PERFORMANCE OF TEXTURE COMPRESSION ALGORITHMS IN LOW-LATENCY COMPUTER VISION TASKS

Jakub Žádník, Markku Mäkitalo, Jussi Iho, Pekka Jääskeläinen

Tampere University, Finland
{jakub.zadnik, markku.makitalo, jussi.iho, pekka.jaaskelainen}@tuni.fi

ABSTRACT

Deep learning has been successfully used for computer vision tasks, but its high computational cost limits the adoption in lightweight devices such as camera sensors. For this reason, many low-latency vision systems offload the inference computation to a local server, requiring fast (de)compression of the source images. Texture compression is a compelling alternative to existing compression schemes, such as JPEG or HEVC, due to its low decoding overhead, straightforward parallelization, robustness, and a fixed compression ratio. In this paper, we study the impact of lightweight bounding box–based texture compression algorithms, BC1 and YCoCg-BC3, on the accuracy of two computer vision tasks: object detection and semantic segmentation. While JPEG achieves superior per-pixel error rate, the YCoCg-BC3 encoding can provide comparable vision accuracy. The BC1 encoding results in significant degradation of vision performance. However, by retraining the FasterSeg teacher network with a BC1-compressed dataset, we reduced its segmentation mIoU loss from 2.7 to 0.5 percent. Thus, both BC1 and YCoCg-BC3 encoders are suitable for use in low latency vision systems, since they both achieve significantly higher encoding speed than JPEG and their decoding overhead is negligible.

Index Terms— Image Compression, Computer Vision, Texture Compression, Low Latency

1. INTRODUCTION

Computer vision is a necessary component of latency-oriented applications such as autonomous driving or industrial control. The growing trend of multi-access edge computing [1] enables to use lightweight devices as "dumb sensors" connected via a high-speed network, such as 5G, to edge servers where the neural network (NN) inference takes place. For this purpose, compression can be necessary to limit the signal bandwidth and speed up the data transfers.

Figure 1 illustrates a compression ratio required to achieve 10, 20, and 50 ms end-to-end latency in a control loop with a FasterSeg semantic segmentation network [2] running on an edge server at 163.9 frames per second (FPS) (as reported by the authors). The horizontal stripes represent compression ratios of texture compression formats and a typical compression ratio range of joint photographic experts group (JPEG) with quality parameters between 20 and 100 (blue region). The diagonal lines represent a bitrate and a compression ratio when all the time remaining from FasterSeg inference is spent on a network transfer, without any room for (de)compression latency or the latency of the network itself. To meet a given latency budget, the compression ratio or the bitrate must increase by an increment dictated also by the encoding/decoding time of the (de)compression algorithm. Thus, fast (de)compression is essential for transferring over low-bitrate channels or preventing quality degradation due to strict compression ratio requirements.

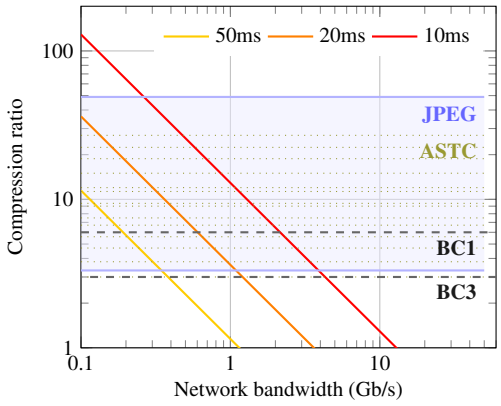

Fig. 1. Theoretical minimum compression ratio required to achieve an end-to-end latency of 10, 20 and 50 ms assuming a FasterSeg network in the loop.

Most video and image compression algorithms are optimized for human perception which, however, might not necessarily be optimal for computer vision algorithms [3]. Recently, adapting a compression algorithm for machine perception has been a subject of multiple studies. However, the works concentrate mostly on JPEG [3–5] or high efficiency video coding (HEVC) algorithms [6], or designing a custom algorithm to be integrated as an application-specific integrated circuit (ASIC) [7].

JPEG [8] is a popular image compression format often used for its simplicity. Compared to JPEG and other codecs based on frequency transform and entropy coding, such as HEVC, texture compression has the advantage of a fixed compression ratio, and therefore a random access property where each pixel can be addressed directly from the compressed representation. Furthermore, texture decompression is an extremely fast process, further accelerated with specialized hardware on virtually all modern graphics processing units (GPUs) to allow fast mapping of textures on a rendered scene. On the other hand, compressing into modern texture formats, such as adaptive scalable texture compression (ASTC) [9], is usually more complex and time-consuming than JPEG encoding if high quality is desired.

To assess the practicality of using texture compression in low latency computer vision pipelines, this paper compares real-time BC1 and YCoCg-BC3 texture compression algorithms and an offline high-quality ASTC encoder against the well-known JPEG format in terms of computer vision performance. We evaluate the computer

vision performance on two common computer vision tasks using state-of-the-art real-time neural networks: object detection with a YOLOv4 network [10] and semantic segmentation with a FasterSeg network [2]. To verify the low-latency potential of the bounding box–based BC1 and YCoCg-BC3 encoding algorithms, we provide their runtime measurements and demonstrate their low complexity against the ASTC encoder. Furthermore, we show that in the case of the more complex FasterSeg teacher network, the loss of vision accuracy caused by BC1 compression can be recovered from 2.7 to 0.5 percentage points (pp) by retraining the network with a compressed dataset.

2. TEXTURE COMPRESSION

BC1 [11] can be considered the most basic texture compression format. The encoder splits an image into 4×4 blocks and selects two endpoints representing the two "most opposite" colors in the block. Each pixel is then represented as a 2-bit weight pointing at either one of the two endpoints or one of two values linearly interpolated equidistantly between the two endpoints. Each BC1-compressed block occupies 64 bits, therefore its compression ratio is 6, assuming an RGB source image with 8 bits per channel.

A similar principle is used in other BCn formats, such as BC3, which compresses an alpha channel into its own 64-bit block on top of a BC1 block with compressed RGB channels. A YCoCg-BC3 modification of this format [12] exploits the human vision being most sensitive to luminance by storing the Y channel into the high-precision alpha block while the Co and Cg channels are encoded with lower precision in the BC1 block. By using two 64-bit blocks, the compression ratio of (YCoCg-)BC3 is only 3.

The core of the ASTC [9] format also follows the endpoint-based principle but expands on the number of possible configurations. For example, it supports block partitioning where each partition has its own set of endpoints, which allows resulting colors to reach beyond a line in a color space. Also, ASTC allows adjusting the quantization precision tradeoff between the endpoints and weights, which is a fixed parameter in the BC1 and BC3 formats. A unique feature of ASTC is the support for different input block sizes from 4×4 to 12×12 pixels. Since the encoded block has always 128 bits, by scaling the input block size it is possible to adjust the rate-distortion tradeoff. More specifically, the achievable compression ratio of ASTC ranges from 3 to 27 on RGB images with 8 bits per channel.

Apart from the endpoint-based BCn and ASTC formats, various other texture compression formats exist, such as ETC1, ETC2, or PVRTC. We chose the BC1 format since it is the simplest widely used format and faster to encode than ETC1 or ETC2 [13]. YCoCg-BC3 allows for superior quality in comparison to BC1, while retaining similar simplicity. The third evaluated format, ASTC, provides us one of the highest quality achievable with existing texture compression formats at respective bitrates and also reaching lower bitrates comparable with JPEG at lower quality levels.

Unlike traditional image/video formats mentioned earlier, texture compression specifies only the data layout of the encoded block while the encoding procedure depends on the heuristics of a particular encoder. Thus, the encoding complexity varies significantly based on the required quality vs. speed tradeoff. The following section explains the choice of the evaluated algorithms.

3. EXPERIMENTAL SETUP

We implement fast GPU-based BC1 and YCoCg-BC3 algorithms from [12] on an Intel UHD 620 GPU integrated within a laptop processor instead of a discrete GPU in order to emulate a resource-constrained scenario. BC1 and YCoCg-BC3 offer only one bitrate level and encoding configuration, therefore, the measurement consists of only a single data point for each. As the low-latency counterpart of BC1 and YCoCg-BC3, we use a GPU implementation of JPEG: `GPUJPEG` compression from [14][1]. `GPUJPEG` was evaluated at quality levels 20, 30, ..., 90, 95 and 100 without subsampling.

