KMeans Colouring Clustering (Quantization)

Table of Contents

Set-up and Imports

In [1]:

```
!pip install colour-science
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
import colour
import cv2
import functools
```

```
Requirement already satisfied: colour-science in c:\users\prannaya
\appdata\local\programs\python\python37\lib\site-packages (0.3.16)
Requirement already satisfied: six in c:\users\prannaya\appdata\lo
cal\programs\python\python37\lib\site-packages (from colour-scienc
e) (1.12.0)
Requirement already satisfied: imageio in c:\users\prannaya\appdat
a\local\programs\python\python37\lib\site-packages (from colour-sc
ience) (2.9.0)
Requirement already satisfied: scipy<2.0.0,>=1.1.0 in c:\users\pra
nnaya\appdata\local\programs\python\python37\lib\site-packages (fr
om colour-science) (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\prannaya
\appdata\roaming\python\python37\site-packages (from scipy<2.0.0,>
=1.1.0->colour-science) (1.20.3)
Requirement already satisfied: pillow in c:\users\prannaya\appdata
\roaming\python\python37\site-packages (from imageio->colour-scien
ce) (8.2.0)
```

Alter Brightness of an Image

In [2]:

```
def alter(img, value):
    h, s, v = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2HSV)) # Split in H, S,
 V
    v = np.where((v+value >= 0)&(v+value <= 255), v, v+value) # Adds the value,
while bounding the value
    return cv2.cvtColor(np.array(np.stack((h,s,v), axis=-1), np.uint8), cv2.COL
OR_HSV2BGR) # Merge and convert back to BGR
```

Standard Cluster with KMeans

Done in the LAB colour space, since RGB values do not change in a way that is close to how human eyesight perceives changes in colours, but LAB does.

In [3]:

```
def cluster(image, clusters):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    clf = MiniBatchKMeans(n_clusters = clusters, batch_size = 4096)
    labels = clf.fit_predict(image.reshape(-1, 3))
    return cv2.cvtColor(clf.cluster_centers_.astype("uint8")[labels].reshape(im
age.shape), cv2.COLOR_LAB2BGR, cv2.CV_8U)
```

Brightening and Darkening before and after Clustering

This is a brightening and darkening function which has parameters for value lightened and value darkened.

In [4]:

```
def cluster_brightened(image, clusters, lighten_value, darken_value):
    return alter(cluster(alter(image, lighten_value), clusters), -darken_value)
```

Standardising Colours

This is a function to make sure that all the output classes in the quantized image are the same.

This functon is very unstable and a better implementation is needed.

In [5]:

```
standardise_colours = lambda quantized_image: cv2.cvtColor(
    functools.reduce(
        lambda quantized_image,i:np.where(
            np.stack(
                [np.any(
                    quantized_image == functools.reduce(
                        lambda x, j: (colour.delta_E(i, j), j) if(colour.delta_
E(i, j) < x[0]) else x, # x is a tuple (max_value, map_colour), find map_colour
for min value of colour.delta_E
                        np.unique(quantized_image.reshape(-1, 3), axis=0), # Re
fers to unique class pairs
                        (float("inf"), 0) # Initial Value, with max_value of in
f and map_colour of 0
                    )[-1], # checks for map_colour
                    axis=-1 # Checks if any of the values at pixel-level is equ
al to map_colour
                )]*3,
            axis=-1), # Makes an image of shape the same as the quantized imag
e, with each pixel having the same L,A,B value
            np.full(quantized_image.shape,i), # Multiplies each trio to form of
a full colour image
            quantized_image # Initial Value is the Image
        ),
        cv2.cvtColor(np.array([[[127,127,127],[128,0,0],[0,0,128]]], np.uint8),
cv2.COLOR_BGR2LAB)[0], # unique values
        cv2.cvtColor(quantized_image, cv2.COLOR_BGR2LAB) # Initial Value is qua
ntized_image in L*AB Format
    ),
    cv2.COLOR_LAB2BGR, cv2.CV_8U # Converts Altered Image back to BGR
)
```

Testing

Testing Function

In [6]:

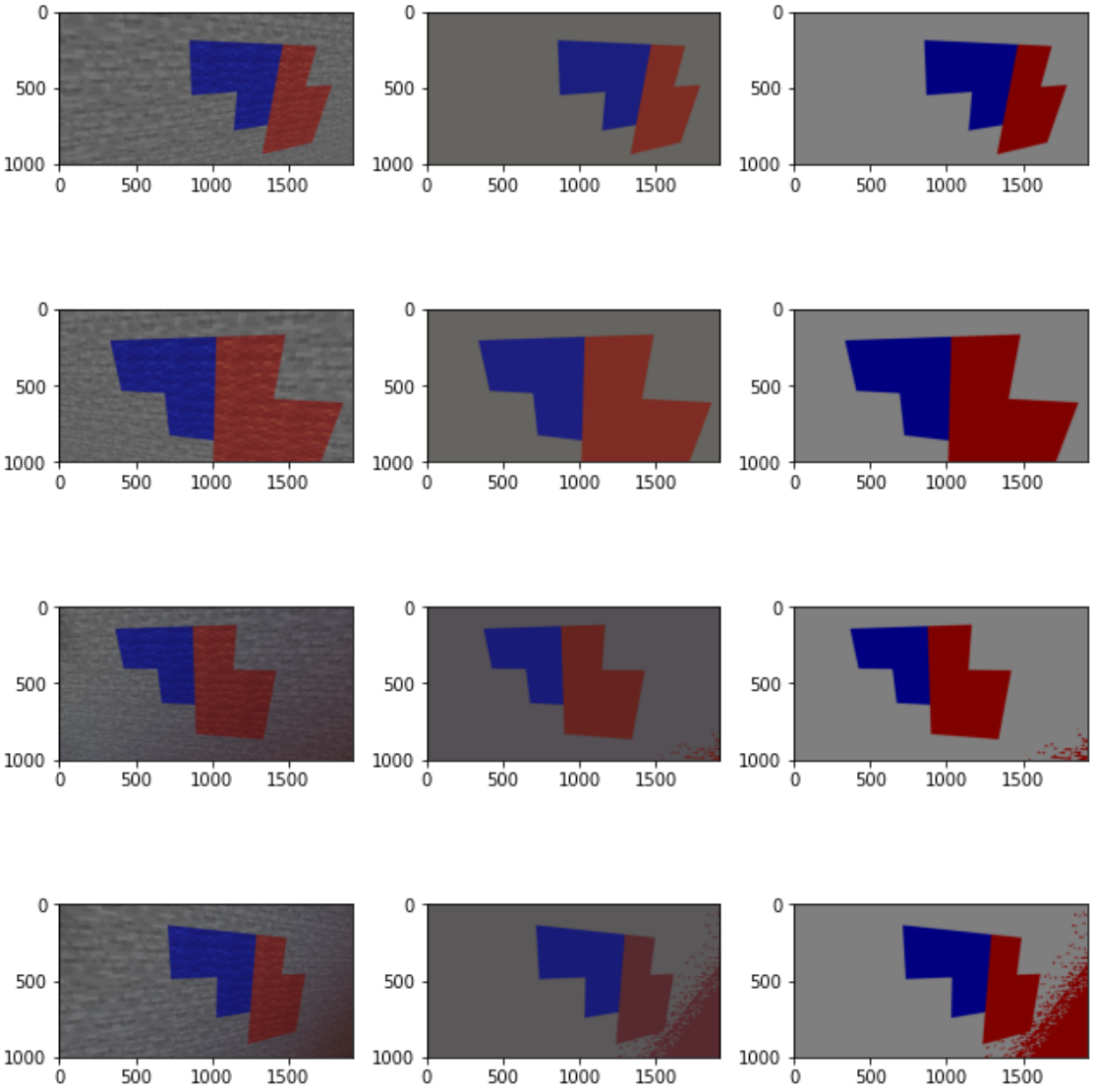
```
def cluster_test(clustering_function, lighten_value, darken_value):
    fig, axes = plt.subplots(4,3,figsize=(9,10))
    [
      [
        axes[i, j].imshow(img) for j,img in ( # Plot on Axes
            lambda img: (
                lambda img, q: list(enumerate((
                    img, q,
                    standardise_colours(q) # Standardized Image
                )))
            )(
                img, # Original Image
                clustering_function(img, 3, lighten_value, darken_value) # Quan
tized Image
            )
        )(
            cv2.imread(f"image_{i}.png") # Takes in the Images
        )
      ] for i in range(4)
    ]
    fig.tight_layout()
```

Unit Testing

The first image is the unquantized image, the second is the quantized image and third is the standardised image such that the grey is (127,127,127), the red is (128,0,0) and the blue is (0,0,128). We see that the darkness causes a faulty cluster.

In [7]:

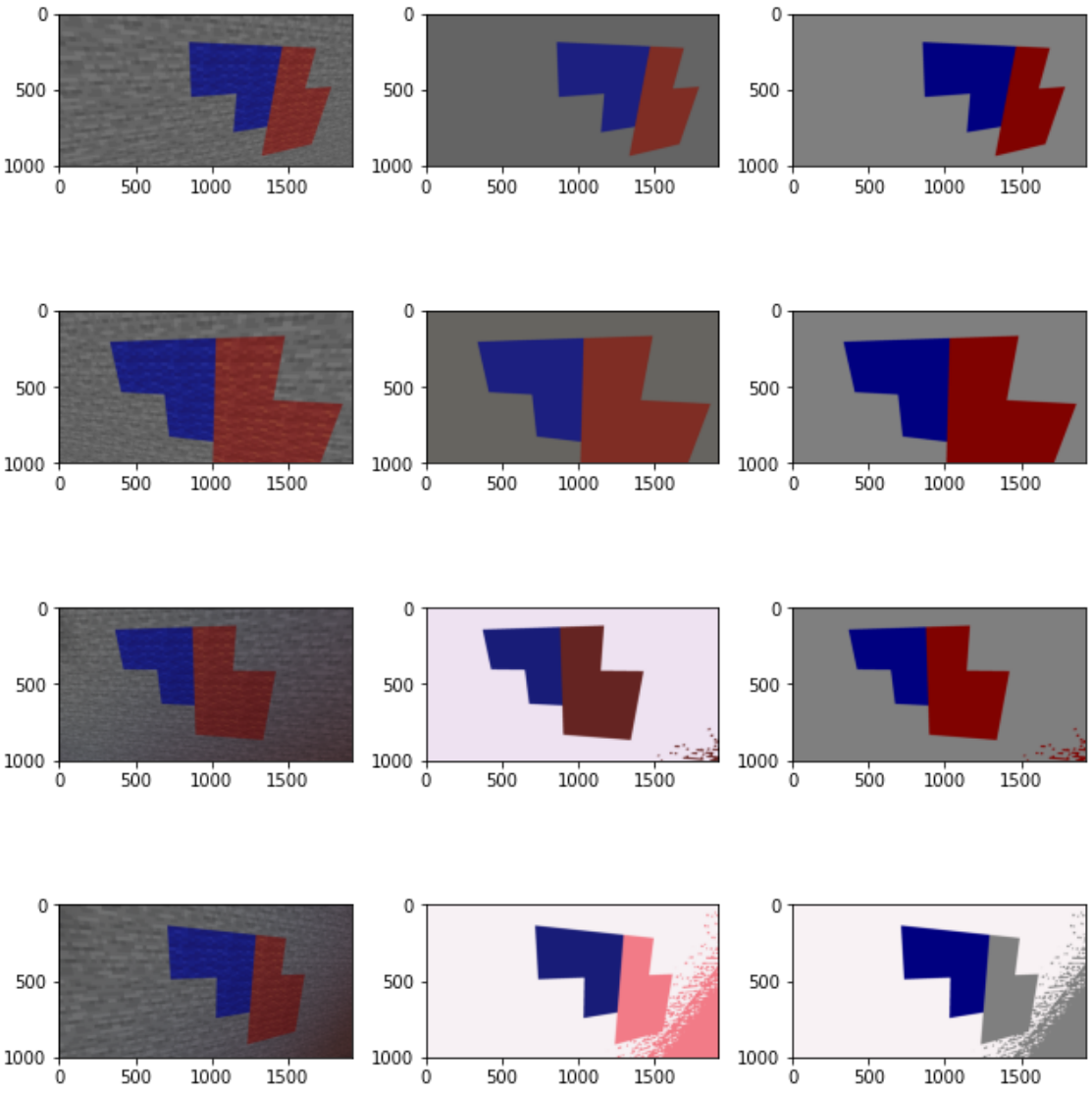
```
cluster_test(cluster_brightened, 0, 0)
```



We see that brightening the image can help

In [8]:

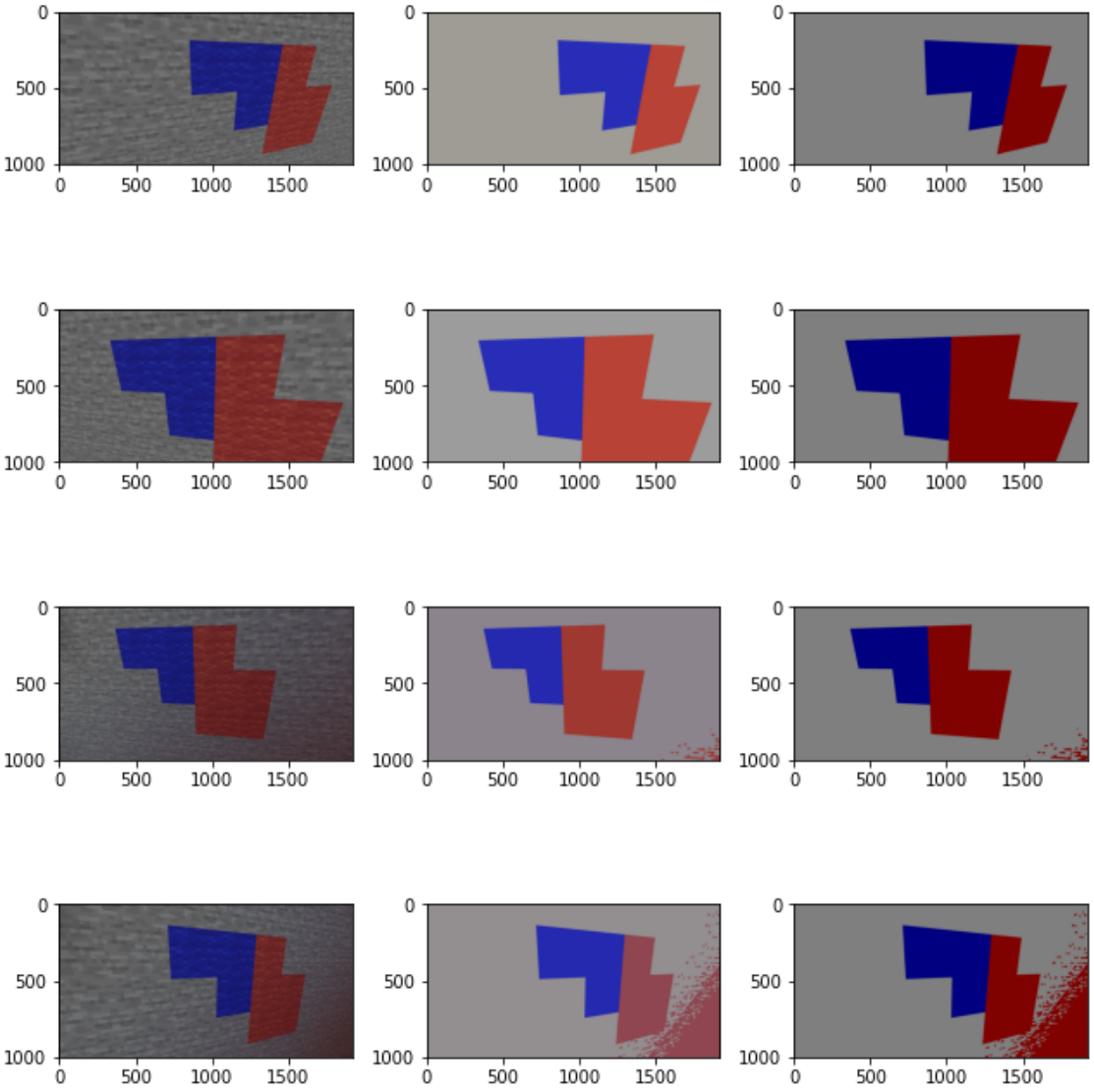
```
cluster_test(cluster_brightened, 100, 100)
```



if we do it too bright, we see that it does an effective job but the colour standardisation fails.

In [9]:

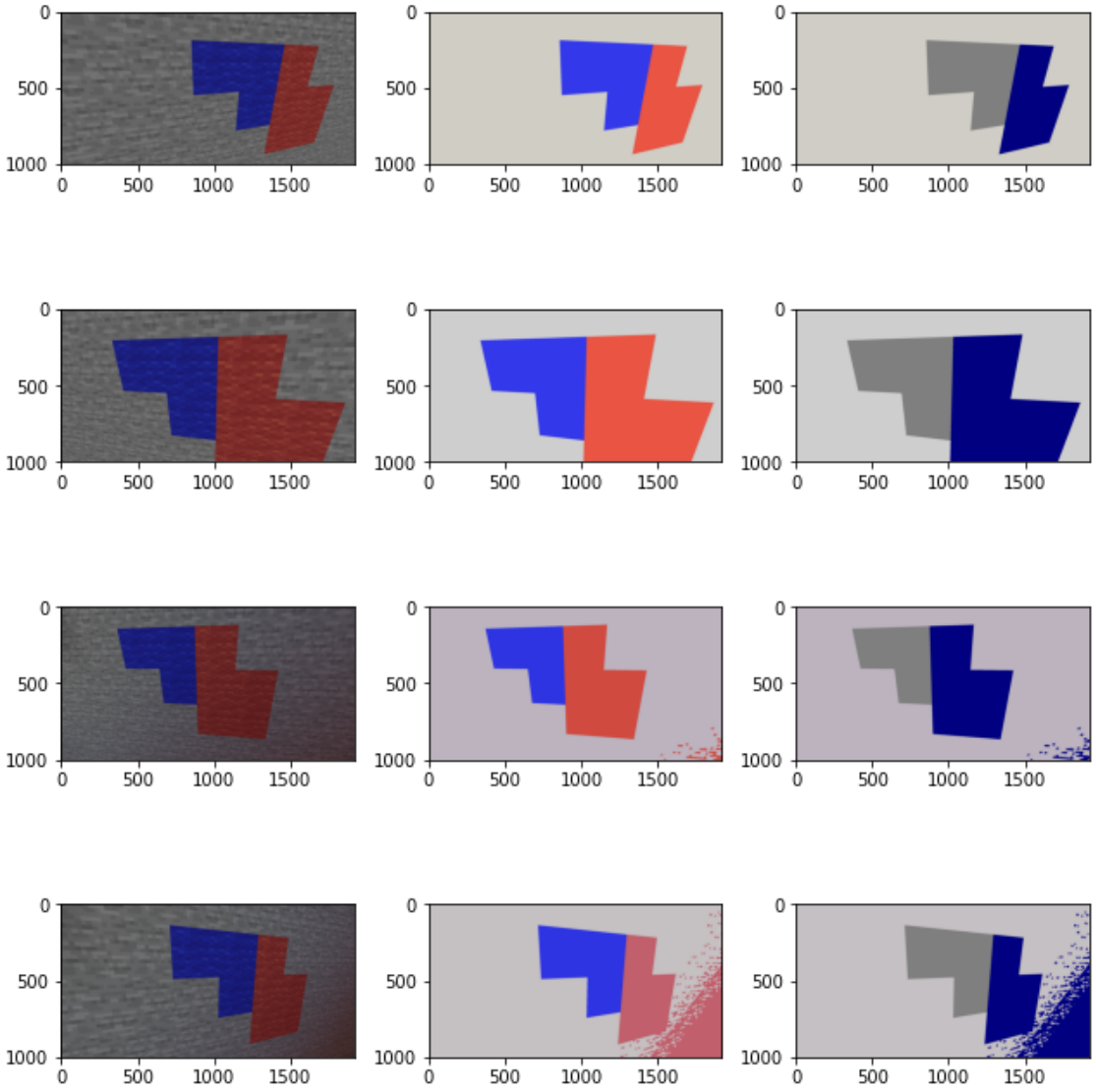
```
cluster_test(cluster_brightened, 200, 200)
```



Hence, we must darken it less than we brightened it

In [10]:

```
cluster_test(cluster_brightened, 200, 150)
```



Overall, the best clustering and standardisation will be with bright and uniform lighting with minimal reflection.