Week 1 Computer Vision Notes

Maxwell Li

3-25-2020

Contents

Section 1: Getting Started with Images

Introduction

Some things we will learn
- What is color?
- How does a camera see an image?
- How to read/write images?
- How to manipulate pixels?
- Alot more!

How is An Image Formed

Throughout history the phenomena of imaging has been explored by many ancient civilizations. The most used primitive form of a camera that has been used was the pinhole camera. The pinhole camera works by taking the light waves that bounce off of an object and projecting them through a small hole.

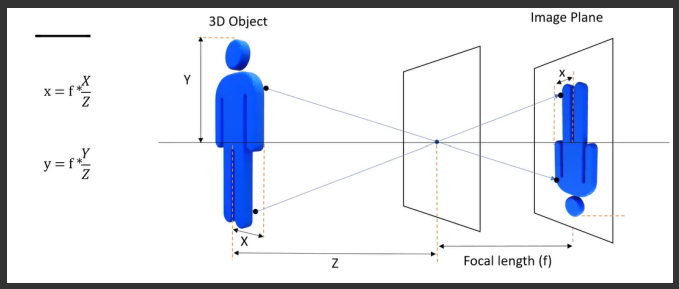

Fig. 1: PinHole Camera Example

Image Formation

Inside of a camera there is a sensor that is a large grid of nodes. These nodes are what eventually become "pixels"'. Each node on the sensor gets a value between 0 and 255 which represents the digital grayscale image. The value for each node is determined by the strength of the light that hits it. The stronger the light the closer the value is to 255 as 255 is white and 0 is black. Each pixel is essentially an 8 bit representation of the intensity of the light at that position on the sensor.

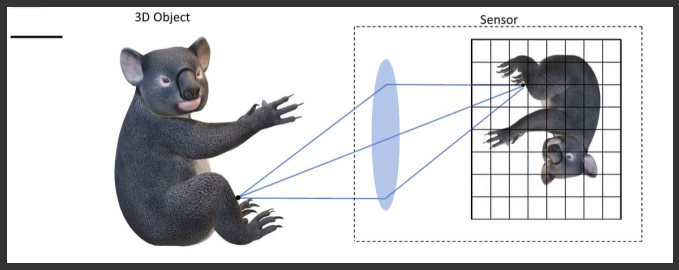

Fig. 2: Greyscale Digitalization

Bayer Pattern

Every pixed only records one color: red, green, or blue. The human eye is much more sensitive to green light than blue or red. The full RGB Image is then formed through a process called demosaicing where 2 two missing values from each pixel are calculated from the neighbouring pixels
For example: If a pixel records the red value, based on the pixels around it the pixel will interpolate what the correct hue for the 2 missing colors.

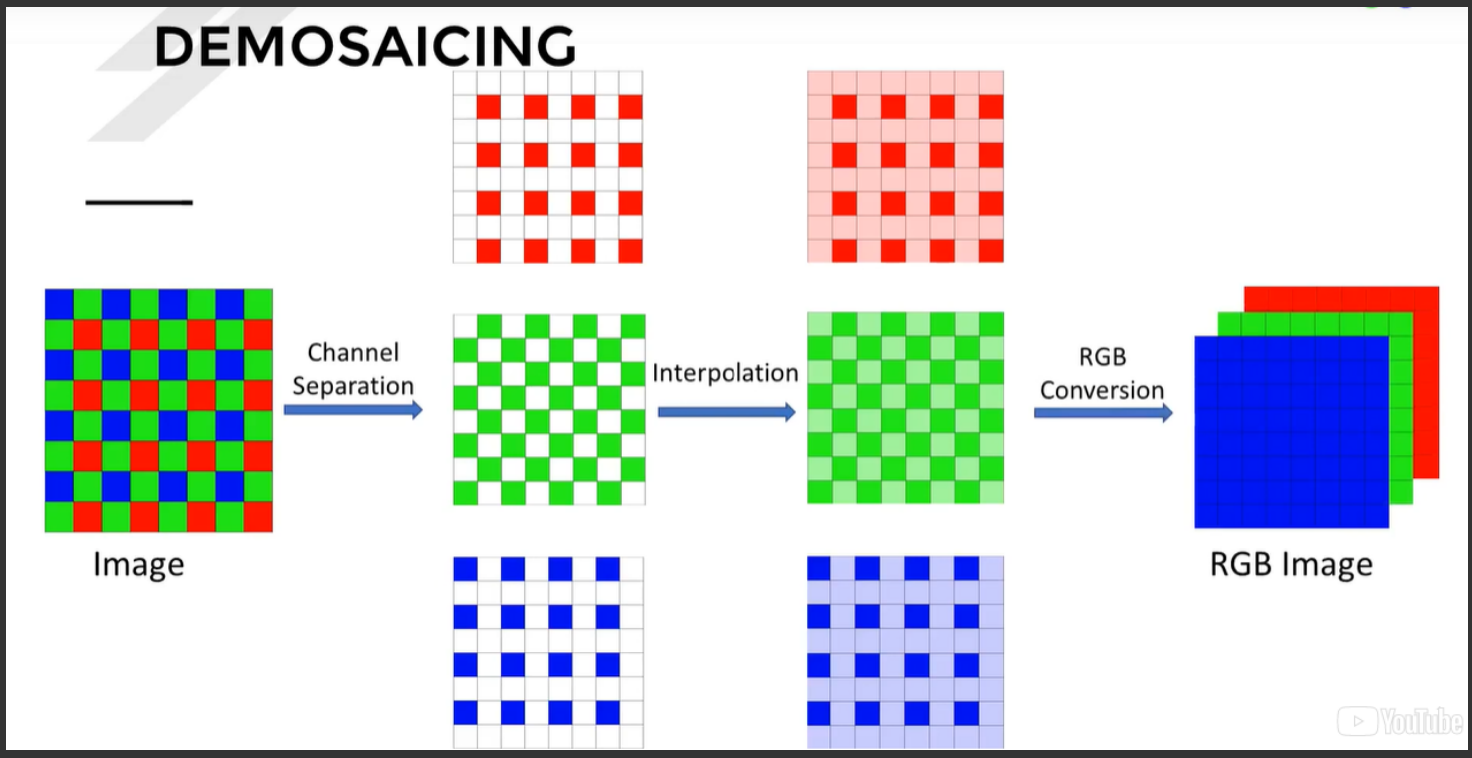

Fig. 3: Process of Demosaicing

Image Format

Once a RGB image is created, they are typically stored and compressed as either Joint Photographic Expert Group (JEPGs) or Portable Network Graphics (PNGs).

Image Storage

Within a JPEG file the image is broken up into two parts. There is the Image Header and the Data.
- Image Header
  - Width
  - Height
  - No. of channel
  - Color Profile
  - No. of bits per pixel

The second part is the data which actually contains the RGB values

Image in OpenCV

When an image is first read in openCV the image is decompressed and stored in a standardized format. All images are stored into the **Mat Class** if the C++ version is being used, or a **Numpy array** if Python is being used.

Mat Class

The Mat class is similar to a JPEG in structure but the difference is that instead of using RGB values it used BGR. It uses bgr because of historic reasons, aka that is how they initially did it and it makes no sense to back through and change all of the backend code now.

Reading Images in OpenCV

Image Formats

OpenCV can use two different formats, JPEG and PNG

JPEG

1) Most Popular
2) Loss in image format
3) Small Filesize

PNG

1) Very Popular
2) Lossless and lossy options
3) Supports transparency

Fig. 4: JPEG vs PNG.

Transparent Images

Contains all three regular channels (Red, Green, Blue) and also contains an 'Alpha' channel that acts as a mask (255 or 0). The value that is put into the alpha channel is determined by the Opacity of the pixel. If the pixel is "Transparent", then there will be a 0 for that pixel in the Alpha channel. If the pixel is not "Transparent", then it will recieve a value of 255.
***Note: Alpha channels do not neccesarily have to be binary. They can be tertiary or higher if for some reason that kind of masking is wanted/needed. Aka, there can be partial transparencies

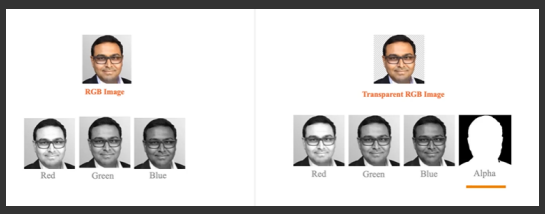

Fig. 5: RBG vs Transparent RGB.

Reading The Image (Week One: Getting Started With Images)

To load an image through OpenCV use the **imRread** function. This function is able to handle most, if not all of the image types that you may want to upload (these include JPEG, PNG, etc)

Function Syntax

$loadedimage = cv2.imread(filename, flags)$
this function (imread) will return the image as a matrix or None if something happened that it was not able to resolve.
This function has **two arguments**:

1) The **path** of the image file: this can be absolute or relative and is a mandatory argument
2) **Flags:** -1, 0, or 1. ***DEFAULT IS ONE***
   $cv2.IMREAD\_UNCHANGED$ or -1: Loads an image including the alpha channel (unchanged)
   $cv2.IMREAD\_GRAYSCALE$ or 0: Loads the image in grayscale
   $cv2.IMREAD\_COLOR$ or 1: Loads a color image and throws out transparency

Manipulating Pixels (Week One: Getting Started With Images)

Accessing Values

Since the openCV returns things as a matrix of values, you can just access them using the subscript notation of matricies. IMPORTANT: Numpy saves the matrix in row column format which is the opposite of x,y format. For example if you want to access the pixel to the right of the top left instead of (1,0), which is the cartesian representation of it, it would be [0,1]. This is an important distinction as it will allow you to actually access the correct variables.

$$print(testImage[0, 0])$$

Modifying Values

Just use a normal python = operation for this. You can either modify a single value or a whole region...that is next
There are a few reasons why people would want to change the values of the pixels. For one, if there is a known issue with the image or a known change that the user might want to change, they need to be able to access and modify the values. For example maybe the edges of the image are blurry so you just set them all to 0 so that there are no false positives.

$$testImage[0, 0] = 200$$
$$print(testImage)$$

Manipulating Groups of Pixels (Week One: Getting Started With Images)

How To Access A Region

ROI - Region of Interest
$$test\_roi = testImage[0 : 2, 0 : 4]$$

This code will return a resulting matrix the is 2 rows tall, and 4 columns long. This is a bit counterintuitive because if you count up from 0 (where subscript position starts), and go to the last number then both dimensions are missing 1 element. This is because python is inclusive of the lower bound but non inclusive of the upper bound.
0:2 tells python, hey! Look in this matrix and take out rows 0 and 1, stop before two, and also take columsn 0, 1, 2, 3 and stop before four. Now stich this all together into a matrix and return it! This way the programmer can select a specific part of the iamge that they are interested in using aka, Region of Interest.
To set a region of interest to a specific value all you have to do is take the region, and use the equals operator.

$$testImage[0 : 2, 0 : 4] = 111$$