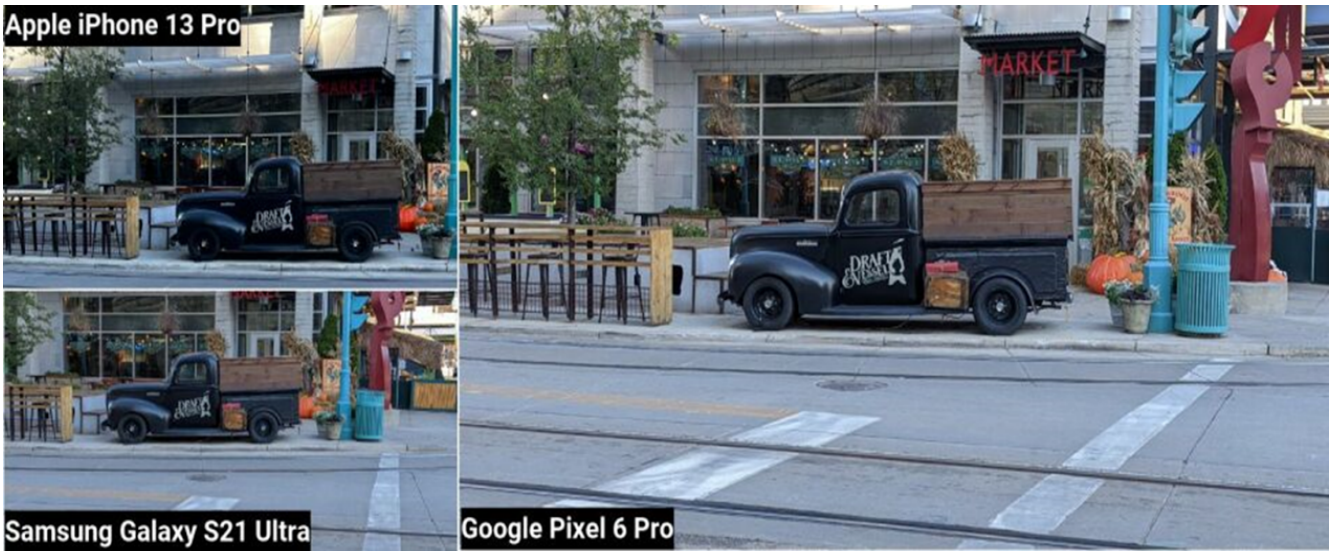
Learning Google Pixel’s Photo Finishing HDR tonemapping Algorithm - Haley So

Team members: Haley So

Introduction:

In any imaging pipeline, there will be some amount of post-processing done to take the raw image to the photo-finished image for the user. From the raw image, one needs to demosaic, motion deblur, deal with vignetting, adjust exposure, tonemap, color balance, adjust saturation, adjust texture, and perform RGB toning, among other things. In the past, many parts of this could often be done by a skilled photographer. Today, however, where everyone has a camera in their pocket, the camera turns into a tool for capturing memories. Ideally, what one captures on their phone wouldn’t require a lot of fiddling with different sliders to make the image beautiful and representative of what they saw.

In the world of mobile photography, each company has their own tailored algorithm that takes the 10-12 bit raw image down to a standard 8 bit image for the display to be able to show. In this way, they each have their own ‘tastes’ for what a nice image looks like.



(courtesy of Steven Winkelman)

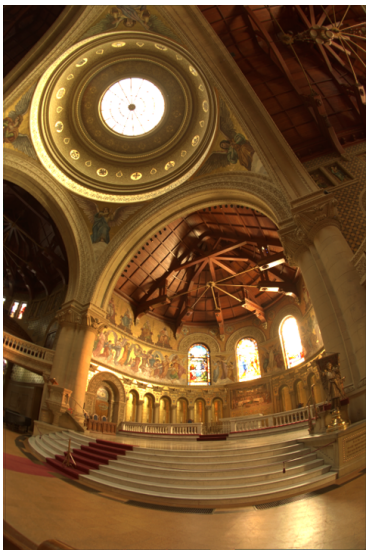
We can see for example, Apple has higher contrast and tends to have a darker tone, whereas google pixel is much brighter. Samsung seems to have more muted colors in this example. But these pipelines are not easily accessible to us. In this project, we will see if we can learn the Google Pixel’s finishing pipeline’s high dynamic range (HDR) tonemapping operator from small patches. (Quick note: Because Google doesn’t have the the functionality to tap out the image right before tonemapping and right after tonemapping, what we will try to learn is actually a little more complicated than just the tonemapping operator.)

Motivation:

Learning the HDR tonemapping operator would not only allow people to quickly test the different looks or tastes of images of each company or even do style transfer, but also this can be used for the live viewfinder. When one opens their camera app on their phone, the image they see before they press the shutter is often not the same image that results from the imaging pipeline. Ideally they would be very similar, so a small and hence fast network that predicts how the image would actually turn out would be useful in bridging the gap.

Background:

When images are captured in 10, 12, (anything above 8 bits), to be able to display them on standard displays, we need to compress the signal to 8 bits. This compression is called tonemapping. One could simply clamp the image for displaying, but that leads to loss of information in the brights.



(Clamped to 255.)

The simplest tonemapping function is to take the log of the signal since it roughly follows the brightness perception of the human eye, the curve that shows how the luminance gets mapped to a perceived brightness. In this way, more precision is given to the darker areas. This simple global tonemapping operator however isn't always able to produce an image similar to what a human actually saw since our eyes have incredible dynamic range. So there's a whole line of research going into how to create good tonemapping operators.

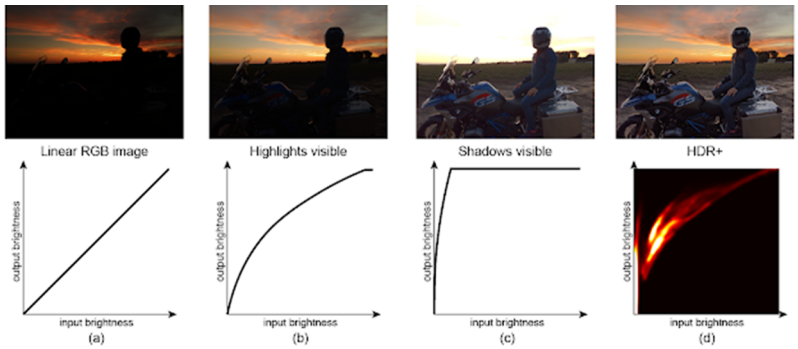
Algorithms fall under 2 main categories: global and local tone mapping operators. Global tonemapping operators are simple functions that apply pixel-wise, and local tonemapping operators try to maintain local contrast by looking at the region around the pixel to decide the output value. Cerda-Company [1] summarizes some of the best tonemapping operators.

TMO	Global/Local	HVS	Luminance	Color
Ashikhmin	L	✓	✓	
Drago	G		✓	
Durand	L		✓	✓
Fattal	L		✓	
Ferradans	H	✓	✓	✓
Ferwerda	G	✓	✓	✓
iCAM06	L	✓	✓	✓
KimKautz	G		✓	
Krawczyk	L		✓	
Li	L		✓	
Mertens	-			
Meylan	L	✓	✓	✓
Otazu	L	✓	✓	
Reinhard	G		✓	
Reinhard-Devlin	G		✓	

Table 1: Summary of used TMO's characteristics. Second column shows whether the TMO is global (G), local (L) or hybrid (H). Third column shows whether it is inspired by the Human Visual System, and following columns show whether it processes luminace and color information.

On the left are two examples of local tonemapping operators (Fattal, Durand) and the one on the right is a global operator (Reinhard).

Another way to look at this is by looking at how the input brightness is mapped to the output brightness. Google HDR+ illustrates this well. We can see with simple mappings, either the details in the darks or the brights are lost. In their pipeline, their tonemapping function is much more complicated.

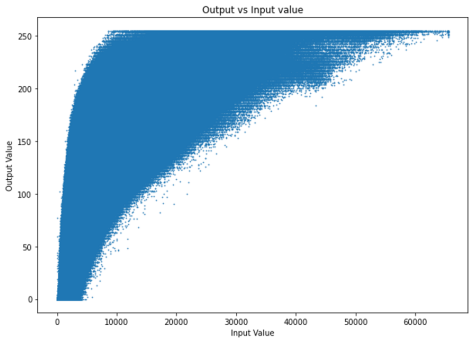


Method:

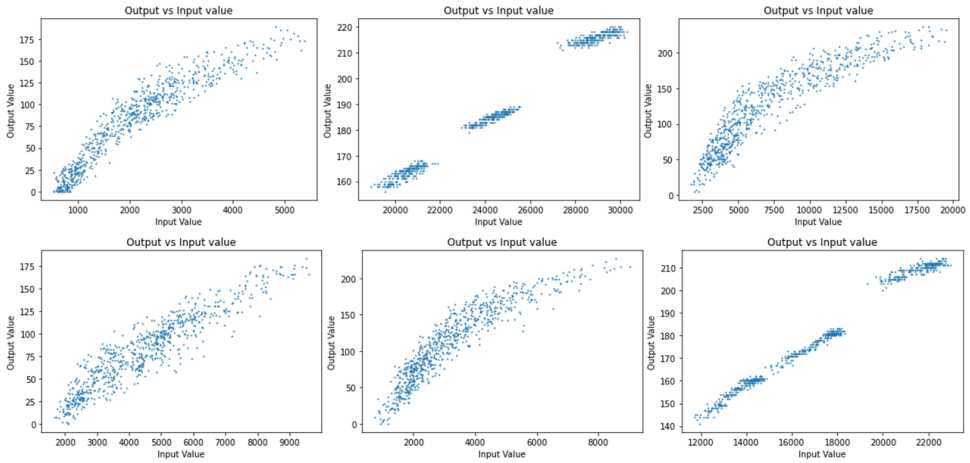
In this project, we will be trying to learn the tonemapping/photo-finishing function from a dataset of Google Pixel raw images and the corresponding photo-finished images. The dataset is the HDR+ Dataset that contains 153 image pairs. We reserve 10% of the images (15) to be our test dataset which the networks will never see.

