Universal Style Transfer via Feature Transforms

Team : **Mandelbrot**

Team Members : Krishna Mahesh Teja
Sai Krishna Charan Dara (Team Leader)
Sai Deva Harsha Annam
Venkata Susheel Voora

Introduction

- Universal style transfer aims to transfer arbitrary visual styles to content images.
- Given a pair of examples the content and style image, universal style transfer aims to synthesize an image that preserves some notion of the content but carries characteristics of the style.
- The key challenge is how to extract effective representations of the style and then match it in the content image

Introduction Contd..

- Two different kind of approaches exist for this problem in the literature
  - Optimization based methods that use covariance matrix to formulate the correlation between features. They can handle arbitrary values with a good visual quality but have high computational costs.
  - Feed forward based methods that work for a fixed number of styles with a compromised visual quality but with a good computational efficiency

- Objective is to develop a universal style transfer approach with a decent visual quality and efficiency.

Approach

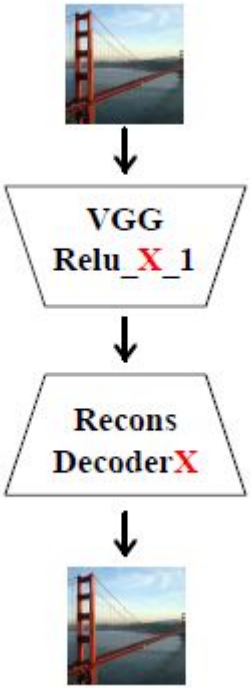
- Aims to synthesize an image that preserves some notion of the content image but carries characteristics of the style image.

We design a universal style transfer pipeline in 3 steps -

- Reconstruction
  - Encoder and Decoder
- Single level Stylization
  - Whitening and Coloring Transform
- Multi level Stylization

In each intermediate layer, our objective is to transform the extracted content features such that they exhibit the same statistical characteristics as the style features of the same layer.

Reconstruction



- We use the VGG-19 network as the feature extractor (encoder), and train a symmetric decoder to invert the VGG-19 features to the original image.
- We trained all the 5 layers using Microsoft COCO dataset.
- To evaluate with features extracted at different layers, we select feature maps at five layers of the VGG-19, i.e., Relu_X_1 (X=1,2,3,4,5), and train five decoders accordingly.
- Loss function used is

$$L = \|I_o - I_i\|_2^2 + \lambda \|\Phi(I_o) - \Phi(I_i)\|_2^2$$

  where ϕ is the VGG encoder that extracts the Relu_X_1 features.
- Once trained, encoder and decoder are fixed through all the experiments.

Structure of Encoders and Decoders of VGG (1 - 5)

VGG-1

```
from torchvision import models

vgg19 =
models.vgg19(pretrained=pretrained)
     features = list(vgg19.features)

self.encoder =
nn.Sequential(*features[:4])
```

VGG-19 Network as the feature extractor (Pretrained Encoder)

Trained symmetric decoder to invert VGG-19 features

```
self.decoder = nn.Sequential( # Sequential,
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,3,(3, 3)),
                              )
```

VGG-2

```
self.encoder =
nn.Sequential(*features[:9])
```

VGG-19 Network as the feature extractor (Pretrained Encoder)

```
self.decoder = nn.Sequential( # Sequential,
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(128,64,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,64,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,3,(3, 3)),
                              )
```

Trained symmetric decoder to invert VGG-19 features

VGG-3

```
self.encoder = nn.Sequential(*features[:18])
```

VGG-19 Network as the feature extractor (Pretrained Encoder)

```
self.decoder = nn.Sequential( # Sequential,
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(256,128,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(128,128,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(128,64,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,64,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,3,(3, 3)),
                              )
```

Trained symmetric decoder to invert VGG-19 features

VGG-4

```
self.encoder = nn.Sequential(*features[:27])
```

VGG-19 Network as the feature extractor (Pretrained Encoder)

Trained symmetric decoder to invert VGG-19 features

```
self.decoder = nn.Sequential( # Sequential,
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(512,256,(3, 3)),
                                    nn.ReLU(),
                                    nn.UpsamplingNearest2d(scale_factor=2),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(256,256,(3, 3)),
                                    nn.ReLU(),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(256,256,(3, 3)),
                                    nn.ReLU(),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(256,256,(3, 3)),
                                    nn.ReLU(),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(256,128,(3, 3)),
                                    nn.ReLU(),
                                    nn.UpsamplingNearest2d(scale_factor=2),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(128,128,(3, 3)),
                                    nn.ReLU(),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(128,64,(3, 3)),
                                    nn.ReLU(),
                                    nn.UpsamplingNearest2d(scale_factor=2),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(64,64,(3, 3)),
                                    nn.ReLU(),
                                    nn.ReflectionPad2d((1, 1, 1, 1)),
                                    nn.Conv2d(64,3,(3, 3)),)
```

VGG-5

```
self.encoder = nn.Sequential(*features[:36])
```

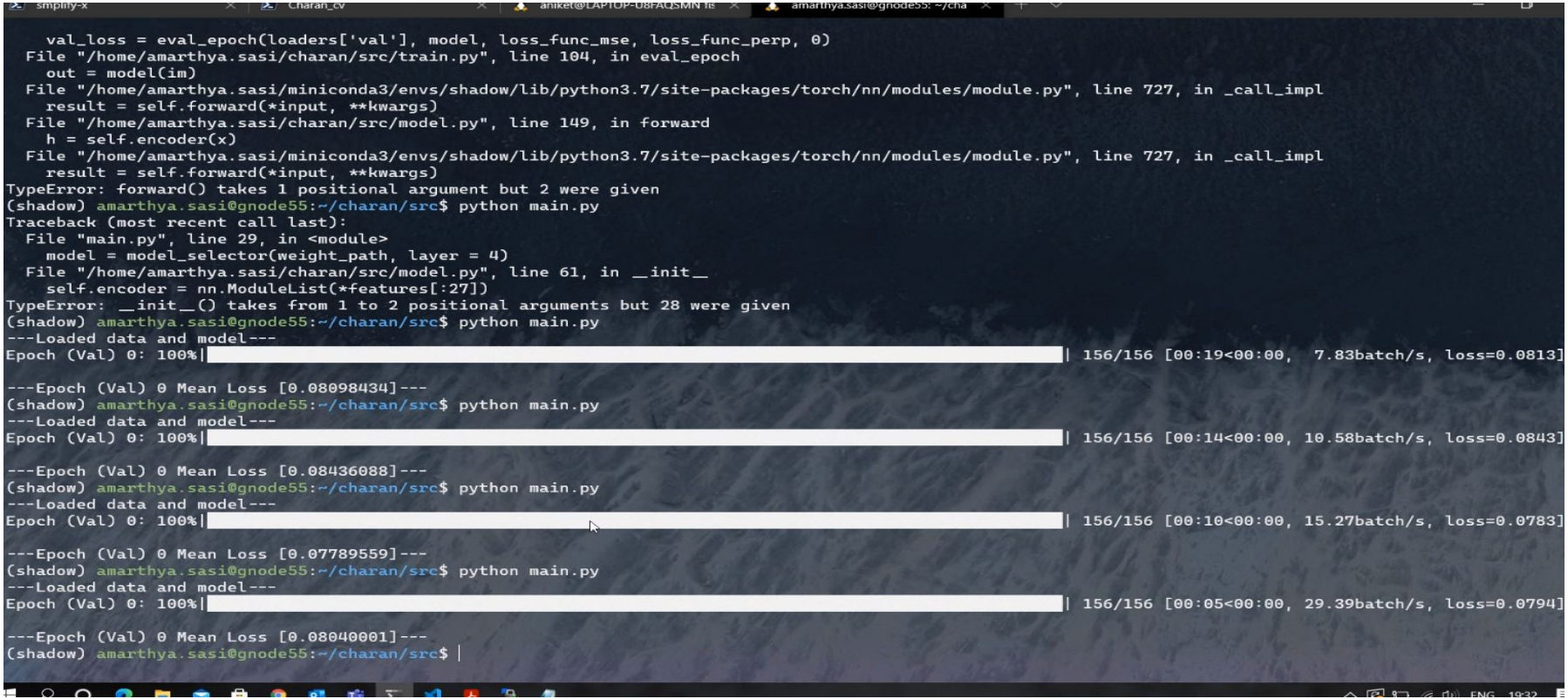
VGG-19 Network as the feature extractor (Pretrained Encoder)

Trained symmetric decoder to invert VGG-19 features

```
self.decoder = nn.Sequential( # Sequential,
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(512,512,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(512,512,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(512,512,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(512,512,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(512,256,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(256,256,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(256,256,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(256,256,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(256,128,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(128,128,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(128,64,(3, 3)),
                              nn.ReLU(),
                              nn.UpsamplingNearest2d(scale_factor=2),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,64,(3, 3)),
                              nn.ReLU(),
                              nn.ReflectionPad2d((1, 1, 1, 1)),
                              nn.Conv2d(64,3,(3, 3)),
                              )
```

Trained Decoder Results

VGG (1-4) Mean Loss over Test Data Set



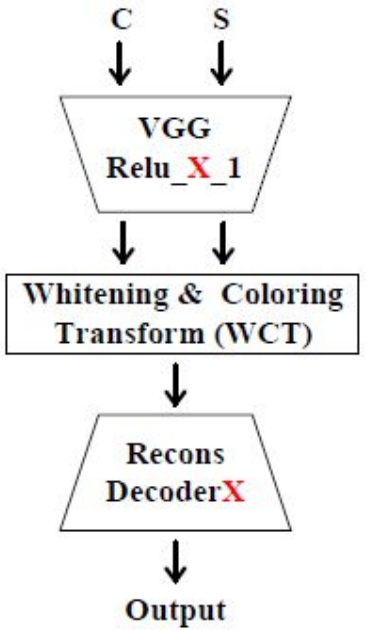
VGG-5 Mean Loss over Test Data Set

```
Traceback (most recent call last):
  File "main.py", line 36, in <module>
    train(loaders, model)
  File "/home/amarthya.sasi/charan/src/train.py", line 138, in train
    val_loss = eval_epoch(loaders['val'], model, loss_func_mse, loss_func_perp, 0)
  File "/home/amarthya.sasi/charan/src/train.py", line 105, in eval_epoch
    loss = loss_func_mse(out,im) + loss_func_perp(out,im)
  File "/home/amarthya.sasi/miniconda3/envs/shadow/lib/python3.7/site-packages/torch/nn/modules/module.py", line 727, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/home/amarthya.sasi/charan/src/train.py", line 65, in forward
    x = block(x)
  File "/home/amarthya.sasi/miniconda3/envs/shadow/lib/python3.7/site-packages/torch/nn/modules/module.py", line 727, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/home/amarthya.sasi/miniconda3/envs/shadow/lib/python3.7/site-packages/torch/nn/modules/container.py", line 117, in forward
    input = module(input)
  File "/home/amarthya.sasi/miniconda3/envs/shadow/lib/python3.7/site-packages/torch/nn/modules/module.py", line 727, in _call_impl
    result = self.forward(*input, **kwargs)
  File "/home/amarthya.sasi/miniconda3/envs/shadow/lib/python3.7/site-packages/torch/nn/modules/conv.py", line 423, in forward
    return self._conv_forward(input, self.weight)
  File "/home/amarthya.sasi/miniconda3/envs/shadow/lib/python3.7/site-packages/torch/nn/modules/conv.py", line 420, in _conv_forward
    self.padding, self.dilation, self.groups)
RuntimeError: Input type (torch.cuda.FloatTensor) and weight type (torch.FloatTensor) should be the same
(shadow) amarthya.sasi@gnode50:~/charan/src$ python main.py
---Loaded data and model---
Epoch (Val) 0:   6%|▌         | 10/156 [00:02<00:43,  3.36batch/s, loss=0.0742]

---Epoch (Val) 0 Mean Loss [0.07544522]---
(shadow) amarthya.sasi@gnode50:~/charan/src$ python main.py
---Loaded data and model---
Epoch (Val) 0: 100%|██████████| 156/156 [00:22<00:00,  6.85batch/s, loss=0.078]

---Epoch (Val) 0 Mean Loss [0.07693266]---
(shadow) amarthya.sasi@gnode50:~/charan/src$ python main.py
---Loaded data and model---
Epoch (Val) 0: 100%|██████████| 156/156 [00:22<00:00,  6.82batch/s, loss=0.078]

---Epoch (Val) 0 Mean Loss [0.07693266]---
(shadow) amarthya.sasi@gnode50:~/charan/src$
```

Reconstruction Losses over Test Data Set (CoCo)

VGG Type	Mean Loss
VGG-1	0.0804
VGG-2	0.0778
VGG-3	0.0843
VGG-4	0.0809
VGG-5	0.0769

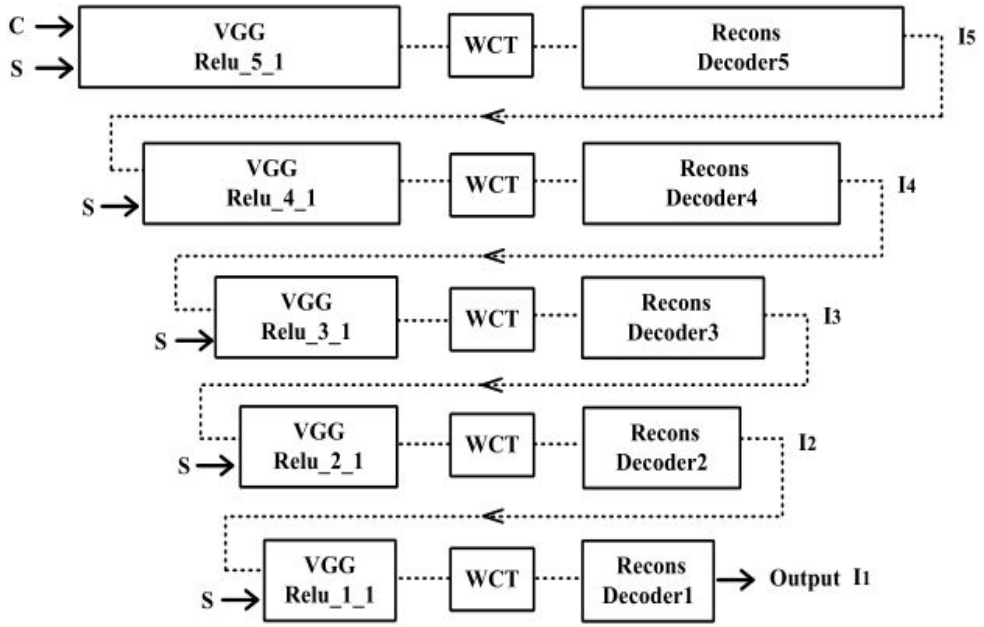
Single level stylization



- Given a pair of content image I_c and style image I_s, we first extract their vectorized VGG feature maps f_c and f_s at a certain layer (e.g., Relu_4_1), where H_c, W_c (H_s, W_s) are the height and width of the content (style) feature, and C is the number of channels.
- The decoder will reconstruct the original image I_c if f_c is directly fed into it.
- Now we use WCT to directly transform the f_c to match the covariance matrix of f_s.
- First we apply the whitening transform and make the covariance matrix to Identity..
- Now we apply the Coloring transform to match the covariance matrix to that of the style image.

Multi Level Stylization

- We extend single-level to multi-level stylization in order to match the statistics of the style at all levels. The result obtained by matching higher level statistics of the style is treated as the new content to continue to match lower-level information of the style.



