Traffic_Light_Classifier-zh

November 26, 2018

0.1

notebook

1.
2.
3.
4.
5. > 90100

0.1.1

'(IMPLEMENTATION)''(QUESTION)'
1. **90** 2.

1 1.

31484 * 904 * 536 * 44
- 4.0

1.0.1

```
In [2]: import cv2 # computer vision library
        import helpers # helper functions

        import random
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg # for loading in images

        %matplotlib inline
```

1.1

1484

- 80
- 20
- 3

1.2

In [4]:
```
IMAGE_DIR_TRAINING: the directory where our training image data is stored
IMAGE_DIR_TEST: the directory where our test image data is stored
```

```
  File "<ipython-input-4-e292d0ecadef>", line 1
IMAGE_DIR_TRAINING: the directory where our training image data is stored
                                                 ^

SyntaxError: invalid syntax
```

In [3]:
```
# Image data directories
IMAGE_DIR_TRAINING = "traffic_light_images/training/"
IMAGE_DIR_TEST = "traffic_light_images/test/"
```

1.3

IMAGE_LIST“”“”“”

helpers.pyload_datasetglob load_dataset

IMAGE_LIST

In [4]:
```
#```python
# Using the load_dataset function in helpers.py
# Load training data
IMAGE_LIST = helpers.load_dataset(IMAGE_DIR_TRAINING)
```

1.4

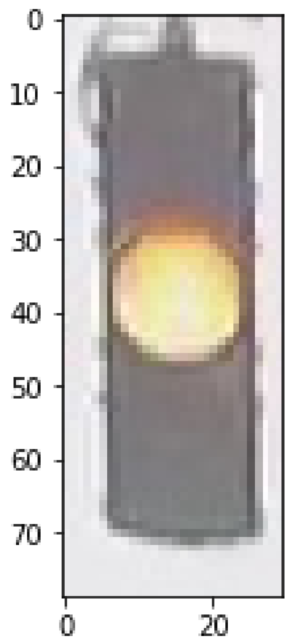
1.2.

1.4.1

IMAGE_LIST * * *

In [7]:
```
## TODO: Write code to display an image in IMAGE_LIST (try finding a yellow traffic ligh
## TODO: Print out 1. The shape of the image and 2. The image's label

# The first image in IMAGE_LIST is displayed below (without information about shape or l
index =390
selected_image = IMAGE_LIST[index][0]
label = IMAGE_LIST[index][1]
plt.imshow(selected_image)
print(selected_image.shape)
print(label)

for img in IMAGE_LIST:
    if img[1] == 'yellow':
        img_yellow = img
        break
plt.imshow(img_yellow[0])
print(img_yellow[0].shape)
print(img_yellow[1])
```

```
(46, 17, 3)
red
(79, 30, 3)
yellow
```



2 2.

2.0.1

2.0.2

""""

3 01[][]
[1,0,0][010][001]

###

- 32x32px
-

```
In [5]: # This function should take in an RGB image and return a new, standardized version
        def standardize_input(image):

            ## TODO: Resize image and pre-process so that all "standard" images are the same siz
            standard_im = np.copy(image)
            standard_im = cv2.resize(standard_im,(32,32))
            return standard_im
```

2.1

- 1

[][0,0,0]1[0,1,0]

###

```
In [6]: ## TODO: One hot encode an image label
        ## Given a label - "red", "green", or "yellow" - return a one-hot encoded label

        # Examples:
        # one_hot_encode("red") should return: [1, 0, 0]
```

```
# one_hot_encode("yellow") should return: [0, 1, 0]
# one_hot_encode("green") should return: [0, 0, 1]

def one_hot_encode(label):

    ## TODO: Create a one-hot encoded label that works for all classes of traffic lights
    one_hot_encoded = []
    if label == 'red':
        one_hot_encoded = [1,0,0]
    elif label == 'yellow':
        one_hot_encoded = [0,1,0]
    elif label == 'green':
        one_hot_encoded = [0,0,1]
    return one_hot_encoded
```

2.1.1

`test_functions.py`
`test_one_hot(self, one_hot_function)`one_hot_encodeone_hot_labelTEST PASSED

```
In [6]: # Importing the tests
        import test_functions
        tests = test_functions.Tests()

        # Test for one_hot_encode function
        tests.test_one_hot(one_hot_encode)
```

TEST PASSED

2.2 STANDARDIZED_LIST

```
In [7]: def standardize(image_list):

            # Empty image data array
            standard_list = []

            # Iterate through all the image-label pairs
            for item in image_list:
                image = item[0]
                label = item[1]

                # Standardize the image
                standardized_im = standardize_input(image)

                # One-hot encode the label
```

```
        one_hot_label = one_hot_encode(label)

        # Append the image, and it's one hot encoded label to the full, processed list o
        standard_list.append((standardized_im, one_hot_label))

    return standard_list

# Standardize all training images
STANDARDIZED_LIST = standardize(IMAGE_LIST)
```

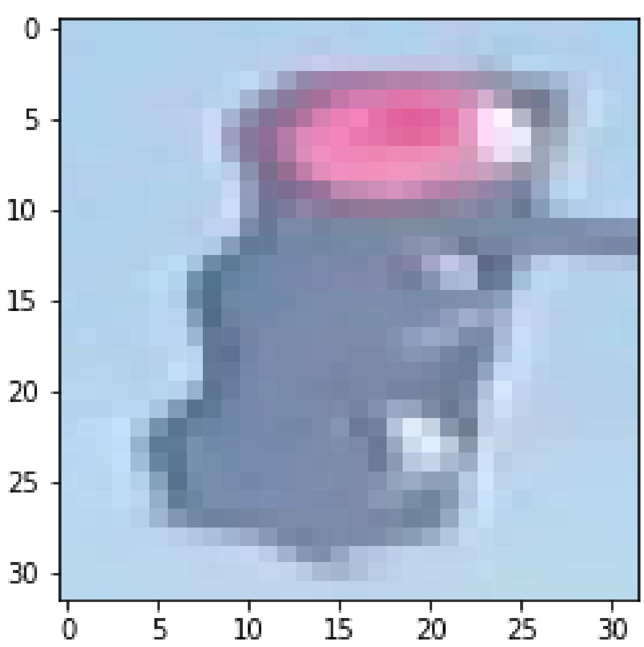
2.3

STANDARDIZED_LISTIMAGE_LIST

In [8]:
```
## TODO: Display a standardized image and its label
index_test = 8
img_standard = STANDARDIZED_LIST[index_test][0]
img_standard_label = STANDARDIZED_LIST[index_test][1]
print(img_standard.shape)
print(img_standard_label)
plt.imshow(img_standard)
```

```
(32, 32, 3)
[1, 0, 0]
```

Out[8]: <matplotlib.image.AxesImage at 0x7fd7601c6f28>



3 3.

HSV

1.
   - HSV3
   - notebook
2.

notebook

3.1

HSV

3.2 RGBHSV

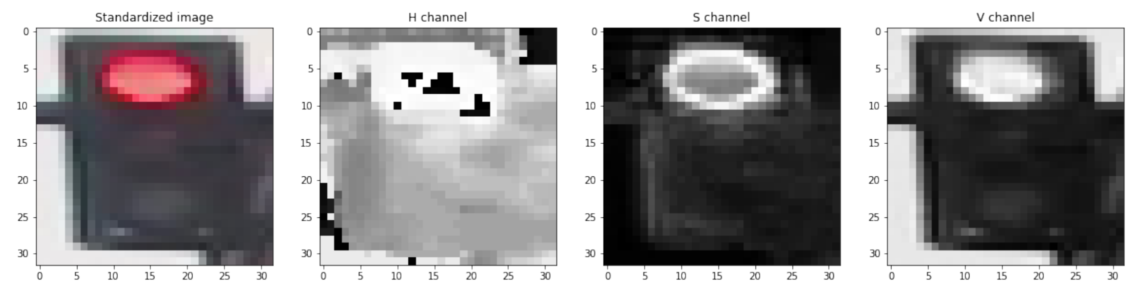
RGBHSV

```
In [16]: # Convert and image to HSV colorspace
         # Visualize the individual color channels

         image_num = 0
         test_im = STANDARDIZED_LIST[image_num][0]
         test_label = STANDARDIZED_LIST[image_num][1]

         # Convert to HSV
         hsv = cv2.cvtColor(test_im, cv2.COLOR_RGB2HSV)

         # Print image label
         print('Label [red, yellow, green]: ' + str(test_label))

         # HSV channels
         h = hsv[:,:,0]
         s = hsv[:,:,1]
         v = hsv[:,:,2]

         # Plot the original image and the three channels
         f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20,10))
         ax1.set_title('Standardized image')
         ax1.imshow(test_im)
         ax2.set_title('H channel')
         ax2.imshow(h, cmap='gray')
         ax3.set_title('S channel')
         ax3.imshow(s, cmap='gray')
```

```
        ax4.set_title('V channel')
        ax4.imshow(v, cmap='gray')
```

```
Label [red, yellow, green]: [1, 0, 0]
```

```
Out[16]: <matplotlib.image.AxesImage at 0x7fd72946deb8>
```



HSV
RGB/HSV

```
In [95]: def y_value_sum(hsv_img):
             y_sum = list(range(3))
             y_sum[0] = np.sum(hsv_img[5:12,5:25,2])*0.1/(7 * 20)
             y_sum[1] = np.sum(hsv_img[11:22,5:25,2])*0.1/(11*20)
             y_sum[2] = np.sum(hsv_img[21:28,5:25,2])*0.1/(7*20)
             return y_sum
         ## TODO: Create a brightness feature that takes in an RGB image and outputs a feature v
         ## This feature should use HSV colorspace values
         def create_feature(rgb_image):

             ## TODO: Convert image to HSV color space
             hsv = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2HSV)
             ## TODO: Create and return a feature value and/or vector
             feature = []
             feature = y_value_sum(hsv)
             return feature
```

3.3

```
In [92]: def y_color_sum(hsv_img):
             y_sum = [0,0,0]

             lower_r_1 = np.array([0,43,46])                              #choice red mask
```

```
    upper_r_1 = np.array([10,256,256])
    mask_r_1 = cv2.inRange(hsv_img, lower_r_1, upper_r_1)

    lower_r_2 = np.array([156,43,46])
    upper_r_2 = np.array([180,256,256])
    mask_r_2 = cv2.inRange(hsv_img, lower_r_2, upper_r_2)

    masked_image_r1 = np.copy(hsv_img)
    masked_image_r1[mask_r_1 == 0] = [0, 0, 0]

    masked_image_r2 = np.copy(hsv_img)
    masked_image_r2[mask_r_2 == 0] = [0, 0, 0]
    masked_image_r = masked_image_r1 + masked_image_r2

    a = masked_image_r[:11,:,0]
    y_sum[0] = np.sum(a > 0)

    lower_y = np.array([26,43,46])                        #choice yellow mask
    upper_y = np.array([34,256,256])
    mask_y = cv2.inRange(hsv_img, lower_y, upper_y)

    masked_image_y = np.copy(hsv_img)
    masked_image_y[mask_y == 0] = [0, 0, 0]
    b = masked_image_y[11:21,:,0]
    y_sum[1] = np.sum(b > 0)

    lower_g = np.array([35,43,46])                        #choice green mask
    upper_g = np.array([99,256,256])
    mask_g = cv2.inRange(hsv_img, lower_g, upper_g)

    masked_image_g = np.copy(hsv_img)
    masked_image_g[mask_g == 0] = [0, 0, 0]
    c = masked_image_g[21:,:,0]
    y_sum[2] = np.sum(c > 0)

    # Plot the original image and the three channels
#     f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20,10))
#     ax1.set_title('Standardized image')
#     ax1.imshow(cv2.cvtColor(hsv_img, cv2.COLOR_HSV2RGB))
#     ax2.set_title('r channel')
#     ax2.imshow(masked_image_r)
#     ax3.set_title('y channel')
#     ax3.imshow(masked_image_y)
#     ax4.set_title('g channel')
#     ax4.imshow(masked_image_g)
    return y_sum
# (Optional) Add more image analysis and create more features
def color_feature(rgb_image):
```

```
        ## TODO: Convert image to HSV color space
        hsv = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2HSV)
        ## TODO: Create and return a feature value and/or vector
        feature = []
        feature = y_color_sum(hsv)

        return feature
```

3.4 13

1.y33 2.hsv0601200

4 4.

RGB

estimate_label

###

```
In [116]: # This function should take in RGB image input
          # Analyze that image using your feature creation code and output a one-hot encoded lab
          def estimate_label(rgb_image):

              ## TODO: Extract feature(s) from the RGB image and use those features to
              ## classify the image and output a one-hot encoded label
              #predicted_label = []
              up_mid_down_sum_list = create_feature(rgb_image)
              r_y_g_sum_list = color_feature(rgb_image)
              max_index_1 = up_mid_down_sum_list.index(max(up_mid_down_sum_list))
              max_index_2 = r_y_g_sum_list.index(max(r_y_g_sum_list))
              if r_y_g_sum_list == [0,0,0]:
                  predicted_label = [0,1,0]
              elif max_index_1 == 0 and max_index_2 == 0:
                  predicted_label = [1,0,0]
              elif max_index_1 == 2 and max_index_2 == 2:
                  predicted_label = [0,0,1]
              else:
                  predicted_label = [0,1,0]
          #     if r_y_g_sum_list[0]
          #     if max_index_2 == 0 :
          #         predicted_label = [1,0,0]
          #     elif max_index_2 == 2 :
          #         predicted_label = [0,0,1]
          #     else:
          #         predicted_label = [0,1,0]
```

```
        return predicted_label
```

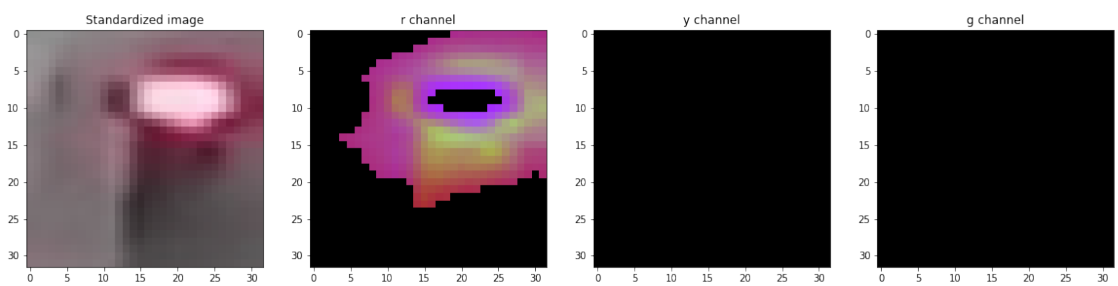
4.1

notebook “”
“” 1. 90 2.

4.1.1

standardize

```
In [11]: # Using the load_dataset function in helpers.py
         # Load test data
         TEST_IMAGE_LIST = helpers.load_dataset(IMAGE_DIR_TEST)

         # Standardize the test data
         STANDARDIZED_TEST_LIST = standardize(TEST_IMAGE_LIST)

         # Shuffle the standardized test data
         random.shuffle(STANDARDIZED_TEST_LIST)

In [91]: color_feature(STANDARDIZED_TEST_LIST[30][0])

Out[91]: [212, 0, 0]
```



4.2

“”

MISCLASSIFIED

```
In [117]: # Constructs a list of misclassified images given a list of test images and their labe
          # This will throw an AssertionError if labels are not standardized (one-hot encoded)

          def get_misclassified_images(test_images):
              # Track misclassified images by placing them into a list
              misclassified_images_labels = []
```

```
    # Iterate through all the test images
    # Classify each image and compare to the true label
    for image in test_images:

        # Get true data
        im = image[0]
        true_label = image[1]
        assert(len(true_label) == 3), "The true_label is not the expected length (3)."

        # Get predicted label from your classifier
        predicted_label = estimate_label(im)
        assert(len(predicted_label) == 3), "The predicted_label is not the expected le

        # Compare true and predicted labels
        if(predicted_label != true_label):
            # If these labels are not equal, the image has been misclassified
            misclassified_images_labels.append((im, predicted_label, true_label))

    # Return the list of misclassified [image, predicted_label, true_label] values
    return misclassified_images_labels


# Find all misclassified images in a given test set
MISCLASSIFIED = get_misclassified_images(STANDARDIZED_TEST_LIST)

# Accuracy calculations
total = len(STANDARDIZED_TEST_LIST)
num_correct = total - len(MISCLASSIFIED)
accuracy = num_correct/total

print('Accuracy: ' + str(accuracy))
print("Number of misclassified images = " + str(len(MISCLASSIFIED)) +' out of '+ str(t
```

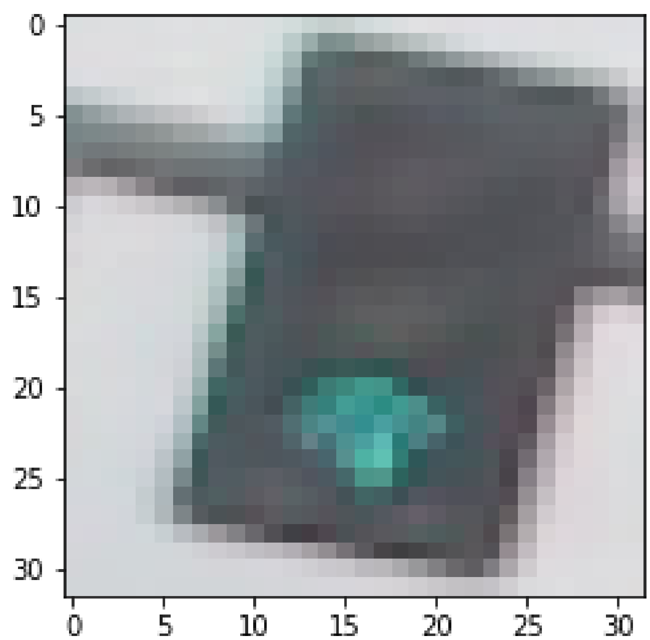
```
Accuracy: 0.936026936026936
Number of misclassified images = 19 out of 297
```

###
MISCLASSIFIED

In [143]:
```
# Visualize misclassified example(s)
## TODO: Display an image in the `MISCLASSIFIED` list
image_miss_index =3
image_miss = MISCLASSIFIED[image_miss_index]
plt.imshow(image_miss[0])
## TODO: Print out its predicted label - to see what the image *was* incorrectly class
```

```
print(image_miss[1])
print(image_miss[2])
up_mid_down_sum_list1 = create_feature(image_miss[0])
print(up_mid_down_sum_list1)
r_y_g_sum_list1 = color_feature(image_miss[0])
print(r_y_g_sum_list1)
```

```
[0, 1, 0]
[0, 0, 1]
[11.244285714285715, 11.539545454545456, 11.322142857142858]
[0, 0, 64]
```



2

1. 2. 3.

4.3

""

`MISCLASSIFIED` **[misclassified_image, predicted_label, true_label]**

[0,1,0]

```
In [144]: # Importing the tests
          import test_functions
```

```python
tests = test_functions.Tests()

if(len(MISCLASSIFIED) > 0):
    # Test code for one_hot_encode function
    tests.test_red_as_green(MISCLASSIFIED)
else:
    print("MISCLASSIFIED may not have been populated with images.")
```

TEST PASSED

5 5.

1. 90 2.
   -

5.0.1

* > 95 * * 100