The Use of Low-Cost Sensors and a Convolutional Neural Network to Detect and Classify Mini-Drones

Austin Florio

The University of New Haven

THE USE OF LOW-COST SENSORS AND A CONVOLUTIONAL NEURAL NETWORK TO DETECT AND CLASSIFY MINI-DRONES

A THESIS

Submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

BY

Austin Florio

University of New Haven

West Haven, Connecticut

August 2020

THE USE OF LOW-COST SENSORS AND A CONVOLUTIONAL NEURAL NETWORK TO DETECT AND CLASSIFY MINI-DRONES

APPROVED BY

[illegible]

Eric A. Dieckman, Ph.D.
Committee Chair

[illegible]

Cheryl Li., Ph.D.
Committee Member

Gokhan Egilmez

Gokhan Egilmez, Ph.D.
Committee Member

[illegible]

Cheryl Li, Ph.D.
Program Coordinator

RSHarichandran

Ronald S. Harichandran, Ph.D., P.E., F.ASCE
Dean of the Tagliatela College of Engineering

[illegible]

Mario Thomas Gaboury, J.D., Ph.D
Interim Provost

ACKNOWLEDGEMENTS

I would like to acknowledge everyone who has helped with the completion of this thesis.

First, this research would not have been possible to complete without my advisor, Dr. Eric Dieckman, for his guidance and continuous support throughout this project. His expertise was invaluable, and he has imparted a depth of knowledge to me that I will carry with me throughout my life and future career.

Secondly, I am indebted to Dr. Cheryl Li for providing me with a strong foundation in the background knowledge of both mechatronics and programming throughout my mechatronics concentration.

Thirdly, I am extremely thankful to my grandmother, Phyllis Cardullo, for proofreading and recommending the grammar revisions that would have the best impact for the reader.

Lastly, I would like to thank my friends and family for their constant and loyal support.

Abstract

The increasing commercial availability of mini-drones and quadrotors has led to their greater usage, highlighting the need for detection and classification systems to ensure safe operation. Instances of drones causing serious complications since 2019 alone include shutting down airports [1-2], spying on individuals [3-4], and smuggling drugs and prohibited items across borders and into prisons [5-6]. Some regulatory measures have been taken, such as registration of drones above a specific size and the establishment of no-fly zones in sensitive areas such as airports, military bases, and national parks. While commercial systems exist to detect drones [7-8], they are expensive, unreliable, and often rely on a single sensor. This thesis will explore the practicality of using low-cost, Commercial-off-the-shelf (COTS) sensors and machine learning to detect and classify drones.

A Red, Green, and Blue (RGB) USB camera [9], FLIR Lepton 3.0 thermal camera [10], miniDSP UMA-16 acoustic microphone array [11], and a Garmin LIDAR [12] were mounted on a robotic sensor platform and integrated using a Minisforum Z83-F with 4GB RAM and Intel Atom x5-Z8350 CPU to collect data from drones operating in unstructured, outdoor, and real-world environments. Approximately 1,000 unique measurements were taken from three mini-drones – Parrot Swing, Parrot Quadcopter, and Tello Quadcopter – using the RGB, thermal, and acoustic sensors. Deep Convolutional Neural Network (CNNs), based on Resnet-50 [13-14], trained to classify the drones, achieved accuracies of 96.6% using the RGB images, 82.9% using the thermal images, and 71.3% using the passive acoustic microphone array.

Table of Contents

List of Tables

List of Figures

Chapter 1. Existing Approaches to Detect and Classify Mini-Drones

The increasing availability of low-cost, easy-to-fly drones and quadcopters have led to their use in criminal activities including disrupting air travel, spying on neighbors, and smuggling drugs and contraband. In 2019, a drone was spotted in the airspace of Newark airport, causing the temporary suspension of all flights and leaving dozens of aircraft circling the airport [1]. Serious invasions of privacy include instances of voyeurism that can happen to anyone who is not aware that drones are in the area [4]. Drones were intercepted attempting to smuggle $306,000 worth of drugs across the United States' border from Mexico [5] and illicit materials into the Fort Dix prison over at least seven separate incidents [6].

There is a clear need to prevent such incidents from occurring. Drone detection and classification applications have been gaining in popularity over the years, and a number of systems have been developed to detect and classify drones. These include systems based on passive monitoring of Radio Frequency (RF) communications, active radar, optical sensors in the visible and infrared spectrum, passive and active acoustic sensors, and active LiDAR.

Passive radio frequency-based detection of drones has successfully been proven both cost-effective and feasible throughout the years. One example this research involved detecting and classifying Unmanned Aerial Vehicles (UAVs) using a multistage detector system to distinguish the signals from the UAV controller from both background noise and interference signals. The first stage of this research has a Markov models-based naïve Bayes decision mechanism to detect any RF signals that obtained a detection accuracy of 99.8% with a false alarm of 2.8%. The second stage detects whether there are any signals from WI-FI and Bluetooth emitters through the bandwidth and modulation features of the acquired RF signal. Once the UAV controller signal is detected, the signal's three most significant features are determined through the neighborhood component analysis and is then inputted into five different machine learning techniques, obtaining a classification accuracy of 98.13% through the k-nearest neighborhood classifier [15]. Another research involves RF-based low-signal-to-noise UAV classification using convolutional neural networks. This research uses fifteen off-the-shelf drone RF signals to obtain RF time-series images and spectrograms for the training of the convolutional neural network. The spectrogram drastically outperformed the time-series images when the Signal to Noise Ratio (SNR) was reduce. The overall classification accuracy of the spectrogram-based CNN varied from 92% to 100% for a signal-to-noise ratio range of -10 dB to 30 dB [16]. An example of another research using cost-effective RF-based detection of drones includes exploring the areas of active tracking and passive listening. These approaches were validated and could observe that the drone's propellers emitted frequency of less than 100 Hz. [17]. Unfortunately, this radio frequency-based detection relies heavily on communication between the drone and operator, which would not be required for future drones in implementing artificial intelligence (AI) systems.

The standard active radar system has difficulty detecting and classifying mini-drones due to the drones' small radar cross section and resemblance to birds, which are of similar physical size flying at equivalent altitude and speed. However, recent studies have shown positive results incorporating micro-Doppler effects, which are frequency modulations on the return signal caused by the target's mechanical vibration or rotation [18]. The components on a small consumer drone that are non-plastic, such as the battery, motors, and camera, have a significant return in radar signature compared to the plastic materials, such as propellers [19]. A further study collected data on different birds and drones using K-band and W-bands, and by incorporating the micro-Doppler effect, the study showed a significant difference between their signatures [20].

Another type of approach is to use optical sensor that are usually incorporated with a machine learning technique. One research integrated a static wide-angle camera and a lower-angle camera mounted on a rotating turret with YOLOv3 architecture to autonomously detect and track drones. The overall system has the static wide-angle camera mounted on a stationary platform that is able to adjust the angle depending on the demand, a narrow-angle camera, with zoom capability, mounted on a rotating turret, and the connected to the main computational device through ethernet with the YOLO architecture. These cameras are the same model RGB high performance industrial cameras, except for the professional zooming capability on the narrow-angled camera. The static wide-angle camera's output frame was overlaid with the zoomed camera's output frame to use memory and time efficiently. This system was compared with the two conventional object detection approaches: the Haar classifier with Adaboost algorithm, and the Gaussian mixture model background subtraction algorithm. These object detection approaches were used on 20 videos containing 800 frames of complex background and containing various objects such as different birds and planes. The results showed that the YOLO system had a 91% true positive, which is 7% lower than the highest detection approach, and this model had no false alarms, unlike the other approaches [21]. The results of this experiment can be seen in Table 1.

Table 1: Results of Deep Learning-Based Strategies for Detection and Tracking of Drones Using Several Cameras. The results of the RGB camera system involving the static camera and rotating zoom camera are shown below with the different types object detection techniques.

	True Positive	**False Alarm**
Lightweight YOLO	0.91	0
Cascaded Haar	0.95	0.42
Gaussian Mixture Model Back. Sub.	0.98	0.31

A different optical research integrates thirty HD cameras and thirty microphones into an array to detect and classify aircrafts using YOLO and CNN. The dataset includes multiple fixed-wing aircrafts, helicopters, and consumer drones. The fixed-wing aircrafts, helicopters, and some of the images of the drones were from the FGVC-aircraft dataset, while the drone dataset is also extracted from the captured camera array. The test images included 300 drone images, 100 helicopter images, and 100 fixed-wing aircrafts. The classification results show that the "Aeroplane" class obtained a 96.03% accuracy, the "Helicopter" class received a 90.47% accuracy, and the "Drone" class obtained a 52.13% accuracy. The reasoning for the low percentage in drone accuracy is due to the complexity of the backgrounds and the need for more images to be collected [22].

A very popular use of drone classification methods is the use of acoustic sensors. One research incorporated acoustic sensors to be used alongside feature extraction and Support Vector Machines (SVMs) to classify UAVs at distances of up to 50 m. based on vehicle noise [23]. A similar project using feature extraction in both the time and frequency domains to deal with noisy environments resulted in classification accuracy above 96% [24]. Another drone detection research explored incorporating low-cost hardware components, comprised of two different arrays of three or six microphones, to identify and classify drones using nearest neighbor rule. These arrays consist of low-cost omnidirectional miniature microphones, the processor STM32F405RG, and the drones that are included in the project are the Quadcopter DJI P3, Quadcopter CX 10, and the Sennheiser MKH 8040. Advanced array processing methods are utilized to obtain the normalized Power Spectral Density that is unique to each of the drones. The nearest neighbor rule estimates the closest similarity to each drone using the normalized spectrum over the frequency and time and the library data stored on the flash memory of the microcontroller. The use of this detection and classification system "has yet to fail" in a noise free environment for the preliminary experiments; however, this system still needs to be tested in realistic environments [25].

A very well detailed acoustic research that is relatable to this thesis is the creation of an audio pattern recognition system capable of detecting the number of DJI phantoms on scene with Convolutional Neural Networks. The equipment involved in the experiment are the two DJI phantoms standard 3's and the Sony ECM-DS70p-portable stereo. With the Sony EXM-DS70p, the audio samples collected were used to create the raw spectrograms, log-Mel-spectrograms, harmonic-percussive source separation and raw audio waveforms. Using both custom and augmented datasets, the experiments performed are: PCA and TSNE visualization of the SMILE988 features, Random Forest Algorithm applied to the SMILE988 features, Deep Neural Network with SMILE988 features, Deep Neural Network with SMILE988 reduced features, Convolutional Neural Network with 3-channel spectrograms, and Convolutional Neural Network with 2-channel Spectrograms with Harmonic and Percussive content into individual channels. An additional two experiments are included in the custom collected dataset, which are:

Convolutional Neural Network with Raw audio waveforms, and Generative Adversarial Networks for Data Augmentation. The convolutional neural network's average accuracy performed better with the augmented dataset over the custom collected dataset, as shown in Table 2. The main difference between this acoustic research and this thesis is that the acoustic research used multiple methods to classify audio samples to the classes: one drone, two drones, or background noise; while this thesis explores the classification of each individual drone in different settings [26].

Table 2: Results of Multiple Drone Detection and Acoustic Scene Classification with Deep Learning. The acoustic research performed different experiments using custom and augmented datasets. The table shows the average classification percentage obtained for the dataset and experiment.

Average Classification Accuracy (%)	**Custom Dataset**	**Augmented Dataset**
Random Forest Algorithm with SMILE988 Features	73.3	63.3
DNN with SMILE988 Features	84.2	76
DNN with SMILE200 Features	91.3	69
CNN Raw Spectrograms	66.3	90.3
CNN Log Spectrograms	57.3	91
CNN with Mel-Spectrograms with 128 Mels	68	73.6
CNN with Log-Mel Spectrograms with 40 Mels	72	85.6
CNN with Log-Mel Spectrograms with 60 Mels	73.3	87
CNN with Log-Mel Spectrograms with 80 Mels	66.3	87
CNN with Log-Mel Spectrograms with 128 Mels	72.7	85.3
CNN with Log-Mel Spectrograms with 200 Mels	73.7	87
Harmonic Percussive Source Separation	79	81
CNN with Raw Audio Files	70.6	N/A

Light Detection and Ranging (LiDAR) and Laser Detection and Ranging (LADAR) sensors, one of the newest techniques to detect and classify drones, measures the distance between the sensor and target by emitting a light and having it reflect off the surface back to the sensor. One example of research conducted is the fusion of a 3D LiDAR sensor integrated with a pair of cameras that is used for object detection and classification in maritime environments. The LiDAR initiates the object detection and classification by obtaining the spatially distinct features, and then the global LiDAR frame is converted to the camera frame, which allows the camera to extract the color-based features in the region. Both the Support Vector Machine (SVM) and Multi-Variant Gaussian (MVG) classifiers had amazing classification accuracies detecting objects, such as a specific tower, dock, and different buoys [27]. A research was performed to develop a new 3D LADAR to detect small drones up to 2 km. using the Variable Radially Bounded Nearest Neighbor (V-RBNN) method. The V-RBNN was proven to be much more reliable when compared to the conventional Radially Bounded Nearest Neighbor (RBNN) clustering method, which had difficulty due to the variation of the drone's shape and size at different distances. This experiment was based only on augmented datasets and future work for this research would include adding data for birds, as well as acquiring real sensor data [28]. Another research was performed to expand the tracking, detection, and classification of low flying objects, such as mini-UAVs in real-time using LiDAR. The UAV's have typical movement patterns that can be analyzed, allowing a precise prediction of the movement and UAV classification. Experimental data using the LiDAR was collected in the field with several different mini UAVs, using four 360° LiDAR sensors mounted to a car. This system allowed the car to be protected from the UAV threats withing the radius of 35 m. [29].

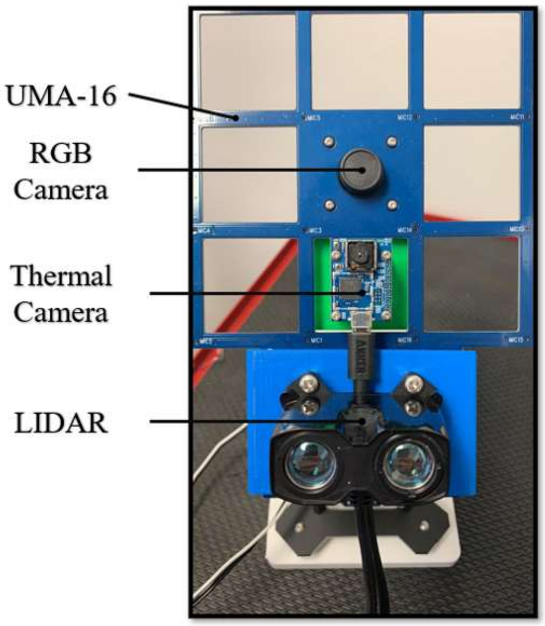
Combinations of sensors have also been studied to improve classification over a range of environmental conditions. A very similar research to this thesis is the integration of the RGB images, thermal images, acoustic data, and transmitters and receivers to detect, classify, and track drones using convolutional neural networks. This research incorporated a FLIR Breach PTQ136 thermal sensor, Sony HDR-CX405 video camera, a Boya BY-MM1 acoustic sensor, and a

NooElec Nano 2+ Software Defined Radio receiver and G-STAR IV BU-353S4 GPS receiver to track active ADS-B transponders on airborne devices. Data collected from the Hubsan H107D+, DJI Phantom 4 Pro, and DJI Flame Wheel F450 drones was evaluated using convolutional neural networks to compare performance from sensor fusion to each individual sensor. The fused data classified the drone correctly for 78% of the detection opportunities, with a more robust system. The equipment cost in this research greatly exceeded the cost of equipment in this thesis, primarily because the purpose of this thesis was to explore the performance of low-cost sensors [30].

Chapter 2. Building a Robotic Sensor Platform

Sensors

Using a variety of sensor types is necessary to create a robust system that can work in a variety of environmental conditions. The sensors integrated into this project included optical (ELP 2.0-megapixel USB camera, $65), thermal (Lepton FLIR 3.0, $240), acoustic (miniDSP UMA-16 microphone array, $275), and LiDAR (LiDAR-Lite v3HP, $150) sensors, as shown in Figure 1. Years ago, these sensors were extremely expensive, but affordable low-cost sensors are now easily obtainable in the marketplace. The cost of whole drone detection systems cost thousands of dollars, which prevents these systems from being widespread. The purpose of this project is to create a system that gives comparable or reasonable performance at a much lower cost. By the end of this project, the outcome will quantify the limitations of each sensor modality to steer future work in optimization of sensors and machine learning processing to create a low-cost drone detection system.



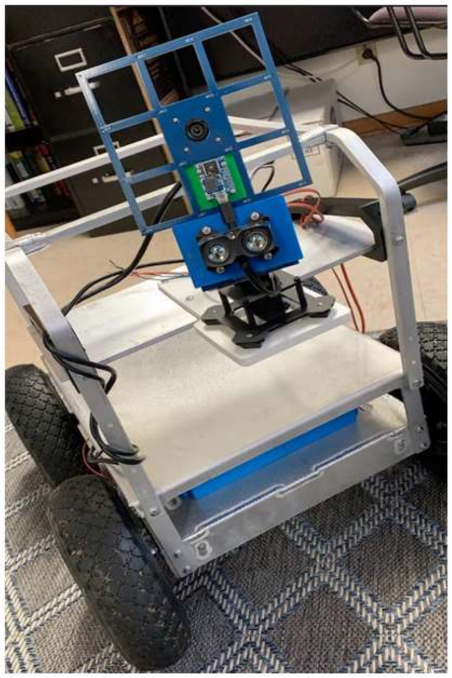


Figure 1: Sensors Diagram. The diagram of the sensors, on the left, shows the UMA-16, RGB camera, thermal camera, and LiDAR sensors. These sensors are attached to a pan and tilt mount that provides 180° to both the horizontal and vertical direction. The right image shows the front profile of the robot with the sensors.

The ELP 2.0 megapixel USB camera provides a max resolution of 1920x1080 at 50 FPS, with a 70° Field of View (FOV) [10]. The minimum illumination for this camera is 0.1 lux, with signal to noise ratio of 40 dB and of 65 dB of dynamic range. The overall dimensions are 38 x 38 x 25mm and the camera requires 5 Vdc. Additional specifications are given in Table B-1.

The Lepton FLIR 3.0 is an enhanced infrared sensor with 160 x 120 active pixels and thermal sensitivity of less than 50 mK [9]. The f/1.1 lens provides a horizontal FOV of 57° and diagonal FOV of 71°. The output allows 14-bit video over SPI, 8-bit with Automatic Gain Control (AGC) applied, or 24-bit with AGC and colorization applied. This thermal sensor has a low operating power of 140 mW (typ), 650 mW during shutter event, and 5mW during standby. The overall dimensions are 11.8 x 12.7 x 7.2 mm. Unfortunately, this model does not have the radiometry temperature feature, which allows temperature scales in the thermal images. Additional specifications are given in Table B-2.

The miniDSP UMA-16 is a sixteen-channel rectangular microphone array using Knowles SPH1668LM4H MEMS capsule microphones laid out in a Uniform Rectangular Array (URA) [11,32]. The acoustic array contains a nanoSharc kit that has a 400MHz SHARC ADSP21489 +500MHZ multicore CPU that provides significant processing power for high SNR PDM to PCM conversions and multichannel low latency USB audio. This sensor has 24-bit resolution and a sampling rate of up to 48kHz. Additional specifications are given in Table B-3.

Lastly, the LiDAR-Lite v3HP has a range of 5 cm to 40 m. with a resolution of +/- 1.0 cm [12,31]. The typical accuracy is +/- 2.5 cm at distances greater than 2 m, and +/- 5 cm at distances less than 2 m, indicating that this sensor performs better at distances greater than 2 m. The LiDAR has a greater than 1 kHz update rate and an optical aperture of 12.5 mm. This sensor has a nominal wavelength of 905 nm, 1.3 W peak laser power, beam diameter of 12x2 mm, and beam divergence of 8 mRadian. The device communicates through Inter-Integrated Circuit (I2C) and Pulse Width Modulation (PWM). The overall dimensions of the unit are 40.18 x 54.99 x 35 mm. Additional specifications are given in Table B-4 - B-5.

Mobile Sensor Platform

The robot kit IG42-SB4, a four-wheel differential-drive all-terrain robot platform, was purchased from SuperDroid Robots. This kit included the aluminum chassis, motor plates, wheels, motors, motor drivers, transmitter and receiver, 12V batteries, hardware kit, and a roll-cage that was reimplemented as a sensor rack. The wheel motors, 10A regulated fuse, kill switch, and motor driver were secured to the enclosed lower level of the robot, as can be seen in Figure 2, while the 12V batteries and sensor electronics are mounted in the middle section.

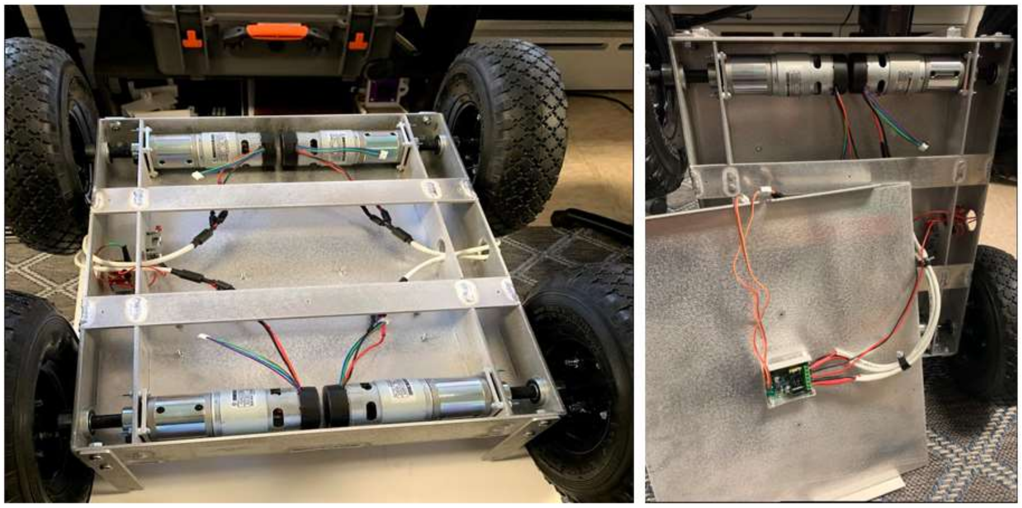


Figure 2: Robot's Lower Section. The lower section is where the motor controller, fuse, kill switch, and motors are stored. This is enclosed and prevents any of the components from potential damages.

The sensor rack was used to mount all the sensors, servo motors, and the monitor that displaced the graphical user interface. Custom mounts were 3D printed to incorporate the sensors as one unit attached to two servo motors (RobotGeek RGS-13) to give a 180° vertical and horizontal view. A FlySki I6 receiver is used to drive the robot and control the pan-tilt sensor mount.

Sensors and GUI Programming Outline

The data collection routines and Graphical User Interface (GUI) were built in Python and Arduino, while the machine learning processing was performed offline using Matlab. Data collection from the RGB video, thermal camera, and acoustic microphone array uses a Raspberry Pi 3 B+, while the LiDAR sensor was programmed using an Arduino Mega 2560. Unfortunately, the LiDAR became unresponsive during testing, and no data was collected for analysis. This issue was most likely due to the LiDAR's hardware due to the LiDAR lack of response to the any of the successfully programs that previously worked with the LiDAR.

The GUI was created in Python using Tkinter and showed streaming data from the RGB video camera (as both RGB and HSV values), the thermal camera, and an acoustic waveform from a single microphone channel (Figure 3) [33-34]. This streaming data could be recorded to disk in timestamped files for offline analysis.

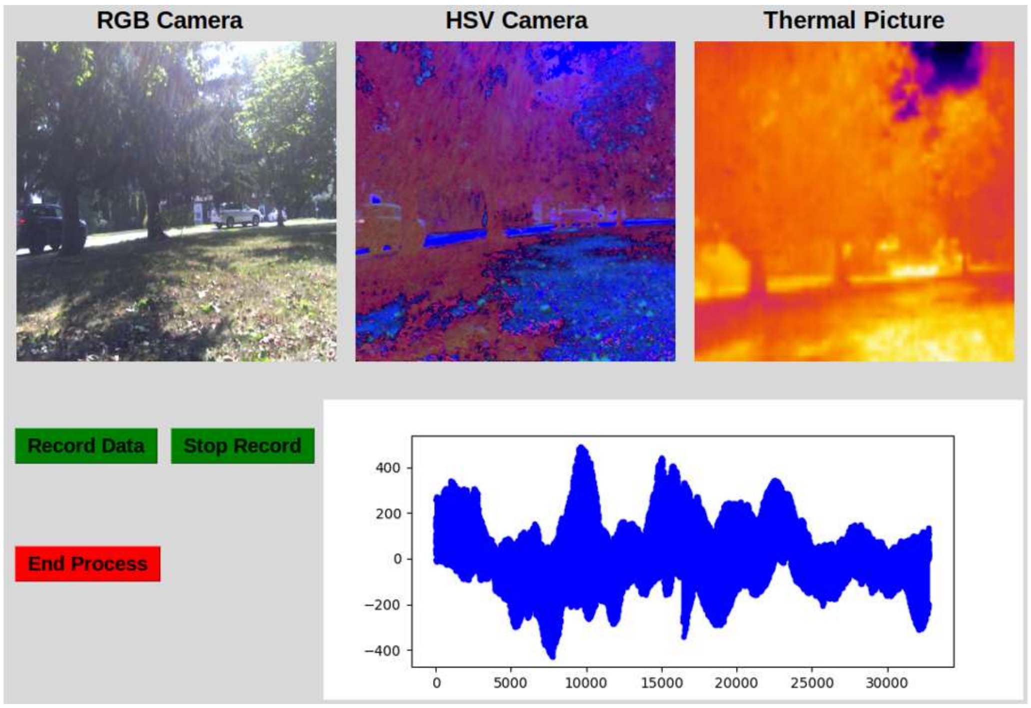


Figure 3: Graphical User Interface. The GUI shows the RGB camera produced the RGB and HSV images in the top left and top middle section, the thermal camera was displayed in the top right, the acoustic signal displayed in the bottom right, and the controls on the bottom left.

While the Raspberry Pi was able to handle the computational tasking from each sensor individually, it was not up to task for data collection from the entire suite of sensors. Instead a Minisforum Z83-F minicomputer with 4 GB RAM, Intel Atom Quad-Core CPU, and 64 GB storage was configured to be dual boot between Windows 10 and Linux Mint and used to collect all data.

However, the program then had to be compacted due to the time delay in the video stream in order to obtain as much real-time data as possible. The issue was the length of time the acoustic array required to process all the data points and plot on the GUI, preventing the other sensors from performing. The primary solution to fix this issue was the creation of a multi-threading process allowing the cameras and acoustic sensors to perform simultaneously.

While the GUI is unable to be considered real-time data, the program comes close to it. Either a more powerful minicomputer or further optimization of the program would be required to accomplish this fully. The data collection program can be seen in Appendix H.

Chapter 3. Data Collection

Drones

Three drones were obtained and used for the data collection process: the Parrot Quadcopter, the Parrot Swing, and the Tello Quadcopter (Figure 4). The Parrot Quadcopter is formally known as the Parrot Mambo Fly that has a 550 mAH. battery pack, ultrasonic vertical stabilization, horizontal camera stabilization, a range of 100 m. with Parrot Flypad, and a 60 FPS. vertical camera. The Parrot Mambo is 7.1 x 7.1 x in. with bumpers and weighs 2.22 oz [35]. The Parrot Swing has a max speed of speed of 19 mph., a 60 FPS. vertical camera for speed measurement, ultrasonic vertical stabilization, horizontal camera stabilization, a 550 mAh. battery, and a range of 60 m. with the Parrot Flypad. The dimensions are 13 x 5 x 5 in. and a weight of 10.4 oz [36-37]. The Tello Quadcopter has a max speed of 8 m/s. and connects to the Tello app on the iPhone or android devices and requires Wi-Fi. The drone has built in functions that include: Range Finder, Barometer, LED, Vision System, Wi-Fi, and 720p Live View. Also, the battery is 1.1 Ah., the overall dimensions are 3.9 x 3.6 x 1.6 in.; and the weight is approximately 80 g [38].

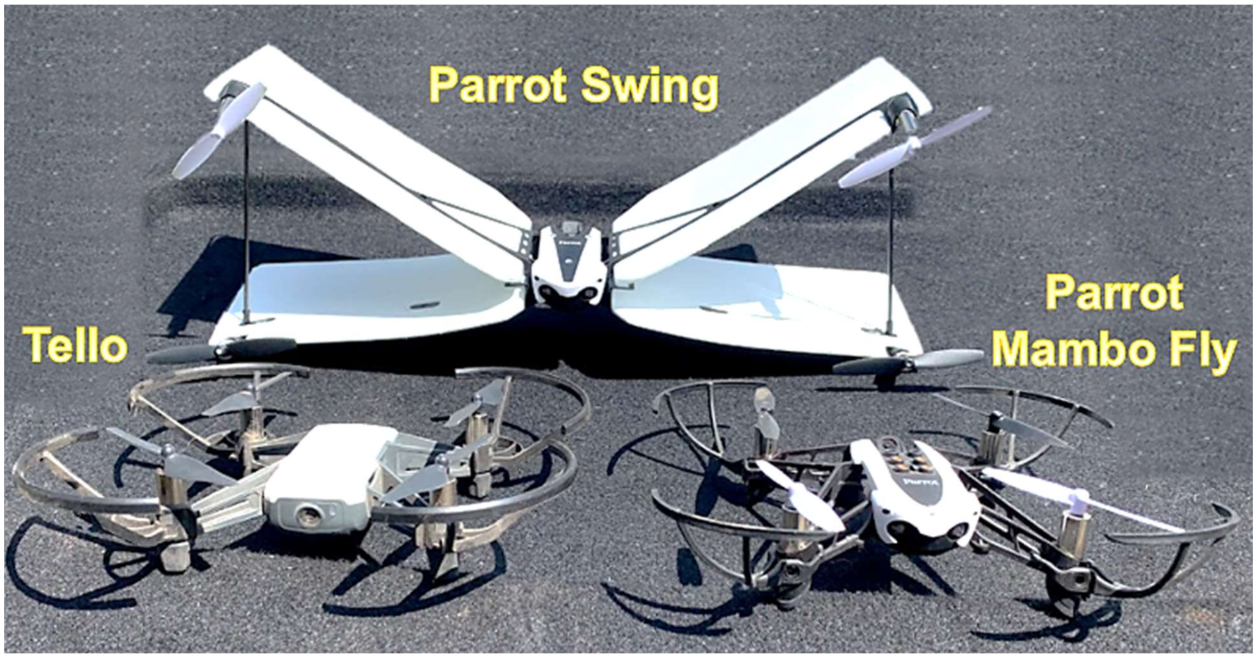


Figure 4: Parrot Swing, Parrot Quadcopter, Tello Quadcopter. The Parrot Swing is in the back, the Parrot Quadcopter is in the front right, and the Tello Quadcopter is in the front left.

Each of the drones was flown separately and at different angles to the sensors. The quadcopters were relatively easy to fly but the Parrot Swing was more challenging, especially if there was any wind causing the drones to leave the area of the sensors. All drones fly typically in the same manner, with the propellers parallel to the ground, with the side dipping downward in a certain direction allowing movement in that direction. The main reason these drones were included in the project is that the Parrot Swing is different size and shape compared to the two mini-quadcopters, which are very similar in size and appearance. However, the Parrot Swing is very similar in appearance to the Parrot Quadcopter at certain flight angles.

Capturing Data

Approximately 1,000 measurements were recorded from each of the video camera, thermal camera, and acoustic microphone array for each of the three drones (Table 3). Since creation of a robust machine learning algorithm requires training on data from realistic real-world environments, data was collected from five different scenes. These scenes were determined by including different variations of areas in which drones could possibly appear, using a street-based view to a sky-based view with variations in backgrounds. All data was manually filtered to make sure the drone was visible in the RGB or thermal image (partially occluded views were allowed).

Table 3: Overall Data Collection. The table shows the amount of the filtered data obtained from each sensor at the different locations. The total amount of the filtered data of each sensor is located at the right side of the table. For instance, the total amount of RGB images collected at Location 1 was 132 images, and the overall total amount of images collected was 992 images.

	Location 1	**Location 2**	**Location 3**	**Location 4**	**Location 5**	**Total**
Parrot Quad. RGB	**132**	**121**	**277**	**146**	**316**	**992**
Parrot Swing RGB	**60**	**70**	**523**	**65**	**456**	**1174**
Tello Quad. RGB	**155**	**119**	**299**	**80**	**403**	**1056**
Parrot Quad. Thermal	**128**	**120**	**250**	**154**	**328**	**980**
Parrot Swing Thermal	**56**	**61**	**439**	**62**	**495**	**1113**
Tello Quad. Thermal	**132**	**115**	**295**	**82**	**418**	**1042**
Parrot Quad. Acoustic	**112**	**122**	**232**	**131**	**160**	**757**
Parrot Swing Acoustic	**45**	**64**	**428**	**57**	**365**	**959**
Tello Quad. Acoustic	**138**	**116**	**227**	**66**	**313**	**860**

The first data collection location was in the front yard angled towards the right side of a house (Figure 5). The RGB camera had a good background view of the trees, the street, grass, and the sky. Due to mounting and FOV differences, the thermal camera always shows a slightly smaller portion of the image at a slightly different angle. Ideally, throughout the data collection people were walking by or riding bikes, cars were driving by, and other realistic noise sources are captured in the data set. A sample of the RGB and thermal image for the first location can be seen in Figure 6. The closer the drone came to the sensors, the less it could move due to its quickly disappearing out of the picture frame quickly.

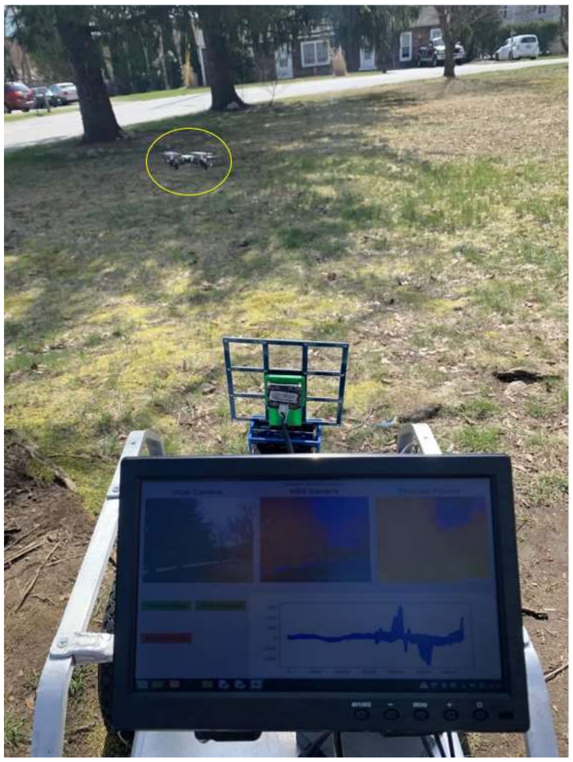


Figure 5: Data Collecting Setup. The general setup for the collecting process of the drones at the first location.

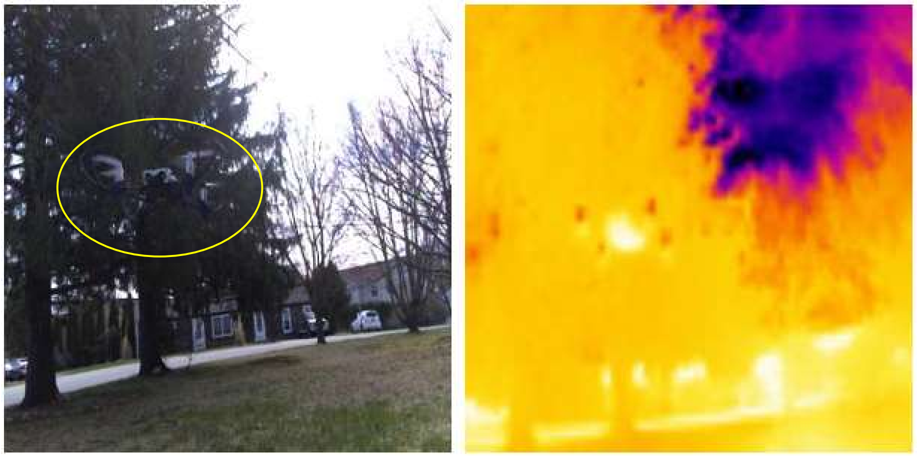


Figure 6: First Location Data Sample. These images represent the data from the sensors obtained from the first location with the Parrot Quadcopter. The left image is the obtained RGB image and the right image is the collect thermal camera. The images above are within the same second of each other.

The second data collection location was in the front yard with the sensors aimed towards the sky (Figure 7). The RGB and thermal camera have shots of moving tree branches and clouds, with relatively stationary trees and telephone wires in the background. Other background noise variables included birds, cars driving by, people mowing lawns, or people talking. There was a considerable amount of wind during this time that hindered the drones from being stable in front of the sensor.

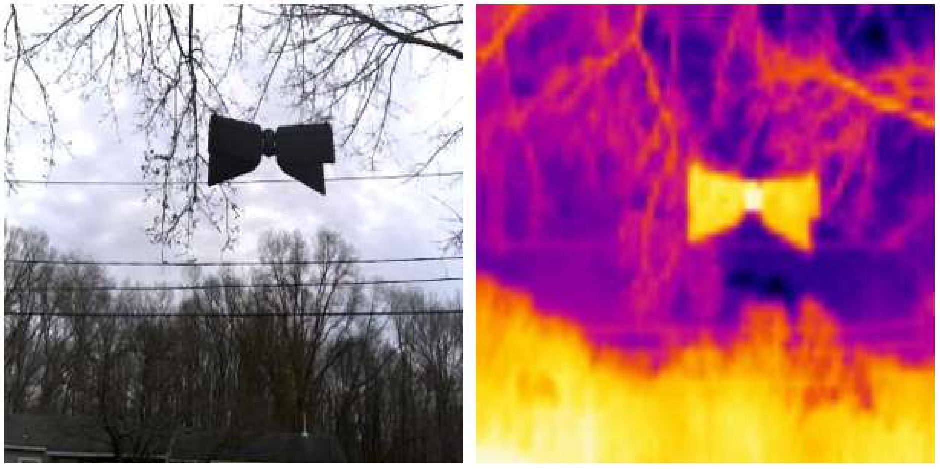


Figure 7: Second Location Data Sample. These are sample images that were obtained from the second location with the Parrot Swing. The left image is the RGB image and the right is the thermal image within the same second of data collection.

The third data collection location was in the backyard, angled toward a wide-open sky (Figure 8). There were plenty of clouds to have a moving background, with a few tree branches in view. Other possible random noises were children playing, dogs barking, birds, and people doing yard work.

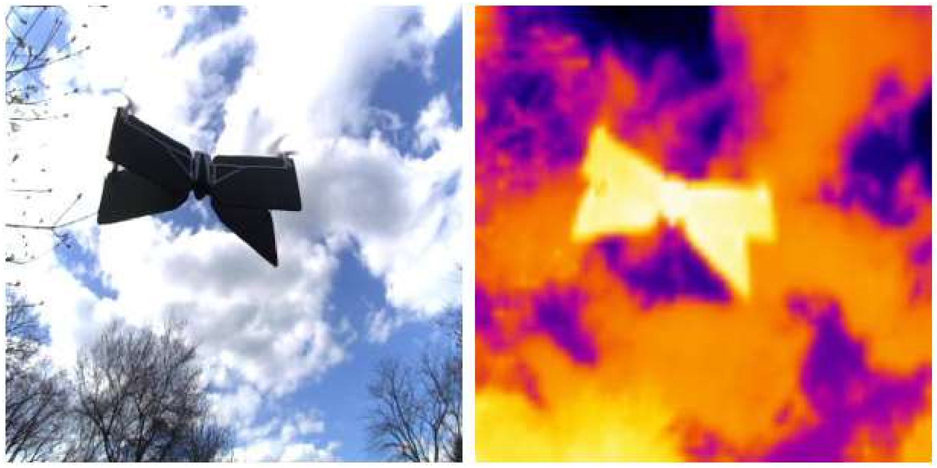


Figure 8: Third Location Data Sample. These are sample images that were obtained from the third location with the Parrot Swing. The left image is the RGB image and the right is the thermal image within the same second of data collection. This utilizes the moving clouds for constant changing background more than the other locations.

For the fourth location, the robot's sensors were in the direction that had the camera view a majority of trees with the sky visible, allowing the clouds to roam the background (Figure 9). The visible variations were ideally based around birds and clouds, while the acoustic noise was ideally based on neighborhood sounds. Unfortunately, this was a windy day, and the Parrot Swing collided with a tree, fracturing the propeller and grounding the drone. A replacement Parrot Swing was used for the remaining measurements. This new model had a wing material that was softer and more elastic, which made it noticeably more comfortable to control but likely caused significant variation in the Parrot Swing dataset.

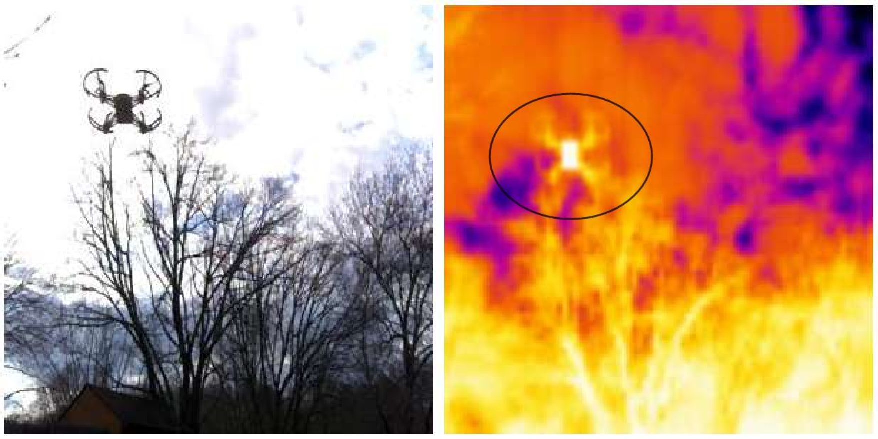


Figure 9: Fourth Location Data Sample. These are sample images that were obtained from the fourth location with the Tello Quadcopter. The left image is the RGB image and the right is the thermal image within the same second of data collection. The tree and moving clouds background provide decent variance for both sensors shown.

The fifth data collection location was the front yard with the camera lens incorporating a house, the street, parked cars, plants, power lines, garbage bins, trees, plants, and the sky (Figure 10). There are many variations in this scenery with the noise variance of the neighborhood, cars, people walking or riding bikes, and animals. This location allowed for the greatest distances of fifty to ninety feet away from the sensors.

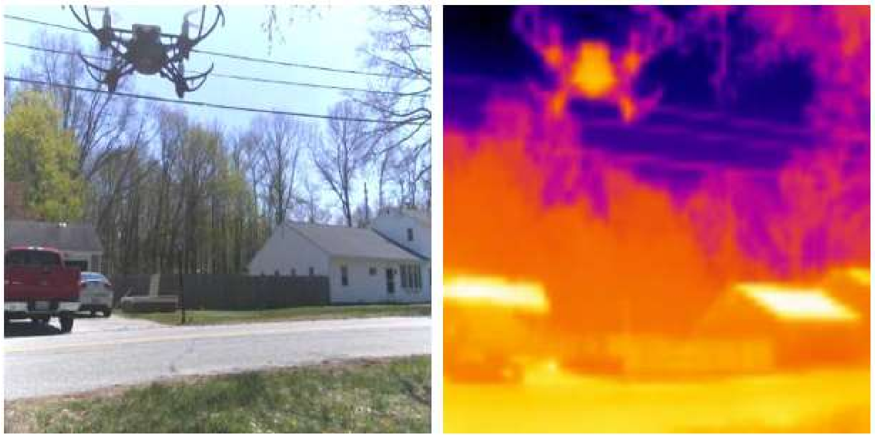


Figure 10: Fifth Location Data Sample. These are sample images that were obtained from the fifth location with the Parrot Quadcopter. The left image is the RGB image and the right is the thermal image within the same second of data collection. The inclusion of the hot roof of the house and cars were to help determine thermal sensor capabilities, with the temperature reaching to 89° F that day.

Variation and Noise

Variations and noises are critical for the machine learning process to learn and prevent overfitting. Real data is not perfect, and in the case of detecting the drone, the drone does not contain only a sky background. There are also people, cars, animals, and much more that will be in a real application of the machine learning process. With no variation, the prediction of a drone with a lamp post may cause severe problems with the prediction method.

For the first location, one of the main visual variations were cars and people walking their dogs. For acoustic, there were cars driving by, people walking, construction on a house across the street and two houses to the left, as well as other neighborhood sounds were occurring. For the thermal images, the drone was flown into different conditions, such as the hot and cold regions of the visual sensor. A few examples of these variations and noises can be seen in the following figures, Figures 11 - 13.

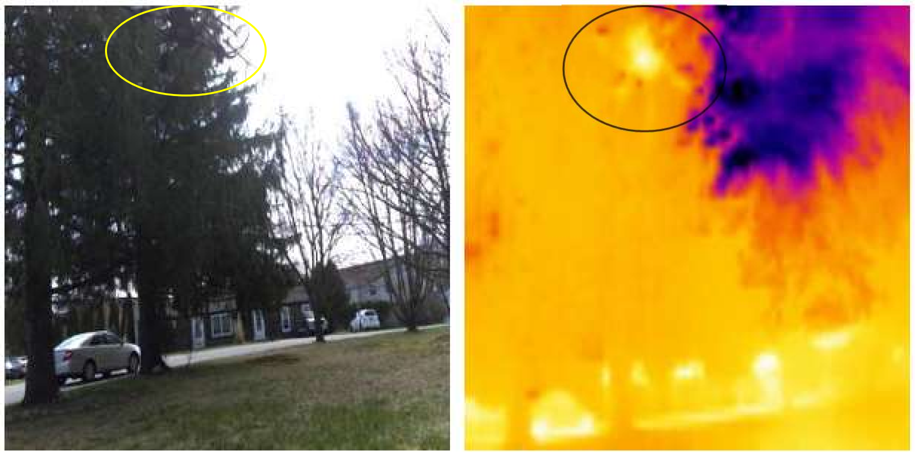


Figure 11: Location 1 Car Variations. Data collection of the Parrot Quadcopter, top center of both images, with a vehicle driving in the background. This type of variation is common for visual and noise variance with all drones and throughout all locations.

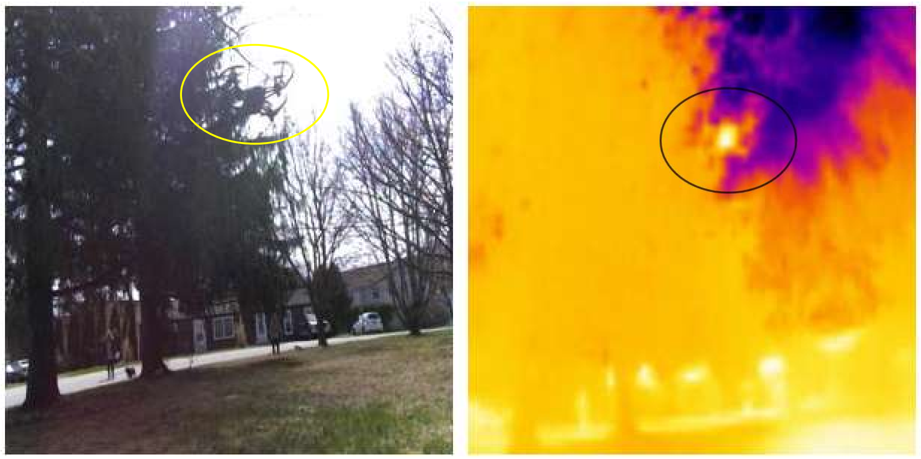


Figure 12: Location 1 People Walking Dog Variation. The data collection of the Tello Quadcopter, top center of both images, with people walking dogs in the background at the first location. This is difficult to see on the thermal camera, but the heat signatures moving are still visible and, as a result, help the machine learning process.

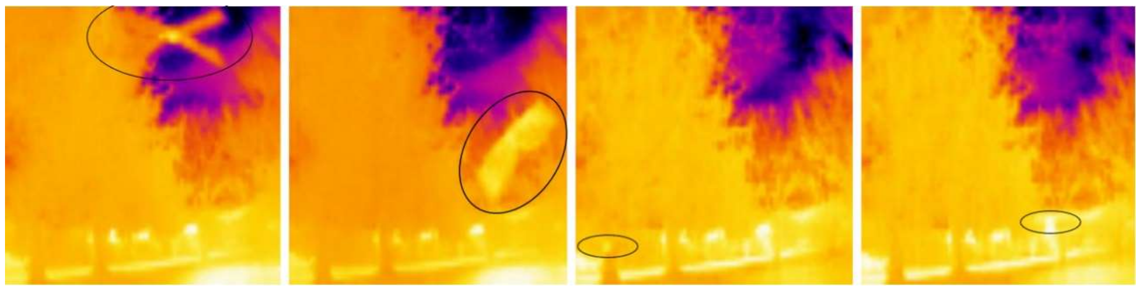


Figure 13: Location 1 Temperature Variation. The series of images are the Parrot Swing thermal variations in size and position throughout the heatmap at first location. This type of variation is performed on all drones at all locations. However, there are different heat intensities at each location.

For the second location, the variations and noises were very similar to the first: the variations and related noise involved the drone flying in a different setting, the distance of the drone, the wind blowing the trees, cars driving, neighborhood sounds, and flying the drone in hot and cold regions of the thermal image. The top portion of the vehicles can be seen in the RGB camera; however, the thermal camera was angled too high to include the vehicle in the thermal images. A temperature variation of the second location can be seen in Figure 14.

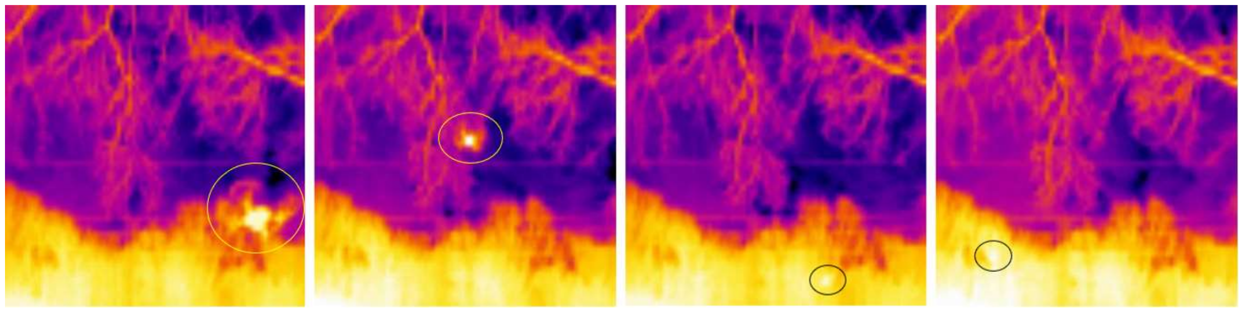


Figure 14: Location 2 Temperature Variation. The series of images of the Tello Quadcopter were collected and show the thermal variations in distance and position throughout the heatmap at the second location. The drone is very difficult to see the more heat intense areas in the thermal images as the drone is flown further away from the sensors.

For the third location, the drone's visible variations were the clouds moving in the background, the tree branches swaying, birds flying, and the drone moving in all areas of thermal intensity regions. The noise variance was neighborhood noise, which included cars, people talking, and people mowing the lawn. In order to help show some of these variations and noise variance, Figure 15 – Figure 16 are shown below. Figure 15 shows the sample at the given time with a bird in the background. Figure 16 shows the time-lapse of the sky over 30 seconds with the Tello Quadcopter on the top section with the calm sky, and the Parrot Swing on the bottom section with the cloudy sky.

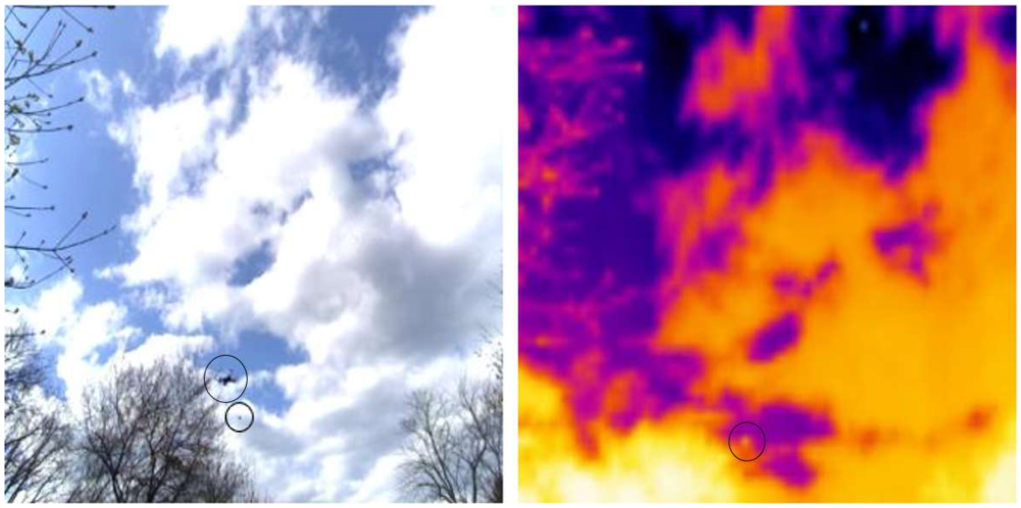


Figure 15: Location 3 Bird Variance. During the data collection of the Parrot Quadcopter the drone was able to fly with a bird further behind the drone at the second location (a circle was inserted to help identify the bird). On the thermal camera, the drone is hard to see due to the size, and the bird is not visible.

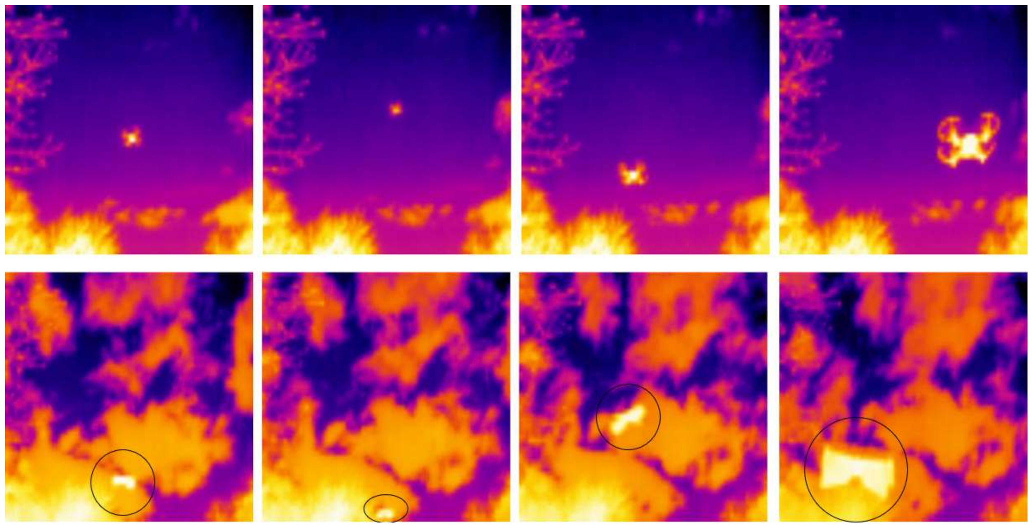


Figure 16: Location 3 Thermal Time-lapse. Tello Quadcopter and Parrot Swing thermal time-lapse over 30 seconds.

The fourth location’s variations and noise variance included the standard variations as above with the drone in all thermal regions and neighborhood noises. Uniquely, the fifth location captured a significant number of vehicles passing by compared to the previous locations, a person on a bike, a new Parrot Swing, as well as having the drone go much farther than the previous data collection processes. Also, the neighborhood noise variance was still accounted for in this process. A person can be seen riding the bike on a hot day next to the Parrot Quadcopter in Figure 17. The roofs of the houses are so hot that the thermal images are much harder to see compared to the other location's thermal images.

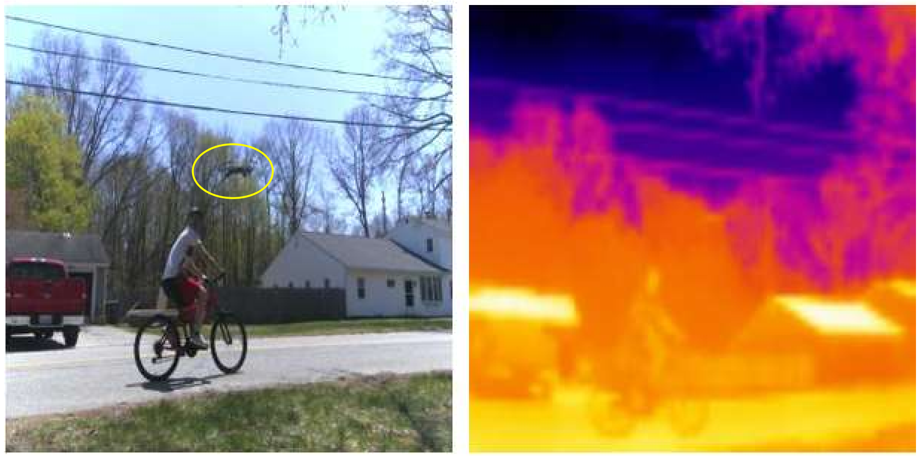


Figure 17: Location 5 Biker. Parrot Quadcopter next to a person on a bike at fifth location.

Potential Machine Learning Issues

When the data was being filtered, a few potential issues in the machine learning process were likely to happen. The first issue was the drone’s distance from the camera. The drones were in the frame of the camera beyond 50 ft., and they were hard to distinguish from the background with human eyes. For instance, Figure 18 shows the Parrot Quadcopter in the middle of the road, and the drone is small and blurry. The distance from the sensors to the opposite end of the street was approximately 50 ft., and the distance from the sensors to the fence was approximately 90 ft.

In Figure 19, the two other drones were having the same issue without the fence, the Tello Quadcopter on the top section and the Parrot Swing on the bottom section. The drones never went past the fence, and for the majority of the time were around the lamp post to respect the neighbor's property. When the quadcopters were across the street, the trees made it extremely difficult to detect the drone with the human eye. The white Parrot Swing had similar issue, with the features becoming difficult to distinguish at the same distance, especially if the drone was in front of the white house.

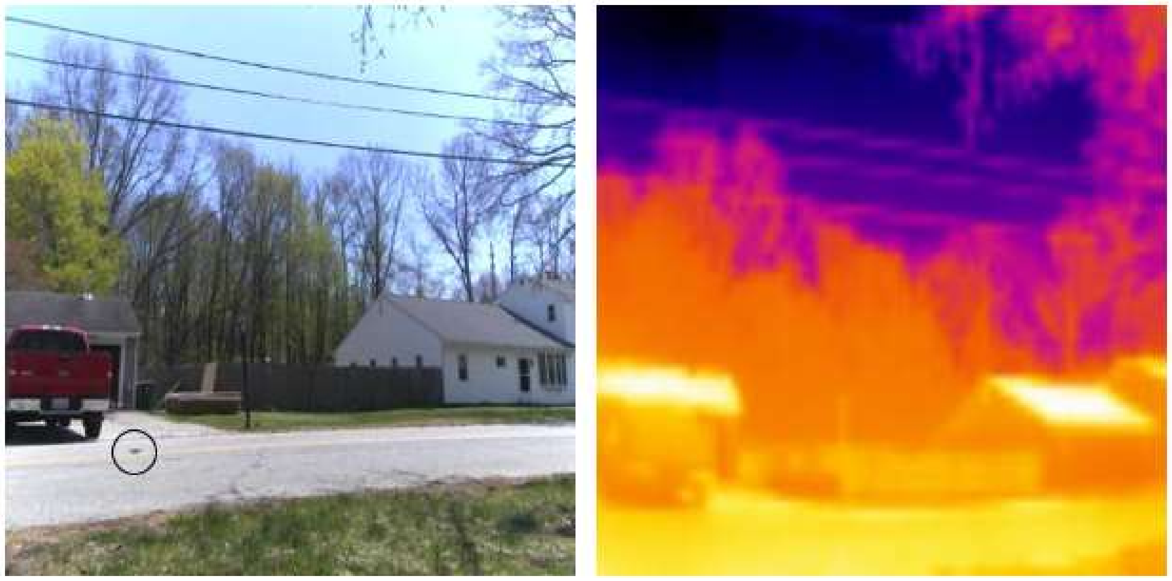


Figure 18: Location 5 Camera Distance Issue. The quadcopter has landed in the middle of the road.

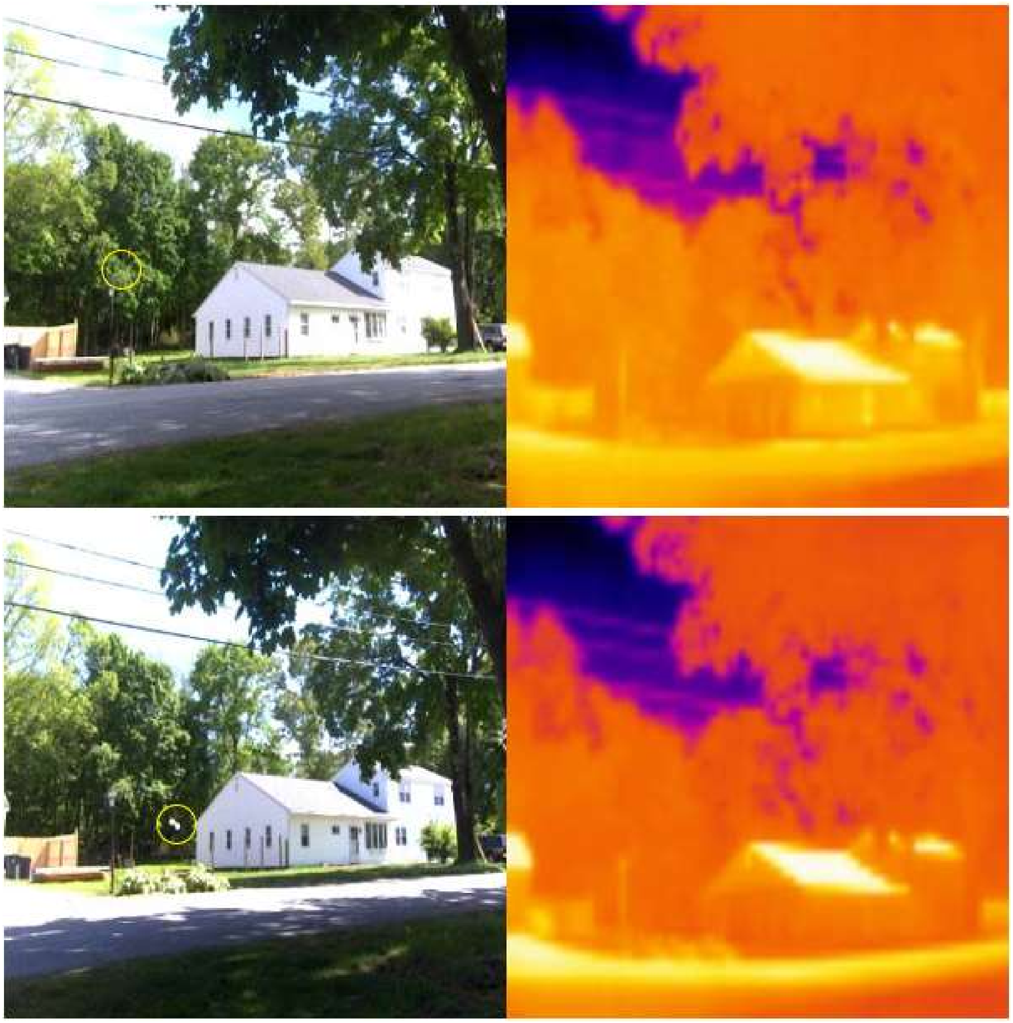


Figure 19: Camera Distance Issue. Tello Quadcopter and Parrot Swing near lamp post. However, the RGB camera is difficult to see and the thermal camera does not appear to detect the drones.

