Ürün Görsellerinin (RGB) Renk Kanalları ile K-Means Kümeleme

BLM 5110 Makine Öğrenmesi

Metin Uslu - 235B7014

Yapılan çalışmanın kodlarına ColorClustering{.ipynb, .py ve .html} dosyaları üzerinden erişebilirsiniz. Projenin tüm repository'sine aşağıdaki adres üzerinden erişim sağlayabilirsiniz. Aşağıda belirtilen açıklamaların çok daha fazlasına ColorClustering notebook dosyası üzerinden erişebilirsiniz.

GitHub: https://github.com/metinuslu/blm5110_color_clustering

1. Özet: Yaptığınız çalışmayı, elde ettiğiniz sonuçları özet olarak veriniz.

Bu çalışmada Roboflow platformundan alınan farklı çözünürlükte ve doğal ortamlarda çekilmiş, kırmızı, yeşil, mavi, beyaz ve gri renk çeşitliliğinde ürün görselleri R,G,B renk kanalları üzerinden kümelenerek renkler bazında kümelenmesi amaçlanmıştır. Öncesinde resimlerin array haline getirilmesi, ardından renk kanallarının orijinal ve normalize histogramlarının elde edilmesi ve görselleştirilmesi sağlanmıştır. Devamında kendimiz implemente ettiği K-Means algoritması kullanılarak her resim için R, G ve B normalize histogramları üzerinden kümele çalışması yapılmıştır. Küme merkezleri rastgele belirlenmesi ile initialize edilen algoritmanın %49 bir doğruluk performansı elde edilmiştir. Yine 10 farklı deneme ile elde edilen performans aralığı %32 ile %54 arasında olduğu gözlemlenmiştir.

2. Giriş: Ödev konusunu tanıtan 1 paragraflık bir giriş yapınız. Bu çalışmanın nerelerde kullanılabileceğinden bahsediniz.

Bu çalışma da Ürün görsellerinin R, G, B renk kanalları/özellikleri kullanılarak benzer renkte bulunan ürün görsellerinin kümelenmesi için yapılan bir çalışmasıdır. Bu çalışma aşağıdaki amaçlar için kullanılabilir;

1. Ürün görselleri üzerinde Renk özelliklerinin tespit edilmesinde kullanabilir.
2. Yine aynı zamanda ürünlerin renklerini kullanarak birbirlerine aynı ya da yakın renklere sahip ürünlerin tespit ederek bunlar gruplanabilir.
3. Bu aynı zamanda sonsuz sayıda bulunan ürün renk uzayını azaltmaya da (reduce) de katkı sağlayabilir. Mesela bazı renkleri (Bknz: Siyah, Gri, Kahverengi)koyu tonlar olarak, bazı renkleri (Bknz: Beyaz, Açık gri, Bej, Açık Sarı) açık tonlar, bazı renkleri(bordo, Koyu Kırmızı, Kırmızı, Turuncu) kızıl tonlar olarak indirgeyebilirler. Bu da Modelleme açısından bakıldığında "High Cardinality" problemine de çözüm sağlayacaktır.

3. Sistem Tasarımı: Sisteminizin işlem adımlarını kısaca anlatınız.

I. Resimlerin formatını hazır kütüphaneler kullanarak çözerek görüntüye ait matrisi elde ediniz. Bir görüntü piksellerin (R,G,B) bileşenlerinden oluştuğu bir matristir.

create_img_to_arr fonksiyonu kullanılarak her bir resim dosyası array haline getirilmiştir. Burada resim dosyaları okunurken OpenCv library'si kullanılmıştır. OpenCv ile resimler varsayılan olarak

BGR olarak okunmaktadır. Buna dikkat edilerek resim nesneleri BGR to RGB çevrilmiştir. Fonksiyon hem RGB hem de BGR resim arrayleri return etmektedir.

Task 1: Create Image to Array

Description: Resimlerin formatını hazır kütüphaneler kullanarak çözerek görüntüye ait matrisi elde ediniz.Bir görüntü piksellerin (R,G,B) bileşenlerinden oluştuğu bir matristir.

[7]:
```
def create_img_to_arr(img_path_list):
    img_rgb_arr = []
    img_bgr_arr = []

    for file in img_path_list:
        img_bgr = cv2.imread(file)
        img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

        red, green, blue = cv2.split(img_rgb)
        # red, green, blue

        height, width, channels = img_rgb.shape
        # height, width, channels

        img_rgb_arr.append(img_rgb)
        img_bgr_arr.append(img_bgr)

    return img_rgb_arr, img_bgr_arr
```

[8]:
```
img_rgb_arr, img_bgr_arr = create_img_to_arr(img_path_list=file_names)
```

II. Resimleri oluşturan piksellerin (R,G,B) değerlerine göre her resmin renk histogramını (Her resimde R, G, B bileşenleri için ayrı ayrı olmak üzere toplam 3 histogram) elde ediniz. Histogram eldesi için de hazır kütüphane kullanabilirsiniz. Bir pikselin R,G,B bileşenlerinin değeri 0-255 arası değiştiği için her histogram dizisi 256 elemanlı olmalıdır.

img_to_hist_arr fonksiyonu kullanılarak her bir resim nesnesinin R, G ve B histogramları OpenCv library'si kullanılarak elde edilmiştir. Fonksiyon R, G ve B için ayrı ayrı histogram arrayleri return etmektedir.

Task 2: Create Color Histogram for each R,G,B Components and Chart

Description: Resimleri oluşturan piksellerin (R,G,B) değerlerine göre her resmin renk histogramını (Her resimde R, G, B bileşenleri için ayrı ayrı olmak üzere toplam 3 histogram) elde ediniz. Histogram eldesi için de hazır kütüphane kullanabilirsiniz. Bir pikselin R,G,B bileşenlerinin değeri 0-255 arası değişti için her histogram dizisi 256 elemanlı olmalıdı.

[9]:
```
def img_to_hist_arr(img_arr, is_rgb=True):
    hist_r_arr = []
    hist_g_arr = []
    hist_b_arr = []

    for i in range(0, len(img_arr)):
        img = img_arr[i]
        height, width, channels = img.shape

        if is_rgb:
            hist_r = cv2.calcHist([img], [0], None, [256], [0, 256]).ravel()
            hist_g = cv2.calcHist([img], [1], None, [256], [0, 256]).ravel()
            hist_b = cv2.calcHist([img], [2], None, [256], [0, 256]).ravel()
        else:
            hist_r = cv2.calcHist([img], [2], None, [256], [0, 256]).ravel()
            hist_g = cv2.calcHist([img], [1], None, [256], [0, 256]).ravel()
            hist_b = cv2.calcHist([img], [0], None, [256], [0, 256]).ravel()

        hist_r_arr.append(hist_r)
        hist_g_arr.append(hist_g)
        hist_b_arr.append(hist_b)

    return hist_r_arr, hist_g_arr, hist_b_arr
```

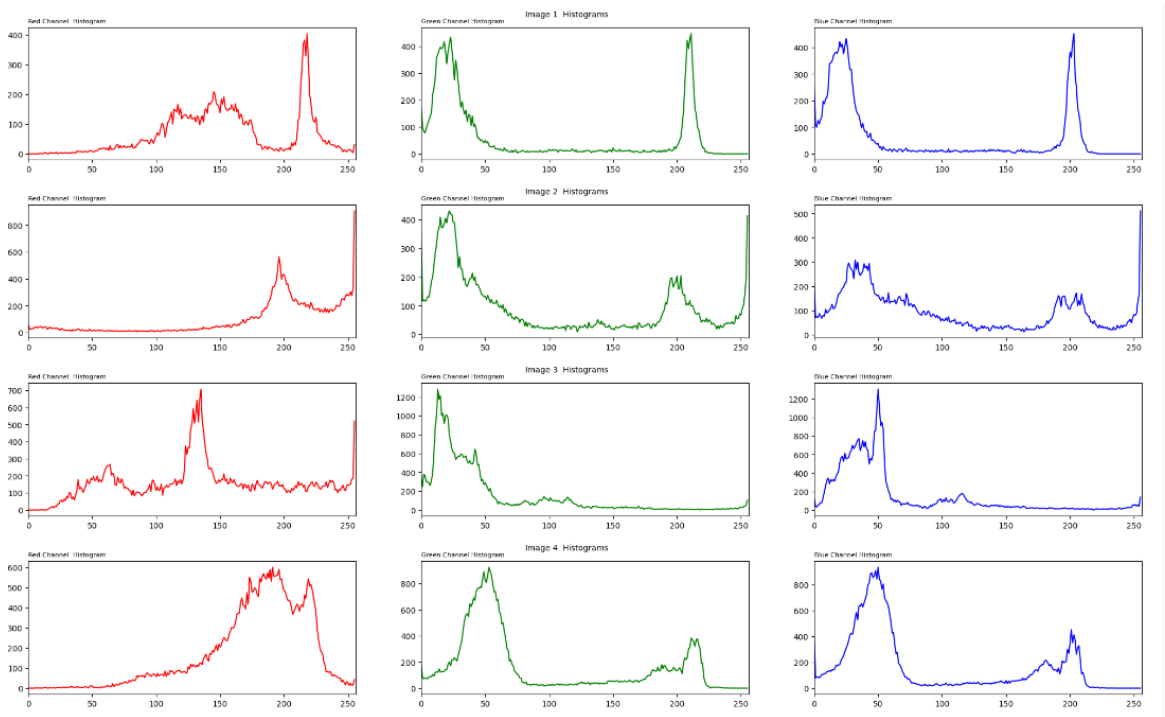
[10]:
```
hist_r_arr, hist_g_arr, hist_b_arr = img_to_hist_arr(img_arr=img_rgb_arr, is_rgb=True)
```

R, G ve B Histogramlarının görselleştirilmesi için **plot_histogram** fonksiyonu yazılmıştır. Örnek birkaç resim için çıktıları aşağıda paylaşıyorum. ColorClustering Notebook (.ipynb, .html) üzerinden tamamına erişebilirsiniz.

Plot Histograms RGB Components

[12]:
```
def plot_histogram(hist_r_arr, hist_g_arr, hist_b_arr, normalized_msg=""):

    for i in range(0, len(hist_r_arr)):
        hist_r = hist_r_arr[i]
        hist_g = hist_g_arr[i]
        hist_b = hist_b_arr[i]

        plt.figure(figsize=(25, 3))
        plt.subplot(131)
        plt.plot(hist_r, color='red')
        plt.title(f'Red Channel {normalized_msg} Histogram', size=8, loc='left')
        plt.xlim([0, 256])

        plt.subplot(132)
        plt.plot(hist_g, color='green')
        plt.title(f'Green Channel {normalized_msg}Histogram', size=8, loc='left')
        plt.xlim([0, 256])

        plt.subplot(133)
        plt.plot(hist_b, color='blue')
        plt.title(f'Blue Channel {normalized_msg}Histogram',size=8, loc='left')
        plt.xlim([0, 256])

        plt.suptitle(f'Image {i+1} {normalized_msg} Histograms', size=10)
        plt.show()
```

[13]:
```
plot_histogram(hist_r_arr, hist_g_arr, hist_b_arr)
```



III. Her resimde, her renk bileşeni için histogram dizisindeki sonuçları resimdeki toplam piksel sayısına bölerek dizi elemanlarının değerini [0-1] aralığına normalize ediniz.

img_to_norm_hist_arr fonksiyonu kullanılarak R, G ve B histogramları 0 - 255 aralığından 0 - 1 aralığına normalize edilmiştir.

Task 3: Normalizing Color Histogram for each R,G,B Components

Description: Her resimde, her renk bileşeni için histogram dizisindeki sonuçları resimdeki toplam piksel sayısına bölerek dizi elemanlarının değerini [0-1] aralığına normalize ediniz

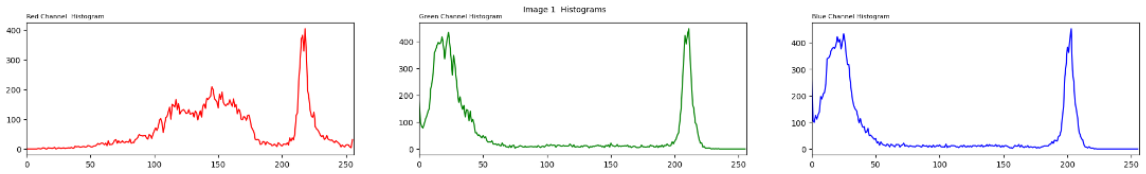
[14]:
```python
def img_to_norm_hist_arr(img_arr, is_rgb=True):
    norm_hist_r_arr = []
    norm_hist_g_arr = []
    norm_hist_b_arr = []

    for i in range(0, len(img_arr)):
        img = img_arr[i]
        height, width, channels = img.shape

        if is_rgb:
            hist_r = cv2.calcHist([img], [0], None, [256], [0, 256]).ravel()
            hist_g = cv2.calcHist([img], [1], None, [256], [0, 256]).ravel()
            hist_b = cv2.calcHist([img], [2], None, [256], [0, 256]).ravel()
        else:
            hist_r = cv2.calcHist([img], [2], None, [256], [0, 256]).ravel()
            hist_g = cv2.calcHist([img], [1], None, [256], [0, 256]).ravel()
            hist_b = cv2.calcHist([img], [0], None, [256], [0, 256]).ravel()

        total_pixel = height * width
        norm_hist_r_arr.append(hist_r / total_pixel)
        norm_hist_g_arr.append(hist_g / total_pixel)
        norm_hist_b_arr.append(hist_b / total_pixel)

    return norm_hist_r_arr, norm_hist_g_arr, norm_hist_b_arr
```

[15]:
```python
norm_hist_r_arr, norm_hist_g_arr, norm_hist_b_arr = img_to_norm_hist_arr(img_arr=img_rgb_arr, is_rgb=True)
```

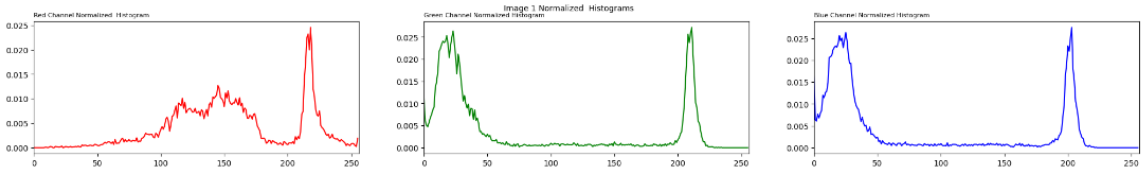
R, G ve B Histogram ve Normalize Histogram çıktılarını bir örnek resim üzerinden paylaşıyorum.

Histogram vs Normalized Histogram Chart

[16]:
```python
plot_histogram(hist_r_arr[0:1], hist_g_arr[0:1], hist_b_arr[0:1], normalized_msg="")
```



[17]:
```python
plot_histogram(norm_hist_r_arr[0:1], norm_hist_g_arr[0:1], norm_hist_b_arr[0:1], normalized_msg="Normalized ")
```



IV. Resimlerin histogramlarını benzerliklerine göre k=5 değeri için(5 renk sınıfı olduğu için) k-means yöntemiyle kümeleyiniz. K-means'de başlangıç adımında rasgele seçeceğiniz k resmin histogramını k cluster'ın başlangıç merkezi olarak kullanınız.

KMeans class içerisine **fit**, **predict**, uzaklık hesaplama yöntemlerinin (Euclidean ve Manhattan) tercihi için **calculate_distance** ve centroid değerlerinin rastgele belirlenmesi için **initialize_centroids** fonksiyonları yazılmıştır.

Task 4: K Means Algorithms for k=5

Description: Resimlerin histogramlarını benzerliklerine göre k=5 değeri için(5 renk sınıfı olduğu için) k-means yöntemiyle kümeleyiniz. K-means'de başlangıç adımında rasgele seçeceğiniz k resmin histogramını k cluster'ın başlangıç merkezi olarak kullanınız.

[18]:
```
# norm_hist_r_arr, norm_hist_g_arr, norm_hist_b_arr
histograms = np.array([np.concatenate((r, g, b)) for r, g, b in zip(norm_hist_r_arr, norm_hist_g_arr, norm_hist_b_arr)])
```

KMeans Algorithms

[138]:
```
class KMeans:
    def __init__(self, n_clusters=5, max_iters=100, distance_metric='euclidean'):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.distance_metric = distance_metric
        self.centroids = None
        self.clusters = None


    def initialize_centroids(self, X):
        centroids_indices = np.random.choice(X.shape[0], self.n_clusters, replace=False)
        centroids = X[centroids_indices]
        return centroids

    def calculate_distance(self, x, y):
        # Uzaklık hesaplama (Öklid veya Manhattan)
        if self.distance_metric == 'euclidean':
            return np.dot(x - y, x - y)
        elif self.distance_metric == 'manhattan':
            return np.sum(np.abs(x - y))
        else:
            raise ValueError("Geçersiz uzaklık metriği. 'euclidean' veya 'manhattan' seçiniz.")

    def fit(self, X):
        self.centroids = self.initialize_centroids(X)

        for _ in range(self.max_iters):
            # Atama aşaması
            self.clusters = np.array([np.argmin([self.calculate_distance(x, y) for y in self.centroids]) for x in X])

            # Güncelleme aşaması
            new_centroids = np.array([X[self.clusters == k].mean(axis=0) for k in range(self.n_clusters)])

            # Eğer merkezler değişmiyorsa döngüyü bitir
            if np.all(self.centroids == new_centroids):
                print("Cluster Converged")
                break

            self.centroids = new_centroids

    def predict(self, X):
        return np.array([np.argmin([self.calculate_distance(x, y) for y in self.centroids]) for x in X])
```

[171]:
```
# # KMeans sınıfını oluştur
kmeans = KMeans(n_clusters=5, max_iters=150, distance_metric='euclidean')

# # Veriyi kümele
kmeans.fit(histograms)

# # Kümeleme sonuçları
clusters = kmeans.predict(histograms)
print(clusters)
```

```
Cluster Converged
[1 0 1 0 0 1 1 1 1 1 0 0 1 2 1 1 1 1 4 1 0 0 2 0 0 0 4 0 4 0 0 0 0 0 0 0
 0 0 0 0 0 4 0 2 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 2 0 4 3 4 0 4 4 0 4 0 4 4 0 0
 4 0 4 0 4 4 0 2 0 1 0 0 1 3 0 0 0 3 0 0 0 0 0 0 3 0 2]
```

4. Deneysel Sonuçlar: Sistem başarısını değerlendirmek için aşağıdaki işlemleri yapınız.

Model performasının ölçülmesi için her bir küme için {0..4} her kümede baskın olan Ürün Rengi o kümenin doğru sonucu olarak kabul edilip, elimizde bulunan bildiğimiz (ground truth) gerçek Ürün Renkleri ile Kümelerdeki Ürün Renklerinin karşılaştırılması ile Accuracy hesaplanmıştır. Yine öncesinde kümelerdeki Cluster x Count dağılımları ve ClusterId ve Color x Count dağılımları ColorClustering notebook dosyasında paylaşılmıştır.

a. Küme merkezlerini rasgele belirleyerek her k değeri için kümeleme işlemini 10 defa tekrarlayınız. Her kümeleme sonunda, kümelerde doğru cluster'da olan resim yüzdesini hesaplayınız.

Burada Cluster performansının ölçülmesi için **cluster_evaluate** isimli bir fonksiyon yazılarak kümeleme işleminin performansı ölçülmüştür. 10 tekrar için ayrı ayrı Confusion Matrix sonuçlarına ColorClustering notebook dosyası üzerinden erişebilirsiniz.

Cluster Performance Evaluation

[354]:
```
def cluster_evaluate(df, label_1="", label_2="", labels=[], visualise=False):
    accuracy = accuracy_score(df[label_1], df[label_2])
    print(f"Accuracy: {accuracy}")
    if visualise:
        conf_matrix = confusion_matrix(df[label_1], df[label_2], labels=labels)
        plt.figure(figsize=(8, 6))
        sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['red', 'blue', 'green', 'white', 'gray'], yticklabels=['red', 'blue', 'green', 'white', 'gray'])
        plt.xlabel('Tahmin Edilen Renk')
        plt.ylabel('Gerçek Renk')
        plt.title('Confusion Matrix')
        plt.show()
```

a. Küme merkezlerini rasgele belirleyerek her k değeri için kümeleme işlemini 10 defa tekrarlayınız. Her kümeleme sonunda, kümelerde doğru cluster'da olan resim yüzdesini hesaplayınız.

[352]:
```
# df_cluster_result = pd.DataFrame()
df_cluster_result = df_data[['FileName', 'FilePath','Color']]
for i in range(0, 10):
    kmeans = KMeans(n_clusters=5, max_iters=150)
    kmeans.fit(histograms)
    clusters = kmeans.predict(histograms)
    df_cluster_result[f'ClusterId_{i+1}'] = clusters

df_cluster_result
```

```
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
Cluster Converged
```

[352]:

	FileName	FilePath	Color	ClusterId_1	ClusterId_2	ClusterId_3	ClusterId_4	ClusterId_5	ClusterId_6	ClusterId_7	ClusterId_8	ClusterId_9	ClusterId_10
0	original1.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original1.jpg	red	1	4	4	0	4	0	0	2	4	0
1	original10.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original10.jpg	red	1	4	4	0	4	0	0	2	4	0
2	original11.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original11.jpg	red	1	4	4	0	4	0	0	2	4	0
3	original12.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original12.jpg	red	1	4	2	0	4	4	0	2	4	0
4	original13.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original13.jpg	red	1	4	4	0	4	4	0	2	4	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	original5.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original5.jpg	white	1	4	2	0	4	4	0	2	4	0
96	original6.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original6.jpg	white	1	4	2	0	4	4	0	2	4	0
97	original7.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original7.jpg	white	3	3	1	1	2	3	3	3	1	3
98	original8.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original8.jpg	white	1	4	2	0	4	4	0	2	4	0
99	original9.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original9.jpg	white	0	0	3	2	3	1	2	0	2	4

100 rows × 13 columns

[357]:
```
for i in range(0, 10):
    df_multi_cluster = df_cluster_result.groupby(by=[f'ClusterId_{i+1}'], as_index=False)['Color'].apply(lambda x: x.value_counts().idxmax())
    df_multi_cluster.rename(columns={'Color':f'ClusterColor_{i+1}'}, inplace=True)
    df_cluster_result[f'ClusterColor_{i+1}'] = df_cluster_result[f'ClusterId_{i+1}'].map(df_multi_cluster.to_dict()[f'ClusterColor_{i+1}'])
    print(f"Cluster_{i+1}")
    cluster_evaluate(df=df_cluster_result, label_1="Color", label_2=f"ClusterColor_{i+1}", labels=['red', 'blue', 'green', 'white', 'gray'], visualise=False)
    print("* " * 10)
```

```
Cluster_1
Accuracy: 0.54
* * * * * * * * * *
Cluster_2
Accuracy: 0.38
* * * * * * * * * *
Cluster_3
Accuracy: 0.5
* * * * * * * * * *
Cluster_4
Accuracy: 0.37
* * * * * * * * * *
Cluster_5
Accuracy: 0.33
* * * * * * * * * *
Cluster_6
Accuracy: 0.49
* * * * * * * * * *
Cluster_7
Accuracy: 0.38
* * * * * * * * * *
Cluster_8
Accuracy: 0.32
* * * * * * * * * *
Cluster_9
Accuracy: 0.38
* * * * * * * * * *
Cluster_10
Accuracy: 0.33
* * * * * * * * * *
```

b. Hazırlayacağınız karışıklık matrisi (confusion matrix) üzerinde sonucu gösteriniz.

Burada Modeli oluşturduğumuz ve sadece 1 tekrar elde ettiğimiz Model Performansı ve Confusion Matrix yer almaktadır. 10 tekrar ile elde ettiğimiz Confusion Matrix'ler notebook içerisinden erişebilirsiniz.

b. Hazırlayacağınız karışıklık matrisi (confusion matrix) üzerinde sonucu gösteriniz

[245]:
```
df_cluster = df_data.groupby(by=['ClusterId'], as_index=False)['Color'].apply(lambda x: x.value_counts().idxmax())
df_cluster.rename(columns={'Color':'ClusterColor'}, inplace=True)
df_cluster
```

[245]:

	ClusterId	ClusterColor
0	0	blue
1	1	red
2	2	green
3	3	white
4	4	gray

[254]:
```
df_evaluate = pd.merge(df_data, df_cluster, on='ClusterId', how='left')
df_evaluate
```

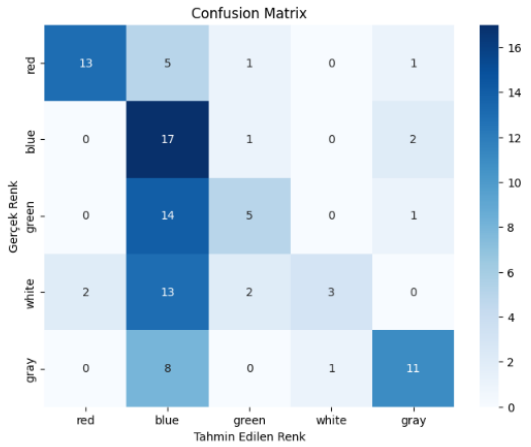
[254]:

	FileName	FilePath	Color	ClusterId	ClusterColor
0	original1.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original1.jpg	red	1	red
1	original10.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original10.jpg	red	0	blue
2	original11.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original11.jpg	red	1	red
3	original12.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original12.jpg	red	0	blue
4	original13.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\red\original13.jpg	red	0	blue
...	...	...	...	...	...
95	original5.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original5.jpg	white	0	blue
96	original6.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original6.jpg	white	0	blue
97	original7.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original7.jpg	white	3	white
98	original8.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original8.jpg	white	0	blue
99	original9.jpg	C:\Users\metin\Desktop\YTU\2-BLM5110-ML\Homeworks\blm5110_hw2-clustering\data\roboflow\white\original9.jpg	white	2	green

100 rows × 5 columns

```
[338]: cluster_evaluate(df=df_evaluate, label_1="Color", label_2="ClusterColor", labels=['red', 'blue', 'green', 'white', 'gray'], visualise=True)
```

```
Accuracy: 0.49
```



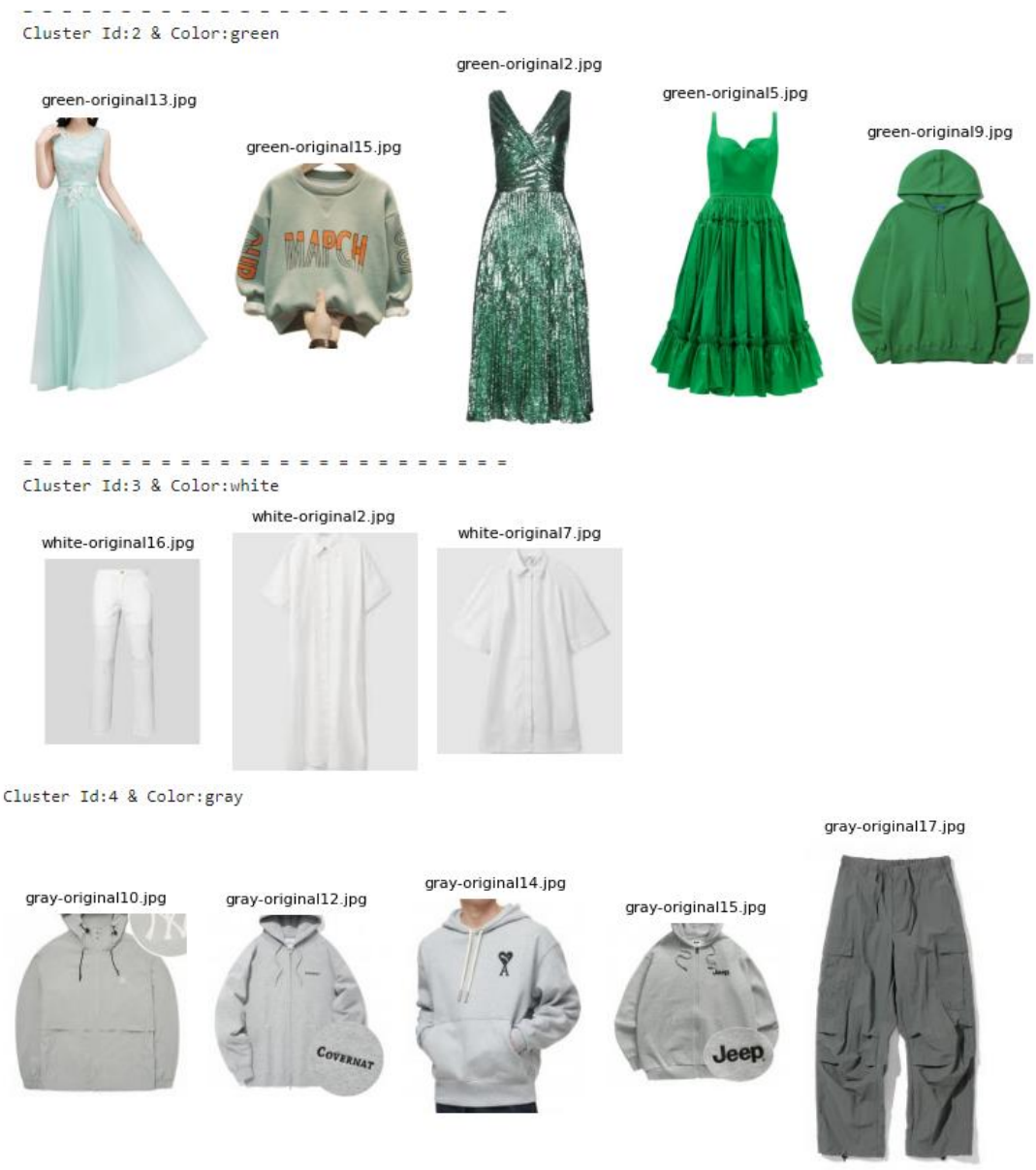
c. Her sınıf için doğru cluster'a yerleşmiş 5 örnek ve yanlış cluster'da bulunan 1 örnek resmi veriniz.

Doğru Cluster Edilmiş 5 Örnek

```
[307]: # Doğru Yerleşmiş Örnekler
cluster_color_dict = df_evaluate[['ClusterId', 'ClusterColor']].drop_duplicates().sort_values(by=['ClusterId'], ascending=True).reset_index(drop=True).to_dict()['ClusterColor']
for i in range(0, 5):
    color_name = cluster_color_dict[i]
    print("Cluster Id:{} & Color:{}".format(i, color_name))
    image_paths = df_evaluate[(df_evaluate['ClusterId']==i) & (df_evaluate['Color']==df_evaluate['ClusterColor'])]['FilePath'].head(5).to_list()

    X = 10  # Her satırdaki resim sayısı
    display_images(image_paths, X)
    print("= " * 25)
```

```
Cluster Id:0 & Color:blue
```


```
= = = = = = = = = = = = = = = = = = = = = = = = =
Cluster Id:1 & Color:red
```



Yanlış Cluster Edilmiş 5 Örnek

```
[309]: # Yanlış Yerleşmiş Örnekler
cluster_color_dict = df_evaluate[['ClusterId', 'ClusterColor']].drop_duplicates().sort_values(by=['ClusterId'], ascending=True).reset_index(drop=True).to_dict()['ClusterColor']
for i in range(0, 5):
    color_name = cluster_color_dict[i]
    print("Cluster Id:{} & Color:{}".format(i, color_name))
    image_paths = df_evaluate[(df_evaluate['ClusterId']==i) & (df_evaluate['Color']!=df_evaluate['ClusterColor'])]['FilePath'].head(5).to_list()

    X = 10  # Her satırdaki resim sayısı
    display_images(image_paths, X)
    print("=" * 25)
```

```
Cluster Id:0 & Color:blue
```

red-original10.jpg

red-original12.jpg

red-original13.jpg

red-original19.jpg

red-original2.jpg

```
= = = = = = = = = = = = = = = = = = = = = = = = =
Cluster Id:1 & Color:red
```

white-original12.jpg

white-original15.jpg

5. **Sonuç: Sizce kümeleme işlemi başarılı oldu mu? Yanlış kümelerde olan resimler sizce neden yanlış kümelerde yer aldı? Sistemin genel başarısını yorumlayınız. Başarısızlık sebebi olduğunu düşündüğünüz problemlerin giderilmesi için varsa önerilerinizi belirtiniz.**

Bu çalışmada Kümeleme performansı oldukça düşük bir başarıma sahip olduğunu tekrarlar ile görülmüştür. Hem tekli hem de 10 tekrarlı k adet farklı merkez noktası ile kümeleme algoritmasını (Accuracy) performansını ölçtüğümüz de performans maksimum %54'e ulaştığı gözlemlenmiştir. Bu da modelin performansının iyi olmadığını göstermektedir. Başarısızlığın sebepleri olarak;

1. Farklı Çözünürlük
   - ❖ Çözünürlükler ön işleme ile belirli bir x,y çözünürlüğüne çekilebilir.
2. Doğal Ortam
   - ❖ Tüm resimlerde doğal ortamların farklı olması, noiselere sebebiyet vermektedir. Bu sebeple eğer sağlanabiliyor ise aynı şartlar altında doğal ortamlarda resimlerin alınması.
3. Alt ve Üst kıyafetlerin birlikte bulunması sebebiyle birden fazla renk içerebiliyor olması

- ❖ Modelleme öncesi ön işleme adımı olarak semantic segmentasyon ürünler birbirlerinden ayrıştırılabilir.

4. Bazı ürünler için mankenli görsellerinin (yüz, boyun, el, vs.) bulunması resim üzerinde gürültüye(noise) sebep vermektedir
   - ❖ Face detection ile mankenli görsellerin exclude edilmesi ya da detect edilen bölgenin croplanması oradan alınması yapılabilir.
5. Işık ve Diğer Etkenler
   - ❖ Mümkünse tüm etkenlerin stabilizasyonunu sağlanması ya da tüm resimlerde aynı ön işleme teknikleri kullanılarak resimler de aynı standartizenin oluşmasının (variance indirgenmesi) sağlanması sağlanması
6. Daha Fazla Veri yada Veri Çeşitliğinin Artırılması
   - ❖ Eğer mümkünse daha fazla veri ile modellemenin tekrar edilmesi,
   - ❖ Eğer mümkün değil ise Data Augmentation ile her bir renk için ürün görsel çeşitliliğinin artırılması