# NAME: SHUBHAM S. JAGTAP

# THE SPARKS FOUNDATION

# GRIPJAN21 Computer Vision Internship

## Task 2: Color Identification in Images using KMeans and OpenCV

### In this notebook, we will extract colors from images using KMeans Machine Learning algorithm and filtered images from a collection of images based on RGB values of colors.

### Load required packages

In [99]:
```
import numpy as np
import cv2
from collections import Counter
import os
import matplotlib.pyplot as plt
from skimage.color import rgb2lab, deltaE_cie76
from sklearn.cluster import KMeans
```

### Convert the resized image into HEX to use them as labels for the pie chart

In [100]:
```
def RGB2HEX(color):
    return "#{:02x}{:02x}{:02x}".format(int(color[0]), int(color[1]),
int(color[2]))
```

### Define a function to load and convert image to RGB for our model

In [101]:
```
def get_image(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image
```

We now define the complete code as a method that we can call to extract the top colors from the image and display them as a pie chart. I've named the method as get_colors and it takes 3 arguments:

```
image: The image whose colors we wish to extract.
number_of_colors: Total colors we want to extract.
show_chart: A boolean that decides whether we show the pie chart or not
```

First, we resize the image to the size 600 x 400. It is not required to resize it to a smaller size but we do so to lessen the pixels which'll reduce the time needed to extract the colors from the image. KMeans expects the input to be of two dimensions, so we use Numpy's reshape function to reshape the image data

Second, KMeans algorithm creates clusters based on the supplied count of clusters. In our case, it will form clusters of colors and these clusters will be our top colors. We then fit and predict on the same image to extract the prediction into the variable labels.

We use Counter to get count of all labels. To find the colors, we use clf.cluster*centers*. The ordered_colors iterates over the keys present in count, and then divides each value by 255. We could have directly divided each value by 255 but that would have disrupted the order.

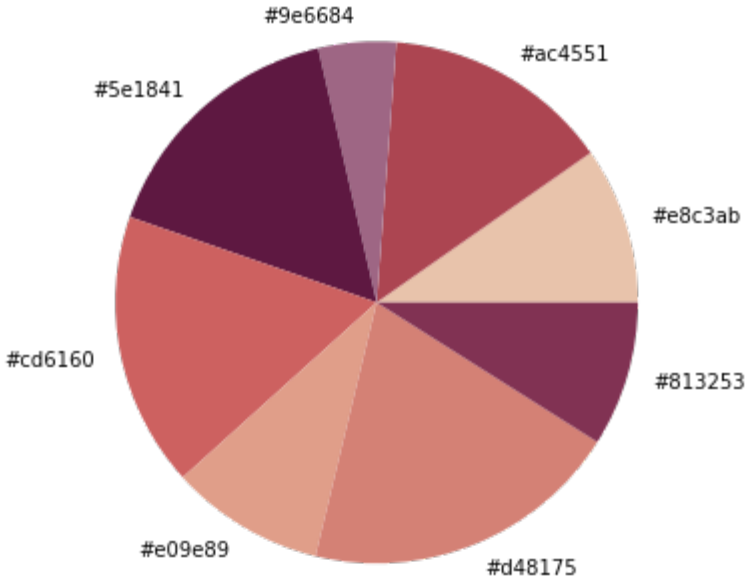
Next, we get the hex and rgb colors. As we divided each color by 255 before, we now multiply it by 255 again while finding the colors. If show_chart is True, we plot a pie chart with each pie chart portion defined using count.values(), labels as hex_colors and colors as ordered_colors. We finally return the rgb_colors which we'll use at a later stage.

In [102]:
```
def get_colors(image, number_of_colors, show_chart):

    modified_image = cv2.resize(image, (600, 400), interpolation = cv
2.INTER_AREA)
    modified_image = modified_image.reshape(modified_image.shape[0]*mo
dified_image.shape[1], 3)

    clf = KMeans(n_clusters = number_of_colors)
    labels = clf.fit_predict(modified_image)

    counts = Counter(labels)
    # sort to ensure correct color percentage
    counts = dict(sorted(counts.items()))

    center_colors = clf.cluster_centers_

    # We get ordered colors by iterating through the keys
    ordered_colors = [center_colors[i] for i in counts.keys()]
    hex_colors = [RGB2HEX(ordered_colors[i]) for i in counts.keys()]
    rgb_colors = [ordered_colors[i] for i in counts.keys()]

    if (show_chart):
        plt.figure(figsize = (8, 6))
        plt.pie(counts.values(), labels = hex_colors, colors = hex_col
ors)

    return rgb_colors
```

Let's just call this method as get_colors(get_image('sample_image.jpg'), 8, True) and our pie chart appears with top 8 colors of the image

In [103]:
```
get_colors(get_image("F:\Python\lena.jpg"), 8, True)
```

Out[103]:
```
[array([232.01310137, 195.82633162, 171.52293814]),
 array([172.54146067,  69.84099589,  81.8531336 ]),
 array([158.61066069, 102.52328791, 132.62670347]),
 array([94.08675516, 24.14817977, 65.19529613]),
 array([205.62786031,  97.74750985,  96.72957098]),
 array([224.11271067, 158.88671875, 137.53366397]),
 array([212.9942685 , 129.06768232, 117.49059148]),
 array([129.15623264,  50.5408872 ,  83.06853121])]
```




## Search Images using color

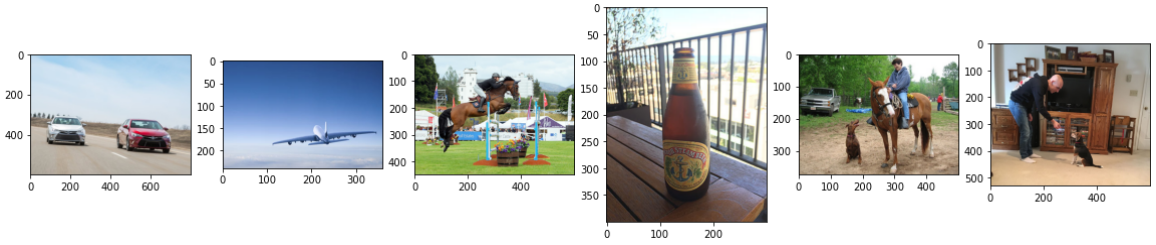
We'll now dive into the code of filtering a set of five images based on the color we'd like. For our use case, we'll supply the RGB values for the colors Green, Blue, Yellow and Red and let our system filter the images

In [123]:
```
IMAGE_DIRECTORY = 'F:\Python\object-detection-deep-learning\images'
COLORS = {
    'GREEN': [0, 128, 0],
    'BLUE': [0, 0, 128],
    'YELLOW': [255, 255, 0],
    'RED' : [128, 0, 0]
}

images = list()

for file in os.listdir(IMAGE_DIRECTORY):
    if not file.startswith('.'):
        images.append(get_image(os.path.join(IMAGE_DIRECTORY, file)))
```

## Show all images

We first show all the images in the folder using the below mentioned for loop. We split the area into subplots equal to the number of images. The method takes the arguments as number of rows = 1, number of columns = all images i.e. 6 in our case and the index.

In [120]:
```
plt.figure(figsize=(20, 10))
for i in range(len(images)):
    plt.subplot(1, len(images), i+1)
    plt.imshow(images[i])
```



## Match Images with color

We now define a method match_image_by_color to filter all images that match the selected color.

In [121]:
```
def match_image_by_color(image, color, threshold = 60, number_of_color
s = 10):

    image_colors = get_colors(image, number_of_colors, False)
    selected_color = rgb2lab(np.uint8(np.asarray([[color]])))

    select_image = False
    for i in range(number_of_colors):
        curr_color = rgb2lab(np.uint8(np.asarray([[image_colors
[i]]])))
        diff = deltaE_cie76(selected_color, curr_color)
        if (diff < threshold):
            select_image = True

    return select_image
```

## Explanation:

We first extract the image colors using our previously defined method get_colors in RGB format. We use the method rgb2lab to convert the selected color to a format we can compare. The for loop simply iterates over all the colors retrieved from the image.

For each color, the loop changes it to lab, finds the delta (basically difference) between the selected color and the color in iteration and if the delta is less than the threshold, the image is selected as matching with the color. We need to calculate the delta and compare it to the threshold because for each color there are many shades and we cannot always exactly match the selected color with the colors in the image.

```
By saying green, the user can mean light green, green or dark green. We ne
ed to scan through all possibilities.
```

If we extract say 6 colors from an image, even if one color matches with the selected color, we select that image. The threshold basically defines how different can the colors of the image and selected color be.

Let's consider the case where we are trying to find images with color Green. If the threshold is too high, we might start seeing blue images in our search. Similarly, on the other hand, if the threshold is too low, then green might not even match images that have dark green in them. It's all based on what is required in the situation at hand and we can modify the values accordingly. We need to carefully set the threshold value

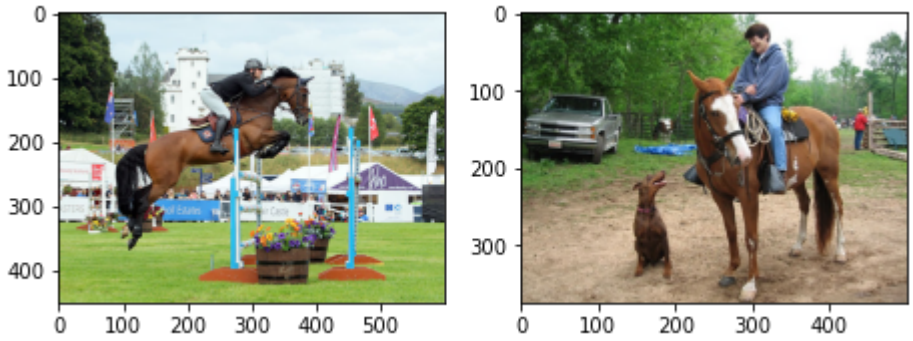
## Show selected images

We define a function show_selected_images that iterates over all images, calls the above function to filter them based on color and displays them on the screen using imshow

In [117]:
```
def show_selected_images(images, color, threshold, colors_to_match):
    index = 1

    for i in range(len(images)):
        selected = match_image_by_color(images[i],
                                        color,
                                        threshold,
                                        colors_to_match)
        if (selected):
            plt.subplot(1, 5, index)
            plt.imshow(images[i])
            index += 1
```

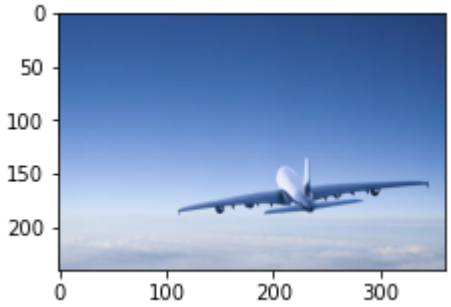
## Search for GREEN

In [118]:
```
plt.figure(figsize = (20, 10))
show_selected_images(images, COLORS['GREEN'], 60, 5)
```



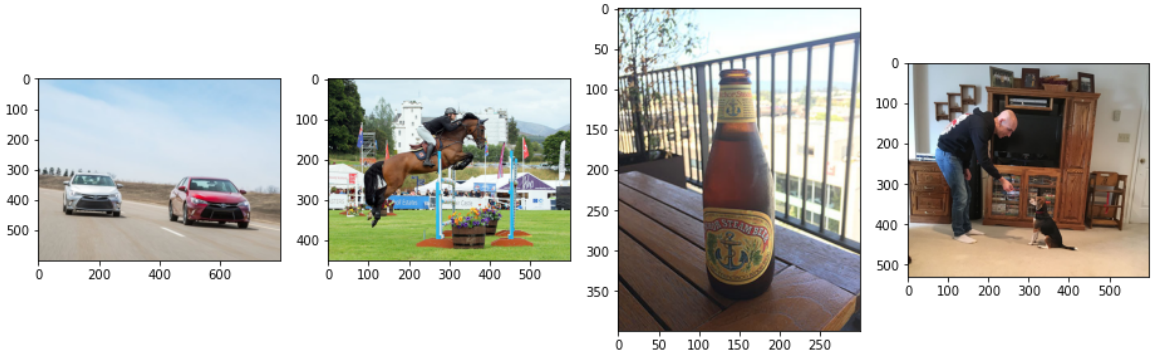
## Search for BLUE

In [122]:
```
plt.figure(figsize = (20, 10))
show_selected_images(images, COLORS['BLUE'], 60, 5)
```



## Search for RED

In [113]:
```
plt.figure(figsize = (20, 10))
show_selected_images(images, COLORS['RED'], 60, 5)
```



## Search for YELLOW

In [114]:
```
plt.figure(figsize = (20, 10))
show_selected_images(images, COLORS['YELLOW'], 60, 5)
```