# JPEG-resistant Adversarial Images

**Richard Shin**
Computer Science Division
University of California, Berkeley
ricshin@cs.berkeley.edu

**Dawn Song**
Computer Science Division
University of California, Berkeley
dawnsong@cs.berkeley.edu

## Abstract

Several papers have explored the use of JPEG compression as a defense against adversarial images [3, 2, 5]. In this work, we show that we can generate adversarial images which survive JPEG compression, by including a differentiable approximation to JPEG in the target model. By ensembling multiple target models employing varying levels of compression, we generate adversarial images with up to 691× greater success rate than the baseline method on a model using JPEG as defense.

## 1 Introduction

Image classification models has been a highly-studied domain for *adversarial examples*. While adversarial images generated against these models are nevertheless very close to the original image according to $L_\infty$ or $L_2$ norm, unnatural high-frequency components or random-looking dot patterns are sometimes noticeable. As such, some proposed defenses to adversarial examples involve an initial input transformation step which attempts to remove such unnatural-looking additions.

In particular, several papers [3, 2, 5] have proposed and evaluated JPEG compression as a potential method for preventing adversarial images. To summarize, they compress and then decompress an image using JPEG before providing it to the image classification model. Since JPEG is a lossy image compression method designed to preferentially preserve details important to the human visual system, the hope is that JPEG compression will keep the aspects of the image important for classification, but discard any adversarial perturbations. The method is appealingly simple and, according to the previous work, can reduce the attack success rate of adversarial examples.

However, these previous works did not consider an *adaptive* adversary that can change the adversarial example generation method to counteract this defense. Previous work [6] has shown that adaptive adversaries can defeat other input transformation-based defenses, such as median filtering and bit depth reduction. Can an adaptive adversary also defeat the JPEG compression defense? Our answer is yes. While JPEG compression is not differentiable, we show how to closely approximate it using only differentiable operations. Afterwards, we can use existing adversarial example generation methods against the combination of differentiable JPEG compression and the original image classification model. Our empirical results using the ImageNet dataset show that we can generate adversarial examples which attack a model with JPEG compression just as well as a plain model. Furthermore, by attacking an ensemble of models where each applies a different level of JPEG compression, we can ensure that the resulting adversarial examples can successfully fool classification models using the JPEG defense at a variety of settings, or no JPEG defense at all.

## 2 Preliminaries

Consider an image $x \in [0,1]^{H\times W\times C}$ and an image classifer $C(x) \in \Delta^K$ where $\Delta$ is the probability simplex and $K$ is the number of classes. A *non-targeted* adversarial example $x'$ satisfies $\arg\max_k C(x)_k \neq \arg\max_k C(x')_k$ and $\|x' - x\|$ is small, for some distance metric $\|\cdot\|$.

Given a loss function $\ell(\cdot,\cdot)$, we attempt to find $x'$ by solving $\arg\max_{x'} \ell(C(x), C(x'))$ s.t.$\|x'-x\| < d$. In this work, we used two methods for approximating a solution this optimization problem:

- Fast gradient sign method [4]: $x' = x + \epsilon \cdot \text{sign}(\nabla_{x'}[\ell(C(x), C(x'))]_{x'=x})$. This method ensures that $\|x' - x\|_\infty = \epsilon$.
- Iterative FGSM [7]: $x^{(i+1)} = x^{(i)} + \frac{\epsilon}{N} \cdot \text{sign}(\nabla_{x'}[\ell(C(x), C(x'))]_{x'=x^{(i)}})$, for all $i = 0, \cdots, N-1$. $x = x^{(0)}$ and $x' = x^{(N)}$. This is equivalent to the fast gradient sign method run for $N$ iterations.

After each update, we apply clipping to $[0, 1]$ to ensure that the image remains within bounds. If the true class $y$ is known for a given image $x$, we can instead solve $\arg\max_{x'} \ell(\mathbf{1}_y, C(x'))$ s.t.$\|x'-x\| < d$, where $\mathbf{1}_y \in \mathbb{R}^K$ is a one-hot vector with the $y$th element set to 1. We use this variant for our experiments.

We also consider a *targeted* attack, satisfying $\arg\max_k C(x')_k = y'$ and $\|x' - x\|$ is small, by optimizing $\arg\min x' \ell(\mathbf{1}_{y'}, C(x'))$ s.t.$\|x' - x\| < d$ where $y'$ is the target class. For compactness, the remainder of the exposition assumes a non-targeted attack.

## 3 JPEG compression

JPEG compression first converts the image into a luma-chroma color space (YCbCr), downsampling the chroma channels, transforming each 8x8 block with a two-dimensional discrete cosine transform, and then quantizing of each block. Afterwards, lossless compression techniques are used on this output, but they are not relevant for our purposes. In this section, we describe how to perform each step using differentiable operations, straightforward to implement in common libraries like TensorFlow [1].

**Color space conversion.** While digital images are most commonly displayed using the RGB color space, JPEG uses the YCbCr color space. The conversion is an affine transformation $y = Mx + b$, where $x \in \mathbb{R}^3$ is a color in RGB, $y \in \mathbb{R}^3$ is in YCbCr, and $M \in \mathbb{R}^{3\times3}$, $b \in \mathbb{R}^3$ are coefficients given in the appendix.

**Chroma subsampling.** In the YCbCr color space, the Y component represents the overall brightness (luminance) of the pixel, whereas Cb and Cr encode the color (chrominance) separate from the brightness. As humans can discern finer detail in brightness versus color, JPEG downsamples the Cb and Cr channels by a factor of two in each dimension. We achieve this by using 2x2 average pooling with a stride of 2.

**Block splitting.** The subsequent parts of the compression method deal with 8x8 blocks in each channel. Due to the chroma subsampling, note that the blocks for Cb and Cr correspond to 16x16 blocks in the original image, which is why we needed padding to the next multiple of 16.

**Discrete cosine transform.** Let us represent a 8x8 block represented as matrix $B \in \mathbb{R}^{8\times8}$ into a 64-dimensional vector, such that $v_{8i+j} = B_{i,j}$ $(0 \leq i, j \leq 7)$. Then we can implement the discrete cosine transformation as $w = A \odot Gv$ where $A \in R^{64}$ is a scaling factor, $\odot$ is the Hadamard (element-wise) product, and $G \in R^{64\times64}$ contains the DCT coefficients, which are shown in the appendix. Afterward, we rearrange the DCT output $w \in \mathbb{R}^{64}$ into $C \in \mathbb{R}^{8\times8}$, where $C_{i,j} = w_{8i+j}$ $(0 \leq i, j \leq 7)$. The top left corner of $C$ $(i = 0, j = 0)$ contains the lowest-frequency component of the original 8x8 block, while the bottom right corner $(i = 7, j = 7)$ contains the highest-frequency component.

**Quantization.** Other than the chroma subsampling step, the other steps above are lossless transformations if performed with sufficient precision. Their purpose is to enable the quantization step to efficiently discard a large amount of data while preserving as much as possible the appearance of the image to the human eye.

In JPEG, we use a *quantization matrix* $Q \in 8 \times 8$ to create $D_{i,j} = \left\lfloor \frac{C_{i,j}}{Q_{i,j}} \right\rceil$, where $\lfloor\cdot\rceil$ represents rounding to the nearest integer. The design of $Q$ reflects characteristics of the human visual system:

the values of $Q$ in the bottom right are larger than in the top left, so as to preferentially discard the high-frequency components. Using larger values for $Q$ results in greater compression, as after quantization, more entries of $D$ become 0.

**Differentiable approximation to rounding.** $\lfloor\cdot\rceil$ has derivative 0 nearly everywhere, which is not compatible with the gradient-based methods used for generating adversarial examples. We instead used the approximation $\lfloor x\rceil_{approx} = \lfloor x\rceil + (x - \lfloor x\rceil)^3$, which has non-zero derivatives nearly everywhere, and close to $\lfloor x\rceil$ (maximum discrepancy occurs at $n + 0.5$ for integers $n$, where $\lfloor n + 0.5\rceil_{approx} - \lfloor n + 0.5\rceil = 0.125$).

**Decoding.** Recovering an RGB image involves performing the inverse of the above steps in reverse order. Element-wise multiplication with $Q$ (lossily) reverses quantization; the inverse discrete cosine transform is a similar linear operation with different coefficients; conversion from YCbCr to RGB is also an affine transformation.

## 4 Creating JPEG-resistant adversarial images

Originally, we classify each image $x$ by using the classifier $C(x)$. Instead, the JPEG-based defense uses $C(JPEG(x, q))$, where $JPEG(x, q)$ compresses $x$ using quality level $q$ (an integer between 0 and 100) and then decompresses the result to produce a slightly transformed version of $x$. The premise is that for an adversarial example $x'$ created from $x$, even if $\arg\max_k C(x)_k \neq \arg\max_k C(x')_k$, then $\arg\max_k C(JPEG(x, q))_k = \arg\max_k C(JPEG(x', q))_k$ for some suitable $q$.

To attack this defense, we will use $JPEG_{\text{diff}}(x, q)$, a differentiable approximation to JPEG described in Section 3. In other words, our optimization problem is $\arg\max_{x'} \ell(C(x), C(JPEG_{\text{diff}}(x', q)))$ s.t.$\|x' - x\| < d$. This requires evaluating the gradient $\nabla_{x'}[\ell(C(x), C(JPEG_{\text{diff}}(x', q)))]$ which is straightforward to do using common libraries like TensorFlow.

However, we found that using this gradient could result in adversarial examples which are overly specialized to the quality level used. To ensure that we can generate robust adversarial examples, we also tried using an *ensemble* of models employing different $q$, also including a model without any JPEG compression.

Our ensemble weights the gradients computed from each model based on the relative magnitude of the losses. Given various quality levels $q_i$, and setting $L_i = \ell(C(x), C(JPEG_{\text{diff}}(x', q_i)))$, we use the gradient $\sum_{i=1}^{n} \left(1 - \frac{\exp(L_i)}{\sum_i \exp(L_i)}\right) \nabla_{x'}[L_i]$ for the untargeted attack. Intuitively, when a given value of $L_i$ is smaller, we place a larger weight on $\nabla_{x'}[L_i]$ for the combined gradient, in order to increase $L_i$ by a greater amount.

## 5 Experimental results

For our experiments, we used a pretrained ResNet-50 model[1] and the first 1000 images of the ILSVRC 2012 object recognition validation set. We resized all images to $224 \times 224$ without preserving the aspect ratio and used single-crop evaluation. We used 25, 50, and 75 as the quality value $q$ in JPEG compression; in the ensemble for the gradient, we also include a model with no JPEG compression (so the ensemble contains 4 different models). For evaluating the JPEG defense, we used the JPEG encoder and decoder provided by TensorFlow; for attacking the JPEG defense, we implemented $JPEG_{\text{diff}}$ as described in Section 3. For consistency, all models reduce their input to 8 bits, by truncating down to a multiple of $1/255$. For targeted attacks, we used $y' = (y + 500) \mod 1000$ as the target class. Table 1 shows our main results. We explain their implications in this section.

**JPEG compression on benign images.** JPEG compression may cause the system to misclassify benign inputs, as reported by Guo et al [5]. Row 1 of Table 1 shows that at the fairly low quality level of $q = 25$, the accuracy goes down by 10 percentage points.

[1]Obtained from `https://github.com/tensorflow/models/tree/master/research/slim`

| | | | | | | Result on eval. with model | | | |
|---|---|---|---|---|---|---|---|---|---|
| # | U/T | Attack method | Model for $\nabla$ | Mean $L_2$ | Mean $L_\infty$ | $q=\infty$ | $q=75$ | $q=50$ | $q=25$ |
| 1 | | None | N/A | 0 | 0 | 70.7% | 66.3% | 64.7% | 60.2% |
| 2 | U | FGSM, $\epsilon=\frac{1}{255}$ | $q=\infty$ | 1.51 | 1/255 | 12.2% | 36.8% | 45.8% | 51.6% |
| 3 | | | $q=25$ | 1.51 | 1/255 | 59.9% | 51.6% | 45.3% | 22.5% |
| 4 | | | ensemble | 1.51 | 1/255 | 20.9% | 27.8% | 31.5% | 31.1% |
| 5 | U | FGSM, $\epsilon=\frac{3}{255}$ | $q=\infty$ | 4.51 | 3/255 | 4.8% | 14.6% | 23.1% | 35.4% |
| 6 | | | ensemble | 4.51 | 3/255 | 9.2% | 10.1% | 10.1% | 15.0% |
| 7 | U | FGSM, $\epsilon=\frac{9}{255}$ | $q=\infty$ | 13.44 | 9/255 | 5.9% | 8.8% | 11.3% | 16.9% |
| 8 | | | ensemble | 13.44 | 9/255 | 4.5% | 5.0% | 5.6% | 7.9% |
| 9 | U | I-FGSM, $\epsilon=\frac{1}{255}$ | $q=\infty$ | 1.00 | 1/255 | 1.6% | 34.6% | 47.4% | 52.0% |
| 10 | | | $q=25$ | 0.97 | 1/255 | 62.2% | 51.8% | 46.6% | 12.9% |
| 11 | | | ensemble | 1.02 | 1/255 | 9.1% | 18.0% | 23.6% | 27.9% |
| 12 | U | I-FGSM, $\epsilon=\frac{3}{255}$ | $q=\infty$ | 2.24 | 3/255 | 0.2% | 9.5% | 30.4% | 44.2% |
| 13 | | | q = 50 | 2.13 | 3/255 | 23.3% | 6.5% | 0.6% | 45.9% |
| 14 | | | ensemble | 2.42 | 3/255 | 1.0% | 1.1% | 4.6% | 10.6% |
| 15 | T | I-FGSM, $\epsilon=\frac{7}{255}$ | $q=\infty$ | 3.89 | 7/255 | 0.992 | 0.096 | 0.008 | 0.001 |
| 16 | | | $q=25$ | 4.11 | 7/255 | 0.013 | 0.033 | 0.142 | 0.645 |
| 17 | | | ensemble | 4.41 | 7/255 | 0.972 | 0.905 | 0.842 | 0.691 |
| 18 | T | I-FGSM, $\epsilon=\frac{9}{255}$ | $q=\infty$ | 4.75 | 9/255 | 0.983 | 0.255 | 0.018 | 0.003 |
| 19 | | | $q=75$ | 4.82 | 9/255 | 0.723 | 0.946 | 0.060 | 0.007 |
| 20 | | | ensemble | 5.33 | 9/255 | 0.985 | 0.963 | 0.934 | 0.829 |

Table 1: Summary of results; **please refer to body for explanation**. U/T indicates untargeted/targeted. For U, % is the adversarial input's accuracy (*lower* number is a better attack). For T, fraction is success rate (*higher* number is a better attack). $q=\infty$ indicates no JPEG defense. I-FGSM uses $N=10$ iterations.

**Untargeted FGSM.** In row 3, using gradients from a model including $JPEG_{\text{diff}}(x, 25)$ reduces accuracy to 22.5%, from 51.6% in the unadaptive attack (row 2). However, the attack does not generalize to a JPEG-less model ($q=\infty$), with 59.9% accuracy. Ensembling gradients (row 4) leads to an attack with greater success for $q=\infty, 75, 50$, but with modest declines for $q=25$. With larger $\epsilon$ (rows 5-8), the attacks are more successful, but the defense is also weaker (e.g., 16.9% accuracy for $q=25$ defense in row 7).

**Untargeted I-FGSM.** Comparing row 9 to row 2, I-FGSM attacking $q=\infty$ is more successful against an undefended model (1.6% vs 12.2%). However, results against $q=75, 50, 25$ remain similar. In row 10, the $q=25$ attack again fails to generalize to other $q$ values used as defense; similarly for $q=50$ with $\epsilon=3/255$ in row 13. Comparing row 11 to row 4 and row 14 to row 6, I-FGSM with the ensemble performs better than FGSM.

**Targeted I-FGSM.** For a targeted attack to succeed, the resulting image must be classified as a specific class out of 1000. This is more difficult than the untargeted attack (where any of 999 classes suffices). Indeed, we see in row 15 that the unadaptive attack is 99.2% successful against the undefended model, but only 0.1% successful with $q=25$ defense (for targeted, *higher* numbers are better). However, the adaptive attack against the ensemble fares much better: against $q=25$ defense, the success rate rises to 69.1% and 89.2% on rows 17 and 20; hence at $\epsilon=7/255$, our success rate is **691$\times$ greater**. Attacks against only one $q$ do not transfer well; on row 16, the $q=25$ attack only succeeds 1.3% on the undefended model ($q=\infty$ column), and on row 19, the $q=75$ attack succeeds 6% on the $q=50$ defense and $0.7\%$ on the $q=25$ defense.

## 6 Conclusion

In this paper, we showed how to defeat the JPEG defense by performing an adaptive attack with a differentiable JPEG approximation. By ensembling target models that use varying amounts of compression, our adversarial examples generalize to models with and without this defense.

## References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Li Chen, Michael E. Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression, 2017.

[3] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M. Roy. A study of the effect of jpg compression on adversarial images, 2016.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations, 2017.

[6] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defenses: Ensembles of weak defenses are not strong. *arXiv preprint arXiv:1706.04701*, 2017.

[7] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

## A JPEG compression details

**RGB to YCbCr.**

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

where all values are in $[0, 255]$.

**Discrete cosine transform.** We expressed the 2D discrete cosine transform over 8x8 blocks, used in JPEG, as $w = A \odot Gv$.

$$\begin{aligned} G &\in \mathbb{R}^{64\times 64} \\ G_{8u+v,8x+y} &= \cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right) && x, y, u, v \in [0, 7] \\ A &\in \mathbb{R}^{64} \\ A_{8u+v} &= \frac{1}{4}\alpha(u)\alpha(v) && u, v \in [0, 7] \\ \alpha(u) &= \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

**Quantization matrix.** For the luminance channel (Y), we use the following quantization matrix $Q$:

$$Q_Y = s \cdot \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 10 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}^T$$

For the chrominance channels (Cb and Cr):

$$Q_C = s \cdot \begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}^T$$

$s$ is a scalar determined from the quality level $q$:

$$s = \begin{cases} \frac{50}{q} & \text{if } q < 50 \\ 2 - \frac{2q}{100} & \text{otherwise} \end{cases}$$