A.L.V.I.N.N. Final Report

Autonomous Learning Vehicle Incorporating Neural Networks

Dec1709

Rockwell Collins – Josh Bertram

Drs. Jones & Zambreno

Bijan Choobineh – Team Leader
Darren Davis – Team Leader
Tracy La Van – Communication Lead
Jesse Luedtke – Key Concept Holder
David Schott – Key Concept Holder
Robert Stemig – Webmaster

dec1709@iastate.edu
http://dec1709.sd.ece.iastate.edu/

Revised: December 6th, 2017

IOWA STATE UNIVERSITY

ALVINN

Table of Contents

1 Introduction

This project was in collaboration with Rockwell Collins to use computer vision, machine learning, and neural networks to have a drone detect objects in the air. Objects detected could include airplanes, helicopters, other drones, flying birds or mammals, and stationary items such as buildings. This required the team to acquire theoretical knowledge in the computer vision and machine learning fields to be able to identify and implement the software and hardware requirements capable of performing these tasks. Furthermore, part of the project consisted of determining which machine learning algorithms are particularly suitable for the object recognition task.

Computer vision tools were used to preprocess images from a picture and video stream, as well as extracting key features of objects. Machine learning was then incorporated to perform teaching operations for identifying objects or distinguishing between different objects based on those key features. Since this problem is interdisciplinary in nature, the team had to investigate to what extent computer vision techniques could support the machine learning based detection algorithms.

The next few sections will describe the purpose of this project, the intended users, and the operating environment, as well as the desired final outcomes or goals of the project.

1.2 Purpose

Rockwell Collins is an aerospace electronics company for military and commercial aircraft. This technology could be used in a military setting; object detection could be used to survey an area before sending in troops by detecting enemy forces or identifying bombing locations of strategic targets. Commercial use could include finding hotspots in forest fires, locating remote wreckage sites, finding lost hikers, or finding survivors of a natural or manmade disaster. The diversity of these different use-cases shows the extensibility, power, and usefulness of the project.

1.3 Goals

Rockwell Collins' primary goal for this project was to detect objects in the air. To accomplish this goal, the project was broken into smaller, more manageable parts. First, the team invested time in expanding their knowledge base on computer vision, machine learning, and neural networks. Upon acquisition of this knowledge, the next part the team decided on which software to use and how to use it for object detection. To accomplish this goal, detection began with detecting a simple object (the letter 'X') on a plain background. The difficulty of detecting an object then was increased by adding in background noise, similar objects, and permutations of the same object. These techniques used for detecting a simple object were applied towards detecting more complicated, airborne objects – specifically airborne aircrafts. The system was taught how to detect these objects in the same manner. With these parts in place, the final phase implemented the parts together into a customized object detection pipeline.

2 Deliverables

This section describes the necessary deliverables and project timeline for the project.

2.1 Overview

To meet the goals outlined in the proposal, the following deliverables were necessary:

Phase I: Education and determining what software/hardware to use

- Learn about computer vision, neural networks, and machine learning
- Determine what hardware to use
- Determine what software to use based on its capabilities and ability to integrate with the selected hardware

Phase II: Detecting a simple object

- Image processing - show an image, gray scale, threshold, ORB feature detection, homography, and object matching (detecting an 'X')
- Detect an 'X' with background noise in an image
- Detect multiple 'X's in an image (with and without background noise)

Phase III: Extending Phase I by detecting objects other than 'X's

- Detect airborne objects using elementary machine learning techniques
- Incorporate cascade classifier training using OpenCV and its Caffe framework.

Phase IV: Detect and classify basic images such as 'X's or handwritten digits with a simple neural network

- Training a network with images of 'X's and handwritten digits
- Discover the limitations of simple neural networks
- After detecting an object, report possible object identifications with associated confidence levels

Phase V: Detect and classify complex images with more advanced and deeper neural network models

- Possibly use Convolutional Neural Networks
- Use complex images that contain many objects in different locations and orientations
- Detect multiple objects and multiple complex objects within an image
- Identify objects even if objects have several different appearances (e.g. detecting a bird with wings by its sides and with its wings expanded)

Phase VI: After successfully being able to identify objects, begin feeding the system back-to-back images, building up to real-time image capture and video feed

Possible Bonus Features to implement

- Tracking movements of identified objects
- Feature Recognition: Ability to recognize various features present on objects (i.e. colors, symbols, unique traits, etc.)
- Use multiple cameras
    - 360° View: Like Nissan's 360° view technology around their vehicles, have a 360° view of what is around the system laterally and what is above and/or below the system
    - Explore whether thermal imaging or night vision could be incorporated to the proposed model
- Track fellow systems in air and follow the system via autopilot

2.2 Timeline

The timeline below in *Figure 1* shows the goal of the project, directly related to the phase descriptions above. The first semester was heavily focused on education, determining the hardware to use and selecting which software interfaces work well with the hardware to accomplish the goals set forth for the project. The second semester was heavily focused on producing the deliverables outlined within each phase of the project.

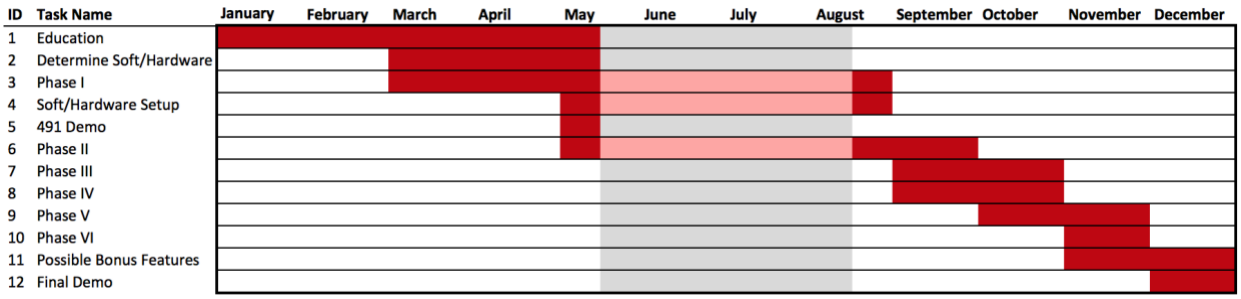

Figure 1: Project Timeline

2.2.1 First Semester

The project breakdown for the first semester of senior design can be seen below in *Table 1*. Most this semester was spent on education on the topic, on the chosen hardware, and on software that will integrate with the hardware. Various phases may be lightly touched on through education, but no serious programming was accomplished until the second semester.

Week Beginning/Ending			Task(s)	Assignment	Deliverables
February	13	19	Education	All Members	
	20	26			
March	27	5			
	6	12	Begin Setup & Coding	All Members	Design Document v1
	13	19	Begin Phase I	All Members	[Spring Break]
	20	26			
	27	2			Project Plan v2
April	3	9			
	10	16			
	17	23			Design Document v2 & Project Plan v3
	24	30			491 Demo
May	1	7	Begin Phase II	All Members	

Table 1: First Semester Project Breakdown

2.2.2 Second Semester

The project breakdown for the second semester of senior design can be seen below in *Table 2*. This semester was focused on using the education from first semester and applying it towards programming the software and hardware to produce the desired deliverables. Assignments for phases were broken up and sometimes assigned to specific team members once the education phase from the first semester was completed. Waiting to assign tasks allowed the team to judge the strengths of each member more

accurately. This also allowed the team to work on multiple phases at once and extend the amount of time needed on a phase, if necessary.

Week Beginning/Ending			Task(s)	Assignment	Deliverables
August	21	27	Continue Phase II	All Members	Phase I
September	28	3	Begin Phase III	All Members	
	4	10			Phase II
	11	17			Phase III
	18	24	Begin Phase IV	All Members	
	25	1			
October	2	8	Begin training with FlightGear & gathering images for detection	Bijan, Robert	Phase IV
	9	15	Begin Phase V	All Members	
	16	22			
	23	29			Phase V
November	30	5	Begin Phase VI	All Members	
	6	12	Begin Preparing for Demo & Testing	All Members	
	13	19	Begin Final Documentation; Begin Poster	Darren, Tracy, Jesse; David, Bijan, Robert	Phase VI
	20	26		Darren, David	[Thanksgiving Break] Final Testing, FlightGear
December	27	3	Final Demo/Presentation Preparation	All Members	Final Report, Poster
	4	10			492 Demo & Presentation

Table 2: Second Semester Project Breakdown

3 Literature Review

Below are several resources and research studies previously done on the topic of image detection using machine learning and computer vision. Besides those listed below, machine learning work has been done before on image detection and movement. For instance, Google has Google Draw Neural Network.

Deep learning has been a very popular area of research so there have been many reputable papers published over the years. Additionally, a specific type of neural network called a Convolutional Neural Network (CNN) has become popular recently and many more papers on the subject have been published. Although some sources the team considered were at a higher level of abstraction in terms of their discussion, the references are incredibly advantageous and provided the team with a theoretical understanding of neural networks and machine learning.

3.1 Previous Literature

One important aspect in the A.L.V.I.N.N. project was trying to understand the interplay between machine learning and computer vision. Asaeedi *et al*[1]. describes topological and geometric algorithms that could be of use for detecting (ideally) highly distinguishable points in feature detection tasks. The paper discusses a generalization of convex hull called α-convex hull that also incorporates the smoothness of the computed convex hull spanning all desired points. From object decomposition to shape approximation, or even pathfinding: alpha shapes have an enormous range of potential applications. The application of α-convex hull which is particularly relevant to the A.L.V.I.N.N. project is point pattern matching. This is because point pattern matching could be used to more accurately discriminate between objects detected by faster/naive approaches. The algorithm described in the paper should perform admirably in object detection tasks across images that contain a lot of noise; however, this also comes at the price of heightened computational cost. The anticipated benefits and costs of integrating such a computationally complex algorithm into the object detection pipeline must be carefully weighed.

The paper by Yann Lecun *et al.*[2] is one of the first demonstrating the concepts of a CNN. It describes the fundamental differences from a traditional feedforward neural network that make a CNN more applicable for image classification. For example, people are able locate and identify specific objects in images because human visual cortexes notice the spatial orientation of colors and shapes. CNNs achieve this by subsampling small blocks within an image. This preserves the spatial structure of objects in images and allows a CNN to locate an object regardless of its location, rotation, and scale. Understanding the way neural networks function is essential for the A.L.V.I.N.N. project and this paper gave the team a good insight into CNNs.

The CNN referenced in Yann Lecun *et al.* is named LeNet and it performs five steps when analyzing an image. The steps are convolution, max pooling, convolution, max pooling, and finally the resulting data is passed to a traditional feed forward neural network. The convolution layer applies six different trained filters to the input image. The filters extract six feature maps from the image which are then compressed during the max pooling layer. These steps are repeated once until the processed data is passed into a fully connected feed forward neural network.

---

[1] Asaeedi, Saeed, et al. "α -Concave hull, a generalization of convex hull." Theoretical Computer Science, vol. 1309.7829, 30 Sept. 2013, doi:10.1016/j.tcs.2017.08.014.

[2] LeCun, Yann, et al. "Gradient-Based learning applied to document recognition." Proceedings of the IEEE, vol. 86, no. 11, Nov. 1998, pp. 2278–2324., doi: 10.1109/5.726791.

In the paper "Video Super-Resolution with Convolutional Neural Networks" by Kappeler *et al.*[3], the issue of video super resolution among CNNs is addressed. In this paper, the team introduced a video SR algorithm using CNNs. The proposed CNN exploits spatial as well as temporal information. Having investigated different architectures each had shown their advantages and disadvantages. Using motion compensated input frames, filter symmetry enforcement and a pre-training method they could improve the reconstruction quality and reduce the training time. Finally, an adaptive motion compensation scheme to deal with motion blur and fast moving objects was introduced. The results presented an algorithm that outperforms the current state-of-the art algorithms in video SR. The potential of use of this in real-life application could help with the issue of motion-blur from the camera or the objects in motion.

The same authors, Kappeler *et al.*, published a similar piece later in 2016 titled "Super-Resolution of Compressed Videos using Convolutional Neural Networks." [4] The paper tackles the issue regarding being able to generate high and ultra-quality video efficiently. The paper discusses how this topic has been analyzed for a long time in which low quality video is used, and higher quality video is an estimated output of the low-quality video. What the paper introduces, is a neural network approach to the problem, using neural networks to be able to take in a low-quality video stream and generate a high quality or ultra-quality video stream respectively. To do this, their proposed algorithm first trains a network using three layers of convolution on a training set. To compensate for motion blur in video, they used the Druleas Algorithm accordingly which uses the (CLG-TV) approach to be able to effectively handle even large displacements. The paper concludes by discussing their implementation and verifying the conversion from low to high quality data. The paper discusses that to the best of their ability there is no other existing algorithm that using CNNs to approach the SR algorithm problem. This paper has a significance to the project, since both working with the Caffe framework and with input data coming into the aircraft detection network. The ability to get a clearer higher quality picture would make the detection rate go up accordingly and thus make aircraft detection more efficient and accurate.

There are several different premade neural networks available on the Internet today. One such neural network is ImageNet. ImageNet was designed to improve the performance and accuracy of CNNs. The paper by Krizhevsky *et al*[5]*.* goes into detail on the differences made to the structure of the network and the training process. CNNs before ImageNet were typically trained on datasets with tens of thousands of images. To be able to identify thousands of different objects, millions of data points are needed. The problem with using millions of data points, is that the network needs to have a much larger learning capacity compared to previous designs. The creators of ImageNet achieved this by simply adding more layers to the different stages inside a CNN.

[3] Kappeler, Armin, et al. "Video Super-Resolution with Convolutional Neural Networks." IEEE Transactions on Computational Imaging, vol. 2, no. 2, 30 Mar. 2016, pp. 109–122., doi:10.1109/TCI.2016.2532323.

[4] Kappeler, Armin, et al. "Super-Resolution of compressed videos using convolutional neural networks." 2016 IEEE International Conference on Image Processing (ICIP), 19 Aug. 2016, pp. 1150–1154., doi:10.1109/icip.2016.7532538.

[5] Krizhevsky, Alex, et al. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM*, vol. 60, no. 6, 2017, pp. 84–90., doi:10.1145/3065386.

The paper by Simonyan *et al.*[6] covers a deep neural network, known as VGG, that was designed so that it was less complicated and deeper than previous neural network models. This model also changed some of the hyper parameters specific to CNNs such as the kernel size and stride. VGG altered these parameters so that more information was processed for each image. In the A.LV.I.N.N. project, DetectNet uses ImageNet which is trained in similar fashion to VGG as mentioned in this paper. ImageNet, as mentioned in previous reviewed papers, is trained on 1.2 million images covering a thousand different classes. This creates a drawback in that it requires a slower, lengthy training process. To avoid this problem, the team gained access to Iowa State University's high-performance computing (HPC) cluster to use its computing power for training.

Dehmamy *et al.*[7] discussed how they trained a deep neural network with a rectified linear unit (ReLU). The paper went into detail about low-lying layers of a neural network could be replaced with activated layers that have been condensed using eigenvectors and matrices. The paper detailed how one could solve the first layer of the neural network and then solve subsequent layers. They also proposed a way that this training could be done recursively. The goal was to reduce the computational complexity of training neural networks. This paper was more of an informational piece for the A.L.V.I.N.N. project due to its use of a deep neural network. It was interesting to see the layers of the filters created on the MNIST dataset of handwritten digits because the team originally experimented with MNIST while learning about neural networks.

The above papers and articles provided the team with a foundation to building the A.L.V.I.N.N. project from the ground up. Though the projects were not all similar in nature, the insights proved to be valuable when learning about various neural networks and how machine learning and computer vision relate to one another. Two articles were found that specifically used machine learning with drones and the NVIDIA Jetson TX1 board which were instrumental in helping the team decide which neural networks to consider for the project. These two articles are discussed in more detail in the next section.

3.2 Related Work

In recent years, the use of drones has exploded from a personal toy to an industrial tool. Drones are being used and considered in various industries, such as Amazon contemplating package delivery via drone, mines managing equipment and aggregate wit drones, and police are using drones for search and rescue operations. Drone utilization has become possible due to embedded systems, neural networks, and computer vision working together to quickly process and identify objects within an image.

While researching previous work, the team found two similar projects both focusing on drones and autonomous flight. Although these projects incorporated tracking, tracking is not a focus in the A.L.V.I.N.N project; however, it has been included it as a feature that could be added in the future. The paper from

[6] Simonyan, K., Zisserman, A. 2015, 'Very Deep Convolutional Networks for Large-Scale Image Recognition' Paper presented to International Conference on Learning Representations, San Diego, CA, May 7-9, 2015. viewed 10-22-17, https://arxiv.org/pdf/1409.1556v6.pdf.

[7] Dehmamy, Nima, et al. "Convergence of Deep Neural Networks to a Hierarchical Covariance Matrix Decomposition." Computing Research Repository, vol. 1703.04757, 14 Mar. 2017, arxiv.org/abs/1703.04757.

Han *et al*[8]. does an additional comparison on NVIDIA boards which has proved useful as the client suggested the use of an NVIDIA Jetson TK1 or TX1 board for this project.

When considering these boards, there was an immediate distinction in size, with the TK1 being considerably smaller and more likely to fit on any drone. As discussed in Han *et al.*, the TX1 has similar power requirements to that of the TK1, but the TX1 is three times faster at object detection. The authors' comparison for tracking the TK1 and TX1 both processed at 71 fps. This means the KCF algorithm the authors were using is light enough that it did not require the additional computing power of the TX1. They also presented a way to make the TX1 smaller than the TK1 by using a carrier board. The team's choice to use the TX1 board over the TK1 was confirmed after reading the Han *et al.* findings.

The other interesting part of Han *et al.* is their research and decision process for choosing a neural network. This has been a main topic of debate in the A.L.V.I.N.N. project as well. On one hand, speed is a major concern when dealing with detecting objects in real-time but, on the other hand, accurate classification could be just as important depending on the use of the system. They looked at "off-the-shelf" networks such as Fast R-CNN, Faster R-CNN, and YOLO Detector. In the end, they went with their own small and relatively shallow network. Based on these findings, efforts were made to split the A.L.V.I.N.N. project into two teams: some members focused on developing a network while the remaining members are experimented with "off-the-shelf" models. These "off-the-shelf" models include DetectNet, based on the Google LeNet network structure for accuracy, Single Shot MultiBox Detector (SSD), YOLO version 2, Tiny YOLO, and Google MobileNets.

DetectNet is an NVIDIA network that comes preinstalled on the TX1 board. The decision was made to try this network instead of switching to one of the R-CNNs mentioned in both papers. Smolyanskiy *et al.*[9] stated R-CNNs are not ideal for running in real-time on the TX1 and Han *et al.* confirms this by stating they only switch to it when accuracy is too low with the KCF tracking model. These confirmations, along with having DetectNet already on the board, let the team to believe this would be a promising approach to begin working with a neural network.

Both Han *et al.* and Smolyanskiy *et al.* discuss the YOLO network and Smolyanskiy *et al.* also talks about the SSD network. Smolyanskiy *et al.* brought up support issues with the SSD network and thus used a modified version of YOLO. Han *et al.* decided against using the unmodified YOLO network because it did not provide the accuracy they wanted; however, since their publication, there is a new version of YOLO that promises to have improved accuracy while being even faster. There is a newer version of TensorRT (NVIDIA's deep learning inference optimizer) that supports the SSD and YOLO networks.

One idea brought up in Han *et al.* that the team had not considered was switching between networks for different tasks. Han *et al.* used a very fast model for tracking and switched to a more accurate R-CNN when accuracy was reduced past their desired threshold. If one network cannot provide both speed and accuracy, a potential solution is to use one network geared towards fast results and another geared towards object

---

[8] Han, S. Shen, W. Liu, Z. *Deep Drone:* (2016) *Object Detection and Tracking for Smart Drones on Embedded System.* Stanford University. Retrieved from https://web.stanford.edu/class/cs231a/prev_projects_2016/deep-drone-object__2_.pdf.

[9] Smolyanskiy, N. Kamenev, A. Smith, J. Birchfield, D. (2017) *Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness* Retrieved from https://arxiv.org/pdf/1705.02550.pdf.

prediction accuracy. Speed would be important if, when flying along a designated route, the drone needs only to detect objects to avoid a collision. Whereas, accuracy would be important in purposefully trying to classify the objects the drone comes across in its path.

These two papers provided useful research findings. The papers confirmed the team's choice in using the NVIDIA Jetson TX1 board for the project. The papers provided insight into different neural networks that have been used for accuracy and speed and have confirmed that some of the networks being considered for this project have been successfully used in other projects with similar goals. Though the idea ended up not being implemented, the idea for the use of more than one network for different flight scenarios expanded the realm of possibilities that the A.L.V.I.N.N. project could tackle to achieve its goals.

4 Design

This section covers the various requirements for the project, including non-functional, functional, and resource requirements. In addition, it covers safety considerations for the implementation of Project A.L.V.I.N.N. and the associated professional ethics and standards.

4.1 Project Requirements and Specifications

The specific objective of the project was to detect and identify airborne aircraft through means of computer vision and neural networks. The project incorporated open-source deep learning frameworks and open source computer vision library to deploy a neural network on an embedded board. Alongside the specific objective, the team was required to meet specific functional and non-functional requirements. The non-functional requirements set criteria for the performance of the system while the functional requirements define the behavior of the system.

The team determined the resources needed to accomplish the goal of the project. Since the work was done in a lab environment, the resource requirements are specific to the lab. Incorporating the project into a different environment, such as a drone, will change the resources required. The following sections will breakdown the non-functional, functional, and resource requirements for the project.

4.1.1 Non-Functional Requirements

The following list includes the non-functional requirements for the project:

- Time performance: The object detection processing pipeline for a single image should take no longer than 300 ms and continuous video feed should process no less than three frames per second.
- Extensibility: The data should be outputted in an elastic way such that future users could easily integrate and use the data in other systems (i.e. using JSON-format).
- Object detection accuracy: The image processor should be able to identify a desired object 50% of the time on images where the object in question is clearly discernible by a human eye.
- Reliability: The image processor should be able to run sustainably for 30 minutes without crashes or other execution interruptions (i.e. due to overheating, memory leaks, or other software bugs).

4.1.2 Functional Requirements

The following list includes the functional requirements for the project. The image processor must be able to

- process a single picture, string of pictures, and/or continuously moving video.
- detect stationary objects apart from the background.

- detect moving objects apart from the background.
- detect multiple objects of the same or different type in a single image.
- report confidence levels of any identified objects.

4.1.3 Resource Requirements

The following list includes the resource requirements for this project:

- NVIDIA Jetson TX1 board - $344 (possible educational discount available)
    - or NVIDIA Jetson TK1 board - $192
- Logitech c270 webcam - $22
- USB hub - $10
- 16GB flash drive - $10
- 8GB SD card - $10
- Other requirements:
    - Standard computer equipment: Tower/monitor, mouse, keyboard, cables
    - Monitor with HDMI capabilities
    - Access to Iowa State University's High Performance Computing Lab

4.2 Safety Considerations

As with any new technology that is investigated, it is important to consider safety issues that may arise with the technology. An immediate concern is using the project to detect specific objects as an incorrect detection could lead a user to make an inappropriate decision, e.g. an incorrect detection of in a military setting could lead to disastrous consequences. Similarly, failing to make a detection could lead a drone using the technology to crash into an airborne object.

To run a neural network takes up a good amount of computing power. This project is meant to be used on an embedded board that lies inside a drone. Due to the space constraints of a drone and the computing power necessary to use the board to detect objects, overheating is a concern. Overheating or overuse of the boards memory could lead to technical malfunctions resulting in the drone failing to correct report airborne objects or cause the drone to crash.

4.3 Standards and Ethics

When working on this project, the team will be adhering to several standards. First, all code that will be written will standardized and documented, making it easy to maintain, understand, and debug if necessary. To ensure high software quality and team cohesion, software peer reviews will be incorporated to ensure that all team members are familiar with each other's code and purpose. For any code written in C++, C++14 standards[10] will be used. Furthermore, since a lot of the code base will initially be written using the Python programming language, PEP 8 recommended style guide for Python will be followed. There are several IEEE standards which are applicable to the project. These standards include floating point standardization, test methodology standardization, as well as code standardization. It will be important to adhere to these

[10] "ISO - International Organization for Standardization." Information technology – Programming languages – C++, 30 Nov. 2017, www.iso.org/standard/64029.html.

standards for when the code is transferred to the client's system. The Code of Ethics[11], Code of Conduct[12], and Computer Technology Standards[13] of IEEE will be followed throughout this project.

[11] IEEE Code of Ethics. IEEE, www.ieee.org/about/corporate/governance/p7-8.html.

[12] IEEE Code of Conduct. IEEE, June 2014, m.ieee.org/about/ieee_code_of_conduct.pdf.

[13] "Computer Technology Standards." IEEE Standards Association. IEEE, 2017. Web. 31 Mar. 2017. http://standards.ieee.org/cgi-bin/lp_index?status=active&%3Bpg=40&%3Btype=standard&%3Bcoll=15.

5 Design

This section provides a high-level overview of the system in a block diagram and provides a short discussion about the different approaches available and the approach that was taken with the project.

5.1 Proposed System Block Diagram

A very high-level overview of the system can be seen in *Figure 2* below. The system input is either a video feed provided by an open source flight simulator, FlightGear, or individual images. OpenCV is used to capture the input and resize it to network specific dimension. The information is then sent to the Caffe deep learning framework where the trained neural network will predict what the image contains. Finally, the system outputs to the user its prediction on what it believes the object in the image is as well as its confidence levels.

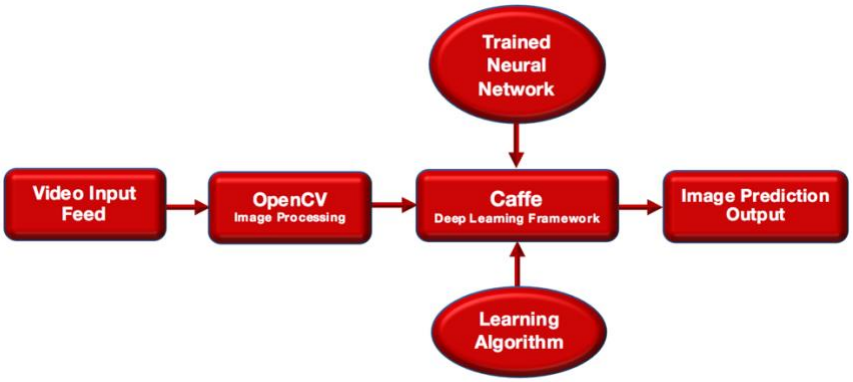

Figure 2: A.L.V.I.N.N. Block Diagram

5.2 Assessment of Proposed Methods

The following subsections describe an approach and solution for every aspect of A.L.V.I.N.N. These include hardware, software, image preprocessing, computer vision, and neural networks.

5.2.1 Hardware and Software

In terms of hardware, the NVIDIA Jetson TK1 and TX1 boards were suggested by the client. After considering both boards, the team has decided to go with the TX1 board which can be purchased as a kit incorporating I/O, HDMI, USB, WI-FI, and a camera interface. NVIDIA also offers CUDA, an application programming interface, to deploy the neural network architecture onto the GPU. All requirements for meeting the goals outlined for the project should be possible using these predefined interfaces. Other possible interfaces, outside the scope of this project, would include the onboard GPS and the autopilot system Rockwell Collins has running on the drone.

In terms of software, there are multiple viable options. MATLAB is a programming environment and language all in one. Used by engineers and scientist for visualizing data to develop and running algorithms in the areas of robotics, signal processing, and image processing. MATLAB presents itself as a viable option because it has toolboxes for computer vision, machine learning, and neural networks. OpenCV is an open source computer vision library that can be used with many of today's popular programming languages. The functions in OpenCV are specifically written towards computer vision with focus on areas such as two-dimensional and three-dimensional features, motion tracking, learning, and artificial neural networks. Google's TensorFlow is another open source library which focuses more on machine learning and neural

networks and is available for C++ or Python programming languages. Caffe is an OpenCV framework that can be used for machine learning and neural networks.

The team has decided to go with Python as the programming language of choice for libraries because more team members have used it than C++. For those team members who have not used either language, Python is said to be the easier of the two to learn. OpenCV's focus is computer vision thus it has more functions to work with for this project. OpenCV is open source versus a hefty price tag for MATLAB. After consulting different popular technologies and frameworks, it was decided that the Caffe deep learning framework would be the best choice due to its expressive architecture, extensible code base, and speed optimizations.

5.2.2 Image Preprocessing and Feature Detection

Images are the heart of this project and without images, there are no objects to detect. Because of this, being able to read an image and work with it is an important first step. Image processing involves altering the image to help acquire specific information quickly. For example, a green square could be found by removing all other colors first and then looking for specific details, or key features, of a square. Other image processing techniques can help remove noise such as discolored pixels that appear as specks or dots in an image.

Image preprocessing was done entirely in OpenCV. To begin this process, the team first learned how to read an image into the program. This was a straightforward technique requiring one line of code. With access to the image, work could begin by processing the image. The first processing technique used involved converting from a colored image to a gray scaled image. At such an early stage in the project, colors are not a big concern so it is easiest to eliminate them. This takes away the complexity of working with the red, green, and blue color intensities out of the equation and leaves just the one-color intensity.

Next, blurring techniques were considered to help remove noise and smooth out rough edges. In its simplistic form, blurring works by looking at a pixel and the surrounding pixels to determine what color it should be. The blurring methods explored included OpenCV's blur, Gaussian blur, median blur, and bilateral filter.

After blurring techniques, thresholding was the next image processing technique that was considered. This technique involves setting a threshold for the color intensity. A limit is set based on the gray scale intensity or shade of gray. Shades either above or below the limit can be removed while the remaining shades can be converted to all one shade. The thresholding technique along with blurring can be used to remove noise and provide a clean black and white image.

Once blurring and thresholding techniques were applied, an attempt was made at detecting features in an image. There are a few techniques available for detecting features that were considered including Harris

Corner Detector[14], Shi-Thomasi Corner Detector[15], Scale-Invariant Feature Transform (SIFT) [16], Speeded-Up Robust Features (SURF)[17], FAST Algorithm for Corner Detection[18], Binary Robust Independent Elementary Features (BRIEF)[19], and Oriented FAST and Rotated BRIEF (ORB)[20]. After learning about each one, it was decided to focus on ORB since it was developed by OpenCV combining the FAST and the BRIEF techniques and matches performance of the SIFT and the SURF technique.

After detecting features in an object, the next logical step was trying to match features in one image to those in another image. Two methods for matching features were used: a brute force matcher and a Fast Library for Approximate Nearest Neighbors (FLANN)[21]. When comparing the same image both methods worked equally well. Matching an image to the geometrically similar image but on a different scale and matching an image to an image with multiple objects was also tried. The brute force method made more matches between images; however, not all of them were correct. The FLANN method made fewer matches compared to the brute force method, but the method also had fewer errors.

These techniques are time consuming and provided poor results when trying to match objects based on features. To keep processing time of the system to a minimum image preprocessing will be limited to resizing. For resizing, an image will be converted to a size that works best with a given neural network.

5.2.3 Computer Vision

Traditional computer vision methods utilize similar techniques as those discussed in Section 5.2.2, to detect objects in an image. First a set of images containing the object are processed and key features are extracted into a vector representing the object. Labels can be applied to these image sets if multiple objects are to be distinguishable.

---

[14] "Harris Corner Detection." Harris Corner Detection — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners.

[15] "Shi-Tomasi Corner Detector & Good Features to Track." Shi-Tomasi Corner Detector & Good Features to Track — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi.

[16] "Introduction to SIFT (Scale-Invariant Feature Transform)." Introduction to SIFT (Scale-Invariant Feature Transform) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro.

[17] "Introduction to SURF (Speeded-Up Robust Features)." Introduction to SURF (Speeded-Up Robust Features) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html#surf.

[18] "FAST Algorithm for Corner Detection." FAST Algorithm for Corner Detection — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html#fast.

[19] "BRIEF (Binary Robust Independent Elementary Features)." BRIEF (Binary Robust Independent Elementary Features) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html#brief.

[20] "ORB (Oriented FAST and Rotated BRIEF)." ORB (Oriented FAST and Rotated BRIEF) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html#orb.

[21] "Feature Matching." Feature Matching — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html#matcher.

Test images are then processed by the same methods and the key feature vector extracted. To classify an object the test vector is compared to the training vectors. Based on the number of key feature matches and a threshold value, the image is classified as the label on the vector it best matches if the number of matches are greater than the threshold value.

Computer vision has several methods for evaluating models and improving the model's algorithmic capabilities. Simple model evaluation includes cross validation - a technique that partitions the original sample into a training set and a testing set. A simple type of cross validation called the holdout method. Some more advanced methods of evaluating models include repeated holdout and cross validation. The repeated holdout method is the holdout method repeated multiple times with the dataset redistributed across the training and test sets each iteration. Using the repeated holdout method provides a better estimate on how well the model performs on a random test set while providing an idea about the model's stability. There are numerous methods for performing cross validation. The cross-validation method considered is the k-fold cross validation method. The dataset is divided into k equally sized groups; these groups are called folds. One-fold is left to be used as the test group while the model is trained on the other groups before testing. The process is repeated until each group has been tested and the results averaged to estimate the algorithms capability.

OpenCV has a FaceRecognizer module that can be trained to detect and identify a person by their face. The FaceRecognizer project is not new and there are various tutorials online about how to get started with it. The team decided this was worth exploring to begin exploration in computer vision.

5.2.4 Neural Networks

With the application revolving around neural networks, focus for the project was spent learning about neural network frameworks, such as TensorFlow and Caffe. Initially, the team practiced using these frameworks to create a handwritten digit classifier using a feedforward neural network. This network's structure was based on the format of the MNIST dataset. This dataset is comprised of 60,000 training images written by 250 people and 10,000 testing images written by a separate set of 250 people. Images contain a single centered digit with a 28 by 28 resolution. The benefit to this approach is the simplified design however it also creates many other problems. More information about this design can be found in Section 8.3.

CNNs were also explored due to their success in image recognition. Their advantages make them a better choice for A.L.V.I.N.N. when compared to feedforward neural networks. CNN's are modeled after parts of the visual cortex and are specialized to recognize complex images without drastically increasing processing time per image. They can achieve this by extracting multiple feature maps from an image and as well as compressing the information in the image. The main disadvantage of CNNs is their increased complexity and lengthy training time. These challenges were mitigated by implementing CNNs designed by researchers. Most of the information the team used to learn and implement these networks is contained in Section 3.

5.3 System Constraints

Several system constraints have been identified with the project. Knowing the system would be placed on a drone, the size of the embedded board had to be considered. The NVIDIA Jetson TX1 used by the team is the Developer Kit which may be too large to fit in a drone. NVIDIA offers a stripped-down version that a user can modify to reduce its size. Other physical limitations of the NVIDIA Jetson TX1, such as computing

power and memory, limit what the project can achieve. These limitations affect the speed and accuracy of the networks.

The system is also constrained by how well the neural network is trained. If a system is only trained to detect aircraft, then it will not detect other obstacles such as birds or a balloon that has floated away from a child. Additionally, the amount and type of data used to train the system affects how confident the network is in classifying an object. If a large amount of data is used to train a system, a high-performance computing cluster may be needed to train the networks at a quicker rate than a general computer. Not having access to such a cluster could hinder network retraining abilities.

Assumptions were made during the development process that could potentially create system constraints. The system was intended for aeronautical uses and not for uses on the ground or surveying the terrain from above. It is assumed that the system will be used commercially and not intended for household use.

6 Challenges

The immediately observable challenges faced in this project are knowledge and theory related. The machine learning field was unfamiliar territory to most of the team. The vastness of this field and the open-endedness of the client's expectations resulted in the need to be careful before investing a lot of time in a technique or method that could not be incorporated into the project. Gleaning advice and information from the advisors and project stakeholders, who are much more knowledgeable on machine learning and computer vision, was paramount to a successful project.

The NVIDIA Jetson TX1 board was acquired for this project and comes with no pre-installed software required for basic operations. Since the team did not incorporate the board until later in the project, not enough time was allocated for initial setup and debugging of the board. This was also a challenge for the team as setting up a board is something no team member had done prior to this project. Another challenge with the board included getting the onboard camera working. The team was unable to get this working but could use a USB web camera instead.

Besides a feed from a video camera the team wanted to a use prerecorded video and a video streamed from another source. This presented a challenge as the team incorporated the use of GStreamer, a multimedia framework that the team has never used before. Additionally, the team was unable to incorporate prerecorded video and GStreamer feeds into NVIDIA's proprietary neural network, DetectNet. The issues with the onboard camera and DetectNet are ongoing issues that other users reported experiencing. To date, it is not known whether a fix has been issued to resolve these problems.

This project involved a client, two advisors and six team members. For many team members, this was their first experience being on a large project and part of a bigger team. With so many people involved, communication was going to be key to keep everyone on the same page because lack of communication or lack of clarity led to a few problems, luckily none that hindered development. With the client having an open-ended vision and only very general expectations set forth for the project, the team had to make sure contact with the client was established to ensure the client's satisfaction with the direction in which the project moved. Additionally, team members had other classes and tasks that require their attention outside of this project; members had to be diligent in their work and avoid any lapses in accountability. At times, these other commitments resulted in the slowing of forward progress on the project.

7 Testing and Development

Due to the many components and complexity of A.L.V.I.N.N., thorough testing was needed to ensure a reliable design. This testing process was split into three major sections: image preprocessing and feature detection, computer vision, and neural networks.

7.1 Image Preprocessing and Feature Detection

An image of an 'X' was used to test the effectiveness of different image preprocessing techniques and feature detection methods. The 'X' provides key features of internal and external corners as well as multiple edges. By creating an 'X' in Microsoft Paint the team could control the shape of these corners, the smoothness of the edges, and add noise. Two 'X' images, one with sharp corners and smooth edges and another with rounded corners and jagged edges, were used for testing. These images were used to test blurring and thresholding methods resulting in black and white images that the team compared for increased edge smoothness and reduction of noise.

To further verify the effectiveness of image preprocessing, the team utilized feature detection to identify key features. A rougher edge will contain possible key features that a smooth edge doesn't. Noise in an image may also be mistaken as a key feature. By incorporating feature detection, the team could evaluate how well noise was reduced and edges smoothed by seeing a reduction in the number of features detected.

Matching verifies two identical images by comparing key features. To test how well features could be matched the team compared the same image at different scales (e.g. 'X' vs. 'x'). The testing was also extended to include trying to match an image (e.g. 'X' with 'X') when multiple objects (e.g. 'A B C D …') are present.

7.2 Computer Vision

Regarding computer vision, a basic object recognition use-case was explored using a OpenCV library. Experiments began by using pre-existing code to explore how to perform detection of faces using OpenCV's FaceRecognizer module. Information on how to use the FaceRecognizer was provided by Geitgey[22] and OpenCV's "Cascade Classifier Training". [23] The initial test methods compared the results obtained to results that have been published using the holdout method, a model evaluation method. The method involved dividing a dataset into a training set and a test set with labels for each data point. After the model was trained on the training set, the team could determine how well the model did by comparing its predictions of the testing set with its labels.

As the algorithms improve, more advanced testing techniques will need to be used to thoroughly test the algorithms' capabilities. Some more advanced methods include repeated holdout and cross validation. The repeated holdout method is the holdout method repeated multiple times with the dataset redistributed across the training and test sets each iteration. Using the repeated holdout method provides a better

---

[22] Geitgey, Adam. "Machine Learning Is Fun! Part 4: Modern Face Recognition with Deep Learning." Medium. Medium Corporation, 24 July 2016. Web. 31 Mar. 2017. <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78#.7732t9p7a>.

[23] "Cascade Classifier Training." Cascade Classifier Training — OpenCV 2.4.13.2 Documentation. OpenCV Dev Team, 16 Dec. 2016. Web. 31 Mar. 2017. <http://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html>.

estimate on how well the model performs on a random test set while providing an idea about the model's stability. There are numerous methods for performing cross validation. The cross-validation method considered is the k-fold cross validation method. The dataset is divided into k equally sized groups; these groups are called folds. One-fold is left to be used as the test group while the model is trained on the other groups before testing. The process is repeated until each group has been tested and the results averaged to estimate the algorithms capability.

7.3 Neural Networks

This section covers final testing of the system with each neural network being tested individually. Testing began by looking at a test set of 10,000 images contained within the MNIST dataset. The team recorded the number of correctly classified images by the trained neural network to find its accuracy. Then the team looked further into training convolutional neural networks on much larger datasets with a variety of images. Finally, several pre-trained neural networks were tested against a small set of data. The results were measured a network's ability to detect and classify objects along with its associated confidence level. System performance was measured on these same neural networks to analyze speed, memory usage, and power consumption while streaming video.

As stated in Section 3.1, training neural networks, especially CNNs, can be a time-consuming task. The number of images a network is trained on affects the performance of the network. To achieve a quality network that can accurately detect and classify an object requires training on thousands of images containing the object and thousands of images without the object. Training on this large of a data set can set a project back if the appropriate amount of time is not given. To determine the time constraints and best method for training, the team used a personal laptop and the high-performance computing cluster provided by Iowa State University. A network was trained using both computing devices and the times compared.

To test a network's ability to detect and classify objects, different sets of images were used. The different sets of images contained no aircraft, a single aircraft and multiple aircraft. The set containing no aircraft was used to determine if the network falsely detected an aircraft. The sets with aircraft contained different types of aircraft, in different positions, at different distances from the camera, and over varying backgrounds. This determined if the network performed well at detecting and classifying an object over a variety of conditions or only in specific situations.

Testing the speed, power, and memory performance metrics was done well running a prerecorded video through the system. Timers were used in the code to capture the time when a video frame was passed to the neural network and the time the network finished processing it. From this, the number of frames per second each network was capable of processing was determined. NVIDIA includes a system monitor on the TX1 enabling data, such as memory and power demands, to be collected while the system was running.

Image size has a huge effect on the performance of the system. Larger images contain more data resulting in detecting and classifying objects with higher confidence; however, it takes longer to process. Smaller images are faster to process because they have less data to process but less data results in lower confidence in the classification. Google's MobileNets was tested with three different image sizes to see how processing speed and object detection and classification were influenced by image size.

The team used different statistical methods to compare the testing results of each neural network. Since classification outcomes (aircraft or not) were known on the test data, a confusion matrix was used to

analyze the classification performance of the different neural networks. A resulting table of confusion for aircraft was constructed for each network. From the table, several statistics were analyzed including sensitivity (true positive rate), specificity (true negative rate), precision, negative predictive values, accuracy, miss rate (or false negative rate), and fall out (or false positive rate).

8 Results

This section covers the results of the testing covered in Section 7. Due to timing constraints within the second semester, the implementation of a neural network from scratch was unattainable. As a result, all members of the team focused on finding existing neural networks that would fit the goals of this project. Seven neural networks total were tested: DetectNet, SSD: Single Shot MultiBox Detector, YOLOv2, Tiny YOLO, and Google MobileNets with images sizes of 150x150PX, 300x300PX, and 450x450PX. The existing neural networks incorporated image processing and computer vision into their design; therefore, after running initial testing on image processing and computer vision during the first semester, it was unnecessary to explore these areas further.

8.1 Image Preprocessing and Feature Detection

The results from image preprocessing of a simple 'X' were encouraging. For an initial test, a smoothing, or blurring, process was implemented to a red 'X' help smooth the edges and eliminate noise. *Figure 3* shows the original image and the image after it has been converted to grayscale. At this stage, the 'X' still had rough edges and noise surrounding it.

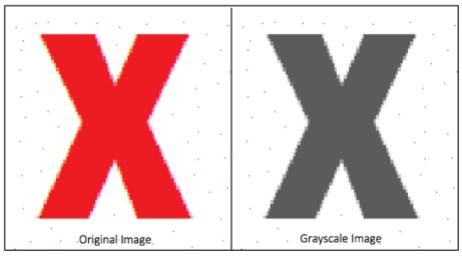

Figure 3: Original and Grayscale Images

The blurring methods explored to smooth edges and eliminate noise included a normalized box filter, a Gaussian filter, a median filter, and a bilateral filter. The results of these methods on the original grayscale 'X' can be seen in *Figure 4*. Both the Gaussian blur and median blur methods appear to have removed the noise. The Gaussian filter also provides a smoother edge than the other three methods.

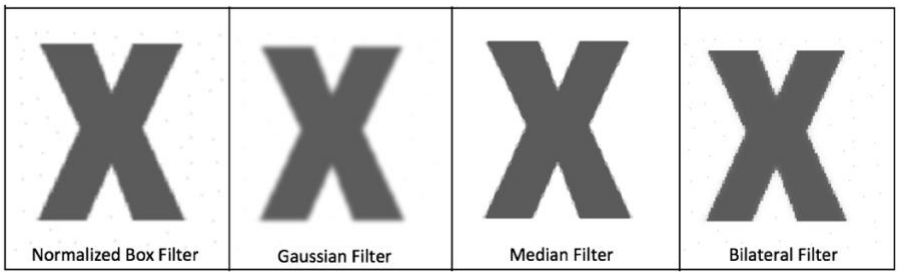

Figure 4: The four different filters used on the grayscale 'X'

For a better comparison, two thresholding techniques were applied to each blurred image: a basic thresholding technique and an adaptive thresholding technique. The results of applying these two techniques is shown in *Figure 5* below. Using a high threshold value, everything in the image that was not already white was converted to black. This showed that more noise was present in the image after blurring than what initially appeared.

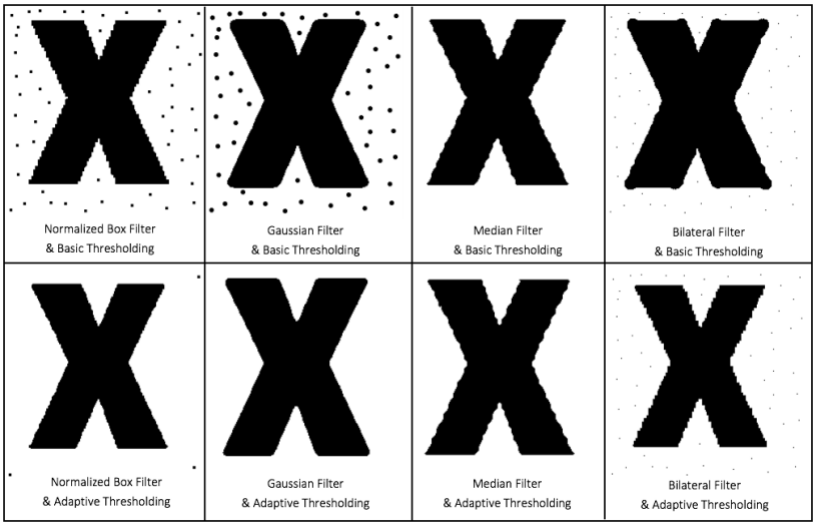

Figure 5: All four filter techniques used in conjunction with a basic and an adaptive thresholding technique using a high threshold value

By adjusting the threshold value, the amount of black noise appearing can be reduced, as seen in *Figure 6*. Comparing the results, it was decided that the combination of the Gaussian filter and adaptive thresholding provided the most noise reduction and smoothest edges without having to lower the threshold value.

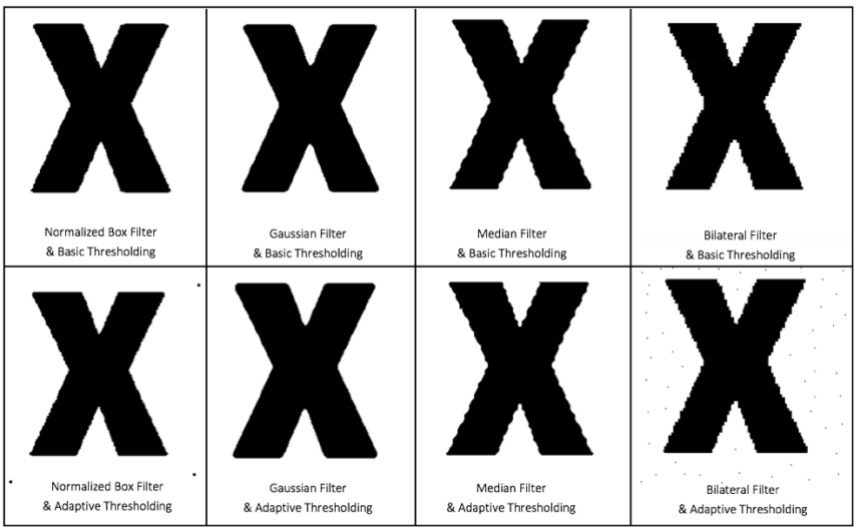

Figure 6: Basic and adaptive thresholding using lower threshold values

Feature detection began with a basic 'X'. Feature detection identified key points in the image that were used to recognize the same type of key features in other images when scanning for potential matches (i.e., detecting similar objects). The rough edges of the original image resulted in too many key points, making it hard to be able to detect other 'X's from the original 'X'. The image was blurred and the thresholding technique applied to reduce the number of key points. After blurring and thresholding, the edges of the original image still had many points that stood out.

Next, the above process was repeated on an 'X' that already had smooth edges and did not need blurring or thresholding techniques applied to it. Beginning with smooth edges allowed the number of key features

to be significantly reduced. The key features of both the original 'X' with blurring and thresholding techniques applied and the smooth 'X' can been identified in green in *Figure 7* below.

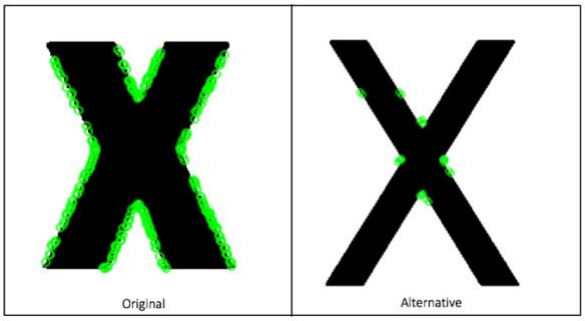

Figure 7: Feature Detection with the original 'X' after blurring and thresholding and with smooth-edged 'X' with key features or key points are shown in green

The feature matching tests that have been performed so far have had unsatisfying results. The first issue was matching key features on the 'X' with similar key features on other object types resulted in too many incorrect matches. *Figure 8* shows feature matching displayed through green lines going from the 'X' to the 'B' and the 'n'. The second issue was matching the wrong points on the correct object. *Figure 8* also shows this issue with two green lines between the 'X' objects with one line going from the bottom center of one 'X' to the top center of the other 'X'. Finally, there was the issue of not matching key points between geometrically similar objects. The small 'x' in *Figure 8* shows no matching points in common with the comparison 'X'.

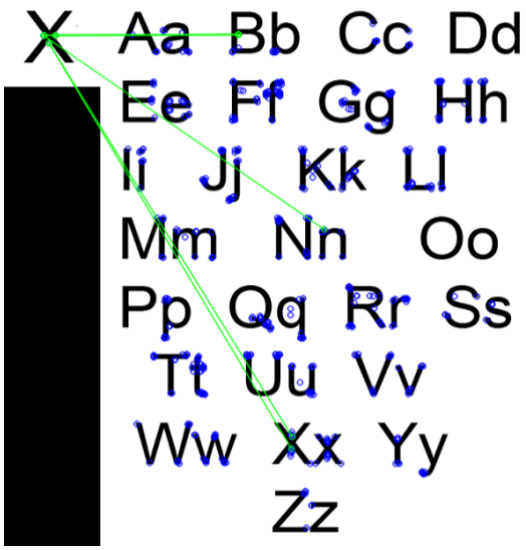

Figure 8: Feature Matching

Though the thresholding results were promising, the data obtained from matching objects was too inconsistent to be of use. These inconsistencies could be related to the chosen feature detection methods. With the off-the-shelf neural networks incorporating image processing into their design, attempts to improve these results were abandoned.

8.2 Computer Vision

Work began by feeding simple images containing faces to the program and converting these images into grayscale. The program was then extended to include the capability to distinguish between faces belonging to different people (by constructing a labeled training dataset). Upon consolidation of this feature, a video stream was fed from the webcam to the facial recognition software. A green square box was drawn around the recognized faces, as well as displaying a name on the command-line, should a familiar face have been encountered. *Figure 9* shows an example of someone being recognized.

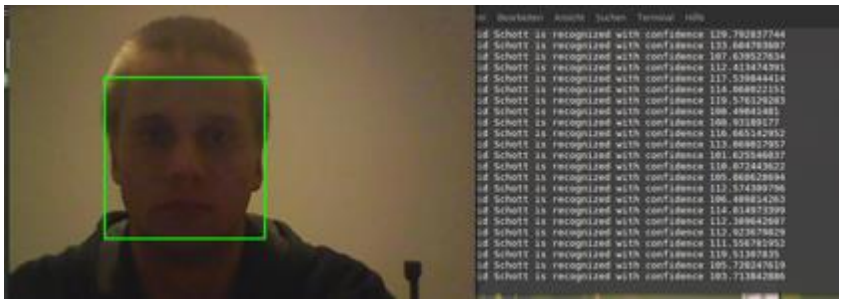

Figure 9: Facial recognition of David Schott

For the module to be able to successfully detect, box, and identify a team member showed promising results. However, with the off-the-shelf neural networks incorporating computer vision into their design, further explanation into computer vision was deserted.

8.3 Neural Networks

To become more familiar with the concept of neural networks and their limitations, a basic neural network was implemented which classified handwritten digits (see Section 5.2.4 for more details). To sum up the findings, the neural network implemented was slow and poor at recognizing complex images. For these reasons, this type of neural network model will not work for the system. The images taken by the system will be very complex and with a lot of noise or distortions. Also, it is important that the images are processed as fast as possible to work towards being able to feed the system continuous video footage.

The teams first neural network design, the handwritten digit classifier, was structured initially with three layers. The first layer contained 784 input neurons, one for each pixel in the image. The second layer, or the hidden layer, contained 15 neurons to compress the information of the previous layer. The final layer consisted of 10 output neurons where each neuron reports the confidence of the class for the input image (numbers 0 through 9). Additionally, the team modified the number of layers as well as the number of neurons in each layer. Increasing both parameters slowed down training but increased the accuracy of the model.

The results from testing the three-layered network with the MNIST dataset showed that 93% of the digits were correctly classified. While this may sound impressive, it was not very reliable when comparing the results to other solutions that solve the same problem. This approach for processing images was naïve because only centered digits could be correctly classified, the network had trouble with digits that were rotated, and the model ails when processing higher resolution images with millions of pixels. Compared to other solutions available, this method is not recommended due to its poor accuracy.

After finishing the work with traditional feed-forward neural networks and understanding the limitations they suffer from, the team decided to explore CNNs because they specialize in image classification. Some

of the major problems with the previous design was that as the images became larger, the network size increases exponentially thus slowing down training and execution speeds. Most importantly, when a feed forward neural network processes an image, spatial locality is not preserved, the image is seen in one dimension when it really is a two-dimensional input. CNNs solve both major problems by introducing image convolution with kernels and by subsampling images down further. This is apparent when observing the testing data collected by the team.

For investigating training speeds, testing began by modifying existing Caffe training programs. A total of 8000 images were used from ImageNet for testing with half of the images containing aircraft and half of the images not containing aircraft. Training took place on two different systems: a personal laptop and the high-performance computing cluster provided by Iowa State University. Specifications from both systems can be seen in *Table 3* below.

Device	Processor	Number of Cores	Memory	Accelerator Card
Lenovo G50-45 Personal Laptop	1x 1.8 GHz 4-core AMD A4-6210	4	8 GB	None
ISU's HPC Node (GPU)	2x 2.0 GHz 8-core Intel E5 2650	16	64 GB	NVIDIA K20 Kepler GPU

Table 3: System Specifications

Initially, 20,000 iterations were run for training. The HPC can run these iterations in a timely manner; however, on the personal laptop it was going to take too long, possibly several weeks. The number of iterations were then reduced to 3,000 and the lapsed training time on both the typical Laptop and Iowa State's Computing cluster was then monitored in a log file. A graph summarizing the elapsed time for the first 3000 iterations can be seen below in *Figure 10*.

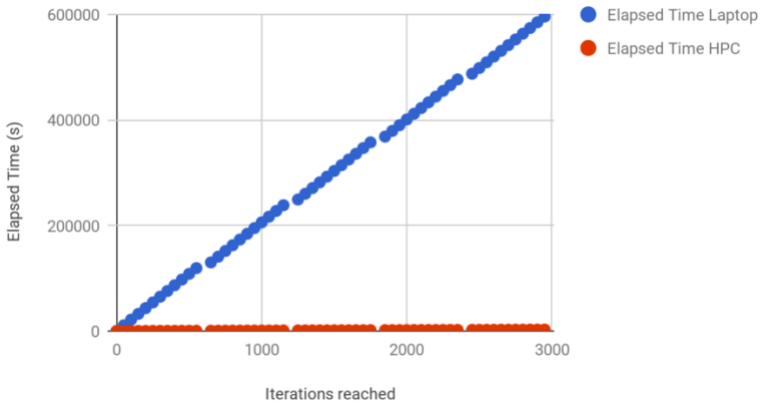

Figure 10: Training Time of Personal Laptop vs. HPC Cluster Node

Based on these training results, the neural network achieved a classification accuracy of 90.9%. The training on the HPC GPU node led to a significant speed improvement over training on a CPU-only personal laptop. GPU acceleration allowed the intensive mathematical computations to be performed 221.3 times faster. In theory, the performance gains from training on the HPC node could be even greater if the step size was adapted to fully utilize the extra memory available on the HPC node.

Using the HPC node to train or retrain a neural network would have been very useful for this project. However, since a neural network was not built from the ground up and since retraining was not utilized due to timing constraints, the team did not need to use the HPC cluster after this initial testing because the off-the-shelf neural network models used were pre-trained.

Five different pre-made neural networks were tested and evaluated on their performance: DetectNet, SSD: Single Shot MultiBox Detector, YOLOv2, Tiny YOLO, and Google MobileNets. Google MobileNets was tested using three different image sizes, 150x150 PX, 300x300 PX, and 450x450 PX, making the total number of neural networks analyzed seven. Each neural network was tested on a set of 30 images containing plane and non-plane objects. For each image, the objects in the image that could potentially be identified by a neural network were recorded. Within these 30 images, a total of 39 aircrafts and 14 other objects (non-aircraft) were identified.

The images were then processed through each of the networks and the data was collected on whether the objects in the images were correctly identified as an aircraft or ignored. For example, one image tested contained two objects: a chinook hauling a Jeep. A neural network should classify the chinook as an aircraft while ignoring the Jeep. If the network correctly identified the chinook as an aircraft at 50% confidence or above, it received a score of '1' for that classification. If it did not detect the aircraft, did not correctly identify the aircraft, or did not have a confidence above 50%, it received a score of '0'. For the Jeep, if a network did not classify the Jeep as an aircraft or if its confidence was below 50%, it received a score of '1'. If a network classified the Jeep as an aircraft at or above 50% confidence, it received a score of '0'.

A table of confusion for aircraft was then constructed for each neural network. The fashion in which the data was scored in the table can be seen in *Table 4*. From these tables, several statistics were analyzed.

		Actual Class	
		Aircraft	Non-aircraft
Predicted Class	Aircraft	True Positives	False Positive (Type I Error)
	Non-aircraft	False Negative (Type II Error)	True Negative

Table 4: Table of Confusion for Aircraft

From these tables, several statistics were analyzed. Sensitivity (true positive rate) was measured to see how each network could correctly identify an aircraft in an image when one was present. Specificity (true negative rate) was measured to see if a network could correctly identify that no aircraft was present in an image when there was no aircraft in the images. Precision was considered to see the percent of aircraft being correctly classified from the total number of aircraft the network detected. The negative predictive value was analyzed to see what percent of objects correctly classified as not an aircraft from the total number of objects the network not classified as not an aircraft. Accuracy was considered to analyze the total percent of classifications correct. Miss rate (false negative rate) was used to see the percent of time the network didn't think an object was an aircraft but it really was an aircraft and conversely, the fallout

(false positive rate, or "false alarm") was used to see the percent of time the network thought the object was an aircraft when it was not an aircraft.

The results of these performance statistics for each network can be seen below in *Table 5*. It should be noted that specificity was not included in the table below because DetectNet was the only neural network of the seven had false positives (Type I errors). These false positives can be identified when considering the fallout and precision results, making a specificity column unnecessary.

Model	Sensitivity	Precision	Negative Predictor Value	Accuracy	Miss Rate	Fallout
DetectNet	76.92	88.24	52.63	75.47	23.08	28.57
SSD	56.41	100.00	45.16	67.92	43.59	0.00
YOLOv2	71.79	100.00	56.00	79.25	28.21	0.00
Tiny YOLO	46.15	100.00	40.00	60.38	53.85	0.00
MobileNets 150	25.64	100.00	32.56	45.28	74.36	0.00
MobileNets 300	66.67	100.00	51.85	75.47	33.33	0.00
MobileNets 450	71.79	100.00	56.00	79.25	28.21	0.00

Table 5: Performance Statistics in Percents

A few interesting observations were noted but not fully understood. Out of the 25 images containing at least one aircraft, there were seven images in which all seven of the neural networks successfully identified the aircraft present in the image and there were four images that no network could detect an aircraft present. There seems to be no consistent feature that helped or hindered the networks in their detection of these 11 images.

Another observation was that four of the neural networks, DetectNet, Tiny YOLO, MobileNets 300 and MobileNets 450, detected two to three additional objects within the 30 images that were not listed as an object to detect or ignore. Part of the reason these objects were not considered was because the detection was made within another object, for instance, DetectNet detected the right rotor of a heliplane and an aircraft along with detecting the entire heliplane as an aircraft.

Model	FPS	Memory Usage (GB)	Power (mW)
DetectNet	9.60	2.34	10120
SSD	5.57	2.57	11555
YOLOv2	1.75	3.23	11994
Tiny YOLO	6.20	2.21	10508
MobileNets 150	4.15	2.73	6678
MobileNets 300	3.56	2.84	7292
MobileNets 450	3.02	3.04	9166
Idle CPU	n/a	1.16	3343

Table 6: Performance Metrics

To gain a better understanding of how each neural network performed under a continuous video stream, the team recorded frames per second (FPS), memory usage in GB, and power in mW. The performance metrics for each network can be seen in *Table 6 above*. Performance of each network was then analyzed

by looking at the throughput per Watt where throughput is defined as the number of frames per second processed by the network. These results can be seen in *Figure 11*.

As suspected, the resizing of an image affects the speed and accuracy of a neural network. For Google's MobileNets, smaller images, 150x150PX, processed more frames per second and consumed less power but the accuracy of these smaller images suffered as a result. The 450x450PX images were superior in terms of accuracy but the network was slower to process the data.

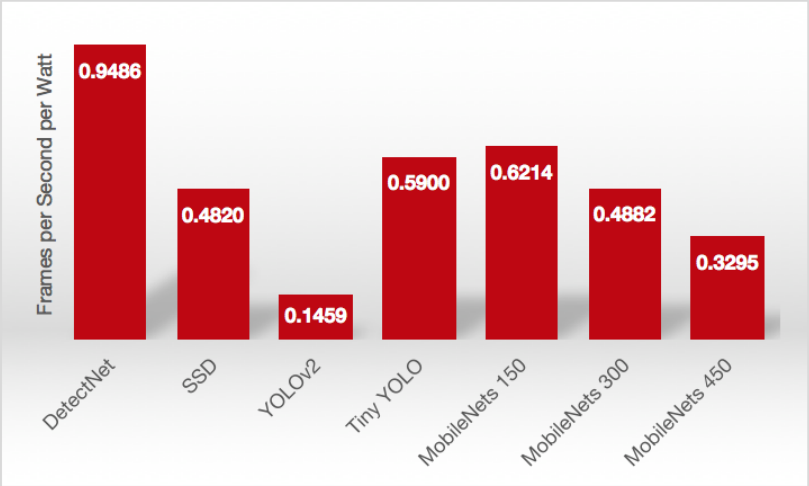

Figure 11: Image throughput per Watt

Each neural network had different strengths and weaknesses. YOLOv2 and Google MobileNets 450 had greater accuracy towards detecting aircraft and the latter consumed less power. However, neither network performed well for real-time video processing. DetectNet had the highest probability of detection, the lowest miss rate, and the best FPS performance, but it also had the greatest fallout. For this project, both speed and accuracy are essential to this real-time system to correctly classify objects and perform quickly. Comparing both performance and statistical metrics, Google's MobileNets 300, though not the best in any category, was fairly reliable in all areas.

9 Concluding Results and Future Work

The primary objective of the project was to appreciate what it is like working on a large project with a multidisciplinary engineering team. Through the development of a project plan and a design document, the team could begin working towards the end goal. Along the way, the team expanded their knowledge base on image processing, computer vision, machine learning, and neural networks.

From there, the team explored what research had already been done on the topics and what was already available that would facilitate incorporating computer vision and machine learning into the project. Using the background experience from team members, previously coded examples, advisor expertise, and tutorials, the team experimented with different neural networks to see how each one performed. Based on the performance of each neural network the team made a final decision on which one to use with the hardware suggested by the client.

Due to the time constraints imposed on the team members while working on this design, there are several ways to advance this study. One such way to improve results would be to address the challenges presented in Section 6. The team could have spent more time debugging the streaming errors with DetectNet, since DetectNet had promising results, and possibly considered using a different board that supports more network functions.

Another option would be to spend the time building a neural network from the ground up to train and apply constraints that specifically meet the needs of the project instead of using a network that has been pre-trained. As seen in Hans *et al.*, using different neural networks for different tasks could be implemented as well. If a neural network was implemented from scratch, the team could revisit their efforts with image processing and computer vision. For image processing, more time would need to be spent understanding how the feature detection and matching functions work along with investigating how changing parameters could improve results. For computer vision, the team could take a more in depth look at what OpenCV has available or explore other options.

Regarding testing, a better method of testing should be developed. More data in the form of images, both with and without planes, should be trained and tested on the neural networks. There should also be an attempt to remove data bias present from having only 16.7% of the test images not containing planes. Furthermore, the team should harness the retraining capabilities of the neural networks to produce better results overtime.

10 Bibliography

Listed below are works consulted for or referenced within this document that were used for the project.

Asaeedi, Saeed, et al. “α -Concave hull, a generalization of convex hull.” Theoretical Computer Science, vol. 1309.7829, 30 Sept. 2013, doi:10.1016/j.tcs.2017.08.014.

“BRIEF (Binary Robust Independent Elementary Features).” BRIEF (Binary Robust Independent Elementary Features) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html#brief.

"Cascade Classifier Training." Cascade Classifier Training — OpenCV 2.4.13.2 Documentation. OpenCV Dev Team, 16 Dec. 2016. Web. 31 Mar. 2017. <http://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html>.

"Computer Technology Standards." IEEE Standards Association. IEEE, 2017. Web. 31 Mar. 2017. <http://standards.ieee.org/cgi-bin/lp_index?status=active&%3Bpg=40&%3Btype=standard&%3Bcoll=15>.

Dehmamy, Nima, et al. “Convergence of Deep Neural Networks to a Hierarchical Covariance Matrix Decomposition.” Computing Research Repository, vol. 1703.04757, 14 Mar. 2017, arxiv.org/abs/1703.04757.

Deshpande, Adit. "A Beginner's Guide to Understanding Convolutional Neural Networks." A Beginner's Guide to Understanding Convolutional Neural Networks – Adit Deshpande – CS Undergrad at UCLA ('19). N.p., 20 July 2016. Web. 31 Mar. 2017. <https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>.

“FAST Algorithm for Corner Detection.” FAST Algorithm for Corner Detection — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html#fast.

“Feature Matching.” Feature Matching — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html#matcher.

Geitgey, Adam. "Machine Learning Is Fun! Part 4: Modern Face Recognition with Deep Learning." Medium. Medium Corporation, 24 July 2016. Web. 31 Mar. 2017. <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78#.7732t9p7a>.

Han, S. Shen, W. Liu, Z. *Deep Drone:* (2016) *Object Detection and Tracking for Smart Drones on Embedded System.* Stanford University. Retrieved from https://web.stanford.edu/class/cs231a/prev_projects_2016/deep-drone-object__2_.pdf.

“Harris Corner Detection.” Harris Corner Detection — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners.

IEEE Code of Conduct. IEEE, June 2014, m.ieee.org/about/ieee_code_of_conduct.pdf.

IEEE Code of Ethics. IEEE, www.ieee.org/about/corporate/governance/p7-8.html.

"Introduction to SIFT (Scale-Invariant Feature Transform)." Introduction to SIFT (Scale-Invariant Feature Transform) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro.

"Introduction to SURF (Speeded-Up Robust Features)." Introduction to SURF (Speeded-Up Robust Features) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html#surf.

"ISO - International Organization for Standardization." Information technology – Programming languages – C++, 30 Nov. 2017, www.iso.org/standard/64029.html.

Kappeler, Armin, et al. "Video Super-Resolution with Convolutional Neural Networks." IEEE Transactions on Computational Imaging, vol. 2, no. 2, 30 Mar. 2016, pp. 109–122., doi:10.1109/TCI.2016.2532323.

Kappeler, Armin, et al. "Super-Resolution of compressed videos using convolutional neural networks." 2016 IEEE International Conference on Image Processing (ICIP), 19 Aug. 2016, pp. 1150–1154., doi:10.1109/icip.2016.7532538.

Krizhevsky, Alex, et al. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM*, vol. 60, no. 6, 2017, pp. 84–90., doi:10.1145/3065386.

LeCun, Yann, et al. "Gradient-Based learning applied to document recognition." Proceedings of the IEEE, vol. 86, no. 11, Nov. 1998, pp. 2278–2324., doi: 10.1109/5.726791.

LeCun, Yann, et al. "The MNIST Database." MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, yann.lecun.com/exdb/mnist/.

Mallick, Satya. "Home." Learn OpenCV. Big Vision LLC, 14 Nov. 2016. Web. 31 Mar. 2017. <http://www.learnopencv.com/image-recognition-and-object-detection-part1/>.

Ng, Andrew. "Machine Learning." Machine Learning | The Open Academy. Open Academy, n.d. Web. 31 Mar. 2017. <http://theopenacademy.com/content/machine-learning>. Ng, Andrew. "Machine Learning." Machine Learning | The Open Academy. Open Academy, n.d. Web. 31 Mar. 2017. <http://theopenacademy.com/content/machine-learning>.

Nielsen, Michael A. "Neural Networks and Deep Learning." Neural Networks and Deep Learning. Determination Press, 01 Jan. 1970. Web. 31 Mar. 2017. <http://neuralnetworksanddeeplearning.com/>.

"ORB (Oriented FAST and Rotated BRIEF)." ORB (Oriented FAST and Rotated BRIEF) — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html#orb.

Qureshi, Shehrzad. "Computer Vision Acceleration Using GPUs." Computer Vision Acceleration Using GPUs (n.d.): n. pag. AMD Developer. AMD, June 2011. Web. 31 Mar. 2017. <http://developer.amd.com/wordpress/media/2013/06/2162_final.pdf>.

Shimodaira, Hiroshi. "Learning and Data Note." Multi-Layer Neural Networks (2015): n. pag. Web. <https://491dec1709.slack.com/files/daschott/F46274YA1/learning.zip>.

“Shi-Tomasi Corner Detector & Good Features to Track.” Shi-Tomasi Corner Detector & Good Features to Track — OpenCV 3.0.0-Dev documentation, OpenCV.org, docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi.

Simonyan, K., Zisserman, A. 2015, ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’ Paper presented to International Conference on Learning Representations, San Diego, CA, May 7-9, 2015. viewed 10-22-17, https://arxiv.org/pdf/1409.1556v6.pdf.

Smith, Steven W. "Nueral Networks (and More!)." The Scientist and Engineer's Guide to Digital Signal Processing. San Diego, CA: California Technical Publ., 1999. 451-80. Print.

Smolyanskiy, N. Kamenev, A. Smith, J. Birchfield, D. (2017) *Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness* Retrieved from https://arxiv.org/pdf/1705.02550.pdf.