# Finding Waldo: An Investigation into Machine Learning for Object Detection

Jon Camarillo, Maria Contreras, Mayleen Cortez and Andrew Martos
Spring 2019, MATH 398 - Advanced Research Investigations
California State University Channel Islands

## 1. Introduction

In this paper, we discuss a project that we have worked on throughout the Spring 2019 semester at California State University Channel Islands as part of an undergraduate mathematics research course. Our project was to create a machine learning model that can detect the character Waldo from Martin Handford's *Where's Waldo* series. In the following sections, we discuss the background and motivation for the project, elaborate on the data collection and labeling process, briefly talk about the Single Shot multibox Detector algorithm, reflect on our results and review paths forward.

## 2. Background and Motivation

In this section, we present computer vision, machine learning and their relevance. We also introduce the use of Amazon Web Services (AWS) in our project. The section ends with a discussion about our project goals and the motivation behind this research.

2.1. **Computer Vision and Machine Learning.** Computer Vision (CV) is the field of research that investigates how we can get machines to do the high-level visual processes that humans naturally do. In other words, CV researchers work to teach computers to see and understand image data [1]. For example, when we go to the park, we can easily classify what we see: person, dog, tree, runner, child, etc. We can also identify where in the park they are. For example, we see the runner on the south end of the park running towards the north end and we can follow them with our eyes as long as nothing obstructs our view. For humans, classifying and localizing objects is a trivial task, but this is not the case for computers. One reason CV problems have yet to be fully solved is that our understanding of human vision and how it works is limited [1]. Richard Szeliski, research scientist and founding director of the Computational Photography group at Facebook, states that "perceptual psychologists have spent decades trying to understand how the visual system works [but] a complete solution to this puzzle remains elusive" [10]. Another reason that computer vision has proved far more difficult than previously thought is that the visual world is inherently complex. Any given item or object can look different under contrasting lighting conditions and there are often several objects in our visual field. These objects sometimes partially obstruct our view of other items [1]. Despite these problems, there has been progress made in CV. Szeliski's *Computer Vision: Algorithms and Applications*[1] highlights some of the high-level problems that have seen success, such as medical imaging and motion capture.

In 1955, John McCarthy of Dartmouth College created the term "artificial intelligence" to introduce a new field in computer science. This field researches how to create machines that demonstrate intelligence. Intelligence is the ability to observe one's surroundings and make decisions based on those observations [8]. At first, researchers in the field aimed to teach computers intelligence by focusing on teaching them how to think abstractly. In recent times, the main approach has shifted towards Machine Learning (ML), which is based on statistics more than logic and reasoning. According to Chris Meserole, an expert in artificial intelligence, "the core insight of machine learning is that much of what we recognize as intelligence hinges on probability rather than reason or logic" [8]. Recognizing someone, planning a trip, plotting a strategy—each of these tasks demonstrates intelligence. However, instead of depending on our ability to reason abstractly and logically, these tasks first rely on our ability

[1] Interested readers can go to http://szeliski.org/Book for more information Richard Szeliski and his book.

to accurately assess how likely something is—though, we don't always realize this is what our brain is doing. ML algorithms use data, experience, and interaction to learn how to do something better. This is, in some sense, mimicking intelligence [8]. In recent years, the use of ML in computer vision research has become a growing trend [9]. We will be using machine learning techniques to tackle the problem of teaching a computer to find Waldo.

2.2. **Amazon Web Services**. According to Google, cloud computing is "the practice of using a network of remote servers hosted on the Internet to store, manage and process data, rather than a local server on a personal computer." Cloud computing is preferred for training ML models because of the large datasets involved. Of the various cloud-computing services available (Google Cloud Machine Learning, IBM Watson, Azure Machine Learning), we will use Amazon SageMaker via Amazon Web Services (AWS) to train and implement our ML model. With SageMaker, we are able to build, train and deploy ML models with relative ease.

Another tool that AWS provides as part of their Machine Learning cohort is the DeepLens video camera. According to Amazon, DeepLens is "the world's first deep learning-enabled wireless video camera." When we train a model in SageMaker, we are able to send the model to DeepLens to test the model in real-time. We hope that once we have a viable model to detect Waldo from still images, we can then send our model to a DeepLens device, aim the camera at any Waldo puzzle and find him in real-time. Both SageMaker and DeepLens come with pre-trained models and examples that a user can work through before starting their own project. We have the options to create projects from scratch, use AWS built-in models, or modify pre-existing models.

