Algorithm-Aware Neural Network Based Image Compression for High-Speed Imaging

Reid Pinkham
University of Michigan, Ann Arbor, MI
Facebook Reality Labs, Redmond, WA
pinkhamr@umich.edu

Tanner Schmidt
Facebook Reality Labs, Redmond, WA
tanner.schmidt@fb.com

Andrew Berkovich
Facebook Reality Labs, Redmond, WA
andrew.berkovich@fb.com

***Abstract*—In wearable AR/VR systems, data transmission between cameras and central processors can account for a significant portion of total system power, particularly in high framerate applications. Thus, it becomes necessary to compress video streams to reduce the cost of data transmission. In this paper we present a CNN-based compression scheme for such vision systems. We demonstrate that, unlike conventional compression techniques, our method can be tuned for specific machine vision applications. This enables increased compression for a given application performance target. We present results for Detectron2 Keypoint Detection and compare the performance and computational complexity of our method to existing compression schemes, such as H.264. We created a new high-framerate dataset which represents common scenarios for wearable AR/VR devices.**

Keywords*—*Video compression, convolutional neural networks, high-speed video, keypoint detection

I. Introduction and Background

Enabling real-time vision systems in power constrained, mobile systems such as AR/VR devices presents a significant challenge. This is largely due to the cost of transmitting and processing high-dimensional image data [1], [2]. Depending on system architecture and sensor interface, communication costs (energy/bit) vary greatly. Wired connections such as MIPI can consume $<$30 pJ/bit [3] while wireless interface such as WiFi or Low Energy Bluetooth (BLE) can consume orders of magnitude more energy (e.g. 407 nJ/bit [4]). Thus, data transmission for a 1 mega-pixel (MP) sensor running at 30 FPS can consume anywhere from $<$10mW for MIPI-like interfaces to nearly 100W for uncompressed, wireless interfaces.

This problem is compounded when operating image sensors at high frame rates as both datarate and power consumption of transmission scale linearly with frame rate. The benefits of high-speed image capture, namely reduced computational complexity and improved tracking performance, have been demonstrated for various computer vision (CV) applications in the context of AR/VR [5], [6]. However, leveraging these benefits in wearable mobile systems remains challenging due to bandwidth and power limitations. Data compression has become a widely adopted technique to reduce the cost of video transmission in both sensor systems and more broadly in communication systems. While there have been extensive previous works on video compression [7] and real time video compression [8], these algorithms are typically tuned for perceived visual image quality. For machine perception systems in which humans do not consume images, such image quality metrics are ill-equipped to drive the optimization of compression algorithms. We argue that compression algorithms should be optimized with the full machine perception pipeline in mind.

Similar to work presented by Mnih, *et al.* [9], we pose the problem of image compression as a control task in which we aim to determine a minimal set of pixels to transmit for each frame captured by a sensor. The sensor transmits the set of sparse pixel data to a receiver which builds a full-frame representation of the compressed image by simply updating pixel values in a frame buffer. Thus decompression only requires a simple memory update and no computation. Our approach leverages a neural network with a large receptive field to select regions of a scene with relevant information. In addition, our method enables either end-to-end optimization of image compression for a variety of loss functions—this includes performance of a particular task (e.g. tracking accuracy), or more traditional image quality metrics (e.g. mean squared error).

Our main contributions are as follows:

- We introduce a lightweight CNN-based method to select and transmit a subset of image data for each frame based on scene content and dynamics
- We show how our method can be tuned for performance of a particular downstream algorithm
- We demonstrate the feasibility of our method compared to a traditional video compression method (H.264)
- We introduce a new dataset of high-speed videos for training and benchmarking

II. Method

In this section we describe the operation and topology of our compression network. An hourglass style encoder-decoder network processes uncompressed frames to produce a pixel-wise decision map indicating which pixels should be transmitted over the link and which should not. Only the transmitted pixels are used to reconstruct an image at the receiver (or aggregator) as follows: (1) the first image captured is transmitted as a full frame, (2) subsequent images come in

Work completed during the first author's internship at Facebook Reality Labs

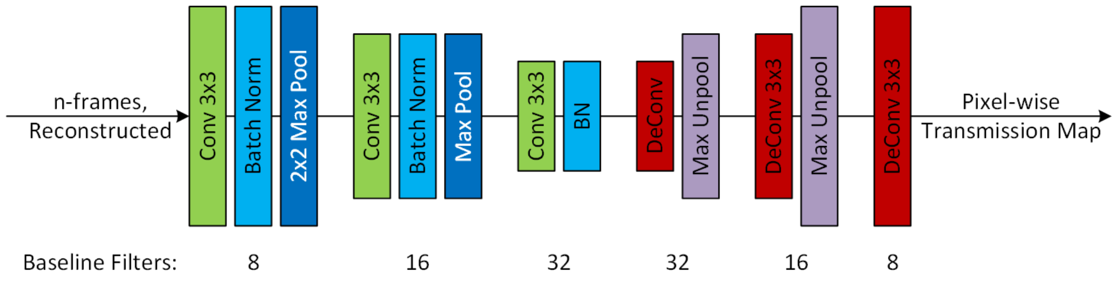

Fig. 1. Baseline network with three 3x3 convolution layers, and three 3x3 deconvolution layers. The ReLU activation is used after each Conv or DeConv layer. The final output is a pixel-wise confidence level to transmit each pixel.

the form of pixel updates, (3) new pixel data replaces the old values in the reconstructed image and pixels which are not updated remain unchanged. This method provides a continuous approximation of the full-frame image captured by the sensor.

Our baseline network topology is shown in Figure 1. As input. the network receives the most recent reconstructed frame and n previously captured full frames. The most recently reconstructed frame serves as a feedback mechanism so the network can identify and correct portions of the image which are degraded from compression. The final post sigmoid output of the network is a pixel-wise confidence that each pixel should be transmitted. These values are then thresholded to create a binary transmission map.

The baseline version of the network uses $n = 2$ as the number of full frames at the input. For a 512x512 input image, this results in 377 MOps/image, or 1440 ops/pixel. Max pooling and unpooling are used after each convolutional layer to reduce or increase the intermediate size, respectively.

During training, the reconstructed images use soft updates to create a differentiable approximation of the hard pixel update. Soft updates interpolate pixel values between their old and new values based on the confidence of the prediction.

In order for the network to understand the reconstructed input, it is trained on a series of frames we denote as rollouts. The length of these rollouts are increased throughout training so the network can learn to recognize and correct for artifacts. The learning rate was scaled inversely with the rollout length to reduce the emphasis of progressively longer single rollouts.

The loss function is formed by the sum of two competing terms: a sparsity penalty which is the average confidence of all pixels L_{spar} and the downstream or application loss L_{app}. For the baseline version, this is the mean squared error (MSE) between the reconstructed image and the ground truth image. These two losses compete against each other during training since achieving lower MSE drives the network to transmit more pixel data at the expense of a larger sparsity penalty. The values of W_{app}, W_{spar} are treated as hyperparameters and are fixed at the start of training. The final loss is as follows:

$$L = W_{app} \cdot L_{app} + W_{spar} \cdot L_{spar}$$

Our compression method has the flexibility and adaptability to train on other loss signals such as a backpropagated signal from a downstream algorithm. To demonstrate the tunability of our compression method, we use the state of the art Detectron2 [10] R-CNN model with the ResNet-50 back-end trained for Person Keypoint Detection[1]. The application loss with compressed images is computed against the pre-computed, ground truth, application output and backpropagated through our compression network. This allows our compression network to be tuned to increase compression while maintaining application performance.

We primarily focus on the effect of compression on the downstream algorithm performance and therefore do not consider traditional compression quality metrics. We use a sum of the relevant loss terms from the Detectron2 loss function including the bounding box and keypoint predictions.

To the best of our knowledge there is no publicly available dataset of high-framerate videos relavent to AR/VR systems. Along with this work we are releasing a representative high-framerate dataset of 512x512 greyscale video clips collected at a fixed framerate of 500FPS. Our dataset represents a diverse set of environments and camera dynamics. In total, our dataset contains over 378k frames representing 12.6 minutes of real-time data spread across 34 different scenes. We split it randomly into a training and test set at the per-clip granularity.

III. Ablation

In addition to our baseline network topology, we considered three similar designs and evaluated their performance when trained on MSE loss. We first introduced pixel age (time since last update) as input to the network. This helped the network identify regions which are "stale", increased total computation costs by 5%, and resulted in more fine-grained transmission maps with a slight increase in sparsity.

The other two network variants we considered either doubled the number of filters for each layer (Large) or doubled the number of filters while reducing the total depth to four (Shallow). Each of these networks increase the computational footprint of the compression network and were used to evaluate if an increased network capacity improved performance.

All networks display similar performance metrics (Table I). Versions of the network with similar MSE loss were chosen for comparison. All but the large configuration achieve greater than 91% sparsity. The shallow network performed the best

[1]We take Detectron2 as an off-the-shelf algorithm and do not modify or tune the weights for our application. A co-optimization between the downstream algorithm and our compression network may be beneficial, but it outside the scope of this work.

TABLE I. PERFORMANCE OF DIFFERENT NETWORK TOPOLOGIES.

	MOps	Depth	MSE	Sparsity
Baseline	377	6	$2.05 \cdot 10^{-4}$	91.1%
Age	396	6	$2.03 \cdot 10^{-4}$	91.3%
Shallow	755	4	$2.02 \cdot 10^{-4}$	91.5%
Large	1359	6	$2.06 \cdot 10^{-4}$	88.4%

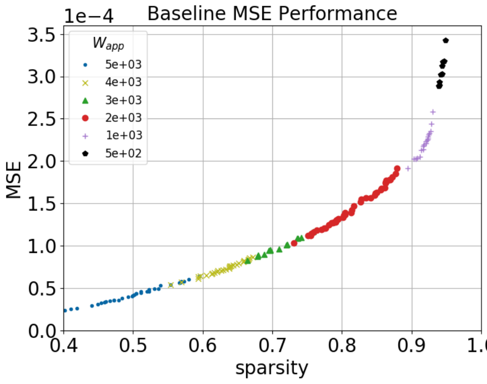

Fig. 2. MSE performance of the baseline network when trained with various values of W_{app}. Each datapoint represents a point in training which forms the performance frontier.

overall, with 91.5% sparsity and the lowest of the selected MSE losses.

We also studied the effect of the weighting between W_{spar} and W_{app}. This proved to be a good way to push the sparsity towards a particular target level. Figure 2 shows how we can adjust performance of the baseline network by varying W_{app} from 500 to 5000 with a fixed value of $W_{spar} = 1.0$.

IV. PERFORMANCE

Across the dataset, the baseline network consistently achieves sparsity levels above 90% with minimal degradation of keypoint tracking performance. Sample transmission maps and reconstructed frames are shown in Figure 3.

A. Power Analysis

Net energy reduction resulting from video compression heavily depends on the cost (energy/bit) of transmitting data from the sensor, and the cost (ops/W) of compute. As an example, we can assume an efficiency of processing on par with recent CNN edge-accelerators, such as the edge-TPU, which have efficiencies around 2 TOPS/Watt [11]. Our compression method begins to save energy when the transmission cost exceeds 100 pJ/bit, which is approximately the cost of high-speed wired interconnects such as LVDS/MIPI. Nearly all wireless interfaces, such as WiFi and BLE, consume orders of magnitude more energy per bit.

B. DVS-style encoding

We formulate video compression as a control task for selecting a subset of pixel data to transmit for each frame. One of the simplest selection mechanisms is based on a pixel-wise temporal contrast. The decision to transmit a pixel is based on whether the temporal derivative of a pixel's output value exceeds some fixed threshold. Variants of this compression scheme have been implemented [12] and are generally referred to as Dynamic Vision Sensors (DVS). In our implementation of DVS-style compression we compute temporal contrast as the absolute difference between pixel values in the current frame and pixel values stored in the reconstructed frame buffer (generated by pixel updates). We transmit pixel values only when temporal contrast exceeds a fixed threshold.

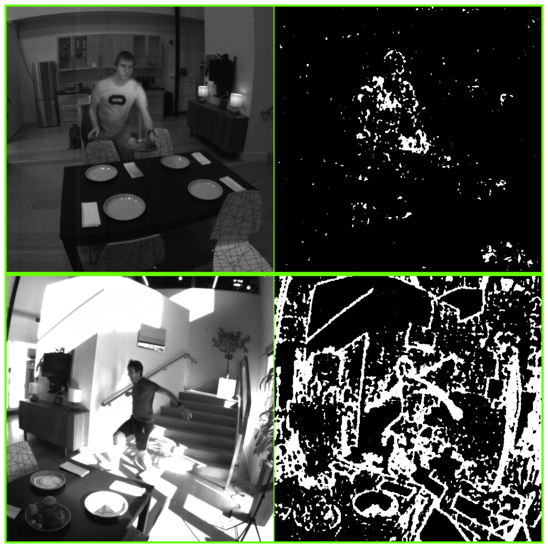

Fig. 3. Examples of transmission maps (right) and their corresponding reconstructed frames (left). White regions indicate pixels which were transmitted. The top/bottom row show scenes with a static/dynamic camera position.

TABLE II. COMPARISON OF JOINT TRACKING PERFORMANCE AT A SPARSITY OF ≈94%. LOSS: TOTAL DETECTRON2 LOSS [10]. BOX/KEYPOINT: ERROR IN BOUNDING BOX/KEYPOINT PREDICTION.

	Loss	Box	Keypoint	MSE	Sparsity
Baseline	15.36	0.34	14.36	$2.93e^{-4}$	94.0%
DVS	15.01	0.34	14.02	$1.98e^{-4}$	94.2%
Baseline-D2	14.47	0.31	13.55	$5.35e^{-4}$	94.4%

C. Optimizing for Keypoint Detection

We optimize the baseline network for use with the Detectron2 keypoint detection network by changing L_{app} to include accuracy of keypoint detection instead of MSE. We refer to this retrained version of the baseline network as Baseline-D2. Table II shows performance of the baseline network before and after being optimized for keypoint detection. By changing our loss function, the performance for all Detectron2 loss signals decreased. Interestingly the Detectron2 application performs noticeably better with the Baseline-D2 network, even though the DVS-style encoding scheme achieves less than half the MSE. Similarly, the Baseline-D2 network has the highest MSE of any method we evaluated with our dataset. This indicates that optimizing the network for keypoint detection allows it to identify the most essential pixels for the task and ignore those in regions not relevant to the detection task.

D. Comparison with H.264 encoding

H.264 encoding is currently the most widespread method to encode raw video. It supports both variable quality, target bitrate, and computational complexity. H.264 encoding makes use of spatial and temporal transforms to compress video to

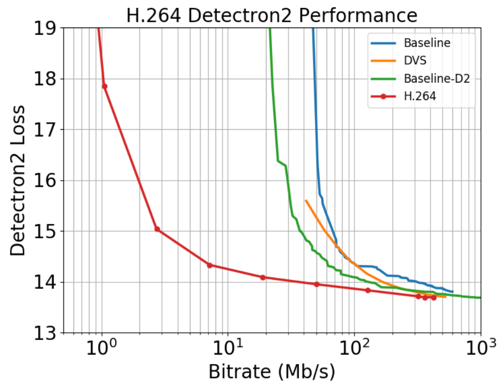

Fig. 4. H.264 encoding performance with Detectron2. The right side of the graph corresponds to uncompressed data. 94% sparsity ≈ 60 Mb/s.

very high levels. We encoded our test set using the target bitrate mode. This allowed us to sweep across multiple performance points of H.264 encoding and understand how it performs in terms of MSE as well as Detectron2 loss. The performance across the simulated range is shown in Figure 4. We estimate the rough computational cost of H.264 encoding using the FFMPEG [13] tool by measuring the time to encode a video. At the same bitrate as our baseline, H.264 encoding took 348 operations per pixel. Compared to our baseline method which uses 1440 Ops/pixel.

Both our method and H.264 encoding are expected to use similar amounts of energy for encoding. H.264 requires high precision operations for its Discrete Cosine Transform operation which uses 16-bit values [14]. Quantization of neural networks has been shown to be effective down to 8-bit precision [15]. Additionally, the decoding process for H.264 is nearly as complex as encoding which effectively doubles the number of operations required per pixel, while our decoding method only requires a memory update. Taking both precision and decoding differences between our method and H.264 into account, the total energy cost is similar.

V. Discussion and Future Work

The compression method presented here uses a very simple compression mechanism driven by an intelligent per-pixel prediction. Fundamentally, this simple compression mechanism has its limitation. By only compressing pixel-level information in the time domain we cannot utilize spatial compression and can only choose to disregard certain regions of the image or video. This forms a key difference between conventional video compression methods such as H.264 and ours. This fundamental limit manifests itself as the sharp degradation in performance as our compression method is pushed to very high levels of sparsity. This limit will be different for all applications, but will always be at a lower sparsity than a method which can effectively compress both temporally and spatially. However, because this compression scheme relies on simple and highly parallelizable operations, it is uniquely well-suited for high-speed video applications. We expect that co-optimization of algorithm and compression will push the performance of our method closer to that of H.264 performance.

Previous efforts to create a CNN based compression scheme, such as DVC [16], have seen good success and offered competitive results compared to H.264 encoding. However, these compression methods require three or more orders of magnitude more computation than H.264. DVC requires 409k Ops/pixel, compared to our method requiring 1.4k Ops/pixel. This difference in computation illustrates the advantage of our method and makes it practical for low-power AR/VR systems.

VI. Conclusion

Both performance and complexity of various real time AR/VR applications benefit from high framerate video streams. However, transmitting these video streams across wired and wireless interfaces consumes a significant amount of energy. This necessitates video compression prior to transmission.

In this paper, we present a lightweight CNN based method to perform real time video compression. This new compression technique can be fine tuned for a particular downstream processing algorithm. Using our simple compression network, we are able to compress high-speed video tenfold to provide a net energy savings across common wired and wireless interfaces.

Acknowledgment

The authors kindly acknowledge the input and helpful feedback from Hans Reyserhove, Syed Shakib Sarwar, Chiao Liu, Richard Webb, and our other colleagues at FRL.

References

[1] Liu *et al.*, “Intelligent Vision Systems – Bringing Human-Machine Interface to AR/VR,” in *IEEE IEDM*, 2019, pp. 218–221.

[2] Horowitz, “Computing’s energy problem (and what we can do about it),” in *IEEE ISSCC*, Feb 2014, pp. 10–14.

[3] Lee and Jang, “A 6.84 Gbps/lane MIPI C-PHY Transceiver Bridge Chip with Level-dependent Equalization,” *IEEE Trans. on Circuits and Systems II: Express Briefs*, pp. 1–1, 2019.

[4] Siekkinen *et al.*, “How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4,” in *IEEE WCNC*, April 2012, pp. 232–237.

[5] Galoogahi *et al.*, “Need for speed: A benchmark for higher frame rate object tracking,” *CoRR*, vol. abs/1703.05884, 2017.

[6] Kowdle *et al.*, “The Need 4 Speed in Real-Time Dense Visual Tracking,” *ACM Trans. Graph.*, vol. 37, no. 6, Dec. 2018.

[7] Tabatabai *et al.*, “Motion Estimation Methods for Video Compression—A Review,” *Journal of the Franklin Institute*, vol. 335, no. 8, pp. 1411 – 1441, 1998.

[8] Westwater and Furht, *Real-time video compression: techniques and algorithms*. Springer, 2007, vol. 376.

[9] Mnih *et al.*, “Recurrent models of visual attention,” in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[10] Wu *et al.*, “Detectron2,” https://github.com/facebookresearch/detectron2, 2019.

[11] Edge TPU performance benchmarks. Google LLC. [Online]. Available: https://coral.ai/docs/edgetpu/benchmarks/

[12] Lichtsteiner *et al.*, “A 128 x 128 120dB 30mW asynchronous vision sensor that responds to relative intensity change,” in *IEEE ISSCC*, 2006, pp. 2060–2069.

[13] ffmpeg Developers. ffmpeg tool. [Online]. Available: https://ffmpeg.org/

[14] Wiegand *et al.*, “Overview of the H.264/AVC video coding standard,” *IEEE TCSVT*, vol. 13, no. 7, pp. 560–576, 2003.

[15] Han *et al.*, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[16] Lu *et al.*, “Dvc: An end-to-end deep video compression framework,” in *IEEE CVPR*, 2019, pp. 11 006–11 015.