The Effects of L1 Piece-wise Flattening on Image Compression

Corey Wingo
The University Of Texas at Dallas
csw170030@utdallas.edu

Sravan Kumar Guduru
The University Of Texas at Dallas
sxg210117@utdallas.edu

Kartikey Gupta
The University Of Texas at Dallas
kxg220013@utdallas.edu

May 9, 2023

Abstract

Piece-wise Image flattening can remove high and low-level noise from an image while preserving it's structure. Such an effect is useful for image compression without reducing the dimensions of the image while maintaining the *essence* of the image. Our team measures the impact that image flattening can have on various image formats and their respective compression methods.

1 Introduction

Typical image smoothing techniques remove high-level noise from an image, e.g., the bilateral filter. This filter operates locally in the sense that it filters every pixel individually based on it's neighborhood which has the effect of blurring an image.

L1 Piece-wise Flattening [2] operates globally by constructing a global matrix containing pixel similarity between each pixel and it's neighborhood. This allows the method to remove both high-level and low-level noise from an image. In other words, image flattening converges two pixels that have similar colors and no edges between them.

Our team had the idea of measuring the impact of removing such noise from an image on compression. It was based on intuition that less noise promotes better compression. Another potential improvement to compression was the notion that if color gradients were removed from an image and replaced with a converged color then image compression methods would be able to make better approximations.

We first tested our theory by creating two 500x500px PNG images of differing complexity and measuring their sizes, as seen in Figure 1. Both images were exported with maximum compression and no metadata. The left image contains only 1 color while the second image contains a gradient of that same color.

This confirms that our idea might work. Now all that is left is to implement L1 piece-wise flattening and measure the results.

Figure 1: A size comparison between a simple and gradient image

2 Related Work

The bilateral filter is another method of removing noise from an image while preserving some structure and edges. However, It doesn't achieve the effect of converging gradients into single colors and instead blurs them, as seen in Figure 2. Note that the ridge at the top of the image, an edge feature, is not preserved in the flattened image. This is because our team could not implement edge-preserving for L1 piece-wise flattening. We hope to remedy that in the future.

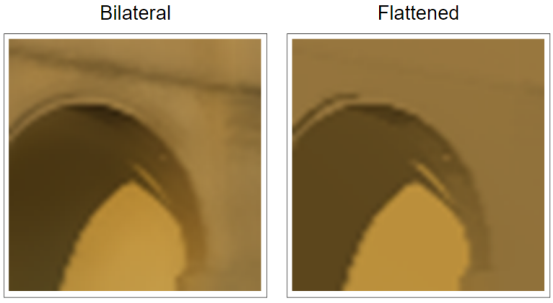

Figure 2: An image comparison between an image processed with a bilateral filter and an image processed with L1 flattening

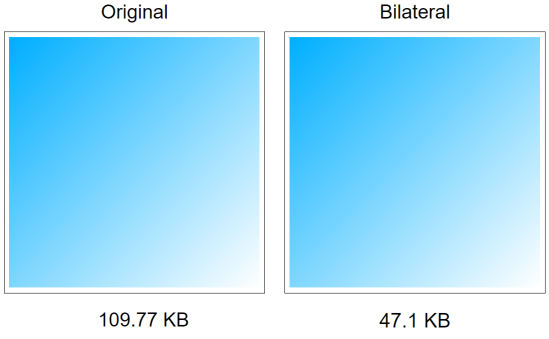

Figure 3: A size comparison between an image and it having been processed with a bilateral filter

Still, processing an image with a bilateral filter, as seen in Figure 3, results in a very similar image while reducing the size substantially. It should also be considered that the bilateral filter takes orders of magnitude less time to compute than L1 piece-wise flattening and uses substantially less VRAM. This makes it possible to filter large images, whereas L1 flattening is limited to smaller images.

3 Method

The essence of the L1 piece-wise flattening algorithm is distilled into two rules:

1. If two neighboring pixels have similar colors, pull those colors closer together in the transformed image.

2. If a color discontinuity exists between two neighboring pixels, keep those colors apart in the transformed image.

A color discontinuity is an abrupt change in color which can be controlled by a parameter λ to increase the "flattening" of the image.

3.1 Constructing the Affinity Matrix

To implement these two rules, a global matrix M is constructed by computing the affinities between two neighboring pixels. That is, for every pixel p_i and a pixel p_j within p_i's neighborhood assign M_{si} to w_{ij}, the affinity between p_i and p_j, where s is the index of the number of pixels pairs, k, within the image. If there are n pixels within an image and the size of the neighborhood is h, then $k = nh$ and $0 \leq s \leq k$.

$$M_{si} = w_{ij}$$

$$M_{sj} = -w_{ij}$$

The affinity of two pixels is the function of the L1 norm of the pixels, where σ is the lightness weight.

$$w_{ij} = e^{-\sigma \|p_i - p_j\|}$$

Once M has been constructed, we construct an identity matrix L comprised of the M matrix and it will serve as the global matrix that we wish to

optimize using the Split-Bregman regularization optimization algorithm.

$$L_{3k\times 3n} = \begin{pmatrix} M & & \\ & M & \\ & & M \end{pmatrix}$$

3.2 L1 Solver

The intermediate variables for the Split-Bregman algorithm is the flattened RGB values of the image. That is, $z = (z_r^T, z_g^T, z_b^T)^T$ where z_x is a vector contaning all of the x values of the image. Essentially z is a 1-D flattened vector of the 2-D matrix image.

The image flattening procedure in Algorithm 1 is given an input image as a 1-D flattened vector, the threshold stop condition ϵ, the maximum number of iterations R, the image approximation term θ, the regularization term weight λ, and outputs a flattened image as a 1-D flattened vector. The flatness of the image and the vector have no correlation.

The algorithm begins by initializing the intermediate variables and then iteratively solving their convex optimizations and placing it within the transformed image z. The algorithm typically converges to a solution within 2-3 iterations. Further iterations result in the transformed image reverting back into the original image.

Algorithm 1 Split-Bregman for Piece-wise Image Flattening

procedure IMAGEFLATTEN(z^{in}, ϵ, R, θ, λ)
  $z^0 = z^{in}$
  $d^0, b^0 = 0$
  while $i < R \wedge \|z^i - z^{i-1}\| > \epsilon$ **do**
    $v = \theta z^{in} + \lambda L^T(d^i - b^i)$
    $z^{i+1} = solve(\theta I + \lambda L^T L, v)$
    $d^{i+1} = shrink(Lz^{i+1} + b^i, 1/\lambda)$
    $b^{i+1} = b^i + Lz^{i+1} - d^{i+1}$
    $i = i + 1$
  end while
  return z^{i+1}
procedure SHRINK(y, λ)
  return $\frac{y}{\|y\|} max(\|y\| - \lambda, 0)$

3.3 Implementation

It should be noted that the L matrix is abnormally large since L has the shape $3k \times 3n$, where n is the number of pixels within an image and $k = nh$ is the number of pixel pairs within an image with a neighborhood size. For a 100×100 image and using a neighborhood size $h = 5$, L roughly has the shape $250,000 \times 10,000$. That is a total of 2.5 billion cells, or 20GB worth of data when using 64-bit floats. Hence, sparse matrices should be used for the storage of L.

Our team used Python and CuPy, a CUDA-based GPU acceleration library with support for sparse matrices. Our implementation is roughly 400 LoC [1]. Flattening a 150×150 image uses approximately 7GB of VRAM and takes roughly several minutes. The computation time could be significantly reduced to several seconds by optimizing the affinity matrix construction to use matrix-agnostic code.

4 Experiments

All of the experiments was performed on a dataset of 5 PNG images found online, which were converted to JPG and WebP images, for a total of 15 images. The images were downscaled to between 100×100 and 200×200 so that our team's GPU could handle processing them.

4.1 Hardware

The hardware used for these experiments are listed below.

1. CPU: Ryzen 5 5600X 6-core
2. GPU: Nvidia RTX 3070 8GB
3. RAM: 32GB DDR4-3600

4.2 Parameters

The following parameters are used to achieve the results: $R = 4$, $h = 5$, $\alpha = 20.0$, $\epsilon = 0.01$, $\theta = 50.0$, $\lambda = 128.0$, and $\sigma = 0.9$. Increasing the λ parameter increases the flattening effect and decreases the size of the resulting image.

4.3 Evaluation Metrics

The only metric used is the size reduction between the original image and the flattened image. The metric is meaningless when the original and flattened images are of different image formats. Each image format was configured to use maximum compression. Particularly, Lossy WebP is used as opposed to lossless.

4.4 Results

Firstly, the flattened images retain the essential structure of the images, as seen in Figure 4 and Figure 5. Meanwhile, they reduce the size of the image substantially depending on the image format used. Figure 6 shows the size ratio between the original image and flattened image over the dataset for various image formats.

Over all image formats image flattening reduces image size to 48% of the original without reducing dimensions and preserving structures. The WebP format has the strongest results of reducing image size to 33% of the original and JPEG has the weakest results with 66%.

Figure 4: A size comparison between the image car.png and car_flattened.png

Our team believes that WebP benefits the most from flattening an image because of Adaptive Block Quantization (ABQ). While Prediction Coding is the main breadwinner for WebP against JPEG for normal images, flattened images bring its other features to light. ABQ segments an image into groups of blocks with similar features and tuns each block for efficient compression so that bits are used more effectively. Image flattening transforms an image so that areas of the image converge into one color. This greatly boosts the effectiveness of ABQ.

Figure 5: A size comparison between the image field.webp and field_flattened.webp

4.5 Limitations

Notice that while the flattened image in Figure 5 has a significantly decreased size compared to the original and the flattened image in Figure 4 has a slightly decreased size compare to it's original. The image format different is irrelevant, the same result is observed when **car.webp** is used.

This is due to a limitation of image flattening: dense salient features cannot be flattened. In other words, if an image has too much color discontinuity within a small space where each of the colors are orthogonal then image flattening will result in a less

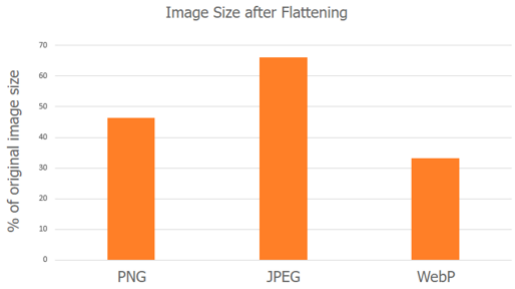

Figure 6: Image compression of flattened images for various image formats

influenced transformed image. The less an image is influenced by flattening, the more it is alike to it's original image, resulting in less compression.

5 Conclusion

Our team has flattened 15 images across the 3 most commonly used image formats and found that on average images were reduced to 48% of it's original size and an average of 33% for WebP images. These results still retain the essential structure of the image. However, we found that L1 piece-wise flattening is an expensive process in terms of memory allocation and computation time.

References

[1] Edge-preserving image smoothing repository. https://github.com/CS6384-S23-Group-Project/EPIS/blob/main/epis.py, 2023. 3

[2] Sai Bi, Xiaoguang Han, and Yizhou Yu. An l1 image transform for edge-preserving smoothing and scene-level intrinsic decomposition. *ACM Trans. Graph.*, 34(4), jul 2015. 1