Cole Hoffer
choffer@mit.edu
6.815

PSET 10: Style Transfer

Introduction

I chose to do my final project/pset on the style transfer option listed in the handout. This is because I thought it would be cool to do a project more geared towards machine learning, as our past assignments didn't cover it even though it came up in lecture a decent amount. From the machine learning options listed in the handouts, I thought this was by far the most interesting and stimulating idea, plus it didn't involved Halide, and the pains of Halide.

Background

Style transfer applications are built upon the backbone of the deep learning "revolution" and image recognition applications. The big push for style transfer applications came from the "Image Style Transfer Using Convolutional Neural Networks" paper linked in PSET 10 by Gatys et. al.. This was the first majorly successful attempt to use deep learning image classification architectures to perform style transfer, and what my project was based around. Since that paper, there has been a good deal of work expanding upon those ideas to make the style transfer even more realist. In "Deep Photo Style Transfer" by Luan et. al., the researchers were able to use a "Matting Laplacian to constrain the transformation from the input to the output to be locally affine in colorspace". In simpler terms, they were able to more successfully retain the photo realism of the style reference image and more accurately apply it to the base content image through semantic segmentation. Another expansion came from Zhang et. al. in "Multi-style Generative Network for Real-time Transfer". This paper was unique in that instead of using typical feed forward network to generate the images, the researchers introduce a new "CoMatch Layer" that helps the model learn to match the second order statistics as image style representation, which in terms decrease generation time and improves the final generated output.

Implementation

Model

For all three of the papers mentioned in the background, the VGG-19 model trained on ImageNet weights was the go to choice for style transfer projects. The VGG model group is popular for many deep learning vision problems, and as suggested, the VGG-19 specifically has 19 layers total layers. For style transfer, we actually only need to worry about some of smaller subset of layers, as the researchers in the Gatys et. al. paper were able to show an original image could be relatively reconstructed just by using five layers (one from the first five layer blocks). Thus we can pull data from those five layers of the style reference model, and use them in the last block (block 5) in our base image model.

Loss

The overall loss of our model is actually a summation of three different loss components, the base image content loss, the style loss, and the total variation loss. The base image content

loss implements the same content loss function from the Gatys et. al. paper linked from the PSET, where the loss is a simple squared-error loss between layers of the original base image A_l and the recently generated image G_l (equation 1).

The style loss is also implements from the Gatys et. al. paper but it is a bit more complicated, as the “representation of the style of an input image”, uses a “feature space designed to capture texture information”. In as plain english as possible, we are essentially trying to find the feature correlation between different layers in the model, which will turn allow us to obtain a texture representation that omits the arrangement of the pixels. We can calculate these feature (F_l) correlations using a Gram matrix (equation 2). Then we can calculate the total style loss by finding the square-error between the feature correlations of the generated and original images, which is then normalized against the input image dimensions (equation 3).

Finally, while not included in the Gatys et. al. all paper, I included a third and final loss function that calculated the variance across the base, style reference and generated images. This step was a part of many non-academically-published style transfer projects, with the idea being that it can help prevent the generated image from straying too far from either of the input images. Luckily for me, tensorflow has a built in function called “tf.image.total_variation”, which literally calculated the variation between given images, so it was simple enough to just use that.

Equation 1: $$L_{content} = \frac{1}{2} \sum_{l} (G_l - A_l)^2$$

Equation 2: $$Gram(a, b)_l = \sum F_{a,l} * F_{b,l}$$

Equation 3: $$L_{style} = (\frac{1}{4 * channels * width * height})^2 \sum_{l} (Gram(G_l) - Gram(A_l))^2$$

Training specs and implementation decisions

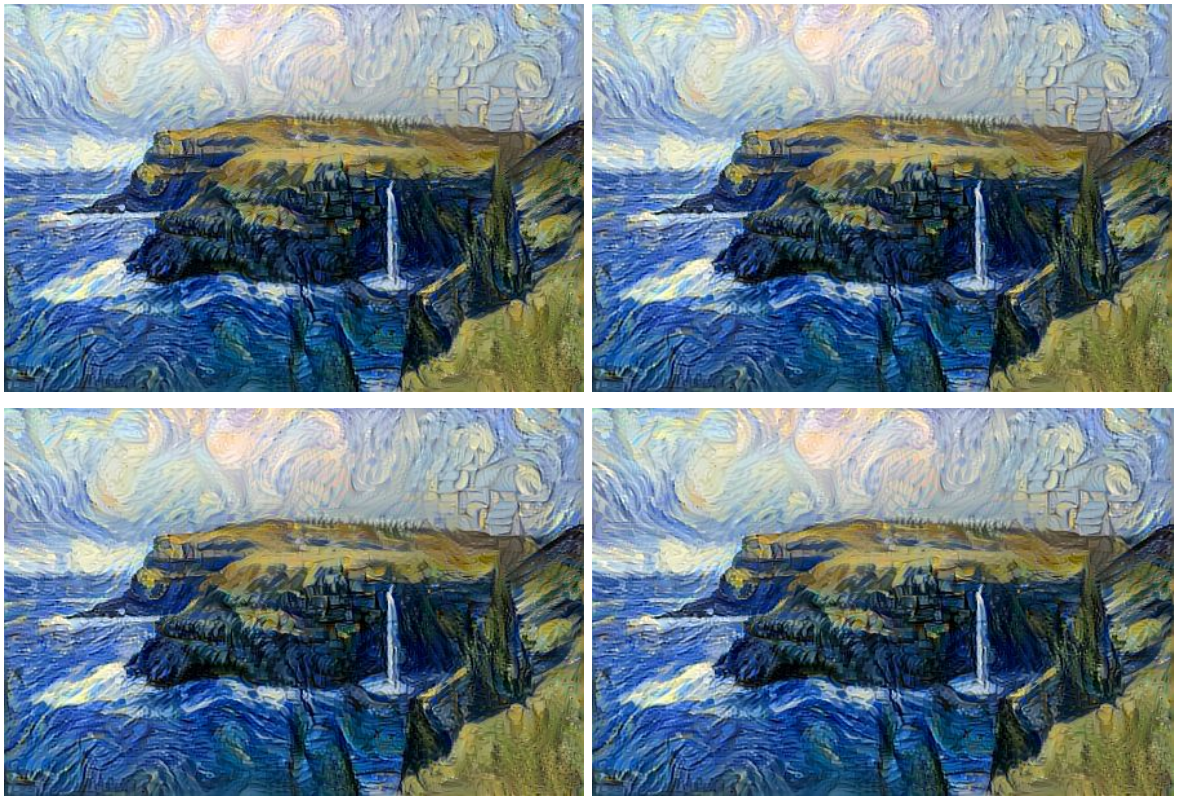
- For the optimizer, I again followed the Gatys et. al. all lead and used the L-BFGS optimizer (nice and easy to use with scipy). Another one that seemed to be commonly used was the popular Adam optimizer, but I had some trouble getting it to work well with my code so that’s why I stuck with L-BFGS.
- After a couple of trials, I found that 100 iterations of my implementation was a good balance between output quality and time constraints. Once past 100 iterations, the style didn’t need to change much, and I actually saw that physical artifacts from the style reference image starting the sneak in (like mountains from the Starry Night image).
- For the loss weighting, the big thing was just making sure the style loss weight was much higher than the base content weight. In the Gatys et. al. paper, their best results seemed to be a ratio of 100:1 (when comparing 10:1, 100:1, 1,000:1 and 10,000:1). For my implementation, I ended up using a ratio of 20:1, but I think that was in part because I may have been performing more iterations then they were, and thus had more time to bring in style components.

- I also hardcoded in a requirement for images to be 300x450 pixels, just because that ensured that the style and based images matched as much as possible before hand, and because it was a good compromise between image quality and computation time.

Results/Examples (it worked!)

Base - Style / Iteration 0 - Iteration 20 / Iteration 40 - Iteration 60 / Iteration 80 - Iteration 100

Faroe Islands → Starry Night



NYC → “Rain Princess”