CS512 Project: An implementation of Laplacian-steered neural style transfer

Olugbenga Abdulai
Department of Computer Science
Illinois Institute of Technology

November 29, 2020

Background

Neural style transfer is the process of rendering a content image in different styles. It is an application of computer vision that has been studied for decades with the first major breakthrough by Gatys et al. [1] where a deep convolutional neural network (CNN) is used. The gist of the neural style transfer algorithm as described in that paper [1] is that a CNN is able to extract content information from a content image and style features from an artistic image and thus these can be recombined by an optimization procedure to obtain appealing stylistic outputs. Figure 1 illustrates an example of transferring the style in “Dwelling in the Fuchun Mountains” (by Gongwang Huang) to an image of the great wall of China.

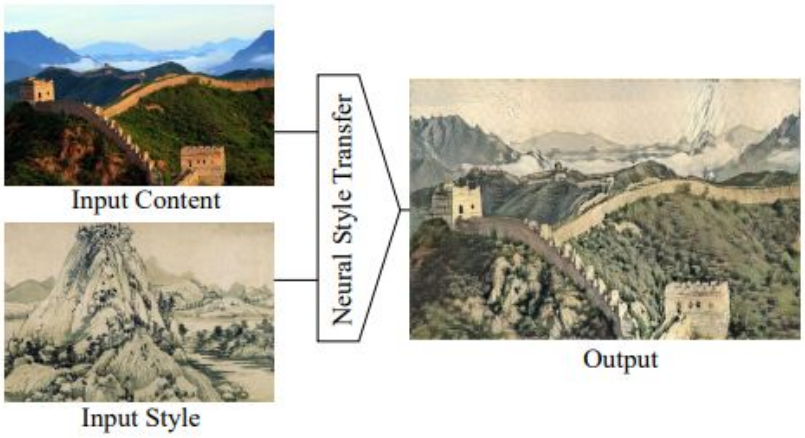

Fig 1: An illustration of neural style transfer results

A limitation of the results of the original implementation by Gatys et al. [1] is that it doesn’t capture properly the low-level features of the content image and thus the produced images are not very visually appealing with distortions and unappealing artifacts present.

Proposed Solution

In this report, we review a modification of the original implementation (*Gatys-style*) via an algorithm termed “Lapstyle” [2] which seeks to improve results by incorporating a Laplacian loss component into the original loss function. The original loss function optimizes the style and content representations using weights as hyperparameters.

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}}$$

$$\mathcal{L}_{\text{content}} = \frac{1}{N_{l_c} M_{l_c}(\boldsymbol{x}_c)} \sum_{ij} (F_{l_c}(\hat{\boldsymbol{x}}) - F_{l_c}(\boldsymbol{x}_c))_{ij}^2$$

$$\mathcal{L}_{\text{style}} = \sum_{l} w_l E_l(\hat{\boldsymbol{x}}, \boldsymbol{x}_s)$$

$$E_l(\hat{\boldsymbol{x}}, \boldsymbol{x}_s) = \frac{1}{4N_l^2} \sum_{ij} (G_l(\hat{\boldsymbol{x}}) - G_l(\boldsymbol{x}_s))_{ij}^2,$$

$$G_l(\boldsymbol{x}) = \frac{1}{M_l(\boldsymbol{x})} F_l(\boldsymbol{x})^\top F_l(\boldsymbol{x})$$

The equations above are used in Gatys-style. The total loss is a weighted sum of the content and style losses. The content loss is a normalized mean squared error comparing the target image to the content image at each chosen intermediate content layer. The style loss is similar but uses comparisons between *gram matrices* for the mean squared error. The gram matrix of an image denotes the style representation via correlations between different filter responses. The image synthesis is done as follows:

- Set the input of the CNN as the content image. Do a forward propagation and save the response of the content layer;
- Set the input of the CNN as the style image. Do a forward propagation. Compute and save the Gram matrices of all style layers;
- Initialize the target image. This could be randomly initialized, or copied from the content image;
- Iterate until reaching N iterations:
    - Do a forward propagation, and compute the total loss;
    - Do a backward propagation to update the generated image.

To better preserve the detail structures of the content image, more constraints can be added to the loss function. The Laplacian filter [3] is ubiquitous in computer vision as a second-derivative edge detector and thus can help extract these detail structures of the content image better. Thus, with a Laplacian loss, we steer the stylized image towards having a similar Laplacian to that of the content image and we define the Laplacian loss as the mean squared distance between the two Laplacians.

$$D = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

Fig 2: Laplacian filter

To apply a Laplacian filter to an image, we simply convolve the image with the filter. The Laplacian loss is defined as:

$$\mathcal{L}_{\text{lap}} = \sum_{ij} (D(\boldsymbol{x}_c) - D(\hat{\boldsymbol{x}}))_{ij}^2$$

The augmented total loss function is given as:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}} + \gamma \mathcal{L}_{\text{lap}}$$

Where γ is a hyperparameter controlling the weight of the Laplacian loss. This new optimization objective governs the Lapstyle algorithm.

Implementation Details

Input

The input images are fed into the program and resized to (300x400) pixels. A preprocessing step is done to normalize the image pixels to the standard expected by the VGG-19 model. Hence we subtract an array of pixels ([103.939,116.779,123.68]) corresponding to each channel before feeding the images as input to the model.

Model Definition

The VGG model is loaded from a *.mat* file [6] with layers defined as in the Figure 3. The content layer is chosen as *conv4_2* and the style layers are chosen as *'conv1_1', 'conv2_1',*

'conv3_1', 'conv4_1', 'conv5_1'. As suggested in the paper, one or more laplacian layers are added and a pxp pooling layer is added prior to it for smoothing.

```
['conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

   'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

   'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
   'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

   'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
   'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

   'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
   'relu5_3', 'conv5_4', 'relu5_4']
```

Fig 3: Layers in the VGG-19

Due to the fact that we're making modifications to the internal structure of the model and obtaining results from intermediate layers, we define custom functions and use Tensorflow 1.x in the code. We define a custom convolution function to cater for the convolution with the Laplacian filter and then the further layers as required. We also define a function for the pooling operation in the pooling layers. The convolutions use 'same' padding and the pool size defaults to 2, except for the pooling on the Laplacian layers which vary according to the depth of the layer. A single Laplacian layer is used.

Model Deep-dive

We define a function for the modeling and optimization operations termed 'styled_output'. It returns the styled output images from the chosen intermediate layers at each optimization iteration. The weighting assigned to the content and style layers are outlined in Table 1. We use a larger weight for the style layers and distribute it equally per layer, hence the layer weights of 0.2 per style layer (⅕).

Content layer	**Weight**	**Style layers**	**Weight(s)**
Conv4_2	5	Overall Per layer: • Conv1_1 • Conv2_1 • Conv3_1 • Conv4_1 • Conv5_1	5e2 • 0.2 • 0.2 • 0.2 • 0.2 • 0.2

Table 1: Weight hyperparameters

A Tensorflow computation graph is initialized and we run the preprocessed content and style images through the network in a forward propagation and then save the outputs from the specified intermediate layers in python dictionaries. The gram matrices for the style layers are normalized as the definition requires.

For the backpropagation step, our target image is initialized as the content image. It can be chosen as a noisy image correlated with the content image or just using the content image also suffices. The former option was tried but produced worse results. The content loss, laplacian loss and style losses are initialized to zero and then updated per iteration as the model performs the back pass steps. For convenience, we use the Adam optimizer although the paper suggests to use the LBFGS optimizer. The Adam optimizer suffices still. The hyperparameters used are as follows:

- Learning rate = 1.0
- beta1 = 0.9, beta2 = 0.999
- Epsilon = 1e-8
- Number of iterations = 1000

Every 50 iterations, we save the target image for reference. Before saving the image we revert the pixel processing step done prior to feeding it to the model by adding the pixels per channel back to the image. We also clip the image to the ‘0-255’ range and finally convert the tensor to an image with the PIL python library.

An important point to note is the results do not match those in the paper and certain modifications to the algorithm are made. This is discussed in detail in the results section.

Program Instructions

The code in the notebook (titled *LS-NST*) is well documented with comments and properly modularized. Simply run all cells to obtain the results. Hyperparameters have been defined as constants and can be changed easily as necessary. File paths can also be adjusted to suit your local directory.

