Learning Resolution-independent Image Representations

Jon C. Hammer and Michael S. Gashler
Department of Computer Science and Computer Engineering
University of Arkansas
Fayetteville, AR, USA
{*jhammer, mgashler*}*@uark.edu*

***Abstract*—Humans are well-known to be highly effective at comprehending continuous patterns within digital images. We present a collection of methods that enable analogous capabilities in deep neural networks. These methods train neural networks to represent images with continuous resolution-independent representations. They utilize an MCMC algorithm that directs attention during the learning phase to regions of the image that deviate from the current model. An encoding hypernetwork learns to generalize from a collection of images, such that it can effectively compute resolution-independent representations in constant time. These methods have immediate applications in super-resolution scaling of images, image compression, and secure image processing, and additionally suggest improved capabilities for image processing with neural networks in several future applications.**

***Keywords*-Neural Networks; Super Resolution; Hypernetworks; MCMC**

I. Introduction

The pixel values in a digital image may be viewed as samples drawn from a function of the form $f(x) = i$, where $x \in \mathbb{R}^2$ represents the image coordinates, $i \in \mathbb{R}^c$ is the corresponding intensity for that coordinate, and c is the number of channels of the image. Rather than measuring f directly, it is common for a digital camera to sample it at uniform intervals. The resulting discrete sequence $\{i_{0,0}, i_{0,1}, \dots \ i_{H-1,W-1}\}$ of intensity values constitutes a representation of the underlying continuous image from which it is sampled.

Although generally useful for computer graphics, this discrete representation does have several important drawbacks. For example, applying image transformations, such as scaling and rotation, often requires resampling. In practice, the ability to do so directly is often limited (e.g. the digital camera does not have the ability, the subject has moved, etc.), so new intensity values must be synthesized using the pixels that have already been captured, adding additional noise. The discrete representation also has a space requirement proportional to the number of samples taken rather than to the actual entropy of the image. A large image with little entropy will require more space than a small image with large entropy.

A continuous representation of the image would not suffer from these drawbacks, but since f is not typically available, a learned approximation, $\hat{f}$, can be used instead. Such an approximation can be learned by examining the pixels of the discrete image representation as long as the underlying model has sufficient representational capacity.

A popular choice of learning model is the neural network, especially a multilayer perceptron (MLP). Assuming several base conditions have been met, MLPs have been shown to be universal function approximators [1]. As such, they possess the capability to accurately model f (or any other function, for that matter) under some setting of the weights and biases θ. We define $\hat{f}(\theta, x) = i$ to be a MLP that maps normalized image coordinate pairs x to a vector of intensity values i, where $i \in \mathbb{R}^c$. When fully trained, $\hat{f}(\theta, x) \approx f(x)\ \forall x \in \mathbb{R}^2$. In practice, we limit ourselves to $x \in [0, 1]^2$ to avoid dependence on the exact dimensions of the image. Given θ, the intensity for any individual coordinate can be approximated by performing a feed-forward pass through $\hat{f}$. Similarly, an entire image can be constructed by feeding multiple coordinates in as a batch.

Importantly, as long as θ is available, $\hat{f}$ can be used to generate discrete representations of f with arbitrary pixel sampling frequencies. In other words, images can be generated at any resolution. As a result, θ itself constitutes an alternative representation of the original image f, one that is invariant with respect to the sampling resolution.

The parameters θ of $\hat{f}$ can be learned for any given image using traditional gradient descent-based techniques. A set of N $\langle x_i, f(x_i)\rangle$ tuples can be obtained directly from the pixels of the available image, where N is the number of pixels. This information can be used as training data for an ordinary regression model.

Interestingly, it is also possible to train another model, an **encoder**, to predict a reasonable image encoding θ for a single image. The encoder is given as input all available intensity values from an original image and outputs the set of parameters to be used by $\hat{f}$, making it a type of hypernetwork [2]. The encoder is trained on a collection of images with a reconstructive error metric utilizing $\hat{f}$. The encoder learns to recognize features that are common to multiple images and associates them with the parameters of $\hat{f}$ necessary to properly reconstruct the image that was provided as input. A well-trained encoder should even be able to produce reasonable encodings of images it has never seen by generalizing from the images it was trained on.

Proc. 2018 IEEE 17th Int'l Conf. on Cognitive Informatics & Cognitive Computing (ICCI*CC'18)
Y. Wang, S. Kwong, J. Feldman, N. Howard, P. Sheu & B. Widrow (Eds.)
978-1-5386-3360-1/18/$31.00 ©2018 IEEE

With this work, our primary contributions are as follows:

- We show that θ can be learned using the discrete representation of an arbitrary image using traditional gradient-descent techniques.
- We introduce a Markov-chain Monte Carlo (MCMC)-based technique that yields improved reconstruction accuracy and lowers training time compared to standard batch processing of pixels.
- We present a deep convolutional encoder that is capable of generating reasonable θ values for unknown images using a single forward pass through the network. Our encoder generates the weights of $\hat{f}$ directly, rather than requiring extensive training for each image.
- We discuss several practical applications of our work, including image resizing, compression, and security.

This paper is outlined as follows: In section II, we analyze related works. In section III, we detail the process by which θ can be learned iteratively for a single image. In section IV, we show how a generalized encoder can be trained to output useful image representations directly. In section V, we discuss several applications of a resolution-invariant image representation, including arbitrary scaling, compression, and security. In section VI, we evaluate our claims using single images and groups of related images. Section VII concludes our work.

II. Related Works

Neural networks have been used for several state-of-the-art applications in image processing, including recognition [3], [4], [5], completion [6], [7], style transfer [8], [9], and generation, especially using Variational Autoencoders [10] and Generative Adversarial Networks [11], [12]. Many of these approaches create approximate image representations implicitly in order to accomplish their stated goals. In our work, we focus entirely on the problem of generating reasonable continuous representations of images.

Several other works have put more focus on learning image representations, including PixelRNN [13], which uses a recurrent network to generate pixels one a time, and Ashmore et. al [14], who suggested learning image representations in order to capture state from either a single image or a sequence of images. Our approach differs from PixelRNN in that we can generate an entire image in parallel, as our model does not make use of recurrent connections (e.g. LSTM). Ashmore's approach uses a type of autoencoder to learn image state, while our approach more closely resembles a hypernetwork.

A. Hypernetworks

A hypernetwork refers to a neural network capable of generating the weights for another neural network. For example, Ha et al. used hypernetworks to generate adaptive weights for recurrent neural networks [2]. Stanley used an approach called HyperNEAT to make a neural network that was highly amenable for evolving images [15]. In our work, we train a hypernetwork to learn how to approximate resolution-independent representations of images from raw pixel values. More specifically, our hypernetwork learns from many images how to comprehend what is implied by the discrete sampling of pixels in digital images by digital cameras.

B. Super-resolution

Recently, the problem of resizing an arbitrary image, especially to increase the resolution, has been addressed by deep convolutional neural networks [16], [17]. For example, Dong et. al [16] demonstrated that super-resolution can be achieved for a single image by training a deep network to map between low and high resolution versions of that image. One application of our work is super-resolution, as an image can be reconstructed using any arbitrary resolution, including larger sizes. Our work differs in that a higher resolution version of the original image is not required for training. Our work also produces results a single pixel at a time, rather than one image at a time, although an entire image may be produced using batching. This allows the networks to be significantly smaller and faster to evaluate.

Generative adversarial networks have also been applied to this problem. For example, Ledig et. al [18] have demonstrated that GANs can effectively generate the fine image texture details that are often missing from other super-resolution approaches, which indicates that they are capable of accurately approximating the original image. Our work differs in that we can train networks and produce scaled results using images of any arbitrary resolutions. As such, our approach is considerably more flexible.

C. Compression

Although not our primary focus, one tangential application of our research is the ability to compress images with reasonable effectiveness, as the size of the resolution-independent representation depends on the entropy of the image, rather than the discrete sampling rate. Several others have also used neural networks to compress images, including Toderici et. al [19]. These approaches tend to rely on learning some form of mapping between the original image and the compressed version in order to maximize compression ratios. Our approach is conceptually much simpler, as θ directly corresponds to the compressed image representation. Due to the simplicity, however, our method is unlikely to outperform established SOA techniques in this particular field.

III. Learning a Representation for Single Images

A. Formulation

Training an MLP to fit to a single image f can be a relatively straightforward process when the problem has been well-formulated. The goal is to find a setting of the

parameters of the network θ_f that minimizes some reconstruction error E with respect to the pixels of the original image. More formally,

$$\theta_f = \text{argmin}_{\theta_f}(E[\hat{f}(X, \theta_f), f(X)]) \quad (1)$$

In this equation, E refers to some error metric, and X represents a $B \times 2$ matrix, where B is the batch size, containing the normalized image coordinates for each pixel in the batch. Both $\hat{f}(X, \theta_f)$ and $f(X)$ produce a $B \times c$ matrix representing a c channel intensity (e.g. 3 for RGB, 1 for grayscale, etc.) for each sample. Image coordinates $(0, 0)$ and $(1, 1)$ refer to the top-left and bottom-right corners of the image, respectively.

If E is differentiable, equation 1 can be evaluated using traditional gradient descent-based techniques. For each pixel batch, the appropriate image coordinate vector X is calculated and used as the input to the model. The pixel intensity values themselves are used as labels. If an image contains N pixel values, then at most N unique training samples are available for training $\hat{f}$.

B. Error Metrics

While the choice of error metric E is theoretically arbitrary, certain metrics work better than others in practice. For example, in our testing, variants of Sum-squared Error (SSE) seem to be highly susceptible to a local optimum in which the network learns to output a blank image (all intensities are either 0 or 1). Experimentally, we found that using mean Cross-Entropy Error does not have these limitations and indeed does perform well for most images. Therefore, the remainder of this paper assumes that mean Cross-Entropy Error is used for E.

C. Model

The topology of $\hat{f}$ controls the degree to which it is capable of approximating f. If the topology is too restrictive, $\hat{f}$ will not be able to capture the fine details of the original image, resulting in a blurred reconstruction. Conversely, as we will discuss further in sections IV and V, it is desirable to have as small a representation of θ_f as possible. Ideally, the number of weights would be proportional to the entropy of the image in order to guarantee the network has the capacity necessary to fully represent f, but a small fixed topology can be effective in practice. For example, a $2 \rightarrow 100 \rightarrow 50 \rightarrow c$ fully connected topology with tanh or relu nonlinearities is sufficient to represent many low resolution images. The choice of nonlinearity has an effect when training has not yet fully converged or if the topology is too restrictive. Otherwise, such a choice has demonstrated much less significance.

D. Training

$\hat{f}$ can be trained using mini-batch gradient descent, where the batch size varies between 1 and N, where N is the number of pixels in the image. There are several different strategies that could be used to select the pixels to use in each mini-batch: A naïve approach would be to group pixels into blocks of a certain width and height as is often done when compressing images (e.g. JPEG). The pixels within a particular block would always be part of the same batch, and blocks could be chosen for training randomly. This approach tends to compromise between learning high-frequency and low-frequency image components based on the size of the blocks, buts it tends to converge slowly in practice.

A related method is to choose pixels from the image randomly to fill each mini-batch. Random sampling allows the network to efficiently learn the low-frequency components of an image due to the potentially large spacial separation between the pixels in the mini-batch. As a result, this method is likely to converge more quickly than the block-based method at the beginning of training. However, since the separation between pixels is typically much larger than the block-based approach, learning high-frequency image components can be more difficult when using random sampling. This effect tends to slow progress once the low-frequency portions of the image have been learned.

A potential solution to this problem incorporates the structure of the image. At a high level, pixels that can already be reconstructed accurately should not be weighted the same as pixels that are are far from their correct intensities. Training would be more efficient if those pixels with large reconstruction error were more likely to be placed in a batch than those with small reconstruction error, similar to the idea of boosting in ensembles.

Pixel weights can be calculated for all of the pixels of an image by forward-propagating each pixel coordinate and calculating an appropriate reconstruction error (e.g. $|y - \hat{y}|^\gamma$). This particular approach tends to be expensive in practice because calculating all pixel weights requires a forward propagation of the entire image for each training batch.

We propose a significantly more efficient alternative approach based on the Metropolis algorithm [20], a Markov Chain Monte Carlo (MCMC) approach often used to generate random samples from an arbitrary probability distribution. The first batch contains pixels chosen uniformly from the original image, but each batch afterwards is generated using the process outlined in Algorithm 1. For each element in the batch, we generate a candidate pixel in the same neighborhood as the original. We then calculate reconstruction errors for both the original pixel and the candidate. (We use $\gamma = 0.125$ for our experiments.) These are used as estimates of the probability density function required by the Metropolis algorithm. Finally, we apply the Metropolis algorithm directly, choosing probabilistically whether or not

Algorithm 1 Applying the Metropolis-Hastings algorithm to pixel selection

function NEXT-BATCH-METROPOLIS(f, $\hat{f}$, b_{prev}, σ, γ)
  $b \leftarrow b_{prev}$
  for $i \in [0, len(b) - 1]$ **do**
    $c_i \leftarrow b_i + \mathcal{N}(0, \sigma)$
    $p(b_i) \leftarrow |f(b_i) - \hat{f}(b_i)|^\gamma$
    $p(c_i) \leftarrow |f(c_i) - \hat{f}(c_i)|^\gamma$
    if $\mathcal{U}(0,1) < \frac{p(c_i)}{p(b_i)}$ **then**
      $b_i \leftarrow c_i$
    end if
  end for
  return b
end function

to replace the original pixel with the candidate.

This algorithm tends to cause training to focus on high-frequency portions of the image, such as edges. As those pixels are better represented in the training batches, the network is given more motivation to correctly reconstruct detailed portions of the image. As a result, this method can provide sharper reconstructions. Especially when combined with the random-pixel method to learn low-frequency portions of the image, this MCMC-based approach can lead to faster convergence and lower overall reconstructive error compared to either of the other methods alone.

IV. Learning a Generic Encoder

In the previous section, we presented an iterative process for training a network to generate a resolution-independent image representation. This section generalizes upon that capability by demonstrating that we can train an encoding hypernetwork, e, to compute θ_f directly. This encoder offers significantly greater utility, as it computes θ_f in a single forward pass and learns to generalize effectively from multiple images. e maps from the pixels of the original image to a resolution-independent representation, θ_f. With this formulation, $\hat{f}$ no longer needs to be trained iteratively. Instead it is used as part of an objective function to train the parameters of e, θ_e, which now constitutes the complete set of all optimizable variables.

Here, we show that e can be trained by examining numerous images, extracting common features and mapping them to image representations that can then be used by $\hat{f}$ directly, without the need for additional training at inference time. As a result, if e has already been sufficiently trained, θ_f can be calculated for an arbitrary image using a single forward pass through the encoder network. Compared to the approach outlined in the previous section, this approach is asymptotically faster. Evaluating e for a single input is a constant time operation, compared to the linear cost of fully training a network to accomplish the same goal, which is clearly an improvement.

In addition to the methods outlined in the previous section, additional procedures unique to this problem are also needed in order to generate high quality encodings across multiple images. In this section, we examine these additional procedures, as well as the architecture of the complete encoding hypernetwork in detail.

A. Model Architecture

Deep convolutional neural networks have demonstrated that they are capable of extracting high-level features from images, especially for the purposes of image recognition, segmentation, and generation [3], [4], [5], [10], [11], [12]. The same properties are desirable in creating a generic encoder, as we desire to associate image features with appropriate image representations. As a result, we will use a similar structure as the basis for our encoder.

The architecture of our model is shown in Figure 1. We use several layers of convolution, separated by rectified linear activation units and 2×2 max-pooling operations to extract and downsample important image features. We then attach two fully connected layers separated by additional relu activations to map between the extracted features and the appropriate image representation θ_f.

Adding a layer of activation units after the output layer would have the effect of enforcing constraints on the weights of the MLP $\hat{f}$. For example, a tanh activation layer would constrain all values of θ_f to fall in the range [-1, 1], providing a form of regularization. However, in order to provide e with maximal flexibility to reconstruct images, we have chosen to omit such an activation layer in our model. Studying the effects of different activation layers is left as an area for future work.

B. Training

In order to train e in a traditional supervised fashion, we would need to have access to a dataset mapping images to an appropriate representation, θ_f. While such a dataset could be generated given a large enough collection of images using the techniques outlined in Section III, doing so would require training a network to completion for each image in that collection, which is not practical.

Instead, we use the function $\hat{f}$ as an objective metric to guide training directly. We select an image from the training collection and forward-propagate it through e to obtain an initial image encoding θ_f. That encoding is split into individual vectors, each of which is reshaped to form the weight and bias matrices used by $\hat{f}$. $\hat{f}$ is then evaluated using a mini-batch of pixels selected from the chosen image, producing a $B \times c$ matrix of reconstructed pixels. As in Section III, we calculate the gradient of some reconstruction error metric (e.g. Cross-Entropy) with respect to each of the components of θ_f, which is then backpropagated back into e in order to update the encoder's parameters θ_e. We repeat the process K times using the MCMC algorithm presented

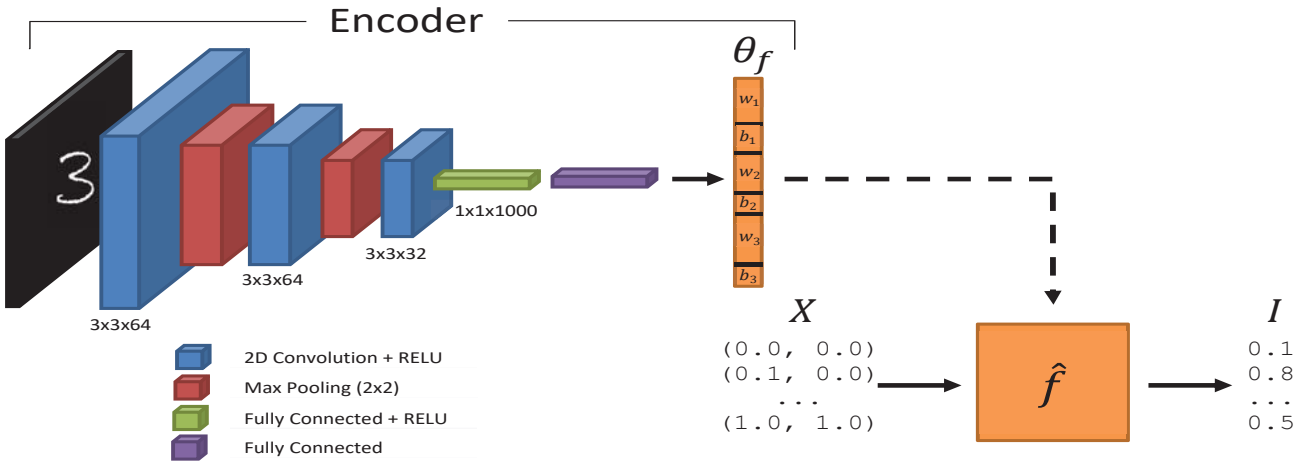

Figure 1: Architecture of the encoder. We use convolution and max pooling to identify important image features and fully connected layers to learn the mapping to θ_f.

in Section III to select which pixels to use in each mini-batch. After the K iterations have passed, another image from the training set is selected, and the algorithm repeats until e produces encodings of sufficient quality.

As with most other neural networks, it is possible for e to overfit, causing it to produce encodings that do not generalize effectively. In addition to other common training procedures, we found that by using smaller values of K (e.g. $K = 10$) and by making use of slightly weaker optimizers (e.g. RMSProp [21] instead of ADAM [22]), we can counteract these types of issues in practice.

V. Applications

There are several practical applications for resolution-independent image representations, including, but not limited to, image resizing, compression, and security. In this section, we survey these applications.

A. Image Resizing

A straightforward application of our research is the ability to resample images to arbitrary resolutions. This is achieved by feeding a different set of relative pixel coordinates x' to $\hat{f}$ than the model was initially trained on. If W_d and H_d represent the desired width and height in pixels, x' can be calculated as:

$$\begin{aligned} h &= \left\lfloor \frac{i}{W_d} \right\rfloor \\ w &= i \bmod W_d \\ x'_i &= \left[\frac{w}{W_d - 1}, \frac{h}{H_d - 1} \right], \text{where } 0 \leq i < W_d \times H_d \end{aligned} \tag{2}$$

Indeed, we demonstrate in Section VI that simple images such as MNIST digits can easily be rescaled roughly 20x larger, from 28 x 28 pixels to 512 x 512 pixels, with little perceptual loss in quality, while larger images can be comfortably be scaled by a lesser amount.

B. Compression

Another interesting application of this research is the ability to efficiently compress images, especially higher-resolution ones. Our generation network $\hat{f}$ is designed to encode images using relatively few weights (e.g. 5,503 for the topology given in Section III with 3 channels). For the purposes of storage or transmission, only those weights need to be persisted, rather than each of the individual pixels of the original image. As a result, the resolution-independent encoding can be smaller than the traditional representation.

For example, the famous "Lenna" image often used with Image Processing research is a 220 x 220 pixel RGB image, requiring 145,200 bytes uncompressed (220 x 220 x 3 bytes per pixel). The corresponding scale-independent representation would require 22,012 bytes uncompressed (5,503 weights x 4 bytes per weight), an 84% reduction in size or roughly a 7:1 compression ratio. For comparison, a standard JPEG compressed version of the same image with the highest quality level requires 47,145 bytes, which is a 67% reduction in size or a 3:1 compression ratio.

With many lossy compression algorithms, ours included, a tradeoff can be made between file size and reconstruction quality. By limiting the topology of $\hat{f}$, we can simultaneously constrain the capacity of the neural network while introducing additional reconstruction error. In Section VI, we will demonstrate the results of a more complete compression test, showing how as the size of the network topology correlates positively to the reconstruction quality.

C. Security

A final application of our technique is relevant to information security, particularly as an additional layer of "security through obscurity". Sensitive images that have been encoded

to a resolution-independent format would not be able to be viewed by a malicious entity without understanding the significance of the values, similarly to how most binary formats cannot easily be read without understanding the file format in which they were saved.

Assuming an attacker did understand that the individual bits of an image's resolution-independent representation could be interpreted as 4-byte floating point values, those numbers would still be meaningless without the associated network topology, which would not necessarily have to be encoded into the file format itself, and an understanding of what the inputs and outputs to the network represent.

As resolution-independent representations are not currently commonplace in many real-world scenarios, entities interested in preserving the privacy of their users could adopt our technique as an additional line of defense beyond other orthogonal approaches, such as strong encryption or multi-factor authentication.

VI. Evaluation

In this section, we validate the claims that have been made so far in the paper. Firstly, we demonstrate that a resolution-independent image encoding can be learned for a single image using a small MLP. Then we demonstrate our results for training a single generic encoder that outputs reasonable image encodings directly. Next, we compare our resampling approach to several others in common use. Finally, we perform a comparison between the size of the resolution-independent encoding and reconstructive accuracy, showing that the two are positively correlated.

For our evaluation, we will use examples from the MNIST database of handwritten digits, photo #6 from the Kodak dataset, and the famous "Lenna" photo often used in image processing. The resolutions of each (in pixels) are 28 x 28, 192 x 128, and 220 x 220, respectively. The MNIST images are grayscale, while the others are traditional RGB images.

A. Resolution-independent Image Encodings

We first demonstrate that the techniques discussed in Section III can be applied to learn a resolution-independent image encoding directly. For this experiment, a small network was trained on a single image from the MNIST dataset of handwritten digits using the ADAM optimizer and a learning rate of 0.001 for 50,000 iterations. This network was then used to reconstruct the image used for training. We repeated the process for the first 20 elements of the MNIST dataset to generate the entries in Figure 2. Notice that simple images can be reconstructed relatively easily and that most images show little perceptible error.

Next we show how the choice of batching method affects the convergence of training. In Section III, three alternative methods were proposed: block training, random pixel sampling, and an MCMC-based approach. For this test, two individual models were created, one for a single MNIST image, and another for the Lenna image. Both networks were trained to convergence using each of the three approaches. For the MCMC-based approach, several iterations of random sampling were used at the beginning of training before switching to the MCMC algorithm as discussed in Section III. The resulting training curves are given in Figure 3. Note that the figure on the left is displayed in log-scale to better visualize the differences between each method.

In both cases, block sampling proved to be the least effective of the three approaches, as it converged slower and had a higher resulting reconstruction error. The MCMC-based approach clearly outperformed random sampling for the MNIST test, but the two approaches produced virtually identical results for the Lenna test. We believe the MCMC method tends to perform better when there are steep color gradients as opposed to shallow ones, as demonstrated by its performance on the MNIST image.

B. General Image Encoder

In this section, we evaluate the performance of our generic encoder. For this test, an encoder with the architecture given in Figure 1 was created and trained on 10,000 images from the MNIST dataset using the training procedure outlined in Section IV. That encoder was then used to recreate 10 additional images that were not part of the training set at their original resolutions, as well as the first 10 images in the training set. The results are given in Figure 4.

We see that the encoder was generally able to provide image encodings that allowed for good reconstructions of both the images that were seen previously and those that were not. Importantly, it seems that some of the imperfections present in several of the images were effectively corrected by the encoder. For example, the digit 8 in row 2 appears to have a ! symbol next to it in the original image, but that symbol was removed by the encoder. There were also holes in each of the two previous examples (the 0 and 9) that were filled during reconstruction. This demonstrates that the encoder does appear to have some understanding about the content of the images. It is not simply memorizing and parroting the encoding space.

C. Scaling

As mentioned in Section V, one application of our work is the ability to resample an image at a higher resolutions. To demonstrate the utility, Figure 5 shows how various interpolation algorithms perform on one particular element of the MNIST dataset. For this experiment, the original image, which has a resolution of 28 x 28 pixels, was upscaled to 512 x 512 pixels (roughly 20x larger), using various common methods for image interpolation. Our method captures the unique features of the original image while avoiding unnecessary blurring or other artifacts.

(a) Original Images (b) Reconstructed Images

Figure 2: A network has been trained on each individual image to generate a corresponding resolution-independent encoding. Images were then reconstructed at the original resolution.

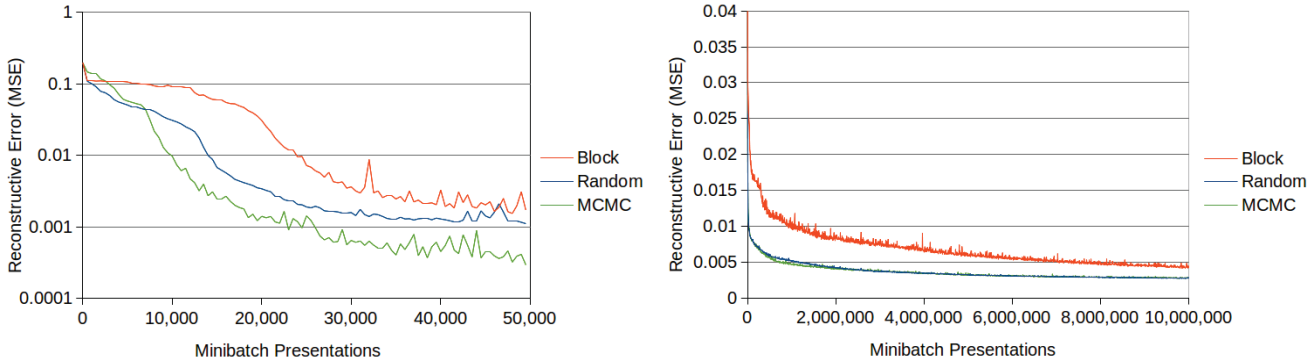

Figure 3: A comparison of three batching methods on two different images. Left: an arbitrary example from the MNIST dataset. Right: Lenna.

(a) The first 10 training images. (b) Reconstructions of the first 10 training images. (c) The 10 testing images. (d) Reconstructions of the 10 testing images

Figure 4: A generic encoder was trained on 10,000 images from the MNIST dataset. Reconstructions were produced without any additional training.

D. Encoding Size and Reconstruction Error

For our last experiment, we compare how the size of θ affects reconstructive quality. For this test, we trained several networks to reproduce three images: an example from the MNIST database, photo #6 from the Kodak dataset, and the Lenna image. The size of the topology of $\hat{f}$ was varied to produce larger (and more expressive) encodings. The results are given in Figure 6. Intuitively, as the size of the encoding increases, so does the reconstructive power of the network. Note that relatively small topologies are sufficient to reconstruct images clearly.

VII. Conclusion

In this work, we discussed a process for learning resolution-independent image encodings. We also demonstrated that a deep convolutional encoder can be trained to produce reasonable encodings for images, avoiding the cost of training a complete neural network for each individual image. We validated our claims with well-known standard datasets. These new methods offer capabilities in such areas as super-resolution scaling, image compression, secure image processing, and other image processing applications.

References

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[2] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[3] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058–1066.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

(a) NN Interpolation (b) Linear Interpolation (c) Cubic Interpolation (d) Sinc Interpolation (e) Our Approach

Figure 5: Various means of upscaling an element from the MNIST database. Images were increased in resolution from 28 x 28 pixels to 512 x 512 pixels.

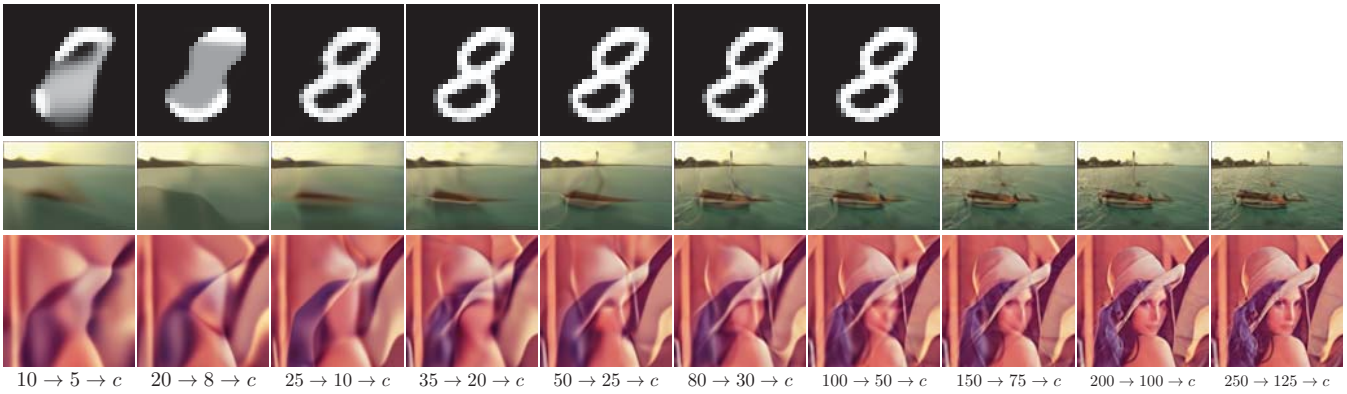

Figure 6: Larger topologies yield better image reconstructions. From top: an example from the MNIST dataset, Kodak #6, Lenna, the topology of $\hat{f}$ using $\tanh()$ activations.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 107, 2017.

[7] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, "High-resolution image inpainting using multi-scale neural patch synthesis," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 2, 2017, p. 3.

[8] F. Luan, S. Paris, E. Shechtman, and K. Bala, "Deep photo style transfer," *CoRR, abs/1703.07511*, 2017.

[9] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman, "Controlling perceptual factors in neural style transfer," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[12] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International Conference on Machine Learning*, 2017, pp. 214–223.

[13] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[14] S. C. Ashmore and M. S. Gashler, "Practical techniques for using neural networks to estimate state from images," in *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*. IEEE, 2016, pp. 916–919.

[15] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[16] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[17] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1646–1654.

[18] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*,

"Photo-realistic single image super-resolution using a generative adversarial network," *arXiv preprint*, 2016.

[19] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017, pp. 5435–5443.

[20] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[21] 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.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.