Machine Learning Based Citrus Orchard Health Analysis with Autonomous Drone Technology

Rehaan Ahmad

Email: rehaan.contact@gmail.com

Brian Yang

Email: brianayang@gmail.com

Abstract:

A crucial aspect of the economy in Florida is their lucrative orange industry. In 2008-2009, Florida produced 79% of the oranges harvested in the United States with 459,100 acres of land. However, this industry is at risk with the spread of diseases deadly to orange trees and rendering oranges unsellable. Due to the damage wreaked by a disease known as citrus greening, spread by the Asian Citrus Psyllid, a total of 665,000 acres of commercially produced citrus in 2010 was reduced to only 473,000 acres of commercially producing citrus in 2011, a decrease of 29% over a year. Existing drone technology that uses infrared cameras can give an idea of the overall health of an orchard, but infrared does not provide enough detail about potential disease outbreaks. Solutions involving hyperspectral cameras can identify what disease a tree may possess and various characteristics of a tree, but these methods are quite expensive. Moreover, these aforementioned existing technologies focus on analyzing tree health when obtaining data about fruit health should be prioritized as they directly impact commercial output. However, it is notable that there is a vast amount of data and computation necessary to analyze the health of an overall orchard. In this paper, a TensorFlow based convolutional neural network (with OpenCV pre-processing algorithms) is implemented to detect faults in orange orchards utilizing an off the shelf drone with a standard RGB camera.

Keywords: Machine Learning; Tensorflow; OpenCV; Python; Drone

Introduction: In 2004, the state of Florida produced 240 million boxes of oranges. In 2016, that number is going to drop to 70 million boxes of oranges. The number one cause of these orange trees' deaths are diseases such as citrus greening. Diseases such as citrus greening harm the fruit and render it unsaleable. In addition, due to the sheer size of orange orchards, it takes an extensive time for a team of humans to check through the farm. So far, there are two main technologies to get tree health, infrared (Mattar, 2011) and hyperspectral analysis (Okamoto, 2009). Most of these technologies utilize drones as drones are the most efficient way to map out an orchard. Studies utilizing a normal RGB camera have been conducted before, using algorithms such as watershed paired with other image processing techniques. The implementation of a convolutional neural net (Sabzi, 2017) in conjunction with drone technology (Prinsloo, 2017) is significantly cheaper while producing more specific results in terms of orchard health. It is capable of providing location-based information about the disease frequently as well as giving accurate results about fruit health.

Materials and Methods:

2.1 Image Acquisition

The most efficient way to capture data around an orchard utilizes an autonomous drone flying in a preset route, taking pictures of orange trees. The optimal pattern for the drone to fly in order to maximize efficiency and simplify the projection from the set of images in the drone to the set of values in the user interface is a snake pattern. Let x, i and j be local variables to be utilized as part as sequences. Suppose

each image taken by the drone takes a value k_x, where k_1 is the first image taken, k_2 is the second image taken, etc. along the snake path illustrated by the diagram below, and each position on the user interface grid takes a value $\omega_{[i][j]}$, where i is the row value and j is the column value in the orchard (recall that the user interface is intended to be a heat map of the orchard. Let m be the number of rows in the orchard, and n be the number of columns. Then the function $k_x \to \omega_{[i][j]}$ can be represented parametrically as

$$k_x = \omega_{[m-\lfloor \frac{x-1}{n} \rfloor][\frac{\mu_1+(-1)^{\lfloor \frac{x-1}{n} \rfloor}\mu_1+\mu_2+(-1)^{\lfloor \frac{x-1}{n} \rfloor+1}\mu_2}{2}]}$$

$$\mu_1 = x - n\lfloor \frac{x-1}{n} \rfloor$$

$$\mu_2 = n + n\lfloor \frac{x-1}{n} \rfloor - x + 1$$

if the drone is to fly in a snake pattern (Chau, 2017). This function utilizes a string of basic mathematical operations which can be implemented into a python program to map the images taken (and the values derived from each image) by the drone to particular positions on the User Interface (Of course, the number of deformed and normal oranges for each tree will be variables that belong to different classes, where classes are defined as individual orange trees in the orchard).

Let us describe this function in further detail. The parameters μ_1 and μ_2 are used to represent the column number for the drone at different times of its flight. When the drone is in a row where it is travelling to the right, i.e. an "odd" row (based off of the diagram below), the value $\lfloor \frac{x-1}{n} \rfloor$ is even, hence $(-1)^{\lfloor \frac{x-1}{n} \rfloor} = 1$ and the μ_2 terms cancel out, so the column value is μ_1. Likewise, when the drone is travelling to the left, i.e. an "even" row, the value $\lfloor \frac{x-1}{n} \rfloor$ is odd, thereby $(-1)^{\lfloor \frac{x-1}{n} \rfloor} = -1$ and the μ_1 terms cancel out, so the column value is μ_2. This is important due to the information that for an increasing x (assume that $in + 1 \leq x \leq (i+1)n$ for some positive integer i), the value μ_1 is a positive

linear function of slope 1, while the value μ_2 is a negative linear function of slope 2. This is credible due to the information that as the drone travels to the right, the column value increases, yet when the drone travels to the left, the column value decreases.

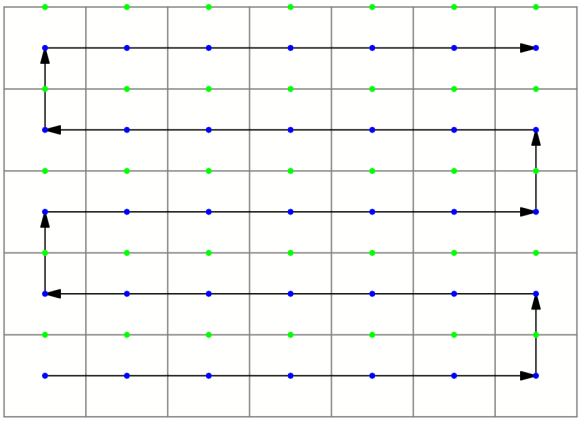
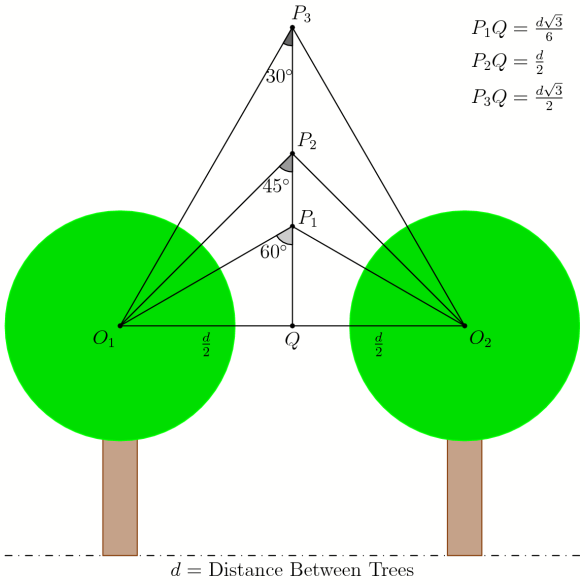


Figure 1: This following grid demonstrates the flight path of the autonomous drone in the orchard. Blue points represent stopping points for the drone to take pictures, green points represent orange trees, and the vectors represent the flight path. At each blue point, the drone will rotate itself to face the green point (tree) directly in front of it and take a picture. As certain vectors intersect green points, the drone is programmed to increase in height and fly over these trees to prevent collision. It takes the drone $\upsilon(mn-1)$ *time to cover the orchard, where* υ *is the amount of time it takes to travel from tree to tree, dependent upon the distance between consecutive trees*

Acquisition of the images must also occur with the drone at the correct height. A drone that is too high may not catch accurate, precise images of oranges and may catch oranges from other orange trees, producing inaccurate results uploaded to the User Interface. The drone must be flown at a low enough height such that the important deformities that may be spotted on an orange can be manifested to the neural network and the farmer itself who views these images. However, a “safety net” can be imagined to exist around the drone, where the safety net cannot intersect with the canopy of the orchard. Hence, the

drone cannot fly too low either, where there is a greater potential of collision. In addition, as the drone's field of vision is a camera, flying too low gives the drone a potential of catching an orange close up and taking too many pixels on the image. The OpenCV preprocessing algorithm that identifies individual oranges puts a constraint around the number of orange pixels, and having too many orange pixels in a contour results in detecting multiple oranges when in reality only one orange was present.

After various tests, which included taking pictures of trees at elevations of $20°$, $30°$, $45°$, $60°$, and $70°$, the optimal elevation for the drone, with respect to the tree height (this value would be implemented by the farmer, who is familiar with the nature of his/her orange trees), is a $45°$ degree angle of elevation from the center of the canopy of the orange tree, and at a position directly in between orange trees. This value can be mathematically represented. If the average tree height was h and the distance between consecutive trees was d, then the drone would fly at a height of $h + \frac{d}{2}$ taking pictures of orange trees.

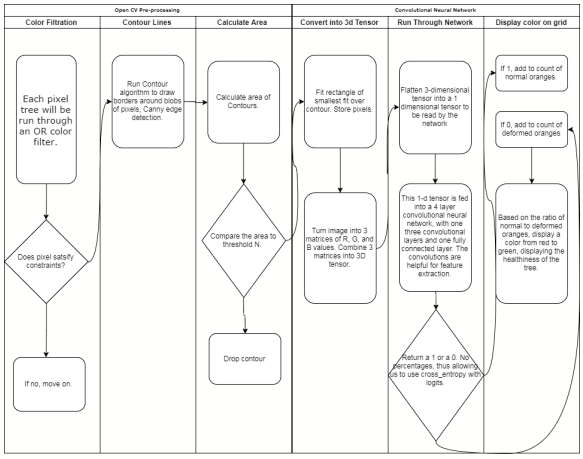


Theory/Calculations:

3.1 Algorithm Outline

The algorithm consists of two main stages, the OpenCV pre-processing, and a three dimensional

convolutional neural network. The pre-processing involves detection of the oranges on the trees themselves, whereas the neural network is used for classifying each oranges as edible or inedible.



Our algorithm will be fed individual pictures of trees taken from a 45^{o} angle as mentioned above. Obviously, feeding entire pictures of trees will be too time consuming and ineffective for a machine learning program. However, a machine learning network should be able to deal with individual oranges and classify them effectively. To cut the problem size down for our neural network, an image processing library called OpenCV in Python 3.1 was used. For proof of concept in this program, it is assumed that every orange on the tree is indeed mature. There are currently methods of detecting immature citrus using a standard color camera (Lee, 2016).

3.3 Pre-Processing

The first step in pre-processing is to threshold out all “orange” pixels. However, the definition of “orange” is rather loose. To formalize these pixels, color filters are created to cluster colors together. R, G, B values were split up into two categories, values that would represent an orange, and values that

wouldn't represent an orange. The "filter" is a set of ranges like so

$$\text{Min1} = [\text{min red value}, \text{min green value}, \text{min blue value}]$$

$$\text{Max1} = [\text{max red value}, \text{max green value}, \text{max blue value}]$$

In the detection of mature oranges, the color filters were developed by hand. After observing R, G, B values of mature oranges in different lightings (incoming sunlight, clouds, etc.) four different color ranges were developed.

The color filterings filter out the mature oranges on the tree. However, there are still many false positives of oranges(unnecessary pixels, oranges from tree in the background, etc.). To eliminate these oranges, the size of each cluster of pixels will need to be determined. To determine this contours are drawn around each object on the screen. To find contours the Canny edge detection algorithm (Canny, 1986) is implemented. Canny edge detection consists of

1. Gaussian blur
2. Four convolutions of the image with edge detector kernels
3. Computation of the gradient direction
4. Non-maximum suppression
5. Thresholding with hysteresis

Once the edges of orange blobs are acquired, the contour of the orange pixel clusters can be found and the approximate rectangular area can be calculated. The obvious possibility of a rotten orange not being orange is discussed on page 8. Because the outer contours are being looked at, even if the oranges have

major deformities on them, they can still be located.

Now all the orange blobs that are too small to be considered oranges can be filtered out. To figure the smallest area allowed, a formula is derived based on the angle of depression, the distance from the base of the tree, and the width of the tree. The formula for our threshold is a function of $\theta = 45^\circ$ (angle of elevation, see Image Acquisition), l (displacement from the base of the tree, will vary between orchards), r (radius of the tree's canopy), x (average diameter of an orange), $z = 20\text{mm}$ (net focal length).

Note the ubiquitous formula:

Object Size in image = focal length * object size / object distance

The oranges that will appear to be largest in the image will be $d_1 = l\sec(\theta) - r$ from the camera (the camera's shortest distance from the edge of the tree's canopy), and the oranges that will appear to be the smallest will be $d_2 = \sqrt{l^2\sec^2(\theta) - r^2}$ from the camera (the length of a segment connecting the camera to the edge of the tree's canopy, tangent to the tree's canopy).

Hence, the maximum and minimum possible values of the orange's size in the image can be calculated

$$\frac{zx}{d_2} \leq \text{Size of Orange in Image (diameter, mm)} \leq \frac{zx}{d_1}$$

based off of the formula for calculating object size in image (every single term distance is represented in millimeters).

The drone that was used utilizes a 12 megapixel camera, 4000 pixels in width and 3000 pixels in height.

Its camera sensor's dimensions are 6.16 millimeters in width and 4.62 millimeters in height. This is a ratio of 1.54 millimeters in the camera per 1000 pixels, so an image that would appear to be 1.54 millimeters wide would appear to be 1000 pixels wide in the resulting image. This implies a conversion from millimeters to pixels in our given threshold by a multiplication of the given ratio.

$$\frac{1000zx}{1.54d_2} \leq \text{Pixel size of orange in image (diameter)} \leq \frac{1000zx}{1.54d_1}$$

The threshold is tested for validation by substituting placeholder values $z = 20$, $x = 73$ (the average diameter for an orange), and $d = 4000$ (4 meters). Utilizing this formula, a diameter of 237 pixels is obtained, which is a reasonable value for the size of an orange 4 meters away in that image, when the full image is 4000 pixels in width.

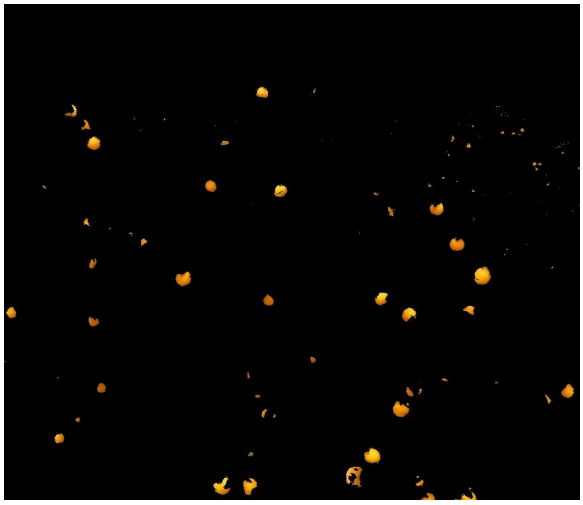
Now it would be reasonable to convert diameter into area. Suppose that by estimate leaves or branches cover parts of oranges such that about $\frac{3}{4}$ of the orange is visible to the camera. The radius of the orange in pixels is $\frac{500zx}{1.54d}$, hence the approximate area of the orange in pixels is $(\frac{500zx}{1.54d})^2\pi$. This area must be multiplied by $\frac{3}{4}$ to satisfy our given estimate, which results in each orange to cover $(\frac{3}{4})(\frac{500zx}{1.54d})^2\pi$ pixels in terms of area. Consequently our final threshold for oranges is the following:

$$(\frac{3}{4})(\frac{500zx}{1.54d_2})^2\pi \leq \text{Number of orange pixels (given the image is 4000 x 3000)} \leq (\frac{3}{4})(\frac{500zx}{1.54d_1})^2\pi$$

The benefit of this resulting inequality is that it is an if statement that detects whether or not a contour has

enough orange pixels to be considered an orange for the neural network to classify. By converting the diameter of an orange in pixels to an approximated area in pixels, the difficulty in identifying oranges within an orange tree is massively reduced. This threshold is part of the pixel-counting algorithm that will eliminate false positives, such as small orange blobs from background orange trees, or large background objects that are orange.

The algorithm is capable of further identifying the center of each orange contour and drawing a square with its circumcenter aligned with the center of each orange contour, such that the square encapsulates the orange completely. Each square is saved as an individual image belonging to a certain class (each class represents an orange tree), and these images are scaled to 72×72. This image scaling is necessary due to the fact that the machine learning algorithm has been trained with images of that size and is hence optimized with classifying images of the same size. The following images displays the algorithm at work.



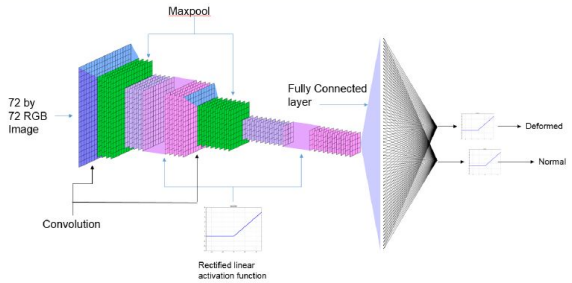
3.4 Convolutional neural network

The goal of the convolutional neural network will be to classify each orange as normal or deformed,

represented as a 1 or a 0 respectively.

3.4.1 Neural Network Structure

The neural network has two convolutional layers, one fully connected layer, as well as one output layer.



The convolutional neural network starts by performing the convolution process of a 72×72 image of an orange, and then creating feature maps with 16 kernels. This process is applied once more with 32 kernels. 5×5 filters are utilized with a stride of 1. The third layer is a fully connected layer for the classification of each orange. The sigmoid function is used as it is differentiable across the domain compared to the rectified linear function (Shang, 2016). Our hidden layer consists of 1024 nodes, which connect to two final nodes in the output layer, represented by $\{0, 1\} = \{\text{deformed}, \text{normal}\}$. This standard convolutional neural network is sufficient for our problem as only two outputs are necessary with a 72×72 image (Krizhevsky, 2012).

3.4.2 Cost Function

The loss/cost of our network is calculated with the *tf.nn.sparse_softmax_cross_entropy_with_logits(prediction,y)* function, and optimized it with an Adam Optimizer, which is a modified version of the gradient descent method. The function to calculate cost is represented as the cross-entropy function:

$$\text{Cost}(P_{\text{output},i,j}, P_{\text{target},i,j}) = \begin{cases} -\log(P_{\text{output},i,j}) & \text{if } P_{\text{target},i,j} = 1 \\ -\log(1 - P_{\text{output},i,j}) & \text{if } P_{\text{target},i,j} = 0 \end{cases}$$

In this equation, $P_{\text{target},i,j}$ represents the target probability for training data i with respect to class j (this value is 0 when the training data i does not pertain to class j, and this value is 1 when the training data i belongs to class j), and $P_{\text{output},i,j}$ is the output probability for training data i with respect to class j. The output probability is calculated by the Softmax Function, which takes a vector of arbitrary real-valued scores and applies weights to change it to a vector of values between zero and one that sum to one. Given this function for calculating cost, we can parametrically define the total cost of an epoch as the summation.

$$\text{Total Cost}(P_{\text{output},i,j}, P_{\text{target},i,j}) = \sum_{1 \le i \le k, 1 \le j \le m} \text{Cost}(P_{\text{output},i,j}, P_{\text{target},i,j})$$

k represents the entire training dataset, and it takes an integer value equal to the number of pieces of data used to train the neural network (in the context of this problem $k = 10000$). m represents the number of output classes for the neural network (in the context of this problem $m = 2$, since our only outputs are "normal" or "deformed" for a given orange). Hence, this function returns a large value for Loss if the target probability subtracted by the output probability for some ordered pairs (i, j) has a greater absolute value. More specifically, $\lim_{x \to 0}(-\log(x)) = \infty$, and as x takes the values $P_{\text{output},i,j}$ or $1 - P_{\text{output},i,j}$, the function causes the cost to be penalized greatly if a particular output probability is distant from the target probability. Calculating cost with *Cross_Entropy_With_Logits* is preferred over using Mean Squared Error to determine the cost, as utilizing logistic regression guarantees a convex function Total Cost. This prevents the possibility of detecting a local minimum during backpropagation by the process of repeated applications of the Adam Optimizer (gradient descent), and instead, a global minimum is approached. Furthermore *Cross_Entropy_With_Logits* more severely penalizes inaccurate

output values, as previously mentioned, compared to the Mean Squared Error function, which can only increase the cost by up to 1 for each training data i. This is important in our application of classification in which there are only two outputs, 1 or 0.

3.4.3 Selection of Optimizer

The AdamOptimizer (which is a modified version of the Gradient Descent Optimization method) is the algorithm that is used to optimize the parameters in our convolutional neural network (parameters include convolution filters and hidden layer weights). It is notable that the function $\mathrm{Total\ Cost}$ can be represented as a function of all input values and all parameters in our neural network structure. The input values have already been provided; hence the function $\mathrm{Total\ Cost}$ is determinable by the parameters in the neural network. Therefore, $\mathrm{Total\ Cost}$ is the same function as $f(\theta)$ as described below. Optimal default settings for the algorithm as used by our machine learning are $\theta_0 = \mathrm{random}$, $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

(Kingma, 2014)

Note that after each repetition of the do-while loop, $f(\theta)$ is recalculated during forepropagation, and the last line of code $\theta_t \leftarrow \theta_{t-1} - \alpha \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ is very similar to the process of gradient descent (as an optimization algorithm), which would be simply $\theta_t \leftarrow \theta_{t-1} - \alpha \nabla_\theta f_t(\theta_{t-1})$.

The AdamOptimizer provides several benefits over the process of gradient descent, as follows [2] [6]:

- The AdamOptimizer naturally performs StepSize (learning rate) annealing. While performing gradient descent, the learning rate θ is reduced after a certain number of epochs. This is not needed when utilizing the AdamOptimizer
- The AdamOptimizer utilizes moving averages of the parameters in the values m_t and v_t (This is similar to momentum), while only g_t represents the parameters in gradient descent.
- These two aspects of AdamOptimizer allows us to use an effective step size and thus approach the minimum of $f(\theta)$ in less epochs.
- The objective function $f(\theta)$ is scaling-invariant, which means the performance is not impacted if the objective function was $g(\theta) = k * f(\theta)$ for some constant k.

3.4.3 Training Dataset

We then trained this number with a dataset of 5000 normal oranges and 5000 deformed oranges. To generate these datasets we used several data augmentation techniques to increase our data set. We initially started off with an original dataset of 1000 normal oranges and 1000 deformed oranges. The following transformations were performed:

1. Gaussian Blur
2. Increased Contrast
3. Horizontal Flip
4. Decreased Contrast

Our oranges were trained in batches of 100 and we had a testing dataset of 1000 total oranges

Results:

We validated our algorithm at a local orange orchard in which 40 trees were present. On each of those 40 trees, the algorithm was able to accurately classify 92% of the oranges on the tree, and it was able to pick up oranges with the color filter 85% of the time.

4.1 Color Extraction and Contour Detection

From an orange tree, our color extraction algorithm would pick out all the "orange" elements from the tree, outputting an image like so:

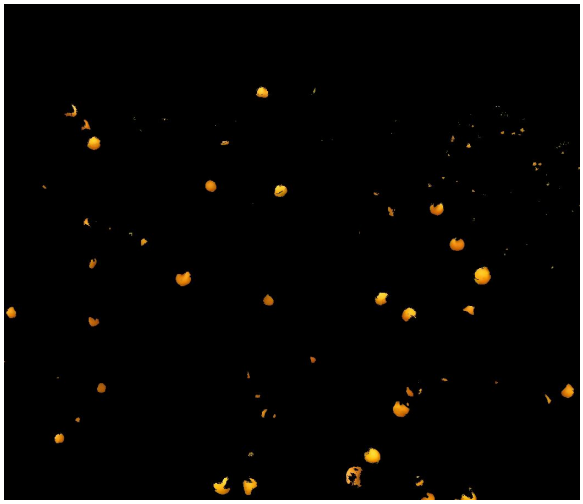


Figure 4.1. The color extraction algorithm extracts all orange from the tree, and takes into account the different shades and varieties of orange that are apparent on the orange fruit. However, notice that no matter how good the extraction algorithm is, there are many false positives as shown in this figure.

Figure 4.2. Each of the oranges in figure 3.1 have a corresponding contour in this image.

The contours help give us a good idea of the area in pixels of each orange area. After validating each area with our threshold function, the algorithm can determine the location of each orange. Even if the orange has a deformed spot, our algorithm could pick up on 84% of the deformed oranges, as the oranges weren't completely non-orange. In addition, our algorithm picked up on 95% of the normal oranges in each picture.

On question that may come up is the ability of the color filter to detect oranges that are deformed, and whose color isn't predominantly orange. However, since the algorithm is looking for contours and boundaries, even if the oranges have spots, the filter is able to detect it.

4.2 Convolutional Neural Network

Our convolutional neural network was tested with a validation set of 2000 labelled oranges, split with 1000 normal oranges, and 1000 deformed oranges.

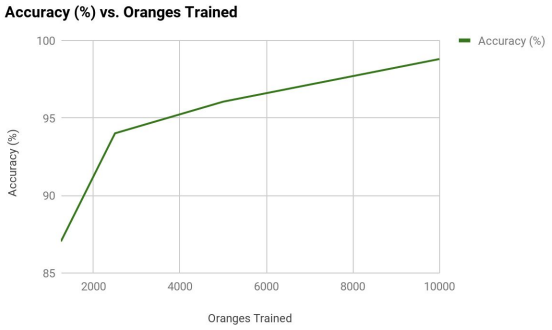


Fig 1: We first trained and tested the convolutional neural net with the original combined dataset size of 1500 oranges. Every increase in data size in the graph is due to data augmentation. The data augmented data was tested each time.

As shown in figure 1, after a while, data augmentation becomes redundant and unnecessary to the neural network. Our final accuracy had an average of 95.6%.

Discussion:

This study experiments with the usage of convolutional neural networks, and utilizes them effectively to classify mature oranges on trees as normal or deformed. The detection of mature oranges themselves wasn't a difficult challenge. Using classical techniques, one can even detect immature oranges with an 83% accuracy (Li, 2016). The focus of this experimentation was to ensure that neural networks can be a reliable method of the detection of deformed oranges.

This algorithm's neural network was trained to only give one of two outputs, normal or deformed. Every orange on the tree would be put into these two categories. This provides numerous upsides along with downsides. Given only two types of labels, training speeds and accuracy improve significantly. In addition, this gives the neural network a larger span of images to work with it. For example, our network wasn't trained with cases of citrus canker disease. However, when tested with 20 different example of

citrus canker, it was effectively able to classify the oranges as “deformed” 19/20, or 95% of the time. However, the problem with this is that we lose track of what type of disease is spreading. Diseases such as citrus greening are much more dangerous than one rotten orange.

This limitation, however, can be solved and the solution would have enough content to fit into another paper itself. Using the fact the citrus greening diseases spread quickly compared to one rotten orange, this problem can be framed as a geo-temporal machine learning problem. Running the drone weekly, and looking at the data over the month, will give sufficient data for a machine learning program. The other crucial piece of information we have is that diseases such as citrus greening spread over location, making this an even simpler problem for a machine learning-based solution.

Figure 3: Data currently would be displayed to the farmer in a heatmap. Currently, the farmer would have to make the decision on how bad the disease is based on the information shown above. The function for each of the colors is $f(O_n, O_d) = \frac{O_n}{O_n + 3O_d}$. *When* $f(O_n, O_d) = 1$ *the output color* $[r, g, b]$ *is* $[0, 255, 0]$, *which changes linearly to* $[255, 255, 0]$ *when* $f(O_n, O_d) = 0.5$, *and then changes linearly to* $[255, 0, 0]$ *as* $f(O_n, O_d)$ *approaches zero. Note that the number of normal oranges in a tree is* O_n *and the number of deformed oranges is* O_d, *and each of those terms must belong to a tree “class”, and are determined by the number of normal and deformed oranges detected by the algorithm on that tree.*

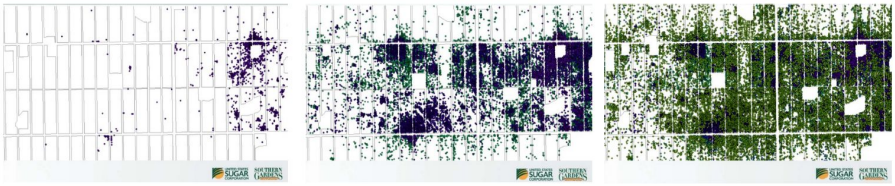


Figure 4: Over the course of 4 years, we can see the cases of citrus greening cluster by location. Purple dots represent asymptomatic cases of citrus greening, while green dots represent symptomatic cases of citrus greening. [5]

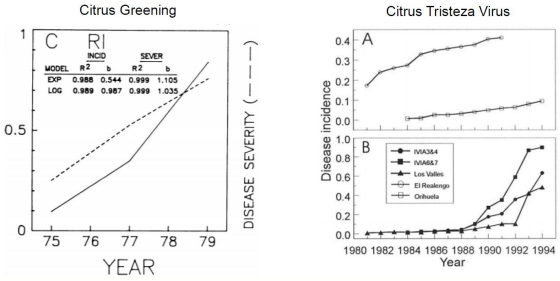


Figure 5: Different diseases have different levels of disease severity and disease incidence over time as shown in cases of citrus tristeza and citrus greening.

A possible extension of this project could be to implement the aforementioned geo-temporal machine learning analysis of the orchard.

Conclusion:

This study presented a comprehensive method that is able to count, detect, and classify oranges on orange trees in an orchard. A proof of the most efficient location and route for a drone to pass through an orchard was also presented. Utilizing a 3D model we were able to create color filters and were able to effectively detect most orange pixels in a picture. The canny edge detection algorithm was then utilized to draw contours and box each individual orange, and a threshold was provided to remove any false positives.

A 4-layer convolutional neural network was also implemented, giving the algorithm the ability to classify many different sorts of orange disease that it hadn't been trained with, because of this convolutional neural network's usage of numerous filters (i.e., citrus greening and citrus canker were both identified as "deformed" by our algorithm). Furthermore, accuracy of detection was not hindered by foreign objects in the image such as leaves.

The creation of a user interface with colors of each entry determined by the weighted ratio between normal and deformed oranges synchronized with the drone's route is crucial for the farmer. Only with a user interface is a farmer capable of immediately reacting to a spreading disease by travelling to the indicated location. In conjunction with the graph representing the percent average quantities of normal and deformed oranges, the farmer can also determine whether or not diseases are spreading by the trends displayed.

In summary, the application of drone technology with a convolutional neural network prepared by an OpenCV algorithm is a viable orchard management method for not only oranges, but for apples, strawberries, lemons, and other fruits as well. This technique would not only reduce the cost of caring for an orchard's health, it would increase the financial output of orange plantations significantly due to its ability to find diseased trees and thus prevent these diseases from becoming major outbreaks.

References:

Canny, John. "A Computational Approach to Edge Detection."

Chau, Chi-kin. "Autonomous Recharging and Flight Mission Planning for Battery-Operated Autonomous Drones."

Okamoto, Hiroshi "Green Citrus Detection Using Hyperspectral Imaging." *Computers and Electronics in Agriculture*, Elsevier, 17 Mar. 2009, www.sciencedirect.com/science/article/pii/S0168169909000258?via%3Dihub.

Julien, Yves. Temporal Analysis of Normalized Difference Vegetation Index (NDVI) and Land Surface Temperature (LST) Parameters to Detect Changes in the Iberian Land Cover between 1981 and 2001. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1012.6299&rep=rep1&type=pdf.

Kim, Yun H. "Crop Pests Prediction Method Using Regression and Machine Learning Technology: Survey." IERI Procedia, No Longer Published by Elsevier, 7 June 2014, www.sciencedirect.com/science/article/pii/S2212667814000100.

Kingma, Diederik P. "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION."

Krizhevsky, Alex. "ImageNet Classification with Deep Convolutional Neural Networks."

Lee, Won S. "Machine Vision System for Determining Citrus Count and Size on a Canopy Shake and Catch Harvester." *Machine Vision System for Determining Citrus Count and Size on a Canopy Shake and Catch Harvester*, www.scopus.com/record/display.uri?eid=2-s2.0-70350298950&origin=inward&txGid=c486945cac94703778df2bf325e3b44c.

Lee, Won S. "Immature Green Citrus Detection Based on Colour Feature and Sum of Absolute Transformed Difference (SATD) Using Colour Images in the Citrus Grove." Computers and Electronics in Agriculture, Elsevier, 26 Apr. 2016, www.sciencedirect.com/science/article/pii/S0168169916301272.

Li, Han, et al. “Immature Green Citrus Fruit Detection and Counting Based on Fast Normalized Cross Correlation (FNCC) Using Natural Outdoor Colour Images.” *SpringerLink*, Springer US, 12 Mar. 2016, link.springer.com/article/10.1007/s11119-016-9443-z.

Sabzi, Sajad “A New Approach for Visual Identification of Orange Varieties Using Neural Networks and Metaheuristic Algorithms.” *Information Processing in Agriculture*, Elsevier, 27 Sept. 2017, www.sciencedirect.com/science/article/pii/S2214317317300884.

Prinsloo, Tania. “The Influence of Drone Monitoring on Crop Health and Harvest Size.” *The Influence of Drone Monitoring on Crop Health and Harvest Size - IEEE Conference Publication*, ieeexplore.ieee.org/document/8016168/?reload=true.

Shang, Wenling. “Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units.”