CONTEXT ADAPTIVE THRESHOLDING AND ENTROPY CODING FOR VERY LOW COMPLEXITY JPEG TRANSCODING

Xing Xu†, Zahaib Akhtar‡, Ramesh Govindan†, Wyatt Lloyd†, Antonio Ortega‡

† Department of Computer Science, ‡ Department of Electrical Engineering
University of Southern California

ABSTRACT

The ever increasing quantity of user generated photos, nearly all compressed using JPEG, has created a growing storage burden on photo storage and sharing services. This creates the need for compression techniques that take JPEG compressed images as inputs. In this paper we propose two novel very low complexity codecs, ROMP and L-ROMP to recompress JPEG photos, achieving increased coding efficiency by making use of very large entropy coding tables. ROMP is a lossless JPEG recompression codec that achieves 15% average gains over JPEG, while L-ROMP is a lossy codec that can achieve 29% average compression gains over JPEG, by applying coefficient thresholding based on a perceptual criterion to a JPEG image before using the entropy coding of ROMP.

Index Terms— JPEG, image compression

1 Introduction

With the wide availability of digital cameras, users are taking more photos, at higher resolution and quality, than ever before. Huge volumes of images are then stored, and shared with other users, leading to a dramatic growth in the popularity of services such as Facebook, Flickr, and Instagram that provide image storage. For the purpose of this paper we distinguish archival from non-archival photo storage services. In the former, captured images are stored at their original (very high) quality settings, providing essentially extended or back-up storage for digital cameras. Instead, in the latter, images are stored in order to be shared; e.g., blogs, online social networks, email, chat, etc. As an example of such a non-archival storage, Facebook alone had 350 million photo uploads per day and stored over 250 billion photos in 2013 [1]. In non-archival applications photos are often resized and stored at lower quality in order to reduce bandwidth and storage needs. Still, given the huge volumes involved, photo storage costs can represent a significant burden on these services. Also, while theoretically users could choose to remove these images some time after they have been made shared, in practice, many users keep them there indefinitely, adding to the storage costs.

In this paper we propose techniques to decrease significantly the image storage requirements, with minimum impact on quality and access latency. We focus on non-archival storage services. In this type of service, when a user uploads photos, a transcoder typically reduces the photo file-size by resizing, reducing quality (e.g., through re-quantization), stripping off headers or a combination thereof. Some services may also transcode images to other more storage-efficient formats such as WebP or JPEG2000. When a user sends a request to download a previously uploaded photo, the transcoder converts the photo from the format used for storage in backend, to one suitable for delivery to a user device (adjusting image size and quality as needed).

Our proposed approach specifically targets reduction of storage costs, but also leads to additional benefits (e.g., reduction of internal bandwidth required for transferring images*) and is in general designed so that it can be seamlessly combined with existing systems. This has three major implications. First, as JPEG[3] is the dominant format for image capture and sharing, our techniques will take JPEG images as input and provide JPEG images as decoded output. Second, most systems already transcode uploaded images to lower quality. Thus, we will apply our method to these lower quality images, rather than to the high quality JPEG images generated by cameras. Third, we will aim at very low complexity, in order to minimize latency on the download path and to lower transcoding cost and increase scalability in large scale systems.

Two types of re-encoding approaches could be considered, namely, i) techniques closely associated to JPEG, usually providing lossless performance, e.g., JPEG Progressive, JPEG Arithmetic or PackJPG, ii) alternative coding techniques based on more recently developed codecs, such as WebP or JPEG2000. We do not consider the latter approaches because these codecs are not as widely supported as JPEG among typical client devices, and would require excessive transcoding overhead (e.g., more than a second) if they were used solely for storage (and JPEG images had to be sent to the client devices). This will be discussed in Section 4.

Our approach leverages two unique attributes of non-archival photo storage. First, the encoder (at image upload) and decoder (at image download) are co-located and are both part of the storage infrastructure; second, constraints on memory usage can be relaxed. Based on these two facts, we introduce a novel JPEG recompression technique. ROMP (Recombination Of Many Photos) that is a direct extension of JPEG, but makes use of a large number of context-dependent Huffman tables. Because encoder and decoder are co-located, these tables can be stored along with the images, leading a negligible storage overhead in a large scale system. These tables allow ROMP to trade-off runtime memory for encoding speed, achieving some of the benefits of context adaptation but without the complexity associated to techniques such as context adaptive arithmetic coding.

To achieve additional gains, we also introduce L-ROMP (Lossy-ROMP), a fast, lossy recompression technique. Traditional recompression techniques (e.g., re-quantizing by changing JPEG’s quality parameter) degrade image quality sub-optimally, especially when incremental storage savings are required (due to rounding effects when applying successive quantization). L-ROMP is based on DCT coefficient thresholding (setting some of the coefficients to 0), has very low complexity and allows full control of the addi-
tional distortion introduced. We introduce techniques for L-ROMP that use an approach based on visual perception criteria in order to choose the thresholds in order to minimize the perceptual impact of thresholding.

To evaluate ROMP and L-ROMP, we perform experiments on several image sets and show that ROMP reduces storage requirements by 15% beyond the JPEG standard compression while having low decompression time (30-50ms). Its performance dominates all image compression techniques we have been able to find: its competitors either have much higher decoding complexity, or much lower compression gain. L-ROMP can, in some cases, reduce storage requirements by 29% with a comparable perceptual quality.

There exist some prior works that focus on lossless compression [4, 5, 6, 7, 8]. Compared to these schemes, ROMP, can achieve competitively high compression at very low complexity. Transcoding from JPEG to other compressed formats, such as WebP [9] or JPEG2000 [10] normally introduces high complexity due to transforming to pixel domain and encoding to the new format. L-ROMP avoids these conversions and always stays in DCT domain, which has advantages in terms of complexity. As will be described, successive changes of quantization parameters in JPEG can introduce significant quality degradations [11]. L-ROMP, instead, only introduces small and perceptually lossless changes. L-ROMP is inspired by prior work on thresholding which optimizes the rate/distortion tradeoff [12, 13, 14], but is simpler and more computationally efficient.

2 Context-Adaptive Lossless Coder

In order to satisfy the system design constraints, ROMP makes use of a large set of entropy coding tables generated from a collection of images, where each of the tables is optimized for a specific context. Our approach proceeds block by block and does not involve any transformations or re-orderings of DCT coefficients, essentially using entropy tables that have exactly the same structure as that of a typical JPEG entropy coding table. This ensures that the complexity of the system is very low (see Fig. 1): essentially equivalent to JPEG entropy decoding followed by JPEG entropy coding.

As will be discussed next, in order to improve coding efficiency we take advantage of statistical dependencies that are exploited by other mechanisms (e.g., intra prediction, adaptive arithmetic coding) in state-of-the-art codecs. The key difference is that by exploiting these dependencies using a large number of fixed pre-computed tables (which increases memory requirements for the codec) we have significantly lower complexity than competing approaches. Note that this increase in memory is reasonable within a high performance photo sharing system, where there are essentially few constraints on memory usage.

Context-Sensitive Coding. ROMP exploits the freedom to have large coding tables by designing context-sensitive coding tables that result in lossless\(^b\) compression. Recall that JPEG’s Huffman tables are used to code runsizes, that is, information about a run of consecutive zeros followed by a non-zero coefficient of a given size. Huffman codes for these runsizes are designed based on their expected frequency of occurrence based on average image statistics. ROMP learns context-sensitive Huffman tables by learning the empirical probability of occurrence of runsizes from a corpus of images. This learning leverages the availability of such corpora in a large-scale photo sharing service. A Huffman table will contain the set of variable length codes assigned to each of the possible runsizes values, and each table will be optimized for a context based on position and energy information.

Position Dependence. Specifically, we consider position dependent tables, so that in principle a different table could be used for each of DCT coefficient position along the zig-zag scan. This allows us to use codes that exploit the fact that longer run-lengths are more likely as the frequency increases. This follows from the fact that, for natural images, it is known that non-zero coefficients are increasingly unlikely at higher frequencies [18].

Energy Dependence. Furthermore, we also take into account energy-dependence by creating additional contexts (for each position) based on the energy of other coefficients within the block (intra-block energy) and of neighbouring blocks (inter-block energy). Using intra-block energy provides additional information beyond position alone, e.g., in a block with less accumulated energy up to position \(p-1\), the coefficient at position \(p\) is likely to be smaller. Inter-block energy exploits similarity between neighbouring blocks and is likely to be more effective for high resolution images, for which an \(8 \times 8\) represents a relatively small portion of the image and neighbouring blocks are more likely to be similar.

To reduce the computation cost, for a given runsize that occurs at zigzag position \(p\) of the \(n\)-th block, we use the average of the observed coefficient sizes in a block as an estimate of intra-block energy:

\[
\text{intra}(n, p) = \frac{1}{p-1} \sum_{i=1}^{p-1} \text{SIZE}(b_n(i))
\]

where \(b_n\) denotes the \(n\)-th block, and \(b_n(i)\) denotes the coefficient at position \(i\), \(\text{SIZE}(\cdot)\) denotes the bits required to represent the amplitude of the coefficient, \(\text{maxSIZE}(\cdot)\) is the observed maximum coefficient size for position \(i\) of images in the training set. Similarly, the inter-block energy value is estimated based on the average sizes of coefficients in nearby blocks: \(F\) nearby zigzag positions in \(B\) adjacent prior blocks (coefficients at positions \(p, p+1, \cdots, p+F-1\) of blocks \(b_{n-1}, b_{n-2}, \cdots, b_{n-B}\)):

\[
\text{inter}(n, p) = \frac{1}{B \cdot F} \sum_{i=n-B}^{n-1} \sum_{j=p}^{p+F-1} \frac{\text{SIZE}(b_{i}(j))}{\text{maxSIZE}(j)}
\]

Learning the Context-Sensitive Tables. ROMP uses a triple \(< p, i, e >\) to define context: zigzag position \(p\), intra-block energy \(i\) and inter-block energy \(e\). It generates a Huffman table for each of these contexts from a training set of images. For runsize that occurs in any image in the training set, ROMP first determines its context triple and then gathers it together with other runsizes belonging to the same context. After gathering all the runsizes for each context, ROMP can generate a table for this context based on the number of occurrences of each, including runsizes that have 0 occurrence; if all the runsizes in one context are 0 occurrence, then a default Huffman table will be used. ROMP pre-defines 20 different energy levels for both intra-energy and inter-energy, which leads to \(64^2 \times 20 \times 20 = 25600\) different contexts and Huffman tables to be learned.\(^c\) These

\(^a\)Lossless with respect to the uploaded JPEG image.

\(^b\)These tables take up less than 7MBs of the memory, a negligible memory usage increment for modern machines.

\(^c\)We use \(F = 5\) and \(B = 3\) in ROMP.

\(^d\)There are 64 different zigzag positions in a \(8 \times 8\) block.
tables are quite different: take the example of End-Of-Block (EOB) symbol, which always takes 4 bits in default JPEG huffman table, ROMP’s table of context $< 5.0.0 >$ uses 1 bit for it, while context $< 2.5.9 >$ uses 9 bits! These different Huffman tables allow ROMP to achieve better compression over standard JPEG.

Using Context-Sensitive Tables for En-/Decoding. Given the learned Huffman tables, the en-/decoding of ROMP is easy: ROMP parses the decoded symbols and for every runsize it computes the corresponding triple $< p, i, e >$, based on causal information, and uses the table corresponding to that context to encode runsize.

In summary ROMP operates as follows (Fig. 1): 1) From a training set of images, ROMP learns a Huffman table for each unique context (i.e., for each unique combination of position, intra- and inter-block energy). 2) When an image is uploaded, ROMP decodes it using default JPEG table, then uses the learned context-adaptive entropy tables to re-code the image and 3) Before delivering the image to the user, ROMP reverses its context-adaptive entropy code, and then applies the default JPEG entropy code.

3 Fast Near-Lossless Thresholding

ROMP’s entropy coder is lossless with respect to the uploaded JPEG. In this section, we describe L-ROMP, which introduces loss (or distortion) in uploaded images as a way of achieving further savings in photo storage.

As discussed previously, users upload high quality JPEG images and many photo sharing services, e.g., Facebook, change the JPEG quality parameter ($Q_P$) to a lower level in order to ensure predictable storage usage, a step that introduces additional distortion [11]. Fig. 2 shows how generating a JPEG image with $Q_P$ in two steps (i.e., encoding first with $Q_P$, and then requantizing to $Q_P$) can be significantly worse than encoding directly the original with $Q_P$. In Fig. 2, “raw” corresponds to encoding the original raw image (raw $\rightarrow Q_P$), while “JPEG” corresponds to re-encoding a JPEG image (raw $\rightarrow Q_P$). This penalty arises from the rounding effects when a quantized coefficient is first reconstructed, then divided by a different quantization parameter and again quantized.

L-ROMP avoids re-quantizing coefficients, but introduces distortion by carefully setting some non-zero (quantized) coefficients to zero, a specific example of thresholding [12]. While more general forms of thresholding have been explored in other contexts, we are not aware of it being considered as an alternative to re-quantization in large photo sharing services. The intuition behind thresholding is that, by setting a well-chosen non-zero coefficient to zero, we decrease the number of runsizes to be encoded or generate an earlier end of block, in both cases reducing the overall rate. Optimization of coefficient thresholding has been considered from a rate-distortion perspective in the literature [16, 12]. Here we use a simplified version where only coefficients of size equal to 1 (i.e., 1 or $-1$) can be removed. This means that the distortion increase for any coefficient being removed will be the same $^6$. Thus, we only need to decide if for a given coefficient the bit-rate savings are sufficient to remove it. We make the decision by introducing a rate threshold and only thresholding a coefficient if the bits saving by doing so would exceed this threshold.

However, setting too many coefficients to zero within a block can introduce local artifacts (e.g., blocking). Thus, L-ROMP uses a perceptual threshold $T_p$ that limits the percentage of non-zero coefficients that will be set to zero. By doing this L-ROMP can guarantee that the block-wise SSIM respect to the original JPEG is always higher than $1 - \frac{T_p}{s}$. For example, if we use $T_p = 0.1$ (i.e., we can threshold at most 10% of the non-zero coefficients), then block-wise SSIM metric is guaranteed to be higher than 0.947. The proof of such bound is based on results from [17] and is omitted for brevity.

Finally, L-ROMP can be easily introduced into ROMP’s pipeline: before applying the context-sensitive entropy coding, L-ROMP’s thresholding can be applied to each block (Fig. 1). No changes are required to ROMP’s entropy coder.

4 Evaluation

In this section, we evaluate ROMP and L-ROMP, by comparing their compression and complexity performance to other state-of-the-art alternatives. Our evaluations use an implementation of ROMP and L-ROMP on top of libjpeg-turbo [18], a fork of the Independent JPEG Group libjpeg [19] SIMD [20] accelerated C library JPEG codec. Our implementation is optimized to reduce complexity, for example, by caching intermediate results. We compare ROMP and L-ROMP to all lossless JPEG codecs we are aware of that have publicly available implementations, including JPEG Standard, JPEG Optimized, JPEG Progressive, JPEG Arithmetic, MzsJPEG [21] and PackJPEG [22, 8]. For lossy codecs, we include WebP and JPEG2000 into comparison.

Our evaluation uses two sets of images. First we use the MIT-Adobe FiveK Dataset [23], which contains 5000 photos in raw format taken by SLR cameras. We transcode them to quality parameter 75, at two resolutions: 1152×864 and 2048×1536, in order to simulate photos typically uploaded to photo sharing services. We also use the Tecnick image set [24]: 100 images of maximum resolutions 1200×1200 in raw format (PNG). For ROMP and L-ROMP, we randomly select a group of images for training and test on the rest of the images, e.g., for FiveK sets, we train on 1,000 and test on 4,000 images.

We evaluate our recompression schemes and alternatives on two metrics: compression ratio, and encoding/decoding time. Compression ratio is computed with respect to the original image’s size in JPEG Standard: \[ \frac{T_s}{s}, \] where $s$ is the size of an image generated by a scheme and $s'$ is the file size of the image coded with JPEG. The encoding time is the time to recompress from JPEG and the decoding time is the time to decompress back to JPEG.

Compression Ratios. Table 1 (left) shows the compression ratio for all low complexity schemes to FiveK image sets. We see that ROMP provides the highest compression ratio among lossless codecs. It achieves a 15% compression ratio over JPEG Standard. L-ROMP achieves a 29% compression ratio. We further examine the distribution of compression ratios of images and find that ROMP provides a compression ratio over 15% for 95% of images while L-ROMP provides a compression ratio over 28% for 95% of images. This demonstrates both schemes provide good compression for almost all images on average.

We also evaluate using images of different resolutions and quality parameters of the Tecnick image set. Table 1 (left) shows the most common parameters: resolution of 1200×1200, and quality parameter of 75. For ROMP and L-ROMP, we observe consistent compression results as compared to FiveK image sets (15% and

---

$^6$Note that the actual MSE will be different if two coefficients have same value 1, but different frequency weights in the quantization matrix. However, by ignoring this difference we take into account the different perceptual weighting given to each frequency and obtain better perceptual quality.
be because it affects user-perceived delay. ROMP’s decoder is slightly faster than JPEG Arithmetic, comparable to JPEG Progressive and MozJPEG, and much faster than other schemes: $4.8 \times$ faster than PackJPG, $10 \times$ faster than WebP and $20 \times$ faster than JPEG2000. L-ROMP’s decoder is identical to ROMP, but after thresholding the image becomes smaller, which makes it ~20% faster than ROMP, a more significant advantage comparing to alternatives. Interestingly, decoding is faster than encoding for most of the schemes; but for all the high complexity alternatives (PackJPG, WebP, JPEG2000), the decoding is actually slower.

This experiment shows that ROMP and L-ROMP achieve both high compression ratios (15 – 28% ) and very low complexity (< 50ms encoding/decoding time for a 1200×1200 image). By contrast, the other high-compression scheme, PackJPG, has a compression ratio of 20%, but its decoding time is over 178ms, more than 5 times the complexity of ROMP. Other lossy codecs have even higher complexities. We also conduct experiments on higher quality and resolution images to show how decoding time scales with each of these factors. In general, we observe that schemes with low decoding complexity scale well, including ROMP and L-ROMP; the decoding time for PackJPG, WebP and JPEG2000 scale poorly with image size and image quality. For example, PackJPG takes 531 ms to decode a 2048×1536 image. WebP needs 811 ms and JPEG2000 requires almost 3000 ms, limiting in photo sharing services unless the client can decode these formats.

**5 Conclusion**

Motivated by the need for additional tools for managing storage in large photo sharing services, this paper explores the problem of image recompression and proposes two low complexity recompression schemes, ROMP and L-ROMP, that produce perceptually lossless compression with gains of 15-29% on a large corpus of images. Compression gains of this magnitude can substantially reduce storage requirements at these services; this brings collateral benefits including internal bandwidth reduction, etc.