(c) Multi-level stylization

Advantages of MultiLevel Algorithm

- The multi-level algorithm generates stylized images with greater visual quality, which are comparable or even better with much less computational costs
- It clearly shows that the higher layer features capture more complicated local structures, while lower layer features carry more low-level information (e.g., colors)
- Therefore, it is advantageous to use features at all five layers to fully capture the characteristics of a style from low to high levels.
- Higher layer features first capture salient patterns of the style and lower layer features further improve details!

WCT

- Given a pair of content image I_c and style image I_s, we first extract their vectorized VGG feature f_c and f_s .
- We want to adjust f_c with respect to the statistics of f_s to transfer the style.
- The goal of WCT is to directly transform the f_c to match the covariance matrix of f_s It consists of two steps, i.e., whitening and coloring transform.
- This WCT is applied in the pipeline before passing to the decoder

Whitening Transform

- We make the feature map uncorrelated i.e we want the covariance matrix to be diagonal
- In Whitening Transform we also want all the eigenvalues of the resultant feature map to be equal i.e we want the covariance matrix to be Identity.
- This can be achieved in many ways. The transform used in this paper is

$$\hat{f}_c = E_c \, D_c^{-\frac{1}{2}} \, E_c^\top \, f_c$$

where D_c is a diagonal matrix with the eigenvalues of the covariance matrix $f_c \, f_c^\top \in \Re^{C \times C}$, and E_c is the corresponding orthogonal matrix of eigenvectors, satisfying $f_c \, f_c^\top = E_c D_c E_c^\top$.

The resultant feature maps are uncorrelated $(\hat{f}_c \hat{f}_c^{\,\top} = I)$

Whitening Transform

- Whitened features still maintain global structures of the image contents, but greatly help remove other information related to styles.
- The whitening step helps peel off the style from an input image while preserving the global content structure. The outcome of this operation is ready to be transformed with the target style.

Coloring Transform

- Now we transform our output of whitening transform to a feature map such that the covariance is same as that of the style.
- This is reverse of whitening transform.

$$\hat{f_{cs}} = E_s \, D_s^{\frac{1}{2}} \, E_s^{\top} \, \hat{f_c}$$

where D_s is a diagonal matrix with the eigenvalues of the covariance matrix $f_s \, f_s^{\top} \in \Re^{C \times C}$

and E_s is the corresponding orthogonal matrix of eigenvectors

The final covariance matrix equals that of style $(\hat{f_{cs}} \, \hat{f_{cs}}^{\top} = f_s \, f_s^{\top})$.

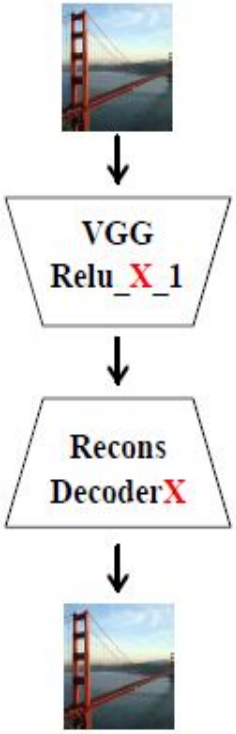
User control on the strength of stylization effects

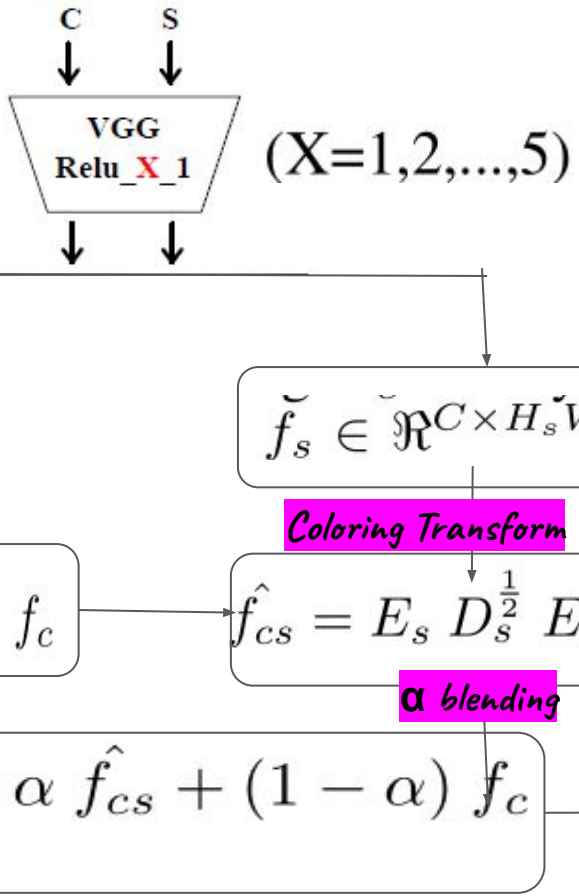
- After the WCT, we may blend the output feature map $\hat{f}_{cs}$ with the content feature fc before feeding it to the decoder in order to provide user controls on the strength of stylization effects:

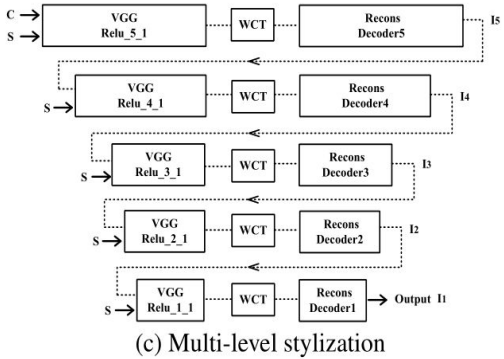
$$\hat{f}_{cs} = \alpha \, \hat{f}_{cs} + (1 - \alpha) \, f_c \, ,$$

  where α serves as the style weight for users to control the transfer effect.

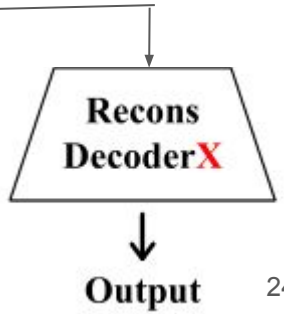
Overall Flow Diagram







(c) Multi-level stylization



RESULTS

Single Stage Stylization outputs Eg 1

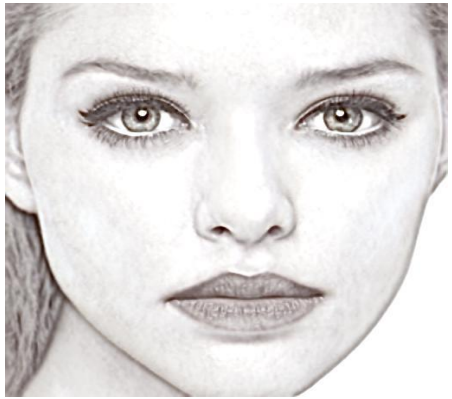
Content

Style

Single Stage Stylization outputs Eg 2

Content

Style



Conclusions

- ➔ It clearly shows that the higher layer features capture more complicated local structures, while lower layer features carry more low-level information (e.g., colors).
- ➔ Lower level nicely captures the edges and higher level captures the global structure.
- ➔ Hence we go to multi-level to include both higher and lower level information.

Output => Alpha = 0.6 | Multi Level Eg:1

Content

Style

Output

Intermediate Levels output - Level 5 to Level 1 Eg:1

I_5

I_4

I_1

I_3

I_2

Output => Alpha = 0.6 | Multi Level Eg:2

Content

Style

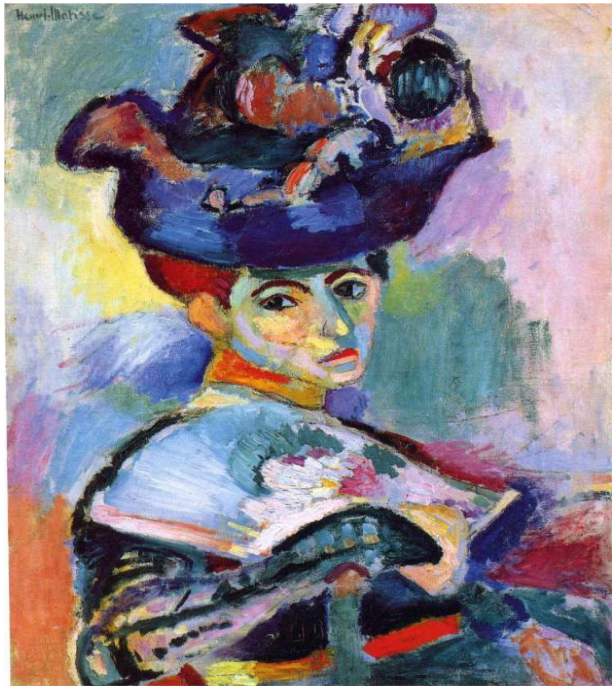
Intermediate Levels output - Level 5 to Level 1 Eg: 2



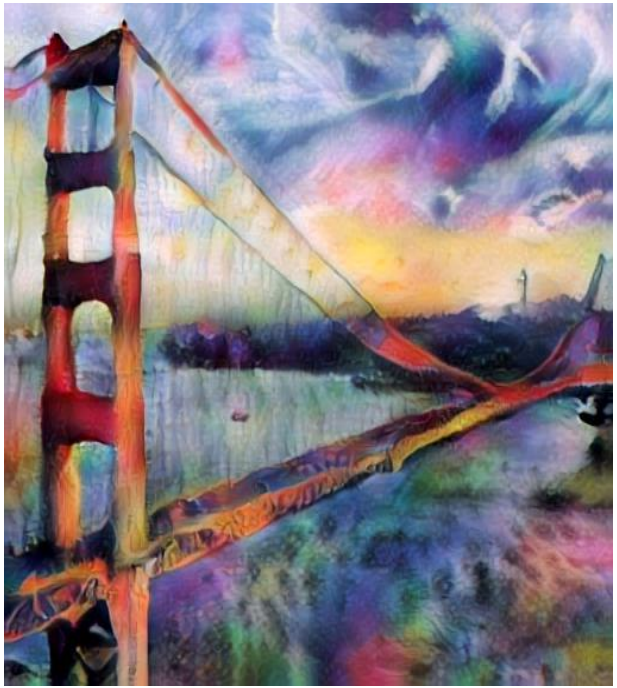


Output => Alpha = 0.6 | Multi Level Eg:3

Content



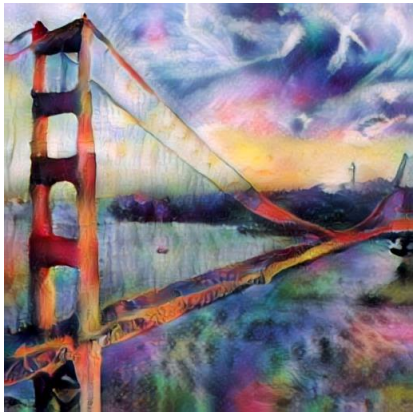
Style



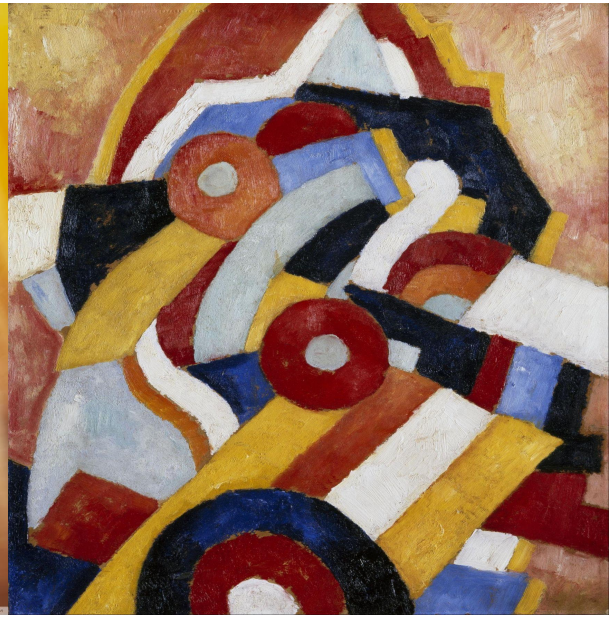
Intermediate Levels output - Level 5 to Level 1 Eg:3







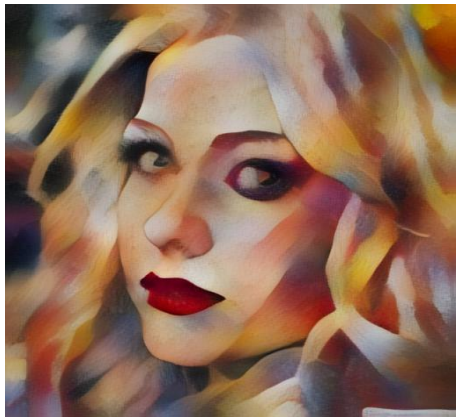
Output => Alpha = 0.6 | Multi Level Eg:4



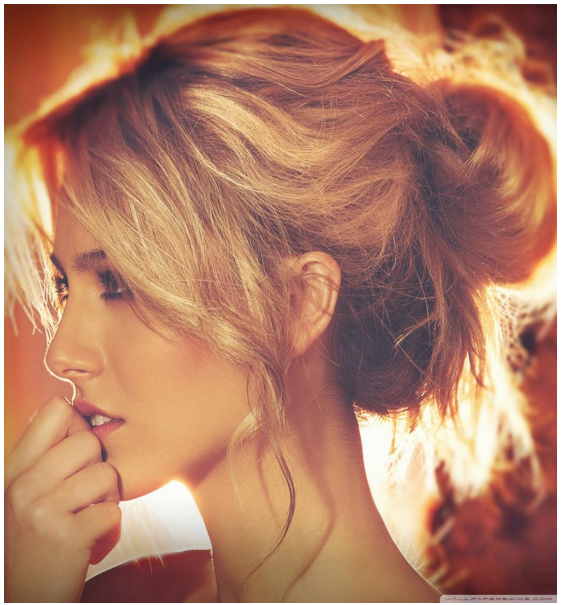
Content

Style

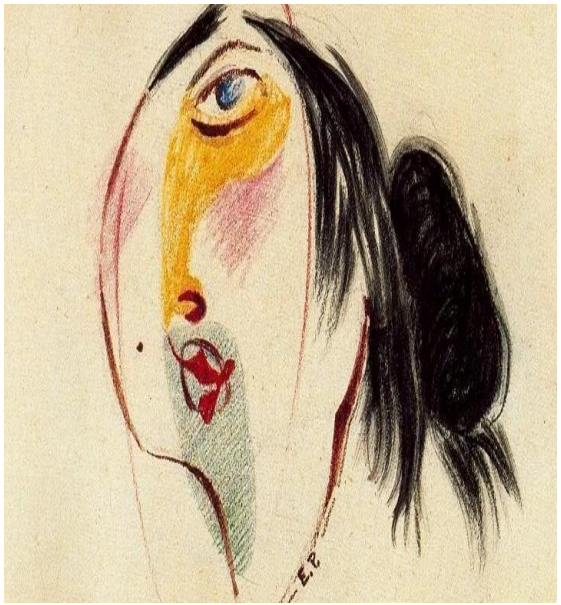
Intermediate Levels output - Level 5 to Level 1 Eg:4


