Implicit Neural Representations using XCiT Transformers

Lorenzo Liso [*1] **Francesco Deaglio** [*1] **Clara Teissier** [1]

Abstract

Compressing images while maintaining a faithful representation is one of the longest-running challenges in the context of deep learning. Compression is a problem with many applications (reducing memory space, reducing communication bandwidth...) and a myriad of solutions using the latest architectures such as CNNs (Cavigelli et al., 2017), autoencoders (Kingma & Welling, 2013; Theis et al., 2017) and GANs (Agustsson et al., 2019). Now the trend is transformers: in this paper, we use image codes learned with XCiT transformers to achieve reliable compression using as little as one bit per pixel. As far as we know, this is one of the first papers to use transformers to compress images and may be the starting point for more powerful models.

1. Introduction

An extremely large number of images are compressed every day mainly to reduce storage or required communication bandwidth. There are different techniques, which can be divided into two macro-categories: lossy and lossless, depending on whether the compressed image is similar or the same as the original. Generally, compression algorithms based on deep learning techniques belong to the first strategy, and our solution is lossy as well. The most famous lossy compression algorithm is JPEG (Hamilton, 2004), in which the image is transformed with DCT (Discrete Cosine Transform), quantized and finally an entropic encoding is applied. In our solution, we replace the first part with a learned representation (which we will call `ImageCode`), obtained according to the architecture described in the second section. In the third section we will show the results obtained on CIFAR10 (Alex Krizhevsky & Hinton, 2010), DIV2K (Agustsson et al., 2019) and KODAK (Kodak, 1991), achieving excellent compression using one bit per pixel, exploiting an `ImageCode` of 256 values on 4 bits.

Our strategy is similar to COIN++ (Dupont et al., 2022), where a learned latent vector is used as a modulation of the network layers. However, it differs in the way used to learn this vector: in COIN a SIREN-based MLP is used while we employ a combination of MLPs and transformers.

2. Models and method

The starting architecture for our project is borrowed from the paper Shape Transformers (Chandran et al., 2022), where it is used to represent the domain of statistically-plausible 3D shapes in a compact and controllable way. Having to solve a totally different problem, we have made several changes, which we will explain in the following subsections.

At a high level, RGB colors and positions (in the range [-1,1]) are extracted from the image and independently passed through two networks to increase dimensionality. Coordinates and corresponding colors are concatenated to create tokens that are fed as input to a cascade of XCiT transformers (Ali et al., 2021). In addition to these tokens, an extra learnable token is passed as input and its output is our compressed representation, which we call `ImageCode`. This vector is used to modulate both the XCiT decoder and the output neural network, which is used to reconstruct colors point by point. A more detailed representation is available in figure 1.

The model is trained by minimising the mean squared error between the original and the compressed image. We tried other techniques (L1, PSNR and weighted combinations) but all produced worse results (in terms of convergence time and quality).

2.1. Color and Position MLPs

We use multi-layer perceptrons in different stages of our architecture, to map inputs into a higher or lower dimension. This increases the expressive power of our model by being able to have high-dimensional tokens. Without MLPs, tokens would be 5-dimensional elements, being the concatenation of colors ($3d$) and positions (2d). Using MLPs we can map positions and colors to any arbitrary dimension. We found no advantage in giving more elements to positions

[*]Equal contribution [1]Department of Computer Science, ETH, Zürich, Switzerland. Correspondence to: Lorenzo Liso <loliso@student.ethz.ch>, Francesco Deaglio <fdeaglio@student.ethz.ch>, Clara Teissier <cteissier@student.ethz.ch>.

Deep Learning, ETH Zürich, 2022

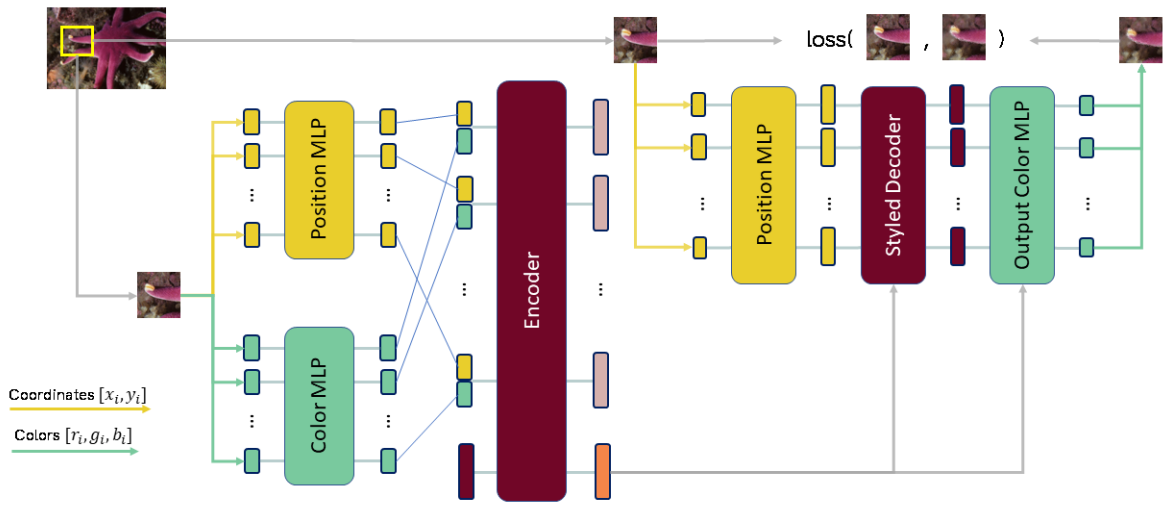

Figure 1. Our architecture . The `ImageCode` is represented in orange.

than colors (or vice versa). Therefore, if the `ImageCode` has dimensionality n, `Position MLP` and `Color MLP` will each produce $n/2$ elements.

Each MLP has four hidden layers twice the size of the output. We have found that this slightly improves quality compared to having the same size of the output while increasing it further slows down the training without any consistent benefit.

While the `color MLP` follows the classical architecture, for the `position MLP` and `output color MLP` we have implemented specific functionalities. In particular the `position MLP` applies the same positional encoding as in NeRF (Mildenhall et al., 2021) with the number of frequencies set to 16.
The `output color MLP` has its activation modulated by `ImageCode` as follows: a linear layer maps the image code to a modulation vector that has the size of the summation of the hidden layers dimensionality. During the forward pass, every activation is modulated by shifting (i.e summing) the values of the activation with the corresponding values in the modulation vector. More details can be found in (Dupont et al., 2022)

2.2. Encoder and Decoder

Our transformer-based architecture is implemented as a sequence of XCIT blocks (Ali et al., 2021) with the LPI layer removed as we are not interested in modelling local patch interaction and substituting it with a fully connected layer. We decided on using the XCIT cross-covariance attention method to avoid the memory bottleneck given by the standard attention matrix, as its number of elements depends on the squared length of the input/output sequence which grows rapidly with the image resolution.
The goal of the encoder is to produce the `ImageCode` that represents the whole image. To do so a strategy similar to the [CLS] token in BERT (Devlin et al., 2018) is used: a learnable extra token is added to the encoder input and the corresponding token in the output sequence represents the `ImageCode`. This extra input token is not position encoded and the other tokens' output are not used.
The decoder architecture takes as input the sequence of tokens produced by the `Position MLP` and the `ImageCode` and generates the sequence of tokens used by the `Output Color MLP` to produce the final RGB values. The `ImageCode` is used to modulate the intermediate activation that serves as input to each decoder block, injecting information to each transformer block. In detail every transformer block has a 5-layer MLP (of which the first 4 are shared among all the encoder blocks) that maps the `ImageCode` to the dimensionality of the input token, then the final modulated input is obtained by multiplying every input token with the resized `ImageCode`.

2.3. Quantization

The `ImageCode` is a tensor of n elements on 32 bits, but do we really need that much precision? The answer is no, and as we will show in the next section, 4 bits are more than enough. To do this, we have quantized the image codes by following a simple algorithm. First, we calculate the mean and standard deviation for each dimension and normalise the `ImageCodes`. At this point, we clamp all values in the range $[-3, 3]$ (if they are outside this range, we simply set them to the nearest extreme) and divide into 2^{nbits} intervals. Each value is represented with the index of the interval to which it belongs (between 0 and $2^{nbits} - 1$) and stored.

To reconstruct the image, the `ImageCodes` is dequantized (by adding the mean and scaling the standard deviation) and used to modulate the decoder as described before.

Furthermore, by quantizing the `ImageCodes` we can compress it further using entropy encoding. Since it is a lossless technique (so it does not alter the reconstruction) we did not elaborate further. We merely mention that by using a trivial technique such as run-length encoding on an alphabet of a few symbols, we can achieve a substantial improvement.

3. Results

In image compression, the main challenge is to minimize the number of bits per pixel (low bpp) while maintaining high quality (high PSNR). In our model, there are two possible ways to achieve this: using an `ImageCode` with few elements but high precision or an `ImageCode` with many elements quantized on few bits. The second solution is by far the best for us.

We trained the model on CIFAR10 and evaluated it on CIFAR10, DIV2K and KODAK. In the next subsections, we will discuss the results obtained using each dataset.

3.1. CIFAR10

Given the limited resources available, we used this dataset for training. The first experiments were performed without quantization with an `ImageCode` of 32 elements and allowed us to achieve decent but not satisfactory results (20 of PSNR). In subsequent iterations we used progressively larger `ImageCode` (64, 128, 256 elements) quantized on 2, 4, 8, 16 bits (the "original" version is a `torch.float32` tensor).

Table 1. PSNR for different `ImageCode` dimensions. Green cells indicate a compression of 1 bpp. The **bolded** cell is our best model, used to evaluate on the other datasets. An extended version, with standard deviations, is reported in the appendix (*Table 3*).

	original	quantized			
dim	32 bit	2 bit	4 bit	8 bit	16 bit
32	19.9	15.7	19.6	19.9	19.9
64	21.1	15.7	20.6	21.0	21.0
128	23.5	17.2	22.8	23.4	23.4
256	26.6	18.8	**25.6**	26.5	26.5

From the reported results, we can observe that by increasing the size of the `ImageCode` the quality improves significantly and that the quantization does not negatively impact performance, being able to reach even 4 bit with a satisfactory result.

To compare our model on this dataset, we use measurements made in COIN++ (based on the benchmarking tool `compressai` (Bégaint et al., 2020)). We add our results to the chart (Figure 2), shown on multiple lines depending on the size of the `ImageCode` used (e.g. `ImageCode-256`). As can be seen, our model is slightly worse (but competitive) at low bit-rates and outperforms static techniques such as JPEG. The PSNR of our model saturates for high bit-rates. However, we believe that with progressively larger `ImageCode` sizes, we can compete even in that regime. It was not possible for us to test due to hardware limitations.

3.2. KODAK and DIV2K

We evaluated our model, trained on CIFAR10, on these two datasets as well. Working with 32x32 images, we had to patch the pictures (dividing them into 384 patches) and compress each patch individually. This brought excellent results, which we report in Table 2. We also report a reconstructed KODAK image here and some others in the appendix.

Table 2. PSNR of our best model (`ImageCode-256`) with datasets KODAK and DIV2K. An extended version, with standard deviation, is reported in the appendix (*Table 4*).

	original	quantized			
	32 bit	2 bit	4 bit	8 bit	16 bit
KODAK	31.7	20.9	29.4	31.4	31.4
DIV2K	34.2	19.3	30.3	33.9	34.0

Regarding the KODAK dataset, our model obtains comparable results to INR (Strümpler et al., 2022) and Ballé (Ballé et al., 2016) on this dataset at 1 bpp. These models (and others, the best on this dataset is the one proposed by Xie (Xie et al., 2021)), using global and not patch compression, manage to push to even lower bitrates (up to 1 bit per 5 pixels). Presumably, using entropy encoding on the 4-bit image we would be able to reach those bit-rates but we have no data to support this claim.

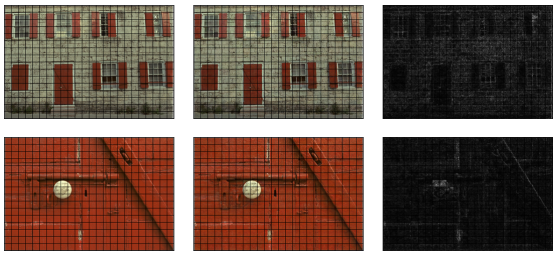

Figure 3. Comparison original (left) vs compressed at 1 bpp (center) vs residuals (right). Pictures from KODAK dataset. When reconstructing the image, we left the grid visible to emphasise that each patch was compressed individually.

As for DIV2K, we have no other results to benchmark our model. In the appendix, we provide some reconstructed

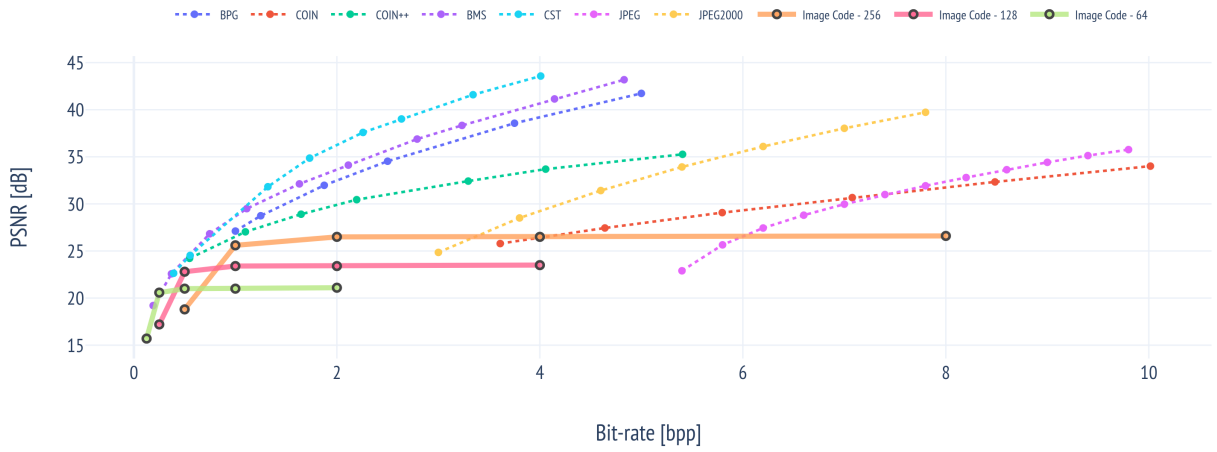

Figure 2. Comparison on CIFAR10. Benchmark for the other models are taken from COIN++ (based on `compressai`).

images from DIV2K and a graphical comparison of the various quantization levels.

4. Discussion and Conclusion

In this work, we showed how the expressive power of transformers can be used for image compression. Using an autoencoder-like architecture we were able to remove the complexity of using meta-learning algorithms, moreover, the latent vector produced by the encoder seems to be very robust with respect to quantization with only a very small decrease in performance. For memory and time constraints we have kept the hidden dimension of the transformer blocks to the same size as the `ImageCode`, but as the network is not part of the compression these dimensions could be increased hopefully resulting in better reconstruction quality.
In our approach we followed the same patch technique used in COIN++ for high-resolution images, leading to blocking artifacts in the whole image reconstruction. It would be interesting to leverage the image to patch embedding and local patch interaction (LPI) (Ali et al., 2021) to test on the whole image or still increase the patch size to a resolution greater than 32x32.
In addition to the lossless compression using entropy coding, a loss augmentation considering the compression rate could be applied, pushing the latent space of the encoder to be more inclined to quantization or entropy encoding.
Furthermore, more schemes of positional encoding could be tried such as the learned positional encoding used in general vision transformers (Dosovitskiy et al., 2020).

A third parameter that we have not considered is the compression time (as compression is often used in latency-sensitive applications, like file sharing in messaging applications). Depending considerably on the hardware used and having limited resources, we decided not to optimize this metric. It would be interesting to fine-tune this model by also applying compression techniques to the network itself (e.g. Structured pruning (Anwar et al., 2017; Zhu et al., 2021)).

References

Agustsson, E., Tschannen, M., Mentzer, F., Timofte, R., and Gool, L. V. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 221–231, 2019.

Alex Krizhevsky, V. N. and Hinton, G. Cifar10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`, 2010.

Ali, A., Touvron, H., Caron, M., Bojanowski, P., Douze, M., Joulin, A., Laptev, I., Neverova, N., Synnaeve, G., Verbeek, J., et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34: 20014–20027, 2021.

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

Ballé, J., Laparra, V., and Simoncelli, E. P. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

Bégaint, J., Racapé, F., Feltman, S., and Pushparaja, A. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

Cavigelli, L., Hager, P. A., and Benini, L. Cas-cnn: A deep convolutional neural network for image compression artifact suppression. pp. 752–759, 05 2017. doi: 10.1109/IJCNN.2017.7965927.

Chandran, P., Zoss, G., Gross, M., Gotardo, P., and Bradley, D. Shape transformers: Topology-independent 3d shape models using transformers. *Computer Graphics Forum*, 41(2):195–207, 2022. doi: https://doi.org/10.1111/cgf.14468. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14468.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Dupont, E., Loya, H., Alizadeh, M., Golinski, A., Teh, Y. W., and Doucet, A. Coin++: neural compression across modalities. *Transactions on Machine Learning Research*, 2022(11), 2022.

Hamilton, E. Jpeg file interchange format. 2004.

Kingma, D. and Welling, M. Auto-encoding variational bayes. *ICLR*, 12 2013.

Kodak. Kodak dataset. http://r0k.us/graphics/kodak/, 1991.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Strümpler, Y., Postels, J., Yang, R., Gool, L. V., and Tombari, F. Implicit neural representations for image compression. In *European Conference on Computer Vision*, pp. 74–91. Springer, 2022.

Theis, L., Shi, W., Cunningham, A., and Huszár, F. Lossy image compression with compressive autoencoders, 2017. URL https://arxiv.org/abs/1703.00395.

Xie, Y., Cheng, K. L., and Chen, Q. Enhanced invertible encoding for learned image compression. In *Proceedings of the 29th ACM International Conference on Multimedia*, pp. 162–170, 2021.

Zhu, M., Tang, Y., and Han, K. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

A. Table 1 extension

Table 3. PSNR for different `ImageCode` dimensions. Green cells indicate a compression of 1 bpp. The **bolded** cell is our best model, used to evaluate on the other datasets. This is an extended version with standard deviations.

	original	quantized			
dim	32 bit	2 bit	4 bit	8 bit	16 bit
32	19.96±1.06	15.72±0.42	19.64±0.99	19.93±1.05	19.93±1.05
64	21.09±1.06	15.73±0.38	20.58±0.96	21.06±1.06	21.06±1.06
128	23.52±1.05	17.23±0.45	22.85±0.94	23.47±1.06	23.47±1.05
256	26.60±1.15	18.78±0.43	**25.65±0.97**	26.54±1.13	26.55±1.13

B. Table 2 extension

Table 4. PSNR of our best model (`ImageCode-256`) with datasets KODAK and DIV2K. This is an extended version with standard deviations.

	original	quantized			
	32 bit	2 bit	4 bit	8 bit	16 bit
KODAK	31.73±3.03	20.92±1.26	29.38±2.20	31.43±3.04	31.44±3.05
DIV2K	34.16±4.51	19.31±2.11	30.26±3.24	33.94±4.59	33.97±4.60

C. KODAK Highest and Lowest PSNRs

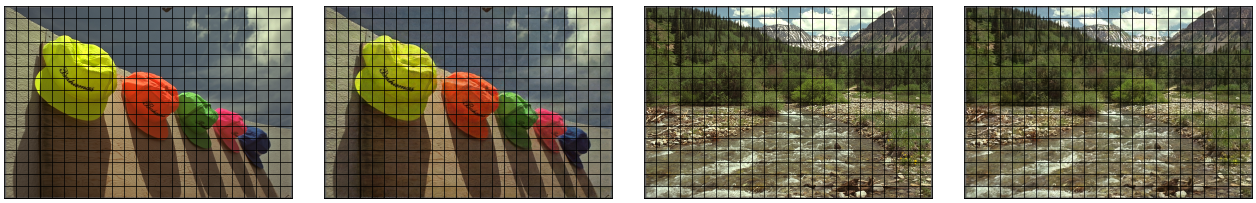

Figure 4. We report a couple (left: original, right: quantized on 4 bit) for the best and worst score we achieve on this dataset. The first couple is the picture with the best PSNR (35.47) while the second one has the lowest one (25.20)

D. DIV2K compressed images

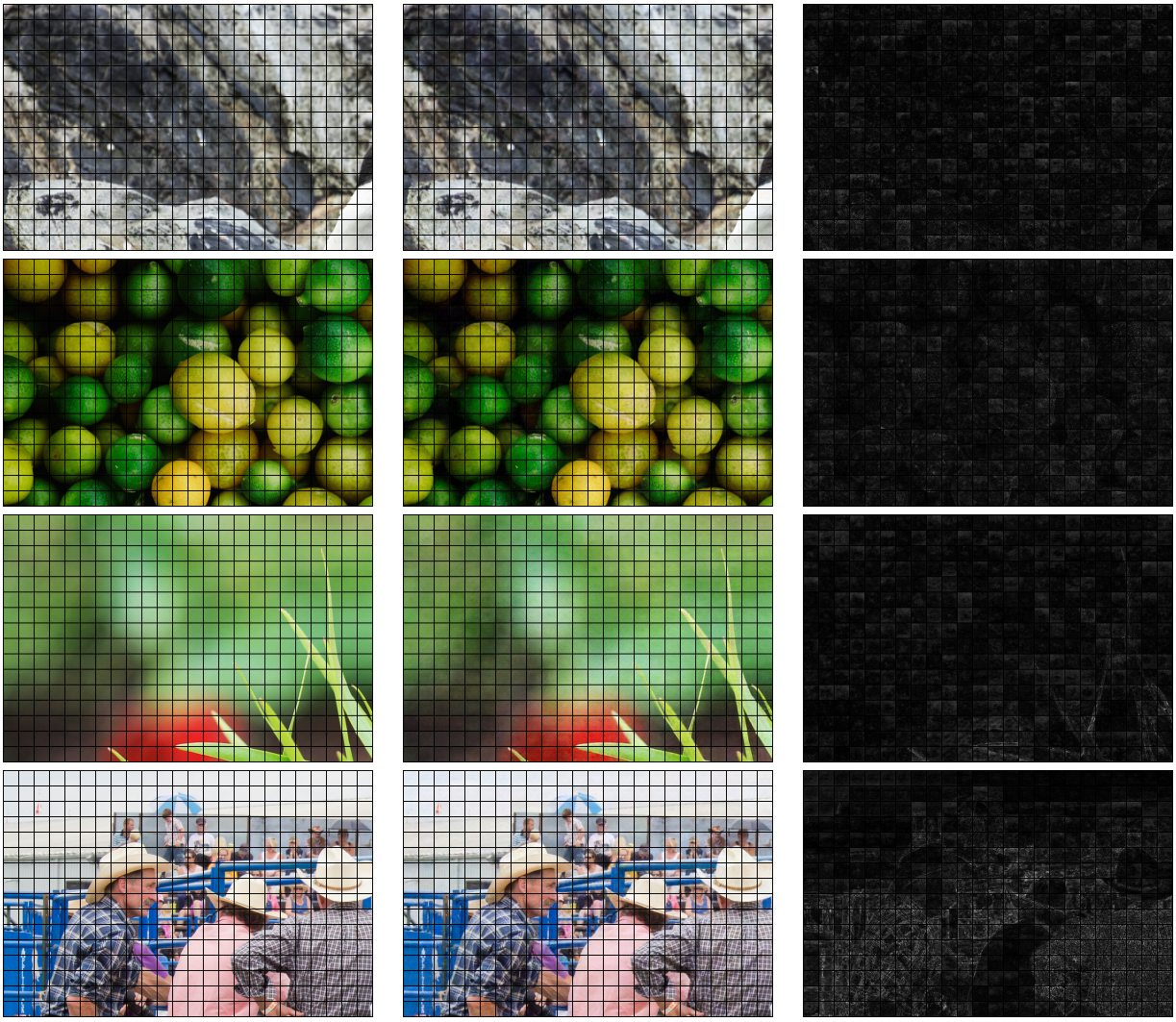

Figure 5. Comparison original (left) vs compressed at 1 bpp (center) vs residuals (right). Pictures from DIV2K dataset.

E. Quantization quality

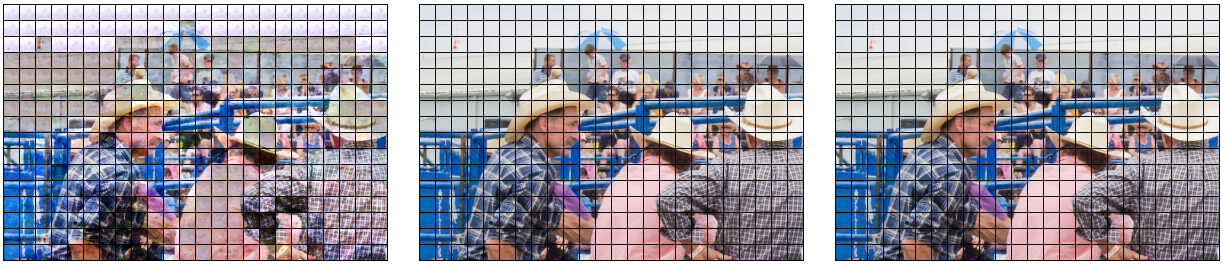

Figure 6. Comparison 0.5 bpp (left) vs 2 bpp (center) vs 4 bpp (right). Pictures from DIV2K dataset. The original and the 1 bpp version can be found in the section above. Compressed using a 256 elements `ImageCode`, so 1 bpp means that every element is quantized on 4 bits. As shown in the previous chapters, there is no great advantage to using more than 4 bits. However, using less makes the image almost unrecognisable.