Computer Vision Progress Report

Jiajun Li, Hanliang Jiang, Haoxiang Li

Project definition and goals

After carefully reviewing the algorithm and experiment in the paper, we proposed a few detailed contents of the experiment. First of all, we configure all the runtime environments needed and write batch test code to analyze average overhead, average compression rate, average peak signal-noise ratio, etc. For the next step, we decide to reproduce the fine granular scalability feature of the algorithm, e.g. under the condition of displaying an image with network congestion. Finally, we plan to implement the real world application of this algorithm, which is adding a runtime processing feature and using this compression method for runtime image transfer.

Member roles and collaboration strategy

Three members in the group share the work evenly across different work, including runtime environment configuration, paper reading and programming. We contact and discuss with each other closely through online methods, and we talk about arrangement and progress very often after class.Through working together, we learned a lot of new skills on paper reading, debugging and project planning. Collaboration greatly accelerated our learning steps.

Proposed approach

Reference: Jeon, Seungmin and Choi, Kwang Pyo and Park, Youngo and Kim, Chang-Su. Context-Based Trit-Plane Coding for Progressive Image Compression. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2023.

External code: [janghl/CTC: Context-Based Trit-Plane Coding for Progressive Image Compression, CVPR2023 (github.com)]

Step 1: Writing batch test code to analyze average overhead, average compression rate, average peak signal-noise ratio.
Code :[CTC/test.py at master · janghl/CTC (github.com)]

Step 2: Reproduce the fine granular scalability feature of the algorithm under the condition of displaying an image with network congestion

Step 3: Adding runtime processing feature and using this compression method for runtime image transfer
Pseudo code: CTC/runtime_pseudo_code.py at master · janghl/CTC (github.com)

Data

Original experiment data:
Enc 18.8sec, dec time: 23.779, bpp: 1.81974 , psnr: 36.4747
Enc 3.6sec, dec time: 10.395, bpp: 0.61312 , psnr: 41.3318
Enc 13.9sec, dec time: 22.690, bpp: 1.86190 , psnr: 38.0138
Enc 3.8sec, dec time: 10.786, bpp: 0.74463 , psnr: 40.5065
Enc 9.4sec, dec time: 17.928, bpp: 1.33325 , psnr: 38.5821
Enc 6.2sec, dec time: 16.754, bpp: 1.29720 , psnr: 39.1081
Enc 3.3sec, dec time: 12.729, bpp: 0.95321 , psnr: 41.3068
Enc 13.1sec, dec time: 23.374, bpp: 2.09269 , psnr: 37.4792
Enc 3.7sec, dec time: 10.853, bpp: 0.71696 , psnr: 41.9422
Enc 3.4sec, dec time: 14.399, bpp: 1.00675 , psnr: 40.1330
Enc 8.2sec, dec time: 15.007, bpp: 0.83537 , psnr: 41.7387
Enc 14.7sec, dec time: 32.712, bpp: 2.53743 , psnr: 34.1152
Enc 3.0sec, dec time: 13.635, bpp: 1.14445 , psnr: 39.8162
Enc 10.0sec, dec time: 21.610, bpp: 1.65495 , psnr: 37.0825
Enc 4.0sec, dec time: 12.861, bpp: 0.85498 , psnr: 40.2908
Enc 8.1sec, dec time: 20.176, bpp: 1.61987 , psnr: 39.3121
Enc 3.3sec, dec time: 12.083, bpp: 0.83187 , psnr: 40.6688
Enc 15.3sec, dec time: 25.361, bpp: 1.98022 , psnr: 38.3941
Enc 9.5sec, dec time: 25.536, bpp: 2.14559 , psnr: 37.0420
Enc 3.9sec, dec time: 13.963, bpp: 1.25277 , psnr: 39.5259
Enc 3.4sec, dec time: 13.338, bpp: 1.02759 , psnr: 39.8424
Enc 3.5sec, dec time: 17.888, bpp: 1.49886 , psnr: 38.3688
Enc 3.5sec, dec time: 12.433, bpp: 1.19255 , psnr: 41.0277
Enc 8.8sec, dec time: 18.603, bpp: 1.51090 , psnr: 39.4401
Average Encoding Time: 7.465 seconds
Average Decoding Time: 20.507 seconds
Average Compression rate: 0.89454

Initial results

We have compiled and run the external code and implemented step 1. The results are shown in the Data Section. Bpp stands for bits per pixel, indicating how many bits are used to store each pixel. A lower bpp means better image quality but requires more storage space. Psnr stands for

Peak Signal-to-Noise Ratio, which is the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is most commonly used to measure the quality of reconstruction of lossy compression, the signal in this case is the original data, and the noise is the error caused by compression.

Current reservations and questions

We are not a hundred percent sure whether step 2 and step 3 are easy to implement. I was wondering if you could give us insight about whether the pseudo code for step 3 is applicable, which is collecting part of the encoded data and reproducing an image with a low resolution.