Neural Network Pastiche

Cara Todd, Mengjia Luo, Neha Manu, John Herman
Department of Computer Science, University of Virginia, Charlottesville, VA 22904
[cet9me, ml6uk, nm8ff, jmh2bj]@virginia.edu
We would not mind having our final write-up posted.

1. Introduction

The projects goal was to apply an artistic style to non-art images by taking two images, one piece of art (the style image), and one non-art image (the content image) and applying the style of the style image to the content image while keeping the content recognizable. There have already been papers exploring this topic. The current developments with style transfer either focus on either making style transfer faster, real time style transfer, or improving the visuals. Most importantly, transferring high level style instead of just local style. The current state of style transfer is very good at transferring certain art and artists, mainly those with very distinct textures. This project is concerned with increasing the quality of style transfer, not speed.

2. Related Work

There were several papers that we consulted at the different stages of our project. Our primary goal, which was basic style transfer, was implemented based off of *A Neural Algorithm of Artistic Style* by Gatys et al. We also referenced https://github.com/titu1994/Neural-Style-Transfer while writing the code. Once we acheieved basic functionality, we attempted to improve the quality of the output images using Markov Random Fields (MRFs). The MRFs would allow more high level style information to transfer, giving the resulting image a more believable and sophisticated look. This was based on research conducted by Li and Wand in *Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis*. As part of MRFs, we used the PatchMatch algorithm within *PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing* by Barnes et al.

3. Model

The core of the model is a pretrained VGG-16 neural network model with weights attached. It is given either the content image, or a white noise image. Then, each iteration the algorithm performs gradient descent backpropogation, where it optimizes the original image to minimize loss between the content image and the style images.

Content and style images were preproccessed by zero-centering by mean pixel and switching channel ordering (RGB to BGR). We selected the fifth layer of content image for representing high level content and used all five layers of style image to represent style by applying Gram Matrix to extract feature maps. We then performed ten iterations of forward pass and backpropagation through gradient descent to minimize the losses calculated by mean squared difference. The image was deprocessed by adding back the mean and switching channel ordering. The code is included in main.py.

When we tried to implement style transfer using MRFs,we used the PatchMatch to optimize the style loss calculation in feed forward. The content loss calculation was the same. We first tried to implement this the brute force way as outlined in the MRF paper. For each patch in style image, we found the nearest neighbor patch using normalized cross-correlation via an additional convolutional layer. We then constructed an image pyramid, in which each layer is scaled using the output from its previous layer.This is included in mainmrf.py. We also tried to implement the Patch Match algorithm in Barnes' paper. This initializes random offsets in sample image then iteratively propagates to nearby pixels. If certain ones match the conditions, then it's a good patch match. This is included in mainpmmrf.py.

4. Challenges

One of the first difficulties we encountered was acquiring a laptop with a powerful GPU so training would not take hours to complete. Furthermore, the need for specialized hardware became a bottleneck on development, as only one person could fully run the program at a time. We also had installation issues with Keras and making it compatible with the CUDA backend.

In our implementation of MRFs, the library we tried to use, Images2Neibs, was theano based and could not be implemented with tensorflow. It is more frustrating because

Figure 1. A basic style transfer with the input content (left) and style images (center), which result in the output image on the right.

Figure 2. A basic style transfer with the same content image (left) as Figure 1 and different style image (center), which result in the output image on the right.

online documentation indicates that the libaray is compatible with tensorflow. This backend issue prevented us from finishing the MRF implementation.

5. Results

The results from our initial implementation of basic style transfer were quite successful. We used several different content and style images to get a variety of output images, which can be seen in Figures 1, 2, and 3. Our program is also capable of transfering multiple styles onto a single image, which can be seen in Figure 4. We could also adjust the weights to change how much influences style and content separately have on the generated image. The differences can be seen in Figure 5. Another option is to select a different layer for content extraction. The differences can be seen in Figure 6.

Unfortunately, we could not fully implement the MRF loss function due to backend issues outlined in Challenges. Therefore we do not have results for it. However, we have included our codes for both the brute force implementation of MRFs and the PatchMatch algorithm implementation of MRFs in our deliverable.

6. Individual Contributions

Cara Todd: Installation, understanding and configuring the neural network models.

Mengjia Luo: Image processing, patchmatcher, research, and write-up

Neha Manu: Patchmatcher, results, and write-up

John Herman: Project proposal and parse arguments

Figure 3. Another basic style transfer with different content (left) style images (center) from those in Figures 1 or 2, which result in the output image on the right.

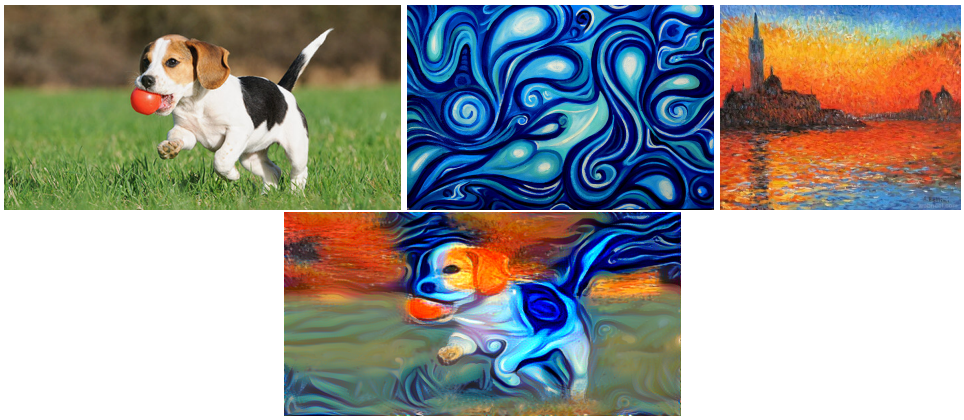

Figure 4. A multiple style transfer with the input content (top left) and two style images (top center and right), which result in the output image below them.

Figure 5. A multiple style transfer with varying weigths for the two style images. The first row includes the content and style input images. From left to right in the middle row, the wave to swirl weigth ratio changes from 0-5 to 1-2 to 1-1 (equal weight). In the bottom row, the weights change to 2-1 on the left and 5-0 on the right.

Figure 6. A style transfer with varying layers for content extraction. The content and style input images are shown on the top row. The bottom row from left to right shows the output images for content extraction at layers 3, then 4, and then 5.