Gábor Vecsei

# Plant Seedlings Fun with Computer Vision

54 voters

last run 4 months ago · Python notebook · 3689 views
using data from Plant Seedlings Classification · Public

Tags data visualization data cleaning image processing plants

Notebook

# Plant Seedlings Segmentation with pure Computer Vision

First of all, thanks for the popularity of this kernel. I hope it will help for you to create more accurate predictions

In [1]:

```
%matplotlib inline
import os
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import numpy as np
from glob import glob
import seaborn as sns
```

In [2]:

```
BASE_DATA_FOLDER = "../input"
TRAin_DATA_FOLDER = os.path.join(BASE_DATA_FOLDER, "train")
```

## Read images

First, I'll just read all the images. The images are in BGR (Blue/Green/Red) format because OpenCV uses this.

Btw... If you'd like to use RGB format, than you can use it, it won't effect the segmentation because we will use the HSV (Hue/Saturation/Value) color space for that.

In [3]:

```
images_per_class = {}
for class_folder_name in os.listdir(TRAin_DATA_FOLDER):
    class_folder_path = os.path.join(TRAin_DATA_FOLDER, class_folder_name)
    class_label = class_folder_name
    images_per_class[class_label] = []
    for image_path in glob(os.path.join(class_folder_path, "*.png")):
        image_bgr = cv2.imread(image_path, cv2.IMREAD_COLOR)
        images_per_class[class_label].append(image_bgr)
```

## Number of images per class

In [4]:

```
for key,value in images_per_class.items():
    print("{0} -> {1}".format(key, len(value)))
```

```
Common wheat -> 221
Black-grass -> 263
Charlock -> 390
Scentless Mayweed -> 516
Maize -> 221
Cleavers -> 287
Loose Silky-bent -> 654
Small-flowered Cranesbill -> 496
Shepherds Purse -> 231
Sugar beet -> 385
Common Chickweed -> 611
Fat Hen -> 475
```

## Plot images

Plot images so we can see what the input looks like

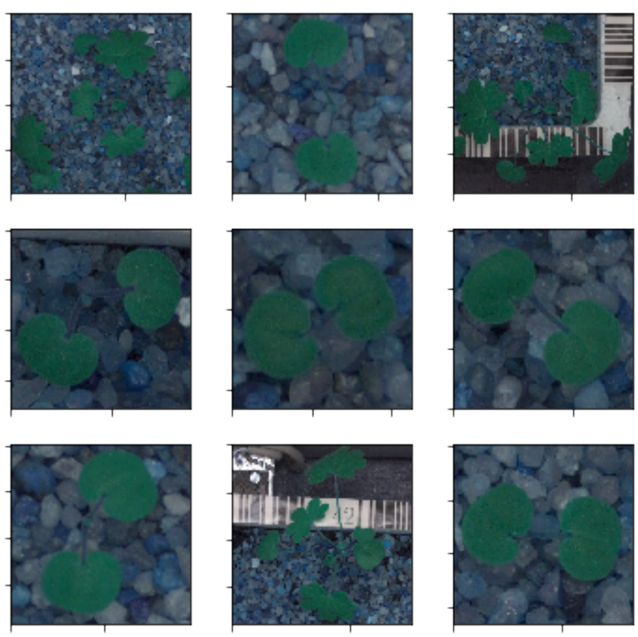
In [5]:

```
def plot_for_class(label):
    nb_rows = 3
    nb_cols = 3
    fig, axs = plt.subplots(nb_rows, nb_cols, figsize=(6, 6))

    n = 0
    for i in range(0, nb_rows):
        for j in range(0, nb_cols):
            axs[i, j].xaxis.set_ticklabels([])
            axs[i, j].yaxis.set_ticklabels([])
            axs[i, j].imshow(images_per_class[label][n])
            n += 1
```

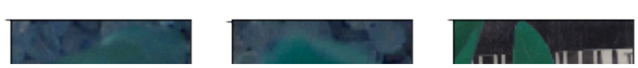
In [6]:

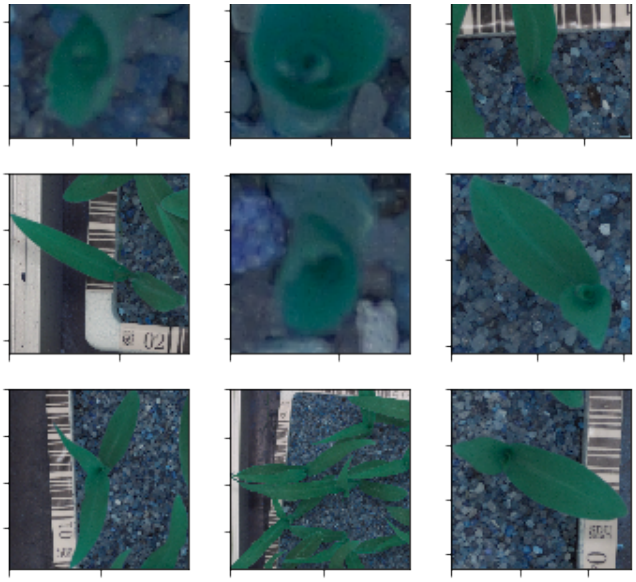
```
plot_for_class("Small-flowered Cranesbill")
```



In [7]:

```
plot_for_class("Maize")
```





### Preprocessing for the images:

Now comes the interesting and fun part!

I created separate functions so if you'd like to use these it is easier.

In the next block I'll explain what I am doing to make the segmentation happen.

In [8]:

```
def create_mask_for_plant(image):
    image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

Notebook Code Data (1) Comments (19) Log Versions (6) Forks (11)

```
    lower_hsv = np.array([60 - sensitivity, 100, 50])
    upper_hsv = np.array([60 + sensitivity, 255, 255])

    mask = cv2.inRange(image_hsv, lower_hsv, upper_hsv)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11,11))
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    return mask

def segment_plant(image):
    mask = create_mask_for_plant(image)
    output = cv2.bitwise_and(image, image, mask = mask)
    return output
```

```
def sharpen_image(image):
    image_blurred = cv2.GaussianBlur(image, (0, 0), 3)
    image_sharp = cv2.addWeighted(image, 1.5, image_blurred, -0.5, 0)
    return image_sharp
```

The `create_mask_for_plant` function: This function returns an image mask: Matrix with shape `(image_height, image_width)`. In this matrix there are only `0` and `1` values. The 1 values define the interesting part of the original image. But the question is...How do we create this mask?

This is a simple object detection problem, where we can use the color of the object.

The HSV color-space is suitable for color detection because with the Hue we can define the color and the saturation and value will define "different kinds" of the color. (For example it will detect the red, darker red, lighter red too). We cannot do this with the original BGR color space.

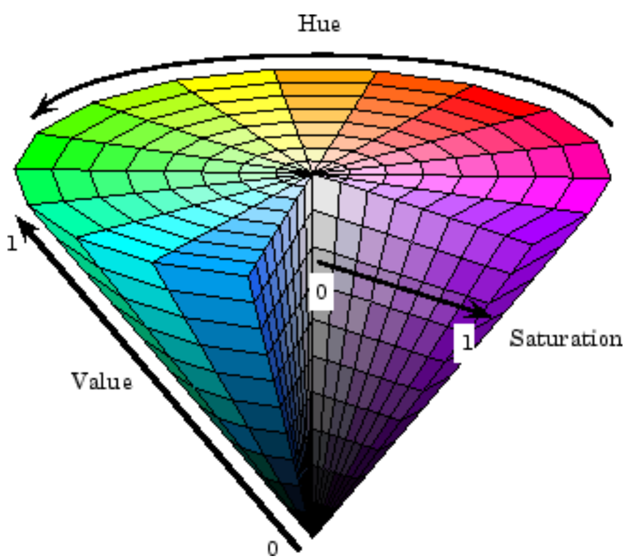



*image from https://www.mathworks.com/help/images/convert-from-hsv-to-rgb-color-space.html (https://www.mathworks.com/help/images/convert-from-hsv-to-rgb-color-space.html)*

We have to set a range, which color should be detected:

```
sensitivity = 35
lower_hsv = np.array([60 - sensitivity, 100, 50])
upper_hsv = np.array([60 + sensitivity, 255, 255])
```

After the mask is created with the `inRange` function, we can do a little *CV magic* (not close to magic, because this is almost the most basic thing in CV, but it is a cool buzzword, and this opertation is as awesome as simple it is) which is called *morphological operations* (You can read more here
(https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm))

(https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)).

Basically with the *Close* operation we would like to keep the shape of the original objects (1 blobs on the mask image) but close the small holes. That way we can clarify our detection mask more.

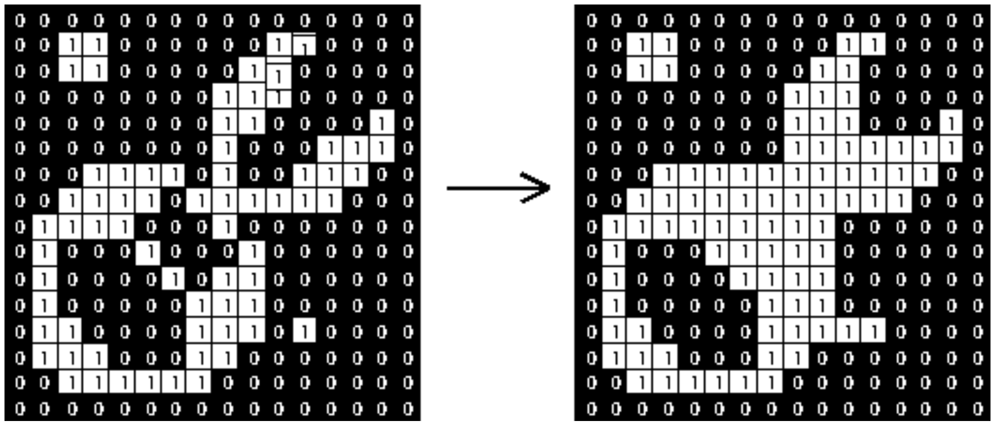


*image from https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm (https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm)*

After these steps we created the mask for the object.

In [9]:

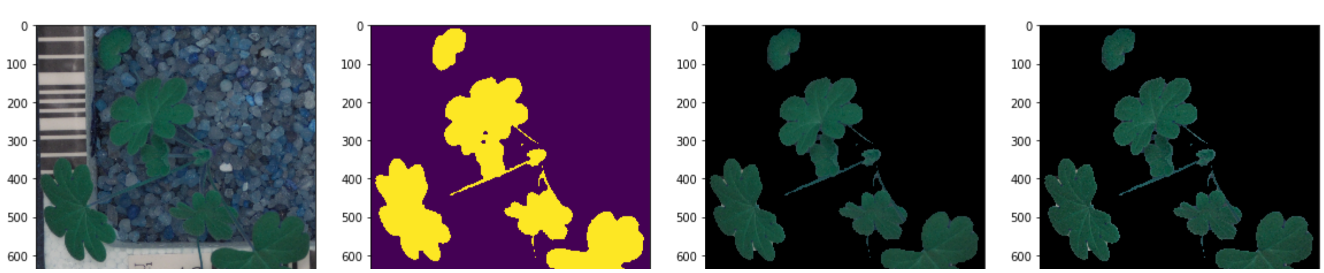
```
# Test image to see the changes
image = images_per_class["Small-flowered Cranesbill"][97]

image_mask = create_mask_for_plant(image)
image_segmented = segment_plant(image)
image_sharpen = sharpen_image(image_segmented)

fig, axs = plt.subplots(1, 4, figsize=(20, 20))
axs[0].imshow(image)
axs[1].imshow(image_mask)
axs[2].imshow(image_segmented)
axs[3].imshow(image_sharpen)
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x7f204e735198>
```



After this step we can see that the image on the right is more recognizable than the original image on the left.

From the mask image what we created (because we need that for the segmentation), we can extract some features. For example we can see how the area of the plant changes based on their classes.

Of course from the contours we can extract much more information than the area of the contour and the number of components, but this is the one I would like to show you.

Additional read: https://en.wikipedia.org/wiki/Image_moment (https://en.wikipedia.org/wiki/Image_moment)

In [10]:

