# تشخیص و طبقه بندی بیماری برگ گیاه با استفاده از تکنیک های پردازش تصویر

**استاد راهنما: دکتر شبنم شادرو**

**دانشجو: نفیسه فغانی**

**بهار 1403**

**چکیده**:

کشاورزی یکی از اساسی‌ترین فعالیت‌های اقتصادی و زیربنایی هر کشور است که نقشی حیاتی در تأمین امنیت غذایی، ایجاد اشتغال و توسعه روستایی ایفا می‌کند. بیماری‌های گیاهان کاهش کمیت و کیفیت محصولات، افت اقتصادی و حتی گرسنگی را به همراه می آورند. این بیماری‌ها می‌توانند به سرعت گسترش یابند و باعث از بین رفتن بخش عظیمی از محصولات کشاورزی شوند. به همین دلیل، تشخیص زودهنگام و دقیق این بیماری ها اهمیت فراوانی دارد.

استفاده از تکنیک‌های پردازش تصویر برای تشخیص بیماری‌های گیاهان، یکی از پیشرفته‌ترین روش‌ها در این زمینه است. این تکنیک‌ها می‌توانند به سرعت و با دقت بالا علائم بیماری را شناسایی کنند، که این امر موجب می‌شود تا اقدامات کنترلی و درمانی به موقع انجام شود. همچنین، این روش‌ها می‌توانند هزینه‌ها و زمان لازم برای تشخیص را کاهش دهند و به بهبود مدیریت منابع و افزایش بهره‌وری کشاورزی کمک کنند.

سیستم ارائه‌شده این توانایی را دارد که بیماری‌های برگ گیاهان را در چهار گروه rust، powdery mildew، bacterial blight و Cercospora leaf spot با دقت 85.71 درصد تشخیص داده و طبقه‌بندی کند.

**کلمات کلیدی**: *پردازش تصویر، K-Means، Local binary pattern، support vector machine، تشخیص بیماری گیاه*

**مقدمه:**

کشاورزی به عنوان یکی از پایه‌های اصلی اقتصاد هر کشور، نقشی حیاتی در تأمین امنیت غذایی و توسعه اقتصادی دارد. بیماری‌های گیاهی، اگرچه به ظاهر کوچک و تاثیرات محدودی دارند، می‌توانند به طور قابل توجهی به کاهش عملکرد و کیفیت محصولات کشاورزی منجر شوند. تشخیص به موقع بیماری‌های گیاهی از طریق فناوری‌های مدرن مانند پردازش تصویر، امکان می‌دهد تا با سرعت و دقت بالا علائم ابتلا به بیماری را شناسایی و تحلیل کرده و اقدامات پیشگیری و درمانی مناسب را به صورت بهینه و به موقع انجام دهیم. این اقدامات نه تنها به بهبود عملکرد کشاورزی کمک می‌کنند بلکه از خسارات اقتصادی و زیست‌محیطی ناشی از بیماری‌های گیاهی جلوگیری می‌کنند، که به طور کلی موجب بهبود وضعیت کشاورزی و اقتصاد کشور می‌شود.

بیماری‌های گیاهی از جمله عواملی هستند که می‌توانند به شدت بر کشاورزی تأثیر بگذارند. هر یک از بیماری‌های Cercospora leaf spot ، bacterial blight، rust و powdery mildew دارای خصوصیات و علائم خاصی هستند که می‌توانند به طور مستقیم یا غیرمستقیم به کاهش عملکرد و کیفیت محصولات کشاورزی منجر شوند. علائم این بیماری ها معمولا در برگ گیاه مشهود و قابل تشخیص هستند.

استفاده از پردازش تصویر در تشخیص بیماری‌های گیاهان نسبت به استفاده از چشم غیر مسلح، از مزایای بسیاری برخوردار است. این فناوری قادر به شناسایی علائم کوچک و غیر قابل رؤیت به چشم غیر مسلح است که ممکن است از دید بیرونی نادیده گرفته شوند. سیستم‌های پردازش تصویر با دقت و سرعت بالا عمل می‌کنند که این امر منجر به تشخیص زودهنگام و پیشگیری از گسترش بیماری‌ها می‌شود. همچنین، این روش به دلیل قابلیت تکرار پذیری بالا، از مشکلاتی که ممکن است در تشخیص دستی ایجاد شود، جلوگیری می‌کند و به تصمیم‌گیری‌های دقیق‌تر و مؤثرتر در مدیریت بیماری‌های گیاهان کمک می‌کند. به طور کلی، استفاده از پردازش تصویر در کشاورزی نه تنها به بهبود کارایی و افزایش بهره‌وری منجر می‌شود بلکه از لحاظ اقتصادی و زیست‌محیطی نیز مزایای قابل توجهی را به همراه دارد.

این پژوهش به منظور حل مشکلاتی که با استفاده از چشم غیر مسلح در تشخیص بیماری‌های گیاهی پیش می‌آید، طراحی شده است. دیتاست مورد استفاده در این مطالعه، بخشی از دیتاست PlantVillage است که شامل چهار گروه بیماری Cercospora leaf spot ، bacterial blight، rust و powdery mildew می‌باشد. این دیتاست شامل 560 نمونه تصویر میباشد که از آن‌ها 115 تصویر مربوط به bacterial blight ، 120 تصویر مربوط به Cercospora leaf spot، 200 تصویر مربوط به powdery mildew و 125 تصویر مربوط به rust می‌باشد. این سیستم از چندین مرحله مختلف تشکیل شده است که شامل قطعه‌بندی تصاویر، استخراج ویژگی‌های مربوط به بیماری، تشخیص و طبقه‌بندی می‌باشد. در بخش بعدی از این پژوهش، به بررسی دقیق‌تر ساختار این سیستم و مراحل مختلف آن می‌پردازیم.

**ساختار سیستم:**

***پیش پردازش تصویر:***

```
1    import os
2    import cv2
```

وارد کردن کتابخانه های مورد نیاز:

1- کتابخانه‌ی os را وارد می‌کنیم که برای ارتباط با سیستم عامل استفاده می‌شود و به کمک آن تعامل با فایل‌ها و دایرکتوری‌ها در سیستم عامل انجام میشود و برای کار با مسیرهای فایل استفاده میشود.