Output => Alpha = 0.6 | Multi Level Eg:5

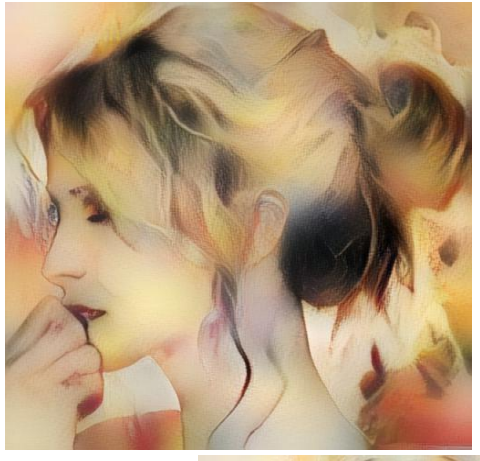


Content



Style

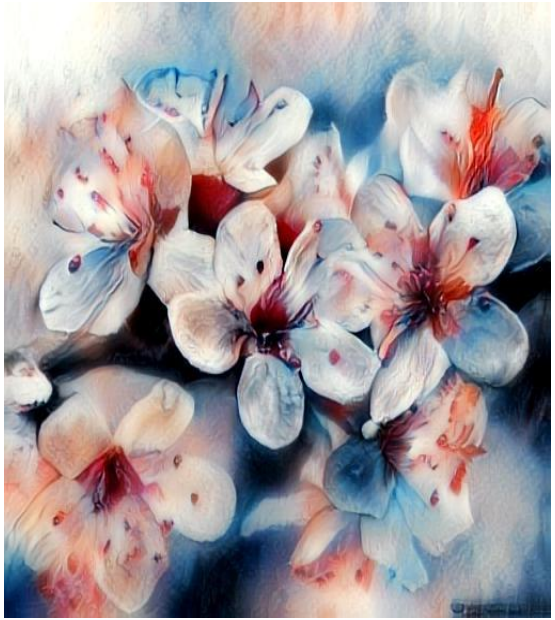
Intermediate Levels output - Level 5 to Level 1 Eg:5


