← Go Back to Introduction to Computer Vision

Course Content

FAQ - Plant Seedlings Classification

Q1. How should one approach the Plant Seedlings Classification project?

- Before starting the project, please read the problem statement carefully and go through the criteria and descriptions mentioned in the rubric.
- Once you understand the task, download the dataset and import it into a Jupyter notebook or Google Colab to get started with the project.
- To work on the project, you should start with image preprocessing and image visualization.
- Once data is preprocessed, you can use the data to build a model, check its performance based on the desired metric, and if it is not good then train other neural networks.
- You should include all the models that you have trained in your notebook.
- It is important to close the analysis with key findings and conclusions.

Q2. How to import the dataset on Google Colab?

You can follow the below steps

1) Download the data from the Olympus (images.npy and labels.csv) and upload it into your drive.

2) Mount your google drive using the code below

```
from google.colab import drive
drive.mount('/content/drive')
```

3) Reading the dataset using the example code below

```
import numpy as np
import pandas as pd
```

```
np.load(path+file_name )
pd.read_csv(path+file_name )
```

You should split the data into train and test using the sklearn train_test_split function after the necessary preprocessing.

Q3. How to plot an image from the numpy array?

You can follow the below steps

1) The images.npy has the images converted into arrays and stored row-wise so each index value represents an image.

2) You can use matplotlib's plot function to plot each image from the numpy array as shown below.

plt.imshow(images[i])

Q4. What can be done if the given dataset is imbalanced?

The below code can be used to treat the class imbalance by increasing the weights of the minority classes.

```
from sklearn.utils import class_weight

labelList = Labels.Label.unique()
class_weights = class_weight.compute_class_weight(class_weight = "balanced",
                                                  classes = np.array(labelList),
                                                  y = y_train.values.reshape(-1)
                                                  )
class_weights = dict(zip(np.array(range(len(labelList))), class_weights))
#print calculated class weights
class_weights
```

Q5. How to split the images.npy and labels.csv into train and test?

We can split the data into train and test using the train_test_split function as shown below:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(images,labels , test_size=0.1, random_state=1,stratify=labels)

Q6. I keep getting the below error while executing the fit function for training CNN. Why?

```
ValueError: Shapes (None, 12) and (None, 10) are incompatible
```

This error is due to the incompatible shape of the output layer. The number of units at the output layer should be equal to the number of labels. Change the Dense output shape into 12 because we have 12 labels in this dataset.

```
model.add(Dense(12, activation="softmax"))
```

Q7. What should the test accuracy of the CNN model be and how should I improve it?

You can definitely try to decrease the generalization error till the optimal point. There is no specific target accuracy for any real-time problem it depends on the problem as well as the complexity of the dataset.

You can work on the below option to get better performance.

1. Tune Parameters
2. **Image Data Augmentation (Please try this)**
3. Deeper Network Topology
4. Handle Overfitting and Underfitting problem

Q8. How do you download the HTML file from Google Colab?

Google Colab does not provide an option to download the HTML version of the code. You can download the .ipynb file from Colab and open this downloaded file in the Jupyter notebook. Jupyter provides the option of downloading an HTML version of your code.

Q9. Is it mandatory to use Google Colab for this project?

Yes, it's not mandatory. However, you can try to use either Jupyter Notebook or Google Colab based on your convenience. We prefer using Google Colab for Deep Learning since installing TensorFlow is never an issue.

Q10. I keep getting the below error while running the code on the Jupyter notebook. Why?

```
ModuleNotFoundError: No module named 'google.colab'
```

If you are using a Jupyter notebook, then you need not install google.colab. This library is only used on Google Colab and it comes pre-installed on it.

Q11. Why do images become black after applying Gaussian blurring and normalization?

Please apply the Gaussian blurring first and then normalization.

You should use the appropriate function to display the image.

Actually, cv2_imshow() function plots the image with the 0-1 pixel intensity (since you have normalized the images). When you plot an image with 0-1 pixel intensity, it gives black in color. Therefore, either you can multiply 255 while displaying the image or you can use plt.imshow function which does this denormalization internally and plots the image.

1. Using cv2_imshow()

```
for i in range(5) :
   cv2_imshow(X[i]*255.0)
   #plt.imshow(X[i])
```

2. Using plt.imshow()

```
import pylab as plt
for i in range(5) :
   plt.imshow(X[i]*255.0)
```

Q12. What should be the input shape to the CNN model if the error is caused due to the input shape and size of the image?

The input shape to the input layer of the CNN model should be equal to the size of the image.

For example: if the shape of the image is 128×128×3 then the input shape to the first layer of CNN should be 128×128×3

Q13. What should the order of different layers in a Convolutional Neural Network be? Where should I put my Batch Norm, Dropout, Activation, and Pooling layers? Are there any guidelines regarding the same?

Dropout vs Batch Normalization - Standard deviation issue

There is a big problem that appears when you mix these layers, especially when Batch Normalization is right after Dropout.

Dropouts try to keep the same mean of the outputs without dropouts, but it does change the standard deviation, which will cause a huge difference in the Batch Normalization between training and validation. (During training, the Batch Normalization receives changed standard deviations, and accumulates and stores them. During validation, the dropouts are turned off, the standard deviation is not a changed one anymore, but the original. But Batch Normalization, because it's invalidation, will not use the batch statistics, but the stored statistics, which will be very different from the batch statistics)

So, the first and most important rule is: don't place a Batch Normalization after a Dropout (or a SpatialDropout).

Usually, try to leave at least two convolutional/dense layers without any dropout before applying a batch normalization, to avoid this.

Dropout vs Batch Normalization - Changing the zeros to another value

Also important: the role of the Dropout is to "zero" the influence of some of the weights of the next layer. If you apply a normalization after the dropout, you will not have "zeros" anymore, but a certain value that will be repeated for many units. And this value will vary from batch to batch. So, although there is noise added, you are not killing units as a pure dropout is supposed to do.

Dropout vs MaxPooling

The problem of using a regular Dropout before a MaxPooling is that you will zero some pixels, and then the MaxPooling will take the maximum value, sort of ignoring part of your dropout. If your dropout happens to hit a maximum pixel, then the pooling will result in the second maximum, not in zero.

So, Dropout before MaxPooling reduces the effectiveness of the dropout.

Batch Normalization vs Activation

Depending on the activation function, using a batch normalization before it can be a good advantage.

For a 'ReLU' activation, the normalization makes the model fail-safe against a bad luck case of "all zeros freeze a ReLU layer". It will also tend to guarantee that half of the units will be zero and the other half linear.

For a 'sigmoid' or a 'tanh', the Batch Normalization will guarantee that the values are within a healthy range, avoiding saturation and vanishing gradients (values that are too far from zero will hit an almost flat region of these functions, causing vanishing gradients).

There are people that say there are other advantages if you do the contrary, I'm not fully aware of these advantages, I like the ones I mentioned very much.

Dropout vs Activation

With 'ReLU', there is no difference, it can be proved that the results are exactly the same (Links to an external site.)Links to an external site.

With activations that are not centered, such as 'sigmoid' putting a dropout before the activation will not result in "zeros", but in other values. For a sigmoid, the final results of the dropout before it would be 0.5.

If you add a 'tanh' after a dropout, for instance, you will have the zeros, but the scaling that dropout applies to keep the same mean will be distorted by the tanh.

MaxPooling vs Activation

There is nothing much here. If the activation is not very weird, the final result would be the same.

Conclusions:

Find the appropriate order of layers which is often useful

- Group1
  - Conv
  - Batch Norm
  - Activation
  - MaxPooling

    - Dropout or SpatialDropout
- Group2
    - Conv
    - ----- (there was a dropout in the last group, no Batch Norm here)
    - Activation
    - MaxPooling
    - Dropout or SpatialDropout (decide to use or not)
- After two groups without dropout, can use Batch Norm again