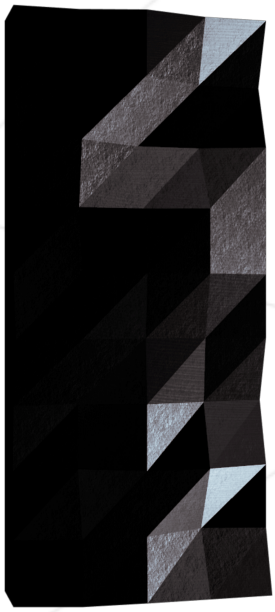


Leaffliction

Computer vision

Summary: Image classification by disease recognition on leaves

Version: 1.00

Contents

Chapter I

Foreword

"No," Arthur said, "look, it's very, very simple.... All I want... is a cup of tea. You are going to make one for me. Now keep quiet and listen."

And he sat. He told the Nutro-Matic about India, he told it about China, he told it about Ceylon. He told it about broad leaves drying in the sun. He told it about silver teapots. He told it about summer afternoons on the lawn. He told it about putting the milk in before the tea so it wouldn't get scalded. He even told it (briefly) about the East India Trading Company.

"So that's it, is it?" said the Nutro-Matic when he had finished.

"Yes," said Arthur. "That is what I want."

"You want the taste of dried leaves boiled in water?"

"Er, yes. With milk."

"Squirted out of a cow?"

"Well in a manner of speaking, I suppose..."

"I'm going to need some help with this one."

Douglas Adams, The Restaurant At The End Of The Universe

Chapter II

Introduction

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g. in the forms of decisions.

Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that make sense to thought processes and can elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

Chapter III

General rules

- You have to render your modules from a computer in the cluster either using a virtual machine:
  - You can choose the operating system to use for your virtual machine
  - Your virtual machine must have all the necessary software to realize your project. This software must be configured and installed.
- Or you can use the computer directly in case the tools are available.
  - Make sure you have the space on your session to install what you need for all the modules (use the goinfre if your campus has it)
  - You must have everything installed before the evaluations
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.
- You are free to use any language of your choice for this project but we advise you to do it in **python** because there are many libraries that can help you, if you do this project in **python**, it must respect the norm.
  - pip install flake8
  - alias norminette_python=flake8
- By Odin, by Thor ! Use your brain !!!

You are free to use any language of your choice for this module.

Chapter IV

Mandatory part

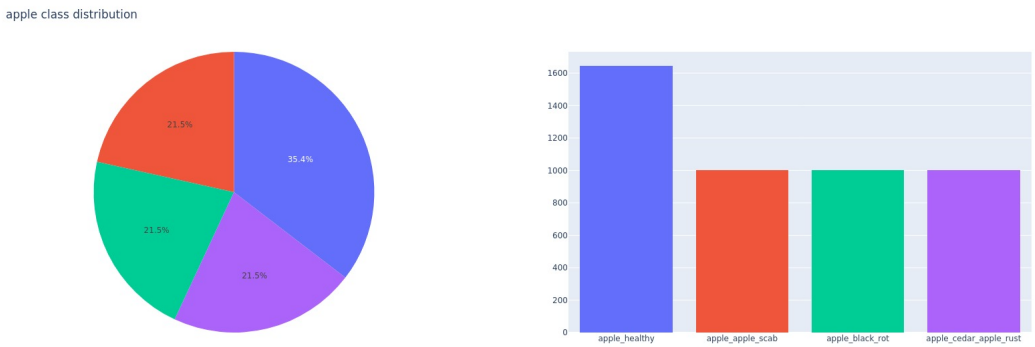
IV.1 Part 1: Analysis of the Data Set

Write a program named Distribution.[extension] that takes as arguments a directory and fetches images in its subdirectories. This program, and therefore you, must then extract and analyze/understand the data set from the images and prompt pie charts and bar charts for each plant type - your program must also retrieve the name of the directory in order to name the chart's colones accordingly.

```
$> find . -maxdepth 2
./Apple
./Apple/apple_healthy
./Apple/apple_apple_scab
./Apple/apple_black_rot
./Apple/apple_cedar_apple_rust
```

```
$> ./Distribution.[extension] ./Apple
```

Example of expected output:



Your program must work with all the directory in the data set

IV.2 Part 2: Data augmentation

You realize that the data set is not balanced.
You have to balance your number of images for each variety and disease.

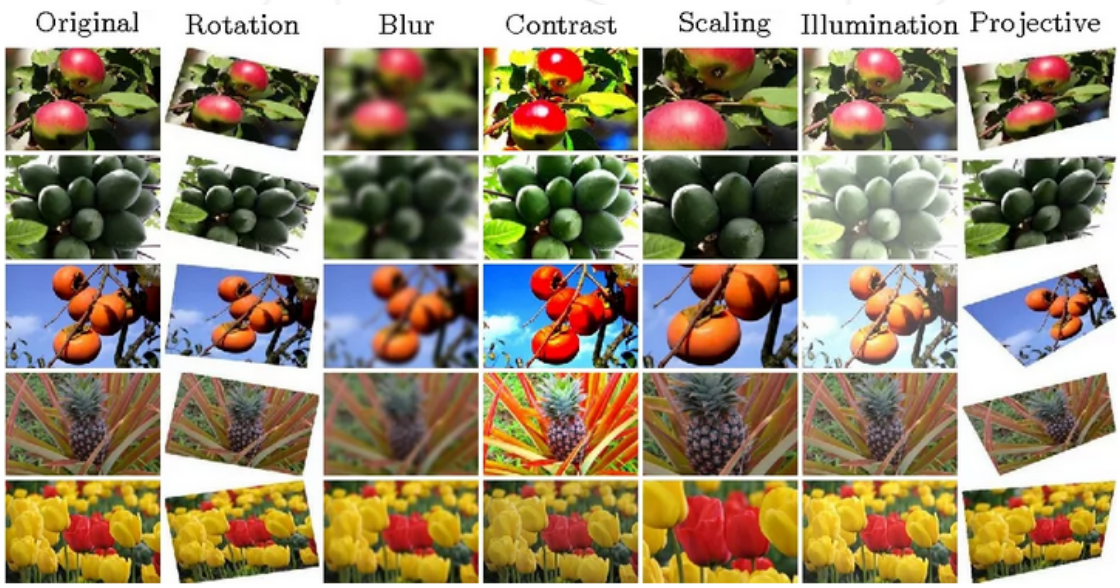
Here is a list of examples of techniques you can use:

- Flip
- Rotate
- Skew
- Shear
- Crop
- Distortion

You must display 6 types of data augmentation for each image given to your program, they must also be saved with the original file name followed by the name of the type of augmentation.

```
$> ./Augmentation.[extension] ./Apple/apple_healthy/image (1).JPG
```

Example of expected output:




```
$> ls
image (1)_Flip.JPG
image (1)_Rotate.JPG
image (1)_Skew.JPG
image (1)_Shear.JPG
image (1)_Crop.JPG
image (1)_Distortion.JPG
```

You have to place the new images in the existing directory to balance your data set. You will need to return this data set in an "augmented_directory" for evaluation.

IV.3 Part 3: Image Transformation

You can use whatever language you want. However, we recommend that you choose a language with a library that facilitates image-processing workflow like plantCV

Different methods of direct extraction of characteristics from an image of a leaf need to be implemented. Once again, you must display at least 6 image transformations.

```
$> ./Transformation.[extension] ./Apple/apple_healthy/image (1).JPG
```

Example of expected output:

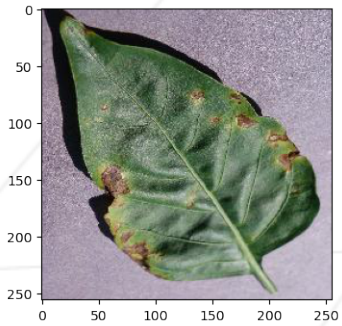


Figure IV.1: Original

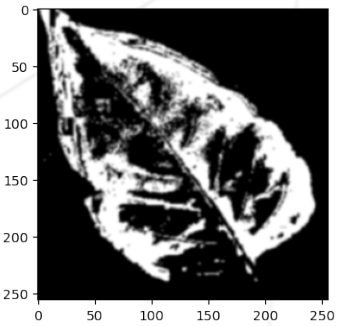


Figure IV.2: Gaussian blur

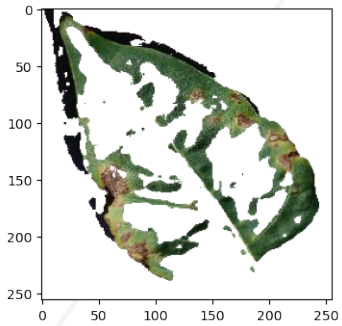


Figure IV.3: Mask

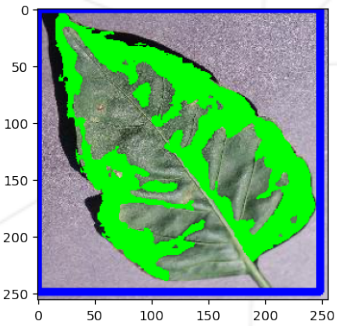


Figure IV.4: Roi objects

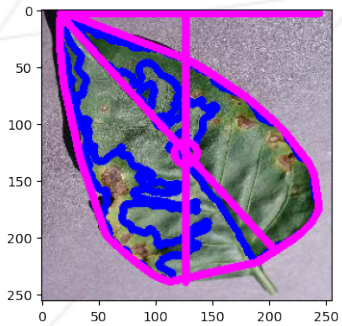


Figure IV.5: Analyze object

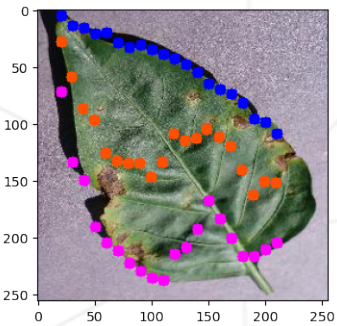


Figure IV.6: Pseudolandmarks

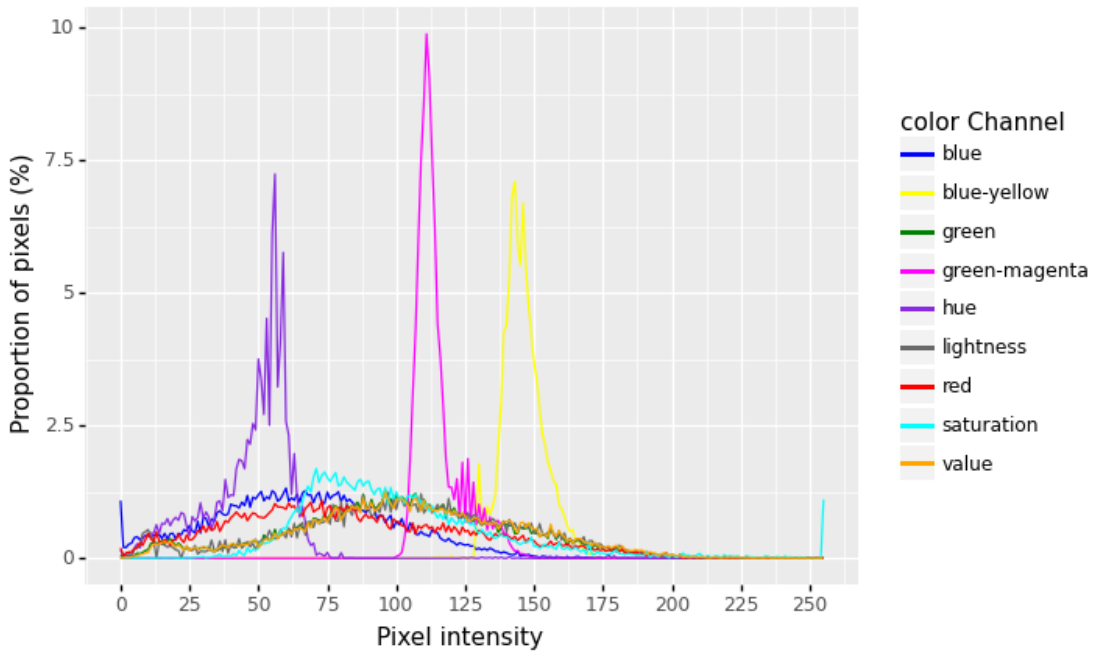


Figure IV.7: Color histogram

If your program is given a direct path to an image, it must display your set of image transformations. However, if it is given a source path to a directory filled with multiple images, it must then save all the image transformations in the specified destination directory.

For example:

```
$> ./Transformation.[extension] -src Apple/apple_healthy/ -dst dst_directory -mask
```

Think to make your own usage to facilitate the choice of the arguments with ./Transformation.[extension] -h

To allow a fast evaluation, it goes without saying that you will create your own data set from the images by following all these steps beforehand. Beware, the evaluation will assess whether your program is working well on small data sets.

IV.4 Part 4: Classification

In this part, you must write a program named train.[extension] that takes as parameter a directory and fetches images in its subdirectories. It must then increase/modify those images in order to learn the characteristics of the diseases specified in the leaf. Those learnings must be saved and returned in a .zip that also includes your increased/modified images.

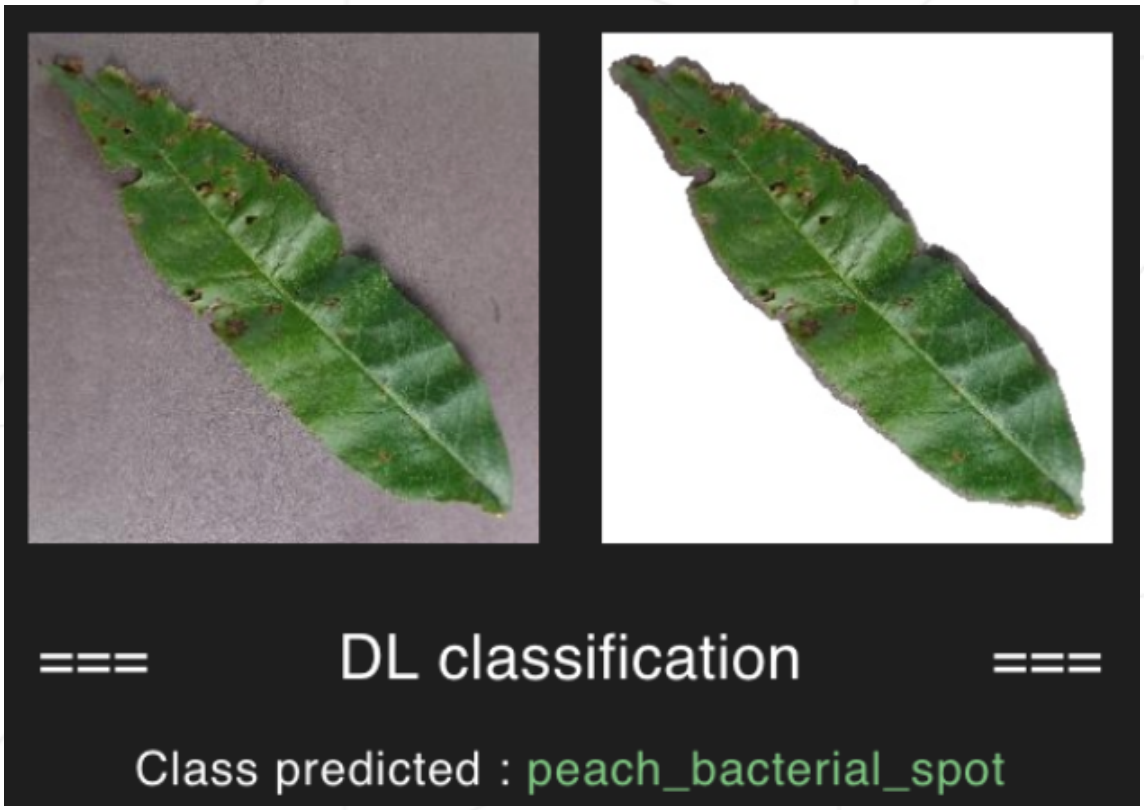
```
$> find . -maxdepth 2
./Apple
./Apple/apple_healthy
./Apple/apple_apple_scab
./Apple/apple_black_rot
./Apple/apple_cedar_apple_rust
```

```
$> ./train.[extension] ./Apple/
```

For evaluation, you will need to write a second program that retrieves your learnings. It must take as arguments a path to an image from your learnings, displays the original image and the transformed image, and that gives the type of disease specified in the leaf.

For example:

```
$> ./predict.[extension] ./Apple/apple_healthy/image (1).JPG
```



You have to separate your data set in two parts, one for Training and one for Validation. The predictions of your validation set must also have an accuracy above 90% (you must be able to prove it in an evaluation, with a minimum of 100 images in the validation set).

Make sure you don't have any knowledge about the validation set or on overfitting prior to the assessment, so your results don't look suspicious.

Chapter V

Turn-in and peer-evaluation

Turn your work in using your `GiT` repository, as usual. Only your programs and the signature.txt must be present in your repository.

Indeed, you must also deposit a signature.txt file at the root of your Git repository in addition to your code. This file must be a hash of your DB which includes everything you need to run your programs correctly, i.e. your dataset in sha1 format and the training of your model.

Then, retrieve the signature from the `".zip"` file of your data set in `sha1` format. Below are 4 command examples for a `directory.zip` file:

- Windows: `certUtil -hashfile directory.zip sha1`
- Linux: `sha1sum directory.zip`
- For Mac M1: `shasum directory.zip`
- MacOS: `shasum directory.zip`

This is an example of what kind of output you will get:

- `7a18a838d2203cc7d6e8c4c521fdd4dd214aa560 directory.zip`

`It is of course FORBIDDEN to turn in your data set in your Git repository. During the defense, the signature of the signature.txt file will be compared with the one of your data set. If the two of them are not identical, your grade will be 0.`