EN-250 Introductory Quantitative Biology

Professor Strigul

Final Report

YOLO: Real-Time Object Detection for Wildfires

By Luke McEvoy, Alexander Murtagh, Jayden Pereira, Brandon Wang

"We pledge our honor that we have abided by the Stevens Honor System"

5/18/20

Table of Contents

Introduction

With the emergence of computerized object detection and deep convolutional networks, we have witnessed a new and powerful form of data analytics. Beginning with basic object detection technology, the field has developed into a deep learning based detection method known as YOLO: You Only Look Once. YOLO is a real-time object detection algorithm that avoids spending too much time on generation region proposals. Instead of locating objects perfectly, it prioritizes speed and recognition. A single neural network predicts bounding boxes and class probabilities directly from full images or videos in one evaluation. YOLO makes more localization errors but is far less likely to predict false detections where nothing exists.

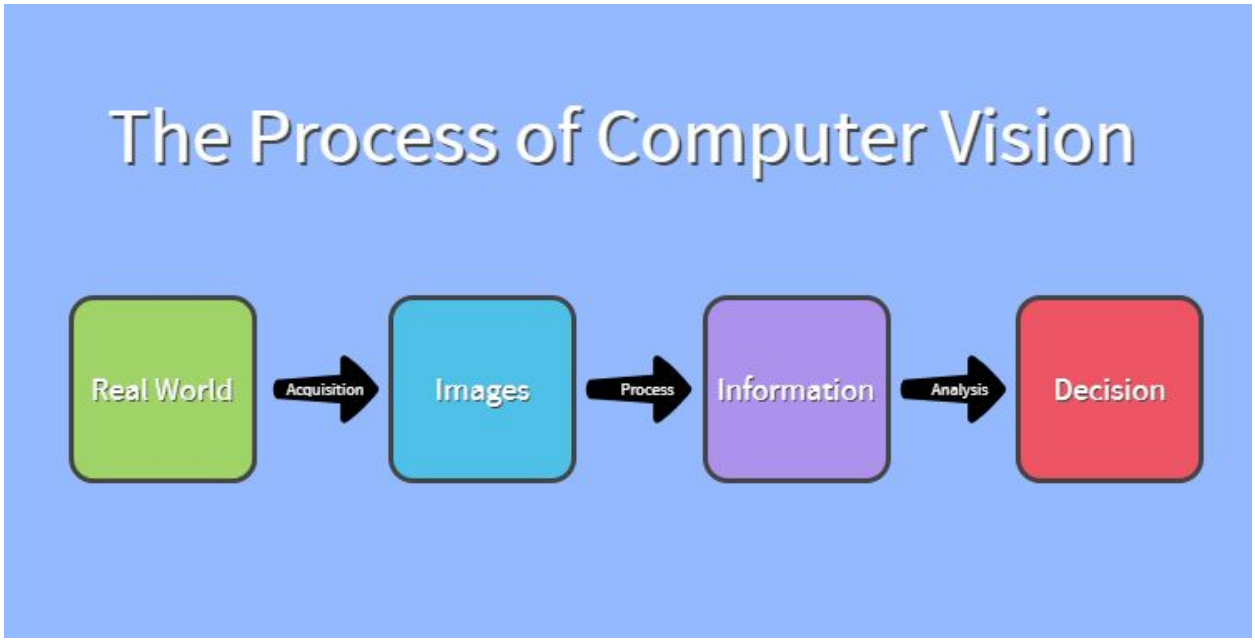

Figure 1: The Process of Computer Vision

Forest fires have been increasing and causing serious damage all over the world. Over the past 10 years, there have been an average of 67,000 forest fires annually and an average of 7 million acres burned annually. With these losses, the suppression costs are seen to have an average of $2 million a year. Although forest fires can be spotted and extinguished, it depends on the size of the fire when it comes to losses. They can have a devastating impact on the acres they affect, the lives they affect, and the structures they destroy. Not only can fires affect the area, but they can also affect the health of those around them through chronic inhalation. The number of personnel with corresponding loss statistics for the years 2015 to 2018 can be found below in Table 1.

Table 1: FS and DOI Personnel and Loss Statistics

	2015	2016	2017	2018
Personnel				
FS Firefighters	10,000	10,000	10,000	10,000
DOI Firefighters	3,997	4,129	4,514	4,492
Losses				
Firefighter Fatalities	13	12	14	19
Structures Burned	4,636	4,312	12,306	25,790

This is where deep learning based detection can shine and be very useful in today's world. Real-time object detection method is on the rise and scientists are branching out in terms of how it can be used to improve life. By using YOLO, computers can be trained to detect different types of tinder and kindling and determine how likely it is to burn. Also, this can directly lead to figuring out the likelihood of an area catching

fire based on what is in the vicinity and if the objects are highly flammable. With the rapid growth of YOLO, potential risks of forest fires can even be detected with satellite imaging to see which areas are more prone to catch fire. These locations can then be marked using Geographic Information System (GIS) Mapping to give governments, scientists, and firefighters a head start when fighting these fires and potential models for the spread of the fire.

History

Beginning in the late 1990s, researchers began to make what would be the first of a series of breakthroughs in object detection technology. The first research to do so created what became known as the Viola Jones Detectors. Prior to this research had to develop and design sophisticated feature representations which could only be handled by what were considered powerful computers at the time. The VJ Detectors, as they became known, worked using sliding windows that would cover an entire image and scale when a potential object was found. Additionally, the VJ Detector made significant improvements in a computer's ability to process images at a relatively fast pace. This was primarily accomplished through three main features known as integral image, feature selection, and detection cascades.

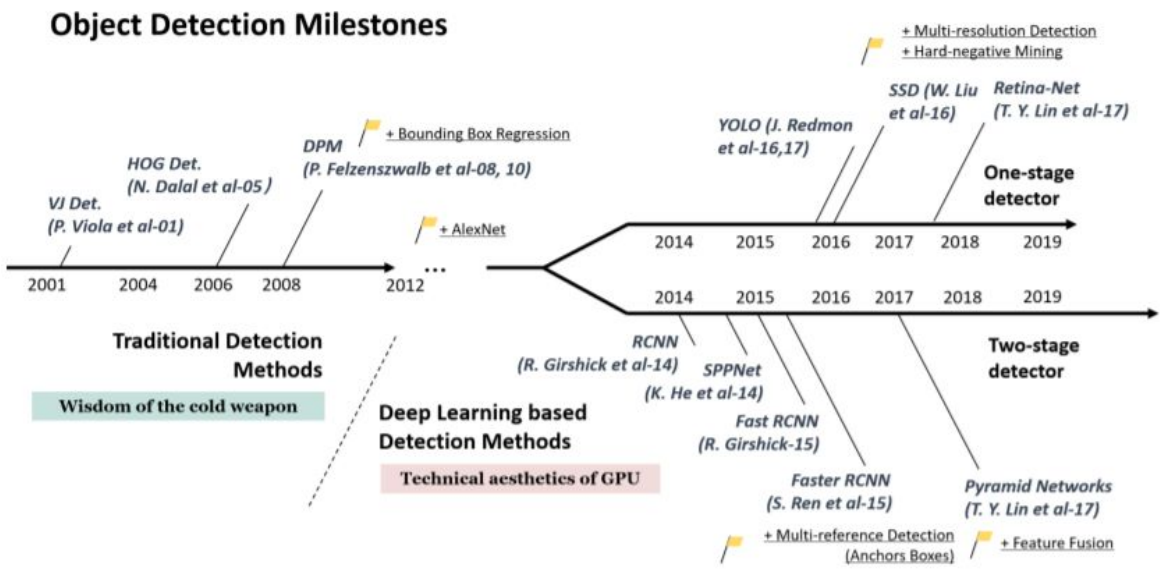

Figure 2: A Road Map of Object Detection

The next major milestone wouldn't come until 2012 with the emergence of deep convolutional networks that could more accurately detect objects. This would lead into a split in the field in which both one and two stage detectors could be developed. YOLOwould become the first one-stage detector when it was proposed by Joseph Redmon and his fellow researchers in 2015. At the time YOLO was the fastest method of object detection, however it suffered from shortcomings like a drop of the localization accuracy compared with two-stage detectors, especially for some small objects.

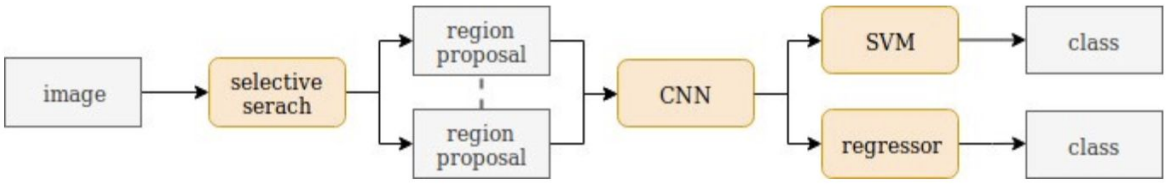

Figure 3: Typical Process of Object Detectors for Classification

Applications

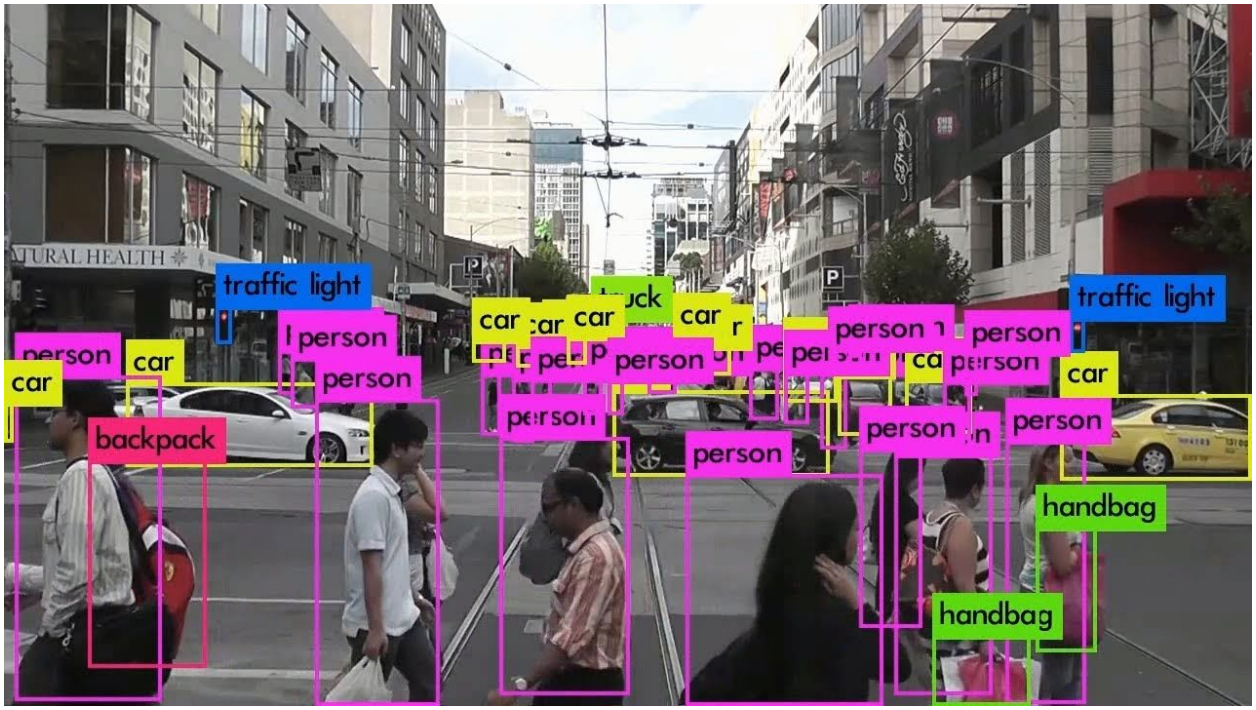

Figure 4: YOLO Object Detection of an Everyday Picture

There are countless applications of YOLO industry - ranging from the medical field to traffic logistics to business analytics. An in depth example of YOLO is Object Detection and Classification with Applications to Skin Cancer Screening, conducted by Jonathan M. Blackledge and Dmitry A. Dubovitskiy. Another application is self-driving cars, where computers in a vehicle identify road signs, distance between vehicles, and lane conditions while traveling. Next, identification of personnel via face images is possible as well from this scientific innovation. Face verification is popular in items such as computers, phones, and airports. In addition, doctors use computer vision to make

analyses on unhealthy esophagitis and use trained image data sets to make quantifiable decisions of the patient's status and/or condition.

In our case we are using YOLO to identify the risk of an area contributing to a forest fire based on the nature of the objects present. By identifying and classifying various items such as trees, rocks, grass, and water, we can accurately develop a model to tell us the risk of fire spreading in that particular area. This system can be integrated into almost any smartphone, which many fire departments already supply. This will help firefighters to more accurately combat high risk areas of the fire and prevent them from missing critical information. The data from smartphones can be captured and relayed to other systems, like the national arcGIS Wildfire Public Information Dashboard, through existing apps from ESRI, the company that develops the arcGIS software. From there, the data can be used by firefighters, scientists, government officials, and other individuals to help combat any wildfires based on the locations where they receive data from. This data can even be overlayed into larger maps that model wind speed and weather patterns so firefighting resources can be more effectively used.

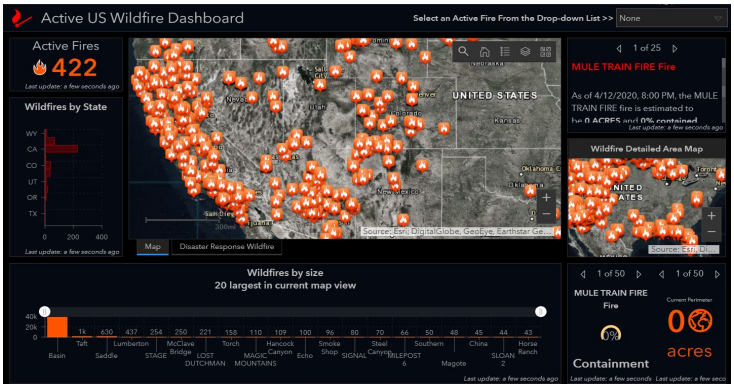

Figure 5: Wildfire Public Information Dashboard

Mathematical Model

YOLO and computer vision as a whole relies on convolutional neural networks and deep learning (shown below). Convolutional neural networks (CNNs) pool images through different stages to analyze input more in depth as the pipeline grows. Furthermore, this system architecture is similar to that of the human retina as it attempts to digest all the input data present in the outside world every second.

In addition, the multi-part loss function is used to optimize the predictions of bounding box size, shape, and classes of objects which is essential for getting the standard deviation of image processing and classification error to a minimal. Furthermore, the multi-part loss function does this error reduction by penalizing wrong classification and misplaced box coordinates. Moreover, the function shown in figure 10 is an amalgamation of a few functions that combine to create a multi-part loss function. This function optimizes the predictions set up by the neural network while reducing error of wrong classification of objects if the bounding box is incorrectly placed.