Output => Alpha = 0.6 | Multi Level Eg:6

Content

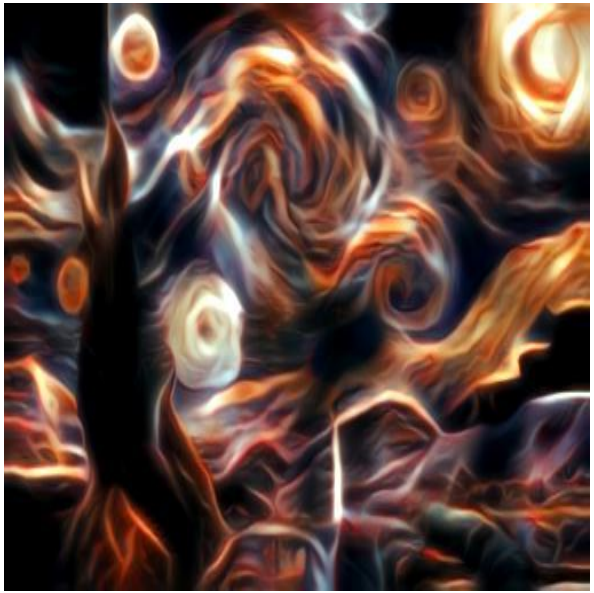
Style



Output => Alpha = 0.6 | Multi Level Eg:7

Content

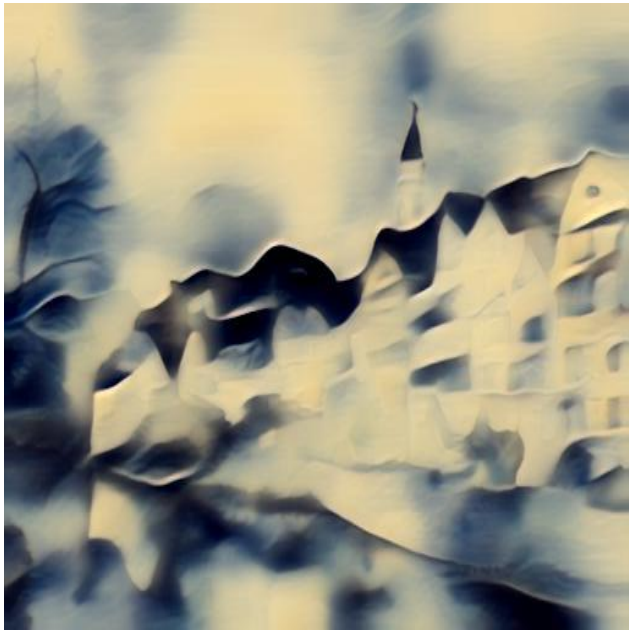
Style



Output => Alpha = 0.6 | Multi Level Eg:8

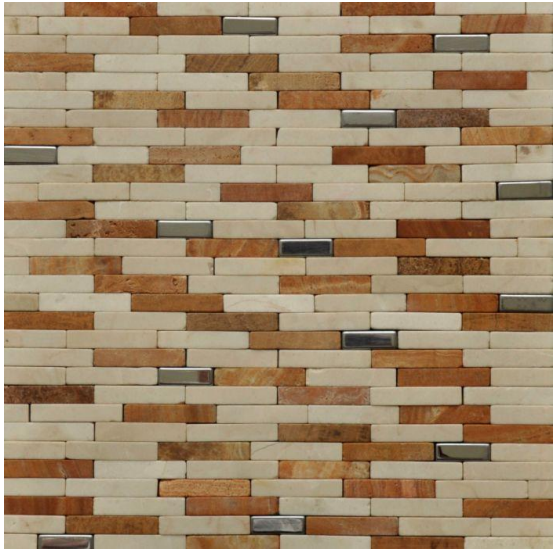
Content

Style



Output => Alpha = 0.6 | Multi Level Eg:9

Content



Style

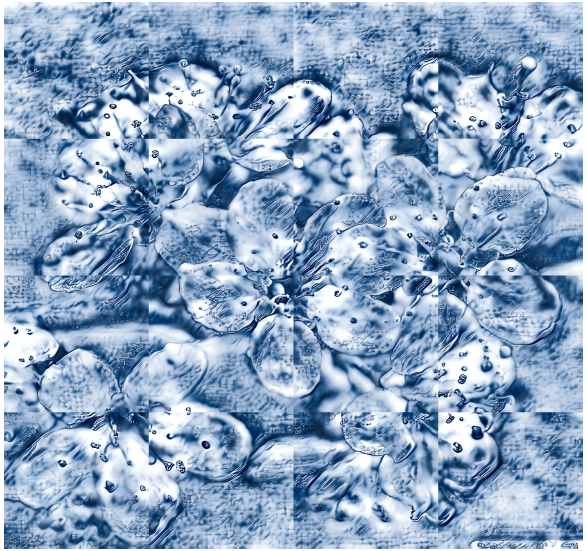
Analysis Studies

Study - 1
Patchwise Style Transform

Universal Style Transfer on Patched Content - 1

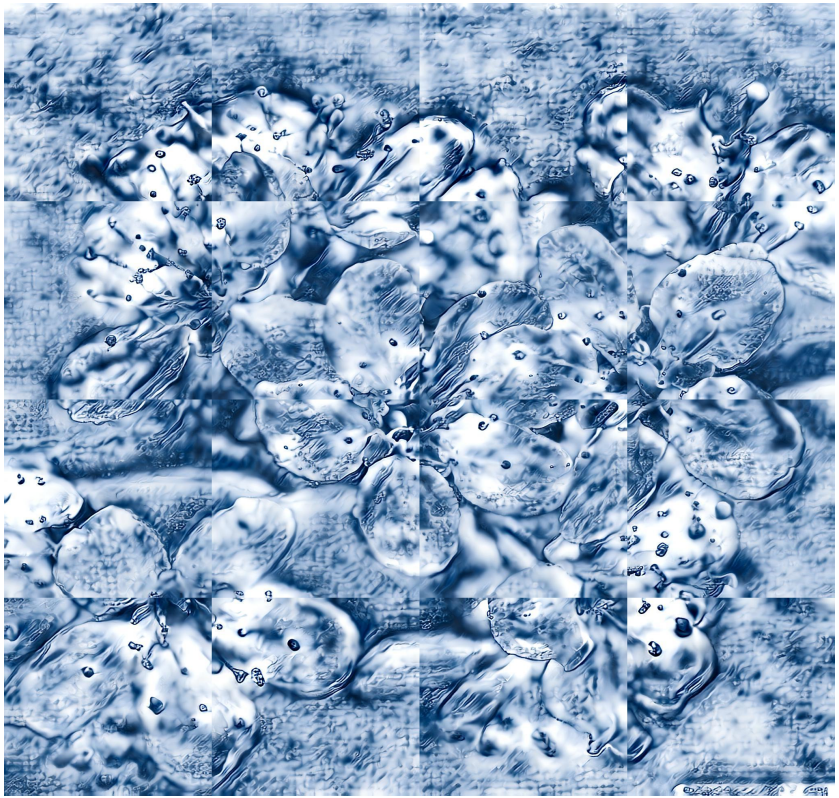
Content

Style



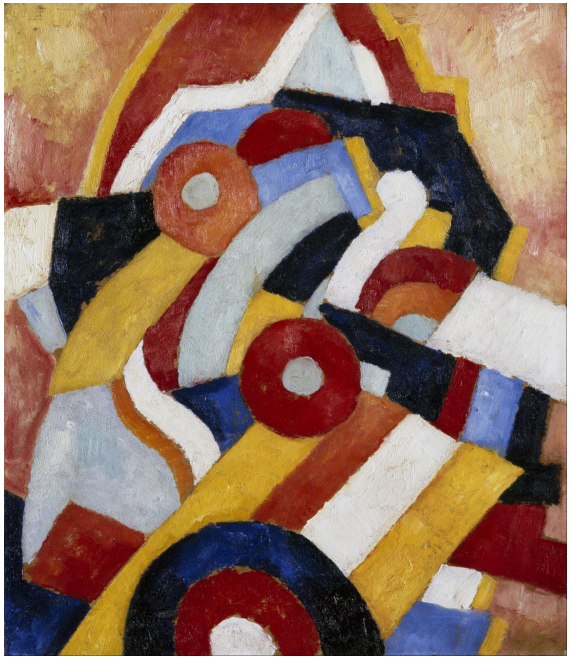
Output
Alpha =1
MultiLevel

Comparison without and with Patch - 1



Universal Style Transfer on Patched Content - 2

Content



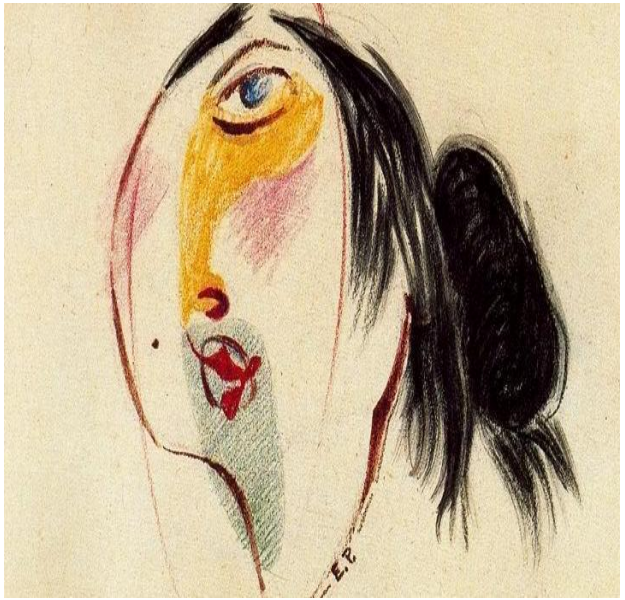
Style

Output
Alpha =1
MultiLevel

Comparison without and with Patch - 2

Universal Style Transfer on Patched Content - 3

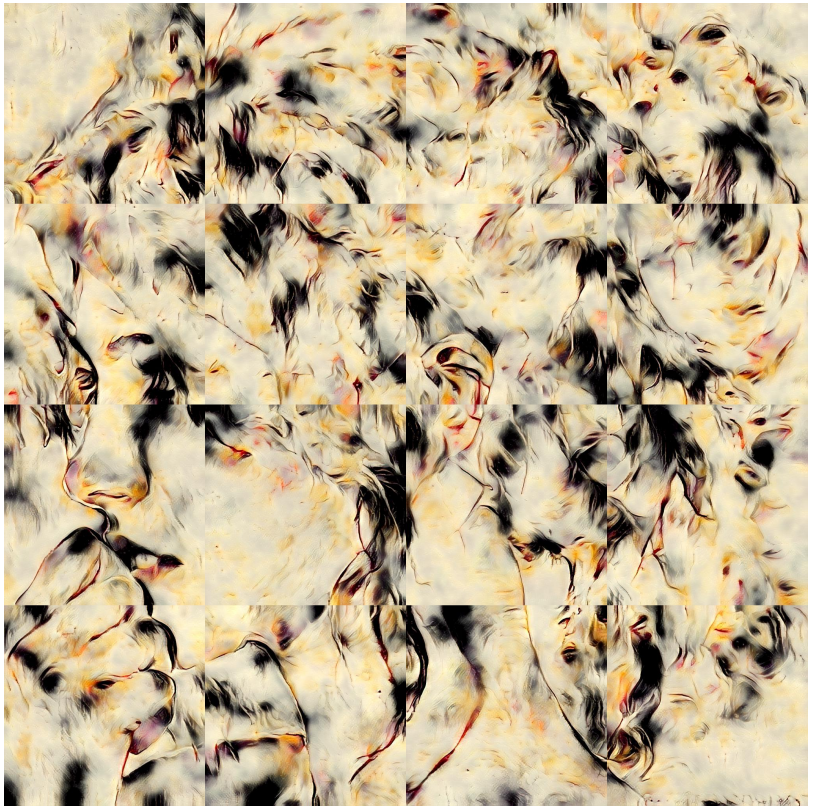
Content



Style

Output
Alpha =1
MultiLevel

Comparison without and with Patch - 3



Conclusions

- We can observe that there is lack of continuity between Patches.
- A little intensity variations are present between the Patches.
- We can see the Content Image is preserved with Style features.
- It is different from normal WCT because, WCT takes global features into account, whereas with patch it is local features (within the patch).

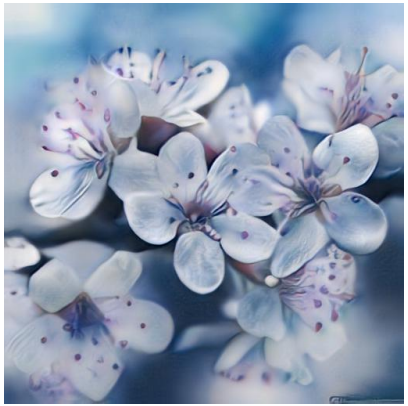
Advantage with this output

➔ If we post process this image to remain continuity between patches, we can see that output will resemble the artistic effect.

