Image Style Transfer Using Convolutional Neural Networks

A

Project Report

submitted in partial fulfillment of the

requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

Specialization in

BUSINESS ANALYTICS AND OPTIMIZATION

by

Name	Roll No.
VIMANSHU RAJ CHANDRA	R103216116
VAIBHAV SRIVASTAVA	R103216113
SURBHI MEHTA	R103216131
SHINJANA MISRA	R103216128

under the guidance of

Dr. Tanupriya Choudhury

Associate Professor, Department of Informatics

Department of Informatics

School of Computer Science

University of Petroleum & Energy Studies

Bidholi, Via Prem Nagar, Dehradun, UK

May – 2019

CANDIDATE’S DECLARATION

We hereby certify that the project work entitled **“ Image Style Transfer Using Convolutional Neural Networks”** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in BAO and submitted to the Department of Informatics at School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **January**, **2019** to **May**, **2019** under the supervision of **Dr. Tanupriya Choudhury, Associate Professor, Department of Informatics**.

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

Vimanshu Raj Chandra	**Vaibhav Srivastava**	**Surbhi Mehta**	**Shinjana Misra**
R103216116	**R103216113**	**R103216131**	**R103216128**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 19/05/2019

Dr. Tanupriya Choudhury
Project Guide

Dr. T.P. Singh
Head – Department of Informatics
School of Computer Science
University of Petroleum & Energy Studies
Dehradun – 248 001 (Uttarakhand)

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Dr. Tanupriya Choudhury**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank to our Head of the Department, **Dr. T.P. Singh**, for his great support in doing our project in **Image Style Transfer Using Convolutional Neural Networks** at **SoCS**.

We are also grateful to **Dr. Manish Prateek Professor and Director, SoCS**, UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our **friends** for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support they have given us.

Name	**Vimanshu Raj Chandra**	**Vaibhav Srivastava**	**Surbhi Mehta**	**Shinjana Misra**
Roll No.	**R103216116**	**R103216113**	**R103216131**	**R103216128**

ABSTRACT

Image processing is very difficult to render using conventional methods as they do not provide image representations for explicit semantic information to allow separation of content and style of images to be processed individually for better processed images. Convolutional Neural Networks are capable of making high-level semantic information of images explicit by its capability of rendering image representations.[5] Further, A Neural Algorithm of Artistic Style is used in this project to separate content and style of images and later club them to produce new images with minimal content and style losses. This project looks into the potential of this method for high level image processing and manipulation via deep image representations. Also, the concept of Data Augmentation has been looked at in this project.

Keywords: Image processing, Semantic information, Data Augmentation

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Convolutional Neural Networks

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.[7]

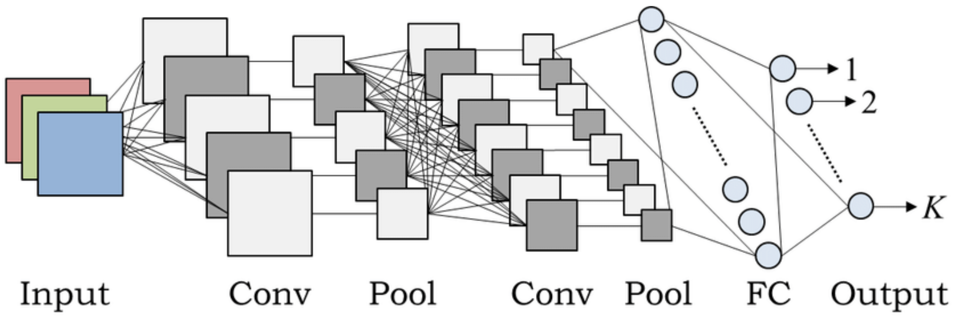


Fig 1.1 An example of CNN architecture [8]

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:[7]

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

1.2 CNN for Image Styling

Convolutional Neural Networks trained with sufficient labeled data on specific tasks such as object recognition learn to extract high-level image content in generic feature representations that generalise across datasets[2] and even to other visual information processing tasks[1] including texture recognition and artistic style classification.

