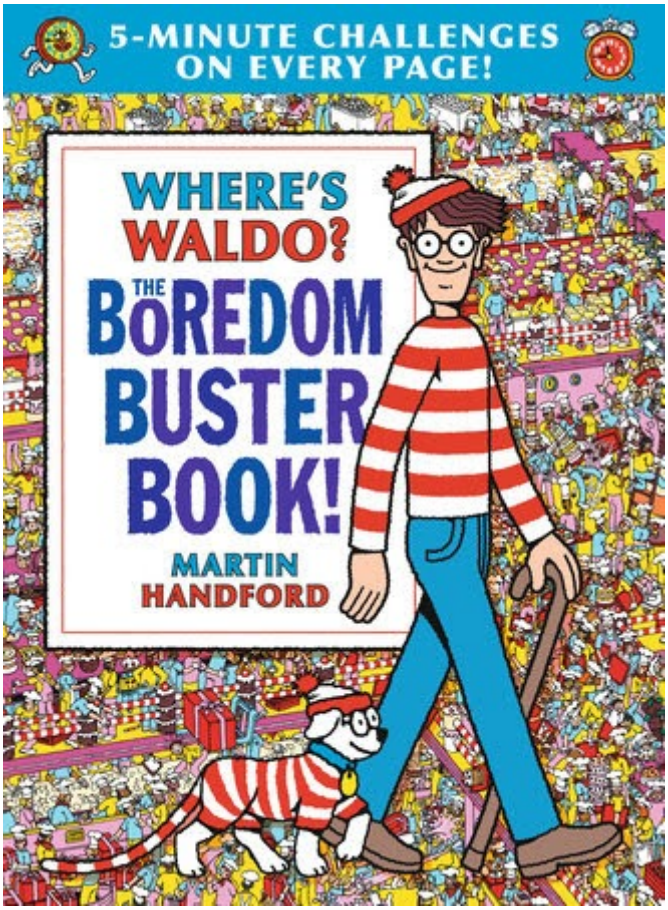
I have always thought that computer vision is a super interesting field of computer science. The idea that computers can understand images and automate visual tasks is crazy to me, and I wanted to learn more about it. When I took a Deep Learning and Neural Networks class this past fall, I was instantly intrigued by Convolutional Neural Networks (CNNs). CNNs can do lots of cool things regarding image recognition, so for my final project I decided to do a fun experiment on a classic visual challenge - could I make a program that solves 'Where's Waldo' puzzles for me?



A 'Where's Waldo' book, a strong source of childhood frustration

As a kid, I enjoyed doing 'Where's Waldo' books. For those who do not know, the premise is quite simple. Each page of the book is an elaborate scene full of people and commotion, and amongst all that action, there is one Waldo. He is small and purposefully hard to find. While I am now at an age where I can quickly solve 'Where's

Waldo' puzzles (most of the time, some are truly unfair and take me ages) I wanted to know how a computer would fare. With the motivation of making a computer successfully find Waldo, I started working.

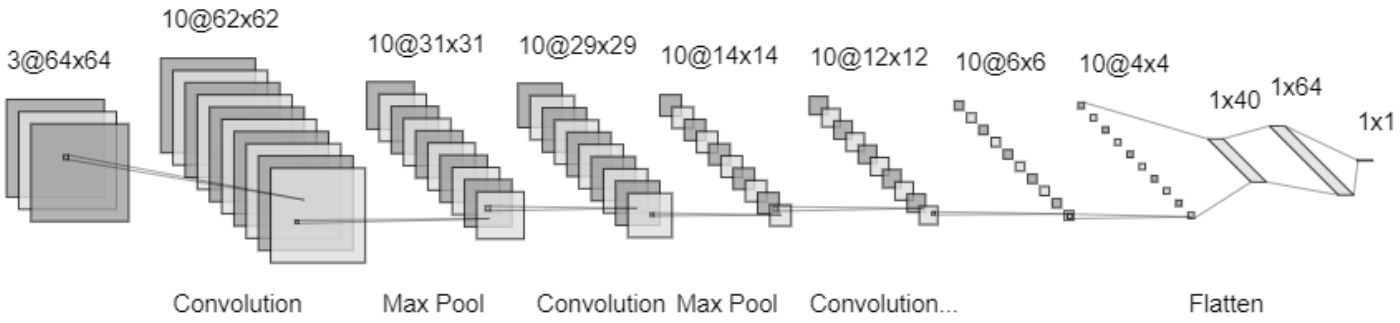
I wanted to train a CNN to look at an image and output a probability that Waldo is in that image. Meaning, for the image of Waldo (like below) we want our CNN to output a high probability:

CNN: "There is a 99% chance that Waldo is in the image"

On the other hand, we want it to output a very low probability when it looks at a 'non-Waldo' image - i.e. a different part of the puzzle **not** containing Waldo.

CNN: "There is a <1% chance that Waldo is in these images"

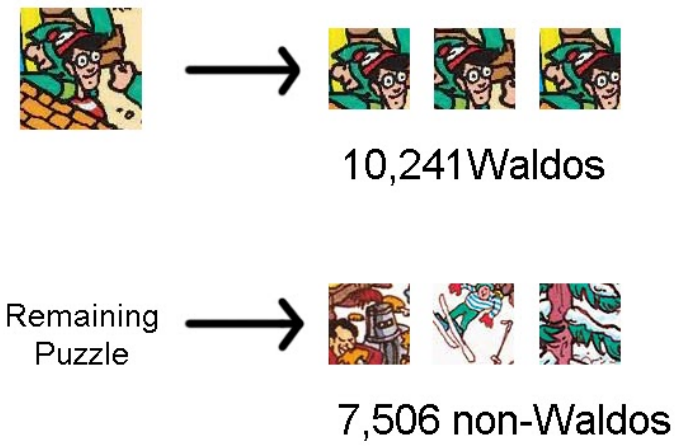
This is well within the realm of possibilities for a CNN, and I will explain why without going into too much technical jargon. Our CNN works by applying multiple *filters* to an image we give it. These filters search over the entire image for different things: horizontal edges, vertical edges, specific lines, etc. Each one of these filters outputs a *feature map*, which is exactly what it sounds like: a map of the features the filter was looking for and where they are in the image. If you repeat the process over and over again, running filters over the original image and over feature maps, our CNN "learns" to see and understand what is present in the image. At first, these filters do not know what to look for, but through repeated trials, it will learn what features make up Waldo (glasses, stripes, his face, etc.), and when it detects these specific features, the network triggers activations that output a higher probability of that image being Waldo.



The Waldo CNN: stacks of feature maps funneling into one single probability output

Before building the CNN above, I had to consider a big problem: the dataset. When training a CNN, or any neural network for that matter, the more *balanced* your data is and the more data you have, the better. Usually, 1000s of both 'positive' (in our case, images of just Waldo) and 'negative' (parts of the 'Where's Waldo' that do not contain Waldo) images are needed to properly train a CNN to predict whether a given image has Waldo in it. If the dataset is *unbalanced*, say we have ~20 Waldo images and 1000+ non-Waldo images, it will always predict an image does not contain Waldo (because 980/1000 of the images are not Waldo, and 98% is a good accuracy!).

