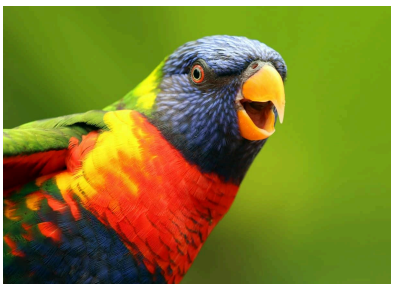
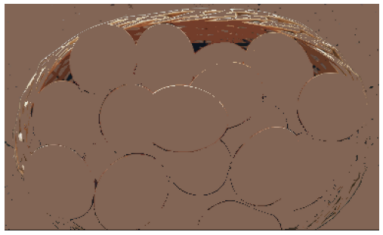
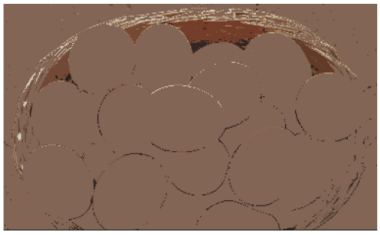
Deniz Soral
Matthaniel Hesbian
Intro ML
18 December 2025
Link to Code
Link to Self Score

For my project, I took images, lowered their resolution by grouping pixels into larger blocks, ran DBscan with 5 features, and ran k-means++ based on mean color, x, and y values of those DBscan clusters.

I chose this structure of ensemble (DBscan → k-means) because I wanted to isolate specific colors first, without creating the 'pinwheel' of death you get with k-means. Furthermore, needing to choose the amount of clusters with an image with tens and tens of thousands of individual points, with tens of thousands of unique colors, would have created poor distinctions and bad groupings. I also chose DBscan for the first step to factor in x and y values when considering pixel groupings; for an example, only DBscan would reasonably be able to distinguish between two similar colors in very very remote areas of the image, whereas K-means would likely create very large groupings and smush distant places together. DBscan would also be able to find longer lines of similar color, whereas k-means is much more likely to separate them. Then, after isolating key colors using DBscan, I used k-means to control the number of key colors I derived, keeping it low by dividing the number of DBscan clusters by 80. While it still factors x/y, I made the k-means model prioritize RGB values and cluster *size* over anything else meaning those last clusters should keep preserving the large distinctions created by DBscan, while grouping together the small clusters (which stops small clusters from being assimilated into the larger blobs). Lastly, choosing k-means first would have just widely group colors together, losing a lot of the detail and noisy colors in the image; to prevent this, I used DBScan with a min_sample_size of 1 to preserve detail without just making things noise, then used k-means to categorize where the noise should fall into.

This model is probably best used in two ways: finding key-colors in an image, or setting up a simple image recognition model/dataset. Because the model is able to slowly narrow down colors and preserve the widely different colors in noise, it's pretty good at defining what the key 40 or so colors are. I could also see this being used in an image recognition model, by clustering and finding patterns which could then be used to train a model like this (based on overall features of clusters and with a decision tree, or maybe using a neural network). I will say though, that with images that don't have much color, or are so so colorful that every block of pixels is very different, the model won't perform well as it won't create distinct clusters (and thus not find patterns) or will create so many clusters with no patterns between them, and essentially make up connections.

Here are some examples of the model (working well and not working well):

 Base Image	 3141 clusters	 39
	 2604	 32