Standardizing the Data

October 14, 2018

0.1 ### Day and Night Image Classifier

The day/night image dataset consists of 200 RGB color images in two categories: day and night. There are equal numbers of each example: 100 day images and 100 night images.

We'd like to build a classifier that can accurately label these images as day or night, and that relies on finding distinguishing features between the two types of images!

Note: All images come from the AMOS dataset (Archive of Many Outdoor Scenes).

0.1.1 Import resources

Before you get started on the project code, import the libraries and resources that you'll need.

```
In [5]: import cv2 # computer vision library
        import helpers

        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg

        %matplotlib inline
```

0.2 Training and Testing Data

The 200 day/night images are separated into training and testing datasets.

- 60% of these images are training images, for you to use as you create a classifier.
- 40% are test images, which will be used to test the accuracy of your classifier.

First, we set some variables to keep track of some where our images are stored:

```
image_dir_training: the directory where our training image data is stored
image_dir_test: the directory where our test image data is stored
```

```
In [6]: # Image data directories
        image_dir_training = "day_night_images/training/"
        image_dir_test = "day_night_images/test/"
```

0.3 Load the datasets

These first few lines of code will load the training day/night images and store all of them in a variable, `IMAGE_LIST`. This list contains the images and their associated label ("day" or "night").

For example, the first image-label pair in `IMAGE_LIST` can be accessed by index: `IMAGE_LIST[0][:]`.

```
In [7]: # Using the load_dataset function in helpers.py
        # Load training data
        IMAGE_LIST = helpers.load_dataset(image_dir_training)
```

---

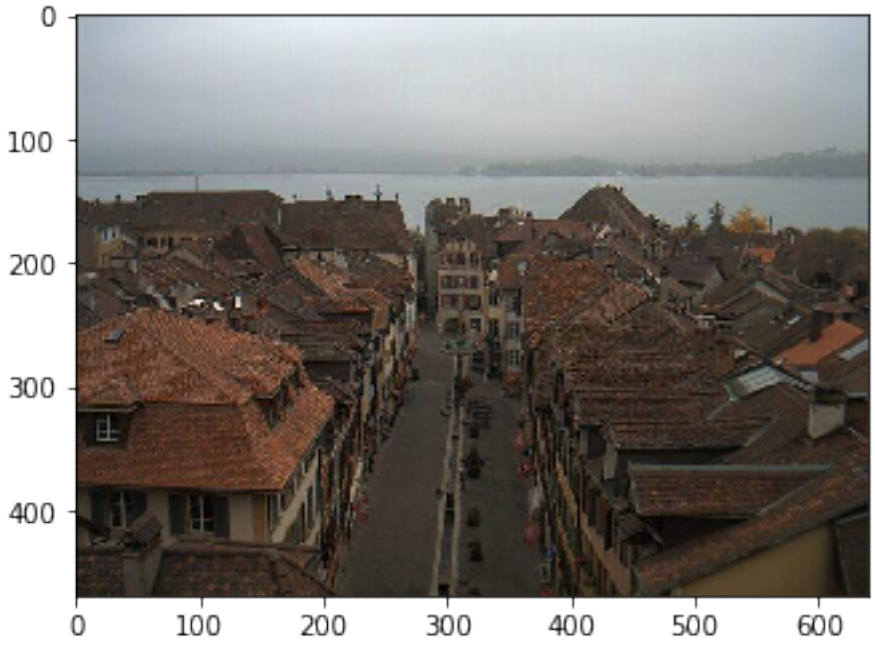
1 1. Visualize the input images

```
In [8]: # Print out 1. The shape of the image and 2. The image's label

        # Select an image and its label by list index
        image_index = 0
        selected_image = IMAGE_LIST[image_index][0]
        selected_label = IMAGE_LIST[image_index][1]

        # Display image and data about it
        plt.imshow(selected_image)
        print("Shape: "+str(selected_image.shape))
        print("Label: " + str(selected_label))
```

```
Shape: (469, 640, 3)
Label: day
```



2 2. Pre-process the Data

After loading in each image, you have to standardize the input and output!

