The methods used in this project are based on the perceptual hash algorithm. The perceptual hash algorithm describes the images in a hash that can be easily compared against. It takes a little to compute the whole set of images but the simplicity of finding the most similar is as simple as doing a comparison against a subset of images or *is close* comparison. This algorithm doesn't neccesarily have a great record of being efficient or capable of learning what an image actually is of. That can be done with a neural network running images from a training set to have a more decision oriented approach than a brute force checking against all options.

The first step of the process as described in the hackerfactor resource is to shrink the image down to an 8x8 image. For this example 8x8 is more than enough but the higher the resolution the more details are preserved so you get a more exact match rather than a similarity. The second step is to convert all of the color to grayscale, conviently all the images are already grayscale but this implementation will work with colors as well. This condenses the colorspace from 64^3 to a mere 64 colors. This reduces the hashing of the image as well as eliminate the problems with scaling, aspect ratio, or brightness/contrast. It will however not match against rotated images, good thing our training set has the rotated images!

Computing the average of the 64 colors in the image $of the 8x8 grid$ is a simple sum and division by 64. Using this number you set each bit of a 64-bit integer if it is in excess of the mean the bit is set to 1 otherwise it's set to 0. This 64 bit number is now the unique intendifier of this image. Comparing the training set against the single picture is fast.

To actually compare the hashes of each image we use *Hamming Distance* which is just comparing the difference between two strings. Since we can assert that our hash will always be of equal length we can calcualte how different the two numbers are and conclude the *similarity* of the two hashes represents how similar the two images are on a pixel-by-pixel basis.

To speed this up even more it would be possible to have a handpicked subset of images that best simulate each mutation of the group. Running the initial image in question against each set would give a good assumption of which group it *could* be. Further hash comparisons in that group can confirm if there is a better match. If a majority of the second pass would be worse other groups could be tested.

Given an image from each group, this algorithm has 100% accuracy.