A1. CV. Basic Image Classification

March 2, 2020

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.

[153]:
```
import cv2 # computer vision library
import helpers

import numpy as np
import matplotlib.pyplot as plt

%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`

[154]:
```
# 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][:]`.

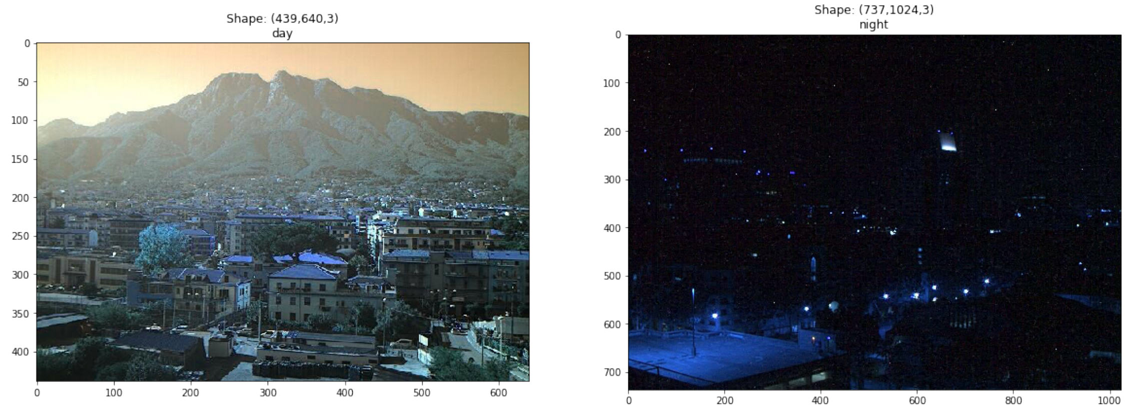
[155]:
```
# Using the load_dataset function in helpers.py
# Load training data
IMAGE_LIST = helpers.load_dataset(image_dir_training)
```

0.4 Visualize sample day and night images

[156]:
```
# Select an image and its label by list index
image_index = 6
selected_image = IMAGE_LIST[image_index][0]
selected_label = IMAGE_LIST[image_index][1]

## TODO: Create a subplot of a day image and a night image. The titles should␣
↪consist of the shape and label
# of the image

#Find beginning of night images
nightIndex = 0
done = False

while(not done):
    if(IMAGE_LIST[nightIndex][1] == "night"):
        done = True
    else:
        nightIndex = nightIndex + 1

f,(ax1,ax2) = plt.subplots(1, 2, figsize=(20,10))

#Title needs to be formatted as: Shape: img.shape\nIMAGE_LIST[i][1] Maybe␣
↪approach via removing elements of tuple and using the values
#to generate String values?

title1 = 'Shape: (' + ','.join(str(v) for v in selected_image.shape) + ')\n' +␣
↪IMAGE_LIST[image_index][1];
ax1.set_title(title1)
ax1.imshow(selected_image)

image_index = nightIndex
selected_image = IMAGE_LIST[image_index][0]
selected_label = IMAGE_LIST[image_index][1]
```

```
title2 = 'Shape: (' + ','.join(str(v) for v in selected_image.shape) + ')\n' +␣
↪IMAGE_LIST[image_index][1];
ax2.set_title(title2)
ax2.imshow(selected_image)
```

[156]:
```
<matplotlib.image.AxesImage at 0x7f1520b43290>
```



0.5 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.

[158]:
```
## Standardize the input images
# Resize each image to the desired input size: 600x1100px (hxw).
# This function should take in an RGB image and return a new, standardized␣
↪version
def standardize_input(image, width, height):

    # Resize image and pre-process so that all "standard" images are the same␣
↪size
    # cv2.resize
    dimensions = (width, height)
    #Standardize first -> BRG to RGB
    #image_copy = np.copy(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    #Resize image
    standard_im = cv2.resize(image, dimensions, interpolation = cv2.INTER_AREA)

    return standard_im

# With each loaded image, we also specify the expected output.
# For this, we use binary numerical values 0/1 = night/day.
# Examples:
# encode("day") should return: 1
# encode("night") should return: 0
def encode(label):

    numerical_val = 0
    if(label == "day"):
        numerical_val = 1
    elif(label == "night"):
        numerical_val = 0

    return numerical_val


## Standardize the output using both functions above, standardize the input␣
↪images and output labels
def standardize(image_list):

    # Empty image data array
    standard_list = []

    # Iterate through all the image-label pairs
    for item in image_list:

        # Standardize the image
        standardImage = standardize_input(item[0], 1100, 600)

        # Create a numerical label
        numLabel = encode(item[1])

        # Append the image, and its one hot encoded label to the full,␣
↪processed list of image data
        #standard_list.append((image_list[item][0],image_list[item][1]))
        standard_list.append((standardImage, numLabel))

    return standard_list

# Standardize all training images

## TODO: Code the needed functions in the helpers file in order to return a␣
↪standardized list.
# NOTE: Adding copy of helpers' necessary functions here because for some␣
↪reason it won't work using them from helpers.py
# The code is identical but it only works if it's on the same notebook not␣
↪really sure why ()
```

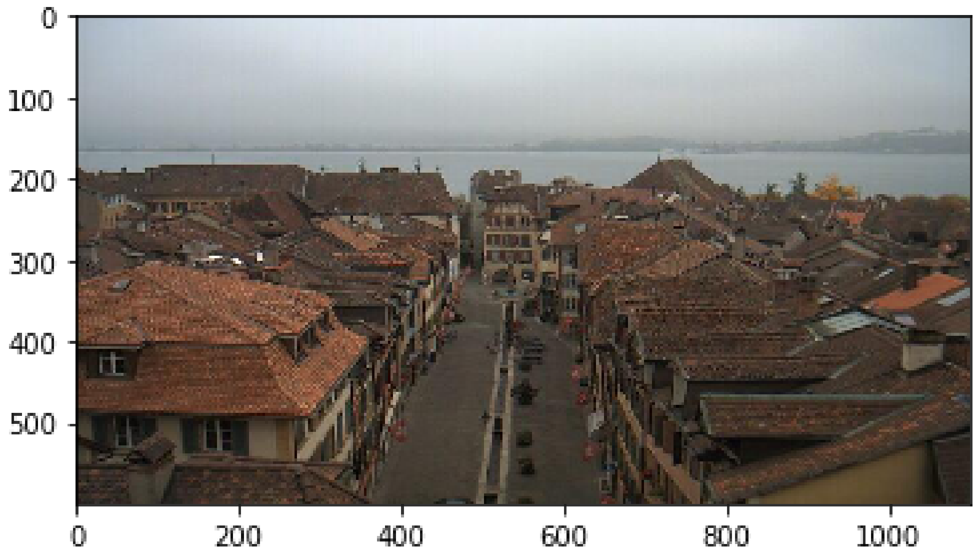
```
STANDARDIZED_LIST = standardize(IMAGE_LIST)
```

0.6 Visualize the standardized data

Display a standardized image from STANDARDIZED_LIST.

[159]:
```
# 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
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
```



1 Feature Extraction

Create a feature that represents the brightness in an image. We'll be extracting the **average brightness** using HSV colorspace. Specifically, we'll use the V channel (a measure of brightness), add up the pixel values in the V channel, then divide that sum by the area of the image to get the average Value of the image.

1.0.1 Find the average brigtness using the V channel

This function takes in a **standardized** RGB image and returns a feature (a single value) that represent the average level of brightness in the image. We'll use this value to classify the image as day or night.

[160]:
```
# Find the average Value or brightness of an image
def avg_brightness(rgb_image):
    ## TODO: Get the average brightness from an image using the HSV color space.

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

    # Add up all the pixel values in the V channel
    v = hsv[:,:,2]

    totalSum = 0
    for element in v:
        for pixel in element:
            totalSum = totalSum + pixel

    # find the avg
    avg = totalSum/(1100*600)

    return avg

avg_brightness(selected_image)
```

[160]:
```
115.09862424242424
```

[161]:
```
# Testing average brightness levels
# Look at a number of different day and night images and think about
# what average brightness value separates the two types of images

# As an example, a "night" image is loaded in and its avg brightness is␣
↪displayed
image_num = 239
test_im = STANDARDIZED_LIST[image_num][0]

avg = avg_brightness(test_im)
print('Avg brightness: ' + str(avg))
plt.imshow(test_im)
```

```
Avg brightness: 99.23268181818182
```

[161]:
```
<matplotlib.image.AxesImage at 0x7f152098dbd0>
```

2 Classification and Visualizing Error

In this section, we'll turn our average brightness feature into a classifier that takes in a standardized image and returns a `predicted_label` for that image. This `estimate_label` function should return a value: 0 or 1 (night or day, respectively).

2.0.1 Build a complete classifier

Complete this code so that it returns an estimated class label given an input RGB image.

[162]:
```python
# This function should take in RGB image input
def estimate_label(rgb_image):
    ## TODO: Use the avg brightness feature to predict a label (0, 1)
    # Extract average brightness feature from an RGB image
    avg = avg_brightness(rgb_image)

    predicted_label = 0

    # Define a threshold value
    threshold = 103

    if(avg >= threshold):
        predicted_label = 1

    # if the average brightness is above the threshold value, we classify it as␣
↪"day"
    # else, the pred-cted_label can stay 0 (it is predicted to be "night")

    return predicted_label
```

2.1 Testing the classifier

Here is where we test your classification algorithm using our test set of data that we set aside at the beginning of the notebook!

Since we are using a pretty simple brightess feature, we may not expect this classifier to be 100% accurate. We'll aim for around 75-85% accuracy usin this one feature.

2.1.1 Test dataset

Below, we load in the test dataset, standardize it using the `standardize` function you defined above, and then **shuffle** it; this ensures that order will not play a role in testing accuracy.

[163]:
```
import random

# 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)
```

2.2 Determine the Accuracy

Compare the output of your classification algorithm (a.k.a. your "model") with the true labels and determine the accuracy.

This code stores all the misclassified images, their predicted labels, and their true labels, in a list called `misclassified`.

[164]:
```
# Constructs a list of misclassified images given a list of test images and␣
↪their labels
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]

        ## TODO:
        # Get predicted label from your classifier
        predicted_label = estimate_label(im)

        # Compare true and predicted labels
        # If these labels are not equal, the image has been misclassified.␣
↪Append a tuple of
        # image, prediction, and true label to the misclassified list.

        if(true_label != predicted_label):
            misclassified_images_labels.append((im,true_label,predicted_label))

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

[165]:
```
# Find all misclassified images in a given test set
MISCLASSIFIED = get_misclassified_images(STANDARDIZED_TEST_LIST)

## TODO: Calculate the accuracy of the classifier. Accuracy = number of correct␣
↪/ total number of images
# Accuracy calculations
total = len(STANDARDIZED_TEST_LIST)

accuracy = (total-len(MISCLASSIFIED))/total

print('Accuracy: ' + str(accuracy))
print("Number of misclassified images = " + str(len(MISCLASSIFIED)) +' out of␣
↪'+ str(total))
```

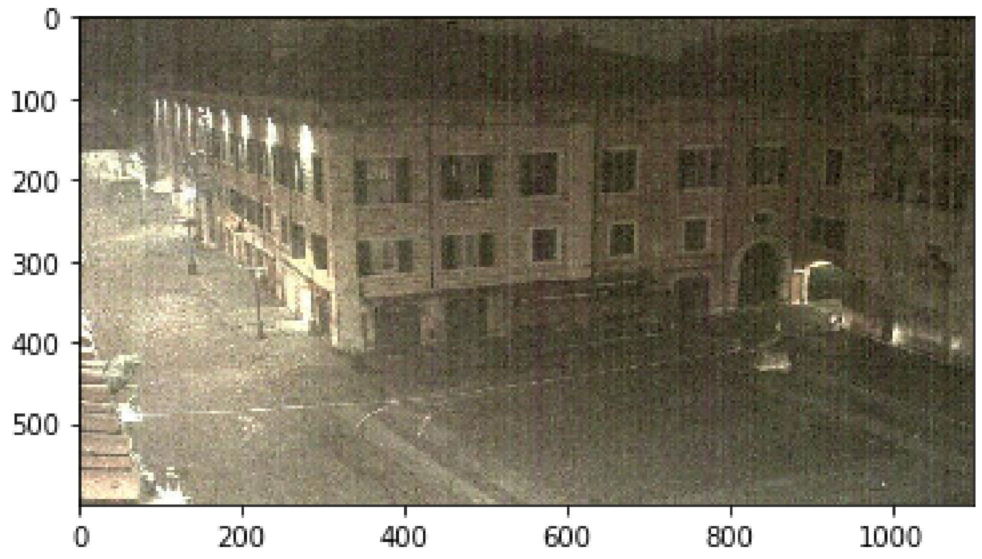
```
Accuracy: 0.925
Number of misclassified images = 12 out of 160
```

### Visualize the misclassified images

Visualize some of the images you classified wrong (in the `MISCLASSIFIED` list) and note any qualities that make them difficult to classify.

[166]:
```
# Visualize misclassified example(s)
## TODO: Display an image in the `MISCLASSIFIED` list
sampleMisclassified = 11          #CHANGE this value to see different␣
 ↪misclassified images
plt.imshow(MISCLASSIFIED[sampleMisclassified][0])
## TODO: Print out its predicted label - to see what the image *was*␣
 ↪incorrectly classified as
print('Predicted label: ', MISCLASSIFIED[sampleMisclassified][2], '\nTrue label:
 ↪ ', MISCLASSIFIED[sampleMisclassified][1])
```

```
Predicted label:  1
True label:  0
```



[ ]: