INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 04, Issue 01, January 2024, pp : 473-482

e-ISSN : 2583-1062

Impact Factor : 5.725

IMAGE COMPRESSION FOR MOBILE DEVICES

Sahana S N[1], Priya Khanapure[2], Dr. Mohammad Rafi[3]

[1,2,3]Computer Science & Engineering, University BDT College of Engineering. Hadadi Road,
Davanagere, Karnataka, India - 577004

ABSTRACT

Efficient image compression is paramount for optimizing the performance of mobile devices by reducing file sizes, enhancing loading times, and conserving storage and data resources. This abstract explores key strategies for image compression, including lossy and lossless compression methods, appropriate file formats such as JPEG and PNG, resolution adjustments, and progressive loading techniques. It also emphasizes the importance of adaptive image loading, modern formats like WebP, and the use of image optimization tools and APIs. By considering user preferences, caching mechanisms, and automated compression within mobile apps, developers can achieve a harmonious balance between image quality and efficient resource utilization, ultimately improving the mobile user experience.

Keywords— Image compression; machine learning; JPEG; WebP; HEIC;

1. INTRODUCTION

Image Compression is an approach of reducing the storage space for a graphic file in bytes without disturbing the original image quality to obnoxious level. This reduction in storage size enables storing of more images in a minimal memory space. It also drops the time accommodation for transmission of the image over the Internet and also its downloading time.

There are many different approaches to compressing image files in order to reduce their storage size. From the Internet, the compression can be categorised into two types namely Lossy and Lossless Compression based on there graphic file resizing process where the primary intension is to maintain image quality without any intact. Nasir Ahmed, T. Natarajan, and K.R. Rao introduced a type of compression called Lossy in 1974. It is the most commonly used type of compression. The Lossy compression technique is based on transform coding, especially Discrete Cosine Transform (DCT)

which compress multimedia data (audio, video & images). Lossless compression invented in 1838 and its algorithm specified in 1996. It is a class of data compression approach allowing perfect reconstruction of the original data without loss of information.

Mobile devices are often used for taking pictures. People today expect a single photo to be automatically uploaded to different cloud services, compressed, edited, and shared on social media apps. This puts a lot of strain on cellular networks and storage space. High-resolution sensors in mobile cameras add to this pressure. Social media and cloud storage apps often show pictures at different resolutions, so the uploaded picture has to be compressed many times on the server. This is why it's important to minimize the size of image files without sacrificing quality. This is also important because it keeps energy consumption low, which is crucial for mobile and cloud services.

4 1, January 2024 January

2. RELATED WORK

As of now, the most popular open-source JPEG codecs such as libjpeg do not offer any direct options to manage the desired output file size or image quality [1]. When adjusting the quantization table, users are asked to choose a quality factor between 0 and 100. This factor will be used to scale the table. Sometime, the user is asked to give technical parameters that dictate how the algorithm behaves instead, rather than specifying their actual needs in a meaningful or measurable way. Have you ever thought about the impact of compression on images with intricate details and complex visual elements? It's worth considering that such images may not compress as effectively, leading to a lower quality output. When it comes to finding the best way to compress images, some people may use exhaustive searches or binary methods. However, these approaches can take a long time, especially when dealing with multiple images. Therefore, it is important to find more efficient methods that can help achieve optimal compression results without taking up too much time to achieve optimal compression results quickly and efficiently, there are methods available that also minimize energy consumption. It's essential to prioritize energy use and output quality [2].

Google's WebP compressor offers options to target a specific image quality, as specified in PSNR, or file size. However, this comes with the downside of longer compression time. The WebP manual explains that this extra time is due to "several passes of partial encoding" needed to meet the given constraints[3]. In contrast, our approach requires no additional recompressions.

Said and Pearlman [1].have developed the S+P Transform-based compression algorithm which can be used for both lossless and lossy compression. This transform only needs integer addition and bit shift operations, and with careful scaling and truncation, it keeps the number of bits used to represent the transformed image low. The S transformation truncation removes redundancy from the least significant bit. Instead of performing the prediction as a post-transform operation, Said and Pearlman developed a scheme to predict high-frequency components from a combination of computed low and high-frequency components. This transformation produces superior outcomes when compared to JPEG LS. Calderbank et al. [4] developed the wavelet transform which was created to help compress images. It's a popular method for compressing images. In wavelet transformation-based compression, the input sequence is split into even and odd samples. It is followed by a repeated alternate application of lifting and dual lifting steps. When lifting steps are applied, the odd samples are filtered and the results are then subtracted from the even samples. Lifting makes it possible to obtain an integer transform simply by using truncation without sacrificing invertibility. The dual lifting step does the opposite, it involves applying a filter to the even samples and subtracting the result from the odd samples. After N such applications of dual and primal lifting operations, the even and odd samples produce the low and high-pass coefficients respectively.

The inverse transform can be obtained by simply reversing the operation and flipping the signs. The lifting scheme implementation of the wavelet transform has numerous advantages over the traditional wavelet transform. Wu and Memon [3] developed CALIC (Context-based Adaptive Lossless Image Coder), a lossy+residual approach. The algorithm uses a gradient-based non-linear prediction to get a lossy image and a residual image in the first stage. In the second stage, it uses arithmetic coding to encode the residuals based on the conditional probability of the symbols in different contexts. CALIC also has a mechanism to automatically trigger a binary mode which is used to code either uniform or binary sub-images or both. CALIC uses a nonlinear predictor that adapts itself to the image gradient near the predicted pixel. This predictor enhances the accuracy of traditional DPCM predictors, particularly in regions with abrupt transitions.

The JPEG standard defines coding schemes for image compression using Huffman and arithmetic encoding techniques [4]. In JPEG's lossless mode, a predictive coding technique is used to compress the image without losing any information. The algorithm is easy to implement in hardware due to its simplicity. There are various prediction methods available, but the most commonly used one is based on the three nearest neighbours (upper, left, and upper-left). The prediction is formed using previously encoded pixels in the current row of the image, and the prediction error is then subjected to entropy coding. In LOCO-I [5], context definition is based on the local image gradients that allow one to capture the level of activity surrounding a pixel.