Solution code You are encouraged to try to complete this code on your own, but if you are struggling or want to make sure your code is correct, there will be solution code in **the next Notebook: Average Brightness Feature Extraction** in the `helpers.py` file. You can jump ahead and look into that python file to see complete `standardize_input` and `encode` function code. For this day and night challenge, you can often jump one notebook ahead to see the solution code for a previous notebook!

2.0.1 Input

It's important to make all your images the same size so that they can be sent through the same pipeline of classification steps! Every input image should be in the same format, of the same size, and so on.

TODO: Standardize the input images

- Resize each image to the desired input size: 600x1100px (hxw).

```
In [12]: # This function should take in an RGB image and return a new, standardized version
         def standardize_input(image):

             ## TODO: Resize image so that all "standard" images are the same size 600x1100 (hxw
             standard_im = cv2.resize(image, (1100, 600))

             return standard_im
```

2.0.2 TODO: Standardize the output

With each loaded image, you also need to specify the expected output. For this, use binary numerical values 0/1 = night/day.

```
In [13]: # Examples:
         # encode("day") should return: 1
         # encode("night") should return: 0

         def encode(label):

             numerical_val = 0
             ## TODO: complete the code to produce a numerical label
```

```
    if(label == "day"): return 1
    else:
        return 0
```

2.1 Construct a STANDARDIZED_LIST of input images and output labels.

This function takes in a list of image-label pairs and outputs a **standardized** list of resized images and numerical labels.

This uses the functions you defined above to standardize the input and output, so those functions must be complete for this standardization to work!

```
In [14]: 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)

        # Create a numerical label
        binary_label = encode(label)

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

    return standard_list

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

2.2 Visualize the standardized data

Display a standardized image from STANDARDIZED_LIST.

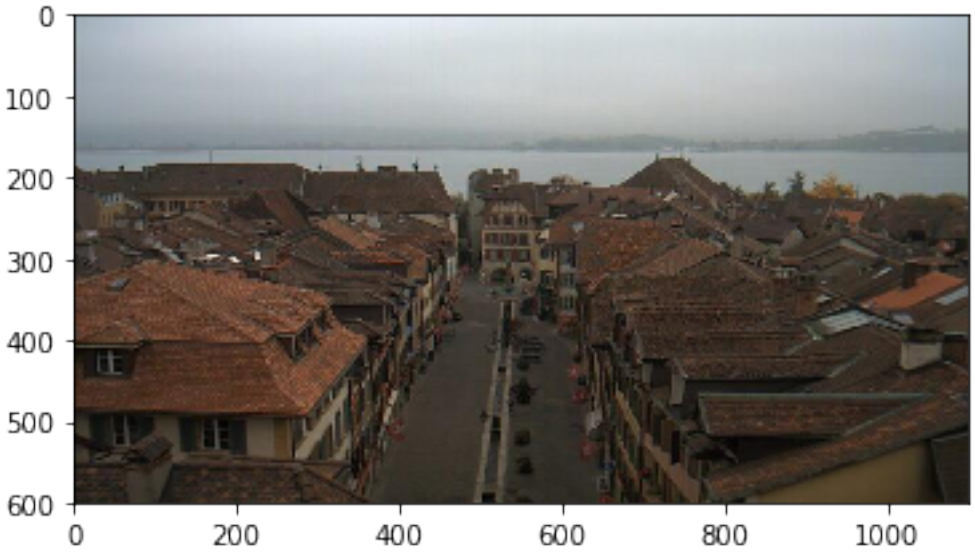
```
In [15]: # Display a standardized image and its label

# Select an image by index
image_num = 0
selected_image = STANDARDIZED_LIST[image_num][0]
selected_label = STANDARDIZED_LIST[image_num][1]

# Display image and data about it
## TODO: Make sure the images have numerical labels and are of the same size
```

```
        plt.imshow(selected_image)
        print("Shape: "+str(selected_image.shape))
        print("Label [1 = day, 0 = night]: " + str(selected_label))
```

```
Shape: (600, 1100, 3)
Label [1 = day, 0 = night]: 1
```



In [ ]: