Image Compression and Decompression using Adaptive Interpolation

SUNIL BHOOSHAN¹, SHIPRA SHARMA²
Jaypee University of Information Technology
¹Electronics and Communication Engineering Department
²Computer Science and Engineering Department
Waknaghat, Solan
INDIA
shipra.sharma@juit.ac.in, sunil.bhooshan@juit.ac.in

Abstract: A simple and fast lossy compression and decompression algorithm for digital images is proposed. The method offers varying compression ratios (depending on dimensions of the image) and the acquired decompressed image is close to the original one. A selectable tradeoff between storage size and image quality is allowed, making it possible to adjust degree of compression. Compared to JPEG, it provides us better compression ratio. The suggested method does not restrict itself to any particular type of image.

Key–Words: Lossy image compression, Interpolation, Compression ratio

1 Introduction

Storage and transmission of digital images has become more of a necessity than luxury these days. Hence the importance of image compression. It involves minimization of the number of information carrying units, pixels. This means that an image where adjacent pixels have almost the same values leads to spatial redundancy. Any lossy compression technique has to achieve high compression ratios while maintaining high visual quality of the decompressed image [1].

During the last two decades, various lossy and lossless image compression techniques have been developed as discussed in [2]. Lossy coding methods provide high compression ratios but do not recover the exact data. While lossless methods recover the exact original data but do not compress the image to such extent as the former method. So, high compression ratios can be achieved for images where some loss of data does not matter. When a compressed image is decompressed to the original one, some distortion/loss is acceptable provided image quality is not compromised. This is because HVS (Human Visual System) does not detect slight changes in the image. The same has been elaborated in [2]. Therefore, pixel values may differ slightly in the original and decompressed images and HVS will not detect the difference between them [3]. One method as in [4] gives good results but is hardware dependent. Recently neural networks have also been used [5] for compressing images, but they have low compression rates. For implementing interpolation on an image it has to be considered in spatial domain. This has been discussed and implemented in [6] for linear interpolation. Various approaches exist in literature which propose interpolation techniques for “perceptually motivated” coding applications [7], [8] and [9].

Our purpose in this paper will be to discuss possible algorithm for compressing a still digital image and then acquiring back the original image from the compressed one. We use non-linear interpolation to reconstruct the original image from the compressed one. The proposed decompression algorithm can also be universally used to enlarge any given image. We then compare our method with JPEG format.

The rest of the paper is organized as fol-
In section 2 the compression and de-compression methods are proposed. In section 3 some computational results will show that our proposed method indeed produces good image quality. Discussion and conclusion will follow in section 4.

2 Procedure

The procedure is outlined for grayscale images but may be extended to colored images. The image is considered to be a smooth function of \( x \) and \( y \), even though it may not seem so visually.

Consider a \( b \) bit grayscale image of size \( m \times n \) which is to be compressed. Since we are considering \( b \) bits therefore, gray level values, of the pixel in this image, will range from 0 to \( 2^b - 1 \). Here 0 represents black and \( 2^b - 1 \) represents white and similarly intermediate values represent the transition from black to white.

2.1 Compression

To compress the image we proceed in the following manner. We start from the first pixel of the image and select other eight adjacent pixels such that a square is formed. The zoomed in pixels in Figure 1 show how a set is formed.

To form such sets (squares) we first check whether \( m \) and \( n \) are divisible by 3. If this is not the case, then zero padding is done so that \( m \) and \( n \) become divisible by 3. This padding may lead to some error which will be removed when original image is acquired back after decompression. The image is then divided into non-overlapping squares and no pixel is missed as shown by original image in Figure 2. The center pixel from each set is chosen and a new image is formed using these pixels.

Consider an Image in matrix form which is of size \( (1024, 1024) \). The starting pixel is at position \((1, 1)\). Figure 3 represents a part of this matrix. To compress it by the described method we first check whether 1024 is divisible by 3. As it is not, we pad the image by 2 rows and 2 columns. The squares (sets) are formed starting from the first pixel. The center pixels (boundaries highlighted) in Figure 3 are selected and stored in a new matrix, say \( N \) shown in Figure 4. As can be seen in Figure 3 pixel values at position \((2,2), (2,5), (5,2)\) becomes pixel values at position \((1,1), (1,2), (2,1)\) respectively in Figure 4. Similarly, the whole image is compressed.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>175</td>
<td>174</td>
<td>193</td>
<td>183</td>
<td>183</td>
<td>175</td>
<td>183</td>
<td>175</td>
<td>183</td>
</tr>
<tr>
<td>2</td>
<td>174</td>
<td>175</td>
<td>174</td>
<td>183</td>
<td>183</td>
<td>183</td>
<td>175</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>3</td>
<td>173</td>
<td>172</td>
<td>171</td>
<td>161</td>
<td>161</td>
<td>151</td>
<td>161</td>
<td>151</td>
<td>161</td>
</tr>
<tr>
<td>4</td>
<td>172</td>
<td>171</td>
<td>161</td>
<td>151</td>
<td>161</td>
<td>151</td>
<td>161</td>
<td>151</td>
<td>161</td>
</tr>
<tr>
<td>5</td>
<td>171</td>
<td>170</td>
<td>160</td>
<td>150</td>
<td>160</td>
<td>150</td>
<td>160</td>
<td>150</td>
<td>160</td>
</tr>
<tr>
<td>6</td>
<td>170</td>
<td>169</td>
<td>159</td>
<td>149</td>
<td>159</td>
<td>149</td>
<td>159</td>
<td>149</td>
<td>159</td>
</tr>
<tr>
<td>7</td>
<td>169</td>
<td>159</td>
<td>149</td>
<td>139</td>
<td>149</td>
<td>139</td>
<td>149</td>
<td>139</td>
<td>149</td>
</tr>
<tr>
<td>8</td>
<td>168</td>
<td>158</td>
<td>148</td>
<td>138</td>
<td>148</td>
<td>138</td>
<td>148</td>
<td>138</td>
<td>148</td>
</tr>
<tr>
<td>9</td>
<td>167</td>
<td>157</td>
<td>147</td>
<td>137</td>
<td>147</td>
<td>137</td>
<td>147</td>
<td>137</td>
<td>147</td>
</tr>
<tr>
<td>10</td>
<td>166</td>
<td>156</td>
<td>146</td>
<td>136</td>
<td>146</td>
<td>136</td>
<td>146</td>
<td>136</td>
<td>146</td>
</tr>
</tbody>
</table>

Figure 4: Compressed Matrix, \( N \)
2.1.1 Variants of Compression

The above method can be applied for different compression ratios. Keeping in the vein of things, one out of every four, sixteen, twentyfive pixels and so on can be chosen so that, different compression ratios on images of different sizes can be obtained.

2.2 Decompression

This method involves application of second order interpolation over the compressed image. For this, first let us discuss about the information given by pixels. Any pixel in the image has two values. First is its position \((x, y)\) in the image and second is its gray value \(G\) at that position.

Figure 5 shows how a \(n^{th}\) pixel which is at position \((x_p, y_q)\) and has gray value \(G_n\), can be plotted in this coordinate system. As can be seen in the Figure 5, we plot \(x_p\) or \(y_q\) on the \(\xi - axis\) depending on whether we are moving in \(x\) or \(y\) direction (in the image) respectively and on \(G - axis\) we plot the gray value \(G_n\).