Lempel and Ziv and Welch [6] proposed the LZW coding, which uses an adaptive coding method that does not require the data that is to be compressed to be available at the start. Rather, their technique generates codes as it examines the source file from beginning to end. In Huffman coding, each symbol of the uncompressed data is replaced by a code [7]. The symbol codes are of variable length and symbols that occur more frequently in the uncompressed data have codes of smaller length that symbols that occur less frequently.

Several papers have studied the issue of predicting the effects of compressing images that have already been compressed. One such study is by Chandra et al. They focused on JPEG image transcoding [4] and developed a method that generates a statistical table to determine if recompressing a particular image is worthwhile in terms of reducing its size while keeping an acceptable level of quality. To navigate this table, they used parameters of the compressed JPEG image obtained from manipulating DCT coefficients. Our feature design was inspired by the idea of using coefficients of image spectral transformations.

Coulombe and Pigeon proposed a predictor for JPEG file size and image complexity in a series of papers, along with similar techniques presented in [14, 5, 15, 16, 17, 11, 18]. The key idea behind these works is to use the compression ratio of an already encoded image as input to nearest neighbour and table-based predictors. This approach assumes that file size and quality increase with the quality factor, and that neighbour values of quality factor correspond to similar compression ratio and quality level.

Our approach differs in that it does not require the input image to be compressed and instead works on uncompressed images. In recent news, our experimental results have shown a significantly lower error in our model compared to Coulombe and Pigeon's predictor, despite using less input information. This demonstrates the effectiveness of our approach and its potential to improve image compression and quality in the future.

Some papers discuss how to extract image features for various purposes. Some of these features are complex and mostly related to computer vision problems. Others focus on low-level features, such as discrete wavelet transform coefficients transform coefficients are used by Hanghang et al. [5] to indicate the amount of blur in the images and by Viola et al. [6] to detect faces. These coefficients are used to determine the amount of blur in images or to detect faces. However, we don't believe that features designed for vision problems can be applied to our issue.

3. ALGORITHMS USED

There are various algorithms commonly used for image compression on mobile devices. Some of the popular ones are:

The JPEG algorithm is widely used for compressing photographic images by selectively discarding less significant image data.

2. PNG (Portable Network Graphics) uses lossless compression to retain all original image data without quality loss. It's suitable for sharp edges, text, and graphics.

3. WebP is a modern image format developed by Google that provides both lossless and lossy compression. It offers smaller file sizes compared to JPEG and PNG while maintaining good image quality.

4. HEIC is a newer image format introduced by Apple that uses advanced compression techniques to achieve high-quality images with smaller file sizes. However, it may not be widely supported by older mobile devices.

Different image compression algorithms are used on mobile devices. The algorithm chosen depends on factors such as the image type, preferred image quality, and compatibility with the device or platform.

4. METHODOLOGY

Digital image compression [9] is a topic extensively studied in multimedia processing. It is used to reduce the size of digital images by taking advantage of the correlations that exist between neighboring pixels. The compression is achieved by removing these redundancies before compression, leading to a more effective compression.

An image compression system consists of a compressor and a decompressor. The compressor has two parts: preprocessing and encoding. The decompressor has decoding and postprocessing stages. A diagram shows the compression process in Figure 1**.** To prepare the image for the encoding process, the preprocessing stage performs any number of operations that are application-specific. After decoding the compressed file, the postprocessing stage can be utilized to eliminate any undesirable artifacts introduced during compression. The compressor can be broken down into stages

as in Figure 2. The first stage in preprocessing is data reduction. The image data can be reduced by grey level and/or spatial quantization, or can undergo any desired image improvement (for example, noise removal). In the process of image compression, the second step is the mapping process. This process maps the original data of the image into a different mathematical space. This is done to make it easier to compress the data. After the mapping process, the next step is the quantization stage. This stage converts the continuous data from the mapping stage into discrete form.

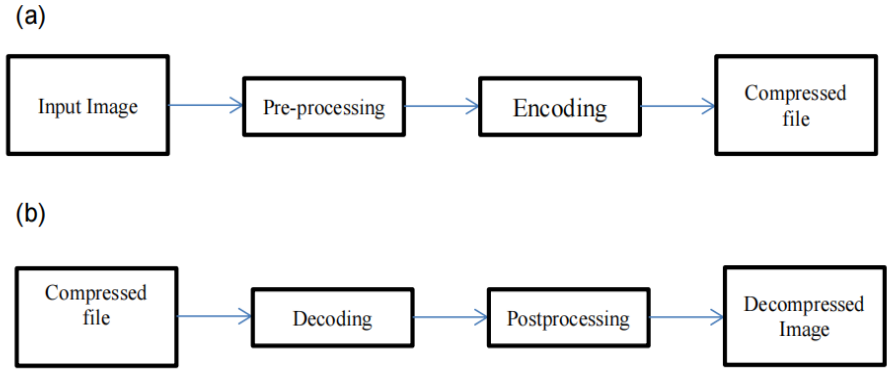


Figure 1: compression system model.
(a) Compression. (b) Decompression.

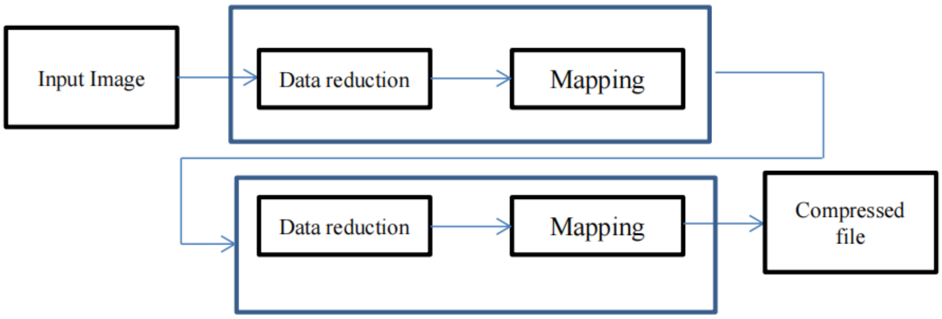


Figure 2: compression system model.

Encoding data involves several stages before the final stage of coding. The compression process can have several stages. When it's time to decode the compressed file, the process is divided into two stages. The first stage is about reversing the original coding by mapping the codes back to the original quantized values. In the second stage, these values are processed again by a stage that performs an inverse mapping to reverse the original mapping process. After that, the image can be postprocessed to improve the appearance of the final image. The decompressor is divided into stages shown in Figure 3.

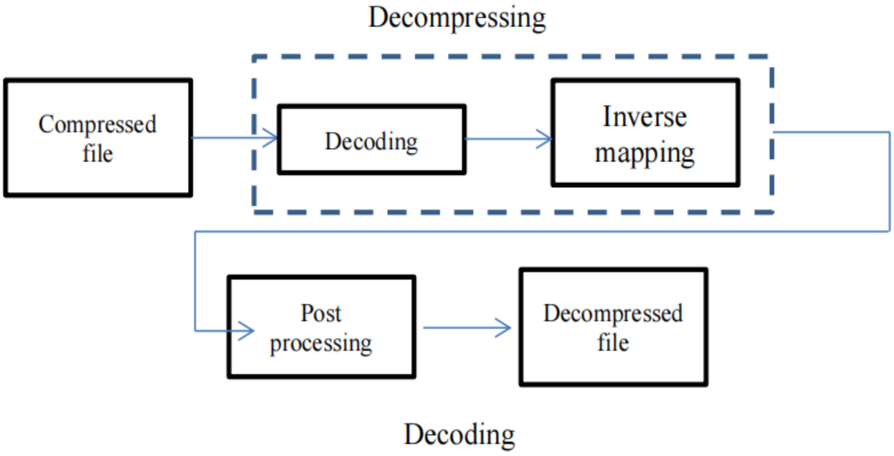


Figure 3: The Decompressor.

Developing [10] a compression algorithm requires customizing it to the specific application. Before compression can begin, the image must undergo preprocessing, which involves removing noise, enhancing or quantizing the image. The aim of preprocessing is to prepare the image by removing any unnecessary information based on the application's requirements.

Mapping is crucial because image data typically has high correlation. If the value of one pixel is known, it is likely that the adjacent pixel value is similar. To remove this type of data redundancy, a mapping equation must be found. However, the quantization process is not reversible, so it may result in some information loss. Moreover, since it is not reversible, the inverse process is not included in the decompression model.

The coding stage is critical in image compression algorithms. The coder establishes a one-to-one mapping, where each input maps to a unique output. This stage is reversible. The code can be an equal-length code or an unequal-length code with variable-length codewords. In data compression, an unequal-length code is often the most efficient, but it requires more overhead during coding and decoding.

The file that is created as a result of the compression process is known as the compressed file. This file is then used to reconstruct the original image, resulting in the decompressed image. The compression ratio is the ratio of the original image size to the size of the compressed image. The compression ratio [10] is calculated using the following equation:

$$Compression\ Ratio = \frac{Uncompressed\quad file\quad size\,(bytes)}{Compressed\quad file\quad size\ \ (bytes)} \qquad (1)$$

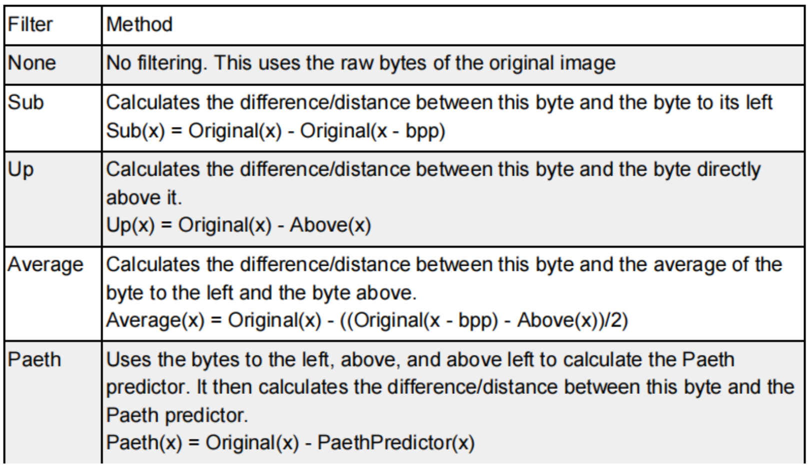
Another image compression terminology is the number of bits per pixel. For N × N image, this is determined as follows:

$$Bits\ Per\ Pixel = \frac{Number\quad of\quad bits}{Number\quad of\quad Pixels} \qquad (2)$$

A. PNG Compression

One form of lossless compression [11] for images is PNG compression. PNG, or Portable Network Graphic, was designed to be an open source replacement of the GIF (Graphics Interchange Format). PNG is a better alternative to GIF for a couple of reasons. GIF was patented and could only display up to 256 colors. PNG, on the other hand, is more suitable for icons and logos as it doesn't lose any part of the original image while compressing. For photographs, JPEG compression is more appropriate. PNG compression consists of three steps: Filtering, LZ77 Compression, and Huffman Coding. The last two are also known as DEFLATE Compression.

PNG Compression begins with Filtering, which makes compression more efficient. There are five filtering options available, which we will explain below:

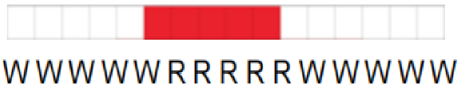


Filter	Method
None	No filtering. This uses the raw bytes of the original image
Sub	Calculates the difference/distance between this byte and the byte to its left Sub(x) = Original(x) - Original(x - bpp)
Up	Calculates the difference/distance between this byte and the byte directly above it. Up(x) = Original(x) - Above(x)
Average	Calculates the difference/distance between this byte and the average of the byte to the left and the byte above. Average(x) = Original(x) - ((Original(x - bpp) - Above(x))/2)
Paeth	Uses the bytes to the left, above, and above left to calculate the Paeth predictor. It then calculates the difference/distance between this byte and the Paeth predictor. Paeth(x) = Original(x) - PaethPredictor(x)

It is important to understand that image filtering is based on byte comparison, not pixel comparison. In an image with 8-bit coloring, each pixel is one byte, so the comparison is based on pixels. But for images with higher colors, the comparison is based on the blue component of each pixel, which is one byte, and the comparison is aided by the number of bytes per pixel (bpp). The bpp value is always a whole number and is rounded up to 1 if it is less than 1. The calculation is crucial for accurate filtering of the PaethPredictor is explained below:

```
PaethPredictor (x) {
estimate = Original (x - bpp) + Above(x) - Above (x - bpp);
distanceLeft = Absolute (estimate – Original (x - bpp));
distanceAbove = Absolute (estimate - Above(x));
distanceAboveLeft = Absolute (estimate - Above (x - bpp);
if distanceLeft is less than distanceAbove and distanceAboveLeft
return distanceLeft;
else is distanceAbove is less than distance AboveL eft return distance Above;
else  eturn distanceAboveLeft;
}
```

