CMPT 365 - Self Proposed Project Report

Zhong Yu Quan

Abstract

The essence of this project are Python scripts that take an image file (in the sample examples, a movie banner) and convert it into an image using a 128 color palette using KMeans clustering to quantize the colors of the original image into 128 clusters. The Python script then parses a video file (the movie referenced in the banner) by keyframes, downscaling individual images and determining an average RGB value for each keyframe to store as a 16x9 sized “pixel” of the original KMeans color palette. I then use Floyd-Steinberg dithering to generate the original image using the 128 color palette, spreading out the quantization error across adjacent pixels. Finally, I use my folder of 128 downscale keyframes from the image, each representing a color in the 128 color palette, to recreate the original image through the Floyd-Steinberg dithering output.

Introduction

I based my initial image quantization efforts on reducing the color space of the image. I thought of this idea based on the concept of GIFs only saving a 256 color palette as sufficient in information reproduction. I wanted to find a way to reduce the color space of the movie banners into a smaller palette so that I could parse key frames to save as pixel representations. The algorithm I decided on was K-Means clustering as discussed by Tomas Mikolov. K-Means clustering chooses a specified number of random centroids in vector space (in our case RGB values). It then assigns every pixel in a sample to the nearest centroid and recalculates the centroid value based on the average of pixels assigned. This is repeated until the centroids no longer move. I chose to use 128 colors as my image base as examples on the internet showed little perceivable change in image quality between full color space to reduced 128 color space. Very noticeable artifacts began to arise when the color space was reduced further to a palette of size 64.

The next step involved the parsing of key frames to accurately represent each palette color. I decided to downscale key frames by simply averaging pixels over a specified factor range. I chose to preserve the original aspect ratio of the key frame and chose to reduce the size of images by the same ratio height wise and width wise. However, when summing pixel RGBs together, I chose to average the sum of squares of RGB values and then take the square root of this result. This resulted in brighter, better representative downscaled images than simply taking the average of RGBs; I took inspiration for this idea from Manohar Vanga in “Averaging RGB Colors the Right Way”. It was important to note, despite my best efforts, not all palette colors could be represented by key frames present in the movies. I incrementally increased the allowed error ranges and reparsed the movie files over many hours of computer processing time but many cluster values still were not accounted for. Thus, I chose to tint a few key frames in order to best represent my final output image.

Floyd-Steinberg dithering is a technique commonly used by the GIF format when recreating images with a reduced color palette of 256 colors. I highly valued this technique of dithering by diffusing the quantization error onto other subsequent pixels in the image as it reduced many artifacts in my output image that would otherwise be present if the pixels are directly replaced by K-Means predicted pixels. Further, the dithering effect appears to increase the image depth of the recreated image. I implemented this technique in a top to bottom, left to right raster for generating my output image. It is interesting to note that the calculations for this dithering technique are much slower than other dithering techniques as each individual pixel must calculate its closest palette color and then spread the corresponding quantization error to adjacent pixels. This is a process that iterates across the entire image and took a fair bit of time for my script to complete.

Solution

The libraries used in my Python implementation are: numpy, scikit-image, scikit-learn, and pyAV. These can be installed by the following commands:

- pip install numpy
- pip install scikit-image
- pip install scikit-learn
- pip install av

Numpy was used for the handling of large number arrays representing the pixel color channels for each image file. Scikit-image was used to convert jpg images to numpy arrays and to save output numpy arrays back into jpg format. Scikit-learn was used to calculate the KMeans clusters (the 128 color palette) for each sample input image. AV was used to load mp4 video formats into a Python container to parse the individual keyframes in a movie.

The steps for my solution can be broken up by the three scripts I created to implement this project: *parseMovie.py, genDither.py, and buildImage.py*.

parseMovie:

I first passed a jpg image into Python to be processed as a numpy array of pixels. This array was then shuffled and randomly sampled for 5000 pixels. This sampling step is incredibly important in increasing the efficiency of the KMeans Clustering algorithm as vector quantization for the entire pixel array would be extremely difficult. The sample is still sufficient to generate a suitable image and further errors will be reduced by Floyd-Steinberg dithering. The sample of the original pixel array was used to run KMeans Clustering to generate a palette of 128 cluster colors that could be used to represent and recreate the original image. I have saved an example of what an image recreation with 128 colors would look like in spiderReduced.jpg and strangeReduced.jpg. I then created a function to downscale a picture by a factor by averaging ranges of pixels into individual pixels. Since I was working with 720p video keyframes, I chose a factor of 45 downscaling in order to get output images of size 16x9. I also created two helper functions that averaged the color of the downscale image and then compared this average color with the 128 saved palette colors against an error threshold. If the downscaled image was sufficiently close to a color, it would be saved in a directory, essentially building a table of downscaled keyframes to represent the 128 colors in the color palette obtained from KMeans clustering. Finally, I wrote a script to parse through a movie file by keyframes and save the results of downscaled keyframes that matched a palette color. This turned out to be an incredibly arduous and time consuming process that took multiple iterations through the movie as many colors in the original palette from KMeans were very similar and would take precedence in being saved as a keyframe.

genDither:
In this step I coded an implementation of Floyd-Steinberg dithering to generate an array of numbers representing which palette color to use for each pixel in an original image. I loaded in the original image in a numpy array and created a function to compare a pixel with the color palette from KMeans to return the closest color and associated error. I then iterated through the original image's pixel array, running the Floyd-Steinberg algorithm to pass this error to subsequent adjacent pixels. The final output array of palette colors matched to pixel locations in an image was saved in the local folder.

buildImage:
In this step, I took in the array generated by genDither and the image palette files produced by parseMovie. I created a big array to represent the size of the original image multiplied by the size of the jpg pixels (16x9) that would be substituted in as pixel replacements. I then iterated through this large array and filled in pixel information for each palette color jpg called by the dither array. This big array was saved to create a final output image.

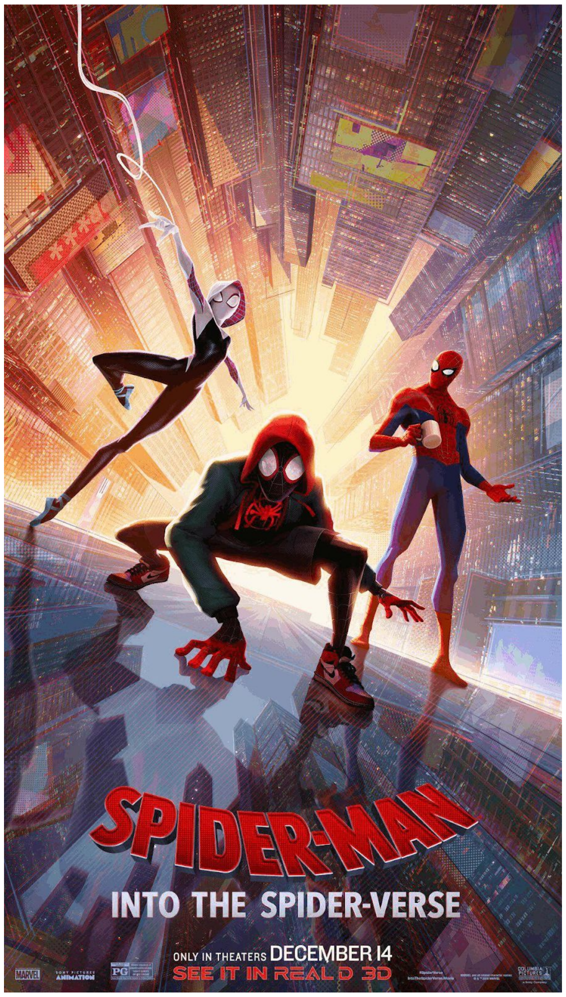
Sample Results
A backup repository of my project directory can be found on:
https://drive.google.com/drive/folders/14juS0ZOi8s85wquRcgver8GQnR2qMzUS?usp=sharing
This includes my sample output pictures, strangeFinal.jpg and spiderFinal.jpg. The two output images from the project are very large and cannot be compressed into the 30mb submission limit on Coursys.

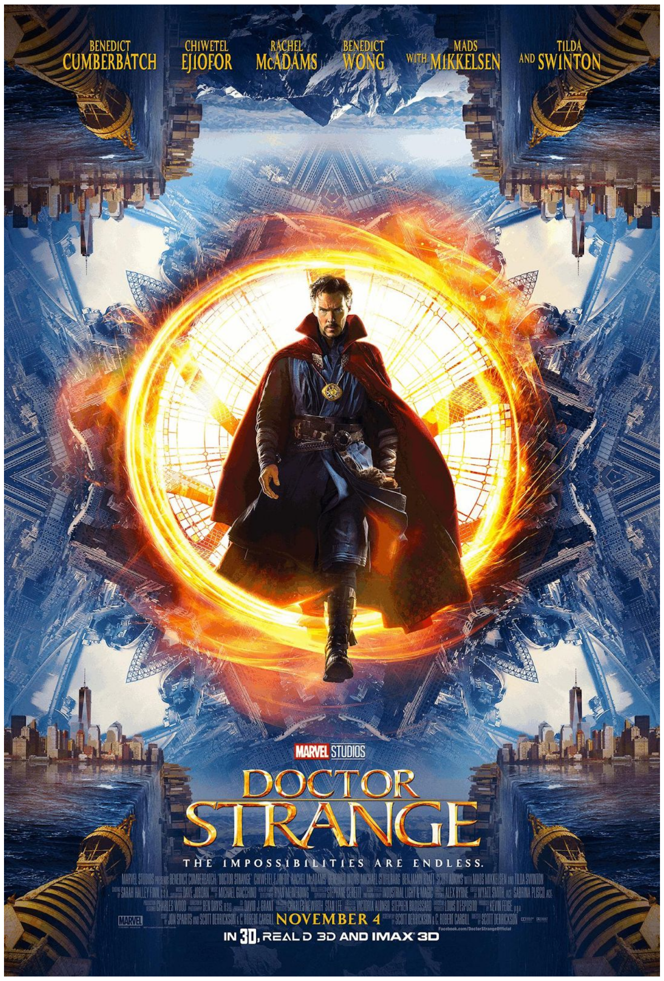
The original images I used are: spider.jpg and strange.jpg

SPIDER-MAN
INTO THE SPIDER-VERSE
ONLY IN THEATERS DECEMBER 14
SEE IT IN REAL D 3D

BENEDICT CUMBERBATCH
CHIWETEL EJIOFOR
RACHEL McADAMS
BENEDICT WONG
WITH MADS MIKKELSEN
AND TILDA SWINTON
MARVEL STUDIOS
DOCTOR STRANGE
THE IMPOSSIBILITIES ARE ENDLESS.
NOVEMBER 4
IN 3D, REAL D 3D AND IMAX 3D

When they are quantized into a range of 128 colors by KMeans Clustering, the output references images are: spiderReduced.jpg and strangeReduced.jpg





The final output images generated from movie keyframes and Floyd-Steinberg dithering are spiderFinal.jpg and strangeFinal.jpg

spiderFinal.jpg:
https://drive.google.com/file/d/11IYxMYH8dgP_SVQmAM34Fgx-hPRGfeiB/view?usp=sharing
strangeFinal.jpg:
https://drive.google.com/file/d/11MIMk8iyppSonEblh3s7PNjH4cyuftz_/view?usp=sharing

These final output images look flattened or squished relative to the original movie banners as each individual pixel is replaced by a 16x9 sized keyframe from the 720p movie. This means that the width of the original image is stretched out by a ratio of 16 whereas the height only increases by a ratio of 9. I wanted to maintain the ratio of the original keyframes to preserve some visual information inside the palette pixel, and thus chose not to reduce my downscaled images into a square pixel shape. However, I believe it is clear that my output image is very similar to the original image movie banner outside of its dimensions. The major differences arise from difficulty in finding an adequate movie keyframe to represent a palette color, leading to an increase in the allowable error threshold. This means that some palette colors may be represented by a keyframe that is quite different from the intended average color.

References

Floyd-Steinberg Dithering. (2000). Retrieved from https://www.visgraf.impa.br/Courses/ip00/proj/Dithering1/floyd_steinberg_dithering.html

Mikolov, Tomas. (2012). Color Reductions Using K-Means Clustering. Brno University of Technology.

Vanga, Manohar. (2017). Averaging RGB Colors the Right Way. Retrieved from https://sighack.com/post/averaging-rgb-colors-the-right-way