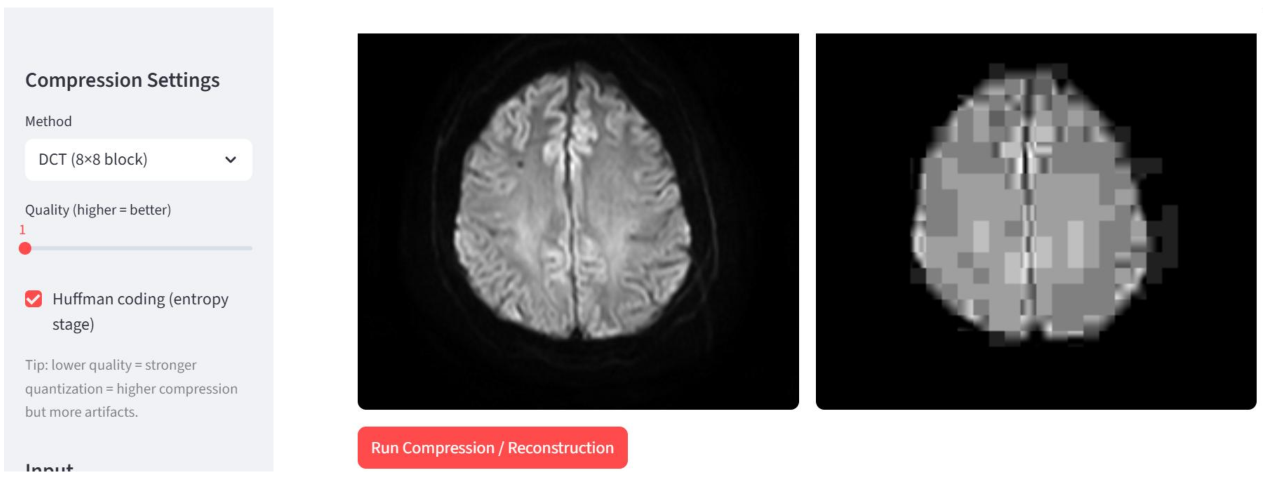
Lossy Compression

Python version

DCT



Compression Settings

Method

DCT (8×8 block)

Quality (higher = better)

1

☑ Huffman coding (entropy stage)

Tip: lower quality = stronger quantization = higher compression but more artifacts.

Debug Info

Original image range: 0 - 255

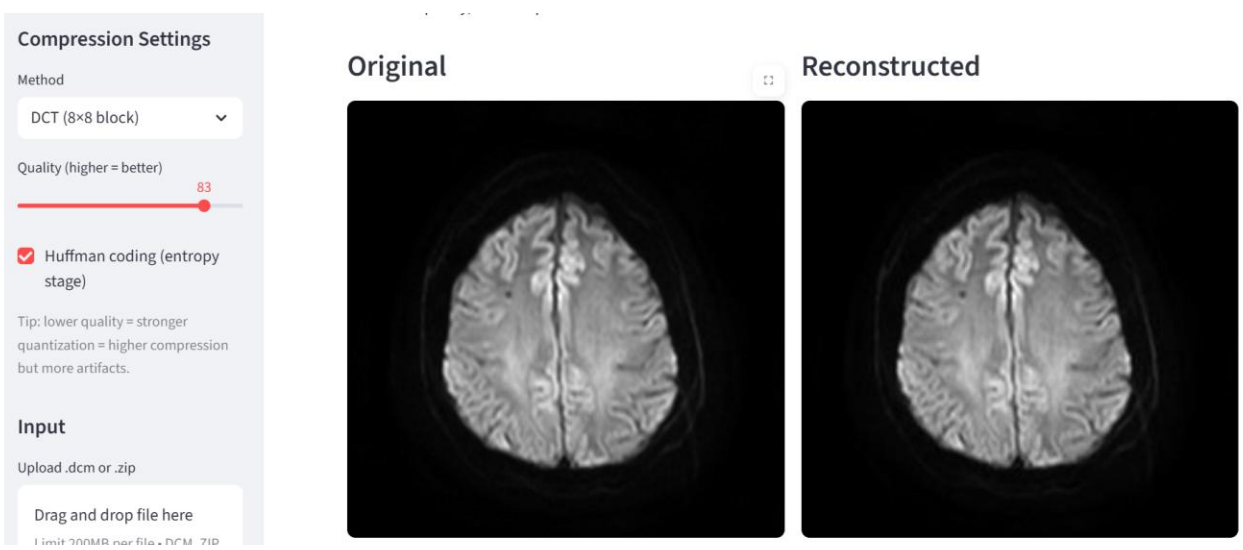
Reconstructed image range: 0 - 255

Original shape: (224, 224), Reconstructed shape: (224, 224)

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
151.12	26.34	0.9802	72.47×

Fig.1



Compression Settings

Method: DCT (8×8 block)

Quality (higher = better): 83

[✓] Huffman coding (entropy stage)

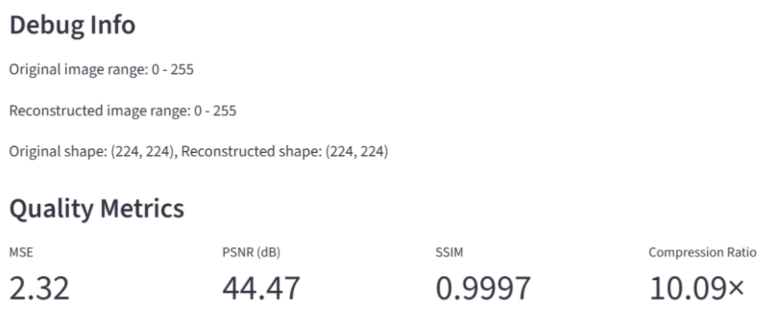
Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Limit 200MB per file • DCM, ZIP



Debug Info

Original image range: 0 - 255

Reconstructed image range: 0 - 255

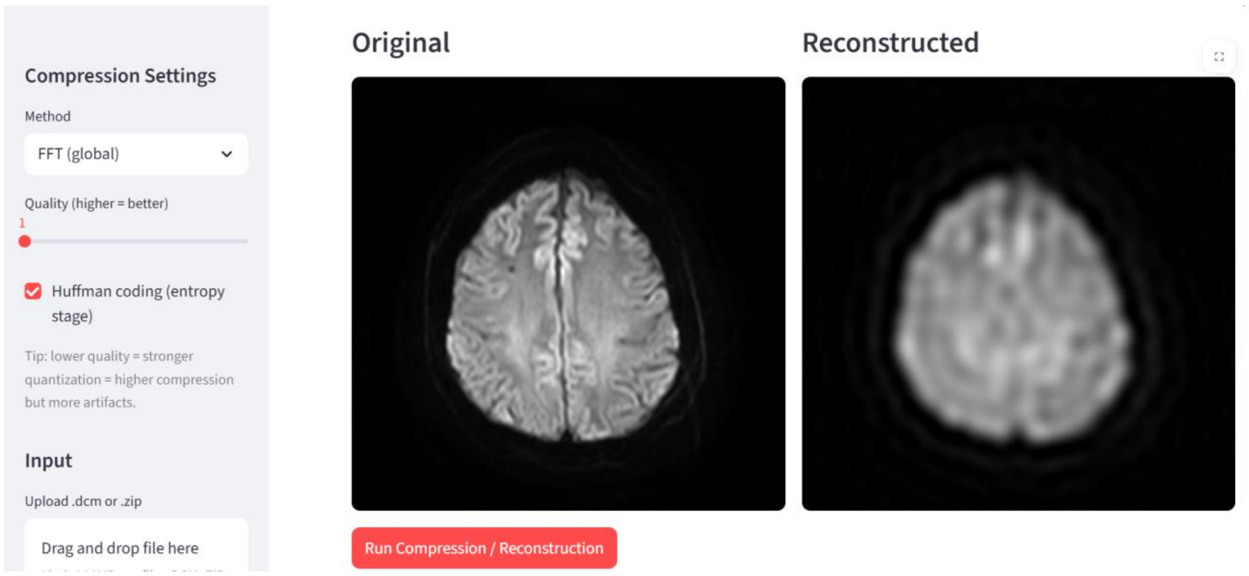
Original shape: (224, 224), Reconstructed shape: (224, 224)

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
2.32	44.47	0.9997	10.09×

Fig.2

FFT



Compression Settings

Method

FFT (global)

Quality (higher = better)

1

☑ Huffman coding (entropy stage)

Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Debug Info

Original image range: 0 - 255

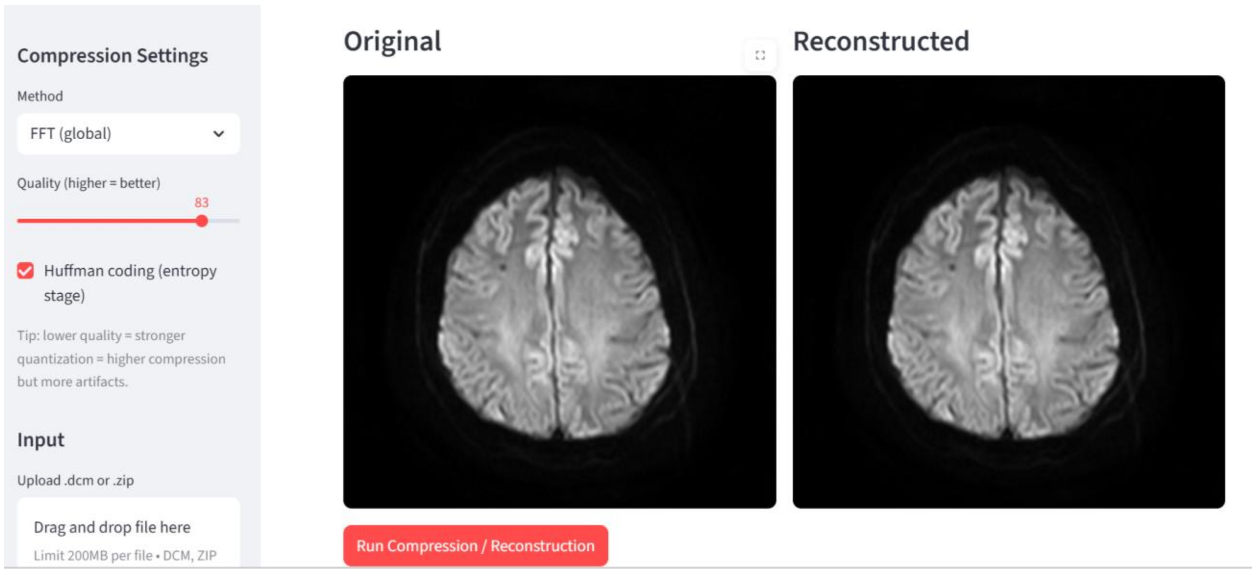
Reconstructed image range: 0 - 209

Original shape: (224, 224), Reconstructed shape: (224, 224)

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
129.25	27.02	0.9832	28.29×

Fig.3


Compression Settings

Method

FFT (global)

Quality (higher = better)

83

[✓] Huffman coding (entropy stage)

Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Limit 200MB per file • DCM, ZIP

Debug Info

Original image range: 0 - 255

Reconstructed image range: 0 - 255

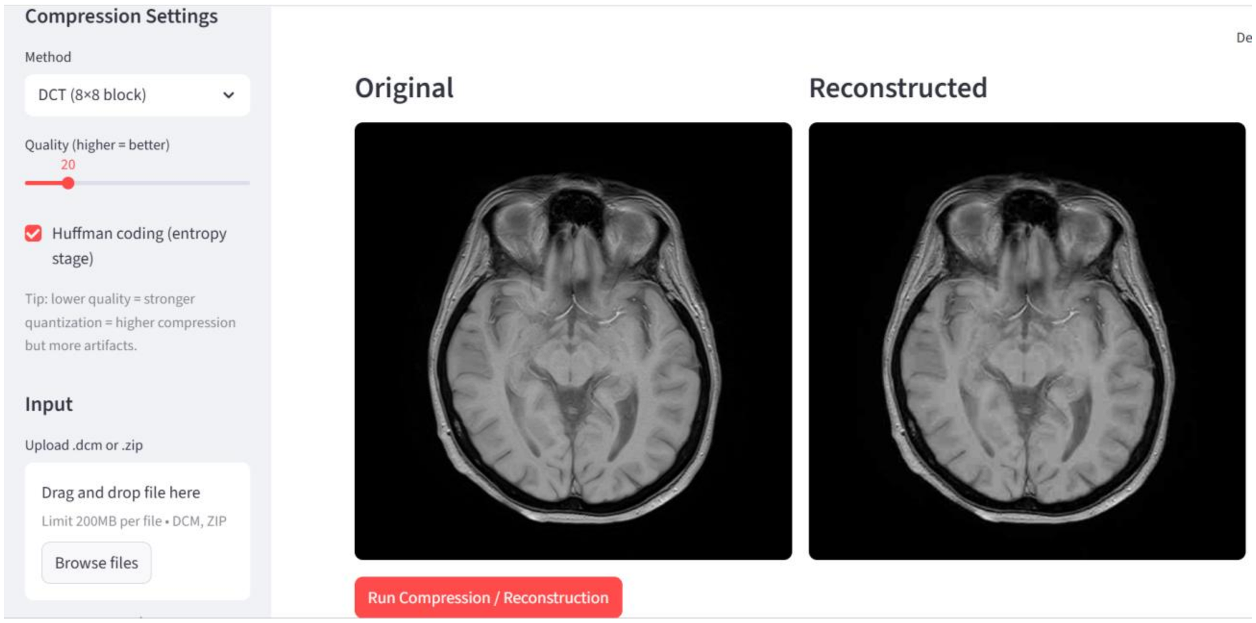
Original shape: (224, 224), Reconstructed shape: (224, 224)