After the filtering is completed, PNG compression uses LZ77 Compression. LZ77, or Lempel-Ziv 1977, was created by Abraham Lempel and Jacob Ziv in 1977. LZ77 uses a sliding window technique that keeps track of a certain number of previous bytes. The larger the window the easier and more efficient the compression. Using the window, LZ77 looks for repeated sequences in the pixels. The format of the compression is the set of pixels leading up to a repetition, then designate the distance back that the repetition starts and the length of the repetition. For example: (Using W to represent a white pixel, R to represent a red pixel, D for distance, and L for length)



This line of pixels could be compressed to W [D=1, L=4] R [D=1, L=4] [D=10, L = 5].

Since the last set of 5 white pixels has already been expressed earlier in the line, the compression merely denotes the distance back to the beginning of the first set of 5 white pixels and then states that the following 5 pixels should be copied.

Another example:

W R W W R W W R W W R W

This line would be compressed to WRW[D=3,L=9] After the LZ77 compression is complete, PNG compression then uses Huffman coding to finish the job. Huffman coding is used to look at the frequency of literals, characters, numbers, or bytes, in a file and then compress the file by using a few bits to represent those literals. A literal typically requires a byte or multiple bytes, but using one to a few bits to represent them significantly reduces the file size. The literals that are most frequent in the file are represented with the fewest number of bits, while literals that are less frequent or only appear a few times in the file are represented with more bits. The literals and their frequency are used to construct a

Huffman tree. A Huffman tree is constructed by ordering the literals by their frequency in ascending order. A recursive method is then used to combine the two smallest items and then place them as one item back in the list. Once this process is completed a Huffman tree allows the proxy bits to be calculated. A Huffman tree is read by how many times you move left or right from the nodes, starting from the head node, to get to the desired literal. The head node is the node usually represented at the top center of the tree. Every left move is a 0 and every right move is a 1. An example of a 16x16 8-bit color image and You can check out the Huffman tree that was made from the color of the pixels in Figure.4.

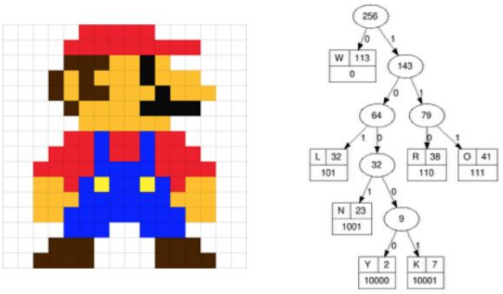


Figure 4: Huffman tree created from the color of the pixels

The generator used to create the Huffman tree flipped all nodes from "64" downwards to fit the tree into a smaller image dimension. Normally, all the nodes under “64” would be shown on the opposite side as the smaller weights are placed on the left and larger weights on the right.

The Huffman code used to represent the white pixels would be just one bit, ‘0’, as white is the most common color in the image. On the other hand, the Huffman code used to represent the yellow would be five bits, ‘10000’.

PNG compression has two options for using the Huffman tree. It can read the file, calculate the frequencies of the literals within the image, and then generate a Huffman tree specific to the image or it can use a standard Huffman tree. If a Huffman tree is created specifically for an image, its details must be included in the compressed file. On the other hand, if a standard Huffman tree is used, the compressed file must only indicate which standard Huffman tree is being used. In the case of PNG, the output of LZ77 compression or the sequence of bytes with distances and lengths represents the literals. For instance, in the given example, if WRW [D=3, L=9] is the most frequent literal in the image after LZ77 compression, it can be represented with only 7 bits instead of 7+ bytes. If this pattern repeats 100 times within an image, what would typically require 700 bytes can be compressed to only 88 bytes.

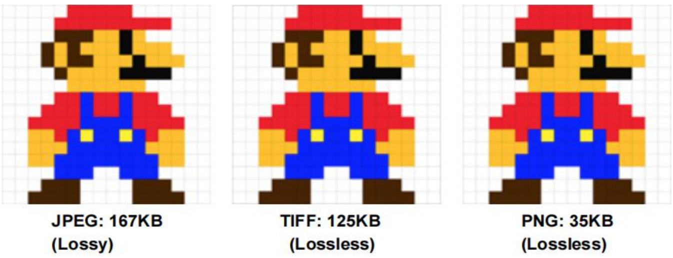


Figure 5: Comparisons of Image File Size

B. JPEG Compression

One form of Lossy Compression [11] for images is JPEG compression. This compression method allows for the user to alter the degree of compression by changing the quantization table, which will adjust the storage space and image quality. This quantization table is the reason that JPEG is lossy compression, as certain information will be irreversibly lost. JPEG compression also uses Discrete Cosine Transformations, which essentially takes a number of patterns created from different cosine lines and adds them together to form a specific image. JPEG undergoes multiple steps, including converting the color format to YCbCr, grouping the image, performing DCT, quantizing, and finally organizing with Huffman coding.

To make compression easier, it's best to convert the image to YCbCr color format. Y represents brightness and Cb and Cr represent the chroma components of blue and red respectively. The human eye relies on brightness for the perception of an image, so we can get better compression by working with the brightness component Y. The Cb and Cr components are also used to assist with compression. If you wanted the greatest possible quality, however, you can use the RGB color format. However, this does not allow for a smaller file size, which is one of the main reasons to use JPEGs. The next step is dividing the image into groups. Each group is made up of 8 by 8 pixels. The rest of the compression method is performed on the Y, Cb, and Cr components separately.

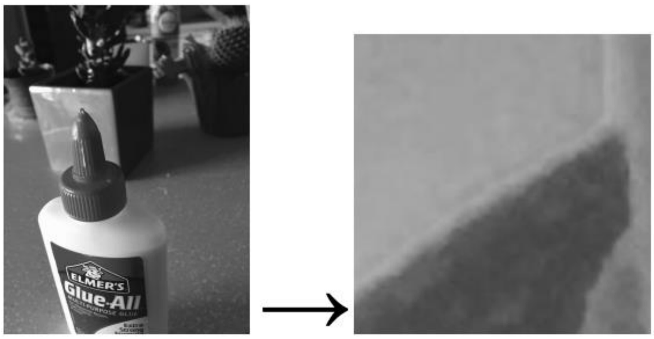
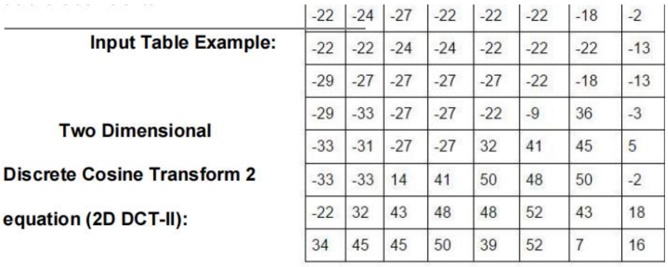


Figure.6

For each component we take the component's value of each pixel and place it in a table respective to where it is. These values range between 0 and 255, but in order to use discrete cosine transforms we need to alter it to closely resemble a cosine line. This involves subtracting every value by 128, making the range between -128 and 127. With these values, we apply the Two Dimension Discrete Cosine Transform 2, which will give us the Coefficients.



Input Table Example:

Two Dimensional

Discrete Cosine Transform 2

equation (2D DCT-II):

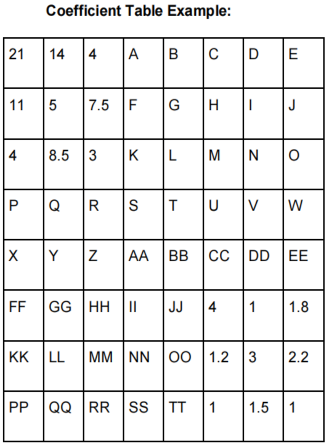
-22	-24	-27	-22	-22	-22	-18	-2
-22	-22	-24	-24	-22	-22	-22	-13
-29	-27	-27	-27	-27	-22	-18	-13
-29	-33	-27	-27	-22	-9	36	-3
-33	-31	-27	-27	32	41	45	5
-33	-33	14	41	50	48	50	-2
-22	32	43	48	48	52	43	18
34	45	45	50	39	52	7	16

$$G_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y} \cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

Figure.7

We create a table similar to the previous and place each coefficient in it. This table is special, as each spot does not correlate with the original image, but rather the pattern table that shows the possible patterns different cosine lines can create. Each

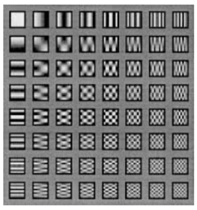
value in the coefficient table represents the influence of each pattern. The patterns in the top left are very simple while the patterns in the bottom right are highly complex. This often results in more influence being placed in the top left rather than the bottom right. The idea is that by adding enough of these patterns together, and taking into account influence, we can recreate an image.



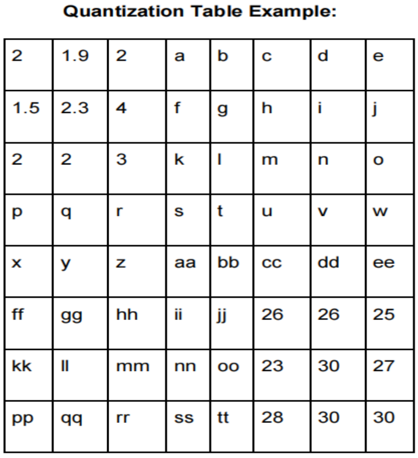
Coefficient Table Example:

21	14	4	A	B	C	D	E
11	5	7.5	F	G	H	I	J
4	8.5	3	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	AA	BB	CC	DD	EE
FF	GG	HH	II	JJ	4	1	1.8
KK	LL	MM	NN	OO	1.2	3	2.2
PP	QQ	RR	SS	TT	1	1.5	1

DCT Table:



Using the current coefficient table would recreate the exact image we started, but because we're attempting to compress the image we begin the next step, which is quantization. We take a quantization table, which can differ between programs and attempts. When we change the quantization table, it affects the balance between image quality and file size. We divide each value in the coefficient table by its respective quantization table and round it to the nearest whole number. When we change the quantization table, it affects the balance between image quality and file size. We divide each value in the coefficient table by its respective quantization table and round it to the nearest whole number.

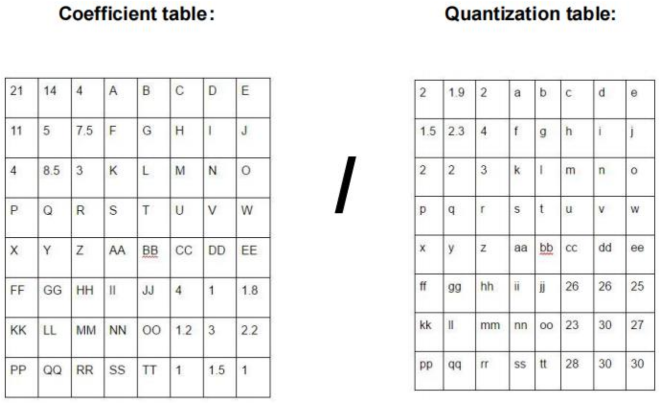


Quantization Table Example:

2	1.9	2	a	b	c	d	e
1.5	2.3	4	f	g	h	i	j
2	2	3	k	l	m	n	o
p	q	r	s	t	u	v	w
x	y	z	aa	bb	cc	dd	ee
ff	gg	hh	ii	jj	26	26	25
kk	ll	mm	nn	oo	23	30	27
pp	qq	rr	ss	tt	28	30	30

Figure.8

Values in the top left of the quantization table are often smaller than values on the bottom right. This allows us to keep the necessary coefficients while eliminating the less influential ones. To demonstrate If we take 21 from the Coefficient table and divide it by 1 from the quantization table, we round to 11. This value still has influence over the image. On the other hand, if we take 1 from the coefficient (the bottom right) and divide it by 30 from the coefficient table we get .033, which rounds to 0. This means that the pattern no longer has influence on the image. All of the least influential values are eliminated while leaving the ones with greater influence. The resulting image should be close to the same as the old one, with some differences that aren't always apparent.



Coefficient table:

21	14	4	A	B	C	D	E
11	5	7.5	F	G	H	I	J
4	8.5	3	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	AA	BB	CC	DD	EE
FF	GG	HH	II	JJ	4	1	1.8
KK	LL	MM	NN	OO	1.2	3	2.2
PP	QQ	RR	SS	TT	1	1.5	1

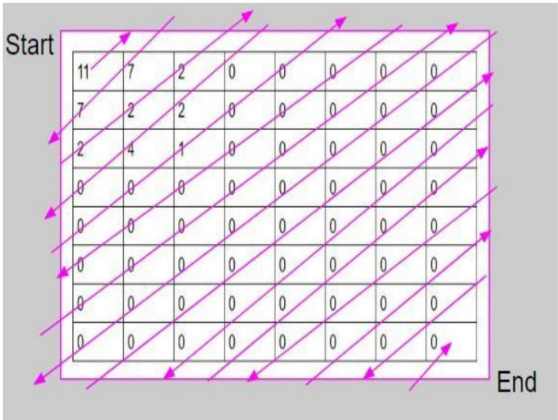
/

Quantization table:

2	1.9	2	a	b	c	d	e
1.5	2.3	4	f	g	h	i	j
2	2	3	k	l	m	n	o
p	q	r	s	t	u	v	w
x	y	z	aa	bb	cc	dd	ee
ff	gg	hh	ii	jj	26	26	25
kk	ll	mm	nn	oo	23	30	27
pp	qq	rr	ss	tt	28	30	30

Figure.9

We calculate the quantized table by dividing the coefficient table by the quantization table and rounding to the nearest whole number. We have gotten rid of unnecessary information, but the recreated image will be largely the same. This is what makes JPEG compression Lossy instead of lossless. Now we need to take the quantized coefficients and encode them. We can use Huffman Coding, a lossless method, to reduce the list. This will allow us to keep the same values we have now. If we take the quantized table and go in a zigzag pattern to organize the values for Huffman Coding then we will end up with several 0's in a row. This will allow us to compress more efficiently.



11	7	2	0	0	0	0	0
7	2	2	0	0	0	0	0
2	4	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure.10

The example quantized table would result in this for Huffman Coding:

11,7,7,2,2,2,0,2,4,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.

This is how to encode for JPEG, a lossy compression method. To decode, we simply reverse the order of steps. Because we return with new coefficients the final image will look different. However, it will be very close to the original. The final compressed image can look more or less accurate depending if the user wants less or more reduction in the file sizes respectively.

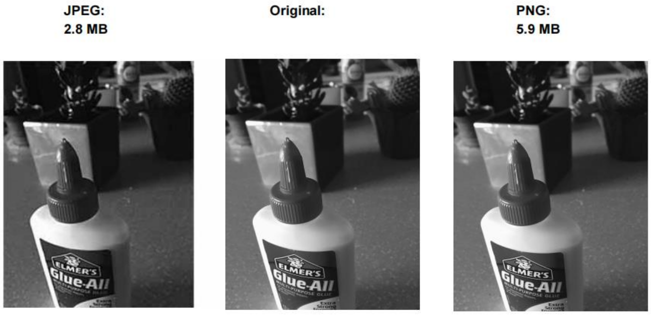


Figure. 11

C. We bP Compression

Google's WebP compressor does provide options to more accurately target a particular image quality. The WebP manual [12] explains this additional time by performing 'several passes of partial encoding' to meet the given constraints. the WebP encoder already includes support for targeting both quality and file size, enabling a direct comparison with the current implementation. We use the Peak Signal-to-Noise Ratio (PSNR) as the image quality metric, as it is the metric used by WebP. It supports the multiple-pass option to increase accuracy. It essentially performs partial compression to ensure that data loss is minimized. As a result, its cost and transportation time are higher compared to lossy compression.

Experiment 1: SSIM vs. BPP Plots for WebP and JPEG

We have conducted a study on the rate-distortion trade-off of JPEG and WebP image compression techniques. Our focus was on the SSIM vs. bits per pixel (bpp) plots for both techniques. We began by taking a source PNG image and compressing it to JPEG and WebP using all possible quality values ranging from 0-100. For each quality value, we plotted the SSIM and bpp achieved for both JPEG and WebP. We have generated SSIM vs. bpp plots for three images selected from the three public data sets we used in our study.

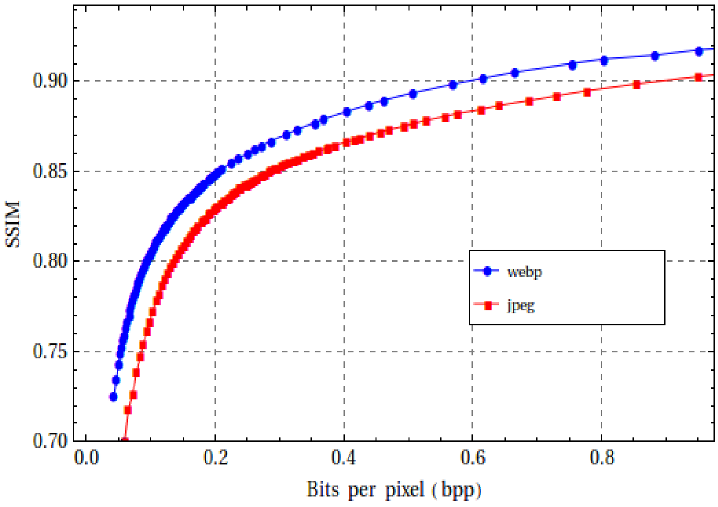


Figure 12: SSIM vs. BPP for Lenna

Figure 2: SSIM vs. BPP for kodim19.png from the Kodak dataset.

We analyzed the average behavior of the SSIM vs bpp plot for the Kodak and Tecnick data sets. To get the average behavior, we compressed all images in a dataset using WebP and JPEG at 100 different quality values. We then plotted the average of SSIM and bpp for each quality value. However, the image crawl dataset consists of diverse images that can't be easily aggregated. Below are the plots that demonstrate the average behavior of SSIM and bpp for the Kodak and Tecnick data sets respectively.

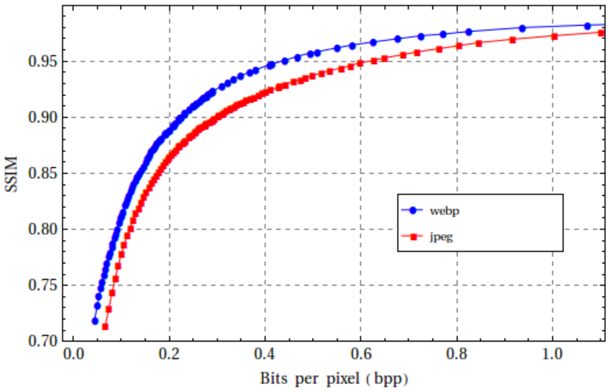


Figure 13: SSIM vs. BPP for RGB_OR_1200x1200_061.png from the Tecnick dataset

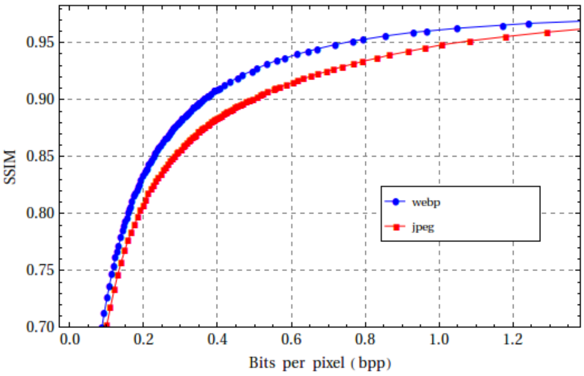


Figure 14: SSIM vs. BPP for the Kodak dataset

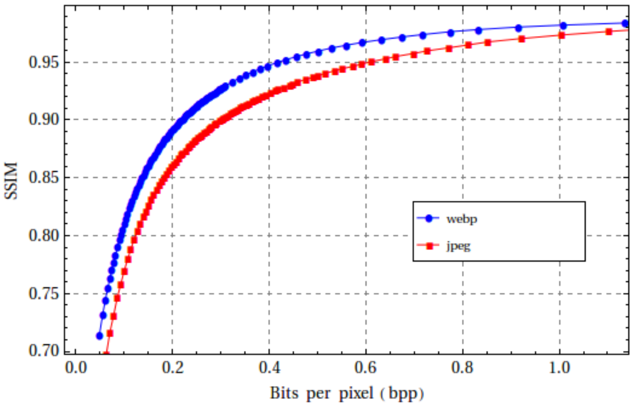


Figure 15: SSIM vs. BPP for the Tecnick dataset

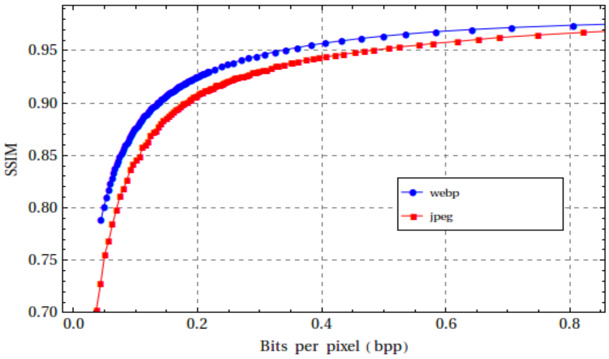


Figure.16

Based on the charts provided, it is clear that WebP consistently requires fewer bits per pixel than JPEG to achieve the same SSIM index. Based on the charts provided, it is clear that WebP consistently requires fewer bits per pixel than JPEG to achieve the same SSIM index.

D. HEIC Compression

There is an algorithm called JPEG2000 that is useful for compressing images. However, recent research suggests that High-Efficiency Image Coding (HEIC) is even better. HEIC processes image information using 10 bits, resulting in sharper images compared to JPEG2000. There are different versions of High-Efficiency Image File Format (HEIF), designed for storing images, image sequences, or videos using different codecs. Apple uses a variant of HEIF called HEIC in its iOS and macOS operating systems. HEIC employs HEVC/H.265 for compressing content. To differentiate between image and image sequence files from video content on its devices, Apple uses the .heic file extension. For video content encoded with HEVC, Apple uses the .mov file extension. The figure below illustrates the difference between HEIC and JPG image compression.

Figure 17: Difference between the HECI compressed image and JPG compressed image

The performance of HEIC, as assessed through PSNR, consistently outperformed that of both JPEG and JPEG2000, as depicted in Figure 2. Across all rates, the PSNR values for the HEIC method were higher than those for JPEG2000 and JPEG. Furthermore, the rate of PSNR reduction at high compression rates was notably lower for the HEIC method compared to other methods.

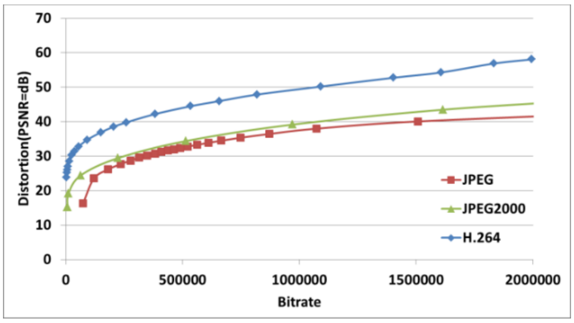


Figure 18: Difference between the HECI compressed mage, JPG compressed image as well as the JPEG2000

5. RESULTS

Including results for why JPEG is better **the** other.