➔ If we increase number of patches, the output will be like ***SuperPixels Image.***

Study - 2
Varying Alpha

Varying Alpha Eg 1



Alpha = 0.2

Alpha = 0.4

Alpha = 0.6

Alpha = 1

Varying Alpha Eg 2

Alpha = 0.2

Alpha = 0.4

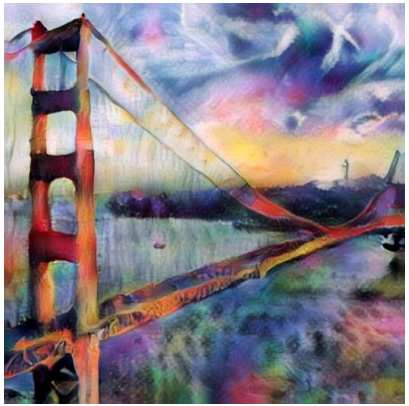
Alpha = 0.6

Alpha = 1

Varying Alpha Eg 3

Alpha = 0.2

Alpha = 0.4



Alpha = 0.6

Alpha = 1

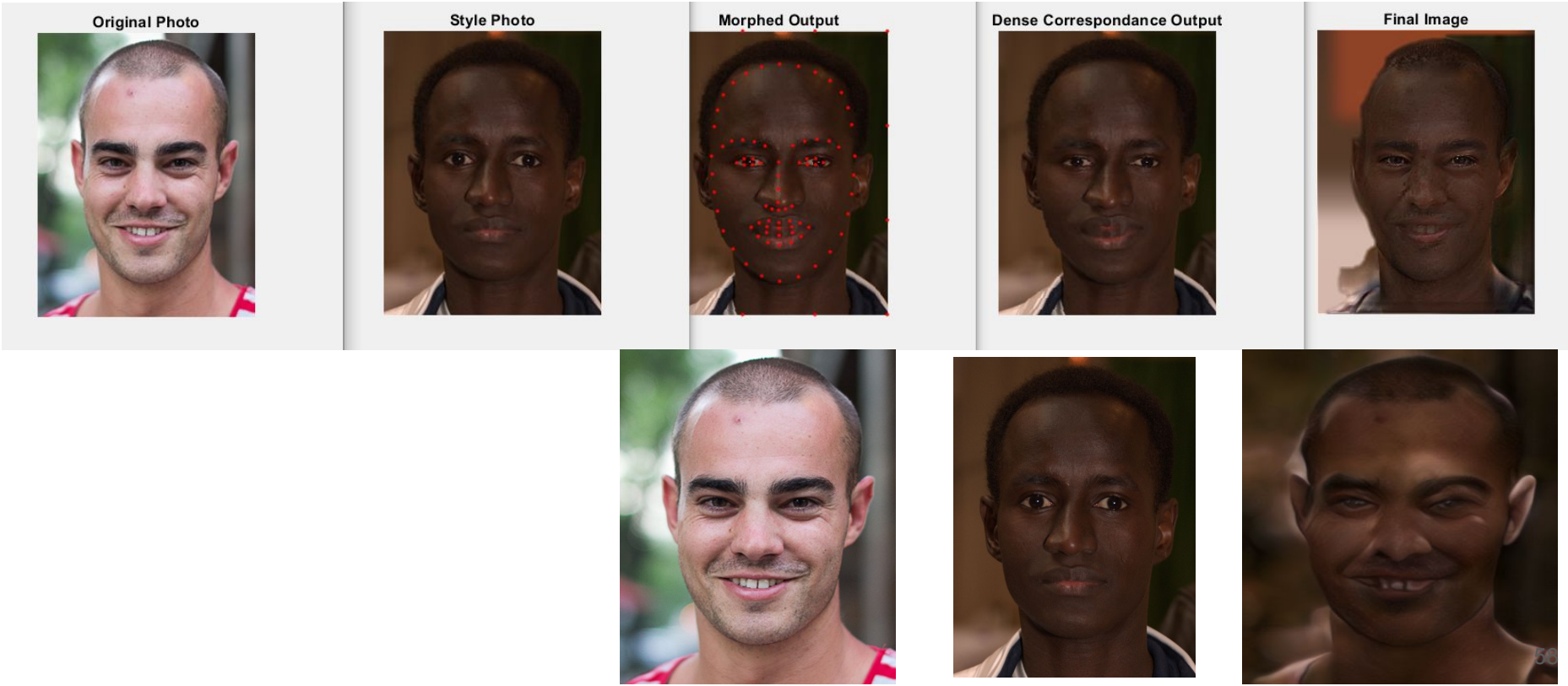
Conclusion

➔ Decreasing Alpha means adding more content features which can be observed in the above results.

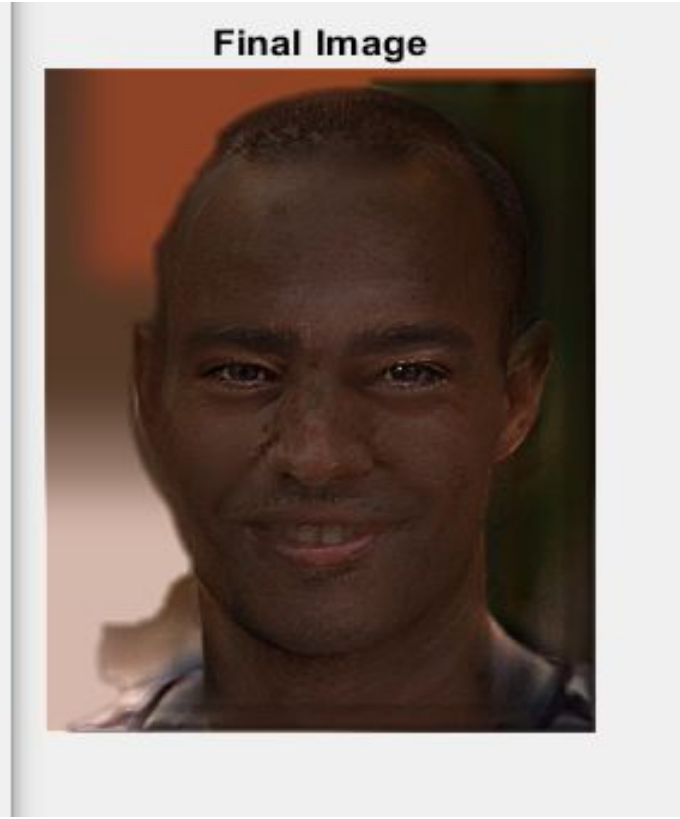
$$\hat{f_{cs}} = \alpha \, \hat{f_{cs}} + (1 - \alpha) \, f_c \, ,$$

Study - 3
Style Transfer on Headshot Portraits

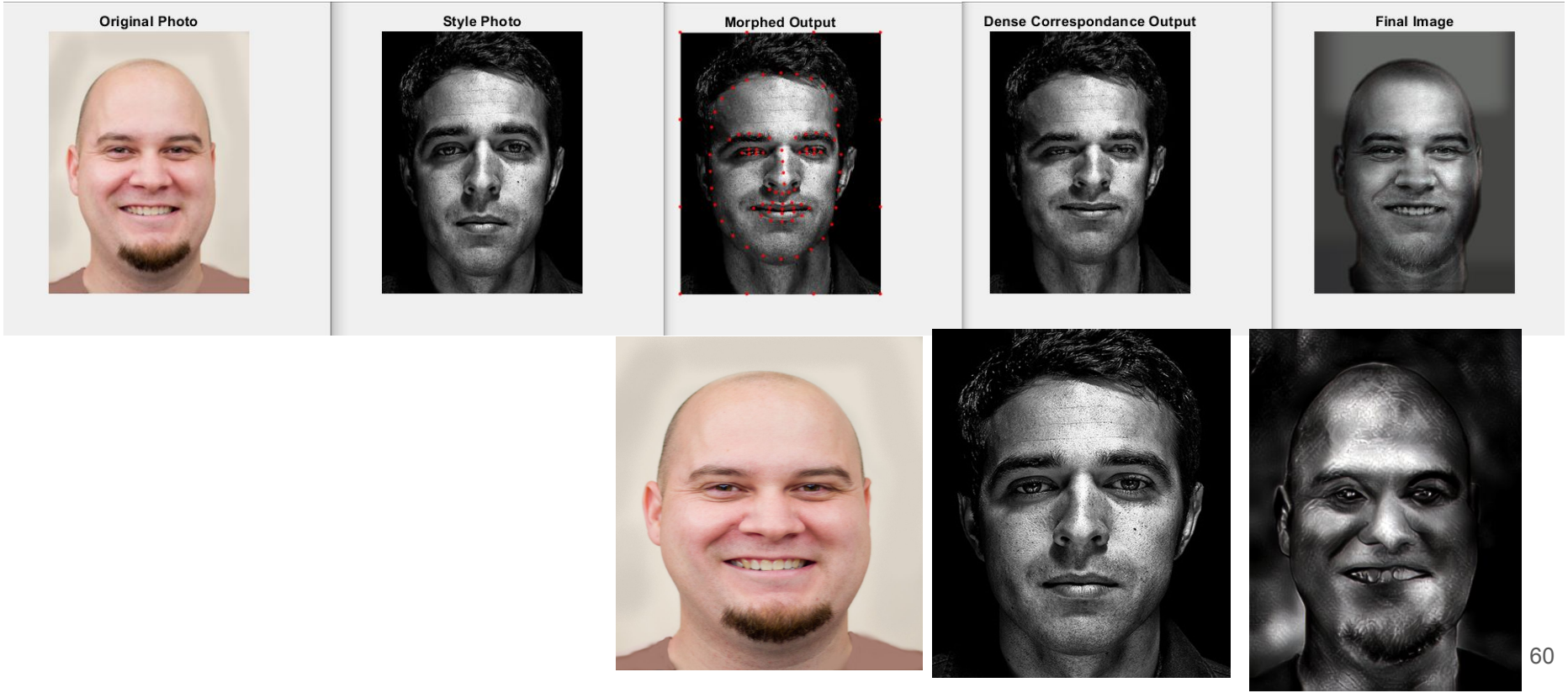
Using Headshot Transfer Vs Using WCT Style Transfer - 1



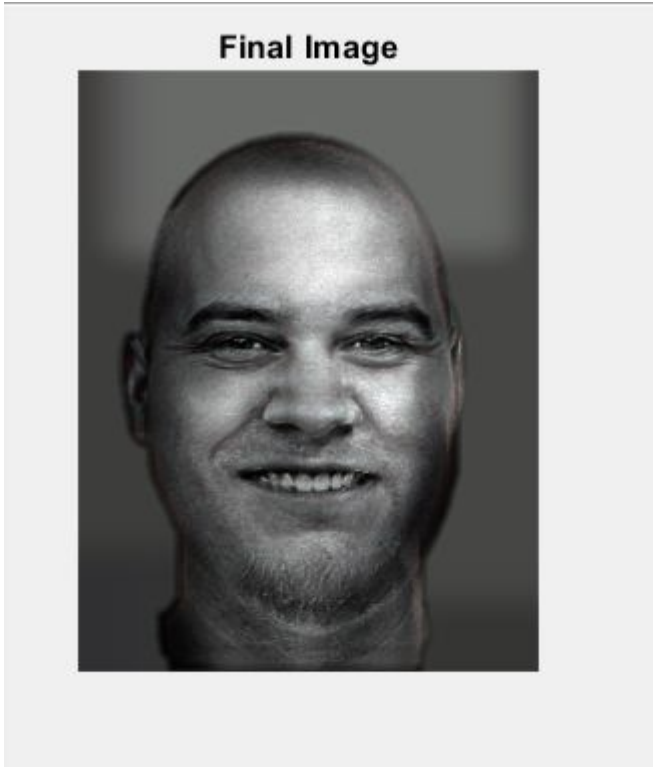
Headshot Style Transfer Vs WCT Style Transfer - 1

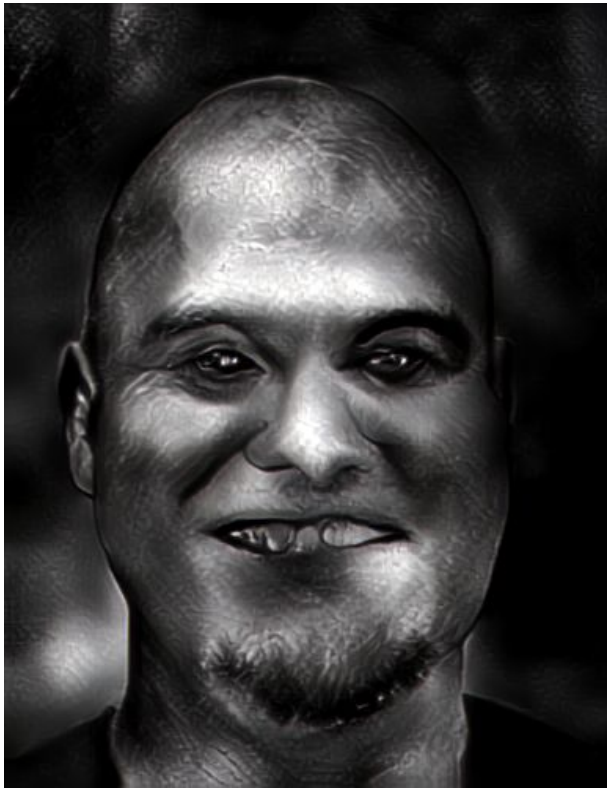


Using Headshot Transfer Vs Using WCT Style Transfer - 2

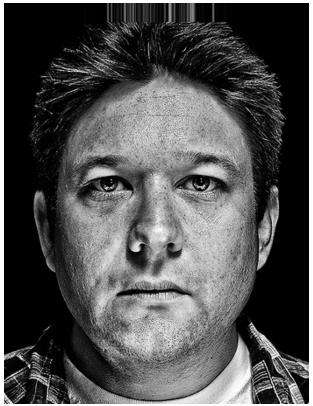


Headshot Style Transfer Vs WCT Style Transfer - 2

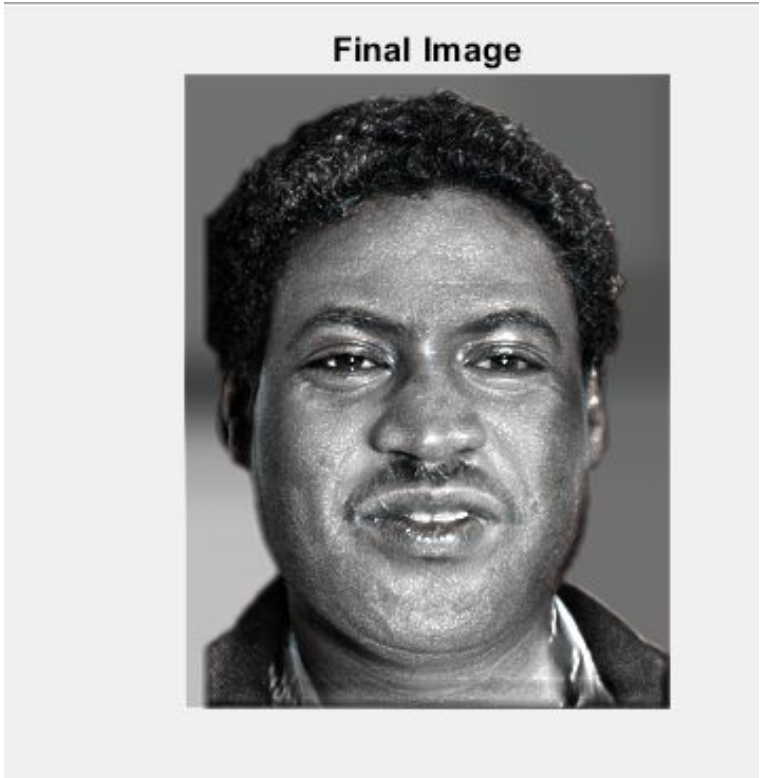




Using Headshot Transfer Vs Using WCT Style Transfer - 3



Headshot Style Transfer Vs WCT Style Transfer - 3



Conclusions

- We can see that the background matching is not happening nicely using WCT compared to Headshot style Transfer.
- A little artefacts are present in the output images.
- Some areas in the output, style transfer didn’t happen as expected.

Thank You