Results and Discussion

For the sample content image, style image, and expected result [2] in Figure 4, the results (using the hyperparameters as listed in the prior section) at various iteration points are shown in Figure 5. The results clearly do not match the expected results

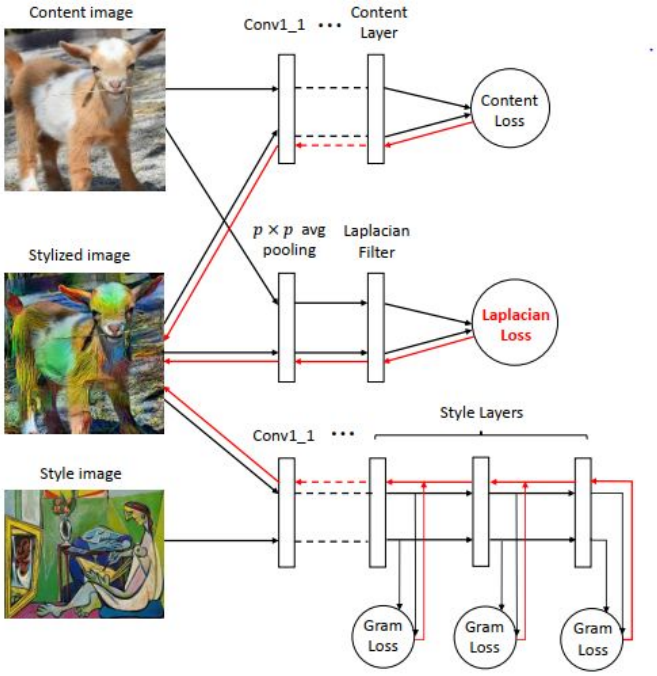

Fig 4: Network Architecture of Lapstyle. Black and red lines are forward and backward passes, respectively. Dashed lines indicate that there are unshown intermediate layers.

Fig 5: (L-R) Results at 50, 500, and 1000 iterations

On closer inspection it appears that the content image is being overshadowed by the style representation. The weights were adjusted to varying degrees but the results hardly showed any improvements. After intense debugging and some online research, it is discovered that

with eager execution on Tensorflow 2 coupled with a ‘total variational loss’, we can achieve similar goals in the task.

TensorFlow's eager execution [4] is an imperative programming environment that evaluates operations immediately, without building graphs: operations return concrete values instead of constructing a computational graph to run later. This eases the debugging problems that come with Tensorflow graphs in Tensorflow 1. On the downside, we are unable to fully customize (to the best of my knowledge) the internal workings of the model to add the Laplacian layers, but the variational loss component suffices to tend to the issue of unappealing results. The total variational loss imposes an explicit regularization term on the high frequency components of the image.

That said, we deviate from the Lapstyle algorithm and append the objective function with the total variational loss component instead, making for much less complex code. The new notebook is titled *LS-NST-2.* The few changes made are outlined below:

- Load the pre-trained VGG-19 model from Keras [5] instead with imagenet weights and discarding the top layer.
- Changing the content layer to ‘conv5_2’ as it produced better results over ‘conv4_2’
- Adding a total variational loss component to the objective function
- Some hyperparameter changes such as:
    - Style weight as 1e-3 and content weight as 1e4
    - Total variational weight as 3e1
    - Learning rate as 0.02
    - Epsilon as 0.1

The results using the prior images after 1000 iterations are shown in Figure 6 with the inclusion of other test images and results as shown in the paper. Figure 7 shows the expected results from the paper using the Lapstyle algorithm. From visual observations it is obvious that the total variational loss results (Figure 6) outperform the Lapstyle results (Figure 7) in most of the images. In Figure 7, Lapstyle is shown on the far right side.

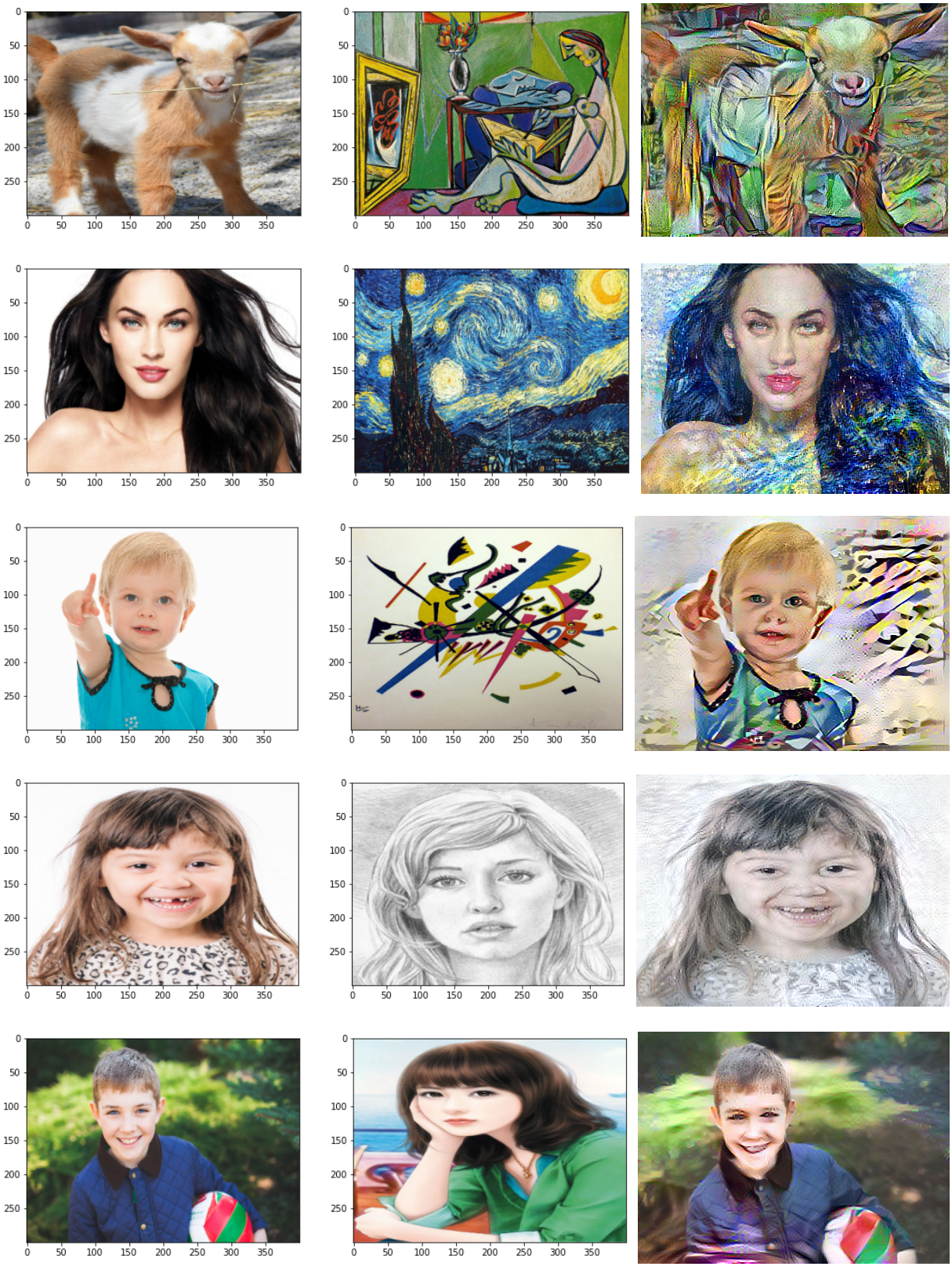

Figure 6: Results of using the total variational loss. The generated images are more visually appealing than the results from the Laplacian loss

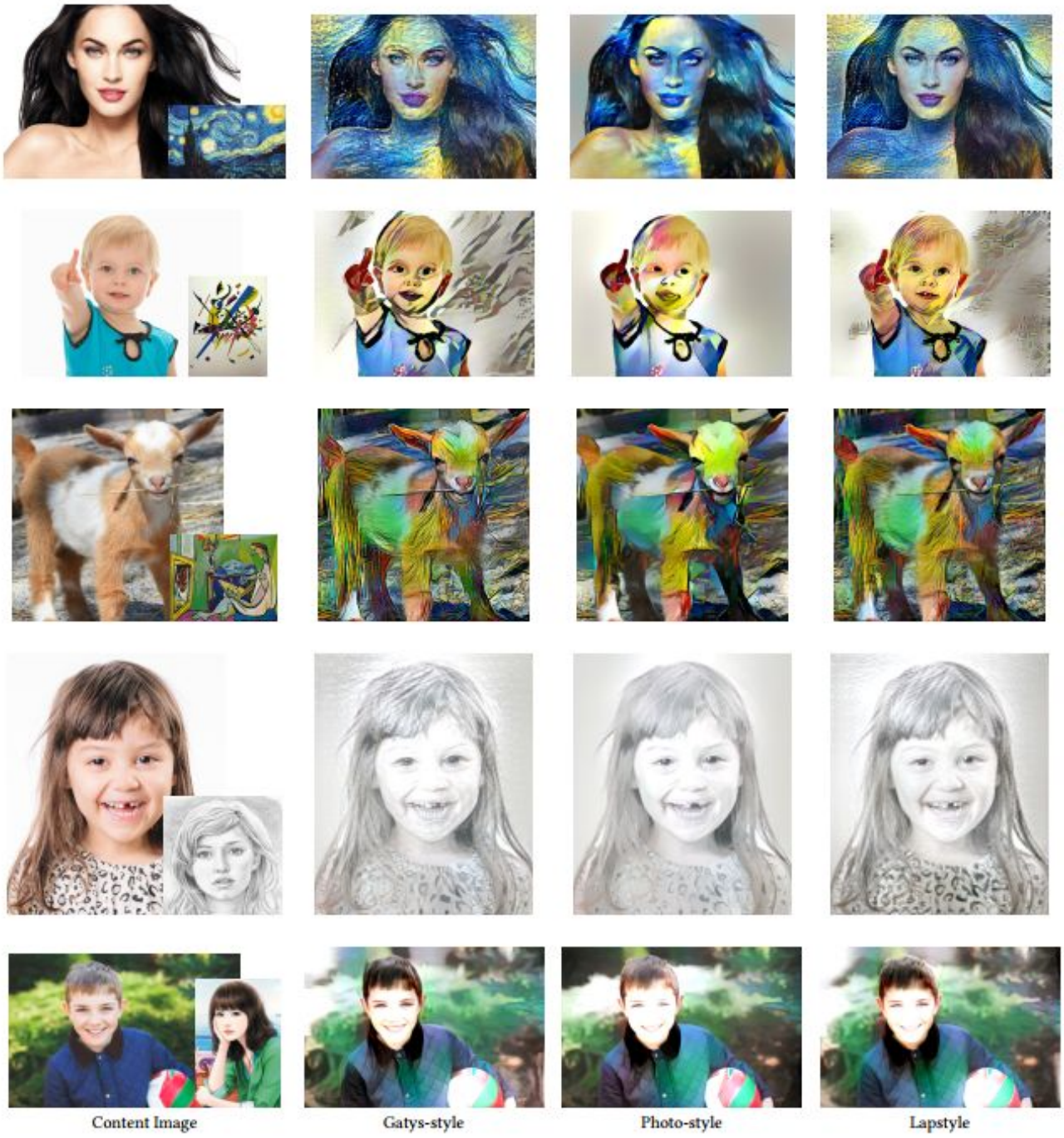

Figure 7: Results from Lapstyle. In the fourth image from the top, the girl's face color is completely lost which shouldn't be the case.

Conclusion

In this report, we tackle the problem of distortions and artifacts present in the results of the original Gatys-style algorithm by incorporating a Laplacian loss component in the objective function. With the unusual results obtained, we sought to try a variational loss method instead to solve the problem and obtained even better results than what the Gatys-style paper presents. We have proven with examples that the images produced by this variational loss method are more visually appealing, while remaining almost equally stylish.

Data

- https://drive.google.com/drive/folders/12C0_DLxGJo61N-unUX0uWa2cwZ4SvP0I?usp=sharing

References

[1] Gatys et al. 2015. *A Neural Algorithm of Artistic Style*. arXiv:1508.06576v2 [cs.CV]

[2] Shaohua Li et al. 2017. *Laplacian-Steered Neural Style Transfer*. arXiv:1707.01253v2 [cs.CV]

[3] Simon JD Prince. 2012. *Computer vision: models, learning, and inference*. Cambridge University Press.

[4] https://www.tensorflow.org/guide/eager

[5] https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG19

[6]https://www.kaggle.com/teksab/imagenetvggverydeep19mat?select=imagenet-vgg-verydeep-19.mat