---

\[\text{Note: L-ROMP’s algorithm is optimized for perceptual quality, not PSNR. L-ROMP can be tuned to optimize for PSNR, and that would lead to relatively higher compression ratio as compared to WebP and JPEG2000.}\]

---

**Table 1. Left:** Compression ratio over JPEG Standard for FiveK image sets (1152×864 and 2048×1536, QP=75) for low complexity schemes, and for Tecnick image set (1200×1200, QP=75). **Right:** Encoding and decoding complexity comparison on Tecnick image set (1200×1200, quality parameter is 75).

<table>
<thead>
<tr>
<th>Lossless</th>
<th>Compression Ratio (100%)</th>
<th>Complexity (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FiveK</td>
<td>Tecnick</td>
<td>Tecnick</td>
</tr>
<tr>
<td>1152</td>
<td>2048</td>
<td>Enc. Dec.</td>
</tr>
<tr>
<td>Lossy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L-ROMP</td>
<td>28.00%</td>
<td>29.10%</td>
</tr>
<tr>
<td>WebP</td>
<td>N/A</td>
<td>29.71%</td>
</tr>
<tr>
<td>JPEG2000</td>
<td>N/A</td>
<td>28.34%</td>
</tr>
<tr>
<td>ROMP</td>
<td>15.44%</td>
<td>16.07%</td>
</tr>
<tr>
<td>PackJPG</td>
<td>N/A</td>
<td>21.53%</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>10.00%</td>
<td>11.50%</td>
</tr>
<tr>
<td>MozJPEG</td>
<td>5.90%</td>
<td>6.41%</td>
</tr>
<tr>
<td>Progressive</td>
<td>4.60%</td>
<td>4.90%</td>
</tr>
<tr>
<td>Optimized</td>
<td>2.49%</td>
<td>2.97%</td>
</tr>
</tbody>
</table>

**Fig. 2.** Rate/distortion performance of L-ROMP, compared to re-quantization from raw image and from JPEG image using Tecnick images of 1200×1200. **(a):** Using PSNR as the quality metric, and showing performance on JPEG images of four QPs (70,80,86,90). **(b):** Using MS-SSIM metric, and focusing on JPEG images of QP=75; “o” marks the perceptually lossless setting of L-ROMP.

29%). For lossy alternatives, to make the comparison fair, we compare the compression ratio of the same achieved PSNR: we tune each lossy alternative’s “quality degradation” parameter to provide the same PSNR of L-ROMP’s perceptually lossless setting. Note that in all cases images are first encoded using JPEG and then re-encoded with one of the algorithms under consideration in order to mimic the settings in a photo sharing system where JPEG images are uploaded. WebP and JPEG2000 achieve similar compression ratios compared to L-ROMP\(^8\), but with unacceptably high complexity as will be discussed next. In terms of lossless performance, we see that ROMP’s compression ratio is over 15%; more importantly, ROMP’s compression ratio increases as the quality parameter increases beyond 75, achieving up to 7% higher compression gain than JPEG Arithmetic, and 10% higher compression than Progressive and MozJPEG: given the trend towards higher quality images, this is a desirable property. PackJPG has higher compression gain, but its complexity is also significantly higher.

**Complexity.** Table 1 (right) compares the encoding and decoding complexity of ROMP, and L-ROMP against other alternatives. For encoding time, ROMP is comparable to JPEG Arithmetic, and much faster than other competitors. Compared to ROMP, L-ROMP’s additional step of thresholding does not induce any extra complexity. Decoding time is the more relevant metric for photo sharing services.
6 References


[21] Mozilla, *mozjpeg*, Available at [https://blog.mozilla.org/research/2014/03/05/introducing-the-mozjpeg-project/](https://blog.mozilla.org/research/2014/03/05/introducing-the-mozjpeg-project/).