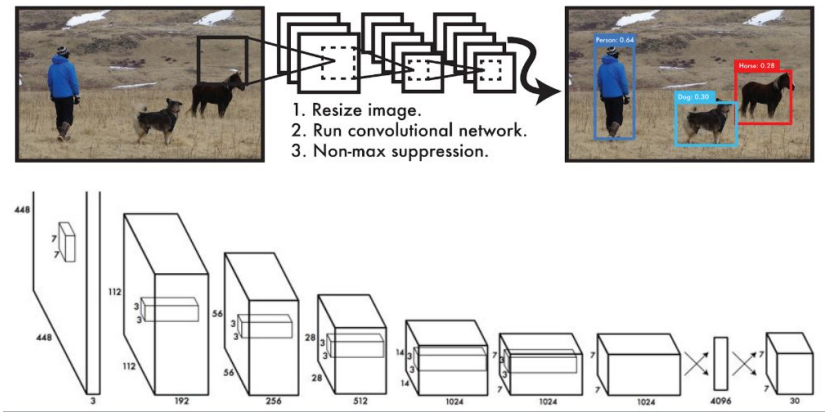

Figure 6: Convolutional Neural Network Model Applied to a Picture

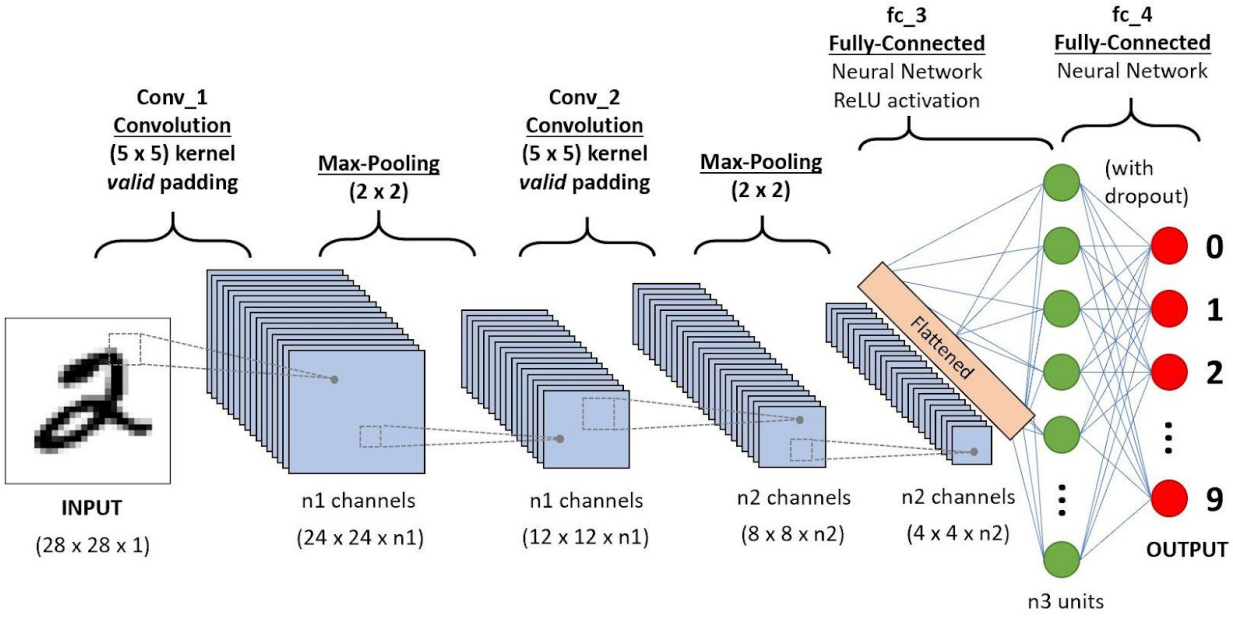

Figure 7: Convolutional Neural Network Background

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Figure 8: Mathematical Model and Equations Behind YOLO

Original Research

Our research project will apply the theoretical and mathematical concepts of YOLO: Real-Time Object Detection to a scalable algorithm, backed by a trained machine learning data set of concentrated areas of forests. For our research, our data set will consist solely from forests in California. California specifically suffers from frequent wildfire outbreaks. Using YOLO we will be able to develop a model to detect high risk areas to slow and stop the spread of wildfires. Moreover, the pivotal conceptual principles that will back our project are convolutional neural networks, image processing, and data visualization.

Figure 9: Wood Debris Example Picture 1

Figure 10: Wood Debris Example Picture 2

Figure 11: Wood Debris Example Picture 3

The likelihood of fire spread across a region in a percentage can be generalized into the formulas:

$$S_g = \frac{\sum_{i=1}^{n}(y_i \cdot z_i)}{x}$$

$$S_c = S_g \cdot \Delta C$$

For the first formula, $\mathbf{S_g}$ = general likelihood of fire spread across a region, **n** = total number of objects, **x** = total area, **y** = area of object, **z** = ease of ignition. This will aggregate all of the objects in our pictures as bounded by the computer vision algorithm into a ratio of how much shown in the picture is actually flammable and likely to spread fire that is added into the region.

The variable z can be classified by using table 2:

Table 2: Ease of Ignition

Ease of Ignition Based on General Flash Point and Flammability		
z = 1	z = 0.6	z = 0
Wood	Grass	Water
	Leaves	Rock

All wood is characterized as a constant due to the generalization of averaging out the relationship between area and ease of ignition in wood. As the wood's surface area decreases, flammability increases (i.e. twigs), since this is an inverse relation we have decided to generalize the ease of ignition in order to optimize the computational complexity of the model.

For the second formula, $\mathbf{S_c}$ = climate-based change in likelihood of fire spread across a region, **ΔC** = change of humidity based on average climate of location. ΔC can be found by simply obtaining the average humidity in the region and then measuring the humidity in the region at a specific time, which can be seen below. This allows us to take our relative area model and define it with even more points of data that can alter the likelihood of fire spread. In particular, if the climate in a certain area becomes drier than others, the fire would most likely be spreading to the drier region.

$$\Delta C = \frac{C_{avg} - C_{measured}}{C_{avg}}$$

Before the images are able to be effectively trained in a CNN, the training images need to be altered. The characteristics of depth, concentration of brush, and static of image need to be quantified. With this, using image processing libraries such as cv2 and matplotlib.pyplot.

```
3   import cv2
4   import matplotlib.pyplot as plt
```

Figure 12: Library Imports

The program opens an image and assigns it to a local variable called forest. Next, a threshold function is applied to the local image, known as forest. This function is called adaptiveThreshold and transforms a grayscale image to a binary image according to the formula:

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxValue} & \text{if } \texttt{src}(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

Figure 13: Binary Threshold Formula

adaptiveThreshold takes in parameters: source image, destination image, adaptive thresholding algorithm to use, thresholding type, blocksize, constant subtracted from mean.

```
2
3   import cv2
4   import matplotlib.pyplot as plt
5
6   forest = cv2.imread('test1.JPG', 0)
7   th2 = cv2.adaptiveThreshold(forest, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
8                               cv2.THRESH_BINARY, 11, 8)
9   plt.imshow(th2, cmap='gray')
10  plt.show()
11
```

Figure 14: Reading and Manipulation of Input Image

Using matplotlib.pylot, the altered forest image is converted to a grayscale and output.

