Leaf Segmentation and Counting in Plant Phenotyping

Shashank Shekhar
Computer Science
North Carolina State University
Raleigh, North Carolina
sshekha4@ncsu.edu

Abstract — In this paper, I investigate the problem of segmenting and counting leaves from an RGB image of a plant, an important task in plant phenotyping. The problem is approached in two steps – Leaf Segmentation followed by Leaf Counting. Two deep learning architectures are used to accomplish this task – a U-Net model for image segmentation and a VGG-Net like model for counting the number of leaves. Despite the small number of training samples available in this dataset, as compared to typical deep learning image sets, I obtain satisfactory performance on leaf segmentation. Simple data augmentation strategies have been used to create about 60K new images for the purpose of training the VGG-Net Model. Given that the network requires over 1 Million images, the augmented images were not enough resulting in lower performance for the counting problem.

I. INTRODUCTION

The major motivation towards working on this problem is that the traditional manual measurement of plant traits is a slow, tedious and expensive task. The manual measurement techniques use sparse random sampling followed by the projection of the random measurements over the whole population which might incorporate measurement bias. Moreover, plant phenotyping is a bottleneck in modern plant breeding and research programs. Hence, this served as a motivational factor to develop a neural network-based architecture that can accurately perform leaf segmentation and counting. The goal of this project is to use Deep Neural Network Architectures to perform image segmentation of a leaf and count the number of leaves in an image. There were a few challenges faced while working on this problem. The original dataset of images available to train the model is very small in size. Also, the original images are available in different light exposures. Some images are taken in very low light while others are taken in bright light conditions. For some images for which the image shot was taken from a close distance, some leaves are partially cropped. Some of the plants are very small in the image where almost whole of the image is occupied by background such as soil, moss, pot/tray etc. There are some images with partially overlapping leaves making it difficult to count the number of leaves.

II. METHODOLOGY

Goal here is to create two separate models that can accurately perform leaf segmentation and counting tasks. The data to work on this problem is obtained from Computer Vision Problems in Plant Phenotyping (CVPPP) challenge 2017. Only the training dataset is publicly available and therefore I split the available data randomly into train, validation and test sets. The dataset contained images of Arabidopsis and Tobacco plants organized into 4 directories, namely A1, A2, A3 and A4. Directories A1 and A2 contain Arabidopsis plant images taken from growth chamber experiments with larger but different field of view covering many plants and then cropped into a single plant. Directory A4 contains the images of Arabidopsis plant collected using a time-lapse camera. Directory A3 contains images of the Tobacco plants with a field of view chosen to encompass a single plant. Each RGB image is accompanied by a binary segmentation mask and an image with leaf centers denoted by single pixels. An excel file containing the counts for each of the plants in also provided. In total, there are 783 plant images. The data split is done to keep 683 train images, 50 validation and 50 test images.

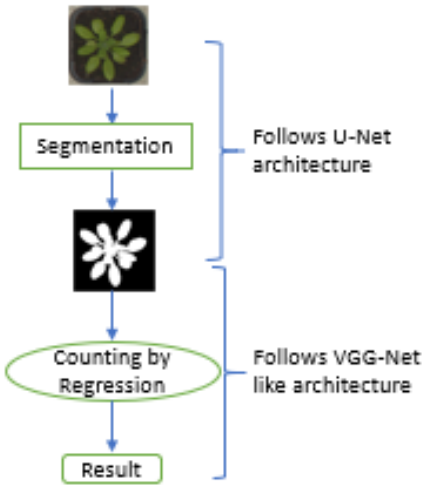


Fig 1: Block Diagram of the Approach

The original data given had images numbered in random order with some missing image numbers. A few preprocessing steps were used to order all the train images and collect them together. The original images provided were in different sizes. Some images were of the size 512 * 512 while others were 224 * 224. The advantage of using smaller images is that it allows for greater data augmentation for the same space available. Hence, I converted all the images to 224 * 224 size. Then I used the image data generator to augment the original images along with their segmentation masks with the following transformations: rotation range of 15, width shift range of 0.1, height shift range of 0.1, shear range of 0.15, zoom range of 0.1, channel shift range of 10., horizontal flip, vertical flip. These transformations were used to create 100 images for each original image generating 68.3K augmented images combined with 683 original images to give a total of 68,983 images. The above augmentations are done in line with those in [1].

The approach used for this problem has been taken from [1] with some modifications. Since I was able to augment only 60K images rather than 1.5 M images as done in [1], I used U-Net model which uses lesser number of parameters and hence lesser number of samples required for training the model. [1] uses SegNet model which is more appropriate when larger number of training samples are available.

All the 68K generated RGB images each having a shape (224*224*3) were given as input to the U-Net Model [2] as shown in Fig 2. U-net is synonymous to an encoder-decoder architecture. It is a deep learning framework based on Fully Connected Network comprised of two parts – a contracting path similar to an encoder to capture context via a compact feature map and a symmetric expanding path similar to a decoder to allow for precise localization. This helps in retaining boundary information. U-net is computationally efficient, is trainable is smaller dataset and is trained end-to-end and hence I preferred it for the problem of image segmentation over other networks. Below is the block diagram showing the structure of U-Net model.

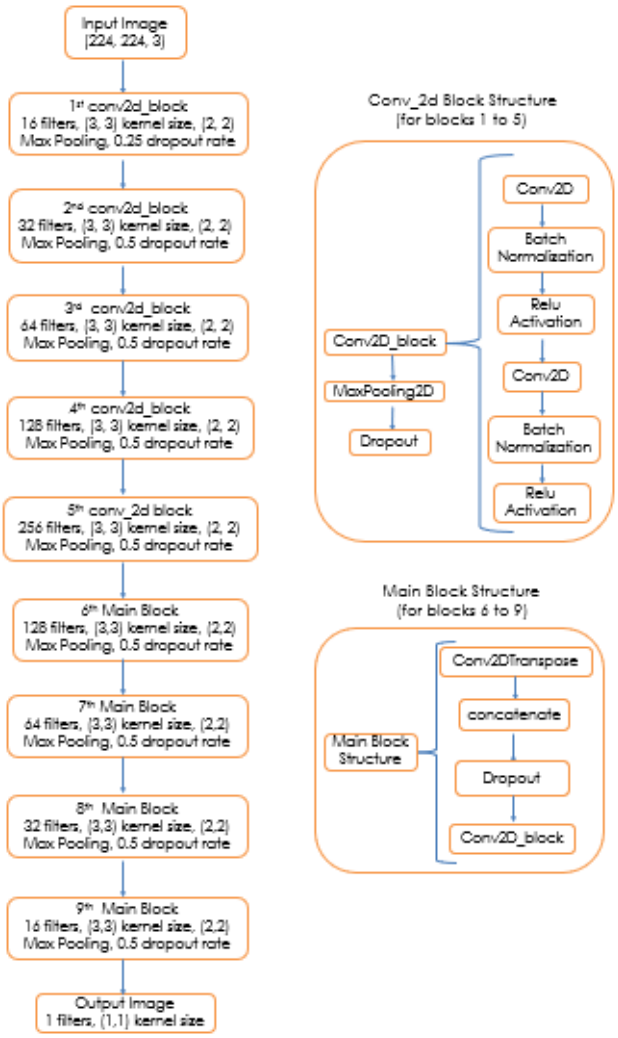


Fig 2: Block Diagram of U-Net Architecture

The downsampling path is composed of 4 blocks. Each block is composed of:

- 3*3 Convolution Layer + Relu activation function with batch normalization
- 3*3 Convolution Layer + Relu activation function with batch normalization
- 2*2 Max Pooling
- 0.5 dropout rate

The number of filters doubles with each block starting with 16 filters for the 1st block to 128 filters for the 4th block. 5th block is the part of the network between the contracting and the expanding paths. This block represents the bottleneck and is similar to the top 4 blocks and uses 256 filters.

The upsampling path is also composed of 4 blocks. Each block is composed of:

- Deconvolution layer
- Concatenation with the corresponding layer from the contracting path
- 0.5 dropout
- 3*3 Convolution Layer + Relu activation function with batch normalization
- 3*3 Convolution Layer + Relu activation function with batch normalization

The number of filters halves with each block starting with 128 filters for the 6th block to 16 filters for the 9th block. The last layer generates the output segmented image.

The segmented images are provided as inputs to the VGG-Net like Model as shown in Fig. 3

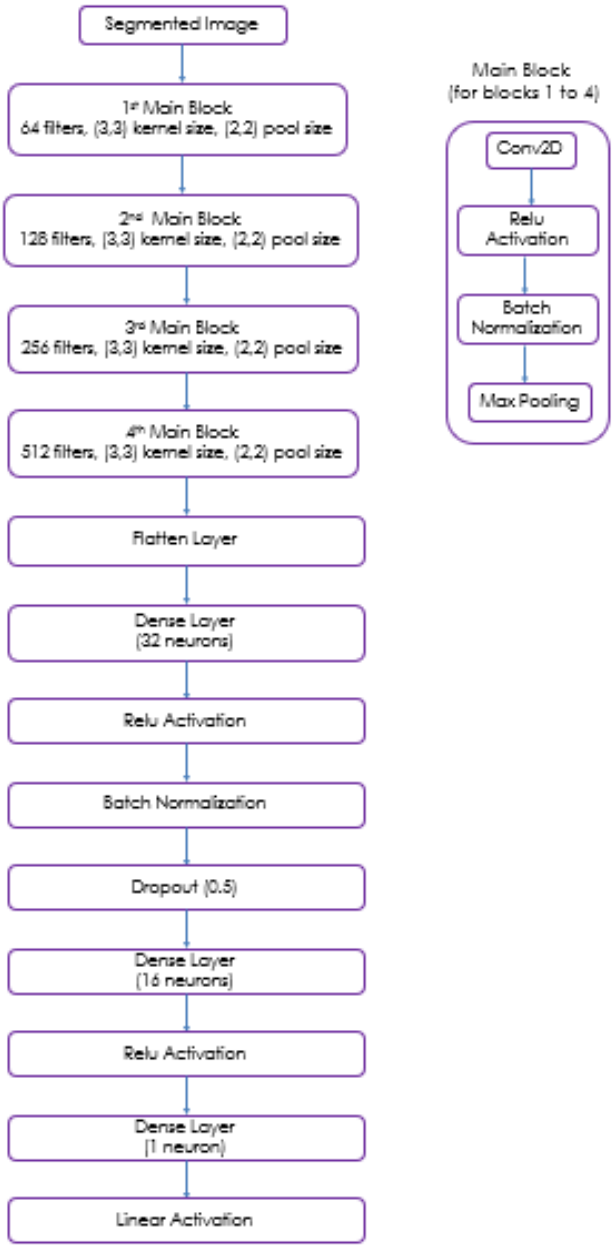


Fig 3: Block Diagram of VGG-Net like Architecture

The model has a structure similar to VGGNet – 16 [3]. The initial part is composed of 4 main blocks. Each block is composed of:

- 3*3 Convolution Layer + Relu activation function with batch normalization
- 2*2 Max Pooling

The number of filters doubles with each block starting with 64 filters for the 1st block to 512 filters for the 4th block. The fourth block is connected to a fully connected layer with 32 neurons

Link to Code

having Relu activation and Batch Normalization. The dropout used in this layer is 0.5. This connects to a fully connected layer with 16 neurons and Relu activation. This then connects to the output layer with linear activation function. The variations from VGG-16 in this model (from the Fully Connected Layer to the Output layer) in this model for purpose of regression are referenced from [4]. The VGGNet–16 [3] is used for classification problems and hence the last layer in it has neurons equal to the number of classes and activation function is softmax. The modified version of VGGNet-16 used for this problem referenced from [4] has a single output with linear activation for regression. This is also a reason for not using a pretrained VGGNet-16 Model since its weights are trained with a loss function suitable for classification tasks. Therefore, I trained this model with 68K images from scratch instead of using a pre-trained model. The implementation required the following python packages: Tensorflow 1.13.0rc1, keras 2.2.4, keras-metrics 1.1.0, keras-preprocessing 1.0.9, opencv-python 4.1.0.25, scikit-learn 0.19.2, numpy 1.16.2, pandas 0.23.4, matplotlib 2.2.3. The code for this implementation is available in the link mentioned in the footer of the page. The code is separated into 2 files:

- Data Preprocessing
- Model Creation, Hyperparameter Tuning and Results

III. MODEL TRAINING AND HYPERPARAMETER SELECTION

All the models below were run on Dell Precision 5810 machine with:

- 6-Core 3.5 GHz Intel Xeon CPU E5-1650v3
- NVIDIA Quadro K22002 with 4GB memory
- Windows 10 Operating System
- 32 GB memory
- 2 TB SATA Storage

For the U-Net based Model, Kernel Size and Batch Size were the hyperparameters chosen. All the models were trained for a maximum of 40 epochs with the patience level set to 5. For all the below models, the learning rate was set to 0.0055 and the pool size was fixed to (2,2) with a dropout of 0.5.

Below table shows the value of f1 score / dice coefficient for the hyperparameters chosen:

Kernel Size / Batch Size	8	16	32
(2,2)	0.9532	0.9516	0.9402
(3,3)	0.9751	0.9642	0.9568
(5,5)	0.9458	0.9396	0.9315

Table 1: Hyperparameter Tuning for U-Net Model

The best model obtained (dark green color) is for a Kernel Size of (3,3) and Batch Size of 8.

Fig. 4 shows the loss (training and validation) vs the number of epochs (40 epochs) for the above best model.

Fig. 5 shows the F1 / Dice Score (training and validation) vs the number of epochs (40 epochs) for the above best model.

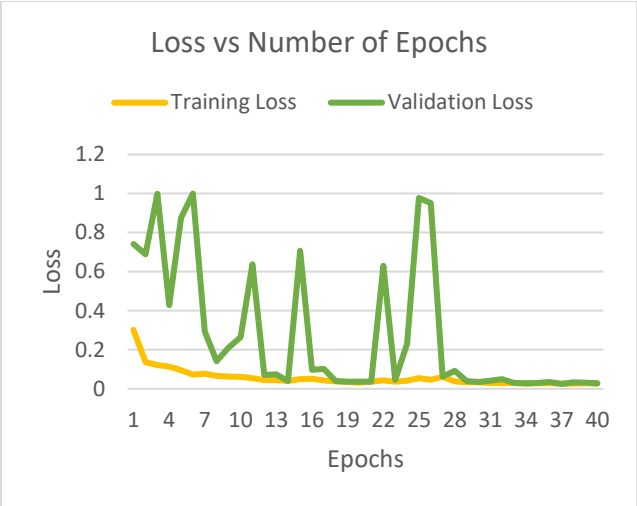


Fig 4: Line Graph for Loss vs Number of Epochs

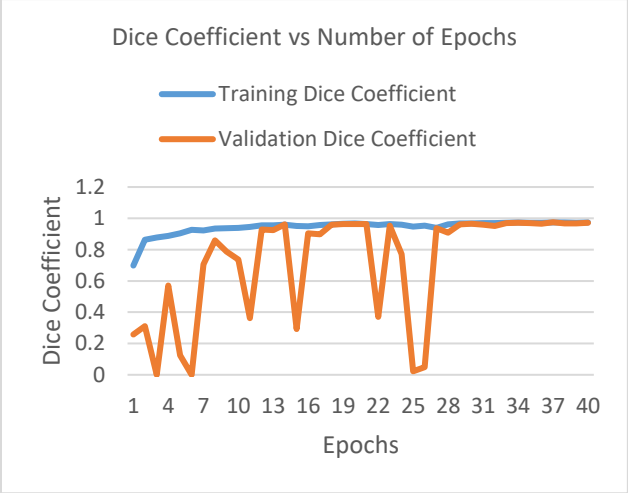


Fig 5: Line Graph for Dice Cofficient vs Number of Epochs

For the VGG-Net Model, Kernel Size and the Batch Size were the hyperparameters chosen. All the models were trained for 20 epochs with a patience level set to 5. The learning rate was set to a value of 0.01 and the pool size was fixed to (2,2) with a dropout set to 0.5.

Below table shows the loss values for the hyperparameters chosen:

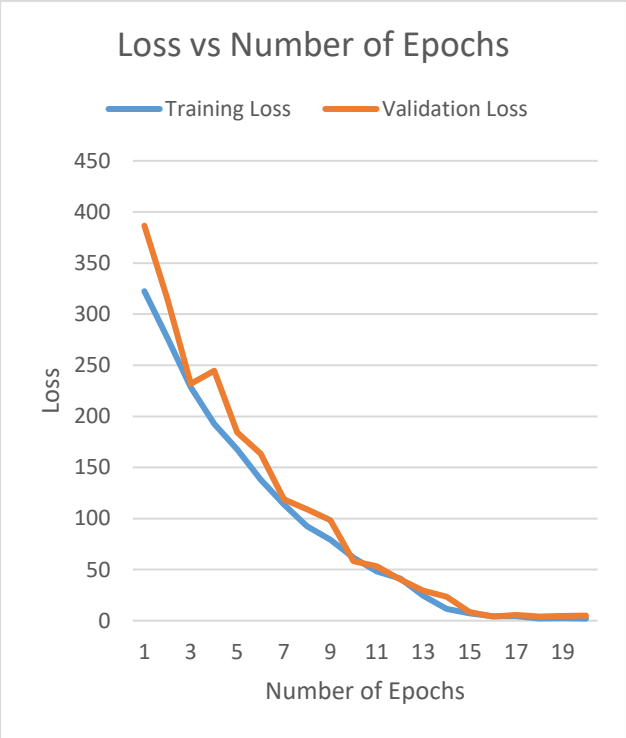
Kernel Size / Batch Size	32	64
(2,2)	4.06	7.3208
(5,5)	9.542	11.2568

Table 2: Hyperparameter Tuning for VGGNet-16 like Model

The best model obtained (dark green) is for a Kernel Size of (2,2) and Batch Size of 32.

Link to Code

Below graph shows the loss (training and validation) vs the number of epochs (20 epochs) for the above best model.

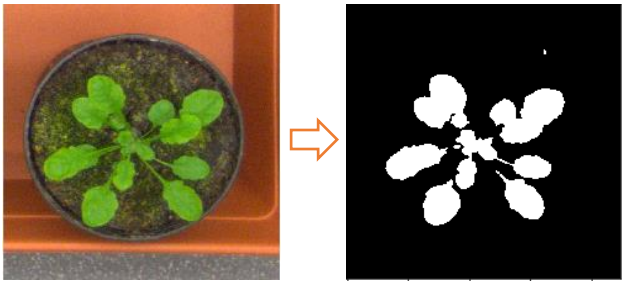


IV. Evaluation

For the evaluation of the given models, there were two separate assessments.

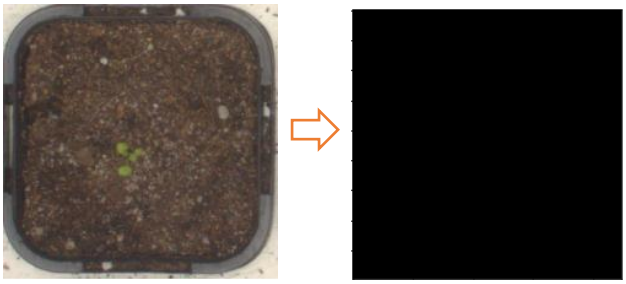
The first assessment was done for the segmentation masks generated. Below are the segmented images obtained using the above trained model for 3 different plant sizes:

A. Large Plant / Corresponding Segmentation Mask:



B. Medium Plant / Corresponding Segmentation Mask:

C. Small Plant / Corresponding Segmentation Mask:



We can observe that the segmentation results for the Large and the Medium Sized Plants are satisfactory whereas for the Smaller sized plants are unsatisfactory. We also observe that the leaf stems which join to the main trunk are missing for some leaves and the leaf margins are not as smooth as the original image. These are some parts of the segmentation task that can further be improved upon. The formal evaluation of the segmentation task was done based on F1 score / Dice Coefficient. F1 score is a value between [0, 1] indicating the overlap between the true and predicted binary segmentation results. The F1 score generated from the baseline model [1] calculated using the results for precision and recall obtained by them are shown below:
Precision = 0.98 and Recall = 0.99
Therefore, F1 = (2 * precision * recall) / (precision + recall)
F1 = (2 * 0.98 * 0.99) / (0.98 + 0.99) = 0.9849
The F1 score obtained for the best model after hyperparameter tuning implemented above using U-Net architecture is 0.9698
The model above has an average performance given the results of the baseline model.

The second assessment was done for the leaf counting problem. Two metrics were used for interpreting the results – Count Difference and Absolute Count Difference. A lower value of count difference suggests that the model is less biased towards overestimate or underestimate. A lower value of Absolute Count Difference suggests that the Average Performance of the model is better. The count difference and the absolute count difference values obtained for the baseline [1] model are -0.33 and 1.00 respectively. These values obtained for the above best model after hyperparameter tuning are 0.7874 and 2.58 respectively. The above models' performance is slightly below average and can be improved by using more augmented images for training the above model. This performance will also improve if the results of the segmentation masks are also improved. This comes directly from the fact that segmentation masks are direct inputs to the VGGNet-16 like model. Further improvements might be possible by providing original RGB images as inputs along with the segmentation masks. This should help the model to recover the missed plant regions as well as to reject the false detections from the original image with the help of segmentation masks.

Link to Code

References

[1] Aich, S., & Stavness, I. (2017). Leaf counting with deep convolutional and deconvolutional networks. In Proceedings of the IEEE International Conference on Computer Vision (pp. 2080-2089).

[2] U-Net: Convolutional Networks for Biomedical Image Segmentation by Olaf Ronneberger, Philipp Fischer & Thomas Brox

[3] Very Deep Convolutional Networks For Large-Scale Image Recognition by Karen Simonyan & Andrew Zisserman

[4] A Comprehensive Analysis of Deep Regression Paper by Stéphane Lathuilière, Pablo Mesejo, Xavier Alameda-Pineda, Radu Horaud