Lossy GIF Compression Using Deep Intrinsic Parameterization

Anuj Pahuja  
Carnegie Mellon University  
apahuja@andrew.cmu.edu

Simon Lucey  
Carnegie Mellon University  
slucey@cs.cmu.edu

Abstract

With billions of animated GIFs being shared and viewed every day, it has become imperative for GIF hosting websites to serve content with minimal lag. To cater to the ever-decreasing attention span of a wide audience with different connectivity issues, it makes sense to suitably compress GIFs during transmission. We present a unique and interpretable approach to lossily compress GIFs or any temporal sequence of frames through a CNN based image parameterization technique and a simple scalar quantization scheme. Contrary to learned compression techniques, our approach is instance-specific and self-supervised.

1. Introduction

Animated GIFs have become a staple of daily communication and entertainment. GIFs have been around for a long time, but their popularity has surged dramatically in recent years, owing primarily to the increasing use of messaging apps and the meme culture. With more than 2 billion GIFs served on the GIPHY website alone [1], GIFs have started to dominate our digital interactions. It is then reasonable to design algorithms for their efficient storage and transfer.

While the adoption of high-speed internet has popularized the use of GIFs, it has also resulted in a decrease of our attention span, with viewers abandoning a video if the startup delay is beyond 2 seconds [2]. High traffic websites coupled with viewers having a sub-par internet speed often results in higher loading times for all kinds of media, eventually leading to the viewer moving on to something next. Serving suitably compressed GIFs for low latency can prove to be a viable solution for such scenarios.

The resurgence of neural networks has led to new and improved methods for video compression, a domain which otherwise had reached its peak with the highly efficient HEVC codec [7]. Although HEVC is the gold standard for video encoding, the compression quality is still dependent on the quality of input frames. We aim to use the natural image prior of CNNs to tackle that issue and doing more perceptually-driven compression. On the other hand, end-to-end video compression using neural networks is still in its infancy. We propose a unique end-to-end ‘multi-frame’ compression technique that is primarily suited for animated GIFs but could also be extended to videos through keyframing. For spatial compression, our method extends upon the observation that untrained CNNs act as a regularizer for inverse tasks [8], and the concept of intrinsic dimensionality of a neural network [3]. For temporal compression, we propose a Markovian technique with a simple scalar quantization scheme. Our method is instance-specific and hence devoid of any dataset bias. We posit that our method is simple, unique and interpretable that combines the recent advances in deep learning for the task of sequential multi-frame compression.

2. Related Work

Since GIF itself is a compressed format, not much work has been done to further compress it. We treat GIF as a sequence of temporally related frames, which can be considered as a specific form of video. For an exhaustive review of video compression techniques using neural networks, we guide the readers to [5].

There are two relevant works that do end-to-end machine learning based video compression. Wu et al. [9] encode the key frames and use a frame interpolation based technique. Rippel et al. [6] follow a traditional video coding pipeline and construct the building blocks using ML techniques. Our proposed approach is a different take on video coding and does not conform to the usual binary stream based pipeline followed by the aforementioned methods.

3. Methodology

We first formulate the building blocks of our approach in Sections 3.1 and 3.2 and then describe the pipeline in Section 3.3.

3.1. Deep Intrinsic Parameterization

We parameterize an image $y \in \mathbb{R}^{3 \times H \times W}$ as a deep neural network by fitting a generator network to $y$. Assuming a randomly initialized CNN $f_\theta$ as the generator network and
a random code tensor/vector \( z \in \mathbb{R}^{C \times H' \times W'} \), we get the desired parameterization by fitting the CNN to \( y \) using a suitable loss function like \( L2 \). We can simply write the procedure as below:

\[
E = ||y - f_\theta(z)||^2_2, \\
\theta^* = \arg \min_\theta E, \\
y^* = f_\theta(z)
\]

where \( \theta^* \) are the weights of network corresponding to a local minima and \( y^* \) is the desired parameterization. \( f_\theta \) could be any neural network in practice, but an added advantage of using a CNN is the implicit regularization that it provides in the form of high impedance to noisy data [8].

While the above described parameterization works well, it can be wasteful to parameterize a single image using all the weights of a high-capacity CNN. Li et al. [3] describe the intrinsic dimension of a network as the minimum number of parameters/constraints required by the network to do a task with a certain accuracy. Building upon the same idea, we can find the intrinsic dimension of our image parameterization and use that to train in a much smaller subspace of parameters.

For a network \( f_\theta \) with \( D \) trainable parameters, we represent the parameter vector in the parameter space as \( \theta^{(D)} \in \mathbb{R}^D \). To train the network in a smaller subspace of dimension \( d \), \( \theta^{(D)} \) is modified as:

\[
\theta^{(D)} = \theta_0^{(D)} + M v^{(d)}
\]

where \( M \) is a \( D \times d \) random projection matrix, \( \theta_0^{(D)} \) is some initialization of \( \theta^{(D)} \) and \( v^{(d)} \) is the trainable parameter vector in a smaller subspace. \( \theta_0^{(D)} \) and \( M \) are randomly generated once and kept frozen during the optimization. \( M \) is kept sparse using the method described in [4] to fit in the GPU memory. While columns of \( M \) can be explicitly orthogonalized, we rely on the approximate orthogonality of high dimensional random vectors, as also followed by [3]. With a suitably chosen \( d \) such that \( d < D \), we can achieve a desired reconstruction quality that can be quantified by a certain PSNR or SSIM score. This \( d \) would be the intrinsic dimension of our image parameterization.

Using this approach, we can lossily parameterize an image \( y \) using just \( v_t^{(d)} \) (\( d \) fp32 parameters). This \( d \)-dimensional encoding allows us to represent any image in \( d \times 4 \) bytes and decode it using the CNN architecture and the seed that was used for the random number generator.

3.2. Residual encoding

Deep intrinsic parameterization allows us to efficiently encode an image, but there is a lot of spatial and textural correlation among the frames of a GIF. This residual between frames may be modeled using fewer bits than the bits for a whole frame. Based on this assumption, if \( v_t^{(d)} \) and \( v_t^{(d)} \) represent the learned subspace vectors of frames \( t + 1 \) and \( t \) of a GIF with a continuous scene, we can write

\[
v_{t+1}^{(d)} = v_t^{(d)} + \delta_t^{(d)}
\]

\[
\delta_t^{(d)} = c k_t^{(d)}
\]

where \( c \) is a constant floating point scalar (hyperparameter) and \( k_t^{(d)} \) is a \( d \)-dimensional trainable integer vector whose entries are clamped to lie in \([-2^{l-1}, 2^{l-1} - 1]\), so that \( k_t^{(d)} \) can be represented in \( d \times l \) bits. This integer quantization is not differentiable in practice. Hence, we compute and apply the gradients with respect to floating point weights during backpropagation, cast them to integer and do the clamping during forward pass. This allows the optimization to happen with floating point weights but the \( k_t^{(d)} \) encoding is always a quantized integer vector.

Using this approach, we can encode a frame residual between two frames in \( d \times (l/8) \) bytes.

3.3. Pipeline

Given a GIF with \( N \) frames, we encode frame 1 of the GIF using 3.1 to get a \( d \)-dimensional fp32 encoding. We then use 3.2 to get a \( d \)-dimensional \( (l/8) \) bytes residual encoding of frame 2. The initial encoding plus residual encoding gives the full \( d \)-dimensional fp32 encoding of frame 2 which is then used for frame 3. This encoding happens sequentially on a per-frame basis to get \( N \) frame encodings in \( d \times 4 + (N - 1) \times (l/8) \) bytes.

For each frame, decoding can be done through a single CNN pass using a) the knowledge of CNN architecture and b) the pseudo-random number generator seed that will populate the weights of CNN, the random input tensor \( x \), and the sparse random projection matrix \( M \).

4. Experimental Protocol

4.1. Dataset

We will source a diverse dataset of GIFs from the internet, considering a) variation in spatial resolution, b) number of frames per GIF, and c) relative motion change between the frames.

4.2. Network Architecture and Hyperparameters

We will use a standard encoder-decoder CNN with skip connections as our network, similar to the one described in [8]. Our proposed approach is otherwise agnostic to the choice of CNN architecture. For a suitable value of \( d \), we will do an iterative search starting from \( d = 2500 \) and stop when the reconstruction reaches a reasonable metric value (PSNR/SSIM). For \( l \), we will start with a value of 4 and test other values in ablation studies. For Section 3.1, we will...
use $L_2$ loss function. The scalar $c$ can first be learned end-to-end for a sample pair of frames to get a scale estimate of $c$, and then kept fixed for all the sequences.

4.3. Experiments

**Performance Metrics.** For evaluating the performance of our approach, we will primarily compute SSIM and LPIPS [10], which are perceptual metrics averaged over all frames. We will also report the PSNR scores. We will compare against the method proposed by Wu et al. [9] and H.264 encoding over multiple bit-rates. [9] achieves compression quality better than H.264 codec.

**Ablative studies.** Since the bitrate can be adjusted either by varying the subspace dimension ($d$) or by varying the level of integer quantization ($l$), we will do ablation studies on a subset of the dataset and report the observed performance metrics, showcasing the relative importance of one over the other. We will also compare the effect of this ablation on GIFs of different motion complexities from the dataset.

**Robustness to noise.** We will also conduct an experiment by artificially adding noise to a raw sequence of frames before encoding it and compare the decoded result with the original source.

4.4. Expected outcomes and Significance

**Motion difference among frames.** We expect the metrics to be better for GIFs that are less dynamic than the ones with relatively more motion between frames. This should especially be evident for sequences where the scene changes drastically. The proposed method could easily be adopted for such GIFs by using keyframes but that is not in the scope of this paper.

**Varying compression factor.** We expect the quality of compressed GIFs to increase when we increase the subspace dimension ($d$) and/or decrease the level of integer quantization (by increasing $l$).

**Robustness.** We expect our method to be robust to noisy images compared to standard encoding as we are using CNNs to parameterize the images which have proven to be a good prior for natural images [8].

5. Preliminary results

In Figures 1 and 2, we show the results after deep intrinsic parameterization on sample images of size $256 \times 160$ and $384 \times 576$. Using a subspace dimension ($d$) of 5750 for the first image and 10000 for the second image, we are able to achieve a PSNR of 26 on both the images, while getting perceptually convincing compression results. For the same network architecture, we find that an image of higher spatial resolution requires a larger subspace to achieve a desired accuracy compared to lower resolution images. This implies that the intrinsic dimension required to parameterize an image directly varies as a function of the image resolution.

References