Figure 15: Example 1 Input Image for Static Algorithm

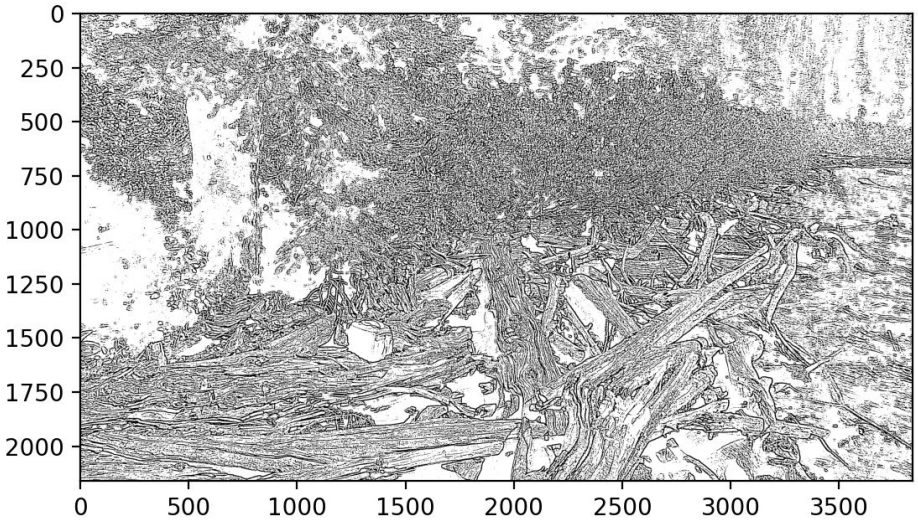

Figure 16: Example 1 Output Image for Static Algorithm

Figure 17: Example 2 Input Image for Static Algorithm

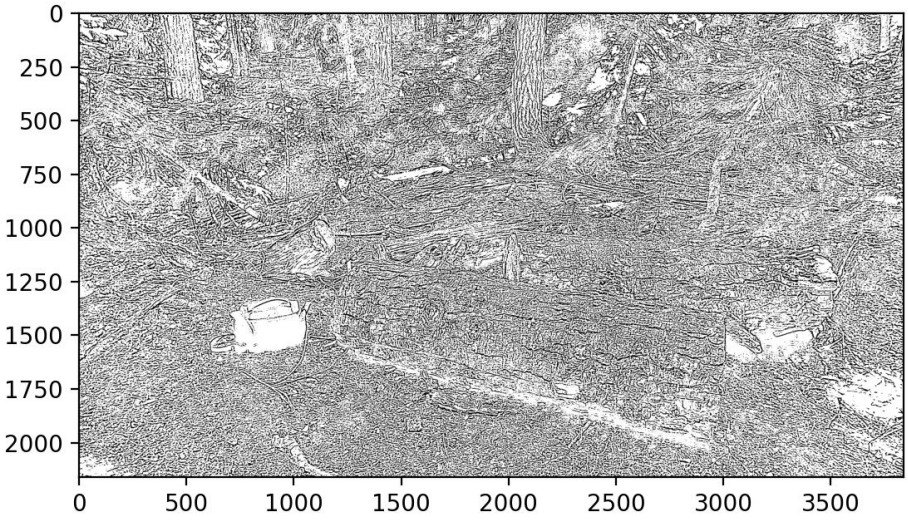

Figure 18: Example 2 Output Image for Static Algorithm

Figure 19: Example 3 Output Image for Static Algorithm

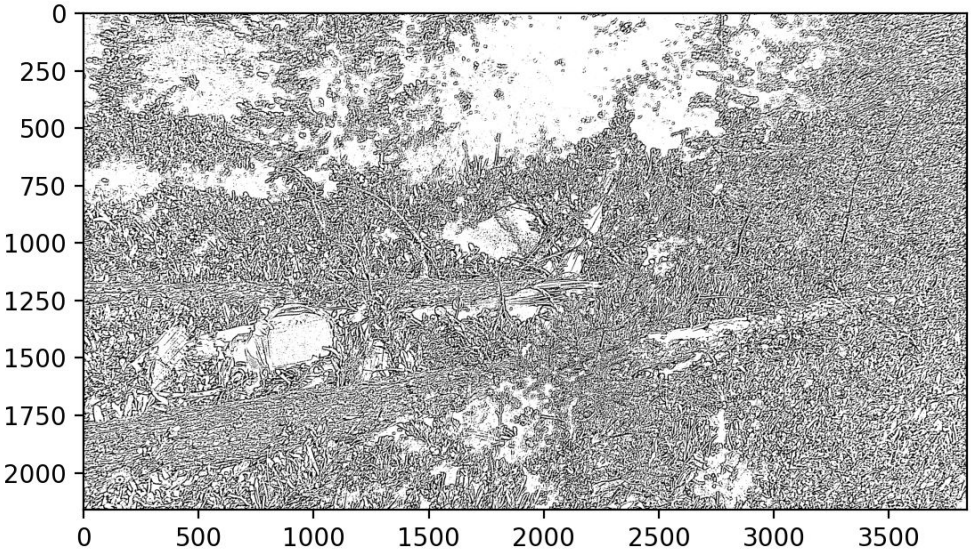

Figure 20: Example 3 Output Image for Static Algorithm

This manipulation to the input images allows for a better analysis of depth and brush concentration of images. Moreover, computations can be made to determine the static of input images. This parameter can in turn be passed into the mathematical model previously described.

The next part is the depth perception analysis of input images. Similar to before, when determining brush concentration, the packages numpy, matplotlib.pyplot, and cv2 are imported.

```
5   import numpy as np
6   import matplotlib.pyplot as plt
7   import cv2
```

Figure 21: Library Import

Second, the input images are read and assigned to a variable called depth_forest. Also, a variable called sample is declared. Sample is the depth_forsest converted to a RGB image scale from its original BGR scale.

```
8
9   depth_forest = cv2.imread('test2.JPG')
10  sample = cv2.cvtColor(depth_forest, cv2.COLOR_BGR2RGB)
11
```

Figure 22: Reading and Manipulating Input Image

Sample is then converted from a RGB image to a HSV image.

```
12  # convert from RGB to HSV
13  hsv = cv2.cvtColor(depth_forest, cv2.COLOR_RGB2HSV)
14
```

Figure 23: Convert Image from RGB to HSV Color Scale

HSV (hue, saturation, value) models the way paints of different colors mix together. The saturation dimension resembles various tints of brightly colored paint. The value dimension resembles the mixture of those paints with varying amounts of black or white paint. Using this personifies depth in images.

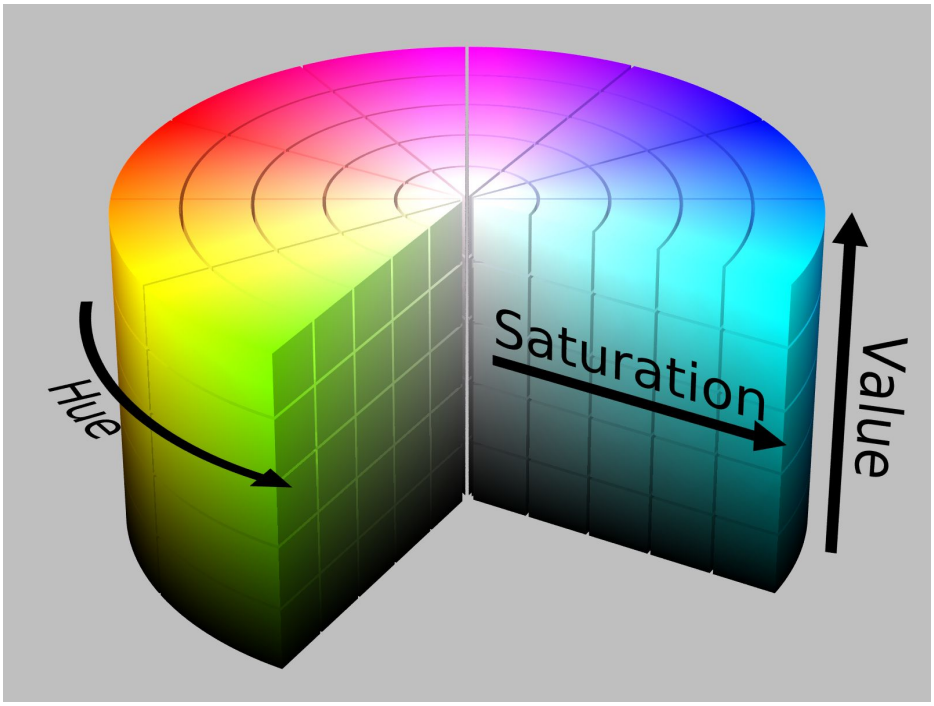

Figure 24: HSV Color Scale Representation

We then define our color selection criteria in HSV value. Further improvement is to be done with this snippe. The declaration of lower and upper hue bounds should be dynamic to the average hue value of the image. In its current state, the lower_hue and upper_hue are hard coded since the images have similar ranges in brightness, but this approach is not optimal.

```
14
15   # 1. define our color selection criteria in HSV value
16   lower_hue = np.array([50, 50, 50])
17   upper_hue = np.array([200, 200, 200])
18
```

Figure 25: Declare HSV Hue Bounds

Next, the image is masked using HSV and output.

```
19   # 2. mask the image using HSV
20   mask_hsv = cv2.inRange(hsv, lower_hue, upper_hue)
21   masked_image = np.copy(sample)
22   masked_image[mask_hsv==0] = [0,0,0]
23
24   plt.imshow(masked_image)
25   plt.show()
```

Figure 26: Filter Image of Hue Values Outside Declared Bounds

HSV (hue, saturation, value) models the way paints of different colors mix together. The saturation dimension resembles various tints of brightly colored paint. The value dimension resembles the mixture of those paints with varying amounts of black or white paint. Using this personifies depth in images.

Figure 27: Example 1 Input Image for Depth

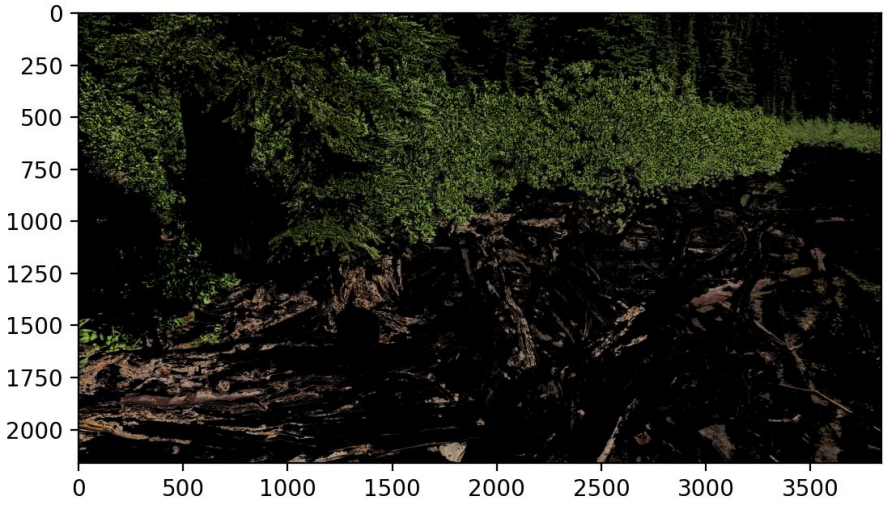

Figure 28: Example 1 Output Image for Depth

Figure 29: Example 2 Input Image for Depth

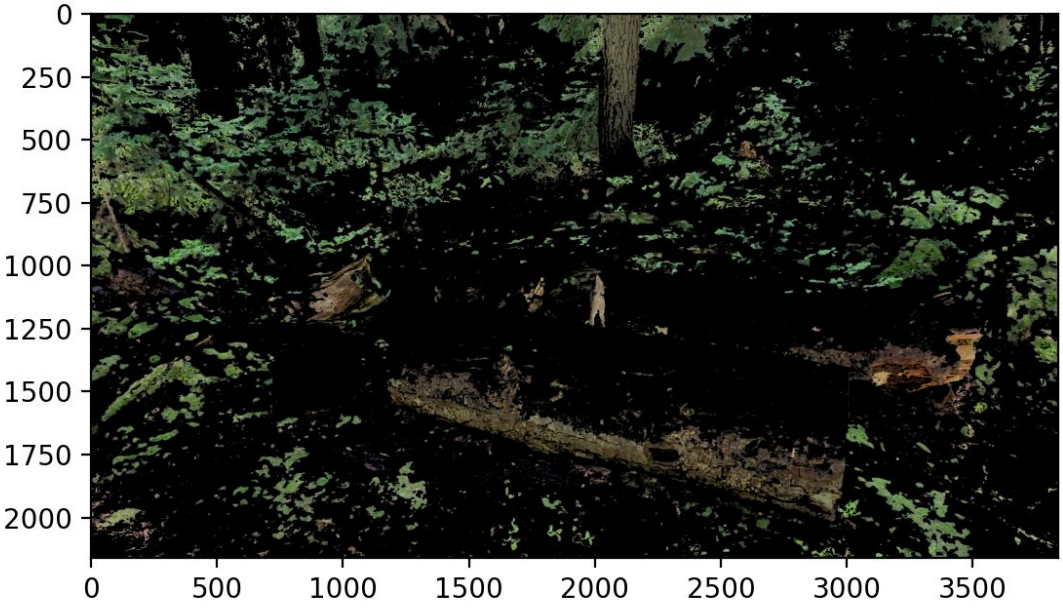

Figure 30: Example 2 Output Image for Depth

Figure 31: Example 3 Input Image for Depth

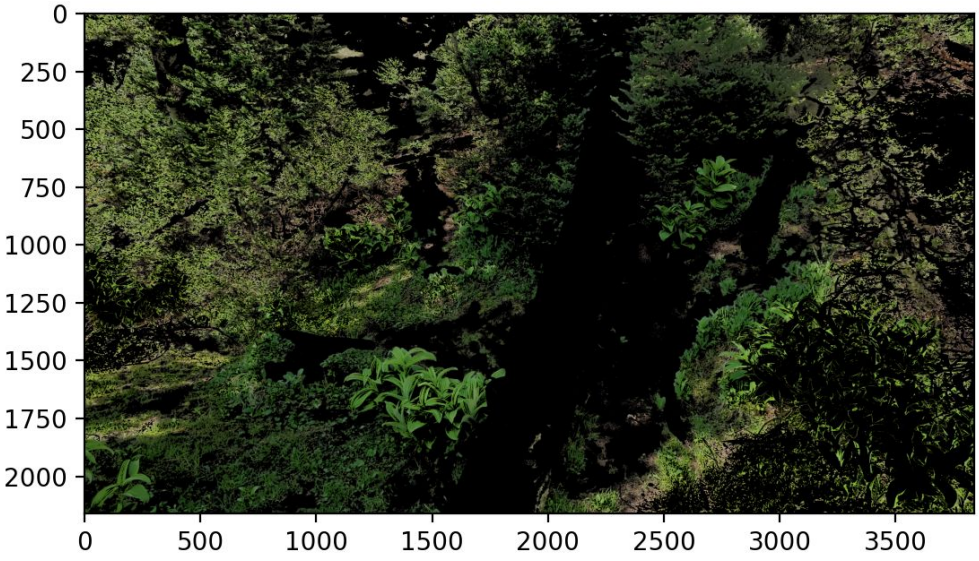

Figure 32: Example 3 Output Image for Depth

Figure 33: Example 4 Input Image for Depth

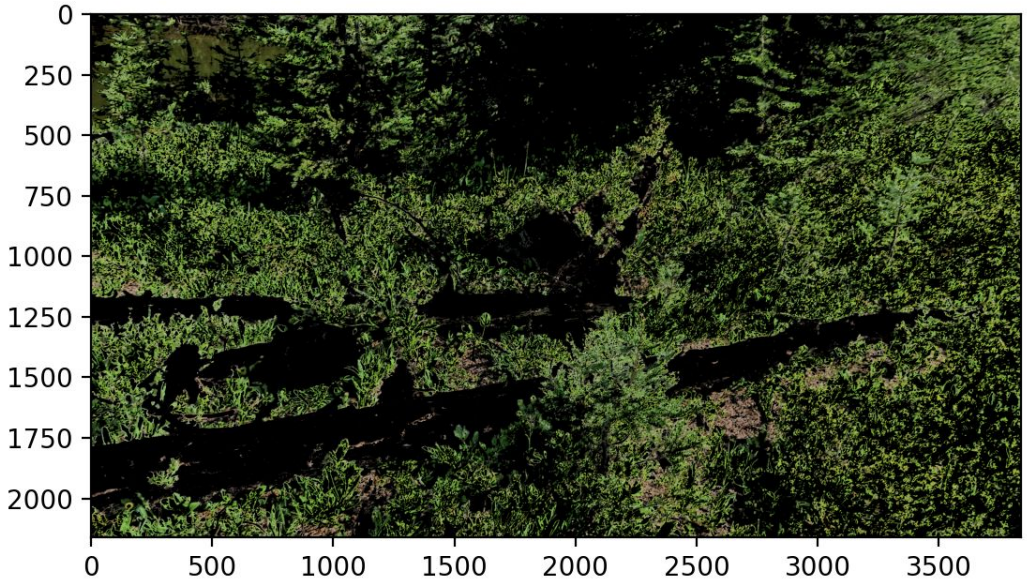

Figure 34: Example 4 Input Image for Depth

Future Plans

In order to continue to develop and build upon our research we must complete the following items:

1. Scale to categorize image static and depth. Doing this will allow the CNN mathematical model to take them in as parameters.
2. Determine the size of the image. The input data has a black bag in every image that can act as a control. In the research's current state, this control is neglected.
3. Making programs dynamic to average RGB or HSV image value. In current state, image processing bounds (ex. Low_hue and high_hue) are hard coded.
4. Developing a python program in conjunction with yolo to plot on an active ArcGIS map allowing for future modeling prediction.

References

- http://people.du.ac.in/~vbhatnagar/publications/ISAST_IS_1_08.pdf#page=36
- https://pjreddie.com/darknet/yolo/
- https://arxiv.org/pdf/1506.02640.pdf
- https://arxiv.org/pdf/1905.05055.pdf
- https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad
- https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006
- https://fas.org/sgp/crs/misc/IF10244.pdf