What's in the Image?
Explorable Decoding of Compressed Images
Supplementary Material

Yuval Bahat and Tomer Michaeli
Technion - Israel Institute of Technology, Haifa, Israel
{yuval.bahat@campus,tomer.m@ee}.technion.ac.il

A. Performance Comparison

Our JPEG decoding framework is the first to facilitate exploration of the abundant plausible images corresponding to a given compressed JPEG code, and therefore cannot be compared to any existing method. Nonetheless, it produces high quality outputs even prior to applying any user editing. To evaluate the quality of pre-edited outputs and compare it with that of existing JPEG artifact removal methods, we perform experiments using two datasets commonly used for evaluating artifact removal, namely the LIVE1 [40] and BSD-100 [36] datasets, containing 29 and 100 images, respectively.

Methods for removing JPEG artifacts strive to achieve one of two possible goals; either they attempt to minimize outputs' distortion with respect to corresponding ground truth (GT) uncompressed images, or they try to maximize their outputs' perceptual quality. We quantitatively evaluate the performance with respect to each of these different goals, by adapting the commonly used metric for each goal: Distortion is evaluated by measuring peak signal to noise ratio (PSNR) w.r.t. GT images, while perceptual quality is evaluated using both the naturalness image quality evaluator (NIQE, [38]) no-reference score, and the learned perceptual image patch similarity (LPIPS, [39]) full-reference score. We compare our method with DnCNN [19] and AGARNet [21], the only AR methods handling a range of compression levels (like ours) whose code is available online. As these methods aim for minimal distortion rather than perceptual quality, we train a variant of our model using the L_1 penalty instead of the full penalty in (1), which is based on adversarial loss. Our Y channel reconstruction L_1 model uses 370 output channels for each convolution operation but the last, instead of the 320 channels used for our perceptually oriented Y channel model. Like AGARNet, we train this variant using a wider range of QFs, spanning from 1 to 90. To allow color image evaluation, we couple our L_1 model with an L_1-trained variant of our chroma reconstruction model as well. We consider our L_1-minimized model and our main model as two different configurations, denoting them by "Ours, L_1" and "Ours, GAN", respectively. Since the available pretrained AGARNet model only handles single channel images (only the Y channel), for evaluating color images we augment their reconstructed Y channel with the C_b and C_r channels decoded by the standard JPEG pipeline.

Note that our main (GAN) model takes an additional input signal z, facilitating user exploration. However, evaluating post-exploration outputs is an ill-defined task that mostly depends on the user's intentions. We circumvent this problem by drawing 50 random z signals per image, then averaging PSNR, NIQE and LPIPS scores over the resulting 50 outputs per image, over the entire evaluation set.

Quantitative evaluations of both metrics on color and grayscale images, on both datasets, are presented in Figs. 11 and 12 respectively. We present scores corresponding to the standard JPEG decompression in all our comparisons, and for the no-reference NIQE scores present also the values corresponding to the GT images. The results on color images (Fig. 11) indicate that our distortion minimizing model (blue), trained to minimize distortion, compares favorably both with DnCNN (brown) and with AGARNet (orange) in terms of reconstruction error (bottom row), on both datasets, especially on severely compressed images (low QFs). When evaluating on grayscale images (Fig. 12), this model is on par with the competition. As can be expected, PSNR scores of our main model ("Ours, GAN"), trained for perceptual quality, are significantly lower, even slightly below JPEG image scores.

As for perceptual quality, NIQE scores (where lower is better) on the top rows suggest that our GAN-trained model (pink) performs well across all evaluated QFs and both datasets, obtaining similar scores to those of the GT images. Evaluating using the full-reference LPIPS score (middle rows) also indicates high perceptual quality. As expected, both competing methods and our distortion minimizing model perform significantly worse, as they were all trained to minimize distortion.

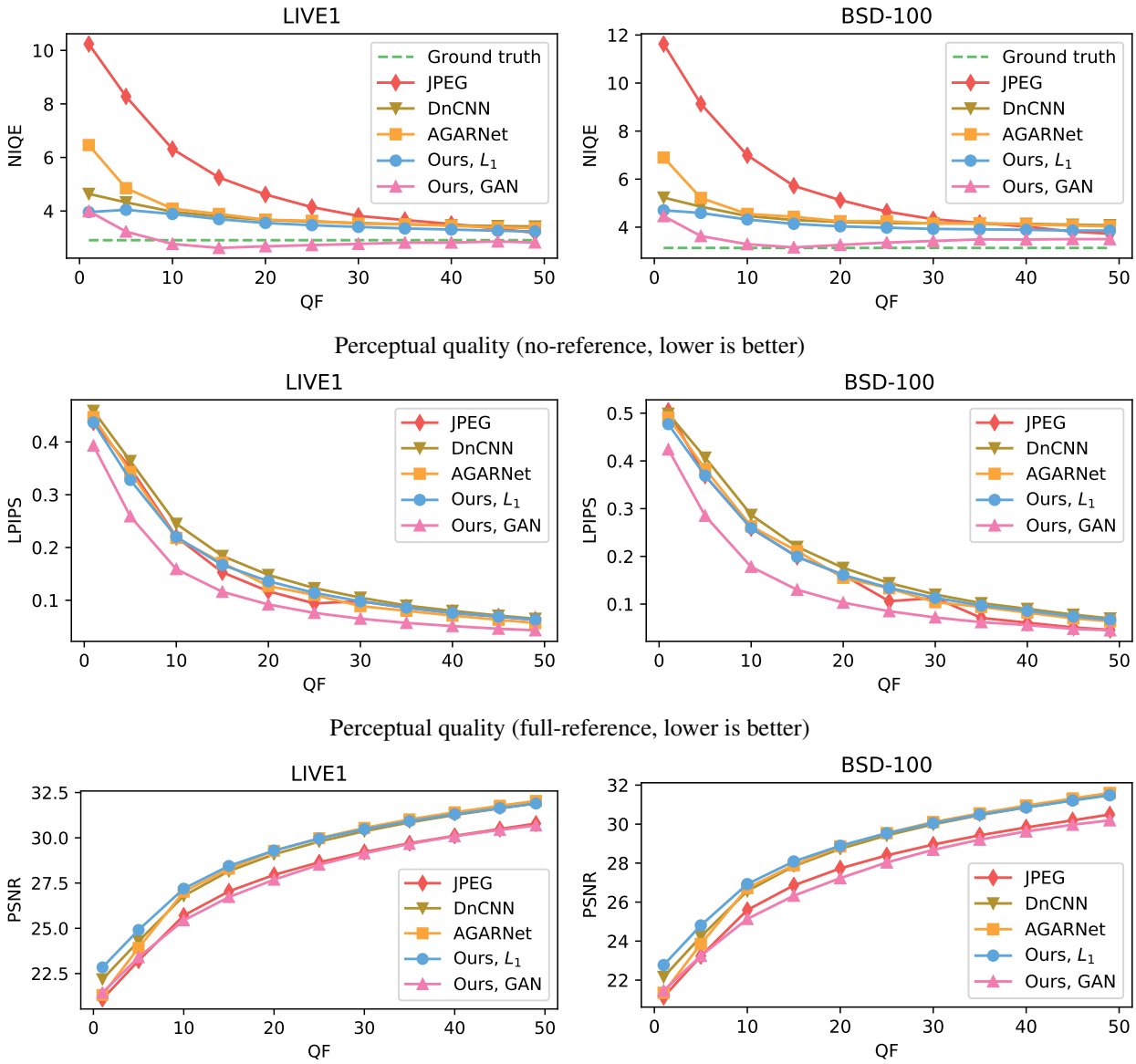

Figure 11. **Quantitative performance evaluation - color images.** Comparing performance on the LIVE1 [40] (left column) and BSD-100 [36] (right column) datasets, in terms of no-reference perceptual quality (top row) and full reference perceptual quality (middle row) and image distortion (bottom row), using the NIQE (↓), LPIPS (↓) and PSNR (↑) metrics, respectively. Please see details in Sec. A.

Finally, the advantage of our method in terms of perceptual quality is evident in a qualitative (visual) comparison, as presented in Fig. 9 for the case of severely compressed images (QF=5). This advantage in perceptual quality persist across different compression levels, as demonstrated using a more moderate compression level (QF=10) in Fig. 13. We do not present the corresponding uncompressed GT images in these figures, since the GT images are as valid a decoding as any other output of our network, due to its inherent consistency with the input code. For the curious readers however, we present these corresponding GT images in Fig. 14.

B. Exploration Tools

Our framework's GUI (depicted in Fig. 15) comprises many editing and exploration tools that facilitate intricate editing operations. As we explain in Sec. 5, these tools work by triggering an optimization process over the space of control signals z, optimizing one of several possible objective functions $f(\cdot)$. This is analogous to traversing the manifold

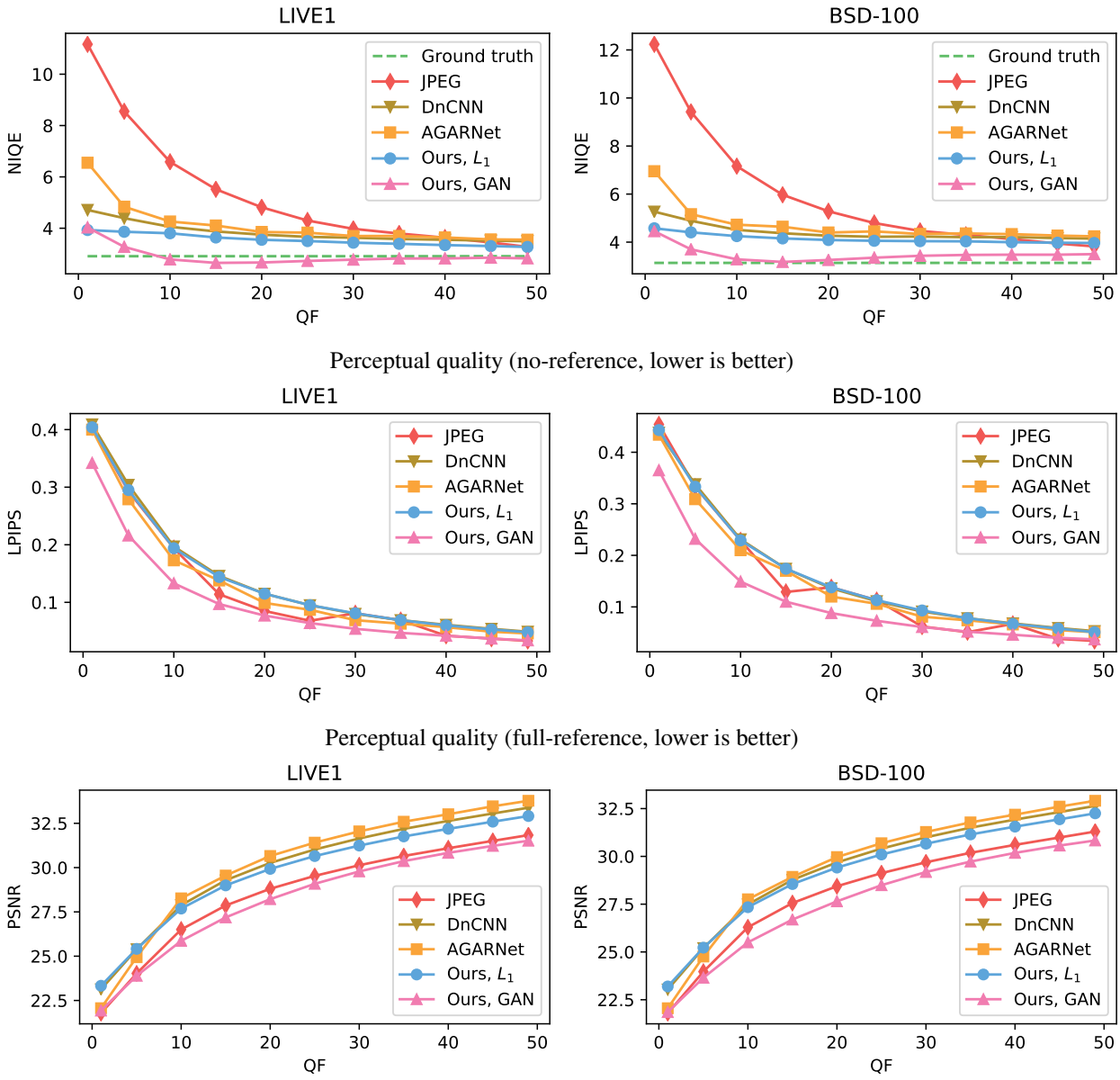

Figure 12. **Quantitative performance evaluation - grayscale images.** Comparing performance on the LIVE1 [40] (left column) and BSD-100 [36] (right column) datasets, in terms of no-reference perceptual quality (top row) and full reference perceptual quality (middle row) and image distortion (bottom row), using the NIQE (↓), LPIPS (↓) and PSNR (↑) metrics, respectively. Please see details in Sec. A.

of perceptually plausible images learned by our network, while always remaining consistent with the compressed image code. We introduce a novel automatic exploration tool, allowing users to explore solutions of interest at a press of a button, as well as some JPEG-specific tools. In addition, we incorporate most editing tools introduced for explorable super resolution [7], by adapting them to the JPEG decompression case.

Editing can be applied to the entire image, or to a specific region marked by the user. Some tools enable more precise editing, by employing Microsoft-Paint-like buttons, including pen and straight line (with adjustable line width), as well as polygon, square and circle drawing tools.

We denote an output image prior to minimizing each objective f by $\hat{x}_0 = \psi(x_Q, z_0)$, where signal z_0 is either a neutral (pre-editing) control signal or the result of a prior editing process. Note that any function $f(\cdot)$ computed on the entire image can alternatively be computed on a specific

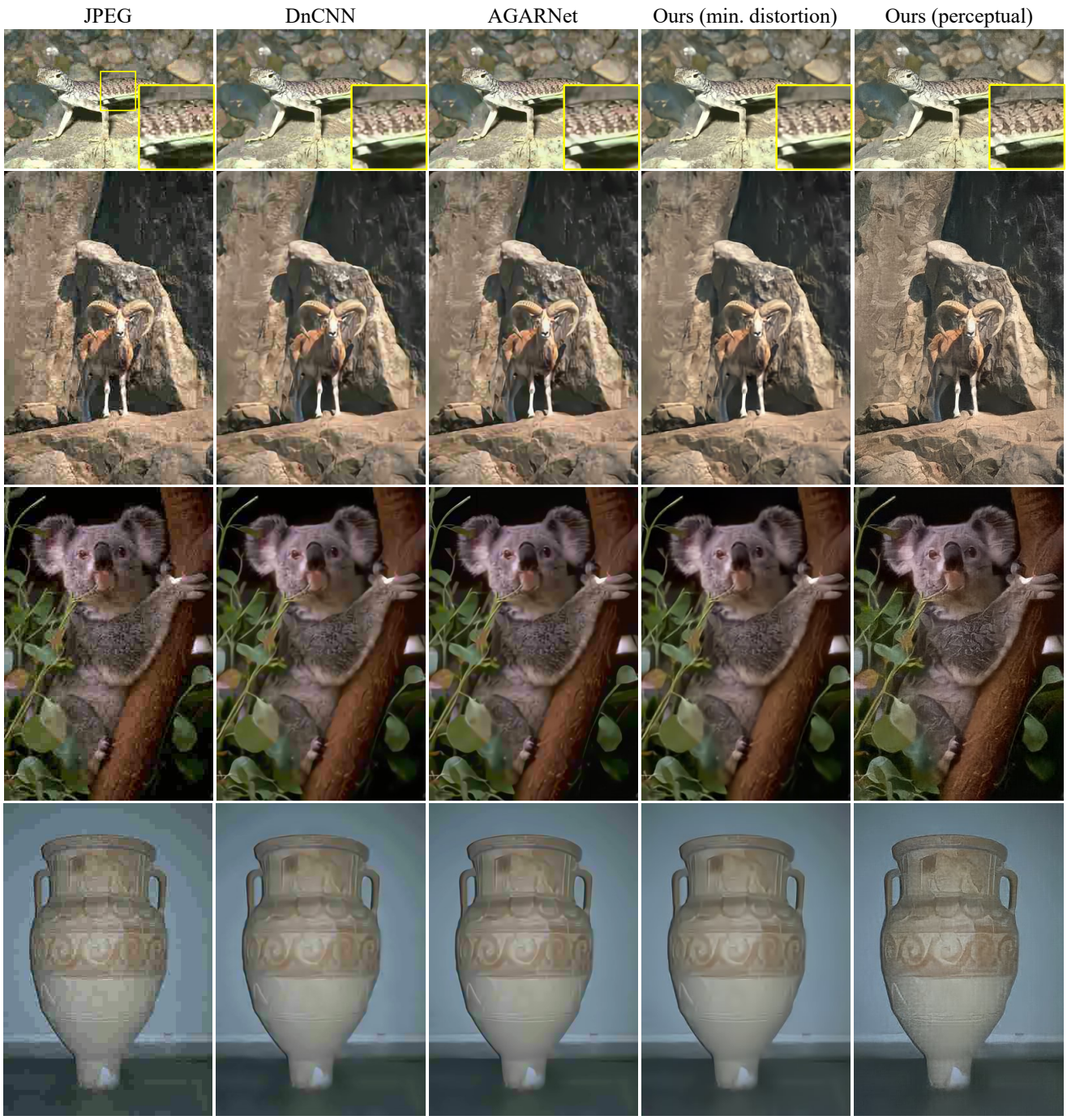

Figure 13. **Qualitative comparison using QF=10.** From left to right: Outputs by the JPEG decompression pipeline, DnCNN [19] AR method, AGARNet [21] AR method, a variant of our model trained to minimize distortion and our GAN-trained model. Similar to the results presented in Fig. 9 for the harsher compression level (QF=5), our GAN-trained model (right) produces sharper and more photo-realistic outputs for the QF=10 case as well, supporting quantitative findings in Figs. 11 and 12, which indicate a significant perceptual quality advantage across a range of compression levels. Images taken from the BSD-100 [36] test set.

region thereof, by masking out the rest of the image. We use $P(\cdot)$ to denote a patch extraction operator[1], for those objective functions below that expect this kind of input. We

[1]Objective functions operating on image patches (rather than directly on the image itself) use partially overlapping 6×6 patches. The degree of overlap varies, and indicated separately for each tool.

Figure 14. **Ground truth uncompressed images corresponding to images in Figs. 9 and 13.**

next describe the different available objective functions and the way they are utilized in our GUI.

B.1. Variance Manipulation

This is a set of tools which operates by manipulating the local variance of all partially overlapping image patches in the selected region. We employ cost function $f(\hat{x}) = \sum(\mathrm{Var}(P(\hat{x})) - \mathrm{Var}(P(\hat{x}_0)) \pm \delta)^2$, where the sum runs over all overlapping patches, and optimize over z to modify (increase or decrease) the local, per-patch variance by a desired value δ. The effect of applying this tool is demonstrated in Fig. 16(a) between the left and middle images.

B.2. Encouraging Piece-wise Smoothness

This tool acts by minimizing the total variations (TV) in an image or a region: $f(\hat{x}) = \mathrm{TV}(\hat{x})$. In particular, we minimize the sum of absolute differences between each pixel in the image and its 8 neighboring pixels. This function can be minimized for a single region, or simultaneously minimized for several marked image areas. The effect of applying this tool is demonstrated in Fig. 16(a) between the middle and right images.

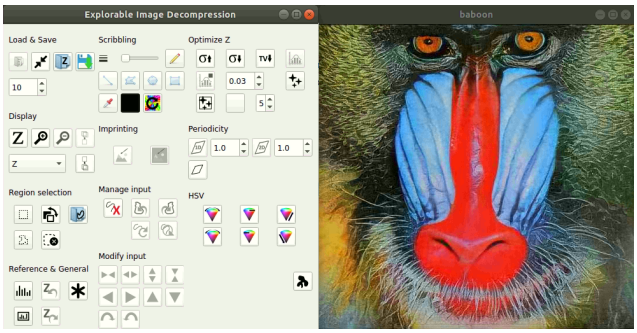

Figure 15. **Our Exploration GUI.** Performing explorable image decompression of a compressed Mandrill image, using our graphical user interface.