2- کتابخانه OpenCV را وارد می‌کنیم که برای خواندن تصاویر و انجام عملیات پیچیده بر روی تصاویر اعم از تبدیل فرمت ها ، اعمال فیلترها و ... استفاده میشود.

```
5    # Define the directories
6    directories = ['D:/plant-disease/data/powdery mildew',
7                   'D:/plant-disease/data/rust',
8                   'D:/plant-disease/data/bacterial blight',
9                   'D:/plant-disease/data/Cercospora leaf spot']
```

یک لیست از مسیرهای دایرکتوری‌ها را تعریف می‌کند که هر کدام از آن‌ها متناظر با یک نوع بیماری گیاهی هستند. هر دایرکتوری به عنوان مکان ذخیره تصاویر مربوط به یک نوع خاص از بیماری گیاهی استفاده شده است. این اطلاعات به منظور بارگذاری داده‌های تصویری از دایرکتوری‌ها، پردازش تصاویر و آماده‌سازی آنها برای استفاده در مدل یادگیری ماشین استفاده میشود.

```
12  # load images from directories
13  def load_images_from_directories(directories):
14      images = []
15      labels = []
16      for directory in directories:
17          label = os.path.basename(directory)   # last part(tail) of the path
18          for filename in os.listdir(directory):  # list of all files and directories in the specified path
19
20              img_path = os.path.join(directory, filename)
21              img = cv2.imread(img_path)
22              if img is not None:
23                  images.append(img)
24                  labels.append(label)
25      return images, labels
```

این تابع به طور خاص برای بارگذاری تصاویر از دایرکتوری‌های مختلف طراحی شده است. این تابع دو لیست images و labels را بر اساس دایرکتوری‌های داده شده برمی‌گرداند. این دو لیست شامل تصاویر و برچسب‌های متناظر با هر تصویر هستند.

ورودی‌ها: (directories) یک لیست از مسیرهای دایرکتوری‌ها که هر کدام شامل تصاویر یک نوع خاص از بیماری گیاهی هستند.

تابع به ترتیب هر دایرکتوری موجود در directories را باز می‌کند. اخرین قسمت مسیر هر دایرکتوری که متشکل از نام بیماری تصاویر موجود در آن است را به عنوان برچسب ذخیره میکند. سپس به ازای هر فایل موجود در دایرکتوری ، مسیر فایل را تشکیل میدهد و تصویر را با استفاده از cv2.imread() بارگذاری می‌کند. اگر تصویر موجود باشد ، تصویر به لیست images اضافه می‌شود و برچسب متناظر با آن به لیست labels اضافه می‌شود. در نهایت، تابع دو لیست images وlabels را برمیگرداند

```python
# preprocess images
def preprocess_image(img, size=(256, 256)):
    # Resize image
    resized_img = cv2.resize(img, size)

    # Convert to HSV format
    hsv_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2HSV)

    # Apply median filter
    median_filtered_img = cv2.medianBlur(hsv_img, 5)

    # Enhance image contrast using histogram equalization
    # Focus on Value channel because it directly corresponds to the brightness of the image
    hsv_img[:, :, 2] = cv2.equalizeHist(median_filtered_img[:, :, 2])

    # Convert back to BGR
    enhanced_img = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2BGR)

    return enhanced_img
```

این تابع، یک تصویر را پیش‌پردازش می‌کند و بهبود میدهد تا برای مراحل بعدی پردازش تصویر آماده شود .

این تابع یک تصویر ورودی را می‌گیرد و آن را به چندین مرحله پردازش می‌کند تا بهبود یابد.

ورودی ها:

**img** : تصویری که باید پیش‌پردازش شود.

**Size**: اندازه‌ای که تصویر به آن تغییر اندازه داده می‌شود. مقدار پیش‌فرض (256, 256) است که اندازه استاندارد است.

تصویر را به اندازه مشخص شده (256x256) تغییر اندازه می‌دهد.

```python
    # Resize image
    resized_img = cv2.resize(img, size)
```

ورودی: تصویر اصلی

خروجی: تصویر تغییر اندازه داده شده

```python
    # Convert to HSV format
    hsv_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2HSV)
```

تصویر را از فرمت BGR به HSV تبدیل می‌کند. فرمت HSV برای تنظیم روشنایی مناسب تر است.

ورودی: تصویر تغییر اندازه داده شده

خروجی: تصویر در فرمت HSV

```
36      # Apply median filter
37      median_filtered_img = cv2.medianBlur(hsv_img, 5)
```

فیلتر میانه را با اندازه کرنل 5 x 5 اعمال می‌کند. (برای کاهش نویز در تصویر)

ورودی: تصویر در فرمت HSV

خروجی: تصویر فیلتر شده

```
39      # Enhance image contrast using histogram equalization
40      # Focus on Value channel because it directly corresponds to the brightness of the image
41      hsv_img[:, :, 2] = cv2.equalizeHist(median_filtered_img[:, :, 2])
```

کانال V (روشنایی) را در تصویر HSV با استفاده از هموارسازی هیستوگرام بهبود می‌دهد.

(برای بهبود کنتراست تصویر برای وضوح بیشتر)

ورودی: کانال V از تصویر فیلتر شده

خروجی: تصویر HSV با کانال V بهبود یافته

```
43      # Convert back to BGR
44      enhanced_img = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2BGR)
```

تصویر را از فرمت HSV به BGR برمی‌گرداند. فرمت اصلی BGR برای نمایش و پردازش‌های بعدی استفاده می‌شود.

ورودی: تصویر HSV با کانال V بهبود یافته

خروجی: تصویر نهایی بهبود یافته در فرمت BGR

در نهایت تصویر نهایی پیش‌پردازش شده که شامل تغییر اندازه، کاهش نویز، و بهبود کنتراست است توسط تابع برگردانده میشود.

## ***قطعه بندی تصویر:***

وارد کردن کتابخانه های مورد نیاز:

```python
import cv2
import numpy as np
from sklearn.cluster import KMeans
```

1- کتابخانه OpenCV را وارد می‌کنیم که برای انجام عملیات پیچیده بر روی تصاویر استفاده میشود.

2- کتابخانه NumPy را وارد می‌کنیم که برای کار با آرایه‌های چند بعدی (تصویر آرایه چند بعدی است) و دستکاری و تغییر شکل ماتریس‌های تصویری استفاده می‌شود.

3- KMeans را از کتابخانه Scikit-learn وارد می‌کند که یک الگوریتم محبوب برای خوشه‌بندی داده‌ها است. برای گروه‌بندی یا خوشه‌بندی داده‌ها به k گروه مختلف بر اساس ویژگی‌های مشابه استفاده می‌شود. در پردازش تصویر، KMeans می‌تواند برای بخش‌بندی تصویر، فشرده‌سازی تصویر، و استخراج ویژگی‌ها استفاده شود.

```python
# k-means clustering for image segmentation
def segment_image(img, k=3):
    # Reshape the image into a 2D array where each row represents a pixel and its BGR values
    x = img.reshape((-1, 3))  # row dim: unknown(pixels)  col dim:3(b,g,r)   image.shape -> (unknown, 3)

    # fix the seed of the random number generator used for centroid initialization (random_state=0)
    # ensures that the K-Means algorithm produces the same results each time it's running on the same data
    kmeans = KMeans(n_clusters=k, random_state=0).fit(x)

    # Get the cluster centers (which are the dominant colors) and assign each pixel to its nearest cluster center
    segmented_img = kmeans.cluster_centers_[kmeans.labels_]

    # Reshape the segmented image back to its original shape
    segmented_img = segmented_img.reshape(img.shape)

    # Convert the segmented image to unsigned 8-bit integer format (the values were floating points)
    segmented_img = segmented_img.astype(np.uint8)

    # returns the trained K-Means clustering model and the segmented image
    return segmented_img, kmeans
```

این تابع برای بخش‌بندی تصویر با استفاده از خوشه‌بندی K-Means طراحی شده است. این تابع تصویر ورودی را به k خوشه تقسیم می‌کند و یک تصویر بخش‌بندی شده به همراه مدل K-Means آموزش دیده شده را برمی‌گرداند.

ورودی ها:

Img: تصویر ورودی که قرار است بخش‌بندی شود.

k: تعداد خوشه‌ها برای الگوریتم .K-Means مقدار پیش‌فرض 3 است.

```
9       x = img.reshape((-1, 3))  # row dim: unknown(pixels)  col dim:3(b,g,r)
```

تصویر را به یک آرایه دو بعدی تبدیل می‌کند که هر سطر نمایانگر یک پیکسل و مقادیر BGR آن است. (1- به معنای مشخص نبود تعداد سطر های خروجی است)

خروجی: آرایه دو بعدی با ابعاد (3، تعداد پیکسل ها)

```
13      kmeans = KMeans(n_clusters=k, random_state=0).fit(x)
```

مدل K-Means را با تعداد خوشه‌های k و با مقدار دهی اولیه تصادفی ثابت آموزش میدهد.
خروجی آن مدل K-Means آموزش دیده است و ورودی آن آرایه دو بعدی x است.

از آن جا که مقدار دهی اولیه تصادفی ثابت است، در هر بار اجرا بر روی یک دیتای ثابت، خروجی ثابتی را تولید میکند.

```
17      segmented_img = kmeans.cluster_centers_[kmeans.labels_]
```

دریافت مراکز خوشه‌ها و تخصیص هر پیکسل به نزدیک‌ترین مرکز خوشه: هر پیکسل را به نزدیک‌ترین مرکز خوشه اختصاص می‌دهد و تصویر بخش‌بندی شده را ایجاد می‌کند.
kmeans.labels_ برچسب هایی که برای پیکسل های تصویر استفاده شده اند، میباشد (0 و 1 و 2)

cluster_centers_ مراکز خوشه ها (رنگ های غالب تصویر) میباشد.

خروجی: تصویر بخش‌بندی شده با مقادیر مرکز خوشه‌ها

```
19          segmented_img = segmented_img.reshape(img.shape)
```

تصویر بخش‌بندی شده را به شکل اصلی آن بازمی‌گرداند.

ورودی: تصویر بخش‌بندی شده به شکل آرایه دو بعدی

خروجی: تصویر بخش‌بندی شده به شکل اصلی

```
22          segmented_img = segmented_img.astype(np.uint8)
```

تصویر بخش‌بندی شده را به فرمت 8 بیتی بدون علامت تبدیل می‌کند.

خروجی: تصویر بخش‌بندی شده به فرمت صحیح

در نهایت تابع تصویر بخش‌بندی شده که به k خوشه تقسیم شده است را به همراه مدل K-Means آموزش دیده شده، برمیگرداند.

```python
def select_infected_segment(img, label, kmeans):
    # Calculate the mean color of each segment
    mean_colors = kmeans.cluster_centers_  # mean_colors.shape --> (k, 3)

    # Calculate distance(column-sum norm) between each mean color and a reference color for infection
    if label == 'powdery mildew':
        reference_color = np.array([200, 200, 200])  # Very light gray color
    elif label == 'rust':
        reference_color = np.array([139, 69, 19])  # Reddish-brown color
    elif label == 'bacterial blight':
        reference_color = np.array([139, 69, 19])  # Darker reddish-brown
    else:
        reference_color = np.array([100, 50, 50]),  # Dark brownish

    distances = np.linalg.norm(mean_colors - reference_color, axis=1)  # axis=1: row wise
    # distances.shape --> (3,1)  each row represents the distance of each cluster from the reference color
    # Select the segment with the smallest distance to the reference color
    infected_segment_label = np.argmin(distances)

    # Create a mask for the infected segment
    infected_mask = (kmeans.labels_ == infected_segment_label).reshape(img.shape[:2])  # [:2]: height & width
    # (65536,1)-->(256,256)
    # Extract the infected segment from the original image
    infected_segment = cv2.bitwise_and(img, img, mask=infected_mask.astype(np.uint8))

    return infected_segment, infected_mask
```

این تابع به منظور انتخاب بخش آلوده شده به بیماری یک تصویر بر اساس برچسب بیماری و مدل K-Means طراحی شده است.

ورودی ها:

Img : تصویر ورودی

Label: برچسب بیماری (مثلا 'rust')

Kmeans: مدل K-Means آموزش دیده که تصویر را بخش‌بندی کرده است.

```
31          mean_colors = kmeans.cluster_centers_
```

میانگین رنگ هر بخش (خوشه) را از مدلK-Means که همان مراکز خوشه ها هستند را بدست می‌آورد. خروجی یک آرایه (k x 3) هست.

```
34      if label == 'powdery mildew':
35          reference_color = np.array([200, 200, 200])  # Very light gray color
36      elif label == 'rust':
37          reference_color = np.array([139, 69, 19])  # Reddish-brown color
38      elif label == 'bacterial blight':
39          reference_color = np.array([139, 69, 19])  # Darker reddish-brown
40      else:
41          reference_color = np.array([100, 50, 50]),  # Dark brownish
42
43      distances = np.linalg.norm(mean_colors - reference_color, axis=1)  # axis=1: row wise
```

رنگ مرجع برای هر بیماری را تعیین می‌کند و فاصله اقلیدسی بین میانگین رنگ هر خوشه و رنگ مرجع را محاسبه می‌کند. خروجی نهایی یک آرایه در ابعاد (k x 1) است.

در واقع از هر سطر ماتریس mean_colors که در آن هر سطر نماینده رنگ یک خوشه است، رنگ مرجع را کم میکند. خروجی این عمل اندازه ای مشابه mean_colors دارد یعنی ( k x 3) است. سپس هر مقدار درون ماتریس را به توان دو می‌رساند. در نهایت، این

مقادیر در هر سطر با هم جمع شده و از آن‌ها جذر گرفته می‌شود. این خروجی همان فاصله اقلیدسی بین میانگین رنگ هر خوشه و رنگ مرجع است.

```
47          infected_segment_label = np.argmin(distances)
```

بخشی که کمترین فاصله اقلیدسی را با رنگ مرجع دارد را انتخاب می‌کند. np.argmin() شماره سطری که کمترین مقدار را دارد برمیگرداند که این در واقع همان شماره برچسب خوشه ای هست که کمترین فاصله اقلیدسی را با رنگ مرجع دارد.

```
49          # Create a mask for the infected segment
50          infected_mask = (kmeans.labels_ == infected_segment_label).reshape(img.shape[:2])
```

ماسکی باینری ایجاد می‌کند که پیکسل‌های مربوط به بخش آلوده را مشخص می‌کند.

در واقع به ازای هر پیکسلی که برچسب آن با برچسب خوشه بیمار یکسان است، مقدار 1 و برای دیگر پیکسل ها مقدار صفر را در نظر میگیرد. در نهایت این ماسک را به یک آرایه دو بعدی با اندازه طول و عرض مشابه تصویر اصلی تبدیل میکند تا بتوان آن را بر روی تصویر اصلی استفاده کرد.

img.shape[:2] دو بعد اول تصویر یعنی طول و عرض را برمیگرداند.

```
52          # Extract the infected segment from the original image
53          infected_segment = cv2.bitwise_and(img, img, mask=infected_mask.astype(np.uint8))
```

با استفاده از ماسک ایجاد شده، بخش آلوده به بیماری را از تصویر اصلی استخراج می‌کند.

ورودی: تصویر اصلی ماسک بخش آلوده (بخش آلوده در آن سفید است و بقیه قسمت ها مشکی هستند)

خروجی: قسمت آلوده به بیماری از تصویر اصلی

ماسک بر روی تصویر اعمال شده و فقط پیکسل‌هایی که متناظر با پیکسل‌های مقدار یک در ماسک هستند، در تصویر نهایی حضور دارند.

در نهایت این تابع بخشی از تصویر که آلوده تشخیص داده شده است را به همراه ماسک دودویی که نشان می‌دهد کدام پیکسل‌ها آلوده هستند را برمیگرداند.

## ***استخراج ویژگی***:

```
from skimage.feature import graycomatrix, graycoprops, local_binary_pattern
import cv2
import numpy as np
```

وارد کردن کتابخانه ها و ماژول های مورد نیاز:

1- سه تابع زیر از کتابخانه skimage.feature وارد میشوند:

graycomatrix: این تابع برای محاسبه ماتریس GLCM استفاده می‌شود که برای تحلیل بافت تصویر کاربرد دارد.

graycoprops: این تابع برای استخراج ویژگی‌های آماری از ماتریس GLCM استفاده می‌شود. مانند کنتراست، همگنی، انرژی و همبستگی و ...

local_binary_pattern: این تابع برای محاسبه LBP استفاده میشود که یک روش موثر برای استخراج ویژگی‌های بافت از تصاویر است.

2- کتابخانه OpenCV را وارد می‌کنیم که برای انجام عملیات پیچیده بر روی تصاویر مانند تبدیل فرمت استفاده میشود.

3- کتابخانه NumPy را وارد می‌کنیم که ابزارهای مختلفی برای کار با آرایه‌ها و ماتریس‌های چند بعدی و انجام عملیات ریاضیاتی سریع و کارا ارائه می‌دهد.

```python
# extract GLCM and LBP features
def extract_features(img):
    if img.ndim == 2:  # ndim: number of dimensions
        gray_img = img
    elif img.ndim == 3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        raise ValueError("Invalid image format")
    # Extract GLCM features
    # the GLCM matrix will be symmetric (ignoring the order of value pairs)
    # levels=256: for an 8-bit input image (number of gray-levels)
    glcm = graycomatrix(gray_img, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)
    #  Measures the intensity contrast
    contrast = graycoprops(glcm, 'contrast')[0]  # The function returns a 2D array (one row per distance-angle pair)
    # similar to contrast but weighs differences linearly rather than quadratically
    dissimilarity = graycoprops(glcm, 'dissimilarity')[0]
    # Higher values indicate more homogeneity
    homogeneity = graycoprops(glcm, 'homogeneity')[0]
    # Higher values indicate more uniformity
    energy = graycoprops(glcm, 'energy')[0]
    # ranges between -1 and 1. Higher values indicate a positive correlation
    correlation = graycoprops(glcm, 'correlation')[0]
    # indicates image texture uniformity
    asm = graycoprops(glcm, 'ASM')[0]
```