Since the compression ratio of both BC1 and YCoCg-BC3 is fixed, we also evaluate different block sizes and presets of a single instruction multiple data (SIMD)-accelerated central processing unit (CPU) encoder of the ASTC format `astcenc`[2], version 2.3. The ASTC encoder was evaluated for three quality presets (fastest, medium and exhaustive) and different input block sizes (4×4, 6×6, 8×8, 10×10 and 12×12). While the fastest preset is optimized for maximum speed, the exhaustive preset is optimized for maximum quality. In practice, the fastest and exhaustive presets are rarely used in production due to low quality and low encoding speed, respectively. However, for our experiments, they provide absolute bounds of the encoder's performance.

For evaluation, we consider two computer vision tasks commonly used in real-time scenarios:

Object Detection YOLOv4 [10] was used as a real-time object detection framework evaluated on the validation set of the COCO dataset, version 2017 [15]. It is worth noting that the COCO dataset contains images already compressed into a lossy JPEG format (the average quality parameter of training and validation images is approximately 94). However, as they have been used in this form to train the YOLOv4 network, they can be treated as the ground truth. The vision performance results are presented as mean average precision for mean intersection over union (mIoU) 50–95% ($\mathrm{mAP}_{0.50-0.95}$).

Semantic Segmentation For the purpose of semantic segmentation we chose a real-time FasterSeg network [2] trained on a high-resolution (2048×1024) Cityscapes dataset [16]. Unlike COCO, the Cityscapes dataset only contains images compressed losslessly. The FasterSeg network consists of two networks: lightweight student (3.4M parameters) and complex teacher (22M parameters), with a mIoU measured separately for each. During training, the student distills the learned knowledge from the teacher network, but also optimizes for the lowest possible latency, unlike the teacher network. Thus, we used the student network for evaluating the vision performance, as it is the one that would be deployed in a real scenario. However, the teacher network is later used in the retraining experiment, as its higher number of parameters represents higher learning capability and allows us to estimate the maximum possible gain achievable with retraining.

4. EVALUATION

4.1. Computer Vision Performance

Figure 2 shows the $\mathrm{mAP}_{0.50-0.95}$ of the YOLOv4 network on a validation set of the COCO dataset for the following formats: ASTC, BC1, YCoCg-BC3 and JPEG with configurations described in the previous section. The bitrate is expressed as bits per pixel (bpp), where in the case of JPEG, the bitrate was averaged over the whole

[1] `https://github.com/CESNET/GPUJPEG`

[2] `https://github.com/ARM-software/astc-encoder`

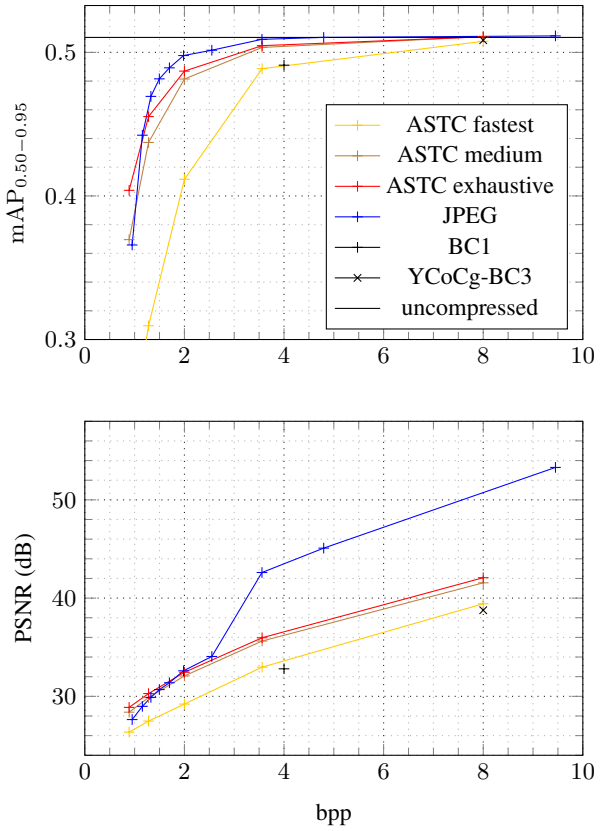

Fig. 2. Mean average precision for intersection over union 50–95% of YOLOv4 (top) and PSNR (bottom) of a COCO validation set compressed with different methods.

