DEEP FEATURE COMPRESSION FOR COLLABORATIVE OBJECT DETECTION

Hyomin Choi and Ivan V. Bajić

School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada

ABSTRACT

Recent studies have shown that the efficiency of deep neural networks in mobile applications can be significantly improved by distributing the computational workload between the mobile device and the cloud. This paradigm, termed *collaborative intelligence*, involves communicating feature data between the mobile and the cloud. The efficiency of such approach can be further improved by lossy compression of feature data, which has not been examined to date. In this work we focus on collaborative object detection and study the impact of both near-lossless and lossy compression of feature data on its accuracy. We also propose a strategy for improving the accuracy under lossy feature compression. Experiments indicate that using this strategy, the communication overhead can be reduced by up to 70% without sacrificing accuracy.

***Index Terms*—** Deep feature compression, collaborative intelligence, compression-augmentation, object detection

1. INTRODUCTION

Mobile and Internet-of-Things (IoT) [1] devices are increasingly relying on Artificial Intelligence (AI) engines to enable sophisticated applications such as personal digital assistants [2], self-driving vehicles, autonomous drones, smart cities, and so on. The AI engines themselves are generally built on deep learning models. The most common way of deploying such models is to place them in the cloud and have the sensor data (images, speech, etc.) uploaded from the mobile to the cloud for processing. This is referred to as the *cloud-only* approach. More recently, with smaller graphical processing units (GPUs) making their way into mobile/IoT devices, some deep models might be able to run on the mobile device, an approach referred to as *mobile-only*.

A recent study [3] has examined a spectrum of possibilities in between the cloud-only and mobile-only extremes. Specifically, they considered splitting a deep network into two parts: the front end (consisting of an input layer and a number of subsequent layers), which runs on the mobile, and the back end (consisting of the remaining layers), which runs on the cloud. In this approach, termed *collaborative intelligence*, the front end computes features up to some layer in the network, then these features are uploaded to the cloud for the remainder of the computation. The authors examined the energy consumption and latency associated with performing computation in this way, for various split points in typical deep models. Their findings indicate that significant savings can be achieved in both energy and latency if the network is split appropriately. They also proposed an algorithm called *Neurosurgeon* to find the optimal split point, depending on whether energy or latency is to be minimized.

The reason why collaborative intelligence can be more efficient than cloud-only and mobile-only approaches is that the feature data volume in deep convolutional neural networks (CNNs) typically decreases as we move from the input to the output. Executing initial layers on the mobile will cost some energy and time, but if the network is split appropriately, we will end up with far less data to be uploaded to the cloud, which will save both transmission latency on the uplink and the energy used for radio transmission. Hence, on the balance, there may be a net benefit in energy and/or latency. Based on [3], depending on the resources available (GPU or CPU on the mobile, speed and energy for wireless transmission, etc.), optimal split points for CNNs tend to be deep in the network.

A recently released study [4] has extended the approach of [3] to include model training and additional network architectures. While the network is again split between the mobile and the cloud, in the framework proposed in [4] the data can move both ways between the mobile and the cloud in order to optimize efficiency of both training and inference.

While [3, 4] have established the potential benefits of collaborative intelligence, the issue of efficient transfer of feature data between the mobile and the cloud is largely unexplored. Specifically, [3] does not consider feature compression at all, while [4] uses 8-bit quantization of feature data followed by lossless compression, but does not examine the impact of such processing on the application. Feature compression can further improve the efficiency of collaborative intelligence by minimizing the latency and energy of feature data transfer. The impact of compressing the input has been studied in several CNN applications [5, 6, 7] and the effects vary from case to case. However, the impact of feature compression has not been studied yet, to our knowledge.

In this work, we focus on a deep model for object detection and study the impact of feature compression on its accuracy. Section 2 presents preliminaries, while Section 3 describes the proposed methods. Experimental results and conclusions are presented in Sections 4 and 5, respectively.

2. PRELIMINARIES

Object detection has been transformed in recent years with the advent of deep models that are able to simultaneously detect, localize, and classify objects in an image. Examples of such detectors include R-CNN [8], SSD [9], and YOLO [10]. This work focuses on YOLO. One of the major innovations of these detectors was that they were trained using a cost function composed of both bounding box error and object class error terms. The YOLO loss function is [10]:

$$\begin{aligned}
&\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]\\
&+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2\right]\\
&+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2\\
&+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2\\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2 \qquad (1)
\end{aligned}$$

where (x_i, y_i) is the center of the ground truth bounding box, w_i and h_i are its width and height, $(\hat{x}_i, \hat{y}_i)$ is the center of the predicted bounding box whose width and height are $\hat{w}_i$ and $\hat{h}_i$, respectively. C_i and $\hat{C}_i$ are the ground truth and predicted confidence scores corresponding to cell i, $p_i(c)$ and $\hat{p}_i(c)$ are the ground truth and predicted conditional probabilities for the object class c in cell i, $\mathbb{1}_{ij}^{obj}$ is equal to 1 if the j-th bounding box in cell i is responsible for prediction (i,e. box j has the largest Intersection-over-Union among all boxes in cell i), and $\mathbb{1}_{ij}^{noobj} = 1 - \mathbb{1}_{ij}^{obj}$. The scaling factors used are $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$.

Our experiments in this work are based on the recent version of YOLO called YOLO9000 [11]. Fig. 1 shows the feature data volume (number of feature samples) at the output of each layer of this model, as well as the cumulative computational cost (normalized execution time) as we move from the input layer towards the output. Computational cost was measured on a desktop machine with a Titan X GPU and Intel i7-6800K CPU over the images from a dataset described in Section 4. As seen in the figure, the feature data volume is fairly small starting with max-pooling layer max_7. Hence, this layer, or other downstream layers seem to be good points to split the network. Note that max-pooling (and other pooling) layers reduce the data volume, so from the point of view of data size, it is always advantageous to split the network at the output of the max-pooling layer rather than at its input.

If we were to split the network at the output of some layer and transfer its feature data losslessly (as 32-bit floating point

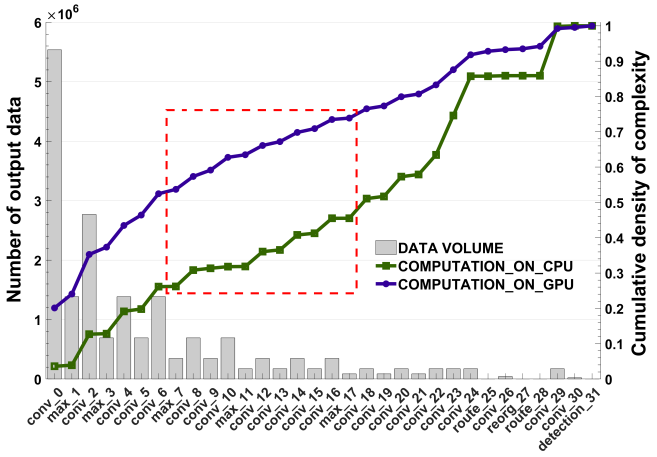


Fig. 1. Cumulative computation complexity and layer-wise output data volume

numbers) to the next layer (in the cloud), the accuracy would clearly stay the same as without the split[1]. This is the approach taken in [3], and is illustrated in Fig. 2(a). But this is inefficient because the data likely contains some redundancy.

A more efficient approach would be to compress the data prior to upload to the cloud. To achieve this, we could quantize the data, say to 8 bits per sample, then encode the quantized data losslessly. This is the approach taken in [4] with a lossless PNG encoder. It is illustrated in Fig. 2(b), where the quantization layer is called the Q-layer. This approach is *near-lossless* because there is some quantization involved, and due to this quantization the accuracy of inference may be affected. An even more efficient approach to data transfer is to employ lossy compression after the Q-layer (Fig. 2(c)), but this will have an even greater impact on the accuracy. These issues are examined in Section 4.

3. PROPOSED METHODS

3.1. Quantization

In order to leverage existing codecs, the feature data is first quantized to 8-, 10-, or 12-bit precision in a Q-layer, which is inserted at the split point. Let $\mathbf{V} \in \mathbb{R}^{N\times M\times C}$ be the tensor containing the feature data at the point of split, with N rows, M columns, and C channels. Let $\min(\mathbf{V})$ and $\max(\mathbf{V})$ be the minimum and maximum value in $\mathbf{V}$, respectively. Quantization with n_{bit}-precision and the corresponding inverse quantization in the inverse Q-layer are performed as

$$\widetilde{\mathbf{V}} = \text{round}\left(\frac{\mathbf{V}-\min(\mathbf{V})}{\max(\mathbf{V})-\min(\mathbf{V})}\cdot(2^{n_{bit}}-1)\right) \qquad (2)$$

$$\widehat{\mathbf{V}} = \frac{\widetilde{\mathbf{V}}\cdot(\max(\mathbf{V})-\min(\mathbf{V}))}{2^{n_{bit}}-1}+\min(\mathbf{V}) \qquad (3)$$

[1]If the data is not corrupted during transmission, as assumed in [3, 4].

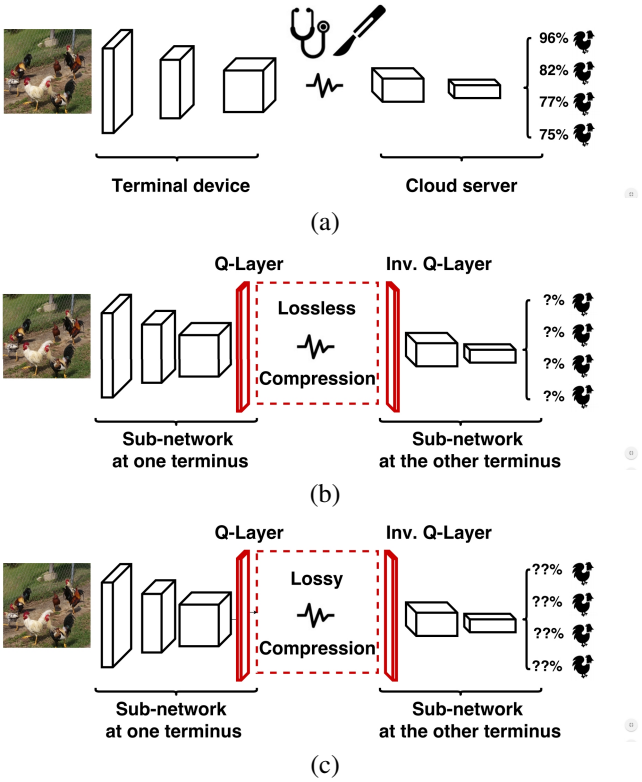
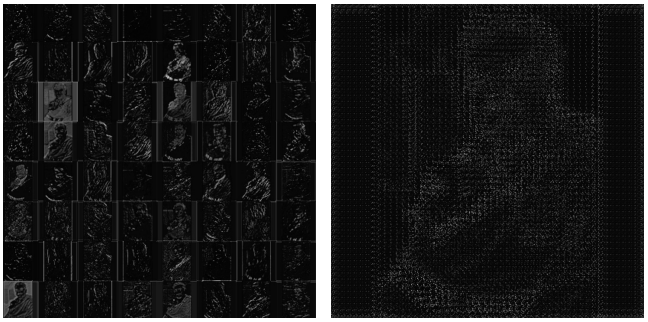


Fig. 2. Three ways for mobile-cloud collaborative intelligence: (a) Lossless transfer, (b) quantization followed by lossless compression, and (c) lossy compression.

where $\widetilde{\mathbf{V}}$ is the quantized feature tensor, $\widehat{\mathbf{V}}$ is the de-quantized feature tensor, and round$(\cdot)$ represents rounding to the nearest integer. min$(\mathbf{V})$ and max$(\mathbf{V})$ need to be stored as 32-bit floats (8 Bytes total) and transferred to the cloud for de-quantization. This is taken into account when computing total bits in the experiments. Note that in some cases, such as when the previous activation layer is sigmoid or ReLU (assumed in [4]), we can consider min$(\mathbf{V}) = 0$ and avoid transmitting it, but for more general activation layers such as Leaky ReLU (which is used in YOLO9000) this parameter is required.

3.2. Compression

Quantized feature tensor $\widetilde{\mathbf{V}}$ can be encoded by a number of existing codecs. If we interpret the $N \times M \times C$ tensor as C frames of size $N \times M$, we could employ a video codec to compress it. We could combine groups of feature channels into larger frames, to end up with less than C frames with larger resolution. Finally, all channels could be combined into a single image. Even in this case there are a number of possibilities, such as tiling (Fig. 3(a)), where the entire channel is placed in the image as a tile, followed by another tile, and so on, and quilting (Fig. 3(b)), where neighboring samples come from different channels. We tested a number of such methods and found that simple tiling by channel index provided the best results, so we use that method from here on. Hence, tiled feature channels are compressed as a still image.



(a) (b)

Fig. 3. Combining feature channels into an image by (a) tiling and (b) quilting.

For compression, we employ high efficiency video coding (HEVC) [12] standard, specifically HEVC Range extension (RExt) [13] which supports 4:0:0 sample format with various bit-depths. HM16.12 [14] in the experiments and all coding tools and configuration follow common test condition [15]. RDOQ tool is turned off and the coding tree unit (CTU) size is set to 16×16, because the feature channel resolution is relatively small deep in the network.

3.3. Compression-augmented training

As will be seen in Section 4, Q-layer quantization followed by lossless compression has little effect on the accuracy. However, lossy compression may affect the accuracy, especially when the quantization parameter (QP) is high. This loss in accuracy can be somewhat compensated by compression-augmented training. Instead of using the network parameters (weights) supplied with the model, we re-train the model by considering lossy compression at the point of split. During training, at each forward pass through the network, the feature data at the split point is tiled and compressed using a randomly chosen QP value. In our experiments we used QP in the range [Lossless, 22, 27, 32, 37]. After compression, the decompressed data is passed further down the network.

This kind of compression augmentation can be interpreted as a form of regularization, where quantization noise is inserted into an intermediate layer deep in the network. It encourages the network to learn the downstream weights (from the split point) that provide good accuracy when processing decompressed features, and also to learn upstream weights that generate features that are robust to compression.

4. EXPERIMENTS

Following [11], a total of 16,551 images from VOC2007 and VOC2012 datasets [16, 17] are used for training and another 4,952 images from VOC2007 for testing. Twenty different object classes are represented in the dataset.

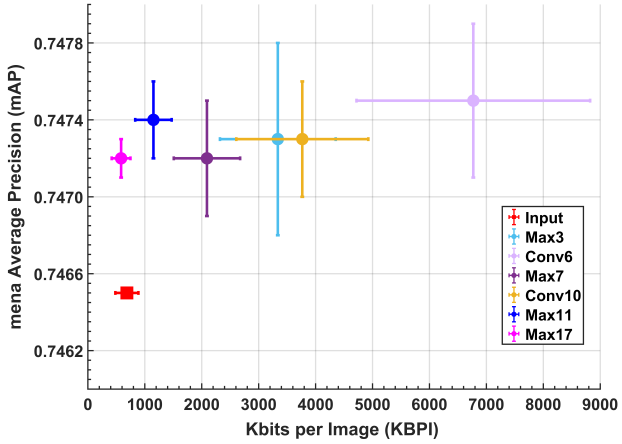


Fig. 4. mAP vs. KBPI for lossless deep feature compression

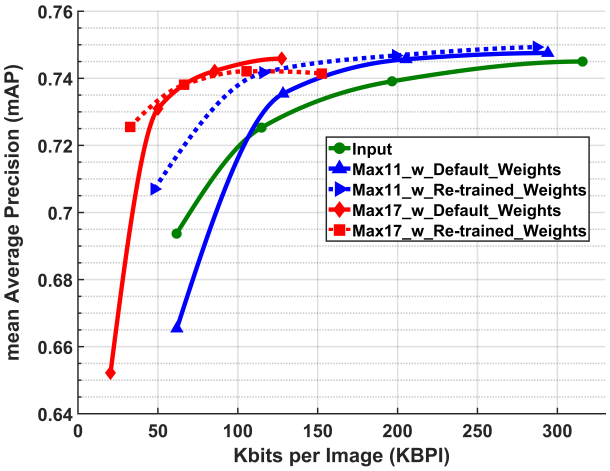


Fig. 5. mAP vs. KBPI for lossy deep feature compression

We first test the impact of lossless compression (after the Q-layer) on accuracy. As is common with multi-class object detectors [18], we use mean Average Precision (mAP) as a measure of accuracy, and look at its variation with 8-bit, 10-bit and 12-bit quantization in the Q-layer. The compression of feature data is quantified using average Kbits per image (KBPI). Fig. 4 presents mAP versus KBPI for various split points in the network. Vertical bars show the standard deviation of mAP at a given average KBPI, while horizontal bars show the standard deviation of KBPI for the corresponding average mAP. The red square indicates the operating point achieved by the cloud-only approach, without network splitting and uploading the input JPEG images to the cloud.

As seen in the figure, when the split point is close to the input (e.g. max_3, conv_6 or conv_10 layers), the data volume is too large, and even with lossless compression of feature data, it is more efficient to simply upload input images to the cloud. But as we move down the network, it becomes more advantageous to upload feature data. Meanwhile, the mAP does not change much - scores around 0.7465-0.7475 are achieved for all the cases. Hence, lossless compression of deep features (following 8-, 10-, or 12-bit quantization) has only a minor influence on accuracy, but also provides limited (if any) bit savings for data transfer to the cloud.

Lossy compression offers significant bit savings, but care must be taken to minimize the loss of accuracy. In order to evaluate the impact of lossy compression, we show mAP vs. KBPI curves in Fig. 5. The green curve corresponds to compressing the input image, as the default cloud-only approach. The blue curves correspond to splitting the network at the output of max_11 layer, and red curves correspond to the split after the max_17 layer. In each case, the solid line corresponds to using default YOLO9000 weights while the dashed line corresponds to using the weights obtained by compression-augmented training, starting from the pre-trained weight, "Darknet19 448x448", for ImageNet classification [19] and following the training procedure in [20]. As seen in the figure, lossy compression can provide significant bit savings over the cloud-only approach, while compression-augmented training further extends the range of useful compression levels for a given mAP.

To quantify the differences between various cases, we adopt a Bjontegaard Delta (BD) approach [21]. Specifically, we use the BD calculation to compute BD-KBPI-mAP, which indicates the average difference in KBPI at the same mAP. The results are shown in Table 1, where the default case against which the comparison is made is the cloud-only approach. As shown in the table, compressing features at the output of max_11 (max_17) while using default weights would give an average saving of 6% (60%) at the same mAP compared to cloud-only approach. Meanwhile, the weights obtained through compression-augmented training would provide an additional bit saving of 39% (10%), for the total of up to 45% (70%) bit savings.

5. CONCLUSIONS

We studied deep feature compression for collaborative object detection between the mobile and the cloud. We examined the impact of compression on detection accuracy and showed that lossless compression of 8-bit (or higher) quantized data does not have much impact on the accuracy. Lossy compression provides higher bit savings, but also affects the accuracy. To compensate for this, we proposed compression-augmented training, which is able to extend the range of useful compression levels for a desired accuracy.

Table 1. BD-KBPI-mAP of lossy feature compression vs. cloud-only approach

Split at	Default weights	Re-trained weights
max_11	-6.09%	$\mathbf{-45.23\%}$
max_17	-60.30%	$\mathbf{-70.30\%}$

6. REFERENCES

[1] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *Int. Journal of Communication Systems*, vol. 25, no. 9, pp. 1101, 2012.

[2] R. Sarikaya, "The technology behind personal digital assistants: An overview of the system architecture and key components," *IEEE Signal Processing Magazine*, vol. 34, no. 1, pp. 67–81, Jan. 2017.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 615–629.

[4] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services," *arXiv preprint arXiv:1801.08618*, 2018.

[5] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *IEEE Int. Conf. Quality of Multimedia Experience (QoMEX'16)*. IEEE, 2016, pp. 1–6.

[6] L. Kong, R. Dai, and Y. Zhang, "A new quality model for object detection using compressed videos," in *Proc. IEEE ICIP'16*, Sep. 2016.

[7] H. Choi and I. V. Bajić, "High efficiency compression for object detection," in *Proc. IEEE ICASSP'18*, Apr. 2018, to appear.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE CVPR'14*, 2014, pp. 580–587.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *Proc. ECCV*, 2016.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE CVPR'16*, Jun. 2016, pp. 779–788.

[11] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proc. IEEE CVPR'17*, Jul. 2017, pp. 6517–6525.

[12] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[13] D. Flynn, D. Marpe, M. Naccari, T. Nguyen, C. Rosewarne, K. Sharman, J. Sole, and J. Xu, "Overview of the range extensions for the hevc standard: Tools, profiles, and performance," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 4–19, 2016.

[14] "HEVC reference software (HM 16.12)," https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.12, Accessed: 2017-05-27.

[15] F. Bossen, "Common HM test conditions and software reference configurations," in *ISO/IEC JTC1/SC29 WG11 m28412, JCTVC-L1100*, Jan. 2013.

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," http://host.robots.ox.ac.uk/pascal/VOC/voc2007/.

[17] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://host.robots.ox.ac.uk/pascal/VOC/voc2012/.

[18] M. Everingham, L. van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (voc) challenge," *Int. Journal of Computer Vision*, vol. 88, no. 2, pp. 330–338, 2010.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and Li F, "ImageNet Large Scale Visual Recognition Challenge," *Int. Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[20] J. Redmon, "Darknet: Open source neural networks in C.," http://pjreddie.com/darknet/, 2013-2017, Accessed: 2017-10-19.

[21] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," Apr. 2001, VCEG-M33.