Assuming that compressed image is of size \((m', n')\) and starts from \((1, 1)\), let us implement this decomposition method considering image as matrix:

1. We acquire the first pixel, \(P_1\), which is at position \((1, 1)\) in the compressed image.
2. The next pixels chosen are \(P_2\) and \(P_3\) which are at position \((1, 2)\) and \((1, 3)\) in the compressed image, considering we are moving in \(x\) direction.
3. Obtain the gray values of \(P_1, P_2\) and \(P_3\) which are \(G_1, G_2\) and \(G_3\) respectively.
4. Figure 2 shows us that originally two pixels were present between any two given adjacent pixels of compressed image. Interpolate two pixels between \(P_1\) and \(P_2\) and two between \(P_2\) and \(P_3\) as follows:
   - Figure 6 depicts the position of \(P_1, P_2\) and \(P_3\) on \(\xi v/s\ G\) axis. \(P_1\) is at \((1, G_1)\) and \(P_2\) at \(((1+3), G_2)\) (leaving positions of two pixels) on this coordinate system and similarly \(P_3\) is at \((1 + 6), G_3\).
   - The second order curve passing through these three points is:
     
     \[
     18G = (G_1 - 2G_2 + G_3)\xi^2 + (-11G_1 + 16G_2 - 5G_3)\xi + 28G_1 - 14G_2 + 4G_3 \tag{1}
     \]
     
     It is depicted in Figure 6.
   - Use Equation (1) to calculate values of the four interpolated pixels, say \(IP_1, IP_2, IP_3\) and \(IP_4\) which are at position 2, 3, 5 and 6 respectively, on \(\xi - axis\) (refer Figure 6). \(IG_1, IG_2, IG_3\) and \(IG_4\) are the calculated gray values of \(IP_1, IP_2, IP_3\) and \(IP_4\) respectively.
   - Store \(G_1, IG_1, IG_2, G_2, IG_3, IG_4\) and \(G_3\) in a new matrix at positions corresponding to the pixel’s position in the decompressed image, i.e., at \((1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)\) and \((1, 7)\).
5. Acquire next two pixels in the compressed image, such that no pixel is missed, in the \(x\) direction as in Step 1 and 2. Repeat Steps 3 to 5 until the whole image is decompressed in this direction.
6. Repeat Steps 1 to 5 to decompress the image in \(y\) direction.
7. When the complete compressed image is decompressed in both the directions, render the reconstructed image.

To make the above steps clear let us consider the compressed matrix of Figure 4. We begin with taking first three pixels \((1, 1), (1, 2)\) and \((1, 3)\). As we are moving horizontally we mark 1, 4, 7 on \(\xi - axis\) and 176, 176.
Using this the second order equation is derived. We find four new points on this curve corresponding to values \( \xi = 2 \), \( \xi = 3 \), \( \xi = 5 \) and \( \xi = 6 \) on \( \xi - \) axis. The values come out to be 174, 174, 180 and 185. These four values are put in a new matrix as shown by dashed arrows in Figure 7.

When this procedure is continued in this direction on the whole compressed matrix the resultant matrix obtained is as shown in Figure 8.

Then the same method is repeated vertically on matrix of Figure 8 as shown by solid arrows in Figure 7, we obtain the final matrix which is shown in Figure 9. This is the final decompressed image.

In the next section we apply this method on some sample images.

## 3 Computational Results

The proposed method is evaluated using some different type of images. The following sub-sections take on the various aspects of applying the above algorithms. For each image the method is compared to well known lossy JPEG compression format.

### 3.1 Various Compression Ratios

Figure 10 shows an image of size 11.3MB. Various compression ratios are applied to it. Figure 11(a) depicts the cropped part of the original image. Figure 11(b), (c), (d), (e) show this part of the decompressed images with compression ratios of 4:1, 9:1, 16:1 and 25:1 respectively. This shows that our method gives tolerable quality of decompressed image till a compression ratio of 25:1 for image of this dimension. After that the image starts blurring to an unacceptable extent.

The Table 1 shows variation in the compression ratio with variation in the size of image. As can be observed, image size is directly proportional to compression ratio.

### 3.2 Results and Comparison

For thoroughness of our comparison various types of grayscale images were used as test images. Two of them are shown in Figure 12. Figure 12(b) is used as it is natural image with varying textures. After applying compression algorithm on it, the resultant
Table 1: Recommended Compression For Various File Size

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Recommended Compression Ratio*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB – 700KB</td>
<td>4 : 1</td>
</tr>
<tr>
<td>700KB – 5.5MB</td>
<td>9 : 1</td>
</tr>
<tr>
<td>5.5MB – 9.5MB</td>
<td>16 : 1</td>
</tr>
<tr>
<td>9.5MB – 14.5MB</td>
<td>25 : 1</td>
</tr>
<tr>
<td>&gt; 14.5MB</td>
<td>36 : 1</td>
</tr>
</tbody>
</table>

*Compression ratio = (Uncompressed Size / Compressed Size) : 1

Table 2 illustrates how our method fares with respect to JPEG. As can be seen, when the image size is large then our algorithm provides better compression than lossy JPEG. But for small images, to maintain the quality of decompressed image and so that minute details are not lost, our method compresses less.

3.3 Universal Decompression Algorithm

The proposed decompression algorithm can also be used to enlarge any given grayscale image. Figure 14(a) shows cropped part of Lena image. When our reconstruction algorithm is applied on it we obtain the image...
<table>
<thead>
<tr>
<th>Image Name</th>
<th>Original Size</th>
<th>JPEG Size</th>
<th>Our Compression Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 1</td>
<td>32.2MB</td>
<td>2.1MB</td>
<td>568.5KB</td>
</tr>
<tr>
<td>Image 2</td>
<td>27MB</td>
<td>1.1MB</td>
<td>474.5KB</td>
</tr>
<tr>
<td>Image 3</td>
<td>469.4KB</td>
<td>98.3KB</td>
<td>101.1KB</td>
</tr>
</tbody>
</table>

Table 2: Comparison with JPEG format

Figure 12: Original Images

(a) Baby

(b) Garden

Figure 13: Decompressed Images

(a) Baby

(b) Garden
3.4 Statistical Assessment

One of the methods for quality assessment is an error measurement that provides a high correlation to perceptual image quality. As mathematical expression of human perception quality metric are not widely agreed upon, the “Mean Square Error” is still commonly used criterion. The average mean square error for an $N \times M$ size image is defined as

$$e_{ms}^2 = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} E(x_{i,j} - x'_{i,j})^2$$  (2)

where $x_{i,j}$ and $x'_{i,j}$ represent the $N \times M$ original and the reproduced images, respectively. The average mean square error is often estimated by average sample mean-square error in the given image which is depicted in Equation 3.

$$e_{mis}^2 \approx \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} (x_{i,j} - x'_{i,j})^2$$  (3)

As our method is lossy, it leads to some degradation of the reconstructed image. As we can see in Figure 13 that visually this is not noticeable. Table 3 shows that even in terms of mean square error the degradation is within tolerable limits.

4 Conclusions

This work documents a theoretical and computational investigation of compressing an image and then acquiring back the original image from it in a very simple manner. The paper proposes a compression algorithm which, while being computationally inexpensive, reduces the image size considerably. The decompression algorithm, based on interpolation, is then implemented on this decompressed image. The results suggest that fairly good compression is obtained in terms of qualitative context. The paper also suggests different ratios of compression which can be applied depending on the size of the original image to maintain the quality of decompressed image. Further on, the decompression algorithm can also be used to enlarge any grayscale image. The techniques suggested here are suitable for applications which cannot handle too many complex calculations, like mobile phones, without compromising on their quality.

References:


### Table 3: Error Estimation

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Original Size</th>
<th>Compressed Size</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 1</td>
<td>32.2MB</td>
<td>568.5KB</td>
<td>$5.0122e^{-005}$</td>
</tr>
<tr>
<td>Image 2</td>
<td>27MB</td>
<td>474.5KB</td>
<td>$1.3241e^{-005}$</td>
</tr>
<tr>
<td>Image 3</td>
<td>469.4KB</td>
<td>101.1KB</td>
<td>$2.8355e^{-004}$</td>
</tr>
</tbody>
</table>