While the tonemapping function we see in (d) cannot be described easily, the idea is that looking at a linear patch, the function for google's tonemapping will be linear or at least well described.

Here is a plot of the input 14bit values of the raw image and the output value after the google photo finishing pipeline. As we can see, it is not a well defined function.



But, if we look at patches of the image, we can see that locally, the function is somewhat discernable. Here are a few plots from randomly sampled 16x16 patches.



Design Choices:

In this experiment, we used a small 5-layer CNN and trained on small patches. The size of the network was chosen to be small for live viewfinder which requires fast updates. The patch sizes we tried were 16x16. A common network architecture for image-to-image translations is the U-Net, but this requires the training and the test images to be of the same size. Here, not only are all the images differently sized, but we are training on small patches, so to run on the test images, we would have had to split up the image into patches and there is no guarantee that the images patches flow from one to the other seamlessly. Therefore, we opt for the classic CNN since we can train on small patches and run on the full image.

In this project, we experimented with using different metrics as the loss function to optimize over to try to get perceptually similar images as the google pixel photo-finishing pipeline outputs.

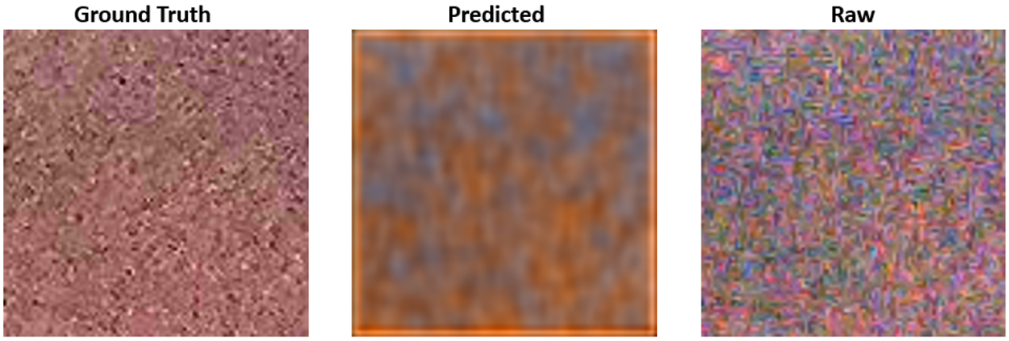
These metrics include:

- minimizing the L1 norm, which is the go-to loss for image restoration
- minimizing L2 norm, another common loss that minimizes the mean-squared error
- maximizing SSim, structural similarity that uses the image's luminance to calculate a score of similarity
- minimizing LPIPS, a learned perceptual image patch similarity that compares the activations of two images from a pre-trained network, often a VGG net
- Minimizing deltaE or the L2 loss in Lab colorspace
- Minimizing L1 loss in Lab colorspace

Results:

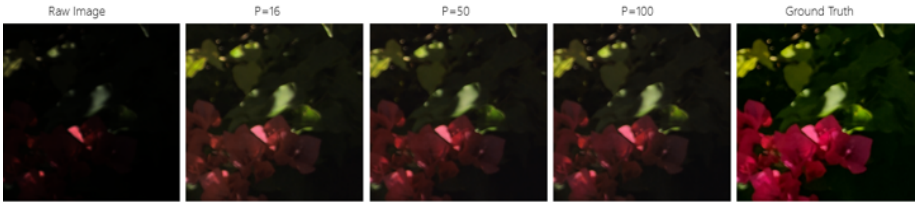
Some insights right away was that the overall colors got blended more and more when using L2 or MSE loss. It seems to settle in an averaging to minimize losses. This is especially noticeable with very vibrant images. This is a 700x700 crop. Ground truth is the photo-finished output from the dataset. Predicted is the output, and Raw is the raw input.

Likewise, taking a look at the little patches themselves, we can see that the details are also spatially blurred a bit.

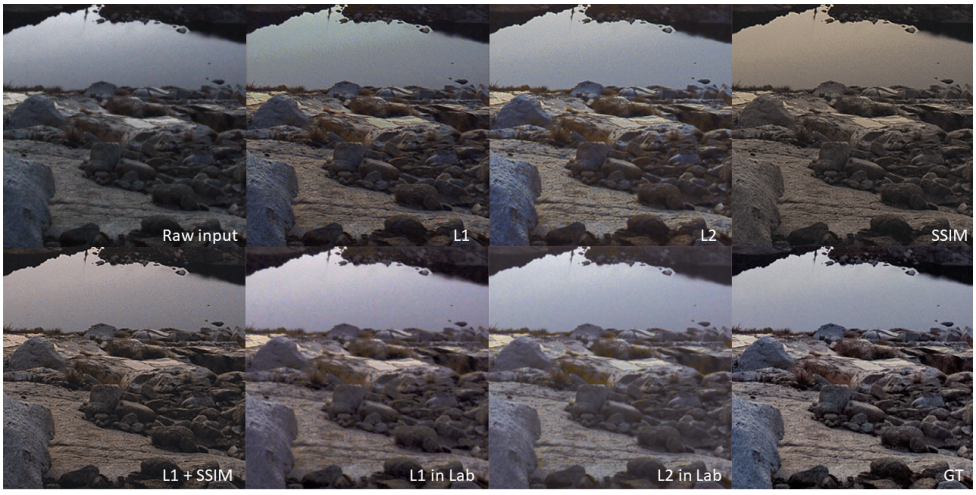


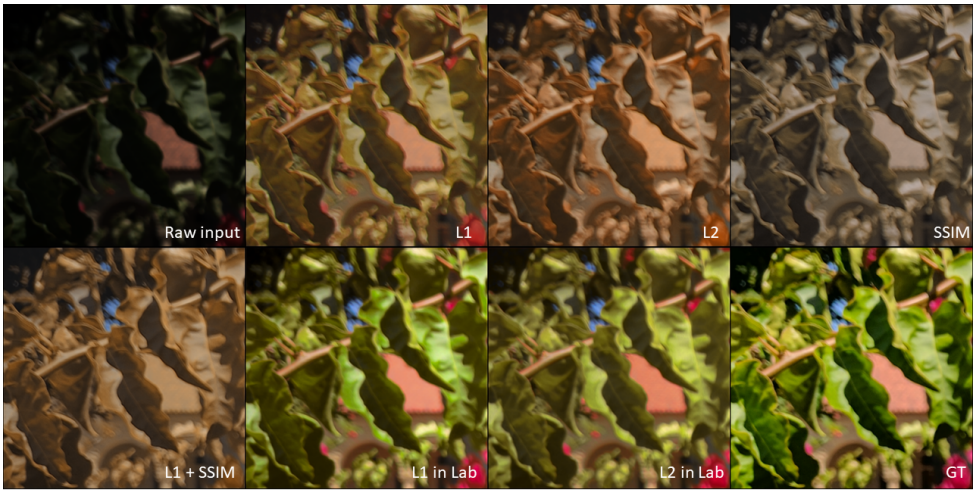
Using just SSIM as the loss produces rather dulled out images. In some cases, the images became completely grayscale. However, the high frequency details were well captured. This makes sense since SSIM only uses the luminance channel.

This is a look into the effects of varying patch size (P). Here all are the same architecture trained with L1 + ssim loss.



All were trained for 600 epochs with learning rates of 0.0001 through 0.01to try to get the best outputs. I also tried a 5 layer ResNet, but the architecture didn't affect performance drastically.





Conclusion and Lessons Learned:

This small network is able to learn a rather good representation of the tonemapping operator. Using Lab space definitely helped. The output looks reasonable, but they are not exactly the same as the Google finishing pipeline outputs.

If I were to do the project again, I would try to learn the HDR tonemapping from images collected right before and right after the tonemapping operation was applied. Learning the whole finishing pipeline with a small network seems like it's not expressive enough to represent all the intricacies. State-of-the-art methods like [Tseng] use networks to represent each module in the pipeline. I would also use a much larger dataset. Google and others have much larger datasets than just a hundred.

Code: https://github.com/haleyso/HDR_pixel

References:

[1]: Xim Cerda-Company, C. Alejandro Parraga, and Xavier Otazu, "Which tone-mapping operator is the best? A comparative study of perceptual quality," J. Opt. Soc. Am. A 35, 626-638 (2018)

[2]: E. Tseng, Y. Zhang, L. Jebe, C. Zhang, Z. Xia, Y. Fan, F. Heide, and J. Chen. “Neural Photo-Finishing.” Siggraph Asia (2022).