A Seed Segmentation Contour Generator and Counter

Chaney Courtney, Mitchell Neilsen
Department of Computer Science
Kansas State University
Manhattan, KS, USA

Abstract

Living in a data-driven world with rapidly growing machine learning techniques, it is apparent that utilizing these methods is necessary to achieve state-of-the-art performance in object detection. Recent novel approaches in the deep-learning field have boasted real-time object segmentation methods given the algorithm is connected to a large validation dataset. Knowing that these algorithms are restricted to a given dataset, it is apparent that the need for data generating algorithms is on a rise. As some object detection problems may suffice with a statically trained deep-learning model, it is true that others will not. Given the no free lunch theorem, we know that no machine learning algorithm can truly generalize to data it has not been trained on; therefore, deep learning models trained on images of cats will not necessarily classify dogs correctly. With modern deep learning libraries being ported for mobile devices, a wide range of utility has been made apparent for plant researchers around the world. One such usage of these real-time approaches is to count and classify seed kernels, replacing monotonous-human-error-beridden tasks. Plant scientists around the world have daily jobs of counting seeds by hand, or using multi-thousand dollar devices to automate the task. It is apparent that many third world countries, where such consumer devices do not exist or require too many resources, could benefit from such an automated task. PhenoApps, an organization started within Kansas State University, has been supplying a subset of these countries with modern phones for such uses. With the following seed segmentation algorithm, and the usage of modern mobile devices, scientists can count seeds with the click of a button and produce results in split-seconds. The algorithms proposed in this paper achieve multiple novel implementations. Mainly, Rice's Theorem was used to show that object detection in clusters is an undecidable task for Turing Machines. Along with this, the novel implementations include an Android application which can segment seed kernels, and a machine learning algorithm which can accurately generate contour data sets. The data generator provided in this paper is an effective start for the later usage of deep learning models and is the first step for a real-time dynamic and static seed counter.

Keywords: no free lunch theorem, semi-clustered, Rice's Theorem, Chomsky Hierarchy, Watershed

1 Introduction

One of this paper's main contributions is to answer if there exists an accurate and robust algorithm for image segmentation of various types of semi-clustered seeds which can be efficiently executed on mobile devices. We propose a novel approach that is an optimization and variation of the Extended Watershed algorithm. Previous work on the Watershed algorithm effectively used cross product calculations to estimate clusters of seeds during counting[1]; however these techniques execute poorly in consideration to time complexity on mobile devices due to the limited clock frequency and number of cores on the system. Modern smart phones have brought appealing improvements to these calculations as most phones wield multi-core processors and some even have dedicated GPUs. Yet with these modern improvements to phones, the time complexity of the Extended Watershed algorithm is lacking the qualifications for a quick and snappy application. The gains of such an algorithm is two-fold for future developments. The first and obvious gain is the ability to segment and count various seeds quickly. Secondly, the algorithm can be used as a validation and training data set generator for machine learning algorithms. For example, state-of-the-art deep neural networks provide real-time image segmentation given the network has available contour point training data[2][4][3]. Again, these DNNs can effectively learn the above algorithm and test on a disjoint dataset, giving developers and users the power of a generalized seed detector[5]. Another novel contribution to seed counting is the ability to train this algorithm for various seeds. The given algorithm proposes two specific parameters relating to the area and perimeter of a given seed's contour. Formally, we suggest a property, P, where L(P) is the set of Turing Machines

which accept a predictable semi-clustered seed contour, the languages in P. As shown in 3 set P is a non-trivial property; therefore, by Rice's Theorem the languages describing the algorithms to count these matrices are undecidable in Chomsky's Hierarchy[6]. This paper will prove the non-trivial nature of said property P. The implementation approximating this undecidable language is an iterative simulation of the suggested algorithm using a random parameter search.

2 Methodology

The proposed methods within the following sections describes an effective replacement of the Extended Watershed code. The classical preprocessing for Watershed and subsequent label search simply takes too long to calculate on modern phones[7]. Even sub-4K images passed to this algorithm take up to one minute, refer to Section 3. The purpose of this methodology is two-fold for this paper. Firstly, the implementation of a fast and accurate object segmentation algorithm. Secondly, it is known that the detection of clustered objects is a non-trivial property of images; therefore, this paper aims to support this belief with empirical data. Formally, there exists two languages L1 and L2 which can be represented by two Turing Machines M1 and M2. These two machines have similar algorithms and will always halt on any input, therefore they are decidable and represent a recursive language. If fed the same dataset of seed contour images, M1 will predictably count the contours; however M2 will not. We define predictability by specifically having an accuracy score of greater than or equal to 99, where the total countable objects are given as a test and the count is given as output from the machine. In the property P we specifically say semi-clustered contours as we do not expect or aim to count seeds which are overlaid, similarly we need some contours to be disjointly connected from other contours to obtain ground truth knowledge on the object being counted.

2.1 Definition of Contour

Images are simply matrices of pixels, when these pixels are connected to create an enclosed space we call this a contour. Specifically, the pixels which bound these contours have equal intensity (given an RGB scale). The problem described in this paper is when a user needs to count the number of objects represented by these contours, there may be the case that two or more of these objects contours are one contour. For modern image processing libraries, the representation of these objects' contours are one; therefore the reason for such formalizations are apparent in order to maintain an accurate experiment[8]. Again, we cannot expect to effectively count a clustered image without some ground truth knowledge of the object being counted. The algorithm proposed will produce an undecidable Turing Machine which from our experiments can effectively count seed contours.

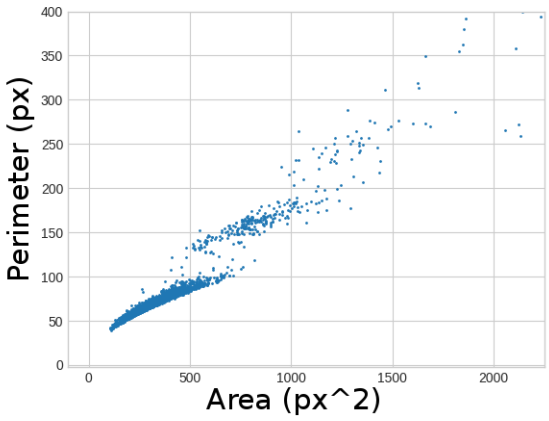

Figure 1: Contour perimeter versus area graph

2.2 Hypothesis

The following sections will delve into the mathematical model which the proposed algorithm uses to segment objects and count seeds. The aim for this methodology is not to replace the classical Watershed algorithm, but to provide a separate, similar, implementation for mobile devices. We expect the count to vary depending on the configuration of clusters in the image. We assume that the number of cluster contours is less than the number of singular seed contours. We will use this assumption to create a dataset of ground truths for the seeds' area, perimeter and inflection points.

Acknowledging Figure 1, which is a plot of an image's contour's areas and perimeters, it is apparent that there is a positively correlated relation between these two variables. This specific data was taken from the Poppy 3960 image (which contains 3960 poppy seeds). There are a few possible clusters visualized. There is one obvious large cluster which is our expected ground truth data (at the lower left hand side). Next, there is a smaller cluster which seems to have larger areas and perimeters. This is effectively showing us that our assumption for this image is true, in that we have a large cluster of ground truth seeds which have small areas (thus most likely just a single seed); however, we also have a cluster of points which represent contour clusters which have a higher combined area and perimeter. As we follow the curve in the data we can see that the sparsity increases therefore there are less

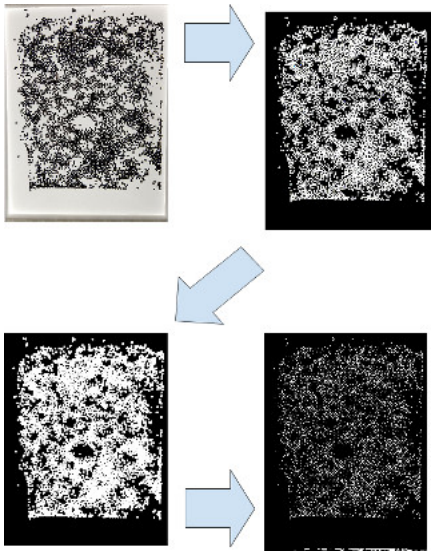

Figure 2: Preprocessing steps

contour clusters than single seed data.

2.3 Preprocessing

In Figure 2 we see four steps total. Before these steps, an image is fed to the algorithm. Once the image is loaded, a histogram is created to represent the different pixel intensities across the image. An average of the lowest and highest found intensities is used as a threshold to the inputted image. This step helps smooth the input image and eliminate any reflective noise. Next, OpenCV's following functions are used sequentially: morphologyEx, dilate, distanceTransform, and threshold[7][8]. Similar to the Extended Watershed algorithm, these steps all together help separate seeds[7]. Before we call OpenCV's find contour function we want to ensure that they are relatively separated; otherwise, we will obtain a higher number of clusters than single seed contours. With these preprocessing steps complete, we are ready to begin counting contours.

2.4 Image Processing

In the previous Extended Watershed algorithm the subsequent step to preprocessing would be to call the OpenCV connectedComponentsWithStats[8] function which returns an array of contours with other statistics like area. We replaced this function by calling findContours and calling a separate function to calculate area and perimeter; we found that this way is much faster than the former. Following this processing the Extended Watershed algorithm would call the watershed function and start counting labels generated[1]. The code in Figure 3 shows

```
Labels = Watershed(gray)

Frame = MaskBorder(Labels)

Unique = Set()

for i in 0 to Labels.rows - 8
  for j in 0 to Labels.cols - 8
    Unique += Labels[i,j]
    j += 8
  i += 8
```

Figure 3: Code segment that shows the use of the watershed function, and the subsequent pixel search using the aforementioned optimization.

an optimized version of the original Extended Watershed algorithm designed for Android devices. Originally, the post watershed process is to find all unique labels generated. Given a mobile device, this process is extremely slow for high resolution images. The following optimization shows the use of a static integer that essentially skips the checking of some pixels. This is an improvement but is a slightly inaccurate way to count the labels. It is apparent in our results how slow this processing really is, and shows why this new algorithm is needed. Therefore, in replacement of this unique label finding, we introduce the estimated cluster count function, which uses our ground truth knowledge and statistics to predict cluster sizes.

2.5 Estimated Cluster Counting

The accumulation of ground truth knowledge and actual seed counting comes from the estimated cluster counting function. After the first call to findContours[8], the resulting contour array is passed to this function. The function iterates over all the contour points and populates three arrays. Area, perimeter, and inflection point count are saved for each contour processed. Inflection points are the spots on a contours boundary where a convex line turns concave and vice versa. The use of inflection points helps find connected objects by first memorizing how many inflection points are visible on ground truth seeds; this is intuitively remembering features of that seeds. Different types of seeds will have a differing number of default inflection points, and there should be an increase of inflections as the seeds are clustered. OpenCV's convexHull function is:

$$O(Nlog(N))$$

where N is the number of contour points[8]. After the hull points are found, another function is called to calculate

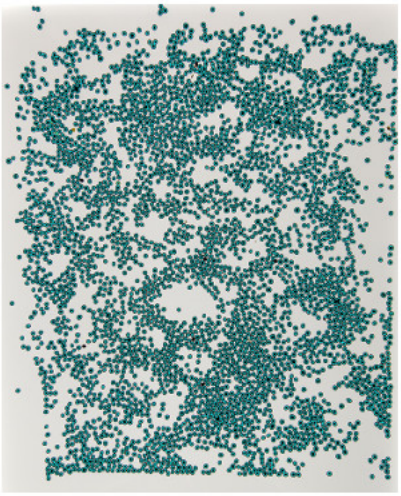

Figure 4: Drawing contours on 3960 seeds.

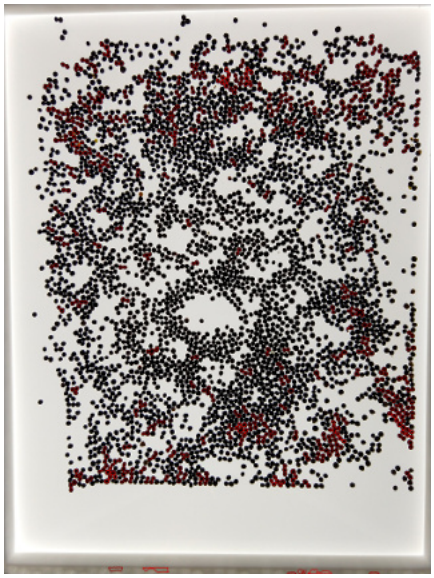

Figure 5: Visualization of clusters.

the convexity defects which are the inflection points, this algorithm is:

$$O(M^2)$$

where M is the number of hull points. Because these calculations are made for each contour type the total runtime of this for loop is:

$$O(NM^2 + N^2 log(N))$$

Subsequently the averages for areas, perimeters and inflection points are calculated which will have a time complexity of:

$$O(N)$$

The next for loop in this function will iterate over all the possible contours which is again N total points. In this loop three scores are calculated by dividing the current contours area, perimeter, and inflection count by the averages of each respectively. If these scores are between 90 and 100 percent accurate they are considered a ground truth. Remembering the assumption that most seeds are alone, the average should be close to the ground truth, therefore these scores should be close to one in order to be reliable data about the seed. These ground truths populate three arrays respectively. Again, the averages of these ground truth arrays are created having similar time complexity as before. This function has one final for loop, again iterating over the possible contour points. A final score is created by taking the minimum of the rounded division of the area divided by the ground truth average and the division of perimeter by the ground truth average. This is considered the estimated count and is returned from the function along with the contour points.

$$C = Min(areas[index]/onesAverageArea, \\ perimeters[index]/onesAveragePerimeter) \quad (1)$$

2.6 Just Ones Average Threshold

After the cluster estimate function is run the function has access to the average area, perimeter, and number of inflection points for a given seed. The next part of the algorithm uses the Pythagorean's theorem, convex hulls, and a ones-average threshold to predict how seeds are connected. First, a for loop iterates over the possible contours, and there is a threshold check to see if the given area is greater than the average seed area. As mentioned earlier, a given count may differ depending on the cluster configuration; therefore, a parameter α is used to represent the number of possible seeds in a contour. The defined threshold is given in Figure 6. Next, the convex hull and defects processing happens again on these contours, but instead of counting inflection points (which is a better overall estimate of the features on a seed) Pythagorean's theorem

```
def justOnesAverage(area, perimeter):
  if area > onesAvgArea * B
    and perimeter > onesAvgPerimeter:
      return True
  return False
```

Figure 6: Python code representing the just ones average threshold.

```
def distanceThreshold(x1, x2, y1, y2):
  dst = math.sqrt((x2-x1) ** 2
          + (y2-y1) ** 2)
  adjOp = (onesAvgPerimeter / 4.0) ** 2
  if dst > A * math.sqrt(2*adjOp):
          inflections = inflections + 1
```

Figure 7: Python code using Euclidean distance and Pythagoreans theorem to create a threshold on the distance from the start and end points of an inflection.

is used to calculate the distance between the start and end points of an inflection. As seen in Figure 7 the function defined is used to count the number of inflections given a certain threshold. The second parameter of this algorithm, β is used to learn how far this distance should be to accurately count a cluster. The threshold is based on the average perimeter of a singular seed; the intuition is to capture the inflections that are large enough to represent seed connections rather than features on the seed itself. A visualization of this distance and inflection points themselves can be see in Figure 8.

Imagine drawing the letter V, the start and end points are when the drawer puts the pen down and picks the pen up respectively. The middle point is the inflection point, this is a simple concave region. Previously, the Extended Watershed algorithm would manually calculate the piecewise cross products of a contours boundary and attempt to detect the inflection regions[1]. The Extended Watershed algorithm deferred in how they used this information, but the code itself was simplified by applying the two formerly introduced functions to calculate convex hulls and then detect defects within the hulls.

A final score is created to represent the count of a given contour. The equation combines the estimated cluster count and the number of threshed inflection points with this formula:

$$Count = (counts[index] + inflections)/2 \quad (2)$$

The above equation represents the final count of a given contour using the calculated inflection defects and the

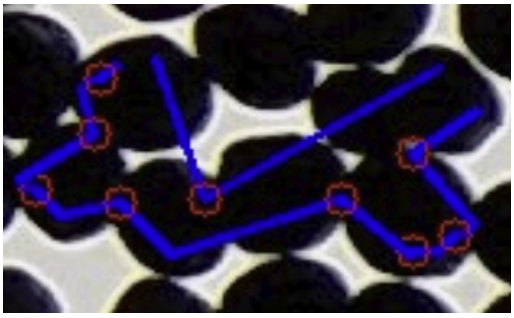

Figure 8: A visualization of the start and end points of inflections.

count given by the estimated cluster function (which was saved into an array earlier).

A final for loop iterates through the count array and sums a final count for the given image. Therefore we can establish the following runtime complexity for the entire algorithm:

$$\begin{aligned} 2 * O(NM^2 + N^2 log(N)) + 2 * O(N) \\ = O(NM^2 + N^2 log(N)) \end{aligned} \quad (3)$$

2.7 Experiment Setup

Similar to the Extended Watershed algorithm, a light box is used to create an even distribution of light among contours to assimilate the most information possible. A phone-holder was designed and 3D-printed to keep a standard distance between the phone and light box. The focal distance between the phone and the light box surface is about 18cm. To find a threshold in order to eliminate noise, we found the smallest seed possible in this setup was about 100 pixels in area, anything smaller than this detected is removed. A caveat of both the Extended Watershed algorithm and this is the possibility of inner contours. Both algorithms use the RETR TREE parameter for OpenCV's findContour function[8], this will essentially eliminate all contours that are within another contour. Therefore, it will be inaccurate for those cases in which multiple seeds are beset a contour. Inner contour processing could be a further extension to this algorithm to relieve these scenarios.

So far the basic idea of contour and seed counting for this algorithm has been explained. Continuing the purpose for this paper, it is apparent that a second algorithm is needed to learn the parameters α and β as mentioned earlier. We introduce an inductive Turing Machine, which is a variant of Turing Machines which may return solutions while never halting. For this experiment, the inductive TM will always halt in a finite number of steps and will save the best results and parameters to reproduce those

results. The algorithm for this inductive TM simply iterates K times, instantiates the counting algorithm with randomly generated values for α and β, runs the algorithm and saves these parameters and the final score to a list. A final score is created by the use of a known test set. As mentioned previously, if the score is at least 99 percent accurate given the known count, then that algorithm is considered accurate and contained in the language of P. It is apparent from the no free lunch theorem that the model created by this algorithm may not accurately classify data that it has not been trained on[5]. Knowing this, we aim to utilize this algorithm to generate labeled data for more effective machine learning algorithms. However, from the data provided we found that there are suitable static values for α and β which will return accurate results, but not necessarily contained in the set P, mentioned earlier. This allows a fast yet unreliable way to count clusters.

3 Results

The following results sections will first describe how object detection is undecidable for Turing Machines, secondly it will report the accuracy and runtime for the algorithm in comparison to the Extended Watershed algorithm.

We claim that image segmentation and object detection is undecidable, and will show this using Rice's Theorem. We know that if all recursively enumerable languages satisfy a property, or no recursively enumerable language satisfies a property that property is trivial[6]. To show a property is non-trivial we can give two examples, a language that does not satisfy the property, and a language that does. We propose two instantiations of the above algorithms, one with an α value of one, and another with an α value of two. This will essentially enable cluster counting for every contour and every contour with an area greater than two of the average seeds size respectively. Using an image with 1025 poppy seeds as our reference the algorithm for an α of one has an accuracy of 0.533, while the algorithm for an α of two has an accuracy of 0.993. Because these two algorithms exist and one satisfies P while the other does not, we can say that P is a non-trivial property. Because P is a non-trivial property, by Rice's Theorem P is undecidable[6]. Which shows that no Turing Machine can promise an accurate count for any cluster of seeds in an image. This justifies the usage of machine learning techniques, and why we are using this algorithm to generate data. Thirty different images were tested on a Google Nexus5X with a high number of seeds to encourage clustering. For some extreme cases such as the image with 3960 poppy seeds, the Extended Watershed took almost a full minute to process, while the proposed algorithm takes a half second for any segmentation. The two algorithms have some innate differences that lead to this result, as mentioned previously the usage of Watershed and unique label finding is a leading cause for a large runtime on high resolution photographs. The Extended Watershed algorithm essentially searches through every pixel of a high resolution photograph, which is a daunting task on a mobile device. Table 2 reports the accuracy of the test set for the Extended Watershed and proposed algorithm. In most cases there is a significant increase in accuracy, but the Extended Watershed algorithm still performs exceptionally.

Table 1: Runtime results from the algorithm running on an Android device.

Label	Runtime EW	Runtime New
poppy825	12.9040	0.4349
poppy925	14.0438	0.4627
poppy1025	15.3391	0.5278
poppy1160	17.0709	0.4687
poppy1260	19.1071	0.4694
poppy1360	20.1819	0.4976
poppy1560	23.0418	0.4394
poppy1760	25.6147	0.4783
poppy1960	28.2106	0.4973
poppy3960	53.3821	0.5742
silphium100	2.9261	0.3940
silphium200	4.2223	0.3881
silphium300	5.5776	0.3835
silphium500	7.3155	0.4637
silphium800	11.5078	0.4376
silphium1200	16.8501	0.4887
silphium1600	20.5449	0.4545
silphium2000	26.2489	0.5691
soy50	2.2948	0.4179
soy100	3.1177	0.4195
soy222	4.3677	0.4508
soy350	5.9584	0.4449
soy500	7.4616	0.5546
wheat200	4.2908	0.4450
wheat300	5.4986	0.4683
wheat400	7.1205	0.4481
wheat500	8.1606	0.4831
wheat700	9.5016	0.5417
wheat800	12.3562	0.4878
wheat900	11.9804	0.5058

Table 2: Accuracy results from the algorithm running on an Android device.

Label	Accuracy EW	Accuracy New
poppy825	0.9406	0.9490
poppy925	0.9459	0.9978
poppy1025	0.9395	0.9980
poppy1160	0.9318	0.9931
poppy1260	0.9460	0.9944
poppy1360	0.9470	0.9985
poppy1560	0.9397	0.9974
poppy1760	0.9477	0.9971
poppy1960	0.9413	0.9914
poppy3960	0.8795	0.9987
silphium100	0.9345	0.9708
silphium200	0.975	1
silphium300	0.88	0.9900
silphium500	0.808	0.998
silphium800	0.8712	0.9975
silphium1200	0.8533	0.9991
silphium1600	0.816875	0.9981
silphium2000	0.8375	0.9945
soy50	0.7692	0.98
soy100	0.9174	0.9900
soy222	0.8648	0.9910
soy350	0.82	1
soy500	0.796	0.994
wheat200	0.965	1
wheat300	0.9566	0.9933
wheat400	0.96	0.9975
wheat500	0.902	0.9920
wheat700	0.7385	0.9857
wheat800	0.7612	0.995
wheat900	0.7711	0.9922

4 Conclusions

The implementations contained within this paper are available on Github, an open source version control website. From the results it is apparent that the runtime for segmenting seeds has been decreased dramatically. Future work will delve into the recent culmination of GPU libraries for mobile devices, which may open up more possibilities for these devices. The sources available on Github include an Android application that utilizes this implementation, the application will be a part of the PhenoApps organization on Github which started a suite of applications for phenotypical research[9]. Secondly, we were able to show that all such algorithms have no promise in returning an accurate result; however, with suitable knowledge on the seed counted, an accurate result

Table 3: Values of α found after the random parameter search.

Label	*alpha*
*poppy*825	2
*poppy*925	5
*poppy*1025	4
*poppy*1160	3
*poppy*1260	5
*poppy*1360	7
*poppy*1560	4
*poppy*1760	5
*poppy*1960	3
*poppy*3960	5
*silphium*100	2
*silphium*200	3
*silphium*300	4
*silphium*500	6
*silphium*800	6
*silphium*1200	7
*silphium*1600	8
*silphium*2000	9
*soy*50	2
*soy*100	3
*soy*222	8
*soy*350	4
*soy*500	5
*wheat*200	4
*wheat*300	5
*wheat*400	4
*wheat*500	4
*wheat*700	5
*wheat*800	6
*wheat*900	6

can be produced. Thoroughly, usage of an inductive Turing Machine innately means we may never get an accurate result, and may never halt. The gains of this paper support the future use of deep learning to generalize over segmentation data. Using these implementations, deep learning algorithms should theoretically be able to not only segment seeds in real time but they should be able to classify them. Another source available on the Github is a python implementation of the above algorithm, this implementation can output Microsoft's COCO data format. This format is typically used in object detectors for deep learning, and is the next step for a real time seed classifier.

References

[1] M.L. Neilsen, S.D. Gangadhara, T. Rife, *Extending watershed segmentation algorithms for high throughput phenotyping*. in Proceedings of the 29th International Conference on Computer Applications in Industry and Engineering, Denver, CO, Sept. 26-28, 2016.

[2] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"* arXiv:1506.01497 [cs.CV]

[3] Kaiming He, Georgia Gkioxari, Piotr Dollr, Ross Girshick, *"Mask R-CNN"* arXiv:1703.06870 [cs.CV]

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, *"You Only Look Once: Unified, Real-Time Object Detection"* arXiv:1506.02640 [cs.CV]

[5] Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning* MIT Press, http://www.deeplearningbook.org, 2016.

[6] John E. Hopcroft, Jeffrey D. Ullman *"Introduction To Automata Theory, Languages, And Computation"*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1990.

[7] P. Soille and L.M. Vincent, *Determining watersheds in digital pictures via flooding simulations* Lausanne-DL tentative, International Society for Optics and Photonics, 1990.

[8] Bradski, G., *"The OpenCV Library"* Dr. Dobb's Journal of Software Tools, 25(11), 120-125, 2000.

[9] T.W. Rife and J. A. Poland, *Field Book: An open-source application for field data collection on Android*. In Crop Science 54, 1624-1627, 2014. DOI: 10.2135/cropsci2013.08.0579.

[10] Chaney Courtney, *"Github account page"* https://www.github.com/chaneylc