Besides ML, Amazon Web Services also offers various Internet of Things (IoT) applications. The term IoT refers to all the things worldwide that can be connected to the internet [12]. This ranges from objects, such as smartphones or washing machines, to people. The IoT is ever-growing, and experts estimate that by 2020 there could be well over 100 billion connected devices all around the world [12]. As access to internet becomes cheaper and more widespread, and as the presence of smart technology grows in each home, it remains clear that the IoT is here to stay. AWS presents solutions to connect our devices and to collect, store and analyze device data. "Smart" devices combine the device in the IoT with ML to improve our daily lives. For example, using SageMaker, we could train a model to detect whenever an individual leaves home and when they return home. We could then further utilize DeepLens to test our model with real-time video feeds. Finally, we can apply AWS IoT services to connect the model to a video camera overlooking the driveway. We may then sync what it detects with the lock on our front door so that when the model observes that our car has left the driveway, it can ensure the front door is locked. Of course, this is just a simplified example of the capabilities of machine learning and the IoT.

2.3. **Goals and Motivation**. The primary goals of our research project can be summarized as follows:

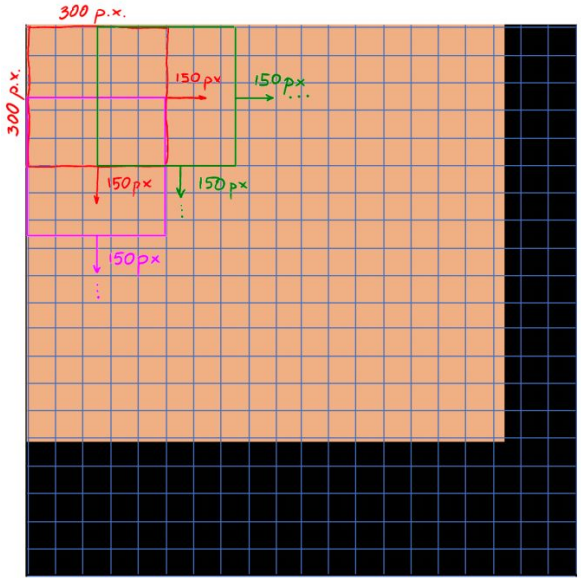
1. To create a Machine Learning model that uses the Single Shot multibox Detector algorithm [2] to find the character Waldo from the children's book series *Where's Waldo* with at least 90% accuracy
2. To train and implement our machine learning model using AWS as our platform
3. To explore the applications of our work, i.e. to ask the question, "How can we extend or modify our model to provide insight to currently open research problems in STEM?"
4. To increase student interest in computer science, data science and mathematics research

The third and fourth goals are also defining motives for this project. Object detection, classification and localization models are relevant in many industries and fields, including surveillance, security and healthcare[2]. We hope that our model can be extended to research problems in Computer Vision, such as finding a specific person in a crowd. We also hope that our project helps to generate excitement about computer science, data science, and mathematics among students and the general public. We chose the character Waldo because the *Where's Waldo* puzzles have intrigued both children and adults worldwide since their inception in 1987. Waldo, being a fun and familiar character, will be an attention-grabber at presentations and talks, giving us the chance to introduce a diverse audience to research in machine learning and computer vision.

## 3. Data Collection

This section describes the process of preparing our data and all that this encompasses, as well as the essential steps that lead up to administering the program used to train our model. In addition to discussing the methods used to prepare our data, we outline our incorporation of Amazon Web Services for this project.

3.1. **Images and Augmentation**. We modified an AWS SageMaker object detection sample project[3] to fit our needs. Using the tutorial provided, we split the project into four phases. For the first phase, we collected high quality 300px by 300px (abbreviated as 300x300 throughout the rest of this paper) images containing Waldo. Some of these images we collected by scanning the puzzles from [5,6] and the rest from [16,17]. The second phase required us to label each image with a bounding box around Waldo. The third phase involved creating JSON files containing information for each image, such as the bounding box coordinates and image dimensions. A JSON file has the extension .json but is just a text file with information ordered in a specific way [13]. The last phase was to train the model on AWS Sagemaker using the images and their JSON files.




[2] More information on the applications of computer vision can be found at https://emerj.com/ai-sector-overviews/computer-vision-applications-shopping-driving-and-more/

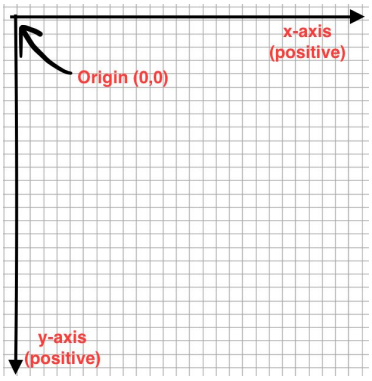
[3] To view the sample object detection project we followed, view this link: https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/object_detection_pascalvoc_coco/object_detection_image_json_format.ipynb

Initially, our dataset contained images of various dimensions, but the underlying object-detection algorithm processes either 300x300 or 512px by 512px images. When we input an image of different dimensions, the algorithm shrinks or stretches the image to the desired dimensions. As a result, the images are distorted. To solve this problem, we used a program[4] to chip each Waldo puzzle image into 300x300 slices. First the image was padded with a black border on the bottom and right side of the image. This ensured that the width and height of the image were divisible by 150px and that every chip in the image would be of size 300x300. We start chipping on the top-left corner of the image. Then we move down 150px from the first chip to chip another 300x300 piece of the image. To chip the next row, the program moved back to the top of the image and over 150 pixels to the right. We continued this process until reaching the bottom right-corner of the image. An illustration of this process is shown below. In the end, only images that contained Waldo or a portion of Waldo were kept. This also helped us augment the dataset.

Since the object detection models we studied were trained and validated with huge training sets, it seemed necessary to have hundreds or thousands of Waldo images to create our model. Through further studying, we learned that having a copious amount of images was not necessary for us to build a successful model. However, this may not be the case for every Machine Learning model [14,15]. In some cases, such as with nonlinear algorithms, the more data the better [15]. In other cases, such as the one that Zach Monge describes in his article, a smaller training set returns just as good results as a larger training set [14]. In the end, it comes down to the diversity of the model's input when used for inference-making, and how bad it will be if the model makes a mistake. The higher the stakes, the more data is necessary. We originally had 300 images of Waldo. Using horizontal flips and the chipping described above, we were able to add to the dataset. However, in our trials to augment the data, we were unable to successfully debug the code. After chipping the larger image, we had trouble with calculating where the bounding boxes should be placed in the chips. Hence, our final set contained only 160 images. The success rate for the validation step was determined by a score called the mean average precision. Validation was conducted using 40 images and the final mAP for our best model was 0.929, which was sufficient enough to provide us with meaningful results based on the nine images that we used for inference. More detailed information about mAP and the results of our model will be discussed in section 5.2

### 3.2. **Labeling and Format**.

To label each image in the training and validation sets, we first used an online program[5] to draw a box around Waldo. This program also gave us the coordinates of the box, using the convention of treating the image like a coordinate plane with the origin in the upper left corner (an illustration of this is shown below).




Using this information, we were able to determine the dimensions of the bounding box. We then gathered the image's dimensions and put it all together in a Microsoft Excel spreadsheet. Finally, we created a python script to transfer the image information contained in the .csv file to a JSON file. To do this, we modified the code provided in the AWS JSON documentation[6]. Our script loops through every row in the .csv file and grabs information from the appropriate column to put that information

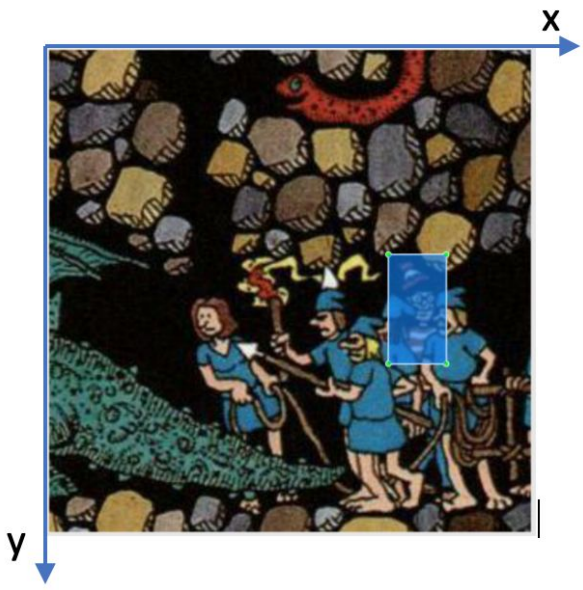
[4] Refer to "ImagePadding.ipynb" at: https://github.com/mayscortez/Finding-Waldo
[5] To access the program, view the following link: http://nicodjimenez.github.io/boxLabel/annotate.html
[6] Found here: https://docs.aws.amazon.com/sagemaker/latest/dg/object-detection.html

into the .json format, instead of hardcoding the values into the .json format manually. Later on, we used a different tool, called LabelImg[7], to label the images in our dataset. With this program, we can open images, place a bounding box around the image, and record the coordinates onto a table using Excel (shown below).

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Filename | $x_1$ | $y_1$ | BB Width | BB Height | Image Width | Image Height |
| 2 | w1.jpg | 47 | 19 | 54 | 67 | 300 | 168 |
| 3 | w2.jpg | 1540 | 367 | 43 | 46 | 1843 | 1309 |
| 4 | w3.jpg | 1388 | 465 | 37 | 38 | 72 | 72 |
| 5 | w4.jpg | 372 | 381 | 17 | 22 | 800 | 600 |

Since LabelImg records image information onto an .xml file, we also have the option of using online tools to convert the .xml files into JSON files. However, for this project, we chose to use the script described above to create the appropriate .json files. Below is an example of a labeled image and the contents of its corresponding JSON file.

```
{"file": "w1.jpg",
  "image_size": [{"width": 300.0, "height": 300.0, "depth": 3}],
  "annotations": [{"class_id": 0, "left": 212.0, "top": 128.0,
       "width": 35.0, "height": 67.0}],
  "categories": [{"class_id": 0, "name": "waldo"}]}
```




The first line of text in the JSON file specifies the filename of its corresponding image. The second line of text contains the image dimensions, which in our case is 300 pixels by 300 pixels. A computer interprets a color image as a three-dimensional matrix, one for each color channel (Red, Green, Blue). This is where the "3" comes from. Next, we have the "annotations" of the image. Class_id is either a 0 for "waldo" or 1 for "not waldo." Next is the top left coordinate, width and height of the bounding box. The final line gives the full class_id.

3.3. **Amazon Web Services**. The required support services for AWS SageMaker are EC2 and IAM (Identity and Access Management) . In general, S3 is used for storing and retrieving data for use in other Amazon computing services. In order for our images and JSON files to link with AWS, everything was stored in S3. To store the data into S3, we placed our images into folders (called buckets). Since the

[7] To view information about LabelImg, view: https://github.com/tzutalin/labelImg

data was to be used in conjunction with SageMaker, the prefix "sagemaker-" was required in the naming of the bucket. For our project, we named the bucket "sagemaker-detectwally."

For computing power, AWS EC2 provided the necessary resources for proper use of the SageMaker notebook, training, and prediction. The SageMaker notebook required an instance of type *ml.t2.medium*, which has relatively low computing power and is less costly. For training, we required the instance type *ml.p3.2xlarge*. This instance type was chosen because it retains more workload memory, which was necessary in this step. As a means to control the use of resources, Amazon limits the number of times a user can utilize these instance types. For the *p3.2xlarge* instance, we had to request limit increases by contacting Amazon support. To start training the model, we had to specify IAM roles to enable communication between S3 and Sagemaker. Aside from our use case, for large-scale projects, IAM allows managers to control who has access to which resources and services.

The interaction of AWS with other services allows for strict control of powerful resources. Once all of the data was placed in S3, training using SageMaker was quick and seamless. During the inference step, we used *ml.m4.xlarg*e instances instead of *p3.2xlarge* because they process less compute-heavy tasks and thus reduce costs.

## 4. Single Shot multibox Detector

In this section, we outline the Single Shot Multibox Detection algorithm [2] that we use to detect and find Waldo.

4.1 **Overview**. SSD uses the VGG16 convolutional neural network as its base. VGG16 tells us that there is a Waldo in the image but it does not tell us where he is. This is where SSD comes in. SSD will find the object we are looking for and place a bounding box around it. Object localization and classification are done in a single forward pass of the network, meaning that we only move forward through the network once. The bounding box regression used is derived from MultiBox. During training, SSD takes an input of images and their associated ground truth boxes. SSD associates a set of default bounding boxes to every feature map cell for multiple feature maps. These default boxes are carefully and manually chosen.

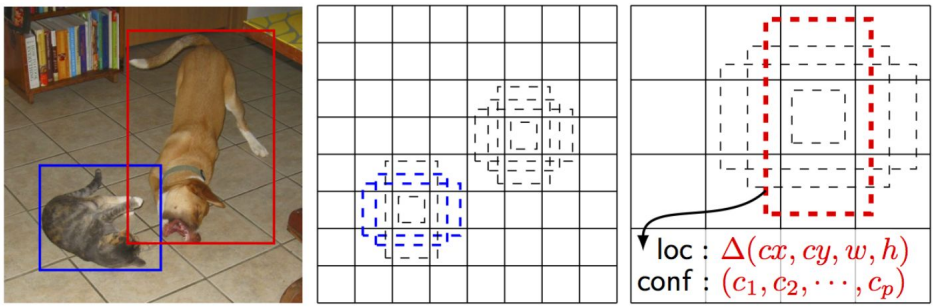



Figure from [2].

4.2 **Model**. SSD is based on a feedforward neural network. Feedforward neural nets take input from the start of the network and pass that input through each layer of the network. Shape offsets and confidences are predicted for each object in the default boxes.The overall objective loss is computed by taking a weighted sum of the localization loss and the confidence loss. The localization loss tells us how close we

are to the original bounding box and the confidence tells us how sure the model is that what we are detecting is actually Waldo. The first few layers of SSD are based off of the VGG16 network. This network is proficient at classifying images. VGG16 works by performing convolutions on the input image. Convolutions are performed by applying a filter or kernel of a fixed size to the entire image. These filters are mathematical operations that bring out certain features in the image.

4.3 **Optimizing Loss.** As stated before, the overall objective loss is a weighted sum of the localization loss and confidence loss.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

N is the number of matched default boxes and if N = 0, we set the loss to 0. The localization loss is calculated by using a smooth L1 function that calculates the loss between the predicted box (*l*) and the ground truth box (*g*). These parameters include offsets for the center (*cx, cy*) of the default box (*d*) and its width(*w*) and height (*h*) which are also regressed.

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

We calculate the confidence loss through a "softmax loss function over multiple classes of confidences"[2]. We set the weight term ($\alpha$) to 1 by cross validation.

