

## ***Final Project Summary***

---

*Michael Hodgetts - 10258130*

*Shehpar Sohail - 10460622*

### **Presentation Title: Image Conversion**

#### **Summary:**

Our final project focuses on utilizing Python code and related packages/modules that will help the user manipulate images into various forms. At the start of the project, we were not familiar with the algorithms to implement that would allow such image conversions. We became eager to learn how to use some of the basic tools/concepts to take an image as input and modify it into several styles.

The research for our project focuses on three areas:

1. Converting a jpeg into a Pencil/Sketch image
2. Converting a jpeg into a Cartoon image
3. Converting a gif into a Mosaic image

To summarize, we quickly learned that the implementation of taking an image and creating the effect of a pencil or cartoon was generally straightforward. However, mosaic imagery was a much bigger lift.

For Pencil and Cartoon images we used the OpenCV package to generate the edges, blurriness, and inversion of the image among other commands to achieve best results. We defined functions for both styles and added line size and blurriness parameters for the user to choose. These functions take the original photo as input and perform the various steps to transform the image.


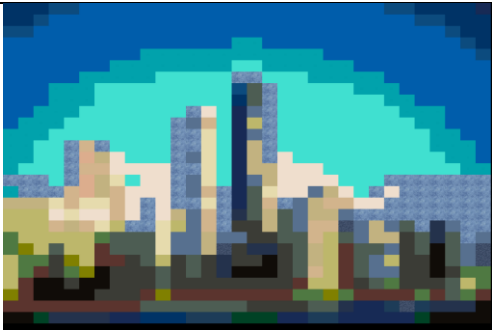


We examined four photos of various styles, i.e., landscape, portrait, urban, and close-up landscape. Once we implemented the algorithms to generate the pencil or cartoon conversions, we created a for loop to examine each photo one at a time. We first displayed the original image for five seconds and then the transformed image for another five seconds. This is a nice feature utilizing the `cv2.waitKey()` function. Since we set up a sequence to display images in a five second duration sequence, the video shows a total of eight images in sequence, all for five seconds each, for each of the two styles of image conversions. In general, we were pleased with the results. The line size parameter is a key factor in bringing out smaller edge features.

The implementation of generating photo mosaics was more involved as it required more steps or functions and coordination. The basic premise to solve is to take a master image and subdivide it into a grid of vertical and horizontal lines. The total number of regions the master image is divided into is dependent on the width and height of the individual tile images. For simplicity, our project keeps the same width and height for all the tile images. The average color of each of the individual tile images is computed and stored.

Then for each tile-sized region of the master image, the average color is compared with the stored average color of the individual tile sized regions to find the best match. Once the closest tile image is selected, it is inserted over the respective grid position of the master image. A key factor is size of the tiles being used to create the photo mosaic. The smaller the dimensions of tiles, the better resolution. Also, the number of tiles to choose from with various color options is a big factor as well. The sequence of the steps utilized were as follows:

1. Load in and store the size of the master image.
2. Load in and store the size of the individual tile images. The `resize_images(tile_location)` function resizes all the tile images to be of the same width and height.
3. Compute the average color of the individual tile images and store them.
4. Split the master image into tile-sized regions depending on the dimensions stored in step (2).
5. Iterate through the tile-sized regions of the master image by calculating and storing the average color, finding the closest match tile image, and pasting the best fit tile in the respective grid position.
6. Save and show the photo mosaic.

Table 1: Example Photo Mosaics Generated Using the Designed Application

Master Image		
		
Mapping of 50 tiles of width 28 pixels and height 21 pixels to the master image		
		
Mapping of 300 tiles of width 28 pixels and height 21 pixels to the master image		
		
Mapping of 300 tiles of width 5 pixels and height 5 pixels to the master image		
		

Clearly, the mapping of 300 tiles of width 5 pixels and height 5 pixels to the master image resulted in the best resolution. Thus, the photo mosaic generated by our application proves our conclusion that size of the tiles being used and the number of tiles to choose from are the main key factors to consider.

Now that we have learned how to process and transform common photos in this project, future research could examine generating mosaics from individual tile images that vary in height and width. In addition, we can examine performing these various conversions on other forms of media i.e. videos, boomerangs, and panoramas.