CNN is capable of high-level explicit image representations because of its many layers. It allows both content and style representations of the respective images.

1.3 Elements Used

Content Representation: The feature responses in higher layers of the network as the content representation.[3]

Content Loss: The squared-error loss between the two feature representations.[3]

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F^l_{ij} - P^l_{ij}\right)^2 \qquad (1)$$

where, ~p and ~x be the original image and the image that is generated, and P^l and F^l their respective feature representation in layer l.

Style Representation: To obtain a representation of the style of an input image, we use a feature space designed to capture texture information.[3][5]

Gram Matrices: Feature correlations are given by the Gram matrices.[3][5]

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}. \qquad (2)$$

where G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l:

Style Loss: Let ~a and ~x be the original image and the image that is generated, and A^l and G^l their respective style representation in layer l. The contribution of layer l to the total loss is then[3][5]

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - A^l_{ij}\right)^2 \qquad (3)$$

Total Loss:[3][5]

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) \qquad (4)$$

We will use A Neural Algorithm of Artistic Style to perform image style transfer [5]. The Algorithm is described in detail in the Methodology section below.

1.4 Pert Chart

The PERT chart given below shows the time required for the project to end, and illustrates the schedule that will be followed for the proper commencement of the project.

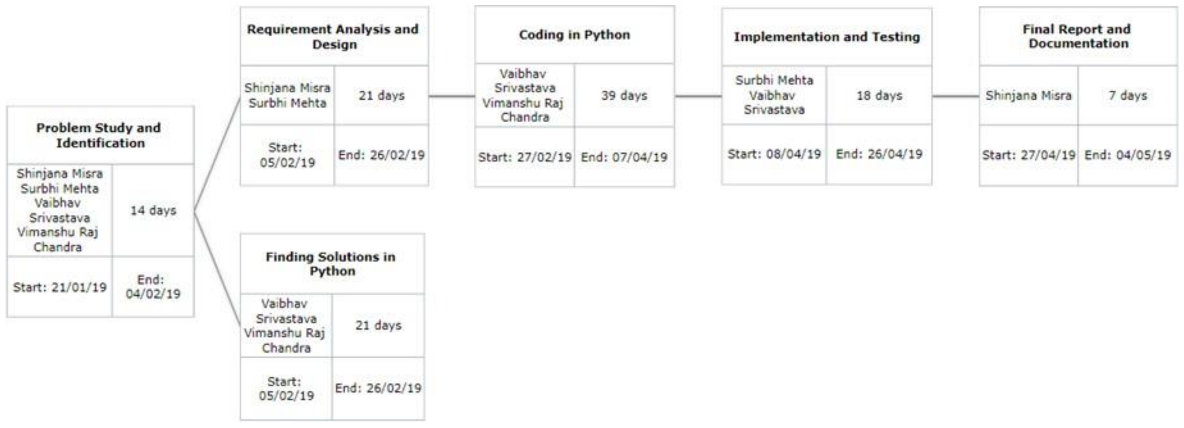

Problem Study and Identification
Shinjana Misra Surbhi Mehta Vaibhav Srivastava Vimanshu Raj Chandra
14 days
Start: 21/01/19
End: 04/02/19
Requirement Analysis and Design
Shinjana Misra Surbhi Mehta
21 days
Start: 05/02/19
End: 26/02/19
Finding Solutions in Python
Vaibhav Srivastava Vimanshu Raj Chandra
21 days
Start: 05/02/19
End: 26/02/19
Coding in Python
Vaibhav Srivastava Vimanshu Raj Chandra
39 days
Start: 27/02/19
End: 07/04/19
Implementation and Testing
Surbhi Mehta Vaibhav Srivastava
18 days
Start: 08/04/19
End: 26/04/19
Final Report and Documentation
Shinjana Misra
7 days
Start: 27/04/19
End: 04/05/19

Fig 1.2 Pert Chart describing project workflow

2. LITERATURE REVIEW

The process of image processing can be made better and more efficient if its semantic information can be extracted explicitly. [1][2]

Convolutional Neural Networks (CNN) is a Deep Learning algorithm[7][8] and provides a solution for this problem as they have the ability to discern and produce image representations both in content and style.

A Neural Algorithm to Artistic Style[5][6] helps in combining content and styles with minimal style and content loss thereby keeping the originality of the images intact at the same time along with producing new image with the combined elements of the two.

The Iterative Waterfall Model, its requirements and its uses are explained well in this reference.[4]

3. PROBLEM STATEMENT

Conventional methods of image processing are taxing and not efficient enough to provide image representations that explicitly show semantic information to allow separation of image content from the image style.

4. OBJECTIVES

1. To use CNN to provide image representations to make the images information explicit.
2. To implement A Neural Algorithm of Artistic Style to combine separate image content and image styles.
3. To combine style of image into another image without losing the content.

5. DESIGN METHODOLOGY

5.1 Workflow

Following is the methodology involved for the real-life implementation of the project[5][6]:

1. Take an input image (image in which style is to be applied).
2. Take a style image (image’s style is to be applied to the previous image).
3. Resize them into the same size and shape.
4. Load a Convolutional Neural Network (CNN).

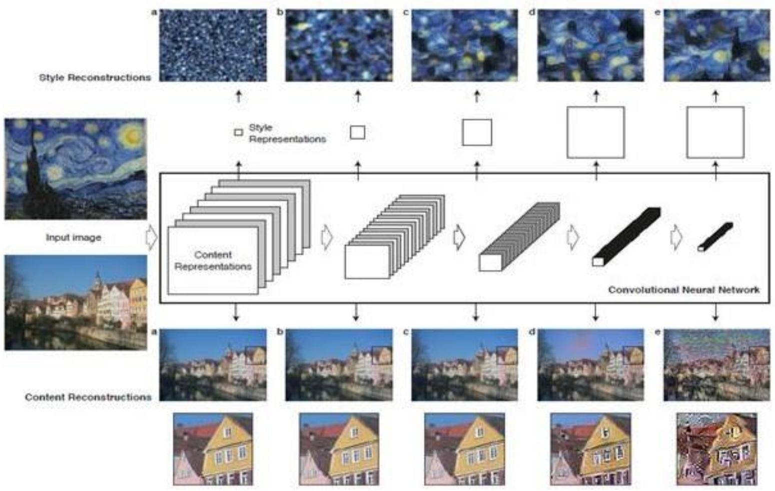


Fig 5.1 Image Representations in CNN

5. Distinguish between the layers that represent the style (basic shapes, colour etc.), and content (image specific features) of the images respectively and separate them.
6. Implement A Neural Algorithm of Artistic Style to extract and store the content and style representations and then transfer the style of the second image onto the first one.
7. Also calculate and optimize content loss (because content is to be preserved), style loss (because style has to be transferred), and total variation loss (because output image has to be de-noised).

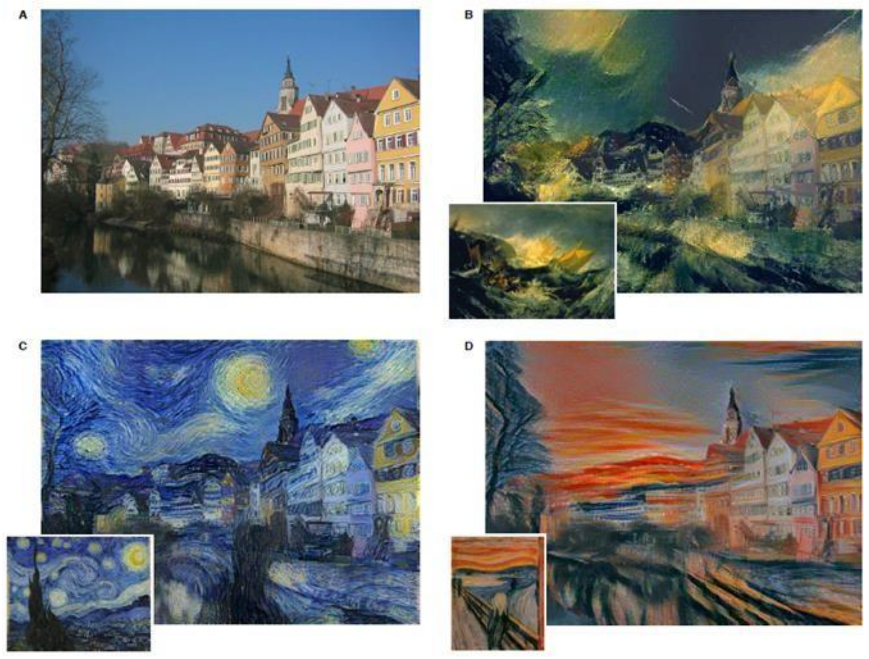


Fig 5.2 Images with context and style mixed

5.2 Software Development Model Used

The project uses the Iterative Waterfall Model for software development because it is a small project and there's no uncertainty of requirements, Also, its sequential design process which leads to no overlapping of different phases which ensures proper departmentalization and control.[4]

The illustration below shows the different phases of the Waterfall Model:

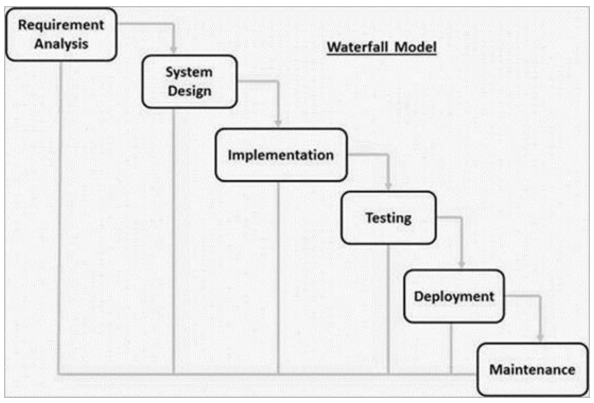
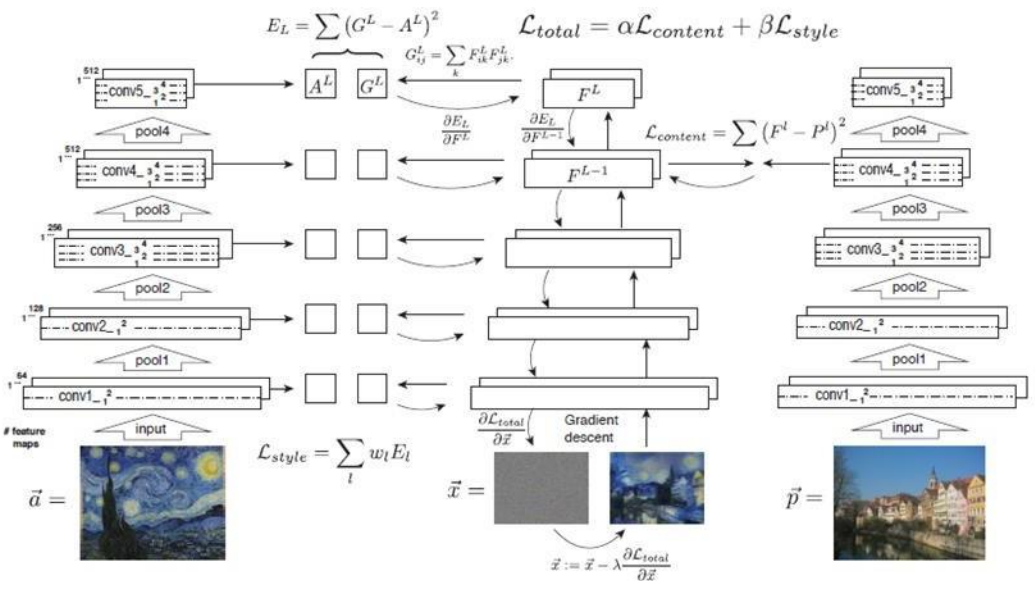


Fig. 5.3 Phases of the Iterative Waterfall Model

5.3 The Algorithm

1. Start with VGG with average pooling (they saw better gradient flow compared to max pooling).
2. Take a content image, a style image, and a target image initialized to random pixel values.
3. Define a content loss (MSE of the feature activations for the content image and the target image)
4. And a style loss (MSE of the Gram matrices of the style image and the target image).
5. Run SGD, optimizing a linear combination of the losses by modifying the pixel values of the target image.

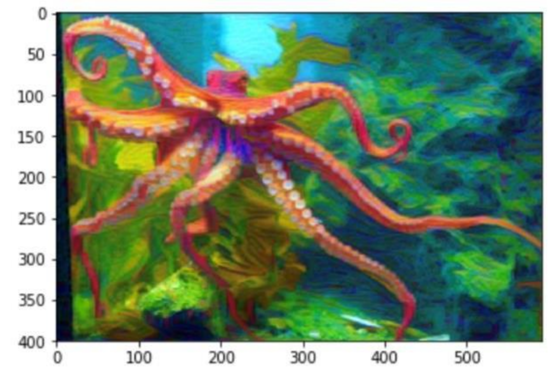


$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_
conv4_
conv3_
conv2_
conv1_
pool4
pool3
pool2
pool1
input
# feature maps
512
256
128
64
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

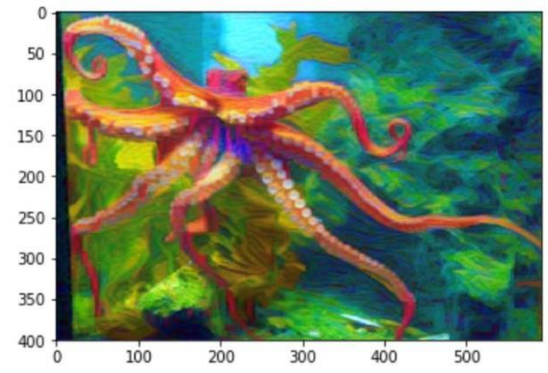
Fig. 5.4 A Neural Algorithm of Artistic Style

6. IMPLEMENTATION

6.1 Output Screens



```
Total loss:  3145412.75
```



```
Total loss:  2023719.125
```

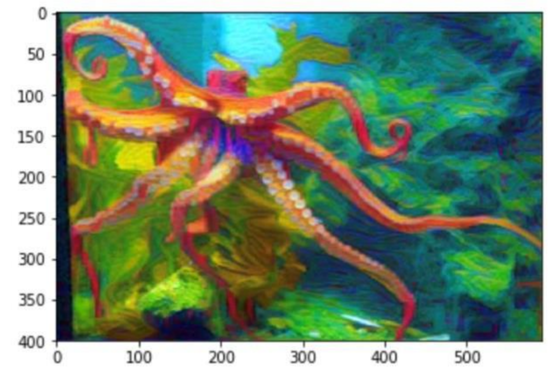


Fig. 6.1 Optimizing the total loss of the images

```
Out[15]: <matplotlib.image.AxesImage at 0x7f0b3b442e80>
```