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

4.4 **Training**. When SSD is being trained, the default boxes that correspond to the ground truth boxes need to be determined. We are selecting from default boxes that vary over location, aspect ratio, and scale for each ground truth box. This matching strategy is done by matching each ground truth box to any of the default boxes with the best Intersection over Union ratio (IoU, also known as jaccard overlap). What this means is that the IoU is calculating the overlap between the ground truth boxes and the default boxes. This is important because if there isn't enough overlap, we know that the default box is not close to the position of the ground truth box. The default boxes are then matched to any ground truth boxes with an IoU higher than a threshold of 0.5. What this does is simplify the learning problem and allows the network to "predict high scores for multiple overlapping default boxes rather than requiring it to pick only the one with maximum overlap"[2].

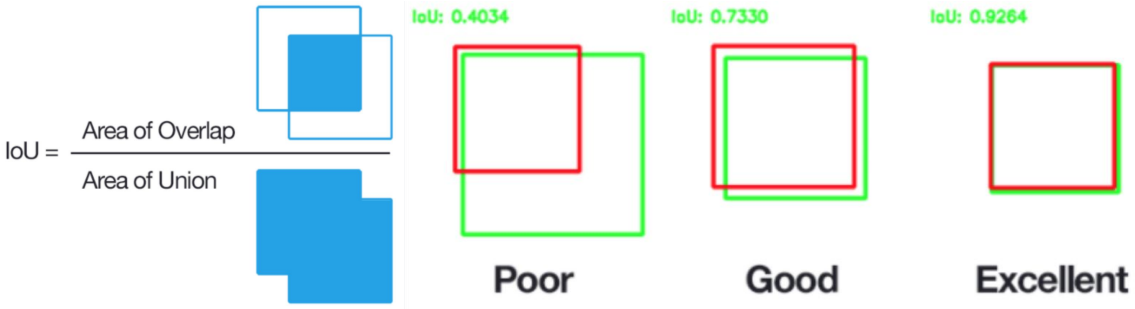



Figure from [pyimagesearch.com].

4.5 **Hard negative mining**. After the matching strategy, most of the default boxes are negatives (meaning those boxes do not have significant overlap with the ground truth box). This becomes even more apparent when the number of default boxes we have is large. This gives us a "significant imbalance between positive and negative training examples" [2]. Hard negative mining is done to sort through all of the delfaut boxes with the highest confidence loss so that we don't end up using all the negative examples. Then, we pick the top ones so that the ratio between the negatives and positives is at most 3:1. This leads to "faster optimization and more stable training" [2]. Having negative samples is also important because the network needs to be explicitly told what isn't Waldo and what constitutes as an incorrect detection.

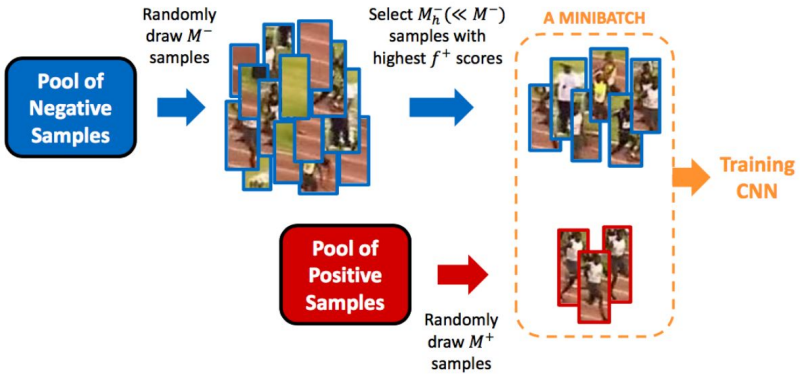



Figure from [towardsdatascience.com].

## 5. Results

In utilizing the Single Shot multiBox Detector algorithm, a defining feature for when a detection is present is a box displayed around predicted objects accompanied with a prediction accuracy. This can then be used to relate the quality of our data and testing metrics. To gain a better understanding of Machine Learning and testing practices, we ran samples using Tensorflow and AWS, and we discuss results obtained for both.

5.1. **Tensorflow.** Using the TensorFlow Object Detection API, we were able to obtain between 86% and 99% accuracy with objects that were detected as Waldo. We also didn't falsely identify any other objects as Waldo. These results were achieved through image augmentation by padding the images to a size that was divisible by 150 pixels, chipping the images down to 300x300 chips, and shifting over by 150 pixels after every chip. Image padding and chipping is standard for datasets that contain large images since simply resizing warps the image and important information may be lost. We used a separate dataset for the TensorFlow Object Detection API that consisted of 44 images. After performing

the data augmentation, the data count increased to approximately 4,800 images with 124 of them having Waldo. The reason we have so many images without Waldo in them is because the Waldo puzzles are very large, but only one tiny piece of the image contains Waldo. Though many images did not contain Waldo, having them as part of the data did not have a significant effect on the results.

5.2. **Amazon Web Services.** In S3, our 160 images were split and stored in two folders. Of our images, 120 were stored in "train" and 40 were stored in "validation." The corresponding JSON files were stored in folders named "train_annotation" and "validation_annotation." We trained the model with a Python program[8] using a Sagemaker notebook instance. The hyperparameters that we manipulated during our test runs were the batch size, epochs, and learning rate, as these were three that, according to the AWS Sagemaker example we used as a template, most influenced the results [18].

The batch size refers to a subset of images that the model trains on. Epoch refers to the number of times the model finishes processing all the batches in the training set. For our set of 120 images, we chose a batch size of 10, and epoch of 120. Doing so meant that the model trained on 10 images before updating its training statistics. Therefore, in one epoch, it trained on 12 batches, each containing 10 images. It did this 120 times. The training statistic that we paid most attention to was the mean Average Precision (mAP). To discuss mAP, we must first define *precision* and *recall.* Precision is the percentage of correct predictions and recall measures how well we find the positives [7]. Mathematically, we can calculate each by the following:

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}.$$

We can then plot the precision against the recall value to form a curve. After smoothing out the curve, we calculate the Average Precision (AP) by finding the area under the curve. When there is more than one category, we have multiple AP. If we take the average AP over all the categories, we are left with the mAP [7].

To begin building a reliable model, we trained twelve models using different combinations of batch size, epochs, and learning rate. Of the twelve, we chose two that had the best mAP score. The two that we selected are the first two listed in Table 1. Using these two models, each was trained two more times without changing their hyperparameters so as to not influence any change to the first score. The model with the higher average mAP was the one we used in our final check for quality. In our case, the model with the original mAP of 0.921 was chosen. To ensure consistency in the base model's mAP score, we trained it six more times. At this point, the model had been trained nine times. The results for the nine training instances are provided in Table-2[9]. We selected the one with the highest mAP and used it for detection during the inference step.

[8] To view this code, see "waldoProjectz.ipynb" at: https://github.com/mayscortez/Finding-Waldo.git
[9] To obtain more information on the trained models, view: https://github.com/jonjxtsu/waldoProject

Results of Twelve Trained Models before Selection of Top-2 Based on mAP Scores

| | Learning Rate | Batch Size | Epochs | mAP |
|---|---|---|---|---|
| 1 | 0.00080 | 10 | 120 | 0.940 |
| 2 | 0.00082 | 10 | 120 | 0.921 |
| 3 | 0.00080 | 10 | 90 | 0.911 |
| 4 | 0.00082 | 10 | 100 | 0.910 |
| 5 | 0.00080 | 15 | 120 | 0.893 |
| 6 | 0.00080 | 5 | 120 | 0.893 |
| 7 | 0.00078 | 10 | 120 | 0.887 |
| 8 | 0.00080 | 10 | 120 | 0.866 |
| 9 | 0.00080 | 10 | 120 | 0.858 |
| 10 | 0.00080 | 10 | 150 | 0.856 |
| 11 | 0.00085 | 12 | 120 | 0.853 |
| 12 | 0.00080 | 10 | 90 | 0.834 |

