METROPOLITAN STATE UNIVERSITY[SM] OF DENVER

Digital Image Processing
Project 2: Visual Saliency

Steven Jennings

17 April 2018

Overview

The goal of this project was to develop an “intelligent” image quality metric based on visual saliency. A secondary objective of this project was to develop a relatively unique quality metric. These goals were to be achieved through the following steps:

- Calculate Visual Saliency maps with the given input images
- Calculate unweighted quality metrics
    - Mean-Squared Error
    - Peak Signal to Noise Ratio
    - Structural Similarity Index
    - Custom Metric
- Calculate weighted quality metrics, based on saliency maps
    - Mean-Squared Error
    - Peak Signal to Noise Ratio
    - Structural Similarity Index
    - Custom Metric

Procedure

It's worth noting that I did not use the provided saliency tool. Since visual saliency is widely developed, I searched for saliency solutions written in languages that I am much more familiar with. I found a good Python saliency solution that generates, thanks to GitHub user shuuchen:

https://github.com/shuuchen/saliency

So this project runs in two parts: one using the Python 2 interpreter to generate the saliency maps, and one to calculate image quality metrics. The results turned out fairly well, considering I took a different approach.

Both programs are set up to compensate one another, so nothing extra needs to be done in order to make both programs work together. Just run the Python 2 program with input images in the /in/ folder, and when the saliency maps are generated (in the /sal/ folder), run the Matlab script to calculate the image quality metrics.

Every quality metric was calculated with conventional formulae, except for my own metric of course. It's also worth noting that for each "weighted" image, I essentially multiplied each value in the images by their corresponding positions in the saliency map to create "weighted images". These weighted images are what were used for the metric calculations.

My Custom Quality Metric

I implemented a semi-unique quality metric that has no official designation. I just call it "my custom metric". My metric assumes the two input images are already normalized between 0 and 1. It compares each pixel value between the two images. If the difference between the pixel values does not exceed the given tolerance value (must be supplied to the function), then the quality score will increase.

For the entire comparison, this score can increase to a potential maximum of 1.0 (or 100%). Comparing two identical images will yield 100%, whereas two completely different images will score much lower, possibly 0%. Depending on the tolerance level given, any score can be theoretically achieved, even if the two images are significantly different.

The possible tolerance levels range from 0 to 1, where 0 is no tolerance, and 1 is infinite tolerance, which both guarantee scores of 0% and 100%, respectively.

For my script, I use a tolerance value of 0.01, which means that the maximum difference between pixel values must not exceed 0.01, or else the score for that particular pixel location will be lost. It's kind of like losing points on an exam.

I admit my custom quality metric is not very sophisticated, but it works pretty well considering it is very simple.

$$CustomMetric = \sum_{i=1}^{r} \sum_{j=1}^{c} \begin{cases} \frac{1}{r*c} \rightarrow |a_{i,j} - b_{i,j}| > t \\ 0 \rightarrow otherwise \end{cases}$$

Where
r = the number of rows in the matrix
c = the number of columns in the matrix
a = the test image/matrix
b = the original image/matrix
t = the tolerance

Assuming each pixel value in images a and b, as well as t, are normalized.

Analysis

Based on the data you can see in the “./out/data.txt” file, it’s clear that using saliency maps to calculate image quality heavily increases the detected “quality” of an image mainly due to the fact that people do not typically consider the background of an image to be as important as the focus of the image (or the foreground). In just about every scenario, a weighted image quality metric shows the quality of the given image to be much more accurate than the unweighted results.

Here is a sample set of output data I have collected:

Image #0
Unweighted MSE: 0.003342
Unweighted PSNR: 24.726066
Unweighted SSIM: 0.999982
Unweighted Custom Metric: 0.099717
Weighted MSE: 0.000188
Weighted PSNR: 33.095516
Weighted SSIM: 0.999999
Weighted Custom Metric: 0.674141

Image #1
Unweighted MSE: 0.003332
Unweighted PSNR: 24.772408
Unweighted SSIM: 0.999982
Unweighted Custom Metric: 0.100811
Weighted MSE: 0.000042
Weighted PSNR: 34.774830
Weighted SSIM: 1.000000
Weighted Custom Metric: 0.886387

Image #2
Unweighted MSE: 0.003334
Unweighted PSNR: 24.632483
Unweighted SSIM: 0.999982
Unweighted Custom Metric: 0.100042
Weighted MSE: 0.000013

Weighted PSNR: 33.543784
Weighted SSIM: 1.000000
Weighted Custom Metric: 0.988379

Image #3
Unweighted MSE: 0.003339
Unweighted PSNR: 24.764409
Unweighted SSIM: 0.999982
Unweighted Custom Metric: 0.100667
Weighted MSE: 0.000111
Weighted PSNR: 32.963030
Weighted SSIM: 0.999999
Weighted Custom Metric: 0.715768

Image #4
Unweighted MSE: 0.003333
Unweighted PSNR: 24.771685
Unweighted SSIM: 0.999982
Unweighted Custom Metric: 0.100091
Weighted MSE: 0.000002
Weighted PSNR: 31.801346
Weighted SSIM: 1.000000
Weighted Custom Metric: 1.000000

Conclusion

It's easy to see that using saliency to calculate weighted image quality yields a better result over unweighted images. This project definitely felt like more of an assignment, but the end of the semester is coming, and I think this course as a whole was a good experience. I look forward to finishing the last project.