Received October 15, 2021, accepted October 26, 2021, date of publication November 4, 2021, date of current version November 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3125009

A Novel Multilevel Lossy Compression Algorithm for Grayscale Images Inspired by the Synthesization of Biological Protein Sequences

MOHAMMAD NASSEF[1,2] AND MONAGI H. ALKINANI[1]

[1]Department of Computer Science and Artificial Intelligence, College of Computer Science and Engineering, University of Jeddah, Jeddah 23890, Saudi Arabia
[2]Department of Computer Science, Faculty of Computers and Artificial Intelligence, Cairo University, Giza 12613, Egypt

Corresponding author: Mohammad Nassef (mnassef@uj.edu.sa; m.nassef@cu.edu.eg)

This work was supported by the University of Jeddah, Jeddah, Saudi Arabia, under Grant UJ-02-071-DR.

ABSTRACT Enormous number of images are generated daily in all areas of life, including social media, medical and navigation images. Moreover, the development of smart phones among other specialized media-capturing devices has witnessed great advances during the last decade. Consequently, the storage, transmission, and analysis of images become essential and frequent tasks. Thus, various research efforts tried to address the image compression problem from different computational perspectives. This article presents a novel multilevel lossy compression algorithm for grayscale images, namely **Image-as-Protein** (*IaP*), that is inspired by the translation of *DNA* sequences into *protein* sequences that occurs inside live beings. Because of the high similarity of the resulting textual *protein* sequence, it can be tackled by general text compression techniques with competitive compression ratios. Various qualitative comparisons and quantitative measures such as *BPP*, *SSIM* and *PSNR* have been carried out on multiple grayscale image benchmark datasets. The experimental results showed that the proposed algorithm is promising compared to the famous JPEG lossy image compression standard.

INDEX TERMS DNA, grayscale images, image compression, image sequences, image storage, JPEG, JPG, lossy compression, protein sequences.

I. INTRODUCTION

Many people deal with enormous amount of images in various aspects of their daily life, from their awakening to their sleep. Social media applications, e-commerce applications, navigation maps, medical diagnoses, governmental procedures are some examples of applications that store and transmit images immensely. Various research efforts tried to address the image compression problem from different perspectives. There are two types of images: raster and vector. Raster images are composed of two-dimensional array of pixels, whereas vector images are stored as geometrical and mathematical expressions [1]. Every pixel in each raster image represents a point in that image, and the resolution and color depth of an image determines its total size.

The associate editor coordinating the review of this manuscript and approving it for publication was Yizhang Jiang.

In a grayscale image, each pixel is stored in one byte and takes a value between zero and 255 according to its intensity. The value of black pixels is zero, whereas the value of white pixels is 255. Grayscale images are widely used by the research community as they are easy to understand and manipulate than color images. So, many research problems can be initially attempted in grayscale level. Furthermore, satellite and medical images are grayscale by nature.

A. IMAGE COMPRESSION

The enhancements applied to the current image capturing devices have increased both the resolution and color depth of images to an amazing degree. Thus, image compression represents a vital and challenging topic because of its direct effect on the storage of images and its transmission over internet. Many image compression techniques have been published in the last decades, some of them are lossless [2, Chapter 7], whereas others are lossy [3]. In lossless image compression, decompression of the compressed image retrieves exactly the original image. On the contrary, in lossy image compression, some insignificant features of the original image are intentionally neglected by the compression algorithm. So, the decompressed image is not exactly the same as the original image. Some fields require the lossless image compression algorithms, especially if any loss in the original image is not tolerated. In this article, ideas from Biology have been employed to develop out-of-the-box lossy image compression algorithm that provides the community with multiple lossy compression options for grayscale images.

B. BIOLOGICAL PRELIMINARIES

A clone of the human's genetic material resides in almost every cell of his body. That genetic material consists of very long double-stranded *DNA* sequences (Figure 1). In a biological process entitled the *Central Dogma* [4], parts of the *DNA* sequences, namely *Genes*, are transcribed (converted) into *mRNA* (simply, *RNA*) sequences which in turn are used to synthesize various corresponding *protein* sequences to perform essential biological tasks.

Sequencing machines have been invented since three decades to convert the complete genetic material of some creature into long digital *DNA* sequences [5]–[8]. These digital *DNA* sequences have been stored into computers for future computational analysis and manipulation. Hence, the role of computer scientists was to build computer algorithms to computationally analyze these very long sequences to discover and treat diseases, and to know variations among creatures. For more details about the computational conversion of *DNA* sequences to *protein* and the computational genome assembly, the reader can refer to [9].

A *DNA* sequence is composed of two complementary strands (Figure 1). The building blocks of each strand are four chemical bases that are digitally represented by the four characters {A, C, G, and T}. The sequence of characters **CGTACGTCCGTGCGTGCGTCCGTA** is an example of a single *DNA* strand with 24 characters. As a computational shortcut, every "T" base in the given *DNA* strand is substituted by a "U" base to obtain the *RNA* sequence with no need to the complementary *DNA* strand. So, any *RNA* sequence can be represented by four characters {A, C, G, and U}. So, the *RNA* sequence corresponding to the past *DNA* strand is **CGUACGUCCGUGCGUGCGUCCGUA**. The *protein* production (translation) process converts every three adjacent *RNA* bases, namely a *Codon*, into one *protein* unit called *AminoAcid* (*AA*). Prior to translating the *RNA* sequence into a *protein*, unnecessary parts of the *RNA* sequence (Introns) are cutted in a process called Splicing, and the remaining intended parts (Exons) are used to build the desired *protein* sequence. Any *protein* sequence is built from 20 different *AminoAcids (AAs)*: {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, and Y}. The *RNA Codons* as well as its corresponding *AminoAcid* characters are listed in Figure 2. Assuming that there are no cuts required, the later *RNA* sequence generates a *protein* sequence with the following eight characters: RTSVRASV.

During the past decades, biologists discovered how the biological systems inside the living creatures are working in a perfect manner. This article proposes a novel multilevel lossy image compression algorithm, namely **Image-as-Protein** (*IaP*) that is inspired by the inherent perfect organization and synthesization of the biological sequences inside living creatures. More specific, *IaP* mimics the biological translation of *DNA* sequences into *protein* sequences that constantly occur inside the cells of live beings. Applying that translation process on pixels of image results in textual protein sequence that can effectively be encoded using general text compression techniques.

The remaining sections of this article are outlined as follows: Section II briefly introduces the research efforts of lossy image compression and how some of these efforts tried to compress images using ideas from Biology. The proposed *IaP* compression algorithm is detailed in Section III, whereas the results are presented and discussed in Section IV. At the end, Section V concludes the article.

II. RELATED WORK

This section highlights the research efforts reviewing the lossy image compression techniques. Next, the JPEG standard and its modern competitors are contrasted from the compatibility perspective. At last, the section reviews the research articles that use *DNA* sequences in image compression.

Hussain *et al.* [10] and Thakur *et al.* [11] published two review articles that summarize various state-of-the-art lossy and lossless image compression methods. The authors talked about lossy image compression methods including Predictive Coding, Transform Coding (including Karhunen-Loeve Transform (KLT), Discrete Cosine Transform (DCT), Discrete Fourier Transform (DFT), and Walsh-Hadamard Transform (WHT)), and the JPEG compression standard. Moreover, they discussed various quantitative quality measures such as *Mean-Square-Error (MSE)*, *Peak-Signal-to-Noise-Ratio (PSNR)*, and *Structure-Similarity-Index-Metric (SSIM)* [12], [13].

JPEG is the most dominant lossy image compression standard invented by the Joint Photographic Experts Group [14]. It has been used for the past three decades to this day to compress images in various quality levels. Images lossy compressed by JPEG take the extension ".jpeg" or ".jpg". However, the quality of compressed images is affected by different artifacts such as ringing, blocking, and blurring effects. As the intended size of the compressed file decreases, the image quality decreases to an unsatisfactory level.

Several research efforts tried to eliminate or reduce the artifacts resulting in the JPEG decompressed images based on various Artificial Intelligence models of Artificial Neural Networks (ANNs) and deep learning [15]–[19]. These ANNs models are trained to learn end-to-end mappings of artifacts features between the distorted images and their

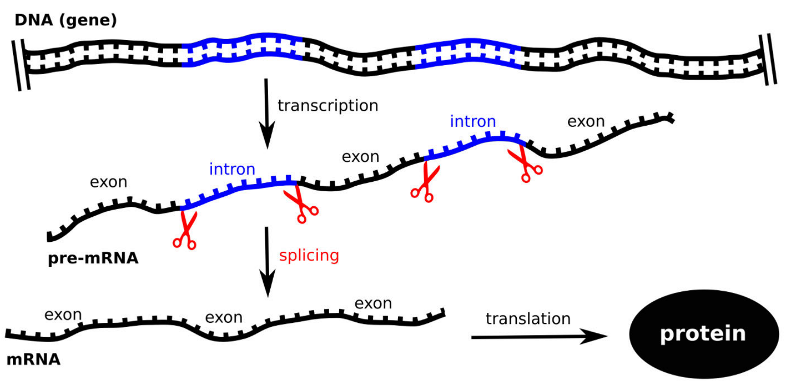


FIGURE 1. The conversion of a *DNA* sequence to a *protein* sequence from the Central Dogma prespective. (www.helmholtz-muenchen.de).

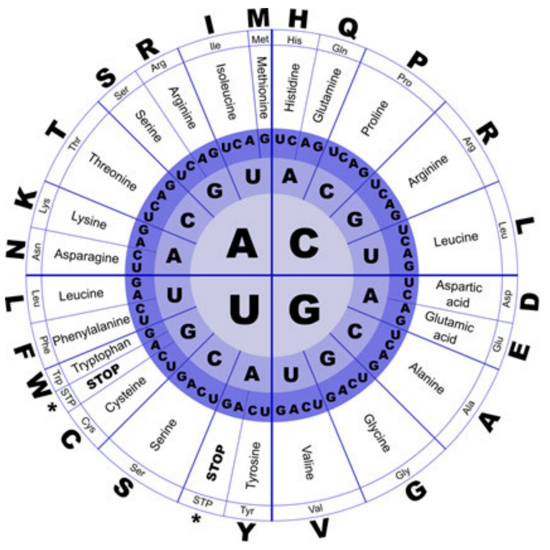


FIGURE 2. The *protein AminoAcids* resulting from the different *RNA codons* [9, p.185].

corresponding original versions. After the learning phase is done, ANNs take a distorted decompressed image with artifacts as an input, and produce an image with higher quality (less or no artifacts) with the absence of its original version. Although these ANNs have massive learning capabilities, their major drawback is the time consumption of their training phase.

Alternatively, the last decade has seen the emergence of promising image compression alternatives. JPEG-XR [20], [21] and JPEG-XT [22], [23] tried to enhance the compression quality with the same compression ratio compared to JPEG. Although they are backward compatible with the JPEG standard, they still face standardization and compatibility issues.

On the other hand, the *Better Portable Graphics (BPG)* [24] standard was invented by Bellard in 2014 to replace JPEG. Although the compression ratio of *BPG* is sometimes competitive to JPEG [25], it still has no built-in native support in popular browsers unless its decoder is implemented in integrated JavaScript libraries. Furthermore, WebP [26] is an image compression standard developed by Google to retain higher image quality for the same compressed size compared to JPEG. WebP provides both lossy and lossless image compression, however, it is still not supported by famous browsers such as Safari [27].

The incompatibility issues of the aforementioned JPEG alternatives forced websites to keep more than one version of the same image, as only portion of the website visitors are using compatible browsers. So, instead of saving storage, these modern alternatives resulted in storage overhead. Moreover, each alternative has its own strengths and weaknesses regarding time consumption, decoder compatibility, and image quality. As a matter of fact, having one compression standard that perform perfectly for all images does not exist till this moment. Thus, in Section IV, the authors decided to compare the proposed *IaP* compression levels with the JPEG compression standard.

Bandyopadhyay and Chakraborty [28] introduced an image compression technique that encodes a given image using four different *DNA* encoding methods and selecting the method with the minimum number of bits in total. It was encoding an image by solely storing the *DNA* sequence of some organism appearing in that image. However, the authors did not clarify how the enormous space of bits representing the image pixels were exactly fitted into the selected *DNA* sequence that is in fact very shorter. Moreover, the decompression process was not illustrated.

Dimopoulou *et al.* [29] proposed an image encoding technique that stores a given image by synthesizing the biological *DNA* sequences corresponding to its Discrete Wavelet Transform (DWT), and then storing the synthesized sequences into a laboratory tube. Although the proposed idea is creative, it is impractical because it is time and money consuming as it needs real sequencing machines as well as biotechnology specialists and assets for both image encoding and restoration.

TABLE 1. The Analogy between the process of biological *protein* Synthesization process (Figure 1) and the proposed compression algorithm (*IaP*).

Biological *protein* Translation	Proposed Image Compression Algorithm (*IaP*)
DNA Sequence	Digital sequence of *DNA* patterns representing image pixels
RNA Exon (*Codon*)	1st, 2nd, and 3rd characters of pixel's *RNA* pattern
RNA Intron	4th character of pixel's *RNA* pattern
protein AminoAcid Sequence	Image's *OriginalAAs* sequence
protein Folding	Encoding of *OriginalAAs* sequence into a compressed output file

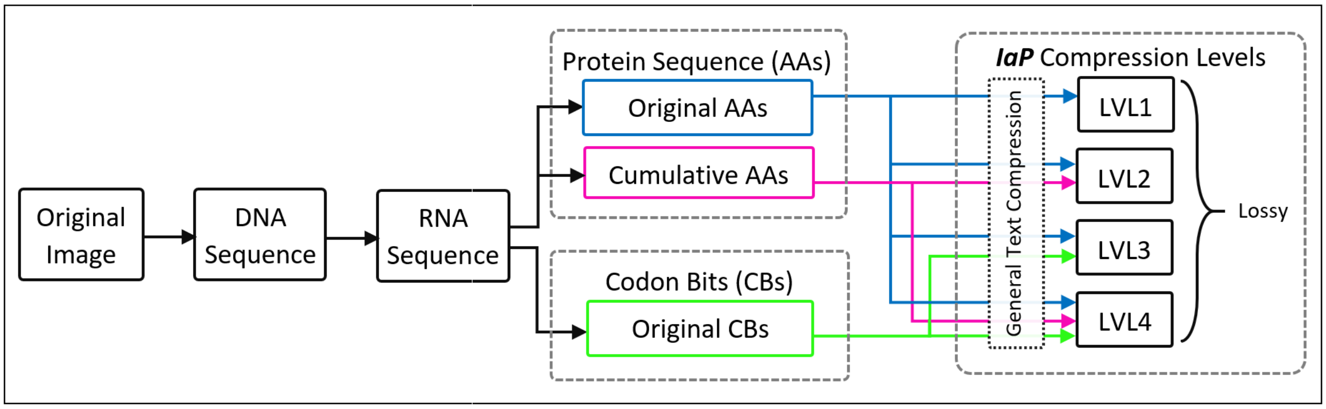


FIGURE 3. The proposed Image-as-Protein (*IaP*) multilevel lossy image compression framework.

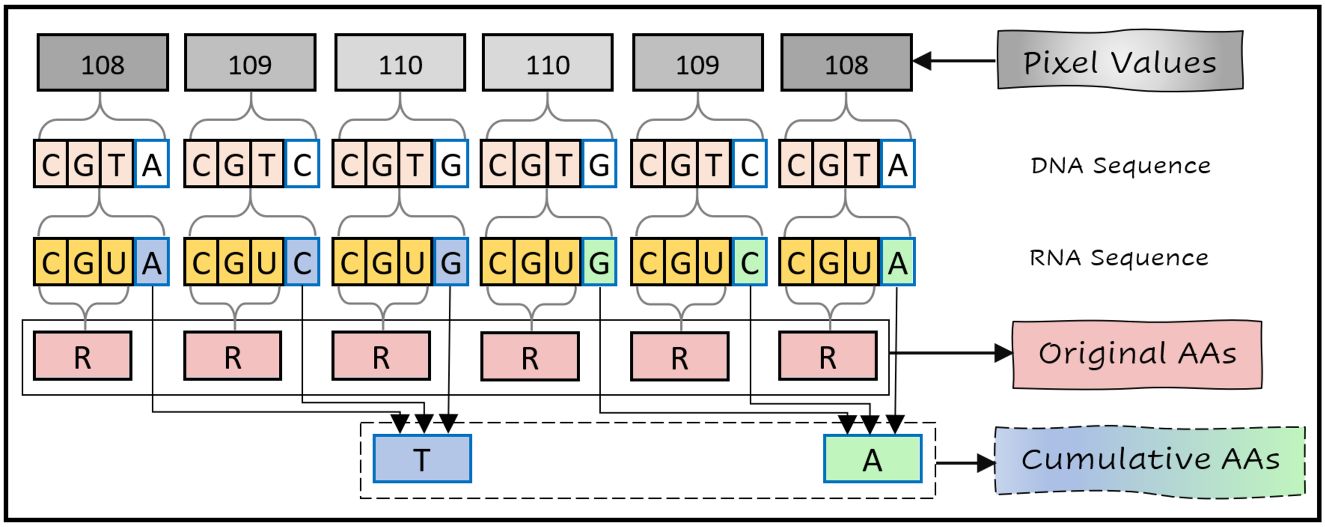


FIGURE 4. An illustrative example of selective translation of six image pixels to the *OriginalAAs* and *CumulativeAAs protein* sequences. The *DNA* codon 'CGT' is converted to *RNA* codon 'CGU' before getting the 'R' AminoAcid.

It is worthy to note that very few research efforts used the *DNA* characters as a different representation of the corresponding image pixels for the purpose of image compression. Nevertheless, the more complex representation of an image as a *protein* sequence has not yet been introduced by any research in the literature.

III. THE PROPOSED ALGORITHM: Image-as-Protein (IaP)

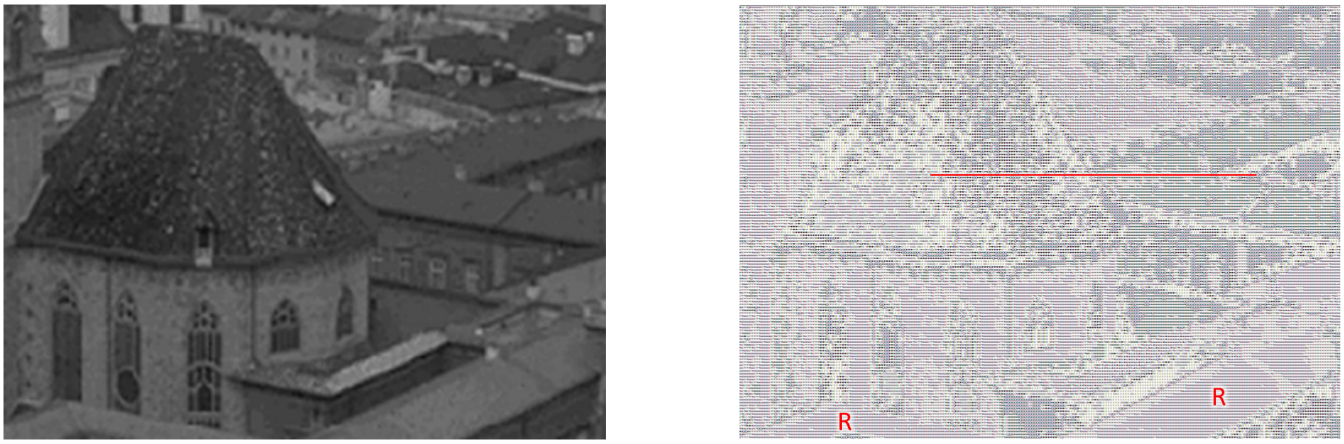
This section details the proposed lossy image compression algorithm with all its levels. It basically depends on compressing a given grayscale image by encoding its pixels into a *DNA* sequence which in turn is translated into a sequence of characters similar to the biological *protein* sequences (Figure 3). Table 1 shows the analogy between the biological *protein* translation process and proposed (*IaP*) compression algorithm.

A. THE COMPRESSION ALGORITHM

Figure 3 depicts the proposed multilevel lossy image compression framework. The compression process is divided into

Algorithm 1 The *IaP* Compression Algorithm

```
Input: An $(m*n)$ sequence of grayscale values $\langle p_1, p_2, \ldots, p_{m*n}\rangle$ corresponding to pixels of rows of input image img with
       dimensions $m$ rows and $n$ columns
Output: Compressed image file (img.IaP)
$DNA\_patterns \leftarrow Patterns\langle AAAA_0, AAAC_1, \ldots, TTTG_{254}, TTTT_{255}\rangle$
$DNA\_seq \leftarrow empty\_string$
for $i \leftarrow 1$ to $m$ do
    for $j \leftarrow 1$ to $n$ do
        $pxl\_value \leftarrow img[i, j]$
        $DNA\_seq \leftarrow DNA\_seq + DNA\_patterns[pxl\_value]$
$RNA\_seq \leftarrow convertDNAtoRNA()$
$RNA\_seq \leftarrow Padding\_RNA\_seq(RNA\_seq, 12)$
$(Acids, Codons, CodonBinaryCodes) \leftarrow loadCodonsTable()$
$OriginalAAs \leftarrow empty\_string$
$CodonBits \leftarrow empty\_string$
for $offset \leftarrow 1$ to $length(RNA\_seq)$ do
    $originalCodon = RNA\_seq[offset : offset + 3]$
    $offset \leftarrow offset + 4$
    $acid \leftarrow getAcidOfCodon(Acids, Codons, originalCodon)$
    $codonBinCode \leftarrow getBinaryCodeOfCodon(Codons, CodonBinaryCodes, originalCodon)$
    $OriginalAAs \leftarrow OriginalAAs + acid$
    $CodonBits \leftarrow CodonBits + codonBinaryCode$
$(img.IaP\_compressed\_file) \leftarrow writeSequencesToFile(OriginalAAs, CodonBits)$
return img.IaP_compressed_file
```



(a) Part of the original "townview" image viewed by image viewer. (https://ccia.ugr.es/cvg/CG/base.htm)

(b) Screenshot of characters in the *OriginalAAs protein* sequence forming the "townview" image. Sequence is exported as a matrix and viewed by the "Notepad++" text editor.

FIGURE 5. **The effect of the Selective Translation step over the *RNA* sequence representing the "townview" image. The red line in subfigure (b) refers to the *OriginalAAs protein* subsequence: "RRRRRPPRPPPPPPHHPLRQIIIISIISIIIMISSSSM QISS IIIIIRSIRTRSSSRRRRSHISSISSIMIMMMMMMQ PHIQMMQIMS QHHQIQQMIIMMQHHMQHQHMMIIIQMMMIMIM QRRQIIISSIIQIIHHHHHHHHHHHHHHHHHHHPHHHHHHHHPPHHHHHHHHPHHHHHHHHHHHQQQIII" that represents a part of the original image in subfigure (a). The red "R" characters in subfigure (b) represent some flat regions in the image with multiple lines containing the "R" character.**

six steps starting by converting the image pixels into a *DNA* sequence, and ending by encoding the corresponding *protein* sequence and its related information into output files. The *IaP* algorithm is summarized in Algorithm 1.

1) STEP 1: SUCCESSIVE ENCODING OF IMAGE PIXELS INTO *DNA* AND *RNA* SEQUENCES

The compression process starts by encoding the input image into a *DNA* sequence, by mapping every pixel of the input

TABLE 2. **The 256 *DNA* patterns (AAAA to TTTT) corresponding to the grayscale pixel values (0 to 256).**

Pixel Value		Corresponding
Decimal	Binary	*DNA* Pattern
0	00000000	AAAA
1	00000001	AAAC
2	00000010	AAAG
3	00000011	AAAT
4	00000100	AACA
..	....	....
108	01101100	CGTA
109	01101101	CGTC
110	01101110	CGTG
111	01101111	CGTT
..	....	....
252	11111100	TTTA
253	11111101	TTTC
254	11111110	TTTG
255	11111111	TTTT

image into a 4-character *DNA* pattern. Table 2 shows an example of mapping some possible image pixels to their corresponding *DNA* patterns. After that, the resulting *DNA* sequence is encoded into an *RNA* sequence which in turn is translated to a *protein* sequence. The translation process is achieved by mapping every 3 *RNA* characters (namely, a *Codon*) into one *AminoAcid (AA)* character in the generated *protein* sequence.

As an example, for a given grayscale images, the mapping of a pixel value to its corresponding *DNA* pattern can be done using bitwise operations over the 8-bits of the pixel value from left to right; assigning A to bits "00", C to bits "01", G to bits "10" and T to bits "11" respectively. However, generating the above *DNA* patterns once, and fetching them directly according to their corresponding pixel values is much faster. From the textual perspective, the conversion from *DNA* to *RNA* depends on replacing each 'T' character in the *DNA* to 'U' in the corresponding *RNA* sequence.

The encoded *DNA* sequence for a given image is four times bigger than the original size of the image itself. That's because every grayscale pixel value (1 byte) is flattened into one corresponding *DNA* pattern (4 characters). Nevertheless, the resulting *DNA* sequence is characterized by high similarity because the alphabet of a given *DNA* sequence contains only 4 characters A, C, G, T. The same merit applies for the *RNA* sequence of that *DNA* sequence as well. However, the neighborhood similarity between subsequent characters in the *DNA* sequence is not high enough compared to its corresponding *protein* sequence, which is formed from 20 different alphabetical characters A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y. That's because every *protein* character can be generated from multiple different *RNA Codons* (as shown in Figure 2). So, subsequent *RNA Codons* would result in subsequent clones of the same *protein* character, especially if the subsequent *RNA Codons* are similar. This fact coincides with image attributes where the neighbor pixels of many areas in an image often have values close to each other.

2) STEP 2: SELECTIVE TRANSLATION OF *RNA* SEQUENCE TO *protein* SEQUENCE

The main idea behind the proposed encoding process is gathering as much common features from the image pixels. Up to this moment, the similarity of the generated *protein* sequence is still not promising, and the compression ratio of the traditionally translated *protein* sequence is not expected to be high enough. For example, if the *DNA* subsequence for the six successive pixels (108, 109, 110, 110, 109, 108) is: CGTA**CGTC**CGTG**CGTG**CGTC**CGTA**. Thereafter, the corresponding *RNA* sequence would be: CGUA**CGUC**CGUG**CGUG**CGUC**CGUA**. The traditional translation of that *RNA* subsequence divides it into the following eight successive *Codons* CGU-**ACG**-UCC-**GUG**-CGU-**GCG**-UCC-**GUA**, which in turn are translated into the *protein* subsequence **RTSVRASV** through the mapping shown at Figure 2. Alternatively, to increase the text similarity, the authors divided the resulting *protein* sequence into two subsequences: *OriginalAAs* and *CumulativeAAs*. Because most of the neighbour pixels in flat regions of a given image are correlated to each other (with a small difference in their values), the difference between the *DNA* patterns of two successive pixels will almost be in the 4th right most character of the pattern. So, the same aforementioned *RNA* sequence can be partitioned by the biological Exons/Introns Splicing perspective as follows: CGU-**A**-CGU-**C**-CGU-**G**-CGU-**G**-CGU-**C**-CGU-**A**. Subsequently, while converting the *Codons* in a given *RNA* sequence into AAs in a *protein* sequence, it is better to first accumulate the left *Codon* (first three characters) of each *RNA* pattern: (CGU-CGU-CGU-CGU-CGU-CGU), resulting in the more homogeneous AAs sequence (R**R**R**R**R**R**), namely (*OriginalAAs*). From the biological perspective, this is analogous to accumulating the Exons that form the final *RNA* strand. After building the first part of the *protein* sequence (*OriginalAAs*), the second part (*CumulativeAAs*) is built by accumulating the 4th character of each *Codon* in the *RNA* sequence (A-C-G-G-C-A) that will generate the AAs subsequence (TA). Figure 4 illustrates how the *OriginalAAs* and *CumulativeAAs* are generated.

So, the resulting *protein* sequence is RRRRRR-TA instead of RTSVRASV that would result from the normal *protein* translation process. At the end, two *protein* subsequences are generated: *OriginalAAs* and *CumulativeAAs*. The idea of selective translation led to higher text similarity inside the *OriginalAAs* subsequence, especially for very long sequences resulting from an encoded image. Moreover, as will be discussed in **Step 6**, the general lossless text compression techniques are expected to give very promising results when applied to the *OriginalAAs* subsequence.

It is worthy to note that the size of the resultant *protein* sequence is identical to the size of the original uncompressed image. That's because every pixel is encoded into 4-characters in the *DNA* sequence, then every *Codon* picked from the *DNA* sequence is translated into one AA character in the resultant *OriginalAAs protein* sequence. Moreover, the *OriginalAAs* sequence can give

TABLE 3. ***DNA Codons* and their corresponding *CodonBits* for the different *AminoAcids*. The M and W *AminoAcids* do not need *CodonBits*. Three extra fake characters (J, X, and Z) are used to overcome the need for 3-bit binary codes for the L, R, and S *AminoAcids*: (L→J, R→X, and S→Z).**

Codon	*AminoAcid*	*CodonBits*	*Codon*	*AminoAcid*	*CodonBits*	*Codon*	*AminoAcid*	*CodonBits*	*Codon*	*AminoAcid*	*CodonBits*
GCA	A	00	CAC	H	0	CCA	P	00	UCG	S	10
GCC	A	01	CAU	H	1	CCC	P	01	UCU	S	11
GCG	A	10	AUA	I	00	CCG	P	10	ACA	T	00
GCU	A	11	AUC	I	01	CCU	P	11	ACC	T	01
UGC	C	0	AUU	I	10	CAA	Q	0	ACG	T	10
UGU	C	1	AAA	K	0	CAG	Q	1	ACU	T	11
GAC	D	0	AAG	K	1	AGA	$R \rightarrow X$	0	GUA	V	00
GAU	D	1	CUA	L	00	AGG	$R \rightarrow X$	1	GUC	V	01
GAA	E	0	CUC	L	01	CGA	R	00	GUG	V	10
GAG	E	1	CUG	L	10	CGC	R	01	GUU	V	11
UUC	F	0	CUU	L	11	CGG	R	10	UGG	W	
UUU	F	1	UUA	$L \rightarrow J$	0	CGU	R	11	UAC	Y	0
GGA	G	00	UUG	$L \rightarrow J$	1	AGC	$S \rightarrow Z$	0	UAU	Y	1
GGC	G	01	AUG	M		AGU	$S \rightarrow Z$	1	UAA	*	00
GGG	G	10	AAC	N	0	UCA	S	00	UAG	*	01
GGU	G	11	AAU	N	1	UCC	S	01	UGA	*	10

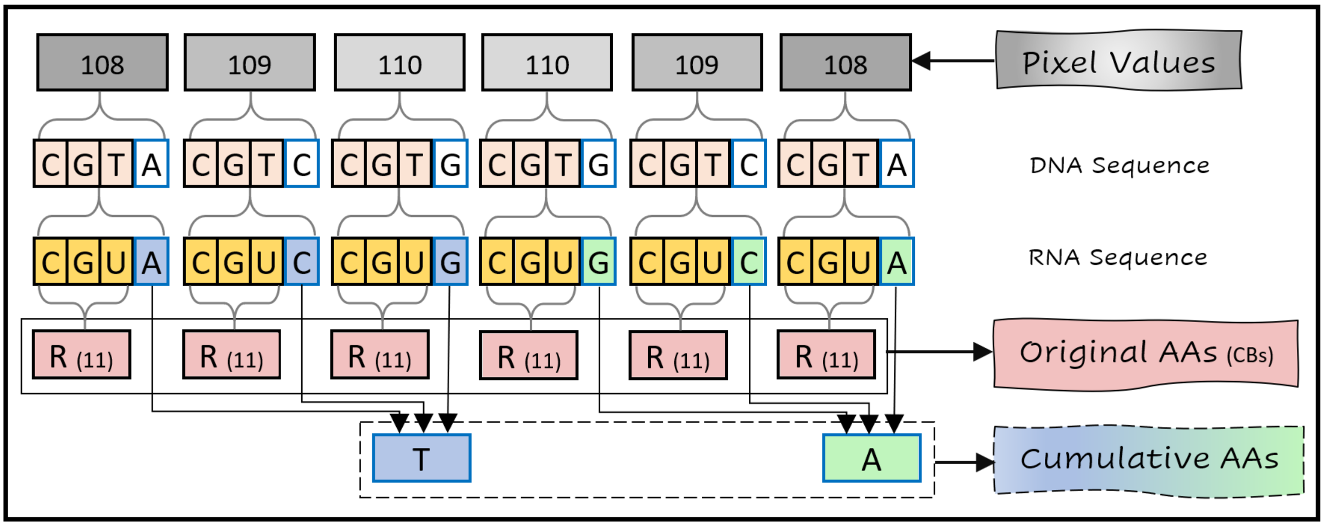


FIGURE 6. **The *CodonBits (CBs)* associated with the *OriginalAAs* sequence resulted from the illustrative image pixels in Figure 4. According to the CBs in Table 3, the 'R' *AminoAcid* results from 4 different codons. However, the CBs associated with the *RNA* codon 'CGU' and the 'R' *AminoAcid* are '11'. So, when reaching an 'R' *AminoAcid* during decompression, the bits '11' tells the exact codon that the 'R' *AminoAcid* came from.**

