# Photo Forensics From Rounding Artifacts

Shruti Agarwal
Hany Farid
shruti_agarwal@berkeley.edu
hfarid@berkeley.edu
University of California, Berkeley
Berkeley, CA

## ABSTRACT

Many aspects of JPEG compression have been successfully used in the domain of photo forensics. Adding to this literature, we describe a JPEG artifact that can arise depending upon seemingly innocuous implementation details in a JPEG encoder. We describe the nature of these artifacts and show how a generic JPEG encoder can be configured to explain a wide range of these artifacts found in real-world cameras. We also describe an algorithm to simultaneously estimate the nature of these artifacts and localize inconsistencies that can arise from a wide range of image manipulations.

## KEYWORDS

Image Forensics, Image Analysis, JPEG Compression

**ACM Reference Format:**
Shruti Agarwal and Hany Farid. 2018. Photo Forensics From Rounding Artifacts. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

Within the field of photo forensics [17], format-based techniques exploit specific artifacts introduced by the choice of image format and compression scheme. A variety of such forensic techniques have been developed based on the JPEG compression format: The quantization values used to quantize the discrete cosine transform (DCT) coefficients are used to identify the recording camera manufacturer and model [16, 22, 23]; anomalies in the distribution of DCT coefficients or the first digit of DCT coefficients are used to reveal multiple JPEG compressions [2, 6, 7, 19, 29, 32, 36]; and artifacts introduced by the JPEG blocking are used to localize tampering [20, 28, 31, 42]. More recently, machine learning has been employed to automatically learn and detect anamolous JPEG artifacts [5, 35].

It has recently been shown that artifacts introduced by the choice of rounding operator used to quantize the DCT coefficients can be used to localize tampering and identify specific encoders [1, 9].[1] This artifact – termed a JPEG dimple – manifests itself as a single darker or brighter pixel in each $8 \times 8$ intensity block. In our earlier work [1], we only considered the impact of floating-point to integer rounding that occurs in the final quantization of DCT coefficients. While this was sufficient to capture the basic artifact, it failed to explain large variations in these artifacts in real-world cameras. Here we extend this earlier work as follows: (1) We show the presence of a wide variation in rounding-based artifacts across dozens of cameras; (2) We show how a detailed model of a JPEG encoder can capture most of these variations; and (3) We propose an algorithm to automatically localize inconsistencies in these artifacts that may arise from a broad range of manipulations.

The analysis of rounding errors in the domain of digital signal processing is well studied [41]. Such rounding errors, arising due to the use of fixed-point arithmetic, have been previously modeled as uniformly distributed white noise [4, 21, 34]. Using this model, the authors in [38] analyzed the rounding errors introduced during various fast fixed-point DCT implementations proposed in [10, 26]. In [8] and [33], the authors analyzed the propagation of rounding errors in JPEG compression. Similar to this previous work, we also analyze the rounding errors introduced during fixed-point JPEG compression. Our analysis, however, is targeted towards understanding and re-producing the rounding errors observed in real-world cameras. Unlike previous work, that derive the rounding error for a given JPEG implementation, we aim to estimate the JPEG compression parameters that give rise to a specific artifact, and exploit these artifacts to detect and localize image manipulation.

We will start by describing a generic JPEG encoder along with some design decisions that fix certain implementation details while allowing others to be configurable. We then show how different choices of the configurable parameters give rise to different artifacts found in many real-world cameras. Lastly, we describe an algorithm to localize inconsistencies in these artifacts that arise from a variety of image manipulations.

## 2 JPEG COMPRESSION

The JPEG image standard is the most popular lossy compression scheme for still photographic images [40]. The JPEG encoding of a 3-channel RGB color image consists of seven basic steps: (1) convert from RGB to luminance/chrominance (YCbCr); (2) optionally subsample each channel by a factor of two or more (the chrominance channels are typically subsampled but the luminance channel is not); (3) partition each channel into non-overlapping $8 \times 8$ pixel


*Woodstock '18, June 03–05, 2018, Woodstock, NY*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

[1] Jessica Fridrich and colleagues have previously observed this JPEG artifact and routinely eliminated it as part of their PRNU analysis [18].

blocks; (4) convert the luminance values from unsigned to signed integers ( e.g., from [0; 255] to [−128, 127] ); (5) convert each block to the frequency domain using a 2-D discrete cosine transform (DCT); (6) quantize each DCT coefficient where the amount of quantization depends on the spatial frequency and channel (the lower spatial frequencies are typically quantized less than higher frequencies, as is the luminance channel as compared to the chrominance channels); (7) entropy encode the quantized DCT coefficients.

Because performing the 2-D DCT in step (5) is the most computationally demanding step of JPEG compression, a number of efficient DCT implementations are available [37]. The popular JPEG compression library, JPEGLib [2], for example, provides three different DCT implementations each yielding a different speed and accuracy trade-off. Inspired by this implementation, the pseudo-code block-jpeg in Appendix C is a generic JPEG algorithm in which we have fixed some parameters (fixed-point arithmetic, 32-bit precision, two's-complement representation, and a more efficient pair of 1-D DCTs), while leaving other parameters configurable. In total, there are 87 of these configurable parameters, which as we will see, can have a impact on the accuracy of the final JPEG compression. These parameters are:

(1) $d(\cdot)$ is one of the three 1-D DCT implementation, $\mathrm{d}_1(\cdot)$, $\mathrm{d}_2(\cdot)$, or $\mathrm{d}_3(\cdot)$ (see Appendix A)
   - $\mathrm{d}_1(\cdot)$ is based on the Ligtenberg-Vetterli (LV) algorithm [39] which yields 1-D DCT coefficients each scaled by a factor of 2. Our implementation is performed with a single matrix multiplication, but this algorithm can be more efficiently implemented with 29 additions and 13 multiplications.
   - $\mathrm{d}_2(\cdot)$ is based on the Loeffler-Ligtenberg-Moschytz (LLM) algorithm [30] which yields 1-D DCT coefficients each scaled by a factor of $2\sqrt{2}$. Our implementation is again performed with a single matrix multiplication, but this algorithm can be more efficiently implemented with 29 additions and 11 multiplications.
   - $\mathrm{d}_3(\cdot)$ is based on the Arai-Agui-Nakajima (AAN) algorithm [3] which yields 1-D DCT coefficients each scaled by a different value given by $\vec{a}$, Equation (1). This algorithm is implemented with 29 additions and 5 multiplications.

$$\vec{a} = \begin{pmatrix} 2\sqrt{2} \\ 4\cos(\pi/16) \\ 4\cos(2\pi/16) \\ 4\cos(3\pi/16) \\ 2\sqrt{2} \\ 4\cos(5\pi/16) \\ 4\cos(6\pi/16) \\ 4\cos(7\pi/16) \end{pmatrix} \quad (1)$$

   Each of the above 1-D DCT implementations consists of three basic steps:
   - Computation of constants each of which is computed as a floating-point value and converted, using the flt2fix operator (see Appendix B), to a fixed-point representation with an assumed 13-bit precision (see $c_j$ in $\mathrm{d}_1(\cdot)$, $\mathrm{d}_2(\cdot)$, and $\mathrm{d}_3(\cdot)$, $j \in [0, 7]$.).
   - Computation of an eight-point 1-D DCT. Each of the resulting DCT coefficients differ from the desired DCT coefficients by a multiplicative factor, defined as the coefficient scale. The scale for $\mathrm{d}_1(\cdot)$, $\mathrm{d}_2(\cdot)$, and $\mathrm{d}_3(\cdot)$ is $2^{13} \cdot 2$, $2^{13} \cdot 2\sqrt{2}$, and $2^{13} \cdot \vec{a}$, respectively.
   - Element-wise division of the DCT coefficients followed by rounding in order yield properly scaled coefficients. In $\mathrm{d}_1(\cdot)$ and $\mathrm{d}_2(\cdot)$, this *de-scaling* operation is performed in lines 3-4, and in lines 5-6 for $\mathrm{d}_3(\cdot)$ (see Appendix A).

(2) $\vec{s}_h$ and $\vec{s}_v$ are each 8-D vectors consisting of de-scaling constants used to control the horizontal ($h$) and vertical ($v$) scale of the 1-D DCT coefficients in $d(\cdot)$.
(3) $s$ is a scalar that can be used to de-scale the 2-D DCT coefficients after two 1-D DCT computations.
(4) $Q$ is an $8 \times 8$ matrix of DCT-coefficient quantization values. These values may or may not be scaled depending on the de-scaling constants of $\vec{s}_h$, $\vec{s}_v$, and $s$.
(5) $f_e(\cdot)$ and $f_o(\cdot)$ are rounding operators used to de-scale even and odd DCT coefficients after 1-D DCT in $d(\cdot)$. We distinguish between even and odd coefficients because they typically involve different computations.
(6) $f_s(\cdot)$ is a rounding operator used to de-scale the 2-D DCT coefficients at line 8 of block-jpeg.
(7) $f_q(\cdot)$ is a rounding operator used to quantize the final DCT coefficients by non-power-of-two values (line 14 of block-jpeg).
(8) $f_2(\cdot)$ is a rounding operator used to quantize the final DCT coefficients by power-of-two values (line 10 of block-jpeg). We distinguish between this and $f_q(\cdot)$ because division by a power of two followed by rounding can be efficiently implemented with a simple bit-shift. Note that unlike $f_q(\cdot)$ and $f_2(\cdot)$ which are used to quantize the DCT coefficients, the operators $f_e(\cdot)$, $f_o(\cdot)$, and $f_s(\cdot)$ are used in the computation of the DCT coefficients. See the next section for a description of the four rounding operators that we consider and see Appendix B for pseudo-code for these operators.

## 2.1 Rounding Artifacts

The parameters $f_e(\cdot)$, $f_o(\cdot)$, $f_s(\cdot)$, $f_q(\cdot)$, and $f_2(\cdot)$ defined above require a choice of rounding operators. The following four standard rounding operators will be considered:

- The round operator rounds floating-point values to the nearest integer: round(1.5) = 2 and round(−1.5) = −2.
- The halfup operator is similar to round(·) except in the case of a tie, where values are rounded towards the largest nearest integer: halfup(1.5) = 2 and halfup(−1.5) = −1.
- The trunc operator rounds towards the nearest integer with the smaller magnitude (i.e., towards zero): trunc(1.5) = 1 and trunc(−1.5) = −1.
- The floor operator rounds towards the smallest nearest integer (i.e., towards negative infinity): floor(1.5) = 1 and floor(−1.5) = −2.

Note that because the typical JPEG encoder is implemented using fixed-point arithmetic, these rounding operations are only necessary after a division (in our case, either quantization or de-scaling). As such, we bundle rounding and division into a single function (see Appendix B).

[2] http://www.ijg.org/

The choice of which of the four rounding operators to use (round, halfup, trunc, floor) depends largely on a speed-accuracy trade-off. For example, the floor operator with fewer operations than the round operator is more efficient. On the other hand, as we will show below, the floor operator yields values that are consistently smaller than their original values, whereas the round operator does not.

To see the nature of these biases, we will consider the most common case of dividing by a power of 2 (for the purposes of exposition, we consider power-of-two divisors, but the basic bias which we will show manifests itself for non-power-of-two divisors). Let the random variables $\Delta_f$, $\Delta_r$, $\Delta_h$, and $\Delta_t$ represent the difference between the floating-point values $x/2^k$ and the integer values obtained after floor$(x, 2^k)$, round$(x, 2^k)$, halfup$(x, 2^k)$, and trunc$(x, 2^k)$ operations respectively, where $x$ is an integer (assumed to be drawn from a zero-mean distribution), $k$ is a positive integer, and the rounding operation occurs after the division $x/2^k$.

Following [21, 38], we assume that: (1) the discrete random variable $\Delta_*$ is uniformly distributed in a finite interval, (2) $\Delta_*$ is uncorrelated with the input $x$, and (3) $\Delta_*$ at any step in block-jpeg is independent of its value at any other step. Under these assumptions, we will derive the bias (if any) that occurs following each of the four rounding operators.

Per our assumption, $\Delta_f = \text{floor}(x, 2^k) - x/2^k$ is uniformly distributed over $i/2^k$ where $i$ is an integer in the interval $[-(2^k - 1), 0]$. The expected value of the random variable $\Delta_f$ is:

$$\begin{aligned}
\mathbb{E}\left[\Delta_f\right] &= \frac{1}{2^k} \sum_{i=-(2^k-1)}^{0} \frac{i}{2^k} \\
&= \frac{1}{2^{2k}} \sum_{i=-(2^k-1)}^{0} i \quad = \quad -\frac{1}{2^{2k}} \sum_{i=0}^{2^k-1} i \\
&= -\frac{1}{2^{2k}} \left( \frac{2^k(2^k-1)}{2} \right) \quad = \quad -\frac{(2^k-1)}{2^{k+1}}. \qquad (2)
\end{aligned}$$

Note that for $k = 1$, this expected value is -0.25, and for large values of $k$, this expected value approaches -0.5. That is, the floor operator introduces a consistent negative bias.

The random variable $\Delta_h = \text{halfup}(x, 2^k) - x/2^k$ also follows a uniform distribution over $i/2^k$ where $i$ is an integer in the interval $\left[-(2^{k-1} - 1), 2^{k-1}\right]$. The expected value of $\Delta_h$ is:

$$\begin{aligned}
\mathbb{E}\left[\Delta_h\right] &= \frac{1}{2^k} \sum_{i=-(2^{k-1}-1)}^{2^{k-1}} \frac{i}{2^k} \\
&= \frac{1}{2^{2k}} \sum_{i=-(2^{k-1}-1)}^{2^{k-1}} i \\
&= \frac{1}{2^{2k}} (2^{k-1}) \quad = \quad \frac{1}{2^{k+1}} \qquad (3)
\end{aligned}$$

When $k = 1$, this expected value is 0.25. In contrast to the floor operator, where a bias of $-0.5$ is introduced as $k$ increases, increasing values of $k$ in the halfup operator lead to an exponentially smaller bias.

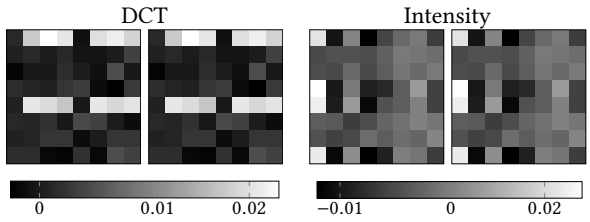


**Figure 1: Shown in the left half of each panel are the average $8 \times 8$ DCT blocks extracted from images compressed using JPEGLib (left) and the corresponding optimal estimate compressed with** block-jpeg **(right). The $L_1$ estimation error is $10^{-17}$. Shown in the right half of this figure is the same artifact in the intensity domain.**

While the floor and halfup operators introduce a consistent bias, a similar analysis of the expected values for the round and trunc operators reveal no such bias. The systematic bias introduced by the floor and halfup rounding operators, used throughout the JPEG compression pipeline, leads to specific artifacts that vary across camera manufacturers.

## 2.2 JPEG Artifacts

The above rounding operators are used at each of four steps in block-jpeg: Once during each of two 1-D DCTs ($f_e(\cdot)$, $f_o(\cdot)$), once during 2-D de-scaling ($f_s(\cdot)$), and once during quantization $\left(f_q(\cdot), f_2(\cdot)\right)$. Because the AC-coefficients of natural images are zero-mean [27], the negative and positive biases described above will lead to non-zero-mean distributions which in turn will lead to perceptual artifacts in the average $8 \times 8$ DCT/intensity block. As will see later, these artifacts depend on both the specific rounding operator used in each of these steps as well as the de-scaling and quantization factors. First, however, we will describe how to estimate these JPEG compression parameters.

# 3 JPEG PARAMETER ESTIMATION

Let $X$ be the expected value of the rounding artifact, estimated by averaging all $8 \times 8$ DCT blocks from images compressed with a fixed JPEG encoder and quality. We seek the block-jpeg parameters $\vec{\theta} = [d(\cdot), \vec{s}_h, \vec{s}_v, s, Q, f_q(\cdot), f_2(\cdot), f_e(\cdot), f_o(\cdot), f_s(\cdot)]$ that yield an artifact $Y_{\vec{\theta}}$ that minimizes the following $L_1$ objective function

$$\vec{\theta}_m \quad = \quad \arg\min_{\vec{\theta}} \left[ \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} \left| X_{i,j} - Y_{\vec{\theta},i,j} \right| \right], \qquad (4)$$

Each $Y_{\vec{\theta}}$ is computed by compressing a fixed set of 10 grayscale images with block-jpeg and averaging all non-overlapping $8 \times 8$ DCT blocks. In order to analyze only rounding artifacts, we subtract the *floating-point-average*[3] DCT block from each $Y_{\vec{\theta}}$. Each grayscale image is of size $512 \times 512$ yielding a total of $40,960$ $8 \times 8$ blocks. The parameter $\vec{\theta}_m$ that minimizes the objective in Equation (4) is estimated using a brute force search over a reduced parameter search space as described next.

[3]The floating-point-average DCT block, obtained using floating-point DCT, with respective quantization table does not introduce any integer rounding bias.

**Table 1: The ground truth (row 1) and estimated (rows 2-7) parameter configurations for JPEGLib. The parameters in rows 2-7 each have the same $L_1$ estimation error (within floating-point precision). The $*$ indicates that all four rounding operators give the same $L_1$ error.**

| $d$ | $(s_h^e, s_h^o)$ | $(s_v^e, s_v^o)$ | $s$ | $f_2(\cdot)$ | $f_q(\cdot)$ | $f_s(\cdot)$ | $f_e(\cdot)$ | $f_o(\cdot)$ |
|---|---|---|---|---|---|---|---|---|
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^0$ | round | round | * | halfup | halfup |
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^0$ | round | round | round | halfup | halfup |
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^0$ | round | round | halfup | halfup | halfup |
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^0$ | round | round | trunc | halfup | halfup |
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^0$ | round | round | floor | halfup | halfup |
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^2$ | round | round | trunc | halfup | halfup |
| $d_2$ | $(2^{11}, 2^{11})$ | $(2^{15}, 2^{15})$ | $2^1$ | round | round | trunc | halfup | halfup |

The full search space for the 87 parameters in block-jpeg is enormous. For example, even if we (arbitrarily) restrict each element in the 8-D vectors $\vec{s}_h$ and $\vec{s}_v$ to 10 possible values, the search space over just these parameters is $10^{16}$. As such, we restrict the parameter search space as follows: (1) the values in $\vec{s}_h$, $\vec{s}_v$ and $s$ are restricted to be a power-of-two; (2) the values for $\vec{s}_h$ and $\vec{s}_v$ are chosen such that the DCT coefficients after each 1-D DCT can have a maximum scale of $2^3$; (3) the values in $\vec{s}_h$ at all even/odd positions are restricted to have the same value. The same constraint is separately applied to $\vec{s}_v$, yielding a total of two parameters for $\vec{s}_h$ and two parameters for $\vec{s}_v$; (4) Finally the quantization table is assumed to be known and fixed.

The above constraints reduce the search space to a total of $3,145,728$ parameter combinations: three possible values for $d(\cdot)$; $16 \times 16 \times 4$ possible values for even and odd de-scaling constants in $\vec{s}_h$, $\vec{s}_v$, $s$; $4^5$ possible values for $f_s(\cdot)$, $f_e(\cdot)$, $f_o(\cdot)$, $f_2(\cdot)$, and $f_q(\cdot)$ as each function can take on one of four possible values, floor, halfup, round, or trunc.

## 3.1 Software Artifacts

In this section, we estimate the parameters for the common JPEGLib library.[4] This library offers three variations of 1-D DCT implementations, float DCT, fast-integer DCT, and slow-integer DCT. We analyze the slow-integer DCT because, unlike the other implementations, this version is captured by our model ($d_2(\cdot)$ in Appendix A).

The slow-integer DCT implementation uses a 13-bit precision to represent the floating point constants. All the DCT coefficients are divided by $2^{11}$ after the first 1-D row-DCT, followed by a division with $2^{15}$ after the second 1-D column-DCT. Each of these divisions are implemented with the bit shift version of the halfup rounding operator (see Appendix B). The remaining scale of $2^3$ (see Section 2) is multiplied with the quantization table and removed during quantization with the round operator used for both power and non-power of two's. JPEGLib does not perform a 2-D de-scaling operation (step 3 in Section 2). This yields the following ground truth values for the parameter $\vec{\theta}$: $d = d_2$, $(s_h^e, s_h^o) = (2^{11}, 2^{11})$, $(s_v^e, s_v^o) = (2^{15}, 2^{15})$, $s = 1$, $f_2(\cdot) =$ round, $f_q(\cdot) =$ round, $f_s(\cdot) = *$, $f_e(\cdot) =$ halfup, and $f_o(\cdot) =$ halfup. The $*$ indicates that the value of the rounding operator doesn't matter as the divisor is 1.

[4]The parameters estimated from MatLab-compressed images and TurboJPEG-compressed images were found to be identical to JPEGLib-compressed images.

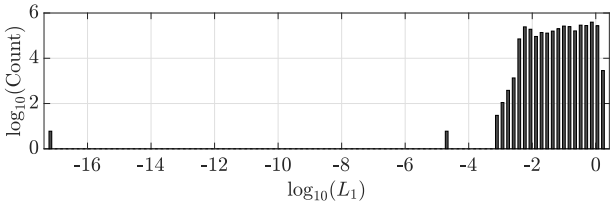


**Figure 2: A histogram of $L_1$ parameter estimation errors for JPEGLib, displayed on a** log-log **scale.**

We construct the average DCT block $X$ in Equation (4) with the same 10 grayscale images that were used to compute $Y_{\vec{\theta}}$. The quantization table is fixed to consist of both power and non-power of twos.

Shown in the left half of Figure 1 is the average $8 \times 8$ DCT blocks compressed using JPEGLib (left) and the corresponding estimated artifact (right). Shown in the right half of this figure is the same artifact in the intensity domain. The first row of Table 1 enumerates the ground-truth parameters for JPEGLib. Rows 2-7 of this table enumerate the estimated parameters that yielded a minimum $L_1$ error of $10^{-17}$. Rows 2-5 correspond to the correct parameters with only the rounding operator for $f_s(\cdot)$ varying. This is expected because the scale factor $s = 1$ and therefore there is no artifact introduced by this rounding operator. The parameters in rows 6-7 correspond to a configuration with a slightly different scale factor $s$ paired with the trunc operator. These results show that although we can estimate the correct parameters, these parameters are not necessarily unique.

Shown in Figure 2 is histogram of $L_1$ parameter estimation errors (on a log-log scale). There are six parameter configurations with a minimal error of $10^{-17}$. The next smallest error is on the order of $10^{-5}$ and the largest error is on the order of $10^0$. Only $0.0002\%$ of the total possible parameter configurations yield the minimum $L_1$ error, indicating that although not unique, they are distinct.

## 3.2 Camera Artifacts

In this section we show that a diverse set of 24 cameras introduces artifacts that can be approximately modeled with the parameters specified in our generic JPEG encoder block-jpeg. Each group in Figure 3 corresponds to one of the 24 different cameras. Shown from left to right are; the average $8 \times 8$ DCT block $X$, the corresponding optimal estimate $Y_{\vec{\theta}}$ of these artifacts, the average $8 \times 8$ block in the intensity domain and the estimated average block in the intensity domain. For each camera, we downloaded between 25 and 100 images from Flickr with intact metadata, ensuring that the images were of the same orientation and quality. This yielded a total of $1,694$ images. The average DCT blocks, computed over all non-overlapping $8 \times 8$ blocks across all images for each camera, is shown in the left-half of each panel in Figure 3 (to aid visualization, the large-magnitude DC-term and the first horizontal and vertical AC-terms are zeroed-out). Note that the artifacts introduced by different cameras vary in terms of their overall structure and magnitude (the panels are displayed on different intensity scales, as specified below each panel). The DCT block $Y_{\vec{\theta}}$ is constructed from the estimated parameters $\vec{\theta}$.

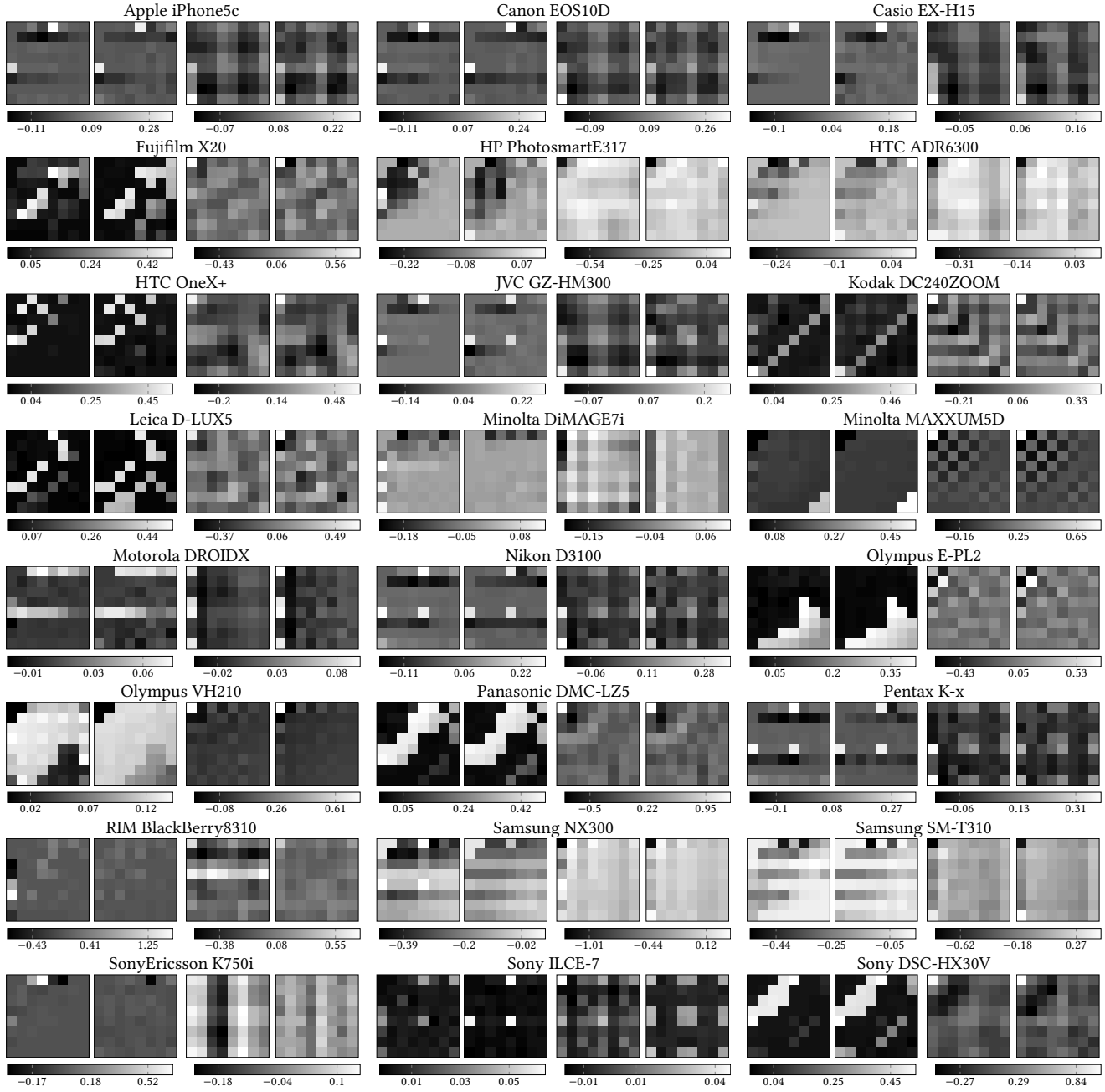


**Figure 3: For each of the 24 cameras we show, from left to right, the actual $8 \times 8$ average DCT block from JPEG images of the camera, the average DCT block after compression with the estimated parameters, the actual average block in intensity domain and the estimated average block in intensity domain.**

As in the previous section, the minimization does not yield a unique set of parameters. Despite the lack of parameter uniqueness, we see that our model captures a broad range of compression artifacts. At the same time, there are some artifacts that are not captured by our model. The impact of this, along with the lack of uniqueness, on the forensic application will be addressed in Section 4.

## 3.3 Recompression Artifacts

We have shown that different encoders introduce different JPEG artifacts due to the differences in JPEG compression parameters. We next begin investigating how these artifacts may be used to forensically detect and localize manipulations.

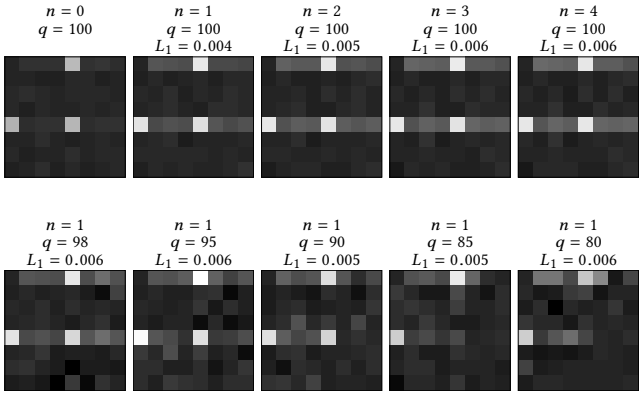


**Figure 4: The effect of multiple compressions on the JPEG artifact. Shown in the top row is the average $8 \times 8$ DCT artifact for singly compressed images ($n = 0$) at the highest quality ($q = 100$), compressed using the estimated parameters of a SONY ILCE-7 camera. Also shown in this row is the effect of multiple compression ($n = 1, 2, 3, 4$) also at the highest quality, recompressed using JPEGLib. The $L_1$ error is between the recompressed and original (far-left) DCT blocks. Shown in the bottom row is the effect of double compression ($n = 1$) at increasingly lower quality second compressions ($q = 98, 95, 90, 85, 80$).**

A manipulation of a JPEG image will generally require, at a minimum, at least two compressions. The first compression will occur on the device and the second will typically occur by the photo-editing software. Because these compressions will likely employ different encoders and different quantization values [22], we will consider the impact of these multiple compressions on the original, device-induced, artifacts. We will also consider the impact of cropping that will lead to multiple compressions in which the $8 \times 8$ JPEG lattice will be misaligned.

Throughout this section we will analyze 10 original images compressed with the estimated parameters for a Sony ILCE-7. We will then use JPEGLib to induce a second compression where qualities are specified in a range of 100 (highest) to 1 (lowest).

Shown in the top row of Figure 4 is the average 8×8 DCT block for images that have been compressed once ($n = 0$) and recompressed one ($n = 1$) to four ($n = 4$) times, each at the highest quality of 100. Shown below each panel is the $L_1$ error between the singly compressed image and the recompressed versions. The artifact in the recompressed images is effectively a super-position of the artifacts from the original Sony ILCE-7 and the JPEGLib artifacts, Figure 1, with the artifact corresponding to the second compression becoming slightly more pronounced with more recompressions.

Shown in bottom row of Figure 4 is the effect of recompression as a function of compression quality. Each panel shows the average $8 \times 8$ DCT block for images that have been recompressed once with a quality ranging from 98 to 80. Shown below each panel is the $L_1$ error between the singly compressed image and the recompressed versions. As the recompression quality decreases, the original artifact is diminished.

Shown in Figure 5 is the effect of cropping on the JPEG artifact. Shown in the left of this panel is the original JPEGLib artifact in the DCT and intensity domain. Shown on the right is the same artifact after cropping off the DCT lattice and saving in the lossless PNG format. Although the artifact is significantly different in the DCT domain[5], we see that, as expected, the cropping simply corresponds to a shift in the intensity domain.

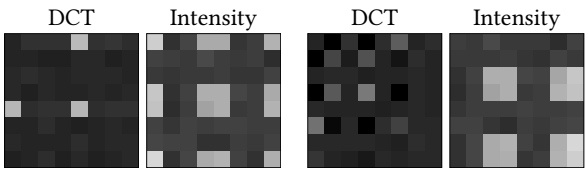


**Figure 5: The effect of cropping on the JPEG artifact. Shown on the left are the original average $8 \times 8$ DCT and intensity blocks. Shown on the right, is the impact of cropping the image off the DCT lattice and re-saving in the lossless PNG format.**

## 4 FORENSICS

As shown in the previous section, a manipulated and resaved image will either contain a combination of compression artifacts (for a high-quality second compression), a diminished artifact (for a low-quality second compression), or a completely new artifact (for cropping prior to a second compression). In either case, we expect a recompressed image to contain artifacts – it just may be difficult to precisely model and estimate these artifacts. In particular, we cannot know the original compression parameters and there is no practical way of considering all possible unknown initial parameters and possible cropping offsets. As a result, we will employ a data-driven approach to automatically estimate and segment an image based on the JPEG rounding artifacts. While this may seem like the detailed model was unnecessary, we believe that a thorough understanding of these artifacts is inherently valuable to our complete understanding of this forensic technique in terms of the source of the artifacts, when these artifacts will be present, and what any counter-forensic techniques might be able to exploit.

In a manipulated image, a portion of the image that was, for example, resized, rotated, median filtered, or digitally inserted from another image, is likely to have JPEG rounding artifacts that are inconsistent with the rest of the image. We develop an algorithm to simultaneously estimate these artifacts and segment an image into regions (if any) that have inconsistent artifacts. The presence of any such regions can be used to identify and localize image manipulation.

This estimation and segmentation problem is formulated within the expectation/maximization (EM) framework [15]. Each 8×8 pixel block is assumed to belong to one of the two classes $C_1$ (original) or $C_2$ (manipulated) each with unknown (and possibly non-existent) rounding artifacts. On each EM-iteration, the rounding artifacts within each class are estimated (as a $8 \times 8$ template in the intensity domain) and the probability that each $8 \times 8$ block belongs to class $C_1$ is computed.

[5]The Fourier transform uses a sine and cosine basis to encode phase by adjusting the relative contribution of these basis functions. In contrast, the DCT, with only a cosine basis, must encode phase shifts by adjusting the relative contributions of the individual cosine harmonics.

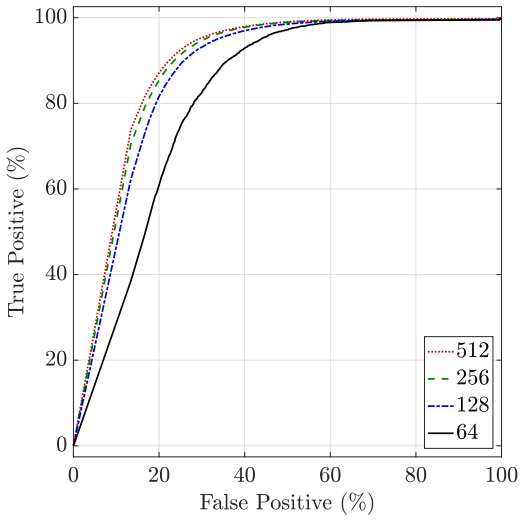


**Figure 6: Detection accuracy as a function of manipulation size, from a maximum size of $512 \times 512$ pixels to a minimum size of $64 \times 64$. The AUC for these manipulation sizes is, from largest to smallest, 0.90, 0.89, 0.87, and 0.81.**

Because the rounding artifacts will be highly correlated across color channels, only the luminance channel is analyzed. In order to reduce the interference of image content, we use the residual of a $3 \times 3$ median filter[6]. The luminance channel is tiled (by 8 pixels) into overlapping square windows $w_i(x, y)$ of size 256, 128, or 64 pixels. For each window, the average block $\vec{b}_i$ of all non-overlapping $8 \times 8$ blocks in this window is computed. We simultaneously estimate the probability that each of these blocks $\vec{b}_i$ belongs to $C_1$ or $C_2$ and the form of the template $\vec{c}_1$ (i.e., the 64 intensity values that make up the rounding artifacts) – the class $C_2$ is not parameterized (i.e., it is an outlier class). This non-parametric second class allows us to contend with the situation when an image has more than two artifacts: one artifact will be characterized by the first class $C_1$ and all others will fall into class $C_2$.

The iterative EM algorithm progresses as follows. The values $\vec{c}_1$ are randomly initialized by drawing 64 values from a uniform distribution in $[0, 1]$. In the E-step, the conditional probability that each block $\vec{b}_i$ belongs to $C_1$ is computed. This conditional probability is estimated by first computing the correlation between $\vec{b}_i$ and the template $\vec{c}_1$ as: $r_i = \vec{b}_i \otimes \vec{c}_1$, where $\otimes$ denotes 2-D correlation. The conditional probability, $P(\vec{b}_i \in C_1 | r_i)$, of each block belonging to $C_1$ given the correlation $r_i$ is then given by Bayes' rule:

$$P(\vec{b}_i \in C_1 | r_i) = \frac{P(r_i | \vec{b}_i \in C_1) P(\vec{b}_i \in C_1)}{P(r_i | \vec{b}_i \in C_1) P(\vec{b}_i \in C_1) + P(r_i | \vec{b}_i \in C_2) P(\vec{b}_i \in C_2)} \quad (5)$$

The conditional probabilities in the numerator and denominator are computed numerically (and offline) as described below. The prior $P(\vec{b}_i \in C_1)$ is assumed to be 0.5. With only two models, the conditional probability for the second class, $P(\vec{b}_i \in C_2 | r_i)$, is simply $1 - P(\vec{b}_i \in C_1 | r_i)$.

In the M-step, the template $\vec{c}_1$ is re-estimated using a weighted average of all blocks $\vec{b}_i$, where the weighting is the probability that

[6]For most cameras, our proposed JPEG artifact manifests as a single darker (lighter) pixel in $8 \times 8$ intensity block, Figure 3. We, therefore, found that a simple median filter, which is good at filtering salt-pepper noise, performed better than other more complex noise filters like Wiener or BM3D [13].

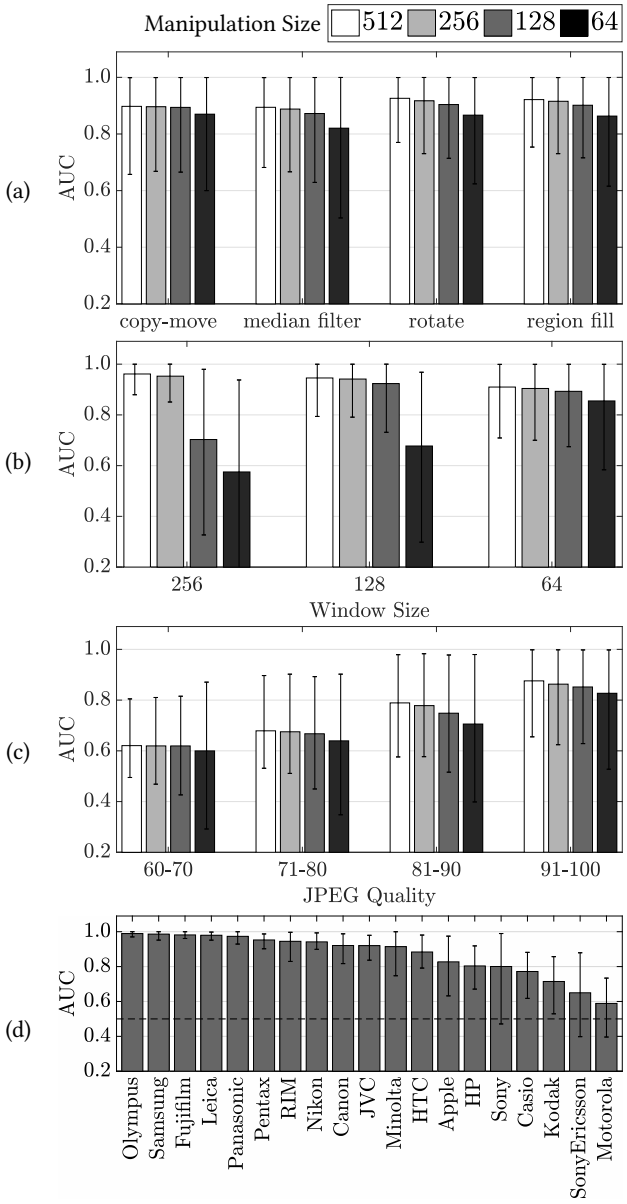


**Figure 7: Shown in each panel is the accuracy of detecting manipulation specified as the average area under the curve (AUC): (a) Accuracy for different manipulation types and sizes; (b) Accuracy for different EM window size $w(x, y)$ and manipulation size; (c) Accuracy for different amounts of JPEG compression and manipulation size; and (d) accuracy for different camera manufacturers. The error bars in each panel correspond to 10% quantile range.**

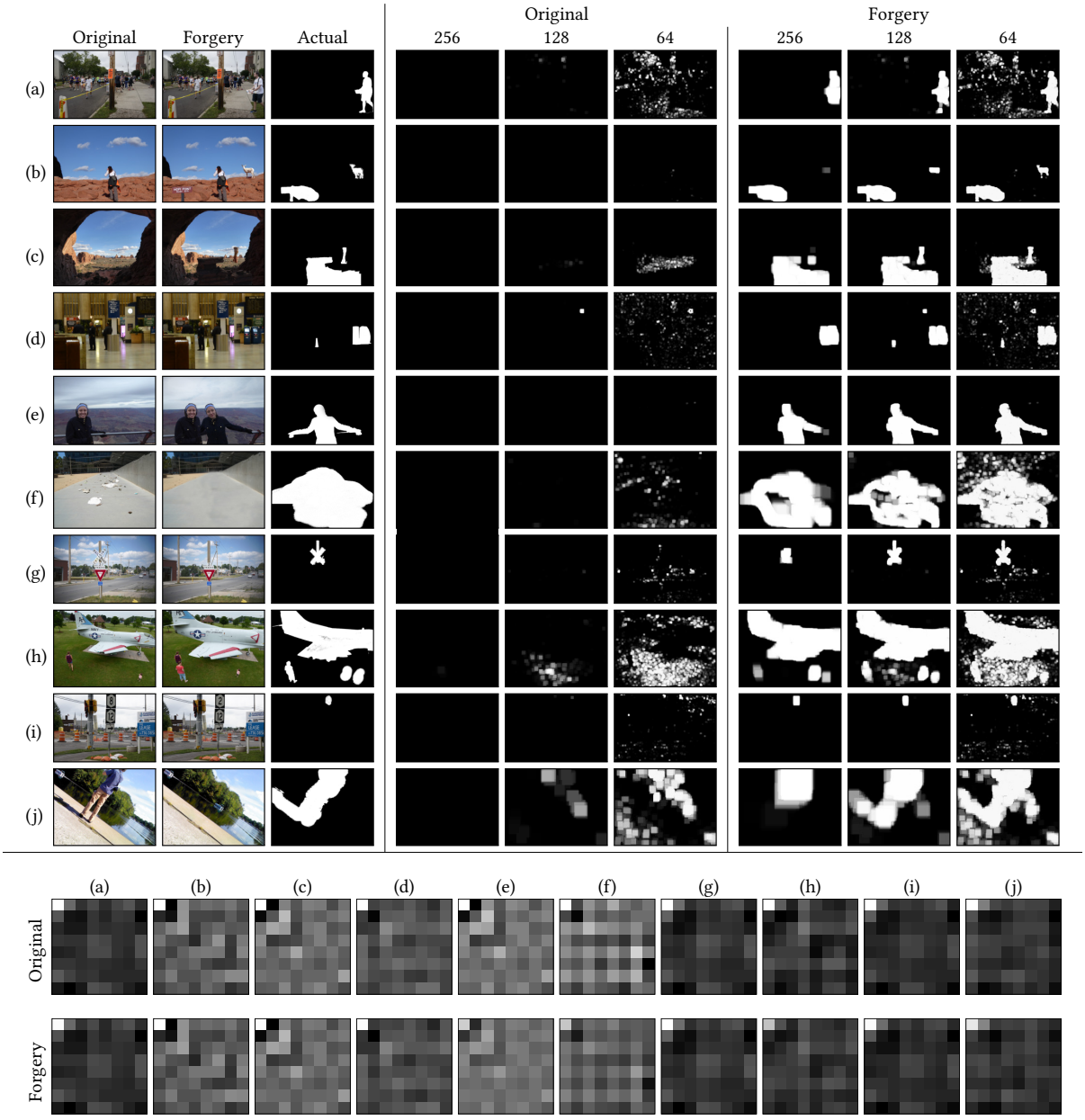


**Figure 8: Shown are examples of image splicing from the NC2017 evaluation dataset. The columns from left to right show the original image, forged image, localization mask, our estimated localization mask (for the original and forgery) using a window size of 256, 128, and 64. Shown below the horizontal divider are the optimal template $c_1$ (Eq. 6) estimated by the EM algorithm for the original and forged images.**

each block belongs to model $C_1$:

$$\vec{c}_1 = \frac{\sum_i P(\vec{b}_i \in C_1 | r_i)\vec{b}_i}{\sum_i P(\vec{b}_i \in C_1 | r_i)} \quad (6)$$

The E- and M-steps are iteratively performed until the difference between successive estimates of $\vec{c}_1$ is below a specified threshold.

The conditional probabilities in the right-hand side of Equation (5) are estimated using a large-scale simulation. The luminance channel of 1000 raw images [14] are extracted and JPEG compressed with 18 random compression settings estimated in Section 3.2. A single window of size $N \times N$ aligned to the $8 \times 8$ JPEG-block lattice was extracted from each of the resulting 18,000 images. All non-overlapping $8 \times 8$ pixel blocks were then averaged to yield an estimate of the rounding artifacts. The correlation between these 18,000 blocks and their matching templates yields an estimate of the conditional probability $P(r_i|\vec{b}_i \in C_1)$. The conditional probability

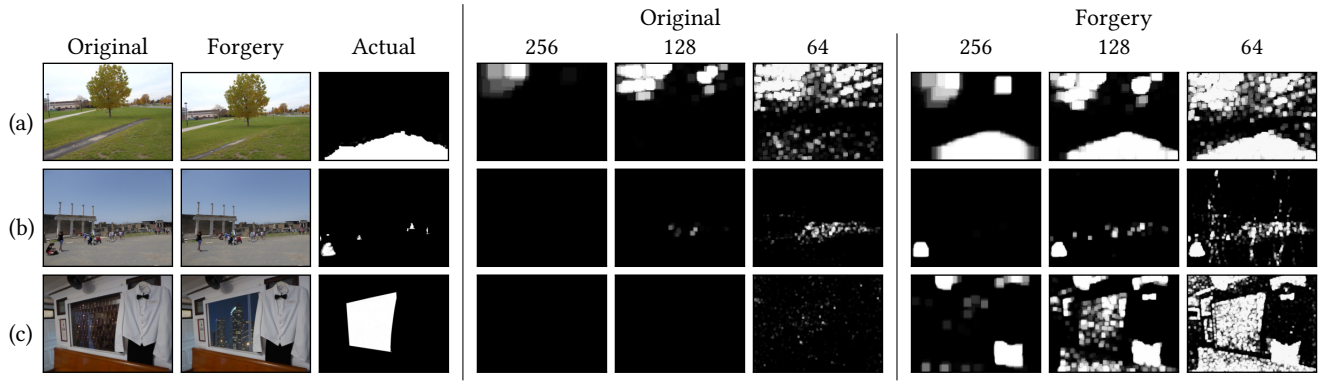


**Figure 9: Three representative failure cases in which: (a) the white exposed regions in the sky are incorrectly flagged as manipulated; (b) multiple small manipulated regions are missed; (c) the large manipulation is missed because in this forgery, a noise residual was added back to the image as part of a counter-forensic attack. The columns from left to right show the original images, forged images, localization mask, our localization mask (for the original and forgery) using a window size of 256, 128, and 64.**

for mis-matched templates $P(r_i|\vec{b}_i \in C_2)$ is computed by generating a new set of 18,000 compressed blocks using floating-point DCT (which does not introduce any rounding bias) and correlating these with the same camera templates.

## 5 RESULTS

We tested our forensic technique on a subset of 580 of the 1,694 images described in Section 3.2. These images spanned 24 different camera models and ranged in resolution from $1200 \times 1600$ to $4000 \times 6000$. Each image was subjected to one of four manipulations: (1) *copy-move*: a region in the image is duplicated; (2) *median filter*: a region is modified with a $3 \times 3$ median filter; (3) *rotation*: a region is randomly rotated by 10 to 80 degrees; and (4) *content-aware fill*: a region is removed using a standard content-aware fill algorithm [12]. The position of the square manipulated region was selected at random and ranged in size from 512, to 256, 128, or 64 pixels. This region may or may not be aligned to the $8 \times 8$ JPEG lattice. For the first set of experiments, the manipulated image was saved as a lossless PNG. In the second set of experiments, the manipulated image was saved as a lossy JPEG with varying qualities using JPEGLib with the slow-integer DCT implementation.

A total of 580 images, four manipulations, and four manipulation sizes yields 9,280 manipulated images. Each image was analyzed with our EM-based detection algorithm with a window size, $w(x, y)$, of size $64 \times 64$. Shown in Figure 6 are the receiver operating characteristic curves (ROC) for each manipulation size and averaged over all images and manipulation types. In this figure, the true positive rate (TPR) corresponds to correctly classifying a manipulated pixel as manipulated, and a false positive rate (FPR) corresponds to incorrectly classifying an original pixel as manipulated. Varying TPRs were achieved by adjusting the probability threshold for classifying a pixel as manipulated or not across all images and manipulation types. The area under the curve (AUC) ranges from a maximum of 0.90 for the largest manipulated region ($512 \times 512$) to a minimum of 0.81 for the smallest manipulated region ($64 \times 64$). The average TPR of 23.4% and 46.7% is obtained at FPR of 5% and 10%. These accuracies are computed on a per-pixel basis and are therefore particularly stringent and do not necessarily reflect the effective accuracy of isolating (if not precisely) a manipulated image region.

Shown in Figure 7(a) is a breakdown of the AUC for each manipulation type and size where we see that detection accuracy is similar regardless of manipulation type and that detection is slightly more difficult for smaller manipulations.

Shown in Figure 7(b) is the AUC for varying EM window size (previously fixed at $64 \times 64$), manipulation size, and averaged overall all manipulation types. As expected, the AUC is higher for a larger window size because it affords a more accurate estimate of the rounding artifact. Also as expected, the AUC is lower when the window size is larger than the manipulation size. For example, the AUC with a window size of 256 and manipulation size of 512 is 0.96, which falls to 0.58 for the same window size and manipulation size of 64.

Shown in Figure 7(c) is the effect of JPEG compressing the manipulated images using JPEGLib with varying qualities (on a scale of 100 (highest) to 1 (lowest)). As expected, the AUC is lower for lower quality images. At typical qualities of 80 and above, however, the AUC remains relatively high.

Shown in Figure 7(d) is a breakdown of the AUC for different camera manufacturers with a fixed manipulation size of $128 \times 128$. Each bar in this graph corresponds to all models per manufacturer. The AUC varies from 0.99 to 0.59 with Olympus being the easiest to detect and Motorola being the most difficult to detect. This is consistent with the magnitude of the rounding bias for these manufacturers, Figure 3.

Shown in Figure 8 are example images from the Nimble Challenge 2017 (NC2017) evaluation dataset [7]. Shown, from left to right, are an original image, a manipulated image, the ground-truth manipulation mask, and our estimated manipulation masks for the original and manipulated images, each with a window size of 256, 128, or 64. Each mask is displayed on a scale of [0, 1], where a value

[7] www.nist.gov/itl/iad/mig/nimble-challenge-2017-evaluation

**Table 2: The average AUC for each forgery type. Our proposed forgery detection performs better or similar to previous techniques.**

| | copy-move | median filter | rotate | region fill |
|---|---|---|---|---|
| BAG [28] | 0.58 | 0.66 | 0.61 | 0.69 |
| CDA [29] | 0.66 | 0.65 | 0.65 | 0.68 |
| I-CDA [7] | 0.71 | 0.74 | 0.74 | 0.68 |
| BG-CDA [6] | 0.66 | 0.63 | 0.66 | 0.66 |
| FDF-A [2] | 0.51 | 0.51 | 0.50 | 0.48 |
| CAGI [20] | 0.58 | 0.61 | 0.58 | 0.51 |
| SB [11] | 0.64 | **0.81** | 0.82 | 0.81 |
| OURS | **0.83** | 0.79 | **0.84** | **0.84** |

0 indicates an original pixel and a value of 1 indicates a manipulated pixel. The examples correspond to object insertion, object removal, object replacement, and copy-move manipulation . From top to bottom, the original JPEG images are recorded from (a) Sony Nex-5T, (b)-(e) Panasonic DMC-ZS40, (f) Sony DSC-S600, (g) Sony Nex-5T, (h) Panasonic DMC-ZS40, (i) Sony Nex-5T, and (j) Asus Nexus 7. In each case, a smaller window size is better at capturing small manipulations, but also leads to more false positives. Shown below the horizontal divider are the optimal template $c_1$ (Eq. 6) estimated by the EM algorithm for the original and forged images. The templates in each column are displayed on the same intensity scale. In each case, the estimated templates exhibit the same type of artifacts shown in Figure 3, indicating that our EM algorithm is latching onto the proposed rounding artifacts.

Lastly, the examples in Figure 9 correspond to representative failure cases. In each example, the failure occurs regardless of the window size.

## 6 COMPARISON WITH RELATED WORK

We compare our proposed method with seven other popular JPEG-based forensic algorithms: two are based on JPEG blocking artifacts (BAG [28] and CAGI [20]), three are based on AC coefficient distributions (CDA [29], I-CDA [7], and BG-CDA [6]), one is based on first digit distribution of AC coefficients (FDF-A [2]), and one is based on noise residuals (SB [11]).

In each case, we used the publicly available implementation for each algorithm. Specifically, implementations for BAG, CAGI, CDA, and FDF-A are downloaded from the Matlab image forensics toolbox [8] [43], and the remaining implementations were downloaded from the respective authors' webpages. The tampering probability maps from BAG, BG-CDA, and FDF-A algorithms are normalized by a logistic non-linearity as specified in [24]. The block size of $64 \times 64$ and the unsupervised scenario was used for our evaluation of SB. Except where noted above, we used the default parameters for all of the techniques.

In [24], the author evaluated different algorithms on a dataset [25] where forgeries were created using raw camera images. Since our algorithm relies on the artifacts associated with JPEG implementation within cameras, this dataset cannot be used to evaluate our algorithm. Instead, we evaluate all the algorithms on the diverse forgeries described in Section 4. We present the results for all four types of forgeries of size $512 \times 512$, recompressed with a quality in the range $[80, 100]$.

The overall performance of SB is the closest to the accuracy of our algorithm. The average AUC of SB, however, is only 0.64 for copy-move forgery compared to 0.83 obtained by our algorithm, Table 2. As the forged region in copy-move forgeries is taken from the same image without further modifications, the SB algorithm, which relies on differences in noise statistics, fails to detect this type of forgery. The slightly lower performance of our algorithm to detect median filter is due to the fact that median filter does not completely remove the artifact from the forged region.

## 7 DISCUSSION

We have shown how certain design decisions in a JPEG encoder can lead to a variety of different artifacts in JPEG-compressed images. We used a generic JPEG encoder to demonstrate that a limited set of values for a few compression parameters are sufficient to explain the structure and magnitude of the artifacts in many cameras. We also showed how to exploit this artifact to detect a range of manipulations from copy-move, to median filtering, re-sampling, and content-aware fill. Our approach out performs other JPEG-based forensic techniques.

There are, however, some limitations to our approach. Like most forensic techniques, our new technique suffers from the robustness vs. detection trade-off in which larger integration windows afford more robustness but also miss detecting small manipulations. Our technique is less effective in cameras with smaller magnitude artifacts that result from specific JPEG implementation choices by the camera manufacturer. And, as with most forensic techniques, our technique suffers from anti-forensic attacks where the artifact of interest can be added back to the image after manipulation.

In spite of these limitations, we believe that the current forensic technique nicely complements previous techniques and opens up new areas of investigation. For example, because the artifacts vary across camera manufacturer, the rounding artifacts can be used to identify an image as originating from a small class of cameras. And, because audio and video encoders implement similar transformations as JPEG encoders, seemingly innocuous encoding details may introduce similar artifacts which can in turn be forensically exploited.

## ACKNOWLEDGMENTS

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA FA8750-16-C-0166). The views, opinions, and findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

## REFERENCES

[1] S. Agarwal and H. Farid. 2017. Photo Forensics from JPEG Dimples. In *IEEE International Workshop on Information Forensics and Security* (Paris, France).
[2] I. Amerini, R. Becarelli, R. Caldelli, and A. Del Mastio. 2014. Splicing forgeries localization through the use of first digit features. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*. 143–148.
[3] Y. Arai, T. Agui, and M. Nakajima. 1988. A fast DCT-SQ scheme for images. *IEICE TRANSACTIONS (1976-1990)* 71 (1988), 1095–1097.

[8] https://github.com/MKLab-ITI/image-forensics

[4] CW Barnes, B. Tran, and S. Leung. 1985. On the statistics of fixed-point roundoff error. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 33 (1985), 595–606.

[5] M. Barni, L. Bondi, N. Bonettini, P. Bestagini, A. Costanzo, M. Maggini, B. Tondi, and S. Tubaro. 2017. Aligned and non-aligned double JPEG detection using convolutional neural networks. *Journal of Visual Communication and Image Representation* 49 (2017), 153–163.

[6] T. Bianchi and A. Piva. 2012. Image Forgery Localization via Block-Grained Analysis of JPEG Artifacts. *IEEE Transactions on Information Forensics and Security* 7 (2012), 1003–1017.

[7] T. Bianchi, A. De Rosa, and A. Piva. 2011. Improved DCT coefficient analysis for forgery localization in JPEG images. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2444–2447.

[8] J. C. Bins, B. A. Draper, W. AP Bohm, and W. Najjar. 1999. Precision vs. error in JPEG compression. In *Parallel and Distributed Methods for Image Processing III*. 76–88.

[9] Nicolò Bonettini, Luca Bondi, Paolo Bestagini, and Stefano Tubaro. 2018. JPEG Implementation Forensics Based on Eigen-Algorithms. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 1–7.

[10] N. Ik Cho and S Uk Lee. 1991. Fast algorithm and implementation of 2-D discrete cosine transform. *IEEE Transactions on circuits and systems* 38 (1991), 297–305.

[11] D. Cozzolino, G. Poggi, and L. Verdoliva. 2015. Splicebuster: A new blind image splicing detector. In *IEEE International Workshop on Information Forensics and Security*.

[12] A. Criminisi, P. Pérez, and K. Toyama. 2004. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing* 13 (2004), 1200–1212.

[13] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. 2007. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on image processing* 16, 8 (2007), 2080–2095.

[14] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato. 2015. RAISE: A Raw Images Dataset for Digital Image Forensics. In *ACM Multimedia Systems Conference*. 219–224.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)* (1977), 1–38.

[16] Hany Farid. 2006. *Digital Image Ballistics from JPEG Quantization*. Technical Report TR2006-583. Department of Computer Science, Dartmouth College.

[17] H. Farid. 2016. *Photo Forensics*. MIT Press.

[18] Miroslav Goljan, Jessica Fridrich, and TomÃąÅą Filler. 2009. Large scale test of sensor fingerprint camera identification. *Proc. SPIE* 7254 (2009), 72540I–72540I–12.

[19] J. He, Z. Lin, L. Wang, and X. Tang. 2006. Detecting Doctored JPEG Images Via DCT Coefficient Analysis. In *European Conference on Computer Vision*. Graz, Austria.

[20] C. Iakovidou, M. Zampoglou, S. Papadopoulos, and Y. Kompatsiaris. 2018. Content-aware detection of JPEG grid inconsistencies for intuitive image forensics. *Journal of Visual Communication and Image Representation* 54 (2018), 155–170.

[21] L. B. Jackson. 1970. On the interaction of roundoff noise and dynamic range in digital filters. *Bell System Technical Journal* 49 (1970), 159–184.

[22] Eric Kee, Micah K. Johnson, and Hany Farid. 2011. Digital Image Authentication from JPEG Headers. *IEEE Transactions on Information Forensics and Security* 6 (2011), 1066–1075.

[23] Jesse D Kornblum. 2008. Using JPEG quantization tables to identify imagery processed by software. *Digital Investigation* 5 (2008), S21–S25.

[24] P. Korus. 2018. Large-Scale and Fine-Grained Evaluation of Popular JPEG Forgery Localization Schemes. *CoRR* abs/1811.12915 (2018).

[25] P. Korus and J. Huang. 2017. Multi-Scale Analysis Strategies in PRNU-Based Tampering Localization. *IEEE Transactions on Information Forensics and Security* 12 (2017), 809–824.

[26] S. Kuhr. 2002. *Implementation of a JPEG decoder on a 16-bit microcontroller*. Ph.D. Dissertation. Masters Thesis, Department of Mathematics and Computer Science, Fachhochschule Stuttgart, Germany February.

[27] E. Y. Lam and J. W. Goodman. 2000. A mathematical analysis of the DCT coefficient distributions for images. *IEEE Transactions on Image Processing* 9 (2000), 1661–1666.

[28] W. Li, Y. Yuan, and N. Yu. 2009. Passive detection of doctored JPEG image via block artifact grid extraction. *Signal Processing* 89, 9 (2009), 1821–1829.

[29] Z. Lin, J. He, X. Tang, and C. Tang. 2009. Fast, automatic and fine-grained tampered JPEG image detection via DCT coefficient analysis. *Pattern Recognition* 42, 11 (2009), 2492–2501.

[30] C. Loeffler, A. Ligtenberg, and G. S. Moschytz. 1989. Practical fast 1-D DCT algorithms with 11 multiplications. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*. 988–991.

[31] W. Luo, Z. Qu, J. Huang, and G. Qiu. 2007. A Novel Method for Detecting Cropped and Recompressed Image Block. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*. 217–220.

[32] B. Mahdian and S. Saic. 2009. Detecting double compressed JPEG images. In *3rd International Conference on Crime Detection and Prevention*. 1–6.

[33] Matthieu Martel. 2002. Propagation of roundoff errors in finite precision computations: a semantics approach. In *European Symposium on Programming*. 194–208.

[34] A. V. Oppenheim and C. J. Weinstein. 1972. Effects of finite register length in digital filtering and the fast Fourier transform. *Proc. IEEE* 60 (1972), 957–976.

[35] Jinseok Park, Donghyeon Cho, Wonhyuk Ahn, and Heung-Kyu Lee. 2018. Double JPEG detection in mixed jpeg quality factors using deep convolutional neural network. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 636–652.

[36] Alin C. Popescu and Hany Farid. 2004. Statistical Tools for Digital Forensics. In *International Conference on Information Hiding* (Toronto, Canada). 128–147.

[37] K Ramamohan Rao and Ping Yip. 2014. *Discrete cosine transform: algorithms, advantages, applications*. Academic press.

[38] Il D. Tun and S. Uk Lee. 1993. On the fixed-point-error analysis of several fast DCT algorithms. *IEEE Transactions on Circuits and Systems for Video Technology* 3 (1993), 27–41.

[39] Martin Vetterli and Adriaan Ligtenberg. 1986. A discrete Fourier-cosine transform chip. *IEEE Journal on Selected Areas in Communications* 4 (1986), 49–61.

[40] Gregory K. Wallace. 1991. The JPEG Still Picture Compression Standard. *IEEE Transactions on Consumer Electronics* 34 (1991), 30–44.

[41] Bernard Widrow and Istvá Kollár. 2008. Quantization noise: roundoff error in digital computation, signal processing, control, and communications. (2008).

[42] S. Ye, Q. Sun, and E.C. Chang. 2007. Detecting Digital Image Forgeries by Measuring Inconsistencies of Blocking Artifact. In *IEEE International Conference on Multimedia and Expo*. 12–15.

[43] M. Zampoglou, S. Papadopoulos, and Y. Kompatsiaris. 2017. Large-scale evaluation of splicing localization algorithms for web images. *Multimedia Tools and Applications* 76, 4 (2017), 4801–4834.


# A DCT ALGORITHMS

$\mathrm{d}_1(\vec{v}, \vec{s}, f_e(\cdot), f_o(\cdot))$ ▷ Ligtenberg-Vetterli-DCT

1: $c_0 = c_4 = \mathrm{flt2fix}(1/\sqrt{2}, 13)$
$c_j = \mathrm{flt2fix}(\cos(j\pi/16), 13)$ ▷ $j \in \{1, 2, 3, 5, 6, 7\}$

2: $\vec{\mathcal{V}} = M\vec{v}$ ▷ 1-D DCT, $M$ in Eq. 7

3: $\vec{\mathcal{V}}(0) = f_e(\vec{\mathcal{V}}(0), s(0)); \vec{\mathcal{V}}(2) = f_e(\vec{\mathcal{V}}(2), s(2))$
$\vec{\mathcal{V}}(4) = f_e(\vec{\mathcal{V}}(4), s(4)); \vec{\mathcal{V}}(6) = f_e(\vec{\mathcal{V}}(6), s(6))$

4: $\vec{\mathcal{V}}(1) = f_o(\vec{\mathcal{V}}(1), s(1)); \vec{\mathcal{V}}(3) = f_o(\vec{\mathcal{V}}(3), s(3))$
$\vec{\mathcal{V}}(5) = f_o(\vec{\mathcal{V}}(5), s(5)); \vec{\mathcal{V}}(7) = f_o(\vec{\mathcal{V}}(7), s(7))$

**return** $\vec{\mathcal{V}}$

$\mathrm{d}_2(\vec{v}, \vec{s}, f_e(\cdot), f_o(\cdot))$ ▷ Loeffler-Ligtenberg-Moschytz-DCT

1: $c_0 = c_4 = 1 << 13$
$c_j = \mathrm{flt2fix}(\sqrt{2}\cos(j\pi/16), 13)$ ▷ $j \in \{1, 2, 3, 5, 6, 7\}$

2: $\vec{\mathcal{V}} = M\vec{v}$ ▷ 1-D DCT, $M$ in Eq. 7

3: $\vec{\mathcal{V}}(0) = f_e(\vec{\mathcal{V}}(0), s(0)); \vec{\mathcal{V}}(2) = f_e(\vec{\mathcal{V}}(2), s(2))$
$\vec{\mathcal{V}}(4) = f_e(\vec{\mathcal{V}}(4), s(4)); \vec{\mathcal{V}}(6) = f_e(\vec{\mathcal{V}}(6), s(6))$

4: $\vec{\mathcal{V}}(1) = f_o(\vec{\mathcal{V}}(1), s(1)); \vec{\mathcal{V}}(3) = f_o(\vec{\mathcal{V}}(3), s(3))$
$\vec{\mathcal{V}}(5) = f_o(\vec{\mathcal{V}}(5), s(5)); \vec{\mathcal{V}}(7) = f_o(\vec{\mathcal{V}}(7), s(7))$

**return** $\vec{\mathcal{V}}$

$$M = \begin{pmatrix} c_0 & c_0 & c_0 & c_0 & c_0 & c_0 & c_0 & c_0 \\ c_1 & c_3 & c_5 & c_7 & c_0 & c_0 & c_0 & c_0 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{pmatrix} \quad (7)$$

$d_3(\vec{v}, \vec{s}, f_e(\cdot), f_o(\cdot))$ ▷ Arai-Agui-Nakajima-DCT

$c_1 = \text{flt2fix}(\cos(4\pi/16), 13)$
$c_2 = \text{flt2fix}(\cos(6\pi/16), 13)$
$c_3 = \text{flt2fix}((\cos(2\pi/16) - \cos(6\pi/16)), 13)$
$c_4 = \text{flt2fix}((\cos(2\pi/16) + \cos(6\pi/16)), 13)$
$m_0 = v(0) + v(7)$; $m_7 = v(0) - v(7)$
$m_1 = v(1) + v(6)$; $m_6 = v(1) - v(6)$
$m_2 = v(2) + v(5)$; $m_5 = v(2) - v(5)$
$m_3 = v(3) + v(4)$; $m_4 = v(3) - v(4)$
$n_0 = m_0 + m_3$; $n_1 = m_1 + m_2$
$n_2 = m_1 - m_2$; $n_3 = m_0 - m_3$
$n_4 = m_4 + m_5$; $n_5 = m_5 + m_6$
$n_6 = m_6 + m_7$; $n_7 = m_7$
$o_0 = (n_0 << 13)$; $o_1 = (n_1 << 13)$
$o_2 = (n_3 << 13)$; $o_3 = (n_7 << 13)$
$o_4 = (n_2 + n_3)c_1$; $o_5 = (n_4 - n_6)c_2$
$o_6 = n_4 c_3$; $o_7 = n_6 c_4$; $o_8 = n_5 c_1$
$\vec{\mathcal{V}}(0) = f_e(o_0 + o_1, s(0))$; $\vec{\mathcal{V}}(2) = f_e(o_2 + o_4, s(2))$
$\vec{\mathcal{V}}(4) = f_e(o_0 - o_1, s(4))$; $\vec{\mathcal{V}}(6) = f_e(o_2 - o_4, s(6))$
$p_1 = f_o(o_3 + o_8, s(1))$; $p_2 = f_o(o_3 - o_8, s(3))$
$p_3 = f_o(o_5 + o_6, s(5))$; $p_4 = f_o(o_5 + o_7, s(7))$
$\vec{\mathcal{V}}(1) = p_1 + p_4$; $\vec{\mathcal{V}}(3) = p_2 - p_3$
$\vec{\mathcal{V}}(5) = p_2 + p_3$; $\vec{\mathcal{V}}(7) = p_1 - p_4$
**return** $\vec{\mathcal{V}}$

# B ROUNDING OPERATORS

flt2fix$(x, k)$ ▷ convert from floating- to fixed-point

$z = \text{int}(x \times 2^k + 0.5)$ ▷ eliminate fractional part
**return** $z$

floor$(x, y, b = 1)$ ▷ Compute $z = \text{floor}(x/y)$

**if** $b == 1$ **then**
  $z = x >> (\log_2(y))$ ▷ bit shift
**else**
  $z = \text{int}(x/y)$ ▷ ignore fractional part
**return** $z$

halfup$(x, y, b = 1)$ ▷ Compute $z = \text{halfup}(x/y)$

$x = x + (y >> 1)$ ▷ add 0.5 before rounding
$z = \text{floor}(x, y, b)$
**return** $z$

trunc$(x, y, b = 1)$ ▷ Compute $z = \text{trunc}(x/y)$

$\text{s} = \text{sign}(x)$
$x = |x|$
$z = \text{floor}(x, y, b)$
$z = s \times z$ ▷ set the correct sign
**return** $z$

round$(x, y, b = 1)$ ▷ Compute $z = \text{round}(x/y)$

$\text{s} = \text{sign}(x)$
$x = |x|$
$x = x + (y >> 1)$ ▷ add 0.5 before rounding
$z = \text{floor}(x, y, b)$
$z = s \times z$ ▷ set the correct sign
**return** $z$

# C JPEG IMPLEMENTATION

block-jpeg$(I)$

**Require:** $d(\cdot)$, $s$, $\vec{s}_h$, $\vec{s}_v$, $Q$, $f_q(\cdot)$, $f_2(\cdot)$, $f_e(\cdot)$, $f_o(\cdot)$, $f_s(\cdot)$
▷ $d(\cdot)$: 1-D DCT operator (see $d_1$, $d_2$, and $d_2$)
▷ $\vec{s}_h$: 8 1-D row DCT divisors for de-scaling
▷ $\vec{s}_v$: 8 1-D column DCT divisors for de-scaling
▷ $s$: 1 scalar for 2-D de-scaling
▷ $Q$: $8 \times 8$ 2-D DCT (scaled) quantization values
▷ $f_q(\cdot)$: rounding operator after quantization
▷ $f_2(\cdot)$: rounding operator after quantization with power of two
▷ $f_e(\cdot)$: rounding operator for even DCT coefficients in 1-D DCT de-scaling
▷ $f_o(\cdot)$: rounding operator for odd DCT coefficients in 1-D DCT de-scaling
▷ $f_s(\cdot)$: rounding operator 2-D de-scaling
$I = I - 128$ ▷ normalize into [-128,127]
▷ 2-D block DCT from two, 1-D DCTs
**for all** rows $i$ in $I$ **do** ▷ 1-D DCT on row
  $\mathcal{I}(i, :) = d(I(i, :)', \vec{s}_h, f_e(\cdot), f_o(\cdot))$
**for all** columns $j$ in $\mathcal{I}$ **do** ▷ 1-D DCT on column
  $\mathcal{I}(:, j) = d(\mathcal{I}(:, j), \vec{s}_v, f_e(\cdot), f_o(\cdot))$
▷ Scale and quantize DCT coeffficients
**for all** rows $i$ in $\mathcal{I}$ **do**
  **for all** columns $j$ in $\mathcal{I}$ **do**
    $\mathcal{I}(i, j) = f_s(\mathcal{I}(i, j), s)$ ▷ 2-D de-scale
    **if** $Q(i, j)$ is a power of 2 **then**
      $\mathcal{I}(i, j) = f_2(\mathcal{I}(i, j), Q(i, j))$ ▷ quantize
    **else**
      $\mathcal{I}(i, j) = f_q(\mathcal{I}(i, j), Q(i, j), 0)$ ▷ quantize
**return** $\mathcal{I}$