Learned Neural Iterative Decoding for Lossy Image Compression Systems

Alexander G. Ororbia*, Ankur Mali†, Jian Wu‡,
Scott O'Connell†, William Dreese†, David Miller†, C. Lee Giles†

$\star$ Rochester Institute of Technology, Rochester, NY, 14623, USA

$\dagger$ Pennsylvania State University, University Park, PA, 16802, USA

$\ddagger$ Old Dominion University, Norfolk, VA, 23529, USA

Abstract

For lossy image compression systems, we develop an algorithm, *iterative refinement*, to improve the decoder's reconstruction compared to standard decoding techniques. Specifically, we propose a recurrent neural network approach for nonlinear, iterative decoding. Our decoder, which works with any encoder, employs self-connected memory units that make use of causal and non-causal spatial context information to progressively reduce reconstruction error over a fixed number of steps. We experiment with variants of our estimator and find that iterative refinement consistently creates lower distortion images of higher perceptual quality compared to other approaches. Specifically, on the Kodak Lossless True Color Image Suite, we observe as much as a 0.871 decibel (dB) gain over JPEG, a 1.095 dB gain over JPEG 2000, and a 0.971 dB gain over a competitive neural model.

Introduction

Image compression is a problem that has been at the core of signal processing research for decades. Recent successes in the application of deep neural networks (DNNs) to problems in speech processing, computer vision, and natural language processing have sparked the development of neural-based approaches to this challenging problem. However, most efforts strive to design end-to-end neural-based systems, which require designing and training effective encoding and decoding functions as well as quantizers. This poses a particular challenge for back-propagation-based learning since the quantizer is discrete and thus not differentiable. In order to learn encoders, most work generally attempts to formulate relaxations or differentiable ("soft") approximations to quantization [1]. Furthermore, most image compression systems decode spatial blocks separately, without considering the spatial dependencies with surrounding blocks. Our proposed algorithm takes a novel, neural network decoding approach that exploits spatially non-causal statistical dependencies, with the potential for achieving improved decoding accuracy given an encoded bit stream.

Instead of building an end-to-end system, we seek to answer the question: given any encoder and quantization scheme what would an optimal non-linear decoder look like? Focusing on the decoder of the system allows us to side-step issues like approximating quantization and to take advantage of well-developed encoders and quantization operations (i.e., as in JPEG/JPEG 2000) while still giving us the opportunity to reduce distortion through improved decoder optimization. To this end, we propose a novel *non-linear* estimator as an image decoder. While our approach is general and can handle any encoder, here we focus on JPEG-encoded bit-streams. While the local memory of recurrent neural networks (RNNs) is typically used to capture short-term causal context, we re-purpose this memory to gradually improve its reconstruction of image patches, an approach we call *iterative refinement*.

1 Related Work

Traditional lossy compression techniques for still images exploit the fact that most of the image energy is concentrated in low spatial frequencies, with sparse content found in high frequencies. Thus, strategies that combine transform coding with optimized bit allocation are used to give good compression performance at acceptable computational and memory complexity as compared to high-dimensional vector quantization. JPEG employs an 8×8 block discrete cosine transform (DCT), followed by run-length coding that exploits the sparsity pattern of the resultant frequency coefficients [2]. JPEG 2000 (JP2), which breaks the image into tiles of any size (such as 64×64 sized tiles as we do in this paper), leverages a multi-scale discrete wavelet transformation (DWT) and applies a uniform quantizer followed by an adaptive binary arithmetic coder and bit-stream organization [3].

Although widely used, DCT and DWT are "universal", i.e., they do not exploit the particular pixel distribution of the input images. Statistical learning techniques, in contrast, have greater power to represent non-linear feature combinations and to automatically capture latent statistical structure. For instance, using a sample of the Outdoor MIT Places dataset, JP2 and JPEG achieve 30 and 29 times compression, respectively, but a neural-based technique achieves better peak signal-to-noise ratio (PSNR) at just a quarter of the bitrate of JPEG and JP2 with comparable visual quality [4]. Van den Oord et al. proposed a neural network that sequentially predicts image pixels along two spatial dimensions [5].

In [6], a representation learning framework using a variational autoencoder for image compression was developed. Toderici et al. (2015) proposed a framework for variable-rate image compression and an architecture based on convolutional and deconvolutional LSTM models [7]. Later on, Toderici et al. (2016) proposed several architectures consisting of an RNN-based encoder and decoder, a binarizer, and an ANN for encoding binary representations. This was claimed to be the first DNN architecture that outperformed JPEG in image compression across most bitrates on the Kodak image dataset [4]. Other methods have been proposed using autoencoders, generative adversarial networks, etc. [8, 9].

Most prior work designs encoders and thus quantization schemes, e.g., [7]. Our method is different from these in two ways: 1) we propose an iterative, RNN-based estimator for decoding instead of using transformations, 2) our algorithm introduces a way to effectively exploit both causal and non-causal information to improve low bitrate reconstruction. Our model applies to any image decoding problem and can handle a wide range of bitrate values.

2 A Nonlinear Estimator for Iterative Decoding

2.1 The Iterative Refinement Procedure

In order to achieve locally optimal decoding of data points, we propose an iterative, nonlinear process called *iterative refinement*. This process treats the act of reconstructing images from a compressed representation as a multi-step reconstruction problem, which will require a model to improve its recreation of some target sample over a bounded number of passes, K.

With respect to the data, let us decompose a single image $\mathbf{I}$ into a set of P pixel patches, or $\mathbf{I} = \{\mathbf{p}^1, \cdots, \mathbf{p}^j, \cdots, \mathbf{p}^P\}$ (non-overlapping for JPEG, overlapping for JP2). Assuming column-major orientation, each input patch is of dimension $\mathbf{p}^j \in \mathcal{R}^{d^2 \times 1}$ (a flattened vector of the original $d \times d$ pixel grid). Each patch $\mathbf{p}^j$ would have a corresponding latent representation, or rather, a quantized symbol representation $\mathbf{q}^j$ (as given by the decoding of the bit-stream

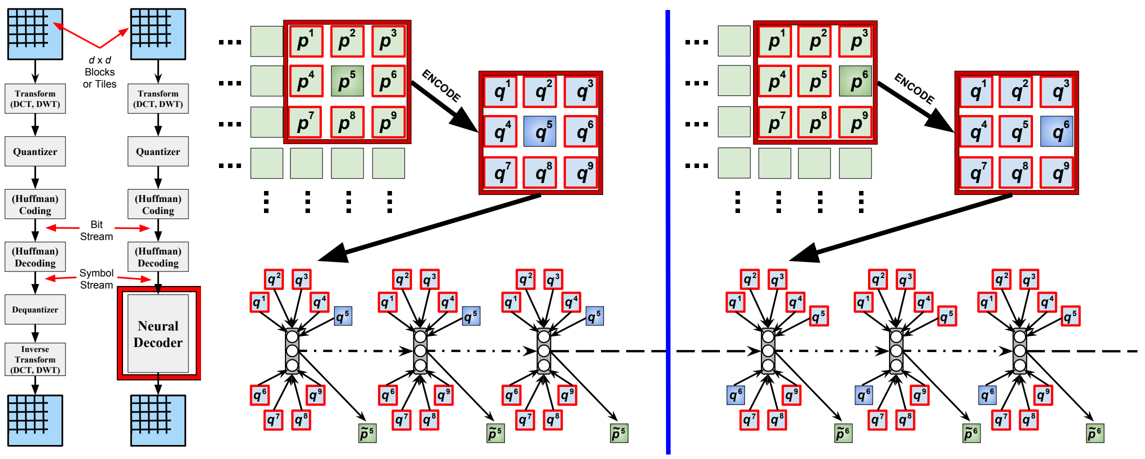

Figure 1: Iterative refinement (2 episodes, $K = 3$) using an Elman RNN estimator (top-level view). Dot-dashed arrows indicate the latent state across an episode, since back-propagation through time is only applied within an episode. The dashed arrow indicates the memory created at an episode's end, carried over to the next one. This visual also shows how a patch on the borderline is handled.

input to the overall decoder), of dimension $\mathbf{q}^j \in \mathcal{R}^{d^2 \times 1}$. We could furthermore treat these variables $\mathbf{p}^j$ and $\mathbf{q}^j$ as mini-batches of B samples (meaning we are operating on B images in parallel), where a particular instance/sample is indexed by the integer b, i.e., $\mathbf{p}^{j,b}$, meaning the dimensions of each would then be $\mathbf{p}^j \in \mathcal{R}^{d^2 \times B}$ and $\mathbf{q}^j \in \mathcal{R}^{d^2 \times B}$, respectively.

Our nonlinear estimator for a locally optimal decoder is defined by parameters $\Theta = \{\Theta_s, \Theta_t, \Theta_d\}$ and takes in N neighboring patches/blocks as input. The overall estimator's general form requires three key functions:

- $\mathbf{e} = e(\mathbf{q}^1, \cdots, \mathbf{q}^N; \Theta)$, a transformation function of a set of encoded neighboring patches (in the form of quantized symbol sequence vectors) for a current target patch that yields a vector summary of the spatial context.
- $\mathbf{s}_k = s(\mathbf{e}, \mathbf{s}_{k-1}; \Theta)$, a state function that combines a filtration with the vector summary of input spatial context.
- $\widetilde{\mathbf{p}}^j_k = d(\mathbf{s}_k; \Theta)$, a reconstruction function that predicts a given target (pixel) patch at step k (within a reconstruction episode).

These ingredients highlight that our estimator requires a way to summarize (encoded) spatial context and state information as well as a transformation of the current state to pixel space.

Using the general specification of the nonlinear decoder above, we can describe the full algorithm for iterative refinement in reconstructing images that are decomposed into multiple, spatially-related patches. First, the encoder, e.g., JPEG, JP2, is used to obtain an *initial* compressed representation of each patch within the original image, i.e., the bit-stream, followed by decoding, e.g., arithmetic or Huffman, of the bit-stream to a quantized symbol sequence. Then, following some defined scanning pattern, each patch of the image will serve as a target for reconstruction for the decoder, of which a set of neighboring quantized symbol blocks will be fed into the neural estimator. The model then works to best estimate each original image patch (which represents a block for JPEG or a tile for JP2). We define the computation for each image patch as a *reconstruction episode* (Figure 1). However, when moving on to a new target, the decoder state carries over, i.e., the model is conditioned on a vector summary of its past reconstruction episodes.

Algorithm 1 The Iterative Refinement algorithm.

```
Input: Image $\mathbf{I} = (\mathbf{p}^1, \cdots, \mathbf{p}^j, \cdots, \mathbf{p}^P)$, current decoder parameters $\Theta_t$, and # of steps $K > 0$. Note: $\mathbf{Q} = (\mathbf{q}^1, \cdots, \mathbf{q}^j, \cdots, \mathbf{q}^P)$, the set of quantized symbol blocks.
function RECONSTRUCT($\mathbf{I}$, $\Theta_t, K$)
    $\mathbf{Q} \leftarrow$ GETQUANTREP($\mathbf{I}$)    ▷ Get quantized symbol representation via JPEG/JP2
    $\mathbf{s}_0 = 0$    ▷ Initialize state
    for $\mathbf{q}^j \in \mathbf{Q}$ do    ▷ Blocks extracted depend on scan-line path
        $(\mathbf{q}^1, \cdots, \mathbf{q}^N) \leftarrow$ GETNEIGHBORS($\mathbf{q}^j, \mathbf{Q}$)    ▷ Get neighboring quantized patches of $\mathbf{q}^j$
        // Begin reconstruction episode
        $\mathbf{e} = e(\mathbf{q}^1, \cdots, \mathbf{q}^N; \Theta)$    ▷ Compute activation for spatial context
        for $k = 1$ to $K$ do    ▷ Conduct $K$ steps of refinement
            $\mathbf{s}_k = s(\mathbf{e}, \mathbf{s}_{k-1}; \Theta)$, $\widetilde{\mathbf{p}}^j_k = d(\mathbf{s}_k; \Theta)$
        $\mathbf{s}_0 = \mathbf{s}_k$, $\widetilde{\mathbf{p}}^j = \widetilde{\mathbf{p}}^j_K$    ▷ Store final decoder state & reconstruction
    return $(\widetilde{\mathbf{p}}^1, \cdots, \widetilde{\mathbf{p}}^j, \cdots, \widetilde{\mathbf{p}}^P)$    ▷ this output is also denoted as $\widetilde{\mathbf{I}}$
```

It is this carry-over of state across episodes that we conjecture gives a fundamental advantage to an RNN-based estimator over an MLP-based one. When decoding a particular patch, an RNN can exploit global state information, from across the whole image (not just a particular patch's neighborhood), through the iterative propagation of state information from all patches to the patches in their neighborhoods. A stateless MLP, in contrast, can only make use of the information contained in a given patch's surrounding neighborhood. Consider the situation where the model is to decode a particular patch $\mathbf{p}^A$ (having just finished decoding $\mathbf{p}^B$, and before that, patches $\mathbf{p}^D$ and $\mathbf{p}^E$), where $\mathbf{p}^B$ and $\mathbf{p}^C$ are in $\mathbf{p}^A$'s neighborhood but $\mathbf{p}^D$ and $\mathbf{p}^E$ are only in $\mathbf{p}^B$'s neighborhood. The MLP can only use (direct) information from $\mathbf{p}^B$ and $\mathbf{p}^C$ when decoding $\mathbf{p}^A$, treating this reconstruction episode independently of the ones it finished before. The RNN, however, through the global connectivity provided through its neighborhood/state function, can also make use of information from $\mathbf{p}^D$ and $\mathbf{p}^E$, since their information was used to update the model's state for $\mathbf{p}^B$ which is used to decode $\mathbf{p}^A$. Note that one could instead modify the stateless MLP to also work with global context, but this would be highly impractical as the model's input dimensionality would explode given that we would have to learn a static mapping for each patch position (since the number of patches to the left, right, top, and bottom for a given a patch is position-dependent). The RNN is a practical way to efficiently exploit global context information as opposed to its array of patch position-dependent MLPs equivalent. To justify the use of K steps of recurrent processing within a decoding episode, we can view the K steps as implicitly forming a K-layer deep MLP, containing K output predictors, with parameters aggressively shared across its layers (this is a similar idea to the equivalence shown between a shallow RNN and a deep residual network [10]). This equivalence allows us to think of the multiple iterations as simply one compact, nonlinear model (conducting a form of posterior sharpening) that can be learned without the extra memory cost that comes with adding extra layers/parameters.