Do you want results only for why JPEG is better then other Using JPEG images has many benefits in comparison to other image file types. Here are some of the key benefits of a JPEG image: Occupies less space on your storage device Easy to download and share across apps, devices, and networks Doesn’t require a lot of edits on contrast, white balance, sharpening, etc. JPEG images have distinct advantages that make them a practical choice for many photographers. One key benefit is that JPEGs are fully processed in-camera, meaning settings like White Balance, Color Saturation, Tone Curve, Sharpening, and Color Space are already applied to the image. This eliminates the need for extensive post-processing, making the image ready to use immediately. Additionally, the format boasts a small footprint, with JPEG images being significantly smaller in size compared to Raw files. This results in reduced storage requirements and faster processing times. The widespread compatibility of JPEG is another notable advantage, as most modern devices and software seamlessly support this format. Furthermore, the smaller size contributes to faster camera write speeds, allowing for a higher number of pictures to be stored in the camera buffer. This, in turn, enables photographers to capture more frames per second and shoot for extended periods without experiencing camera slowdowns. Another aspect that enhances flexibility is the choice of compression levels available for saving JPEG images, offering photographers the option to balance image quality and file size according to their preferences. Lastly, the format facilitates faster and more efficient backups due to its smaller file size, streamlining the overall workflow for photographers. WebP, while offering enhanced compression and quality benefits, comes with a set of considerations that designers and developers should be mindful of. Firstly, the format faces limitations in terms of browser support. While widely adopted browsers like Chrome, Firefox, and Microsoft Edge are WebP-compatible, older browsers such as Internet Explorer may not fully support it. Consequently, there's a need for precautionary measures, such as providing fallback images in alternative formats to ensure a seamless experience across different platforms. Secondly, incorporating WebP into the design process may introduce additional work for designers. They must not only export images in the WebP format but also consider the necessity of creating fallbacks for browsers that do not support it. This extra step requires careful planning and attention to detail during the design phase. Another potential concern is the risk of quality loss. Aggressive compression settings, while reducing file sizes, can sometimes compromise the visual integrity of images. Striking the right balance between file size and image quality becomes crucial to maintain a visually appealing website. It underscores the importance of thoughtful optimization strategies to leverage the benefits of WebP without sacrificing the overall quality of the user experience. JPEG 2000, a format introduced in 2000, is notable for its unique features and characteristics. However, its adoption has been limited, with native support primarily found in Safari, necessitating plugins for other browsers. One significant drawback is its lack of backward compatibility, requiring developers to code in a new standard if they wish to leverage both JPEG 2000 and the original JPEG format simultaneously. Early challenges also revolved around resource intensity, particularly in the encoding process. Modifications to the CPU and additional coding made the process more memory-demanding, a concern that has lessened with the increased capacity of modern machines. Despite this, the initial memory requirements were a notable obstacle when JPEG 2000 was first introduced. The slower and more intricate encoding process of JPEG 2000, compared to the widely used JPEG, contributed to delayed acceptance by websites and camera manufacturers until the format gained broader adoption. Furthermore, JPEG 2000 exhibits less content adaptability than JPEG, making manual adjustment of the bitrate necessary depending on the image content. However, it does offer an advantage in image detail retention during compression, particularly in lossless modes, owing to its foundation in wavelet technology. Despite its strengths, the format's unique characteristics and historical considerations have influenced its adoption and integration in the digital landscape. HEIC, despite its technological advancements and efficiency, has faced challenges in gaining widespread usage, particularly beyond the confines of the Apple ecosystem. The format has encountered limited compatibility with various platforms, as many software applications and platforms have yet to fully support HEIC files. This lack of universal compatibility raises practical concerns for photographers, who may encounter difficulties when attempting to edit or upload their images in their original HEIC format. The limited adoption of HEIC by non-Apple platforms underscores the importance of considering broader compatibility issues when choosing image formats for practical and seamless workflows across different devices and software environments.

6. CONCLUSION

Our paper presents a new method for compressing images. We compared four standard compressors, namely libjpeg, Png, HEIC, and WebP, and developed a technique to automatically adjust image compression quality based on user requirements for file size and image quality. Our approach offers significant improvements over previous methods for JPEG compression. Furthermore, our method achieves higher prediction accuracy in JPEG implementation which taking less time.

7. REFERENCES

[1] Independent JPEG group. http://www.ijg.org/,January 2016.

[2] A. Lewis, S. Ghosh, and N.-F. Tzeng. Run-time energy consumption estimation based on workload in server systems. In Proceedings of the 2008 Conference on Power Aware Computing and Systems,HotPower'08, pages 4–4, Berkeley, CA, USA, 2008.USENIX Association.

[3] WebP manual, Google Developers. https://developers.google.com/speed/webp/docs/cwebp,December 2015.

[4] S. Chandra and C. S. Ellis. JPEG compression Metric as a quality aware image transcoding. In 2nd USENIX Symposium on Internet Technologies & Systems (USITS'99), 1999. C. S. Ellis. JPEG compression metric as a quality aware image transcoding. In 2nd USENIX Symposium on Internet Technologies & Systems (USITS'99), 1999.

[5] H. Tong, M. Li, H. Zhang, and C. Zhang. Blur detection for digital images using wavelet transform. In Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on, volume 1, pages 17–20Vol.1, June 2004.

[6] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In Computer Vision and Pattern Recognition, 2001. CVPR 2001.Proceedings of the 2001 IEEE Computer Society Conference on, volume 1, pages I–511–I–518 vol.1,2001.

[7] K. Fant. A nonaliasing, real-time spatial transform technique. Computer Graphics and Applications, IEEE, 6(1):71–80, January 1986.

[8] U. Javaid. Find & remove similar photos instantly.

[9] http://www.addictivetips.com/windows-tips/find-remove-similar-photos-instantly, June 2010.

[10] S. E. Umbaugh. Computer Imaging: Digital Image Analysis and Processing. CRC Press, 2005.

[11] Hussain AJ, Al-Fayadh A, Radi N. Image compression techniques: A survey in lossless and lossy algorithms. Neurocomputing. 2018 Jul 26;300:44-69.

[12] http://www.math.utah.edu/~gustafso/s2017/2270/projects2017/bradyJacobsonSamuelTeare/GroupProjectCompression.pdf

[13] WebP manual, Google Developers. Tarnation https://developers.google.com/speed/webp/docs/cwebp, December 2015 https://en.wikipedia.org/wiki/Data_compression

[14] https://developers.google.com/speed/webp/docs/webp_study

[15] https://www.math.utah.edu/~gustafso/s2017/2270/projects-2017/bradyJacobsonSamuelTeare/GroupProjectCompression.pdf

[16] https://www.sciencedirect.com/science/article/abs/pii/S0925231218302935

[17] http://ijcem.in/wpcontent/uploads/2015/05/Image_Compression_for_Mobile_Devices_using_Prediction_and_Direct_Coding_Approach.pdf

[18] https://www.smashingmagazine.com/2021/09/modern-image-formats-avif-webp/

[19] https://cviptools.ece.siue.edu/examples.php

[20] https://www.ideamktg.com/blog/is-jpeg-or-png-better