TABLE 4. **Grayscale datasets used for testing the proposed compression algorithm (*IaP*).**

Dataset Title	Image Count	Images Used	URL
1st Waterloo Dataset	12	Grayscale	http://links.uwaterloo.ca/Repository.html
2nd Waterloo Dataset	12	Grayscale	http://links.uwaterloo.ca/Repository.html
California	24	Grayscale	http://sipi.usc.edu/database/database.php?volume=misc
Textures	64	Grayscale	http://sipi.usc.edu/database/textures.zip

an abstract view of the original image (Figure 5). The red line in Figure 5.b refers to the subsequence: "RRRRR PPRPPPPPPHHPLRQIIIISIISIIIMI SSSSMQISSIIIIIIRSIR TRSSSRRRR SHISSISSIMIMMMMMQPHIQMMQIMSQ HHQIQQMIIMMQ HHMQHQHMMIIIQMMMIMIMQRR QIIISSIIQIIHHHHHHHHHHHHHHHHHHHPHHHHHHHP PHHHHHHHPHHHHHHHHHHQQQIII" of the *OriginalAAs* sequence encoding the image. That subsequence highlights the high similarity of the *OriginalAAs* sequence that is gained from the selective translation process.

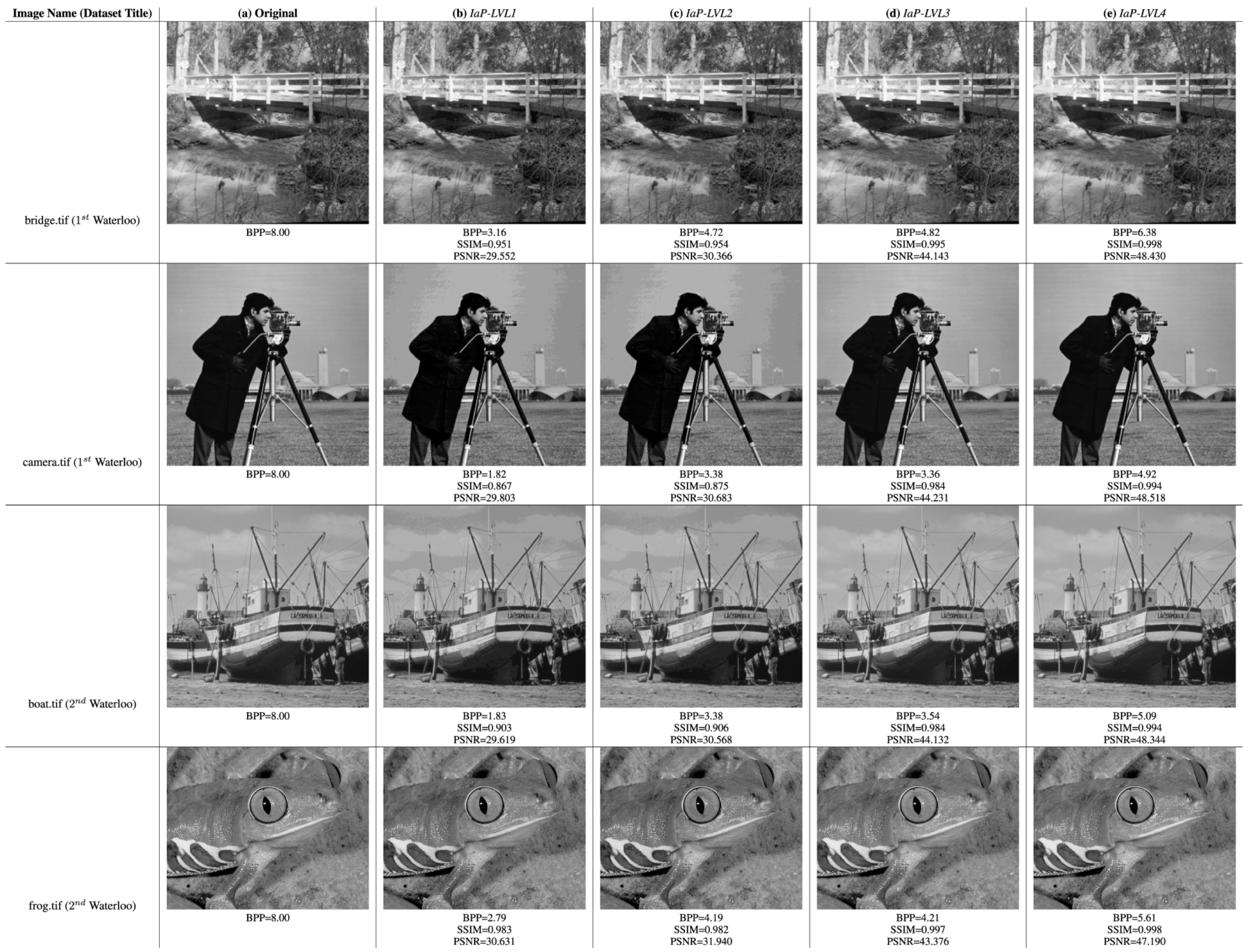


Image Name (Dataset Title)
(a) Original
(b) IaP-LVL1
(c) IaP-LVL2
(d) IaP-LVL3
(e) IaP-LVL4
bridge.tif (1st Waterloo)
BPP=8.00
BPP=3.16 SSIM=0.951 PSNR=29.552
BPP=4.72 SSIM=0.954 PSNR=30.366
BPP=4.82 SSIM=0.995 PSNR=44.143
BPP=6.38 SSIM=0.998 PSNR=48.430
camera.tif (1st Waterloo)
BPP=8.00
BPP=1.82 SSIM=0.867 PSNR=29.803
BPP=3.38 SSIM=0.875 PSNR=30.683
BPP=3.36 SSIM=0.984 PSNR=44.231
BPP=4.92 SSIM=0.994 PSNR=48.518
boat.tif (2nd Waterloo)
BPP=8.00
BPP=1.83 SSIM=0.903 PSNR=29.619
BPP=3.38 SSIM=0.906 PSNR=30.568
BPP=3.54 SSIM=0.984 PSNR=44.132
BPP=5.09 SSIM=0.994 PSNR=48.344
frog.tif (2nd Waterloo)
BPP=8.00
BPP=2.79 SSIM=0.983 PSNR=30.631
BPP=4.19 SSIM=0.982 PSNR=31.940
BPP=4.21 SSIM=0.997 PSNR=43.376
BPP=5.61 SSIM=0.998 PSNR=47.190

FIGURE 7. **Sample compression results of the four lossy image compression levels of *IaP*. In general, *IaP-LVL1* and *IaP-LVL3* introduce better compression ratios and acceptable quality when compared to *IaP-LVL2* and *IaP-LVL4*.**

3) STEP 3: PADDING OF IMAGE PIXELS

To guarantee the complete encoding of the entire image, the count of the image pixels should be divisible by 12. If not, the image pixels can be padded using fake pixel values. The reason behind the value "12" is to guarantee that the *DNA* sequence ends with three complete *DNA* patterns (12 characters) that should result into four *AminoAcid* characters: three *OriginalAAs protein* characters and one *CumulativeAAs protein* character.

4) STEP 4: ENCODING *protein* Sequence(S) INTO BINARY File(S)

According to the selected compression level, the image *protein* sequence(s) are encoded into binary file(s) that represent the compressed image. Although the size of the *OriginalAAs* sequence is identical to the size of the original image, the *OriginalAAs* sequence has very high text similarity, especially in flat regions. So, applying Huffman encoding to the *protein* sequence characters according to their frequencies, and then applying runlength encoding to the entire sequence achieves significant compression ratio compared to the original image file.

5) STEP 5: ENCODING *CodonBits* INTO A BINARY FILE (JUST FOR *IaP-LVL3* and *IaP-LVL4*)

As will be discussed in the next section, the decompression of the compressed *OriginalAAs protein* sequence (*IaP-LVL1*) results in a low-quality decompressed image. Moreover, the encoded *CumulativeAAs* (*IaP-LVL2*) decreases the compression ratio, and it do not noticeably contribute to the quality of the decompressed image.

As a step towards the enhancement of the of the quality of the decompressed image, extra binary bits (namely *CodonBits* (*CBs*)) have been encoded to improve the decompression quality. During decompression, *CodonBits* specify the exact *DNA Codon* used to produce an intended *protein AminoAcid* character during compression. Table 3 shows the 62 *CodonBits* corresponding to the 62 different *Codons* that generate

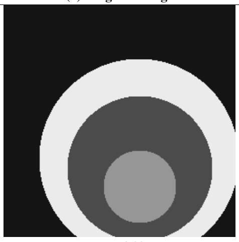
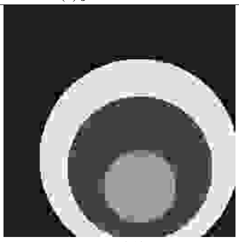
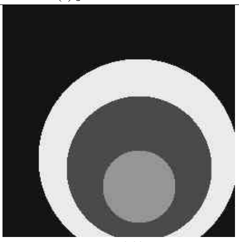
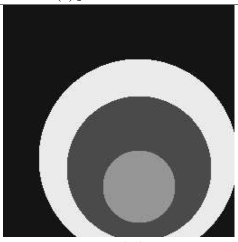
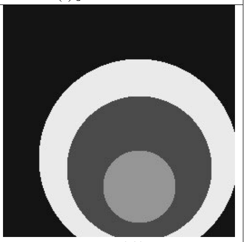
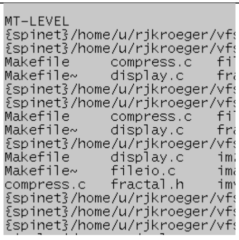
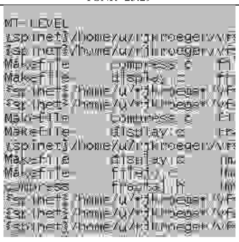
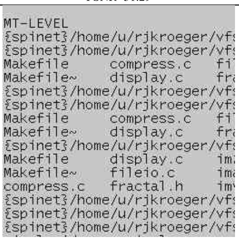
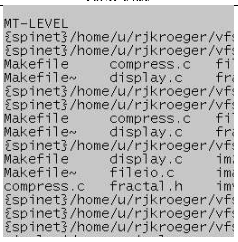
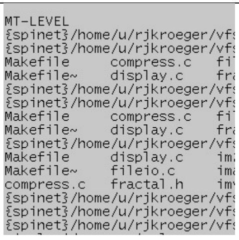
Image Name	(a) Original Image	(b) JPG-0% Version	(c) JPG-30% Version	(d) JPG-60% Version	(e) JPG-90% Version
circles.tif	 BPP=8.00	 BPP=0.18 SSIM=0.845 PSNR=23.29	 BPP=0.39 SSIM=0.950 PSNR=31.29	 BPP=0.53 SSIM=0.959 PSNR=34.33	 BPP=0.89 SSIM=0.991 PSNR=44.78
text.tif	 BPP=8.00	 BPP=0.59 SSIM=0.627 PSNR=14.75	 BPP=1.85 SSIM=0.860 PSNR=21.02	 BPP=2.71 SSIM=0.930 PSNR=26.21	 BPP=4.68 SSIM=0.991 PSNR=37.79

FIGURE 8. Visual comparison between different JPG quality percentages of the same image. (a) an original TIF image from the 1st Waterloo dataset, (b) its JPG compressed image with quality 0%, (c) its JPG compressed image with quality 30%, (d) its JPG compressed image with quality 60%, and (e) its JPG compressed image with quality 90%.

TABLE 5. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL1* vs. JPG-% for the (64 × 64) test images of the 1st Waterloo grayscale dataset. Dark gray rows represent images where *IaP-LVL1* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL1* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL1	JPG	LVL1	JPG	LVL1	JPG
bird.tif	64	90%	1.46	1.42	0.900	0.977	30.25	43.64
bridge.tif	64	88%	3.16	3.14	0.951	0.973	29.55	35.97
camera.tif	64	86%	1.82	1.84	0.867	0.964	29.80	37.84
(*) circles.tif	64	0%	0.14	0.18	0.987	0.845	32.36	23.29
(*) crosses.tif	64	0%	0.15	0.25	0.979	0.779	36.21	21.26
goldhill1.tif	64	89%	2.64	2.70	0.918	0.963	29.34	37.02
(*) horiz.tif	64	0%	0.06	0.16	0.978	0.816	35.50	25.44
lena1.tif	64	91%	2.44	2.40	0.899	0.975	30.04	40.50
montage.tif	64	87%	1.33	1.34	0.910	0.985	30.54	41.48
slope.tif	64	70%	0.56	0.56	0.831	0.981	29.08	41.57
squares.tif	64	95%	0.06	0.14	0.999	1.000	38.78	→100.00←
(*) text.tif	64	0%	0.18	0.59	0.999	0.627	30.56	14.75
Average BPP			1.17	1.23				
Average SSIM					0.935	0.907		
Average PSNR (Max. PSNR capped at 50.00)							31.83	34.40

18 *AminoAcids*. Both M and W *AminoAcids* do not need *CodonBits* as each of them is generated from a single *Codon*.

Figure 6 shows the *CodonBits* ‘**1**1**11**1**1**11**11**11’ corresponding to the illustrative pixel values depicted in Figure 4. According to the CBs in Table 3, the ‘R’ *AminoAcid* results from four different codons: CGA, CGC, CGG, and CGU. For example, the CBs associated with the *RNA* codon ‘CGU’ and the ‘R’ *AminoAcid* are ‘11’. So, when reaching an ‘R’ *AminoAcid* during decompression, the bits ‘11’ in the *CBs* sequence exactly the ‘CGU’ codon that the ‘R’ *AminoAcid* came from.

Although the extra encoded *CodonBits* increase the size of the resulting compressed image, it helps getting a decompressed image that is near enough to the original image. As will be discussed in section IV, *CumulativeAAs* (for *IaP-LVL2* and *IaP-LVL4*) of the compressed image can be safely neglected to increase the compression ratio with limited negative effect on the quality of the decompressed image.

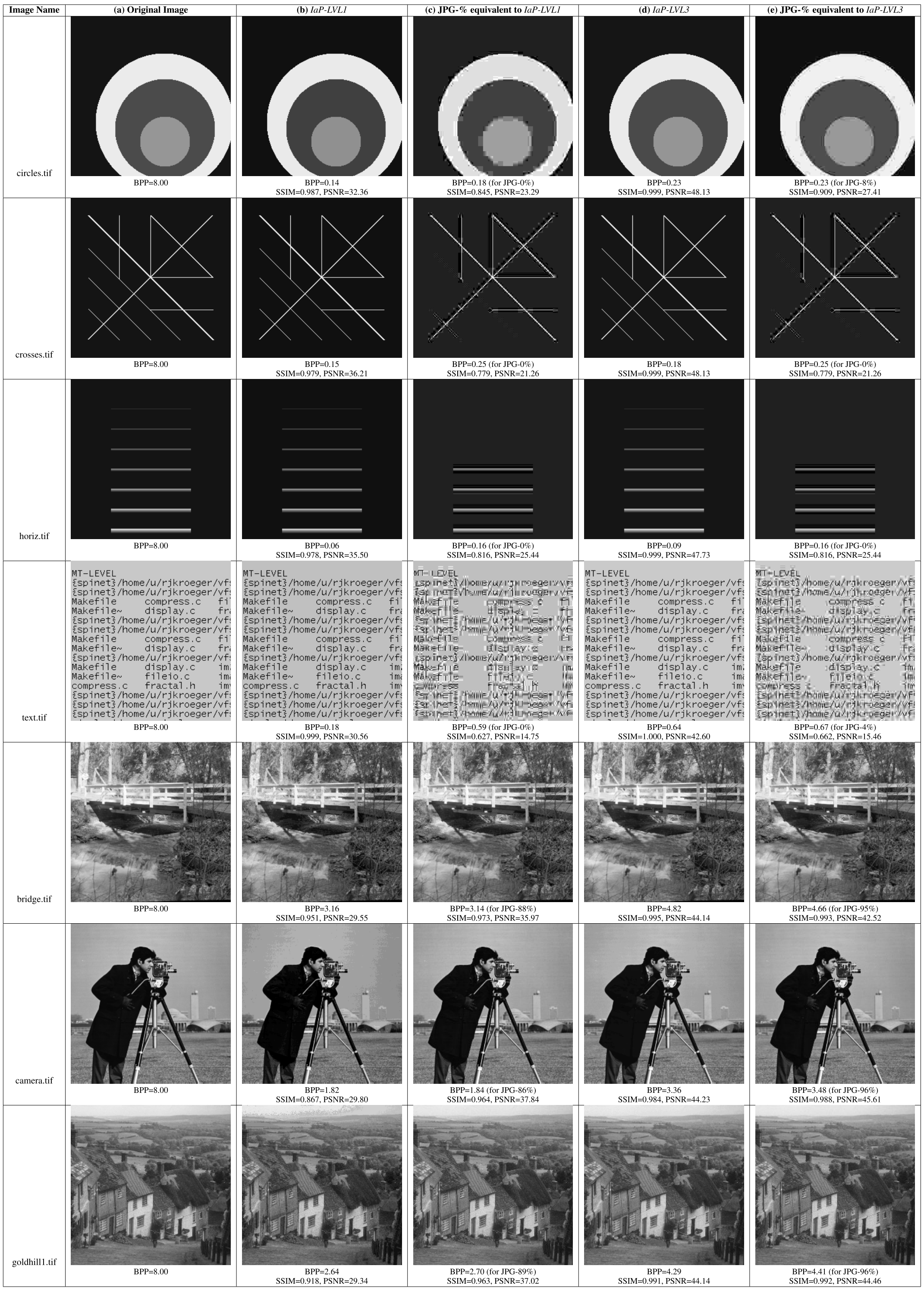


Image Name	(a) Original Image	(b) *IaP-LVL1*	(c) JPG-% equivalent to *IaP-LVL1*	(d) *IaP-LVL3*	(e) JPG-% equivalent to *IaP-LVL3*
circles.tif	BPP=8.00	BPP=0.14 SSIM=0.987, PSNR=32.36	BPP=0.18 (for JPG-0%) SSIM=0.845, PSNR=23.29	BPP=0.23 SSIM=0.999, PSNR=48.13	BPP=0.23 (for JPG-8%) SSIM=0.909, PSNR=27.41
crosses.tif	BPP=8.00	BPP=0.15 SSIM=0.979, PSNR=36.21	BPP=0.25 (for JPG-0%) SSIM=0.779, PSNR=21.26	BPP=0.18 SSIM=0.999, PSNR=48.13	BPP=0.25 (for JPG-0%) SSIM=0.779, PSNR=21.26
horiz.tif	BPP=8.00	BPP=0.06 SSIM=0.978, PSNR=35.50	BPP=0.16 (for JPG-0%) SSIM=0.816, PSNR=25.44	BPP=0.09 SSIM=0.999, PSNR=47.73	BPP=0.16 (for JPG-0%) SSIM=0.816, PSNR=25.44
text.tif	BPP=8.00	BPP=0.18 SSIM=0.999, PSNR=30.56	BPP=0.59 (for JPG-0%) SSIM=0.627, PSNR=14.75	BPP=0.64 SSIM=1.000, PSNR=42.60	BPP=0.67 (for JPG-4%) SSIM=0.662, PSNR=15.46
bridge.tif	BPP=8.00	BPP=3.16 SSIM=0.951, PSNR=29.55	BPP=3.14 (for JPG-88%) SSIM=0.973, PSNR=35.97	BPP=4.82 SSIM=0.995, PSNR=44.14	BPP=4.66 (for JPG-95%) SSIM=0.993, PSNR=42.52
camera.tif	BPP=8.00	BPP=1.82 SSIM=0.867, PSNR=29.80	BPP=1.84 (for JPG-86%) SSIM=0.964, PSNR=37.84	BPP=3.36 SSIM=0.984, PSNR=44.23	BPP=3.48 (for JPG-96%) SSIM=0.988, PSNR=45.61
goldhill1.tif	BPP=8.00	BPP=2.64 SSIM=0.918, PSNR=29.34	BPP=2.70 (for JPG-89%) SSIM=0.963, PSNR=37.02	BPP=4.29 SSIM=0.991, PSNR=44.14	BPP=4.41 (for JPG-96%) SSIM=0.992, PSNR=44.46

FIGURE 9. Visual comparison between the quality of (a) an original TIF image from the 1st Waterloo dataset, (b) its *IaP-LVL1* compressed image, (c) its equivalent JPG-% compressed image, (d) its *IaP-LVL3* compressed image, and (e) its equivalent JPG-% compressed image, based on equal or close *BPP* values between *IaP* levels and JPG quality percentage as illustrated in Tables 5 and 6.

6) STEP 6: COMPRESSING *IaP*'s OUTPUT FILES

In this step, the application of lossless text compression techniques, such as Huffman Encoding [2, Chapter 3] and Run Length Encoding [2, p. 205], to compress the *IaP*'s output files is expected to give very promising compression ratio. Moreover, many general-purpose compression

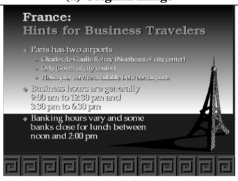
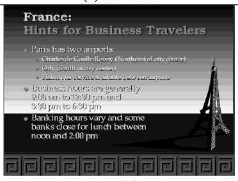
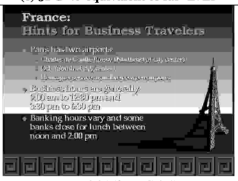
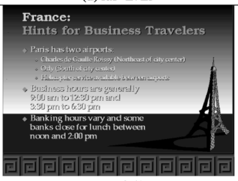
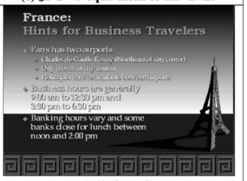
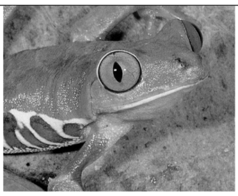
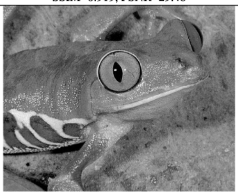
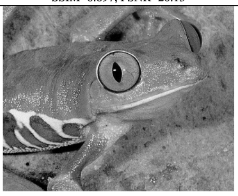
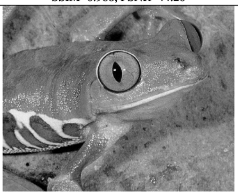
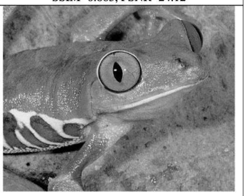
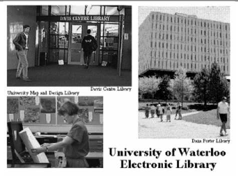
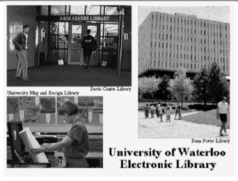
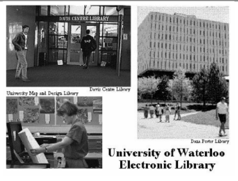
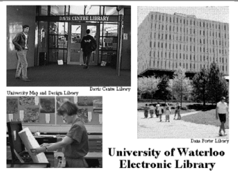
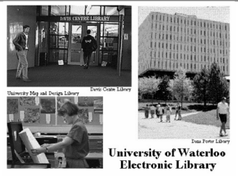
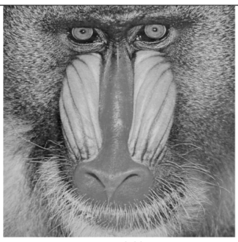
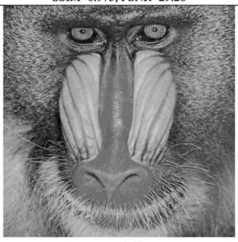
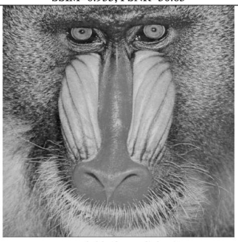
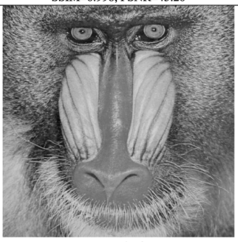
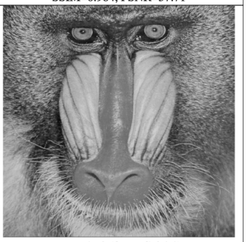
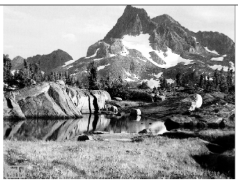
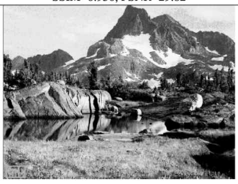
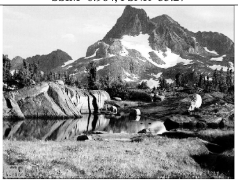
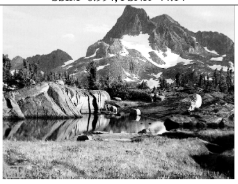
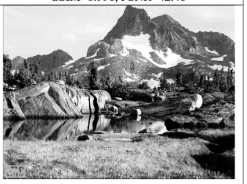
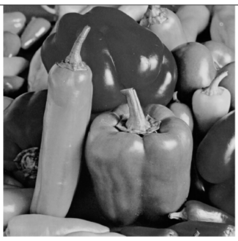
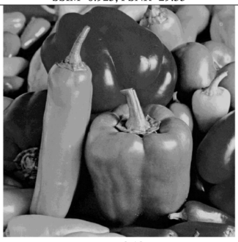
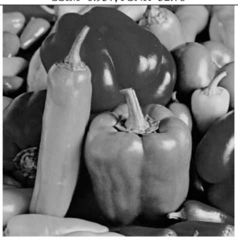
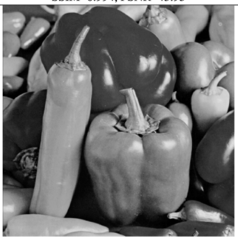
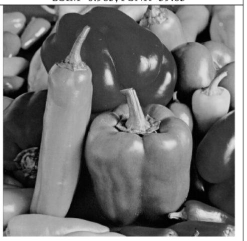
Image Name	(a) Original Image	(b) *IaP-LVL1*	(c) JPG-% equivalent to *IaP-LVL1*	(d) *IaP-LVL3*	(e) JPG-% equivalent to *IaP-LVL3*
france.tif	 BPP=8.00	 BPP=0.25 SSIM=0.919, PSNR=29.48	 BPP=0.24 (for JPG-3%) SSIM=0.697, PSNR=20.13	 BPP=0.51 SSIM=0.988, PSNR=44.20	 BPP=0.52 (for JPG-15%) SSIM=0.863, PSNR=24.12
frog.tif	 BPP=8.00	 BPP=2.79 SSIM=0.983, PSNR=30.63	 BPP=2.76 (for JPG-83%) SSIM=0.920, PSNR=32.12	 BPP=4.21 SSIM=0.997, PSNR=43.38	 BPP=4.43 (for JPG-94%) SSIM=0.988, PSNR=40.80
library.tif	 BPP=8.00	 BPP=2.35 SSIM=0.975, PSNR=29.26	 BPP=2.36 (for JPG-75%) SSIM=0.935, PSNR=30.63	 BPP=3.58 SSIM=0.998, PSNR=45.20	 BPP=3.63 (for JPG-90%) SSIM=0.984, PSNR=37.71
mandrill.tif	 BPP=8.00	 BPP=2.99 SSIM=0.950, PSNR=29.82	 BPP=3.02 (for JPG-87%) SSIM=0.964, PSNR=35.27	 BPP=4.76 SSIM=0.994, PSNR=44.14	 BPP=4.73 (for JPG-95%) SSIM=0.991, PSNR=42.40
mountain.tif	 BPP=8.00	 BPP=3.31 SSIM=0.925, PSNR=29.33	 BPP=3.27 (for JPG-83%) SSIM=0.957, PSNR=32.78	 BPP=4.72 SSIM=0.994, PSNR=43.93	 BPP=4.71 (for JPG-93%) SSIM=0.985, PSNR=39.83
peppers2.tif	 BPP=8.00	 BPP=2.19 SSIM=0.865, PSNR=30.30	 BPP=2.13 (for JPG-91%) SSIM=0.952, PSNR=39.56	 BPP=3.76 SSIM=0.981, PSNR=44.13	 BPP=3.85 (for JPG-97%) SSIM=0.989, PSNR=46.63

FIGURE 10. **Visual comparison between the quality of (a) an original TIF image from the 2nd Waterloo dataset, (b) its *IaP-LVL1* compressed image, (c) its equivalent JPG-% compressed image, (d) its *IaP-LVL3* compressed image, and (e) its equivalent JPG-% compressed image, based on equal or close *BPP* values between *IaP* levels and JPG quality percentage as illustrated in Tables 7 and 8.**

standards, including GZIP, BZIP2, and LZMA, would be used to encode the *IaP*'s output files.

Back to Figure 3, it demonstrates the four different *IaP* compression levels for a given image are formed from different combinations of the three (OriginalAAs, CumulativeAAs, CodonBits) sequences resulting from that image. If a given image ***img*** is compressed by *IaP-LVL1*, its ***img.IaP*** compressed file should only contain its *OriginalAAs* sequence. Whereas, if ***img*** is compressed by *IaP-LVL2*, its ***img.IaP*** compressed file should contain both its *OriginalAAs* and *CumulativeAAs* sequences. Alternatively, compressing ***img*** by *IaP-LVL3* means that both its *OriginalAAs* and *CodonBits*

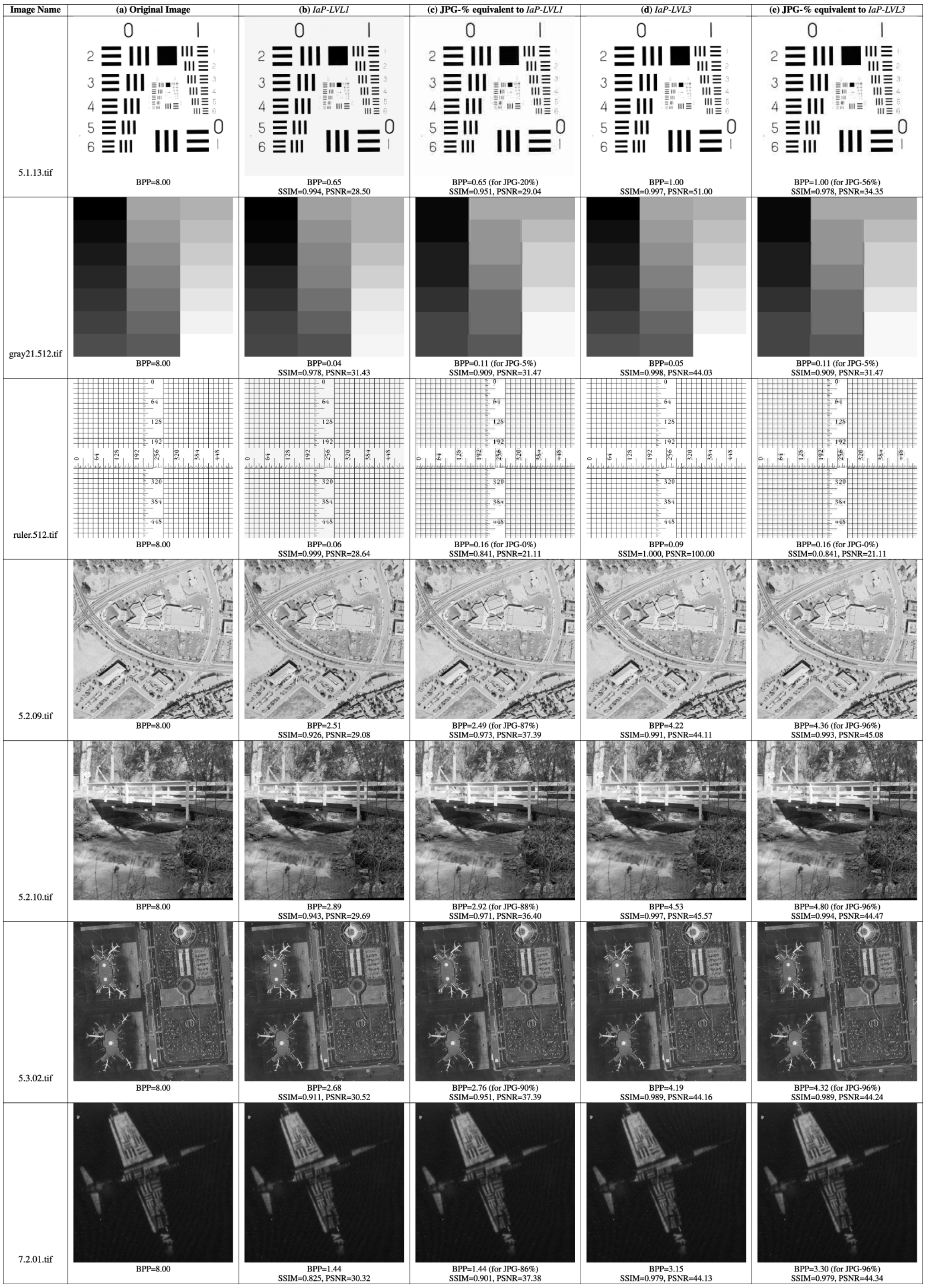


Image Name	(a) Original Image	(b) *IaP-LVL1*	(c) JPG-% equivalent to *IaP-LVL1*	(d) *IaP-LVL3*	(e) JPG-% equivalent to *IaP-LVL3*
5.1.13.tif	BPP=8.00	BPP=0.65 SSIM=0.994, PSNR=28.50	BPP=0.65 (for JPG-20%) SSIM=0.951, PSNR=29.04	BPP=1.00 SSIM=0.997, PSNR=51.00	BPP=1.00 (for JPG-56%) SSIM=0.978, PSNR=34.35
gray21.512.tif	BPP=8.00	BPP=0.04 SSIM=0.978, PSNR=31.43	BPP=0.11 (for JPG-5%) SSIM=0.909, PSNR=31.47	BPP=0.05 SSIM=0.998, PSNR=44.03	BPP=0.11 (for JPG-5%) SSIM=0.909, PSNR=31.47
ruler.512.tif	BPP=8.00	BPP=0.06 SSIM=0.999, PSNR=28.64	BPP=0.16 (for JPG-0%) SSIM=0.841, PSNR=21.11	BPP=0.09 SSIM=1.000, PSNR=100.00	BPP=0.16 (for JPG-0%) SSIM=0.0.841, PSNR=21.11
5.2.09.tif	BPP=8.00	BPP=2.51 SSIM=0.926, PSNR=29.08	BPP=2.49 (for JPG-87%) SSIM=0.973, PSNR=37.39	BPP=4.22 SSIM=0.991, PSNR=44.11	BPP=4.36 (for JPG-96%) SSIM=0.993, PSNR=45.08
5.2.10.tif	BPP=8.00	BPP=2.89 SSIM=0.943, PSNR=29.69	BPP=2.92 (for JPG-88%) SSIM=0.971, PSNR=36.40	BPP=4.53 SSIM=0.997, PSNR=45.57	BPP=4.80 (for JPG-96%) SSIM=0.994, PSNR=44.47
5.3.02.tif	BPP=8.00	BPP=2.68 SSIM=0.911, PSNR=30.52	BPP=2.76 (for JPG-90%) SSIM=0.951, PSNR=37.39	BPP=4.19 SSIM=0.989, PSNR=44.16	BPP=4.32 (for JPG-96%) SSIM=0.989, PSNR=44.24
7.2.01.tif	BPP=8.00	BPP=1.44 SSIM=0.825, PSNR=30.32	BPP=1.44 (for JPG-86%) SSIM=0.901, PSNR=37.38	BPP=3.15 SSIM=0.979, PSNR=44.13	BPP=3.30 (for JPG-96%) SSIM=0.979, PSNR=44.34

FIGURE 11. Visual comparison between the quality of (a) an original TIF image from the California dataset, (b) its *IaP-LVL1* compressed image, (c) its equivalent JPG-% compressed image, (d) its *IaP-LVL3* compressed image, and (e) its equivalent JPG-% compressed image, based on equal or close *BPP* values between *IaP* levels and JPG quality percentage as illustrated in Tables 9 and 10.

sequences are be included in its ***img.IaP*** compressed file. Finally, compressing ***img*** by *IaP-LVL4* implies that all its three sequences are included in its ***img.IaP*** compressed file.

B. THE PROPOSED DECOMPRESSION ALGORITHM

The decompression process should apply all the steps of the compression process in backward. First, the binary

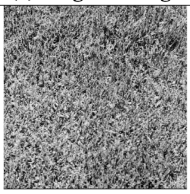
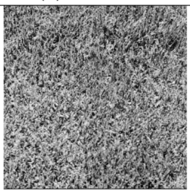
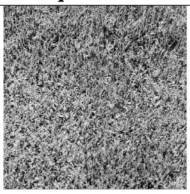
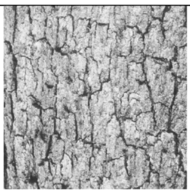
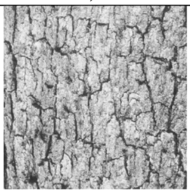
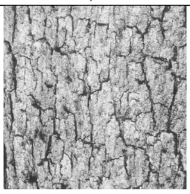
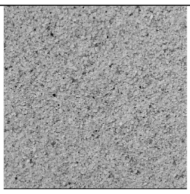
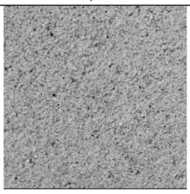
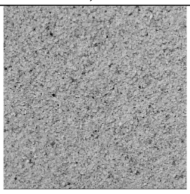
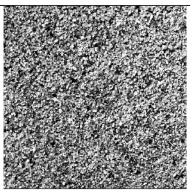
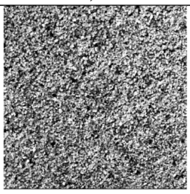
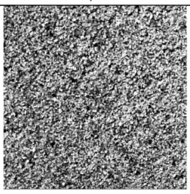
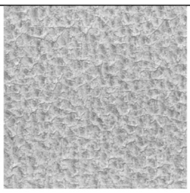
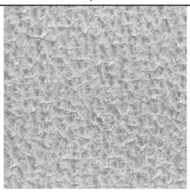
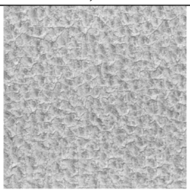
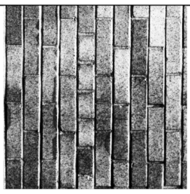
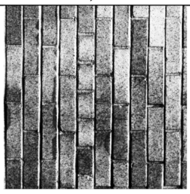
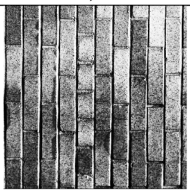
Image Name	(a) Original Image	(b) *IaP-LVL3*	(c) JPG-% equivalent to *IaP-LVL3*
1.1.01.tif	 BPP=8.00	 BPP=5.58 SSIM=0.999, PSNR=44.15	 BPP=5.90 (for JPG-95%) SSIM=0.999, PSNR=42.43
1.3.02.tif	 BPP=8.00	 BPP=4.55 SSIM=0.995, PSNR=44.14	 BPP=4.79 (for JPG-96%) SSIM=0.995, PSNR=44.29
1.2.03.tif	BPP=8.00	BPP=5.87 SSIM=0.999, PSNR=43.72	BPP=6.14 (for JPG-94%) SSIM=0.999, PSNR=40.86
1.1.07.tif	 BPP=8.00	 BPP=4.57 SSIM=0.996, PSNR=44.15	 BPP=4.80 (for JPG-95%) SSIM=0.994, PSNR=42.37
1.2.07.tif	 BPP=8.00	 BPP=5.92 SSIM=1.000, PSNR=44.19	 BPP=6.18 (for JPG-93%) SSIM=0.999, PSNR=39.45
1.3.11.tif	 BPP=8.00	 BPP=4.11 SSIM=0.992, PSNR=44.14	 BPP=4.39 (for JPG-96%) SSIM=0.992, PSNR=44.22
1.2.12.tif	 BPP=8.00	 BPP=5.74 SSIM=0.999, PSNR=44.29	 BPP=5.94 (for JPG-94%) SSIM=0.998, PSNR=40.82

FIGURE 12. Visual comparison between the quality of (a) an original TIF image from the Textures dataset, (b) its *IaP-LVL3* compressed image, and (c) its equivalent JPG-% compressed image, based on equal or close *BPP* values between *IaP-LVL3* and JPG quality percentage as illustrated in Table 12.

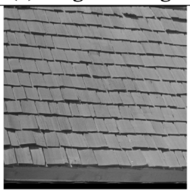
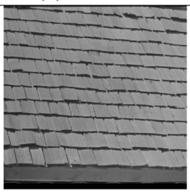
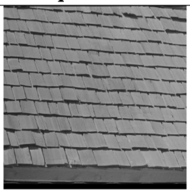
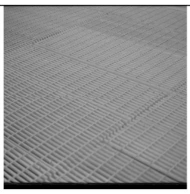
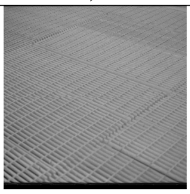
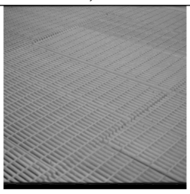
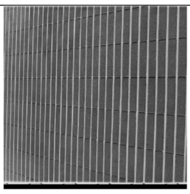
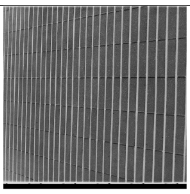
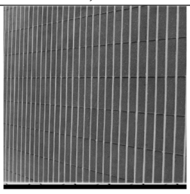
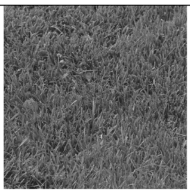
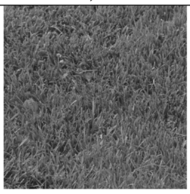
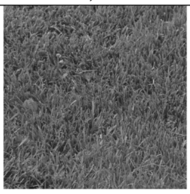
Image Name	(a) Original Image	(b) *IaP-LVL3*	(c) JPG-% equivalent to *IaP-LVL3*
1.4.03.tif	 BPP=8.00	 BPP=3.55 SSIM=0.986, PSNR=44.30	 BPP=3.75 (for JPG-96%) SSIM=0.986, PSNR=44.40
1.4.05.tif	BPP=8.00	BPP=3.31 SSIM=0.985, PSNR=44.30	BPP=3.58 (for JPG-96%) SSIM=0.985, PSNR=44.54
1.4.07.tif	 BPP=8.00	 BPP=3.49 SSIM=0.989, PSNR=44.26	 BPP=3.88 JPG-96% SSIM=0.989, PSNR=44.33
1.4.09.tif	 BPP=8.00	 BPP=4.31 SSIM=0.995, PSNR=44.27	 BPP=4.71 (for JPG-96%) SSIM=0.994, PSNR=44.33
1.4.10.tif	BPP=8.00	BPP=4.47 SSIM=0.995, PSNR=44.15	BPP=4.84 (for JPG-96%) SSIM=0.995, PSNR=44.25
1.5.07.tif	 BPP=8.00	 BPP=4.72 SSIM=0.996, PSNR=44.12	 BPP=5.05 (for JPG-96%) SSIM=0.996, PSNR=44.21

FIGURE 13. Visual comparison between the quality of (a) an original TIF image from the Textures dataset, (b) its *IaP-LVL3* compressed image, and (c) its equivalent JPG-% compressed image, based on equal or close *BPP* values between *IaP-LVL3* and JPG quality percentage as illustrated in Table 12.

compressed file is read and decoded using the Huffman and runlength information implemented in the last compression step to get the encoded *OriginalAAs protein* sequence. Next, a hashmap is built from the reversible mapping of Table 3. Every *protein* character is reversely mapped into its original *RNA Codon*. If a given image was compressed using *IaP-LVL1* and some *protein* character could be obtained from multiple *RNA Codons*, its first *Codon* is picked up. This action affects the quality of the decompressed image compared to the originally compressed image.

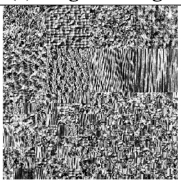
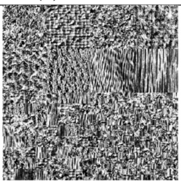
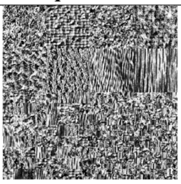
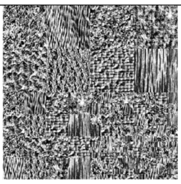
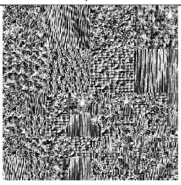
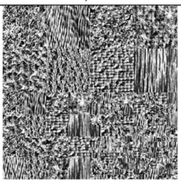
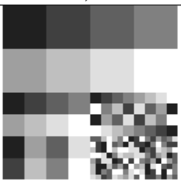
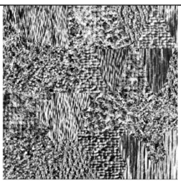
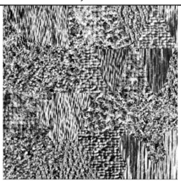
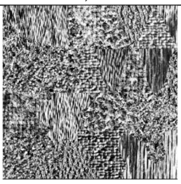
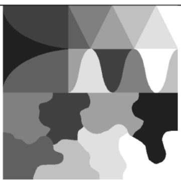
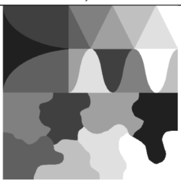
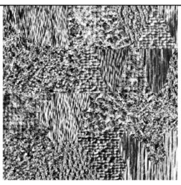
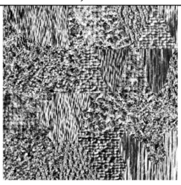
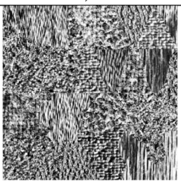
Image Name	(a) Original Image	(b) *IaP-LVL3*	(c) JPG-% equivalent to *IaP-LVL3*
texmos1.p512.tif	 BPP=8.00	 BPP=6.02 SSIM=1.000, PSNR=44.06	 BPP=6.19 (for JPG-94%) SSIM=0.999, PSNR=40.86
texmos2.p512.tif	 BPP=8.00	 BPP=3.93 SSIM=1.000, PSNR=43.94	 BPP=3.96 (for JPG-81%) SSIM=0.991, PSNR=31.49
texmos2.s512.tif	BPP=8.00	BPP=0.05 SSIM=1.000, PSNR=100.00	 BPP=0.11 (for JPG-0%) SSIM=0.995, PSNR=33.36
texmos3.p512.tif	 BPP=8.00	 BPP=5.97 SSIM=0.999, PSNR=43.92	 BPP=6.22 (for JPG-94%) SSIM=0.999, PSNR=40.85
texmos3.s512.tif	 BPP=8.00	 BPP=0.17 SSIM=1.000, PSNR=100.00	BPP=0.18 (for JPG-15%) SSIM=0.963, PSNR=33.92
texmos3b.p512.tif	 BPP=8.00	 BPP=5.99 SSIM=0.999, PSNR=43.90	 BPP=6.22 (for JPG-94%) SSIM=0.999, PSNR=40.85

FIGURE 14. Visual comparison between the quality of (a) an original TIF image from the Textures dataset, (b) its *IaP-LVL3* compressed image, and (c) its equivalent JPG-% compressed image, based on equal or close *BPP* values between *IaP-LVL3* and JPG quality percentage as illustrated in Table 12.

Alternatively, if the same image was compressed using *IaP-LVL3*, then the *CodonBits* stream should be decoded beforehand. In this case, if it is unknown from which *RNA Codon* a *protein* character is generated, then the *CodonBits* of that character are used to exactly know its original *RNA Codon*.

Every decoded *protein* character results in an *RNA Codon* that results in the three leftmost characters of some *DNA* pattern. So, given the original *protein* sequence represented by the *OriginalAAs* sequence, every three *protein* characters will result in three *RNA Codons* that can be used to build three *DNA* patterns.

After decoding three subsequent *RNA Codons* form the *OriginalAAs* sequence, if the image was compressed using *IaP-LVL2*, then one character from the *CumulativeAAs* sequence is decoded to obtain the fourth character for each

TABLE 6. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL3* vs. JPG-xx% for (64 × 64) test images of the 1st Waterloo grayscale dataset. Dark gray rows represent images where *IaP-LVL3* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL3* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL3	JPG	LVL3	JPG	LVL3	JPG
bird.tif	64	98%	3.00	3.23	0.979	0.994	44.05	50.50
bridge.tif	64	95%	4.82	4.66	0.995	0.993	44.14	42.52
camera.tif	64	96%	3.36	3.48	0.984	0.988	44.23	45.61
(*) circles.tif	64	8%	0.23	0.23	0.999	0.909	48.13	27.41
(*) crosses.tif	64	0%	0.18	0.25	0.999	0.779	48.13	21.26
goldhill1.tif	64	96%	4.29	4.41	0.991	0.992	44.14	44.46
(*) horiz.tif	64	0%	0.09	0.16	0.999	0.816	47.73	25.44
lena1.tif	64	97%	4.09	4.08	0.986	0.992	44.13	47.01
montage.tif	64	96%	2.44	2.40	0.987	0.995	44.80	48.03
slope.tif	64	94%	1.22	1.17	0.978	0.996	44.55	50.50
squares.tif	64	95%	0.09	0.14	0.999	1.000	43.27	→100.00←
(*) text.tif	64	4%	0.64	0.67	1.000	0.662	42.60	15.46
Average BPP			2.04	2.07				
Average SSIM					0.991	0.926		
Average PSNR (Max. PSNR capped at 50.00)							44.99	39.02

TABLE 7. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL1* vs. JPG-% for test images of the 2nd Waterloo grayscale dataset. Dark gray rows represent images where *IaP-LVL1* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL1* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL1	JPG	LVL1	JPG	LVL1	JPG
barb.tif	256	92%	2.42	2.40	0.909	0.981	30.05	41.76
boat.tif	256	89%	1.83	1.80	0.903	0.972	29.62	40.54
(*) france.tif	325.5	3%	0.25	0.24	0.919	0.697	29.48	20.13
frog.tif	302.01	83%	2.79	2.76	0.983	0.920	30.63	32.12
goldhill2.tif	256	90%	2.21	2.25	0.893	0.966	29.33	39.30
lena2.tif	256	91%	2.04	1.92	0.885	0.967	30.00	41.16
library.tif	159.5	75%	2.35	2.36	0.975	0.935	29.26	30.63
mandrill.tif	256	87%	2.99	3.02	0.950	0.964	29.82	35.27
mountain.tif	300	83%	3.31	3.27	0.925	0.957	29.33	32.78
peppers2.tif	256	91%	2.19	2.13	0.865	0.952	30.30	39.56
washsat.tif	256	88%	1.66	1.65	0.870	0.947	30.84	39.78
zelda.tif	256	93%	1.89	1.98	0.858	0.971	30.09	42.87
Average BPP			2.16	2.15				
Average SSIM					0.911	0.936		
Average PSNR							29.90	36.33

of the three previously decoded *DNA* patterns. If the given image was compressed using *IaP-LVL1* or *IaP-LVL3*, then the fourth character for each of the three previously decoded *DNA* patterns is randomly guessed. Finally, the subsequent *DNA* patterns are used to lookup their corresponding pixel values as illustrated in Table 2.

IV. RESULTS AND DISCUSSION

This section illustrates all the experimental results conducted on the proposed *IaP* algorithm, including the experimental environment, the implementation details, the attempted datasets and their compression results using *IaP* and JPEG compression algorithms based on quantitative and qualitative measurements.

A. EXPERIMENTAL ENVIRONMENT

The initial prototype of the proposed *IaP* algorithm is implemented in *Python 3*. The *CV2* OpenCV package is used to generate the JPG compressed images from the original TIF images for all the attempted datasets. In addition, the *SSIM*

TABLE 8. ***BPP, SSIM*, and *PSNR* of *IaP-LVL3* vs. JPG-% for test images of the 2nd Waterloo grayscale dataset. Dark gray rows represent images where *IaP-LVL3* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL3* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL3	JPG	LVL3	JPG	LVL3	JPG
barb.tif	256	97%	4.09	3.91	0.987	0.993	44.17	47.18
boat.tif	256	97%	3.54	3.64	0.984	0.991	44.13	47.10
(*) france.tif	325.5	15%	0.51	0.52	0.988	0.863	44.20	24.12
(*) frog.tif	302.01	94%	4.21	4.43	0.997	0.988	43.38	40.80
goldhill2.tif	256	96%	3.83	3.72	0.987	0.989	44.16	44.69
lena2.tif	256	97%	3.70	3.55	0.982	0.990	44.15	46.91
(*) library.tif	159.5	90%	3.58	3.63	0.998	0.984	45.20	37.71
mandrill.tif	256	95%	4.76	4.73	0.994	0.991	44.14	42.40
mountain.tif	300	93%	4.72	4.71	0.994	0.985	43.93	39.83
peppers2.tif	256	97%	3.76	3.85	0.981	0.989	44.13	46.63
washsat.tif	256	95%	2.98	2.83	0.976	0.977	43.10	43.70
zelda.tif	256	98%	3.50	3.67	0.980	0.994	44.15	49.81
Average BPP			3.60	3.60				
Average SSIM					0.987	0.978		
Average PSNR							44.07	42.57

measurements [12], [13] are calculated using the *SSIM* package. Moreover, the *numpy* package is used to build the decompressed images and calculated the *PSNR* values. The runtime was measured using the *timeit* package. Finally, the *zipfile* package was used by *IaP* to encode and decode the *protein* and *CodonBits* streams.

Regarding the general text compression phase applied to the *IaP*'s output files (Figure 3), the authors tried different combinations of lossless text compression algorithms, including Huffman-Encoding and Run-Length-Encoding. Moreover, many general-purpose compression standards, including GZIP, BZIP2, and LZMA, have been attempted to encode the *IaP*'s output files. However, LZMA gave the best compression ratio for the *IaP*'s output files, and BZIP2 comes next. So, the results introduced in this section are based on LZMA encoding of the *IaP*'s output files.

B. DATASETS

The proposed *IaP* algorithm is tested using the datasets listed in Table 4. *IaP* was run over the original grayscale images included in the 1st Waterloo, 2nd Waterloo, California, and Textures datasets. The reader can zoom in all the images included in the following figures to visually assess their quality. However, for more convenient comparison, these images have been uploaded as a supplementary ZIP file at the following link: (https://bit.ly/3BR0wGP). The supplementary ZIP file includes the original TIF images in PNG format, the JPG compressed images, the *IaP* compressed images in ZIP format, and the *IaP* decompressed images with *.IaP* extension.

C. QUANTITATIVE AND QUALITATIVE COMPARISON BETWEEN IaP AND JPG

This subsection starts by comparing the four compression levels of *IaP*. Next, a comparative study between the *IaP* and JPG lossy compression algorithms is introduced. Figure 7 compares the compression ratios and the quality of the four levels of *IaP*. Although the *Bits-Per-Pixel BPP* of *IaP-LVL2* approaches or exceeds the *BPP* of *IaP-LVL3*, the quality of the decompressed image (measured by *SSIM* and *PSNR*) is not significantly improved compared to *IaP-LVL1*. In general, the *BPP*, *SSIM*, and *PSNR* values of the four levels imply that *IaP-LVL2* and *IaP-LVL4* (that include the *CumulativeAAs* component) are not as effective as using the *CodonBits* component.

As shown in Figure 3, the *CumulativeAAs* sequence is the second component of any image compressed by *IaP-LVL2* and *IaP-LVL4*. However, based on the sample results of the four *IaP* levels shown in Figure 7, including the *CumulativeAAs* sequence in the image's compressed file increases its size with insignificant quality enhancement for the decompressed image in both levels. So, the *CumulativeAAs* sequence is ignored analogous to skipping the *Introns* while building the biological *protein* sequence. Consequently, only the results of *IaP-LVL1* and *IaP-LVL3* are considered in this section.

The size of the compressed image, represented by Bits-Per-Pixel (*BPP*), is the main criterion in comparing a specific *IaP* level to JPG. For a given image compressed by some *IaP* level, a corresponding JPG compressed image with the nearest *BPP* value. The quality percentage of the generated JPG image is represented by the expression (JPG-x%), where

TABLE 9. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL1* vs. JPG-% for test images of the California dataset. Dark gray rows represent images where *IaP-LVL1* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL1* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL1	JPG	LVL1	JPG	LVL1	JPG
5.1.09.tif	64	89%	2.39	2.36	0.901	0.935	29.86	37.26
5.1.10.tif	64	85%	2.89	2.84	0.959	0.973	28.79	35.10
5.1.11.tif	64	91%	1.34	1.37	0.822	0.976	24.49	43.89
5.1.12.tif	64	91%	1.75	1.74	0.831	0.982	26.22	42.23
5.1.13.tif	64	20%	0.65	0.65	0.994	0.951	28.50	29.04
5.1.14.tif	64	89%	2.61	2.64	0.933	0.978	30.02	38.59
5.2.08.tif	256	88%	1.89	1.88	0.907	0.969	30.05	39.86
5.2.09.tif	256	87%	2.51	2.49	0.926	0.973	29.08	37.39
5.2.10.tif	256	88%	2.89	2.92	0.943	0.971	29.69	36.40
5.3.01.tif	1024	91%	2.31	3.32	0.881	0.961	30.51	39.63
5.3.02.tif	1024	90%	2.68	2.76	0.911	0.951	30.52	37.39
7.1.01.tif	256	87%	1.87	1.82	0.893	0.952	30.11	38.39
7.1.02.tif	256	83%	0.89	0.88	0.860	0.949	29.05	40.31
7.1.03.tif	256	88%	1.98	2.01	0.902	0.940	29.89	37.79
7.1.04.tif	256	88%	1.90	1.86	0.898	0.955	29.98	39.12
7.1.05.tif	256	87%	2.38	2.38	0.934	0.952	29.70	30.05
7.1.06.tif	256	87%	2.45	2.38	0.930	0.953	29.91	36.14
7.1.07.tif	256	87%	2.25	2.30	0.927	0.947	29.54	36.38
7.1.08.tif	256	89%	1.81	1.78	0.877	0.936	30.72	38.79
7.1.09.tif	256	86%	2.03	2.05	0.921	0.946	29.65	36.71
7.1.10.tif	256	88%	1.90	1.91	0.903	0.955	29.87	38.95
7.2.01.tif	1024	86%	1.44	1.44	0.825	0.901	30.32	37.38
boat.512.tif	256	89%	2.26	2.20	0.906	0.955	29.67	38.71
gray21.512.tif	256	5%	0.04	0.11	0.978	0.909	31.43	31.47
(*) ruler.512.tif	256	0%	0.05	0.54	0.999	0.841	28.64	21.11
Average BPP			1.89	1.95				
Average SSIM					0.901	0.948		
Average PSNR							29.45	36.72

x is the quality percentage. A higher x value means higher *BPP* value and better compression quality (higher *SSIM* and *PSNR* values). After that, the *SSIM* and *PSNR* of that JPG image are calculated.Figure 8 illustrates the (JPG-%) quality percentage for different JPG versions of the same image. The *SSIM* and *PSNR* values of each version is written underneath.

Tables 5 to 12 respectively compare the performance of *IaP* and JPG algorithms over the 1^{st} *Waterloo*, 2^{nd} *Waterloo*, *California*, and *Textures* datasets according to the *BPP*, *SSIM*, and *PSNR* measurements. The *MSE* value between the pixels of the original and decompressed images is calculated using (Equation 1). The *PSNR* value [30] is calculated using (Equation 2) based on the obtained *MSE* value. The *PSNR* value of 100 implies that its *MSE* value is zero. The third column of each of these tables lists the quality percentage of JPG with the nearest *BPP* value for the specified *IaP* level. The next columns compare a specific *IaP* level with JPG with three measures respectively: *BPP*, *SSIM*, and *PSNR*. An (*) preceeding an image name in the following tables refers to a *IaP* compressed image with superior performance compared to its JPEG compressed version with close or equal *BPP* value. In that case, the image row is highlighted in dark gray. In this subsection, an algorithm with better performance for some image usually means that it resulted in a compressed image with smaller *BPP* value but higher quantitative measure(s) (*SSIM*, *PSNR*) for the decompressed image. Alternatively, Figures 9 to 14 visually compare the performance of *IaP* and JPG over some images from the same aforementioned datasets.

$$MSE(x,y) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij} - y_{ij})^2 \tag{1}$$

where:
x = original image
y = decompressed image
m = number of rows in original image
n = number of columns original image
x_{ij} = pixel value from original image
y_{ij} = pixel value from decompressed image

$$PSNR(x,y) = 10\log_{10}\left(\frac{[\max(\max(x), \max(y))]^2}{MSE(x,y)}\right) \tag{2}$$

TABLE 10. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL3* vs. JPG-% for test images of the California dataset. Dark gray rows represent images where *IaP-LVL3* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL3* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL3	JPG	LVL3	JPG	LVL3	JPG
5.1.09.tif	64	96%	4.11	4.04	0.987	0.987	44.15	44.25
5.1.10.tif	64	95%	4.74	4.80	0.996	0.994	44.13	42.51
5.1.11.tif	64	97%	2.86	2.62	0.979	0.990	44.45	48.41
5.1.12.tif	64	97%	3.06	3.07	0.983	0.993	44.27	48.30
(*) 5.1.13.tif	64	56%	1.00	1.00	0.997	0.978	51.00	34.35
5.1.14.tif	64	96%	4.26	4.27	0.992	0.994	44.15	44.95
5.2.08.tif	256	96%	3.52	3.41	0.986	0.989	44.22	45.26
5.2.09.tif	256	96%	4.22	4.36	0.991	0.993	44.11	45.08
5.2.10.tif	256	96%	4.53	4.80	0.997	0.994	45.57	44.47
5.3.01.tif	1024	96%	3.85	3.70	0.984	0.986	44.44	44.54
5.3.02.tif	1024	96%	4.19	4.32	0.989	0.989	44.16	44.24
7.1.01.tif	256	96%	3.50	3.61	0.987	0.987	44.40	44.75
7.1.02.tif	256	96%	2.43	2.39	0.979	0.980	44.15	45.83
7.1.03.tif	256	96%	3.66	3.77	0.985	0.986	43.66	44.45
7.1.04.tif	256	96%	3.53	3.54	0.983	0.987	43.70	44.72
7.1.05.tif	256	95%	4.09	4.00	0.991	0.988	43.96	42.49
7.1.06.tif	256	96%	4.12	4.01	0.991	0.988	43.92	42.47
7.1.07.tif	256	95%	4.03	3.91	0.990	0.986	43.83	42.47
7.1.08.tif	256	95%	3.13	2.99	0.980	0.973	43.63	42.90
7.1.09.tif	256	95%	3.79	3.70	0.990	0.985	44.21	42.61
7.1.10.tif	256	96%	3.56	3.61	0.986	0.988	43.72	44.65
7.2.01.tif	1024	96%	3.15	3.30	0.979	0.979	44.13	44.34
boat.512.tif	256	96%	3.95	3.77	0.986	0.988	43.91	44.61
(*) gray21.512.tif	256	5%	0.05	0.11	0.998	0.909	44.03	31.47
(*) ruler.512.tif	256	0%	0.09	0.54	1.000	0.841	→100.00←	21.11
Average BPP			3.34	3.35				
Average SSIM					0.988	0.978		
Average PSNR (Max. PSNR capped at 50.00)							44.64	42.61

Table 5 shows the compression results of *IaP-LVL1* and *JPG* for the 1st Waterloo dataset, whereas, Table 6 compares *IaP-LVL3* with *JPG* for the same dataset. Table 5 shows the four images "circles.tif", "crosses.tif", "horiz.tif", and "text.tif" where the *SSIM* and *PSNR* measures of *IaP-LVL1* are significantly better than JPG. For the "squares.tif" image, *IaP-LVL1* still has lower *BPP* value and competitive *SSIM* value. At last, JPG has better quality for the rest of images. In Table 6, the performance of *IaP-LVL3* is much better than JPG for the same four images "circles.tif", "crosses.tif", "horiz.tif", and "text.tif" based on equal or close *BPP* values. Moreover, the *IaP-LVL3* gains better performance compared to JPG for images "bridge.tif", "camera.tif", "goldhill1.tif".

By zooming in Figure 9, it is clear that JPG has bad quality for the first four images compared to the same decompressed images of both *IaP-LVL1* and *IaP-LVL3*. For the last three images of the same figure, although JPG has better performance than *IaP-LVL1*, both JPG and *IaP-LVL3* are competing for lower *BPP* and higher *SSIM* and *PSNR* values.

Table 7 shows the compression results of *IaP-LVL1* and *JPG* for the 2nd Waterloo dataset, whereas, Table 8 compares *IaP-LVL3* with *JPG* for the same dataset. As shown in Table 7, *IaP-LVL1* has much better performance for image "france.tif" compared to JPG. For the "frog.tif" and "library.tif" images, *IaP-LVL1* still has better performance compared to JPG. In Table 8, the performance of *IaP-LVL3* is noticeably better than JPG for images "france.tif", "frog.tif", and "library.tif" based on equal or close *BPP* values. Moreover, the performance of *IaP-LVL3* is competitive for images "mandrill.tif", "mountain.tif", "peppers2.tif". Zooming in images of Figure 10 reflects the same aforementioned inductions.

Table 9 shows the compression results of *IaP-LVL1* and *JPG* for the California grayscale dataset, whereas, Table 10 compares *IaP-LVL3* with *JPG* for the same dataset. As shown in Table 9, JPG has the best performance for most of the California images. However, *IaP-LVL1* has better performance for images "5.1.13.tif" and "gray21.512.tif". Moreover, *IaP-LVL1* has the best performance for image "ruler.512.tif". On the contrary, the performance of *IaP-LVL3*, as presented in Table 10, is very competitive to JPG for images with light gray entries. Moreover, *IaP-LVL3* has the best performance for images "5.1.13.tif", "gray21.512.tif",

TABLE 11. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL1* vs. JPG-% for test images of the Textures dataset. Dark gray rows represent images where *IaP-LVL1* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL1* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL1	JPG	LVL1	JPG	LVL1	JPG
1.1.01.tif	256	86%	3.85	3.88	0.989	0.992	29.71	34.64
1.2.01.tif	256	84%	4.35	4.31	0.991	0.994	29.25	33.10
1.3.01.tif	1024	87%	2.92	2.84	0.958	0.981	28.68	35.77
1.1.02.tif	256	87%	3.26	3.26	0.976	0.982	29.92	35.75
1.2.02.tif	256	86%	4.12	4.16	0.983	0.991	28.81	33.97
1.3.02.tif	1024	88%	2.78	2.82	0.922	0.968	28.50	36.03
1.1.03.tif	256	86%	3.24	3.30	0.982	0.983	29.91	34.83
1.2.03.tif	256	85%	4.29	4.30	0.991	0.994	29.29	33.35
1.3.03.tif	1024	89%	2.79	2.68	0.918	0.973	28.04	36.79
1.1.04.tif	256	84%	2.94	2.95	0.981	0.978	30.06	34.12
1.2.04.tif	256	83%	4.39	4.41	0.994	0.995	29.26	32.12
1.3.04.tif	1024	87%	3.44	3.39	0.967	0.973	30.32	34.59
1.1.05.tif	256	85%	2.84	2.80	0.964	0.956	29.83	34.01
1.2.05.tif	256	83%	4.40	4.41	0.988	0.989	29.39	31.96
1.3.05.tif	1024	86%	3.34	3.27	0.978	0.979	29.96	34.07
1.1.06.tif	256	83%	3.37	3.38	0.988	0.981	29.88	32.66
1.2.06.tif	256	83%	4.50	4.48	0.993	0.993	29.28	31.96
1.3.06.tif	1024	86%	2.76	2.71	0.954	0.966	28.80	35.21
1.1.07.tif	256	84%	2.70	2.75	0.968	0.967	29.64	34.46
1.2.07.tif	256	82%	4.35	4.34	0.993	0.992	29.48	31.56
1.3.07.tif	1024	88%	2.42	2.42	0.877	0.952	28.11	36.63
1.1.08.tif	256	82%	1.25	1.26	0.906	0.968	29.50	40.07
1.2.08.tif	256	88%	4.25	4.27	0.990	0.995	29.17	34.97
1.3.08.tif	1024	85%	1.77	1.73	0.885	0.886	29.44	35.44
1.1.09.tif	256	87%	2.11	2.07	0.945	0.983	29.90	39.69
1.2.09.tif	256	90%	4.16	4.24	0.987	0.996	28.96	36.76
1.3.09.tif	1024	92%	2.66	2.54	0.783	0.959	27.02	39.19
1.1.10.tif	256	84%	2.41	2.44	0.972	0.982	29.85	36.50
1.2.10.tif	256	83%	3.78	3.81	0.990	0.993	29.14	33.19
1.3.10.tif	1024	89%	2.26	2.25	0.830	0.969	27.51	39.77
1.1.11.tif	256	84%	2.30	2.26	0.958	0.962	29.87	35.48
1.2.11.tif	256	82%	4.02	3.99	0.992	0.991	29.84	31.77
1.3.11.tif	1024	86%	2.28	2.23	0.917	0.944	29.15	35.74
1.1.12.tif	256	86%	2.64	2.60	0.947	0.949	29.66	35.06
1.2.12.tif	256	85%	4.20	4.15	0.985	0.986	29.78	32.54
1.3.12.tif	1024	88%	2.59	2.51	0.819	0.950	26.94	36.45
1.1.13.tif	256	87%	2.99	2.95	0.966	0.979	29.98	36.10
1.2.13.tif	256	85%	3.86	3.77	0.979	0.987	29.34	33.73
1.3.13.tif	1024	87%	2.48	2.41	0.928	0.947	29.22	35.76
1.4.01.tif	1024	82%	1.71	1.61	0.919	0.926	29.65	35.96
1.4.02.tif	1024	85%	1.87	1.87	0.920	0.933	29.83	36.27
1.4.03.tif	1024	87%	1.97	1.89	0.904	0.918	30.47	36.75
1.4.04.tif	1024	85%	1.25	1.23	0.874	0.907	29.97	37.55
1.4.05.tif	1024	85%	1.64	1.61	0.887	0.920	29.91	36.57
1.4.06.tif	1024	88%	1.92	1.91	0.902	0.936	29.85	37.41
1.4.07.tif	1024	85%	1.84	1.77	0.914	0.914	30.19	35.74
1.4.08.tif	1024	87%	1.61	1.54	0.892	0.942	29.77	38.28
1.4.09.tif	1024	86%	2.65	2.66	0.955	0.947	30.05	34.36
1.4.10.tif	1024	87%	2.77	2.71	0.955	0.970	29.90	35.92
1.4.11.tif	1024	87%	2.93	2.93	0.958	0.982	29.07	36.06
1.4.12.tif	1024	87%	2.74	2.74	0.948	0.977	29.02	36.04
1.5.01.tif	256	90%	1.92	1.91	0.906	0.939	30.72	38.92
1.5.02.tif	256	48%	2.18	2.19	0.994	0.988	28.97	28.21
1.5.03.tif	256	88%	2.77	2.75	0.921	0.969	28.58	36.52
1.5.04.tif	256	85%	2.69	2.66	0.865	0.969	26.17	35.33
1.5.05.tif	256	90%	2.34	2.29	0.784	0.967	26.74	39.29
1.5.06.tif	256	81%	1.25	1.29	0.875	0.873	29.19	35.64
1.5.07.tif	256	88%	3.02	3.04	0.964	0.974	30.04	36.04
texmos1.p512.tif	86%	256	4.44	4.46	0.991	0.995	29.19	33.87
(*) texmos2.p512.tif	50%	256	2.41	2.43	0.991	0.967	29.18	25.54
texmos2.s512.tif	0%	256	0.03	0.11	0.989	0.995	26.60	33.36
texmos3.p512.tif	85%	256	4.39	4.34	0.991	0.994	29.21	33.27
texmos3.s512.tif	0%	256	0.09	0.12	0.991	0.923	26.80	29.10
texmos3b.p512.tif	85%	256	4.41	4.34	0.991	0.994	29.20	33.24
Average BPP			2.84	2.82				
Average SSIM					0.943	0.966		
Average PSNR							29.20	35.02

TABLE 12. ***BPP*, *SSIM*, and *PSNR* of *IaP-LVL3* vs. JPG-% for test images of the Textures dataset. Dark gray rows represent images where *IaP-LVL3* is superior compared to JPG, whereas, light gray rows represent images where *IaP-LVL3* was competitive to JPG with equal or close *BPP* value.**

Image	Size	JPG	BPP		SSIM (0→1)		PSNR (dB)	
	(KB)	Quality %	LVL3	JPG	LVL3	JPG	LVL3	JPG
1.1.01.tif	256	95%	5.58	5.90	0.999	0.999	44.15	42.43
1.2.01.tif	256	93%	5.93	5.91	1.000	0.999	44.31	39.56
1.3.01.tif	1024	95%	4.69	4.65	0.997	0.996	44.15	42.30
1.1.02.tif	256	95%	5.05	5.07	0.997	0.996	44.15	42.39
1.2.02.tif	256	94%	5.69	5.85	0.999	0.998	44.03	40.84
1.3.02.tif	1024	96%	4.55	4.79	0.995	0.995	44.14	44.29
1.1.03.tif	256	95%	5.04	5.15	0.998	0.997	44.15	42.44
1.2.03.tif	256	94%	5.87	6.14	0.999	0.999	43.72	40.86
1.3.03.tif	1024	96%	4.54	4.47	0.995	0.995	44.14	44.23
1.1.04.tif	256	94%	4.75	4.68	0.998	0.995	44.12	40.87
1.2.04.tif	256	93%	5.96	6.13	1.000	0.999	44.92	39.47
1.3.04.tif	1024	95%	5.00	5.17	0.997	0.995	44.15	42.30
1.1.05.tif	256	95%	4.62	4.72	0.996	0.993	44.13	42.30
1.2.05.tif	256	93%	5.97	6.07	0.999	0.998	43.58	39.44
1.3.05.tif	1024	95%	5.02	5.15	0.998	0.997	44.15	42.29
1.1.06.tif	256	94%	5.17	5.28	0.999	0.996	44.14	40.79
1.2.06.tif	256	93%	6.07	6.21	1.000	0.999	44.43	39.46
1.3.06.tif	1024	95%	4.55	4.57	0.996	0.993	44.14	42.28
1.1.07.tif	256	95%	4.57	4.80	0.996	0.994	44.15	42.37
1.2.07.tif	256	93%	5.92	6.18	1.000	0.999	44.19	39.45
1.3.07.tif	1024	96%	4.21	4.33	0.992	0.991	44.13	44.22
1.1.08.tif	256	95%	3.21	2.74	0.987	0.987	44.16	44.28
1.2.08.tif	256	95%	5.82	6.01	0.999	0.999	42.92	42.32
1.3.08.tif	1024	96%	3.72	3.84	0.986	0.985	44.15	44.22
1.1.09.tif	256	96%	3.91	3.72	0.993	0.995	44.14	45.38
1.2.09.tif	256	95%	5.75	5.64	0.999	0.999	44.15	42.34
1.3.09.tif	1024	96%	3.93	3.75	0.987	0.987	44.08	44.26
1.1.10.tif	256	95%	4.21	4.30	0.997	0.996	44.16	43.16
1.2.10.tif	256	93%	5.36	5.46	0.999	0.998	44.01	39.61
1.3.10.tif	1024	96%	3.93	3.97	0.989	0.989	44.11	44.23
1.1.11.tif	256	95%	4.09	4.15	0.995	0.992	44.35	42.67
1.2.11.tif	256	93%	5.55	5.71	1.000	0.998	44.60	39.67
1.3.11.tif	1024	96%	4.11	4.39	0.992	0.992	44.14	44.22
1.1.12.tif	256	96%	4.52	4.72	0.993	0.993	44.15	44.24
1.2.12.tif	256	94%	5.74	5.94	0.999	0.998	44.29	40.82
1.3.12.tif	1024	96%	4.29	4.42	0.991	0.991	44.11	44.23
1.1.13.tif	256	95%	4.68	4.65	0.996	0.995	44.13	42.59
1.2.13.tif	256	94%	5.43	5.55	0.999	0.997	44.51	40.90
1.3.13.tif	1024	96%	4.28	4.46	0.993	0.992	44.15	44.21
1.4.01.tif	1024	95%	3.58	3.57	0.989	0.984	44.27	42.53
1.4.02.tif	1024	95%	3.66	3.69	0.990	0.984	44.16	42.43
1.4.03.tif	1024	96%	3.55	3.75	0.986	0.986	44.30	44.40
1.4.04.tif	1024	96%	3.05	3.14	0.981	0.981	44.31	44.49
1.4.05.tif	1024	96%	3.31	3.58	0.985	0.985	44.30	44.54
1.4.06.tif	1024	96%	3.68	3.65	0.987	0.987	44.24	44.35
1.4.07.tif	1024	96%	3.49	3.88	0.989	0.989	44.26	44.33
1.4.08.tif	1024	96%	3.32	3.36	0.986	0.986	44.22	44.43
1.4.09.tif	1024	96%	4.31	4.71	0.995	0.994	44.27	44.33
1.4.10.tif	1024	96%	4.47	4.84	0.995	0.995	44.15	44.25
1.4.11.tif	1024	95%	4.62	4.70	0.995	0.996	44.19	42.54
1.4.12.tif	1024	95%	4.45	4.46	0.992	0.994	44.16	42.54
1.5.01.tif	256	96%	3.50	3.39	0.983	0.983	44.16	44.29
(*) 1.5.02.tif	256	78%	3.33	3.39	1.000	0.995	46.08	32.26
1.5.03.tif	256	96%	4.55	4.70	0.994	0.994	44.16	44.26
1.5.04.tif	256	94%	4.23	4.22	0.995	0.991	44.14	41.17
1.5.05.tif	256	96%	3.99	3.82	0.988	0.989	44.15	44.52
1.5.06.tif	256	95%	3.26	3.26	0.984	0.975	44.15	42.36
1.5.07.tif	256	96%	4.72	5.05	0.996	0.996	44.12	44.21
texmos1.p512.tif	94%	256	6.02	6.19	1.000	0.999	44.06	40.86
(*) texmos2.p512.tif	81%	256	3.93	3.96	1.000	0.991	43.94	31.49
texmos2.s512.tif	0%	256	0.05	0.11	1.000	0.995	→100.00←	33.36
texmos3.p512.tif	94%	256	5.97	6.22	0.999	0.999	43.92	40.85
(*) texmos3.s512.tif	15%	256	0.17	0.18	1.000	0.963	→100.00←	33.92
texmos3b.p512.tif	94%	256	5.99	6.22	0.999	0.999	43.90	40.85
Average BPP			4.48	4.57				
Average SSIM					0.995	0.993		
Average PSNR (Max. PSNR capped at 50.00)							44.36	41.99

and “ruler.512.tif”. Zooming in these three images at the top of Figure 11 proves this fact. For the last four images in the same figure, JPG is better than *IaP-LVL1*, but *IaP-LVL3* has better performance (*BPP* values) compared to JPG.

Table 11 shows the compression results of *IaP-LVL1* and *JPG* for the Textures dataset, whereas, Table 12 compares *IaP-LVL3* with *JPG* for the same dataset. As shown in Table 11, the performance of *IaP-LVL1* is not bad compared

TABLE 13. **Compression quality of different images using different compressors. Various trials of enhanced *IaP-LVL1* are listed after applying bilateral filter with different parameters and/or adjusting the intensity of the decompressed image. It is clear that Enhanced *IaP-LVL1* still retains lower BPP but higher quality compared to *IaP-LVL2*.**

Image	Dataset	Compressor	BPP	SSIM (0→1)	PSNR (dB)	Bilateral Filter	Intensity Adjustment
bridge.tif	$1^{st} Waterloo$	LVL1	3.16	0.951	29.55		
				0.953	33.54		+7
				0.952	29.25	(3,7,7)	
				0.955	**33.74**	**(3,7,7)**	**+7**
		LVL2	4.72	0.954	30.37		
		JPG-88%	3.14	0.973	35.97		
camera.tif	$1^{st} Waterloo$	LVL1	1.82	0.867	29.80		
				0.921	35.20		+7
				0.888	29.65	(5,11,11)	
				0.949	**35.90**	**(5,11,11)**	**+7**
		LVL2	3.38	0.875	30.68		
		JPG-86%	1.84	0.964	37.84		
boat.tif	$2^{nd} Waterloo$	LVL1	1.83	0.903	29.62		
				0.907	34.54		+7
				0.936	29.58	(5,11,11)	
				0.941	**35.48**	**(5,11,11)**	**+7**
		LVL2	3.38	0.906	30.57		
		JPG-89%	1.80	0.972	40.54		
barb.tif	$2^{nd} Waterloo$	LVL1	2.42	0.909	30.05		
				0.912	34.50		+7
				0.938	29.84	(5,11,11)	
				0.942	**35.05**	**(5,11,11)**	**+7**
		LVL2	3.96	0.913	30.96		
		JPG-92%	240	0.981	41.76		
lena2.tif	$2^{nd} Waterloo$	LVL1	2.04	0.885	30.00		
				0.887	34.52		+7
				0.926	30.04.58	(5,11,11)	
				0.929	**36.00**	**(5,11,11)**	**+7**
		LVL2	3.58	0.889	30.89		
		JPG-91%	1.92	0.967	41.16		
7.1.05.tif	California	LVL1	2.38	0.934	29.70		
				0.942	35.00	(3,5,5)	+7
				0.944	35.15	(3,7,7)	+7
				0.940	29.47	(3,9,9)	
				0.944	**35.25**	**(3,9,9)**	**+7**
				0.941	35.11	(3,11,11)	+7
				0.939	35.03	(5,9,9)	+7
				0.931	34.64	(5,11,11)	+7
		LVL2	3.93	0.938	30.67		
		JPG-87%	2.38	0.952	30.05		

to JPG for most of the images. Conversely, *IaP-LVL3* has the best performance compared to JPG for most of the Textures dataset (Table 12). Figures 12, 13, and 14 visually analyse the decompressed images of *IaP-LVL3* and JPG. With smaller *BPP* value, *IaP-LVL3* achieved better compression ratio with image quality as better as the same JPG images.

D. ENHANCING THE QUALITY OF IaP-LVL1

The aforementioned visual analysis of the decompressed images showed that *IaP-LVL1* is efficient for images with more textures and less gradual regions. That's because of intentionally ignoring the least significant character of each *DNA* pattern corresponding to each pixel, in addition to selecting the first *Codon* corresponding to the *AminoAcid* representing that pixel. Moreover, intentionally ignoring the least significant character of each *DNA* pattern corresponding to each pixel slightly decreases the intensity of that pixel. Table 13 shows various trials of denoising the *IaP-LVL1*'s decompressed images using bilateral filter [31] with different parameters and adjusting the images' intensity. It is clear that the enhanced decompressed images of *IaP-LVL1* still have lower BPP and higher quality compared to *IaP-LVL2*'s decompressed images. Moreover, the enhancement trials shows that adjusting the pixel intensity increases the PSNR value, whereas, applying the bilateral filter with various parameters increases the SSIM value of the *IaP-LVL1*'s decompressed image. On the other hand, thanks to encoding the *CodonBits*, *IaP-LVL3* has competitive compression ratio and quality compared to JPG.

E. COLORED IMAGES

A given 8-bit grayscale image consists of a single channel that is initially encoded into one *DNA* sequence. Alternatively, RGB images contain three channels (Red, Green, and Blue). Until this moment, *IaP* compresses each channel in an RGB image separately. So, the size of the compressed RGB image is approximately three times the size of its compressed grayscale version. This is not the case for JPG images, where the JPG standard depends on downsampling a given RGB image by converting it into a different color space (YCbCr) [32], [33]. As a result, the size of a compressed JPG color image is noticeably smaller than the same color image compressed by *IaP*. This is the reason behind skipping the experimental compression results of color images.

F. RUNTIME COMPLEXITY ANALYSIS

As illustrated in Algorithm 1, most of the *IaP* oparations are proportional to subsequent matrix and vector manipulations with runtime complexity O(m*n), where m and n respectively represent the number of rows and columns in a given image. Of course, the past runtime complexity excludes the compression runtime of the *IaP*'s output files using text compression libraries. Because *IaP* is implemented in Python, measuring its compression/decompression runtime will be an unfair comparison with other highly optimized compression standards such as JPG. Thus, implementing *IaP* in lower level languages (such as Assembly and C) would result in more fair runtime comparison.

G. APPLICATIONS OF IaP

The results shown in this section highlight an important advantage of the *IaP* algorithm that would serve a wide range of online applications (such as social media and navigation maps) with limited bandwidth. Based on the fact that every image compressed by *IaP-LVL3* consists of two components (the *OriginalAAs* and *CodonBits* components), the transfer of image would start by transferring the *OriginalAAs* component (as if the image is only compressed by *IaP-LVL1*). After that, the quality of the displayed image can be further enhanced by applying the *CodonBits* component. On the other hand, *IaP* levels can be used for compressing medical, satellite and security surveillance images according to the acceptable quality levels in each of these fields.

V. CONCLUSION

This article proposed the *IaP* algorithm as a novel multilevel lossy image compression algorithm for grayscale images that is based on encoding image pixels into a sequence analogous to the biological *protein* sequences in living creatures. The four compression levels of *IaP* cumulatively cover different components of the encoded image, however, the 2^{nd} and 4^{th} levels have been excluded from the results because of their inefficient compression ratios compared to the quality of their decompressed images.

Various quantitative and qualitative measures have been applied on different grayscale datasets. Consequently, *IaP-LVL1* introduced acceptable quality with very high compression ratio, whereas, the quality obtained by *IaP-LVL3* was very competitive to JPG. Moreover, *IaP-LVL1* could not retain gradual regions, because *IaP-LVL1* holds only one component of the encoded image that loss the small differences between adjacent pixels of these regions. However, applying an appropriate filter (such as bilateral filter) and intensity adjustmentent can raise the quality of the *IaP-LVL1*'s decompressed images to competitive levels. On the other hand, *IaP-LVL3* uses the CodonBits component that presents successful guidance while decoding the *IaP-LVL3*'s compressed images.

In addition, the introduced multilevel compression levels can effectively be used in multimedia streaming applications, That's because the quality of the streamed image can be enhanced by subsequently transferring and processing separate components of the *IaP*'s compressed image.

Finally, the main target of this article is to draw the attention of the research community to the applicability and efficiency of such biologically-inspired ideas to gain more technological advances in the current era.

ACKNOWLEDGMENT

The authors acknowledge with thanks the University of Jeddah's technical and financial support.

REFERENCES

[1] R. F. Enriquez, *New Basics of Computer Graphics 2020*. Creative Hands Publishing, 2020.

[2] K. Sayood, *Introduction to Data Compression*. San Mateo, CA, USA: Morgan Kaufmann, 2017.

[3] C. Taskin and S. K. Sarikoz, "An overview of image compression approaches," in *Proc. 3rd Int. Conf. Digit. Telecommun. (ICDT)*, Jun. 2008, pp. 174–179.

[4] H. P. Yockey, *The Central Dogma of Molecular Biology*. Cambridge, U.K.: Cambridge Univ. Press, 2005, pp. 20–26.

[5] J. C. Venter, H. O. Smith, and L. Hood, "A new strategy for genome sequencing," *Nature*, vol. 381, no. 6581, pp. 364–366, May 1996.

[6] J. L. Weber and E. W. Myers, "Human whole-genome shotgun sequencing," *Genome Res.*, vol. 7, no. 5, pp. 401–409, May 1997.

[7] J. C. Venter, M. D. Adams, G. G. Sutton, A. R. Kerlavage, H. O. Smith, and M. Hunkapiller, "Shotgun sequencing of the human genome," *Science*, vol. 280, no. 5369, pp. 1540–1542, Jun. 1998.

[8] G. Myers, "Whole-genome dna sequencing," *Comput. Sci. Eng.*, vol. 1, no. 3, pp. 33–43, May 1999.

[9] P. Compeau and P. Pevzner, *Bioinformatics Algorithms: An Active Learning Approach*, vol. 1. La Jolla, CA, USA, 2015.

[10] A. J. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, Jul. 2018.

[11] N. Thakur, A. Gupta, and S. Raju, "Benchmarking image compression: A review," *Int. J. Latest Trends Eng. Technol.*, vol. 9, no. 3, pp. 276–283, 2018.

[12] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[13] D. M. Rouse and S. S. Hemami, "Understanding and simplifying the structural similarity metric," in *Proc. 15th IEEE Int. Conf. Image Process.*, Oct. 2008, pp. 1188–1191.

[14] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991.

[15] G. Amaranageswarao, S. Deivalakshmi, and S.-B. Ko, "Deep dilated and densely connected parallel convolutional groups for compression artifacts reduction," *Digit. Signal Process.*, vol. 106, Nov. 2020, Art. no. 102804.

[16] J. Li, D. Li, C. Chen, Q. Yan, and X. Lu, "A dual-residual network for JPEG compression artifacts reduction," *Signal, Image Video Process.*, vol. 15, no. 3, pp. 485–491, Apr. 2021.

[17] J. Li, Y. Wang, H. Xie, and K.-K. Ma, "Learning local and global priors for JPEG image artifacts removal," *IEEE Signal Process. Lett.*, vol. 27, pp. 2134–2138, 2020.

[18] J. Li, Y. Wang, H. Xie, and K.-K. Ma, "Learning a single model with a wide range of quality factors for JPEG image artifacts removal," *IEEE Trans. Image Process.*, vol. 29, pp. 8842–8854, 2020.

[19] Y. Hu, W. Yang, Z. Ma, and J. Liu, "Learning end-to-end lossy image compression: A benchmark," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Mar. 11, 2021, doi: 10.1109/TPAMI.2021.3065339.

[20] F. Dufaux, G. J. Sullivan, and T. Ebrahimi, "The JPEG XR image coding standard [standards in a nutshell]," *IEEE Signal Process. Mag.*, vol. 26, no. 6, pp. 195–204, Nov. 2009.

[21] F. De Simone, L. Goldmann, V. Baroncini, and T. Ebrahimi, "Subjective evaluation of JPEG XR image compression," *Proc. SPIE*, vol. 7443, Aug. 2009, Art. no. 74430L.

[22] A. Pinheiro, K. Fliegel, P. Korshunov, L. Krasula, M. Bernardo, M. Pereira, and T. Ebrahimi, "Performance evaluation of the emerging JPEG XT image compression standard," in *Proc. IEEE 16th Int. Workshop Multimedia Signal Process. (MMSP)*, Sep. 2014, pp. 1–6.

[23] A. Artusi, R. K. Mantiuk, T. Richter, P. Hanhart, P. Korshunov, M. Agostinelli, A. Ten, and T. Ebrahimi, "Overview and evaluation of the JPEG XT HDR image compression standard," *J. Real-Time Image Process.*, vol. 16, no. 2, pp. 413–428, 2019.

[24] *BPG Image Format*. Accessed: Sep. 28, 2021. [Online]. Available: https://bellard.org/bpg/

[25] D. Yee, S. Soltaninejad, D. Hazarika, G. Mbuyi, R. Barnwal, and A. Basu, "Medical image compression based on region of interest using better portable graphics (BPG)," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 216–221.

[26] *WebP: An Image Format for the Web*. Accessed: Sep. 29, 2021. [Online]. Available: https://developers.google.com/speed/webp

[27] *WebP: Frequently Asked Questions*. Accessed: Sep. 30, 2021. [Online]. Available: https://developers.google.com/speed/webp

[28] S. K. Bandyopadhyay and S. Chakraborty, "Image compression using dna sequence," *Int. J. Comput. Sci. Eng. Technol.*, vol. 1, no. 11, pp. 1–3, 2011.

[29] M. Dimopoulou, M. Antonini, P. Barbry, and R. Appuswamy, "Dna coding for image storage using image compression techniques," in *Proc. CORESA*, 2018, pp. 1–3.

[30] M. Nadipally, "Chapter 2 - optimization of methods for image-texture segmentation using ant colony optimization," in *Intelligent Data Analysis for Biomedical Applications* (Intelligent Data-Centric Systems). New York, NY, USA: Academic, 2019, pp. 21–47.

[31] S. K. Ramanandan, B. Lehmann, and D. Kraus, "Optimal parameters for bilateral filtering and SAS image denoising," in *Proc. Int. Symp. Ocean Electron.*, Nov. 2011, pp. 27–33.

[32] E. Hamilton, "JPEG file interchange format," C-Cube Microsyst., Milpitas, CA, USA, Tech. Rep. 1.02, 2004.

[33] H. Noda and M. Niimi, "Colorization in YCbCr color space and its application to JPEG images," *Pattern Recognit.*, vol. 40, no. 12, pp. 3714–3720, Dec. 2007.

MOHAMMAD NASSEF received the M.Sc. and Ph.D. degrees in computer science from the Faculty of Computer Science and Artificial Intelligence, Cairo University, in 2007 and 2014, respectively. He was an Academic Supervisor for 15 M.Sc. and Ph.D. students, from 2014 to 2020. Moreover, he was an Academic Coordinator of the Computing and Bioinformatics Program, Cairo University, from 2016 to 2019. He has been an Associate Professor with the Department of Computer Science, Faculty of Computers and Artificial Intelligence, Cairo University, since 2019. He is currently a Visiting Professor with the Department of Computer Science and Artificial Intelligence, College of Computer Science and Engineering, University of Jeddah, Saudi Arabia. His research interests include bioinformatics, machine learning, genome and image compression, automated essay scoring, and parallel computing.

MONAGI H. ALKINANI received the Ph.D. degree in computer science from Western University, London, Canada, in 2017. In 2018, he joined the Deanship of Scientific Research at the University of Jeddah, Saudi Arabia, where he served as the Vice Dean for research. He holds an assistant professor position with the Department of Computer Science and Artificial Intelligence. He is a member of the Jeddah Computer Vision Team, where he supervises research activities and teaches image processing and artificial intelligence to bachelor students as well as signal processing to master students in computer science. At the Deanship, he has supervised research in the field of computer vision. He is currently the Dean of the College of Computer Science and Engineering, University of Jeddah. He has been involved in many collaborative research projects financed by various instances, including the Ministry of Education and the University of Jeddah.