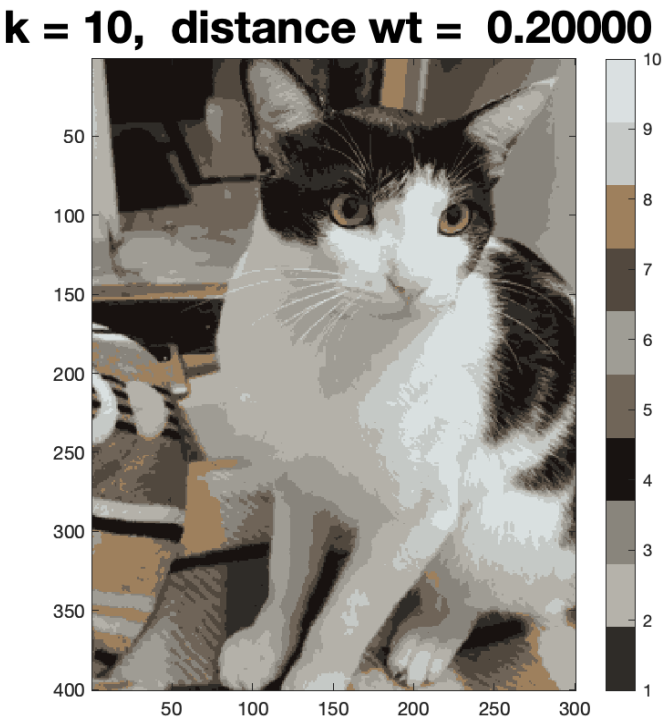
Name: Divyank Kulshrestha

PART A

Code in the file ‘HW_04_Divyank_Kulshrestha.m’ resizes the image to 400x300.

← **Image selected for cartoonization**

PART B



← **Result of K-Means**

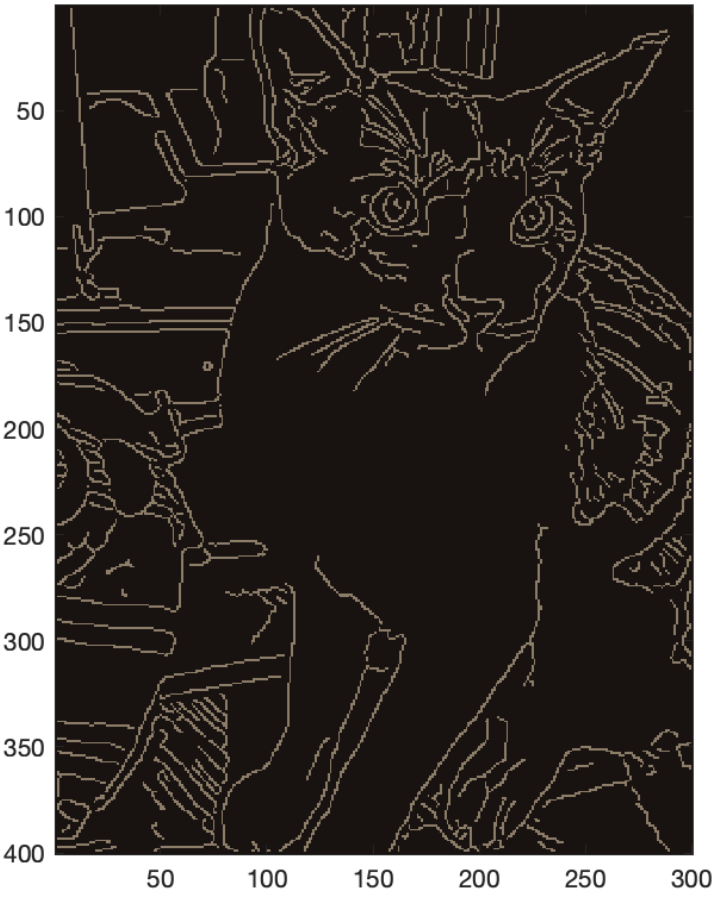
I used 3 attributes for this: Red, Green and Blue channels, all converted to double and having equal weight. Using a filter to ‘clean’ the image resulted in a blurry result, so I decided to not use any sort of filtering in the image and use it as it was.

When trying to use size of image (with weights) as attributes, K-means returned big blobs of color as the result, which was not even close to cartoonization of the image and had lost any image detail.

I tried different values of k, which is the number of clusters, and found out that a high value of k reduces the degree of cartoonization of the image. As we try to partition to a higher number of clusters, each cluster may be assigned its own different color thus pushing the image towards the same pixel colors as in the original image.

Balancing the weight of space and color is important for proper results. Too much space can result in a confusing result while too much color can become distracting. For the colormap, I used the centroid values obtained from the ‘kmeans’ function, replacing the ‘jet’ colorspace, which gave weird colors in the result. Using the centroid values as color space returned the accurate colors for each ‘blob’ in the image.

PART C



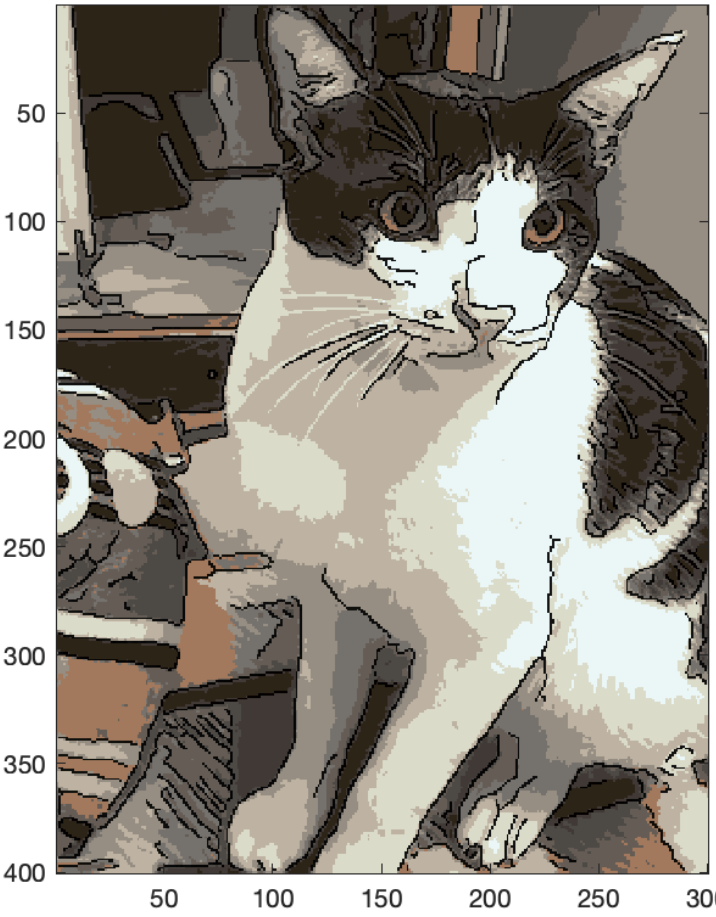
← **Significant edges**

I initially forgot to change the image into grayscale, which gave me an error, but was also easily fixable.

Using the function ‘edge’ gave white edges, which did not look good when fused with the image. To fix that, I inverted the image and obtained black edges.

Using the ‘sobel’ filter initially was not good enough as it failed to find all the significant edges, resulting in loss of edge detail. Although sobel is faster, it is also sensitive to noise and can produce poor edges. Using the ‘canny’ filter, which is more accurate and robust, worked better and retained a lot more structure and detail from the image.

PART D



← **Combined Image from part B and part C**

In order to combine the two images, I used the ‘imfuse’ function and used the ‘blend’ parameter, which uses the weighted average of pixel intensities in the two images.

The result of ‘imfuse’ function had very low contrast so I used ‘histeq’ to improve the contrast of the image.

Edges obtained from the ‘edge’ function were white, so I inverted them to obtain black edges before combining.

PART E

I used the photo of my cat **COCO** who is the most loved member of our family. My family loved this cartoon-ish image. However, being the brat that he is, Coco wasn’t really as interested and just went about his daily business without a care.

PART F (**Conclusion):**

This homework was much more challenging than previous ones. I had to spend a lot of time exploring different ways for simulating k-means in MATLAB. But I really enjoyed working with Coco’s image. Understanding how different attributes affect our k-means result was very intensive and full of hit-and-trials, to get the best results. I also had to find out what ‘replicate’

and ‘maxIter’ attributes mean in order to debug the program. Figuring out how to use centroid values for the colorspace was also a little tricky but required an easy change to fix it.

Working with edges was fun, as I had to try different filters and inversion to find a good balance of detail vs required computation for the edges. Function ‘histeq’ again came to the rescue in order to fix the contrast quickly and easily. Combining the two images and getting the resulting image was very satisfying after spending a lot of time initially.