Machine Learning Engineer Nanodegree

Capstone Project

Surya Krishnamurthy
August 20th, 2018

Neural Style Transfer

I. Definition

Project Overview

Computer vision is a field that has been gaining a lot of momentum in recent times. It powers many emerging technologies like facial recognition, self-driving cars and more. The invention of the Convolutional Neural Networks (CNN), especially, lead to several possibilities. CNNs possess interesting properties that arise out of its ability to learn higher level features from an image.

For this project, we particularly concentrate on **style transfer**. In style transfer, we render a content image in different styles. Several studies exploring how to automatically turn images into synthetic artworks have been performed since the mid-1990s. But, impressive results have been produced only after the post-neural era.

Problem statement

In this project, we will develop an artificial system based on a Deep Neural Network that creates artistic images of high perceptual quality. The algorithm will also help us develop an understanding of how humans create and perceive artistic imagery.

Given an image, the algorithm will separate and recombine the content from the image and style of an arbitrary image using neural representations.
The algorithm will be evaluated based on some quantitative metrics and qualitatively by visualizing the images generated by the algorithm.

Metrics

This paper (section 6.3) built a standardized benchmark to quantitatively evaluate this class of algorithms. The primary evaluation metrics are as follows:

1. Training time
2. Loss comparison with iteration
    a. Total loss
    b. Content loss
    c. Style loss

Further, we qualitatively evaluate the generated image saved during the checkpoints in our model to ensure that the algorithm is indeed producing the expected results. This evaluation relies on the aesthetic judgment of observers and is the criteria for stopping metric.

II. Analysis

Data Exploration

The algorithm depends on the VGG architecture pre-trained on a complex classification task (in our case the ImageNet dataset). So, no datasets were used to train the architecture. The algorithm is also a one-shot generation process - it doesn't require iteration through several images. But pre-trained weights (keras) of VGG16 are required for the algorithm to work.

The algorithm requires 2 major inputs - A content image and a style image whose style will be applied to the content image. The algorithm should work for any arbitrary images and should generate aesthetically pleasing images as long as the style image possess artistic textures that can be applied to the content image. Also note that using images with very low resolution might also degrade the algorithm's performance.

Exploratory Visualization

For the purposes of testing and demonstration we will be using these images:

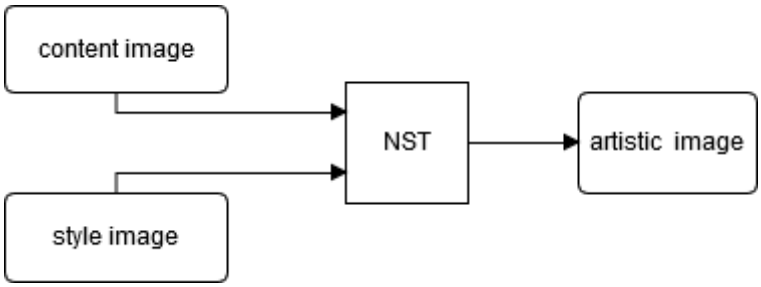
content image

style image

Image on the right is *The Starry Night* by Vincent Van Gogh, 1889 whose style will be applied to the content image on the left.

Algorithms and Techniques

We use Neural Style transfer for the problem. In Neural style transfer, we take a style image and apply its style, texture, and other artistic patterns, to a content image to generate a new visually pleasing image. Unlike conventional image filters that transform the image in the color space, style transfer alters the entire style of the content image while preserving the semantic content.



The method was first proposed by Gatys et al (2015). They proposed to model the content of a photo as the feature responses from a pre-trained CNN, and further model the style of an artwork as the summary feature statistics. The key idea behind their algorithm is to iteratively optimize an image with the objective of matching desired CNN feature distribution, which involves both the photo's content information and artwork's style information.

The algorithm uses perceptual loss functions to minimize the error between the content image and the generated image as well as the style image and the generated image.

Loss function:
In the algorithm, we ultimately minimize the distance of the canvas image from the content representations of the photograph in one layer of the network and the style representations of the painting in a number of layers of the CNN.

Content loss:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2 .$$

Style loss:
For the style loss we compute the correlation between the different filter responses. The feature correlations are given by the Gram matrix.