این تابع تصویر قسمت آلوده به بیماری را به عنوان ورودی میگیرد تا ویژگی هایی مربوط به بافت بیماری را از آن استخراج کند.

```python
    if img.ndim == 2:  # ndim: number of dimensions
        gray_img = img
    elif img.ndim == 3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        raise ValueError("Invalid image format")
```

بررسی ابعاد تصویر با هدف اطمینان حاصل کردن از اینکه تصویر ورودی به فرمت خاکستری باشد. (این امر برای استخراج ویژگی های آتی ضروری است)

img.ndim ابعاد تصویر است که اگر برابر 2 باشد به معنی این است که تصویر از قبل به صورت خاکستری است و فرمت آن قابل قبول است. اگر مقدار آن برابر 3 باشد یعنی تصویر به صورت رنگی است ، بنابراین با استفاده از cv2.cvtColor به خاکستری تبدیل می‌شود. اگر تصویر فرمت نامعتبری داشته باشد (نه 2 بعدی و نه 3 بعدی)، خطایی ایجاد می‌شود.

```
    # Extract GLCM features
    # the GLCM matrix will be symmetric (ignoring the order of value pairs)
    # levels=256: for an 8-bit input image (number of gray-levels)
    glcm = graycomatrix(gray_img, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)
```

استخراج ماتریس GLCM از تصویر خاکستری

distances=[1]: فاصله پیکسل‌هایی که برای محاسبه هم‌پوشانی در نظر گرفته می‌شوند. (در این جا پیکسل های مجاور در نظر گرفته میشوند.)

angles=[0]: زاویه‌ای که برای محاسبه هم‌پوشانی استفاده می‌شود. (در اینجا 0 درجه)

levels=256: تعداد سطوح خاکستری برای تصویر ورودی 8 بیتی.

symmetric=True: ماتریس GLCM متقارن باشد. (ترتیب زوج مقدار های i و j مهم نباشد)

normed=True: ماتریس GLCM نرمالایز شود.

ورودی: یک تصویر در فرمت خاکستری

خروجی: ماتریس GLCM که یک آرایه 4 بعدی به ابعاد (len(distances) x len(angles) x height x width) است.

```
    #  Measures the intensity contrast
    contrast = graycoprops(glcm, 'contrast')[0]  # The function returns a 2D array (one row per distance-angle pair)
    # similar to contrast but weighs differences linearly rather than quadratically
    dissimilarity = graycoprops(glcm, 'dissimilarity')[0]
    # Higher values indicate more homogeneity
    homogeneity = graycoprops(glcm, 'homogeneity')[0]
    # Higher values indicate more uniformity
    energy = graycoprops(glcm, 'energy')[0]
    # ranges between -1 and 1. Higher values indicate a positive correlation
    correlation = graycoprops(glcm, 'correlation')[0]
    # indicates image texture uniformity
    asm = graycoprops(glcm, 'ASM')[0]
```

محاسبه ویژگی‌های مختلف GLCM از ماتریس GLCM

تابع graycoprops یک آرایه دو بعدی برمیگرداند که هر سطر آن متناظر با یک جفت از فواصل و زوایای ورودی در مرحله قبل هستند. از آن جا که ما فقط از یک جفت زاویه و فاصله استفاده کرده ایم، آرایه ما تنها یک سطر خواهد داشت و همان را هم با اندیس 0 انتخاب میکنیم.

Contrast: میزان تضاد شدت نور

Dissimilarity: تفاوت شدت‌ها به صورت خطی

Homogeneity: همگنی تصویر

Energy: یکنواختی تصویر

Correlation: همبستگی بین شدت پیکسل‌ها

Asm: یکنواختی بافت تصویر

```
35        lbp = local_binary_pattern(gray_img, P=8, R=1, method='uniform')
```

استخراج LBP از تصویر خاکستری

P=8: تعداد نقاط در دایره همسایگی که برای محاسبه الگوی باینری محلی استفاده می‌شوند.

R=1: شعاع دایره همسایگی

method='uniform': استفاده از روش یکنواخت LBP که مقاوم به چرخش است (00000010 را مانند 00000001 میداند) و تعداد الگوی کمتری ایجاد می‌کند. (9 الگوی یکنواخت شامل 8 الگوی با 2 تغییر و 1 الگوی با 0 تغییر و 1 الگوی غیر یکنواخت به نمایندگی از همه الگوهای غیر یکنواخت)

ورودی: یک تصویر در فرمت خاکستری

خروجی: یک آرایه دو بعدی که هر پیکسل را به یک مقدار LBP متناظر تبدیل می‌کند.

```
39          lbp_hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, 11), range=(0, 10))
```

ایجاد هیستوگرام از مقادیر LBP برای توصیف توزیع الگوهای باینری محلی در تصویر

lbp.ravel: تبدیل آرایه دو بعدی lbp به یک آرایه یک بعدی. این کار برای ساخت هیستوگرام از تمامی مقادیر LBP در تصویر انجام می‌شود.

bins=np.arange(0, 11): تعریف تعداد بین‌ها (10 بین برای مقادیر 0 تا 9).

در واقع بین ها به این شکل خواهند بود: 0 1 2 3 ..... 8 9 10

range=(0, 10): محدوده مقادیرLBP (0 مینیمم مقدار تا 9 ماکزیمم مقدار).

ورودی: lbp که به صورت یک بعدی تبدیل شده است.

خروجی: lbp_hist که تعداد مقادیر LBP را در هر بین شمارش می‌کند.

```
40          lbp_hist = lbp_hist.astype("float")  # This is necessary for normalization
41          lbp_hist /= lbp_hist.sum()  # Normalize the histogram
```

نرمالیزه کردن هیستوگرام به طوری که مجموع مقادیر آن برابر با 1 باشد.

1- تبدیل مقادیر هیستوگرام به نوع داده‌ای float برای انجام محاسبات اعشاری.

