سروش پسندیده – ۹۹۱۲۷۶۲۱۰۹، سروش فتحی – ۹۹۱۲۷۶۲۷۳۴

در این فاز از ما خواسته شده که مهم ترین نواحی را تشخیص دهیم و اهمیت آنها را مشخص کنیم.

باید نواحی ای را پیدا کنیم که بیشترین اطلاعات را درباره طبقه بندی تصویر به ما می دهند. برای طبقه بندی تصاویر پس از استخراج ویژگی از ناحیه ها، میانگین بردار ویژگی ناحیه ها را بدست میاوریم و از بردار حاصل به عنوان بردار ویژگی تصاویر استفاده میکنیم.

با استفاده از تابع زیر (calculate_mean_imgs_clstrs_features) میانگین بردار ویژگی کلاستر های هر عکس را میتوانیم بدست آوریم. Histograms در حقیقت لیستی حاوی بردار ویژگی تصاویر است.

```
"""
    Extract a feature vector for each image
    by calculating the mean of all its culsters
"""
def calculate_mean_imgs_clstrs_features():
    histograms = []
    for ftr_clstrs_img in all_images_clusters_features:
        histograms.append(np.mean(ftr_clstrs_img, axis=0))
    return histograms
✓ 0.0s
```

از این تابع برای تست کردن درستی میانگین گیری استفاده می‌شد. آنرا توسعه دادیم و به تابع پایین تبدیل کردیم که آنرا در پایین توضیح داده ایم.

با استفاده از تابع زیر (calculate_mean_imgs_clstrs_except_index) میتوانیم میانگین بردار ویژگی کلاستر های یک عکس را (img_num) ، بجز یک کلاستر خاص(_index) در آن عکس، بدست آوریم.

کاربرد این تابع آنجاست که میخواهیم به ازای هر عکس، با حذف هر کدام از کلاستر های آن، تغییرات احتمال لیبل آن عکس را بررسی کنیم که ببینیم بعد از حذف هر کدام از کلاستر های عکس، اطمینان لیبل آن چگونه تغییر میکند تا بر اساس آن ترتیب اهمیت کلاستر های عکس را مشخص کنیم.

```
"""
    Extract a feature vector for each image
    by calculating the mean of its culsters excpet one of them
"""
def calculate_mean_imgs_clstrs_except_index(index_, img_num, remove):
    histograms = []
    for i, ftr_clstrs_img in enumerate(all_images_clusters_features_bak):
        if i == img_num and remove:
            histograms.append(np.mean(np.concatenate((ftr_clstrs_img[:index_],
                                                      ftr_clstrs_img[index_+1:])),
                                                      axis=0))
        else:
            histograms.append(np.mean(ftr_clstrs_img, axis=0))

    return histograms
```
✓ 0.0s

اینجا histograms لیستی شامل میانگین بردار ویژگی کلاستر های عکس هاست که یک کلاستر از یک عکس خاص آن حذف شده و در بردار ویژگی آن عکس اعمال نشده است.

در این قسمت کد برای لیست های کاربردی یک instance ایجاد کرده ایم و تغییرات را روی آنها اعمال میکنیم تا در هنگام تست کردن لیست های اصلی تغییر نکنند و نیاز به ران کردن مجدد کد از اول نباشد.

```
def apply_back_ups():
    all_images_clusters_features_bak = all_images_clusters_features
    k_number_bak = k_number
    available_clusters = [ [ i for i in range(k_number)] for _ in range(n_images)]
```
✓ 0.0s

سپس با استفاده از تابع cal_classify_results میتوانیم به ازای حذف هر یک از کلاستر های یک عکسِ img_num لیست histograms را بدست آوریم. سپس این histograms را به تابع classify می‌دهیم تا classification را انجام دهد. خروجی این تابع (rslt) شامل اطمینان لیبل عکس هاست که در توابع بعدی از آن استفاده میکنیم. (به تعداد k_number_bak که همان تعداد کلاستر های k-means است این کار را میکنیم.) (تابع classify را در انتهای داکیونت توضیح می‌دهیم)

```python
# classifies images while a cluster of an image has been removed.
# This happens for all clusters of that image.
def cal_classify_results(img_num, remove):
    classify_results = []
    for clstr_num in range(k_number_bak):
        histograms = calculate_mean_imgs_clstrs_except_index(index_= clstr_num,
                                                             img_num = img_num,
                                                             remove=remove)
        rslt = classify(histograms, random_labels)
        classify_results.append(rslt)
    return classify_results
```

✓ 0.0s

در نهایت classify_results، یعنی خروجی همین تابع، به ازای هر عکس شامل چندین پارامتر است که یکی از آنها همان اطمینان لیبل عکس مورد نظر (img_num) بعد از حذف هر یک از کلاستر های آن است.

سپس با استفاده از تابع cal_probs (calculate probabilities) به ازای هر عکس اطمینان لیبل بعد از حذف هر یک از کلاستر های آن که در classify_results ذخیره شده بود را جمع آوری کرده و در یک لیست کنار هم قرار میدهیم تا بتوان بعدا اطمینان لیبل هر عکس در نبود هر کلاستر های آنرا با هم مقایسه کرد و کلاستر ها را رتبه بندی کرد.

```python
def cal_probs(classify_results):
    probs = [ [] for _ in range(len(classify_results[0][3])) ]
    for result in classify_results:
        for i, imgs_probs in enumerate(result[3]):
            probs[i].append(imgs_probs)

    return(probs)
```

✓ 0.0s

هر کدام از لیست های داخل لیست probs شامل اطمینان لیبل ها برای یک عکس در نبود هر یک از کلاستر های آن است.

از تابع زیر (show_probs) برای تست کردن درستی حذف هر کلاستر و دیدن تاثیر آن در اطمینان لیبل عکس استفاده می‌شد.

```python
def show_probs(probs, classify_results):
    for i, prob in enumerate(probs):
        print(np.array(prob))
        print(f"acc={classify_results[i][0]},\
              true labels={classify_results[i][1]},\
              predicted labels={classify_results[i][2]}")
        print("---------------------------------")
```

سپس با استفاده از تابع check_clusters_importance اهمیت هر یک از کلاستر های هر عکس و اولویت آنها را بدست می‌آوریم. در ادامه بخش های مختلف آن و روش کارش را توضیح می‌دهیم.

```python
'''
    prob for each image and true label would be checked,
    ckeck in which one of those 15 state has best result for true label

'''
def check_clusters_importance():
    images_clusters_importance = []
    # for each image in test and train
    for img in range(n_images):

        Effect_of_cluster_removal = {}

        classify_results = cal_classify_results(img, remove=False)
        classify_results_remove = cal_classify_results(img, remove=True)

        probs = cal_probs(classify_results)
        probs_remove = cal_probs(classify_results_remove)

        true_label = classify_results[0][1][img]
        # print(true_label)
        # print(classify_results)
        class_probability = probs[img][0][true_label]
        # print(probs)
        # print(class_probability)
        # check the effect of cluster removal
        # for each cluster
        for i in range(len(probs_remove[img])):
            class_probability_after_remove = probs_remove[img][i][true_label]
            Effect_of_cluster_removal[available_clusters[img][i]] = class_probability_after_remove - class_probability


        # this dictionary contains the importance of
        # clusters for an image in ascending order.
        # (first element has the least importance)
        importance_of_clusters = dict(sorted(Effect_of_cluster_removal.items(), key=lambda item: item[1]))
        images_clusters_importance.append(importance_of_clusters)

    return images_clusters_importance
```

✓ 0.0s

روش کار تابع بالا این صورت است که به ازای هر عکس، یک دیکشنری Effect_of_cluster_removal در نظر میگیریم که در آن هر key نشان دهنده شماره کلاستر است و value آن نشان دهنده این است که بعد از حذف این کلاستر اطمینان عکس برای لیبل آن به چه صورت تغییر میکند.

classify_results خروجی تابع cal_classify_results است که نتایج classification را برای حالتی که کلاستری حذف نشده (remove=False) به ما میدهد.

classify_results_remove خروجی تابع cal_classify_results است که نتایج classification را برای حالتی که کلاستر ها در آن عکس حذف شده باشند (remove=True) به ما میدهد.

(نتایج classification همانطور که قبلا گفته شد شامل لیبل های صحیح عکس ها، لیبل های پیش بینی شده برای عکس ها، اطمینان هر لیبل برای هر عکس و accuracy است.)

```
classify_results = cal_classify_results(img, remove=False)
classify_results_remove = cal_classify_results(img, remove=True)
```

probs و probs_remove به ترتیب حاوی اطمینان هر لیبل برای هر عکس در حالتی که کلاستر ها را حذف نکنیم و حذف کنیم است.

```
probs = cal_probs(classify_results)
probs_remove = cal_probs(classify_results_remove)
```

سپس true_label یعنی همان لیبل اصلی عکس و class_probability یعنی اطمینان لیبل اصلی برای عکس را بدست میاوریم.

```
true_label = classify_results[0][1][img]

class_probability = probs[img][0][true_label]
```

سپس به ازای هر کلاستر عکس، اطمینان لیبل عکس در نبود آن کلاستر را بدست میاوریم. همچنین Effect_of_cluster_removal را نیز که بالاتر توضیح داده ایم برای آن کلاستر بدست میاوریم.

در نهایت این دیکشنری را به صورت صعودی sort میکنیم و در importance_of_clusters ذخیره میکنیم (اولین عضو کمترین اولویت را دارد) و به لیست images_clusters_importance که شامل همه‌ی این دیکشنری ها برای همه‌ی عکس هاست اضافه می‌کنیم.

```
# check the effect of cluster removal
# for each cluster
for i in range(len(probs_remove[img])):
    class_probability_after_remove = probs_remove[img][i][true_label]
    Effect_of_cluster_removal[available_clusters[img][i]] = class_probability_after_remove - class_probability
```

با استفاده از تابع remove_least_important_clusters کم اهمیت ترین کلاستر هر عکس را از لیست کلاستر هایش که در فاز قبلی در لیست all_images_clusters_features ذخیره کرده بودیم حذف میکنیم.

```
def remove_least_important_clusters(images_clusters_importance):
    modified_all_images_clusters_features = []
    for i, img_clstr in enumerate(images_clusters_importance):
        keys_list = list(img_clstr.keys())
        least_important_clstr = keys_list[0]
        modified_all_images_clusters_features.append(np.delete(all_images_clusters_features_bak[i],
                                                               available_clusters[i].index(least_important_clstr),
                                                               axis=0))
        del available_clusters[i][available_clusters[i].index(least_important_clstr)]

    return modified_all_images_clusters_features
```

بنابراین لیست modified_all_images_clusters_features شامل بردار ویژگی همه کلاستر های همه عکس هاست که کم اهمیت ترین کلاستر آنها حذف شده است.

تابع apply_back_ups را قبل از شناسایی و حذف کلاسترهای کم اهمیت صدا میزنیم تا همانطور که گفته شد از لیست های اصلی یک instance بسازد و ما با آنها کار کنیم.

```
    apply_back_ups()
✓ 0.0s
```

در این حلقه for به تعداد k_number-1 بار توابع check_clusters_importance و remove_least_important_clusters را صدا میزیم تا همانطور که توضیح داده شد، کم اهمیت ترین کلاستر ها شناسایی و حذف شوند.

```
for i in range(k_number - 1):
    images_clusters_importance = check_clusters_importance()
    all_images_clusters_features_bak = remove_least_important_clusters(images_clusters_importance)
    k_number_bak -= 1
```

