

Approximate Compression – Enhancing Compressibility Through Data Approximation

Harini Suresh, Shashank Hegde, and John Sartori
University of Minnesota

Abstract

Internet-connected mobile processors used in cellphones, tablets, and internet-of-things (IoT) devices are generating and transmitting data at an ever-increasing rate. These devices are already the most abundant types of processor parts produced and used today and are growing in ubiquity with the rapid proliferation of mobile and IoT technologies. Size and usage characteristics of these data-generating systems dictate that they will continue to be both bandwidth- and energy-constrained. The most popular mobile applications, dominating communication bandwidth utilization for the entire internet, are centered around transmission of image, video, and audio content. For such applications, where perfect data quality is not required, approximate computation has been explored to alleviate system bottlenecks by exploiting implicit noise tolerance to trade off output quality for performance and energy benefits. However, it is often communication, not computation, that dominates performance and energy requirements in mobile systems. This is coupled with the increasing tendency to offload computation to the cloud, making communication efficiency, not computation efficiency, the most critical parameter in mobile systems. Given this increasing need for communication efficiency, data compression provides one effective means of reducing communication costs. In this paper, we explore approximate compression and communication to increase energy efficiency and alleviate bandwidth limitations in communication-centric systems. We focus on application-specific approximate data compression, whereby a transmitted data stream is approximated to improve compression rate and reduce data transmission cost. Whereas conventional lossy compression follows a one-size-fits-all mentality in selecting a compression technique, we show that higher compression rates can be achieved by understanding the characteristics of the input data stream and the application in which it is used. We introduce a suite of data stream approximations that enhance the compression rates of lossless compression algorithms by gracefully and efficiently trading off output quality for increased compression rate. For different classes of images, we explain the interaction between compression rate, output quality, and complexity of approximation and establish comparisons with existing lossy compression algorithms. Our approximate compression techniques increase compression rate and reduce bandwidth utilization by up to 10× with respect to state-of-the-art lossy compression while achieving the same output quality and better end-to-end communication performance.

ACM Reference format:

Harini Suresh, Shashank Hegde, and John Sartori
University of Minnesota. 2017. Approximate Compression – Enhancing Compressibility Through Data Approximation. In *Proceedings of ESTIMedia'17, Seoul, Republic of Korea, October 15–20, 2017*, 10 pages.
DOI: 10.1145/3139315.3139319

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions.acm.org.

ESTIMedia'17, Seoul, Republic of Korea
© 2017 ACM. 978-1-4503-5117-1/17/10...\$15.00
DOI: 10.1145/3139315.3139319

CCS Concepts • Information systems → Compression strategies

1 Introduction

Traditional computing systems are increasingly being replaced and outnumbered by mobile and cloud computing systems that rely heavily on communication between internet-connected devices. Along with this shift toward communication-centric computing, technological advances and proliferation of internet-connected devices are constantly increasing the amount of data produced and transmitted in these computing systems [1–4]. This explosion in data communication is particularly prominent for image, video, and audio data. For example, mobile phones with high-resolution cameras like the iSight camera in iPhone6s regularly exchange 12MP images through popular applications like Facebook and Instagram. This represents a more than 100× increase in transmitted data for a single image compared to the images transmitted by earlier phone models. Similar, or even greater increases in communication bandwidth usage are observable across the gamut of popular mobile applications, including streaming digital audio and video content, social media, video conferencing, and cloud storage [5]. In fact, nearly all of the applications that consume the vast majority of all internet bandwidth (e.g., Netflix, Youtube, Facebook, iTunes, Instagram, Snapchat, etc.) are centered around transmission of video, image, and audio content [6]. Network technologies continue to evolve to support increasing loads from an ever-increasing number of communication-centric computing devices and applications, but device technology and application advances, along with usage trends, ensure that the demand for more bandwidth always exceeds the advances in communication and network technology.

With applications and computing platforms straining available communication bandwidth, data compression presents one means of reducing bandwidth utilization. Given trends toward more mobile, parallel, and cloud-based computing, it may often be advantageous to compress data locally before transmitting over a bandwidth-constrained link. At the same time, given that many of the killer applications for communication bandwidth operate on noise-tolerant data (e.g., image, video, audio), lossy compression [7] presents a potentially attractive approach to reduce bandwidth usage further by sacrificing some data fidelity for increased compression rates. In this paper, we recognize that exploiting noise tolerance in these applications to perform less costly *approximate data communication* may be more effective for emerging applications than exploiting noise tolerance to perform approximate computing. Along this vein, we expand on the field of lossy compression by designing a suite of approximate compression techniques for noise-tolerant applications.¹ We use classification of dataset features to create data-aware approximations that enhance the compressibility of data while maintaining acceptable data quality. The goal of our approximate compression techniques is to maximize the increase in compression rate provided by approximation per unit reduction in quality. Our paper makes the following contributions.

- We propose a suite of approximation techniques that enhance the performance of existing lossless compression algorithms for image

¹To distinguish approximate compression from existing lossy compression techniques, we describe approximate compression as data approximation followed by lossless compression.

data. Our techniques leverage image characteristics to increase gains in compression rate per unit quality reduction.

- We show that the approximation technique that maximizes compression rate depends on image characteristics and desired output quality and demonstrate that learning-based techniques (e.g., neural networks) can be used to select an appropriate approximate compression technique for a given data stream.
- Our approximation techniques allow graceful trade-offs between data quality and compression rate, allowing a sweet spot to be selected based on user preferences or application constraints.
- We perform a thorough evaluation of our suite of approximate compression techniques over the trade-off space between quality and compression rate and show that our data-aware approximate compression techniques result in up to 10× improvement in compression rate with respect to state-of-the-art lossy compression techniques, for the same output quality.
- We evaluate the performance of our approximate compression techniques against state-of-the-art lossy compression and show that they improve end-to-end performance by over 2×, on average, while achieving better compression rates for the same output quality. We also show that our approximation techniques achieve performance scalability on massively-parallel processors, indicating that the overhead of performing approximate compression will decrease as processors continue to integrate more parallelism.

2 Background

In his seminal work on information theory in the late 1940s [8], Claude Shannon quantified the information gained from an event, e , in terms of the probability of occurrence of the event, p_e . I.e., $information_e = -\log_2(p_e)$. The entropy, h , of a transmitted data stream is the information content present per symbol in the data stream. I.e., if a source transmits n identical, independently distributed symbols, the entropy of the data stream is given by $h = \sum_{i=1}^n (p_i * -\log_2(p_i))$. The entropy of the data stream also represents the minimum number of bits required to encode the symbols, so that they can be uniquely decoded at the receiver. An encoding scheme is said to be *optimal* when the number of bits used for encoding equals the entropy of the data stream. *Redundancy* is the difference between the symbol set's largest possible entropy and the actual entropy. A data stream is said to have achieved maximum compression when the redundancy is nil.

Statistical data compression methods based on entropy encoding assign shorter codes to frequently-occurring symbols and longer codes to rarely-occurring symbols to reduce the total amount of data in the encoded stream. The achievable compression rate for a data stream depends on the frequencies of occurrence of symbols in the stream, as well as the ability of the model to correctly estimate the relative symbol frequencies. The goal of our approximation techniques is to judiciously increase the occurrence of frequently-occurring symbols and decrease the occurrence of infrequently-occurring symbols in a way that best preserves data quality. This increases compression rate by (1) reducing the average code length for encoded symbols and (2) making it easier to estimate relative symbol frequencies to produce a more optimal encoding.