```
def find_contours(mask_image):
    return cv2.findContours(mask_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPL
E)[-2]

def calculate_largest_contour_area(contours):
    if len(contours) == 0:
        return 0
    c = max(contours, key=cv2.contourArea)
    return cv2.contourArea(c)

def calculate_contours_area(contours, min_contour_area = 250):
    area = 0
    for c in contours:
        c_area = cv2.contourArea(c)
        if c_area >= min_contour_area:
            area += c_area
    return area
```

In [11]:

```
areas = []
```

```
larges_contour_areas = []
labels = []
nb_of_contours = []
images_height = []
images_width = []

for class_label in images_per_class.keys():
    for image in images_per_class[class_label]:
        mask = create_mask_for_plant(image)
        contours = find_contours(mask)

        area = calculate_contours_area(contours)
        largest_area = calculate_largest_contour_area(contours)
        height, width, channels = image.shape

        images_height.append(height)
        images_width.append(width)
        areas.append(area)
        nb_of_contours.append(len(contours))
        larges_contour_areas.append(largest_area)
        labels.append(class_label)
```

In [12]:

```
features_df = pd.DataFrame()
features_df["label"] = labels
features_df["area"] = areas
features_df["largest_area"] = larges_contour_areas
features_df["number_of_components"] = nb_of_contours
features_df["height"] = images_height
features_df["width"] = images_width
```

In [13]:

```
features_df.groupby("label").describe()
```

Out[13]:

| | area | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | count | mean | std | min | 25% | 50% | 75% | max |
| label | | | | | | | | |
| Black-grass | 263.0 | 41793.982890 | 159636.952451 | 0.0 | 1559.750 | 7204.00 | 23556.250 | 2097133.5 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Charlock | 390.0 | 91669.094872 | 150626.968468 | 3177.5 | 12795.125 | 39382.25 | 80786.125 | 1193936.5 |
| Cleavers | 287.0 | 25619.073171 | 24726.375631 | 567.5 | 8368.000 | 18717.50 | 34363.000 | 175187.0 |
| Common Chickweed | 611.0 | 14935.878887 | 20172.494215 | 0.0 | 1917.500 | 5026.00 | 25140.000 | 131654.5 |
| Common wheat | 221.0 | 6949.357466 | 8161.181143 | 0.0 | 1883.000 | 4044.50 | 8953.000 | 57880.0 |
| Fat Hen | 475.0 | 30623.629474 | 67760.225738 | 512.0 | 2804.250 | 6153.50 | 28209.000 | 561161.5 |
| Loose Silky-bent | 654.0 | 24618.366208 | 132431.879535 | 0.0 | 718.500 | 1795.50 | 8252.500 | 1930040.5 |
| Maize | 221.0 | 115311.341629 | 206882.627747 | 851.5 | 3885.500 | 52033.00 | 119356.000 | 1435536.5 |
| Scentless Mayweed | 516.0 | 15517.381783 | 27678.754205 | 0.0 | 1589.250 | 3314.25 | 14044.125 | 198328.5 |
| Shepherds Purse | 231.0 | 38187.586580 | 66440.811037 | 562.0 | 3303.500 | 7435.50 | 45623.250 | 415662.5 |
| Small-flowered Cranesbill | 496.0 | 38126.337702 | 48735.741208 | 678.5 | 7834.625 | 16201.50 | 42556.500 | 306815.5 |
| Sugar beet | 385.0 | 79003.370130 | 112091.057669 | 0.0 | 16029.000 | 40748.50 | 81174.000 | 815699.0 |

12 rows × 40 columns

In [14]:

```
```

**Did you find this Kernel useful?**
Show your appreciation with an upvote

54

Comments (19) All Comments Sort by Hotness

Click here to enter a comment...

leejet • Posted on Version 5 • 4 months ago • Options • Reply 2

Actually, cv2.imread() will read the image in BGR mode. So, you read the image in BGR mode and then convert it to RGB mode. That's why you could show the normal image in pyplot which deal with image in RGB mode.

ABHISHEKDOG... • Posted on Version 3 • 5 months ago • Options • Reply 0

This is a good step to highlight only the green areas for recognition. It will speed up the computation and be more accurate. Thanks

Gábor Vecsei • Posted on Version 4 • 4 months ago • Options • Reply 1

Thanks! Btw, it does not speed up the process (because you have to convert the image from one color space to the other) but it does make your model more accurate because it doesn't have to deal with the background. By "more accurate" I mean a few decimals, but if you are in the first 10-20 in the contest this preprocessing can mean a lot.

ABHISHEKD... • Posted on Version 5 • 4 months ago • Options • Reply 0

What if we grayscale or binarize the image after we get the green and black areas, I think it might speed up the computation as well?

Gábor Vecsei • Posted on Version 5 • 4 months ago • Options • Reply 1

Yeah, it will, if you create or fine tune a model where the input shape accepts the grayscale images. But (for example) if you'd like to fine-tune Xception than the input shape has to be `(299, 299, 3)` and this means that it doesn't matter if the image is garyscale or not.

Jeru666 • Posted on Version 5 • 4 months ago • Options • Reply 0

thanks for sharing this !!!

**Carson S** • Posted on Version 5 • 4 months ago • Options • Reply 0

Nice post. Do you think you could give a little explanation of what you did and why for the preprocessing step?

**Gábor Vecsei** • Posted on Version 5 • 4 months ago • Options • Reply 0

Of course, Later this day or tomorrow I'll write a more detailed explanation.

**Gábor Vecsei** • Posted on Latest Version • 4 months ago • Options • Reply 0

I don't know if you noticed it, but I wrote a more detailed explanation.

**Kalpana Krishna...** • Posted on Latest Version • 4 months ago • Options • Reply 0

Thanks for sharing, great place to get started

**ArsenyAntonov** • Posted on Latest Version • 4 months ago • Options • Reply 0

Thanks for your kernel. Why did you decide you use sharpen filter here?

**Gábor Vecsei** • Posted on Latest Version • 4 months ago • Options • Reply 2

Because as I saw, the images of the plants are blurry and I thought that that sharpening the segmented image would help bring up more details, so I can do the classification more accurately.

Btw. This comes from a previous experience with emotion detection from face images. I had 48x48 grayscale images there and when I applied a sharpening filter the accuracy increased by a few points for the same model.

**Siddique** • Posted on Latest Version • 2 months ago • Options • Reply 0

Many thanks for the code. It was very nice to see the background noise gone. Improved classification accuracy by 5+%.

**Gábor Vecsei** • Posted on Latest Version • 2 months ago • Options • Reply 0

It is really nice to hear that it helped you. If you've got any questions, don't hesitate to ask!

**Jacken** • Posted on Latest Version • 12 days ago • Options • Reply 0

I suppose how you can improve so a lot. Can you give me some advice?

**Mohammed Alb...** • Posted on Latest Version • 2 months ago • Options • Reply 0

Excellent work and great details. It helped me a lot, many thanks.

I trained two models, one with and the other without the background filters and it improved the results a lot from 0.78 to 0.81 on the same CNN layer. Still trying to get a better score.

**城璐** • Posted on Latest Version • a month ago • Options • Reply 0

Many thanks for the tricks!

**AdityaMishra** • Posted on Latest Version • a month ago • Options • Reply 0

Thanks!! The trick to sharpen the images & removing the background noise was really great!! Do point me to other kernels, if they use similar cool features for preprocessing .

**Jacken** • Posted on Latest Version • 12 days ago • Options • Reply 0

Good job! And I have used this trick by means of deeplabV3+ to get a good segmentation.However, I did not get an obvious improvement as others discussed these. I wonder what tricks you used, like param's value,etc. And I also want to know is this a good idea to other transfer learning task or fine tuning task? Thanks a lot.