High Quality JPEG Compressor Detection via Decompression Error

Jan Butora, Patrick Bas

High Quality JPEG Compressor Detection via Decompression Error

Jan BUTORA, Patrick BAS

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
{jan.butora,patrick.bas}@cnrs.fr

Résumé – Dans ce travail, nous allons rechercher quel compresseur JPEG a été utilisé pour développer des images JPEG compressées avec un facteur de qualité de 100. Nous le faisons en inspectant les erreurs d'arrondi de l'image décompressée. Sous quelques hypothèses simples, nous pouvons dériver une distribution probabiliste de ces erreurs d'arrondi et détecter si le signal d'une image JPEG donnée suit une telle distribution. Toutefois, la compression JPEG peut être mise en œuvre de nombreuses façons différentes, ce qui peut affecter considérablement les hypothèses formulées. Cela peut conduire à une sous-performance importante d'un détecteur forensique qui peut être sensible au compresseur. Nos résultats sur le jeu de données Alaska montrent que nous pouvons créer un détecteur d'apprentissage profond qui, avec une précision proche de 100%, détermine quel compresseur JPEG a été utilisé pour la compression de l'image.

Abstract – In this work we will investigate which JPEG compressor was used to develop JPEG images compressed with quality factor 100. We will do this by inspecting the rounding errors of the decompressed image. Under a few simple assumptions, we can derive a probabilistic distribution of such rounding errors and detect whether this signal from a given JPEG image follow such a distribution. However, JPEG compression can be implemented in many different ways, which can greatly affect the assumptions made. This can lead to severe underperformance of a forensic detector that is not aware of possible differences caused by different compressors. Our results on Alaska dataset show that we can create a deep learning detector, which will with accuracy close to 100% correctly classify the JPEG compressor used for the image compression.

1 Introduction

Many digital image forensic tools greatly rely on the knowledge of the processing pipeline of a given image. If a processing pipeline is estimated poorly or is not considered at all, the forensic tool can suddenly suffer from many errors, simply because it is not aware of possible exceptions in the processing. This affects in particular the extremely popular JPEG image format. Even though the JPEG compression pipeline is theoretically fixed, for practical purposes there exists many different versions of implementing the DCT transformation and the final rounding operation [2] . These differences can lead to exploitable properties of the JPEG files, for example the JPEG dimples [1] caused by a bias introduced in the compressor. This can be used for forensics purposes, such as detecting copy-paste forgeries or double-compression. Unfortunately it can also create a lot of potential issues, especially if the detector does not consider possible differences in the compressors and the signal of interest is dependent on the actual implementation of the compression pipeline. It is therefore very useful to be able to distinguish which version of the JPEG compressor was used to create a given JPEG file. In this work we limit ourselves only on JPEG images compressed with Quality Factor (QF) 100, as this is the most sensitive scenario due to much smoother quantization, which keeps the compressor artifacts intact.

In the next section we remind the reader of the specifics of the JPEG compression and recall a model we can derive from this compression pipeline. Section 3 then points out several compression operations as well as steganography as a potential source of corruption of the derived model. In Section 4 we build a classifier which correctly predicts among several JPEG compressor with accuracy near 100%. Finally, the paper is concluded in Section 5.

2 JPEG compression

The JPEG format is a widely used image format, because the JPEG compression provides very good trade-off between image quality and size of the compressed file. The compression is fairly simple and can be summarized in several steps :

1. Transform the RGB color representation of the image into YCbCr (only the luminance channel Y is considered for grayscale images). The Y, Cb, and Cr components resulting from the color conversion are each processed independently.
2. Optionally subsample the two chrominance channels Cb and Cr, and round every pixel to integer value.
3. Subtract 128 from every pixel in order to ensure zero mean.
4. Divide every channel into 8x8 non-overlapping blocks.
5. Perform 2-D DCT transform on every 8x8 block.
6. Quantize every DCT block with a quantization matrix defined by JPEG Quality Factor (one matrix for luminance channel and one matrix for the two chrominance channels)
7. Round the quantized DCT coefficients into integers.

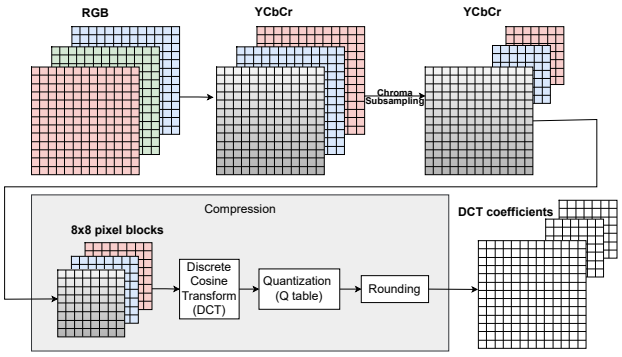

FIGURE 1 – Typical JPEG compression pipeline.

8. Losslessly encode the resulting structure into a JPEG file.

We assumed for simplicity that the original image is of dimensions that are divisible by 8. If that is not the case, the image can be symmetrically padded to comply with the desired format. To retrieve the decompressed image, the compression steps are performed in reverse order. Since steps 2 and 7 in the compression pipeline are lossy operations, we cannot retrieve the original uncompressed image, but we can reconstruct a very close estimate.

2.1 Model

In this Section, we express the Steps 3-7 mathematically in order to derive some properties of the JPEG images. We would like to point out at this point, that the input to step 3 is integer-valued, which will be important later. For better readability, everywhere in this paper, i, j will be strictly used to index in pixel domain and k, l will be indexing in DCT domain. Given a 8×8 block of pixels $\mathbf{x}$, the JPEG compression transforms these pixels into a block of unquantized DCT coefficients

$$\mathbf{d} = \mathrm{DCT}(\mathbf{x} - 128),$$

where $\mathrm{DCT}(\cdot)$ represents the 2-D DCT transformation. These coefficients are then quantized into integer values $\mathbf{c} = [\mathbf{d} \oslash \mathbf{q}]$, where $\oslash$ represents element-wise division and $\mathbf{q}$ is the quantization matrix specified by the Quality Factor used for the compression. Note that due to the rounding operation, the JPEG compression is lossy, because we cannot retrieve the DCT errors that the rounding produces $\mathbf{u} = \mathbf{d} \oslash \mathbf{q} - \mathbf{c}$. As a consequence, when we decompress the DCT coefficients $\mathbf{c}$ back to spatial domain by reverting the compression steps, we obtain a different pixel representation

$$\mathbf{y} = \mathrm{DCT}^{-1}(\mathbf{c} \odot \mathbf{q}) + 128 \neq \mathbf{x},$$

where $\odot$ represents element-wise multiplication and $\mathrm{DCT}^{-1}(\cdot)$ is the inverse 2-D DCT transformation. The compression pipeline is visualized in Figure 1.

3 Reverse JPEG Compatibility

In practice we cannot say much about the error we introduce in the spatial domain with respect to the uncompressed image $\mathbf{y} - \mathbf{x}$, because we cannot retrieve the uncompressed image $\mathbf{x}$ from a given JPEG file. However, assuming that the original uncompressed pixels are integer-valued [1], we can derive a model for the spatial domain rounding error $\mathbf{e} = \mathbf{y} - [\mathbf{y}]$. In fact, we can further assume that the DCT rounding errors u_{kl} are i.i.d. and follow a uniform distribution $u_{kl} \sim U(-1/2, 1/2)$. This is a reasonable assumption easily verifiable as long as the DCT coefficients during compression are rounded to the nearest integer. In such case a recent paper aimed at steganalysis of JPEG images compressed with QF 100 [5] shows that for JPEG images compressed with Quality Factor 100, the spatial domain errors follow the so-called Wrapped Gaussian Distribution

$$e_{ij} \sim \mathcal{N}_W(0, 1/12) \tag{1}$$

(see Figure 2). However many situations destroying this property can arise. We will point out a few of them in the following.

3.1 Effect of Steganography

As the main motivation of this work, we first mention the effect of steganographic embedding on the rounding errors. Indeed, since (1) is a natural behaviour of JPEG images, it can be used for very accurate steganalysis because changing the DCT coefficients increases the variance of the Wrapped Gaussian distribution $e^S_{ij} \sim \mathcal{N}_W(0, 1/12 + v_{ij})$ [5], where v_{ij} depend on the size of the secret message and e^S_{ij} are spatial domain rounding errors coming from a stego image. This increase in variance is very exploitable even for very small steganographic messages, because the Wrapped Gaussian distribution is very sensitive to changes in variance (see Figure 2). Interestingly, the same arguments apply for any method changing the DCT coefficients, not only steganography. However this property can only be exploited if we are sure that the decompression errors of cover images follow (1). If it is not the case, any decision making can possibly introduce a lot of errors.

3.2 Final Compression Rounding

We cannot always assume that the DCT errors are uniformly distributed. For instance, many imaging devices use rounding towards zero (trunc quantizer) as the final step of JPEG compression, which changes the properties of the DCT errors [6]. For example, let c_{kl} be a DCT coefficients that was quantized to zero. Then we can model its DCT error as $u_{kl} \sim U(-1, 1)$ and easy calculation reveals that in an extreme case when a DCT block consists of zero coefficients only, the errors e_{ij} would

1. To the best of our knowledge, every publicly available JPEG compressor converts the pixels into integers before compression.

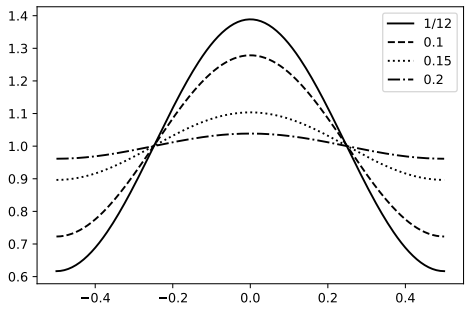

FIGURE 2 – Probability density function of the Wrapped Gaussian distribution $\mathcal{N}_W(0, \nu)$ for different values of ν.

follow Wrapped Gaussian distribution $e_{ij} \sim \mathcal{N}_W(0, 1/3)$, which resembles uniform noise (see Figure 2). Since the vast majority of the DCT coefficients are equal to zero (due to the nature of DCT transformation), this prevents us from using the errors e_{ij} for reliable decision making.

3.3 Chrominance Subsampling

The chrominance subsampling can be implemented in two main ways. First, the more obvious way subsamples in the spatial domain by simple averaging the four neighbouring pixels of the Cb and Cr channels (step 2 of the compression pipeline in Section 2). Even though averaging 4 pixels can end up in non-integer value, the compressors still round the averages to integers before the DCT transformation. We verified this only experimentally and indeed, if we disallow the rounding of the averages to integers, our model on the errors $\mathbf{e}$ is completely broken and resembles uniform noise instead, which is expectable because only a quarter of all pixels entering the JPEG compression have integer value after such subsampling. This is a crucial observation, which will allow us to detect different subsampling methods by inspecting the decompression errors $\mathbf{e}$.

Another way, used for example in libjpeg version 7 [3], uses subsampling in the DCT domain. After step 5 of the compression pipeline, the downsampling is performed on 2×2 neighbouring 8×8 DCT blocks, combining only their low-frequency components, see the details in Section 2B in [8]. In this scenario, the resulting downsampled DCT block is a combination of 4 original neighbouring blocks. Consequently, this breaks the model we assume for the spatial domain rounding errors and they will follow uniform distribution instead.

4 Experiments

We took 25,000 color TIFF images of size 256×256 from ALASKA2 dataset [7] and JPEG compressed them with four different compressors causing different behaviour of the

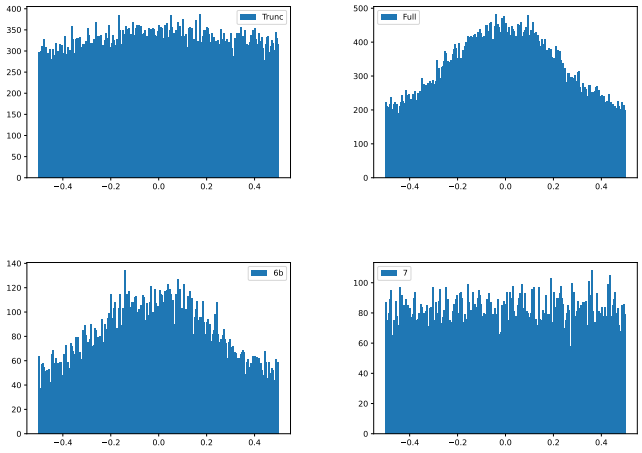

FIGURE 3 – Histograms of rounding errors $\mathbf{e}$ coming from the blue channel Cb compressed with different compressors. Same image was used for every compressor.

errors $\mathbf{e}$. First, we computed the DCT coefficients manually by performing steps 1-7 from the compression pipeline (without chroma subsampling) and with the trunc quantizer as the final rounding operation. This method will be referred to as Trunc. For the other three pipelines, we used libjpeg library with a Python wrapper that supports version selection. [2] For all three compressors, we used the slow DCT method (in fact all available methods produce the same results at QF 100 for a fixed libjpeg version). Once, we did not use any chroma subsampling, which again produces the same results for every libjpeg version. For the other two compressors, we used libjpeg 6b and 7 with default 4 :2 :0 chroma subsampling, which subsamples both chrominance channels by a factor of two in both dimensions. These three methods will be referred to as Full, 6b, and 7. It was shown in [3] that for QF 100 with the default chroma subsampling, only two possible clusters of compressors exist and the selected versions are their respective representatives. Furthermore we observed that mozjpeg [9] produces the same DCT coefficients as libjpeg 6b, but it uses progressive JPEG instead. Since that will not have any effect on the errors $\mathbf{e}$, we excluded mozjpeg from our experiments, since the progressive method can be read form the header. Also note that all the libjpeg versions provide the same results for the luminance channel. We then divide our images into training, validation, and testing sets of sizes 22,000, 1,000, and 2,000.

To train a detector distinguishing a JPEG compressor via the errors $\mathbf{e}$, we used a multi-class EfficientNet-B0 [10]. On input we provide the rounding errors of decompressed chrominance channels $\mathbf{e}^{Cb}$, $\mathbf{e}^{Cr}$ stacked as two channels of an image. Since some of the compressors perform chrominance subsampling, we will upsample the inputs (if needed) to size 256×256. Finally, because the EfficientNet requires three channel input (it was pretrained on color images), we add a 1×1 convolution

2. https ://github.com/martinbenes1996/jpeglib

	Trunc	Full	6b	7
Trunc	1994	6	0	0
Full	0	2000	0	0
6b	0	7	1993	0
7	0	0	0	2000

TABLE 1 – Confusion matrix of the multi-class EfficientNet

into the beginning of the network mapping two channel inputs into three channels. The rest of the hyperparameters are kept as in [4].

4.1 Results

To verify that there is a potential for detection through the errors $\mathbf{e}$, we will first take a look at their behaviour with respect to the tested compressors. In Figure 3 we can observe the histograms of rounding errors $\mathbf{e}^{Cb}$ coming from decompressed Cb channel of a single image. We can observe what we expected. With the trunc quantizer, the Wrapped Gaussian distribution has a very large variance, therefore it almost looks uniform. For compressors Full and 6b, we see a nice Wrapped Gaussian shape, while for 6b the distribution is more noisy, because we only have a quarter of all original pixels, due to the subsampling. The biggest difference comes for libjpeg 7, where we can observe uniform noise, which is due to the subsampling in the DCT domain, as mentioned previously.

In Table 1 we can observe the results of our multi-class EfficientNet trained only on the chrominance rounding errors $\mathbf{e}^{Cb}$, $\mathbf{e}^{Cr}$. For images compressed with the Trunc quantizer, only 6 images got misclassified, a fact that can probably be avoided by adding the errors $\mathbf{e}^{Y}$ coming from the decompressed luminance channel as additional input to the network detector. The only other mistake the detector does is misclassifying libjpeg 6b with subsampling as libjpeg without subsampling. This is most likely due to upsampling of the errors $\mathbf{e}^{Cb}$, $\mathbf{e}^{Cr}$ before feeding them into the network. As this was only done in order to have unified size of the network inputs, we can clearly correct for these mistakes by taking the original size of the chrominance DCT channels into account. All in all, we can observe that inspecting only decompression errors coming from the two chrominance channels gives almost perfect prediction about the compressor used.

5 Conclusions

In this work we verified that JPEG images created by various JPEG compressors produce very different decompression errors and we build a near-perfect deep learning detector distinguishing the compressors by inspecting these decompression errors. Interestingly, we did not even have to use all three color channels for the predictions, allowing potentially even better performance, if we were to use also the luminance channel. This compressor detection can be particularly useful in steganalysis, if the steganalyst builds a detector sensitive to changes in the distribution of the decompression rounding errors. For instance, given an image producing uniformly looking rounding errors does not necessarily mean steganography was used, as was previously believed. We have seen that this can be caused by different rounding operation performed on DCT coefficients or by subsampling of chrominance channels in the DCT domain. In our future work, we plan on investigating the effect of steganography on prediction of the JPEG compressor.

Références

[1] S. Agarwal and H. Farid. Photo forensics from JPEG dimples. In *IEEE Workshop on Information Forensics and Security (WIFS)*, December 4-7, 2017.

[2] S. Agarwal and H. Farid. Photo forensics from rounding artifacts. In C. Riess and F. Schirrmacher, editors, *The 8th ACM Workshop on Information Hiding and Multimedia Security*, Denver, CO, June 22–25, 2020. ACM Press.

[3] M. Beneš, N. Hofer, and R. Böhme. Know your library : How the libjpeg version influences compression and decompression results. In *The 10th ACM Workshop on Information Hiding and Multimedia Security*, Santa Barbara, CA, June 27–29, 2022. ACM Press.

[4] J. Butora and P. Bas. Fighting the reverse JPEG compatibility attack : Pick your side. In *The 10th ACM Workshop on Information Hiding and Multimedia Security*, Santa Barbara, CA, June 27–29, 2022. ACM Press. Under review.

[5] J. Butora and J. Fridrich. Reverse JPEG compatibility attack. *IEEE Transactions on Information Forensics and Security*, 15 :1444–1454, 2020.

[6] J. Butora and J. Fridrich. Steganography and its detection in JPEG images obtained with the "trunc" quantizer. In *Proceedings IEEE, International Conference on Acoustics, Speech, and Signal Processing*, Barcelona, Spain, May 4–8, 2020.

[7] R. Cogranne, Q. Giboulot, and P. Bas. ALASKA–2 : Challenging academic research on steganalysis with realistic images. In *IEEE International Workshop on Information Forensics and Security*, New York, NY, December 6–11, 2020.

[8] R. Dugad and N. Ahuja. A fast scheme for image size change in the compressed domain. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(4) :461–474, 2001.

[9] Mozilla Foundation. mozjpeg : Improved JPEG encoder. https ://github.com/mozilla/mozjpeg, 2014.

[10] T. Mingxing and V. L. Quoc. EfficientNet : Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, volume 97, pages 6105–6114, June 9–15, 2019.