# mean_value+otsu_binarization

February 16, 2020

## 1 Introduction

Here we aim to classify images into daylight landscapes, nightlight landscapes and portraits by designing features and using kNN classifier.

[1]:
```
# all imports
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
```

### 1.1 Average brightness

In the HSV colorspace, the 'value' of each pixel corresponds to the brightness of that pixel. Since, we know that night landscapes are inherently dark and vice versa, we can use average value of the images as a feature to distinguish between the two.

[2]:
```
# get mean value
def get_mean_value(img):
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    mean_value = np.mean(hsv_img[:,:,2])
    return hsv_img, mean_value
```

### 1.2 Otsu binarization

When we binarize a portrait, the face and the background form the foreground and background respectively. Also, there is a clear distinction between the average intensities in the grayscale image corresponding to the binary image. This is not true for any landscape. Hence, we use the difference between average grayscale intensities of the foreground and background of the images as a feature.

[3]:
```
# get otsu binarization
def get_otsu(img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray_img,(5,5),0)

    # find normalized_histogram, and its cumulative distribution function
    hist = cv2.calcHist([blur],[0],None,[256],[0,256])
    hist_norm = hist.ravel()/hist.sum()
```

```
    Q = hist_norm.cumsum()
    bins = np.arange(256)
    fn_min = np.inf
    thresh = -1
    for i in range(1,256):
        p1,p2 = np.hsplit(hist_norm,[i]) # probabilities
        q1,q2 = Q[i],Q[255]-Q[i] # cumulative sum of classes
        if q1 < 1.e-6 or q2 < 1.e-6:
            continue
        b1,b2 = np.hsplit(bins,[i]) # weights
        # finding means and variances
        m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
        v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
        # calculates the minimization function
        fn = v1*q1 + v2*q2
        if fn < fn_min:
            fn_min = fn
            thresh = i
    # find otsu's threshold value with OpenCV function
    ret, otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    # get inter-class seperation of mean intensities
    white_pixels = gray_img[otsu == 255]
    black_pixels = gray_img[otsu == 0]
    mean_diff = white_pixels.mean() - black_pixels.mean()
    abs_diff = abs(mean_diff)
    return otsu, abs_diff
```

[4]:
```
# get image features
feats = []
labels = []
for file in os.listdir('../Train Images/portrait'):
    img = cv2.imread('../Train Images/portrait/' + file)
    hsv_img, mean_value = get_mean_value(img)
    otsu, abs_diff = get_otsu(img)
    feat_vec = [mean_value, abs_diff]
    feats.append(feat_vec)
    labels.append([0])

for file in os.listdir('../Train Images/day-landscape'):
    img = cv2.imread('../Train Images/day-landscape/' + file)
    hsv_img, mean_value = get_mean_value(img)
    otsu, abs_diff = get_otsu(img)
    feat_vec = [mean_value, abs_diff]
    feats.append(feat_vec)
    labels.append([1])
```

```
for file in os.listdir('../Train Images/night-landscape'):
    img = cv2.imread('../Train Images/night-landscape/' + file)
    hsv_img, mean_value = get_mean_value(img)
    otsu, abs_diff = get_otsu(img)
    feat_vec = [mean_value, abs_diff]
    feats.append(feat_vec)
    labels.append([2])

feats = np.array(feats, dtype=np.float32)
labels = np.array(labels, dtype=np.float32)
```

[5]:
```
# train knn classifier
knn = cv2.ml.KNearest_create()
knn.train(feats, cv2.ml.ROW_SAMPLE, labels)
```

[5]: True

[6]:
```
# test knn classifier
for file in os.listdir('../Test Images'):
    img = cv2.imread('../Test Images/' + file)
    hsv_img, mean_value = get_mean_value(img)
    plt.imsave('./HSV Images/' + file, hsv_img)
    otsu, abs_diff = get_otsu(img)
    plt.imsave('./Binarized Images/' + file, otsu, cmap='binary')
    feat_vec = np.array([[np.float32(mean_value), np.float32(abs_diff)]],␣
 ↪dtype=np.float32)
    print(file)
    ret, results, neighbours ,dist = knn.findNearest(feat_vec, 3)
    print( "result:  {}\n".format(results))
```

```
[[110.07635  88.51414]]
test_1.jpeg
result:  [[0.]]

[[49.28376 69.33861]]
test_10.jpeg
result:  [[2.]]

[[70.24407 69.75939]]
test_11.jpeg
result:  [[2.]]

[[150.54861 101.02058]]
test_12.jpeg
result:  [[0.]]

[[96.91976 81.10951]]
test_2.jpeg
result:  [[2.]]
```

```
[[ 62.46497  104.793076]]
test_3.jpeg
result:  [[2.]]

[[77.30093 68.11229]]
test_4.jpeg
result:  [[2.]]

[[127.97794  88.18447]]
test_5.jpeg
result:  [[1.]]

[[87.304085 75.723335]]
test_6.jpeg
result:  [[2.]]

[[176.90126 134.08897]]
test_7.jpeg
result:  [[1.]]

[[77.61969 84.27217]]
test_8.jpeg
result:  [[2.]]

[[168.62503 126.42834]]
test_9.jpeg
result:  [[0.]]
```

[ ]: