Sign Language Translation Using Machine Learning and Computer Vision

Gordon MacMaster

Honor's College Thesis

Advisors: James Eddy, David Darais

University of Vermont

https://gmacmaster.github.io/signSpeak/#/

3/31/2020

THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING &
MATHEMATICAL SCIENCES

Contents

Abstract

Over the past several years there have been incredible advancements in machine learning and object detection. They're now being used in everything from security systems to self-driving cars, to automated sorting facilities. One area that has had little benefit from the advancements in breaking down the communication barriers between the Deaf and those who are unable to understand sign language. This paper examines one attempt to start to address those barriers and how that attempt took an unexpected turn that looked deeper at the differences and challenges among various object detection algorithms, how computing power affects how fast and efficiently code can run, and how difficult it can be to work with people.

Acronyms

Acronyms	Definition
API	Application Programming Interface
ASL	American Sign Language
CEMS	College of Engineering and Mathematical Science
CNN	Convolutional Neural Network
COCO	Common Objects in Context
GPU	Graphics Processing Unit
HCOL	Honors College (UVM)
HPC	High Powered Computer
R-CNN	Regional Convolutional Neural Network
SSD	Single Shot MultiBox Detector
UVM	University of Vermont

Acknowledgments

Introduction

As dictated by the UVM Honor's College curriculum, I have spent the last semester and a half designing, building, and implementing a project with the goal of enhancing independent research and learning. The senior thesis serves as a way for students to gain experience and expertise by conducting a multi-month independent research project culminating in a final presentation. Looking back on the past few months of research, I am certain it has done just that. I have worked on creating a way to recognize and interpret American Sign Language (ASL) using machine learning and object detection. What started with the goal of creating an application that could use a webcam or phone cam to detect ASL turned into a project that dove much deeper into the nuances of data collection, machine learning, and troubleshooting than I could have ever imagined. I first came up with this idea over the summer when I saw a video of an individual who had used machine learning and computer vision to match his hand motions with various hand emojis. I thought this was a unique idea and wondered how it could be expanded and improved upon to recognize ASL.

Going into the project I knew that it would be no small undertaking. Looking back at where I was when I first proposed the idea, I could have never imagined the numerous challenges that I would face, the number of new things I learned, or the conclusions that I would end up reaching. I will be the first to admit that taking this project on was out of both my comfort zone and area(s) of expertise. The project spans several fields that were all unfamiliar to me; each with their own unique aspects and nuances. The project required that I dive into both ASL, a language I had no experience with, and machine learning/object detection, a field I had never worked in. This paper will be a discussion of my thoughts going into the project, differences between planned and executed outcomes, reasons for those differences, findings, and an analysis of the degree to which I think this project can be considered a success.

Research Question

Can machine learning and computer vision be used as an American Sign Language translation tool to translate ASL in live time, thus simplifying communication between the Deaf and individuals who do not know ASL?

What is Object Detection?

At a high level, object detection is the process of identifying and recognizing objects in images and videos and providing information about their location within the image. These objects can be anything from faces to buildings to vehicles to, in my case, letters in the ASL alphabet. Object detection algorithms use large amounts of training data to extract features in order to recognize instances of a specific object. Object detection is commonly used in technologies such as self-driving cars, surveillance systems, and image retrieval (Training). One drawback, however, is the ability of the algorithm to make guesses about unknown objects. For example, if an algorithm that has been taught to recognize different type of apples sees an orange it's impossible for it to ever properly identify the orange until it is taught what an orange it. Nonetheless, object detection is an incredibly powerful tool with uses for which are still being identified.

Existing work

The following section contains research done prior to my thesis. Over the past several months since I initially proposed the idea there have been no announcements of new ASL translation technologies using computer vision or machine learning. I am including this as a way to continue to record ongoing research in the field.

In coming up with the idea, I did not immediately think it was particularly unique or special as I figured that with the growth of machine learning/computer vision and the number of people who use ASL to communicate daily that there had to be multiple applications available that were designed to do exactly what I was thinking. In doing some research into the idea, I quickly realized that this is not the case. There are no fully developed companies or applications that use computer vision or other input devices to recognize and translate Sign Language in real time. The two most developed ideas that I have found come from a group of students at MIT and a company called SignAll. The students at MIT designed a pair of gloves that can be worn by an individual and as they sign the 3d coordinates of their hands are transmitted to a computer via Bluetooth where they are run through a neural network to predict the sign that the user is making (UW). Though the project has a lot of potential, only a prototype has ever been made. Shown in figure 1 is one of the students who helped build the device wearing one of the gloves.

Figure 1

The other idea comes from SignAll, a company working to develop a desktop setup to recognize an individual's hand and body movement as well as their facial expressions. The product utilizes four cameras (figure 2) to recognize the user's movements which are passed through a machine learning algorithm to translate their signs in live time. The cameras capture the user in 3D space and recognize hand gestures though a pair of multicolored gloves worn by the user (SignAll).

Figure 2

Both projects follow a similar idea to mine but differ in that they require a much more advanced setup than I would like to see in my application. The product out of MIT requires that the user be wearing a pair of gloves connected to a powerful desktop computer and the product from SignAll requires a full desktop setup complete with multiple cameras and a pair of gloves.

The idea of the project was to create an application that can run from any web browser or from a mobile app or mobile first website so anyone with a laptop or smartphone can use the application whenever and wherever they are. Just creating an application that could recognize the letters of the alphabet proved to be an incredibly difficult task and it has become obvious to me why there haven't been more advancements in using object detection along with sign language.

Original Project Design

Looking back on what I had originally planned for this, it is clear to me now that I expected that I would get a lot more done on the project than I have. This is because each step has taken far longer than I thought it possibly would. The following is a description of what I originally had in mind for the project, what I wanted to accomplish, and about how long I thought each step would take. This helps to really highlight the magnitude of difference between what was supposed to happen and what did happen, both above what I expected and where I fell short.

Going into the project I knew that what I hoped I could accomplish would be no small task. In order to help myself stay organized and on track I broke the project down into several smaller phases. The phases I identified were as follows: A data collection phase, where I would gather and record the information that I needed to carry out the project. A preliminary implementation phase, where I hoped to create an MVP for the project in which I demonstrate that there is viability to the project using the letters of the ASL alphabet. An improvement phase, where I continue to build out on the features and abilities of the project including more complex gestures. And finally, a reflection phase where I analyze the data I have collected, feedback from users, and looks towards further use cases for the application.

Initially I imagined phase one, the data collection phase to take about a month to complete. Over this past summer I built a web app that allows users to record themselves signing various words and letters. By the time I was ready to have users start collecting data I had included 24 of the 26 letters (all letters less 'J' and 'Z' which require hand movement). By taking pictures of individuals signing using my data collection application I would be able to collect the data required to train a machine learning algorithm to recognize a user signing the various letters of the alphabet. I worked with Bradley Brown, an upper level ASL professor here at UVM and had set up a time at the beginning of October to present to five of his ASL 3 and 4 classes.

During the presentations I explained to the classes what my project was, how to use my data collection site, and the timeline for the project. I anticipated that their part of the project would take about 15 minutes and hoped to give them a month (until the end of October) to send me the data I needed. With busy schedules I thought this was a reasonable timeline for the students.

The second phase of the project was to be where I would start using the data sent to me by students to generate a machine learning algorithm to translate ASL letters in real time. During the month while I was waiting for students to send me their data I planned to research machine learning methods for object detection so that once I gathered all the data I would quickly be able to start using it to build out the functionality of the app. The idea was that by compiling all the pre-labeled photos from the users, I would be able to immediately start training a model. This first implementation phase would serve as a proof of concept as well as an experience base for me to start implementing more and more advanced features into the app. With my little knowledge of machine learning and object detection going into the project I expected it would take me about a month and a half (until about Winter break) to create an MVP that could recognize the ASL letters.

ASL relies very heavily on hand and arm movement as well as facial expressions to transmit messages between parties. As a result, an application that can only recognize static hand gestures is very incomplete. The third phase was supposed to be where I would use the knowledge and insights that I gained in building out the MVP to start implementing these more advanced features. I planned to modify the data collection site over the Winter break to allow for more advanced gestures to be recorded. I would then collect more data after returning to school either by myself or again with the help of the ASL students. Then the new data would be used to implement a machine learning algorithm that could recognize moving signs as well as static signs, so the original functionality of the app was not lost. I expected this phase to be significantly more difficult than the first implementation phase as it would be more complicated to implement a machine learning algorithm that can recognize non-static signs and would require me to decide which signs to focus on to maximize the understanding that can be conveyed through my app without having to gather an impossibly large dataset.

My plan for the final phase was to wrap up the functionality of the app as well as analyze my results in terms of functionality, success of the project, and how well it could serve the Deaf

community. Going into the project I thought it would be a very interesting stand-alone computer science project, but that wasn't enough for me. Throughout the project it has been very important to me that it is also seen as a benefit to Deaf Culture. Thus, the final phase would be a combination of finishing up any leftover requirements for the thesis as well as working to ensure this is an appropriate project from the perspective of the Deaf.

Actual Implementation

Upon starting the project, I quickly realized that the timeline of phases that I had originally laid out before the project started would not be possible to stick to. For the most part this is because I greatly underestimated the amount of time each step would take. As each step progressed, I have fallen farther and farther behind my initial timeline. Though this may seem at first like a very bad thing, the path of learning discovery I have embarked on has taught me things that I never expected to learn and exposed me to new technologies I didn't even know existed. With the limited research that I had done at the time I had proposed my thesis I thought that object detection was much more plug and play than I have come to realize. There are many ways to customize what you are doing to tailor it in different ways to adjust for speed vs performance and other metrics. I also underestimated the computational power that is required to run the calculations to create the algorithms. I quickly realized that my personal computer is not powerful enough which lead me to search out other alternatives such as the UVM supercomputer, DeepGreen, and Google's HPC servers. The following sections are an outline of the phase of the project I have completed as of the time of writing this document and plans going forward through the rest of the year.

Data Collection

Out of all the phases of my project that has caused a delay in my progress, the data collection phase was by far the most slow and cumbersome. I initially presented to the five ASL classes about helping with my project in early October. During my presentation I explained what my project consists of and where I needed help gathering data. At the end of the presentations I had gathered 50 volunteers who were willing to help with the project.

The problems with data collection came in two parts: participation and data quality. In terms of participation, by the end of November, a month after I had initially hoped to have collected all my data, I only had received it from two individuals despite several follow up

emails and requests for people to send it to me. Eventually, Professor Brown offered extra credit on one of their assignments that was due during the final's week of the fall semester. This was incredibly generous of him and I ended up receiving data from about 45 people during that last week of the semester. This just goes to show how unlikely people are to do something if they have no incentive or personal reason to do so.

Data Labelling

The next issue was the quality of data I was sent. When an individual uses my data collection web app, they are given random letters to sign. This is designed so that if they do not get through all the signs that are available there is a good enough mix of data for each of the signs to create a computer vision algorithm to recognize each character. As the user progresses through each sign their webcam captures pictures which are then run through a hand recognition software to determine the location of the hand in the picture. When the user is finished the location of the hands in each photo is displayed in a box. The user is then supposed to move and resize the box, so the hand is properly centered in the box. The students who sent me data did not do this. Figure 3 shows what a properly sized and aligned box looks like.

Figure 3
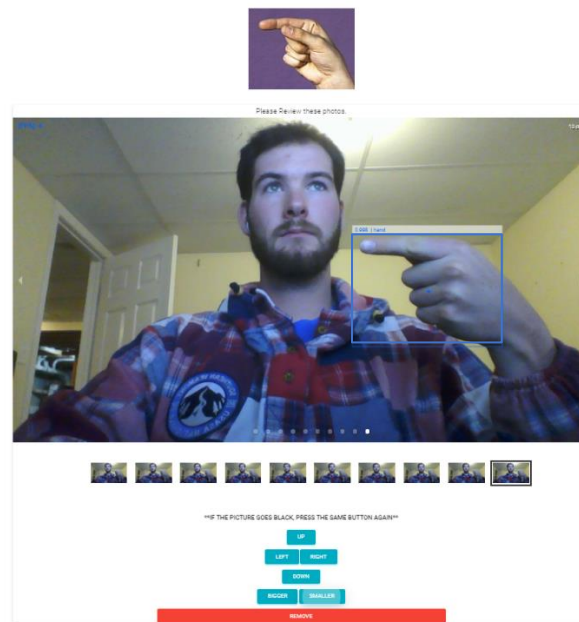
Notice the box is properly aligned around the hand, it's the proper size, and contains all of the unique gestures that makes up the letter. The following picture is an example of what a picture I

was sent looked like. I used an example with myself rather than real data as I told the students their data would not be published anywhere.
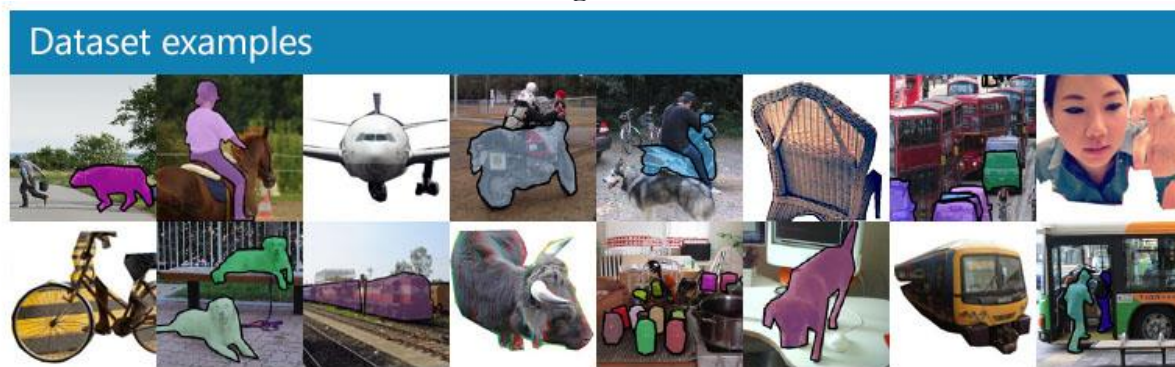
Figure 4
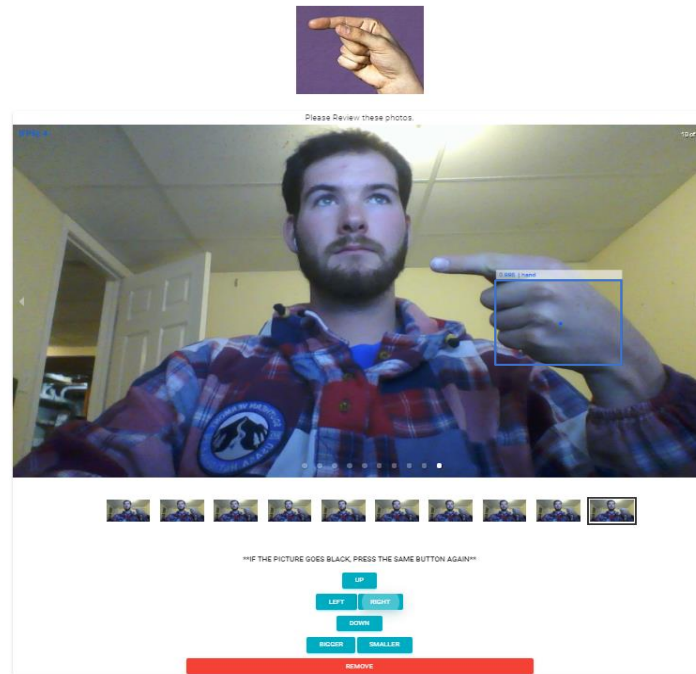
In this picture the box surrounding the hand is far too small and it is not properly centered on the hand to completely enclose all the unique features of the letter. I was sent about 3200 images from the students, all of which I had to manually go through and adjust the bounding boxes so my machine learning program could properly identify the letters. This started out taking me about 1 minute per image. I quickly realized how much total time that would take, and I added some event listeners to the app so that I could move the box with my keyboard. This cut down the time per picture to about 20 seconds. In total I spent 25 hours over the course of winter break and the first two weeks after winter break properly labeling the images. I take full responsibility for this slow down. I did not emphasize the importance of students properly moving the boxes in their images. This shows me how, moving forward, any time I am interacting with a third party I need to very clearly define the requirements and ensure that individuals are properly executing assigned tasks.

Training the Model

        Once I completed cleaning all the data and ensuring the labels were correct, I was finally able to begin training my object detection model. While I was waiting for the ASL classes to send me their data I had spent some time watching tutorials on how to create custom object detection models using Google's TensorFlow Object Detection API. After completing my research, I found that I should use an approach called transfer learning to create my algorithm. Transfer learning is a process where the weights of an existing pretrained model that was created to perform a similar task are retrained on new data in order to perform a new, but similar task. This leads to a more efficient train time and a more accurate model in the end. I found a very easy to follow [tutorial](#) that broke down, step by step, what it takes to create a custom object detection using transfer learning. The tutorial used a model that was trained on the Microsoft COCO dataset which is a large-scale object detection dataset containing 330,000 images consisting of 80 object categories (Common Objects in Context). The model provides an out of the box method to identify everyday objects from animals to people to vehicles. Some examples of objects it can recognize are show in figure 5 below.

Figure 5

(Common Objects in Context)

As the model is very commonly used to identify objects in live video streams with high average accuracy, I figured it would be a good starting point for creating an algorithm that can recognize ASL letters in live time and would allow me to build an application around that ability.

Training on my Data
First Try

        I was first able to start training my model during the first week of February. Following the steps of the tutorial I thought everything was going great through most of them. The first

tutorial I followed is optimized for training on machines with a GPU, which allows for significantly faster training times than on a computer without one. I have a Dell XPS 15 with a Nvidia GeForce GTX 1050 Ti GPU so I figured it would be good enough to train my models. The tutorial trains the object detection algorithm to identify different playing card values. To train the model in the tutorial it takes about three hours, so I figured it would be about the same to train mine. I was very wrong.

I first started training the model at about 6pm on Wednesday February 5th. When training object detection there is a variable called location loss that is output with every iteration of the training. It usually starts somewhere around 10 and training is complete when it reaches about .05. The loss usually falls from 10 to 2 in about the first 1000 iterations and then slows down significantly after that. The first indication of trouble that I noticed was the amount of time it was taking per step (iteration) for my model to train. In the tutorial it was taking about .2-.25 seconds/step to train. My model was taking about 2.3 seconds/step to train, on the order of 10 times slower. In doing some research about my computer I came to find out that my GPU is a low-quality graphics card designed for mobile applications and in my computer treated as a backup number processor, rather than as a true GPU. I figured there was nothing I could do about it and decided to let the algorithm run, even though it would take, to my estimates, about 30 hours. I left my computer running for a day and came back the next night to what I realized would be another setback in the amount of time it took for my model to train. In the tutorial, it took about 50,000 steps to reach the .05 loss goal. After a day I had barely broken 1. I still had a lot of training left after 24 hours.

There was nothing to do other than let the algorithm continue training. I didn't know enough about machine learning to try to figure out how to make it run faster. Letting the algorithm train through the weekend, I woke up on the following Tuesday to the model saying it had finished training. The settings in the algorithm automatically stops training after 200,000 steps. Going into training I thought this would be plenty as the tutorial stopped at 50,000. Luckily, the model had reached a loss of about .07 when it stopped training so I thought that would be good enough. In total the model trained for 200,000 steps at about 2.3 seconds per step. That is 127.8 hours or 5.3 days in total. The final output of the training is shown in the figure 6 below.

Figure 6

```
Administrator: Anaconda Prompt (Anaconda3) - conda  install -c anaconda protobuf
I0210 04:20:05.195069 24568 learning.py:507] global step 199990: loss = 0.1186 (2.301 sec/step)
INFO:tensorflow:global step 199991: loss = 0.0795 (2.305 sec/step)
I0210 04:20:07.500901 24568 learning.py:507] global step 199991: loss = 0.0795 (2.305 sec/step)
INFO:tensorflow:global step 199992: loss = 0.0603 (2.307 sec/step)
I0210 04:20:09.808727 24568 learning.py:507] global step 199992: loss = 0.0603 (2.307 sec/step)
INFO:tensorflow:global step 199993: loss = 0.0935 (2.292 sec/step)
I0210 04:20:12.101595 24568 learning.py:507] global step 199993: loss = 0.0935 (2.292 sec/step)
INFO:tensorflow:global step 199994: loss = 0.1124 (2.307 sec/step)
I0210 04:20:14.410417 24568 learning.py:507] global step 199994: loss = 0.1124 (2.307 sec/step)
INFO:tensorflow:global step 199995: loss = 0.2119 (2.301 sec/step)
I0210 04:20:16.712260 24568 learning.py:507] global step 199995: loss = 0.2119 (2.301 sec/step)
INFO:tensorflow:global step 199996: loss = 0.0633 (2.315 sec/step)
I0210 04:20:19.028065 24568 learning.py:507] global step 199996: loss = 0.0633 (2.315 sec/step)
INFO:tensorflow:global step 199997: loss = 0.0770 (2.305 sec/step)
I0210 04:20:21.333897 24568 learning.py:507] global step 199997: loss = 0.0770 (2.305 sec/step)
INFO:tensorflow:global step 199998: loss = 0.0554 (2.306 sec/step)
I0210 04:20:23.641724 24568 learning.py:507] global step 199998: loss = 0.0554 (2.306 sec/step)
INFO:tensorflow:global step 199999: loss = 0.2397 (2.290 sec/step)
I0210 04:20:25.933593 24568 learning.py:507] global step 199999: loss = 0.2397 (2.290 sec/step)
INFO:tensorflow:global step 200000: loss = 0.0793 (2.296 sec/step)
I0210 04:20:28.231447 24568 learning.py:507] global step 200000: loss = 0.0793 (2.296 sec/step)
INFO:tensorflow:Stopping Training.
I0210 04:20:28.232459 24568 learning.py:777] Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
```

I knew that was far too long if the algorithm didn't work or if I needed to make changes. The tutorial comes with code to test your model once it finishes, both with your webcam and single images. I first ran a single image test. Here is the result:

Figure 7
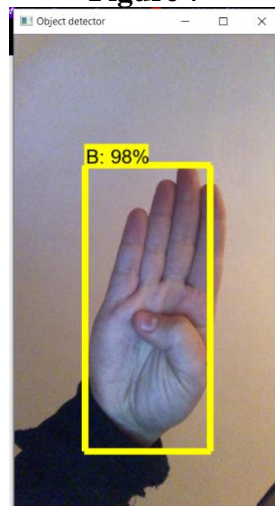
Despite the incredibly long training time, it worked. It correctly predicted the sign and identified where in the image the sign was. I tried a few more images and started to notice my first issue. The first image I tested included the hand in the middle of the frame. The other images I tested had hands positioned to one side or the other. In these images the signs were being correctly

identified, but the boxes were being placed in the wrong places. It appeared that the boxes were being mirrored to the position they were supposed to be. This issue is shown below, in figure 8.

Figure 8
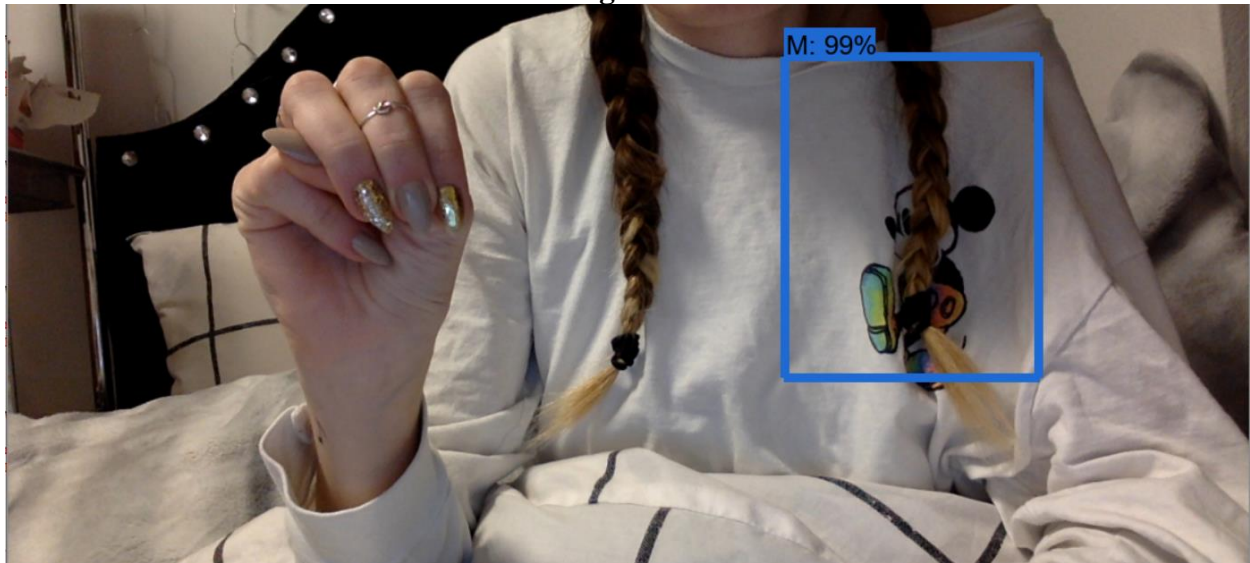
Assuming this was a code problem that I could easily fix, I then tried the live video stream from my webcam. This is where I ran into my next problem. The algorithm was incredibly slow. The video looked like it was running at about 1 frame per second. When it predicted the sign it was, overall, correct most of the time. Oddly enough, the mirroring problem didn't happen in the live stream. One problem fixed, another one created. When I checked in with Professor Eddy in February, this is where I was. I had just finished training my first model and was running into these issues.

Fixing the Issues

Having seen other algorithms in action I knew they were supposed to be able to handle upwards of 50 frames per second. The current model was not fast enough. I set out trying to figure out how to make my algorithm faster and more efficient. A journey I am still on today, even as I write this. The first thing that occurred to me was that my data was not good enough, more specifically the way I had labeled the data. The web app that I built to collect the data automatically generated a CSV file with all of the information about what was in each picture including the class (letter), file name, and hand location. Here is one line from a file generated by a user.

score	sign	signName	xLocation	yLocation	width	height	fileName
0.99339	Y	Letter Y	711.9202	228.0822	137.7215	153.63	124898-Y-4

In the tutorial, users are supposed to label each of their images with a special labelling software. This software then generates an XML file with the same information which is then converted into a CSV. I thought that because I skipped these steps, that might be the cause of my issues. So I spent another 5 hours labelling a sample of 500 images the same way that the tutorial directs you to. The process involved drawing a box around the hand in each image and labelling the sign in each image.

Figure 9
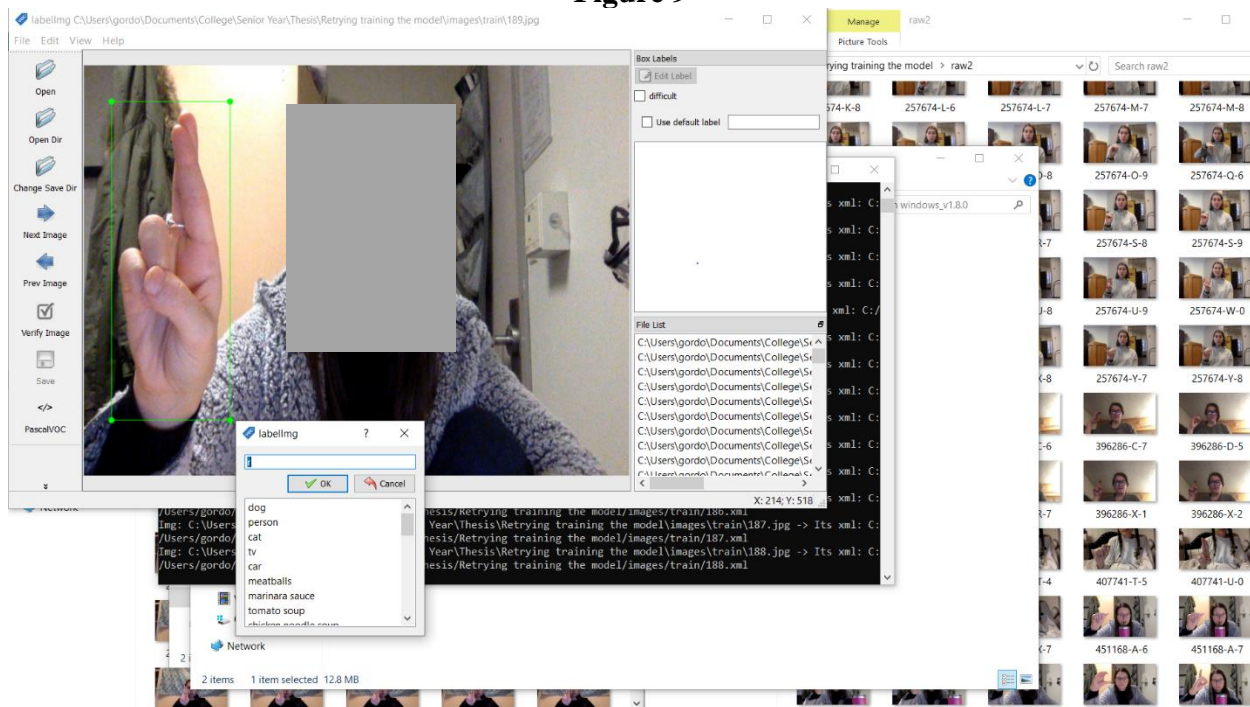
I started training on my computer again and quickly realized that it was not going any faster. Not wanted to wait another 130 hours, I started looking into ways I could use another computer to train on. One of my friends suggested I used Google Colaboratory which allows users to run Jupyter Notebooks on Google's GPU accelerated servers for free. I found a tutorial that provided a Jupyter notebook I could copy and allowed me to run basically the same code I followed in the first example in Colab. Around the same time I learned about Colab, I also found out about UVM's supercomputer, DeepGreen. I now had access to two computers that were wildly more powerful than my own to help train my model on.

Training on Colab
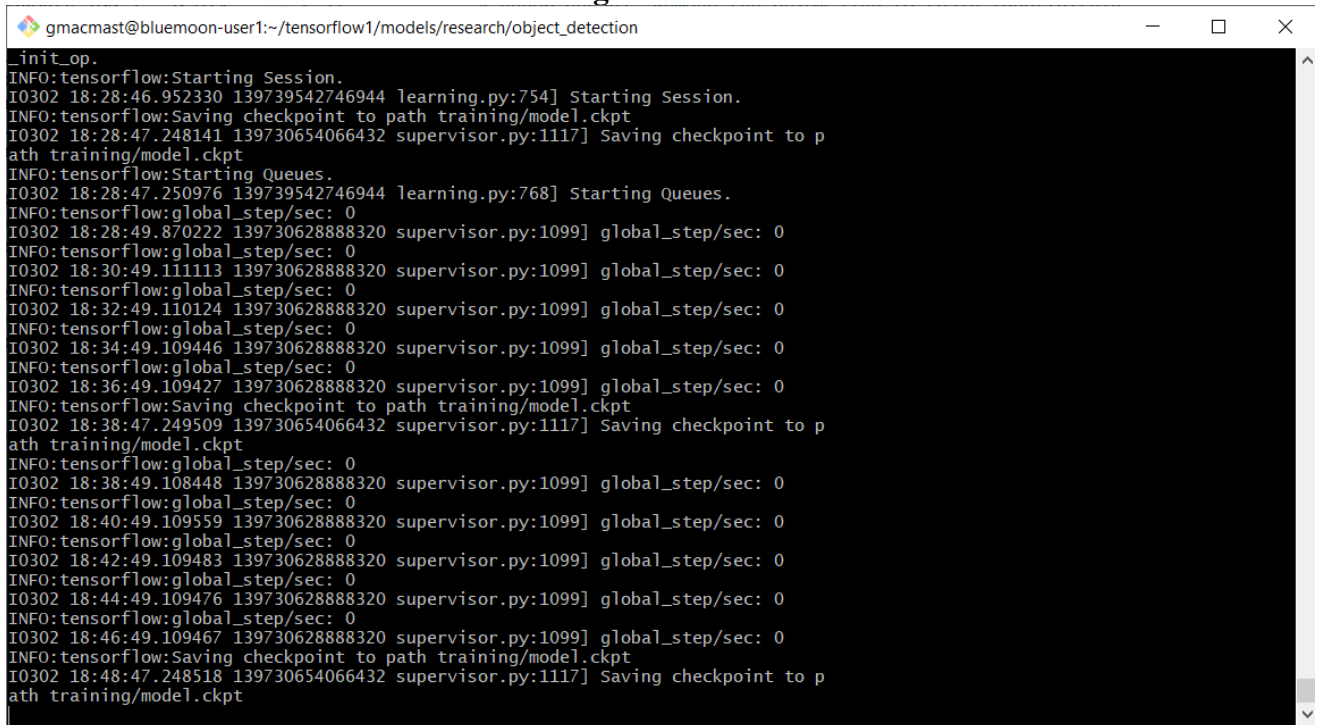
While waiting to be granted access to DeepGreen I started working with Colab. It was quite easy to upload my own data; it uses your Google Drive to save all your data. Training on Colab was significantly faster than on my personal computers. I will go into more specific details about the numbers later, but it was about .23 seconds/set or about 10 times faster than my computer. Training still took a couple of days because there are GPU time limits on your account for the free tier. But in total, it only took about 12 hours of GPU time to train my second model.

Testing the new model, I immediately noticed two keys differences, one good and one bad. Starting with the bad, the new model did not seem nearly as accurate as the first one I created. It failed to pick up some signs that the original model could easily identify and more often it misidentified the letters being shown. On the other hand, the model was significantly faster. It could run a live video stream seamlessly where the original one was incredibly slow. A big improvement in my mind. Now I just had to figure out what was making them so different.

Training on Deep Green

Being unfamiliar with most of the ideas behind object detection, at this point I had two models trained on different machines, both with unique advantages and disadvantages. I thought that maybe training on a third, different machine, would give me the result I wanted. While I had been training on Colab I was granted access to DeepGreen. I hoped that getting my model training on DeepGreen would be as simple as it was to run on my personal machine and on Colab. This was far from the case. The original tutorial I used was written for Windows machines, DeepGreen has a Linux OS, and to makes things more difficult, it is a RedHat distribution. I spent a lot of time trying to convert the original tutorial to work on Linux, but kept running into various OS errors, permission denied errors, and the inability to install some required resources because of the OS. Finally, after hours of weeding through Stack Overflow solutions to the various issues I was able to get to the step to start training. It appeared that everything was all set. The program went through the initialization steps and began training. At this point I started to notice that it didn't seem to be working. Where on my computer and on Colab it would tell me how many steps/second it was preforming and the loss, on DeepGreen they just showed up as 0. The output of my first attempt on DeepGreen is shown in figure 10.

Figure 10

I let the program run for 10 minutes and ultimately gave up as it was not working. I then tried to upload the Jupyter Notebook I used on Colab to DeepGreen as there is a UI that allows you to use them. Once again, I ran into permission denied errors when trying to install some Python packages. Another CS student then showed me a tutorial that included how to create virtual environments on DeepGreen. I thought this would be my solution as long as I could run the Jupyter Notebook within the virtual environment, a task that according to the internet should be pretty easy. Creating the virtual environment was no problem, but every time I opened the notebook in it, the kernel crashed. Unable to fix the problem using any answers I could find online, I thought I was at a loss. As a last attempt I decided, that because I had a virtual environment and Jupyter Notebooks are nothing but Python commands I might be able to just run all the commands from the command line (shown below).

Figure 11

Working my way through the notebook I was able to run every command I needed to get the model to start training and noticed a key part of the algorithms that I had missed up until this point. More on that later. By running the commands from the command line, I was finally, after hours of struggling, able to train my model on DeepGreen. It was not as effective as I hoped, however. On DeepGreen it was taking about 350-425 seconds to run 100 steps, or about 4 seconds per step, which is about 15 times slower than on Colab. I was not sure why it was so much slower on a supercomputer than on a free Google service, but at the time I decided not to let the training on DeepGreen and focus rather on the discovery I made while reading through the code. I hoped to return and try to figure out why is was so slow once I completed more of my project.

The Key Factor I Missed

While I was copying over the commands to DeepGreen I noticed a big difference in the commands to start running the training. I noticed that the code I originally ran on my computer and the code I ran on Colab had different config files. I wasn't sure what that meant but remembered the original tutorial mentioning something about being able to choose which one you wanted. At the time I had so little experience with object detection I decided to just ignore

that possibility and go with what they used. The differences between the commands is shown below.

```
python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/faster_rcnn_inception_v2_coco.config
```
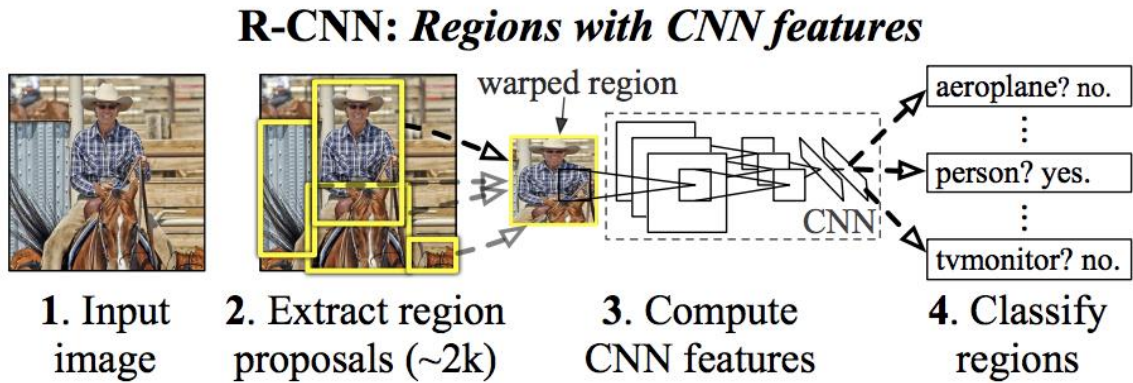
```
python3 object_detection/model_main.py  --pipeline_config_path=/gdrive/My\
Drive/object_detection/models/research/object_detection/samples/configs/ssd_mobilenet_v2_coc
o.config  --model_dir=training/
```

Given that I had such a big difference between my models in terms of speed and accuracy and this was the only difference I had noticed up until this point, I decided to dig a little deeper.

The Algorithms

When I started looking into the differences between the two config files that I unknowingly used I found that over the past several years there have been many iterations, improvements, and optimizations within the Google TensorFlow Object Detection Suite. People have created algorithms that are optimized for different things, often in a tradeoff between speed and accuracy, exactly what I had noticed, only I didn't know why it was happening. I figured out I had used two different models: Faster_RCNN_Inception_V2 and SSD_Mobilenet_V2. Faster_RCNN_Inception is a type of algorithm that uses a convolutional neural network to identify objects within smaller "regions" of an image or frames of a video, hence the name regional convolutional neural network. Traditional neural networks are not good for object detection because a standard convolutional network uses a fixed output layer, which would make it impossible to indentify an unknown number of objects of interest. Each image is broken down into about 2000 regions which are then warped into a square and run through a convolutional neural network that produces a 4096-dimensional vector that contains the information about features within the image. The resulting vector is then fed into a support vector to decode the information about classes in the image and their locations (Gandhi). Figure 12, below, contains a simplified version of this process. The inception part of the name refers to the idea that the model computes several different transformations at the same time in parallel before combining the results into a single output. This allows for a more accurate model, but at a significant increase in computational cost (Xu).

Figure 12

R-CNN: *Regions with CNN features*

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

(Gandhi)

Faster_RCNN is an optimization of the original RCNN algorithm that cuts down on classification time by using a separate network to identify regions and then using the CNN to make predictions about objects in the image. Faster_RCNN created a speed upgrade that finally allowed for R-CNN to be used for live-time object detection. The differences in detection time for the various versions are shown here:

Figure 13

R-CNN Test-Time Speed

- R-CNN: 49
- SPP-Net: 4.3
- Fast R-CNN: 2.3
- Faster R-CNN: 0.2
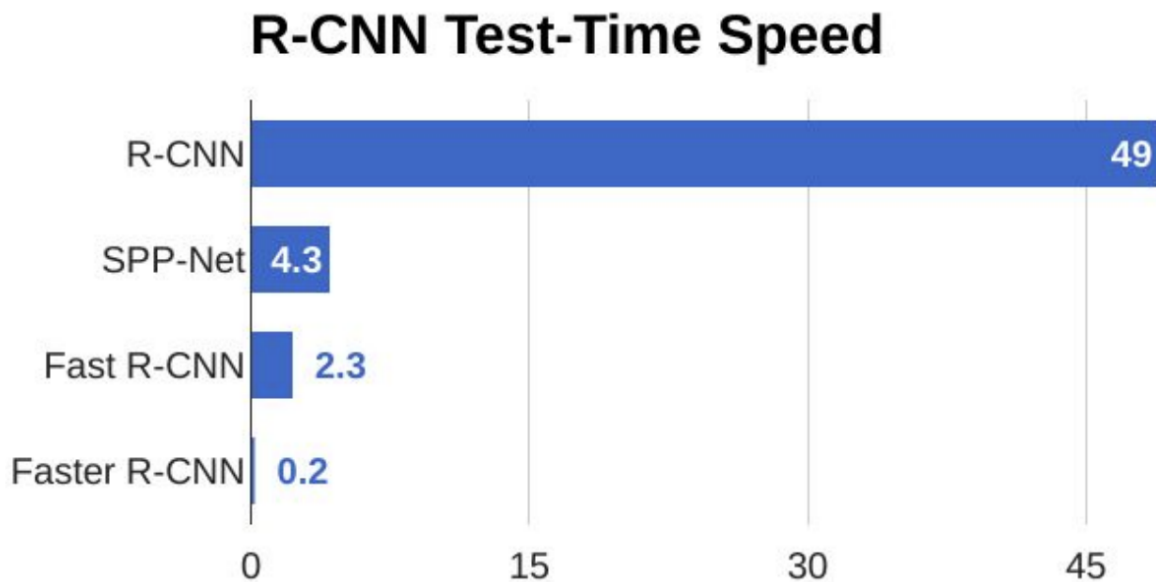
(Gandhi)

While Faster_RCNN is optimized for speed over the original RCNN, its main optimization is towards accuracy rather than speed. Between the two algorithms that I used, SSD_Mobilenet is far more optimized towards speed, at the logical expense of accuracy. SSD referrers to the method of detection used in the algorithm, single shot multibox detector. Like RCNN, SSD also uses a convolutional neural network to identify objects in an image. At its base is VGG-16, a top convolutional neural network built by the University of Oxford. VGG-16 offers high quality classification and is also very good in transfer learning. The name SSD comes from three parts: single shot, multibox, and detector. Single shot refers to the fact that object detection occurs in one forward pass of the network. Multibox is the method of regressively identifying the bounding boxes of the objects. Detector means the network classifies the images as well as identifying their presence. SSD is very popular when speed is a required feature. It has been able to consistently reach 74% accuracy at 59 frames per second on large datasets such as COCO. (Forson). Figure 14 shows the architecture of the SSD algorithm with a 300x300 pixel input.

Figure 14
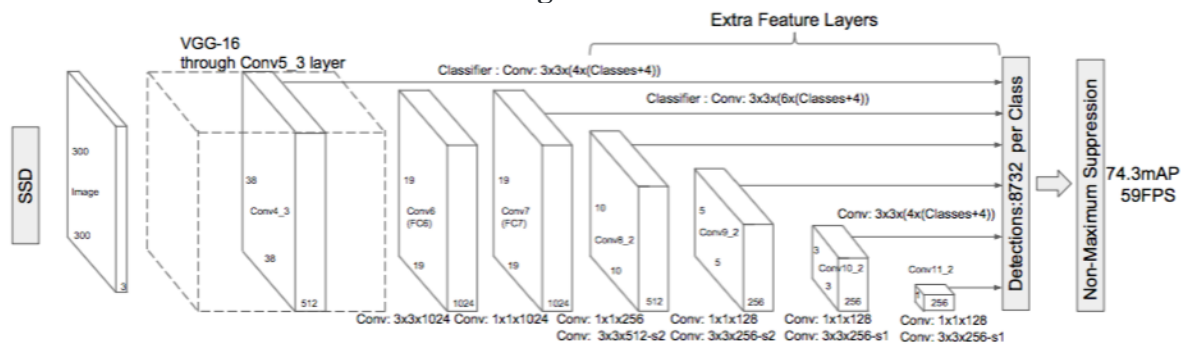
(Forson)

On top of the SSD optimizations, MobileNet adds further optimization when computational power is lacking. It was proposed by Google for mobile and embedded applications where computational power is limited. Mobilenet uses depth wise convolutions which significantly reduces the number of floating-point multiplication operations as compared to normal convolutional networks. This makes for a lightweight neural network. The difference is highlighted here, in figure 15.

Figure 15

(a) Standard Convolution Filters

(b) Depthwise Convolutional Filters

(c) 1 × 1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution
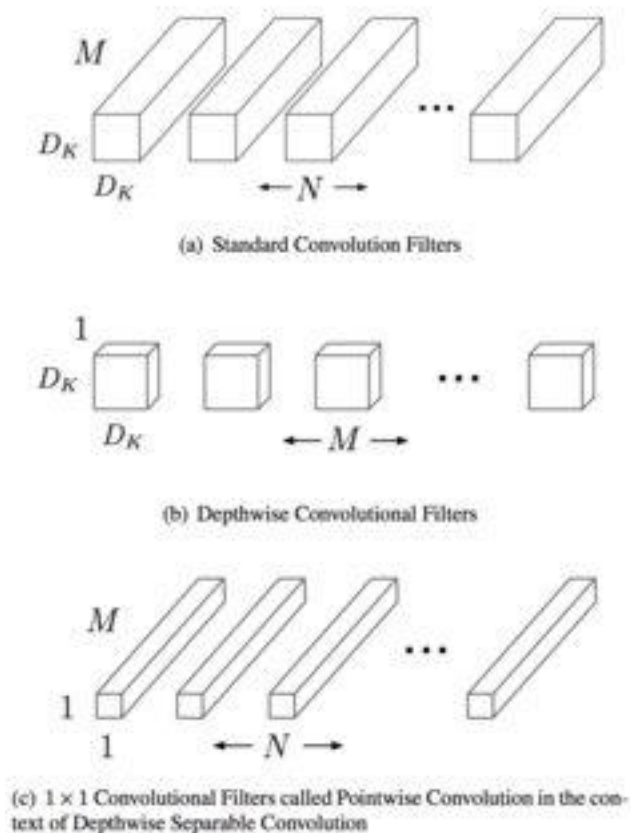
(Forson)

The use of depth wise convolutions greatly reduces the number of calculations required but does so at the loss of accuracy in the network.

There is no perfect answer to the accuracy vs speed problem in object detection. Over the past few years there have been huge advancements in both, and I expect in the coming years to see an even greater increase in the ability of these algorithms to accurately detect objects in ever decreasing amounts of time.

Results

From a pure initial goal perspective, this project was not a complete success, or even much of a success at all. As much as it pains me to admit, now, at the end of March, I am still not even to the point I wanted to be before winter break. As someone who is usually incredibly

driven by results it has been difficult to stay motivated at times because it became clear I wasn't going to get nearly as far on the project as I initially wanted to. Over the past couple of days, I have been spending a lot of time talking about theses with my roommates (they're also in HCOL) and its dawned on me just how much I have learned throughout the course of attempting to complete this project.

What I Learned

Coming into this project I had no experience with either ASL or machine learning/object detection, but if I had been asked, I would have said that I was most nervous to work with ASL. I knew that in order to create a successful object detection algorithm, I had to start with a large amount of high-quality data. I also knew that in order to avoid bias, my data should differ a decent amount in order to be able to predict signs from people other than just me. In order to gather this data, I wanted to work with upper level ASL classes in the fall semester. Before I approached the professors to ask for their help, I made sure to do research about the Deaf and Deaf Culture. Deaf culture is the combination of "beliefs, attitudes, history, norms, values, literary traditions, and art shared by Deaf people." (The Outreach Center) The Deaf are very proud of their culture. They do not view themselves as having a disability and don't want to be "fixed" (Clason). I wanted to make sure it was clear that I wasn't trying to intrude on this culture by building out this project so I made sure to meet with the professors and discuss the app, the difficulty in building it, and gather feedback from them before writing any code. They were very much onboard with the project and ended up being instrumental in my data collection. On top of learning about Deaf Culture, I have also started to learn a little bit of ASL myself. Though I would consider my skills very lacking or almost non-existent, I have been trying to have conversations with my roommate who is in ASL 2 which has helped me gain an idea of where this project would have to go next in terms of getting what is required for communication.

One of the parts of the project that taught me the most was the data collection phase. After I presented to the ASL classes and in taking questions and comments from the students I initially thought the data collection would take less than a month. Everyone who volunteered seemed eager to help and see where the project would go. As weeks started to drag by and I wasn't receiving data from people, even with a couple reminder emails I really started to realize how hard it is to motivate people who don't have stake in a project. Even though the students thought it would be a cool project and wanted to use it once it was done, they didn't want to take

the time to actually help and send data. That was until they were given the possibility of extra credit. Even when that happened, they didn't take the time to properly center the boxes around the hands, leaving me with hours of work to do on my end. I've realized that when getting people to help you with something or work for you its critical to figure out how to properly incentivize them to do it, otherwise they won't do it or won't do it with as much energy as they should. This will be very important to remember in the future. I plan on starting my own company at some point and it will be important when working on future projects, hiring people, or leading people in general to make sure I can figure out what incentivizes them to do their best work, be that money, a challenge, possibility of promotion, or anything else. This has also made me think back to all the requests for feedback I have personally ignored from companies and even UVM. These could be incredibly important to them, but since I didn't have a reason, I never saw the point in responding. Perhaps I will start doing them in the future.

The point of your senior thesis is, at least in part, to become an expert in the area you choose to investigate. I am not sure how much I can consider that true for myself, but I think I have learned a lot in another area: self-teaching. Throughout the course of this project I have really come to understand just how much you can learn on your own. I have never taken a formal class in ASL or machine learning, yet I have still been able to create an application that can recognize basic signs. The internet is an incredibly powerful tool. With it and some personal motivation I think that people can learn to do anything. Going forward I will know how possible it is to do something even if you don't know how. I have also found some new technologies that I will absolutely use in the future, not just related to ASL or machine learning. Google Colab and Jupyter notebooks in general are incredibly powerful tools that have the potential to assist me in a lot of things going forward.

Finally, I learned that not everything in life follows the timeline you lay out for it. When I made my initial plan for my thesis, I thought I was giving myself plenty, if not even a generous amount of time to get each stage done. As timelines for each phase slipped, I found myself farther and farther behind where I wanted to be. It will be important to remember that, especially once I get to the real world and am trying to give estimates to bosses and other stakeholders for when things are going to get done. I am lucky that I have been able to learn so much other than just the result in this project that I am not worried that it was at all a waste of time or not worth it.

The Numbers

As much as it seems to me that most of the value of this thesis has come through discoveries about machine learning and object detection, new ways to run code (like Colab) and how difficult it can be to work with people who aren't fully invested in a project, I was able to collect a decent amount of numerical data to support what I was reading about the differences in object detection algorithms and my understanding of computing power. In the sections below I highlight speed differences between the algorithms running on the same machine and training time differences on different machines. It speaks a lot to how algorithms and computing power can affect the time it takes to run code.

Faster_RCNN_Inception_V2

The first algorithm I will talk about is the Faster_RCNN, the first model I unknowingly used. As a reminder, RCNN is designed to have higher accuracy at the cost of speed of object identification. The first time I was able to start a model training was on my personal computer using Faster_RCNN, it ran at about 2.3 seconds per step, which was 10 times slower than in the tutorial, but I figured that was due to hardware differences between my machine and the one they used. When I was able to get RCNN running on colab (I first unknowingly used MobileNet), it ran much faster than both my computer and the tutorial, at about .1 seconds per step. This is 23 times faster than my personal computer and more than twice as fast as the tutorial.

| Machine	Personal Computer	Colab
Sec/Step	2.302	0.101
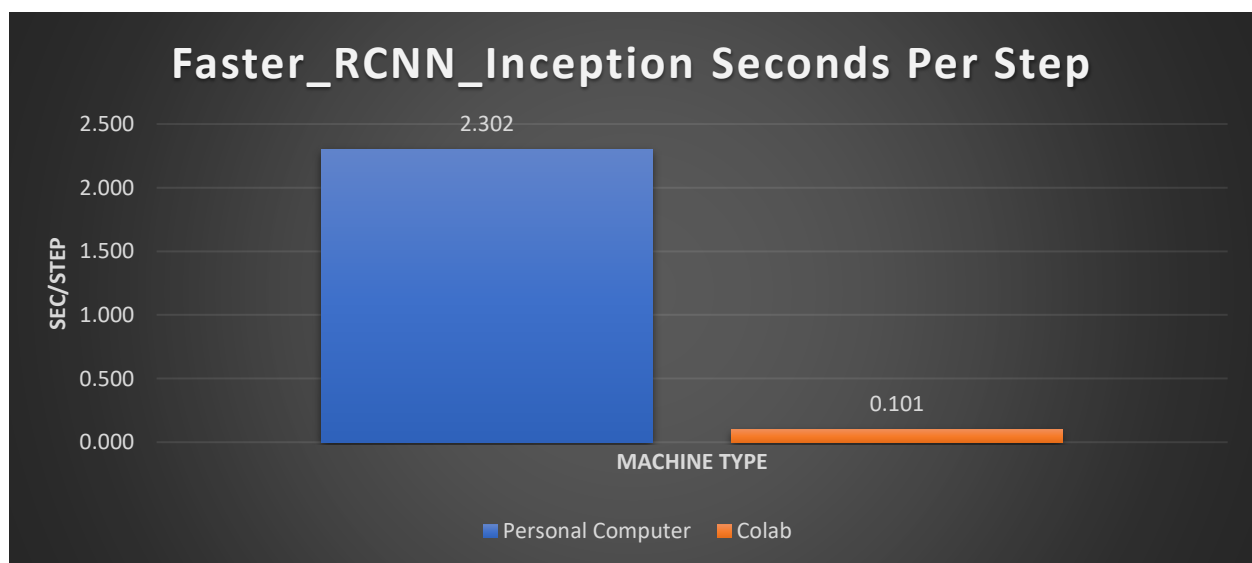
There was also a huge difference in total train time. Where on my computer it took 200,000 steps to get to the .05 loss I was looking for, it only took 50,000 steps on colab. This totaled out to 128 hours training on my computer and only 1.4 hours on Colab. This means that total training on Colab was 92 times faster than it was on my personal computer.

SSD_Mobilenet_V2

   Unlike on my personal computer, the first model I was able to run on Colab was Mobilenet which is designed to be faster but less accurate. Furthermore, Mobilenet is the only model I have been able to run on DeepGreen so far. This has given me an interesting view of the differences in computing power. Starting with Colab, the initial trial trained at .235 seconds per step, which is just about the same as the tutorial I originally watched, leading me to think that I had finally figured out how to replicate what I was seeing. This wasn't entirely correct, but I had gotten a lot closer. From there I then ran the code I used on Colab on DeepGreen and Bluemoon, UVM's HPCs. When I entered the train command, I was incredibly surprised to see that it was only training at 4.1 and 3.5 seconds per step respectively. This is much slower than I was expecting giving Colab is a free service and DeepGreen is a supercomputer. I ran the exact same code and used the same data on both Colab and DeepGreen so I am guessing that the difference must be because there are some settings that are not optimized for machine learning on DeepGreen or the hardware itself that Colab runs on is more optimized towards machine learning and object detection. Once I figured out that I had been using two different models, I also went back and rand Mobilenet on my computer as well on Colab without a GPU to see the difference in times.

| Machine	Personal Computer	Bluemoon	Deepgreen	Colab No GPU	Colab
Sec/Step	13.500	3.546	4.149	15.152	0.235

SSD_Mobilenet Seconds Per Step
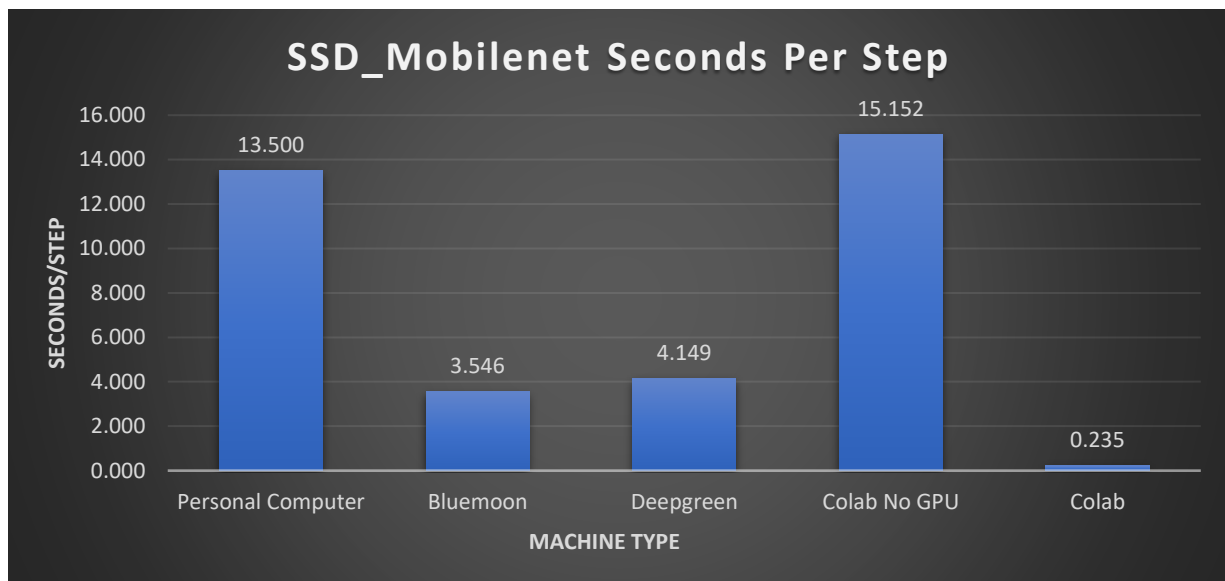
On my computer it took 13.5 seconds per step and on Colab without a GPU even slower at 15.1 seconds per step. This means that training on Colab with a GPU was 17 times faster than on DeepGreen, 59 times faster than my computer, and 65.5 times fast than on Colab without a GPU. The last figure really goes to show how much of a difference a hardware upgrade can make. As a final note, as it took so long on my personal computer, DeepGreen, and Colab with no GPU I decided not to allow them to train completely as they would take too long.

Model Differences

This final comparison I will do is between the two models. Looking at the training times for the two models, on both my computer and on Colab the RCNN model was significantly faster in time per step. On my personal computer RCNN was 5.9 times faster to train than MobileNet and on Colab RCNN it was 2.3 times faster than MobileNet. Initially I found this very interesting because RCNN is optimized for accuracy at the cost of speed, while SSD_MobileNet looks for speed over accuracy. Thinking deeper, I realized that their optimization is for identification and has nothing to do with training times. The two models use different training algorithms, so it makes sense that they have different training times. Realizing that, what I now find most interesting is the change in differences between how much faster RCNN is on my computer vs on Colab. On Colab RCNN trains twice as fast as MobileNet while on my personal computer it trains almost 6 times faster. This suggests that as the hardware becomes more optimized and powerful the difference in training times decreases. It would be very interesting to see the

difference on a machine even more powerful than the version of Colab I was using, such as Colab Pro.

Machine	Personal Computer	Colab
Faster_RCNN_Inception	2.302	0.101
SSD_Mobilenet	13.500	0.235
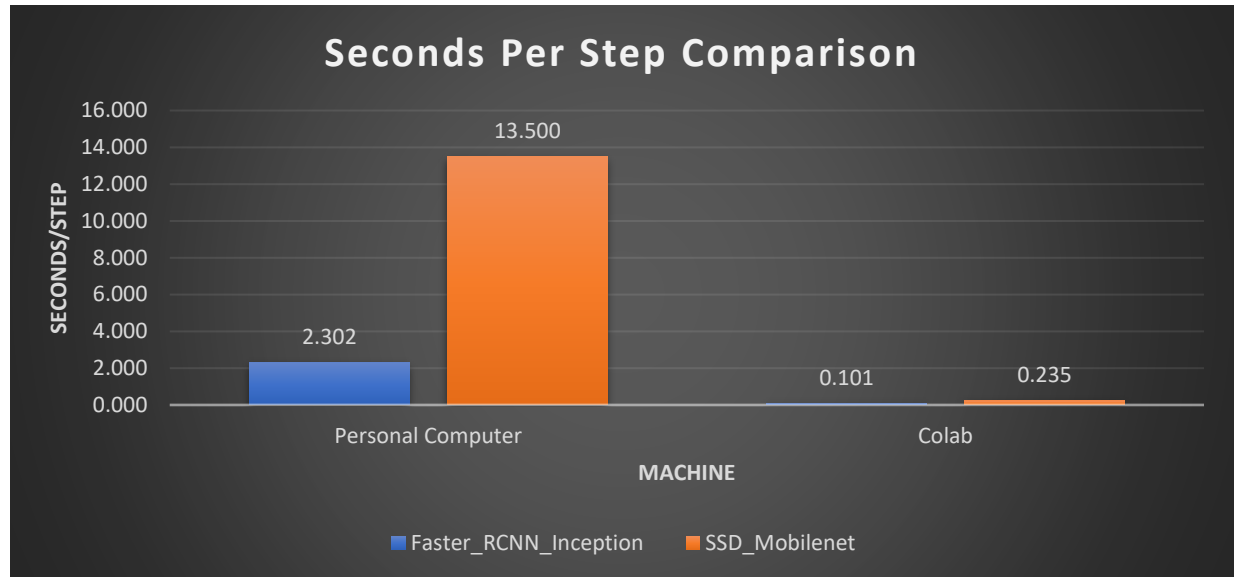
Until now, I have not had the chance to run the RCNN algorithm on DeepGreen. I think it would be very interesting to see how much faster RCNN trains than Mobilenet on another machine, but with everything going on I haven't had time to sit down and try it. If I get some time, I will include those numbers in any future revisions I do.

What I Would Do Differently

At this point in the project I think I finally have enough hindsight to start thinking about the changes that I should have made and how I should have done things differently. Overall this has been an incredible project, but there have been several pain points that I think could have been smoothed out with the information that I have now. First off is the data collection phase. With how hard I now know it is to get people to participate I should have come prepared with the incentive from the beginning. Something like an Amazon gift card, extra credit offered from the beginning, or even a pizza party can go a long way in incentivizing people to help. The next

issue was all of the data labelling I had to do. I needed to make it even more clear how important that part of the process was and made sure people were doing it when they sent me the data. The final change that I would have made was I would have started with MobileNet from the beginning. It has given me the speed I want for my application, but not yet the accuracy. I think with playing around with all of the settings for training such as number of random horizontal and vertical flips, image brightness, and image scale. I think by tweaking these parameters I could have built a model that is both fast and accurate to meet my needs.

Where I Am Now

I have now been working on this thesis for the past 6 months. I have spent countless hours working to build a way to collect data, cleaning the data once I received it, learning about machine learning and object detection, and playing around with various potential ways to improve my models. The one thing I have not spent enough time doing is coding the final project. Over past months I have spent so much time trying to get any type of model that works and writing this paper that I almost forgot that was part of the project. I now have two weeks to build out a project using the models that I have. I would say the best model I have right now is about 60% as accurate as I would like it. Luckily, I finally figured out how to make it fast enough to detect in live time. This will now allow me to play around with different options for the final project. I will be able to spend the next couple of weeks building out a cool project to present. That being said, I am not done trying to make it better. I am still training the model to make it better. I currently have two Colab instances training Mobilenet models as well as on my personal computer, one of which is at training step 800,000 (Figure 16).

Figure 16

Each one has slightly different settings applied in hope of finding one that increases the accuracy to closer to where I would like it. I am not sure how far I will get with the actual coding part of this project, but I am more than happy with everything I have learned, all the discoveries I made, and how much I have produced in this project.

Conclusion

Coming into this project I had a lot of expectations about what I would learn and about where I would be able to get by the time the end of March came around. Now that we're here its very interesting to see how I overestimated parts of the project and underestimated others. In this light I think I greatly overestimated how far I would be able to get. This is both because the steps of the project took longer than I expected and the fact that I have been much busier this year than I thought I would. Looking back, I also think I greatly underestimated how much I would learn about object detection. ASL is incredibly nuanced and complicated which makes creating an object detection algorithm very difficult, but I am more than confident that if I wanted to create an algorithm to detect something like different types of animals or types of food I could do so very quickly and very easily.

After spending all of this time investigating machine learning and object detection, I have gained a new appreciation for the field and have definitely found a new area of interest. It has already started to make me think about my daily life in a different way. I am starting to look for ways I could incorporate object detection into my life or how I could create an app that other people would find useful. Moving forward I want to learn more about the topic and even potentially seek out a career that has something to do with it.

After all of this, comes the big question: Would I consider this project a success? From a pure objective perspective, the answer would be no. I did not accomplish even close to all that I wanted to. I barely have the MVP that I hoped to build before winter break and have not included anything more complicated than static letters, and even that is not yet as accurate as I would like. Despite falling short in this regard, I would absolutely consider this project a success. I wrote in my proposal that a thesis should "challenge a student to take it upon themselves to learn and become experienced in a new topic or idea". The degree to which this project has done that for me has been incredible. Without taking this project on, I would have never, in my time at UVM, had any experience in either ASL or object detection. Now I want to continue learning about both. I am proud of the project that I have built, and I think it will serve me very well going forward. It will give me a lot of good answers to classic interview questions such as "Tell me about a big project you've completed" or "tell me about a time when you failed". I am usually very much a result driven person, but over the course of this thesis I have learned just how much you can gain along the way to those results and how an experience can be valuable even if you don't succeed in your goals.

References

Abhishek, R. Quora, www.quora.com/What-is-MobileNet.

Clason, Debbie. "The Importance of Deaf Culture." Healthy Hearing. N.p., 14 Sept. 2017. Web. 16 Sept. 2019.

"Common Objects in Context." COCO, cocodataset.org/#home.

Forson, Eddie. "Understanding SSD MultiBox - Real-Time Object Detection In Deep Learning." Medium, Towards Data Science, 9 June 2019, towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.

Gandhi, Rohith. "R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms." Medium, Towards Data Science, 9 July 2018, towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

"Object Detection: TensorFlow Lite." TensorFlow, www.tensorflow.org/lite/models/object_detection/overview.

SignAll. SignAll, n.d. Web. 18 Sept. 2019.

TensorFlow. "Training and Serving a Realtime Mobile Object Detector in 30 Minutes with Cloud TPUs." Medium, TensorFlow, 27 Sept. 2018, medium.com/tensorflow/training-and-serving-a-realtime-mobile-object-detector-in-30-minutes-with-cloud-tpus-b78971cf1193

Training Image Classifier Using Tensorflow Object Detection API." GeeksforGeeks, 28 Oct. 2019, www.geeksforgeeks.org/ml-training-image-classifier-using-tensorflow-object-detection-api/

"Understanding the Deaf Culture and the Deaf World." Understanding the Deaf Culture and the Deaf World | The Outreach Center for Deafness and Blindness. The Outreach Center for Deafness and Blindness, n.d. Web. 16 Sept. 2019

"UW Undergraduate Team Wins $10,000 Lemelson-MIT Student Prize for Gloves That Translate Sign Language." UW News. N.p., n.d. Web. 18 Sept. 2019.

Victor Dibia, HandTrack: A Library For Prototyping Real-time Hand Tracking Interfaces using Convolutional Neural Networks, https://github.com/victordibia/handtracking

Xu, Joyce. "An Intuitive Guide to Deep Network Architectures." Medium, Towards Data Science, 15 Aug. 2017, towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41.