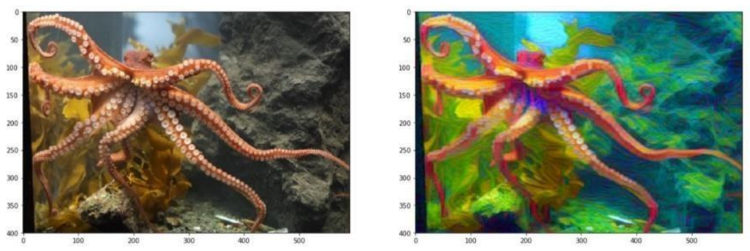


Fig. 6.2 The final output image

6.2 Use Case Diagram

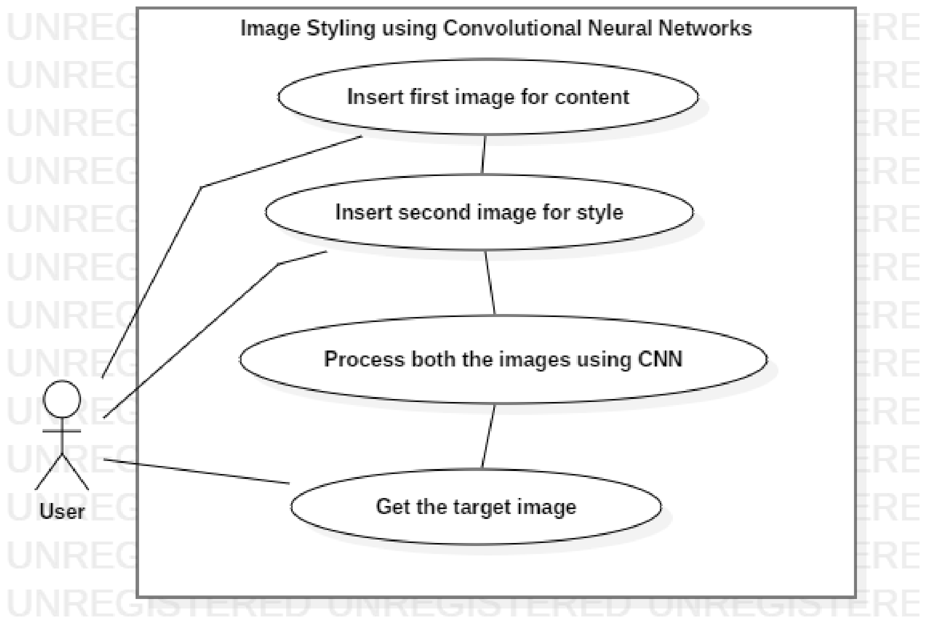


Image Styling using Convolutional Neural Networks
Insert first image for content
Insert second image for style
Process both the images using CNN
Get the target image
User

Fig. 6.3 Use Case depicting the functioning of the system

6.3 Result Analysis

The result shows the implemented A Neural Algorithm of Artistic Style to separate the content and style of the images to form CNN VGG19 network to club the content and style of two different images to form a new image.

7. CONCLUSION AND FUTURE SCOPE

A Neural Algorithm of Artistic Style is used to break down the images into layers to form the CNN VGG19 network for explicit style and content separation to be used to create a new image containing the features of both the images (content from one and style from the other) with minimal losses.

The project can be further implemented to include photo motion and data augmentation in it.

APPENDIX-I: PROJECT CODE SNIPPETS

Module 1: Import of all the basic resources and PyTorch framework

```
# import resources
```

```
%matplotlib inline

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

import torch

import torch.optim as optim

from torchvision import transforms, models
```

Module 2: Load in VGG19-features for feature extraction in the images

```
# get the "features" portion of VGG19

vgg = models.vgg19(pretrained=True).features

for param in vgg.parameters():

    param.requires_grad_(False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

vgg.to(device)
```

Module 3: Load in content and style images

```
def load_image(img_path, max_size=400, shape=None):

    ''' Load in and transform an image, making sure the image

       is <= 400 pixels in the x-y dims.'''


 image = Image.open(img_path).convert('RGB')


    #optimize image size

    if max(image.size) > max_size:

        size = max_size
    else:
        size = max(image.size)


    if shape is not None:
        size = shape


    in_transform = transforms.Compose([
                        transforms.Resize(size),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),
                                             (0.229, 0.224, 0.225))])
    image = in_transform(image)[:3,:,:].unsqueeze(0)


    return image
# load in content and style image
content = load_image('images/octopus.jpg').to(device)
# Resize style to match content
style = load_image('images/hockney.jpg', shape=content.shape[-2:]).to(device)
def im_convert(tensor):
    """ Display a tensor as an image. """


    image = tensor.to("cpu").clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.406))
    image = image.clip(0, 1)


    return image
# display the images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
# content and style ims side-by-side
ax1.imshow(im_convert(content))
ax2.imshow(im_convert(style))
```

Module 4,5: Normalization of the images and to get the content and style representation of an image by passing it forward into VGG19 network

```
# print out VGG19 structure


print(vgg)
def get_features(image, model, layers=None):
    """ Run an image forward through a model and get the features for
        a set of layers. Default layers are for VGGNet matching Gatys et al (2016)
    """
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  '21': 'conv4_2', ## content representation
                  '28': 'conv5_1'}
    features = {}
    x = image
    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features
def gram_matrix(tensor):
    """ Calculate the Gram Matrix of a given tensor
    """

    # get the batch_size, depth, height, and width of the Tensor
    _, d, h, w = tensor.size()

    tensor = tensor.view(d, h * w)

    # calculate the gram matrix
    gram = torch.mm(tensor, tensor.t())

    return gram
# get content and style features
content_features = get_features(content, vgg)
```

```
style_features = get_features(style, vgg)


# calculate the gram matrices for each layer of our style representation

style_grams = {layer: gram_matrix(style_features[layer]) for layer in style_features}


# create a third "target" image and prep it for change

target = content.clone().requires_grad_(True).to(device)
```

Module 6: Defining content_weight and style_weight

```
# weights for each style layer

style_weights = {'conv1_1': 1.,

                 'conv2_1': 0.75,

                 'conv3_1': 0.2,

                 'conv4_1': 0.2,

                 'conv5_1': 0.2}


content_weight = 1 # alpha

style_weight = 1e6 # beta
```

Module 7: Define content, style and total losses

```
# for displaying the target image

show_every = 400


optimizer = optim.Adam([target], lr=0.003)

steps = 2000 # decide how many iterations to update your image (5000)
```

```
for ii in range(1, steps+1):

    # get the features from your target image
    target_features = get_features(target, vgg)

    # the content loss
    content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2'])**2)

    # the style loss
    # initialize the style loss to 0
    style_loss = 0
    # add to it for each layer's gram matrix loss
    for layer in style_weights:
        # get the "target" style representation for the layer
        target_feature = target_features[layer]
        target_gram = gram_matrix(target_feature)
        _, d, h, w = target_feature.shape
        # get the style representation
        style_gram = style_grams[layer]
        # the style loss for one layer, weighted appropriately
        layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram)**2)
        # add to the style loss
        style_loss += layer_style_loss / (d * h * w)
    # calculate the total loss
    total_loss = content_weight * content_loss + style_weight * style_loss


    # update your target image
    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()


    # display intermediate images and print the loss
    if ii % show_every == 0:
        print('Total loss: ', total_loss.item())
        plt.imshow(im_convert(target))
        plt.show()
```

Module 8: Display the target image

```
# display content and final, target image
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(30, 40))
ax1.imshow(im_convert(content))
ax2.imshow(im_convert(target))
```

REFERENCES

[1] M. Berning, K. M. Boergens, M. Helmstaedter, "SegEM: Efficient Image Analysis for High-Resolution Connectomics", Neuron, vol. 87, no. 6, pp. 1193-1206, Sept. 2015.

[2] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, T. Darrell, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", Oct. 2013, [online] Available:

[3] L. A. Gatys, A. S. Ecker, M. Bethge, "Texture Synthesis Using Convolutional Neural Networks", Advances in Neural Information Processing Systems, vol. 28, 2015.

[4] K.K. Aggarwal, Y. Singh. Software Engineering: Third Edition. Dariya Ganj, New Delhi, Delhi : New Age International, January 2008.

[5] "Image Style Transfer Using Convoluted Neural Networks." Internet: https://ieeexplore.ieee.org/document/7780634/references#references, Jan. 1, 2016 [Feb. 12, 2019]

[6] Towards Data Science. "Style Transfer - Styling Images with Convolutional Neural Networks." Internet: https://towardsdatascience.com/style-transfer-styling-images-with-convolutional-neural-networks-7d215b58f461 , Oct. 8, 2018 [Feb. 13, 2019]

[7] Towards Data Science. "A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way." Internet: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, Dec. 15, 2018 [Feb. 12, 2019]

[8] Google. "Convolutional Neural Networks Architecture." Internet: https://www.google.com/search?q=cnn+diagram&rlz=1C1JZAP_enIN705IN705&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjsjqb7iqjiAhXHEHIKHVIQBZgQ_AUIDigB&biw=1364&bih=665#imgrc=TbZdxfRSGhD_MM: