J. Vis. Commun. Image R. 79 (2021) 103208

Contents lists available at ScienceDirect

Journal of Visual Communication and Image Representation

journal homepage: www.elsevier.com/locate/jvci

Image compression optimized for 3D reconstruction by utilizing deep neural networks

Alex Golts [a,*], Yoav Y. Schechner [b]

[a] Rafael Advanced Defense Systems LTD., Israel
[b] Viterbi Faculty of Electrical Engineering, Technion - Israel Institute of Technology, Haifa, Israel

ARTICLE INFO

Keywords:
Image compression
3D reconstruction
Deep learning
Recurrent neural networks

ABSTRACT

Computer vision tasks are often expected to be executed on compressed images. Classical image compression standards like JPEG 2000 are widely used. However, they do not account for the specific end-task at hand. Motivated by works on recurrent neural network (RNN)-based image compression and three-dimensional (3D) reconstruction, we propose unified network architectures to solve both tasks jointly. These joint models provide image compression tailored for the specific task of 3D reconstruction. Images compressed by our proposed models, yield 3D reconstruction performance superior as compared to using JPEG 2000 compression. Our models significantly extend the range of compression rates for which 3D reconstruction is possible. We also show that this can be done highly efficiently at almost no additional cost to obtain compression on top of the computation already required for performing the 3D reconstruction task.

1. Introduction

Image compression is an essential step in many image processing and computer vision pipelines. At times, the sole goal of compression is to deliver images that a human viewer would perceive as having high quality given some compression ratio. However, computer vision systems are often autonomous. This means that evaluation criteria and compression goals should be tailored to computer vision tasks. This paper focuses on the computer vision task of multi-view three-dimensional (3D) reconstruction. Compression is thus jointly optimized with 3D reconstruction.

Compression in this context is important when multi-view data has to be transmitted using limited resources. Specifically, this applies to imaging using drones [1], airplanes [2] and satellites [3], where power and connectivity are limited. For 3D reconstruction, computation is applied on the transmitted images to extract 3D information in the form of volumetric occupancy grids [4,5], point clouds, depth maps or surface mesh models [6]. The images are typically compressed [7], however, the compression methods that are mostly used are well known standards tailored to image quality metrics, and not directly to evaluation metrics of 3D reconstruction.

To solve our problem we use a deep neural network (DNN). DNN-based methods are increasingly successful and popular at solving various image processing and computer vision tasks. In particular, [8] proposed a recurrent neural network (RNN)-based method for image compression that outperforms the well-known JPEG standard [9]. More recently, several non-RNN based compression methods were proposed. They outperformed [8] achieving compression performance competitive [10], or better than [11,12] the JPEG 2000 standard [13]. Some [14] even exceeding the performance of the more recent BPG codec [15]. Neural networks are a natural choice for image compression, as they often compress their input signal, even when compression is not their explicit goal. For the task of stereoscopic 3D reconstruction, [4] proposed an RNN architecture that learns a mapping from images of objects at different viewpoints, to a 3D occupancy grid corresponding to the object's shape. More recently, [5] improved 3D reconstruction performance while using a more computationally efficient non RNN-based approach. When applying image compression to multiple view scenarios where a large overlap exists between adjacent images, some works [16] leverage the redundancy in the overlapped regions to compress better.

We postulate that jointly learning these two tasks (image compression and 3D reconstruction) may lead to compression better suited for 3D reconstruction. [17] showed that learning multiple visual tasks jointly requires less labeled data to achieve the performance obtained by separate learning-based systems. However, this was shown for small

* Corresponding author.
E-mail addresses: alex@golts.net (A. Golts), yoav@ee.technion.ac.il (Y.Y. Schechner).

https://doi.org/10.1016/j.jvcir.2021.103208
Received 20 November 2020; Received in revised form 12 May 2021; Accepted 27 June 2021
Available online 2 July 2021

auxiliary sub-tasks that aid in a more complex grand task. In our case, both compression and 3D recovery are critical components. Moreover, one can hardly expect compression to actually *help* the mission. It is a necessity. As compared to stand-alone compression, it can be expected that tailored compression enhances the mission performance, for a given compression rate.

In the context of image compression, [18] showed that image understanding tasks such as classification and semantic segmentation can be performed directly on compressed representations derived by DNNs. This alleviates the need to decompress files prior to image understanding. They also further show that by jointly training image compression and classification networks, synergies are obtained leading to performance gains in both tasks.

In this work we propose methods to compress images so that they can be optimally used for 3D reconstruction. Our method can work with only negligible computations for image compression, on top of a system for 3D reconstruction based on [4]. It also exceeds the 3D reconstruction performance obtained from images compressed by JPEG-2000 [13] or learned compression, across a wide range of medium to ultra aggressive compression rates that we focused on. Our main focus in this work is on the regime of high compression rates. We find that lower compression rates may be better suited when the goal is to obtain visually satisfying decompressed images. When the ultimate task is different, it is possible to compress images further. We study these limits for the task of 3D reconstruction.

Our approach uses RNN-based basic components for image compression [8] and 3D reconstruction [4]. It does not aim to propose the most current state-of-the-art for these separate tasks. Rather, we wish to illustrate a generic concept involving the multi-task learning of these tasks. We demonstrate the benefits in considering such a framework, while assuming the above RNN-based building blocks. We hope that our findings provides motivation for further research around this subject, which may also be tailored for using more current basic methods.

2. Background

Section 2.1 provides basic background on RNNs and LSTMs. The reader familiar with these topics may skip to Sections 2.2, 2.3, which provide background on specific works in RNN-based image compression and 3D reconstruction that we build on.

2.1. Recurrent neural networks

2.1.1. Motivation

A feedforward neural network's forward pass accepts a single fixed size input vector $\mathbf{x}$. The network applies a fixed amount of computations, depending on the network depth and architecture. The network then outputs a single fixed size output vector $\mathbf{y}$. A common example is image classification, where, $\mathbf{x}$ is an image and $\mathbf{y}$ is a vector of probabilities that the imaged object belongs to a class. Consider for a moment $\mathbf{x} = \{\mathbf{x}_t\}$ to be a movie sequence, where t denotes a time step. The task is to classify events over time [19]. Proper understanding of a moment in the movie should rely on understanding of previous frames. Feedforward neural networks cannot reason about previous events to inform later ones. RNNs address this issue.

Fig. 1 shows a block diagram of an RNN in its cycled (left), and equivalent unrolled form (right). Here $f_{\mathbf{w}}$ is a neural network block with parameters $\mathbf{w}$, which is a recurrence formula applied at every time step t on the current input $\mathbf{x}_t$. A loop shown in Fig. 1 (left) allows information to be passed from one time step of the network to the next. The unrolled form (right) reveals that an RNN can be thought of as simply multiple copies of the same network layer, each passing a message to a successor. The recurrence formula $f_{\mathbf{w}}$ in its general form is

$$\{\mathbf{h}_t, \mathbf{y}_t\} = f_{\mathbf{w}}(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{1}$$

Eq. (1) operates on the current input $\mathbf{x}_t$ and the RNN's hidden state $\mathbf{h}_{t-1}$ from the previous time step. The outputs of Eq. (1) are $\mathbf{h}_t$ and $\mathbf{y}_t$, the RNN's hidden state and output in the current time step, respectively. Contrary to feedforward neural networks, RNNs commonly operate over *sequences* of vectors, rather than an individual vector [20].

2.1.2. Sequential processing of individual vectors

It is sometimes beneficial to apply RNNs even when both the input and output are individual, fixed size vectors (See [21]). For example, in image compression, (Section 2.2), the input is a fixed size single image, and the output is either a binary compressed code, or a fixed size image. An output compressed code is often of a variable length, depending on the desired compression rate. While such compression *can* be performed using a feedforward neural network, it can benefit from the sequential processing power and memory capability of RNNs [22,8].

2.1.3. Vanilla RNNs

In a standard (vanilla) RNN, the block $f_{\mathbf{w}}$ performs a simple recurring operation

$$\mathbf{h}_t = \tanh(\mathbf{W}_{\mathrm{h}}\mathbf{h}_{t-1} + \mathbf{W}_{\mathrm{x}}\mathbf{x}_t), \quad \mathbf{y}_t = \mathbf{W}_{\mathrm{y}}\mathbf{h}_t. \tag{2}$$

Here the tanh activation function is applied elementwise, and $\mathbf{W}_h, \mathbf{W}_x$ and $\mathbf{W}_y$ are learned parameters (weights) of the RNN. Multiple such RNN layers can be concatenated to form deep RNNs. In practice, such standard RNNs suffer from difficulty in learning long-term dependencies throughout a sequence [23]. Therefore, a special kind of RNN called Long-Short Term Memory (LSTM) network, was proposed by [24], and adopted widely and successfully.

2.1.4. LSTM networks

LSTMs are explicitly designed to avoid the long-term dependency problem. When $f_{\mathbf{w}}$ is an LSTM, it has a more complicated structure compared to the vanilla RNN. Its block diagram is shown in Fig. 2. Mathematically, the LSTM recurrence formulae are given by a set of equations as follows. In these equations, $\odot$ denotes element-wise multiplication, σ denotes the Sigmoid function $\sigma(u) = [1 + e^{-u}]^{-1}$, and $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o$ and $\mathbf{b}_c$ are learned bias vectors.

There are three basic *gates* involved in LSTM computation: *forget*, *input* and *output* gates, which are respectively:

$$\mathbf{f}_t = \sigma\big(\mathbf{W}_{\mathrm{fx}}\mathbf{x}_t + \mathbf{W}_{\mathrm{fh}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{f}}\big), \tag{3}$$

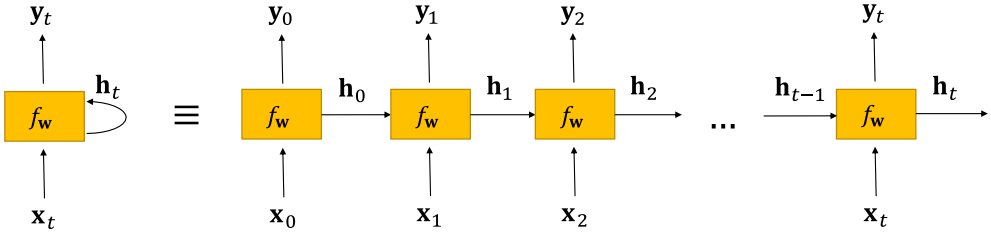


Fig. 1. Block diagram of an RNN in its cycled (left) and equivalent unrolled form (right).

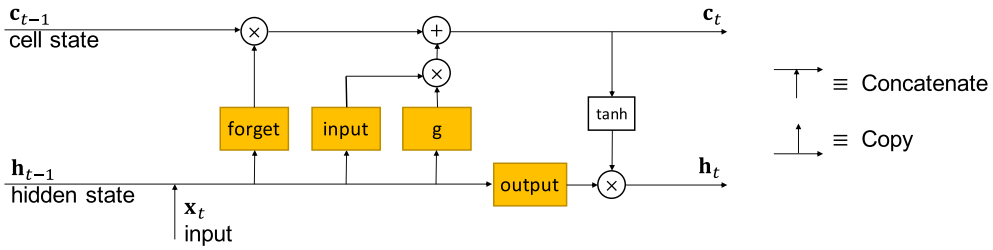


Fig. 2. Block diagram of an LSTM. A notation legend appears on the right.

$$\mathbf{i}_t = \sigma(\mathbf{W}_{\mathrm{ix}}\mathbf{x}_t + \mathbf{W}_{\mathrm{ih}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{i}}), \tag{4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{\mathrm{ox}}\mathbf{x}_t + \mathbf{W}_{\mathrm{oh}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{o}}). \tag{5}$$

Another gate, which usually does not have an explicit name in the literature, is

$$\mathbf{g}_t = \tanh(\mathbf{W}_{\mathrm{cx}}\mathbf{x}_t + \mathbf{W}_{\mathrm{ch}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{c}}), \tag{6}$$

The *cell state* of the LSTM, is defined by

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t, \tag{7}$$

The hidden state $\mathbf{h}_t$ depends on the cell state through

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{8}$$

In the case of an LSTM, the hidden state is also the final RNN output (Eqs. 1,2

$$\mathbf{y}_t \equiv \mathbf{h}_t. \tag{9}$$

Intuitively, and following Eq. (7), the forget gate regulates whether to forget the previous cell state $\mathbf{c}_{t-1}$ in calculating the current cell state $\mathbf{c}_t$. The input gate regulates whether to write the new input $\mathbf{x}_t$ and hidden state $\mathbf{h}_t$ to the current cell state. The gate $\mathbf{g}_t$ weights *how much* of these new vectors to write to the cell state. Finally, the output gate regulates how much to reveal the current cell state for updating the hidden state.

This formulation shows that an LSTM unit explicitly controls the flow from input to output. The gates regulate how much and which information from the previous time steps and current input, proceeds to the next time step. It can be shown [25,26] that in an LSTM, unlike a vanilla RNN, the gradient of the cell state with respect to itself at some previous time step, does not decay exponentially in the time step difference. This is what allows the LSTM to mitigate the vanishing or exploding gradient issue, which exists in vanilla RNNs and may prevent learning of long term dependencies.

Multiple slight alternative formulations of the LSTM equations have been proposed. A more conceptual alternative is the Convolutional LSTM proposed by [27], in which the matrix multiplications in Eqs. (3)–(6) are replaced with convolutional filtering, in analogy to convolutional neural networks (CNNs). This is especially useful for image-related tasks.

2.2. RNN-based image compression

We now briefly discuss an RNN-based method for image compression proposed by [8], which we use in our joint compression and 3D reconstruction framework. The method is a single model architecture capable of producing variable rate compression (see Fig. 3). The encoder $\mathscr{E}^{\mathrm{comp}}$ and decoder $\mathscr{D}^{\mathrm{comp}}$ are RNN-based, therefore the processing occurs sequentially over time t. A single iteration is shown in Fig. 3. In the first iteration, the original image $\mathbf{x}$ is encoded by $\mathscr{E}^{\mathrm{comp}}$ to a vector of length m in floating point representation, per element. Then, a binarizer module $\mathscr{B}$ converts the encoded representation into a binary vector. This binary vector is part of the compressed representation. In each further iteration, a new binary vector of length m is formed and added to the full compressed representation. This way, the number of iterations controls the compression rate.

Decoding applies $\mathscr{D}^{\mathrm{comp}}$ on the binary representation $\mathbf{b}_t$ and outputs either ($\gamma = 0$) a full reconstructed image $\widehat{\mathbf{x}}_t$ or ($\gamma = 1$) a residual from the previous reconstruction $\widehat{\mathbf{x}}_t - \widehat{\mathbf{x}}_{t-1}$. At subsequent iterations, the input to the encoder becomes the residual image $\mathbf{r}_t = \mathbf{x} - \mathbf{x}_t$, which is the difference between the original and reconstructed image. During training, the absolute value of this residual is minimized. The process is defined formally as follows

$$\widehat{\mathbf{x}}_t = \mathscr{D}_t^{\mathrm{comp}}(\mathbf{b}_t) + \gamma\widehat{\mathbf{x}}_{t-1}, \quad \mathbf{b}_t = \mathscr{B}\left[\mathscr{E}_t^{\mathrm{comp}}(\mathbf{r}_{t-1})\right], \tag{10}$$

$$\mathbf{r}_t = \mathbf{x} - \widehat{\mathbf{x}}_t, \quad \mathbf{r}_0 = \mathbf{x}, \quad \widehat{\mathbf{x}}_0 = 0 \tag{11}$$

Here $\mathscr{E}_t^{\mathrm{comp}}$ and $\mathscr{D}_t^{\mathrm{comp}}$ represent the RNN-based encoder and decoder with their states at iteration t, respectively; $\mathbf{x}$ is an original image of size $H \times W \times 3$ and $\widehat{\mathbf{x}}_t$ is its progressive reconstruction, with $\gamma = 0$ for "one-shot" reconstruction[1], and $\gamma = 1$ for additive reconstruction[2]; $\mathbf{r}_t$ is the residual between $\mathbf{x}$ and the reconstruction $\widehat{\mathbf{x}}_t$; and $\mathbf{b}_t \in \{-1, 1\}^m$ is an encoded bit stream produced by a binarizer function $\mathscr{B}$, where m is the number of bits produced per iteration. It depends on H, W and M, the number of output channels from the final layer of $\mathscr{E}^{\mathrm{comp}}$ through $m = (H/16) \times (W/16) \times M$ [3]. The number of RNN iterations N controls the overall compression ratio. A compression rate c is defined as the ratio of the number of bits in the raw image to that of the binarized representation. In our case we have 8 bit-per-pixel RGB images. Thus, in order to obtain a given compression rate c, the number of RNN iterations needs to be set to

$$N = \frac{H \times W \times 3 \times 8}{c \times m}. \tag{12}$$

The encoder consists of a convolutional layer followed by three convolutional RNN layers, i.e. convolutional LSTMs. [8] evaluated additional options for the recurrence unit, besides LSTM. The binarizer in [8] consists of a convolutional layer, followed by a binarizing operation, such as the one used in [22]. In [8], further lossless compression is achieved using entropy coding.

During training, the binarizer in [[22,28]] is a stochastic variable. Assuming the input x is the output of a tanh layer, the binarizer is defined as

$$b(x) = \begin{cases} 1 & \text{with probability } \dfrac{1+x}{2} \\ -1 & \text{with probability } \dfrac{1-x}{2} \end{cases}. \tag{13}$$

For back-propagation, the derivative of the expectation is taken. Since, $\mathbb{E}[b(x)] = x \ \forall x \in [-1, 1]$, the gradients are passed through $b(x)$ un-

[1] Full image reconstruction is computed at each iteration

[2] Only the residual is computed at each iteration. The final reconstructed image is a sum of the outputs of all iterations.

[3] $M = 32$ in [8].

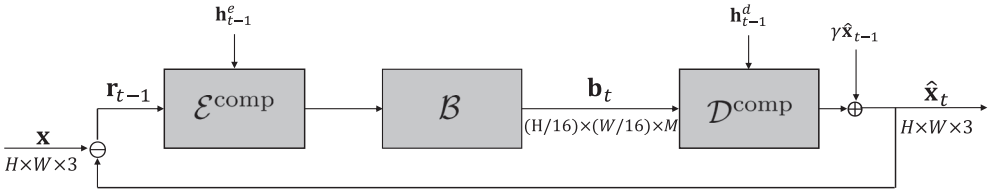


Fig. 3. Compression RNN architecture. Here $\mathbf{x}$ is the original input image, $\mathbf{x}_t$ is the reconstructed (decoded) image after t RNN iterations, $\mathbf{r}_{t-1}$ is the residual image from the previous iteration, $\mathbf{b}_t$ is the compressed binarized representation, and $\gamma \in \{0,1\}$ regulates whether the reconstruction is additive or "one shot". The architecture consists of a convolutional RNN based encoder $\mathscr{E}^{\text{comp}}$ and decoder $\mathscr{D}^{\text{comp}}$, whose hidden states are denoted by $\mathbf{h}^e_{t-1}$ and $\mathbf{h}^d_{t-1}$ respectively, and a stateless binarizer module $\mathscr{B}$.

changed.

The decoder in [8] starts with a convolutional layer, followed by four convolutional RNN layers, with each such layer followed by a depth-to-space[4] layer [29]. Each such layer decreases the depth size by a factor of 4, thus increasing spatial resolution by a factor of 2 in both row and column dimensions. Finally, another convolutional layer is applied to produce a $H \times W \times 3$ reconstructed image.

During training, a weighted L_1 loss on the residual $\mathbf{r}_t$ is minimized

$$L_{\text{comp}} = \beta \sum_{ijt=0}^{HWN} \left|\mathbf{r}_{ijt}\right| \tag{14}$$

where for minibatch size B, $\beta = (B \times H \times W \times 3 \times N)^{-1}$. The sum is over the image spatial dimensions i,j and RNN iterations t.

2.3. RNN-based 3D reconstruction

We now discuss the second important component for our joint framework for compression designed for 3D reconstruction, namely the 3D reconstruction method of [4]. They proposed a network architecture whose input is a single or multiple images of an object, from arbitrary viewpoints. The network output is a reconstruction of the object in the form of a 3D occupancy grid (see Fig. 4). Here, $\{\mathbf{x}_i\}_{i=1}^{V}$ are images of an object from V different viewpoints, and $\mathbf{p}_V$ is the 3D reconstructed occupancy grid produced after V viewpoint. It is represented as Softmax probabilities indicating occupancy of each point in the 3D grid.

The encoder $\mathscr{E}^{\text{3D}}$ and decoder $\mathscr{D}^{\text{3D}}$ are feedforward CNNs. Two architecture variants are proposed in [4]: a shallow and a deeper one with residual blocks [30]. Both encoder variants consist of convolutional layers followed by LeakyReLU nonlinearities, and six MaxPooling operations, each with a stride of 2, for a total encoder stride of 64. Finally, a fully-connected layer is applied, which returns a vector of size K, an embedding of each viewpoint image.

The 3D convolutional LSTM module is a core component proposed in [4]. It allows the network to retain details it has observed in previous views and update the memory when it receives a new image. The module consists of a grid of $4 \times 4 \times 4$ 3D Convolutional LSTM units, each with a hidden state of size N_h. These are different from the convolutional LSTM units in Section 2.2 in two regards: First, the input of the LSTM module is a vector which undergoes multiplication as in a standard LSTM definition. Only the hidden state undergoes convolution. Second, with the hidden state being three-dimensional, the convolution operation is now in 3D. Each such unit is responsible for reconstructing a particular part of the 3D voxel space. In [4] the choices for K and N_h are 1024 and 128, respectively.

During training, a 3D voxel-wise Softmax loss over the final viewpoint's output $\mathbf{p}_V$, is minimized

$$L_{\text{3D}} = \sum \widetilde{\mathbf{p}} \log \mathbf{p}_V + (1-\widetilde{\mathbf{p}}) \log(1-\mathbf{p}_V) \tag{15}$$

Here, $\widetilde{\mathbf{p}} \in \{0,1\}$ is the 3D ground-truth occupancy, and the sum is over the three voxel dimensions (indices omitted for simplicity).

The 3D Convolutional LSTM enables the network to be invariant to the order of viewpoints that are fed to it. Indeed, the V viewpoints consisting each input sequence are selected at random, and there is no single preferred order.

3. Joint compression and 3D reconstruction

The motivation for joining the tasks of compression and 3D reconstruction into a unified framework can be twofold. A primary motivation is to obtain compressed representations better suited for 3D reconstruction, and thus obtain improved 3D reconstruction performance, as compared to when using known compression standards. Another motivation can be in reducing overall computational cost by providing a unified network architecture that would be more efficient than applying sequentially the compression and 3D reconstruction architectures shown in Figs. 3, 4.

We now discuss different unified models that achieve the above needs. In all of the following proposed network architectures, the loss is optimized with respect to all model parameters jointly. No sub-model elements are pre-trained, and no sequential training protocols take place, such as where one sub-model is trained first, then freezed, while another sub-model is trained. The Binarizer in all of the following proposed architectures has no learned parameters, but a gradient is passed through it as explained in Section 2.2.

3.1. 3D reconstruction from decoded images (Sequential model)

Here we propose a sequential approach, as shown in Fig. 5. The compression model (Fig. 3 and 3D reconstruction model (Fig. 4) are simply concatenated. The 3D reconstruction part of the network receives as input a decompressed image. During training, we optimize a loss which depends on viewpoint i as follows

$$L_{\text{total}} = \begin{cases} L_{\text{comp}} & i < V \\ L_{\text{comp}} + L_{\text{3D}} & i = V \end{cases}. \tag{16}$$

For $i < V$ we use $\mathscr{D}^{\text{comp}}$ to obtain a decompressed image $\widehat{\mathbf{x}}_i$, and the compression loss L_{comp} is applied and attempts to make $\widehat{\mathbf{x}}_i$ similar to the original image $\mathbf{x}_i$. In the last viewpoint V, this is done as well, but now, additionally the 3D reconstruction $\mathbf{p}_V$ is calculated, and the corresponding loss L_{3D} is also added to the optimization. For simplicity, we refer to this model from now on as the *sequential* model.

3.2. 3D reconstruction from compressed codes (Direct model)

We also propose a more computationally efficient approach shown in Fig. 6. Here, the output codes following $\mathscr{E}^{\text{comp}}$ and $\mathscr{B}$ are used directly to feed the 3D LSTM module. Thus, applying $\mathscr{E}^{\text{3D}}$ is not required, and some computation is reduced. Here we also optimize the loss from Eq. (16). For simplicity, we refer to this model from now on as the *direct* model.

[4] The depth-to-space or Pixel Shuffle layer rearranges elements of a tensor of shape $\left(*, C \times k^2, H, W\right)$ to a tensor of shape $(*, C, H \times k, W \times k)$. This, followed by a convolution layer is useful for upsampling sub-pixel convolution with a stride of $\frac{1}{k}$

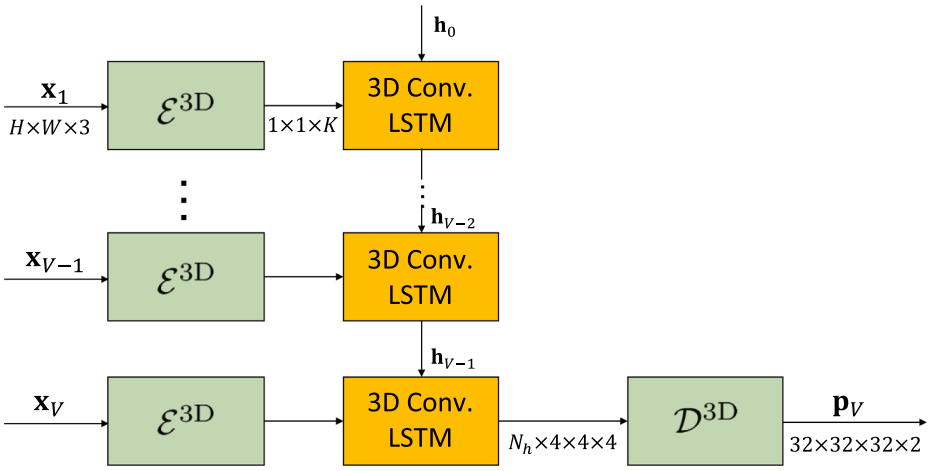


Fig. 4. 3D reconstruction network architecture. Here $\{\mathbf{x}_i\}_{i=1}^{V}$ are the input images, and $\mathbf{p}_V$ is the 3D reconstructed occupancy grid produced after V viewpoints. The encoder $\mathscr{E}^{3D}$ produces a vector of size K from each input image. These vectors enter a $4\times 4\times 4$ grid of 3D convolutional LSTM units, each with a hidden state $\mathbf{h}$ of size N_h. In the final view V, the hidden states are processed by a decoder $\mathscr{D}^{3D}$ to obtain the 3D reconstruction $\mathbf{p}_V$. Note that the same encoder $\mathscr{E}^{3D}$ and 3D convolutional LSTM units, with the same learned parameters are used for processing every viewpoint image.

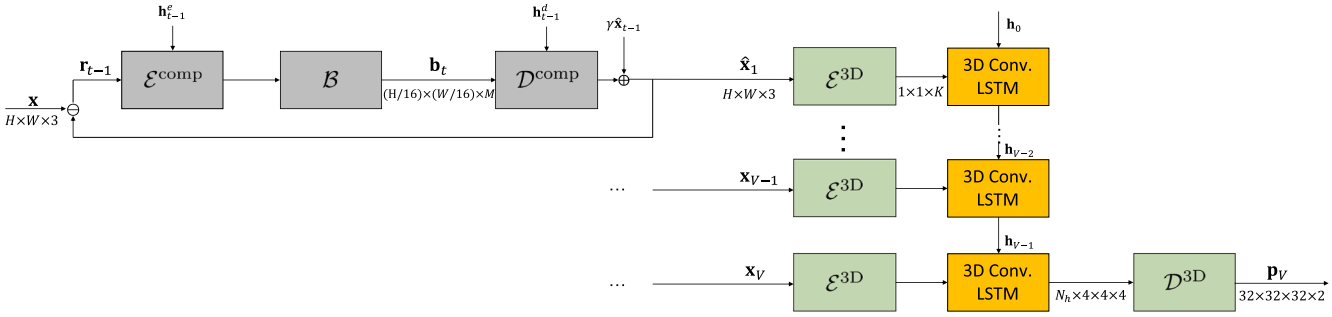


Fig. 5. The *sequential* model recovers a 3D occupancy grid from images decoded by a compression module. Note that the same compression encoder $\mathscr{E}^{comp}$, binarizer $\mathscr{B}$, compression decoder $\mathscr{D}^{comp}$, 3D reconstruction encoder $\mathscr{E}^{3D}$, and 3D convolutional LSTM units, with the same learned parameters, are used for processing every viewpoint image.

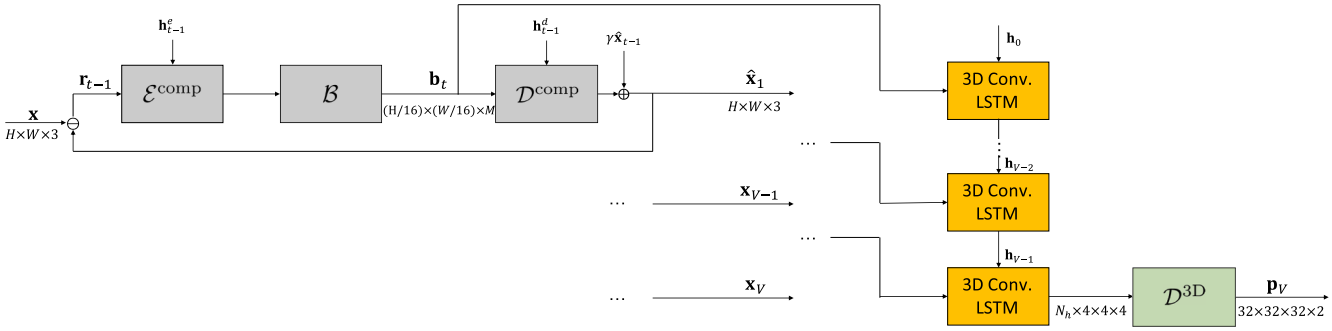


Fig. 6. The *direct* model recovers a 3D occupancy grid directly from the compressed codes. Note that the same compression encoder $\mathscr{E}^{comp}$, binarizer $\mathscr{B}$, compression decoder $\mathscr{D}^{comp}$, and 3D convolutional LSTM units, with the same learned parameters, are used for processing every viewpoint image.

3.3. 3D reconstruction with implicit compression (Implicit model)

Finally, we propose another approach that is significantly more efficient computationally than those in Sections 3.1, 3.2. The idea here is to use the RNN-based 3D reconstruction architecture discussed in Section 2.3, and augment it with a binarizer $\mathscr{B}$ so that compression is obtained implicitly, without explicitly minimizing an image compression loss L_{comp}. The loss we minimize here is simply L_{3D}. This makes sense in a scenario where we wish to solve the 3D reconstruction task by supplying compressed image codes. We do not require that our model reconstruct viewable images. Only that it successfully solves the task at hand, which is to reconstruct a 3D occupancy map. Since we focus on the 3D reconstruction task, we can minimize L_{3D} only.

The binarizer module $\mathscr{B}$ can be thought of as a form of regularization in training the network on the 3D reconstruction task. It imposes a constraint on the minimization of L_{3D}, which is to yield a good 3D reconstruction while requiring the encoded representation to take binary form. This architecture does not use the variable compression rate RNN framework of [8], where multiple RNN iterations controlled the compression rate. Rather, we train a separate slightly modified model for every desired compression rate. The compression rate is now controlled by K, the output vector length of the CNN encoder $\mathscr{E}^{3D}_K$.

The proposed architecture is shown in Fig. 7. Here, the encoder $\mathscr{E}^{3D}_K$ differs from $\mathscr{E}^{3D}$ in that the default choice of K, the encoded vector length is now varied with every compression rate. A separate model is trained for different choices of K obtained by modifying the number of channels in the output of the final convolutional layer of the encoder.

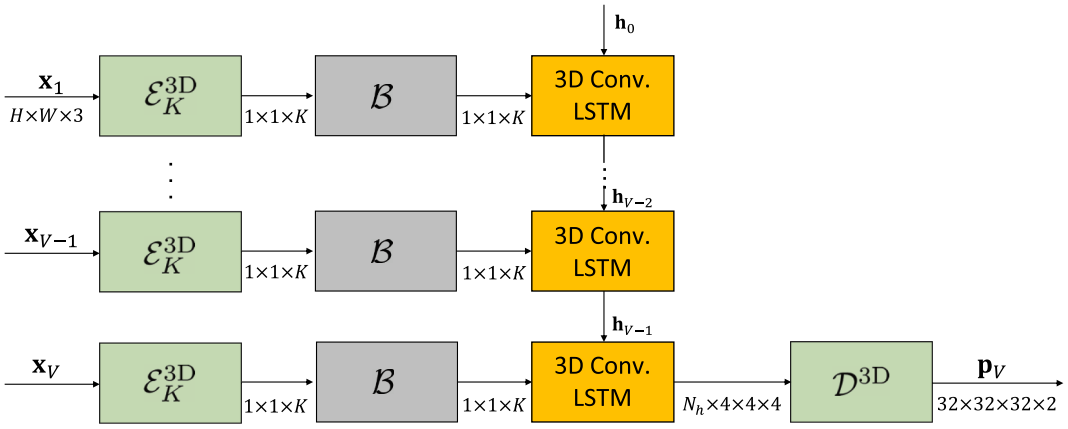


Fig. 7. The *implicit* model architecture recovers a 3D occupancy grid from compressed codes, without ever decompressing them into reconstructed images during training. Only a 3D reconstruction loss is minimized, thus image compression is obtained implicitly. Note that the same encoder $\mathscr{E}_K^{3D}$, binarizer $\mathscr{B}$, and 3D convolutional LSTM units, with the same learned parameters, are used for processing every viewpoint image.

Here K is also the number of bits of the compressed representation (after the binarizer). For simplicity, we refer to this model from now on as the *implicit* model.

Note that the *direct* model from Section 3.2 can be seen as a possible extension of the *implicit* model, in case decompressed viewable images are required. It is similar in structure, except the encoder is $\mathscr{E}^{\text{comp}}$ instead of $\mathscr{E}_K^{3D}$, a decoder is added to produce the decompressed images, and decompression is done recurrently, allowing variable rate compression in a single architecture.

3.4. Implementation details

Our unified models were trained using a minibatch size of 6 so that training could be done on a single GTX 1080 Ti GPU. In Table 1 we compare our reimplementation of [4] to the original work. Note that the model evaluated here is for 3D reconstruction only, without image compression involved. The evaluation criterion is mean Intersection-over-Union (mIoU) between a 3D voxel reconstruction $\mathbf{p}$ and its ground truth voxelized model $\widetilde{\mathbf{p}}$, over the test set. The IoU criterion is defined [[4]] as

$$\text{IoU} \equiv \frac{1(\mathbf{p}>\tau)\cap 1(\widetilde{\mathbf{p}})}{1(\mathbf{p}>\tau)\cup 1(\widetilde{\mathbf{p}})} = \frac{\sum[1(\mathbf{p}>\tau)1(\widetilde{\mathbf{p}})]}{\sum[1(1(\mathbf{p}>\tau)+1(\widetilde{\mathbf{p}}))]}. \tag{17}$$

Here, τ is a threshold we set to 0.4 as in [4], $\cap$ and $\cup$ denote the intersection and union operations, respectively, 1 denotes the Indicator function, and the sums are over the three voxel dimensions. Out of the different architecture variations that [4] experimented with, we chose one of the simplest, termed 3D-LSTM-3. It consists of a network of moderate depth and uses an LSTM recurrence unit with a 3x3x3 convolution kernel. [4] trained the network for roughly 60 epochs. We see that we can train for only 20 epochs and still obtain good performance, only 3% lower than the optimum. Therefore, for practical reasons, we settle on training all of our models for 20 epochs, from here onward. We also see that our implementation results in somewhat improved performance compared to the original implementation of [4]. This performance gain was not found to result from the few deviations from [4] that are known to us, i.e, different minibatch size. We discuss the architecture of the compression part in our *sequential* and *direct* models (Sections 3.1, 3.2) in the appendix.

Table 1
Implementations of the 3D reconstruction method of [4]. The evaluation criterion is mean IoU and standard deviation (STD) across the test set. (*) Here the error measure is an approximation of the STD. [4] quote the mIoU 15% and 85% percentiles. Their difference is approximately 2σ assuming Gaussian distribution.

Implementation	mIoU
Choy et al. (original)	0.54 ± 0.22 (*)
Ours - 20 epochs	0.64 ± 0.20
Ours - 60 epochs	0.66 ± 0.20

In [8], the compression model was trained on small 32x32 image patches. In our *sequential* and *direct* proposed approaches (Sections 3.1, 3.2), we want to build upon the architecture of [8] to train a unified model for both compression and 3D reconstruction. Therefore, we want the input to our models to be full resolution 128x128 images from the ShapeNet dataset used by [4]. Compared to [4,8], we wish to train larger models on larger inputs. We also wish to be able to do this on a single GPU. To handle this challenge, reducing the minibatch size to 6, as mentioned in Section 3.4 was not enough. We needed to further optimize the compression network architecture. We provide elaborate discussion of our choice of compression network architecture in the appendix.

In this work we focus on the ShapeNet dataset of 128x128 rendered images. In different scenarios one may wish to apply our methods on much larger images. Our architectures are fully convolutional, therefore they can be directly be scaled to larger inputs with linear increase in memory and computation. However, the spatial receptive field would not grow unless further architectural changes are made, such as increasing the depth of the network, or using dilated convolutions [31].

4. Results

4.1. 3D reconstruction

In Fig. 8, we report our models' mean Intersection-over-Union (mIoU) scores on the ShapeNet test set, using 5 viewpoints for the 3D reconstruction, as in [4]. Table 2 shows a visual comparison of 3D reconstructed occupancy grids produced for 5 random viewpoints of the objects shown in Tables 4, 6.

We see that for aggressive compression rates of 128 and above, the *sequential* model is superior to a 3D reconstruction network trained on images compressed by JPEG 2000 using a random compression rate at each minibatch. Moreover, for an extremely aggressive compression rate of 384, JPEG 2000 typically compresses images to the extent that no details are seen at all. The mIoU of the network trained on JPEG 2000 compressed images at this point is 0.1, which is meaningless. In our *sequential* and *direct* models, this compression ratio is obtained by using only one RNN timestep. It still produces images that contribute to 3D

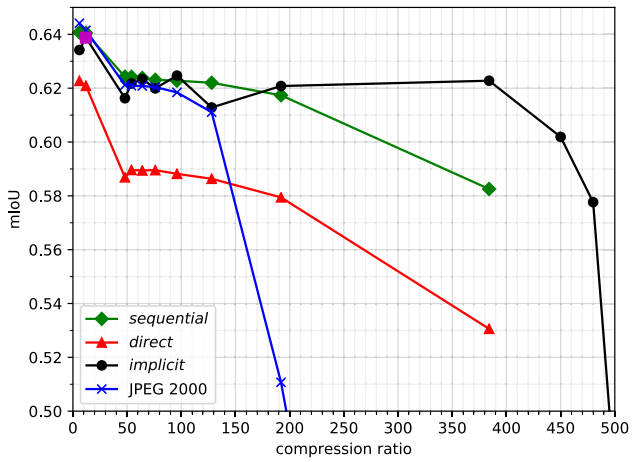


Fig. 8. 3D reconstruction mIoU scores on the ShapeNet test set, using 5 viewpoints. We compare a network as in [4] trained and applied on JPEG 2000 compressed images (blue), to our *sequential* (green), *direct* (red) and *implicit* (black) models. The magenta square at a compression rate of 12 denotes our reimplementation of the original [4] model. The standard deviation (STD) across the test set for all models and compression rates is approximately 0.2.

reconstruction with mIoU > 0.5. We also see that the best results are obtained using the most efficient *implicit* model.

4.2. Statistical significance

The 3D reconstuction IOU metric varies significantly for different test examples, even without compression. This is due to the natural variability between objects. Different objects can be easier or more challenging for accurate 3D reconstruction than others. For the baseline 3D reconstruction model of [4], the STD of the IOU across the test examples is roughly 0.2 (see Table 1). We ask ourselves then, how significant are some of the trends depicted in Fig. 8? To obtain a better insight for this, we separate the test examples into bins of roughly equal IOU STD of 0.04 on the baseline model. For this value of STD, we obtain 7 such bins which contain (in increasing order of mean IOU) 17, 82, 131, 273, 340, 384 and 233 examples. To illustrate this, we show the mIOU results for bin #5 with error bars, in Fig. 9. Here, we see more clearly that the performance trends and differences between models, are of statistical significance. In the two lowest mIOU data bins, the mIOU for all the different models and most compression rates is below 0.2. For such low IOU examples, the evidence for statistical significance is lacking. It may also be explained by the relatively small number of examples that are contained in these low IOU bins.

4.3. Lower compression rates

Note that the original network of [4] naturally obtains an image compression ratio of 1:12, without any use of a binarizer (the magenta square in Fig. 8) or otherwise special design. It can be thought of as a special case of the *implicit* model, with the binarizer module disabled, and set to $K = 1024$ in $\mathscr{E}_K^{3D}$. Thus, the magenta square is plotted on top of the *implicit* model curve.

This work focused on high, limiting compression rates. Still, we wanted to provide a more complete overview of our methods. Therefore, we additionally evaluated all our models for two lower compression rates of 1:12 and 1:6. We can say roughly that all models tend to approach the "ideal" case of the magenta square at these low compression rates.

If we used our *sequential*, *direct* and *implicit* architectures with binarizer, obtaining such low compression rates would become a memory burden. Therefore, we disabled the binarizer and used a 32 bit floating point encoded vector. We set N the number of RNN iterations, or the number of channels in the last encoder layer K (for the *implicit* architecture) appropriately in order to achieve these rates. For the network trained on JPEG 2000 compressed images, we re-trained it

Table 2
Visual comparison of our models' 3D reconstruction capability, for different compression rates. The occupancy grids are produced for the same four objects that were used in Tables 4, 6. The 3D reconstruction for uncompressed images is shown identically in both rows for convenience.

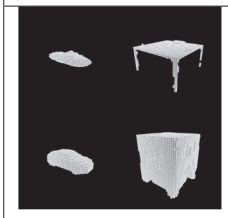
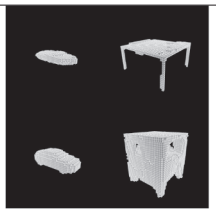
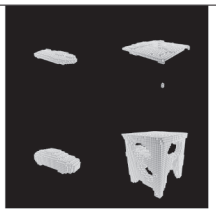
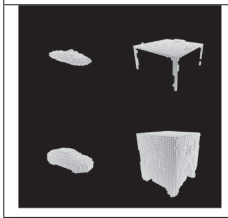
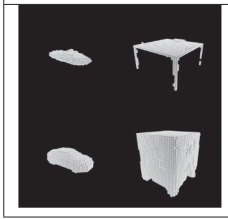
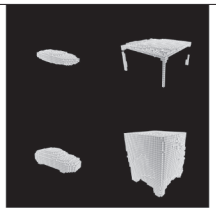
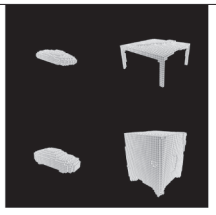
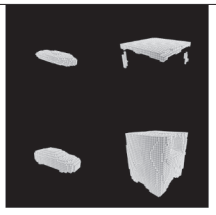
1:1 (uncompressed)	48:1	192:1	384:1	480:1	
					***Sequential* model**
					***Direct* model**
					***Implicit* model**

Table 4
Visual comparison of the *sequential* and *direct* models when probed for their compression capability, for different compression rates. Four different images from the ShapeNet test set are shown. The uncompressed images are shown identically in both rows for convenience.

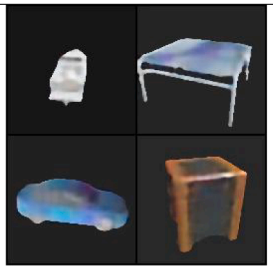
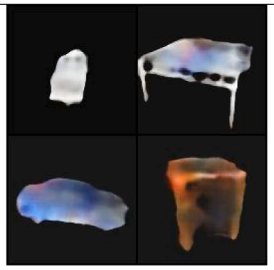
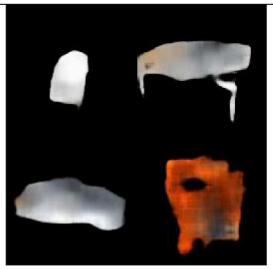
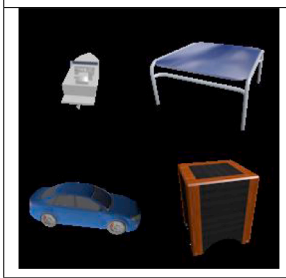
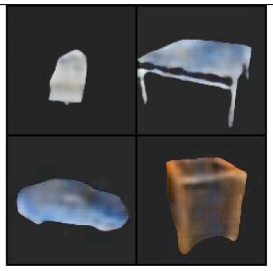
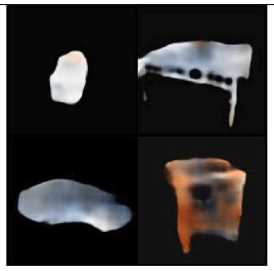
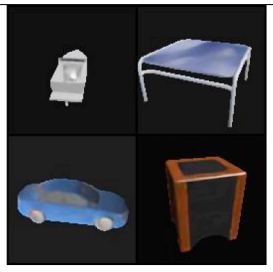
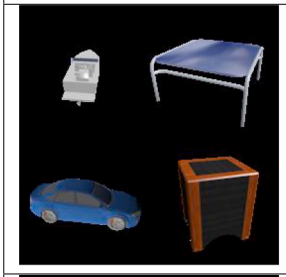
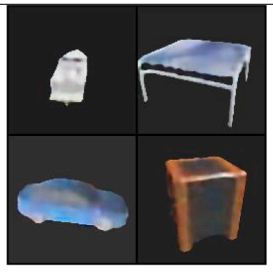
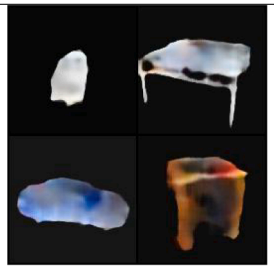
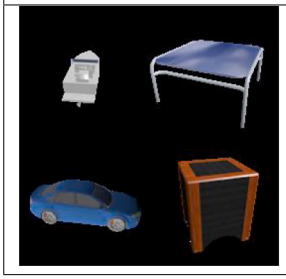
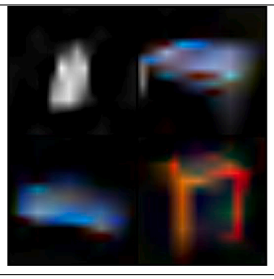
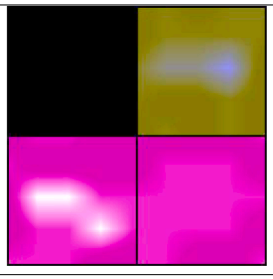
1:1 (uncompressed)	48:1	192:1	384:1	
				***Sequential* model**
				***Direct* model**

Table 6
Visual comparison of different compression methods for different compression rates. Four different images from the ShapeNet test set are shown. The uncompressed images are shown identically in all three rows for convenience.

1:1 (uncompressed)	48:1	192:1	384:1	
				$\mathcal{N}^{\text{original}}$
				$\mathcal{N}^{\text{small}}$
				**JPEG-2000**

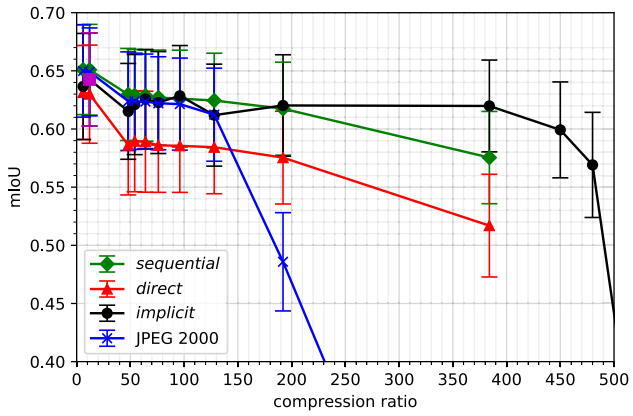


Fig. 9. 3D reconstruction mIoU scores shown with error bars, on a subset of 340 examples from the ShapeNet test set, for which the STD is 0.04 (on the baseline model of [4]), using 5 viewpoints. We compare a network as in [4] trained and applied on JPEG 2000 compressed images (blue), to our *sequential* (green), *direct* (red) and *implicit* (black) models. The magenta square at a compression rate of 12 denotes our reimplementation of the original [4] model.

separately for 1:12 and 1:6 compression rates.

4.4. Run time

Table 3 shows time of execution for different proposed models and building blocks on a single GTX 1080 Ti GPU, during both train (forward and backward pass) and test time (forward pass only)[5]. A typical scenario of 5 viewpoints and 4 RNN iterations was assumed when timing 3D reconstruction and compression modules, respectively. We see that the using $\mathscr{N}^{\text{small}}$ as the compression module in our *sequential* unified model results in a speed-up of 15% in the forward pass compared to using $\mathscr{N}^{\text{original}}$. Using the more efficient *direct* model further improves the relative speed-up percentage to 26%.

This improvement is moderate, implying that the most of the computation lies within the recurrent compression module. One way to increase the speed-up could be in eliminating the variable compression rate constraint and train a model per compression rate. This is done in the *implicit* model, however without reconstructing viewable decompressed images.

The most significant speed-up is not surprisingly obtained by the *implicit* model which only optimizes L_{3D}, and simply augments the 3D reconstruction model by [4] with a binarizer module $\mathscr{B}$ (Section 3.3). Here, the relative speed-up percentage in forward pass reaches 75%. Note that we trained all models on the same data and number of epochs, so these relative speed-ups also indicate the difference between whole training procedures required to obtain a deployable model.

Table 3
Execution time of different models and building blocks. 3D reconstruction was measured for 5 viewpoints and compression was done using 4 RNN iterations. Bold numbers in parentheses denote relative time improvement percentage compared to sequentially applying compression as in [8] followed by 3D reconstruction as in [4]

Model/ Block	Forward pass [msec]	Backward pass [msec]
3D Reconstruction	13	13
3D Reconstruction ($\mathscr{E}^{3D}$ only)	6	≈ 6
Compression - $\mathscr{N}^{\text{original}}$. [8]	40	24
Compression - $\mathscr{N}^{\text{small}}$. (See appendix)	32	20
Sequential model (Section 3.1)	45 **(15%)**	33 **(11%)**
Direct model (Section 3.2)	39 **(26%)**	27 **(27%)**
Implicit model (Section 3.3)	≈ 13 **(75%)**	≈ 13 **(65%)**

[5] The backward pass time of $\mathscr{E}^{3D}$ is approximated. It was not measured directly since it was done through the whole 3D reconstruction module with a single PyTorch.backward() call. It is reasonable to assume that it is identical to the forward pass, as this was the case when measured on the whole 3D reconstruction module. The timing of the joint 3D reconstruction with implicit compression model (Section 3.3) is approximated as identical to just the 3D reconstruction model of [4], since it only adds a single binarizer element on top of it, that is negligible in computational complexity compared to the whole model in both forward and backward passes.

4.5. Compressed image decoding

Although our goal is 3D reconstruction from compressed image representations, it is useful to also be able to decompress into visually satisfying 2D images. Our *sequential* and *direct* models allow this, since they were also optimized for the 2D image decoding task. Here, we evaluate their performance on this task. Fig. 10 shows the compression loss magnitude that our models achieve on the ShapeNet test set, and Table 4 shows a visual comparison. We see that the compressed image decoding performance of the *sequential* model is virtually indistinguishable from that of $\mathscr{N}^{\text{small}}$, a model trained on the image compression task only (See appendix). In contrast, the more computationally efficient *direct* model is somewhat inferior. This could be explained by a harder constraint that this model imposes on the compression codes, requiring them to be suitable for directly feeding the 3D reconstruction model, in addition to allowing for a good compressed image decoding.

In order to gain more qualitative insight on the decompression capability, we show in Table 5 another visual example, with four other images which were manually selected to contain richer texture content. Here we also show JPEG-2000 decompression results for comparison. We can see that for high compression rates, the neural network based compression is superior to JPEG-2000, however in the 48:1 rate, JPEG-2000 manages to keep details better, particularly in areas of rich, high spatial frequency texture. This may show that such image details were not essential to learn good 3D reconstruction, in our case where the 3D models ground truth is relatively low resolution (32x32x32 voxel grids).

5. Discussion

The main conclusions from the paper can be summarized as follows:

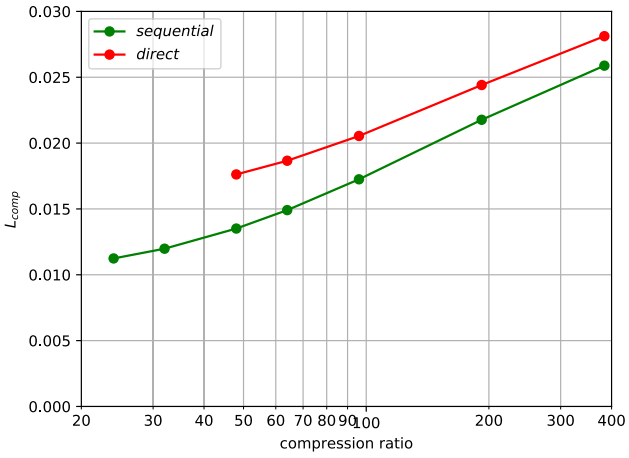
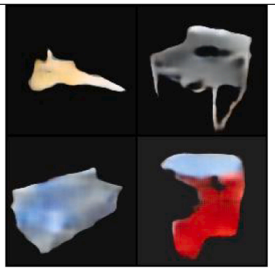
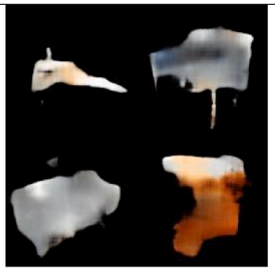
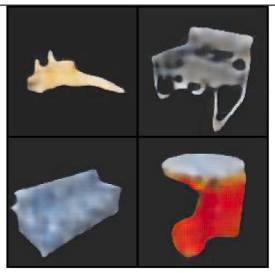
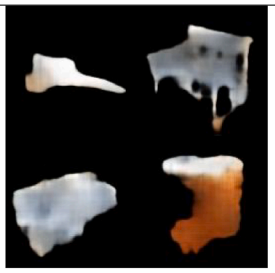
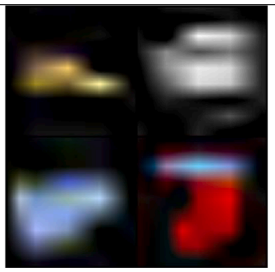
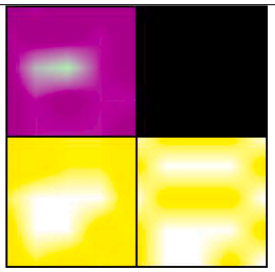


Fig. 10. Compression loss L_{comp} obtained on the ShapeNet test set by the *sequential* and *direct* model architectures.

Table 5
Visual comparison of the *sequential* and *direct* models when probed for their compression capability, and JPEG-2000, for different compression rates. Four different images from the ShapeNet test set are shown. Relatively detailed texture images were manually selected. The uncompressed images are shown identically in both rows for convenience.

1:1 (uncompressed)	48:1	192:1	384:1	
				***Sequential* model**
				***Direct* model**
				***JPEG* 2000**

- Three neural network architectures for joint image compression and 3D reconstruction were proposed which trade-off 3D reconstruction quality, image decompression, and run time.
- Near optimal 3D reconstruction is obtained for a wide range of compression rates, including aggressive rates where JPEG-2000 fails completely.
- The *implicit* architecture reaches the most aggressive compression rates without performance loss, while also being the fastest in run time.

Let us now discuss in more detail. The *sequential* and *direct* model architectures include an RNN-based module dedicated for image compression and allows for compressed image decoding. These architectures are trained to optimize two loss functions simultaneously, one for 2D image compression, and another for 3D reconstruction. As such, they imply significant additional computations (both in train and test time) compared to the 3D reconstruction module alone. The *implicit* model architecture is both more computationally efficient and accurate. It is trained only by minimizing the loss function associated with the 3D reconstruction task. The compression is obtained implicitly via a binarizer module which is inserted as part of the 3D reconstruction network architecture. The computational overhead that this variant imposes on top of an existing 3D reconstruction module alone, is negligible.

In future work, techniques for speeding up training time of current models, especially the *sequential* and *direct*, could be explored. Training speed-up could be obtained by not training from scratch as we did, and instead using pre-trained weights. Training speed-up could also be obtained by using more efficient base models for compression and 3D reconstruction. Finally, existing numerical techniques such as mixed precision training could be applied.

We showed that all our proposed architectures allow for reasonable 3D reconstruction from images compressed aggressively at a compression ratio of 384:1, where JPEG-2000 compression is impractical. Two of our more accurate architectures also outperform RNN-based 3D reconstruction trained on JPEG-2000 compressed images at less aggressive compression rates. Our best performing *implicit* architecture outperforms RNN-based 3D reconstruction trained on JPEG-2000 compressed images for almost the whole range of compression rates, while also reaching accepTable 3D reconstruction performance under even more aggressive compression rates than all other examined variants (up to 480:1).

The *implicit* model yields relatively constant and nearly “ideal”, non-compressed performance throughout a wide range of compression rates. This motivated us to try apply it using even more aggressive compression rates then the previously chosen 384. And indeed we see that reasonable results are still obtained for up to a compression rate of 480, after which they deteriorate quickly.

The *implicit* model is not optimized for acceptable image recovery. It only considers 3D reconstruction. This has limitations, in that such a model could only be used in specific applications, where a human observer is not present, or is not interested in the images, but only in the final 3D reconstructed occupancy grids. However, this also means that this model solves a much simpler problem. It does not need to account

for two constraints simultaneously. Thus, for suitable applications, it is highly beneficial. It achieves superior 3D reconstruction capability over a very broad range of compression rates. And it does so using much less computations.

The somewhat noisy, non-monotonic trend of the mIoU vs. compression rate of the *implicit* model may perhaps be explained by that in contrast to the *sequential* and *direct* models, here, a separate model is trained for every compression rate, so it is reasonable that there can be some degree of inconsistency.

Another reason could be related to our choice of varying the compression rate by modifying the number of output channels from the last layer of $\mathscr{E}_K^{3D}$. This choice may not necessarily be the best one. One could modify the architecture in different ways to control the compression rate, and it may be that restricting the modification to just a single layer is somewhat sub-optimal and may result in some overfitting if the number of neurons in a single layer is too large. This could explain the slightly lower performance counter-intuitively obtained for better compression rates.

Some techniques can be suggested in order to obtain a more smooth performance curve across compression rates. Increasing regularization through weight decay, dropout or other known techniques, could lead to more stability. Additionally, it's possible to use a common backbone architecture with identical weights across all compression rates, while only retraining the last convolutional layer of the encoder, for each rate. This could also reduce variance across compression rates.

We also note that for low compression rates, not using the binarizer and setting K accordingly is one possible design choice, and not necessarily the most optimal one. We could, for instance, use a quantizer that outputs 4, 8 or 16 bits, together with a different choice of K. Or modify the encoder architecture in another way. We leave the focus on moderate to low compression rates for future work.

We find it interesting that 3D reconstruction based on neural networks can be made highly robust to image compression, with only slight performance degradation for an extremely wide range of compression rates. We believe that it may be worth exploring neural networks' robustness to significant image compression for other computer vision tasks as well.

Another promising direction for future work may be to further improve the achievable compression rates, by considering the overlap between adjacent image views in the compression module, instead of compressing the images separately before feeding the compressed representations to the 3D reconstruction module.

We note again that our approach is meant to propose a generic concept which offers benefit in jointly framing the tasks of image compression and 3D reconstruction. Both these separate tasks are well researched and the state-of-the-art in each of them has been improved in recent years over the methods we chose to use in our models.

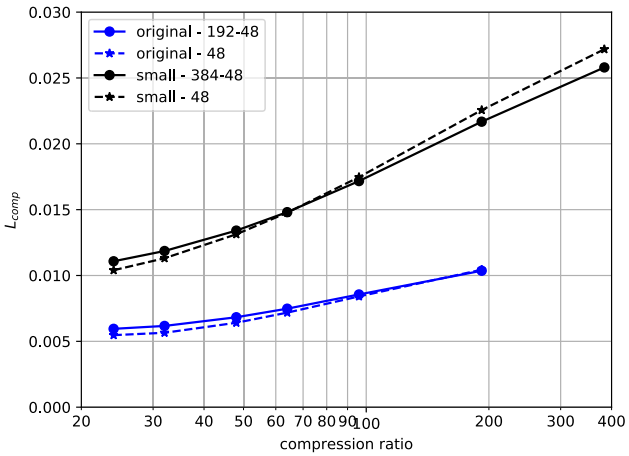


Fig. 11. L_{comp} for $\mathscr{N}^{\text{small}}$ and $\mathscr{N}^{\text{original}}$. Each alternative was trained using a constant number of RNN iterations corresponding to a compression rate of 1:48, and randomly varying up to a rate of 1:384.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank Christopher Choy for his responsiveness to our questions about his work, as well as Alona Golts for her helpful insights.

Yoav Schechner is the Mark and Diane Seiden Chair in Science at the Technion. He is a Landau Fellow - supported by the Taub Foundation. His work is conducted in the Ollendorff Minerva Center. Minvera is funded through the BMBF. This project is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 810370: CloudCT) and the Technion Autonomous Systems Program (TASP).

Appendix A. Appendix - Choice of compression architecture

Due to the challenge in single GPU training of network architectures that extend RNN-based compression modules, on full resolution 128x128 images, we propose a *smaller* compression network architecture $\mathscr{N}^{\text{small}}$ relative to the *original* one $\mathscr{N}^{\text{original}}$ proposed in [8]. We now elaborate the differences between them.

In our proposed smaller architecture, the 3rd (last) convolutional RNN layer of the encoder $\mathscr{E}^{\text{comp}}$ outputs 16 feature maps instead of 512. Therefore, the encoded representation is already compact and the binarizer does not include a convolutional layer. The binarized representation vector length in $\mathscr{N}^{\text{small}}$ is 16, rather than 32 in $\mathscr{N}^{\text{original}}$. This allows us to achieve a more aggressive maximal compression ratio of 384 vs. 192 (The maximal compression rate is obtained when just a single compression RNN iteration is used. Lower rates are obtained as desired using more iterations). The decoder $\mathscr{D}^{\text{comp}}$ in our proposed $\mathscr{N}^{\text{small}}$ consists of only three convolutional RNN layers instead of four, and they are smaller in capacity compared to $\mathscr{N}^{\text{original}}$.

The 3D reconstruction model in [4] used a random number of viewpoints during training. This was useful for obtaining a model that at test time, can reconstruct shape from an arbitrary number of viewpoints. In contrast, [8] trained their image compression model using a constant, maximal number of RNN iterations, which corresponded to a compression ratio of 24:1. We propose to train $\mathscr{N}^{\text{small}}$ and our models based on it, as described in Section 3, using a number of RNN iterations (and thus, compression rate), selected at random in each training iteration. This significantly reduces training time, and also facilitates slightly more robust performance across varying compression rates.

Fig. 11 compares the loss L_{comp} over the ShapeNet test set of $\mathscr{N}^{\text{small}}$ to that of $\mathscr{N}^{\text{original}}$, using both constant and randomly varying number of RNN iterations. We see a significant gap of around a factor of 2 in terms of the loss value in favor of $\mathscr{N}^{\text{original}}$. To gain an intuition of the visual effect of such gap, we show in Table 6 a comparison of a selection of four images from the ShapeNet test set, decoded using the different architectures, at different

compression rates. We see a noticeable difference in quality in favor of $\mathcal{N}^{\text{original}}$, relative to $\mathcal{N}^{\text{small}}$. Despite the difference in visual image quality, we use $\mathcal{N}^{\text{small}}$ throughout our experiments. In Section 4, we show that this choice is still appropriate given that our actual task of interest is that of 3D reconstruction. On this task, the performance that our unified model achieves is only a few percent lower than that achieved using uncompressed images, for a wide range of compression rates.

References

[1] J.D. Renwick, L.J. Klein, H.F. Hamann, Drone-based reconstruction for 3d geospatial data processing, in: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), IEEE, 2016, pp. 729–734.

[2] A. Vetrivel, M. Gerke, N. Kerle, G. Vosselman, Identification of damage in buildings based on gaps in 3d point clouds from very high resolution oblique airborne images, ISPRS Journal of Photogrammetry and Remote Sensing 105 (2015) 61–78.

[3] G. Facciolo, C. De Franchis, E. Meinhardt-Llopis, Automatic 3d reconstruction from multi-date satellite images, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017, pp. 57–66.

[4] C.B. Choy, D. Xu, J. Gwak, K. Chen, S. Savarese, 3d–r2n2: A unified approach for single and multi-view 3d object reconstruction, European Conference on Computer Vision, Springer (2016) 628–644.

[5] H. Xie, H. Yao, X. Sun, S. Zhou, S. Zhang, Pix2vox: Context-aware 3d reconstruction from single and multi-view images, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 2690–2698.

[6] S. Fuhrmann, F. Langguth, M. Goesele, Mve-a multi-view reconstruction environment, in: GCH, 2014, pp. 11–18.

[7] A. Indradjad, A. Nasution, H. Gunawan, A. Widipaminto, A comparison of satellite image compression methods in the wavelet domain, in: IOP Conference Series: Earth and Environmental Science, volume 280, IOP Publishing, 2019, p. 012031.

[8] G. Toderici, D. Vincent, N. Johnston, S.J. Hwang, D. Minnen, J. Shor, M. Covell, Full resolution image compression with recurrent neural networks, in: CVPR, 2017, pp. 5435–5443.

[9] G.K. Wallace, The jpeg still picture compression standard, IEEE transactions on consumer electronics 38 (1992) xviii–xxxiv.

[10] L. Theis, W. Shi, A. Cunningham, F. Huszár, Lossy image compression with compressive autoencoders, arXiv preprint arXiv:1703.00395 (2017).

[11] J. Ballé, V. Laparra, E.P. Simoncelli, End-to-end optimized image compression, arXiv preprint arXiv:1611.01704 (2016).

[12] J. Ballé, D. Minnen, S. Singh, S.J. Hwang, N. Johnston, Variational image compression with a scale hyperprior, arXiv preprint arXiv:1802.01436 (2018).

[13] M. Rabbani, R. Joshi, An overview of the jpeg 2000 still image compression standard, Signal processing: Image communication 17 (2002) (2000) 3–48.

[14] D. Minnen, J. Ballé, G.D. Toderici, Joint autoregressive and hierarchical priors for learned image compression, Advances in Neural Information Processing Systems (2018) 10771–10780.

[15] F. Bellard, BPG Image Format (2014) https://bellard.org/bpg/.

[16] J. Liu, S. Wang, R. Urtasun, Dsic: Deep stereo image compression, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 3136–3145.

[17] A.R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, S. Savarese, Taskonomy: Disentangling task transfer learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 3712–3722.

[18] R. Torfason, F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, L. Van Gool, Towards image understanding from deep compression without decoding, arXiv preprint arXiv:1803.06131 (2018).

[19] C. Olah, Understanding lstm networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs, 2015.

[20] A. Karpathy, The unreasonable effeciveness of recurrent neural networks, http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

[21] P. Shyam, S. Gupta, A. Dukkipati, Attentive recurrent comparators, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 3173–3181.

[22] G. Toderici, S.M. O'Malley, S.J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, R. Sukthankar, Variable rate image compression with recurrent neural networks, arXiv preprint arXiv:1511.06085 (2015).

[23] Y. Bengio, P. Simard, P. Frasconi, et al., Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (1994) 157–166.

[24] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.

[25] J.S. Bayer, Learning sequence representations, Ph.D. thesis, Technische Universität München, 2015.

[26] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International conference on machine learning, PMLR, 2013, pp. 1310–1318.

[27] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-C. Woo, Convolutional lstm network: A machine learning approach for precipitation nowcasting, in: Advances in neural information processing systems, 2015, pp. 802–810.

[28] T. Raiko, M. Berglund, G. Alain, L. Dinh, Techniques for learning binary stochastic feedforward neural networks, arXiv preprint arXiv:1406.2989 (2014).

[29] W. Shi, J. Caballero, F. Huszár, J. Totz, A.P. Aitken, R. Bishop, D. Rueckert, Z. Wang, Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1874–1883.

[30] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Cision and Pattern Recognition, 2016, pp. 770–778.

[31] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, arXiv preprint arXiv:1511.07122 (2015).