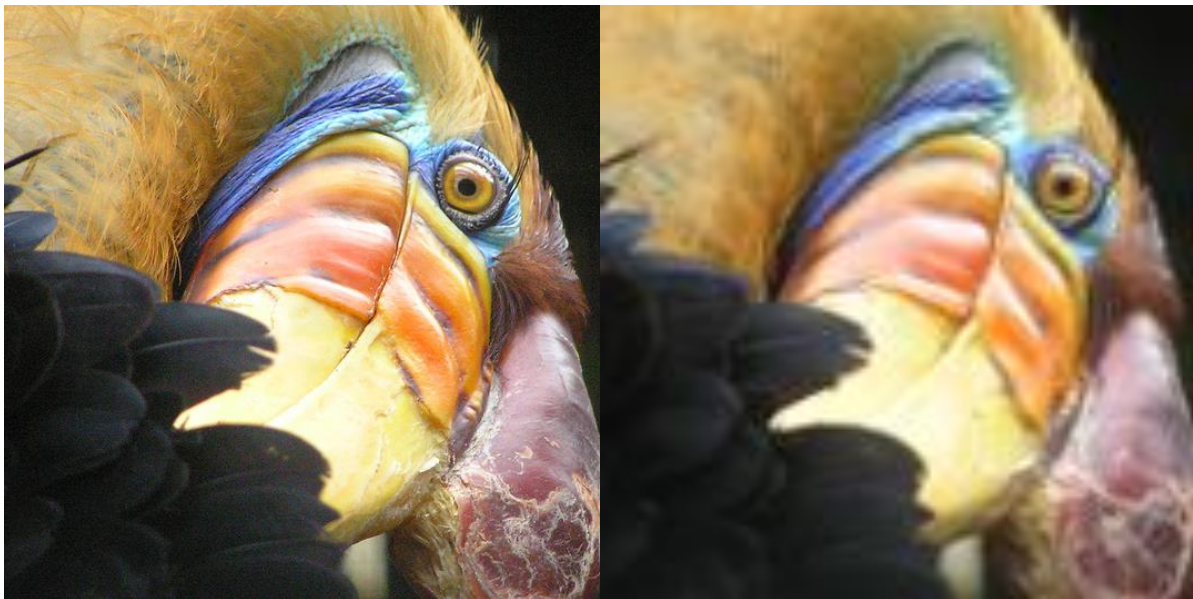


学号：201705130113	姓名：黄瑞哲	班级：计科 17.3
实验题目：K-Means		
实验学时：4	实验日期：2019.12.6	
<p>实验目的：</p> <ol style="list-style-type: none"> 1. 掌握 k 均值中的 KMeans 算法 2. 应用 KMeans 算法实现图片的像素压缩 		
<p>硬件环境：</p> <p>Intel Core i5-8300H @ 2.3GHz</p>		
<p>软件环境：</p> <p>Windows10 Pro 1903</p> <p>Python 3.7</p> <p>Visual Studio Code</p>		
<p>实验步骤与内容：</p> <ol style="list-style-type: none"> 1. 数据集 <p>数据集中包含两张 tiff 格式的图片，一张像素是 538x538 的，另一张是 128x128 的。每个像素点的颜色值是 24 位的，由 3 个 8 位的、范围在 [0, 255] 之间的数构成，分别表示红 R、绿 G、蓝 B。现在我们的任务是在图片中选取 16 个颜色来替代每个像素点中的颜色以实现图片压缩的效果。</p>  2. KMeans 算法 <p>KMeans 是实现聚类的一种算法，分为 3 步</p> <ol style="list-style-type: none"> 一、随机选取 K 个聚类中心 二、使用欧几里得距离计算出每个样本数据与哪一个聚类中心更相似 		

$$c^{(i)} := \arg \min_j \left\| x^{(i)} - \mu_j \right\|^2$$

三、使用该类中所有的样本数据重新计算聚类中心

$$\mu_j := \frac{\sum_i^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_i^m 1\{c^{(i)} = j\}}$$

当中心点变化幅度较小或满足一定的迭代次数后算法停止

3. 图片压缩

算法作用在一个三维空间中，这个空间是以 RGB 三个颜色值为基底的三维空间。那么每个点的 RGB 值便是这个空间中的一个点，将所有像素点的 RGB 值放入该三维空间中然后做 $k=16$ 的聚类，其中 16 个初始聚类中心从所有的颜色值中随机产生。完成聚类后可以知道每个点属于哪一个聚类中心，然后将这个点的 RGB 值替换为对应的聚类中心的 RGB 的值，最终再将图片还原即可。



可以看出颜色的对比度有了明显的减小，而且出现了粗糙的颜色轮廓。

结论分析与体会：

KMeans 算法的思路还是非常清晰的，使用样本数据更新聚类中心，然后根据新的聚类中心来分类，不断循环，最后能够得到一个较好的结果。而且算法实现起来也是非常简单，只需要按照算法一步步实现即可。但是自己手写的 KMeans 算法与现有的算法代码库中的 KMeans 在运行效率上还是有很大的区别：手写的算法迭代 50 多次可能需要将近半分钟的时间，而代码库如 sklearn 中实现的 KMeans 仅仅需要几秒就可以求得结果。

还有一点，python 的绘图库中要求 RGB 的颜色要么是[0, 1]的浮点数，要么是[0, 255]的整型，因此在导入样本数据的时候需要将读入的数据/255 将其转换为[0, 1]之间的浮点数然后再进行运算，否则得到的结果因为会是浮点数，渲染图片的时候将会是错误的。

附录：程序源代码

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image
# from sklearn.cluster import KMeans

class KMeans:
    def __init__(self, n_clusters, max_iter=200):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        m, n = np.shape(X)
        assert self.n_clusters <= m
        self.__random_cluster_centers__(X)
        self.labels_ = [0 for i in range(m)]
        for iter in range(self.max_iter):
            for i in range(m):
                k = 0
                for j in range(1, self.n_clusters):
                    if np.dot(X[i] - self.cluster_centers_[j], X[i] -
self.cluster_centers_[j]) < np.dot(X[i] - self.cluster_centers_[k], X[i] -
self.cluster_centers_[k]):
                        k = j
                self.labels_[i] = k
            for j in range(self.n_clusters):
                sum_x = np.zeros(n)
                sum_cnt = 0
                for i in range(m):
                    if self.labels_[i] == j:
```

```

        sum_x += X[i]
        sum_cnt += 1
    if sum_cnt > 0:
        self.cluster_centers_[j] = sum_x / sum_cnt

def __random_cluster_centers_(self, X):
    import random
    ind = random.sample([i for i in range(len(X))], self.n_clusters)
    self.cluster_centers_ = []
    for i in ind:
        self.cluster_centers_.append(X[i])
    self.cluster_centers_ = np.array(self.cluster_centers_)

if __name__ == "__main__":
    def image_reconstruction(cluster_centers_, labels_, w, h):
        channel = np.shape(cluster_centers_)[1]
        image = np.zeros((w, h, channel))
        ind = 0
        for i in range(w):
            for j in range(h):
                image[i][j] = cluster_centers_[labels_[ind]]
                ind += 1
        return image

    bird = matplotlib.image.imread("exp6/data/bird_small.tiff")
    bird = np.array(bird, dtype=np.uint)
    m, n, channel = np.shape(bird)

    kmeans = KMeans(n_clusters=16, max_iter=50)
    kmeans.fit(np.reshape(bird, (m * n, channel)))

    bird_compressed = image_reconstruction(kmeans.cluster_centers_,
kmeans.labels_, m, n)
    matplotlib.image.imsave("exp6/data/bird_compressed.tiff", bird_compressed)

    plt.figure(1)
    plt.title("Origin image")
    plt.axis("off")
    plt.imshow(bird)
    plt.figure(2)
    plt.title("Compressed image (16 colors)")
    plt.axis("off")
    plt.imshow(bird_compressed)
    plt.show()

```