Video Compression Using Recurrent Convolutional Neural Networks

Cedric Yue Sik Kin
Electrical Engineering
cedyue@stanford.edu

Berk Coker
Computer Science
bcoker@stanford.edu

Abstract

The demand for video streaming has been growing over the past few years. This has made video storage and video transfer a bottleneck for service providers, increasing the need for more robust video compression algorithms. Deep learning has a potential to address this concern. In this paper, we present an auto-encoder system that consists of over-fitted bidirectional recurrent convolutional neural networks to compress videos. Our findings show that RCNNs can learn from temporal information found in consecutive still frames, but fail to achieve state-of-the-art compression rates and speeds due to computational complexities.

1. Introduction

Over the past five years, we have witnessed the rise of streaming services around the world. From Netflix to Youtube, most of the platforms we now use are dominated by video content. According to a Cisco study, video traffic will become 82 percent of the entire consumer internet traffic by 2020 [1]. These predictions show that we will soon be presented with challenges that pertain to video storage and video transfer. A breakthrough in video compression techniques could address these two problems. The implications of such advancements are vast: better compression algorithms could (i) help service providers save on resources such as servers and (ii) give clients an opportunity to have better video experience in lower bandwidths such as in mobile devices.

In this paper, we propose a deep learning approach to video compression. We build an auto-encoder model based on a bidirectional recurrent convolutional neural network with Gated Recurrent Units (GRUs). The encoder accepts a sequence of frames extracted from a video as input, and returns a sequence of corresponding encoded representations as output. Then, the decoder model reverses the compression and predicts the original frames using the intermediary representations. Notice that in auto-encoders, the encoder and the decoder are trained together. But at test time, they can be distributed into different machines.

1.1. Data Compression

Data compression involves encoding information in fewer bytes than the original representation. Compression can be classified into two types: lossy or lossless. Lossy compression reduces the number of bytes of the original representation by removing unnecessary or less crucial information to human perception. For example, the JPEG image compression scheme works by rounding off nonessential information bits.

On the other hand, information is not lost in lossless compression schemes. They usually exploit statistical redundancies to represent data without any loss of information. For example, an image might have areas that do change in pixel information, allowing for more compact representations using run-length encoding in which consecutive data elements are stored as a single value and count. There are many other schemes that reduce file size by eliminating redundancy and keeping all the original information; see Lempei-Ziv [3] and DEFLATE [5].

In this paper, we present an approach to capturing statistical redundancies mentioned above through the use of neural networks. The goal is to maximize statistical learning from video frames while minimizing information loss and maximizing compression rate. In the later parts of the paper, we analyze the learning and the trade-off between information loss and compression rate.

1.2. Video Codecs

There are many video formats, indicated by the extension of the video file. Videos have a lot of information that can be encapsulated into a single container: video stream, audio stream, metadata, subtitles, etc. and the the video format indicates the method used to encapsulate everything. Each of the individual elements can be compressed using their own compression scheme before encapsulation, such as H.264 for the video stream, and AAC for the audio stream for example.

H.264 is an example of a video codec - combination of the words compressor and decompressor. Video codecs are a set of instructions that identify the method used to compress video into fewer bytes, as well as decompressing it when playing back the video.

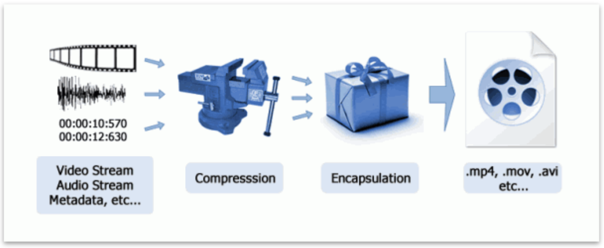

Figure 1. Pipeline for video compression and encapsulation

In the context of this paper, we are only concerned with compressing the visual information in videos. Our approach to video compression is to divide videos into still frames which we use as inputs to our model. Note that this method exponentially increases the size of the input data, requiring much more work to achieve state-of-the-art compression rates. Our goal is to show the opportunity that deep learning presents in this field, that achieves comparable results, which would in return motivate new research.

2. Related Work

We base our work on previous work done in the field of data and image compression and video super-resolution using neural networks.

2.1. Image Compression

The principles of using neural networks for image compression have been know for some time. Jiang wrote a survey of developments of neural network in assisting or even taking over traditional image compression techniques in 1999, by learning more efficient frequency transforms, more effective quantization techniques, etc. [8]. In 2006, Hinton et al. showed an autoencoder architecture that is viable for implementing an end-to-end compression [6]. An autoencoder consists of three parts typically: (1) an *encoder* that takes in the input and encodes it into a (2) *bottleneck* consisting of the compressed data, and (3) *a decoder* whose role is to reconstruct the original input. Hinton et al. trained the entire autoencoder, but during deployment, the encoder and decoder are normally used independently. In 2011, Krizhevsky et al. showed the benefits of encoding the bottleneck as a simple bit vector representation [10]. Such binarization had the following advantages: (1) the bit vectors are easily serializable/deserializable for transmission over the wire, (2) the compression rate can be controlled by putting constraints over the bit allowance, and (3) a binary bottleneck forces the model to learn more efficient representations compared to floating-point layers.

More recently, in 2015, Toderici et al. proposed a general framework for variable-rate image compression and a novel architecture based on convolutional and deconvolutional LSTM recurrent networks [12]. They had better visual quality than (headerless) JPEG, JPEG2000, and WebP, with a storage size reduced by 10% or more. It used the same bit vector representation used by Krizhebsky et al. and accepted fixed size 32x32 inputs.

Toderici et al. followed in 2016 by presenting a full resolution image compressor using recurrent neural networks which could accept any image sizes [13]. Their network consists of an encoding network *E*, a binarizer *B* and a decoding network *D*; *D* and *E* contain recurrent network components. Their system trained by iteratively refining a reconstruction of the original image, with the encoder and decoder using residual GRU layers.

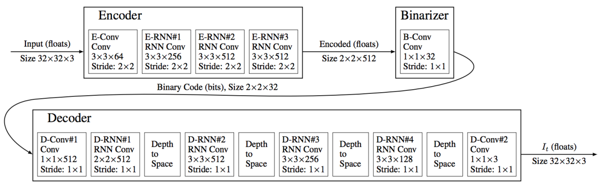
Encoder
E-Conv Conv 3×3×64 Stride: 2×2
E-RNN#1 RNN Conv 3×3×256 Stride: 2×2
E-RNN#2 RNN Conv 3×3×512 Stride: 2×2
E-RNN#3 RNN Conv 3×3×512 Stride: 2×2
Input (floats) Size 32×32×3
Encoded (floats) Size 2×2×512
Binarizer
B-Conv Conv 1×1×32 Stride: 1×1
Binary Code (bits), Size 2×2×32
Decoder
D-Conv#1 Conv 1×1×512 Stride: 1×1
D-RNN#1 RNN Conv 2×2×512 Stride: 1×1
Depth to Space
D-RNN#2 RNN Conv 3×3×512 Stride: 1×1
Depth to Space
D-RNN#3 RNN Conv 3×3×256 Stride: 1×1
Depth to Space
D-RNN#4 RNN Conv 3×3×128 Stride: 1×1
Depth to Space
D-Conv#2 Conv 1×1×3 Stride: 1×1
I_t (floats) Size 32×32×3

Figure 2. A single iteration of the shared RNN architecture used by Toderici et al. in 2017

2.2. Video Super-resolution

Convolutional neural networks have been applied to image super-resolution with state-of-the-art performance. Dahl et al. demonstrated this year a deep network architecture, with pixel recursive super-resolution technique that tried to address the strong prior information that had to be imposed on image synthesis in traditional super-resolution techniques [2]. This year as well, Ledig et al. used a generative adversarial network (GAN) for image super-resolution, capable of infering photo-realistic natural images for 4x upscaling factors [4][11].

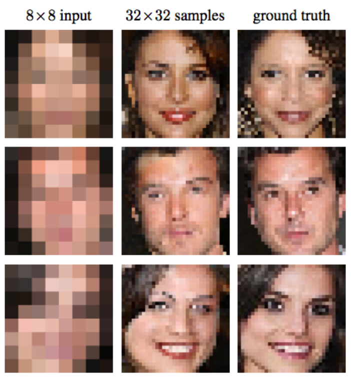

Figure 3. Results achieved by Dahl et al.

But there has been less work in the domain of video super-resolution. Both Huang et al. and Kappeler et al. proposed a CNN that is trained on both the temporal and spatial dimensions of compressed videos to enhance their spatial resolution [7][9]. By taking advantage of the intrinsic temporal dependencies of neighboring video frames, their network is able to perform better and achieve state-of-the-art performance.

3. Methods

3.1. Network Architecture

Our model is a multi-layer, bidirectional, recurrent convolutional neural network that contains two sub-models: the encoder and the decoder. During train-time, the encoder and the decoder are trained together in a single model, where the objective is to retain maximum information after compression and decompression. At test-time, the encoder and the decoder are separated to serve as encoding/decoding modules just as in other compression algorithms. In literature, this architecture is referred to as an auto-encoder. An abstract representation of the model can be found in Figure 4.

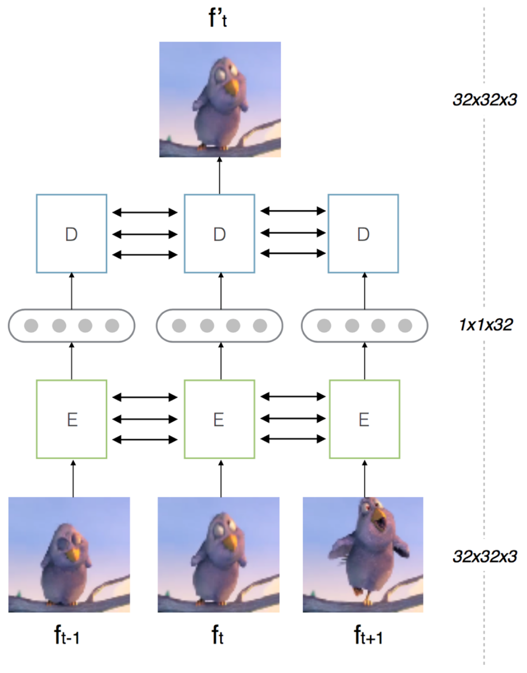

Figure 4. Our architecture. We exploit the temporal dependencies between frames.

The encoder part of our model, marked with the letter E in the figure above, accepts frames of size 32x32x3 at each time step and outputs corresponding vectors of size 1x1x32 for each frame. Then, the decoder, marked with the letter D, uses these vectors the rebuild the frames at each time step. Notice that our model uses a window size of 3 vectors (we experiment with different window sizes) to predict the middle frame: t. This is repeated for each sequence of frames: If we were to proceed to the next time step, we would have predicted the frame at $t + 1$.

3.1.1 Recurrence and Bi-direction

The recurrence and bi-direction in the architecture allows for temporal information to be transferred across time steps, allowing for signal transfer among video frames. We construct each node in the recurrent neural network as a Gated Recurrent Unit (GRU) [13]. The formulation for GRU, with input x_t and hidden state/output h_t is:

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$h_t = (1 - z_t) \circledast h_{t-1} + z_t \circledast \tanh(W x_t + U(r \circledast h_{t-1}))$$

This formulation is repeated twice in the neural network due to the bi-directional structure. The forward and backward weights in the network are separate.

3.1.2 Convolution

Another important factor to note is that the dot products above in the GRU formulation involve convolutions instead of direct dot products. In Figure 5, we demonstrate the entire upstream at a single time-step. Notice that at each layer in the encoder, we use a kernel size of 3x3 and a stride of 2x2. This halves the spatial dimensions each time, achieving the desired compression. On the other hand, in the decoder, we use a deconvolution (transpose of conv2d) at each layer to up-sample in the spatial dimension [4].

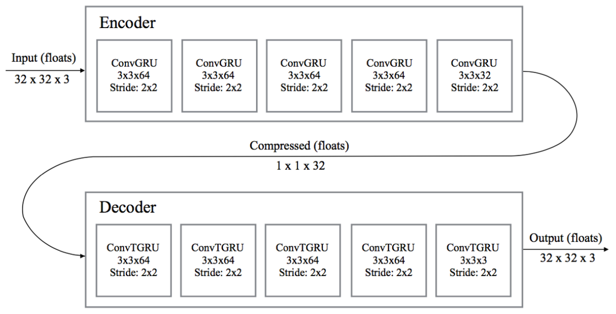
Encoder
Input (floats)
32 x 32 x 3
ConvGRU 3x3x64 Stride: 2x2
ConvGRU 3x3x64 Stride: 2x2
ConvGRU 3x3x64 Stride: 2x2
ConvGRU 3x3x64 Stride: 2x2
ConvGRU 3x3x32 Stride: 2x2
Compressed (floats)
1 x 1 x 32
Decoder
ConvTGRU 3x3x64 Stride: 2x2
ConvTGRU 3x3x64 Stride: 2x2
ConvTGRU 3x3x64 Stride: 2x2
ConvTGRU 3x3x64 Stride: 2x2
ConvTGRU 3x3x3 Stride: 2x2
Output (floats)
32 x 32 x 3

Figure 5. A single iteration of our architecture

3.1.3 Compression

At each time step, we achieve an initial compression rate of $\times 96$. This is because the encoder encodes a frame size of 32x32x3 into a vector representation size of 1x1x32. This comes with a small caveat: while the values stored in frames are of type $int8$, the values stored in vectors are of type $float16$. This, halves the compression rate, resulting in a final compression rate of $\times 48$.

3.2. Training Objective and Evaluation

During training, similar to what Toderici et al. used, an L_1 loss is calculated at each iteration on the absolute residual between the reconstructed and original frame, such that our total loss for the network is:

$$\sum_t |r_t|$$

We experimented with L_2 loss and found that L_1 loss resulted in better performance.

For evaluation, we use the most common evaluation metrics to measure image quality: Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index Measurement (SSIM).

$$\text{PSNR } (\hat{Y}, Y) = 20 \log(s) - 10 \log \textbf{MSE}(\hat{Y}, Y)$$

$$\text{SSIM } (\hat{Y}, Y) = \frac{(2\mu_{\hat{Y}}\mu_Y + c_1)(2\sigma_{\hat{Y}Y} + c_2)}{(\mu_{\hat{Y}}^2 + \mu_Y^2 + c_1)(\sigma_{\hat{Y}}^2 + \sigma_Y^2 + c_2)}$$

where s is the maximum possible pixel value (255 in our case), μ_Y denotes the mean of image Y, μ_Y^2 the variance, $\sigma_{\hat{Y}Y}$ the covariance of the two images and c_1, c_2 typically set to $0.01s^2$ and $0.03s^2$ respectively.

The PSNR is the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Typical values for the PSNR in lossy compression schemes for image or video are between 30 and 50 dB for bit depths of 8 bits. A higher PSNR means more noise has been removed.

SSIM also takes into account the similarity of the edges, i.e. the high frequency content, between the reconstructed image and the original image. Therefore, to have a good SSIM measure, an algorithm needs to remove noise while also preserving the edges in the image.

To get a final value in each metric, we take the average among all of the frames.

3.3. Training Algorithm

For training, we use the Adam optimizer to perform parameter updates based on the gradient of the loss function:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \bigtriangledown X_{t-1}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \bigtriangledown (X_{t-1})^2$$

$$X_t = X_{t-1} - \frac{\alpha m_t}{\sqrt{v_t + \epsilon}}$$

where $\beta_1, \beta_2 \in [0, 1)$ and ϵ are hyperparameters commonly set to 0.9, 0.999 and 1e-8 respectively. m_t is the momentum vector at iteration t, v_t is the velocity vector at iteration t, and α is the learning rate.

3.4. Weight Initialization

We use Xavier initialization:

$$W_{ij} \sim \textbf{Unif}(-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}})$$

where **Unif** denote the uniform distribution, and n_{in} is the number of neurons in the previous layer.

4. Dataset

We train our model by overfitting it to a dataset containing the image frames to a particular video sampled at 24 Hz, with each frame having dimension 160x320. The dataset for which results are included consisted of 1440 frames.

Figure 6. Extract frames from video into a lossless format like png

After the frames are extracted, each frame is split into 32x32x3 quadrants. Each quadrant serves as input to our model. This has two main advantages: (1) it saves computational time, and (2) it allows for variable video sizes as long as the dimension size is a multiple of 32.

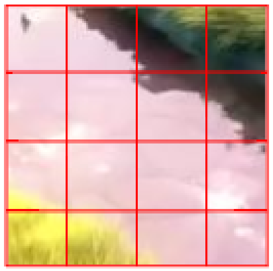

Figure 7. Split frames into 32x32x3 quadrants

5. Experiments

The entirety of this section uses the following hyperparameters in producing the results. These hyperparameter values performed well across the board, and we decided to keep them constant for reasonable comparisons.

Filter Size in Hidden Layers	32
Number of Epochs	5000
Minibatch Size	128
Learning Rate	5e-3
beta1	0.9

Table 1. Final Hyperparameters

5.1. Time-Step Size and Quality

One of the primary experiments we wanted to conduct with our model was to evaluate the information that was being captured through the inclusion of the multiple time-steps, that is, the consecutive sequence of video frames. Since our network was recurrent and bi-directional, our goal was to ensure that this architecture indeed improved the performance. Thus, we experimented with our model by altering the window sizes we used to rebuild the middle frames. The results can be found in Table 2. Notice that we experimented with two different architectures and data: one with 3-layer encoder and decoder with smaller dimensions of data (so that it can be displayed in this paper in), and then one with 4-layer encoder and decoder for the full data. Note that increasing the layers increases the compression: 6-RCNN achieves a compression rate of x32, and 8-RCNN achieves a compression rate of x128. These numbers exclude the size of the model. When the model is factored in, 6-RCNN achieves a compression of x8 and 8-RCNN achieves a compression rate of x10. Remember that we are over-fitting a model for these frames, and therefore need to pass the weights of the decoder to the end-point.

As can be seen in the images displayed in Figure 8, the reproduced images were a little blurry compared to the original images. One can notice this by looking at the tip of the leaves that point up. Regardless, in 6-RCNN, the results were better than JPEG's performance, both in PSNR and SSIM.

	PSNR	SSIM
JPEG (small data)	37.5847 dB	0.9836
6-RCNN (1 Time Step, small data)	37.5971 dB	0.9883
6-RCNN (3 Time Step, small data)	38.8007 dB	0.9910
8-RCNN (3 Time Step, full data)	31.4093 dB	0.9554
8-RCNN (7 Time Step, full data)	31.2896 dB	0.9434
8-RCNN (11 Time Step, full data)	30.0122 dB	0.9435

Table 2. Our results for different time-steps.

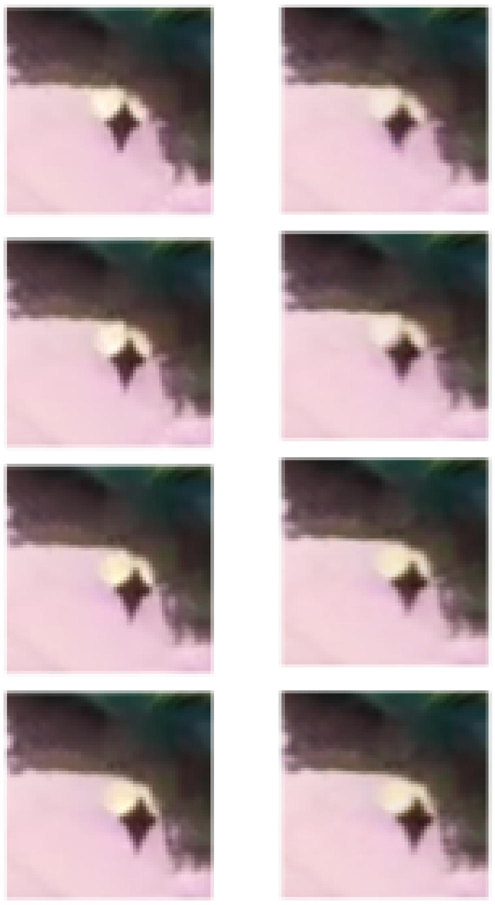

Figure 8. Our results for 6-RCNN. Left: Original. Right: Compressed

Something else we noticed about these different variations of the architecture was that increasing the time-steps improved the performance until a certain point. Notice that the PSNR value goes down for 8-RCNN as we increase the number of time-steps. This, we concluded, was due to the compression rate we were enforcing. At higher compressions rate, wider time-steps did not lead to better results.

The information transferring in our model needs revisiting.

Furthermore, below this threshold, including more time-steps improved the performance but marginally. As can be seen in Figure 9, which demonstrates the loss curves for 8-RCNN, the losses are rather similar. This is also demonstrated in the results table. Our findings showed that our model was indeed transferring information across time steps, and improving

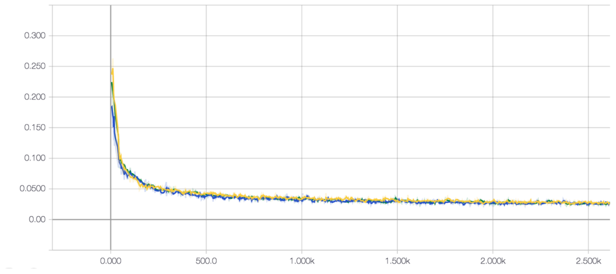

Figure 9. Plot of the loss function. 3, 7, 11

5.2. Compression and Quality

Using our model we experimented with different compression sizes including encoding into 4x4x4 and 2x2x16 dimensions which requires 3 layers, and 1x1x32 dimensions which requires 5 layers. We experimented with the trade-off between compression rate and image quality and settled on using the 1x1x32 encoded size.

6. Conclusion

Given the results from the evaluation metrics, we are able to exploit the intrinsic temporal dependencies between video frames by considering neighboring frames when predicting a video frame. We are able to reconstruct video frames from a compressed size with better performance compared to JPEG compression.

The given results were based on a toy dataset. For future work, we'll focus on extending the model's capacity to reconstruct full video frames and investigate the usefulness of this technique to applications like security footage.

References

[1] Cisco. White paper: Cisco vni forecast and methodology, 2015-2020, 2016.

[2] R. Dahl, M. Norouzi, and J. Shlens. Pixel recursive super resolution. *CoRR*, abs/1702.00783, 2017.

[3] N. et al. Optimized rtl design and implementation of lzw algorithm for high bandwidth applications. *Electrical Review*, 2011.

[4] Z. et al. Deconvolutional networks. *CVPR*.

[5] Z. et al. Deflate compressed data format specification version 1.3. *IEEE Transactions on Information Theory*, 1977.

[6] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[7] Y. Huang, W. Wang, and L. Wang. Bidirectional recurrent convolutional networks for multi-frame super-resolution. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 235–243. Curran Associates, Inc., 2015.

[8] J. Jiang. Image compression with neural networks - a survey. In *Signal Processing: Image Communication 14*, pages 737–760, 1999.

[9] A. Kapperler and Y. D. K. Super-resolution of compressed videos using convolutional neural networks. *International Conference on Image Processing*, 2016.

[10] A. e. a. Krizhevsky. Using very deep autoencoders for content-based image retrieval. *European Symposium on Artificial Neural Networks*, 2011.

[11] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[12] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable rate image compression with recurrent neural networks. *CoRR*, abs/1511.06085, 2015.

[13] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. *CoRR*, abs/1608.05148, 2016.