بسمه تعالی

ارائه دهنده : زهرا منصوری

عنوان : کارگاه پردازش تصویر

فهرست

پروژه دوم: دسته بندی

مجموعه داده انتخابی

داده انتخابی شامل ۴۰۰ عکس گل است که در ۵ دسته ۸۰ تایی دسته بندی شده اند. لینک دریافت فایل تصویر گل ها به شرح زیر است:

“https://www.robots.ox.ac.uk/~vgg/data/flowers/17/”

دسته بندی گل ها در پوشه هایی با نام هر دسته، داخل پوشه images پروژه قرار دارد.

نمونه هایی از تصاویر به شرح زیر است:


image_0241.jpg


image_0242.jpg


image_0243.jpg

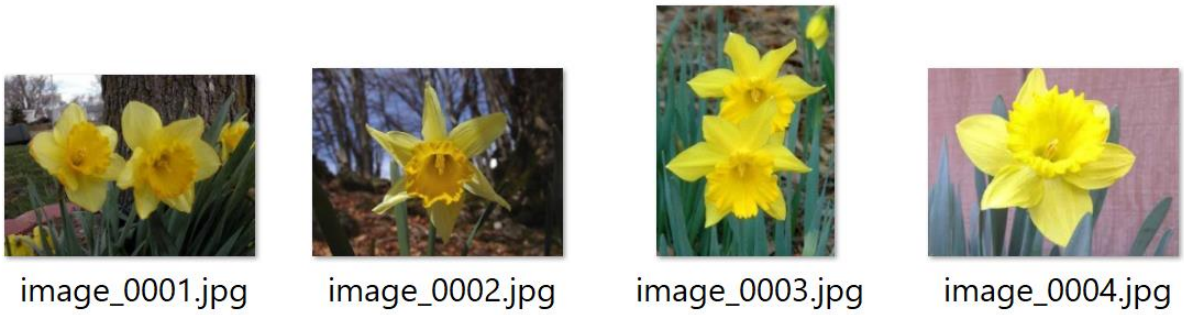

image_0244.jpg


image_0321.jpg


image_0322.jpg


image_0323.jpg


image_0324.jpg



مسئله دسته بندی

برای استخراج descriptor مناسب، باید به ویژگی هایی که باعث ایجاد تمایز بین دسته های مختلف گل ها می شوند؛ توجه کرد.

با توجه به تصاویر بالا، رنگ می تواند یکی از این معیار ها باشد اما به این ویژگی به تنهایی ممکن است نتواند بین همه دسته بندی ها تمایز ایجاد کند. برای مثال، گل هایی که در دسته daffodil قرار می گیرند؛ رنگ زرد دارند و به خوبی می شود به کمک رنگ، این دسته از گل ها را از بقیه دسته ها تشخیص داد؛ اما مثلا رنگ بنفش، توصیف گر خوبی برای ایجاد تمایز بین دو دسته bluebell و crocus نیست.

از طرف دیگر نیز segment کردن تصاویر و به دست آوردن گل موجود در تصویر، گاهی نیاز به حذف background و توجه به قسمت foreground تصویر است.

استخراج ویژگی از تصاویر

برای حل مسئله، ابتدا باید ویژگی های مناسب، شامل لیستی از اعداد که بیانگر هر تصویر است؛ به دست آوریم.

سه دسته از ویژگی هایی که به خوبی می توانند گونه های مختلف گل را از هم جدا کنند؛ شامل رنگ (Color)، بافت (Texture) و شکل (Shape) آنها است.

اما اگر فقط یک بردار از ویژگی ها را انتخاب کنیم؛ این روش نتیجه خوبی نخواهد داشت زیرا این گونه ها دارای ویژگی های مشترک زیادی هستند. برای مثال همانطور که گفته شد؛ گل های دسته bluebell و crocus از نظر رنگ و غیره بسیار مشابه هستند. بنابراین ، ما باید تصویر را با ترکیب توصیف گرهای مختلف ویژگی ، به گونه ای که تصویر را به طور موثرتری توصیف کند ، تعیین کنیم.

Feature descriptor

توصیفگر های استفاده شده در این پروژه عبارتند از:

- رنگ (Color): آمار کانال های رنگی هر تصویر را گرفته و با استفاده از میانگین و انحراف معیار و نمودار هیستوگرام رنگی تصویر، لیست رنگی مناسب با هر تصویر را بر می گرداند.
- شکل (Shape): با استفاده از ویژگی Hu Moments می توان میانگین وزنی از شدت پیکسل های تصویر، به دست آورد.
- بافت (Texture): با استفاده از دو متد Haralick Texture و Local Binary Patterns می توان بافت تصویر را نیز تشخیص داد. برای مثال یکی از روش های محاسبه آن به این صورت است که دایره ای به شعاع دلخواه از مرکزی دلخواه از تصویر، انتخاب کرده و با توجه به شدت، اندازه و ... پیکسل، یک عدد اعشاری بر می گرداند و این کار را روی باقی قسمت های مهم تصویر نیز انجام داده و لیستی از اعداد اعشاری را که بیانگر بافت تصویر است، بر می گرداند.

توضیحات اولیه برنامه

- در خطوط اول برنامه، کتابخانه های لازم import شده است.
- خط ۱۳ برنامه، برای تبدیل تصویر ورودی به اندازه ثابت (۵۰۰ ، ۵۰۰) استفاده می شود.
- خط ۱۴ برنامه، محدوده رنگ ها را در هیستوگرام مربوط به رنگ مشخص می کند.

```python
import os
import cv2
import mahotas
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier

fixed_size = tuple((500, 500))
bins = 8
```

متد های نوشته شده برای استخراج ویژگی

1. fd_hu_moments()

ابتدا تصویر رنگی خود را به تصویری در مقیاس خاکستری تبدیل می کنیم. برای استخراج ویژگی های Hu Moments از تصویر ، از تابع cv2.HuMoments()، از کتابخانه OpenCV استفاده می کنیم. ورودی های این تابع، moment های تصویر است که با استفاده از تابع cv2.moments() آن ها را استخراج کرده و با استفاده از flatten()، آن را به بردار تبدیل می کنیم.

```python
# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

2. fd_haralick()

برای استخراج ویژگی های بافت Haralick تصاویر، از کتابخانه mahotas و تابع mahotas.features.haralick() استفاده می کنیم. قبل از انجام این کار، تصویر رنگی خود را به یک تصویر خاکستری تبدیل می کنیم، زیرا توصیف کننده ویژگی haralick انتظار دارد که تصاویر در مقیاس خاکستری باشد.

```python
# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick
```

3. fd_histogram()

برای استخراج ویژگی های Color Histogram از تصویر ، از تابع cv2.calcHist() استفاده می کنیم. آرگومان ها ورودی این تابع عبارتند از: تصویر ، کانال های رنگی (R,G,B) ، Mask ، histSize (bin) و دامنه برای هر کانال (به طور معمول ۰-۲۵۶).

```
# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()
```

پردازش هر تصویر و تشکیل dataframe ویژگی ها

ابتدا دو لیست خالی global_features و labels را تعریف کرده ایم. این دو لیست قرار است ویژگی های مربوط به هر تصویر و کلاس آن تصویر را نگهداری کنند. لیست train_labels نیز، شامل نام ۵ دسته گلی است که داریم.

سپس با استفاده از دو حلقه for تو در تو، روی پوشه های مربوط به دسته های گل و تصاویر داخل آن پوشه ها گشته و به ازاء هر تصویر، متدهای گفته شده در بخش قبل را صدا زده و feature های به دست آمده به همراه label هر تصویر را به لیست های global_features وlabels اضافه می کنیم.

```
global_features = []
labels = []
train_labels = ['bluebell', 'crocus', 'daffodil', 'lilyvalley', 'snowdrop']

folder = '/Users/zahramansoori/Desktop/ImageProcessing2/images'
for name in train_labels:
    folder = folder + '/' + name
    for filename in os.listdir(folder):
        # print(filename)
        image = cv2.imread(os.path.join(folder, filename))
        image = cv2.resize(image, fixed_size)

        fv_hu_moments = fd_hu_moments(image)
        fv_haralick = fd_haralick(image)
        fv_histogram = fd_histogram(image)

        global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])

        labels.append(name)
        global_features.append(global_feature)
    folder = '/Users/zahramansoori/Desktop/ImageProcessing2/images'
```

در ادامه کد برای شماره گذاری کلاس ها از LabelEncoder() استفاده شده است.

برای normalize کردم محدوه ویژگی ها بین ۰ تا ۱ از متد MinMaxScaler() کتابخانه scikit-learn استفاده کرده ایم.

در نهایت نیز feature های به دست آمده را در df_f و lable های متناظر با هر رکورد df_f را در df_l لود کرده ایم.

```
69
70      print("feature vector size {}".format(np.array(global_features).shape))
71
72
73      print("training Labels {}".format(np.array(labels).shape))
74
75
76      targetNames = np.unique(labels)
77      le = LabelEncoder()
78      target = le.fit_transform(labels)
79
80
81      scaler = MinMaxScaler(feature_range=(0, 1))
82      rescaled_features = scaler.fit_transform(global_features)
83
84
85      df_f = pd.DataFrame(rescaled_features)
86      df_l = pd.DataFrame(target)
87      print(df_f)
88      print(df_l)
```

```
feature vector size (400, 532)
training Labels (400,)
          0         1         2    ...       529       530       531
0    0.000022  0.002057  0.004015  ...  0.777331  0.075766  0.949841
1    0.000000  0.000000  0.000000  ...  0.777338  0.075699  0.949840
2    0.001000  0.009007  0.019085  ...  0.777344  0.075971  0.949843
3    0.000000  0.002226  0.007695  ...  0.777338  0.075936  0.949841
4    0.011868  0.015326  0.005638  ...  0.777342  0.075111  0.949824
..        ...       ...       ...  ...       ...       ...       ...
395  0.180303  0.055102  0.025156  ...  0.778488  0.049694  0.946656
396  0.066348  0.000973  0.000898  ...  0.777321  0.077021  0.949793
397  0.316265  0.954399  0.082664  ...  0.777278  0.080367  0.949904
398  0.002642  0.004192  0.002607  ...  0.777334  0.075376  0.949844
399  0.010937  0.020979  0.020273  ...  0.777499  0.070314  0.950319

[400 rows x 532 columns]
     0
0    0
1    0
2    0
3    0
4    0
..  ..
395  4
396  4
397  4
398  4
399  4
```

محاسبه confusion matrix و accuracy, precision, recall

ابتدا ۷۰ درصد از نمونه ها را برای آموزش به learner خود داده و ۳۰ درصد از آنها را برای تست learner نگه می داریم.

برای محاسبه confusion matrix و خروجی classification report نیز با استفاده از روش هایی که در کارگاه یادگیری ماشین آموختیم؛ تحت عنوان متد ML به خروجی زیر می رسیم.

accuracy حدود ۷۲ درصد شده است؛ که نشان دهنده دقت ۷۲ درصدی learner ما می باشد.

```
def ML():
    x = df_f.iloc[:, :].values
    y = df_l.iloc[:, :].values

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

    clf = DecisionTreeClassifier()
    clf = clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    result = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(result)
    result1 = classification_report(y_test, y_pred)
    print("Classification Report:", )
    print(result1)
    result2 = accuracy_score(y_test, y_pred)
    print("Accuracy:", result2)


ML()
```

```
[400 rows x 1 columns]
Confusion Matrix:
[[14  3  0  1  0]
 [ 1 20  1  0  1]
 [ 0  2 22  1  0]
 [ 3  1  0 14  9]
 [ 1  2  0  7 17]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.78      0.76        18
           1       0.71      0.87      0.78        23
           2       0.96      0.88      0.92        25
           3       0.61      0.52      0.56        27
           4       0.63      0.63      0.63        27

    accuracy                           0.73       120
   macro avg       0.73      0.74      0.73       120
weighted avg       0.73      0.72      0.72       120

Accuracy: 0.725
```