Instance Segmentation via Deep Embeddings - ADVML 2019

Yannay Jaffe Lior Ben-Moshe

https://github.com/lior1990/instance-seg

Abstract

We present new approaches for the Instance Segmentation problem that are based on ”Semantic Instance Segmentation with a Discriminative Loss Function” (Brabandere et al., 2017) framework. We propose different approaches such as: modifying the suggested loss function to take into consideration critical parts of the object (edges, center), using MRF as post-processing action in order to improve the segments’ borders, and building a neural-network that is capable of segmenting a unique object within a group of similar objects. We evaluate our work based on leaf level segmentation of plants from the Computer Vision Problems in Plant Phenotyping (CVPPP 2017) A1 dataset.

1. Introduction

Instance segmentation is defined as the task of assigning a unique label to every object in an image. That is assigning every pixel a label, such that every pixel in the same object receive the same label, and pixels from different objects are labeled differently. It is considered a fundamentally harder problem than semantic segmentation - where instances of the same class are segmented as one. It is often understood as an extension of object detection where, instead of bounding boxes, accurate masks must be predicted.

(Brabandere et al., 2017) obtain masks by two steps: mapping each pixel to a vector in an embedding space, so that vectors that belong to the same instance lie close together, while vectors from different instances are separated by a wide margin; and then using clustering in order to create the mask for each object in the image. The first part is done by a custom loss function.

Our first intuition was to further investigate the custom loss. We were trying to clarify the boundaries between each mask in the image. As a result, we made two modifications to the loss function: adding boundary component to the loss function, and changing the mean calculation of the clusters to be weighted mean with higher weights to the object’s boundaries and center.

In addition, we tried to make sure the boundaries of the masks are well shaped and there are no ”holes” in them. For this purpose we used post-processing Markov Random Field (MRF) method in order to ”clean” each mask.

Another approach we took was to use a different neural-network that was trained to fine-tune each segment, by post processing the distances in the embedding space of every vector from each cluster center.

2. Related Work

(Brabandere et al., 2017) defines a differentiable loss function that operates on the mapping of each pixel in an input image to a point in an n-dimensional feature space, referred to as the pixel embedding. The intuition behind the loss function is that embeddings with the same label (same instance) should end up close together, while embeddings with a different label (different instance) should end up far apart.

This loss function has two competing terms that helps it to achieve the above objective: a term to penalize large distances between embeddings with the same label, and a term to penalize small distances between the mean embeddings of different labels.

They formulate the discriminative loss in terms of push (i.e. repelling) and pull forces between and within clusters. A cluster is defined as a group of pixel embeddings sharing the same label, e.g. pixels belonging to the same instance. The loss consists of three terms:

1. Variance term: an intra-cluster pull-force that draws embeddings towards the mean embedding, i.e. the cluster center.
2. Distance term: an inter-cluster push-force that pushes clusters away from each other, increasing the distance between the cluster centers.
3. Regularization term: a small pull-force that draws all clusters towards the origin, to keep the activations bounded.

The variance and distance terms are hinged: their forces are only active up to a certain distance. Embeddings within a distance of δ_v from their cluster center are no longer attracted to it, which means that they can exist on a local manifold in feature space rather than having to converge to a single point. Analogously, cluster centers further apart than $2\delta_d$ are no longer repulsed and can move freely in feature space. Hinging the forces relaxes the constraints on the network, giving it more representational power to achieve its goal.

The loss function can also be written down exactly. Define C as the number of clusters in the ground truth, N_c as the number of elements in cluster C, x_i as an embedding, μ_c as the mean embedding of cluster C (the cluster center), $|| \cdot ||$ as the L2 distance, and $[x]_+$ = max(0, x) denotes the hinge. δ_v and δ_d are respectively the margins for the variance and distance loss. The loss can then be written as follows:

$$L_{var} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}[||\mu_c - x_i|| - \delta_v]_+^2 \tag{1}$$

$$L_{dist} = \frac{1}{C(C-1)}\sum_{c_A=1}^{C}\sum_{\substack{c_B=1\\ c_A\neq c_B}}^{C}[2\delta_d - ||\mu_{c_A} - \mu_{c_B}||]_+^2 \tag{2}$$

$$L_{reg} = \frac{1}{C}\sum_{c=1}^{C}||\mu_c|| \tag{3}$$

$$L = \alpha \cdot L_{var} + \beta \cdot L_{dist} + \gamma \cdot L_{reg} \tag{4}$$

Where α, β and γ are hyper-parameters.