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

And the contribution of each style representation layer to total loss is

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Total loss:

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

The default values for α and β will be the values recommended in the original paper ($\alpha/\beta \sim 10^{-4}$). The 4th conv layer of VGG16 is used for the content representation and the conv layers 1-4 is used for the style representations

Benchmark

- **Random model**. We expect the model to perform better in creating styled images than a model that generates a random image or applies some random changes to a given content image. We expect to easily accomplish this task.
- **The original paper.** Our main benchmark is to generate images that offer visual experiences similar to that in the original paper shown below. But, our choice of inputs is different from the original paper.

III. Methodology

Data Preprocessing

Since the algorithm uses a VGG network, the input images have to preprocessed so that it is coherent with the format of the input image required by the pre-trained VGG model. Some preprocessing applied include:

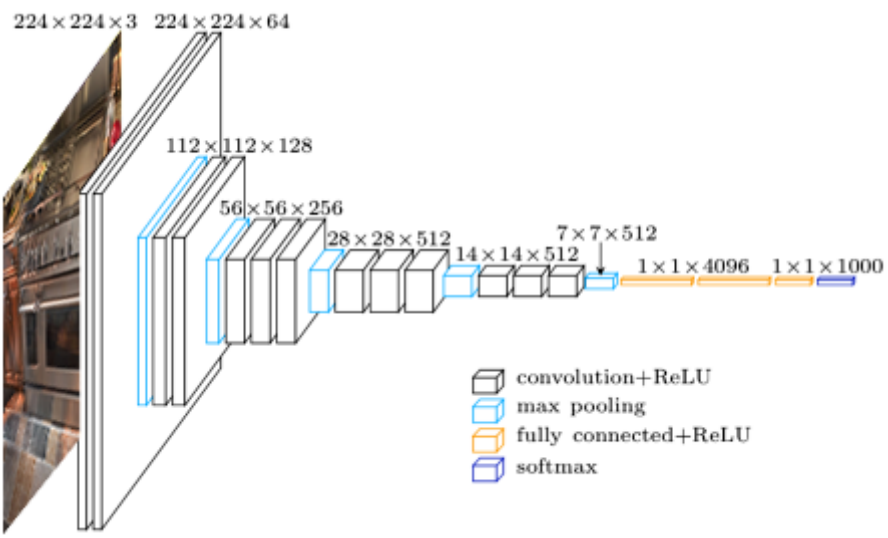
- Conversion of RGB to BGR
- Subtraction of mean values of each channel in the image
- Resizing images to (512, 512, 3) dimensions

Conveniently, keras simplifies some of these preprocessing steps.

The images also had to be converted to tensors with appropriate dimensions to be able to use with the keras deep learning framework.

Implementation

The algorithm was implemented with the help of the keras deep learning framework and other libraries like scipy and pillow.
We used the keras pre-trained VGG16 network. Scipy's l-BFGS optimizer was used, while the loss and gradient calculation was implemented in keras.



VGG16 network

The feature maps of desired layer of the network was then extracted for the content, style and generated image which was used for calculating the losses. Gradients were calculated for the input image with respect to the total loss. The total loss and gradient calculation functions were used with scipy optimizers for updating the generated image.
The checkpointing scheme saved the loss values every time the loss function is computed in the optimizer and the generate images were saved once every few epochs. Callback for displaying the progress of the optimization process was also implemented for the scipy

optimizer. This allowed easy evaluation of the algorithm's performance during the training process.

Post processing

As the generated images needs to be saved in the disk, a post-processing step was required. This phase undid some of the pre-processing steps applied initially. This includes converting BGR back to RGB and adding the mean values to each channel.
After post-processing, the generated images were comprehensive and easy to visualize.

Training environment

The algorithm was trained using tensorflow backend with GPU support. The evaluation metrics (training time) shown are for GeForce 940MX dedicated graphics. They could further be improved by using a better GPU card. Cloud services were also used during the development process. The algorithm is also functional in a CPU environment but the runtime would be way higher.

Abstracting the algorithm

The entire algorithm was then implemented as a simple python module that can be used with other applications if required.

```
from neural_transer import style_transfer

cnt_image = 'img/content.jpg'
style_image = 'img/style.jpg'
output = 'output/'

style_transfer (cnt_image, style_image, output)
```

Refinement

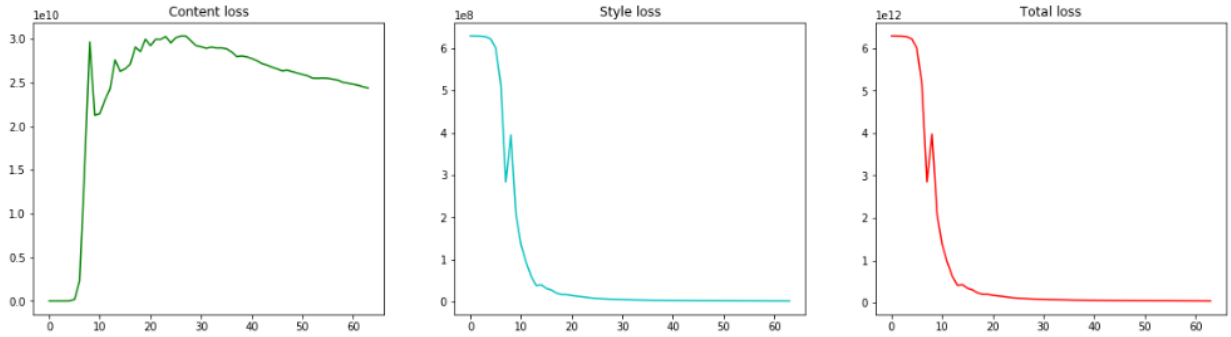
As opposed to the trivial SGD algorithm, we used the Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm as it is shown to converge faster for this particular type of problem. L-BFGS is a second order optimization method that works well compared to other popular methods when memory requirements can't be met.

Further, the default mode for creating the canvas for generating image used the content image. This prevented the algorithm from rebuilding all the main picture elements from the content image.

IV. Results

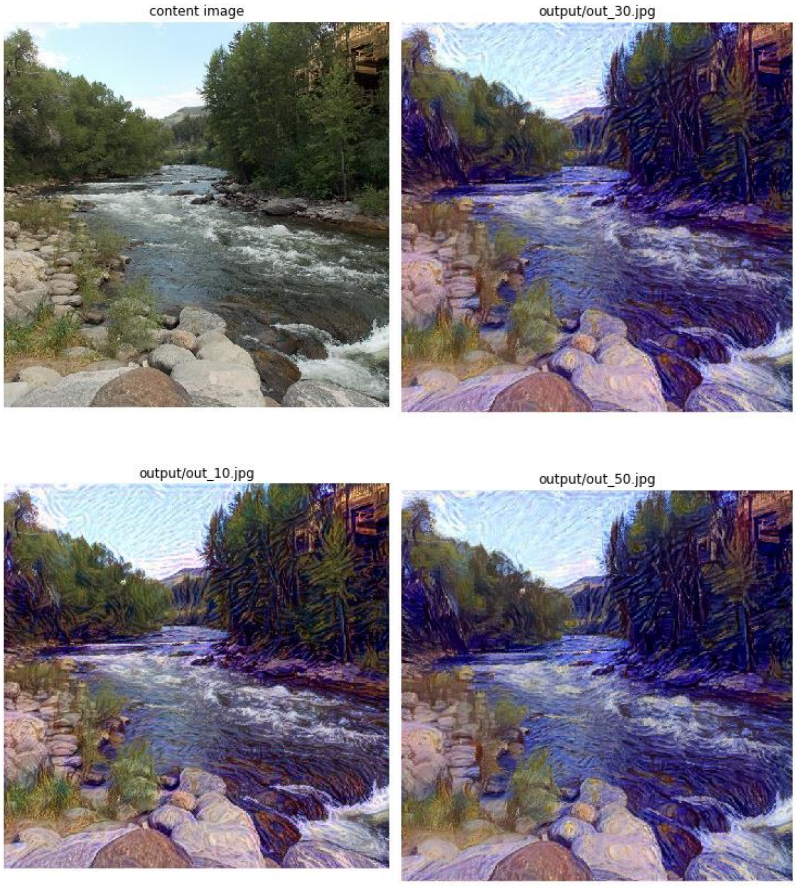
Model Evaluation and Validation

The model gave really good results. The training process was stable and we can observe a steady decrease in the loss values. For a content-image-initialized canvas the loss value stops decreasing after some 40 epochs. As a result, the generated image also doesn't show any improvement after 40 epochs.



