Image analogies V.S. Neural Style Transfer

Overview

My final project proposal is image analogies versus neural style transfer. While the former makes an analogy of a given pair image and another content image, the latter combines the content image and the style image. They try to solve different problems but share the same usage such as image filters. Though neural style transfer is definitely much fancier, I would like to see what image analogies can achieve by simple search. The focus of this project is the implementation of image analogies since it is the computer graphic paper. The implementation of neural style transfer is a part of the tutorial of deep learning framework-Tensorflow. Considering the limited time for the project, I do not try to implement neural style transfer in my own way. In addition to the implementation, I will also compare the results of both and share my thoughts

Image Analogies

Image Analogies takes a pair of image and build the relationship between them and then synthesizes the effect of another image. That is, A: A' = B: B' where the B' is the output of the algorithm while the rest are the inputs. Here is my result. Given the inputs A, A', and B

We can synthesize another picture that has the same effect from A to A'. Here is my result of B':

The challenge of this set is brightening the image. However, my implementation seems to misplace some pixels in the cat's back. I have tested with different parameters but it did not do better. I will introduce the implementation details and their difficulties in the next paragraph, which are likely to be the problem of the result above.

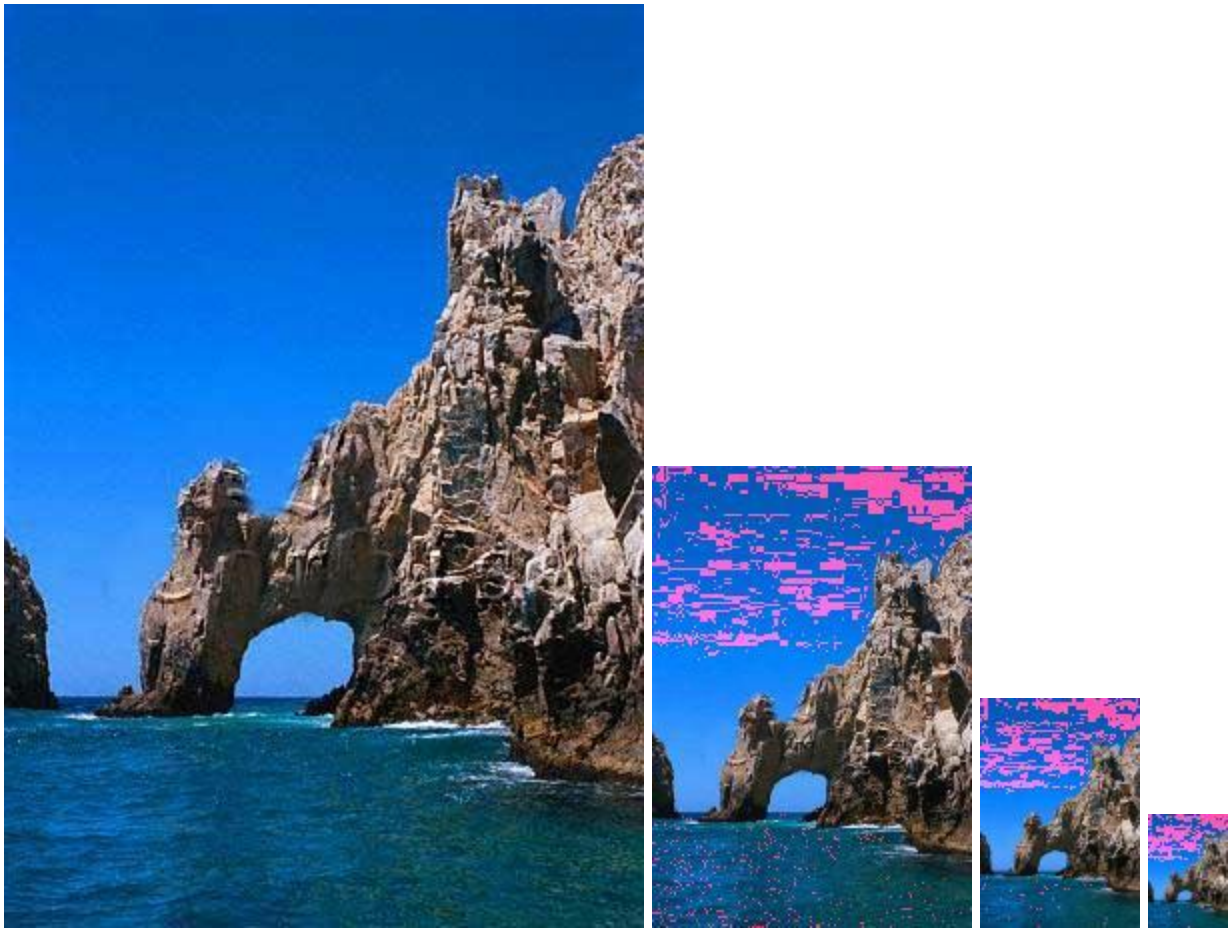
Feature vector: YIQ values

As noted in the paper, YIQ is a better representation than the original RGB value because YIQ keeps track of images' luminance. I don't really know what it means but it is fairly simple to compute the YIQ value, which is just a matrix dot product of RGB. The feature vectors are later being used by the BestMatch algorithm, which determines which byte is the most suitable for being copied into the output B' image. The feature vector affects the whole algorithm because it is how a pixel is represented. I have tried both making the feature vector the whole YIQ values and making it only Y value, but the YIQ one generates the better result, where everything does not lose its shape.

Gaussian Pyramid

To capture more information, not just the surrounding pixels, image analogies utilize gaussian pyramids to gather the coarse pixels, which will be later used as the evaluation metric of the pixel matching. The implementation of the Gaussian Pyramid is straightforward: simply turns the current picture into one fourth repeatedly. Intuitively, I just have to merge every square of 2X2 pixels into one. For simplicity, I choose to take the average value of the 2 by 2 pixels for each channel. Later did I know the smoothing technique matters, which probably renders much more elegant down-scaled pictures. Nonetheless, because of the limited time, I decided to move on and implement the heart of image analogies: bestMatch algorithm.I have tried two method: taking a gaussian pyramid of RGB channels and YIQ channels. The former works much better:

while the YIQ introduce lots of noise. It seems like that taking the average of Y, I, and Q values is a bad idea.



bestMatch: the search algorithm

The core search algorithm is actually two parts: approximate match and coherence match. While the approximate match is the core of this paper, the coherence algorithm makes the pixel connected smoothly. The approximate search provides one candidate match pixel and the coherence match provides another. The search synthesizes the output picture from the coarsest to the finest so that there is more information for the match at each finer level. The original paper uses scan-line order which means rows are completed before columns. I do the opposite because the indexing of width and height are sometimes confusing. I used python3 for constructing the image, where the built-in list and numpy sequence are popular choices for pixels. Numpy and conventional lists takes y-axis value first and then the x-axis. It is good that they are similar. However, the image structure I used with python3 Pillow takes the y-axis first and then the x-axis. My code was not clean enough to make the change of the different axis so that I used the one I got it working (kind of). There is a coherence parameter in the function. The larger it is, the more likely that the algorithm returns the pixel from the coherence match. I choose 0.5 as most of the implementation seems to do it.

Approximate search algorithm

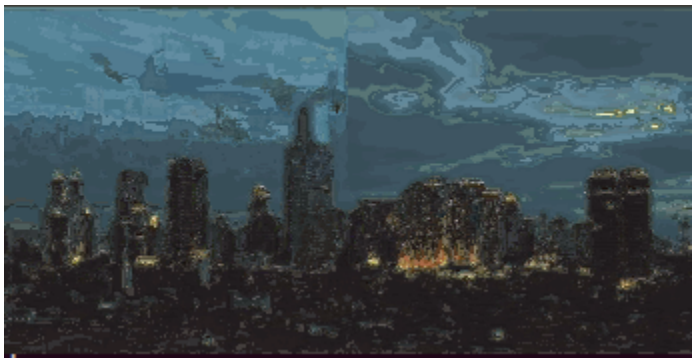
The algorithm is straightforward: take the most suitable pixel and map it from A' to A, which will be later mapped from B' to B. Naively, it can be achieved by brute-force method. Just retrieve vectors pixel by pixel and compute the euclidean distance. Specifically, the search grab 5x5 surrounding pixels, including itself, and the coarser 3x3 pixels and reshape the vector as its feature. However, the search algorithm is wrapped inside a loop for gaussian pyramids so that the time complexity is $O(N^4)$ where N is the number of pixels (assuming A, A', and B share the same size). I successfully get a 80 x 115 image working brute-force algorithm but the program stalls if the size grows four times. In fact, I run the program overnight and the 320 x 426 cannot be shown. Therefore, I have to use the approximate nearest neighbor algorithm for getting the approximated solution. ANN algorithm is recommended in the image analogies paper. It is an approximate algorithm for the nearest neighbor search problem.

Coherence search algorithm

The coherence search algorithm is also about neighbor search problems but the search space is in constant time. Coherence search simply returns the pixel that has already formed in B' that is considered matched. The implementation of this algorithm is much the same in the approximate search but simpler. At this point, I have tested the outputs of approximate match and coherence match. Unfortunately, I did not notice any notable change.

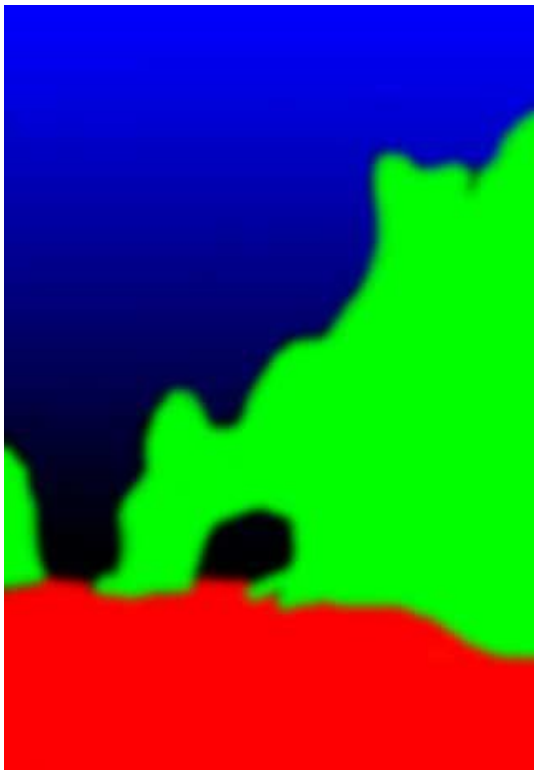
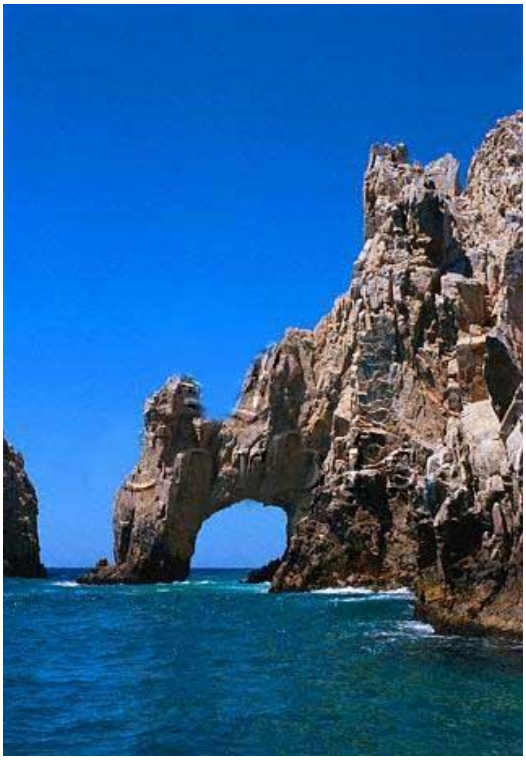
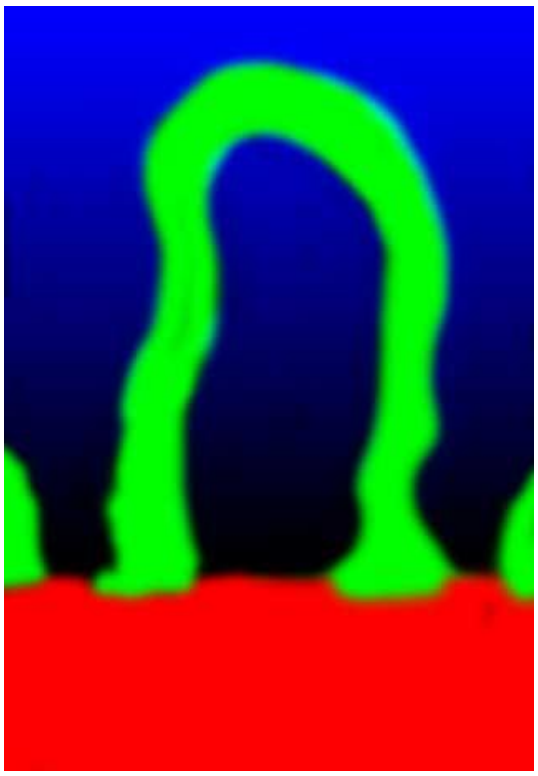
Results

The challenge of this picture is recolorization. Though it looks like a forest in the front but the trees and the sky in the back are blurred and some weird pixels happens in the top sky.



The blurring effect works except the upper-left part looks weird.

CS551 Final Project student: Ping-Jui Liao 933506277 Oregon State University


The rock and the sky are separated but the roar and the sea are inseperable.
The upper left pixels look weird for all images. I tried making changes in many aspects but it failed in different ways. I assume the problem might be related to the gaussian pyramid because

that is the part where the coarser layer is. The Gaussian pyramid is also used in the best match algorithm so that is a big part of the program that might go wrong.

Neural Style Transfer

As I noted in the first paragraph, this project is more about implementing image analogies than neural style transfer because neural style transfer is much more about machine learning than graphics. I simply walk through the tutorial of Tensorflow and grab the script here. My experience of machine learning or deep learning is all from the online courses. I understand the core concept of machine learning and am able to follow the procedure to construct some machine learning models. I have also studied the math and logic behind the neural network. However, I do not have experience in collecting data and building a brand new model on my own. To be honest, I cannot explain why convolution neural networks work here but the others do not. In this section, I will provide what I have learned to explain the algorithm.

Convolutional neural network(CNN)
Convolutional neural networks make cross-relation operations, which is element-wise multiplication. Images are represented with pixels with channels of color, luminance, and other features. Therefore, imagery vision has mostly adopted CNN for the training model. In addition to the cross-relation operations, CNN leverages pooling layers in convolutional layers to reduce the size of representation to speed up computation, as well as make some of the features detect be more robust. As any neural network would do, parameters such as the loss function should be chosen to get the solid result. Like most imagery machine learning, neural style transfer construct models in CNN.

VGG19
VGG is one of the standards of CNN. VGG family share one feature in common: all the convolution layers and pooling layer are uniforms with the same hyperparameters. I am not sure why VGG19 is used in neural style transfer but I believe the reason is to simplify the process.

Loss function

Neural style transfer chooses ADAM, Adaptive moment estimation, as its loss function. ADAM and RMSprop are the two rule-of-thumb loss function. The choice of optimization algorithms does affect the result. However, it is hard to explain one might work while the others are not since the loss in computed repeatedly in a large number of neurons.

Neural Style Transfer at a high-level

Neural style transfer's goal is to find the generated image. The overall workflow is two-step in general: initializing the image randomly and using gradient descent to minimize its cost function. The cost function is simply the linear weighted cost of content image and cost of style image. Two parameters, content parameter and style parameter, must be provided. In this way, The concept of style is defined as the correlation between activations across channels.

Neural Style Transfer V.S Image Analogies

Regarding the inputs

It is obvious that the requirement for neural style transfer is quite loose as compared to image analogies. Not only does neural style transfer require fewer images but it also has no constraints on the given images. Any two given images, in spite of unexpected results, can be run for neural style transfer. On the other hand, image analogies are strict to the input images. First, the image A and A' must be highly related or the result will be completely random. What's more? The size of A and A' should also be the same so that it can perform the matching. This disallows some high-level match for objects. For example, say that A' is scaled by 2, any human can still make analogies based on the A and the scaled A'. Last but not least, this derives another problem that image analogies cannot draw the very first source pair of image; that is, A and A' must be generated by other algorithms or by human effort. This limited image analogies for many use cases.

Regarding the output

As I implemented the image analogies, I realize that all the matching algorithms are mapping pixels from A' to B'. Therefore, the output image cannot have any color that does not belong to A'. Though this might not be a problem if the human cannot detect this physically, the limitation does exist.

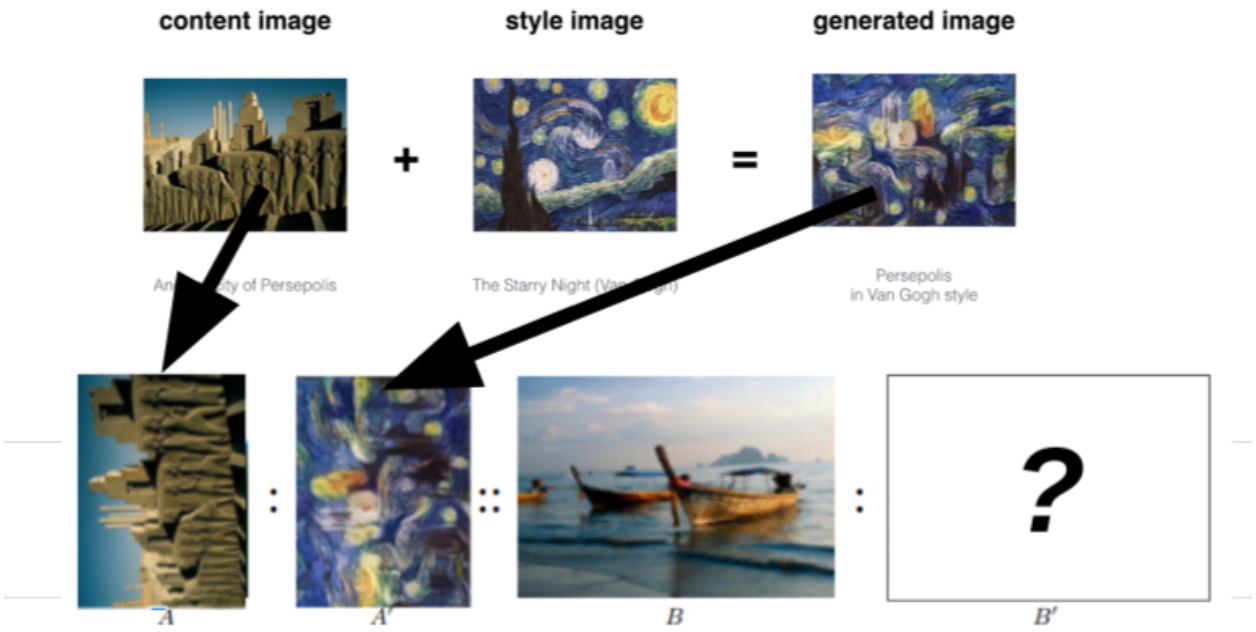
Regarding the building time

The building time of image analogies is much shorter than the neural style transfer because it does not resort to lots of neurons. It would be interesting to compare them in the computer system but I lack the skills to implement the detail of neural style network and approximate nearest neighbor search. I thought it won't be fair since I do not fully understand these libraries.

Style concept in image analogies

The neural style transfer paper claims that they are the first to synthesize pictures with high-level styles instead of texture change. Though style and texture are different things, they sometimes share the same result. As these concepts cannot be strictly defined and proved, the best way to evaluate this is by looking at examples. Here I grab the result of neural style transfer

and its content image and make image analogies of them. The following diagram is the experiment I have in mind.



My experiment should show that whether the “style” can be copied by the image analogies. Here are the results.

A:The content image of NST, A’: The Result of NST

B: a castle B’: poorly behaved


A:The content image of NST, A': The Result of NST


B: a castle B'

We can see that the structure of the castle is still preserved. However, it cannot be considered an art piece at all. Though it might be something wrong with my implementation, I think image analogies cannot make an art piece even if it's fix. That is, the art style really is much high-level structure beyond texture mapping.

Reference

[1] Hertzmann, Aaron, et al. "Image analogies." *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001.
[2] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." *arXiv preprint arXiv:1508.06576* (2015).
[3] Dillon, Joshua V., et al. "Tensorflow distributions." *arXiv preprint arXiv:1711.10604* (2017).
[4] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
[5] Arya, Sunil, et al. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions." *Journal of the ACM (JACM)* 45.6 (1998): 891-923.
[6] Jordan Mecom, "Image Analogies",
http://jmecom.github.io/projects/computational-photography/image-analogies, 2016