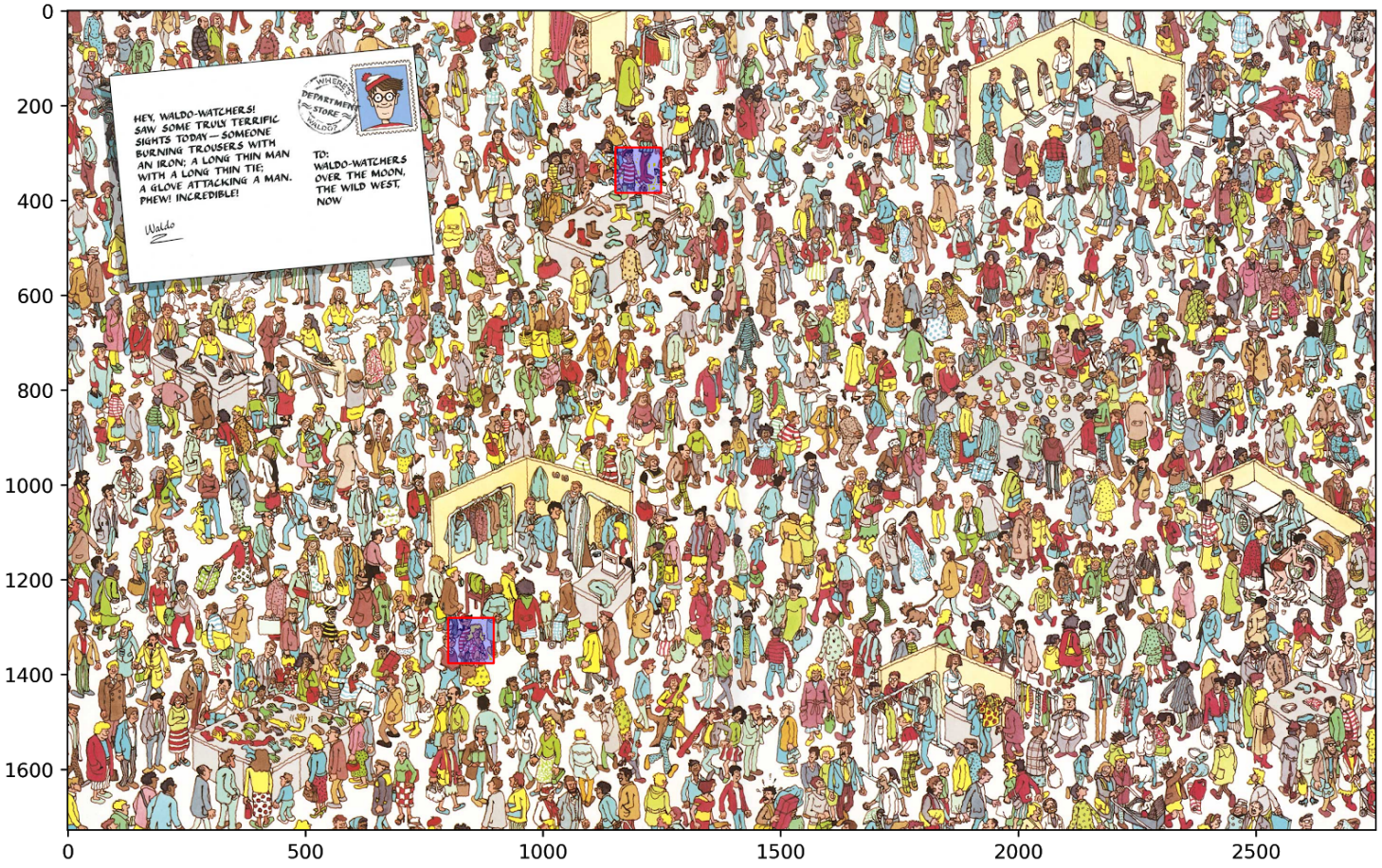
There are only so many 'Where's Waldo' puzzles available online, and for each one of those puzzles, there is only one small section of the image containing Waldo. Figuring out how to acquire thousands of images of not-Waldo and Waldo was the first hurdle to overcome. I ended up writing some code that takes a 96x96 image of Waldo and makes as many 64x64 sub-images as possible. I also performed some data augmentations on each 64x64 like scaling and vertical flip. After manually finding Waldo about ~4 times and cropping him out into a 96x96 image, my code generated thousands of 64x6 sub-images. For my non-Waldo images, I cropped out Waldo from a puzzle and generated 1000s of sub-images in the remaining space.