Loss comparison with iteration

The image generated during the training once every few epochs is shown below. We could see how the algorithm learns to apply the style elements to the content.



In a machine with a GeForce 940Mx GPU, the training took about 554 seconds for 60 epochs.

Benchmarking

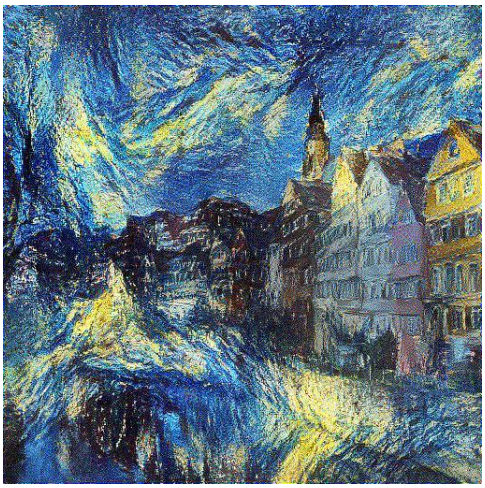
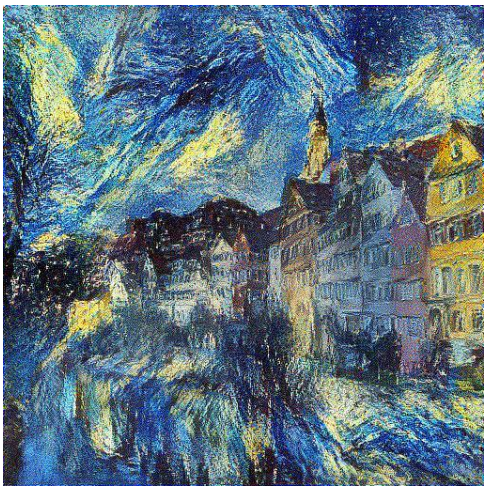
The choice of hyperparameters was different for the images generated for comparison with benchmarks. The following hyperparameters were used

- $(\alpha/\beta \sim 10^{-3})$
- Epochs = 600
- Canvas initialization with white noise image

The algorithm was run on a server instance with a GeForce GTX 1080 Ti GPU. It took about 13 hours to train. It was observed that results were better with this setting but was expensive to execute. The final generated images are presented below.

Justification

The algorithm works well for most images. Better images will be produced based on the compatibility of the style image with the content image.
Some images generated by the algorithm which are comparable with the benchmarks are shown below

V. Conclusion

Free-Form Visualization

The following image generated by the algorithm demonstrates the algorithm's performance.

The algorithm's ability to apply artistic styles to a given content image is apparent from the picture. Algorithm performed well for different content and style images as well.

Reflection

The entire project pipeline can be summarized as below:

1. Initial problem defined and analyzed
2. Potential solutions explored and ideal one identified
3. Benchmark created
4. Algorithm implemented
5. Results evaluated and visualized
6. Reusable module created

The project required implementation of few low-level techniques for calculating the losses and gradients which was particularly difficult to implement. Debugging the algorithm was also quite challenging as they were several functions that required testing.

The algorithm was the most interesting part of the project. It was truly fascinating and helped me understand the capabilities of deep learning technologies. I also gained a lot of experience in implementing deep learning based solutions.

Improvement

There are a lot of scope for improvements in the project. Use of the VGG19 architecture instead of VGG16 can produce more interesting results but would require higher computational resources. Different types of optimizers were not explored in the project, some optimizers can produce significantly different results.

This paper reviews the several improvements made to the algorithm over time. These methods can also be analysed and can be used to improve the performance of the algorithm in areas like training time. This would allow the algorithm to be extended to real-time systems.

Many I/O operations like saving images and printing could be made asynchronous making the training process non-blocking.