B.3. Imposing Graphical User Input

Our GUI comprises a large set of tools to allow imposing a graphical user input on the output image, by minimizing $f(\hat{x}) = \|\hat{x} - x^{\text{scribbled}}\|_1$. The desired graphical content $x^{\text{scribbled}}$ is imposed in a consistency preserving manner, by projecting it onto the set of images that are consistent with the compressed code x_Q. Namely, each block of DCT coefficients $X_D^{\text{scribbled}}$ of the desired input is modified by applying Eq. (2), repeated here for fluency:

$$X_D^{\text{scribbled}} \leftarrow \left(\text{clip}_{[-\frac{1}{2},\frac{1}{2}]}\left(X_D^{\text{scribbled}} \oslash M - X_Q\right) + X_Q\right) \odot M. \tag{2}$$

The modified $X^{\text{scribbled}}$ (depicted, *e.g.*, in the left of the middle pair of images in Fig. 3) is already consistent with the compressed input code x_Q. The user then has the option of translating, resizing or rotating the inserted content using arrow buttons, while consistency is re-enforced automatically after each of these operations. Editing is completed when the user initiates the optimization process, traversing the z space looking for the image $\hat{x}$ that is closest to the desired consistent content $X^{\text{scribbled}}$, while lying on the learned manifold of perceptually plausible images.

The desired input $x^{\text{scribbled}}$ can originate from any of the following sources:

1. *User scribble:* A user can use the Microsoft-Paint-like drawing tools, where scribbling color can be chosen manually or sampled from any given image (including the edited one). Please see Fig. 17 for an example usage of this tool.

2. *Manipulating HSV:* Local hue, saturation and relative brightness (value) of $\hat{x}$ can be manipulated by using one of 6 designated buttons. This results in a desired appearance $x^{\text{scribbled}}$, whose consistency is continuously enforced after each button press, by computing (2). Brightness manipulation was already facilitated in [7] for small image details, but larger regions could not be manipulated, as their HSV attributes

are strictly determined by the low-resolution input. In contrast, JPEG compression often discards information corresponding to these attributes, thus allowing and necessitating their exploration.

3. *Imprinting:* A user can import graphical content, either from within the edited image or from an external one, and then enforce it on $\hat{x}$. The user first selects the desired content to import, and then marks the target region's bounding rectangle on $\hat{x}$. JPEG compression operates on 8×8 pixel blocks, making it highly sensitive to small image shifts. Therefore, to adapt this tool from [7], we propose an option to automatically find a sub-block shifting of the imported content, that yields the most consistent imprinting. Please see Fig. 3 for an example usage of this tool.

Subtle region shifting A variant of the imprinting tool allows applying subtle local affine transformations. It works by imprinting the region of interest onto itself, then allowing a user to utilize the shifting, resizing and rotating buttons to modify the selected region from its original appearance, before triggering the final z optimization process.

B.4. Desired Dictionary of Patches

This tool manipulates target patches in a desired region to resemble the patches comprising a desired source region, either taken from an external image or from a different location in the edited image. The latter case is demonstrated in Fig. 16(b): patches from a small yellow region in the middle image are propagated to the larger yellow region, resulting in the right image.

The corresponding cost function penalizes for the distance between each patch in the target region and its nearest neighbor in the source region. To allow encouraging desired textures across regions with different colors, we first remove mean patch values from each patch, in both source and target patches. To reduce computational load, we discard some of the overlapping patches, by using 2 and 4 rows strides in the source and target regions, respectively. This tool was used for creating the result in Fig. 8 (right image), by propagating patches depicting sand-waves from the center of the image to its upper-left regions.

Ignoring patches' variance A variant of this tool (demonstrated in left image in Fig. 16(b)) allows encouraging desired textures without changing current local variance. To this end, we normalize patches' variance, in addition to removing their mean. Then while optimizing over z, we add an additional penalty that preserves the original variance of each target patch, while encouraging its (normalized) signal to resemble that of its closest (normalized) source patch.

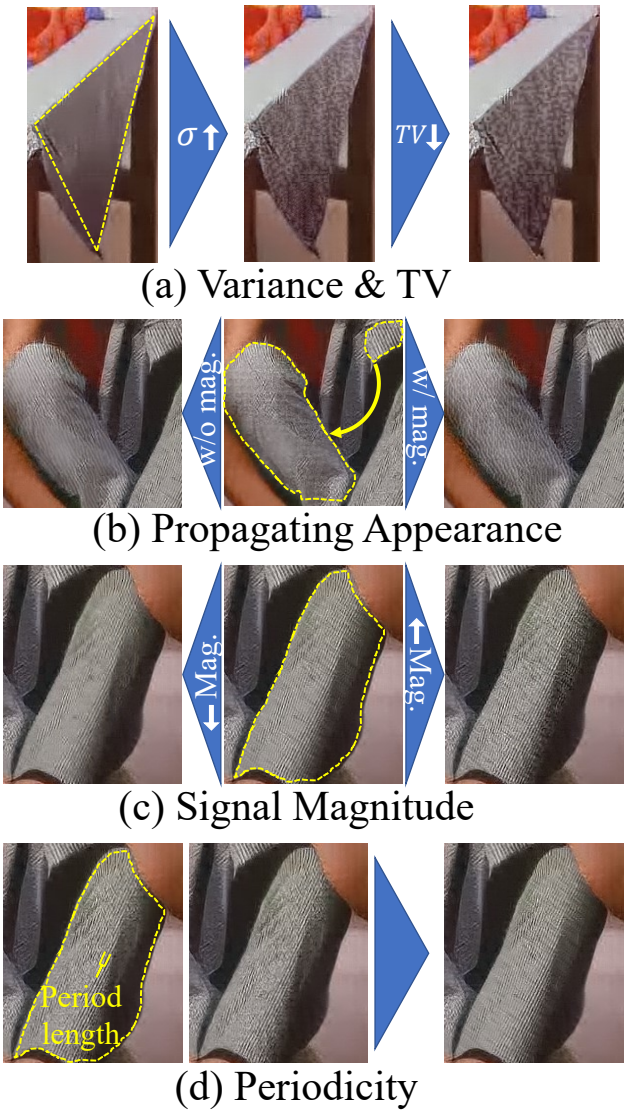

Figure 16. **Exploration tool examples.** (a) A pre-edited region (left) is manipulated to have higher local variance (middle). Then TV minimization is applied for a smoother result (right). (b) Source and target regions are marked by the user (middle). Appearance of patches is then propagated from source to target, while keeping (right) or ignoring (left) source patches signal magnitude. (c) A desired region (middle) is manipulated by increasing (right) or decreasing (left) local signal magnitude. (d) Desired region and period length are marked by user (left). The effect of periodicity encouraging tool is visible between middle and right images.

B.5. Signal magnitude manipulation

An additional tool operating on image patches attempts to amplify or attenuate the magnitude of the signal in existing patches, while preserving existing patch structures. The effect of this tool is demonstrated in Fig. 16(c), both for increasing (right) and for decreasing (left) the signal magnitude within the region marked on the middle image. Similar

to the variance manipulation tool described in Sec. B.1, we use $f(\hat{x}) = \sum \|\tilde{P}(\hat{x}) - (1 \pm \delta)\tilde{P}(\hat{x}_0)\|^2$ as our cost function, where the sum runs over all overlapping patches. It penalizes for the difference between the newly constructed image patches and the $(1 \pm \delta)$ times magnified/attenuated versions of the corresponding existing patches, where operator $\tilde{P}(\cdot)$ extracts image patches and subtracts their respective mean values. This tool was also utilized for creating the result in Fig. 8 (right image), by enhancing the sand-wave appearance of patches propagated to the upper left image regions.

B.6. Encouraging Periodicity

This tool encourages the periodic nature of an image region, across one or two directions determined by a user. The desired period length (in pixels) for each direction can be manually set by the user, or it can be automatically set to the most prominent period length, by calculating local image self-correlation. Periodicity is then encouraged by penalizing for the difference between the image region and its version translated by a single period length, for each desired direction. We used this tool too when creating Fig. 8 (right image), for encouraging the sand-waves to have an approximately constant period length (in the appropriate direction), thus yielding a more realistic appearance. The effect of this tool is also demonstrated in Fig. 16(d), where a desired period length and direction are marked by a user on the left image (yellow curly bracket), as well a region to be manipulated (yellow dashed line). The result after Optimizing over z (right) is a sharper and cleaner appearance of the stripes on the manipulated image region.

B.7. Random Diverse Alternatives

This tool allows exploring the image manifold in a random manner, producing N different outputs by maximizing the L_1 distance between them in pixel space. These images (or sub-regions thereof) can then serve as a baseline for further editing and exploration.

Constraining distance to current image A variant of this tool adds the requirement that all N images should be close to the current $\hat{x}_0$ (in terms of L_1 distance in pixel space).

B.8. Automatic Exploration of Classes

This novel tool allows exploring a predefined set of solutions of interest to a user-marked region at a press of a button. We exemplify it using the case of the 10 possible numerical digits $d \in \{0-9\}$. To this end, we use the d^{th} output of a pre-trained digit classifier as our objective function, $f(\cdot) = \text{Classifier}_d(\cdot)$, and produce 10 different outputs corresponding to the 10 digits, by repeatedly maximizing f over z using a different $d \in \{0-9\}$ each time. The obtained digit-wise optimized outputs are then presented to the user, who can, *e.g.*, examine their plausibility to assess the likelihood of the underlying visual content corresponding to each of these digits, as we show in Fig. 2 for the hour digit on the captured smartphone's screen. This tool can accommodate any type of classifier, and can therefore be very useful in forensic and medical applications (*e.g.* for predicting malignant/benign tumors).

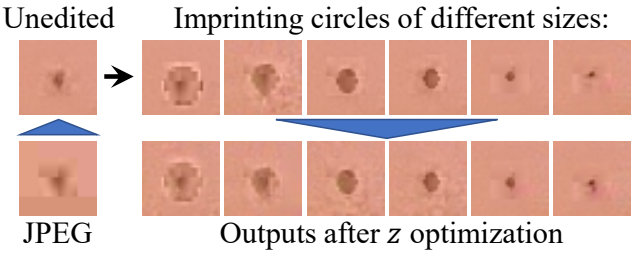

Figure 17. **The making of Fig. 4.** Here, we imprint brown disks of varying radii on the region of the mole. The top row shows the projection of the naively imprinted image onto the set of consistent solutions. The bottom row shows the final output of our method, after determining the control signal z that causes the net's output to resemble the most to the images in the top row.

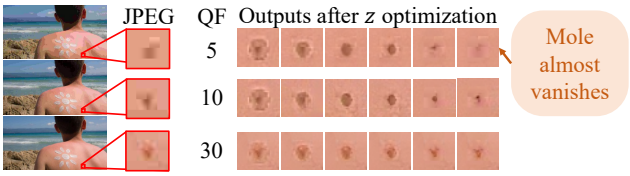

Figure 18. **Effect of QF on exploration space.** The ability to imprint brown disks of different radii on the mole, strongly depends on the QF. When the QF is small (agressive compression), the space of consistent solutions is large. In this case, we can imprint a large mole in one extreme, or completely remove the mole in the other extreme. However, as the QF increases, the space of consistent solutions becomes smaller, and the range of mole sizes that can be imprinted reduces accordingly.

C. Editing Processes and Additional Examples

We next exemplify the exploration and editing process, and illustrate the effect of the quality factor (QF) on the available degrees of freedom.

Figure 17 shows the exploration process of Fig. 4. Here, we attempted to imprint brown disks of varying radii on the unedited image. The top row shows the results of the first stage of the imprinting process, which projects the image with the naively placed brown disk onto the set of images that are consistent with the JPEG code. The second row shows the results of the second stage of the imprinting process, which seeks a control signal z that causes the output of our decoding network to resemble the image produced in the first stage. In this example, the second stage is mostly responsible for smoothing some of the artifacts generated in

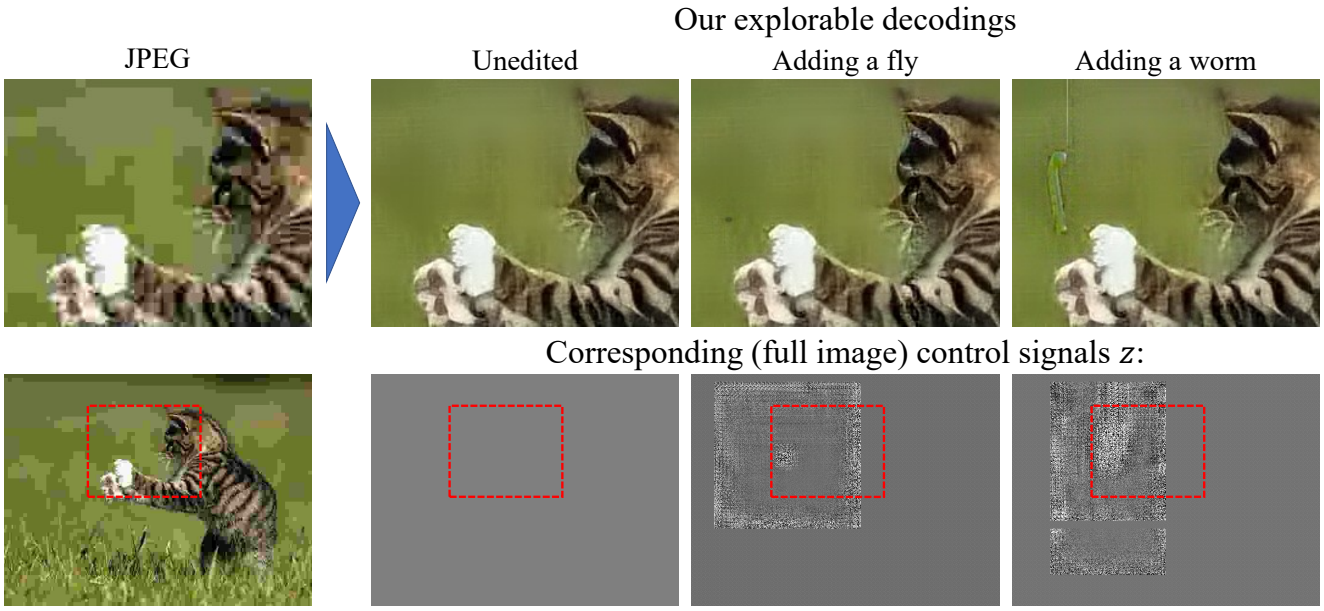

Figure 19. **Possible explanations & corresponding z signals.** Top row: zoom-in on the JPEG pipeline decoding (left) and three plausible (and perfectly consistent) alternative decodings by our method (repeated here for fluency from Fig. 7). Bottom row: z signals corresponding to our alternative decodings (entire image, zoomed region marked in red), displayed by reshaping each $1\times1\times64$ column in the $60\times80\times64$ z signal to an 8×8 block in a 480×640 grayscale image (values were translated from $[-1,1]$ to $[0,1]$ for display purposes). While our pre-edited output (middle-left) corresponds to a constant z, to obtain the alternative appearances (middle-right and right), our imprinting tool's optimization process exploits the model's receptive field and modifies large regions of z (limited to a rectangular window around the modified location), keeping non-modified parts of the image unchanged.

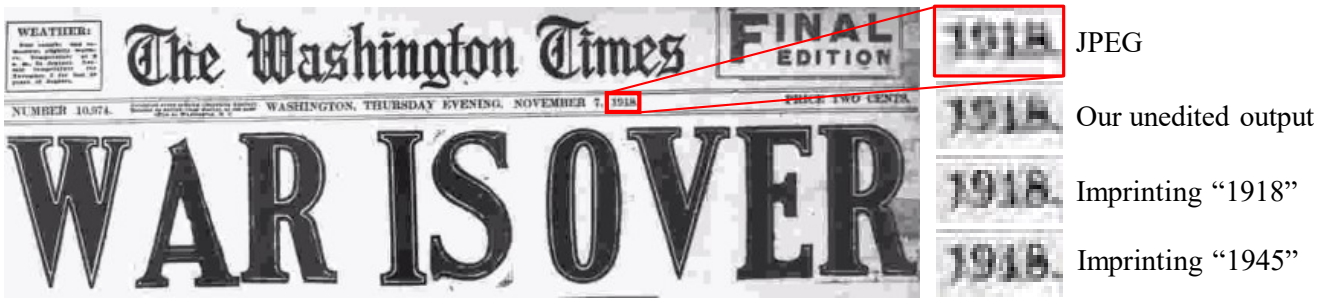

Figure 20. **Which war is over?** Using our framework to attempt imprinting years "1918" vs. "1945" yields a significantly better result for the former, suggesting this compressed archived newspaper dates back to the end of world war I.

the first stage.

Figure 18 illustrates the effect of the QF on the space of images that are consistent with a given JPEG code. As can be seen, when using extreme compression with a QF of 5, we can produce consistent reconstructions with a wide range of mole sizes, from a very large mole on the left, to an almost vanished mole on the right. However, as the QF increases, the set of consistent solutions becomes smaller, making it impossible to imprint very large or very small disks.

Control signal z has a non-local effect, due to the receptive field of our model. This is shown in Fig. 19, depicting different signals z (bottom row) corresponding to different consistent decodings (top row) of a single compressed image. Finally, we present an additional exploration example in Fig. 20, demonstrating a case of exploring corrupted text.

D. Validating our Alternative Modeling of Chroma Subsampling

In an effort to produce higher quality reconstruction of the chroma information, we wish to concatenate the reconstructed luminance information $\hat{x}^Y$ to the input of our chroma reconstruction model. However, this requires handling the dimensions inconsistency between the full-

resolution luminance channel and the subsampled chroma channels, which we do through introducing an alternative modeling of the JPEG chroma subsampling process, as we explain in Sec. 4.1.

To validate this alternative modeling, we looked at the differences (calculated after going back to the pixel domain) between images undergoing the following original vs. alternative subsampling processes:

1. *"4:2:0" JPEG pipeline:* Subsampling chroma channels by a factor of 2 in both axes $\rightarrow$ Computing DCT coefficients for each 8×8 pixels block $\rightarrow$ Right and bottom zero-padding each coefficients block to 16×16 $\rightarrow$ Returning to pixel domain by computing inverse DCT for each 16×16 block.

2. *Our alternative pipeline:* Computing DCT coefficients for each 16×16 pixels block $\rightarrow$ Setting each block's 3 lower-right quadrants to 0, leaving unchanged the 8×8 upper left quadrant coefficients that correspond to low-frequency content $\rightarrow$ Returning to pixel domain by computing inverse DCT for each 16×16 block.

Note that we did not perform any quantization step in either of the alternatives, as we were only interested in the isolated effect of remodeling the subsampling pipeline.

We computed the differences between the two alternatives using the RGB color representation, after concatenating back the non-altered luminance channel (Y) in both alternatives. We experimented using 100 images from the BSD-100 dataset [36], and found that the average root mean square error (RMSE) was a negligible 0.009137 gray levels (corresponding to a PSNR of 88.9dB). This certifies our decision to use the alternative modeling, which allows us to make our chroma reconstruction network aware of the corresponding luminance channel, by concatenating it to the network's input.

E. Full Training Details

We train our model on 1.15M images from the ImageNet training set [34], using batches of 16 images each. We use an Adam optimizer, with learning rates of 0.0001 and 0.00001 for the initialization and consecutive training phases, respectively, and set $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for both generator and critic networks. After ~6 initialization phase epochs, we set λ_{Range} and λ_{Map} from Eq. (1) to 200 and 0.1 respectively, and train for additional ~12 epochs, performing 10 critic iterations for every generator iteration.

For using $\mathcal{L}_{\text{Map}}$ in the latter phase, we feed each batch of images twice, differing by the input control signals $z \in [-1, 1]^{m \times n \times 64}$. We first feed a per-channel constant z, uniformly sampled from $[1, -1]$, using only $\mathcal{L}_{\text{Range}}$ and $\mathcal{L}_{\text{Adv}}$ in (1). We then perform 10 minimization iterations over $\mathcal{L}_{\text{Map}} = \min_z \|\psi(x_Q, z) - x\|_1$, and feed the same image batch with the resulting z, this time using the entire cost function in (1).

To create compressed image input codes, we compress the GT training images utilizing a quantization interval matrix $M = \text{QF} \cdot Q_{\text{baseline}}/5000$, where QF is independently sampled from a uniform distribution over $[5, 49]$ for each image[2], and Q_{baseline} is the example baseline table in the JPEG standard [35]. We use $N_\ell = 10$ layers for both generator and critic models, where convolution operations utilize 3×3 spatial kernels with 320 or 160 output channels for all layers but the last, in the luminance or chroma networks, respectively. We employ a conditional critic, which means we concatenate the generator's input x_Q to our critic's input, as we find it accelerates training convergence.

[2]We omit the upper QF range of $[50, 100]$ when demonstrating our approach, as these higher QF values induce lower data loss, leaving less room for exploration.