A.L.V.I.N.N Design Document

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: April 20, 2017 / Version 2

IOWA STATE UNIVERSITY

Table of Contents

1 Introduction 3
1.1 Project Statement 3
1.2 Purpose 3
1.3 Goals 3
2 Deliverables 4
3 Design 5
3.1 System Specifications 5
3.1.1 Non-Functional Requirements 5
3.1.2 Functional Requirements 5
3.1.3 Standards and Ethics 5
3.2 Proposed Design and Methods 6
3.3 Design Analysis 7
3.3.1 Image Preprocessing 7
3.3.2 Computer Vision 8
3.3.3 Neural Networks 8
4 Testing and Development 11
4.1 Hardware and Software 11
4.2 Process 12
4.2.1 Image Preprocessing 12
4.2.2 Computer Vision 12
4.2.3 Neural Networks 13
5 Results 14
5.1. Image Preprocessing 14
5.2 Computer Vison 16
5.3 Neural Networks 16
6 Conclusion 17
7 References 18
8 Appendix 19

1 Introduction

1.1 Project Statement

This project is 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 will require 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 will consist of determining which machine learning algorithms are particularly suitable for the object recognition task.

Computer vision tools will be used to preprocess images from a picture or video stream, as well as extracting key features of objects. Machine learning will then be 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 will also have to investigate to what extent computer vision techniques could support the machine learning based detection algorithms.

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 of this project is to detect objects in the air. To accomplish this goal, the project will need to be broken into smaller, more manageable parts. First, the team will need to invest time in expanding their knowledge base on computer vision, machine learning, and neural networks. Upon acquisition of this knowledge, the next part will require the team to decide on which software to use and how to use it for object detection. To accomplish this goal, detection will begin with detecting a simple object (the letter 'X') on a plain background. The difficulty of detecting an object can then be increased by adding in background noise, similar objects, and permutations of the same object. These techniques used for detecting a simple object can then be applied towards detecting more complicated, airborne objects. The next immediate goal is teaching the system to detect these objects in the same manner. With these parts in place, the final phase will be implementing them together into a customized object detection pipeline.

2 Deliverables

To meet the goals outlined in the proposal, the following deliverables are 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 & 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 (CNN)
    - 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
    - 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

3 Design

3.1 System Specifications

Design, train, and deploy deep neural network based embedded board for use as a customizable Unmanned Aircraft System (UAS) mission processor. The project will use open-source frameworks and NVIDIA's latest GPU-accelerated deep learning platform to identify objects and fuse sensor data from one or more cameras and potentially other sensors, such as LIDAR.

Initially, use NVIDIA tutorials to implement some type of visual object recognition or similar task and use a mockup environment in the lab. Using flight simulator data, specific images will be fed into the system to build its object detection confidence. As confidence grows, Rockwell Collins can potentially provide some video footage from one of the drone flights this spring or summer. The specific objective of the visual system is to be able to detect and identify moving aircraft through means of computer vision and neural networks.

3.1.1 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 2 seconds.
- Scalability: The application must be able to run advanced mathematical calculations on an NVIDIA TX1 board with hardware constraints in memory and processing speed.
- 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 70% 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).

3.1.2 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.
- The image processor must be able to detect stationary objects apart from the background.
- The image processor must be able to detect moving objects apart from the background.
- The image processor must be able to detect multiple objects of the same or different type in a single image.

The image processor must be able to report confidence levels of any identified objects.

3.1.3 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. 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. The Python Package Index (PyPI) will be used to pin down any dependencies through a requirements.txt file. 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 standards for when the code is transferred to the client's system. The Code of Ethics and Code of Conduct standards of IEEE and ABET will be followed throughout this project.

3.2 Proposed Design and Methods

A very high-level overview of the system can be seen in *Figure 1* below. The system starts off by getting a video input feed from some source. For this system, it will be reading in the video data via a flight simulator. The visual feed is then processed via OpenCV and sent to the Caffe framework to preprocess the image via blurring and smoothing techniques. In post preprocessing, the information is sent to the Caffe machine learning framework where it will take the image and will use its trained neural network as well as its learning algorithms to make a prediction on what the image is. It will proceed to evaluate its own model to determine whether adjustments need to be made to the neural network model. This approach will allow the neural network to be constantly learning. Finally, the system will output to the user its prediction on what it believes the object in the image is as well as its confidence levels.

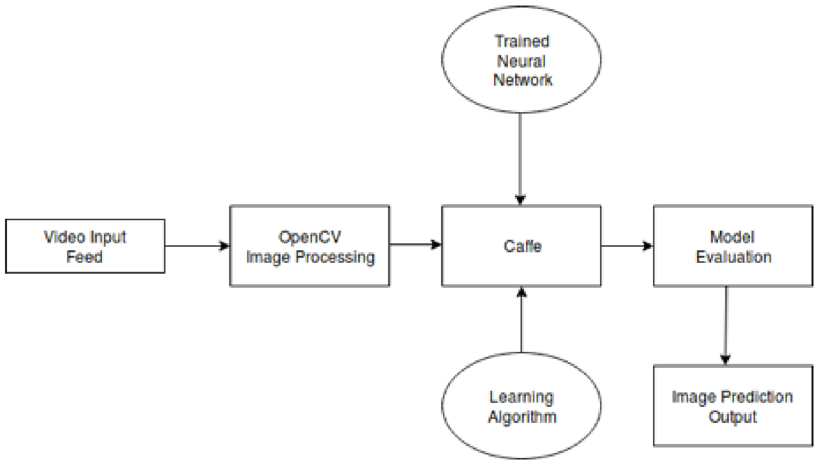


Figure 1: A.L.V.I.N.N Block Diagram

The design for the machine learning network part of the project is broken down into four phases: Preprocessing, Learning, Evaluation, and Prediction (see *Figure 2*). In the Preprocessing phase, raw images and streams of video data will be sent into the image processing unit for filtering and detection of features from which datasets can be created for training and testing. From the training datasets, the information is transferred to the second phase of the project. In this phase, the algorithm will take the incoming labeled data and use machine learning to train a model used to make predictions on the test data. With the model trained, the team can transition into the Evaluation phase where the model will be trained and evaluated using a set of test data. If the evaluation is satisfactory, then the team can move on to the final phase of prediction the recognized object. Using the neural network, the algorithm will then predict what it thinks it sees in new unseen data. This allows the system to inform the user about any found objects and their relative associated information.

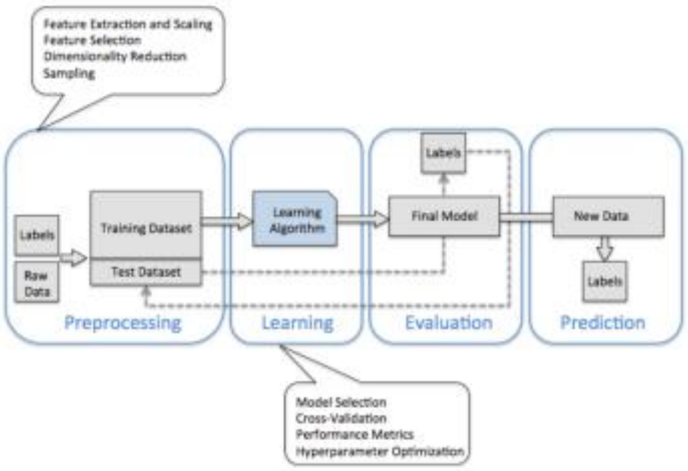


Figure 2: Python Machine Learning Process Diagram

3.3 Design Analysis

3.3.1 Image Preprocessing

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 speckles 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.

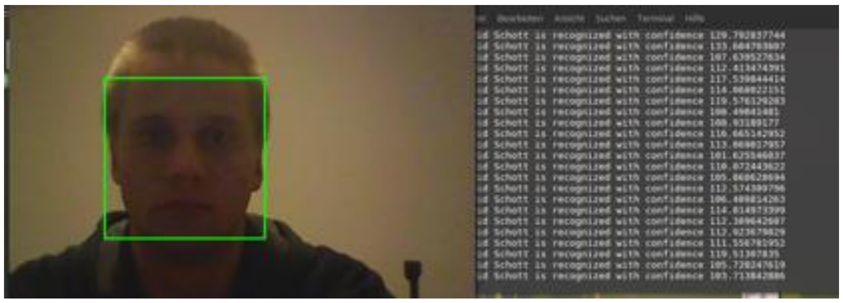
Once blurring and thresholding techniques were applied, an attempt was made at detecting features in an image. There are a few techniques available for detecting features that were considered including Harris Corner Detector, Shi-Thomasi Corner Detector, Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), FAST Algorithm for Corner Detection, Binary Robust Independent Elementary Features (BRIEF), and Oriented FAST and Rotated BRIEF (ORB). 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). 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.

The team has learned a lot about image processing and is pleased with the blurring and thresholding results. The team also has a lot more to learn about image processing and improvements need to be made with matching objects, which could potentially be related to the chosen feature detection method. Being new to this field, it is hard to know whether the results obtained are acceptable or not. Based on the team's knowledge and talking with the project's advisors and client, the team should be able to improve upon these results. More exploration is needed into understanding what each method does and how the parameters can be adjusted to obtain more accurate results.

3.3.2 Computer Vision

Regarding computer vision, basic object recognition use-case was explored using the OpenCV library. Experiments began by using pre-existing code to explore how to perform detection of faces using OpenCV's FaceRecognizer module. 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 3* shows an example of someone being recognized.



```
id Schott is recognized with confidence 129.792837744
id Schott is recognized with confidence 133.604783607
id Schott is recognized with confidence 107.639527634
id Schott is recognized with confidence 112.413474391
id Schott is recognized with confidence 117.539844414
id Schott is recognized with confidence 114.060022151
id Schott is recognized with confidence 119.576129283
id Schott is recognized with confidence 100.49043481
id Schott is recognized with confidence 100.93189177
id Schott is recognized with confidence 116.665142952
id Schott is recognized with confidence 113.060017957
id Schott is recognized with confidence 101.625546037
id Schott is recognized with confidence 110.072443622
id Schott is recognized with confidence 105.060628694
id Schott is recognized with confidence 112.574309796
id Schott is recognized with confidence 106.409814263
id Schott is recognized with confidence 114.014973399
id Schott is recognized with confidence 112.309642607
id Schott is recognized with confidence 112.923679829
id Schott is recognized with confidence 111.556701952
id Schott is recognized with confidence 119.51307835
id Schott is recognized with confidence 105.720247619
id Schott is recognized with confidence 103.713842806
```

Figure 3: Facial recognition of David Schott

The code and documentation of the command-line interface for adding new faces (taking screenshots using a webcam), training the recognizer, and launching it has been uploaded to the group's GitLab repository.

3.3.3 Neural Networks

With the application revolving around neural networks and computer vision, focus for the project was split between learning about neural network frameworks such as TensorFlow or Caffe, as well as learning how computer vision works with the OpenCV framework. So far, the team has practiced using neural network frameworks and OpenCV regarding handwriting analysis and facial recognition respectively. However, this consisted of purely experimenting with pre-existing code. The next steps in the project involve detection of images with cluttered background and being able to send OpenCV data to the neural network to perform object detection quickly.

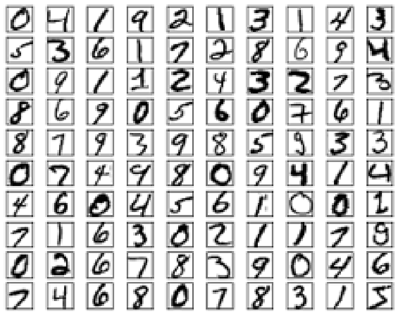


Figure 4: MNIST Sample Images

The first tests with neural networks was with the Caffe framework starting with a classic neural network vision problem which was classifying handwritten digits given by the MNIST dataset. See *Figure 4* for an example of the images used for handwritten digits. A typical naive approach for this specific problem that was used to model the neural network looked like what is depicted in *Figure 5*.

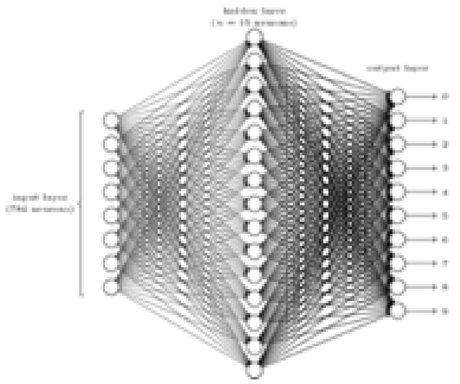


Figure 5: Naive MNIST Image Classifier

The neural network above depicts an input layer of 784 neurons where each neuron corresponds to a pixel in the images of handwritten digits (28×28 pixel images). Hidden layers essentially compress and generalize the information from the input neurons. Finally, each neuron in the output layer outputs the confidence that input image matches its class (numbers 0 through 9).

After training this network with the MNIST dataset, correct classification of the digits was identified 93% of the time. While this may sound impressive, it is not very reliable when comparing the results to other solutions that solve the same problem. This approach for processing images is naive for the following reasons:

1. Only very low resolution images can be evaluated (28×28 pixels).
2. It can only correctly classify digits that are in the same relative position as the training images (cannot classify digits in different locations).
3. There is trouble classifying rotated digits.
4. There is poor accuracy when compared to better solutions.

5. This type of model becomes infeasible when processing higher resolution images with millions of pixels.

After finishing the work with traditional feed-forward neural networks and understanding the limitations they suffer from, it was decided to explore Convolutional Neural Networks which specialize in image classification. Some of the major problems with the previous design was that as the images became more complex, the network size increases exponentially. Most importantly, when a feed forward neural network processes an image, it has no sense of spatial locality, it sees the image 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. Figure 6 shows an example of a simple CNN that also classifies MNIST handwritten digits. One type of CNN that could be explored is called a LeNet CNN. The basic process of this model can be seen in *Figure 6*.

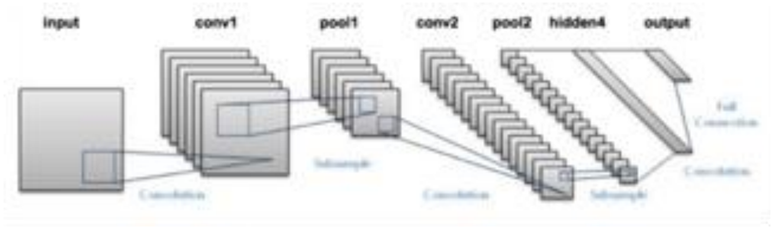


Figure 6: LeNet Convolutional Neural Network

The first step of LeNet is convoluting the input image with six different trained filters. These filters extract six different feature maps that are used in subsequent layers of the network. The input images are of size 32×32 pixels while the six filters are of size 5×5. Each kernel will pass over the entire image while convoluting subsections of the given image. *Figure 7* shows an example of a kernel with its trained weights over a specific section of the input image.

$$Kernel\,1 = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \end{bmatrix} \quad Subsection\ of\ image = \begin{bmatrix} 0 & 34 & 191 & 0 & 0 \\ 0 & 25 & 196 & 0 & 0 \\ 0 & 20 & 192 & 0 & 0 \\ 0 & 0 & 195 & 10 & 0 \\ 0 & 0 & 180 & 5 & 0 \end{bmatrix}$$

Figure 7: Kernel with Trained Weights

Convoluting these two matrices gives a result of 801 (a high positive result means the image section is like a straight vertical line). Passing this kernel over an entire image extracts the image's feature map for this specific kernel which looks for vertical lines. The resulting feature map seen in *Figure 8* is compressed into a smaller feature map using max pooling. These steps are repeated until the processed data is passed into a basic fully connected layer like the Caffe MNIST previously mentioned.

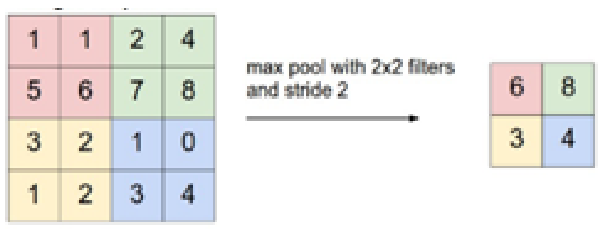


Figure 8: Max Pooling

For the fall semester, a Convolutional Neural Network (CNN) will be implemented because these networks modeled after parts of the visual cortex and are specialized to recognize complex images without increased processing time.

4 Testing and Development

4.1 Hardware and Software

In terms of hardware, the TX1 and TK1 NVIDIA 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. Furthermore, TensorFlow is not supported by the NVIDIA Jetson TX1 embedded board whereas Caffe is supported.

4.2 Process

4.2.1 Image Preprocessing

A basic 'X' was picked to begin testing methods for image preprocessing. The testing process for feature detection and matching will begin with an image of an 'X' containing no noise. The 'X' was made in Microsoft Paint. An image with an 'X' and some background noise (such as colored dots) added around it will also be used.

These images will be used to test blurring and thresholding methods resulting in a black and white image with smoother edges and less noise. Using these methods of blurring and thresholding, the noise should be eliminated or appear lighter in color than what is present in the original images. These methods should also smooth out any rough edges that are present in the original image.

Feature detection should be able to pull some features from the image and be consistent when duplicated with the same image. Matching is used to verify two identical images containing the same features. The same testing process will be followed to confirm success using the same image against an image that is geometrically similar but scaled differently (e.g. 'X' vs. 'x'). This testing will be extended to include trying to match an image (e.g. 'X') with another image containing multiple objects (e.g. 'A B C D …').

Once it has been concluded that matching the same image in multiple scenarios is successful, the next set of tests will begin. One of these tests will be taking similar images (i.e. 'X' vs '**X**') and seeing how they match across multiple scenarios. Another test will include matching similar images with other objects and noise added. For phase II to be complete, a final test of matching multiples of the same object in an image will be performed.

4.2.2 Computer Vision

This section covers testing the capability of the machine learning algorithms being used to predict an object's classification. Initial test will start off basic to verify the algorithms can produce some positive predictions. These initial test methods will compare the results obtained to results that have been published. Comparing to published results entails finding a predefined dataset that has been run on similar algorithms and comparing those results with those from the dataset being run on the algorithms. The holdout method may also be used. This method involves dividing a dataset into a training set and a test set. The model is trained, never seeing the test set until the model is ready to be evaluated. Then it can be determined how well the model did by determining if the prediction matches the test object's label.

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 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 being 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.

Although most the testing will focus on object recognition and prediction, testing can be extended to include the following:

1. Recognize activity: Using elementary machine learning approaches (i.e. training classifiers using labeled datasets containing objects at motion vs. objects at rest), additional capabilities can be added to the image processor such as identifying objects in motion. These can then be verified with typical machine learning model assessment methods such as True Positive Rate or False Positive Rate.
2. Distance of the object from the camera: This is a tricky problem that needs to be approached carefully. One would need to manually measure the actual distance and compare it to the estimated distance supplied by the computer-vision algorithms applied to pictures of objects of a certain (known) distance away.
3. Multiple Object Recognition: Upon successfully identifying different types of flying objects, one way of evaluating performance of the models and identifying problematic regions is through the generation of a confusion matrix. The confusion matrix will help in the understanding the accuracy and misclassification rates of the algorithms being used.

4.2.3 Neural Networks

This section covers final testing with the model being deployed on a neural network. For the variety of tasks that the application should be able to perform, there will be a series of test suites to be able to validate whether models adhere to the client's specifications. Along with the test mentioned in Section 3.4.2, the following is a listing of a few validation tests to be used for the application:

- Provide the application with a series of images of aircraft/other objects with a cluttered background and test whether the system can reliably detect the target image. A low error rate for many images is the goal along with a high confidence level for each classified object in an image.
- Provide the application with a video of an object moving with a cluttered background behind it and test whether the system can reliably detect the moving object.
- Automated cross-validation techniques with labelled images to obtain an averaged model prediction accuracy.

For the above validation tests, the success of the tests will be based on how consistently accurate the application can detect an object (either moving or stationary) through a large amount of trials (preferably in an automated way).

Additional software that may be used for testing and demonstration purposes include a flight simulator and a GUI. A flight simulator will allow a controlled stream of data to enter the system. A GUI could be used to display system results such as identification of detected objects or prediction accuracy. Additional hardware for this project would be in the form of a camera. A camera could be used to import either still pictures or to stream video into the system. However, with the numerous photos available on the internet and the possible use of a flight simulator, an actual camera may not need to be incorporated into the project.

5 Results

5.1. Image Preprocessing

The results from image preprocessing a simple 'X' have been encouraging. Figure 9 shows an original image in red after it has been converted to grayscale and displayed. The 'X' has rough edges and noise surrounding it. A smoothing, or blurring, process was implemented to help smooth the edges and eliminate the noise.

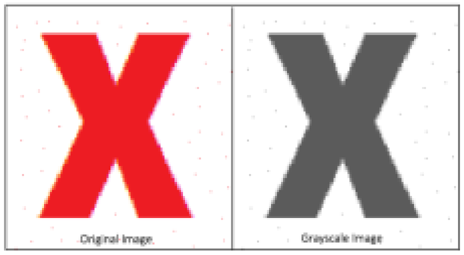


Figure 9: Original and Grayscale Images

The blurring methods explored 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 10*. 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. For a better comparison, two thresholding techniques were applied to each blurred image: a basic thresholding technique and an adaptive thresholding technique.

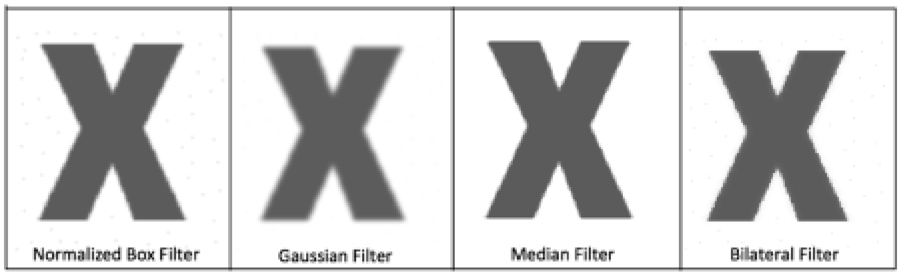


Figure 10: The four different filters used on the grayscale 'X'.

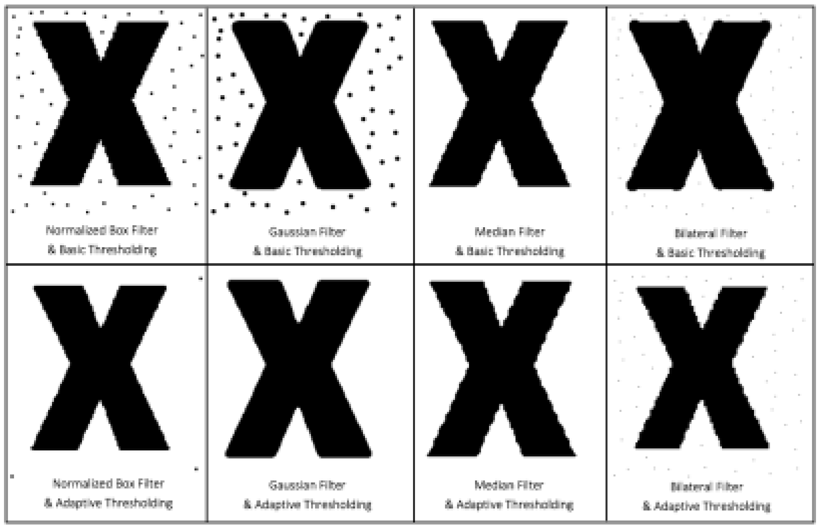

Normalized Box Filter & Basic Thresholding
Gaussian Filter & Basic Thresholding
Median Filter & Basic Thresholding
Bilateral Filter & Basic Thresholding
Normalized Box Filter & Adaptive Thresholding
Gaussian Filter & Adaptive Thresholding
Median Filter & Adaptive Thresholding
Bilateral Filter & Adaptive Thresholding

Figure 11: All four filter techniques used in conjunction with a basic and an adaptive thresholding technique using a high threshold value.

A basic and an adaptive thresholding technique was applied to each image with the results shown in *Figure 11* above. 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 initially thought. However, adjusting the threshold value can take care of this in some cases, as seen in *Figure 12*. 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.

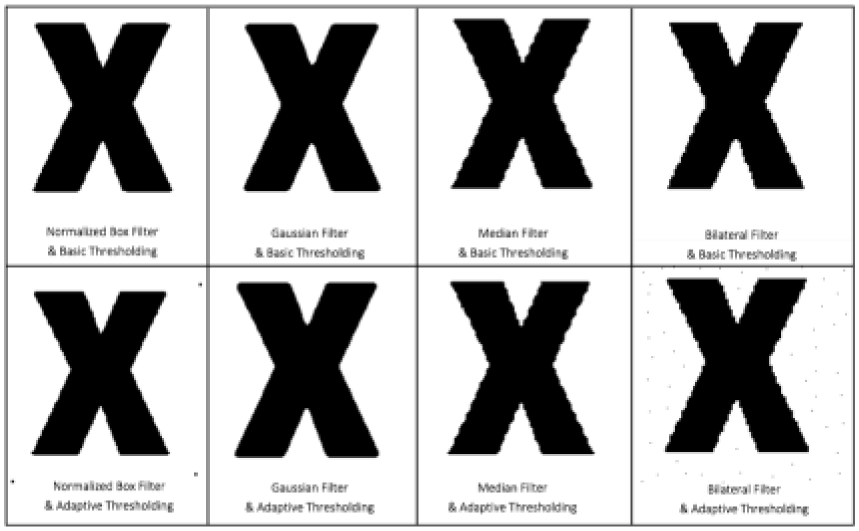

Normalized Box Filter & Basic Thresholding
Gaussian Filter & Basic Thresholding
Median Filter & Basic Thresholding
Bilateral Filter & Basic Thresholding
Normalized Box Filter & Adaptive Thresholding
Gaussian Filter & Adaptive Thresholding
Median Filter & Adaptive Thresholding
Bilateral Filter & Adaptive Thresholding

Figure 12: Basic and adaptive thresholding using lower threshold values.

Feature detection began with the original basic 'X' that was used for image processing. Feature detection identifies key points in the image that are used to recognize the same type of key features in other images when finding potential matches (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 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 points to be significantly reduced. The key features or key points of both the original 'X' with blurring and thresholding techniques applied and the smooth 'X' can been seen in green in *Figure 13* below.

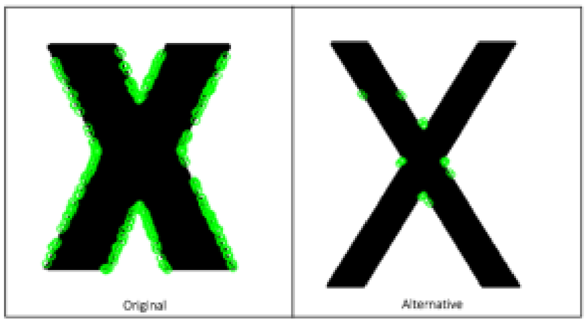


Figure 13: Feature Detection with the original 'X' after blurring and thresholding and with smooth-edged 'X'. Key features or key points are shown in green.

The feature matching tests that have been performed so far have had unsuccessful results. The first issue is matching key points on the 'X' with similar key points on other object types. This can be seen in *Figure*

14 with the lines going from the 'X' to the 'B' and the 'n'. The second issue is matching the wrong points on the correct object. *Figure 14* shows this issue with two 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 is the issue of not matching key points between geometrically similar objects. The small 'x' in *Figure 14* shows no matching points in common with the comparison 'X'. More time needs to be spent understanding how the feature detection and matching functions work and more time needs to be spent investigating how changing parameters could improve results. The team also needs to explore other options for feature detection that may provide better, more reliable results.

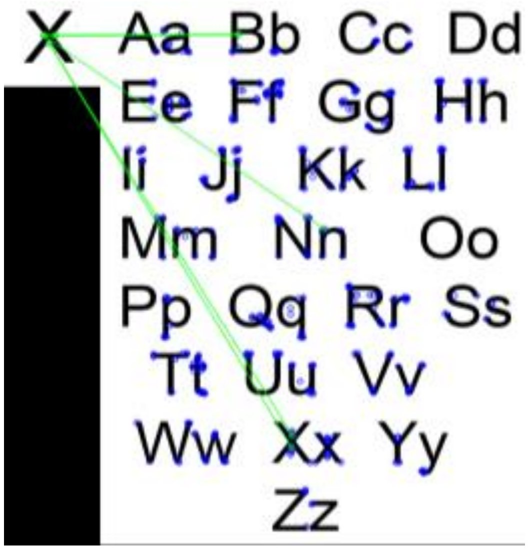


Figure 14: Feature Matching

5.2 Computer Vison

As mentioned in Section 3.3.2, there have been promising results with performing detection of faces using OpenCV's FaceRecognizer module. More investigation is needed before the team can determine how this technology might help with the project.

5.3 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 3.3.3 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 a continuous video footage.

With these results, it was decided that a more sophisticated neural network model should be adopted to combat these issues. For the fall semester, a Convolutional Neural Network (CNN) will be used as these networks are modeled after parts of the visual cortex and are specialized to recognize complex images without increased processing time per image. These networks are much more complex than a traditional neural network which may present further complications in the final design. Fortunately, there are several detailed research papers and implementations of CNNs available to study. These papers should prove to be useful as the project moves forward and if more problems are encountered.

6 Conclusion

The primary objective of the project is to appreciate what it is like to work on a larger project with a multidisciplinary engineering team. Considering that estimation work (at best) is required for a visual object recognition task, this project is being approached from a research perspective. Thus, to optimally achieve the goals set forth for the project, the team will be in close contact with the advisors and the client to evaluate what steps will likely to lead to success. The goal is to complete phases I and II before the summer ends and then use the remaining time in the fall semester to tackle the remaining phases.

The team began the project by expanding their knowledge base on computer vision, machine learning, and neural networks. From there, the team explored what software 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 has been able to work with the different software to see how it performed to make a final decision on what software to use to use with the hardware the client suggested. With this decision, out of the way, work began on the design process to reach the goal of detecting airborne objects. This process consists of preprocessing images, machine learning through training, evaluating and evolving the model, and finally prediction.

By utilizing OpenCV for image processing and the Caffe framework for machine learning, work can begin on building a model that will be able to train, to test, and then to use to predict various objects. OpenCV and Caffe have been used in numerous computer vision projects like the goal of the project, thus there is access to a plethora of information. Likewise, NVIDIA provides interfaces for the chosen hardware, OpenCV, and Caffe. Using these proven technologies will allow more time to concentrate on adapting the technologies to the project goals instead of spending time figuring out how to get the various technologies to communicate with each other.

7 References

Listed below are references that have been used so far as part of the research and testing phases.

"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>.

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/>.

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>.

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/>.

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/>.

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>.

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>.

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.

8 Appendix

PowerPoints:

- The “marketing pitch” for the project.
  - ALVINN_Pitch
- The first approach at what is believed to be the project’s component structure
  - ALVINN_Component_Structure

TX1 and TK1 Boards (Specs & Info):

- http://www.nvidia.com/object/jetson-tx1-module.html
- http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

Previous Literature & Research:

- Asaeedi, Saeed, Farzad Didehvar, and Ali Mohades. *Alpha - Concave Hull, a Generalization of Convex Hull*. Department of Mathematics & Computer Science, Amirkabir University of Technology, n.d. Web. 13 Apr. 2017.
  - https://arxiv.org/ftp/arxiv/papers/1309/1309.7829.pdf
  - Topological and geometric algorithms regarding alpha shapes can be used for image detection as described in the paper. This image handling procedure using the alpha convex hull can prove to be useful for detecting objects in images.

- Lecunn, Y., Bottou, L., Bengio, Y., Haffner, P. (1998) Gradient Based Learning Applied to Document Recognition
  - http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
  - One of the first papers demonstrating the idea of CNN. At the time, computers lacked the computation needed to train these networks. As a result, they did not become popular until around a decade later.

- Krizhevsky, A., Sutskever, I., Hinton, G. E., (2012) ImageNet Classification with Deep Convolutional Neural Networks
  - http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
  - Recent paper on image classification using CNNs. This was the first CNN to win the ImageNet Large-Scale Visual Recognition Challenge.

- Nielsen, Michael. Neural Networks and Deep Learning
  - http://neuralnetworksanddeeplearning.com
  - This source is more of a tutorial than actual research. This is an online book explaining neural networks and solving a common classification problem.

- Simonyan, K., Zisserman, A. (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition
  - https://arxiv.org/pdf/1409.1556v6.pdf
  - This deep network, known as VGG, was designed such that it was simpler and deeper than previous models. This model also changed some of the hyperparameters specific to CNN’s such as the kernel size and stride. VGG altered these parameters so that more information

was processed for each image. The downside to this approach is training becomes much slower.

Machine Learning:

- http://diydrones.com/profiles/blogs/using-tensorflow-to-allow-a-robot-to-identify-objects
- http://theopenacademy.com/content/machine-learning
- https://491dec1709.slack.com/files/daschott/F46274YA1/learning.zip
- Good lightweight article to read for not treating machine learning like magic:
    - http://www.pyimagesearch.com/2014/06/09/get-deep-learning-bandwagon-get-perspective/

Neural Networks:

- Chapter 26 of the free online book. Good overview of using the simplest of Neural Networks. One can download a pdf of each chapter.
    - http://www.dspguide.com/pdfbook.htm

- Online book covering the basics of neural networks with a simple computer vision problem.
    - http://neuralnetworksanddeeplearning.com

- A very basic overview of the concept behind Convolutional Neural Networks and why they are used to classify images.
    - https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks

Computer Vision:

- Slide 2 and 3 of the attached PDF gives a high-level overview of that Traditional Computer Vision pipe-line that is heavy on the Feature Extraction stage and light on the preprocessing and Classification stage.
    - https://491dec1709.slack.com/files/tlavan/F49KXLFH8/rcl_021317_murad.pdf
- Somewhat light article that shows a nice example of combining "Traditional" with Deep Learning Computer vision techies. (cool demo near the end: using the Jimmy Fallon show as an input))
    - https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78#.7732t9p7a
- Slide 9: gives a nice overview of a generic classical Computer Vision pipeline
    - http://developer.amd.com/wordpress/media/2013/06/2162_final.pdf
- Overview of traditional approaches for Computer Vision:
    - http://www.learnopencv.com/image-recognition-and-object-detection-part1/
- Near the end of this slide set, gives the topics covered which does a reasonable job of covering topics useful for doing computer vision.
    - http://www.cs.cmu.edu/~16385/spring15/lectures/Lecture1.pdf

Technologies:

- TensorFlow Video
    - https://www.youtube.com/watch?v=APmF6qE3Vjc
- Intro to Matlab
    - https://491dec1709.slack.com/files/tlavan/F4663EHSR/ee225_matlabintro_2015.pdf
- OpenCV
    - http://www.learnopencv.com/