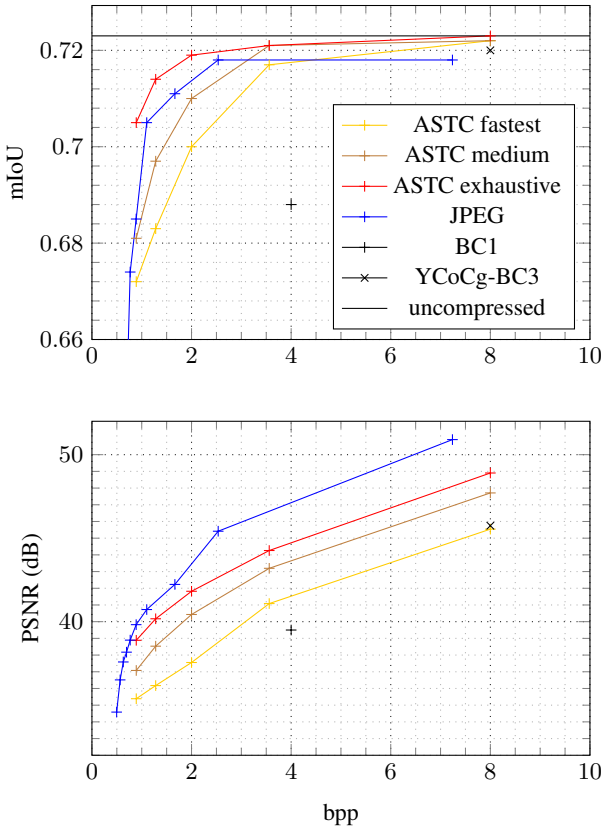

Fig. 3. Mean intersection over union (mIoU) of a FasterSeg student network (top) and PSNR (bottom) of a Cityscapes validation set compressed with different methods.

dataset at each quality level, while the texture compression formats have a constant compression ratio at a given block size. The bottom plot shows the pixel-wise error between the uncompressed and compressed COCO validation set expressed as peak signal-to-noise ratio (PSNR).

Similar comparison is shown in Figure 3 for the mIoU of a FasterSeg student network (top) and PSNR of the validation set of the Cityscapes dataset (bottom). The FasterSeg teacher network reached mIoU 1–2 pp higher in our experiments.

On the object detection task, JPEG outperforms all texture compression algorithms in terms of $\mathrm{mAP}_{0.50-0.95}$, except at the lowest bitrates where the quality degrades faster than the higher ASTC presets. YCoCg-BC3 compression achieves $\mathrm{mAP}_{0.50-0.95}$ only 0.24 pp lower than the uncompressed dataset and BC1 compression degrades the same metric by 1.94 pp. The PSNR results show an opposite trend towards higher bitrates: while the detection precision converges towards the uncompressed level, the PSNR difference increases.

On the semantic segmentation task, JPEG shows a maximum achievable mIoU lower than all ASTC variants and YCoCg-BC3 at high bitrates. YCoCg-BC3 achieves only 0.3 pp less mIoU than uncompressed while BC1 compression degrades the segmentation mIoU by 3.5 pp. The PSNR results, on the other hand, show superior JPEG performance across the whole bitrate range. Interestingly, the exhaustive ASTC preset achieves higher mIoU than JPEG on all bitrates while its PSNR is always lower.

The results show that in terms of only computer vision performance, texture compression algorithms compete better against JPEG towards the higher end of the bitrate spectrum, and also perform better than would be apparent from the PSNR results. BC1 yields inferior performance in all evaluated quality metrics but achieves the fastest compression speed. One factor contributing to the high quality of YCoCg-BC3 is the high precision of the luminance channel which plays an important role for both machine and human perception [3]. Both ASTC and BC1 operate in RGB color space which might be a limiting factor of these formats.

4.2. Retraining

BC1 compression causes a large decrease in computer vision performance, especially for semantic segmentation. However, its fast compression speed and higher compression ratio than YCoCg-BC3 still make it an attractive choice when ultra-fast encoding is required. For this reason, we investigated how much quality could be restored by retraining the FasterSeg network with a BC1-compressed training set of the Cityscapes dataset. We focused only on the teacher network since its higher complexity represents a more achievable learning capability than the student network, which also optimizes for the lowest latency and reduces the number of learnable parameters more than five times.

Figure 4 shows two crops from the Cityscapes validation set (a) with their ground truth segmentation (b). The BC1 compression

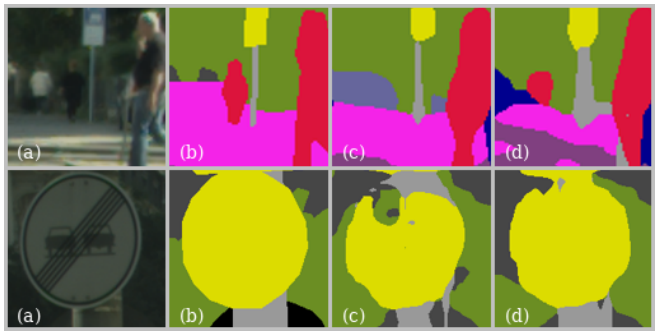

Fig. 4. (a) Two crops from a Cityscapes validation set, (b) their ground truth segmentation, (c) segmentation with a BC1-compressed input by a pretrained FasterSeg teacher network and (d) segmentation with a BC1-compressed input by the same network trained on the BC1-compressed dataset.

causes a missed detection of a person dressed in black (b, top) and a noticeable gap in the traffic sign segmentation (b, botton) when evaluated on the pretrained FasterSeg teacher network. Segmentation by the network retrained on a BC1-compressed dataset at least partially recognizes the missing person (d, top) and restores the original traffic sign shape (d, bottom).

The retraining results are summarized in Table 1. During a 600-epoch training run, the highest validation mIoU was 73.0% at the epoch 590 which is 0.5 pp less than the mIoU achieved with an uncompressed dataset on the pretrained network. Therefore, it is possible to largely overcome artifacts introduced by BC1 compression with retraining.

Table 1. Segmentation accuracy of the FasterSeg teacher network.

compression (eval)	compression (train)	mIoU
BC1	none (pretrained)	70.8 %
BC1	BC1	73.0 %
none	none (pretrained)	73.5 %

4.3. Runtime

Depending on the task and bitrate, ASTC can rival JPEG in terms of computer vision performance. However, while JPEG can be encoded in real time even at 8K resolution [14], the fastest ASTC configuration in our experiments (12×12 with fast preset) still took about 20 ms on average to compress a single Cityscapes image (2048×1024) on a 32-core processor (AMD Threadripper 2990WX) with AVX2 SIMD acceleration.

To verify the runtime performance of BC1 and YCoCg-BC3 algorithms on resource-constrained devices, we reimplemented them on an Intel laptop integrated GPU and measured their encoding speed. The used integrated GPU (Intel UHD 620) has 192 processing elements (PEs) with limited cooling options compared to a typical desktop GPU which can have thousands of PEs (for example NVIDIA GTX 1080 used in [17] has 2560 PEs). The results, together with comparison to related work, are shown in Table 2 and do not include memory transfers from/to a GPU (unless noted otherwise). Since different implementations work with different resolutions, as a common metric we extrapolated the throughput reported for each method, and used it to calculate a hypothetical encoding time it would take the encoder/decoder to process an 8K frame (denoted as T_{8K}).

In our case, the difference between YCoCg-BC3 and BC1 encoding speed is more than 2.7×, although [14] reports a smaller difference on a desktop GPU. From [14] it can also be seen that JPEG encoding on a GPU is about 2.2–2.6× slower than BC1 or YCoCg-BC3 on the same GPU. However, JPEG decompression is even slightly slower than the compression, while texture decompression can be faster by more than an order of magnitude. In our experiments using NVIDIA RTX 2070 GPU, rendering of an 8K frame compressed as BC1 and YCoCg-BC3 took 0.53 and 0.54 ms, respectively, with the latter including the YCoCg → RGB restoration in a fragment shader.

Table 2. Encoding/decoding time of 8K resolution (T_{8K}, milliseconds) and compression ratio (CR) of different coding methods.

method		device	CR	T_{8K}
[12]**	BC1 enc	Intel	6	9.9
	YCoCg-BC3 enc	UHD 620	3	26.8
[12]**	BC1 dec	NVIDIA	6	0.53
	YCoCg-BC3 dec	RTX 2070	3	0.54
`astcenc`	ASTC enc (12×12 fastest)	AMD 2990WX	27	323*
[14]	BC1 enc	NVIDIA GTX 580	6	9.5
	YCoCg-BC3 enc		3	11.3
	JPEG Q95 enc		~4.8	25.1
	JPEG Q95 dec		~4.8	31.3
[13]	BC1 enc	Intel Xeon E5-2699	6	4.5*
	ETC1 enc		6	9.0*
	ETC2 enc		6	11.7*
[17]	JPEG XS+ dec	NVIDIA GTX 1080	2.4†	20.9*
			2.4	26.1*

* Extrapolated when direct 8K results are not available
** Our implementation + Including CPU → GPU memory transfer
† Input with 4:2:2 subsampling

4.4. Complexity Analysis

To illustrate the complexity difference between the evaluated texture compression algorithms, we implemented single-threaded CPU versions of the BC1 and YCoCg-BC3 encoders and compared them to the `astcenc` encoder built with any manual SIMD vectorization disabled and running on a single thread. By purposefully disabling both multi-threading and manual SIMD optimizations it is easier to compare the relative complexity between encoders (the compiler is still allowed to auto-vectorize). Figure 5 shows the average scalar encoding rate in megapixels per second of five `astcenc` presets (black lines), a custom "restrict" preset forcing the `astcenc` encoder to always use only a single encoding configuration (blue), BC1 (+) and YCoCg-BC3 (×). To reduce the variance between datasets, we composed a custom 64-image dataset containing the Kodak dataset [18] as well as a random selection of images from the COCO and Cityscapes validation sets. All benchmarks were performed on a single thread of AMD Threadripper 2990WX.

The comparison shows major differences between the encoding rates: Both BC1 and YCoCg-BC3 are more than an order of a magnitude faster than the fastest `astcenc` preset. Furthermore, the difference between the fastest and exhaustive presets is between two and three orders of magnitude. At the exhaustive preset, `astcenc` per-

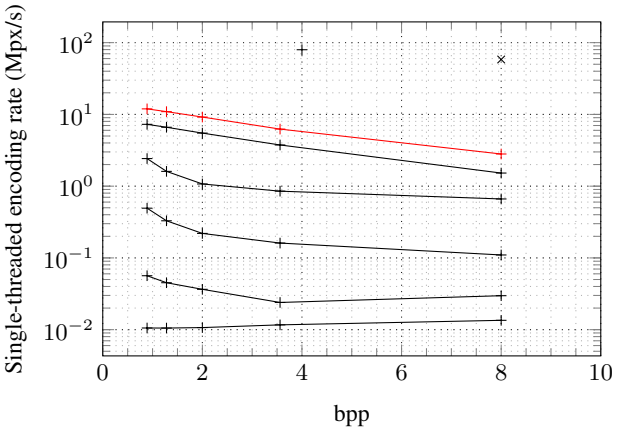

Fig. 5. Encoding rate in megapixels per second of five `astcenc` presets (black line, from bottom to top: exhaustive, thorough, medium, fast and fastest) a custom "restrict" preset (red line, top) at five block sizes (from left to right: 12x12, 10x10, 8x8, 6x6 and 4x4), BC1 (+) and YCoCg-BC3 (×), running on a single core.

forms the most thorough search of the possible configuration space. Scaling down the preset, the encoder is allowed to only search a more and more limited subset of the configuration space. At the fastest preset, the encoder only chooses between a handful of configurations. Further modifying the encoder to only use one configuration ("restrict" preset, shown as a blue line in Fig. 5) and skipping the algorithm for choosing the color endpoint mode (which is redundant in case of only one option), we managed to further decrease the encoding time by over 60%.

The encoding time achieved with the "restrict" preset does not include any configuration space search and most of its runtime is spent calculating the weights' alignment and a refinement pass. At this point, the only way to speed up the calculation is to replace the algorithms with faster variants. Both BC1 and YCoCg-BC3 encoders use a bounding box method for the endpoint estimation which simply selects the minimum and maximum color values within a block and does not rely on the weight alignment step: The weights are directly selected based on their distance to the quantized endpoint values. The refinement pass is also skipped entirely. Combined together, the simplified encoding method leads to more than an order of magnitude improvement of the encoding speed in comparison to the artificial "restrict" preset.

5. RELATED WORK

Compression for Computer Vision Several publications proposed optimizing JPEG for NN inference. In [3], the authors proposed the GRACE method, for optimizing the RGB–YUV transform and quantization of the JPEG algorithm by offline probing a target NN without modifying it. The GRACE method achieved accuracy higher than retraining with unmodified JPEG dataset. [4] and [5] also optimize the JPEG for NN inference by focusing on the quantization table.

In [6], the authors optimized HEVC for machine perception by dynamically adjusting the quality based on a fast saliency map computation. They achieve an encoding speed of ∼10–40 ms per 1080p frame. The authors of [19] optimized the parameters of a low-latency JPEG XS [20] encoder for semantic segmentation task, leading to a significant bitrate reduction at the same accuracy. In [7], the authors propose an ASIC circuit for frame memory compression in the HEVC algorithm optimized for computer vision.

To the best of our knowledge, no work has been published about the effects of texture compression on machine perception.

A different approach from compressing images and tailoring the compression for machine perception is to compress the intermediate NN features [21–23]. Finally, to bridge the gap between the compression for human and machine vision, the moving pictures experts group (MPEG) has launched an exploration on a new paradigm of video coding for machines (VCM), optimizing for both targets in tandem [24, 25]. Both approaches were shown to outperform traditional coding methods in terms of coding efficiency. However, in this work, we focus on the former approach of compressing images, as it is the most common method of supplying inputs to the neural network and does not require any additional architectural changes.

Runtime Measurements Runtime results of GPU execution times of the BC1 and YCoCg-BC3 compression algorithms were published in the original publication [12]. The same algorithms were evaluated in [14] on 1080p, 4K, and 8K resolutions. Real-time BC1 encoding was also implemented on a server-grade CPU as a part of an in-home streaming system [13]. The authors also present their implementation of ETC1 and ETC2 encoders. The ASTC encoding had usually been considered too slow for even real-time purposes. However, in [26], the authors replace the complex configuration search by a NN inference, accelerating the computation by up to 10×. To compare with other types of compression methods, in [17] the authors evaluate the decoding speed of a JPEG XS decoder, including memory transfer times.

6. CONCLUSION

We evaluated the computer vision performance of low-complexity BC1 and YCoCg-BC3 encoders. The results of YCoCg-BC3 encoding show accuracy comparable to both ASTC and JPEG, despite reaching lower PSNR than both. Despite BC1 compression reaching very low accuracy, especially on the semantic segmentation task, retraining results on the FasterSeg teacher network suggest its compression artifacts can be mostly recovered: The mIoU after retraining is only 0.5 pp lower than on the uncompressed dataset, compared to 2.7 pp without retraining.

A modern ASTC encoder, `astcenc` can slightly outperform JPEG in computer vision performance, depending on the task, bitrate and encoding preset. However, ASTC encoding is very slow due to its massive configuration search space and complex algorithms, making it impractical for real-time use. Compared to even the fastest preset, both BC1 and YCoCg-BC3 encoding was faster by more than an order of magnitude when benchmarked on a single thread without vectorization. Previous work also shows that identical GPU-based BC1 and YCoCg-BC3 encoders are faster than a GPU-based JPEG encoder. The texture compression formats, however, reach more than an order of magnitude higher decoding speed, thus significantly outperforming JPEG.

A further study focused on reducing the configuration space and simplification of its core algorithms would allow leveraging the main advantage of the format: the ability to scale its input block size to enable more bitrates, which is a unique property among texture compression formats. Our future work includes adapting simple encoding techniques from BC1 and YCoCg-BC3 to the versatile ASTC format, unlocking lower achievable bitrates than 6 and 3 bits per pixel, respectively. The results of retraining the segmentation network with a BC1-compressed dataset suggest that a significant part of the lost quality can be recovered via retraining.

7. ACKNOWLEDGMENTS

The work was financially supported by the Tampere University ITC Graduate School. This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783162. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Czech Republic, Finland, Spain, Italy. It was also supported by European Union's Horizon 2020 research and innovation programme under Grant Agreement No 871738 (CPSoSAware).

8. REFERENCES

[1] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *Communications Surveys & Tutorials*, vol. 19, no. 3, 2017.

[2] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, "FasterSeg: Searching for faster real-time semantic segmentation," *arXiv:1912.10917*, 2019.

[3] X. Xie and K.-H. Kim, "Source compression with bounded DNN perception loss for IoT edge computer vision," in *Proc. of the International Conference on Mobile Computing and Networking (MobiCom)*, 2019.

[4] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan, "DeepN-JPEG: A deep neural network favorable JPEG-based image compression framework," in *Proc. of the the Design Automation Conference (DAC)*, 2018.

[5] Z. Li, C. De Sa, and A. Sampson, "Optimizing JPEG quantization for classification networks," *arXiv preprint arXiv:2003.02874*, 2020.

[6] L. Galteri, M. Bertini, L. Seidenari, and A. Del Bimbo, "Video compression for object detection algorithms," in *Proc. of the International Conference on Pattern Recognition (ICPR)*, 2018.

[7] L. Guo, D. Zhou, J. Zhou, S. Kimura, and S. Goto, "Lossy compression for embedded computer vision systems," *IEEE Access*, vol. 6, 2018.

[8] G. K. Wallace, "The JPEG still picture compression standard," *Transactions on Consumer Electronics*, vol. 38, no. 1, 1992.

[9] J. Nystad, S. Lassen, A. Pomianowski, S. Ellis, and T. Olson, "Adaptive scalable texture compression," in *Eurographics / ACM SIGGRAPH Symposium on High Performance Graphics*, 2012.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *arXiv:2004.10934*, 2020.

[11] K. I. Iourcha, K. S. Nayak, and Z. Hong, "System and method for fixed-rate block-based image compression with inferred pixel values," US Patent 5,956,431, 1999.

[12] J. M. P. Van Waveren and I. Castaño, "Real-time YCoCg-DXT compression," Technical report, id Software, Inc. and NVIDIA Corp., 2007.

[13] D. Pohl, D. Jungmann, B. Taudul, R. Membarth, H. Hariharan, T. Herfet, and O. Grau, "The next generation of in-home streaming: Light fields, 5K, 10 GbE, and foveated compression," in *Proc. of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2017.

[14] P. Holub, M. Šrom, M. Pulec, J. Matela, and M. Jirman, "GPU-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2K, and 4K video," *Future Generation Computer Systems*, vol. 29, no. 8, 2013.

[15] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[16] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes dataset for semantic urban scene understanding," in *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] V. Bruns, T. Richter, B. Ahmed, J. Keinert, and S. Föel, "Decoding JPEG XS on a GPU," in *Proc. of the Picture Coding Symposium*, 2018.

[18] R. Franzen, "Kodak lossless true color image suite," [online] Available: `http://r0k.us/graphics/kodak`.

[19] B. Brummer and C. de Vleeschouwer, "Adapting JPEG XS gains and priorities to tasks and contents," in *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[20] A. Descampe, J. Keinert, T. Richter, S. Fößel, and G. Rouvroy, "JPEG XS, a new standard for visually lossless low-latency lightweight image compression," in *Proc. of the Applications of Digital Image Processing XL*, 2017.

[21] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection," in *Proc. of the International Conference on Image Processing (ICIP)*, 2018.

[22] H. Choi and I. V. Bajic, "High efficiency compression for object detection," in *Proc. of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[23] J. Shao and J. Zhang, "Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems," in *Prof. of International Conference on Communications Workshops (ICC Workshops)*, 2020.

[24] L. Duan, J. Liu, W. Yang, T. Huang, and W. Gao, "Video coding for machines: A paradigm of collaborative compression and intelligent analytics," *Transactions on Image Processing*, vol. 29, 2020.

[25] Y. Hu, S. Yang, W. Yang, L.-Y. Duan, and J. Liu, "Towards coding for human and machine vision: A scalable image coding approach," in *Proc. of the International Conference on Multimedia and Expo (ICME)*, 2020.

[26] S. Pratapa, T. Olson, A. Chalfin, and D. Manocha, "TexNN: Fast texture encoding using neural networks," *Computer Graphics Forum*, vol. 38, no. 1, 2019.