Table 1

Nine Training Instances of the Best Model and Corresponding mAP Score

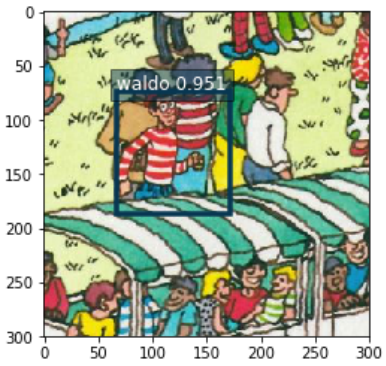
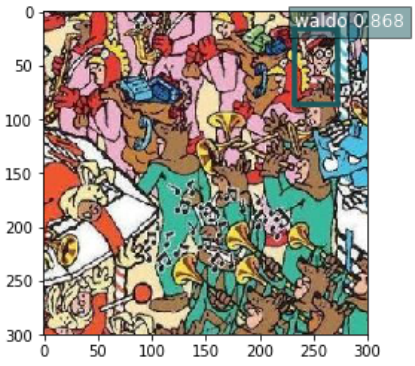
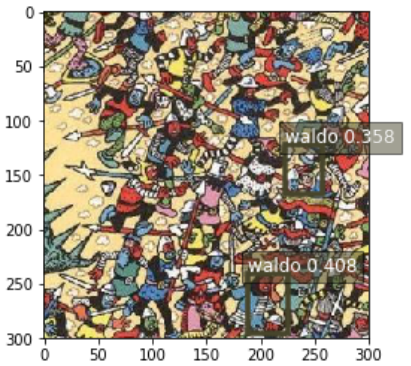
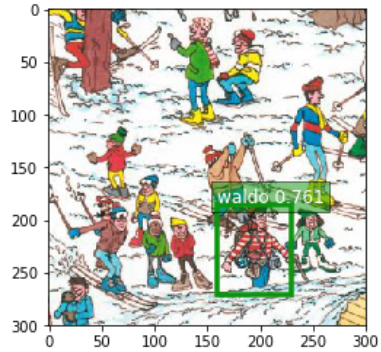
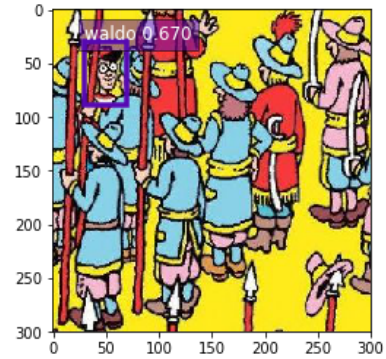
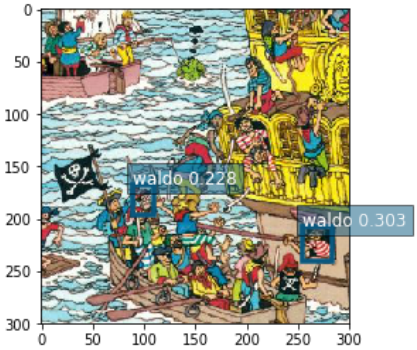
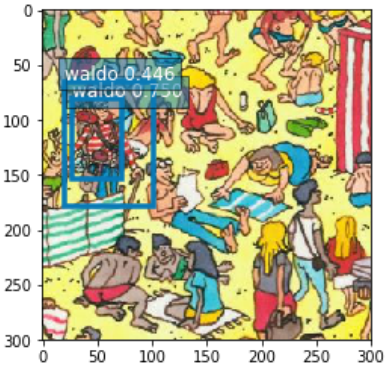
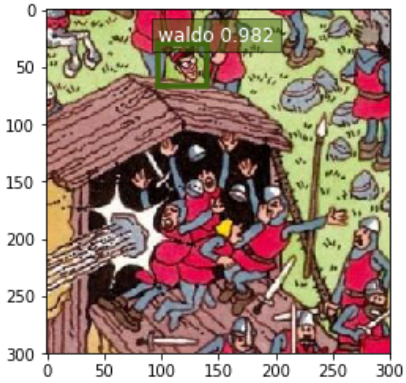
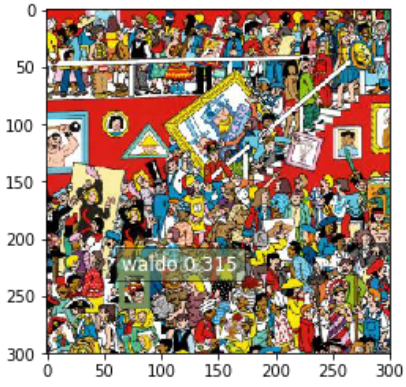
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| mAP | 0.822 | 0.830 | 0.858 | 0.884 | 0.895 | 0.900 | 0.918 | 0.921 | 0.929 |

Learning Rate = 0.00082; Batch Size = 10; Epochs = 120;

Table 2

To enhance the reliability of our results, we tested the model on nine images, ensuring they were ones not used for testing or validation. The nine images were separated into three sets, each set having its own level of difficulty. Difficulty level was determined by image features such as the size of Waldo's body, head size, resemblance to other image entities, and variability of color and patterns. Images that were deemed easily detectable contained clear representations of Waldo's entire body. Images of moderate difficulty showed only Waldo's head with a moderate to significant amount of variability but large enough so as to suggest little obstruction. Images that were considered very difficult exhibited features where Waldo's head was smaller in size compared to those of moderate difficulty, variance of colors and patterns were significant, and entities with similar characteristics as Waldo (red-and-white-striped garments, similar facial shape and hair color) were present.

The following nine images illustrate our results for the inference step.

| EASY | MEDIUM | HARD |
|---|---|---|
|  |  |  |
|  |  |  |
|  |  |  |

We can see that the detection probability throughout all difficulty levels were not consistent. There is a clear decrease in detection probability going from easy to hard difficulty. To improve detection, perhaps a more robust data set containing more images of varying complexities would suffice. Manipulating the parameters might also lead to improvement. Although our detection probabilities were low, we can lend credit to the fact that the model, by observation and not considering the probability, has the ability to detect Waldo accurately. By applying the improvements previously stated, the probabilities and confidence of the model would surely increase.

5.3. **DeepLens.** To use the DeepLens device, we first have to register the device. When registering, we give the device a name and set up some permissions, or IAM roles, for DeepLens. Then, we download a certificate unique to our device and that registration. We must make sure to have a stable internet connection to connect the DeepLens video camera to. Once the device is registered, we can deploy a computer vision project to the device. As mentioned in Section 2.2., we have the option of using sample

projects, modifying sample projects, or creating a project from scratch. A project consists of a machine learning model and a lambda inference function. The machine learning model can be made in SageMaker or in another framework, but depending on where the model comes from changes certain lines in the lambda function script. The lambda function is what applies a machine learning model to a video feed to make inferences in real-time. There are tutorials on the AWS website that guide users step by step in making the lambda function. After creating the model and the inference function, both must be imported and associated, or attached, with a project. When the project has been deployed to the device, the user can view the camera feed live from their computer. We are currently in the process of deploying our project to the DeepLens device.

## 6. Future Work

In this section, our next steps and ideas of what may come from our results and experience are discussed.

- We will explore Amazon Web Services' DeepLens, a video camera that integrates with the cloud computing platform that AWS provides, to determine how we can incorporate it into our current project. We hope to make the jump from running our model on still images to finding Waldo through a live video feed.
- In line with our project goals discussed in Section 2.3, we will consider the extension of our research into other areas of mathematics and its applications in machine learning, object detection, and data analysis. We will consider collaborating with the Environmental Science and Resource Management department on a research project involving object detection.
- We also will continue to spread awareness to the scientific community of the various useful applications that computer vision and Machine Learning provide in research by presenting our work at conferences.

## 7. Conclusion

When we began this project, we did not anticipate what our venture into CV and ML would lead to. We had an idea: to teach a computer to find Waldo. However, we knew almost nothing about Amazon Web Services (AWS) and were unfamiliar with the Single Shot multibox Detector (SSD) algorithm. Learning how to train a Machine Learning model on Amazon Web Service's Sagemaker proved to be difficult, but not impossible. SSD was tough to understand at first, but persistence proved the algorithm's mechanisms were accessible to us. Our first attempts at training the model were unsuccessful, but we learned from our mistakes and fixed them to create a model that finds Waldo with high accuracy. We've been able to create a project that has implications in the use of object detection and Machine Learning in research, all while making the subject material accessible to varying audiences. As we continue to explore AWS and DeepLens, we hope that making the leap from finding Waldo in a still image to finding Waldo in a real-time video setting will increase the usability of our model in other research applications and grab the attention of audiences whenever we present out work. All in all, this experience has strengthened our group work skills, reinforced our communication skills, enhanced our research skills and augmented our technical skills. In a world where machine learning, object detection and computer vision are hot topics, being familiar with the technical jargon and having basic skills or knowledge pertaining to these areas will prove useful in our future academic, research, and career endeavors.

## 8. References


[1] Brownlee, Jason. “A Gentle Introduction to Computer Vision”, *Machine Learning Mastery*, 19 March 2019. Online.

[2] Berg, Alexander C., et al. *SSD: Single Shot MultiBox Detector*. 2016, arxiv.org/pdf/1512.02325.pdf.

[3] EpochFail., Wikimedia, https://creativecommons.org/licenses/by-sa/4.0/legalcode.

[4] Grover, Prince. “Evolution of Object Detection and Localization Algorithms.” *Towards Data Science*, Towards Data Science, 15 Feb. 2018. Online.

[5] Handford, Martin. *Where’s Waldo?* Candlewick Press, Massachusetts. 2007. Print.

[6] Handford, Martin. *Where’s Waldo? In Hollywood.* Candlewick Press, Massachusetts. 2007. Print.

[7] Hui, Jonathan. *mAP (mean Average Precision for Object Detection).* 2018. Online.

[8] Meserole, Chris. “What is Machine Learning?”, *Brookings,* 4 Oct 2018. Online.

[9] Sebe, N., et al. *Machine Learning in Computer Vision.* Springer, The Netherlands. 2005. Print.

[10] Szeliski, Richard. *Computer Vision: Algorithms and Applications,* pp. 3-19, 2010. Online. http://szeliski.org/Book

[11] Yali, Amit and Felzenszwalb, Pedro. *Object Detection*, pp. 537-542, 2014. Online. DOI:10.1007/978-0-387-31439-6_660

[12] Morgan, Jacob. “A Simple Explanation of the ‘Internet of Things’.” *Forbes*, Forbes Media LLC, 13 May 2014. Online.

[13] Amazon Web Services. “Object Detection Algorithm.” *Amazon SageMaker Developer Guide,* Amazon Web Services, 2019. Online.

[14] Monge, Zach. “Does Deep Learning Really Require ‘Big Data’? -- No!” *Towards Data Science,* Medium, 19 Aug 2019. Online.

[15] Brownlee, Jason. “How Much Training Data is Required for Machine Learning?” *Machine Learning Mastery,* 24 July 2017. Online.

[16] Magajna, Tadej. “Here is Wally.” *GitHub.* 12 April 2018. Online.

[17] Constantinou, Valentino. “Hey-Waldo.” *GitHub.* 5 Feb 2019. Online.

[18] Arpin, David. “Amazon SageMaker Object Detection using the Image and JSON format.” *Github*. Amazon Web Services - Labs. 26 July 2018. Online.