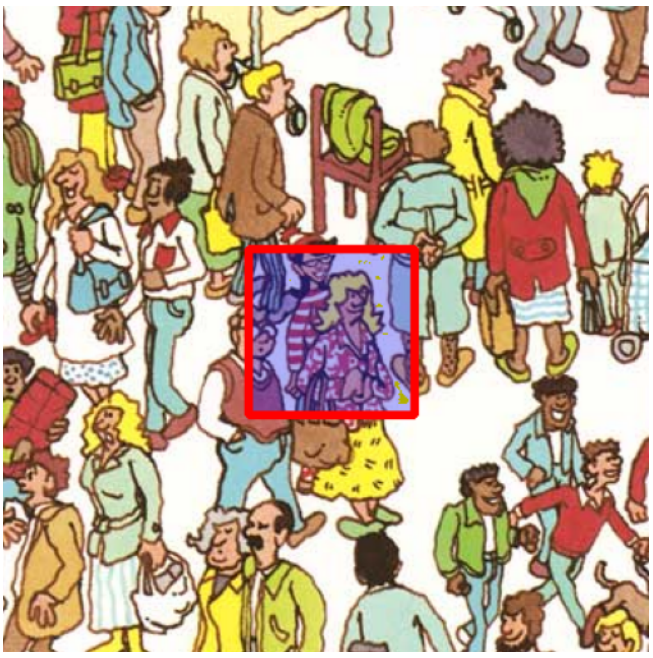
The dataset

Finally, we were able to train the network. It ended up working great, and when given an image, the CNN could predict whether or not Waldo was in it with 99%(!) accuracy. This was a nice checkpoint, but there were more steps that needed to be taken to actually locate him within a puzzle. Object detection, the act of finding an object within a larger image, can be pretty difficult to implement. The state of the art object detection techniques that are the quickest and most precise, such as the YOLO model, require special annotated data that I do not have the time nor the means to create. I settled for a slower, but good enough 'sliding window' algorithm. In essence: we slide a 64x64 window over a 'Where's Waldo' puzzle and have our CNN predict whether or not what the window is seeing contains Waldo. Checking every 64x64 square in the full puzzle would take ages, so I added a *step* parameter allowing our sliding window to check fewer places. It is important to keep the step relatively small so it does not 'step' over Waldo.

