0.4,
#                     "conv4_1" : 0.2,
#                     "conv5_1" : 0.1
                   }
    # compute the correlation (gram matrix)
    gram_layers = {}
    for layer in style_features:
        gram_layers[layer]= gram_matrix(style_features[layer])

    # calculate style loss
    style_loss = 0
    for layer in style_weight:
        style_gram = gram_layers[layer]

        output_gram = output_features[layer]
        # used for normalization
        _,depth, weight, height = output_gram.shape
        output_gram = gram_matrix(output_gram)
        # squared mean of the output and the style gram
        style_loss += (style_weight[layer]*torch.mean((output_gram-style_gram)**2))/depth*weight*height


    return style_loss
```

```python
def gram_matrix(feature):
    t, depth, height, weight = feature.size()
    #input = torch.Tensor(2, 4, 3) # input: 2 x 4 x 3
    #print(input.view(1, -1, -1, -1).size()) # prints - torch.size([1, 2, 4, 3])

    feature = feature.view(depth, height*weight)
    # multiply the matrix with its transpose
    gram_matrix = torch.mm(feature, feature.t())

    return gram_matrix

def create_content_loss(content, output_features):

    content_features = extract_features(content,model)
    content_loss = torch.mean((content_features['conv4_2']-output_features['conv4_2'])**2)
    return content_loss
```

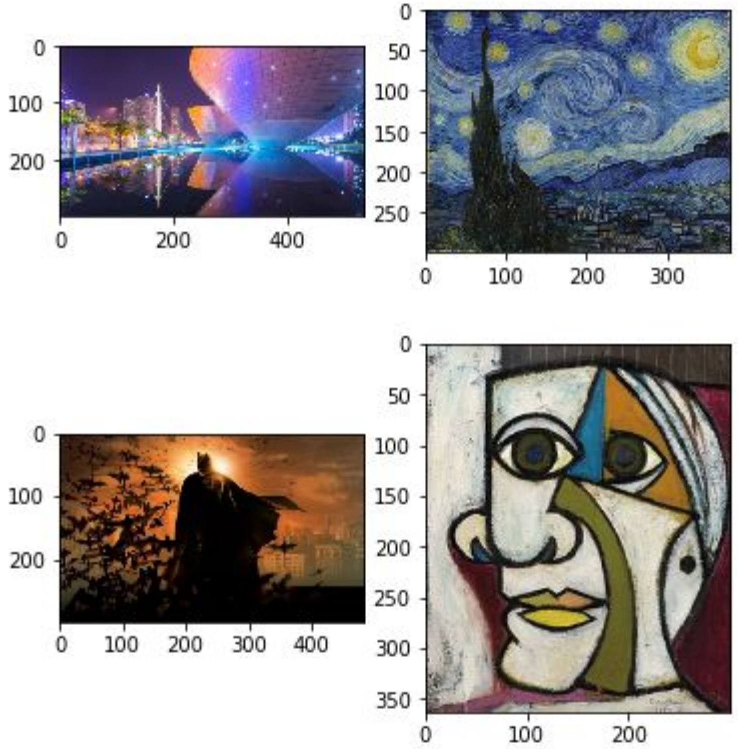
```python
def main():
    # alpha
    content_wt = 5
    # beta
    style_wt = 1e9
#     style_wt2 = 1e6
#     style_wt3 = 1e6

    print_after = 100
    epochs = 2000

    content = Image.open("img_red.jpg").convert("RGB")
    content = transform(content)

    style = Image.open("1-style.jpg").convert("RGB")
    style = transform(style)

#     style2 = Image.open("2-style1.jpg").convert("RGB")
#     style2 = transform(style2)

#     style3 = Image.open("style.jpg").convert("RGB")
#     style3 = transform(style3)


    img_show(content, style)
#     img_show(style3, style2)

    #output is initalized from the content first
    output = content.clone().requires_grad_(True)

    # optimize the output image
    # this is a hyper parameter
    optimizer = torch.optim.Adam([output],lr=0.001)
    # lr - learning rate, 0.07
    # activate output features
    for i in range(1, epochs + 1):
        output_features = extract_features(output, model)

        content_loss = create_content_loss(content, output_features)
        style_loss = create_style_loss(style, output_features)
#         style_loss2 = create_style_loss(style2, output_features)
#         style_loss3 = create_style_loss(style3, output_features)
```

```python
    # activate output features
    for i in range(1, epochs + 1):
        output_features = extract_features(output, model)

        content_loss = create_content_loss(content, output_features)
        style_loss = create_style_loss(style, output_features)
#         style_loss2 = create_style_loss(style2, output_features)
#         style_loss3 = create_style_loss(style3, output_features)

        total_loss = content_wt*content_loss + style_wt*style_loss
#         total_loss = content_wt*content_loss + style_wt*style_loss + st

        if i% 10 == 0:
            print("epoch ", i, " ", total_loss)

        optimizer.zero_grad()
        total_loss.backward()  # This calculates the gradients
        optimizer.step()    # This updates the net

        if i % print_after == 0:
            plt.imshow(tensor_to_Img(output),label="Epoch "+str(i))
            plt.show()
           # plt.imsave(str(i)+'.png',tensor_to_Img(output),format='png')

if __name__ == '__main__':
    main()
```

```python
        if i % print_after == 0:
            plt.imshow(tensor_to_Img(output),label="Epoch "+str(i))
            plt.show()
           # plt.imsave(str(i)+'.png',tensor_to_Img(output),format='png')

if __name__ == '__main__':
    main()
```



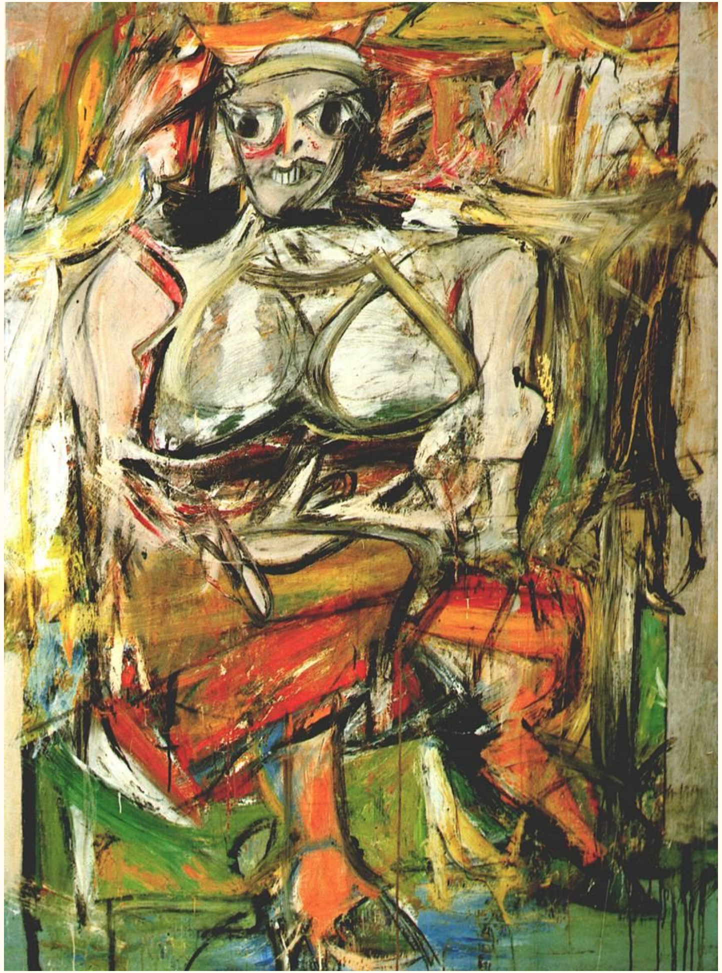
```
epoch  10   tensor(4.7925e+19, grad_fn=<AddBackward0>)
epoch  20   tensor(4.3304e+19, grad_fn=<AddBackward0>)
epoch  30   tensor(3.8863e+19, grad_fn=<AddBackward0>)
epoch  40   tensor(3.4605e+19, grad_fn=<AddBackward0>)
```



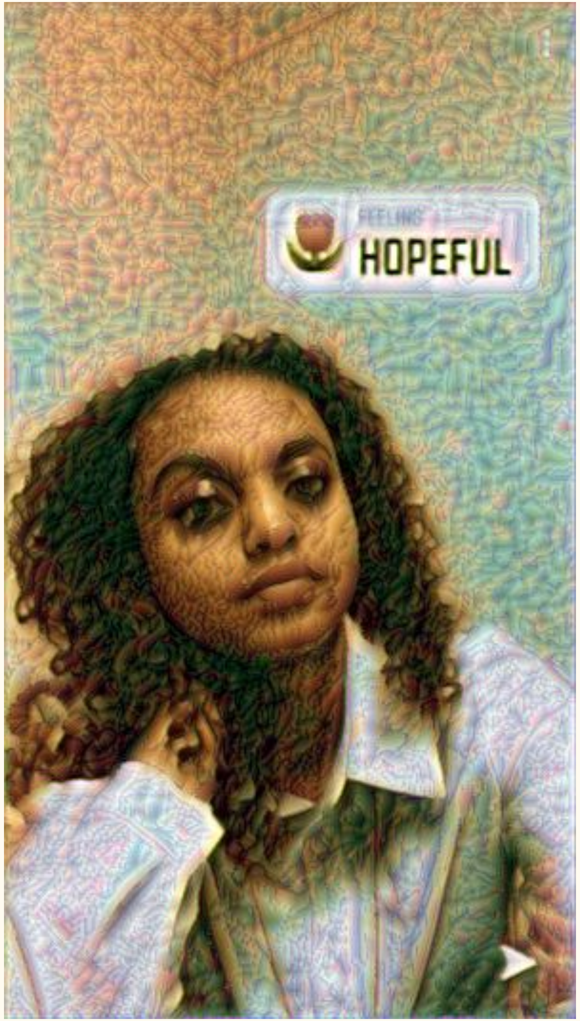
Some outputs

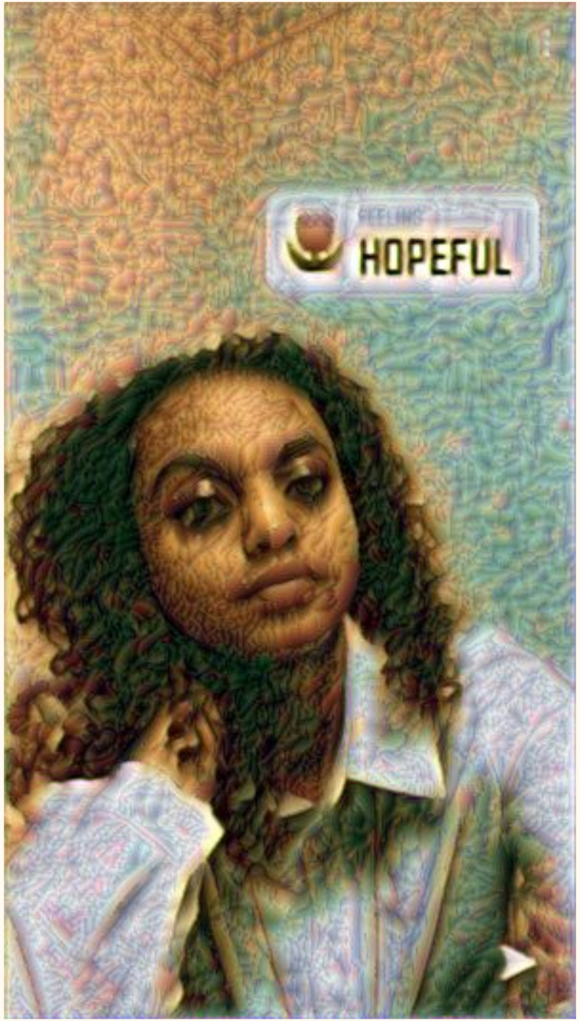


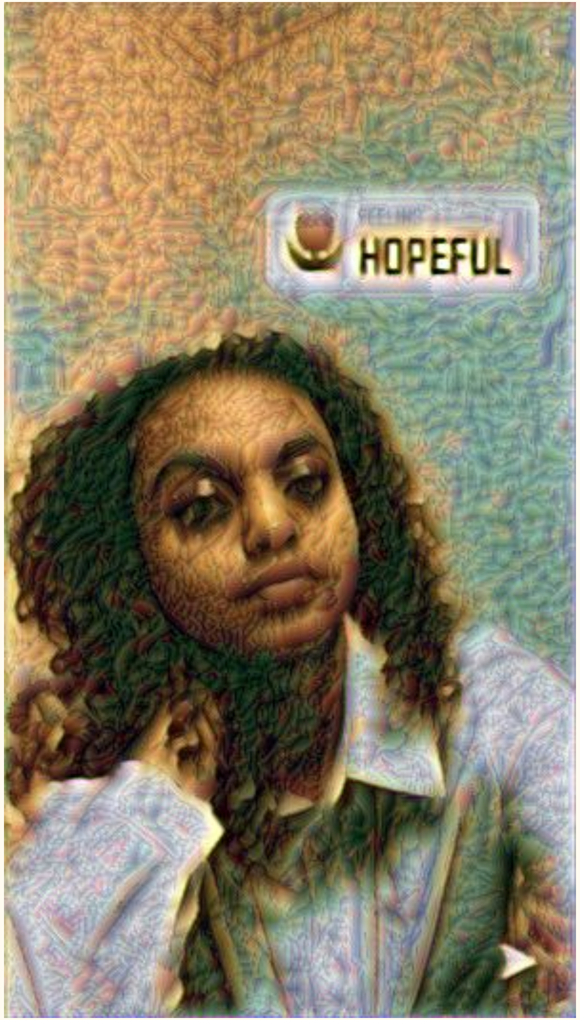

More Iterations with multiple styles




FEELING
HOPEFUL

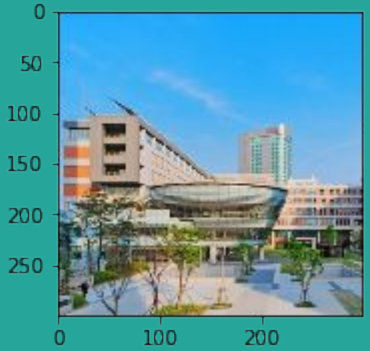
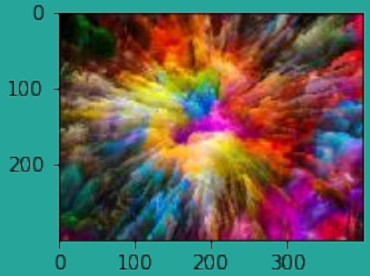
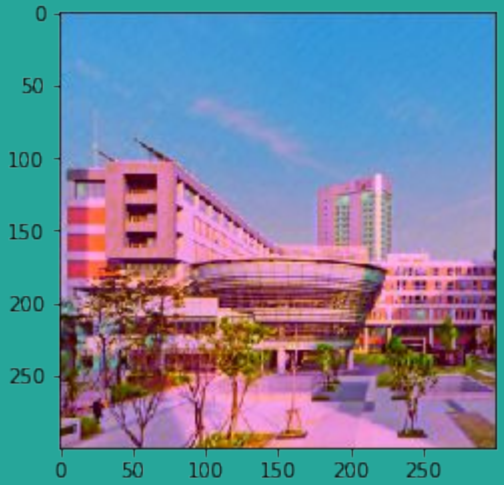
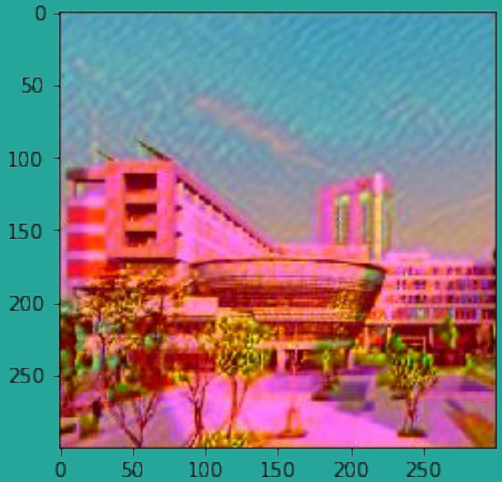
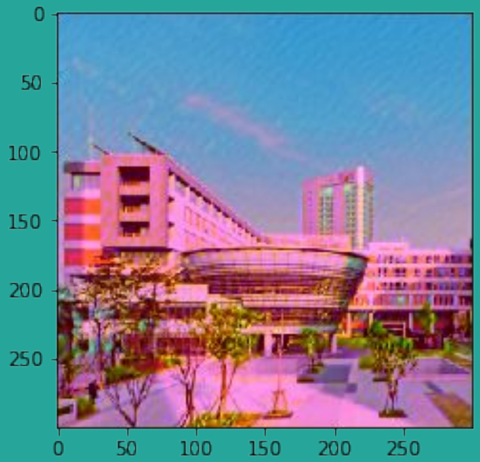
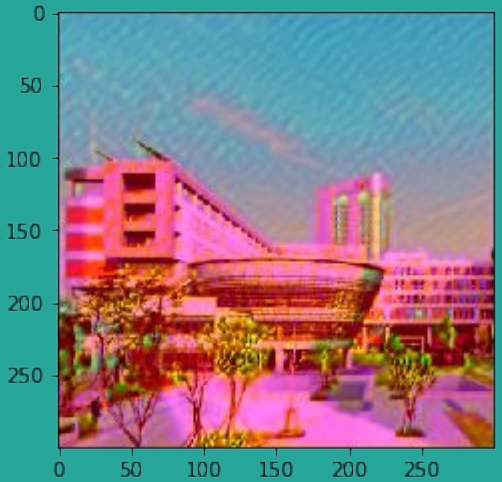


HOPEFUL

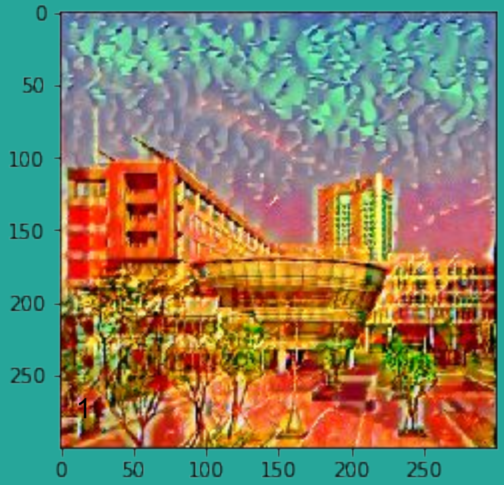
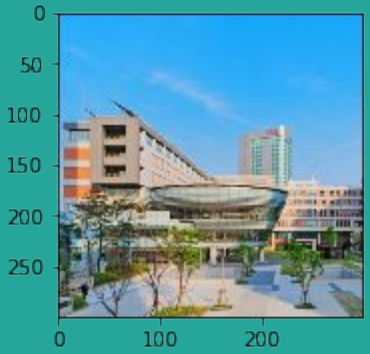
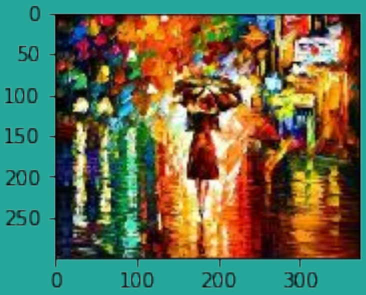
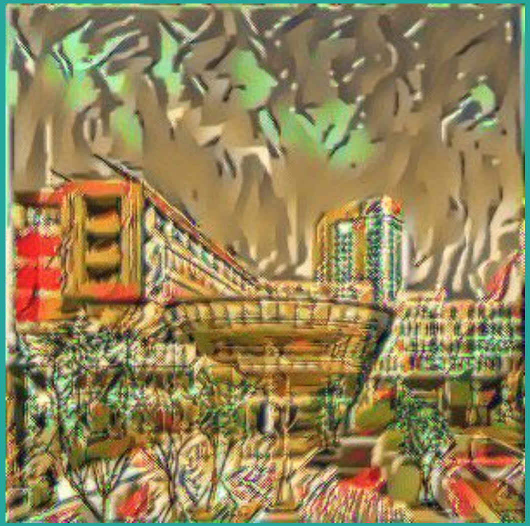

HOPEFUL


HOPEFUL

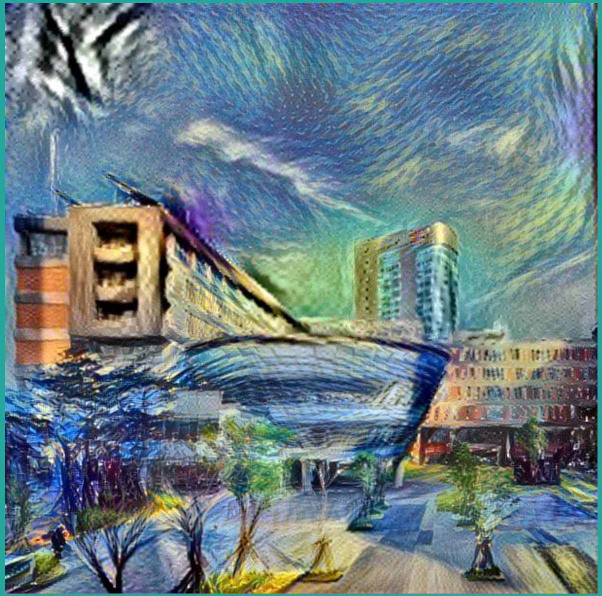

PRIORITY
MAIL
VISIT US AT USPS.COM

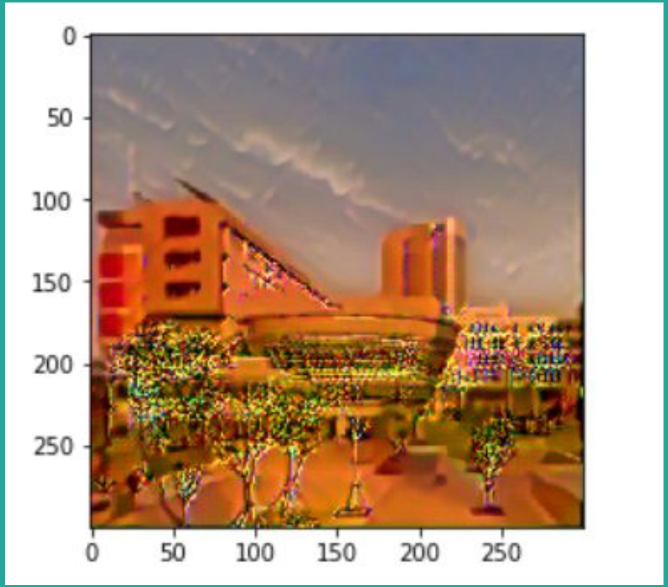
Photo realistic style transfer


0
50
100
150
200
250
0
100
200

0
100
200
0
100
200
300

0
50
100
150
200
250
0
50
100
150
200
250

0
50
100
150
200
250
0
50
100
150
200
250

0
50
100
150
200
250
0
50
100
150
200
250

0
50
100
150
200
250
0
50
100
150
200
250


0
50
100
150
200
250
0 50 100 150 200 250

0
50
100
150
200
250
0 100 200

0
50
100
150
200
250
0 100 200 300


Two Styles applied





Future and current works

- Feedforward Method
- Arbitrary Style Transfer
- Improve computational time
- Improve aesthetics of generated imageF

Conclusion

- The quality of a style transferred image highly depends on the hyper parameters and the convolutional layers used.
- Using higher layers for content and lower layers for style does give a good quality of a generated image. Again, quality is subjective.

Multiple styles- Getting a good neural styled image does require an artistic mind where the content and the style image should be carefully chosen to get the best results.

*** future work teaching the algorithm to predict best style combination for synthesising the best combination of styles…

Thank you

Presented by Rediet Negash