CS663 Project

Adit Srivastava: 22B1269
Jay Mehta: 22B1281
Tanay Bhat: 22B3303

November 2024

Contents

Chapter 1

JPEG For Greyscale Images

1.1 Introduction

The JPEG alogorithm essential comprises of 3 parts: Discrete Cosine Transform, Quantization and Encoding. The quantization step causes loss of information while the encoding step helps compress the image. We shall implement our own version of the JPEG algorithm for greyscale images in python and descibe the results. We also compare the resuts with the inbuilt JPEG algorithm of MATLAB, OpenCV and Python Imaging Library (PIL). We used a dataset of 1500 images of birds called CALTECH-BMP for this purpose.

1.2 Metrics for Comparison

We shall employ the following metrics to compare the results of our implementation with the inbuilt algorithms:

- PSNR (Peak Signal to Noise Ratio)
- SSIM (Structural Similarity Index)
- MSE (Mean Squared Error)
- Compression Ratio (Input Size / Output Size)
- RMSE Vs BPP (Root Mean Squared Error Vs Bits Per Pixel)

This way we get a comprehensive comparison of the results. Let us now look at the results of our implementation.

1.3 Results

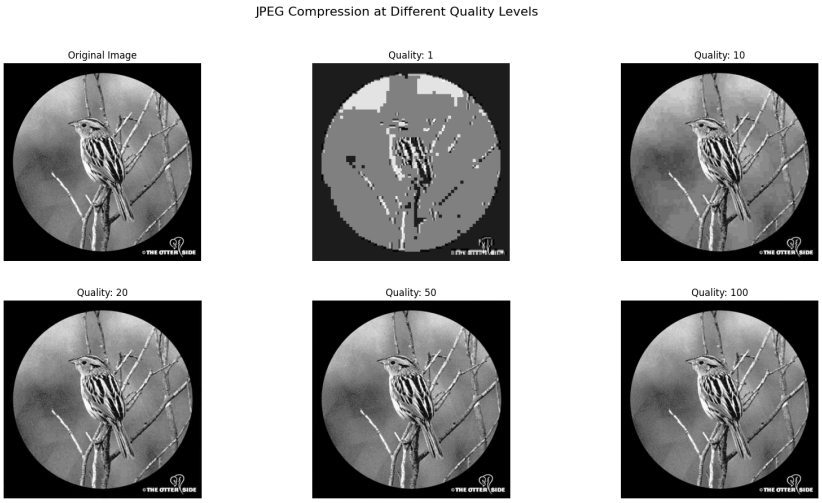


Figure 1.1: Image at various Q values

We see that this image has a large circle at the border. This causes regions of high contrast near the border. Because of this we shall see that the SSIM values fluctuate alot as we vary the qualtiy of compression.

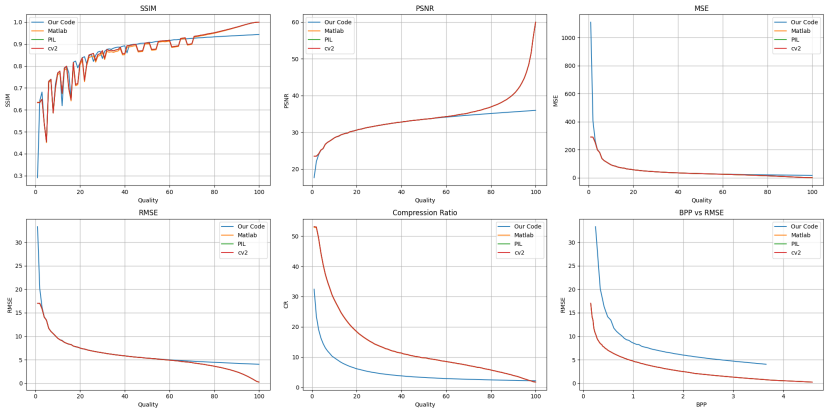


Figure 1.2: Metrics Vs Q

We see that the PSNR obtained by inbuilt implementations of JPEG is about 20 dB higher than our implementation at higher Q values while maintaining a a better compression ratio (Since we have defined compression ratio as the ratio of input size to output size, a higher value is better). We do see quite similar

MSE values for all implementations and the fluctuation in SSIM values is also similar and expected. Let us now look at an image with less contrast.

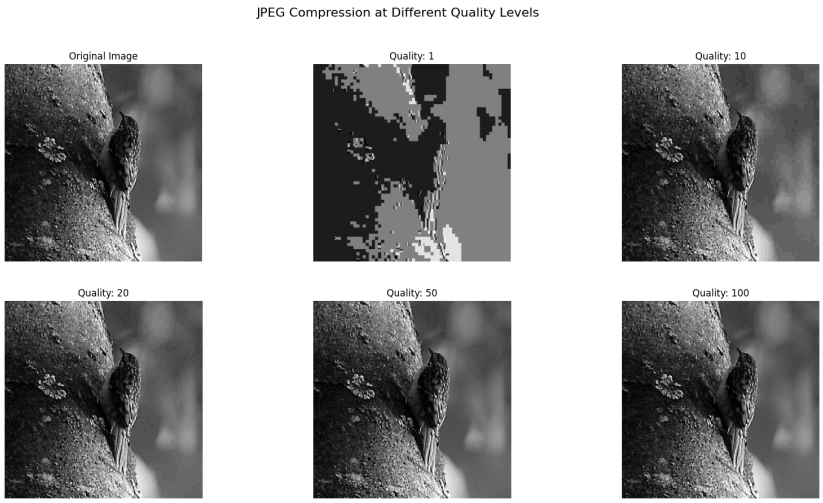


Figure 1.3: Image at various Q values

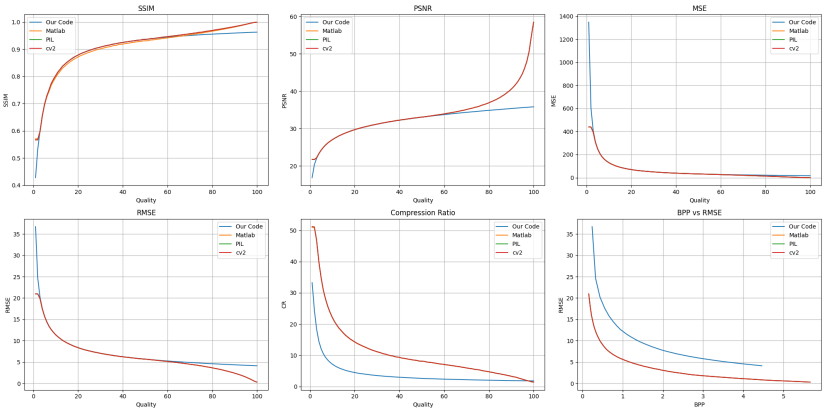
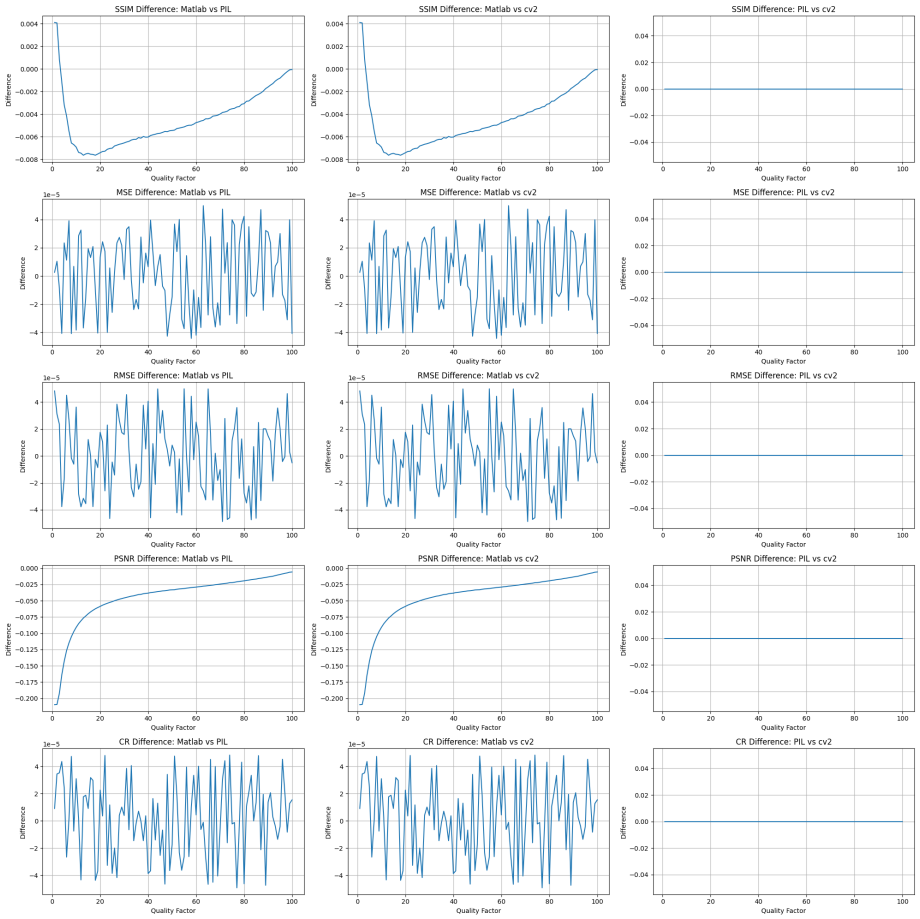


Figure 1.4: Metrics Vs Q

Notice how the varitions in SSIM values are much less in this case. This is because the image has less contrast and hence the SSIM values are more stable. We see that the PSNR values are still higher for the inbuilt implementations of JPEG. The MSE values are also similar. The compression ratio is also better for the inbuilt implementations. It is interesting to note that the the inbuilt implementations seem to follow eaach other quite closely. Let us examine this a bit more.

1.4 Comparison of Inbuilt Implementations

We shall use the metrics obtained by inbuilt implementations of JPEG to compare them with each other by plotting their differences against the Q value.



Here the plot titles A vs B imply that $A - B$ was plotted against quality. We observe that PIL and OpenCV plots are exactly the same as the difference is 0. The MATLAB plot is also very close to the PIL and OpenCV plots however there are some differences as seen by the variations in the metrics. This variations are quite small as seen by the y axis scale and are hence effectively invisible to the naked eye. By plotting difference instead of square difference or absolute difference, we can see places where one implementation performs better than the other. For instance we see that for approximately $Q > 5$, MATLAB has worse performance whencompared on the basis of SSIM. We also see that MATLAB performs worse with respect to PSNR as the plot is always negative.

1.5 RMSE Vs BPP

We also plotted the RMSE values against the BPP values each implementation for 20 different images.

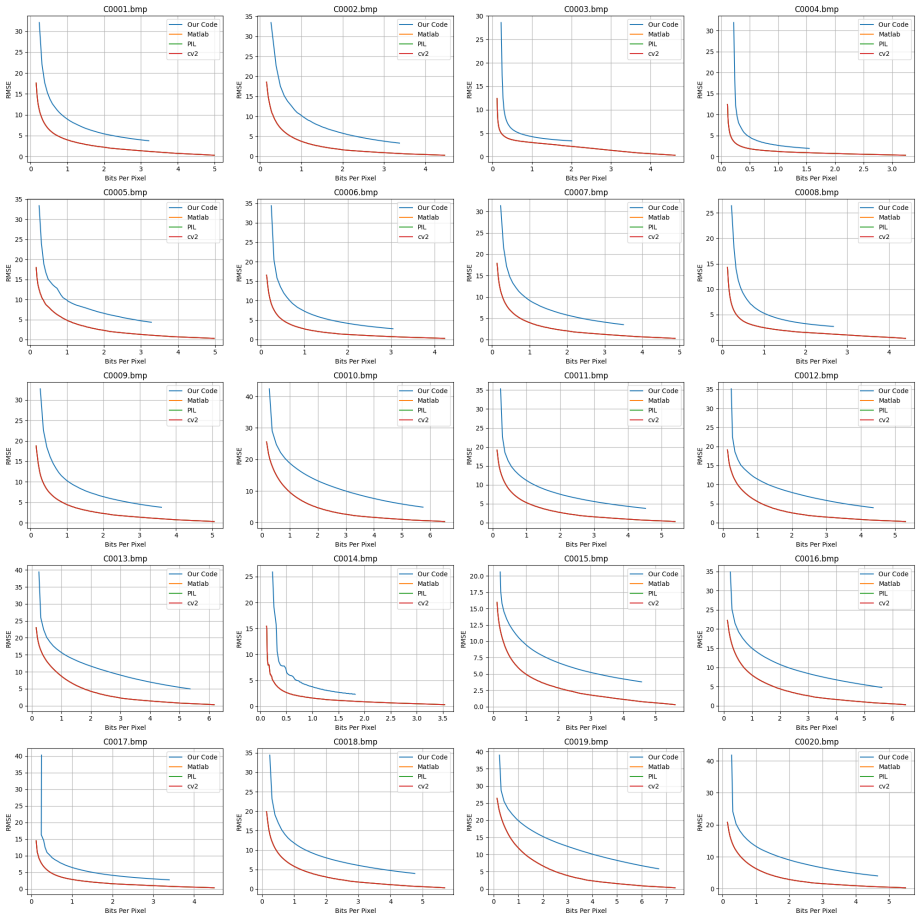


Figure 1.5: RMSE Vs BPP

We see that our algorithm performs worse in general which was expected as current implementations of JPEG are quite optimized. We also see that the inbuilt implementations of JPEG follow each other quite closely.

Chapter 2

JPEG For Colour Images

2.1 Introduction

Having looked at the JPEG algorithm for greyscale images, we shall now extend our implementation to colour images. Inorder to implement JPEG for colour images, we shall separate each channel of the image and compress them separately. We shall then combine the compressed channels to get the final compressed image. In general, the algorith is as follows:

- Convert the image to YCbCr colour space
- Separate the Y, Cb and Cr channels
- Downsample the Cb and Cr channels to half their size along both dimensions
- Apply DCT and quantization to each channel. Note that the quatization used for chrominance channels is generally more agressive than that used for the luminance channel.
- Encode the quantized values to binary files.
- Decode the binary files and combine the channels to get the final image.

The down sampling procedure is carried out only on the chrominance channels and not the luminance channel. This is because the human eye is more sensitive to changes in luminance than chrominance. Hence we can afford to lose some information in the chrominance channels. We shall now look at the results of our implementation. These channels are then upsampled to their original size during the decoding process.

2.2 Results

As with the greyscale images, we shall compare the results of our implementation with the inbuilt implementations of JPEG using the same metrics. First we shall show a few images and their compressed versions at different Q values.

Figure 2.1: Image C0123 at various Q values

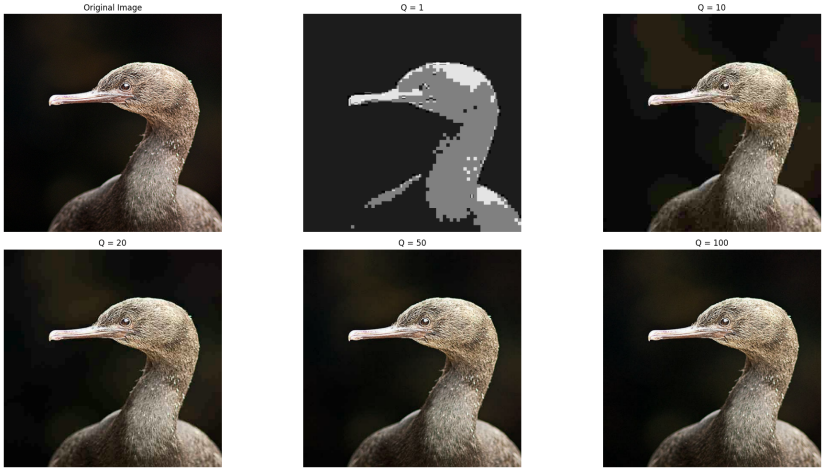


Figure 2.2: Image C0423 at various Q values

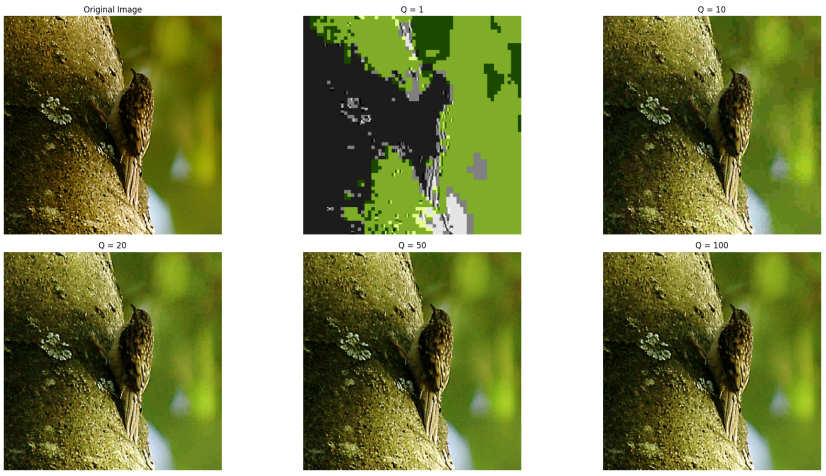


Figure 2.3: Image C0489 at various Q values

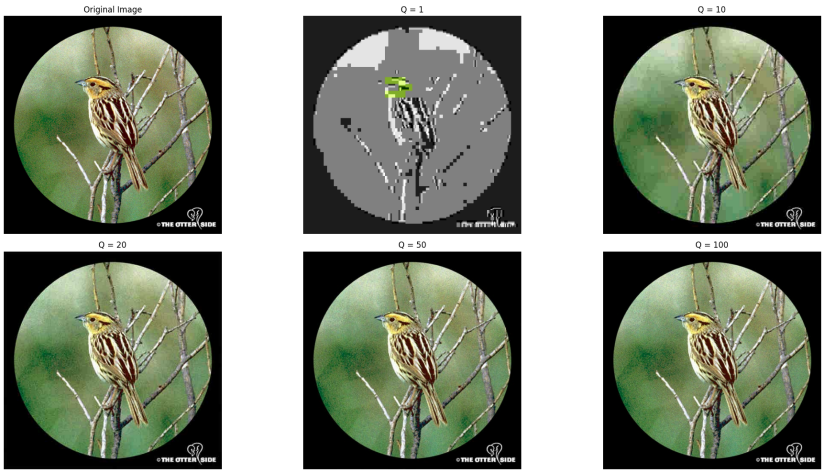


Figure 2.4: Image C1499 at various Q values

Notice how at $Q = 1$, the image is largely greyscale. This is because the chrominance channels have been quantized very agressively and hence most pixel values are close to 0. Hence the Luminance channel dominates. Lets now look at the metrics for images C1499 and C0489:

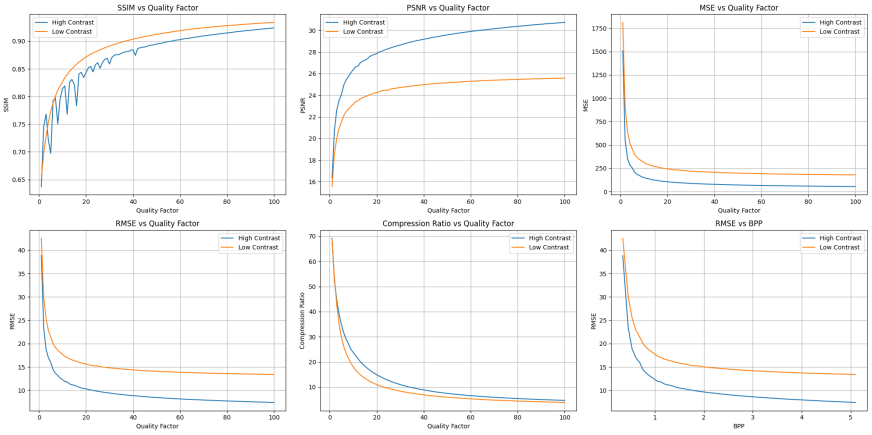


Figure 2.5: Metrics Vs Q

We describe C1499 as the high contrast image and C0489 as the low contrast image. Lets now compare with the inbuilt implementations.

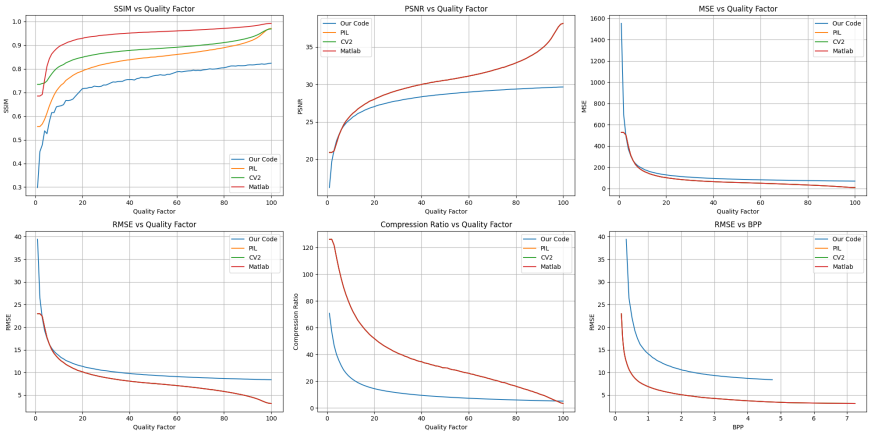


Figure 2.6: Metrics Vs Q

It is interesting to see that while the OpenCV and PIL implementations followed each other perfectlyy in the greyscale case, there is asignificant amount of difference between the two in the colour case when compared on the basis of SSIM. The other metrics are quite similar.

2.3 RMSE Vs BPP

We also plotted the RMSE values against the BPP values each implementation for 20 different images.

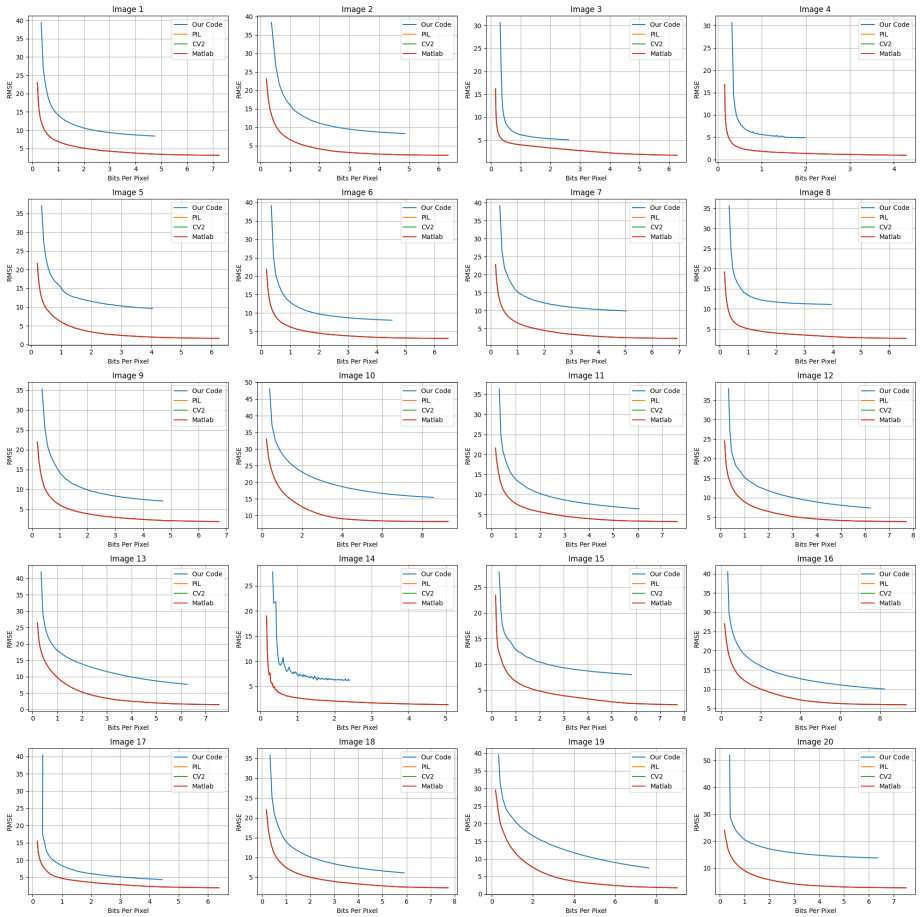


Figure 2.7: RMSE Vs BPP

We see that our algorithm performs worse in general which was expected as current implementations of JPEG are quite optimized. We also see that the inbuilt implementations of JPEG follow each other quite closely.

Chapter 3

ML Based Image Compression

3.1 Introduction

An autoencoder is a neural network architecture that can be used for compressing images. The autoencoder consists of an encoder and a decoder. The encoder (yellow) compresses the image into a lower dimensional representation called a latent vector (blue). The decoder (green) then reconstructs the image from the latent vector. The autoencoder is trained by minimizing the difference between the input image and the reconstructed image. We shall implement an autoencoder for compressing images for the MNIST fashion dataset as it requires less computational power and compare quality of compressed images using simliar metrics as before for various values of latent vector size.

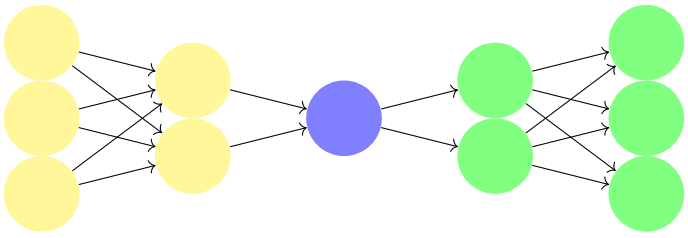


Figure 3.1: Autoencoder Architecture

3.2 Results

We did not use compression ratios as a metric for comparison as the images in the dataset are already quite small in size. Hence we evaluate compression using PSNR, SSIM, MSE and RMSE. We shall now look at the results.

Latent Vector Size	PSNR	SSIM	MSE	RMSE
64	20.73	0.75	654.26	24.49
128	20.81	0.76	641.16	24.24

Table 3.1: Comparison of Metrics for Different Latent Vector Sizes

We see that the values are quite similar even after doubling the latent vector size. This is because the images in the dataset are already quite small and hence the latent vector size does not have a significant impact on the quality of the compressed image. The main drawback of this method is that it is difficult to generalize. If the training dataset was images of birds, the autoencoder would not be able to compress and reconstruct images of other animals that well. Inorder to generalize, we would need a much larger dataset and a much larger neural network. We used a simple architecture for the autoencoder with a single hidden layer in the encoder and decoder.

Chapter 4

Conclusion

We shall conclude this report by presenting more images and their compressed versions using our greyscale and colour JPEG implementations. We shall also outline the contributions of each team member.

4.1 Images

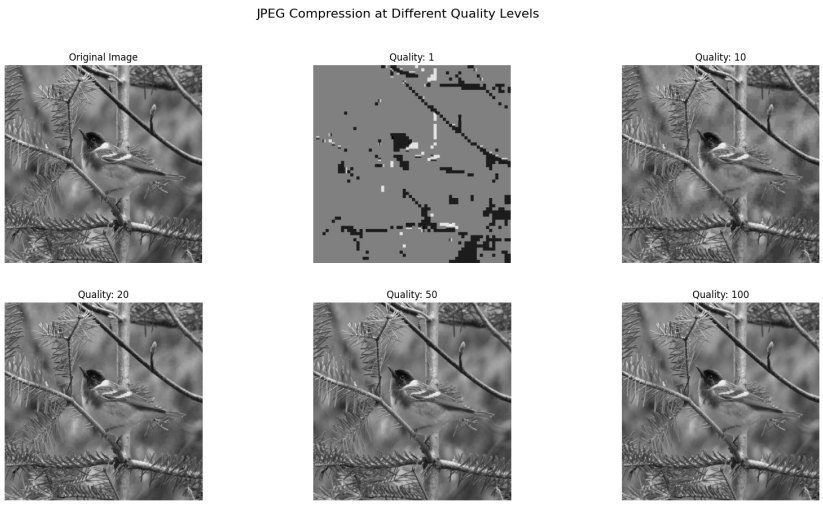


Figure 4.1: Greyscale 1

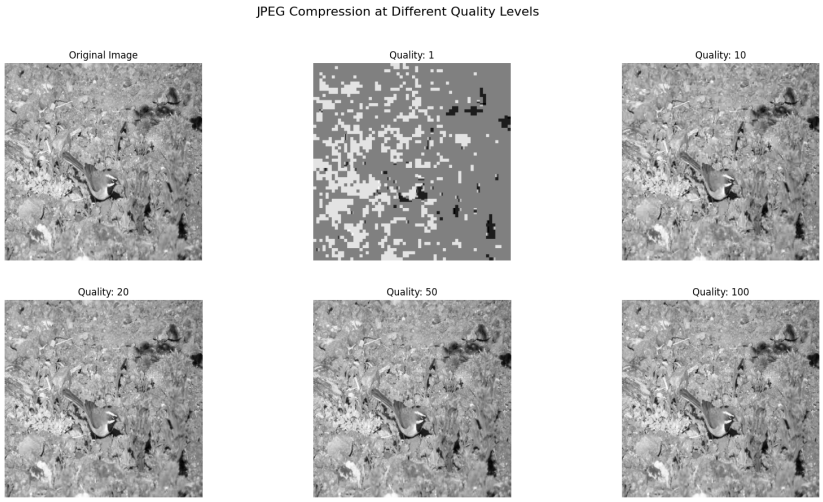


Figure 4.2: Greyscale 2

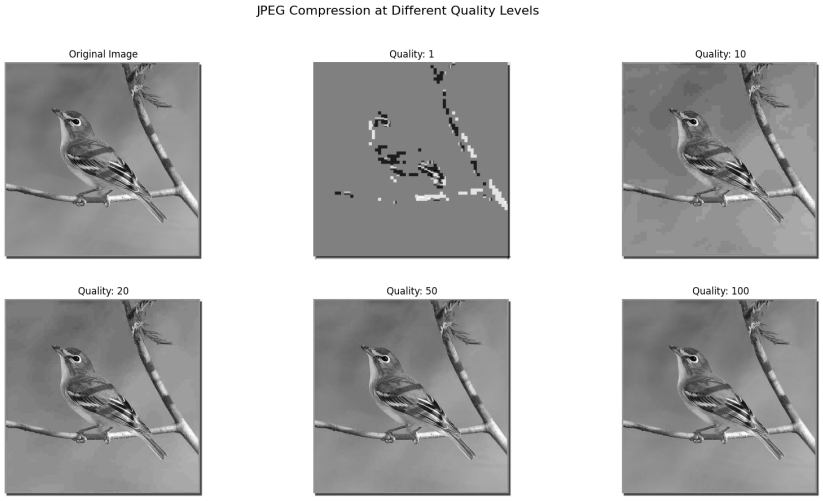


Figure 4.3: Greyscale 3

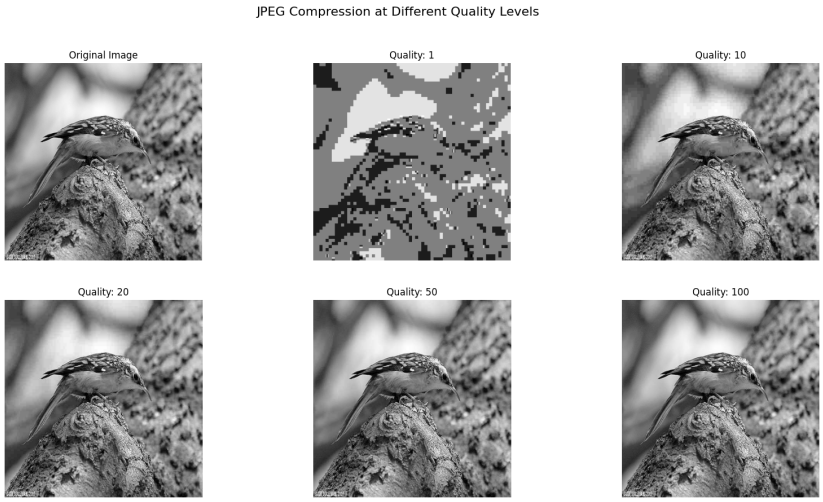


Figure 4.4: Greyscale 4

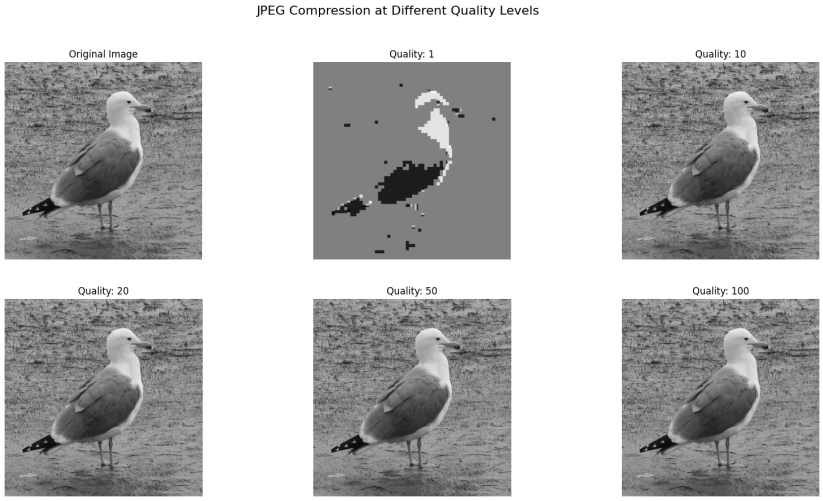


Figure 4.5: Greyscale 5

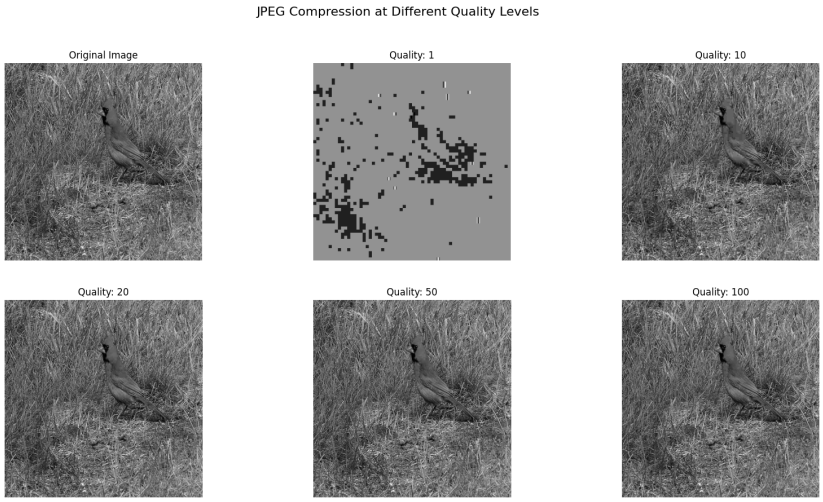


Figure 4.6: Greyscale 6

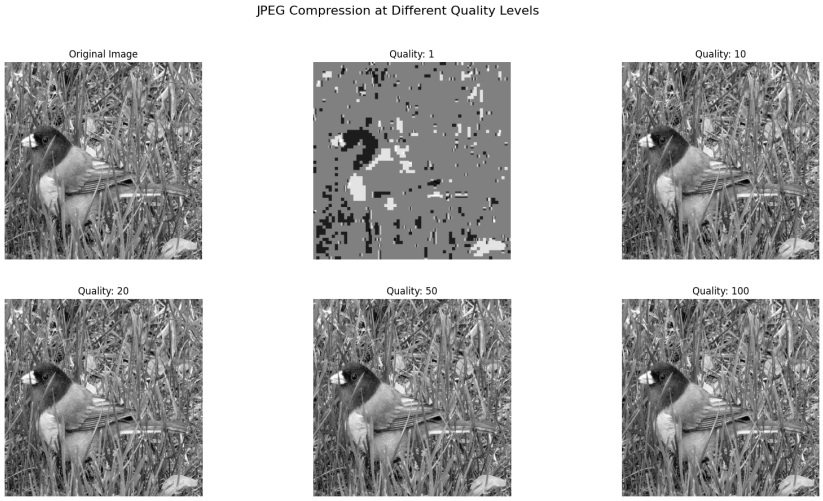


Figure 4.7: Greyscale 7

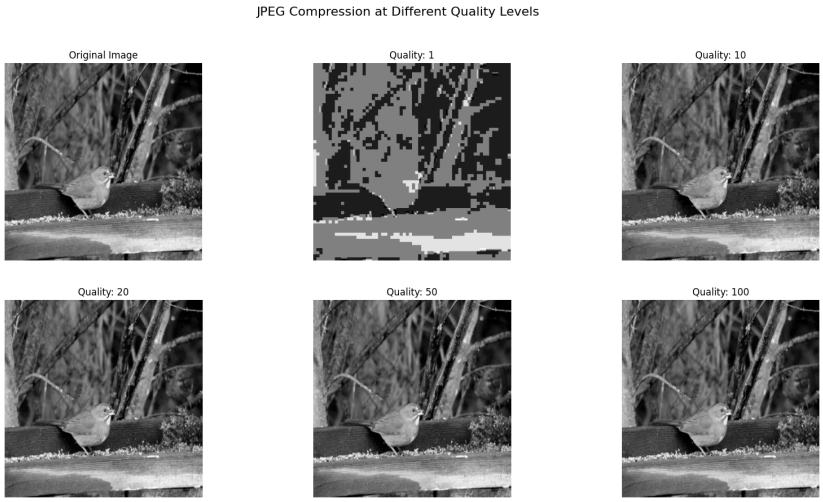


Figure 4.8: Greyscale 8

Figure 4.9: Greyscale 9

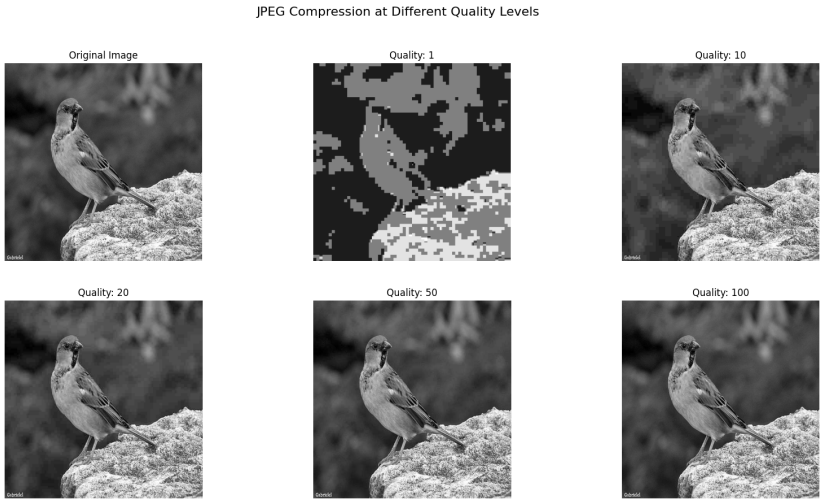


Figure 4.10: Greyscale 10

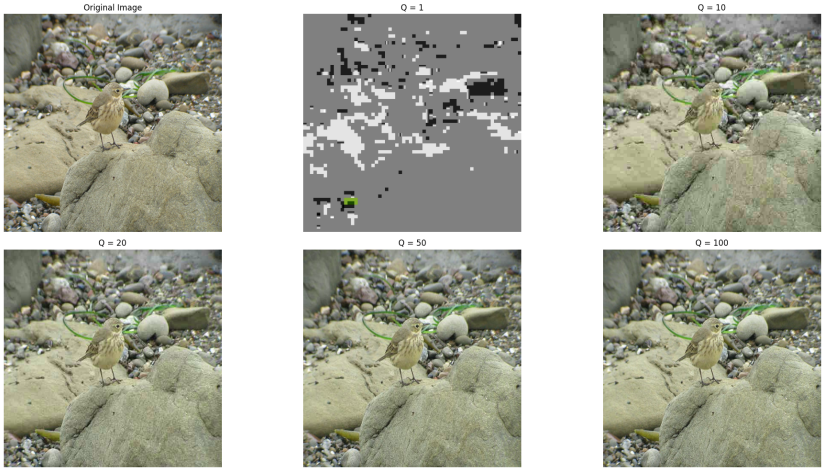


Figure 4.11: Colour 1

Figure 4.12: Colour 2

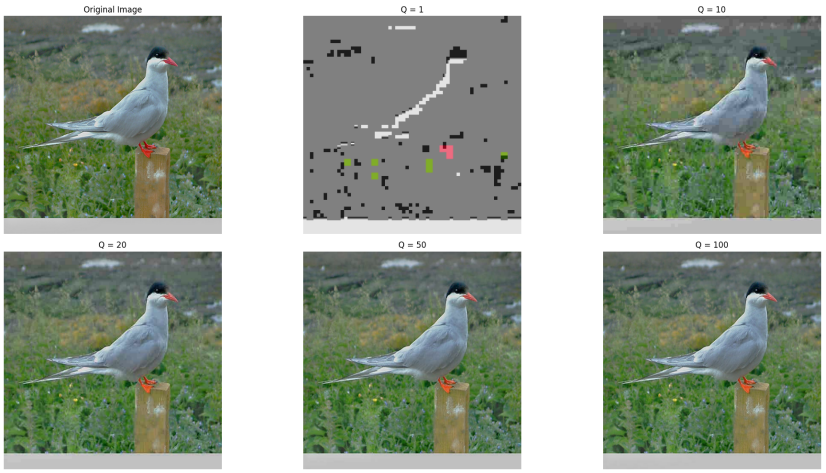


Figure 4.13: Colour 3

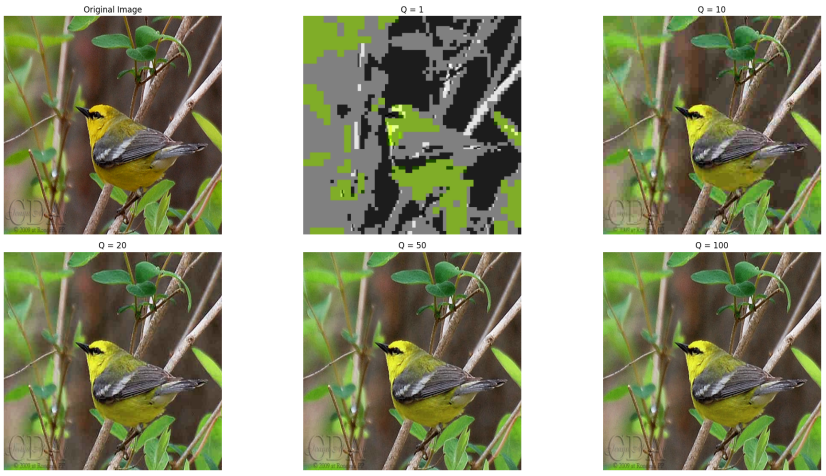


Figure 4.14: Colour 4

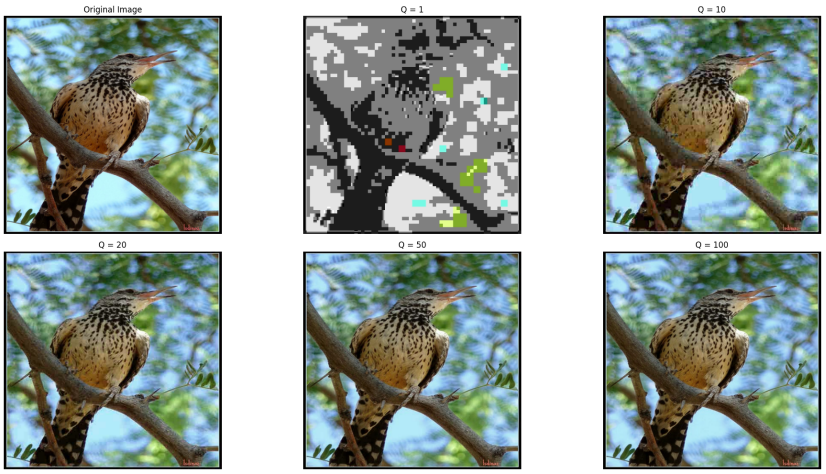


Figure 4.15: Colour 5

Figure 4.16: Colour 6

Figure 4.17: Colour 7

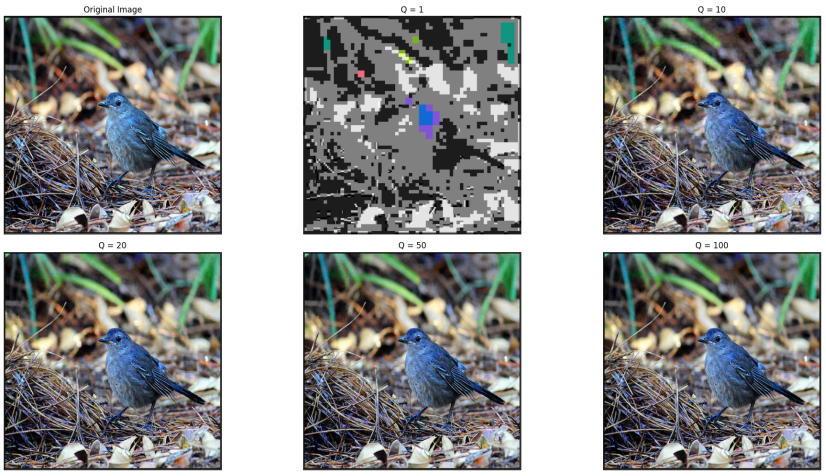


Figure 4.18: Colour 8

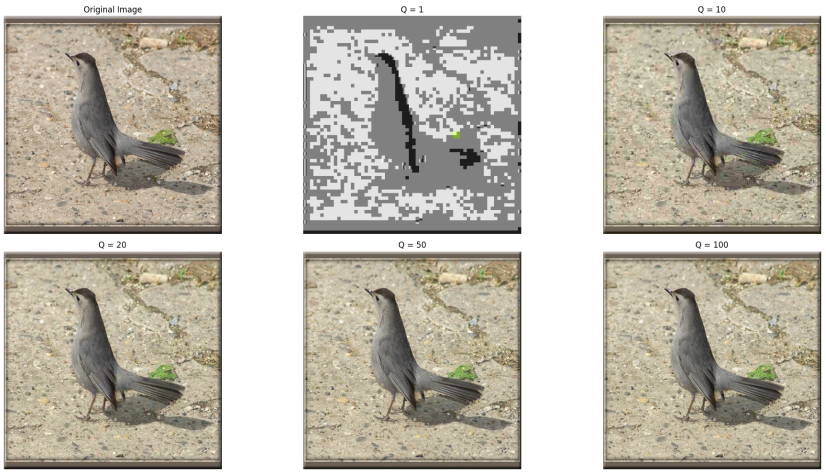


Figure 4.19: Colour 9

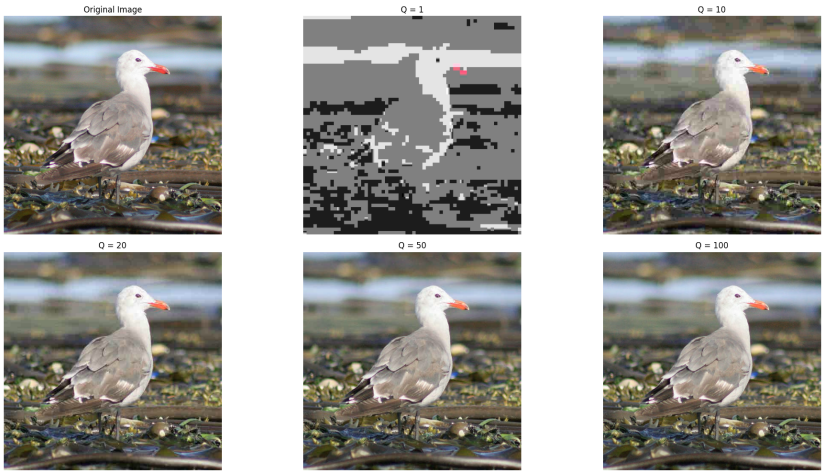


Figure 4.20: Colour 10

4.2 Contributions

- Adit Srivastava: Implemented the autoencoder for image compression. Helped in debugging and evaluating the JPEG algorithm.
- Jay Mehta: Implemented the Huffman encoding and decoding algorithm for the JPEG algorithms. Helped in debugging and evaluating the JPEG algorithm.
- Tanay Bhat: Implemented the DCT and quantization algorithms for the JPEG algorithm. Helped in debugging and evaluating the JPEG algorithm.