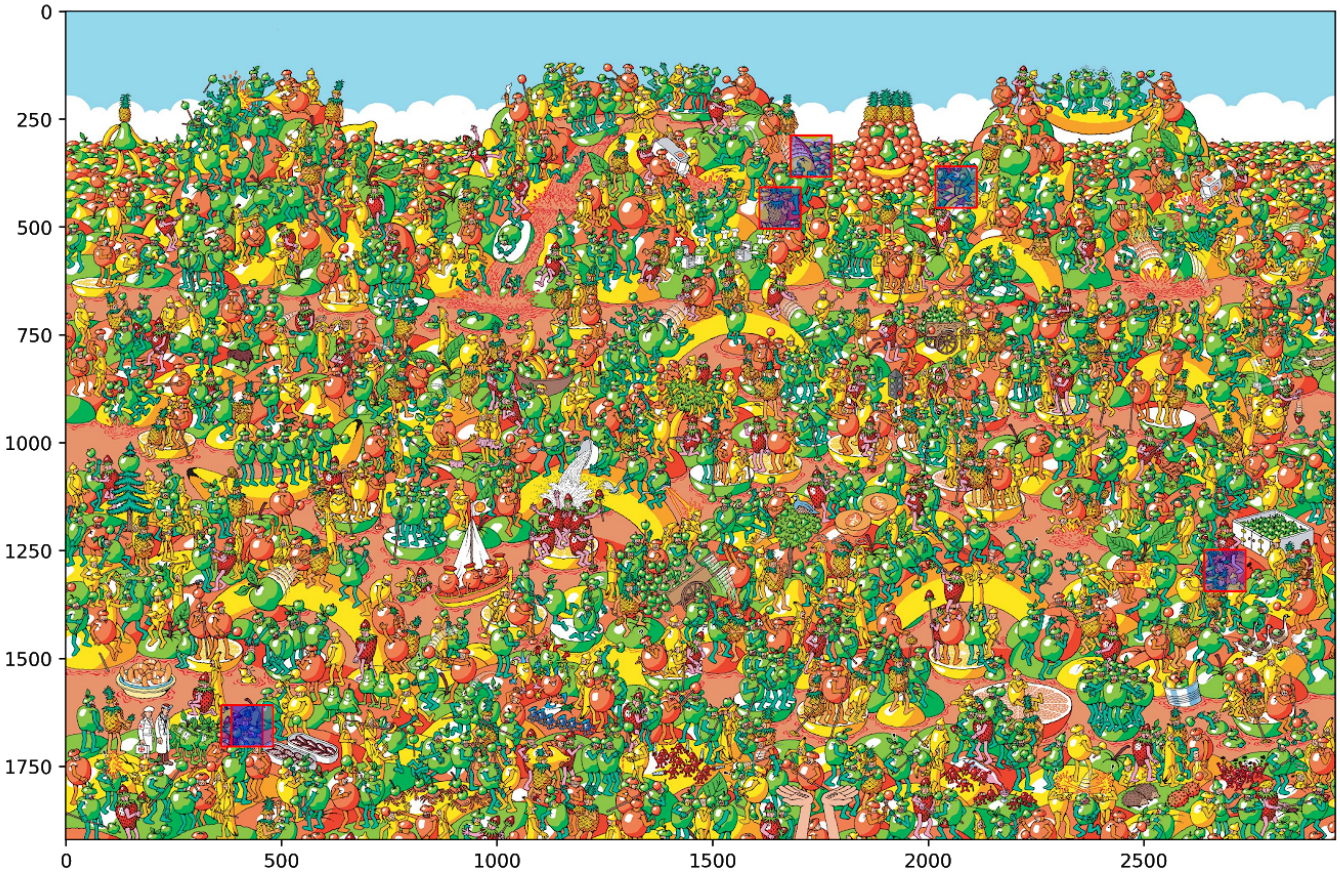
With the network trained and the sliding window in place, I was ready to test it out! Here is how it does on a Waldo puzzle it has never seen before (was not a part of the training set):



It identified two 64x64 windows where it is 99% sure there is a Waldo. Let's zoom in!



It found Waldo and his female counterpart Wendy! The neural network checked 4,558 images and returned only one false positive (if you even consider Wendy a false positive - it is actually considered a bonus to find her according to the book). Not bad at all. Let's do one more.



A much bigger image... let's see what it found. Enhance!

We found Waldo, Wendy, and three false positives. I initially set the step size too big and my sliding window stepped over Waldo the first run through. I decreased the step size, meaning the sliding window checked more locations (less likely to miss Waldo) but it took much longer as a result. For a step size of 24 (meaning the sliding window checks a 64x64 area, then steps 24

pixels), our sliding window and network had to check 9,594 locations. With only three false positives, however, means our network carried a 99.9% accuracy!

Given that this was a week-ish long project, I am very pleased with the results. Particularly, the accuracy of the network was the highlight for me, especially considering that 'Where's Waldo' puzzles are full of people and objects that intentionally look like Waldo to try and trick the reader. The biggest thing I would like to improve at this point is speed. I tried a simpler network with fewer hidden layers, but I ended up using fewer parameters the more convolutional layers I added. This is because the bulk of the parameters comes from the *flattening* of the final convolutional layer and fully connecting it to a dense layer, so if the final convolutional layer is smaller, we can greatly reduce the number of weights. I would also want to look into more ways to optimize the sliding window algorithm and potentially speed up that process as well. I think it could also be interesting to create a VAE to see what specific features it learns to recognize Waldo and find out what features cause false positives to occur.

If you are interested in the **code** of this project, click this link:
https://github.com/bensantos/findingwaldo