Lisandro Nunez
Benjamin Chau

We utilized Professor Daniel's Array2 structure and received some help from the TAs for some clarifying questions about the assignment.

We managed to correctly implement several functions and tasks for compression such as trimming an uneven dimension for an image, changing Rgb values to floating points, floating points to video component color, and packing 2x2 blocks into one. For decompression, we also managed to implement some of the functions successfully such as performing the inverse operations for the discrete cosine transformation, converting component video color to Rgb, and unpacking an array2 of u32s back into their respective decimal numbers.

We unfortunately did not manage to correctly implement some of the specific functionalities when decompressing such as correctly placing the brightness values in a 2x2 block for an expanded 2D matrix.

For the architecture of our program, we primarily constructed three components, the main.rs file, a compression folder, and a decompression folder, which consist of smaller sub-components that function as abstractions to modularize the compression and decompression algorithm. We utilized main.rs file to handle the command-line arguments, read in the input image using csc411_image::Read function, store the image in a new Array2 struct, pass in that Array2 to a compress() function located in the compression folder, and output the compressed image. The same logic applies for decompression but uses decompress functions instead as well as different input that is read from the user.

For compression, we constructed several functions to perform certain operations: compress() which calls smaller functions to compress an image, trim() which removes a column or row from an uneven image, change_to_floating_point which takes in an Array2 of Rgb and converts to floating points. The output Array2 of floats from change_to_floating_point is then passed into a function called rgb_to_component() which changes the rgb float representations into component video color formats of Y, Pb, and Pr and outputs the newly transformed Array2 of component video colors. This Array2 is then passed into another function called pack blocks which compresses the matrix representing the image into a condensed Array2 after doing some bit operations which is then outputted as binary.

For decompression, various functions were also constructed. The decompress() function calls smaller functions to decompress an image. First, it calls unpack_blocks() which takes in an Array2 of u32s to unpack into an Array2 of tuples consisting of a, b, c, d, pb average, and pr average values. That Array2 is then passed into a function called reverse_dct() which performs the inverse operations of the discrete cosine transformation to generate an additional Array2 of tuples consisting of brightnesses. Both Array2s of a, b, c, d, pb average, pr average and brightnesses are passed into a component_video_color() function which generates an expanded image where each pixel is defined as a tuple of y (brightness), pb, and pr. The function returns an Array2 of these tuples and is passed into another function called component_to_rgb() which returns an Array2 of Rgb elements. This Array2 is finally passed into an output function which gives a valid ppm image.

We approximately spent 10 hours analyzing the problems posed in the assignment and approximately 20 hours for solving the problems after our analysis.