At a high-level, our iterative refinement procedure, depicted in Algorithm 1, takes in as input an image I (or mini-batch of images), current model parameters Θ_t, and a predefined number of steps K. First, it creates a quantized symbol representation of the input patches using the subroutine GETQUANTREP($\cdot$), which amounts to using the encoder of a given compression algorithm, i.e., this contains a transform, a quantizer, and a coding procedure such as turbo coding [11], and its corresponding bit-stream decoder. For each target patch $\mathbf{p}^j$, the subroutine $\textsc{getNeighbors}(\cdot)$ is called to extract its local context of N contiguous image patches, or rather, their quantized representations, $(\mathbf{q}^1, \cdots, \mathbf{q}^N)$.[1] Note that (if $\mathbf{I} = \mathbf{Q}$, i.e., starting from a compressed image) one could modify the line with $\textsc{getQuantRep}(\cdot)$ to instead just call the relevant procedure for bit-stream decoding.

2.1.1 Transformation & Reconstruction Function Forms

Both the transformation function $\mathbf{e} = e(\mathbf{q}^1, \cdots, \mathbf{q}^N; \Theta)$ and the reconstruction function $\mathbf{s}_k = s(\mathbf{e}, \mathbf{s}_{k-1}; \Theta)$ can be parametrized by multilayer perceptrons (MLPs). Specifically, we simply parametrize the transformation and reconstruction functions, using parameters $\Theta_t = \{W_1, \cdots, W_N\}$ and $\Theta_d = \{U, \mathbf{c}\}$, respectively, as follows:

$$\mathbf{e} = \phi_e(W_1\mathbf{q}_1 + \cdots + W_N\mathbf{q}_N), \text{ and, } \widetilde{\mathbf{p}}_k = \phi_d(U\mathbf{s}_k + \mathbf{c})$$

where $\phi_e(v) = v$ and $\phi_d(v) = v$, i.e., identity functions, meaning that nonlinear behavior will come from our design of the state-activation function rather than the input block-to-state or state-to-output mappings. Note that the number of parameters could be significantly cut down by tying the block-to-hidden matrices, i.e., $W = W_1 = W_2 = \cdots = W_N$.

2.1.2 State Function Forms

We experimented with a variety of gated recurrent architectures as unified under the Differential State Framework (DSF) [12]. A DSF neural model is generally defined as a composition of an inner function that computes state proposals and an outer mixing function that decides how much of the state proposal is to be incorporated into a slower moving state, i.e., the longer term memory. The DSF models that we will use include the Long Short Term Memory (LSTM) model [13], the Gated Recurrent Unit (GRU) [14], and the Delta-RNN (Delta-RNN or Δ-RNN). We will compare these RNN-based models to a static mapping function learned by a stateless MLP (very much in the spirit of classical predictive coding [15]).

We describe the Δ-RNN form, which is the simplest, most parameter-efficient model we experimented with, since models like the LSTM and GRU can be derived from the same framework as that of the Δ-RNN [12] and integrated into iterative refinement in a similar fashion. The Δ-RNN state function (parameters $\Theta_s = \{V, \mathbf{b}, \mathbf{b}_r, \alpha, \beta_1, \beta_2\}$) is defined as:

$$\mathbf{d}^1_k = \alpha \otimes V\mathbf{s}_{k-1} \otimes \mathbf{e}, \quad \mathbf{d}^2_k = \beta_1 \otimes V\mathbf{s}_{k-1} + \beta_2 \otimes \mathbf{e} \tag{1}$$

$$\widetilde{\mathbf{s}}_k = \phi_s(\mathbf{d}^1_k + \mathbf{d}^2_k + \mathbf{b}) \tag{2}$$

$$\mathbf{s}_k = \Phi((1 - \mathbf{r}) \otimes \widetilde{\mathbf{s}}_k + \mathbf{r} \otimes \mathbf{s}_{k-1}), \text{ where, } \mathbf{r} = \sigma(\mathbf{e} + \mathbf{b}_r), \tag{3}$$

where $\Phi(v) = \phi_s(v) = tanh(v) = (e^{(2v)} - 1)/(e^{(2v)} + 1)$ and $\otimes$ denotes the Hadamard product.

2.2 Learning the Neural Iterative Decoder

To calculate parameter updates for a particular K-step reconstruction episode, one explicitly unrolls the estimator over the length of the episode to create a mini-batch of length K arrays of B matrices (or set of 3D tensors) in order to invoke back-propagation to learn

[1]This also returns the quantized symbol representation $\mathbf{q}^j$ of $\mathbf{p}^j$, also fed into our nonlinear estimator.

the estimator's parameters, Θ. Our objective will be to optimize distortion D, since we are designing a general estimator only for the act of decoding. Note that our estimator must learn to deal with a variable bit-rate, as dictated by training samples. The mean bit-rate of our training dataset (described in detail later) was $R_\mu = 0.525$ with variance $R_{\sigma^2} = 0.671$.

When training decoders using only mean squared error (MSE), we often found that insufficient gains were made with respect to PSNR over JPEG. Motivated to overcome initially disappointing results, we looked to designing a cost function that might help improve our decoders' performance.[2] At training time, we optimize decoder parameters with respect to a multi-objective loss over K-step reconstruction episodes for mini-batches of B target image patches $\mathbf{p}^j$ (channel input) operating over a set of decoder reconstructions $\widehat{\mathbf{p}}^j = \{\widetilde{\mathbf{p}}^j_k, \cdots, \widetilde{\mathbf{p}}^j_K\}$ (channel outputs). We defined the multi-objective loss over a single episode to be a convex combination of MSE and a form of mean absolute error (MAE) as follows:

$$\mathcal{D}(\mathbf{p}^j, \widehat{\mathbf{p}}^j) = (1-\alpha)\mathcal{D}_{MAE}(\mathbf{p}^j, \widehat{\mathbf{p}}^j) + \alpha \mathcal{D}_{MSE}(\mathbf{p}^j, \widehat{\mathbf{p}}^j). \tag{4}$$

α is a tunable coefficient that controls the trade-off between the two distortion terms. In preliminary experiments, we found that $\alpha = 0.235$ provided a good trade-off between the two (using validation set MSE as a guide). The individual terms of the cost are explicitly:

$$\mathcal{D}_{MAE}(\mathbf{p}^j, \widehat{\mathbf{p}}^j) = \frac{1}{(2K)} \sum_{k=1}^{K} \sum_{b=1}^{B} \sum_i |(\widehat{\mathbf{p}}^{j,b}_k[i] - \mathbf{p}^{j,b}[i])| \tag{5}$$

$$\mathcal{D}_{MSE}(\mathbf{p}^j, \widehat{\mathbf{p}}^j) = \frac{1}{(2BK)} \sum_{k=1}^{K} \sum_{b=1}^{B} \sum_i (\widetilde{\mathbf{p}}^{j,b}_k[i] - \mathbf{p}^{j,b}[i])^2. \tag{6}$$

where i indexes a single dimension of a given vector. We speculate that the MAE-based term helps to train a better decoder by reducing outlier errors. MAE is connected to the least absolute deviations statistical optimality criterion, which, in contrast to MSE (least squares), is robust/resistant to outliers in data since it gives equal emphasis to all observed patterns (whereas in least squares, the squaring operator gives more weight to large residual errors). Since PSNR is a function of MSE (and, as is known in the regression literature, using MAE alone can lead to unstable solutions), we felt it unwise to eschew MSE completely and thus combined it with MAE to create a hybrid function.

Learning decoder parameters can be done under an empirical risk minimization framework where we calculate derivatives of our cost with respect to Θ using back-propagation through time (BPTT) [16]. Given that the decoder's state function is recursively defined, we unroll the underlying computation graph over a length K reconstruction episode. With the exception of the very first target, we initialize the initial decoder hidden state with its state at the end of the previous episode. The cost of calculating the gradients may be further reduced by pre-computing the linear combination of projected input blocks, i.e., the transformation function $\mathbf{e} = e(\mathbf{q}^1, \cdots, \mathbf{q}^N; \Theta)$, and reuse $\mathbf{e}$ at each step of the unrolled graph. The BPTT-computed gradients are then used to update parameters using the method of steepest descent. Note that, since $\mathbf{p}^j$ could be a mini-batch, decoding (Algorithm 1) and parameter updating can be carried over B images so long as all images are of the same dimensions.[3]

[2]At test time, performance will still be evaluated by strictly measuring MSE & PSNR.

[3]The patch-by-patch pathway taken across each image within a mini-batch does not have to be the same. One could, in a batch of say 2 images, start from the bottom-left in the 1st and the top-right in the 2nd.

3 Experiments

We implement variations of the estimator state-function described above, i.e., a GRU, an LSTM, and a Δ-RNN. We compare these models to the JPEG and JPEG 2000 (JP2) baselines as well as an MLP stateless decoder. Furthermore, we ran the competitive neural architecture proposed in [4] (GOOG) on our test-sets. All of our decoders/estimators (including the MLP) were all trained using the same cost function (Equation 4). The only pre-processing we applied to patches was normalizing the pixel values to the range $[0, 1]$ (for evaluation we convert decoder outputs back to $[0, 255]$ before comparing to original patches).

3.1 Data & Benchmarks

To create the training set for learning our nonlinear decoders, we randomly choose high resolution 128k images from the *Places365* [17] dataset, which were down-sampled to 512×512 pixel sizes. In addition, we randomly sampled 7168 raw images with variable bit rates from the RAISE-ALL[18] dataset, which were down-sampled to a 1600×1600 size. Each image was first compressed with variable bit rates between 0.35–1.02 bits per pixel (bpp) for a particular encoder (once for JPEG and once for JP2). Similarly, to create a validation sample, we randomly selected 20K images from the *Places365* development set combined with the remaining 1K RAISE-ALL images. Validation samples were also compressed using bitrates between 0.35–1.02 bpp. For simplicity, we focus this study on single channel images and convert each image to gray-scale. However, though we focus on gray-scale, our proposed iterative refinement can be used with other formats, e.g., RGB. We divide images into sets of 8×8 patches/blocks for JPEG (yielding 4096 patches for 512×512 images and 40000 for 1600×1600 images) and 64×64 patches/tiles for JP2 (producing 64 patches for 512×512 images and 625 patches for 1600×1600 images).

We experimented with 6 different test sets: 1) the Kodak Lossless True Color Image Suite[4] (Kodak) with 24 true color 24-bit uncompressed images, 2) the image compression benchmark (CB 8-Bit[5]) with 14 high-resolution 8-bit grayscale uncompressed images down-sampled to 1200×1200 images, 3) the image compression benchmark (CB 16-Bit) with 16-bit uncompressed images also downsampled to 1200×1200, 4) the image compression benchmark 16-bit-linear (CB 16-Bit-Linear) containing 9 high-quality 16-bit uncompressed images downsampled to 1200×1200, 5) Tecnick [19] (36 8-bit images), and 6) the Wikipedia test-set created by crawling 100 high-resolution 1200×1200 images from the Wikipedia website.

3.2 Experimental Setup

All RNN estimators contained one layer of 512 hidden units (initial state was null). Parameters were randomly initialized from a uniform distribution, $\sim U(-0.054, 0.054)$ (biases initialized to zero). Gradients were estimated over mini-batches of 256 samples, each of which is 24×24, or 9 parallel episodes over 9 images, and parameters were updated using RMSprop [20] with gradient norms clipped to 7. All models were trained over 200 epochs. We experimented with two learning rate (η) schedules. The first was simple: start $\eta = 0.002$ and after every 50 epochs, η is decreased by 3 orders of magnitude. The second one was a novel scheme we call the *annealed stochastic learning rate*. In this schedule, with $\eta_0 = 0.002$,

[4]`http://r0k.us/graphics/kodak/`

[5]`http://imagecompression.info/`

Table 1: PSNR of the Δ-RNN-JPEG on the Kodak dataset (bitrate 0.37 bpp) as a function of K.

	$K=1$	$K=3$	$K=5$	$K=7$	$K=9$	$K=11$
PSNR	27.0087	27.3976	27.6619	27.8954	28.2189	28.5093

Table 2: Out-of-sample results for the Kodak (bpp 0.37), the 8-bit Compression Benchmark (CB, bpp, 0.341), the 16-bit and 16-bit-Linear Compression Benchmark (CB) datasets (bpp 0.35 for both), the Tecnick (bpp 0.475), and Wikipedia (bpp 0.352) datasets.

	Kodak				**CB 8-Bit**			
Model	**PSNR**	**MSE**	**SSIM**	MS^3IM	**PSNR**	**MSE**	**SSIM**	MS^3IM
JPEG	27.6540	111.604	0.7733	0.9291	27.5481	114.3583	0.8330	0.9383
JPEG 2000	27.8370	106.9986	0.8396	0.9440	27.7965	108.0011	0.8362	0.9471
GOOG-JPEG	27.9613	103.9802	0.8017	0.9557	27.8458	106.7805	0.8396	0.9562
MLP-JPEG	27.8325	107.1089	0.8399	0.9444	27.8089	107.6923	0.8371	0.9475
Δ-*RNN*-JPEG	28.5093	101.9919	0.8411	0.9487	28.0461	101.9689	0.8403	0.9535
GRU-JPEG	28.5081	102.0017	0.8400	0.9474	28.0446	102.0041	0.8379	0.9533
LSTM-JPEG	28.5247	101.9918	0.8409	0.9486	28.0461	101.9686	0.8371	0.9532
LSTM-JP2	**28.9321**	**98.9686**	**0.8425**	**0.9596**	**28.0896**	**100.9521**	**0.8389**	**0.9562**
	CB 16-Bit				**CB 16-Bit-Linear**			
JPEG	27.5368	114.6580	0.8331	0.9383	31.7522	43.4366	0.8355	0.9455
JPEG 2000	27.7885	108.2001	0.8391	0.9437	32.0270	40.7729	0.8357	0.9471
GOOG	27.8830	105.8712	0.8391	0.9468	32.1275	39.8412	0.8369	0.9533
MLP-JPEG	27.7762	108.5056	0.8390	0.9438	32.0269	40.7746	0.8356	0.9454
Δ-*RNN*-JPEG	28.0093	102.8369	0.8399	0.9471	32.4038	37.3847	0.8403	0.9535
GRU-JPEG	28.0081	102.8649	0.8392	0.9469	32.4038	37.3844	0.8379	0.9533
LSTM-JPEG	28.0247	102.4710	0.8310	0.9471	32.4032	37.3908	0.8371	0.9532
LSTM-JP2	**28.1307**	**100.0021**	**0.8425**	**0.9496**	**32.4998**	**36.5676**	**0.8382**	**0.9541**
	Tecnick				**Wikipedia**			
JPEG	30.7377	54.8663	0.8682	0.9521	28.7724	86.2655	0.8290	0.9435
JPEG 2000	31.2319	48.9659	0.8747	0.9569	29.1545	79.0002	0.8382	0.9495
GOOG	31.5030	46.0021	0.8814	0.9608	29.2209	77.108	0.8406	0.9520
MLP-JPEG	31.2287	49.0012	0.8746	0.9571	29.1547	78.9968	0.8383	0.9497
Δ-*RNN*-JPEG	31.5411	45.6001	0.8821	0.9609	29.2772	76.8000	0.8403	0.9519
LSTM-JPEG	31.5616	45.3857	0.8820	0.9609	29.2771	76.8008	0.8403	0.9519
LSTM-JP2	**31.6962**	**44.0012**	**0.8834**	**0.9619**	**29.3228**	**75.9969**	**0.8411**	**0.9526**

at the end of each epoch, we stochastically corrupt the step-size: $\eta_t = \eta_{t-1} + \mathcal{N}(0, \gamma^t)$, where t marks the current epoch and $\gamma = 0.000001025$ while the η_t is further annealed by a factor of 0.01 every 50 epochs. We found that the second schedule helped speed up convergence. Furthermore, we developed a unique two-step data shuffling technique to prevent formation of any spurious correlations between images and patches. In the first step, we randomly shuffle images before the start of an epoch and during the second step (within an epoch), we randomly shuffle the scan pathway starting points. Specifically, per image, we randomly start at the top-left, bottom-left, top-right, or the bottom-right image corner.[6]

3.3 Results

We evaluate our model on 6 datasets: Kodak True color images, the Image Compression Benchmark (8bit, 16bit, and 16bitlinear), Tecnick and Wikipedia. Every model was evaluated using three metrics, as advocated by [21]. These included PSNR, structural similarity

[6]If a scan starts on the image's left side, we proceed horizontally then vertically. If starting on the right side, we proceed left horizontally then vertically.

(SSIM), and multi-scale structural similarity (MS-SSIM [22], or MS^3IM as shown in our tables). In addition, we report mean squared error (MSE), though PSNR is a function of it.

In Table 2, iterative refinement consistently yields lower distortion as compared to JPEG, JP2, and GOOG. In terms of PSNR (on Kodak) we achieve nearly a 0.9 decibel (dB) gain (with *LSTM*-JPEG) over JPEG and a 1.0951 dB gain (with *LSTM*-JP2) over JP2. With respect to GOOG, our *LSTM*-JP2 estimator yields a gain of 0.9708 dB. Furthermore, note that our nonlinear decoders are vastly simpler and faster than the complex end-to-end GOOG model, which faces the additional difficulty of learning an encoder and quantization scheme from scratch. The results, across all independent benchmarks/test-sets, for all metrics (PSNR, SSIM, and MS-SSIM), show that decoders learned with our proposed iterative refinement procedure generate images with lower distortion and higher perceptual quality (as indicated by SSIM and MS-SSIM). Moreover, the recurrent estimators outperform the MLP stateless estimator, indicating that a decoder benefits from exploiting both causal and non-causal information when attempting to reconstruct image patches.

In Table 1, for our best-performing Δ-RNN, we investigate how PSNR varies as a function of K, the number of steps taken in an episode. We see that raising K improves image reconstruction with respect to dataset PSNR. We also sampled four random images and plotted their PSNR as a function of K.[7] In general, increasing K improves PSNR, but in some cases, we see a diminishing returns effect and even a peak. This might indicate that integrating a decoding-time early-stopping criterion might yield even better overall PSNR.

4 Conclusions

In this paper, we proposed *iterative refinement*, an algorithm for improving decoder reconstruction for lossy compression systems. The procedure realizes a nonlinear estimator of an iterative decoder via a recurrent neural network that exploits both causal and non-causal statistical dependencies in images. We compared our approach to standard JPEG, JPEG-2000, and a state-of-the-art, end-to-end neural architecture and found that our algorithm performed the best with respect to reconstruction error (at low bit rates). Note that our decoder approach is general, which means any encoder can be used, including that of [4].

References

[1] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc Van Gool, “Soft-to-hard vector quantization for end-to-end learned compression of images and neural networks,” *CoRR*, vol. abs/1704.00648, 2017.

[2] S. Takamura and M. Takagi, “Lossless image compression with lossy image using adaptive prediction and arithmetic coding,” in *Proceedings of IEEE Data Compression Conference (DCC’94)*, March 1994, pp. 166–174.

[3] Majid Rabbani and Rajan Joshi, “An overview of the jpeg 2000 still image compression standard,” *Signal Processing: Image Communication*, vol. 17, no. 1, pp. 3 – 48, 2002, JPEG 2000.

[4] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell, “Full resolution image compression with recurrent neural networks,” *CoRR*, vol. abs/1608.05148, 2016.

[5] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu, “Pixel recurrent neural

[7] Available at `https://1drv.ms/b/s!AiNRGVbyH-JUjjVxd7R5Vmy6T2NA`

networks," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016, pp. 1747–1756.

[6] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra, "Towards conceptual compression," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., pp. 3549–3557. Curran Associates, Inc., 2016.

[7] George Toderici, Sean M. O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar, "Variable rate image compression with recurrent neural networks," *CoRR*, vol. abs/1511.06085, 2015.

[8] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár, "Lossy image compression with compressive autoencoders," *CoRR*, vol. abs/1703.00395, 2017.

[9] Oren Rippel and Lubomir D. Bourdev, "Real-time adaptive image compression," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 2922–2930.

[10] Qianli Liao and Tomaso A. Poggio, "Bridging the gaps between residual learning, recurrent neural networks and visual cortex," *CoRR*, vol. abs/1604.03640, 2016.

[11] Patrick Mitran and Jan Bajcsy, "Turbo source coding: A noise-robust approach to data compression," in *Data Compression Conference, 2002. Proceedings. DCC 2002.* IEEE, 2002, p. 465.

[12] Alexander G. Ororbia, II, Tomas Mikolov, and David Reitter, "Learning simpler language models with the differential state framework," *Neural Comput.*, vol. 29, no. 12, pp. 3327–3352, Dec. 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[15] M.W. Spratling, "A review of predictive coding algorithms," *Brain and Cognition*, vol. 112, no. Supplement C, pp. 92 – 97, 2017, Perspectives on Human Probabilistic Inferences and the 'Bayesian Brain'.

[16] Paul J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339 – 356, 1988.

[17] Bolei Zhou, Àgata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba, "Places: A 10 million image database for scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, pp. 1452–1464, 2018.

[18] Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato, "Raise: A raw images dataset for digital image forensics," in *Proceedings of the 6th ACM Multimedia Systems Conference*, New York, NY, USA, 2015, MMSys '15, pp. 219–224, ACM.

[19] Nicola Asuni and Andrea Giachetti, "Testimages: A large data archive for display and algorithm testing," *Journal of Graphics Tools*, vol. 17, no. 4, pp. 113–125, 2013.

[20] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[21] Kede Ma, Qingbo Wu, Zhou Wang, Zhengfang Duanmu, Hongwei Yong, Hongliang Li, and Lei Zhang, "Group MAD competition? A new methodology to compare objective image quality models," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 1664–1673.

[22] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.