2- تقسیم هر مقدار در هیستوگرام به مجموع مقادیر هیستوگرام، که منجر به نرمالیزه شدن مقادیر می‌شود. این کار باعث می‌شود که مقادیر هیستوگرام به صورت نسبی و به شکل توزیع احتمال باشند.

```
43          # Combine features into a single feature vector
44          features = np.hstack([contrast, dissimilarity, homogeneity, energy, correlation, asm, lbp_hist])
45          return features
```

ترکیب تمامی ویژگی‌های استخراج شده به یک بردار ویژگی واحد (یک آرایه یک بعدی)

ورودی: تمام ویژگی های استخراج شده به صورت آرایه های یک بعدی.

خروجی: یک آرایه بک بعدی که همان بردار ویژگی است.

## ***آموزش مدل و ارزیابی:***

```
import numpy as np
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import joblib
from sklearn.metrics import accuracy_score
from preprocessing import directories, load_images_from_directories, preprocess_image
from segmentation import segment_image, select_infected_segment
from feature_extraction import extract_features
```

وارد کردن کتابخانه ها و ماژول های مورد نیاز:

1- کتابخانه NumPy را وارد می‌کنیم که ابزارهای مختلفی برای کار با آرایه‌ها و ماتریس‌های چند بعدی و انجام عملیات ریاضیاتی سریع و کارا ارائه می‌دهد.

2- وارد کردن Support Vector Classifier (SVC) از کتابخانه Scikit-Learn برای ایجاد و استفاده از مدل ماشین بردار پشتیبان

3- وارد کردن ابزارهای پیش‌پردازش از Scikit-Learn

LabelEncoder: برای تبدیل برچسب‌های متنی به عددی.

StandardScaler: برای نرمال‌سازی ویژگی‌ها.

4- وارد کردن ابزارهای تقسیم داده‌ها از Scikit-Learn برای تقسیم داده‌ها به مجموعه‌های آموزش و تست.

5- وارد کردن joblib برای ذخیره مدل های آموزش دیده

6- وارد کردن ابزارهای ارزیابی مدل از Scikit-Learn برای محاسبه دقت مدل

7- وارد کردن توابع و ماژول‌های مربوط به پیش‌پردازش تصویر

directories: مسیرهای مربوط به دایرکتوری‌های حاوی تصاویر.

load_images_from_directories: تابعی برای بارگذاری تصاویر از دایرکتوری‌ها.

preprocess_image: تابعی برای پیش‌پردازش تصاویر.

8- وارد کردن توابع و ماژول‌های مربوط به بخش‌بندی تصویر برای انجام بخش‌بندی تصاویر و انتخاب بخش آلوده‌ی تصویر.

9- وارد کردن توابع و ماژول‌های مربوط به استخراج ویژگی‌ها برای استخراج ویژگی‌های بافت تصویر.

```
13  def prepare_date(features, labels):
14      # Prepare data for SVM classification
15      X = np.array(features)
16      y = np.array(labels)
```

ویژگی‌های استخراج شده از تصاویرو برچسب‌های مرتبط با تصاویر را به یک آرایه NumPy تبدیل می‌کند. (نوع داده قبلی آن ها لیست است)

```
18      # Encode labels
19      le = LabelEncoder()
20      y_encoded = le.fit_transform(y)
```

یک شیء LabelEncoder ایجاد می‌کند و برچسب‌های متنی را به برچسب‌های عددی تبدیل می‌کند.

```
22  # Split the data into training and testing sets
23  # random_state: pass an int for reproducible output across multiple function calls
24  X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,test_size=0.2, random_state=42)
```

داده‌ها را به دو مجموعه‌ی آموزش و تست تقسیم می‌کند. (20 درصد داده ها به عنوان داده های تست انتخاب میشوند)

random_state=42: یک مقدار تصادفی ثابت برای تکرارپذیری نتایج تعیین می‌کند.

```
26      # standardize the feature data ---> SVM performs better
27      scaler = StandardScaler()
28      X_train = scaler.fit_transform(X_train)
29      X_test = scaler.transform(X_test)
```

یک شیء StandardScaler ایجاد می‌کند سپس داده‌های آموزشی را نرمال‌سازی می‌کند و مقیاس‌گذاری می‌کند و پس از آن داده‌های تست را با استفاده از مقیاس‌گذاری انجام شده بر روی داده‌های آموزشی، نرمال‌سازی می‌کند.

```
31      # save the scaler
32      joblib.dump(scaler, 'scaler.joblib')
33
34      return X_train, X_test, y_train, y_test, le
```

شیء StandardScaler را ذخیره می‌کند تا در آینده بتوان از آن برای مقیاس‌گذاری داده های جدید استفاده کرد.

```
34      return X_train, X_test, y_train, y_test, le
```

داده‌های آموزشی و تست، برچسب‌های آموزشی و تست، و شیء LabelEncoder را بازمی‌گرداند تا بتوان در مراحل بعدی از آنها استفاده کرد.

به طور کلی این تابع داده‌های ویژگی و برچسب‌ها را برای استفاده در یک مدل SVM آماده می‌کند. مراحل شامل تبدیل داده‌ها به آرایه‌هایNumPy ، کدگذاری برچسب‌ها، تقسیم داده‌ها به مجموعه‌های آموزش و تست، نرمال‌سازی داده‌ها و ذخیره مقیاس‌کننده است. در نهایت، داده‌های آماده شده بازگردانده می‌شوند تا در آموزش مدل SVM استفاده شوند.

```
39      # Load images and labels
40      images, labels = load_images_from_directories(directories)
```

تصاویر و برچسب‌های آن‌ها را از دایرکتوری‌های مشخص شده بارگذاری می‌کند. خروجی این تابع دو لیست است: یکی شامل تصاویر و دیگری شامل برچسب‌های مرتبط با هر تصویر.

```
42      # Preprocess all images
43      preprocessed_images = [preprocess_image(img) for img in images]
```

یک لیست از تصاویر پیش‌پردازش شده ایجاد می‌کند. (برای هر تصویر در لیست images، تابع preprocess_image فراخوانی می‌شود.)

```
45      # Segmentation
46      infected_segments = []
47      for i, img in enumerate(preprocessed_images):
48          segmented_img, kmeans = segment_image(img)
49          infected_segment, infected_mask = select_infected_segment(img, labels[i], kmeans)
50          infected_segments.append(infected_segment)
```

ابتدا یک لیست خالی برای ذخیره کردن بخش‌های آلوده هر تصویر ایجاد می‌کند.

for i, img in enumerate(preprocessed_images): یک حلقه for برای پردازش هر تصویر پیش‌پردازش شده.

segmented_img, kmeans = segment_image(img): تصویر را به بخش‌های مختلف تقسیم کرده و مدل k-means را به دست می‌آورد.

infected_segment, infected_mask = select_infected_segment(img, labels[i], kmeans):

بخش آلوده تصویر را بر اساس برچسب مربوطه و مدل k-means انتخاب می‌کند.

در نهایت بخش آلوده استخراج شده را به لیست infected_segments اضافه می‌کند.

```
52      # Extract features for all images
53      features = [extract_features(img) for img in infected_segments]  # a list of np arrays
```

ویژگی‌ها را از بخش‌های آلوده تمامی تصاویر استخراج می‌کند و آن‌ها را در یک لیست ذخیره می‌کند. هر ورودی در این لیست، یک آرایه NumPy است که ویژگی‌های مربوط به یک تصویر را نشان می‌دهد.

```
55      X_train, X_test, y_train, y_test, le = prepare_date(features, labels)
```

داده‌های ویژگی و برچسب‌ها را برای استفاده در یک مدل SVM آماده می‌کند و در نهایت داده های آماده شده و شیء LabelEncoder را بازمی‌گرداند.

```
57      # Train the SVM classifier
58      # linear kernel  when the data is linearly separable (low cost)
59      svm = SVC(kernel='linear')
60      svm.fit(X_train, y_train)
```

مدل SVM را آموزش می‌دهد. در ابتدا یک شیء مدل SVM با کرنل خطی ایجاد می‌کند. کرنل خطی زمانی استفاده می‌شود که داده‌ها به صورت خطی قابل جدا شدن باشند و هزینه محاسباتی پایینی دارد. سپس مدل SVM را با استفاده از داده‌های آموزشی آموزش میدهد.

```
62      # Save the trained SVM model
63      joblib.dump(svm, 'trained_svm_model.joblib')
64      print("Model saved as 'trained_svm_model.joblib'")
```

مدل SVM آموزش دیده را در یک فایل با نام trained_svm_model.joblib ذخیره می‌کند.

```
66      # Predict and evaluate the model
67      y_predicted = svm.predict(X_test)
```

برچسب‌های پیش‌بینی شده برای داده‌های تست را با استفاده از مدل SVM آموزش دیده محاسبه می‌کند و آن‌ها را در y_predicted ذخیره می‌کند.

```
69      # Display accuracy
70      accuracy = accuracy_score(y_test, y_predicted)
71      print(f'Accuracy: {accuracy * 100:.2f}%')
```

دقت مدل را با مقایسه برچسب‌های واقعی و برچسب‌های پیش‌بینی شده محاسبه می‌کند و نتیجه را در accuracy ذخیره می‌کند و نمایش میدهد.

## ***طبقه بندی کردن یک تصویر نمونه و دیدن خروجی مدل:***

وارد کردن کتابخانه ها و ماژول های مورد نیاز:

```
from preprocessing import directories, load_images_from_directories, preprocess_image
from segmentation import segment_image, select_infected_segment
from feature_extraction import extract_features
from training import prepare_date
import joblib, cv2
```

1- وارد کردن توابع و ماژول‌های مربوط به پیش‌پردازش تصویر

directories: مسیرهای مربوط به دایرکتوری‌های حاوی تصاویر.

load_images_from_directories: تابعی برای بارگذاری تصاویر از دایرکتوری‌ها.

preprocess_image: تابعی برای پیش‌پردازش تصاویر.

2- وارد کردن توابع و ماژول‌های مربوط به بخش‌بندی تصویر برای انجام بخش‌بندی تصاویر و انتخاب بخش آلوده‌ی تصویر.

3- وارد کردن توابع و ماژول‌های مربوط به استخراج ویژگی‌ها برای استخراج ویژگی‌های بافت تصویر.

4- وارد کردن تابع لازم جهت آماده‌سازی داده‌ها برای آموزش مدل SVM ، شامل تقسیم داده‌ها به بخش‌های آموزشی و تست، استانداردسازی داده‌ها و کدگذاری برچسب‌ها.

5- وارد کردن joblib برای بارگذاری مدل های آموزش دیده و cv2 برای نمایش تصاویر

```
# display the disease type
def display_disease_info(img,label, model, scaler, label_encoder):
```

این تابع برای نمایش نوع بیماری استفاده می‌شود.

پارامتر های ورودی: تصویر ورودی، برچسب بیماری برای تصویر ورودی، مدل آموزش دیده SVM، استانداردساز داده‌ها، کدگذار برچسب‌ها

```
10      processed_img = preprocess_image(img)
11      segmented_img, kmeans = segment_image(processed_img)
12      infected_segment, _ = select_infected_segment(img, label, kmeans)
13      features = extract_features(infected_segment)
```

1- پیش‌پردازش تصویر

2- تقسیم‌بندی تصویر

3- انتخاب بخش آلوده تصویر

4- استخراج ویژگی‌های GLCM و LBP از بخش آلوده‌ی تصویر

```
14      features = scaler.transform(features.reshape(1, -1))  # should be passed as a 2D array
```

ویژگی‌های استخراج شده با استفاده از استانداردساز (scaler) استانداردسازی می‌شوند.
ویژگی‌ها باید به صورت یک آرایه‌ی دو بعدی به استانداردساز داده شوند.

```
15          prediction = model.predict(features)
```

پیش بینی نوع بیماری: مدل آموزش دیده SVM ، نوع بیماری را بر اساس ویژگی‌های استانداردسازی شده پیش‌بینی می‌کند.

```
16      disease_type = label_encoder.inverse_transform(prediction)[0]  # returns a list
```

برچسب پیش‌بینی شده با استفاده از کدگذار برچسب‌ها (label_encoder) به نام بیماری تبدیل می‌شود.

```
18          print(f'Disease Type: {disease_type}')
19          cv2.imshow(f'{disease_type}', img)
20          cv2.waitKey(0)
21          cv2.destroyAllWindows()
```

نمایش تصویر و نام بیماری پیش بینی شده

```
26      # Load images and labels
27      images, labels = load_images_from_directories(directories)
28
29      # Preprocess all images
30      preprocessed_images = [preprocess_image(img) for img in images]
31
32      # Segmentation
33      infected_segments = []
34      for i, img in enumerate(preprocessed_images):
35          segmented_img, kmeans = segment_image(img)
36          infected_segment, infected_mask = select_infected_segment(img, labels[i], kmeans)
37          infected_segments.append(infected_segment)
38
39      # Extract features for all images
40      features = [extract_features(img) for img in infected_segments]  # a list of np arrays
41
42      # prepare data for SVM
43      X_train, X_test, y_train, y_test, le = prepare_date(features, labels)
```

1- بارگذاری تصاویر و برچسب ها

2- پیش پردازش تصاویر

3- بخش بندی تصاویر و انتخاب بخش آلوده هر تصویر

4- استخراج ویژگی های GLCMو LBP از بخش آلوده‌ی تصاویر

5- آماده‌سازی داده‌ها برای مدل SVM ، شامل تقسیم داده‌ها به بخش‌های آموزشی و تست، استانداردسازی داده‌ها و کدگذاری برچسب‌ها

```
45          # load back the model and scaler
46          loaded_scaler = joblib.load('scaler.joblib')
47          loaded_svm = joblib.load('trained_svm_model.joblib')
```

بارگذاری استانداردساز ذخیره شده و مدل SVM آموزش دیده شده

```
49      # Example usage
50      test_image = images[29]
51      label = labels[29]
52      display_disease_info(test_image,label, loaded_svm, loaded_scaler, le)
```

مثال استفاده از تابع display_disease_info

در این مثال، یکی از تصاویر و برچسب مرتبط با آن انتخاب و سپس اطلاعات بیماری برای آن تصویر نمایش داده می‌شود.

تابع display_disease_info را با پارامترهای زیر فراخوانی میشود:

تصویری که می‌خواهیم اطلاعات بیماری آن را نمایش دهیم

برچسب مرتبط با تصویر تست

مدل SVM که قبلاً آموزش دیده و بارگذاری شده است

استانداردسازی که قبلاً آموزش دیده و بارگذاری شده است

کدگذار برچسب‌ها که قبلاً آموزش دیده

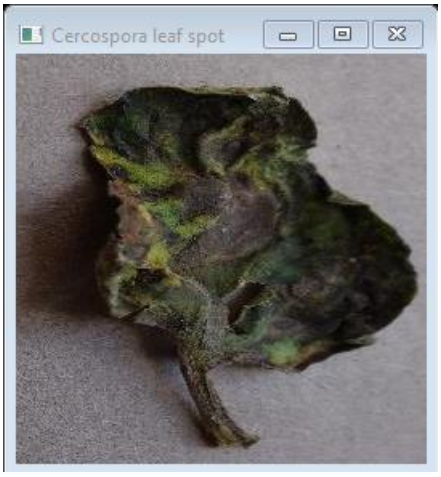
این کد به ما اجازه می‌دهد تا نحوه عملکرد مدل و فرآیند تشخیص بیماری را برای یک تصویر خاص مشاهده کنیم.

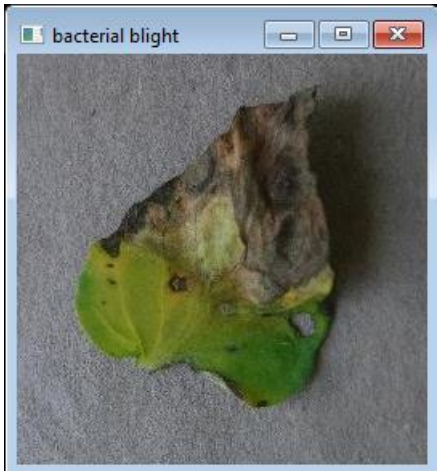
تصویر خروجی در زیر آمده است:


```
D:\plant-disease\venv\Scripts\python.exe D:/plant-disease/codes/predict.py
Disease Type: powdery mildew

Process finished with exit code 0
```

مثال از تصاویری دیگر:


## جمع بندی:

پردازش تصویر نقش حیاتی در تشخیص بیماری‌های گیاهان ایفا می‌کند. این تکنولوژی نه تنها به کشاورزان و متخصصان کمک می‌کند تا به سرعت و دقت بالا بیماری‌های گیاهان را تشخیص دهند، بلکه می‌تواند منجر به کاهش استفاده نادرست از سموم و بهبود کیفیت محصولات کشاورزی شود.

سیستم ارائه‌شده در این مقاله قادر است با دقت 85.71 درصد چهار گروه از بیماری‌های گیاهان شامل Cercospora leaf spot، powdery mildew، rust و bacterial blight را تشخیص دهد. این دقت نشان‌دهنده پتانسیل بالای سیستم در شناسایی بیماری‌های گیاهی و کمک به مدیریت بهینه مزارع است.

چشم‌انداز آینده ما این است که با بهبود و توسعه بیشتر این سیستم، قادر به تشخیص تعداد بیشتری از گروه‌های بیماری گیاهان باشیم و دقت تشخیص را بهبود بخشیم. این هدف می‌تواند از طریق استفاده از داده‌های بیشتر، الگوریتم‌های پیشرفته‌تر و به‌کارگیری تکنیک‌های یادگیری عمیق تحقق یابد. با تحقق این اهداف، می‌توانیم به ایجاد سیستم‌های هوشمندتر و دقیق‌تر در حوزه کشاورزی دست یابیم که به‌طور موثری به افزایش بهره‌وری و پایداری در این صنعت کمک کنند.

## منابع:

[1] S. Aravind, M. Raja, and S. Arun Pandian, "Bacterial foraging optimization based Radial Basis Function Neural Network (BRBFNN) for identification and classification of plant leaf diseases," *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2017, pp. 1-4, doi: 10.1109/ICCIC.2017.8524224.

[2] A. M. Kumar, R. Y. Lokesh, and R. Roopa, "Plant Leaf Disease Detection and Classification using Image Processing," *International Journal of Engineering Research and Technology (IJERT)*, vol. 6, no. 10, pp. 1-4, 2018.

[3]