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
0.32	53.09	0.9999	0.62×

Fig.4

DCT


Compression Settings

Method

DCT (8×8 block)

Quality (higher = better)

20

Huffman coding (entropy stage) [✓]

Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Limit 200MB per file • DCM, ZIP

Debug Info

Original image range: 0 - 255

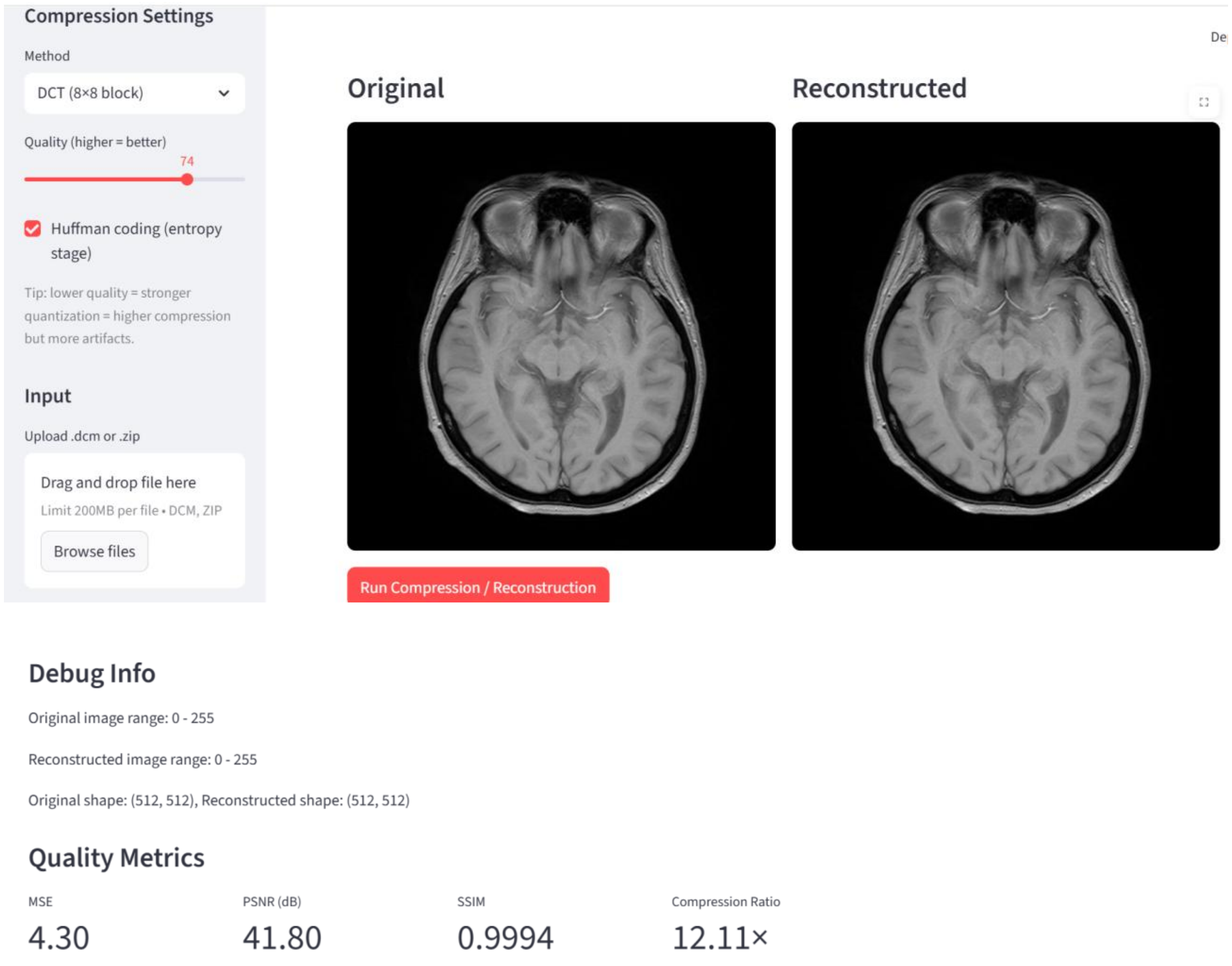
Reconstructed image range: 0 - 255

Original shape: (512, 512), Reconstructed shape: (512, 512)

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
18.56	35.45	0.9975	24.99×

Fig.5


Compression Settings

Method

DCT (8×8 block)

Quality (higher = better)

74

Huffman coding (entropy stage)

Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Limit 200MB per file • DCM, ZIP

Browse files

Original

Reconstructed

Run Compression / Reconstruction

Debug Info

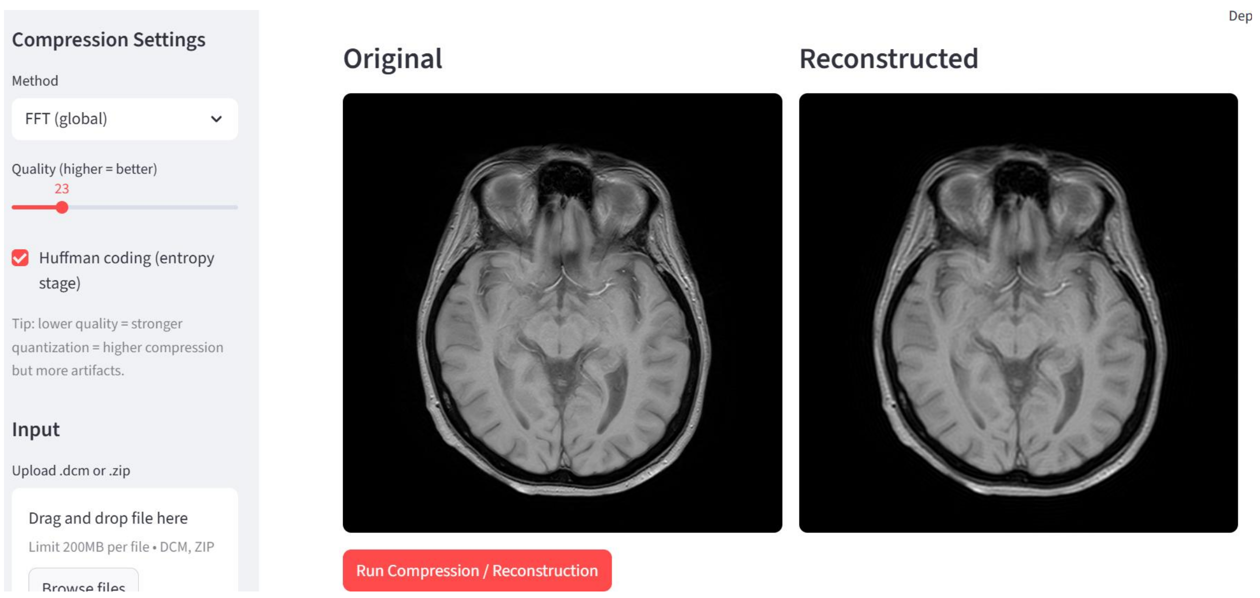
```
Original image range: 0 - 255

Reconstructed image range: 0 - 255

Original shape: (512, 512), Reconstructed shape: (512, 512)
```

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
4.30	41.80	0.9994	12.11×

Fig.6

FFT



Compression Settings

Method

FFT (global)

Quality (higher = better)

23

☑ Huffman coding (entropy stage)

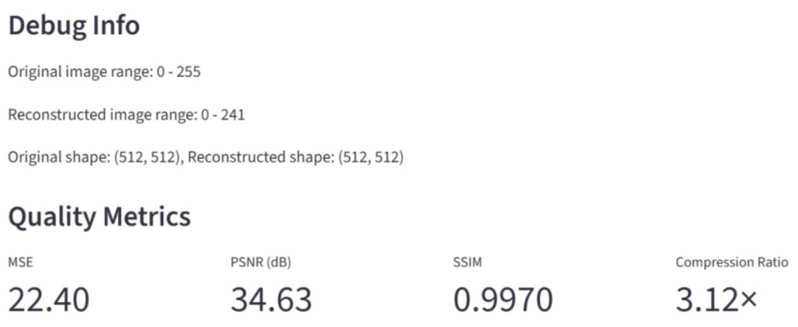
Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Limit 200MB per file • DCM, ZIP



Debug Info

Original image range: 0 - 255

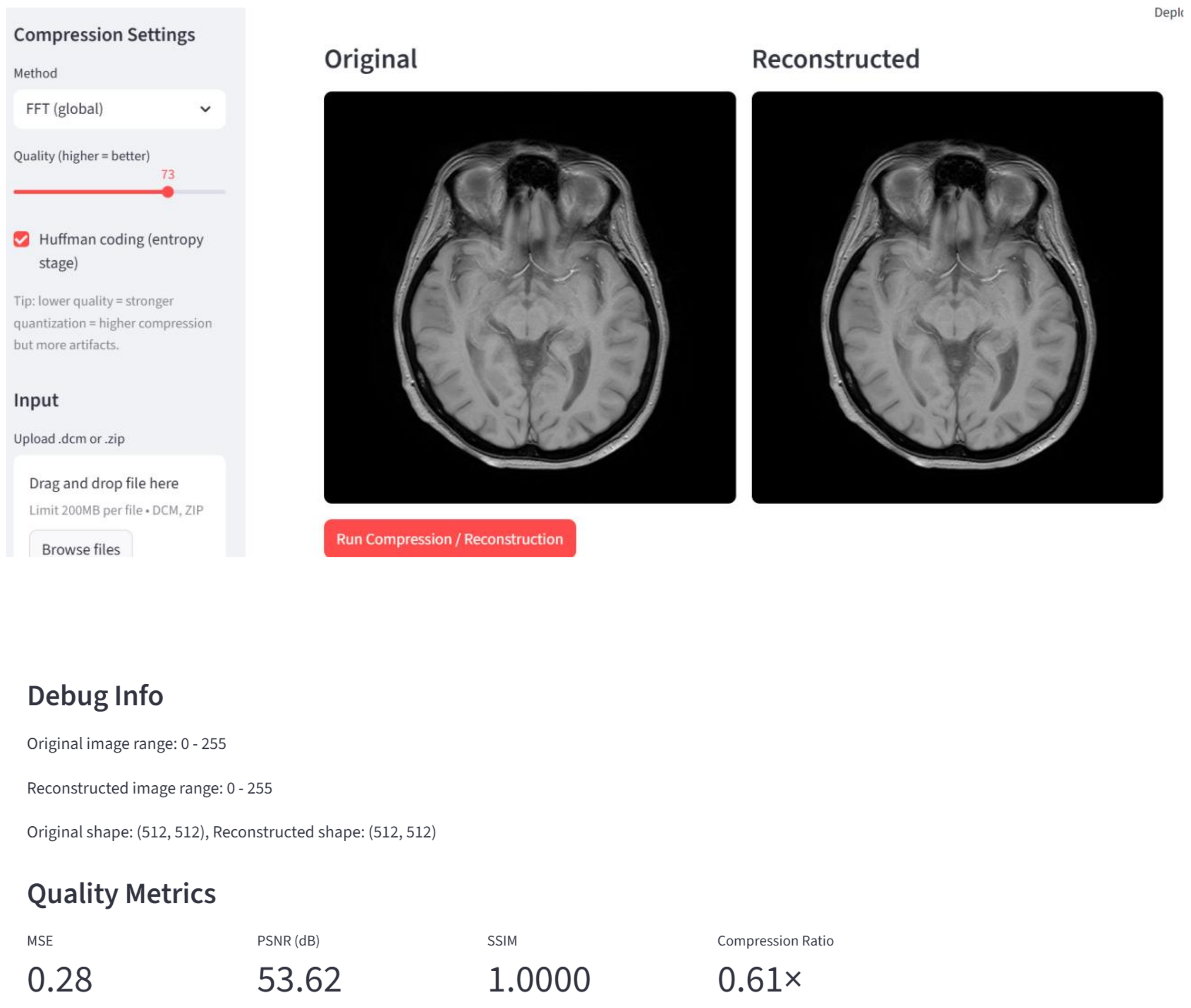
Reconstructed image range: 0 - 241

Original shape: (512, 512), Reconstructed shape: (512, 512)

Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
22.40	34.63	0.9970	3.12×

Fig.7



Compression Settings

Method

FFT (global)

Quality (higher = better)

73

☑ Huffman coding (entropy stage)

Tip: lower quality = stronger quantization = higher compression but more artifacts.

Input

Upload .dcm or .zip

Drag and drop file here

Limit 200MB per file • DCM, ZIP

Debug Info

Original image range: 0 - 255

Reconstructed image range: 0 - 255

Original shape: (512, 512), Reconstructed shape: (512, 512)

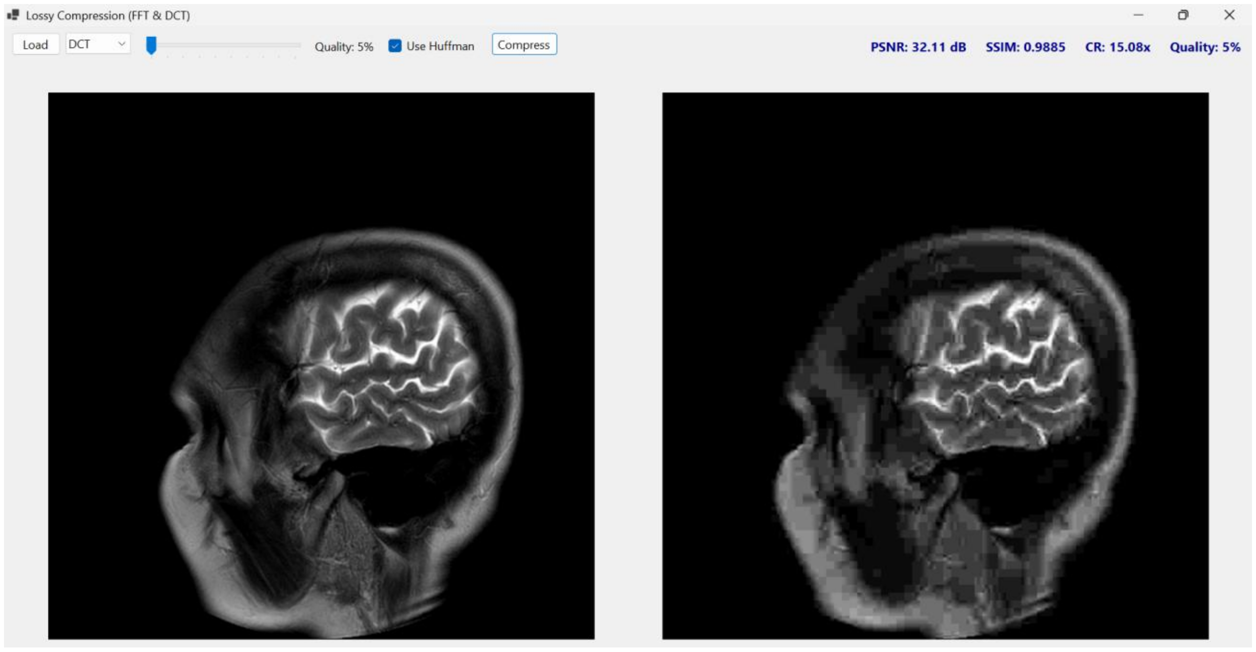
Quality Metrics

MSE	PSNR (dB)	SSIM	Compression Ratio
0.28	53.62	1.0000	0.61×

Fig.8

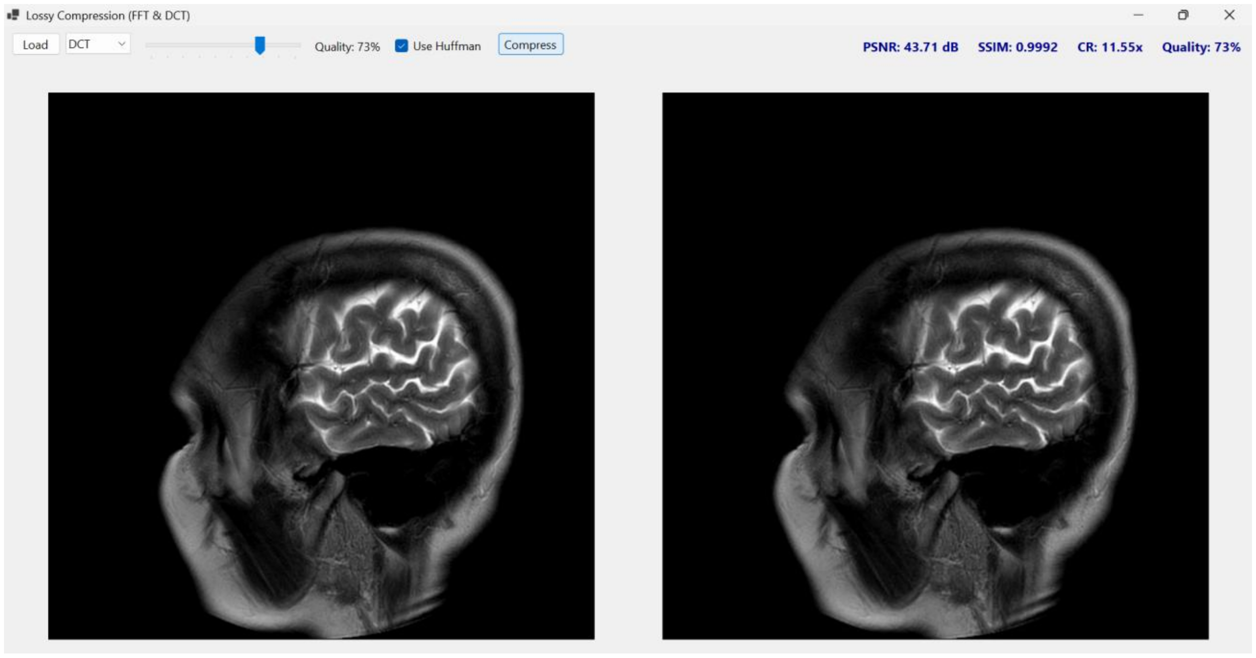
C# version

DCT



PSNR: 32.11 dB SSIM: 0.9885 CR: 15.08x Quality: 5%

Fig.9

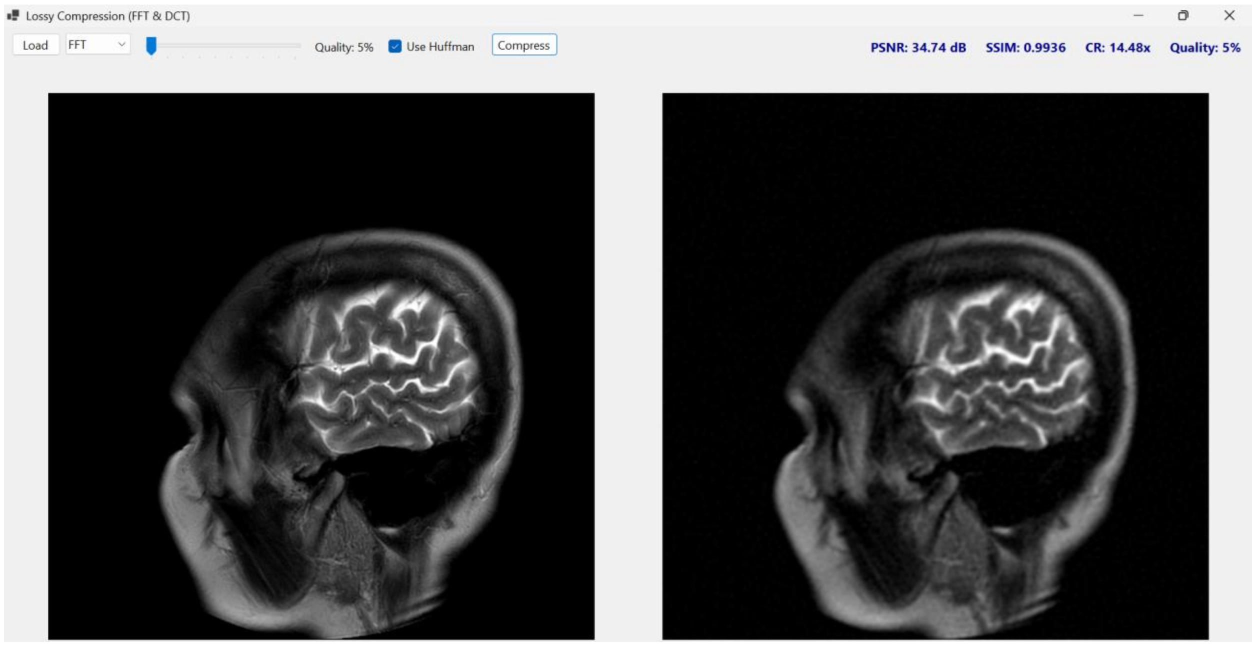


Quality: 73% Use Huffman

PSNR: 43.71 dB SSIM: 0.9992 CR: 11.55x Quality: 73%

Fig.10

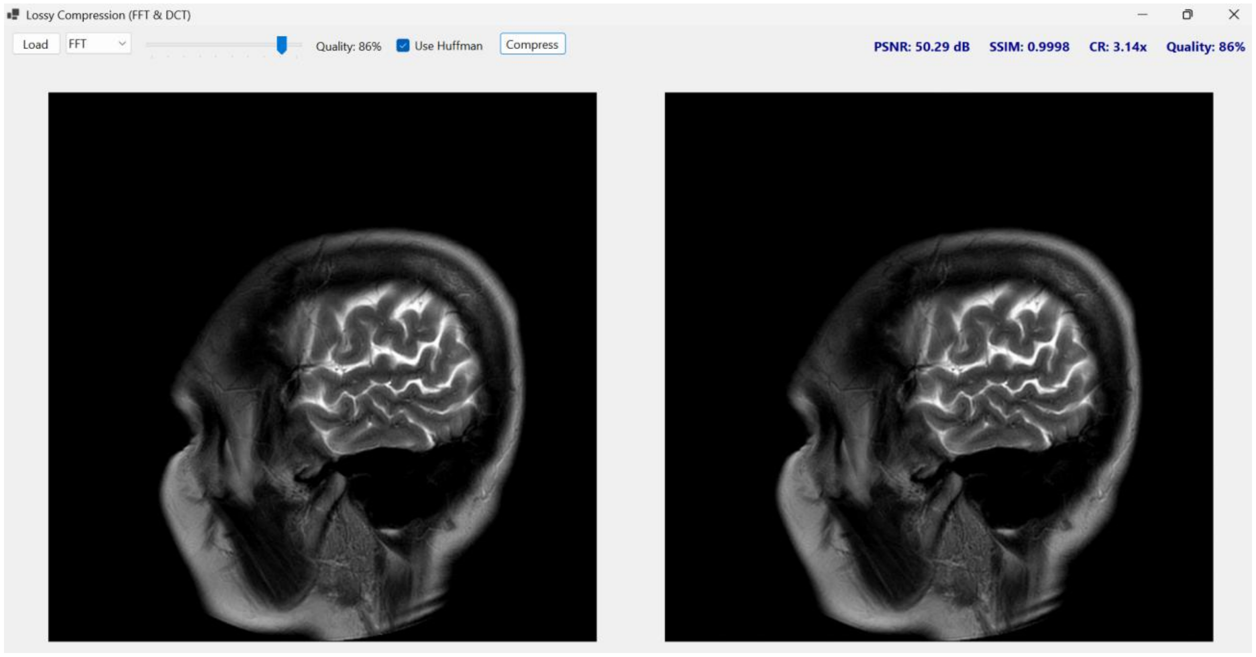
FFT



Quality: 5% Use Huffman

PSNR: 34.74 dB SSIM: 0.9936 CR: 14.48x Quality: 5%

Fig.11

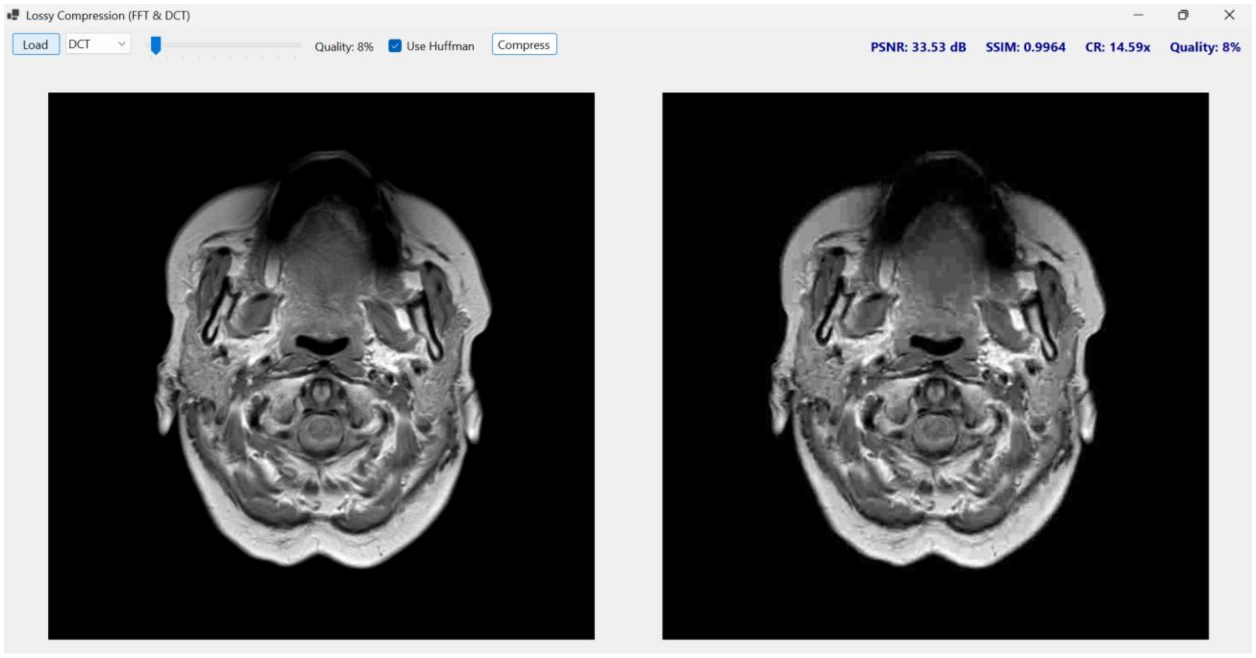


Quality: 86% Use Huffman

PSNR: 50.29 dB SSIM: 0.9998 CR: 3.14x Quality: 86%

Fig.12

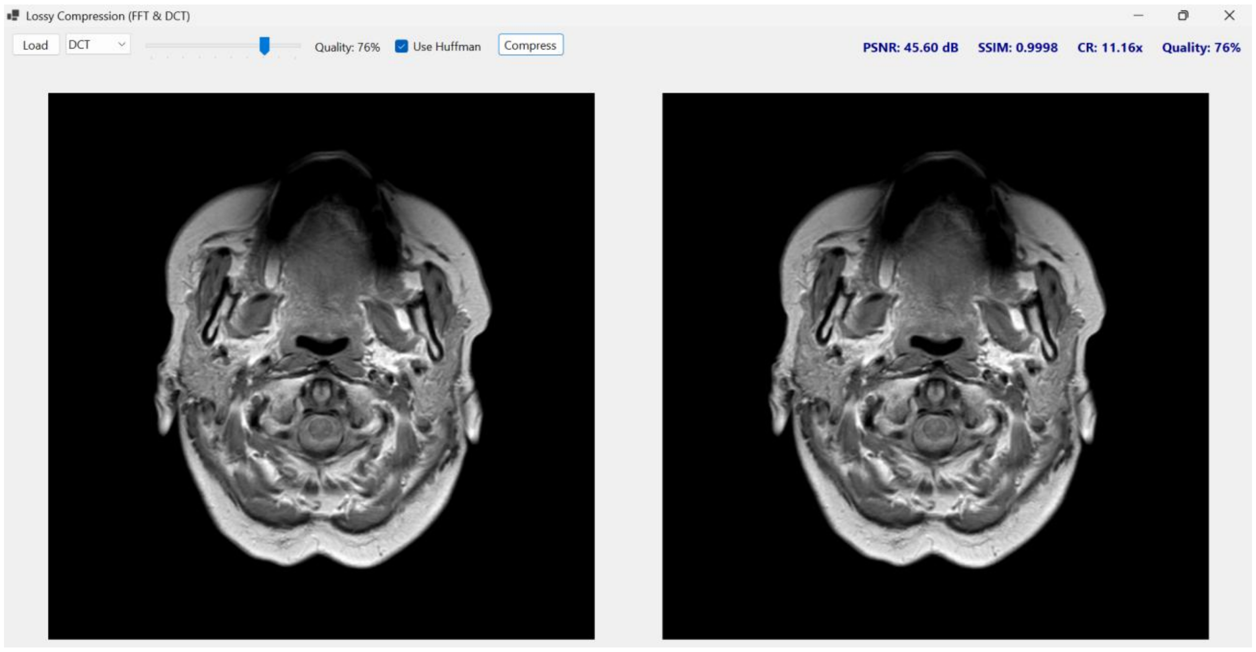
DCT



Quality: 8% ✓ Use Huffman

PSNR: 33.53 dB SSIM: 0.9964 CR: 14.59x Quality: 8%

Fig.13

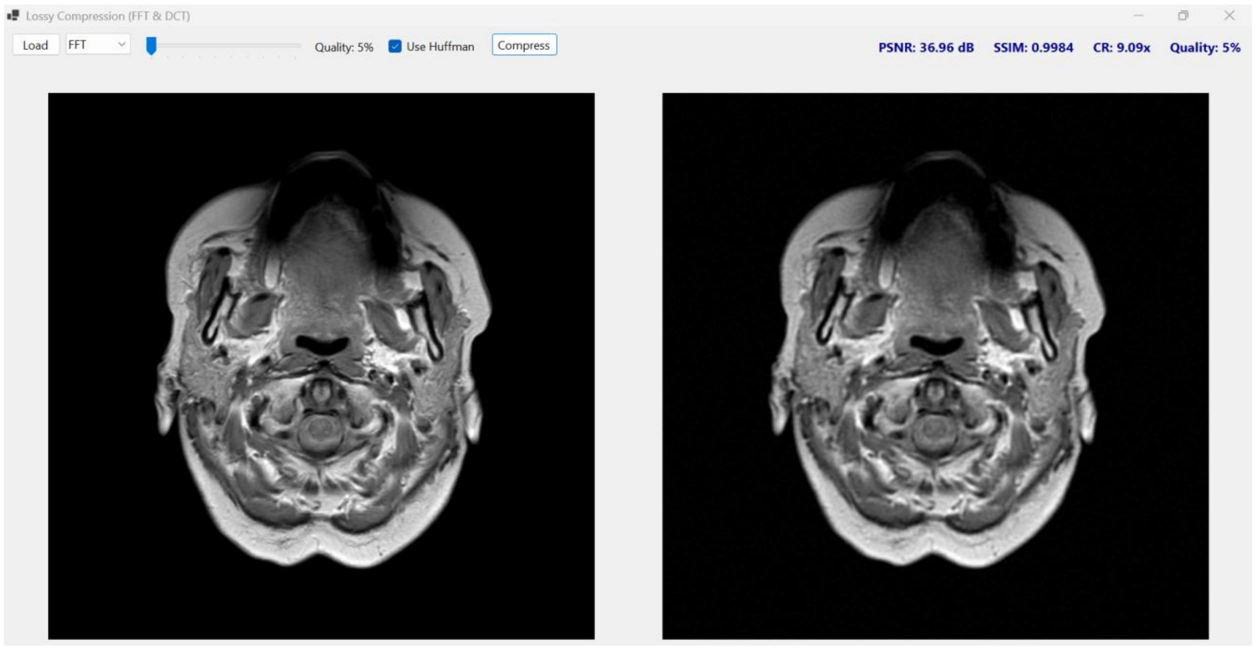


Quality: 76% Use Huffman

PSNR: 45.60 dB SSIM: 0.9998 CR: 11.16x Quality: 76%

Fig.14

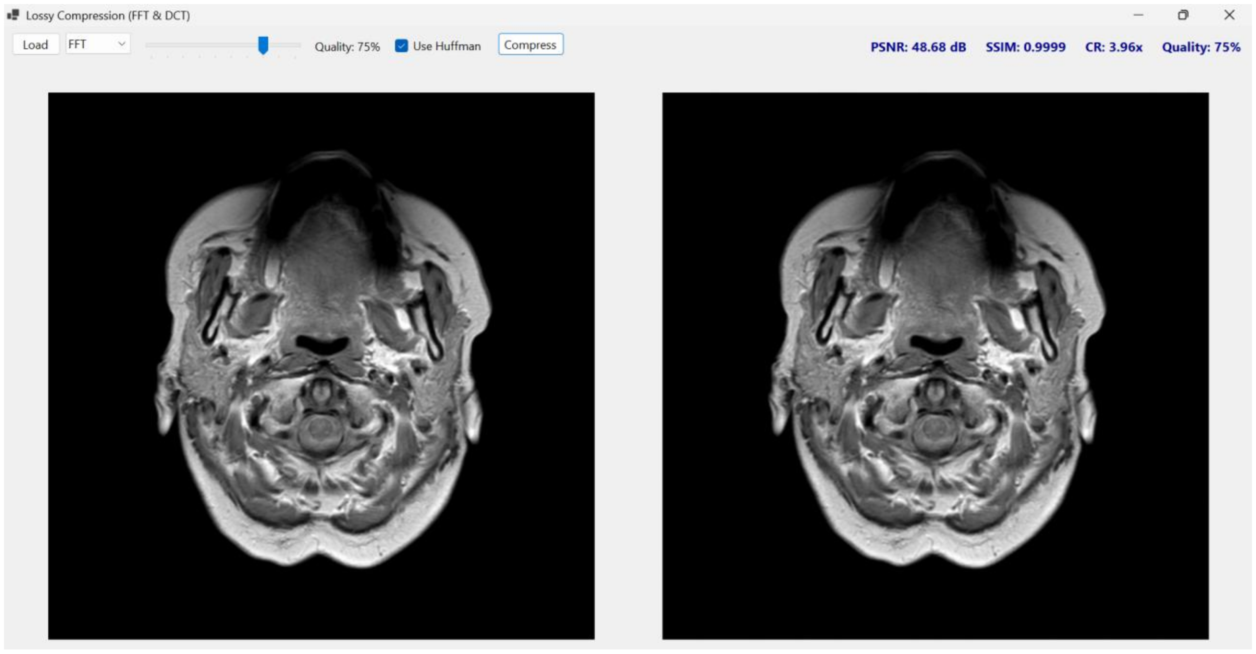
FFT



FFT Quality: 5% Use Huffman ✓

PSNR: 36.96 dB SSIM: 0.9984 CR: 9.09x Quality: 5%

Fig.15



FFT Quality: 75% Use Huffman ✓

PSNR: 48.68 dB SSIM: 0.9999 CR: 3.96x Quality: 75%

Fig.16

Discussion of the Project

In this project I created a full lossy-compression pipeline for medical images using two different programming languages: Python and C#. Both versions follow the same steps. First the image is transformed using either DCT or FFT, then the transformed values are quantised, and finally Huffman coding is applied at the end to squeeze the data even more. I tested everything on several MRI images from the DICOMS folder, and some of the results are shown in the figures in this report.

When the quality setting is high, the reconstructed images look almost the same as the originals. For example, in the Python DCT results (Fig.2) and the Python FFT results (Fig.4), the differences are basically invisible. The PSNR values were around 50 dB and the SSIM was very close to 1.0, which normally means that the image is almost identical to the original. The C# results show the same behaviour. Both the DCT output (Fig.10) and FFT output (Fig.12) look extremely clean at high quality. The only downside at high quality is that the compression ratio stays quite small.

When I lower the quality, the artefacts from the two methods start to look different. In the DCT results (for example Fig.5 in Python and Fig.13 in C#), block artefacts start to appear, especially around edges. This happens because DCT works block-by-block. In contrast, the FFT results (Fig.7 for Python and Fig.15 for C#) don't have blockiness, but they become smoother and sometimes a little blurry. FFT also produces slight ringing artefacts at low quality. Even then, the main structure of the MRI images remains very clear. At these lower settings, the PSNR dropped to around 30 dB, but the SSIM still stayed quite high.

A key requirement in the assignment was adding Huffman coding at the end of the pipeline. Huffman is lossless, so it does not change the reconstructed images at all. The quality stays the same in all the figures. What it does change is the compression ratio. In my results, adding Huffman visibly increased the compression ratio while the images stayed exactly the same. This shows why Huffman is used after quantisation: it gives extra compression “for free” without harming image quality.

Since the task required using two different programming languages, it was interesting to compare Python and C#. The overall results were extremely similar. The Python DCT images and C# DCT images behave almost the same. The same is true for FFT between Python and C#This consistency shows that the compression pipeline works correctly in both implementations.

From experimenting with both transforms, I can say that FFT’s main advantage is that it avoids block artefacts since it processes the whole image at once. Even at medium compression levels, FFT images generally look smoother. On the other hand, DCT is often better at achieving higher compression ratios, because most of the important energy sits in the low-frequency part of each block. The downside is that when the quality becomes too low, DCT’s blockiness becomes very noticeable. So overall, FFT looks cleaner, while DCT compresses more efficiently.

In the end, both the Python and C# implementations worked well and produced clear differences between the two transforms. Huffman coding improved the compression ratios significantly without hurting image quality at all. The results across all the figures in this report show that it is possible to compress MRI images heavily while still keeping the important details intact.