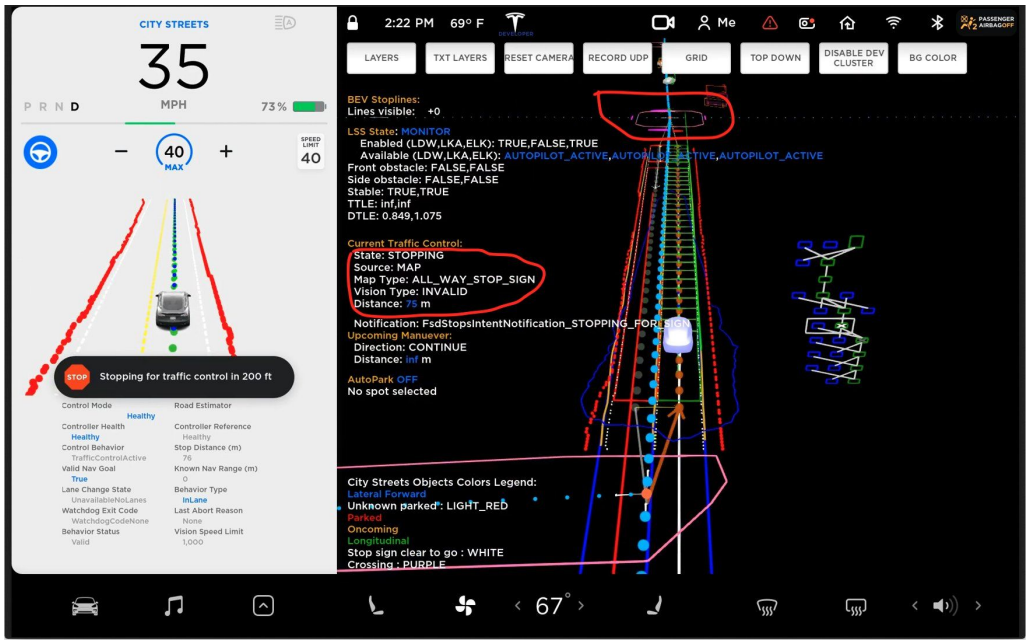
Jaden Booher
February 7, 2023

Wiki Article: Image Processing Tutorial

Introduction

Machine/deep learning and artificial have had a massive impact on the development of technology over the last decade. However, 2023 stands to be the most impactful year yet. With the consistent development of computer vision, the capability for computers to identify and process an image, there is an expansive amount of use cases. From IoT to self-driving, to facial recognition and biometrics, machine learning is undoubtedly going to have a prominent spot in the future of technology. And, at the core of this type of learning is image processing.



Baldwin, Roberto. "Tesla Full Self-Driving System's Beta Developer Settings Leaked."

What is image processing?

Image processing is defined to be "the process of transforming an image into a digital form and performing certain operations to get some useful information from it" (Simplilearn). The purpose of an image undergoing this process is to transform the image from its original form into a new form that can be digested and used by a software. Some of the ways that an image can be processed include, but are not limited to, visualization (finding an object not visible in the image), sharpening and restoration (enhancing the image), recognition, (detecting object in an image), and retrieval (reverse search a database based on the image). In this tutorial, we will

explain in detail some of the different basic image processing that can be done using Pythion's OpenCV library.

Tutorial 1: Changing Color Spaces

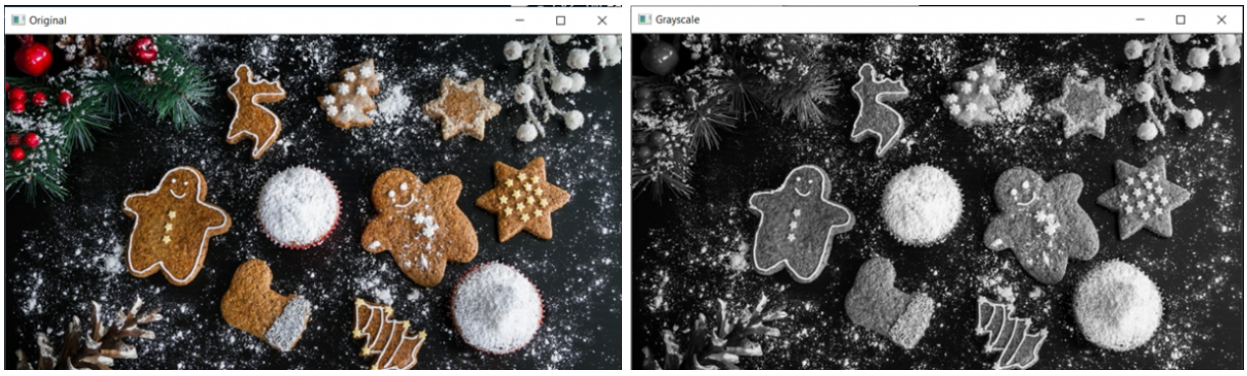
One of the most basic forms of image processing is changing color spaces. There are approximately 150 different color spaces included in the OpenCV library, but this tutorial will focus on the conversion from BGR, which is like RGB but with an inverse subpixel arrangement, to grayscale, which is essentially black and white. Lots of image processing software and algorithms require greyscale, as this removes some unnecessary complexities for the software.

Let's try the following code:

```
import cv2

image = cv2.imread('cookies.jpg') # import image
cv2.imshow('Original',image) # display original image
grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # swap colorspace
cv2.imshow('Grayscale', grayscale) # display grayscale image
```

The resulting images should look like this:



Tutorial 2: Histogram of Color Channels

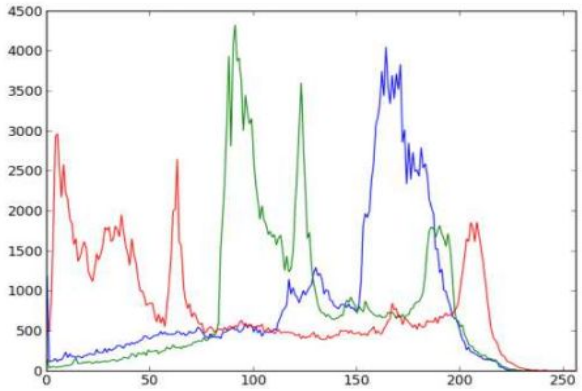
Another useful feature of OpenCV is the color channel histogram function. Color histograms represent the distribution of colors, such as BGR (blue, red, and green), that are present in an image. This works by plotting the pixel values of the entire image, ranging from 0 to 255, on the X-axis while plotting the number of pixels with that value on the Y-axis. By plotting the BGR values, this function allows people to search databases for an image based on their color profiles rather than an image's metadata. This tutorial will show how to create a color histogram using OpenCV the *matplotlib* library.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('house.jpg') # import image
color = ('b','g','r') # define the 3 plots for blue, green, and red

# code for plotting each pixel
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show() # display hisogram
cv2.imshow('Original',image) # display original image
```

The results should look like this:



Tutorial 3: Image Smoothing

A third way to use OpenCV for image processing is image smoothing. Many images, such as the picture of the ducklings below, often contain random noise. This noise provides no valuable information and can sometimes lead to image processing and storage issues. The picture of the ducklings has an excessive amount of noise due to the complexities of their feathers sticking out. This may cause our machine learning models to deviate their attention, so we must use smoothing to reduce the contrast and blur the edges, reducing the noise. The following example will apply a 5x5 averaging filter using OpenCV and *matplotlib*. This works by averaging all the pixels in an image with the kernel, replacing the central pixel with its average. Different-sized kernels will result in different extents of blurring. Here is a visual representation of the kernel we will be using:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('ducklings.png') # import image

kernel = np.ones((5,5),np.float32)/25 # create a kernel to filter with
dst = cv2.filter2D(img,-1,kernel) # convolve the kernel with the image

# display the images
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('An image smoothed with gaussian smoothing')
plt.xticks([]), plt.yticks([])
plt.show()
```

This is what the resulting images should look like:

Tutorial 4: Interactive Foreground Extraction

Have you ever wanted to remove the background of the image but don't know how to use photoshop? Well, using interactive foreground extraction will allow a person to remove the focus of an image using the GrabCut algorithm. This algorithm works by drawing a rectangle around the foreground region, then iterating and segmenting the image until the final, focused image is presented. One of the fascinating aspects of this algorithm, however, is the fact that it can be fine tuned using a "mask," which allows the user to more directly specify the foreground and background. Additionally, this algorithm also leverages "strokes," which allows the user to tell the algorithm if it correctly identified the foreground and background after each iteration, to further finetune the results.

```
import numpy as np
import cv2

img = cv2.imread('person.jpg') # import image

# define mask; this can be further tuned
mask = np.zeros(img.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)

rect = (150,50,500,470) # define rectangle based on pixel locations

# apply grabCut method to extract the foreground
cv2.grabCut(img,mask,rect,bgdModel,fgdModel,20,cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask2[:,:,np.newaxis]

cv2.imshow('Foreground Image',img) # display the resulting image
```

This should be the result: