Vector Quantization Image Compression Report

Omar Eldesoukey
202202155

May 13, 2025

Contents

1 Introduction

This report compares two image compression pipelines using vector quantization: one in the RGB color space, and one in the YUV color space with chroma subsampling. The results, code, and a concise comparison are provided.

2 Summary Table: RGB vs YUV Results

Image	RGB PSNR (R,G,B)	RGB Ratio	YUV PSNR (R,G,B)	YUV Ratio
animal_11.jpg	30.72, 30.66, 30.20	4.00	30.13, 30.55, 29.56	8.00
animal_12.jpg	37.84, 37.77, 37.02	4.00	36.13, 37.23, 35.78	8.00
animal_13.jpg	27.59, 27.76, 27.34	4.00	27.48, 27.54, 27.35	8.00
animal_14.jpg	33.77, 34.16, 33.95	4.00	33.01, 33.80, 33.22	8.00
animal_15.jpg	33.24, 33.28, 33.37	4.00	32.62, 33.16, 33.04	8.00
face_11.jpg	32.96, 32.78, 32.95	4.00	32.45, 33.03, 32.71	8.00
face_12.jpg	32.74, 32.79, 32.77	4.00	32.41, 32.46, 32.30	8.00
face_13.jpg	31.88, 32.12, 32.04	4.00	31.56, 32.00, 31.57	8.00
face_14.jpg	34.03, 34.30, 34.26	4.00	33.04, 34.05, 33.21	8.00
face_15.jpg	32.61, 32.96, 32.33	4.00	31.24, 32.39, 31.28	8.00
nature_11.jpg	30.37, 30.42, 30.91	4.00	29.99, 30.37, 28.50	8.00
nature_12.jpg	31.99, 31.30, 31.52	4.00	29.87, 30.74, 30.23	8.00
nature_13.jpg	31.51, 31.55, 33.22	4.00	30.94, 31.23, 28.09	8.00
nature_14.jpg	30.04, 30.13, 29.74	4.00	29.79, 30.00, 29.72	8.00
nature_15.jpg	31.77, 31.55, 31.28	4.00	31.27, 31.51, 31.13	8.00

3 Very Brief Comparison

- **YUV with chroma subsampling achieves twice the compression ratio of RGB (8.00 vs 4.00) with only a small loss in quality (slightly lower PSNR).**
- **RGB gives slightly better image quality but at a much larger file size.**

4 Main Code: Compression Pipelines

4.1 Main.java: Full Pipeline (RGB and YUV)

```
public class Main {
    public static void main(String[] args) throws Exception {
        // --- RGB PIPELINE ---
        File trainDir = new File(".../data/train");
        List<int[]> redVecs = new ArrayList<>(), greenVecs = new ArrayList<>(),
            blueVecs = new ArrayList<>();
        for (File folder : trainDir.listFiles()) {
            for (File imgFile : folder.listFiles()) {
                BufferedImage img = ImageUtils.loadImage(imgFile.getAbsolutePath
                    ());
                int[][] R = ImageUtils.getChannel(img, 'R');
```

```
            int[][] G = ImageUtils.getChannel(img, 'G');
            int[][] B = ImageUtils.getChannel(img, 'B');
            redVecs.addAll(VQEncoderDecoder.extractVectors(R));
            greenVecs.addAll(VQEncoderDecoder.extractVectors(G));
            blueVecs.addAll(VQEncoderDecoder.extractVectors(B));
        }
    }
    int[][] redCodebook = VectorQuantizer.kMeans(redVecs, 256, 20);
    int[][] greenCodebook = VectorQuantizer.kMeans(greenVecs, 256, 20);
    int[][] blueCodebook = VectorQuantizer.kMeans(blueVecs, 256, 20);

    File testDir = new File(".../data/test");
    for (File folder : testDir.listFiles()) {
        for (File imgFile : folder.listFiles()) {
            BufferedImage img = ImageUtils.loadImage(imgFile.getAbsolutePath
                ());
            int[][] R = ImageUtils.getChannel(img, 'R');
            int[][] G = ImageUtils.getChannel(img, 'G');
            int[][] B = ImageUtils.getChannel(img, 'B');
            int[][] rIdx = VQEncoderDecoder.compress(R, redCodebook);
            int[][] gIdx = VQEncoderDecoder.compress(G, greenCodebook);
            int[][] bIdx = VQEncoderDecoder.compress(B, blueCodebook);
            int[][] rDec = VQEncoderDecoder.decompress(rIdx, redCodebook);
            int[][] gDec = VQEncoderDecoder.decompress(gIdx, greenCodebook);
            int[][] bDec = VQEncoderDecoder.decompress(bIdx, blueCodebook);
            BufferedImage out = ImageUtils.combineChannels(rDec, gDec, bDec)
                ;
            // Save, calculate PSNR, compression ratio...
        }
    }

    // --- YUV PIPELINE ---
    List<int[]> yVecs = new ArrayList<>(), uVecs = new ArrayList<>(), vVecs
         = new ArrayList<>();
    for (File folder : trainDir.listFiles()) {
        for (File imgFile : folder.listFiles()) {
            BufferedImage img = ImageUtils.loadImage(imgFile.getAbsolutePath
                ());
            int[][][] yuv = ImageUtils.rgbToYuv(img);
            int[][] Y = yuv[0];
            int[][] U = ImageUtils.downsample(yuv[1]);
            int[][] V = ImageUtils.downsample(yuv[2]);
            yVecs.addAll(VQEncoderDecoder.extractVectors(Y));
            uVecs.addAll(VQEncoderDecoder.extractVectors(U));
            vVecs.addAll(VQEncoderDecoder.extractVectors(V));
        }
    }
    int[][] yCodebook = VectorQuantizer.kMeans(yVecs, 256, 20);
    int[][] uCodebook = VectorQuantizer.kMeans(uVecs, 256, 20);
    int[][] vCodebook = VectorQuantizer.kMeans(vVecs, 256, 20);

```

```
        for (File folder : testDir.listFiles()) {
            for (File imgFile : folder.listFiles()) {
                BufferedImage img = ImageUtils.loadImage(imgFile.getAbsolutePath
                    ());
                int[][][] yuv = ImageUtils.rgbToYuv(img);
                int[][] Y = yuv[0];
                int[][] U = ImageUtils.downsample(yuv[1]);
                int[][] V = ImageUtils.downsample(yuv[2]);
                int[][] yIdx = VQEncoderDecoder.compress(Y, yCodebook);
                int[][] uIdx = VQEncoderDecoder.compress(U, uCodebook);
                int[][] vIdx = VQEncoderDecoder.compress(V, vCodebook);
                int[][] yDec = VQEncoderDecoder.decompress(yIdx, yCodebook);
                int[][] uDec = VQEncoderDecoder.decompress(uIdx, uCodebook);
                int[][] vDec = VQEncoderDecoder.decompress(vIdx, vCodebook);
                int[][] uUp = ImageUtils.upsample(uDec, Y.length, Y[0].length);
                int[][] vUp = ImageUtils.upsample(vDec, Y.length, Y[0].length);
                BufferedImage outYUV = ImageUtils.yuvToRgb(yDec, uUp, vUp);
                // Save, calculate PSNR, compression ratio...
            }
        }
    }
}
```

5 Supporting Code: Utilities and Quantization

5.1 ImageUtils.java: Color Conversion and Subsampling

```
// RGB to YUV conversion
public static int[][][] rgbToYuv(BufferedImage img) {
    // ... conversion logic ...
}
// YUV to RGB conversion
public static BufferedImage yuvToRgb(int[][] Y, int[][] U, int[][] V) {
    // ... conversion logic ...
}
// Downsample (chroma subsampling)
public static int[][] downsample(int[][] channel) {
    // ... average 2x2 blocks ...
}
// Upsample (nearest neighbor)
public static int[][] upsample(int[][] channel, int targetH, int targetW) {
    // ... duplicate pixels ...
}
```

5.2 VectorQuantizer.java: k-Means Quantization

```
public static int[][] kMeans(List<int[]> data, int k, int maxIter) {
    // Initialize centroids
    // Repeat for maxIter:
    // Assign each vector to nearest centroid
    // Update centroids as mean of assigned vectors
    // Return centroids
}
```

5.3 VQEncoderDecoder.java: Vector Extraction, Compression, Decompression

```
// Extract 2x2 vectors from a channel
public static List<int[]> extractVectors(int[][] channel) { ... }
// Compress a channel using codebook
public static int[][] compress(int[][] channel, int[][] codebook) { ... }
// Decompress using codebook
public static int[][] decompress(int[][] indices, int[][] codebook) { ... }
```

6 Conclusion

This report demonstrates that YUV chroma subsampling with vector quantization achieves much higher compression than RGB (2x), with only a minor drop in PSNR. This matches real-world standards for image and video compression.

End of Report