Our approximate compression techniques are similar to lossy compression, whereby data quality is sacrificed to achieve a higher compression rate. Lossy compression is applicable for applications in which perceptual inaccuracies in the output are permissible. Typical applications of lossy compression include image, video, and audio compression. The loss of output fidelity from the original data stream is either imperceptible to the user or has been accepted by the user for the application. The goal of lossy compression is to maximize the compression rate within the given distortion limits. When the distortion is 0 (i.e., lossless compression), the number

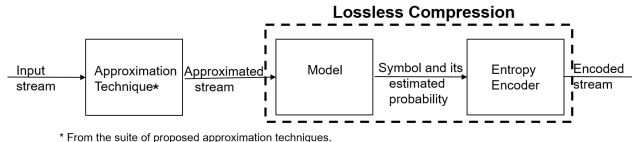


Figure 1. We propose a suite of approximation techniques to enhance the compressibility of the input stream fed to a lossless compression algorithm.

of bits required for encoding is equivalent to the entropy of the symbols generated by the source. Reducing the number of bits required to encode the input stream to below its entropy becomes possible with the introduction of distortions to the data stream.

Lossy compression introduces distortions through transformation and quantization of the input data, which are then losslessly compressed. Transformation algorithms like Discrete Cosine Transform, and Discrete Wavelet Transform are used to transform the input to a form with lower entropy that requires fewer bits for encoding. Transformation may also aid in identifying components of a data stream that are irrelevant to human perception.

This paper demonstrates how differences in data characteristics can be leveraged to enhance data compressibility for image data. In general, approximations made to image data either (1) target the property of the entropy encoder to assign shorter codes to frequently-occurring symbols or (2) impose a more compact data representation. In addition to immediate benefits due to data reduction, these techniques also improve the accuracy of the modeling stage by increasing the probability that relative symbol frequencies are estimated correctly. Both factors improve compression rate.

3 Related Work

Our work on approximate compression techniques can be seen as an extension of the field of lossy compression. Compression of an image is only possible when there is redundancy in the image, equivalently, correlation between pixels. Conventional lossy compression algorithms [9–11] work by transforming an image into a representation where the dependencies between pixels are eliminated, i.e., the pixels are decorrelated and the symbols in the equivalent representation can be encoded independently. Lossy compression techniques based on transforms such as Discrete Fourier Transform (DFT) and Discrete Cosine Transform (DCT) transform images into the frequency domain and identify frequency coefficients, corresponding to different spatial frequencies, that can be approximated with minimal impact on output quality. A segment of the image with fine details and sharp edges contains predominantly high spatial frequencies. Since the human eye is more sensitive to variations in low-frequency regions than high-frequency regions, quantization of high-frequency coefficients to increase compression rate has less of an impact on visual perception of image quality.

Research has also been done on introducing approximations to the transforms used in lossy compression techniques to reduce their complexity at the expense of some accuracy [12, 13]. Approximate transforms that use integer rather than floating point coefficients have been applied to Fast Fourier, Discrete Fourier, Discrete Hartley, and Discrete Cosine transforms [14, 15]. The integer approximations result in lower multiplicative complexity for the computations compared to traditional approaches.

4 Approximation Techniques

This section discusses a suite of approximation techniques that can be used to enhance the compressibility of image data. The approximations introduce controllable distortions to images, allowing graceful trade-offs between output quality and compression rate. As shown in Figure 1, approximation is performed as a pre-processing step to lossless compression to enhance its performance. The goal

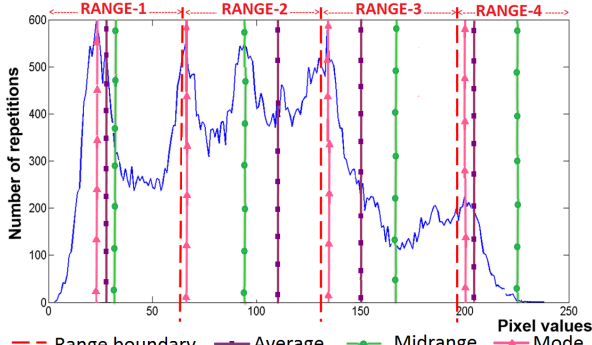


Figure 2. Mode-based and average-based approximations can provide better representative pixel values compared to midrange-based approximation. The distance of approximation used is 32.

of approximation is to maximize the increase in compression rate per unit of quality degradation.

Our approximations target the principle of entropy encoding used in lossless compression. In general, the approximations proposed in this section increase the repetition of frequently-occurring pixel values and reduce the occurrence of infrequently-occurring pixel values. This reduces the average code length, increases the number of pixels represented by shorter codes, and reduces the number of pixels represented by longer codes. This also reduces the number of unique pixel values and increases the probability that modeling correctly estimates the relative pixel probabilities, resulting in an encoding that is closer to optimal. To characterize the extent to which an approximation is allowed to distort an image, we introduce a context-specific metric called *distance of approximation*, which will be explained for each approximation below. Increasing the distance of approximation allows approximation to be performed to a greater extent in exchange for increased compression rate. Sweeping the range of distance of approximation explores the trade-off space between output quality and compression rate.

Midrange, Mode, and Average-based approximations: Our first set of approximation techniques reduce the entropy of image data by analyzing the histogram of pixel values in the image and selecting representative pixel values from the image. To select the representative values, we divide the histogram into equal-sized spectral ranges and select one pixel value from each range to represent all the pixels in the range. After this approximation, many pixel values may be replaced by a nearby approximate pixel value. For this approximation, the distance of approximation is defined as the radius of the spectral ranges that partition the histogram. A distance of d means that $2d$ pixel values are represented by one representative value. A greater distance of approximation corresponds to choosing fewer values to represent the pixel distribution. This improves compressibility by increasing the repetition of fewer unique values but reduces output quality. We explore three ways of selecting the representative value from a spectral range.

The simplest way to select representative values is to use the midpoint of each range. We call this *midrange-based approximation*. Since the set of representative values is static for a given distance of approximation, the approximation time for this technique is not data-dependent. This simplifies computation of the approximate image but may result in lower quality, since the representative values are independent of the underlying pixel distribution. For example, in the pixel value histogram of Figure 2, selecting the midpoint value of 228 to represent the pixel values from 196 to 250 is not very representative of the distribution of pixel values in the range, since the data are clustered around the low end of the range.

One way to make the selection of representative values pixel distribution-aware is to select the most frequently-occurring value to represent each range, i.e., *mode-based approximation*. In case

multiple values tie for the mode, their mean is selected to represent the range. Mode-based approximation can increase quality, since it considers the underlying pixel distribution. The computational complexity for finding the mode ($O(n)$) is greater than for selecting the midpoint, as in midrange-based approximation ($O(1)$).

While using the mode value to represent a range can result in a better representative value, the mode only represents one of the values in the range. To provide an equal weighting of all the pixel values within a range, the average pixel value can be used, i.e., *average-based approximation*. Computing the average can be performed with a parallel reduction algorithm. The complexity of performing histogramming to compute the mode depends on the data distribution, which determines the rate of conflicts that require serialization while updating histogram bins. Figure 3 illustrates how the distance of approximation can be varied for midrange, average, and mode-based approximations to navigate the trade-off between output quality and compressibility.

NxN approximations: The approximation techniques discussed so far look at images from a global perspective and fail to capture local variations within the image. For example, average-based approximation computes a representative value for a range based on all the pixels in the image that fall within the range. Our next set of approximation techniques – NxN approximations – aims to achieve higher quality by applying the previously-proposed techniques to local regions in an image. The image is subdivided into blocks of NxN pixels, as shown in the left subfigure of Figure 4, and mean- or mode-based² approximations are applied locally within each block. This reduces distortions, since the average and mode are localized based on the distribution of pixel values in each block. On the other hand, compressibility is reduced for NxN approximations, compared to the global approximations discussed above, since accounting for local variations in the image increases entropy.

In addition to the distance of approximation, the block size, N , introduces another dimension along which to explore the trade-off between compression rate and image quality. Increasing N tend to increase compression rate at the expense of output quality. Empirically, we select an optimal value of $N = 8$ for our evaluations.

Clustering-based approximation: NxN approximations confine the scope of local variations to square blocks. This is a computationally-simple way to achieve locality but may not maximize entropy reduction, since spatial locality of pixel values may extend beyond the bounds of the NxN square regions. Clustering-based approximation forms arbitrarily sized and shaped local clusters of pixels with similar pixel values. For clustering-based approximation, the distance of approximation is defined as the maximum difference between any two pixel values that are permitted within the same cluster. As with other approximation techniques, a larger distance can increase compression rates at the expense of output quality.

Clusters are formed by traversing the rows of an image and keeping track of the minimum and maximum values observed. One cluster ends and another begins when the maximum and minimum values differ by more than the distance of approximation. Pixels in a cluster are replaced with the average of the values belonging to the cluster. The middle subfigure in Figure 4 illustrates cluster formation. Since rows can be traversed in parallel, cluster formation can be implemented efficiently.

Floodfill-based approximation: Floodfill-based approximation is an extension of clustering-based approximation that extends clusters in multiple directions, rather than just along horizontal scan lines. Since regions with uniform pixel values do not necessarily conform to square regions or horizontal scan-lines, as highlighted in the right subfigure of Figure 4, floodfill-based clustering can

²Note that midrange-based NxN approximation would be the same as the global midrange-based approximation.

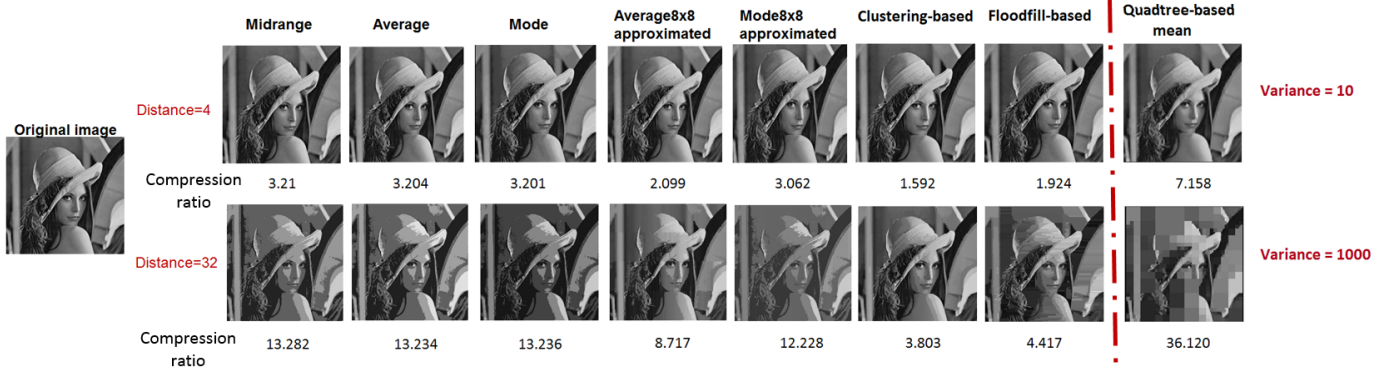


Figure 3. The distance of approximation can be adjusted to trade off output quality for increased compressibility.

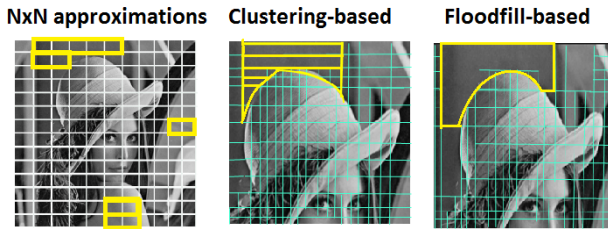


Figure 4. From left to right: (a) NxN approximation captures local variations of pixel intensities within the image. (b) Clustering-based approximation extends locality clusters along horizontal scan-lines. (c) Floodfill-based approximation identifies connected groups of pixels with similar values across multiple dimensions.

exploit spatial value locality to a greater extent than NxN and clustering-based approximations, potentially resulting in greater entropy reduction and increased compressibility. The floodfill algorithm assigns the same average pixel value to an area of connected pixels, where connectivity is decided based on spatial adjacency and distance of approximation, as in clustering-based approximation. To retain the structure of the image, edges in the image are identified using Canny edge detection and are left untouched by the approximation. The possibility of lowering the entropy by increasing distance of approximation is contingent on the presence of regions of connected pixels with similar values in the original image. While floodfill-based approximation can identify larger connected regions than clustering-based approximation, the floodfill algorithm is more complex and less parallelizable than cluster formation and therefore requires more time to perform approximation.

Quadtree-based approximation The approximation techniques discussed so far improve compressibility by reducing image entropy. Motivated by approximation techniques that assign a representative pixel value to a region of pixels with similar values, our next approximation technique aims to reduce image size by storing the image in a compact format that only records the size and average representative value for a region of pixels. To avoid the necessity of storing location information for regions, we base our approximation on a tree data structure called a *quadtree*, in which each internal node has exactly four children. We use a quadtree to recursively subdivide an image into similarity regions based on regional variance, where variance is defined as the average sum of squares of differences between pixel values in the region and the region's mean pixel value. In our approximate quadtree representation, an image region is subdivided into four quadrants whenever the variance of the region is greater than a specified distance of approximation.³ As such, we define the distance of approximation for this

³Quadtrees have been used to perform lossless image compression by subdividing regions with multiple pixel values into quadrants until all regions can be represented by a single pixel value [16].

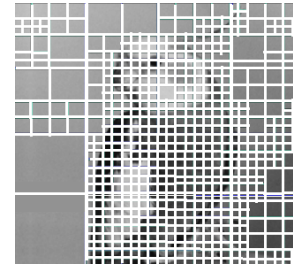


Figure 5. Quadtree-based approximation identifies large blocks of similar-valued pixels that can be represented using just two numbers – the dimension of the block and the mean of the pixel values within the block.

technique as the maximum variance allowed for a similarity region. A larger distance of approximation can result in larger regions and a smaller quadtree size but also allows more distortions.

Figure 5 visualizes the quadtree representation of an image. The structure of the quadtree allows reconstruction of the image based on only the edge size and representative value for each region. This allows for an extremely-compact image representation when the image contains large regions of similar values. For example, the large 64x64 blocks corresponding to the background in the lower left corner of Figure 5 can be represented by only two values – a block compression rate of 2048x. Since the image is transmitted in the approximate compressed format, post-processing is required to convert the quadtree data back into a normal image format.

We define a minimum block dimension of 2 for our quadtree-based approximation, ensuring that the compressed image is at most half the size of the original image. If a block has a variance exceeding the distance of approximation and is at the minimum block size, it is not subdivided further, since doing so would increase the image size in the quadtree representation. The process of quadtree formation and block approximation can be performed in parallel, reducing the time to perform approximation.

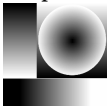





5 Methodology

Sequential evaluations of compression, decompression, and approximation are performed on an Intel Xeon X5675 CPU running at 3.07 GHz. Parallel evaluations of our approximation techniques are performed on an NVIDIA GeForce GTX480 GPU.

Benchmark Images: The benchmark images, summarized in Table 1, are taken from GreySet1 of the Waterloo image repository [17].

Lossless Compression Algorithms: This section discusses the lossless compression algorithms that were evaluated as candidates to compress the approximated image data generated by our approximation techniques. The lossless algorithms considered are 7zip 15.09 [18], BZIP2 1.0.6 [19], CMIX-v7 [20], GZIP 1.6 [21], Infzip [22], LHA 1.14i [23], LZ4 [24], Nanoszip 0.09 [25], LZIP 1.17 [26], LZOP 1.03 [27], PAQ 8o6 [28], RAR 5.21 [29], XZ 5.2.1 [30],

Table 1. Characterization of benchmark images.

Image	Characteristics
 <p>Slope</p>	<ul style="list-style-type: none"> Autokinetic image – appearance changes when viewed from different angles. Artificially generated, but the qualities of blur and noise are similar to natural images.
 <p>Bird</p>	<ul style="list-style-type: none"> Still image. Composed of smooth textured regions – e.g., background, bird’s head – and mildly coarse textured regions like the feathers on the bird’s body.
 <p>Lena</p>	<ul style="list-style-type: none"> Still image. Dominated by smooth regions – her face, body, portions of the background. There are also mildly coarse textured regions – her hat – and complex textured regions – the accessory on her hat.
 <p>Goldhill</p>	<ul style="list-style-type: none"> Still image. Many edges form boundaries between textured regions. Many complex textures, localized to small regions.
 <p>Camera</p>	<ul style="list-style-type: none"> Still image. Composed of equal amounts of smooth textured regions – the coat, the sky – and mildly coarse textured regions – the grass.
 <p>Bridge</p>	<ul style="list-style-type: none"> Still image. Various types of textures in different regions of the image help to camouflage distortions introduced by approximation techniques.

ZLIB 1.2.8 [31], ZPAQ 7.05 [32], ZZIP v0.36c [33], PNG, and GIF using NConvert⁴ [34]. Table 2 shows the compression rates achieved by the different lossless compression algorithms on the original, unapproximated benchmark images.

Metrics for Evaluation: The goal of this paper is to introduce controlled approximations to the images that trade off quality for improved compression. Thus, we need to quantify image quality and compression performance to properly evaluate the approximations. The metrics used to evaluate image quality and compression are discussed below.

Image Quality: The metric used to assess the output quality of an approximated image should quantify the degradation of the image due to approximations, with respect to the original image. Since the acceptability of approximations is user-determined, the approximation metric should correlate well with visual perception of image quality. In image processing, Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR) are the two widely used metrics for quality of recovered images after lossy compression.

Figure 6 compares different quality metrics for images with different distortions. All the images have a PSNR of around 20 dB, and yet, appear visually different. The contrast-stretched image (2nd image), visually appears to be a good approximation to the original image, but it has lower PSNR than the images affected by noise (4th, 5th, 6th images), which look worse. We do not use PSNR

⁴NConvert (GIF) uses LZW compression, which achieves compression only when symbols are small and repetitive; thus, GIF has compression ratios of less than one for some images with fine details (see Table 2).

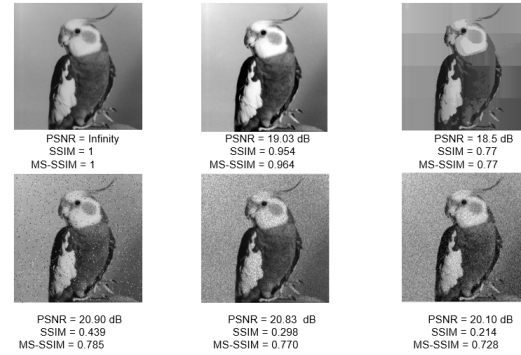


Figure 6. MS-SSIM correlates better with subjective evaluation of visual quality than PSNR and SSIM. From left-to-right, starting across the top row, these images are 1. the original version [17] 2. contrast-stretched 3. mode 8x8 approximated 4. salt and pepper noise 5. Speckle noise 6. Gaussian noise

as a metric because it is not able to differentiate between images that are perceived to have different quality by the human visual system, and it does not correlate well with perceived image quality.

Structural Similarity (SSIM) is a metric that has been developed to quantify the visually-perceived quality of images [35]. The human visual system perceives images by recognizing their structural information, which is provided by the inter-dependency of spatially-close pixels. The SSIM algorithm processes the reference and distorted image in terms of 8x8 pixel blocks. Separate functional measures of luminance – $l(x,y)$, chrominance – $c(x,y)$, and structural similarity – $s(x,y)$ are defined over the local, overlapping windows of the two signals x and y , and are weighted within each window. A general form of SSIM index is obtained by combining the three measures. $SSIM(x, y) = [l(x, y)]^\alpha * [c(x, y)]^\beta * [s(x, y)]^\gamma$, where α , β , and γ are used to assign the relative importance to luminance, chrominance, and structural similarity measures. The SSIM rates the quality of images on a scale between -1 to 1. An SSIM value of 1 indicates identical images, whereas a value of -1 indicates complete uncorrelation between the reference and distorted image.

The quality score provided by SSIM can distinguish between the images, as shown in Figure 6. However, images that are affected by noise (fourth, fifth, and sixth images) get a quality score close to 0, even though the bird in these images is discernible. In the third image of the series, in spite of the visibility of the distortions introduced by approximations, the quality score is higher than what is assigned for the images affected by noise. Hence, SSIM does not correlate well with human perception for these images.

Multi-Scale Structural Similarity Index (MS-SSIM) [36] extends the SSIM technique by evaluating the image at multiple resolutions and viewing distances. MS-SSIM works by computing the SSIM over many iterations by successively down-sampling the reference and distorted images by a factor of two. The results obtained from the different scales of evaluation are weighted differently to give an overall MS-SSIM score. MS-SSIM has been shown to align more closely with subjective evaluations of image quality than SSIM and PSNR [36]. Thus, we select it as our quality metric.

Compression: The bandwidth of a transmission link is the amount of data that can be transferred per unit time across the link. The compression rate achieved when a data stream is compressed is directly proportional to the reduction in bandwidth usage on a transmission link when the compressed data stream is transmitted. Compressing data reduces transmission time at the cost of increased processing time at the sender and receiver for compressing and decompressing the data. The evaluation of the performance of compression algorithms is based on the following two metrics.

Table 2. Compression rates achieved by different lossless compression algorithms for original, unapproximated images.

Compression Algorithm	Switches	Test Images						
		Bird	Bridge	Camera	Goldhill	Lena	Slope	CosinePattern
7zip	-m0=LZMA2	1.898	1.273	1.593	1.362	1.358	4.747	3.875
7zip	-m0=PPMd	2.041	1.193	1.629	1.316	1.325	4.923	6.767
BZIP2	-best	1.952	1.212	1.562	1.308	1.349	4.854	5.887
CMIX		2.716	1.395	1.984	1.535	1.741	8.587	8.720
GZIP	-best	1.570	1.069	1.353	1.130	1.113	3.672	6.489
INFOZIP	-best	1.562	1.065	1.347	1.126	1.110	3.630	6.426
LHA		1.588	1.079	1.388	1.144	1.141	3.668	6.477
LZ4	-9	1.279	1.005	1.175	1.009	1.022	3.107	6.460
NANOZIP (LZHD)	-cd	2.207	1.365	1.730	1.490	1.678	4.795	6.622
LZIP	-best	1.907	1.275	1.599	1.365	1.358	4.838	7.854
LZOP	-9	1.434	1.005	1.253	1.031	1.039	3.394	6.465
NANOZIP (NZCM)	-cc	2.027	1.338	1.614	1.437	1.539	3.079	9.969
PAQ 806	-7	2.624	1.372	1.943	1.511	1.711	7.566	10.386
RAR	-m5	1.818	1.266	1.443	1.388	1.384	3.652	6.663
XZ		1.902	1.275	1.595	1.365	1.357	4.792	7.866
ZLIB		1.580	1.077	1.387	1.154	1.117	3.624	6.538
ZPAQ	-best	2.218	1.289	1.756	1.409	1.451	6.738	8.742
ZZIP	-mx	2.001	1.221	1.586	1.316	1.345	5.288	
NConvert (GIF)		1.382	0.858	1.184	0.959	0.923	2.229	2.204
NConvert (PNG)		1.541	1.057	1.355	1.131	1.096	3.434	8.826

- Compression rate is defined as $\frac{\text{Size of uncompressed file}}{\text{Size of compressed file}}$. Lossy compression rates are reported at a particular output quality value.
- The time taken to approximate and compress the file at the sender and decompress the file at the receiver is measured by the Linux time command and averaged over ten runs.

The choice of a compression algorithm is based on the priority between the resulting file size and the speed of approximation and compression, or decompression, as dictated by the application requirements or user preference. A third factor – the memory requirements of the compression algorithm – also influences the choice. Data Compression finds applications in storage and transmission. Storage favors high data density and faster access times, and as such, the typical priority ladder from top to bottom is (1) compression rate, (2) time to decompress, and (3) time to approximate and compress. For storage applications, e.g., backing up data on a hard drive, the primary concern is reducing the size of the compressed file to meet available storage capacity. This application involves a one-time compression and a many-time decompression. Therefore, longer approximation or compression time does not impact performance as much as a longer decompression time.

Parallelization and Acceleration of Approximation Compression: Increasingly, mobile platforms are integrating more advanced parallel processors, including GPUs [37]. Due to abundant data parallelism inherent in our approximation techniques, parallel implementations of the techniques demonstrate excellent performance scalability as parallelism increases. We implemented and evaluated our approximations on an NVIDIA GeForce GTX480 GPU to measure performance scalability. On average, for different distances of approximation ranging from 2 to 32, parallelization reduces approximation time by 80-230×. While more efficient parallel implementations may exist, even our first-effort parallelizations indicate that (1) parallelization can significantly accelerate our approximation techniques and (2) time to perform approximation should continue to decrease as trends in increasing parallelism continue.

Mobile platforms are also increasingly accelerator-rich, even including hardware accelerator support for conventional lossy compression (JPEG) [38]. Given the increasing trend toward specialization and hardware acceleration motivated by energy efficiency demands, as well as dark silicon [39], we argue that mobile platforms should also include hardware support for lossless compression. Accelerators for lossless compression have been shown to achieve higher throughput than those for lossy compression and can be implemented at lower area and power cost [40, 41]. Furthermore, as we will show in Section 6, approximation followed by lossless compression may often achieve better compression rates

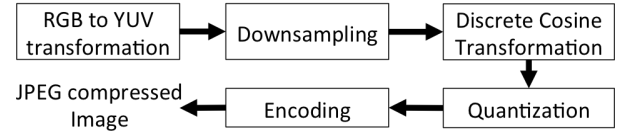


Figure 7. JPEG approximates DCT coefficients corresponding to high frequency components in an image, due to the resulting imperceptibility of distortions.

than conventional lossy techniques, while improving end-to-end data transmission performance for the same output quality. Given that mobile platforms already include accelerators for lossy compression, our results provide strong motivation for acceleration of lossless compression.

6 Results and Analysis

6.1 Exploring the trade-off between output quality and compression rate

This section evaluates the trade-off between the compression rate achieved by the proposed approximation techniques followed by lossless compression and the output quality of the decompressed image. We compare our approximate compression techniques against (1) an elementary approximation technique – sampling the pixels of the image – as well as the state-of-the-art in lossy image compression – JPEG. Image sampling is implemented by selecting the first value in each group of N pixels as the representative value, where N corresponds to the distance of approximation. The performance of the proposed approximate compression techniques are expected to be at least as good as the elementary sampling technique, since the approximations are input-aware.

As shown in Figure 7, JPEG transforms pixel data into the frequency domain using DCT and exploits the human visual system’s reduced sensitivity to distortions in fine image details, such as edges, to identify regions in which distortions can be introduced. Distortions are introduced in the quantization phase by eliminating DCT coefficients corresponding to high-frequency information and preserving low-frequency information. The extent of approximation can be controlled by specifying the desired output quality.

For the sake of analysis, we classify the benchmark images into three categories, based on the characteristics described in Table 1 – (1) autokinetic images, (2) still images dominated by smooth textures, (3) still images dominated by complex textures.

Autokinetic images: Autokinetic images, though synthetically generated, can be classified as continuous-tone images due to their qualities of blur and noise. Hence, the introduction of approximations to these images could take advantage of the a user’s inability to identify distortions in the image, based on the viewing distance.

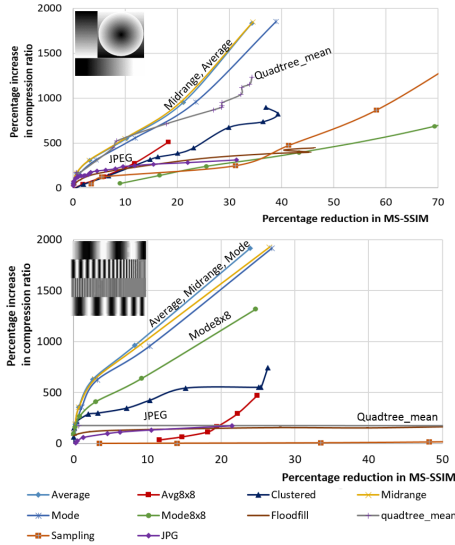


Figure 8. JPEG’s compression rate does not improve significantly, even with high quality degradation. Approximate compression techniques continue to increase compression rate as higher output quality degradation is allowed. *Top: slope Bottom: cosinpattern*

Figure 8 shows the percentage by which compression rate can be increased for a given allowable reduction in MS-SSIM, for the different approximation techniques, sampling, and JPEG. Lossless compression of average- and midrange-based approximations achieves the best compression rate for most of the range of MS-SSIM, closely followed by mode-based approximation. To ensure that compression algorithm performance is not image-specific, the above trends are verified with another autokinetic image – *cosinpattern* [42]. The improvement in compression rate is about 7x at 5% quality reduction and up to 10x at 23% reduction in MS-SSIM for average and midrange-based approximation of autokinetic images.

The presence of regions with uniform pixel values in the image *slope* aids quadtree-based approximation, in contrast with the image *cosinpattern*. Clustering-based approximation performs well because it exploits the uniformity in pixels along the horizontal scan-lines to improve the compression rate.

The DCT coefficients for autokinetic images, shown in Figure 10 are predominantly at 0 or far away from 0. Since JPEG achieves higher compression by approximating coefficients that are close to 0 but not at 0, JPEG cannot provide higher compression, even by sacrificing considerable quality. For the image *slope*, JPEG’s compression rate does not improve substantially even with large quality degradation. For example, in Figure 8 (a), the improvement in compression rate corresponding to a reduction in MS-SSIM from 10% to 30% is only 1.3x. The compression rate of JPEG begins to saturate with further degradation in output quality, due to the lack of coefficients that can be approximated, as seen in Figure 10 (a) (b).

Images with smooth textures: The still images *bird*, *lena*, and *camera* are dominated by the presence of smooth textures, spanning small and large regions. The minor presence of coarse textures help to camouflage the distortions introduced by approximations in the feathers and body of the bird in the *bird* image, the hat and background in *lena*, and the ground in *camera*. Since the images are dominated by regions of smooth textures, quadtree-based approximation provides good compression, since larger low-variance blocks exist in the images. Unlike the large, smooth textured regions in *bird*, the image *lena* has smaller smooth textured regions, negatively impacting the performance of quadtree-based approximation, pushing the cross-over point between JPEG and quadtree-based approximation to 20% reduction in MS-SSIM.

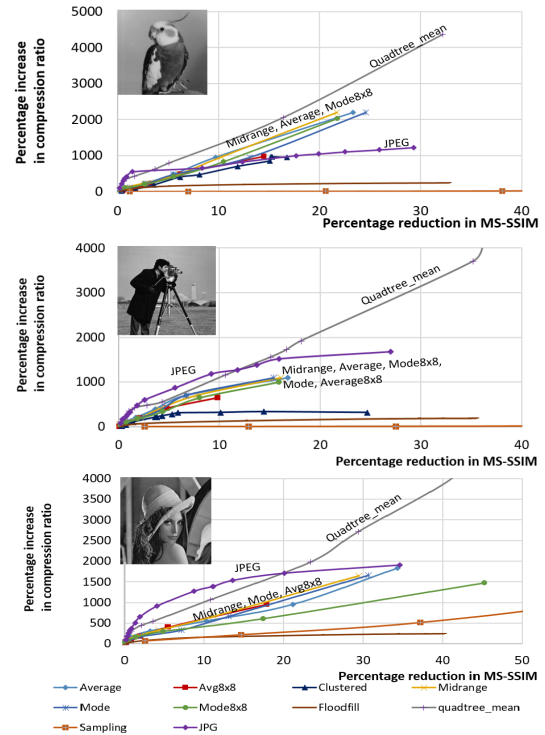


Figure 9. High-frequency image components allow JPEG to introduce imperceptible losses and achieve a good trade-off between compression and output quality. As quality is degraded further, the distortions start to affect perception, and quadtree-based approximate compression achieves a higher compression rate for the same output quality. *From top to bottom: (a) bird (b) camera (c) lena*

High-frequency components resulting from edges, i.e., sharp transitions in intensity values, allow JPEG to provide better compression. From Figure 10 (c) (d) (e), which show the distribution of DCT coefficients for the images, the order of increasing opportunity for benefits with JPEG is *bird*, *camera*, *lena*, based on their scope of approximation. It can be observed from Figure 9 that with the decrease in high frequency components, the range where JPEG is the best compression technique decreases – from up to 20% reduction in MS-SSIM in *lena*, to only up to 3% reduction in MS-SSIM in *bird*. Beyond this region of dominance, JPEG sees little increase in compression rate (only 1.19x) as MS-SSIM decreases from 1.5% to 8.4% for *bird*. Quadtree-based approximation, on the other hand, allows a 1.5x increase in compression rate for only 2% loss in quality in this region of approximation.

Images with coarse textures: The still images *bridge* and *goldhill* are composed of complex textures, which contain sharp transitions in intensity, providing the opportunity for JPEG to achieve superior compression with relatively less loss in perception of visual quality. The dominance of complex textures in the images helps hide distortions effectively when high-frequency components are quantized. From Figure 10 (f) (g), these manifest in the frequency domain as DCT coefficients close to 0, that can potentially be approximated to 0, improving image compressibility. JPEG achieves the best compression rate for up to 25% reduction in quality in *goldhill* and 32% reduction in quality in *bridge*. However, there is a significant drop in quality between successive approximation points, as we move toward the lower quality region. For example, on the JPEG trendline in Figure 11, a quality setting of 3 for JPEG gives an MS-SSIM value of 0.8, whereas a quality setting of 2 causes MS-SSIM to plummet by 56% to 0.35.

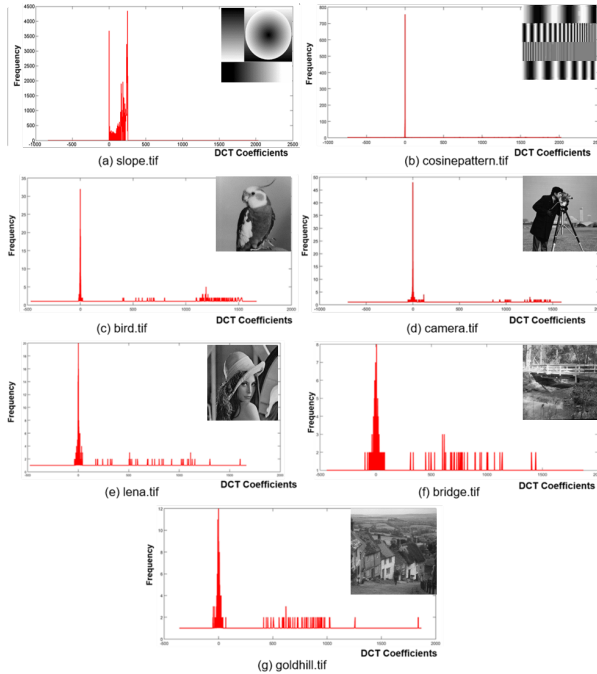


Figure 10. Autokinetic images do not present much scope for JPEG approximation due to the limited presence of DCT coefficients near 0, corresponding to high-frequency components. From top left: Distribution of DCT coefficients for (a) *slope* (b) *cosinepattern* (c) *bird* (d) *camera* (e) *lena* (f) *bridge* (g) *goldhill*

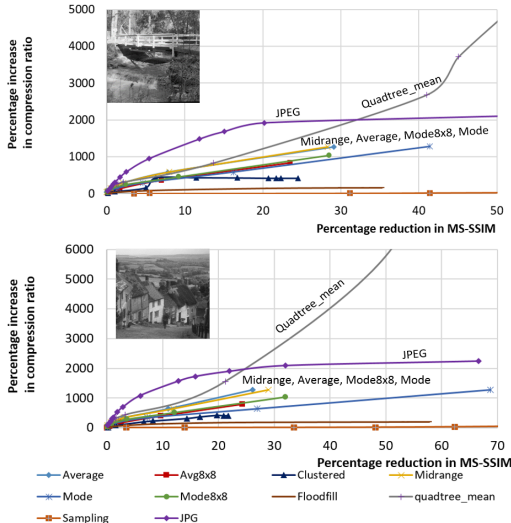


Figure 11. The cross-over point between JPEG and quadtree-based average compression shifts to the right compared to the observations from Figure 9, implying an increase in the presence of high-frequency components.

With such complex textures, quadtree-based approximation performs poorly for high quality requirements, since it is not possible to identify large regions with similar pixel values. When quality requirements are relaxed beyond the cross-over points defined above, the quadtree-based approximation scheme provides 1.3x improvement in compression for 7% reduction in MS-SSIM. For all image categories above, floodfill and clustering-based approximations do not perform well, since they target better output quality, resulting in higher entropy and less scope for improving compressibility.

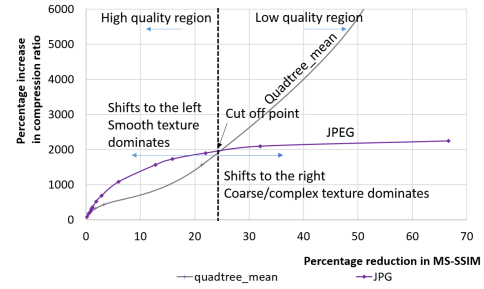


Figure 12. JPEG provides better compression with imperceptible losses for images dominated by high-frequency components. Quadtree-based approximation increases compressibility more when images contain larger smooth regions.

6.2 Recommending approximation techniques based on image characteristics

In this work, we show that the approximate or lossy compression technique that maximizes compression rate for a given image depends on image characteristics and desired output quality. For example, autokinetic images lack high-frequency components that can be quantized by JPEG, so JPEG does not achieve a high compression rates for autokinetic images. Instead, the proposed average- and midrange-based approximations followed by lossless compression provide the best quality vs. compression rate trade-off.

For still images, the choice between compression schemes can be made based on the proportion of high-frequency components in the image and the required application output quality. If only minimal output quality degradation is permissible and the image has many high frequency components, JPEG is the best choice. If higher quality degradation is permissible, quadtree-based approximation maximizes gain in compression rate per unit loss in quality.

The slope of the compression rate vs quality reduction curve varies with the relative composition of edges and smooth textured regions in an image. When an image has many edges, the range of MS-SSIM over which JPEG is the best compression technique increases, since JPEG approximates the high-frequency components in an image, providing higher compression with less degradation of perceived visual quality. In scenarios where image quality degradation is more acceptable, quadtree-based approximation is able to identify large regions with near-uniform pixel values and achieve better compression rates. The conditions for dominance of JPEG and quadtree-based approximation are illustrated in Figure 12.

While the guidelines above provide adequate criteria for selecting an appropriate approximate / lossy compression technique for a given scenario, ideally, the selection of the best approximation technique for a given image should be automated. One solution for automating the selection process is machine learning. We have implemented a convolutional neural network (CNN) using the TensorFlow platform [43], retrained on the inception_v2 model [44, 45], which pretrains a network for generic image classification, such that only the output layer needs to be retrained for the specific classification task. After using a small training set of only 100 images to retrain the network, our CNN is able to predict, with perfect accuracy, the approximation technique that maximizes compression rate for a given output quality for all input images in our test set.

6.3 End-to-end Performance Analysis

The goal of approximate compression is to improve communication performance on bandwidth-constrained channels. In this section, we compare the proposed approximate compression techniques against state-of-the-art lossy compression in terms of end-to-end transmission rate.

Table 3. Average time taken to approximate the benchmark images.

Approximation Technique	Compression Rate	Approximation Time (ms)
Average	19.75	0.1347
Mode	19.80	0.0593
Average8x8	9.43	0.1197
Mode8x8	16.63	0.0315
Midrange	19.82	0.0315
Cluster/Flood	5.17	0.3543
Quadtree	22.84	0.6332

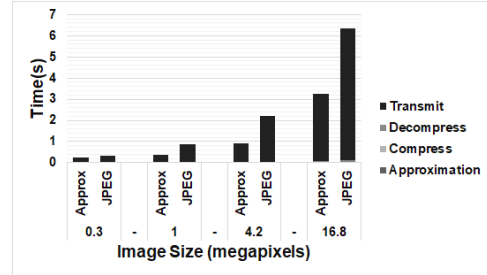
Table 4. Average time taken to compress and decompress the benchmark images using lossless compression algorithms.

Compression Algorithm	Switches	Compression rate	Time to compress (s)	Time to decompress (s)
7zip	-m0=LZMA2	7.916	0.035	0.034
7zip	-m0=PPMd	9.031	0.031	0.047
BZIP2	-best	9.474	0.025	0.019
CMIX		16.175	35.216	35.030
GZIP	-best	9.313	0.017	0.016
INFOZIP	-best	8.497	0.028	0.020
LHA		8.967	0.016	0.013
LZ4	-9	4.371	0.010	0.008
NANOZIP (LZHD)	-cd	9.175	0.025	0.042
LZIP	-best	10.963	0.039	0.019
LZOP	-9	4.762	0.011	0.073
NANOZIP (NZCM)	-cc	11.858	0.205	0.203
PAQ 8o6	-7	14.103	2.736	2.718
RAR	-m5	7.143	0.053	0.024
XZ		10.659	0.035	0.023
ZLIB		8.645	0.015	0.039
ZPAQ	-best	8.626	0.178	0.192
ZZIP	-mx	10.002	0.019	0.016

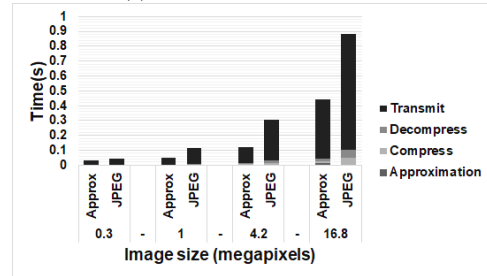
Time to Approximate: Table 3 shows the approximation time for parallel implementations of our approximation techniques, averaged over ten trials, and the compression rate achieved by compressing the approximate images using bzip2. Results are shown for an approximation distance of 32, which resulted in the longest approximation time in our evaluations. Approximation time is averaged over all the benchmark images. A comparison of approximation time (Table 3) and compression time (Table 4) shows that the maximum observed overhead of data approximation is negligible compared to that of compression. This is not surprising, considering that our approximation techniques are highly parallelizable, whereas compression algorithms tend to be fundamentally sequential in nature (though accelerators for compression have been shown to boost compression throughput significantly [40, 41]).

Time to compress and decompress: Table 4 shows compression and decompression times, averaged over ten trials, along with the compression rates achieved for different lossless compression algorithms. The table shows average results over all the approximated benchmark images. CMIX achieves the highest compression rate for all the benchmark images. However, it also has the highest runtime, as it estimates the probability of occurrence of a symbol by combining the results from several different modeling schemes. For latency-critical applications, where fast access times may be more important than maximizing compression rate, the high latency of an algorithm like CMIX may be unacceptable. This is likely the case for many communication-centric applications, since long-latency decompression could result in undesirable buffering time at the destination. As such, we use a different algorithm for our evaluations (bzip2), which has significantly lower latency than CMIX, while still providing fairly comparable compression rates.

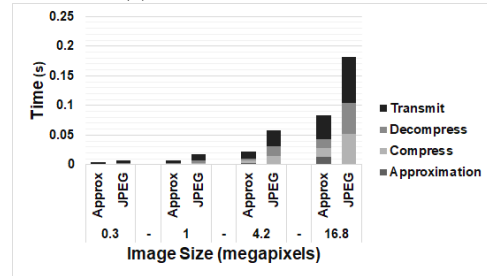
End-to-end Transmission Time: Approximate communication based on approximate or lossy compression involves data compression at the source, transmission of compressed data, and data



(a) 2G network



(b) 3G network



(c) 4G network

Figure 13. Approximate compression reduces end-to-end transmission time compared to state-of-the-art lossy compression (JPEG). Although the relative overhead of compression increases for newer wireless technologies, reduction in data transmission time afforded by compression substantially outweighs compression overhead.

decompression at the destination. In the case of approximate compression, data is approximated prior to compression to increase compressibility. Figure 13 compares end-to-end data transmission time for approximate communication based on the proposed approximate compression techniques against JPEG for various image sizes.⁵ Comparisons are shown for an output quality threshold of 85%. The subfigures in Figure 13 provide comparisons for different wireless network generations with different average data rates [46]. Due to the significant gains in compression rate afforded by data approximation, the increased throughput of lossless compression over lossy compression, and the negligible overhead of data approximation, approximate compression significantly reduces end-to-end transmission time compared to state-of-the-art lossy compression (JPEG). Averaged across different file sizes and network generations, approximate compression reduces transmission latency by 2.1× compared to JPEG, for the same output quality.

Compressed communication makes sense when the reduction in communication latency afforded by transmitting compressed data is greater than the overhead of performing compression and decompression. Because approximate compression significantly improves data compressibility, using approximate communication

⁵Time comparison is also a good proxy for energy, especially for computational stages that represent overheads of approximate compression (e.g., approximation, compression, decompression).

Table 5. Speedup of end-to-end communication time achieved through approximate and lossy compression, compared to communication of uncompressed data.

image size (MP)		2G	3G	4G
0.3	Approx	8.0×	8.0×	7.4×
	JPEG	5.9×	5.8×	5.3×
1.0	Approx	22.4×	21.8×	17.1×
	JPEG	9.2×	9.1×	7.8×
4.2	Approx	133.7×	124.8×	74.3×
	JPEG	53.2×	48.8×	26.3×
16.8	Approx	146.7×	136.1×	78.1×
	JPEG	74.4×	66.0×	30.6×

based on approximate compression for error-tolerant applications significantly outperforms communication of uncompressed data, even considering the overheads of approximation, compression, and decompression. Table 5 shows speedups for approximate communication based on approximate and lossy compression over communication of uncompressed data. Both lossy and approximate compression significantly improve communication performance compared to no compression, and approximate compression outperforms lossy compression by over 2×, on average. Although the relative overhead of compression increases for newer wireless network generations with higher bandwidth, the significant reduction in data transmission latency afforded by compression substantially outweighs compression and approximation overhead.

7 Conclusion

Trends toward mobile, communication-centric processors and applications are increasingly straining available communication bandwidth. In this paper, we leverage the inherent noise tolerance of the most bandwidth-constrained applications (e.g., multimedia streaming) to perform approximate compression and communication. We introduce a suite of characteristic-aware approximate compression techniques that enhance the performance of lossless compression algorithms and allow fluid trade-offs between compression rate and output quality. We show that conventional lossy compression techniques like JPEG may not always achieve the best compression rate for a given level of output quality, and the best approximate compression technique for a given scenario depends on desired quality and image features. Our approximate compression techniques show up to 10× improvement in compression rates over state-of-the-art lossy compression, for the same output quality, demonstrating the importance of selecting the best approximate compression technique for a given scenario. We showed that this selection can be automated through learning-based techniques such as neural networks. Our approximate compression techniques improve end-to-end performance of data transmission by 2×, on average, compared to state-of-the-art lossy compression. Furthermore, our approximate compression techniques scale well on massively parallel processors, indicating that the cost of performing approximate compression before data transmission should continue to decrease in the future as systems become more parallel.

References

- [1] Henry Blodget, Marcelo Ballve, Tony Danova, Cooper Smith, John Heggstuen, Mark Hoelzel, Emily Adler, Cale Weissman, Hope King, Nicholas Quah, John Greenough, and Jessica Smith. The internet of everything: 2015. *BI Intelligence*, 2014.
- [2] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. April 2011.
- [3] Gil Press. Internet of Things By The Numbers: Market Estimates And Forecasts. *Forbes*, 2014.
- [4] Microcontroller Sales Regain Momentum After Slump. www.icinsights.com/news/bulletins/Microcontroller-Sales-Regain-Momentum-After-Slump.
- [5] Responsive web design: Why image optimization is crucial for a mobile friendly experience.
- [6] Sandvine Incorporated ULC. Global internet phenomena. June 2016.
- [7] David Salomon. *Data Compression: The Complete Reference*. Springer, 4th edition, December 2006.
- [8] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.
- [9] Gregory K. Wallace. The JPEG still picture compression standard. *Commun. ACM*, 34(4):30–44, April 1991.
- [10] A. Skodras, C. Christopoulos, and T. Ebrahimi. The JPEG 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5):36–58, September 2001.
- [11] Google. WebP <https://developers.google.com/speed/webp/>, 2016.
- [12] Krisda Lengwehasatit and Antonio Ortega. Scalable variable complexity approximate forward DCT. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(11):1236–1248, November 2004.
- [13] R. J. Cintra and F. M. Bayer. A DCT approximation for image compression. *IEEE Signal Processing Letters*, 18(10):579–582, February 2014.
- [14] Soontorn Oraintara, Ying-jui Chen, and Truong Nguyen. Integer fast fourier transform (INTFFT). In *IEEE Trans. on Signal Processing*, 2001.
- [15] R. J. Cintra. An integer approximation method for discrete sinusoidal transforms. *Circuits, Systems, and Signal Processing*, 30(6):1481–1501, May 2011.
- [16] F Keissarian. A new quadtree-based image compression technique using pattern matching algorithm. In *International Conference on Computational & Experimental Engineering and Sciences (ICES)*, volume 12, pages 137–143, 2009.
- [17] The waterloo image repository. <http://links.uwaterloo.ca/Repository.html>.
- [18] 7ZIP 15.09 <http://www.7-zip.org/download.html>.
- [19] bzip2. <http://www.bzip.org/1.0.6/bzip2-1.0.6.tar.gz>.
- [20] CMIX -v7 <http://www.byronknoll.com/cmixon.html>.
- [21] GZIP 1.6 <http://ftp.gnu.org/gnu/gzip/>.
- [22] INFOZIP <http://www.info-zip.org/pub/infozip/Zip.html>.
- [23] LHA 1.14i <http://www2m.biglobe.ne.jp/~dolphin/lha/lha-download.htm>.
- [24] lz4. <https://code.google.com/p/lz4/>.
- [25] Nanozip 0.09. <http://nanozip.net/>.
- [26] LZIP 1.17 lossless data compressor <http://www.nongnu.org/lzip/lzip.html>.
- [27] LZOP 1.03 www.lzop.org/download/lzop-1.03.tar.gz.
- [28] PAQ8o6 <http://www.mattmahoney.net/dc/>.
- [29] RAR 5.21 <http://www.rarlab.com/download.htm>.
- [30] XZ 5.2.1 <http://tukaani.org/xz/>.
- [31] ZLIB 1.2.8 compression library <http://zlib.net/>.
- [32] ZPAQ 7.05 <http://www.mattmahoney.net/dc/zpaq.html>.
- [33] ZZIP v0.36c <http://archives.damiendebin.net/zzip/download.php>.
- [34] NConvert for png and gif <http://www.nxview.com/en/nconvert/>.
- [35] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, April 2004.
- [36] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1398–1402 Vol.2. IEEE, November 2003.
- [37] NVIDIA Corporation. Nvidia tegra x1, nvidiaf1s new mobile superchip. 2015.
- [38] L. Codrescu, W. Anderson, S. Venkumanhanti, M. Zeng, E. Plondke, C. Koob, A. Ingle, C. Tabony, and R. Maule. Hexagon dsp: An architecture optimized for mobile multimedia and communications. *IEEE Micro*, 34(2):34–43, Mar 2014.
- [39] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor. The greendroid mobile application processor: An architecture for silicon’s dark future. *IEEE Micro*, 31(2):86–95, March 2011.
- [40] Jeremy Fowers, Joo-Young Kim, Doug Burger, and Scott Hauck. A scalable high-bandwidth architecture for lossless compression on fpgas. In *Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM '15*, pages 52–59, Washington, DC, USA, 2015. IEEE Computer Society.
- [41] Mohamed S. Abdelfattah, Andrei Hagiescu, and Deshanand Singh. Gzip on a chip: High performance lossless data compression on fpgas using opencl. In *Proceedings of the International Workshop on OpenCL 2013 & #38; 2014, IWOCCL '14*, pages 4:1–4:9, New York, NY, USA, 2014. ACM.
- [42] Carsten Glasenapp and Hans Zappe. Frequency analysis of maskless lithography system for generation of continuous phase patterns with homogeneous structure depth. *Optical Engineering*, 47:47 – 47 – 7, 2008.
- [43] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudrur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.
- [44] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.
- [45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.
- [46] Comparison of wireless data standards. https://en.wikipedia.org/wiki/Comparison_of_wireless_data_standards.