For the task of segmentation they use post-processing step. The post-processing step uses a clustering approach, which clusters the pixels in the embedding space in order to create masks. For this task a variant of the mean-shift algorithm is applied on the network's output.

3. Our Method

3.1. Custom loss

Our intuition was that some pixels are more important than others, and we should take that into consideration while calculating the loss. This led us to try two different approaches:

- Creating another loss term for the objects' boundaries.
- Changing the clusters' mean to be weighted mean.

3.1.1. Edge Loss

We wanted to put more emphasize on the objects' boundaries so there will be no intersection between the masks of close objects. To do so, we have created another term for the boundaries loss. This term is a variation of the distance term. For every object, we looked at the embeddings of boundary pixels, and demanded that every boundary pixel embeddings of different objects should be far apart from each other. In order to reduce the large computation of the distances between all possible pairs of boundary pixels embeddings, we limited the number of boundary pixels to contribute to that part of the loss. We selected P pixels chosen at random in every training iteration. For our experiments, we used $P = 200$.

Denote by N'_c the maximum between P and the number of points that lies on the edges of a cluster C. δ_e is equivalent to δ_d and controls the inter-cluster push-force. For our experiments we chose $\delta_e = 2(\delta_d - \delta_v)$.

The new loss component can then be written as follows:

$$L_{edge} = \frac{1}{C(C-1)}\sum_{c_A=1}^{C}\sum_{\substack{c_B=1\\ c_A\neq c_B}}^{C}\frac{1}{N'_{c_A}\cdot N'_{c_B}}\sum_{i_A=1}^{N'_{c_A}}\sum_{i_B=1}^{N'_{c_B}} [2\delta_e - ||x'_{c_A}[i_A] - x'_{c_B}[i_B]||]_+^2$$

Where $x'_c[k]$ is the k'th embedding of cluster C boundary. The updated loss can then be written as follows:

$$L = \alpha \cdot L_{var} + \beta \cdot L_{dist} + \gamma \cdot L_{reg} + \psi \cdot L_{edge}$$

Where ψ is another hyper-parameter.

3.1.2. Weighted Mean

The variance term is supposed to penalize large distances between embeddings with the same label. If there is a large distance between some key points of the objects, we want to penalize "harder". The selected key points are:

- top left
- top right
- center
- bottom left
- bottom right

For this purpose, we have changed the cluster mean calculation to be weighted mean with more weight for the points that were described above. This should keep all the surrounding pixels closely together in the feature space, since going farther from them will result in a higher loss.

The updated mean can be then calculated as follows for each cluster C:

$$\mu_c = \frac{\sum_{i=1}^{N_c} w_i \cdot x_i}{\sum_{i=1}^{N_c} w_i}$$

Where w_i is 1 for every regular point, and $w_i \geq 1$ for every selected key point.

3.2. Clustering algorithm

For clustering the results in the embedding space, we used the HDBSCAN algorithm (McInnes et al., 2017). We chose it over other algorithms (such as Mean Shift) due to its quick processing and accurate results even for ”odd-shaped” clusters.

3.3. Post-Processing with MRF

In order to get complete mask and with a ”smooth” shape, we have used Markov Random Field (MRF) method after getting the clustering output. Since MRF can be computationally hard, we have simplified the approach by running MRF per segment that the network has outputted for a given input (except from the background). We convert each segment into binary representation to improve performance. Then, if a pixel from the segment is converted to 0, he is removed from the segment, and a pixel from outside the segment that is converted to 1 joins the segment. Afterwards, we are merging all the segments to a single labeled image.

3.4. Clustering alternatives using neural-network

When we tested our model, we noticed that after running the clustering algorithm, and MRF de-noising, we had descent segments. That means, that we could clearly see that the segmented leaves matched the ground truth, but we believed that we can do better. Even though the segmented image looked nice, we believed that we can fine tune the segments to match the different leaves even better, so we came up with the idea to provide another neural network that will fine tune the segments. The idea here is as follows:

1. Have a trained feature extraction model, and run it on an input image. Get the output which is the embeddings of every pixel.
2. Run a clustering algorithm (HDBSCAN), possibly with MRF afterwards, and get the different segments of the image.
3. Do a ”processing step” on the segmented image, that will output a new ”image” per segment (except the background). The value of every pixel in the ”image” represents a ”likelihood” of the pixel to be included in the segment. For example, if our clustering algorithm found N segments (besides background) in the image, we will have N new ”images”. The value of pixel (i,j) in the k’th ”image” can be interpreted as the likelihood of the pixel (i,j) in the original image to belong the k’th segment.
4. Run all the new ”images” in the clustering neural network, and get a binary mask per new ”image”. A value of 1 in pixel (i,j) of the k’th output from the neural network means that the pixel (i,j) in the original image should belong to the k’th segment. Note that the same pixel may have the value 1 in several output binary masks.
5. Merge all the outputted binary masks from the previous step and get a single labeled mask (with different label per segment).

The processing step we chose is as follows:

1. Per segment we found (except background), we calculate the mean in the embedding space, denote that with μ_c for $c \in \{1...N\}$ (assuming N segments excluding background).
2. Per segment mean in the embedding space, calculate the distance of every pixel from that segment mean. That is assuming the embedding of pixel (i,j) is $x_{i,j}$ we calculate $d^c_{i,j} = ||x_{i,j} - \mu_c||_2$.
3. For every segment found generate a new image I_c, such that $I_c[i,j] = 1 - \frac{d^c_{i,j}}{max_{m,n}(d^c_{m,n})}$
4. Output I_c for all $c \in \{1...N\}$.

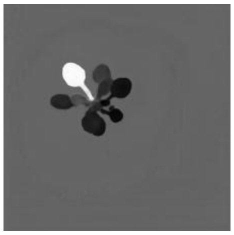


Figure 1. After the processing step. The whiter the pixel, the closer its value to 1.

The merging procedure of all individual segments we chose is as follows:

1. Go through all the individual segments in increasing order of their size.
2. For all pixels that belong to the current segment, assign them a unique label in the merged image (even if the pixel is already labeled).
3. After going through all the individual segments, any remaining unsegmented pixels, mark as background.

Note that by doing so, we override the decisions of small segments by large segments. That in order to avoid ”ghost” segments in the middle of actual true segments.

4. Models architecture

4.1. Model layout

We sketch the overall model in Figure 2, and give further details on the networks’ architecture below.

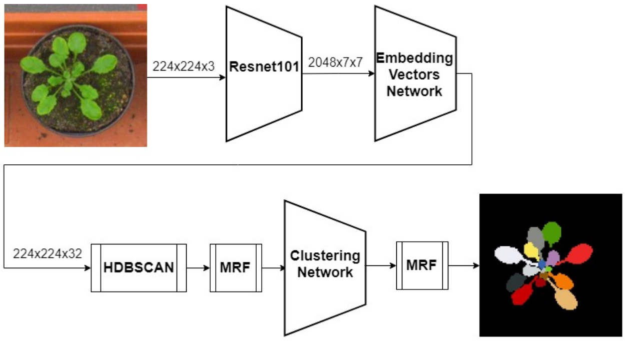


Figure 2. Full network

4.2. Optimizer

Initially, we used the Adam optimizer (Kingma & Ba, 2014) for our training. Due to its well known fast convergence properties. However, we noticed that the network converged to unsatisfying optima. So we changed the optimizer to SGD with momentum, which improved our results, so we kept using that optimizer for both of our networks.

4.3. Embeddings net architecture

We used pre-trained Resnet101 (He et al., 2015) as a backbone for feature extraction, in order to create the mapping between pixel to the embedding space. After that we used a series of convolutional layers to produce the embeddings of each pixel. The output of this network is 32 dimensional vector per pixel.

For training we used Cyclic Learning Rate scheduler (Smith, 2015), with learning rate that varies between $1e-4$ to $1e-6$, that is because the majority of the network is pre-trained and we needed to fine tune using our loss.

4.4. Clustering net architecture

For the clustering network we used a series of convolutional layers with skip connections. The architecture of this network is based on the U-Net architecture (Ronneberger et al., 2015). The full architecture is displayed in Figures 3, and 4. Note that in ”DownSampling Block #1”, we transformed the number of features per input ”pixel” to 64 instead of doubling the amount of input features. The rest of the ”DownSampling Blocks” behave as described in Figure 4.

The loss function we used was the binary cross entropy loss. We added a weight factor of 105 to the leaf segments pixels (the average ratio between single leaf pixels to the rest was $\frac{1}{105}$). For training we used Multi-Step Learning Rate scheduler with $\gamma = 0.1$, and the initial learning rate was 0.01. We decreased the learning after training many epochs, until a final value of $1e-5$. This is due to the fact that we trained a brand new model, and needed fast learning at the beginning of training, and we noticed that the loss was decreasing for many epochs without modifying the learning rate.

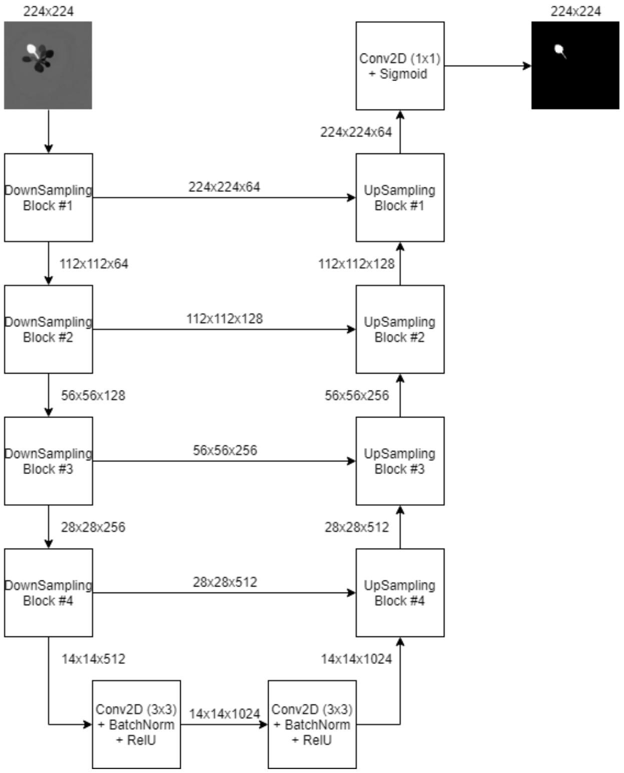


Figure 3. Clustering network

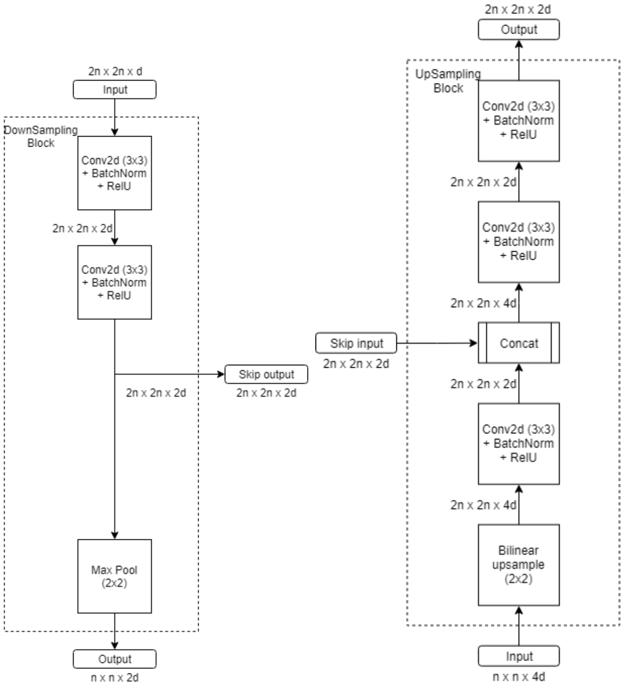
5. Dataset

We tested our method on the CVPPP Leaf Segmentation A1 dataset (Minervini et al., 2015a) (Minervini et al., 2015b). The task is to individually segment each leaf of a plant. The dataset was developed to encourage the use of computer vision methods to aid in the study of plant phenotyping. The A1 dataset training data consists of 128 segmented plants images. We used a total of 108 images for the training and 20 images for the validation/testing chosen at random. We re-scaled all of the images to 224×224, and normalized the images using $\mu = [0.485, 0.456, 0.406], \sigma = [0.229, 0.224, 0.225]$.

5.1. Embeddings net dataset

For training the feature (embeddings) extraction network, we used 78 images of the A1 dataset (out of the 108 training images) chosen at random. Due to the low amount of images we used the following data augmentation methods:

- Random horizontal/vertical flipping.



(a) DownSampling Block (b) UpSamplingBlock

Figure 4. Clustering network building blocks

- Random rotation.
- Random grayscale/color jittering.

5.2. Clustering net dataset

For training the clustering network we used the remaining 30 images in the training set. We created a separated training data for each feature extraction model configuration. The creation of the training data is the same as the processing step described in section 3.4. Except that for the calculation of the mean vector in the embedding space, we used the labels of the ground truth. That is, we separated each image to the individual leaves, and for every leaf, provided a ”probability” for every pixel to be part of that leaf. Because each image had many leaves, we had a total of 495 segmented leaves using this method, so we chose to not use data augmentation on this dataset.

6. Evaluations

6.1. Evaluation Metrics

We report three metrics defined in (Scharr et al., 2015), which are frequently used in leaf segmentation challenges:

- Symmetric Best Dice (SBD): denotes the accuracy of the instance segmentation.
- Foreground-Background Dice (FBD): evaluate a delineation of a plant from the background with respect to the ground truth.
- Absolute Difference in Count ($|DiC|$): the absolute value of the mean of the difference between the predicted number of leaves and the ground truth over all images.

6.2. Evaluation method

For full evaluation of our network we chose the best candidate (according to the cross-validation) from the different feature extraction model variants: with/without modifications to the loss function (edges loss and weighted mean) - a total of four configurations. For every selected feature extraction model we trained the clustering network, and calculated the evaluation metrics on the following:

1. Clustering using HDBSCAN only.
2. Clustering using HDBSCAN, and then using MRF.
3. Clustering using HDBSCAN, and then clustering with our clustering network.
4. Clustering using HDBSCAN, then using MRF, and then clustering with our clustering network (we call it ClusterNet).
5. Clustering using HDBSCAN, then clustering with our clustering network, and then using MRF.
6. Clustering using HDBSCAN, then using MRF, then clustering with our clustering network, and then using MRF.
7. Using the ground truth to extract the mean in the embedding space, and then clustering using the clustering network (to examine the performance of the clustering network on the true instances means in the embedding space).

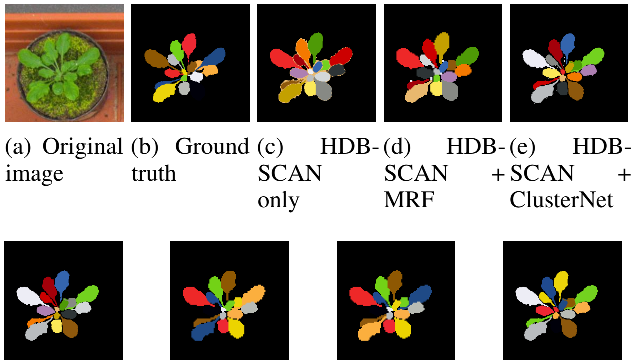
7. Results

After testing our model, we noticed that the additions we made to the loss function didn’t improve its performance, so we will not display here the results for those variants. We will show how by using MRF and the clustering neural network we improved our baseline results. We will compare our results to those mentioned in (Brabandere et al., 2017), however it is important to mention that we couldn’t find the original authors implementation so we couldn’t add our modifications on top of theirs. However, we found an implementation in ”Papers With Code” https://paperswithcode.com in the repository of Almog Elharar https://github.com/alicranck/instance-seg, and used it as a baseline on which we added our modifications. The baseline results are those presented as ”HDBSCAN only”. The results are shown in Table 1.

	SBD	FBD	$\|DiC\|$
RIS + CRF (Romera-Paredes & Torr, 2015)	66.6	—	1.1
MSU (Scharr et al., 2015)	66.7	—	2.3
Nottingham (Scharr et al., 2015)	68.3	—	3.8
Wageningen (Yin et al., 2014)	71.1	—	2.2
IPK (Pape & Klukas, 2014)	74.4	—	2.6
Discriminative Loss (Brabandere et al., 2017)	84.2	—	1.0
End-to-End (Ren & Zemel, 2016)	84.9	—	0.8
Synthetic data (Ward et al., 2018)	90.0	—	—
HDBSCAN only	80.84	90.34	0.9
HDBSCAN + MRF	77.69	90.00	0.9
HDBSCAN + ClusterNet	82.32	94.99	0.85
HDBSCAN + MRF + ClusterNet	81.94	94.94	0.9
HDBSCAN + ClusterNet + MRF	81.22	92.56	0.95
HDBSCAN + MRF + ClusterNet + MRF	80.52	92.29	1.1
True cluster mean + ClusterNet	87.76	95.61	0.0

Table 1. Metrics comparison of different methods.

Figure 5 shows some output examples of the different methods.



(a) Original image (b) Ground truth (c) HDBSCAN only (d) HDBSCAN + MRF (e) HDBSCAN + ClusterNet

(f) HDBSCAN + MRF + ClusterNet (g) HDBSCAN + ClusterNet + MRF (h) HDBSCAN + MRF + ClusterNet + MRF (i) True embedding mean + ClusterNet

Figure 5. Some examples of the validation results.

The best results were achieved by using the following hyper-parameters: $\delta_d = 20, \delta_v = 2, \alpha = 1, \beta = 1, \gamma = 0.001, \psi = 0, \forall i : w_i = 1$.

8. Conclusions and discussion

In this work we have proposed new approaches for loss function modification and extra post-processing steps for the task of instance segmentation.

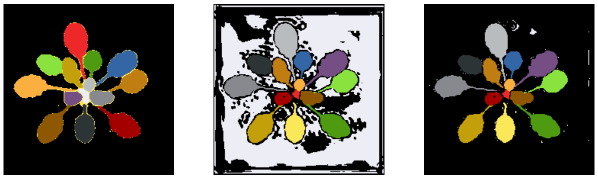
As mentioned, we have seen that our modifications to the loss function didn't yield any improvements. We believe that this is due to including only the edge pixels in the loss term mentioned in section 3.1.1, which caused a non smooth effect on the network. This effect caused sharp variations in the clusters which only hurt the clustering methods we used.

About the weighted mean term as mentioned in section 3.1.2, we believe that it is the right direction, but looking only on 5 pixels per instance is not enough to make much of a difference on the overall learning process. We believe that by merging our approaches we can receive better performance - add a weight term to the loss per pixel, and setting that weight larger at ”interesting” spots, such as making it larger near the edges and smaller at the center (similar to (Ronneberger et al., 2015)). We can also use this whole instance weights to calculate the weighted mean in the embedding space.

On the other hand, we have seen that the post-processing steps we have added were successful. The different combinations of MRF and ClusterNet led to better results over the baseline. We can see that the overall segmentation is working well, and by using those methods we can take it to the next level for making the segments full and well-shaped. By fine-tuning those methods we believe the results can be even better.

As seen in Table 1, when calculating the input for the clustering network based on the true mean of the embeddings we get much better results. This implies that by having a better estimation of the mean in the embedding space, we can greatly improve our performance.

Another thing we noticed is that the clustering network worked well on true instances, but when some ”ghost” instances appeared after the initial clustering, it's performance was greatly reduced. This is due to the fact that the clustering network wasn't designed to distinguish background from foreground (it wasn't trained on entirely background ground truth), but to fine-tune the true foreground segments, so it didn't eliminate ghost segments. Because of that, we tried to rerun our evaluation, but now also treat the background label as another segment, and apply the background label to the reconstructed image last. By doing so, the background label could take over all the ghost instances in the background. Note, that during the training of the clustering network, we used only true leaf instances and not the background. However, the clustering network could figure out that the background cluster is an instance whose label is in the background, this is because during the training of the embeddings network, we treated the background like any other label, so it also got clustered like the leaf instances. The results in Table 1 are those of the evaluation with the treatment of the background as a true label. Figure 6 shows an example for the performance impact due to ”ghost” instance located at the edges of the true instances, and the same result after treating the background as a legitimate label as well.



(a) After HDBSCAN

(b) ClusterNet results

(c) ClusterNet results with background as label

Figure 6. A ghost instance at the edges of the true segments caused the clustering network to perform poorly on the background. After treating the background as label we could eliminate most of the ghost segment.

9. Further Work

There are several more approaches that would be good candidates for a follow-up research:

- Create a dedicated neural network which would distinguish better between the background and the foreground. This will make the task of eliminating ghost labels easier.
- Investigate the merging strategy of individual segments that come as a result from the Clustering Network. For example, finding a better algorithm to solve conflicts between different segments.
- Further explore the colored segment de-noising task and try to use other methods than MRF to do this, by taking into consideration both the accuracy and the speed of the methods.
- Investigate the feature-extraction neural network - its backbone and embedding network architecture. As we've seen we couldn't achieve the results of (Brabandere et al., 2017) in our baseline. This is probably due to difference in the embedding model.
- Search for a way to have better estimates for the true instances means in the embedding space. As we saw, having the true mean can greatly improve the results.
- When creating the training set for the clustering network, use also the means in the embedding space of HDBSCAN clustered instances (and not only of the true labels). This will add data about the distances to not the true mean in the embedding space, and will be more similar to the evaluation conditions.
- Figure out a way for end-to-end training our model. Currently we use two different neural networks, with intermediate processing steps in between.

In addition, we would like to evaluate our method on other datasets such as Cityscapes, Pascal VOC and MSCOCO.

References

Brabandere, B. D., Neven, D., and Gool, L. V. Semantic instance segmentation with a discriminative loss function. *ArXiv*, abs/1708.02551, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. *arXiv*, 1412.6980v9, 2014.

McInnes, L., Healy, J., and Astels, S. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11), mar 2017. URL https://doi.org/10.21105/joss.00205.

Minervini, M., Fischbach, A., Scharr, H., and Tsaftaris, S. Plant phenotyping datasets, 2015a. URL http://www.plant-phenotyping.org/datasets.

Minervini, M., Fischbach, A., Scharr, H., and Tsaftaris, S. A. Finely-grained annotated datasets for image-based plant phenotyping. *Pattern Recognition Letters*, 2015b. URL http://www.sciencedirect.com/science/article/pii/S0167865515003645.

Pape, J.-M. and Klukas, C. 3-d histogram-based segmentation and leaf detection for rosette plants. In *ECCV Workshops*, 2014.

Ren, M. and Zemel, R. S. End-to-end instance segmentation and counting with recurrent attention. *CoRR*, abs/1605.09410, 2016. URL http://arxiv.org/abs/1605.09410.

Romera-Paredes, B. and Torr, P. H. S. Recurrent instance segmentation. *CoRR*, abs/1511.08250, 2015. URL http://arxiv.org/abs/1511.08250.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Scharr, H., Minervini, M., French, A., Klukas, C., Kramer, D., Liu, X., Luengo, I., Pape, J.-M., Polder, G., Vukadinovic, D., Yin, X., and Tsaftaris, S. Leaf segmentation in plant phenotyping: a collation study. *Machine Vision and Applications*, 27, 12 2015.

Smith, L. N. Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472, 2015.

Ward, D., Moghadam, P., and Hudson, N. Deep leaf segmentation using synthetic data. *CoRR*, abs/1807.10931, 2018. URL http://arxiv.org/abs/1807.10931.

Yin, X., Liu, X., Chen, J., and Kramer, D. M. Multi-leaf tracking from fluorescence plant videos. In *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 408–412, Oct 2014.