سپس کلاستر های مهم هر عکس را به صورت زیر نمایش میدهیم: (هر دیکشنری نماینده یک عکس است و مقدار key آن شماره‌ی مهمترین کلاستر و مقدار value تغییر میزان اطمینان لیبل عکسش بعد از حذف آن کلاستر است)

```
images_clusters_importance
✓ 0.0s
```

```
[{14: -0.009999999999999995, 12: 0.009999999999999995},
 {12: 0.0, 14: 0.009999999999999998},
 {12: 0.010000000000000009, 13: 0.010000000000000009},
 {13: -0.02, 14: 0.010000000000000002},
 {13: 0.0, 11: 0.020000000000000004},
 {6: -0.030000000000000006, 2: 0.01999999999999999},
 {14: 0.0, 12: 0.010000000000000009},
 {13: 0.0, 14: 0.0},
 {13: 0.0, 14: 0.0},
 {13: 0.0, 14: 0.0},
 {12: 0.0, 13: 0.0},
 {14: 0.0, 10: 0.010000000000000009},
 {13: 0.0, 14: 0.0},
 {10: -0.010000000000000009, 13: 0.050000000000000044},
 {14: 0.009999999999999898, 13: 0.019999999999999907},
 {13: 0.0, 14: 0.0},
 {8: 0.0, 12: 0.010000000000000009},
 {13: 0.0, 14: 0.0},
 {14: 0.0, 13: 0.010000000000000009},
 {12: 0.0, 11: 0.010000000000000009},
 {11: 0.010000000000000009, 14: 0.050000000000000044},
 {12: 0.0, 13: 0.0},
 {12: -0.010000000000000009, 11: 0.020000000000000018},
 {13: 0.0, 14: 0.0},
 {11: 0.010000000000000009, 12: 0.050000000000000044},
 {5: 0.0, 13: 0.0},
 {9: -0.020000000000000018, 13: 0.0},
 {13: 0.0, 14: 0.0},
 {14: -0.020000000000000018, 6: -0.010000000000000009},
 {14: -0.010000000000000009, 11: 0.0}]
```

دو کلاستر مهم هر عکس بدست آمده است. اندیس ۱ اولین مهمترین و اندیس صفر دومین مهمترین کلاستر است.

تابع classify که پیشتر از خروجی آن استفاده کرده بودیم نیز به این صورت عمل میکند که پس از fit و predict کردن داده ها، لیبل های پیشبینی شده برای داده های تست را در لیست y_pred و میزان اطمینان هر لیبل برای عکس های train و test را در آرایه y_prob ذخیره میکنیم.

```
y_pred = clf.predict(X_test)

y_prob_test = clf.predict_proba(X_test)
y_prob_train = clf.predict_proba(X_train)
y_prob = np.concatenate((y_prob_test, y_prob_train), axis=0)
```

همچنین لیبل درست داده ها را در لیست true_labels ذخیره میکنیم. accuracy را نیز بدست میاوریم.
در انتها این مقادیر را return میکنیم و همانطور که گفته شد در مواقع نیاز استفاده میکنیم.

```
return [accuracy, true_lables, y_pred, y_prob]
```

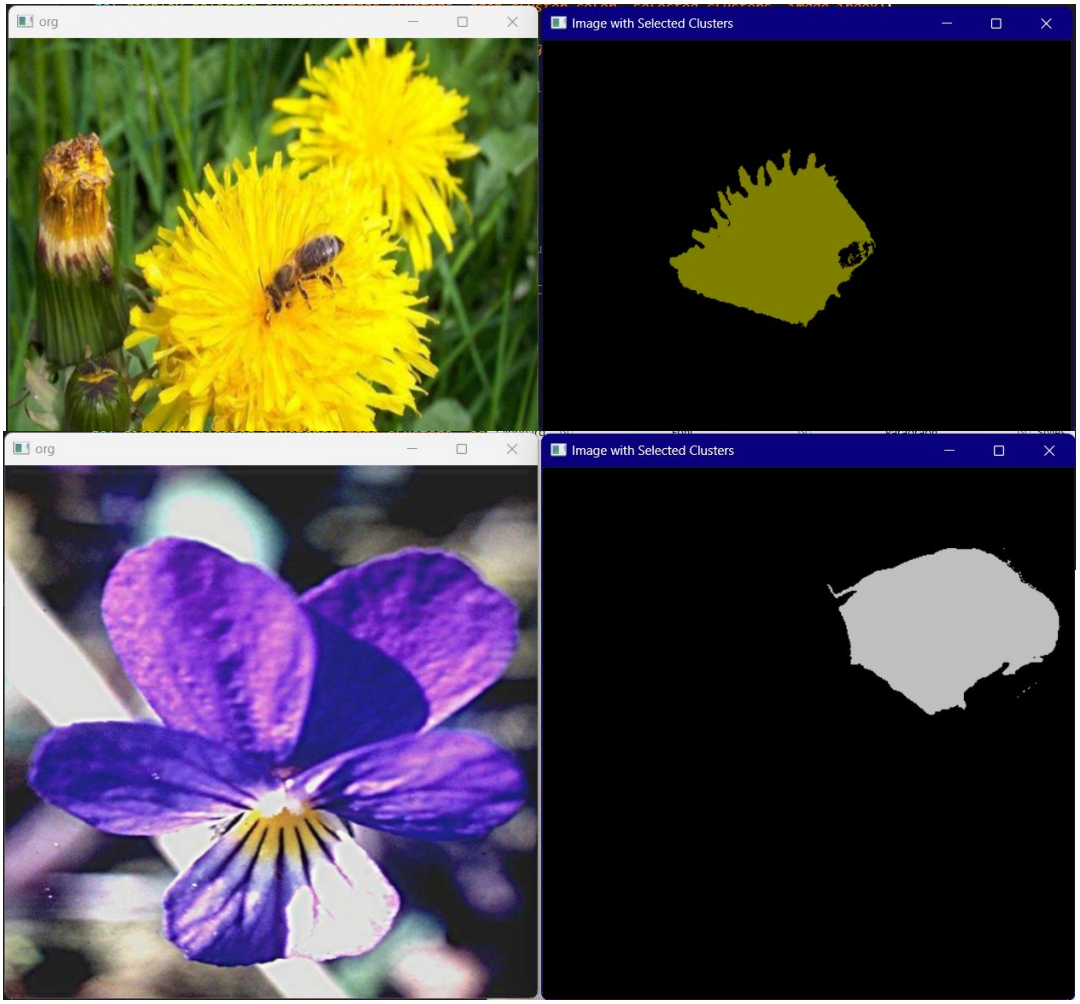
در نهایت با استفاده از تابع display_selected_clusters مهمترین کلاستر هر عکس را نمایش میدهیم.

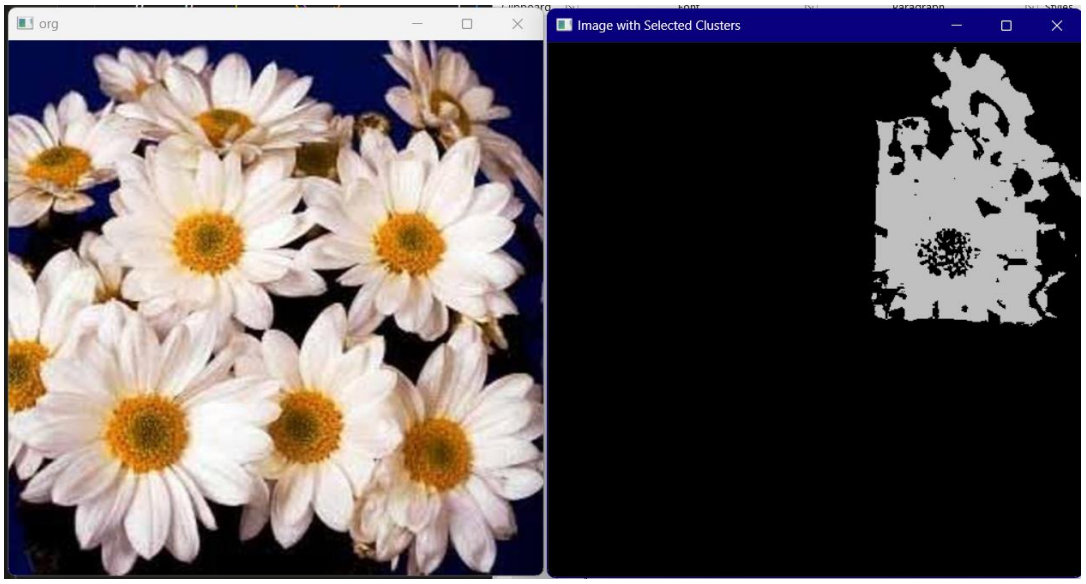
```
import numpy as np
import cv2

def display_selected_clusters(image_clusters, each_cluster_color, selected_clusters, image_index):
    # Create an empty array to hold the HSV values for each pixel
    rgb_image = np.zeros((image_clusters.shape[0], image_clusters.shape[1], 3), dtype=np.uint8)

    # Assign the corresponding HSV values to each pixel for the selected clusters
    for i in range(image_clusters.shape[0]):
        for j in range(image_clusters.shape[1]):
            value = image_clusters[i, j]
            if value == selected_clusters:
                hsv = each_cluster_color[value]
                rgb_image[i, j] = hsv

    # Display the image with corresponding RGB values using cv2.imshow
    cv2.imshow('org', random_values[image_index])
    cv2.imshow('Image with Selected Clusters', cv2.cvtColor(rgb_image, cv2.COLOR_RGB2BGR))


    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

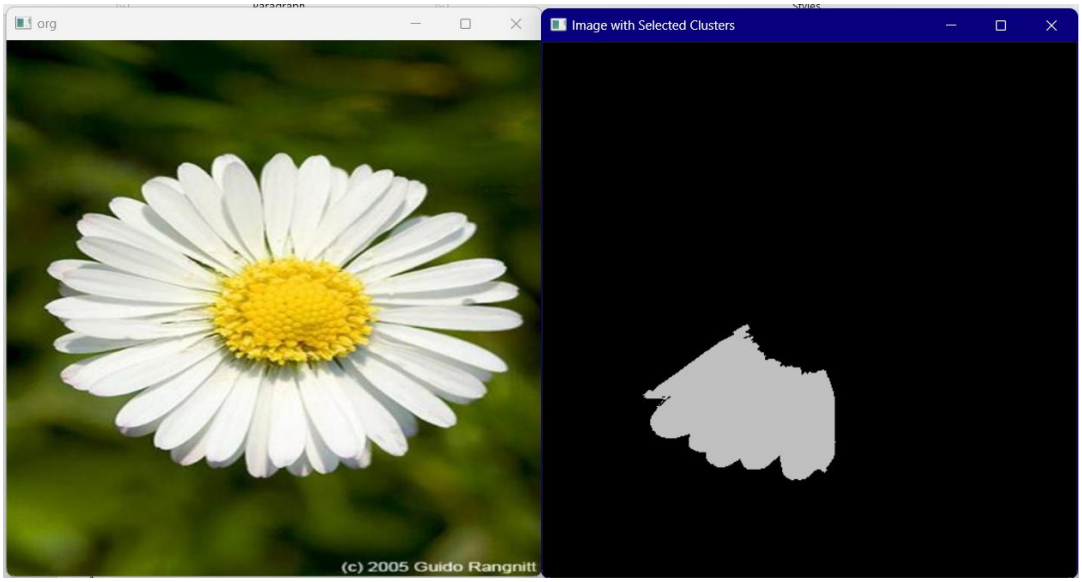
```
for i, image in enumerate(images_clusters_importance):
    important_clusters = list(image.keys())
    display_selected_clusters(new_clusters[i], each_cluster_color, important_clusters[1], i)
```

خروجی این تابع، یعنی مهمترین کلاستر هر عکس، را برای نمونه در اینجا نشان میدهیم:

به دلیل اینکه تعداد کلاستر های هر تصویر ۱۵ عدد است، مهمترین کلاستر بخشی از گل است نه همه‌ی آن.







در بیشتر موارد مهمترین کلاستر به درستی پیدا می‌شود، ولی در چند مورد ممکن است تشخیص صحیح نباشد، مانند نمونه زیر: