



**Credit Hours System**  
**CMPN446**



**Cairo University**  
**Faculty of Engineering**

# **“Image Style Transfer using Convolutional Neural Networks”**

## **Team members:**

- 1- Ahmed Hamada ElHinidy  
“ahmed.el-hinidy.2014@ieee.org - 01227417171”
- 2- Assem Amr Mohamed  
“asemsadek@hotmail.com - 01021533253”
- 3- Lana Tarek Shafik  
“lana.chafik@gmail.com - 01277784025”
- 4- Mohamed Mohsen ElSamanoudy  
“mohamed.m.elsamanoudy@gmail.com - 01009414441”

## **Introduction:**

Neural networks are the state of the art technique used for AI oriented applications nowadays. Convolutional Neural Networks are a special type of neural networks that is used mainly for Computer Vision applications as it respects the layered structure of images and builds upon both the basics of filtering in traditional Image Processing techniques, and also the structure of how we as living organisms perceive and understand images.

## **Project Idea:**

We choose to do a research oriented project where we explore new published papers (2015 and 2016), in order to gain strong understanding of Convolutional Neural Networks, and one of their recent applications which is transferring an artistic style from one image to another, which was traditionally called texture transfer.

We also choose to implement them by hand in order to gain experience with the process of building, debugging, and training Neural Networks, and how to rent online machines to do GPU accelerated training.

In this project we implement and experiment with the following papers:

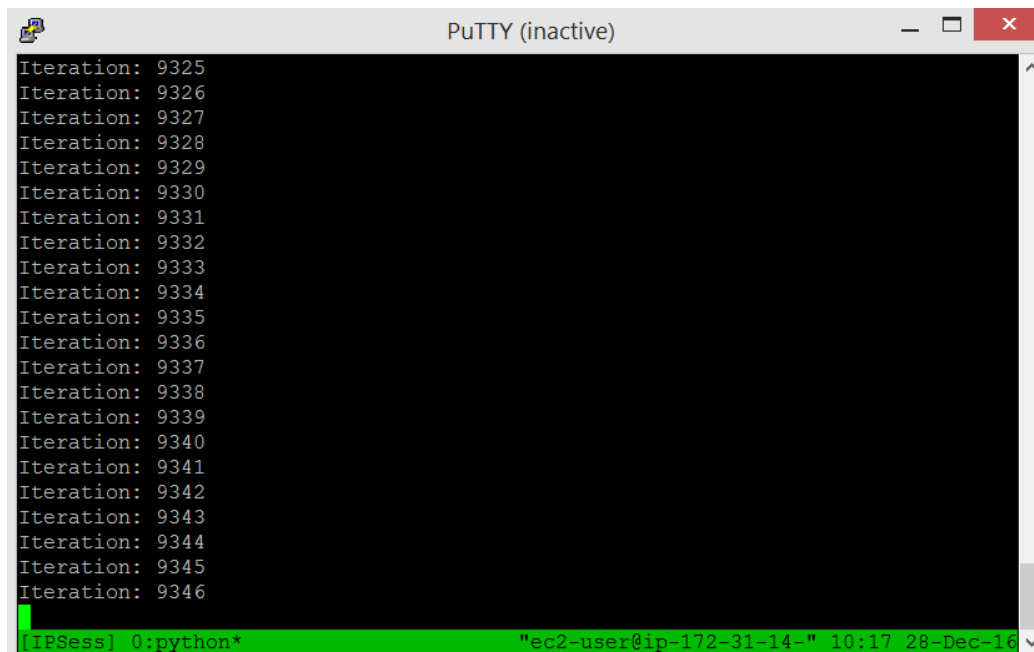
1. Very Deep Convolutional Networks for Large-Scale Image Recognition (2015)
2. A Neural Algorithm of Artistic Style (2015)
3. Perceptual Losses for Real-Time Style Transfer and Super-Resolution (2016)

## **Technologies used:**

There are a variety of available tools for implementing and training neural networks such as Tensorflow (Google), CNTK (Microsoft), Keras, Theano, and Caffe.

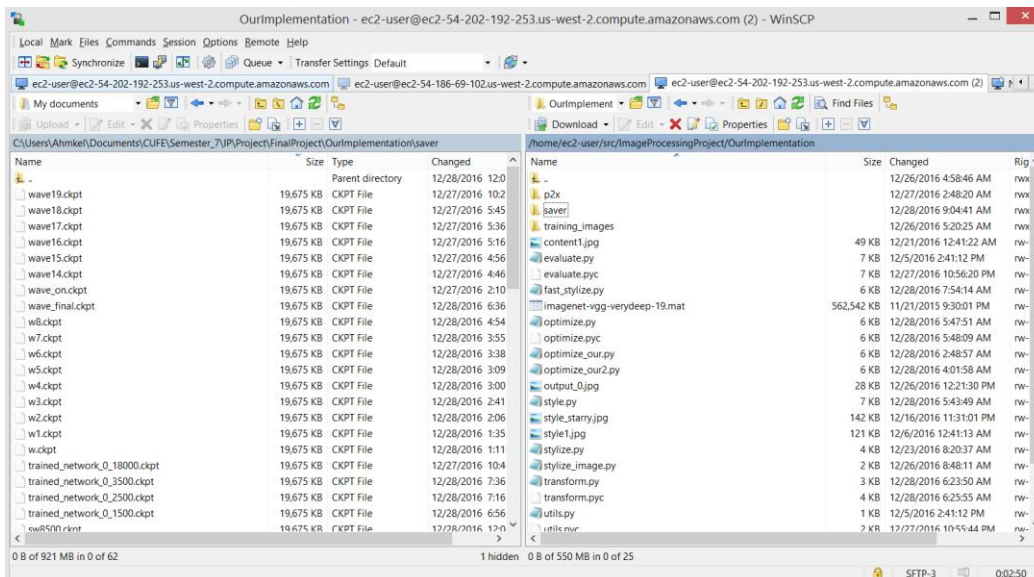
We choose to implement using Tensorflow as it is one of the newest libraries, and it was built to be ready for production level development instead of being suitable only for experiments. The development is done in Python 3.

For the training, Neural Networks training does matrix multiplication intensively, therefore it is much more suitable to utilize GPUs for the training process. In order to be able to train in logical timing, we rented GPU accelerated machines on Amazon Web Services to run the training service on them. The connection was done using SSH and Putty for the console window (terminal) connection, and WinSCP for the file transfer.



```
Iteration: 9325
Iteration: 9326
Iteration: 9327
Iteration: 9328
Iteration: 9329
Iteration: 9330
Iteration: 9331
Iteration: 9332
Iteration: 9333
Iteration: 9334
Iteration: 9335
Iteration: 9336
Iteration: 9337
Iteration: 9338
Iteration: 9339
Iteration: 9340
Iteration: 9341
Iteration: 9342
Iteration: 9343
Iteration: 9344
Iteration: 9345
Iteration: 9346

[IPSSess] 0:python* "ec2-user@ip-172-31-14-" 10:17 28-Dec-16
```



| Name                          | Size      | Type             | Changed         |
|-------------------------------|-----------|------------------|-----------------|
| ..                            |           | Parent directory | 12/28/2016 12:0 |
| wave19.chkpt                  | 19,675 KB | CKPT File        | 12/27/2016 10:2 |
| wave18.chkpt                  | 19,675 KB | CKPT File        | 12/27/2016 5:45 |
| wave17.chkpt                  | 19,675 KB | CKPT File        | 12/27/2016 5:36 |
| wave16.chkpt                  | 19,675 KB | CKPT File        | 12/27/2016 5:16 |
| wave15.chkpt                  | 19,675 KB | CKPT File        | 12/27/2016 4:56 |
| wave14.chkpt                  | 19,675 KB | CKPT File        | 12/27/2016 4:46 |
| wave_on.chkpt                 | 19,675 KB | CKPT File        | 12/27/2016 2:10 |
| wave_final.chkpt              | 19,675 KB | CKPT File        | 12/27/2016 6:36 |
| w8.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 4:54 |
| w7.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 3:55 |
| w6.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 3:38 |
| w5.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 3:09 |
| w4.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 3:00 |
| w3.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 2:41 |
| w2.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 2:06 |
| w1.chkpt                      | 19,675 KB | CKPT File        | 12/28/2016 1:35 |
| w.chkpt                       | 19,675 KB | CKPT File        | 12/28/2016 1:11 |
| trained_network_0_18000.chkpt | 19,675 KB | CKPT File        | 12/27/2016 10:4 |
| trained_network_0_3500.chkpt  | 19,675 KB | CKPT File        | 12/28/2016 7:36 |
| trained_network_0_2500.chkpt  | 19,675 KB | CKPT File        | 12/28/2016 7:16 |
| trained_network_0_1500.chkpt  | 19,675 KB | CKPT File        | 12/28/2016 6:56 |
| new500.chkpt                  | 19,675 KB | CKPT File        | 12/28/2016 1:0  |

| Name                         | Size       | Changed                | Rig       |
|------------------------------|------------|------------------------|-----------|
| ..                           |            | 12/26/2016 4:58:46 AM  | rw-r--r-- |
| p2x                          |            | 12/27/2016 2:48:20 AM  | rw-r--r-- |
| saver                        |            | 12/28/2016 9:04:41 AM  | rw-r--r-- |
| training_images              |            | 12/26/2016 5:20:25 AM  | rw-r--r-- |
| content1.jpg                 | 49 KB      | 12/21/2016 12:41:22 AM | rw-r--r-- |
| evaluate.py                  | 7 KB       | 12/5/2016 2:41:12 PM   | rw-r--r-- |
| evaluate.pyc                 | 7 KB       | 12/27/2016 10:56:20 PM | rw-r--r-- |
| fast_styleize.py             | 6 KB       | 12/28/2016 7:54:14 AM  | rw-r--r-- |
| imagenet-vgg-verydeep-19.mat | 562,542 KB | 11/21/2015 9:30:01 PM  | rw-r--r-- |
| optimize.py                  | 6 KB       | 12/28/2016 5:47:51 AM  | rw-r--r-- |
| optimize.pyc                 | 6 KB       | 12/28/2016 5:48:09 AM  | rw-r--r-- |
| optimize_our.py              | 6 KB       | 12/28/2016 2:48:57 AM  | rw-r--r-- |
| optimize_our2.py             | 6 KB       | 12/28/2016 4:01:58 AM  | rw-r--r-- |
| output_0.jpg                 | 28 KB      | 12/26/2016 12:21:30 PM | rw-r--r-- |
| style.py                     | 7 KB       | 12/26/2016 5:43:49 AM  | rw-r--r-- |
| style_starry.jpg             | 142 KB     | 12/16/2016 11:31:01 PM | rw-r--r-- |
| style1.jpg                   | 121 KB     | 12/6/2016 12:41:13 AM  | rw-r--r-- |
| styleize.py                  | 4 KB       | 12/23/2016 8:20:37 AM  | rw-r--r-- |
| styleize_image.py            | 2 KB       | 12/26/2016 8:48:11 AM  | rw-r--r-- |
| transform.py                 | 3 KB       | 12/28/2016 6:23:50 AM  | rw-r--r-- |
| transform.pyc                | 4 KB       | 12/28/2016 6:23:55 AM  | rw-r--r-- |
| utils.py                     | 1 KB       | 12/5/2016 2:41:12 PM   | rw-r--r-- |
| utils.pyc                    | 3 KB       | 12/27/2016 10:56:44 PM | rw-r--r-- |

# First Paper

In this paper, the content image (which will be stylized) and the style image (from which style is captured) pass through a VGG (architecture of convolution neural network for object recognition developed and trained by Oxford's renowned **Visual Geometry Group** , which achieved very good performance on the ImageNet dataset [1]).

**VGG configurations:** (We used the 19-layers configuration)

| ConvNet Configuration               |                        |                               |  |  |   |
|-------------------------------------|------------------------|-------------------------------|--|--|---|
| A                                   | A-LRN                  | B                             | C  | D  | E   |
| 11 weight layers                    | 11 weight layers       | 13 weight layers              | 16 weight layers                           | 16 weight layers                           | 19 weight layers  |
| input ( $224 \times 224$ RGB image) |                        |                               |  |  |   |
| conv3-64                            | conv3-64<br><b>LRN</b> | conv3-64<br><b>conv3-64</b>   | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                                    |
| maxpool                             |                        |                               |  |  |   |
| conv3-128                           | conv3-128              | conv3-128<br><b>conv3-128</b> | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                                  |
| maxpool                             |                        |                               |  |  |   |
| conv3-256<br>conv3-256              | conv3-256<br>conv3-256 | conv3-256<br>conv3-256        | conv3-256<br>conv3-256<br><b>conv1-256</b> | conv3-256<br>conv3-256<br><b>conv3-256</b> | conv3-256<br>conv3-256<br>conv3-256<br><b>conv3-256</b> |
| maxpool                             |                        |                               |  |  |   |
| conv3-512<br>conv3-512              | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                             |                        |                               |  |  |   |
| conv3-512<br>conv3-512              | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                             |                        |                               |  |  |   |
| FC-4096                             |                        |                               |  |  |   |
| FC-4096                             |                        |                               |  |  |   |
| FC-1000                             |                        |                               |  |  |   |
| soft-max                            |                        |                               |  |  |   |

**There are 3 instances of VGG network:**

- 1- First computes the content features in the feedforward mode.
- 2- Second computes the style features in the feedforward mode.
- 3- Third is trained to minimize the loss between an arbitrary input image and content and style losses.

The weights of VGG network used in the computation of content features and style features in the feedforward mode are also published; they were trained on the ImageNet dataset and gave good results (accuracy > 95%). So, they took advantage of the idea of transfer learning and instead of training the VGG on the style and the content images, they set the weights to the ones trained on the ImageNet dataset.

First, we apply forward propagation on the content image and save the features of 'relu4\_2' ('relu' activation function is performed after each 'conv' layer). Second, we apply forward propagation on the style image and save the features of 'relu1\_1, relu2\_1, relu3\_1, relu4\_1, relu5\_1'. Finally, we train the third instance of the VGG whose input an arbitrary image trying to minimize the loss function which is the summation of content loss and style loss (content loss is the difference between the features extracted from the same layer 'relu4\_2' of the input image and the content features, style loss is the difference between 'gram' [\[2\]](#) matrices of the style image from layers listed above and the corresponding layers in the third instance of VGG).

Total loss =  $\alpha \cdot \text{content loss} + \beta \cdot \text{style loss}$ .

Alpha and beta are parameters which we choose depending on how many of the content image we want in the stylized image.

Minimizing the loss here is done by backpropagating in the arbitrary input image without change the weights.

**Output samples:**

1-

Content Image



Style Image





Stylized Image

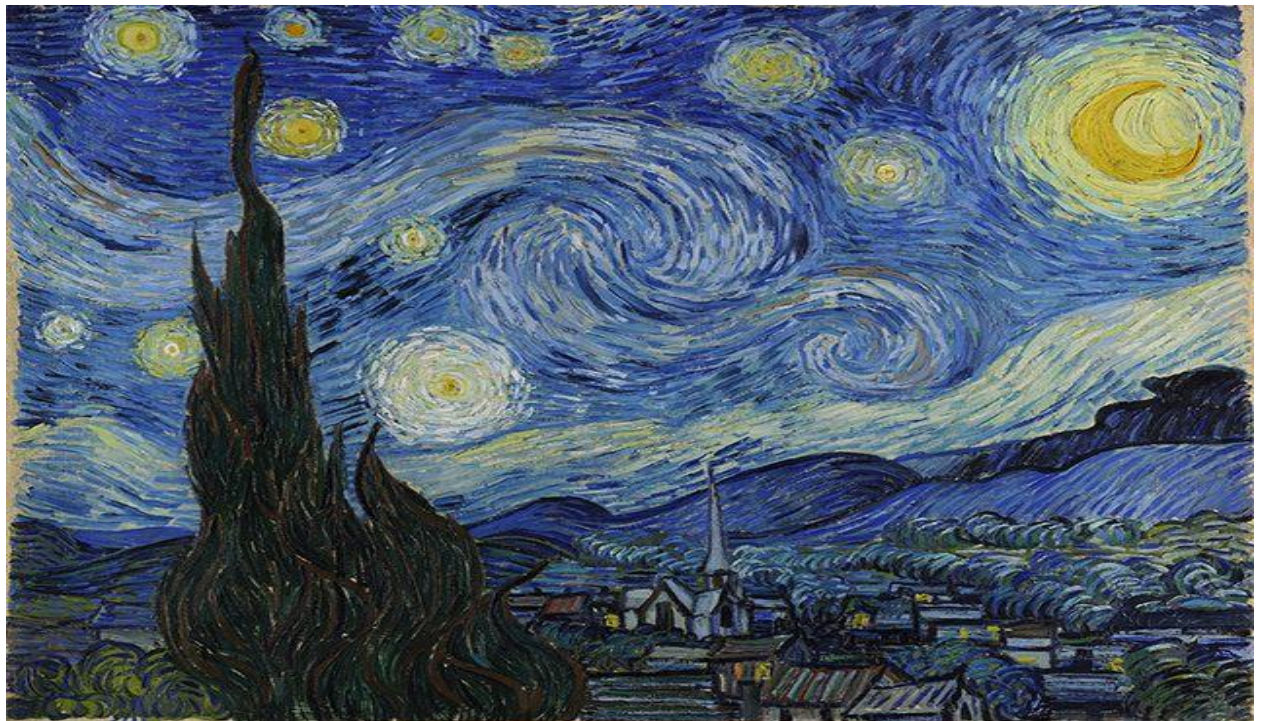


Different Percentages of Style and Content Images



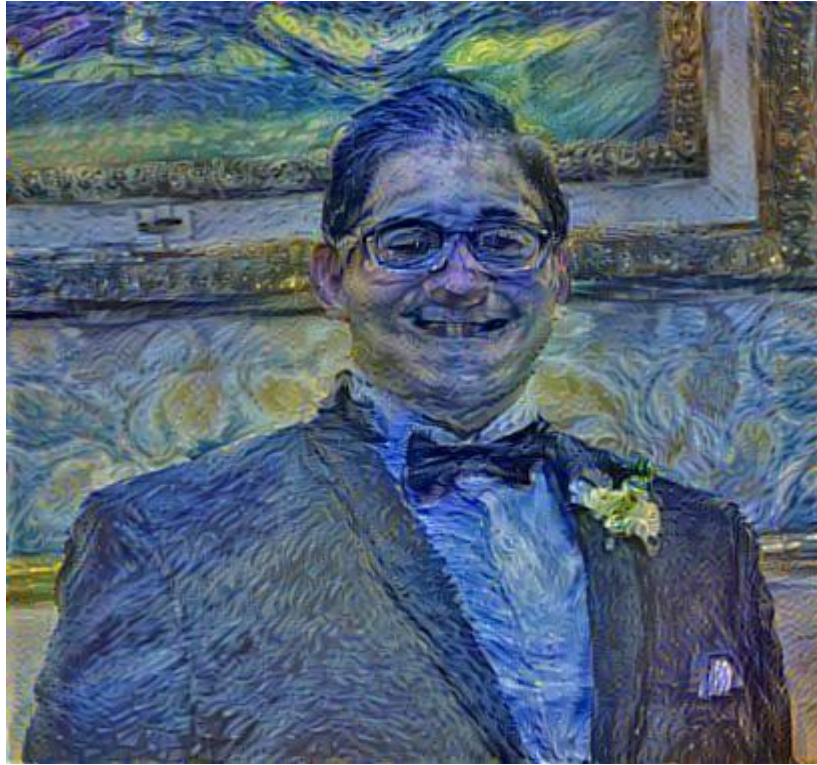
2-

Style Image



Stylized Image





3-

Content Image



Style Image





Stylized Image



## Second Paper

This paper also uses VGG architecture for convolutional neural network but the 16-layers configuration. It also uses another convolutional neural network which consists of 3 layers 'convolution' each followed by 'relu' with downsizing for the output, then, 5 residual layers (residual layer is the summation of current layer and the output of 'convolution' then 'relu' then 'convolution' layers). These 5 residual layers are implement to prevent the gradient during the backpropagation from vanishing. Finally, 3 layers of 'convolution transpose' each followed by 'relu' with upsizing for the output in order to restore original image size for further process (called transform network).

The two instances of VGG network responsible for the forward propagation of the content and the style images are the same as the 1<sup>st</sup> paper. An input image is passed to the transform network and then its output is passed to a VGG network. The final output is used to compute the loss. Also, the loss is computed in the same way as the 1<sup>st</sup> paper.

### **This paper differs only in two things:**

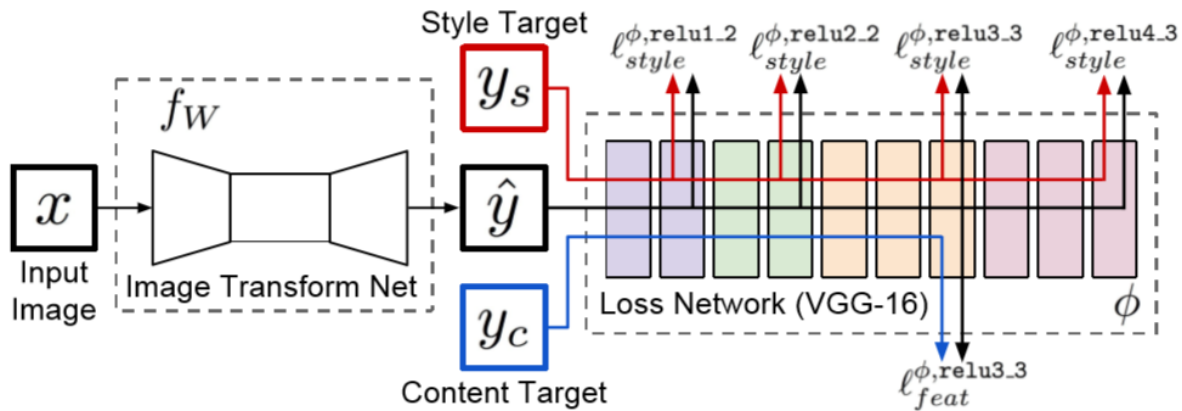
- 1- The input must be the content image not an arbitrary one like the 1<sup>st</sup> paper.
- 2- Minimizing the loss function is done by backpropagating in the weights of the transform network.

### **Transform Network:**

The transform network is mainly composed of 3 layer types: Convolutional, Relu, and Convolutional Transpose layers.

The convolutional layer performs down sampling, to decrease the size of the image. The smaller size image requires less parameters to be trained therefore can be trained faster, then the image is up sampled by the Convolutional Transpose layers to return the input size image. The transform layer takes as an input an RGB image of dimensions (height, width, 3) and outputs a different representation of the image (should be trained to produce a stylized version) of dimensions (height, width, 3)

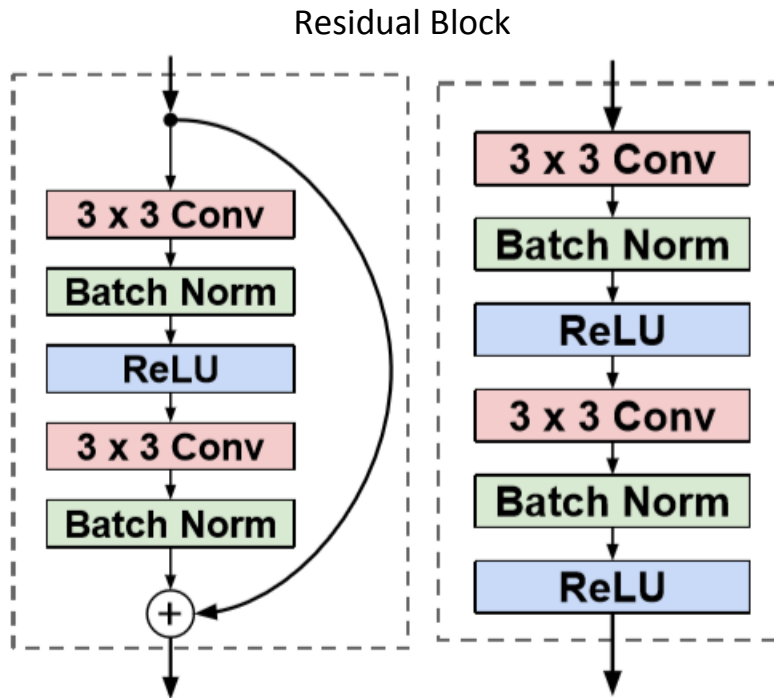
### Total System Architecture (Transform + VGG)



### Transform Network Original Architecture

| Layer                                   | Activation size            |
|---|----------------------------|
| Input                                   | $3 \times 256 \times 256$  |
| $32 \times 9 \times 9$ conv, stride 1   | $32 \times 256 \times 256$ |
| $64 \times 3 \times 3$ conv, stride 2   | $64 \times 128 \times 128$ |
| $128 \times 3 \times 3$ conv, stride 2  | $128 \times 64 \times 64$  |
| Residual block, 128 filters             | $128 \times 64 \times 64$  |
| Residual block, 128 filters             | $128 \times 64 \times 64$  |
| Residual block, 128 filters             | $128 \times 64 \times 64$  |
| Residual block, 128 filters             | $128 \times 64 \times 64$  |
| Residual block, 128 filters             | $128 \times 64 \times 64$  |
| $64 \times 3 \times 3$ conv, stride 1/2 | $64 \times 128 \times 128$ |
| $32 \times 3 \times 3$ conv, stride 1/2 | $32 \times 256 \times 256$ |
| $3 \times 9 \times 9$ conv, stride 1    | $3 \times 256 \times 256$  |

The original architecture of the transform network also contains 5 blocks called residual blocks, each of these blocks takes an input, passes it through 2 convolutional layers, and then outputs a linear combination of the original input and the output of the 2 convolutional layers.



### Training done:

We were able to train 5 Neural Networks, 3 for the same style (Starry Night), 2 of them using the same architecture of the transform network that is mentioned in the paper but they show that the random initialization produces different variations in the transferred style, and the third neural network uses a different architecture for the transform network where we used only 1 residual block instead of 5, the style transfer differed strongly from the other 2 models. The fourth network was given small weight for style transfer, therefore it mostly transfers colors rather than the style. The fifth network was trained on for low number of iterations but gives transfer of colors + dilution of the image details.

Similar Architecture



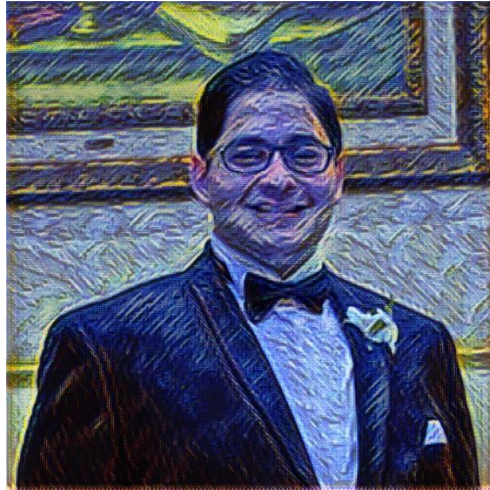


Different Architecture



We can notice that the different architecture (ours) produced stronger style transfer which decreased the strength of the original image's content (the portrait), therefore it is very suitable to use a weighted combination of the stylized and the original image.

50% original + 50% stylized



The original architectures show the transfer of color + the strokes of the drawing which is apparent in the hair and the chin area.

The different architecture shows the transfer of the color as well as the type of coloring (oil) which is apparent in the black suit, but on the other hand some of the details of the image (apparent in the face) are lost. The strokes are also apparent in the background.

### Color Transfer





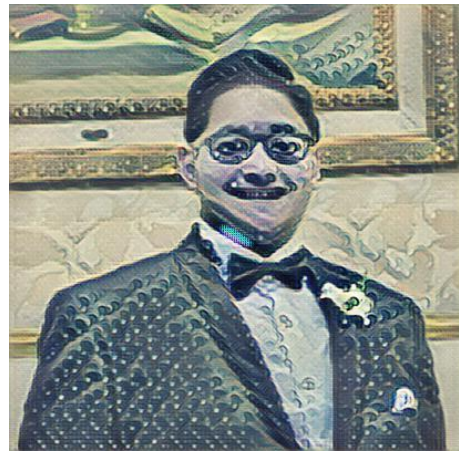
## Color Transfer + Details Dilution



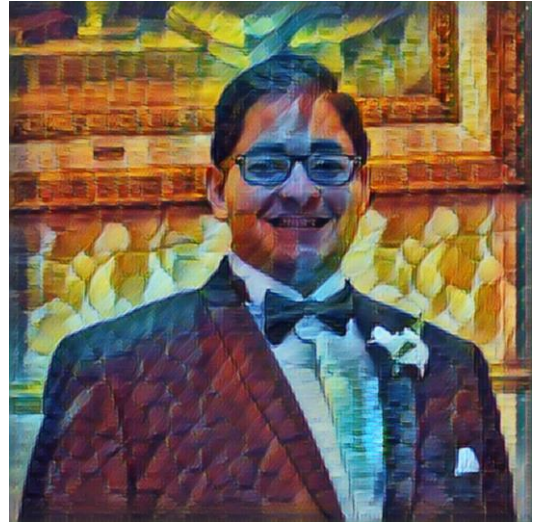
### Online models used:

We have included with the project 2 additional models (wave and rain princess) that were pre-trained on the same architecture that we used, so we were able to load their parameters, the difference between these models and ours is that they were trained using much more powerful GPUs, therefore able to be trained for a higher number of iterations on the data set.

Wave



Rain princess



**Another example (different training iterations):**

Content Image



Stylized Images

(Starry night)

1000 iterations



1500 iterations



3500 iteration



It can be seen that as the number of training iterations increases, the problems with the stylized images decreases. This is very apparent in the bright spots that keeps on decreasing as long as the iterations increase.



## **Comments and Comparison**

The first paper's implementation on our local pc took 6 hours to produce 1 image of high dimensions, but the same image took 45 minutes on a G2.2x Machine Instance on Amazon, but on average the photos with average dimensions took about 13 minutes.

The second paper's models were trained on multiple instances (G2.2xlarge, G2.8xlarge, P2.xlarge). Each 500 iterations took 15 minutes on the G2.2xlarge and G2.8xlarge machines, and 9 minutes on P2.xlarge, and the required number of iterations are 40000 iterations which should take 22, and 12 hours respectively.

The outputs of the first paper showed high style transfer with the required number of iterations, but the second paper provided low style transfer to average based on the number of iterations and the given weights for each loss type.

Due to the restricted resources, we weren't able to complete the 40000 required iterations, but the pre-trained

**References:**

<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>

<https://arxiv.org/pdf/1409.1556.pdf>

<https://arxiv.org/abs/1603.08155>

[http://www.cv-foundation.org/openaccess/content\\_cvpr\\_2016/papers/Gatys\\_Image\\_Style\\_Transfer\\_CVPR\\_2016\\_paper.pdf](http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf)

**Notes:**

[1] The ImageNet project is a large visual database designed for use in visual object recognition (1000 object categories) software research.

[2] Gram matrix shows the correlation between each filter of a certain layer and the rest of filters of the same layer.