CS 440

PROFESSOR WES COWAN

Colorization

MAY 5, 2021

NABHANYA NEB (NN291)

ARPITA RAY (AR1516)

I. Introduction

Our project demonstrates and explores some basic techniques in supervised learning and computer vision. Based on the information we collect from converting the left half of an image to gray scale, we want to recolor the right half of the image as accurately as possible.

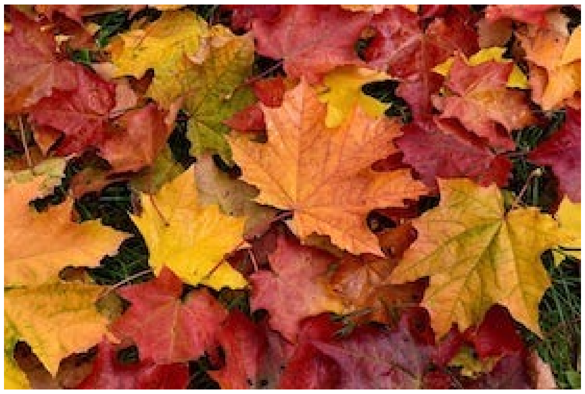

Fig. 1: Original image before any changes

II. The Basic Coloring Agent

The basic coloring agent first converts the image to gray scale by going through every pixel and using the following formula to get the gray value for each pixel:

$$\text{Gray}(r, g, b) = 0.21r + 0.72g + 0.07b.$$

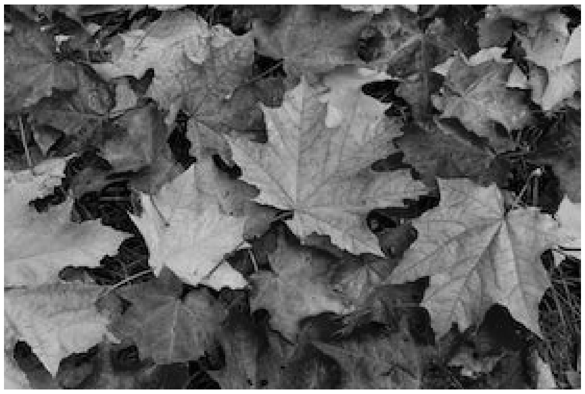

Fig. 2: Image converted to gray scale

Next, we use the k-means clustering algorithm, with k = 5, on the colors present in the training data, which is the left half of the image. We do this to determine the best 5 colors that are representative of the entire image. To implement this algorithm, we first pick 5 random points from the left side of the image and define these are our initial center points for the clusters. We next go through and cluster the remaining the data points according to the center which they are the closest to. We calculate distance by using a formula which takes into account the notion of color difference by weighting the contributions of each color to the total distance. Based on the class notes, we get the final formula to be: dist(color$_1$,color$_2$) =

$$\sqrt{2(r_1 - r_2)^2 + 4(g_1 - g_2)^2 + 3(b_1 - b_2)^2}.$$

Then, we go through each cluster of points and compute the average of the points and define these averages to be our 5 new centers. We now re-cluster the points according to the new centers, and repeat this algorithm t times. In our program, we set t to 100 because we found at this point that the centers were not changing too much and the clusters were staying around the same size. Lastly, we re-colored the left half of our image by setting each pixel in a cluster to the color of the center pixel.

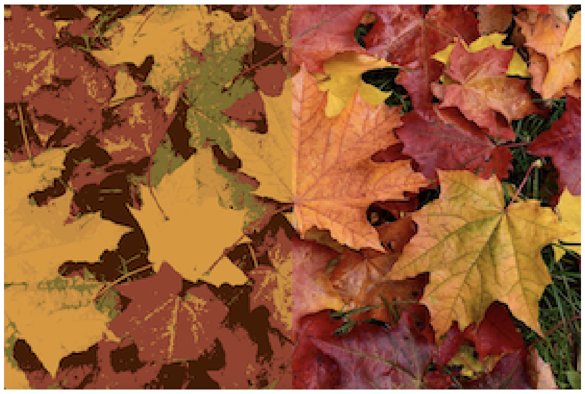

Fig. 3: Left half of image recolored with the 5 colors from the k-means clustering algorithm

Next, we create a dictionary with every (x,y) pixel in the entire image as the key and its corresponding 9 component vector of the surrounding 9 gray values, as the value. We do this so we can easily access these 3x3 patches. Now, we go through each pixel in the right half of the image. For each (x,y) pixel, we get the gray vector that we previously calculated and stored. We now look through the left half of the image to find 6 gray vectors which are the most similar to our current one. The formula we use for this is euclidean distance for a 9 component vector. We go through these 6 vectors and store the color of the middle pixel from the training image, shown above. Next, we pick the the majority color and assign this to our current point. If there is a tie, we break it by picking the most similar vector, which is the one with the smallest euclidean distance from our current vector. We color this point and continue this for every point the right half of our image.

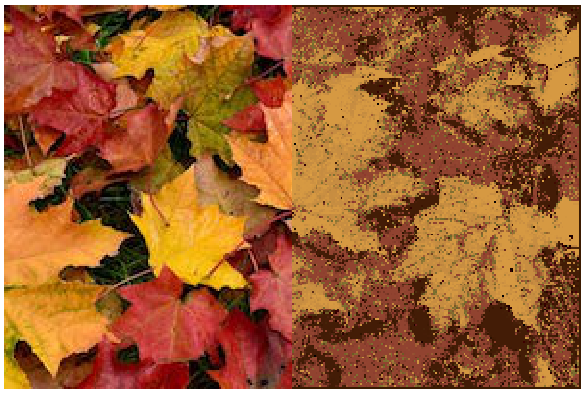
Fig. 4: Right half of image recolored using the basic coloring agent

Visually, this recoloring is decent. There's definitely room for improvement, particularly because there are only five colors, but it's still pretty good considering the limited palette. The coloring captures the basic form of the leaves as well as some shading, but the colors aren't fully accurate.

Numerically, we can measure the quality of the final result by finding the average distance from the color of each point in the recolored image to the corresponding color in the original image. We use the formula from above to find the distance: dist(color$_1$,color$_2$) =

$$\sqrt{2(r_1 - r_2)^2 + 4(g_1 - g_2)^2 + 3(b_1 - b_2)^2}.$$

For our basic agent, we get an average distance value of 96.2.

Bonus: Now, instead of k = 5 for the clustering algorithm, we wanted to find the best number of representative colors to pick for our data. We found this number to be k = 20 for us. The way we found this number was through experimentation. We clustered with various values of k and observed how the image performed. When we increased our k beyond 20, we found that the number of data points in each cluster started getting messy. But, if our k value was too small, the amount of data points in each cluster was too vague and they jumped around a lot. We tested from k = 5 to k = 35, and calculated the numerical average distance value using the formula from above. As we increased our k value to 20, we saw a significant decrease in our number, from 96.2 to 68.5. But, when we continued to increase k more than this, the value didn't decrease as rapidly anymore. So, this meant that k = 20 was probably the elbow for us, which is after the rapid decrease in average distance but before the steady plateau that came after.

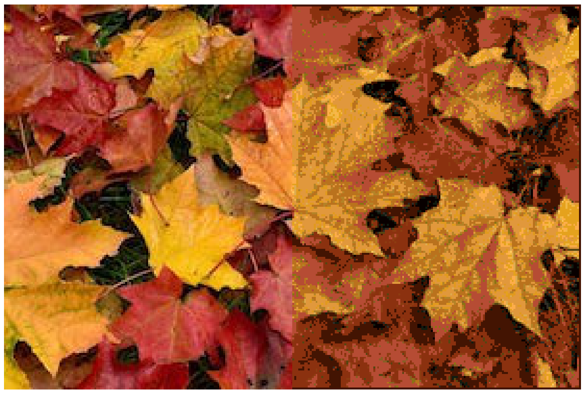
Fig. 5: Right half of image recolored using k = 20

We made the 1 pixel border around the image where we could not calculate 3x3 patches brown instead of black so it was less stark.

III. The Improved Coloring Agent

Next, our acronym for our improved coloring agent is M.Y.T.H.S., which stands for Models You Train Hatch Solutions. We chose this because our improved agent is based heavily based on the training model.

For our input, we take in the 9 component vector for pixels in the gray scale image. We pre-process the data by normalizing the values. This means that we divide all of the values in the vector by 255 so we get values from 0 to 1. Additionally, we add 1 to the beginning of our vector to account for the constant. So, we get $(1, x_1, x_2, ..., x_9)$, with every component being between 0 and 1.

For our output, we have get our r, g, and b values for the (x,y) pixel we are looking at.

For our model, we define it to be $F(x) = \sigma[w{\cdot}x]$. We will be using this model for each r, g, and b. So, we get 3 models: $R(x) = \sigma[wR \cdot x]$, $G(x) = \sigma[wG \cdot x]$, and $B(x) = \sigma[wB \cdot x]$. The wR, wB, and wG variables represent the weights vectors for the r, g, or b models and x represents the input vector described above. We also define our σ as $\sigma(z) = \frac{1}{1+e^{-z}}$.

For our loss function, we define it to be $L(i) = (F(x_i) - y_i)^2$, where i is time passed, $F(x_i)$ is defined above, and y_i is the actual color value of the pixel, which we get from the original image. We will use this in each of the 3 models and use it for the r, g, and b values.

For our learning algorithm, we define it to be $w_{\text{new}} = w_{\text{old}} - \alpha\nabla_{w_{\text{old}}} * L(t)$. We calculate as shown below.

Given: $\sigma(z) = \frac{1}{1+e^{-z}} = \sigma(z) * (1 - \sigma(z))$, $F(x) = 255 * \sigma[w \cdot x]$, and $L(i) = (F(x_i) - y_i)^2$

We calculate:
$\frac{d}{dwR_j}L(i) = \frac{d}{dwR_j}[(F(x_i) - y_i)^2]$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * \frac{d}{dwR_j}[F(x_i) - y_i]$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * \frac{d}{dwR_j}[F(x_i)]$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * \frac{d}{dwR_j}[255 * \sigma(wR \cdot x_i)]$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * 255 * \frac{d}{dwR_j}[\sigma(wR \cdot x_i)]$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * 255 * \frac{d}{dwR_j}[\sigma'(wR \cdot x_i)] * \frac{d}{dwR_j}[wR \cdot x_i]$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * 255 * \frac{d}{dwR_j}[\sigma'(wR \cdot x_i)] * x_{i,j}$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * 255 * \sigma(wR \cdot x_i) * (1 - \sigma(wR \cdot x_i)) * x_{i,j}$
$\frac{d}{dwR_j}L(i) = 2 * (F(x_i) - y_i) * F(x_i) * (1 - \frac{F(x_i)}{255}) * x_{i,j}$

We now have:
$\nabla_{wR}L(i) = 2 * (F(x_i) - y_i) * F(x_i) * (1 - \frac{F(x_i)}{255}) * x_i$
$wR_{\text{new}} = wR_{\text{old}} - \alpha * 2 * (F(x_i) - y_i) * F(x_i) * (1 - \frac{F(x_i)}{255}) * x_i$

We also have:
$wG_{\text{new}} = wG_{\text{old}} - \alpha * 2 * (F(x_i) - y_i) * F(x_i) * (1 - \frac{F(x_i)}{255}) * x_i$
$wB_{\text{new}} = wB_{\text{old}} - \alpha * 2 * (F(x_i) - y_i) * F(x_i) * (1 - \frac{F(x_i)}{255}) * x_i$

These derivations and formulas are from office hours.

The learning algorithm uses this formula to update all three of the weight vectors. We use this equation on 10,000 points, with no repeats, from the left side of the image.

After we finish training, we take the dot product of the final weight vector and the point we are currently on and plug it into $F(x)$ for each r, g, and b model. We multiply these values by 255, and we now finally have our predicted color value.

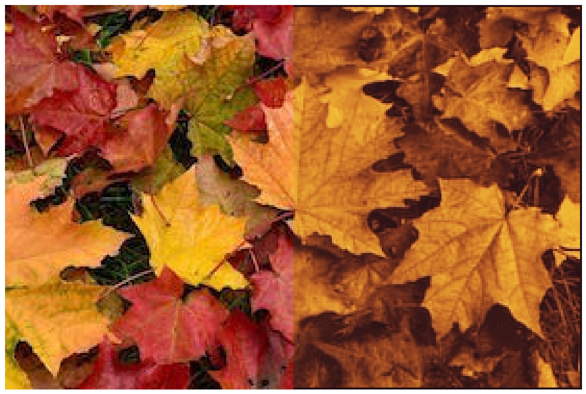
Fig. 6: Right half of image recolored the improved coloring agent

For our parameters, we chose most of them through experimentation. For our alpha, we played around with it and we ended up initializing it to 0.5 and we found this gave us the best result. If alpha was lower the image would lose sharpness. If it was higher the image would be too sharp. For the weight vectors of each model, we initialized them differently for the r, g, and b model. We set our entire initial weight vector to 0.2 for r, 0.4 for g, and 0.3 for b. This is to account for the strength of each color so we could get the best colorization.

When we trained our model, we found that we got the best result from randomly picking some data points to run it on, instead of going through every pixel in the left half. Our image is 300x200 so we randomly picked 10,000 out of the 30,000 pixels in the left half to run it on, without duplicates. This is how we avoided over-fitting. When we trained on all 30,000 points our image didn't look as accurate so this may have been over-fitting. When we ran it randomly on 1/3 of the pixels, we got a better image so this is how we made sure to not over-fit the data.

We used numerical calculations and visual analysis to evaluate the quality and quantity of our model compared to the basic agent. Visually, the improved agent is much more clear than both images from the basic agent. The picture shape is closer to the original, overall. However, the colors are very dulled and are not as accurate as they should be. Our basic agent had similar colors to the improved as well, but the improved does shading and details much better. Numerically, we can measure the quality of the final result by finding the average distance from the color of each point in the recolored image to the corresponding color in the original image. We can use the formula from above to find the distance:
dist(color$_1$,color$_2$)=

$$\sqrt{2(r_1 - r_2)^2 + 4(g_1 - g_2)^2 + 3(b_1 - b_2)^2}.$$

For our basic agent, we get an average distance value of 96.2 for k = 5, and 68.5 for k = 20, as stated above. For our improved agent, we got a value of 55.1. This means that the improved agent actually beat the basic agent numerically, even though it is visually the colors are still a bit off for both agents. The comparison is fair because it just calculates the distance from the pixels in the recolored image to the pixels in the original image and this is a pretty fair description of how well the image did.

With more time, energy, and resources we could improve our improved agent by extending it to use multi-class classification. We could modify our models and loss functions according to this. This would probably produce more accurate colors than our improved agent did. We could use graphical analysis to determine the best number of classes/colors to use as well. This would improve our improved agent to have more color accuracy. Additionally, we would be able to use a larger dimension image. This image would have more pixels, which would help the training and testing data.

Bonus: We based our agent on a parametric model, but a non-linear one because a linear model would not have worked as well. This is because a linear model is unbounded, and we want our values to be bounded from 0 to 255 so it would not make sense to use it. We also don't think the structure of a linear model would accurately predict the values, because the model is too simple for a real colorization.

Acknowledgements

Nabhanya Neb (nn291):
We did both the code and the report together using Zoom. Because of this, we did not have to split anything up and we both contributed equally while working on call. We alternated who was coding/typing to make it fair.

Arpita Ray (ar1516):
The work was evenly divided because we did all of the code on screen-share. We took turns on who would be the person typing each time. For the report, we used Overleaf and did the write up the same way. We worked together on it and alternated who wrote.

Honor Statement:
We have read and abided by the rules of the project, we have not used anyone else's work for the project, and we did all of the work by ourselves.