The second potential issue is the reliability of the thermal images when exposed to an intense area of heat, such as the roof of the house. This issue makes detecting the drones with the human eye incredibly hard. A Parrot Swing was flown directly in front of the sensor while maintaining the roof of the house in the image, and the drone was still hard to visualize, as shown in Figure 20. In the midrange, the drones were still having the same issue; even the cars were hard to distinguish, as shown in Figure 21 with the Tello Quadcopter on the top section and the Parrot

Swing on the bottom section. The roof of the house is expected to significantly skew the data of the machine learning process due to its being too hot.

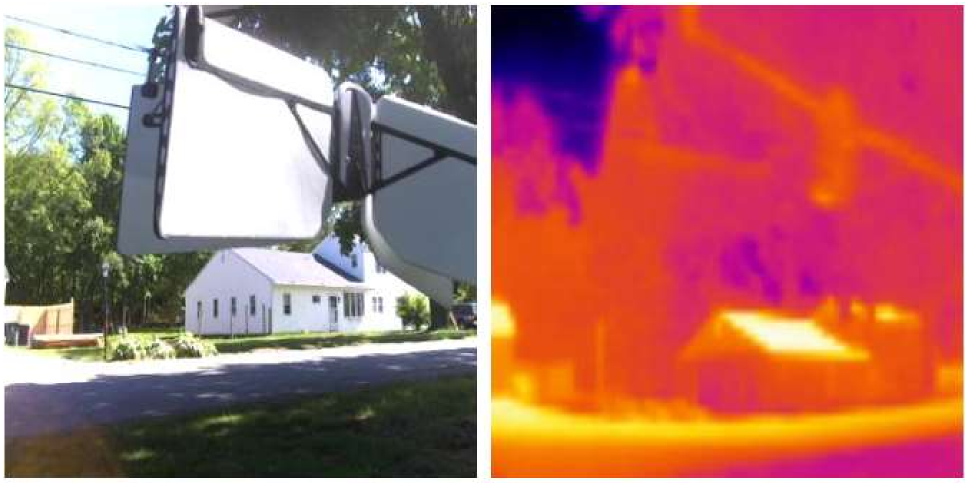


Figure 20: Thermal Roof Issue. The Parrot Swing is close to sensors with extreme thermal intensity background, and it is difficult to see defined features of the drone.

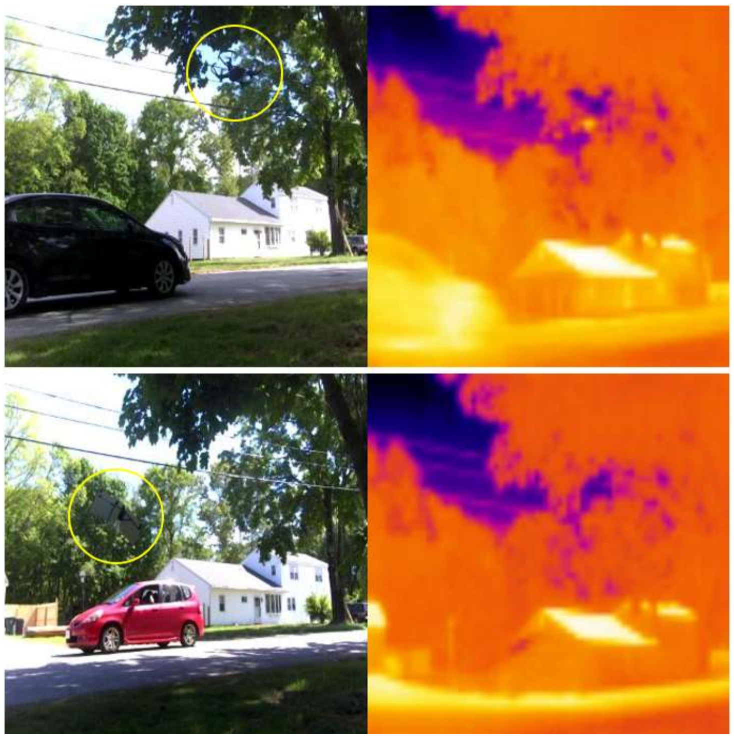


Figure 21: Roof Heat Issue 2. Tello Quadcopter and Parrot Swing with vehicles in extreme thermal intensity background. The drones and vehicle are difficult to detect in the images.

Chapter 4. Machine Learning Application

Neural Network Background

Artificial intelligence – creating computers which demonstrate human behavior – can be thought of as a level above machine learning, which is enabling computers to automatically detect patterns in data and use these "learned" patterns to predict the outcome when given new data without explicitly being programmed. One of the main types of machine learning is the predictive or supervised approach. This approach involves training a system with training sets along with the known outputs. Another main type of machine learning is the descriptive or unsupervised learning approach. This approach involves providing the machine learning algorithm with only inputs and to try to find the patterns in the data.

Traditionally, machine learning algorithms tried to define a set of rules by hand-engineered, but easily explainable, data features, leading to a time-consuming, brittle process that is not scalable in practice. More current deep learning techniques use algorithms inspired by the human brain, such as neural networks, to extract patterns from a set of data[1]. The tradeoff is a limited view into what the computer is "learning", which require large, varied datasets to create robust models.

The goal of a deep learning network is to turn input x into output y in a manner that can be altered to achieve the anticipated results (this is the "training" process). Once the system is trained, this learned set of "weights" allows the correct prediction of output given a new input value. Neural

[1] This background of this chapter is borrowed from Alexander Amini and Ava Soleimany MIT 6.S191: Introduction to Deep Learning, which presented a clear and concise explanation and simplification of the definition of the convolutional network down to the perceptron.

networks accomplish this by combining multiple single neurons, called perceptrons (Figure 22), each of which create output *y* given input *x* as

$$\hat{y} = g(\sum_{i=1}^{n} x_i w_i + bias), \tag{1}$$

Where w are the corresponding weights that are altered in the training process. The nonlinear activation function *g* increases system accuracy by adding in real-world nonlinearities. Commonly used activation functions are the sigmoid, hyperbolic tangent, and rectified linear unit functions.

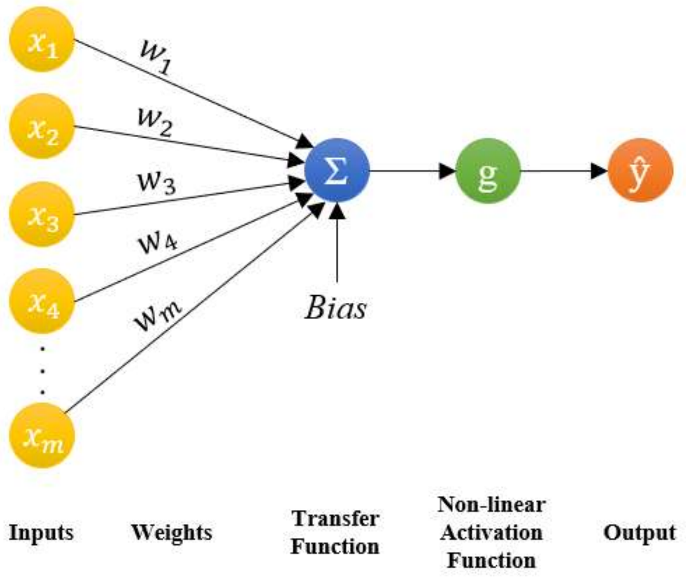


Figure 22: Perceptron Diagram. The figure shows the summation of the inputs, x, being multiplied by the weights, w, with the bias applied, and going through a nonlinear activation function, g, to produce the output, ŷ.

A neural network is created by combining perceptrons, the inputs, and weights, and collapsing them into separate vectors X and W, and then the output is defined as

$$\hat{y} = g(X^T W + bias). \tag{2}$$

When another perceptron is added, it connects to the previous layer with a difference in the weights. These layers are often referred to as dense layers due to all the inputs being densely connected to all of the outputs. The previous figure is then expanded to a single layer neural network shown in Figure 23.

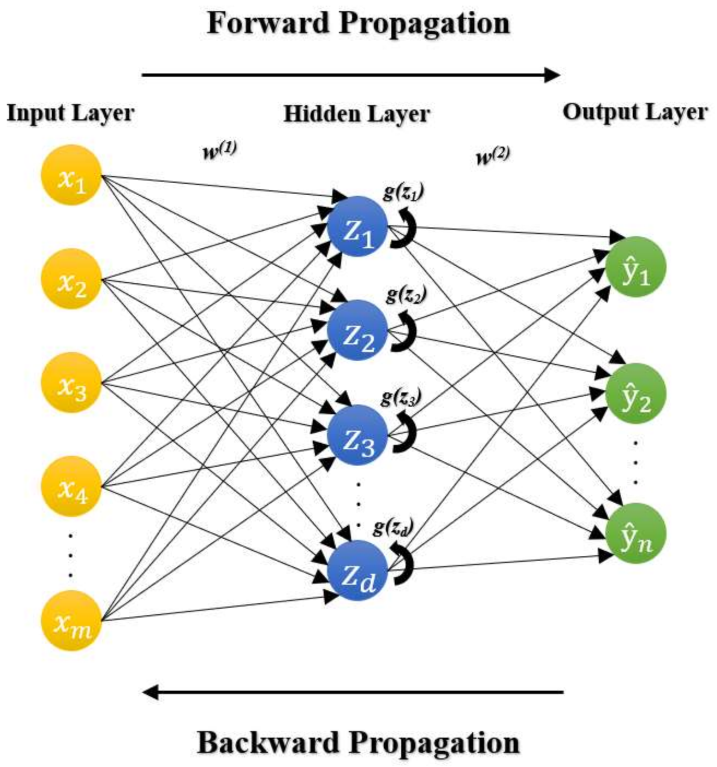


Figure 23: Neural Network Diagram. This shows multiple inputs, perceptrons, and outputs layers in the network.

The hidden layer's output can be determined by

$$z_j = \sum_{i=1}^{m} x_i w_{i,j}^{(1)} + bias^{(1)} \tag{3}$$

and the output layer can be determined by

$$\hat{y}_j = g\left(\sum_{i=1}^{d} x_i w_{i,j}^{(2)} + bias^{(2)}\right). \tag{4}$$

The reason the center is called a hidden layer is that these layers are not directly enforced or observable, unlike the input and output layers, which means that the hidden layer is learned and can be probed to determine what is going on inside the network. The variables $W^{(1)}$ and $W^{(2)}$ represent the weights corresponding to the first or second layer. In order to make the neural network a deep neural network, more hidden layers are incorporated to create a more hierarchical model.

Once the user has labeled, the next step would be to train the model. The first step in achieving this is to tell the network when the prediction is wrong, and this is done by quantifying the error, also known as the loss. There are different types of losses, depending on whether it is classification or regression. When the output is categorical, the system is defined as classification or pattern recognition. However, if the output is real-valued, then the system is known as regression. For classification, the cross-entropy loss would produce an output between 0 and 1 by

$$Loss = \frac{1}{n}\sum_{i=1}^{n} y^i \, log\left(f\left(x^i; W\right)\right) + \left(1 - y^i\right)log\left(1 - f\left(x^i; W\right)\right). \tag{5}$$

In determining the loss of regression, a popular loss is the mean square error, which is

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{6}$$

The y represents the actual and $\hat{y}$ the predicted output. The next objective is to find the ideal set of weights that would result in the minimum amount of loss for the model. The loss is optimized by using the process of gradient descent that maps the set of weights and tries to find the lowest point on the map, known as the local minimum of the loss. This process uses backward propagation to determine the best direction to move with a given loss and a given set of weights [39].

Convolutional Operation Background

Now that this brief description of neural networks can be understood, the idea of how a computer sees an image must be explained. A picture is made of thousands or even millions of pixels. Pixels are the smallest point in the image, and these pixels are displayed in columns and rows to display the image. Depending on the type of image, the pixel is then translated to a number and is able to be processed. Two very common types of images are grayscale and RGB images. In a grayscale image, the pixels are able to be represented by a single number, converting the image into a two-dimensional matrix of numbers based on the brightness values. For an RGB, three two-dimensional matrixes are concatenated on top of each other, one to represent each of the red, green, and blue channels.

For classification, the computer would need to recognize the unique differences between pictures. Each classification class has a set of unique characteristics, called features. If the computer is able to detect enough of the features in that class, the computer would be able to determine what class the image would belong to with high probability. A good approach is to learn the visual features directly from the data and learn the hierarchy of these features as well. In doing so, it would be possible to reconstruct a representation of the class label. Convolution is then used to extract the features and patterns. Rather than having every pixel as its own layer in a neural network, a patch would go through the pixels and connect the patches to the neurons of the hidden layer. A representation of how a patch region influences a single neuron can be seen in Figure 24.

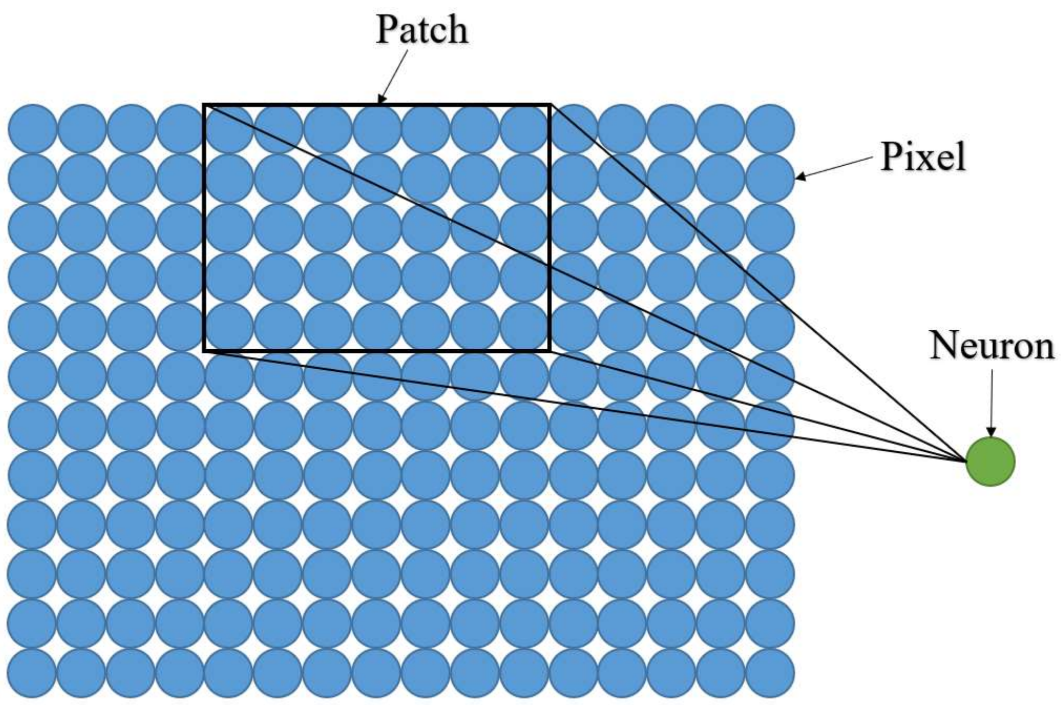


Figure 24: Patch Diagram. The image shows a patch sliding along pixels. Instead of every pixel being set to an individual neuron, the pixels within the patch are incorporated into the neuron.

The patch is slid across the image to define the connections across the input. By doing this, the spatial structure and information are maintained. To learn visual features, those connections in the patches are then weighted and then summed for the input to the following layer.

Images are not strictly compared to another image; there will be certain types of deformations to the images, scale, shift, or rotation. To compensate, the images will be compared patch by patch. Features are the important patches the network looks for, and if rough matches are found, the probability is high that they are the same image class. If the two images share a high number of features, then the objects have a high probability of being the same object. These features are like mini-images and are often referred to as filters. These filters have a set of weights for each pixel and are slid along the image. An example of a triangle being compared to a similar image can be seen in Figure 25.

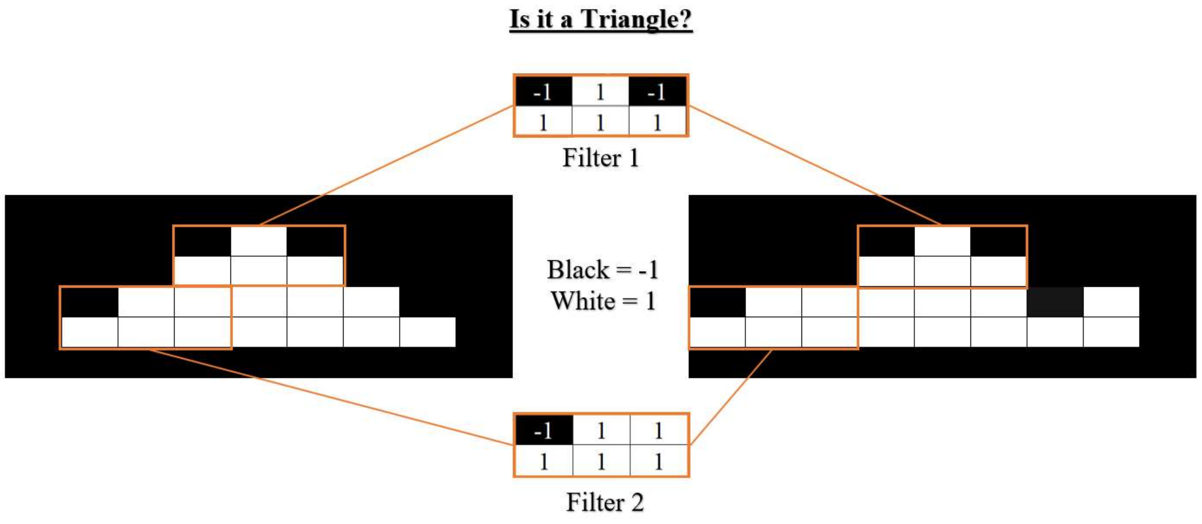


Figure 25: Triangle and Filter Diagram. The filters are unique features that are slid across the pixel to try to find close representations.

When the patch is on top of the image, the output of the hidden neuron layer can be determined by elementwise multiplication of every pixel that the image and filter overlap, and then by totaling all of the numbers to produce the overall output. An example can be seen in Figure 26.

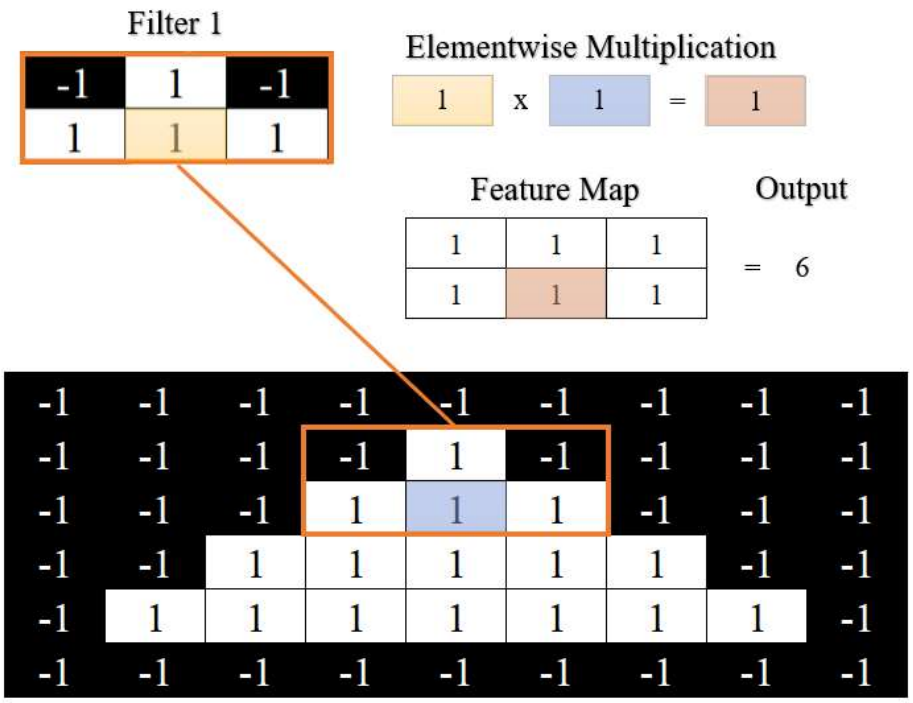


Figure 26: Feature Map. The black pixels are -1 and the white pixels are 1. Applying the filter to an image produces a feature map through elementwise multiplication.

The figure above can be seen with a 3x2 filter and is placed on the image. The elementwise multiplication is performed for every overlapping pixel; since this is a perfect overlap, it is all ones, and they are added to the feature map. Then the overall output of the hidden neural layer is the sum of all the numbers in the feature map to produce 6, the max output of that filter. Changing the weights in the features will significantly impact the feature map and can help sharpen the image or be used for edge detection [40].

Convolutional Neural Network Background

A convolutional neural network process can be described as two sections, the feature learning, and the classification. In feature learning, there are three main steps to consider—first, the convolution operation of extracting features in the image. A single convolutional layer can have multiple different filters, which makes the output layer of a convolution a volume of images that represents the different filters. The number of filters to detect at every layer in a convolutional neural network is set by the programmer and not the network. The rectified linear unit nonlinearity activation function is commonly used to shift all the negative values by zero. Second, the nonlinearity has to be applied to allow the neural network to handle nonlinear data, which enables the network to handle more complex tasks. Finally, the pooling operations has the spatial resolution of the image downsampled and also handles multiple scales of the features within the image. A common pooling technique is max pooling, where another patch is slid along the matrix and takes the maximum value. This is repeated until the image is downsized, and this allows the maintaining of the spatial structure while shrinking the spatial dimension. The classification section then takes the learned features and feeds them into a dense layer to represent the final output of producing the probability distribution of the membership of the image of the different classes. A classic convolution neural network architecture can be seen in Figure 27.

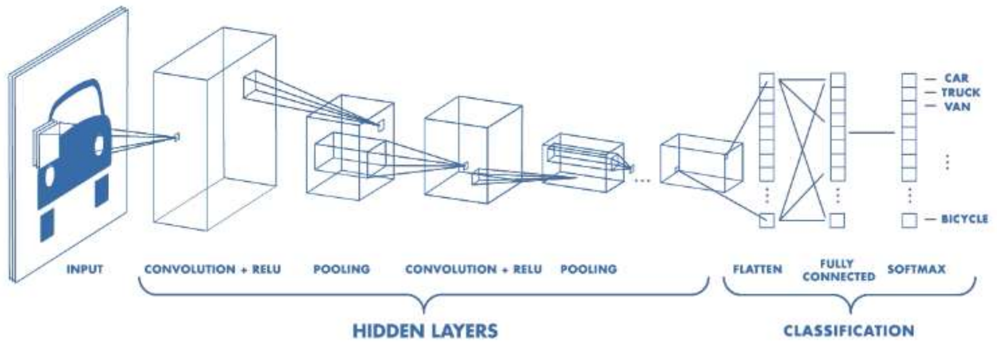


Figure 27: Convolutional Neural Network. This image shows the image of a car as an input. The convolution process is performed and is then sent to the neural networks for classification [41].

Chapter 5. Machine Learning Implementation

Machine Learning Mel Frequency Cepstrum Coefficients Background

Mel Frequency Cepstrum Coefficients (MFCC), illustrated in Figure 28, is a technique for audio processing and is commonly used for speech recognition [42].

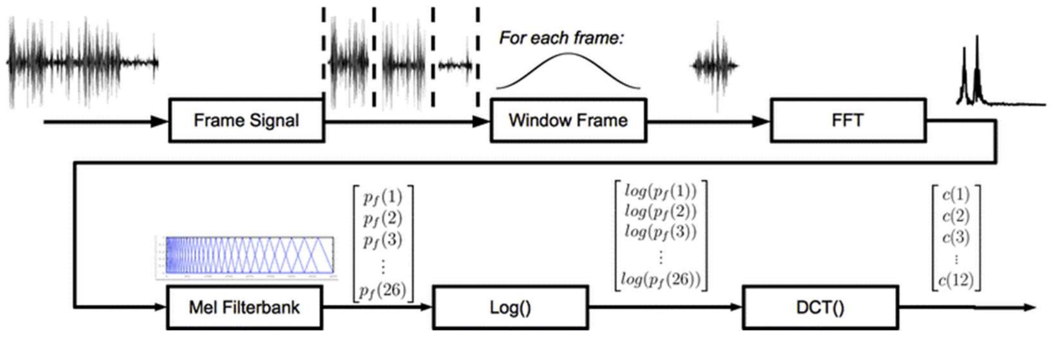

For each frame:
Frame Signal
Window Frame
FFT
Mel Filterbank
$\begin{bmatrix} p_f(1) \\ p_f(2) \\ p_f(3) \\ \vdots \\ p_f(26) \end{bmatrix}$
Log()
$\begin{bmatrix} log(p_f(1)) \\ log(p_f(2)) \\ log(p_f(3)) \\ \vdots \\ log(p_f(26)) \end{bmatrix}$
DCT()
$\begin{bmatrix} c(1) \\ c(2) \\ c(3) \\ \vdots \\ c(12) \end{bmatrix}$

Figure 28: Mel Frequency Cepstral Coefficient Process. This shows the overall process to obtain the Mel Frequency Cepstral Coefficient [44].

A time domain signal is first passed through a high pass filter to reduce noise. The signal is sliced into small frames where it is assumed that the frequency is stationary (typically the frame size range 20 - 40 ms for speech) and a Hamming window is applied to each of the frames to reduce spectral leakage. The N-point Fast Fourier transform is performed on each of the frames to produce the frequency spectrum. The next step is to produce filter banks by applying triangular filters on a Mel-scale to the power spectrum to extract frequency bands. The Mel-scale objective is replicate the human ear's logarithmic perception of sound by being more discriminative at lower frequencies than higher frequencies. Each of the triangular filters in the filterbank has a response of 1 at the center frequency and decreases linearly until it reaches 0 at the center of the adjacent frequency filter, as shown in Figure 29.

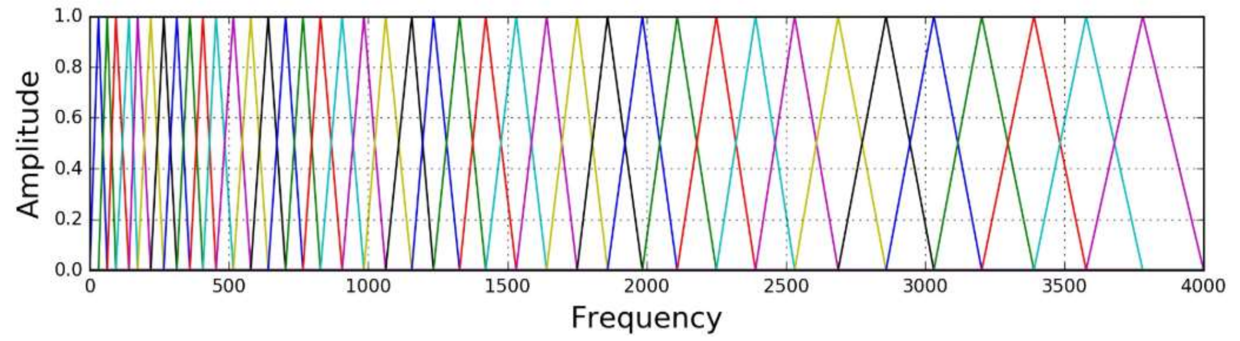
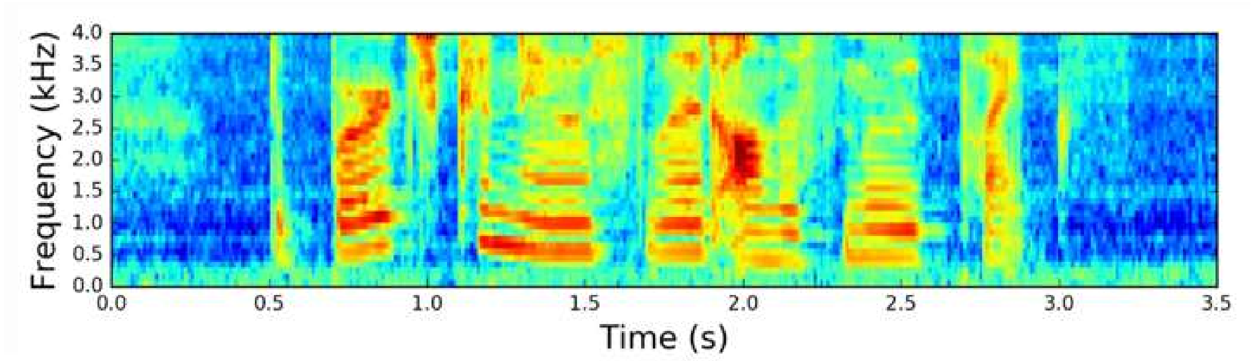
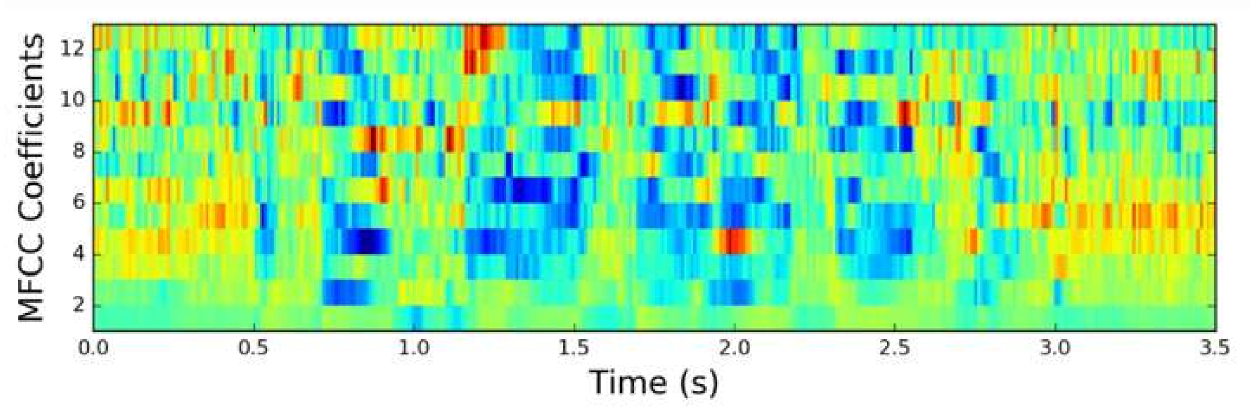


Figure 29: Filterbank on a Mel-Scale. [43].

However, the filterbank coefficients contain highly correlated data that has the potential to be problematic in machine learning systems. This is corrected by applying the discrete cosine transform that produces a compressed representation of the filterbanks, called the Mel Frequency Cepstral Coefficients [43]. A comparison of the Mel filterbank image and the Mel Frequency Cepstral Coefficient can be seen in Figure 30.



a) Normalized Mel Filterbank



b) Normalized Mel Frequency Cepstral Coefficient

Figure 30: Filterbank Vs Mel Frequency Cepstral Coefficient. The top image shows an example of the mean normalized filterbank after the Mel-Scale process and the bottom image shows the mean normalized Mel Frequency Cepstral Coefficient based on the Filterbank [43].

The audio processing script (Appendix I) reads in recorded .wav files using the 'librosa' library, performs the Fast Fourier transfer using 'numpy', and creates the Mel Frequency Cepstral Filterbank and the Mel Frequency Cepstral Coefficient image files using the 'python_speech_features' library [45]. The Python default settings for window length and window step were 25 ms and 10 ms, respectively, and were decreased to 1 ms and 0.4 ms, respectively.

This allowed more details of the image to be displayed since the size of the window, and the amount that the window moves over, are decreased.

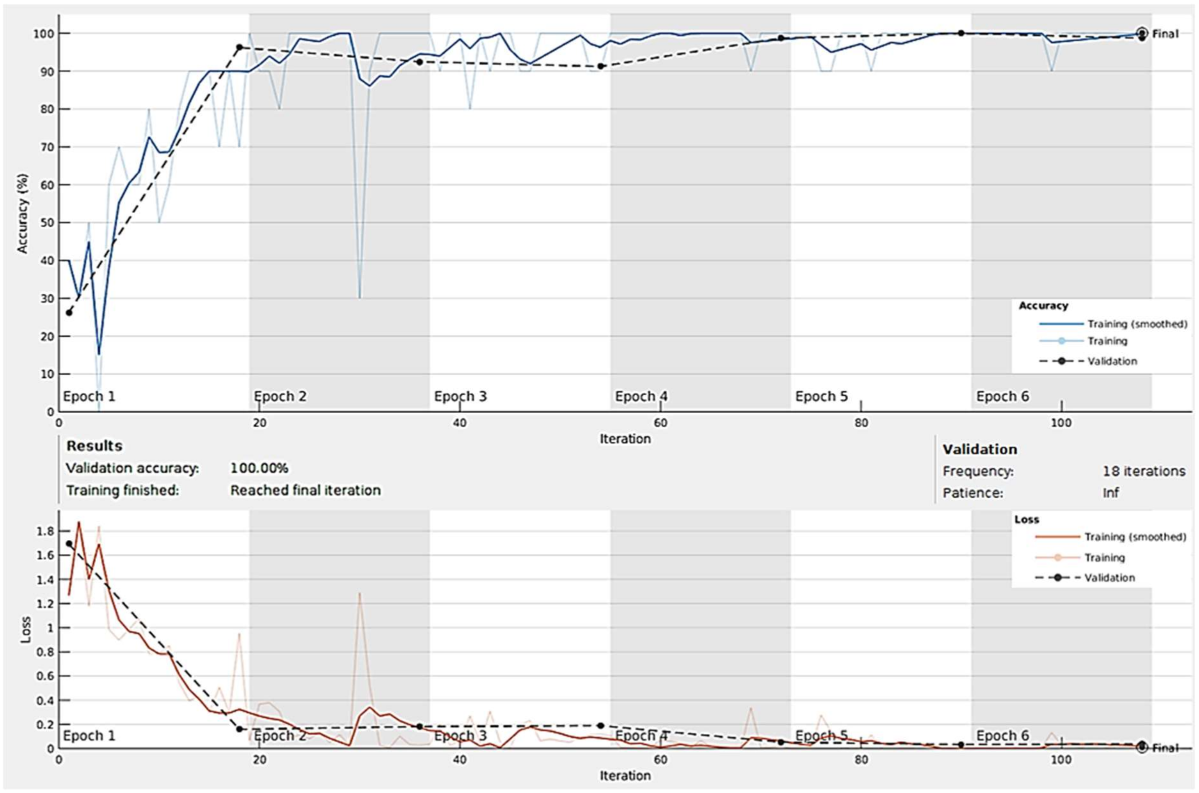
Machine Learning Program Testing

An Asus laptop with 12 GB RAM and an Intel Core i5-8250U CPU was used for the machine learning in Matlab using as inputs the RGB, thermal, and Mel Frequency Cepstral Coefficient images. The machine learning program incorporates Matlab's implementation of the Resnet-50 pre-trained convolutional neural network. The Resnet-50 will be retrained to learn the classes of the Parrot Swing, Parrot Quadcopter, and the Tello Quadcopter based on the input categories acoustic, thermal, and RGB of each of the drones. The minimum number of the data in each category of the three drones will be the max number of data inputs for that category. If a certain drone's category is over the max amount of input data, then the data input into the program will be randomly selected until the max number of input data is reached. After the machine learning process, the accuracy and loss will be evaluated at the end of the process, and additional measures will be performed if needed [46].

To indicate how well this program might perform on the data acquired by the sensors, a test program was created to implement the RGB and acoustic categories from online databases. The RGB category incorporated a database from Caltech101 that was simplified to only the following classes: 'Airplanes', 'Helicopter', 'Ferry', and 'Laptop' [47]. The Mel Frequency Cepstral Coefficients were based on instrumental sounds that were obtained from Kaggle for an audio tagging challenge [48]. The database was simplified to the following classes: 'Acoustic guitar', 'Clarinet', 'Flute', and 'Saxophones'.

For the RGB category, the airplanes class contained 800 images, the helicopter class contained 88 images, the ferry class contained 67 images, and the laptop class contained 81 images.

Overall, the max RGB input data for the CNN was 67 images due to the ferry class containing the lowest amount. This max input is then divided into training and testing images; 70% of the images from each class were set for training the machine learning model, and the remaining 30% were dedicated for the validation testing. This training and testing percentage was used for all the machine learning models in this thesis. It is important for the training images not to contain any of the validation images. The convolutional neural network was performed three times with an average validation accuracy of 99.58%. The convolutional neural network model of training and validation over time is shown in Figure 31. The confusion matrix shows the actual image vs. the predicted guess, as shown in Figure 32.



Results
Validation accuracy: 100.00%
Training finished: Reached final iteration

Validation
Frequency: 18 iterations
Patience: Inf

Figure 31: Practice RGB Training and Validation Over Time. The accuracy is 100% with very low loss. This would be the ideal model to achieve.

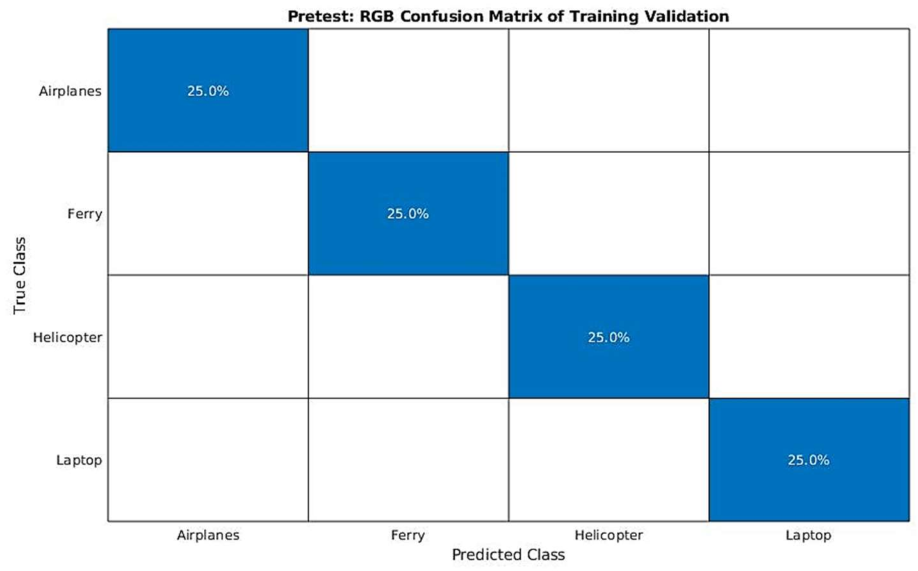
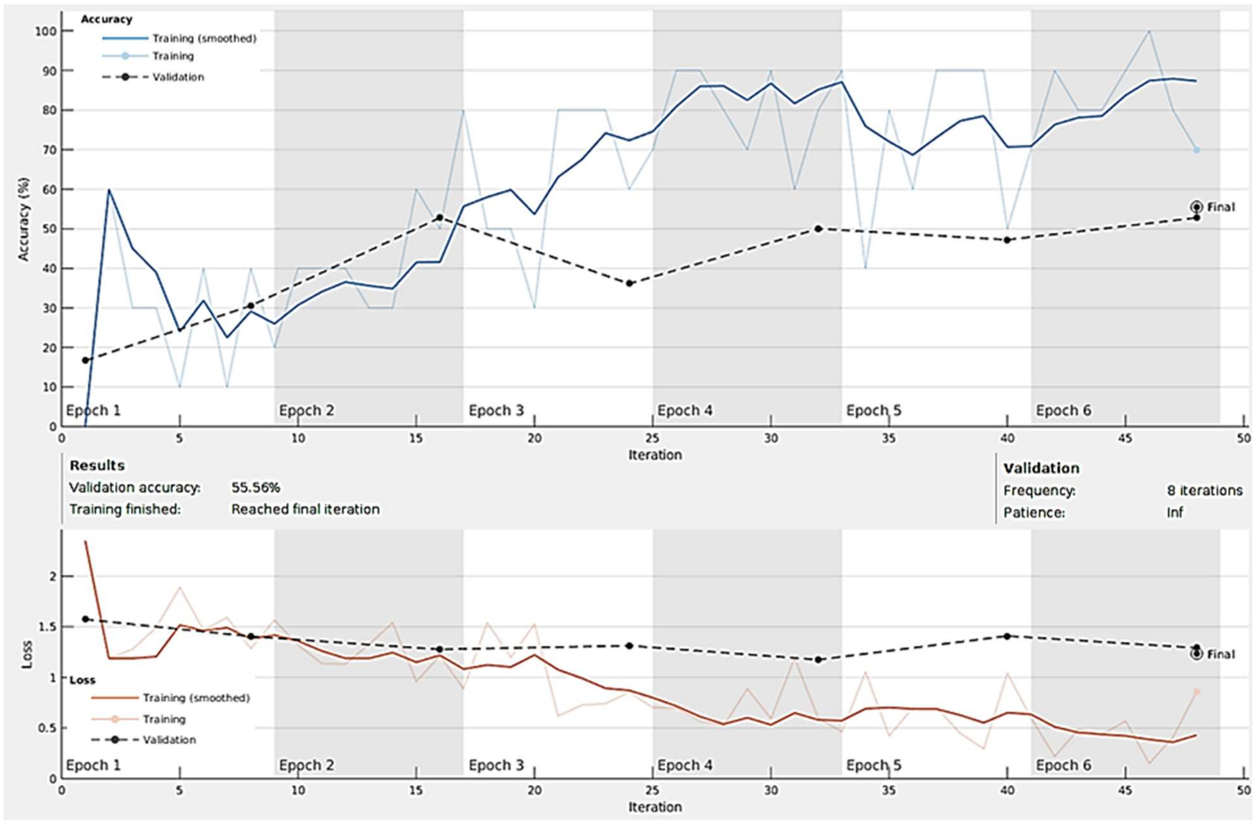


Figure 32: Practice RGB Confusion Matrix. This would be the ideal model to achieve. The blue represents correct predictions and the darker the blue the better the results, the white is neutral, and red is incorrect predictions.

To test the implementation with acoustic data, the wav files from each of the classes were processed as described above to obtain the Mel Frequency Cepstral Coefficient images. All the classes contained 29 wav files, setting that as the max input to the CNN. Unfortunately, this is not a lot of data, and the model was not expected to be reliant. As shown in Figure 33, the training accuracy is approximately 87%; however, the validation accuracy is 55.56%. This means that the test images had the 55.56% accuracy to the training model and that training accuracy and validation trendline gap represents the acoustic data being overfit and more data with variation would be needed. The convolutional neural network was performed three times; the maximum and average validation accuracy obtained was 55.56% and 50.93%, respectively. The confusion matrix of the validation performed can be seen in Figure 34. The acoustic sensor showed positive results for very low data. The validation accuracy between four different instruments with low data was more than twice as accurate than a random guess between the categories.



Results

Validation accuracy:	55.56%
Training finished:	Reached final iteration

Validation

Frequency:	8 iterations
Patience:	Inf

Figure 33: Practice Acoustic Training and Validation Over Time. The accuracy is 55.56% and the loss is high. This is not a good model to produce.

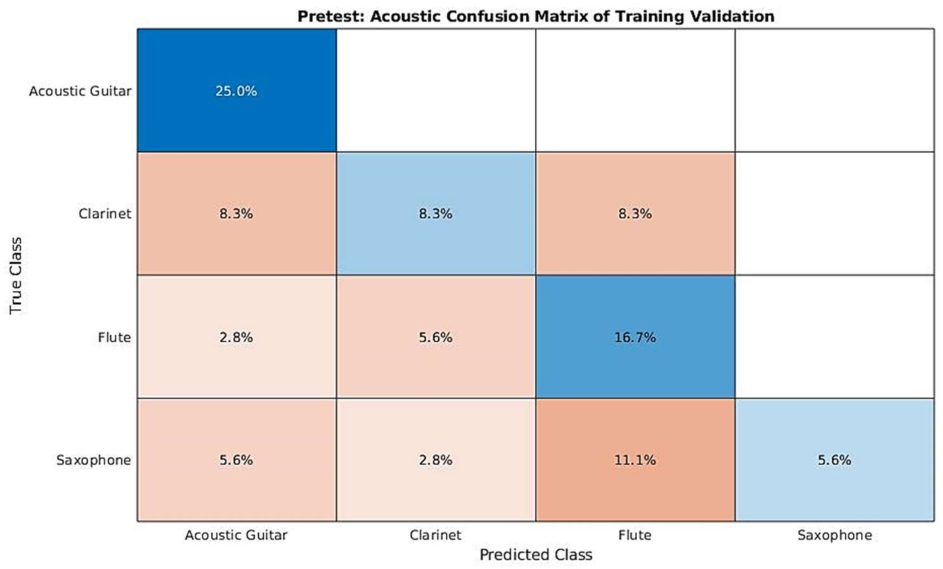


Figure 34: Practice Acoustic Confusion Matrix. There are much more shades of red in this model, which indicates more data is needed.

Machine Learning for Drone Classification

For our drone classification application, the goal is to train a model that will separate images into three classes: Parrot Quadcopter, Parrot Swing, and Tello Quadcopter. The Matlab implementation, shown in Appendix J, creates models for each sensor individually. To determine how performance depends on the amount of data collected from different locations, the process was performed for each location, adding data to the overall set with each new measurement location. For example, the first machine learning performance included the data from the first location, while the fourth performance included the data from the first, second, third, and fourth locations. The detailed amount of input data for each of the machine learning performances can be seen in Table 4. For the final machine learning process including all possible measurements, there will be a total of 1,056 RGB images, 1,042 thermal images, and 860 Mel Frequency Cepstral Coefficient images.

Table 4: Overall Machine Learning Process Inputs. The inputs of the first machine learning process are the data obtained from the first location, Process 2 would include the data obtained from the first and second location, and this repeats to Process 5 including the data obtained from all five locations. The maximum data input to the machine learning process is the minimum input data from the three drone classes.

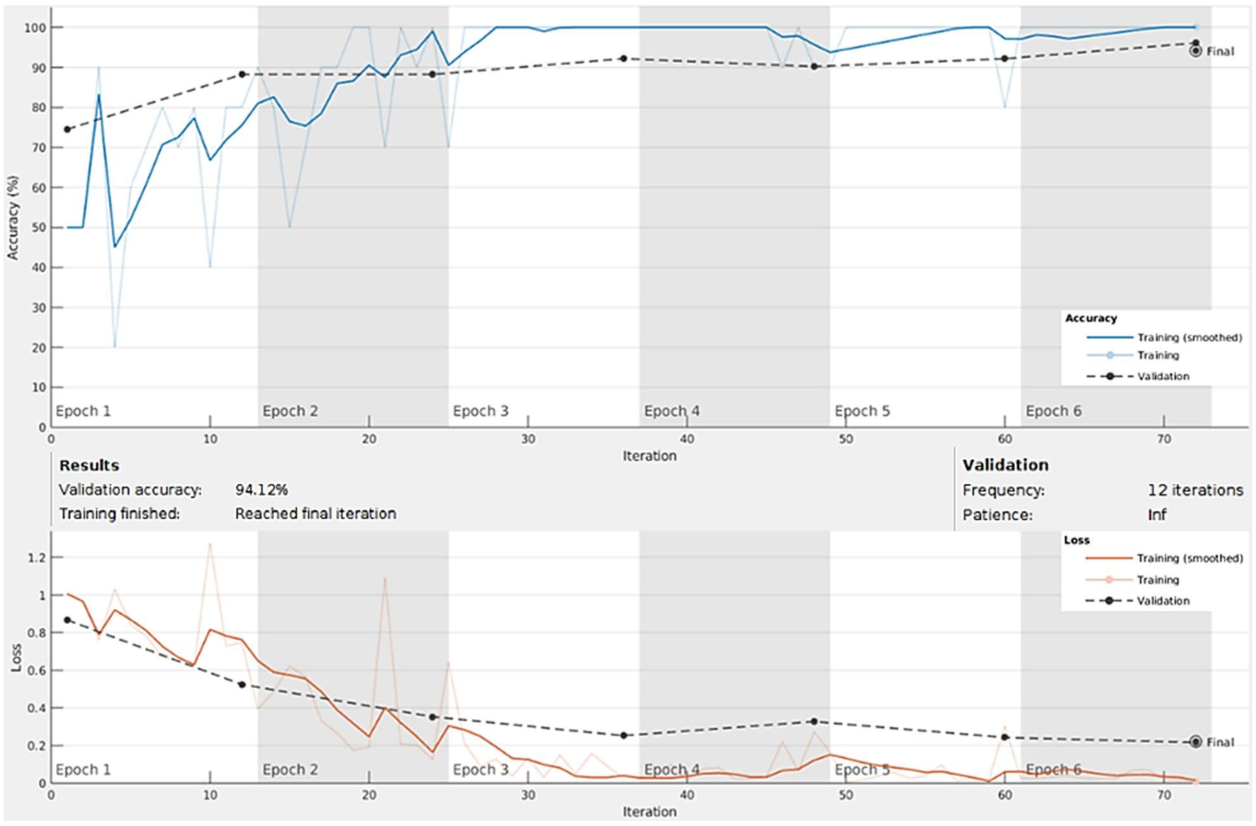
Machine Learning Inputs	**Parrot Quad.**	**Parrot Swing**	**Tello Quad.**	**Max Input**
Process 1 RGB	**132**	**60**	**155**	**60**
Process 2 RGB	**253**	**130**	**274**	**130**
Process 3 RGB	**530**	**653**	**573**	**530**
Process 4 RGB	**676**	**718**	**653**	**653**
Process 5 RGB	**992**	**1174**	**1056**	**992**
Process 1 Thermal	**128**	**56**	**132**	**56**
Process 2 Thermal	**248**	**117**	**247**	**117**
Process 3 Thermal	**498**	**556**	**542**	**498**
Process 4 Thermal	**652**	**618**	**624**	**618**
Process 5 Thermal	**980**	**1113**	**1042**	**980**
Process 1 Acoustic	**112**	**45**	**138**	**45**
Process 2 Acoustic	**234**	**109**	**254**	**109**
Process 3 Acoustic	**466**	**537**	**481**	**466**
Process 4 Acoustic	**597**	**594**	**547**	**547**
Process 5 Acoustic	**959**	**757**	**860**	**757**

These inputs are inputted into the deep convolutional neural network and trained. Due to the set limit, the images inserted into the machine learning program are random. This results in a different validation accuracy. The validation accuracy is determined by taking the input data and having 70% of the data train and 30% of the data tested on the trained data. However, two data samples from each of the sensors and each drone at each location were removed from the database and placed in a separate database for a second validation test. Unlike the first validation test, this second validation shall remain consistent and given to each process for comparison and when this second validation test is performed, the machine learning algorithm will be classifying data from locations it has not learned yet. These data were mostly selected randomly. The RGB and thermal images are the same, however the acoustic were different. The only data samples that were not considered for this second database were when the drone exceeded 50 ft. or when the drone was not visible to the human eye. There was an image where only the drone wing was visible in the RGB and thermal images that were in the second database. As a result, there are ten images of each RGB, thermal, and Mel Frequency Cepstral Coefficients for each drone in the new test database. This second validation test is referred to as the Post-Training Test. Each process was performed three times for all sensor, and the validation accuracy was compared for all three. The max training validation accuracy results achieved for each process can be seen in Table 5. The three evaluations of all the machine learning processes performed are shown in Tables C-1 - C-5.

Table 5: Overall Machine Learning Validation Accuracy. The machine learning process is performed with each set of the input data and is performed three times to compare the accuracy. The results below represent the max validation accuracy achieved from the machine learning process.

Validation Accuracy (%)	**RGB**	**Thermal**	**Acoustic**
Process 1	**94.12**	**66.67**	**66.67**
Process 2	**75.44**	**72.55**	**58.06**
Process 3	**96.60**	**82.88**	**71.26**
Process 4	**78.35**	**62.66**	**60.49**
Process 5	**60.23**	**43.07**	**46.58**

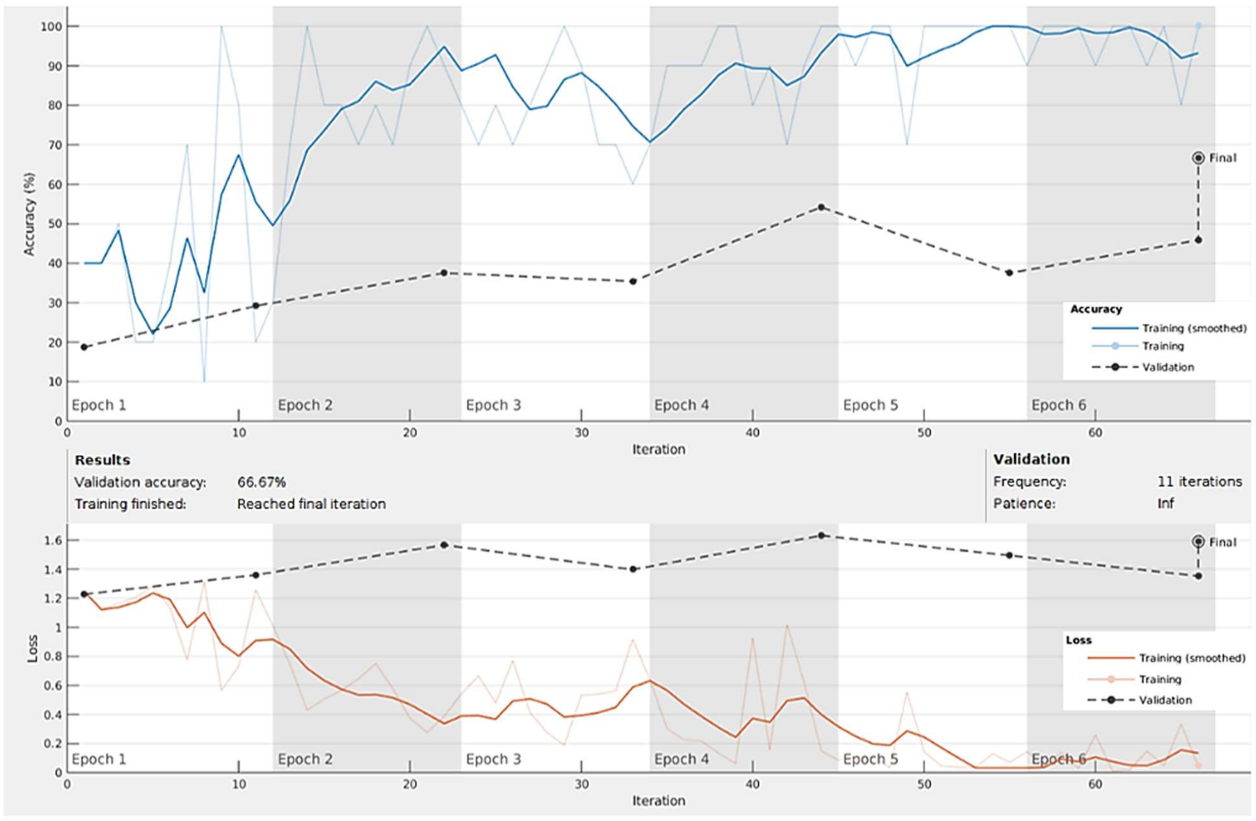
Despite the low amount of data obtained from the first location, the machine learning program still performed well. The lowest amount of data received was the Parrot Swing in all three categories. Therefore, the machine learning used a max input number of 58 images for the RGB, 54 images for the thermal, and 43 images from the acoustics, subtraction of two per category due to the creation of the second validation test. Due to the fact that Parrot and Tello quadcopters contained data samples over the max input limit, the images that were inputted were randomly selected to obtain that max number. In Matlab, the function splitEachLabel(dataset, max input limit, 'randomize') would change the dataset to the max input limit by filling that amount with randomly selected data images from the dataset. The results of the first iteration were very good for the RGB category and showed the other categories had room to improve. The time duration of the machine learning task for each of the categories, RGB, thermal, and acoustic, was 4 minutes and 34 seconds, was 4 minutes and 11 seconds, and 3 minutes and 26 seconds, respectively. Even though the data was very low, the machine learning program was able to show potential. The machine learning program required a greater amount of data than provided for the first location, as well as different scenery and noise to prevent over-fitting the data. The more data that gets added, the longer it takes to finish the machine learning process. The first machine learning process performed in each category can be seen in Figures 35 - 37.



Results
Validation accuracy: 94.12%
Training finished: Reached final iteration

Validation
Frequency: 12 iterations
Patience: Inf

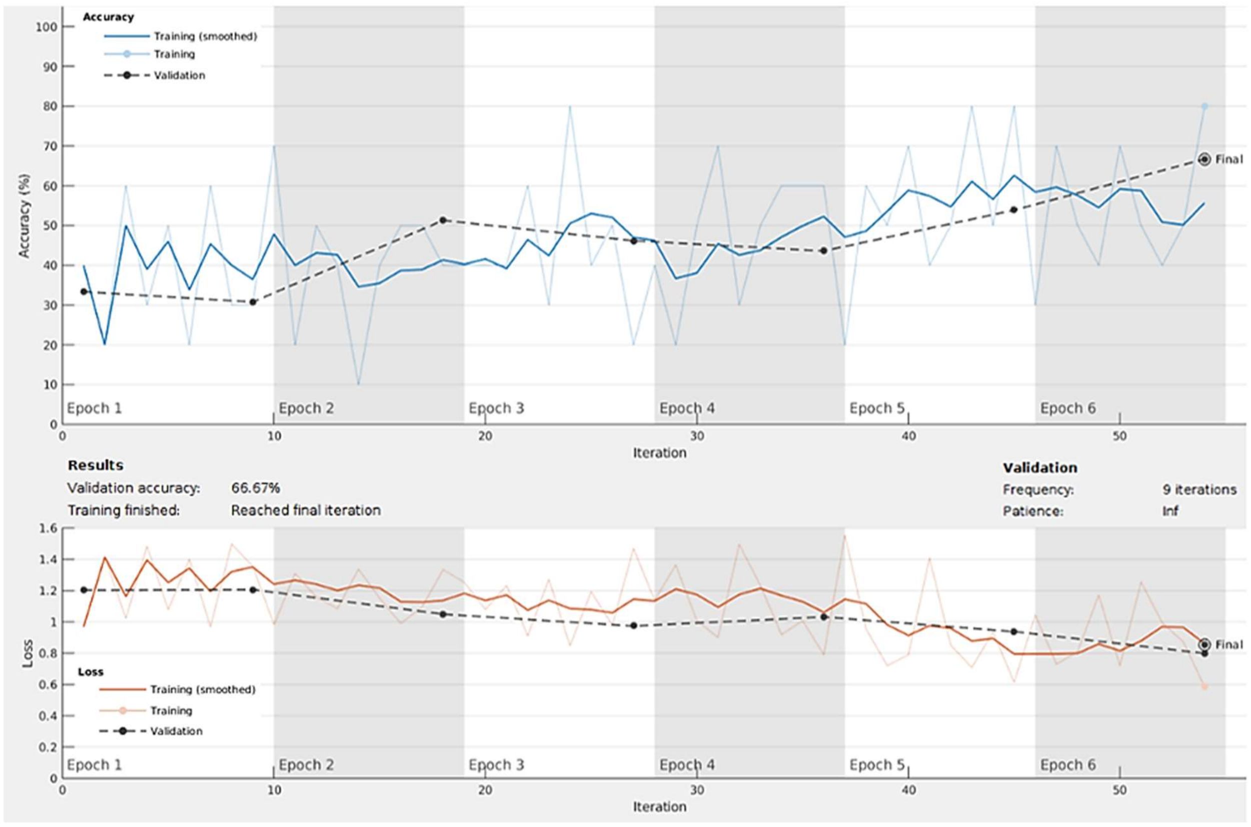
Figure 35: RGB Machine Learning Process Up to the First Location.



Results
Validation accuracy: 66.67%
Training finished: Reached final iteration

Validation
Frequency: 11 iterations
Patience: Inf

Figure 36: Thermal Machine Learning Process Up to the First Location.



Results
Validation accuracy: 66.67%
Training finished: Reached final iteration

Validation
Frequency: 9 iterations
Patience: Inf

Figure 37: Acoustic Machine Learning Process Up to the First Location.

The confusion matrix for the training of the above categories can be seen in Figures 38 - 40. After the convolutional neural network is trained, 30 test images for each category, ten from each location, were given to the network to classify. The confusion matrix for post-training can be seen in Figures 41 - 43.

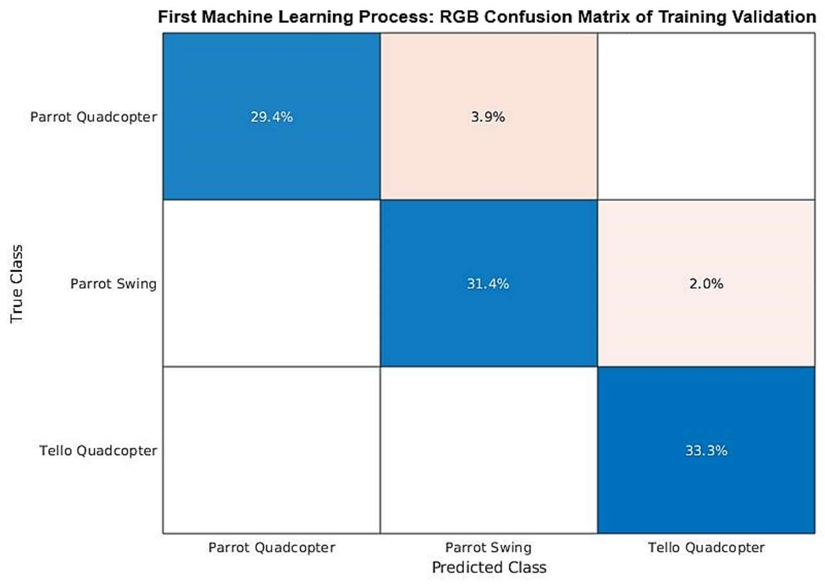


Figure 38: Process 1 RGB Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

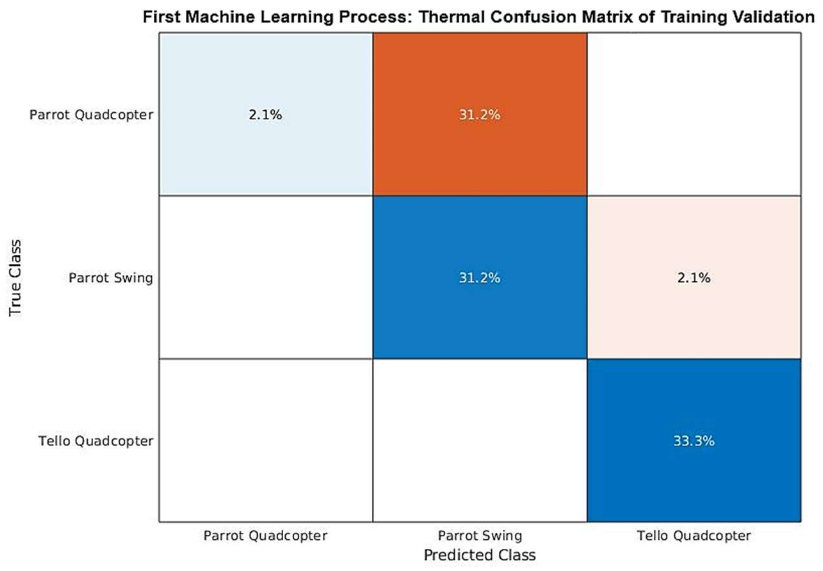


Figure 39: Process 1 Thermal Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

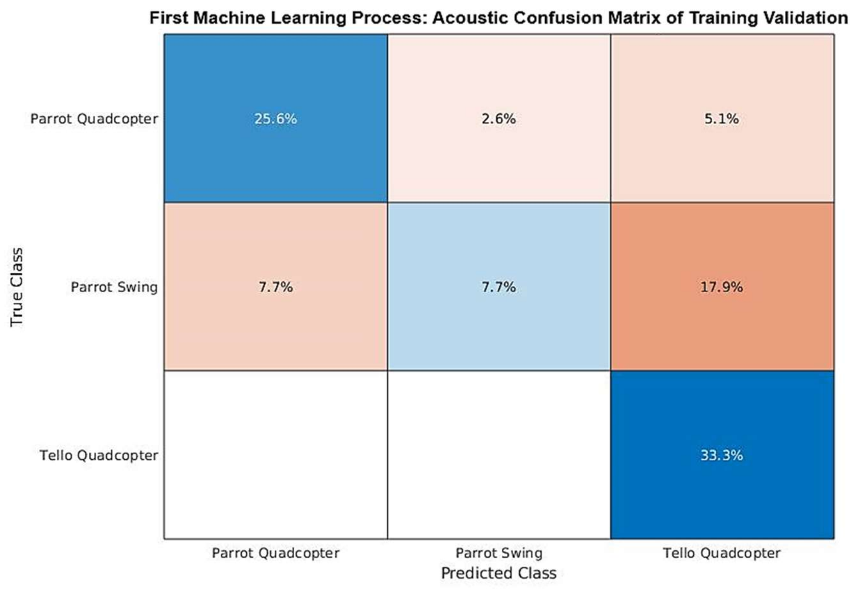


Figure 40: Process 1 Acoustic Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

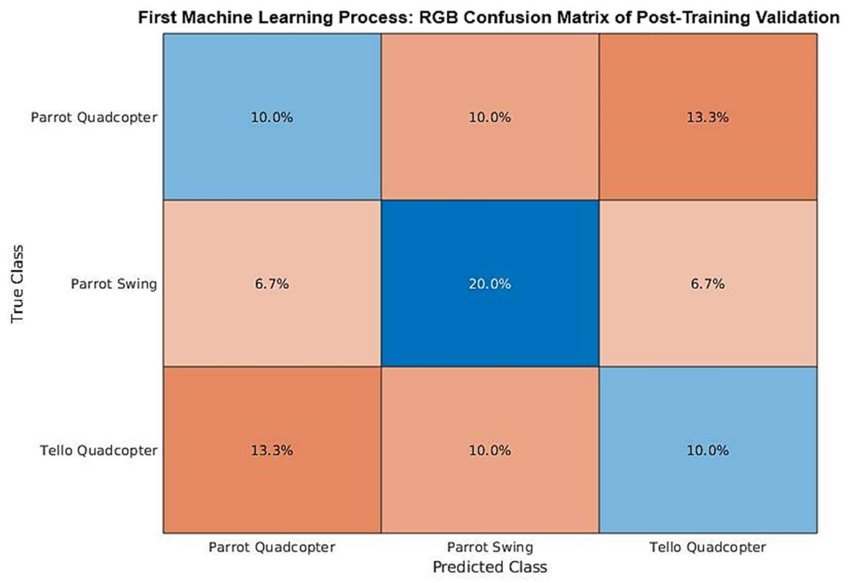


Figure 41: Process 1 RGB Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

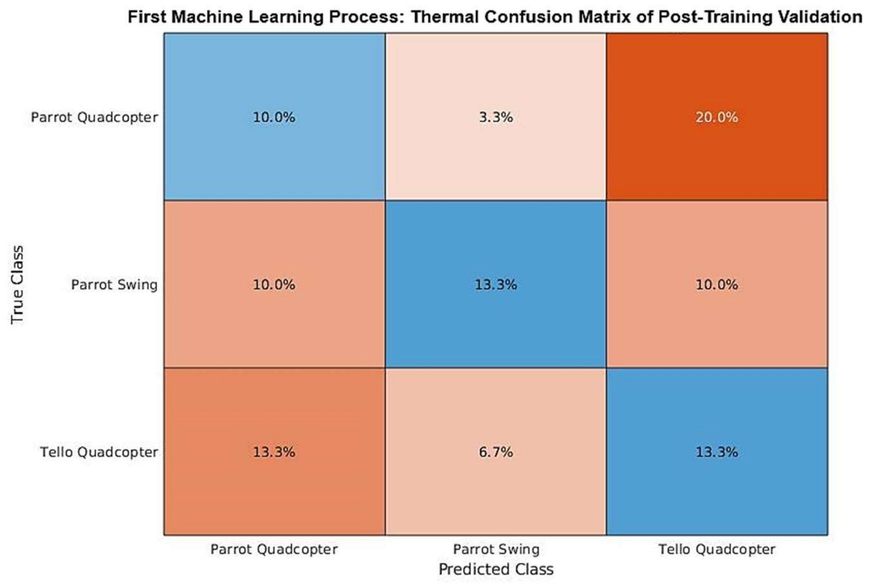


Figure 42: Process 1 Thermal Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

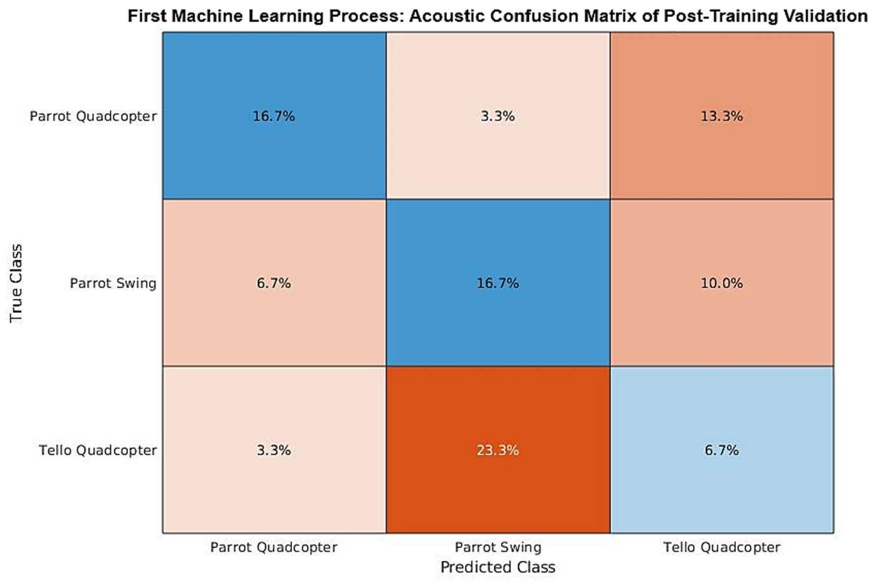
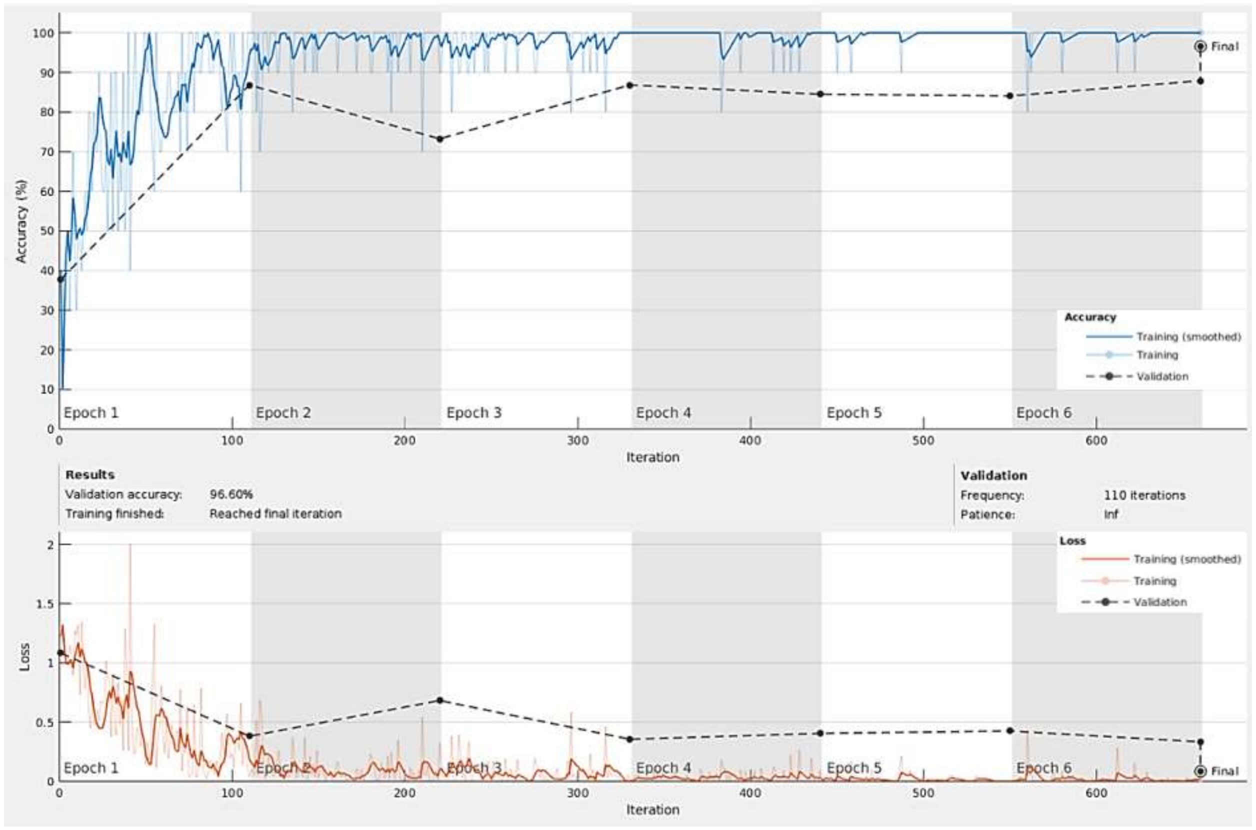


Figure 43: Process 1 Acoustic Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

With the data for the second location being merged with the data from the first, the input data is now more than twice the amount of the previous machine learning performance. The limitation was again the Parrot Swing, allowing the max input data to be 126 RGB images, 113 thermal images, and 105 Mel Frequency Cepstral Coefficient images. The validation accuracy for the RGB category decreased by 18.68 percentage points, the thermal category increased by 5.88 percentage points, and the acoustic category decreased by 8.61 percentage points. The time duration of the machine learning process was 9 minutes and 37 seconds, 8 minutes and 30 seconds, and 8 minutes and 11 seconds for the RGB, thermal, and acoustic categories, respectively. The graphs of the second machine learning processes can be seen in Figures D1 - D3, and the confusion matrices during the training and post-training can be seen in Figures D-4 - D-9.

The validation accuracy of the RGB and the acoustic data decreased; however, this is a positive outcome because it shows a more realistic model representation compared to the first performance due to the increase of the input data by more than double the amount. This increase in the amount of data and different variations makes the machine learning program more likely to detect drones in other environments. The thermal validation accuracy increased to be about as reliable as the RGB camera. As with the previous performance, more data and noise are needed to be able to obtain a reliable representation of the model. The likelihood of increasing the accuracy of the machine model validation accuracy with more data and noise is a strong possibility with only a hundred data points.

With the data from the third location merged with the data from the other location, the input data is around four times larger than the previous process. The Parrot Quadcopter is the limiting class for all three categories. The max number of input data for the RGB category was 524 images, the thermal category was 492 images, and the acoustic category was 460 images. This machine learning process was excellent overall due to the amount of data and the overall performance increased with each sensor. The validation accuracy for the RGB category increased by 21.16 percentage points, the thermal category by 10.33 percentage points, and the acoustic category by 13.20 percentage points, compared to the previous performance. The third machine learning processes' training and loss graphs can be seen in Figures 44-46.



Results
Validation accuracy: 96.60%
Training finished: Reached final iteration

Validation
Frequency: 110 iterations
Patience: Inf

Figure 44: RGB Machine Learning Process Up to the Third Location.

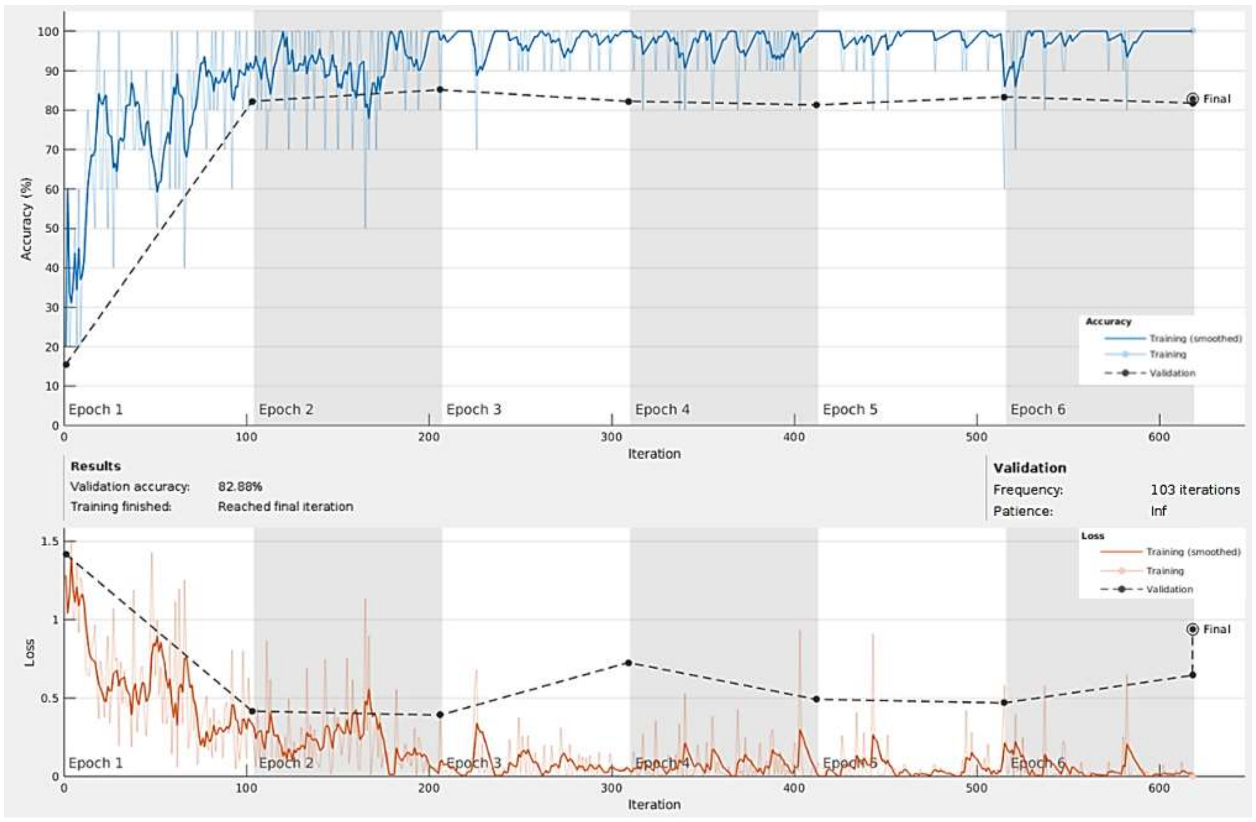


Figure 45: Thermal Machine Learning Process Up to the Third Location.

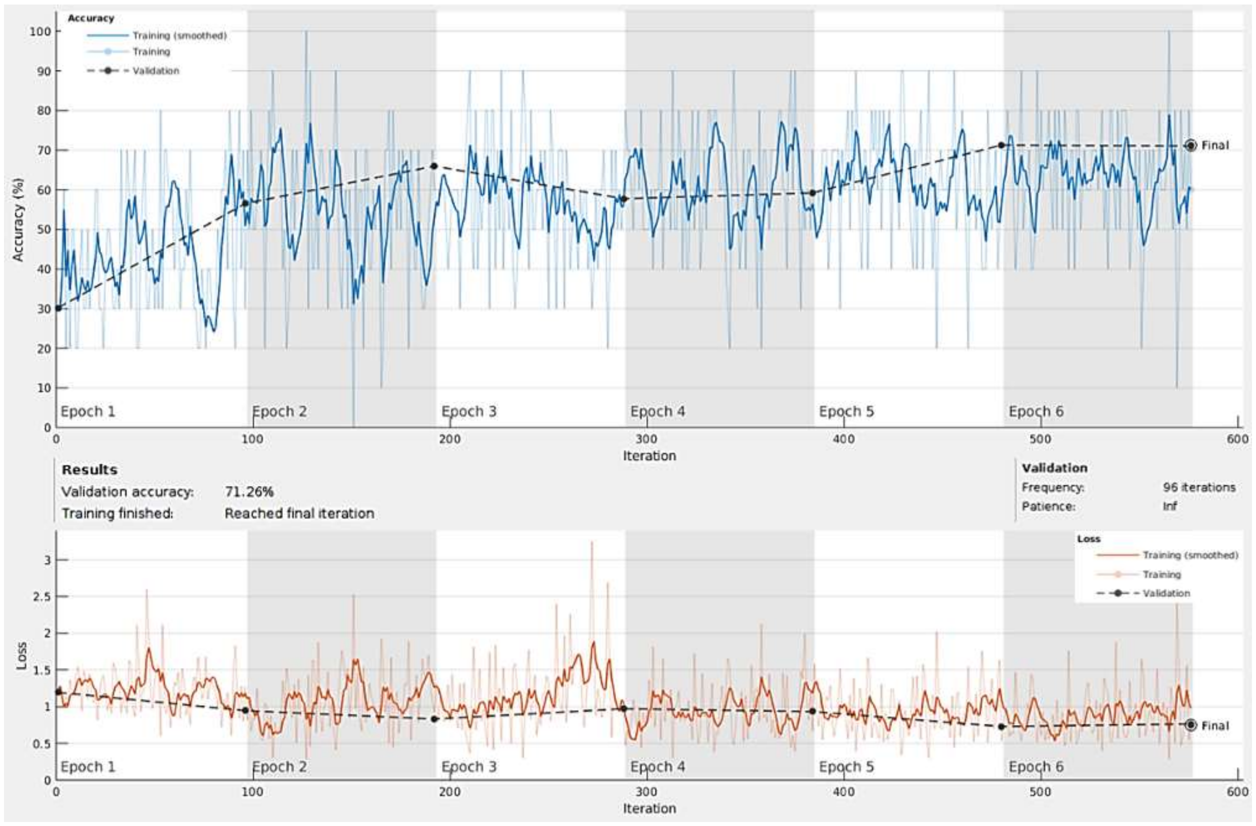


Figure 46: Acoustic Machine Learning Process Up to the Third Location.

According to the post-training image test, the RGB model and thermal category did well with the second dataset of images, in which slightly less than half are drones in a different scenery not seen before. This is an accurate model for the drone under 50 ft. The time duration for the completion of the process was 44 minutes and 13 seconds, 37 minutes and 51 seconds, and 36 minutes and 58 seconds for the RGB, thermal, and acoustic categories, respectively. The third machine learning the confusion matrices during the training and post-training can be seen in Figures 47-52.

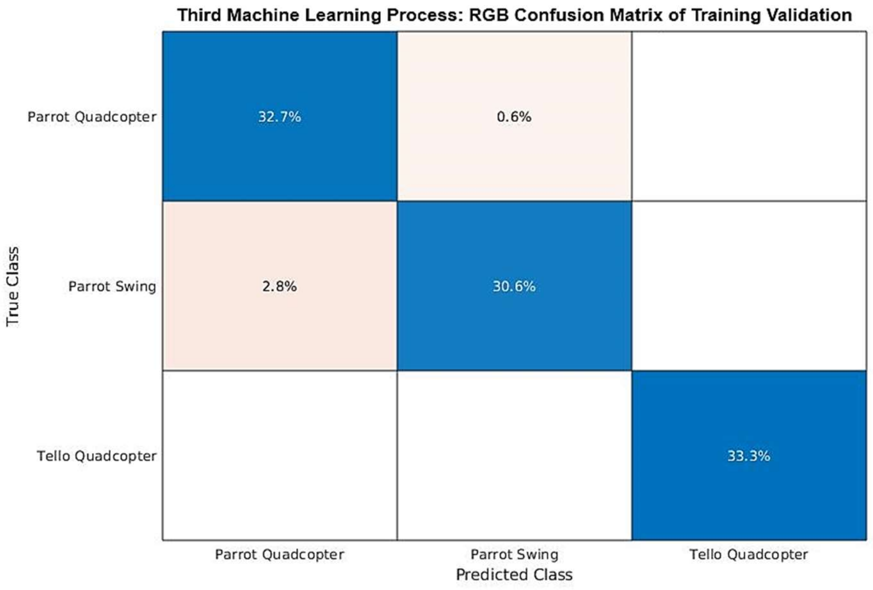


Figure 47: Process 3 RGB Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

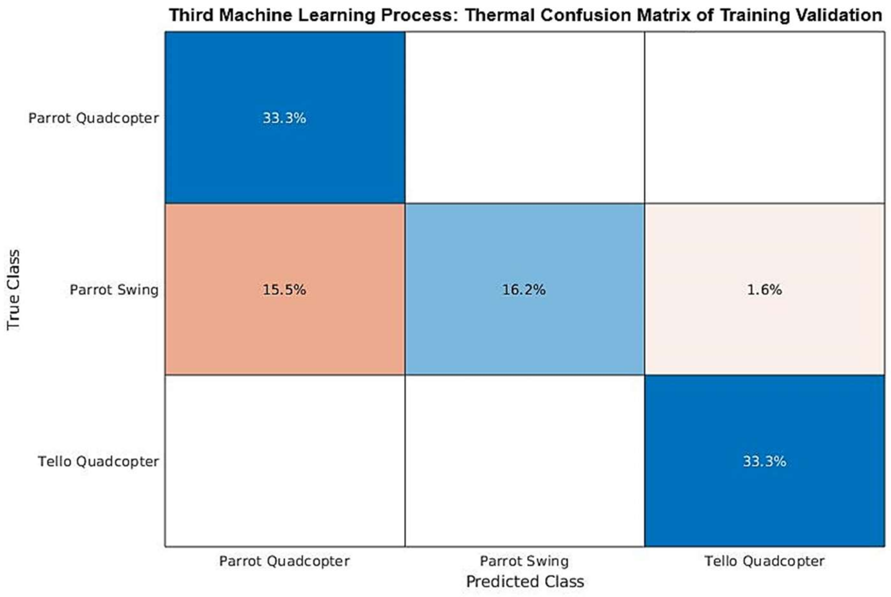


Figure 48: Process 3 Thermal Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

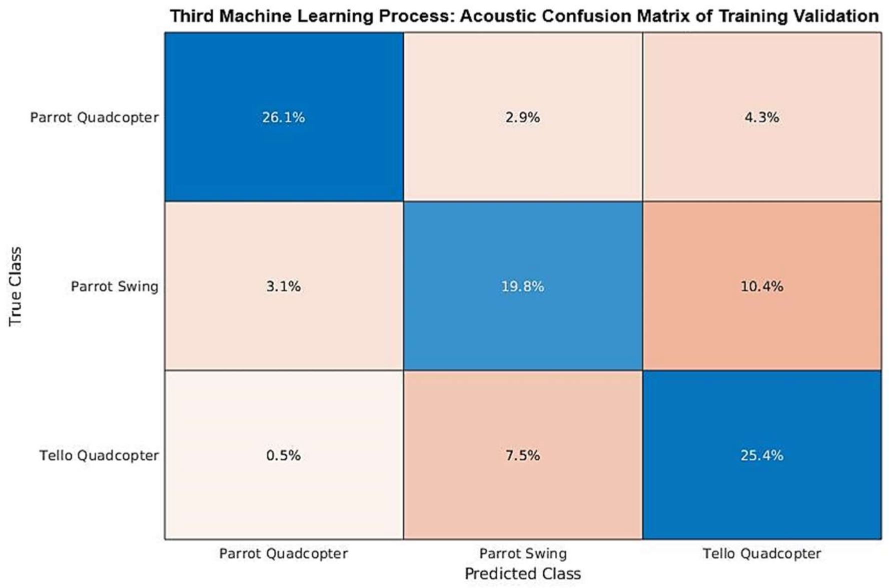


Figure 49: Process 3 Acoustic Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

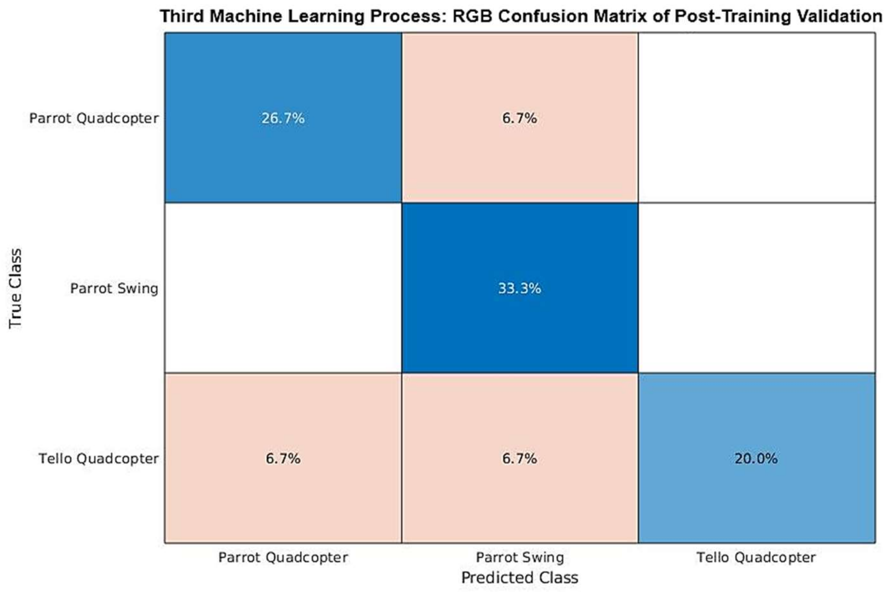


Figure 50: Process 3 RGB Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

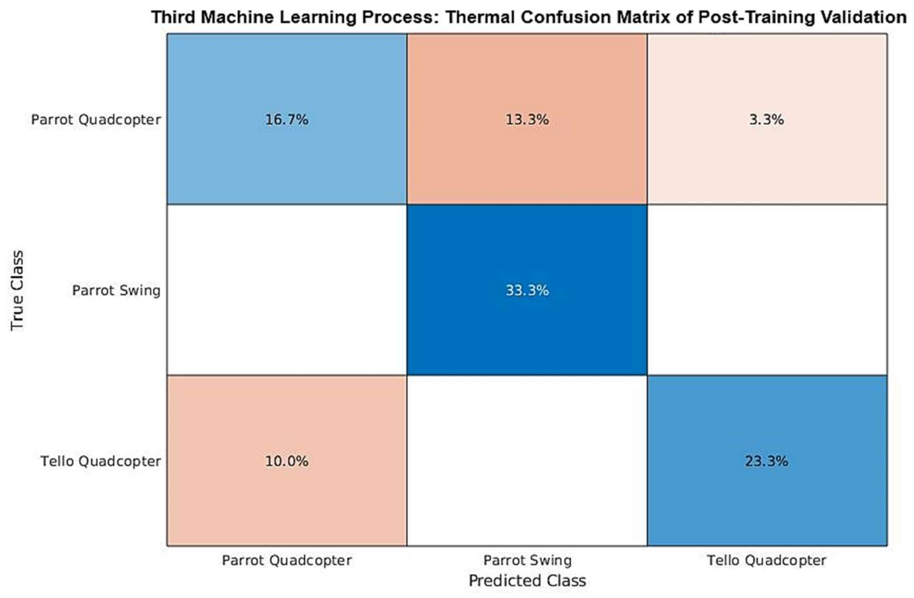


Figure 51: Process 3 Thermal Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

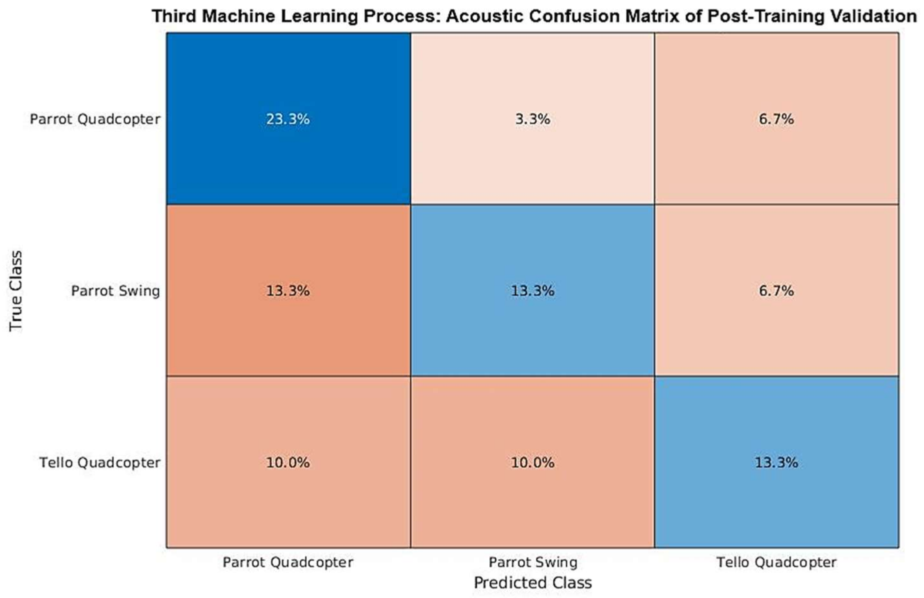


Figure 52: Process 3 Acoustic Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

The data obtained from the fourth location was merged with the data from the previous locations, and the max input data was increased with the Tello Quadcopter setting the input limit for each category. The max input data for the RGB category was 645 images, the thermal category was 616 images, and the acoustic category was 539 images. The validation accuracy for the RGB category was decreased by 18.25 percentage points, the thermal category decreased by 28.96 percentage points, and the acoustic category decreased by 10.77 percentage points when compared to the previous performance. The total time for each of the categories to perform was 50 minutes 11 seconds, 47 minutes and 28 seconds, and 41 minutes and 47 seconds for the RGB, thermal, and acoustic categories, respectively. Even though the drones were exposed to new variations, they were also exposed to the limitations of the sensors. The distance from the sensors to the fence was 70 ft., and the drone almost went over it multiple times. It is also possible that the input data involves the limitations of the sensors captured in the previous iterations as well. The graphs of the fourth machine learning processes can be seen in Figures E-1 - E-3, and the confusion matrices during the training and post-training can be seen in Figures E-4 - E-7.

The fifth and final performance was merging and inputting all the data collected into the machine learning program. The limit of the input data for each of the categories was set by the Parrot Quadcopter setting the max input for the RGB category at 982 images, thermal category at 970 images, and acoustic category at 747 images. As expected, the new data has made the program unreliable. The validation accuracy has decreased in all categories: the RGB category by 18.12 percentage points, the thermal category by 12.00 percentage points, and the acoustic category by 20.91 percentage points when compared to the previous performance. The total time for the categories to complete the performance was 78 minutes and 57 seconds, 77 minutes and 22 seconds, and 60 minutes and 52 seconds for the RGB, thermal, and acoustic category, respectively.

The times for each category to complete for all the processes can be seen in Table F-1. The graphs of the fifth machine learning processes can be seen in Figures 53 - 55.

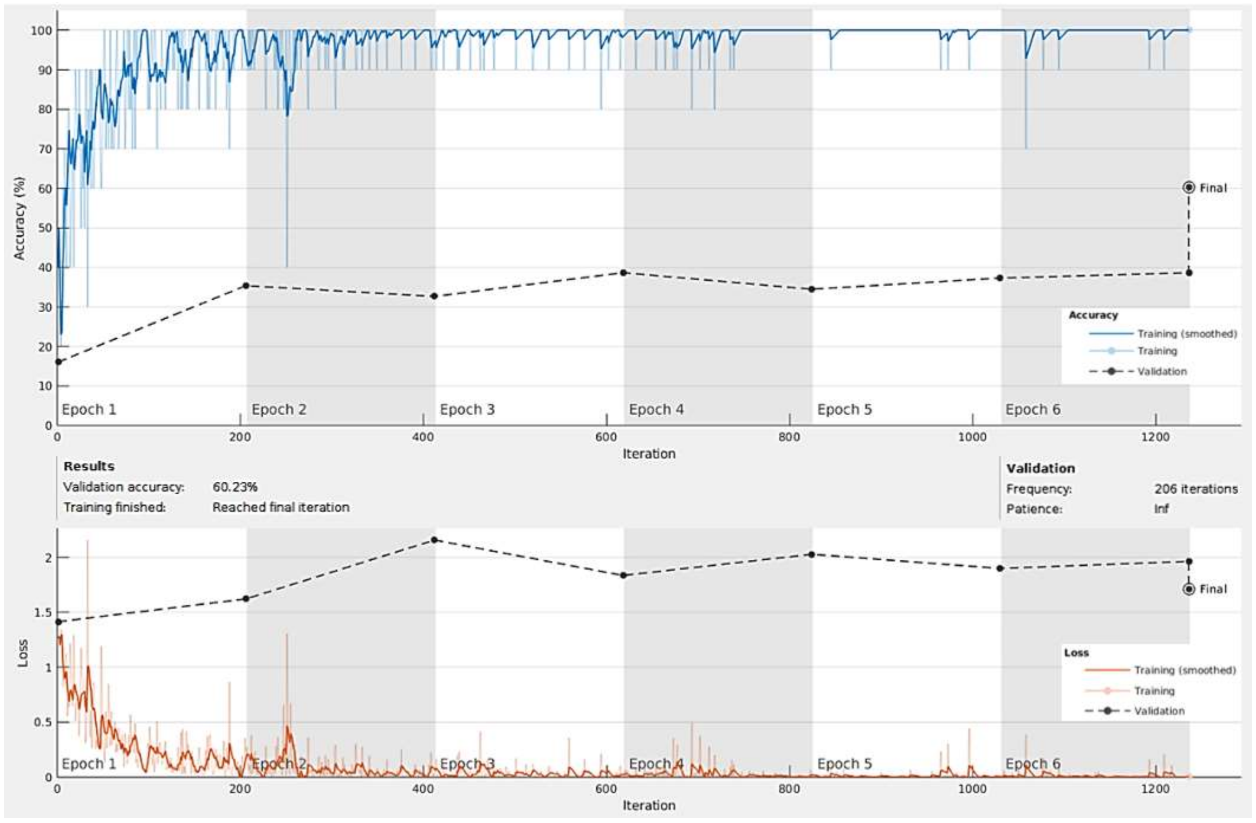
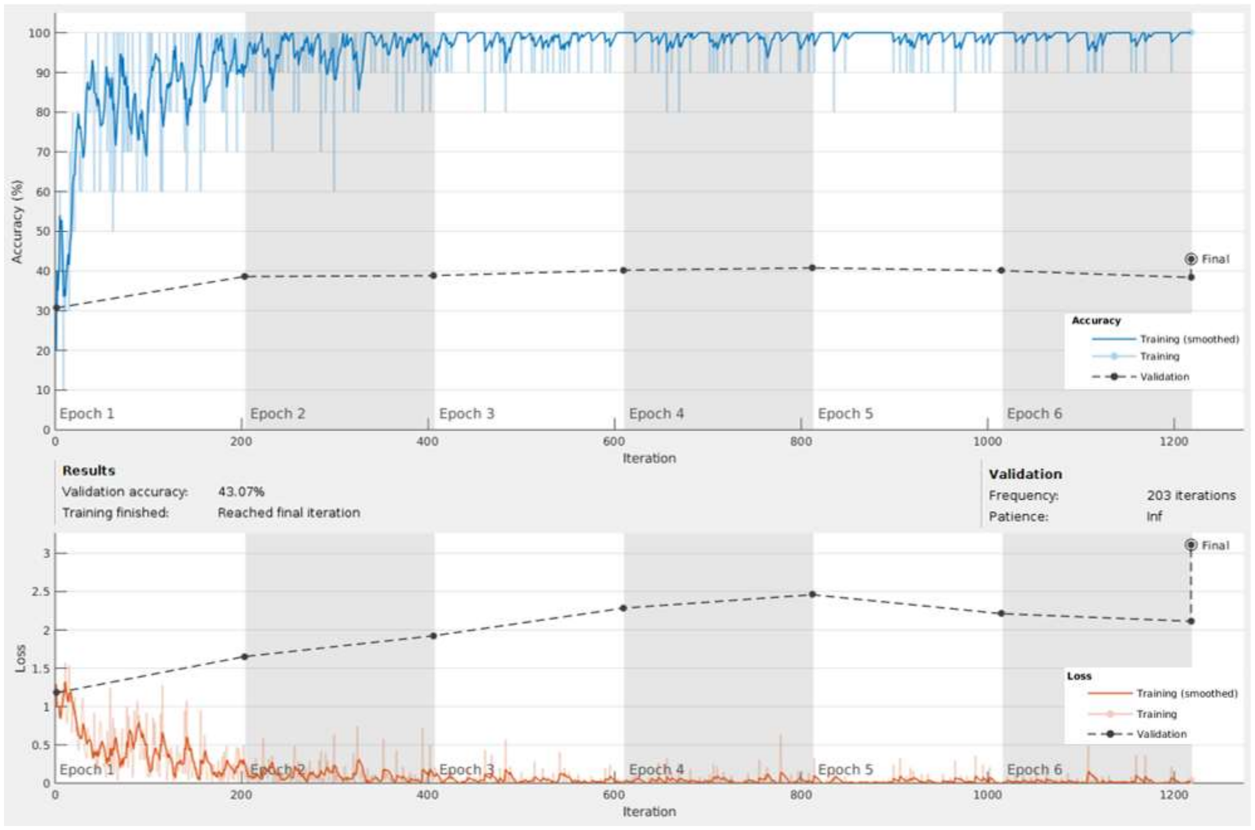


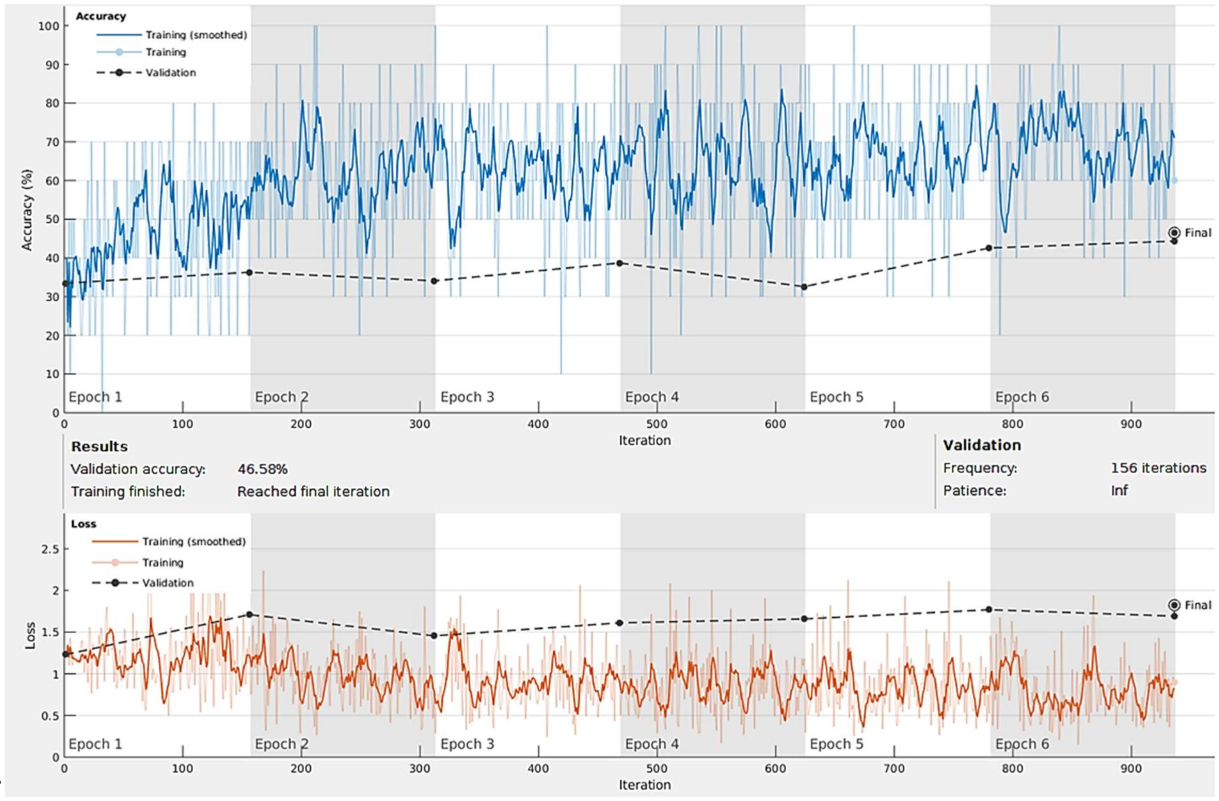
Figure 53: RGB Machine Learning Process Up to the Fifth Location.



Results
Validation accuracy: 43.07%
Training finished: Reached final iteration

Validation
Frequency: 203 iterations
Patience: Inf

Figure 54: Thermal Machine Learning Process Up to the Fifth Location.



Results
Validation accuracy: 46.58%
Training finished: Reached final iteration

Validation
Frequency: 156 iterations
Patience: Inf

Figure 55: Acoustic Machine Learning Process Up to the Fifth Location.

The confusion matrices during the training and post-training can be seen in Figures 56 - 61. Since there are three classes, a random guess is 33.33%, so this performance is still better than randomly picking the drone in all categories. There is a noticeable difference in the accuracy when comparing the training and post-confusion matrices. As previously stated, the second dataset was selective to a degree and remained constant to help indicate if there were limitations in the sensors. The poor accuracy in the confusion matrix based on the primary database and significantly better accuracy in the second database is a decent indication of the limitations are met in all sensor categories.

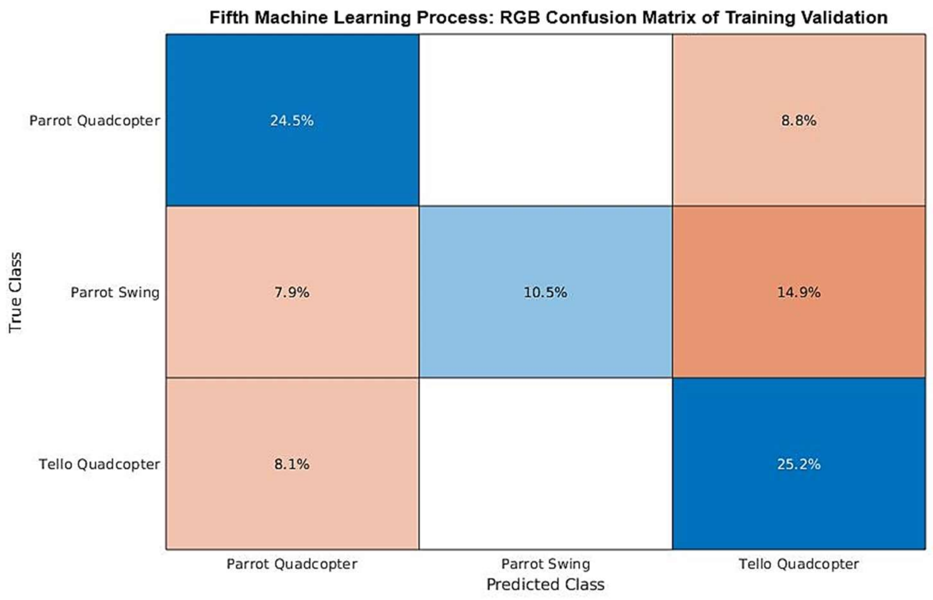


Figure 56: Process 5 RGB Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

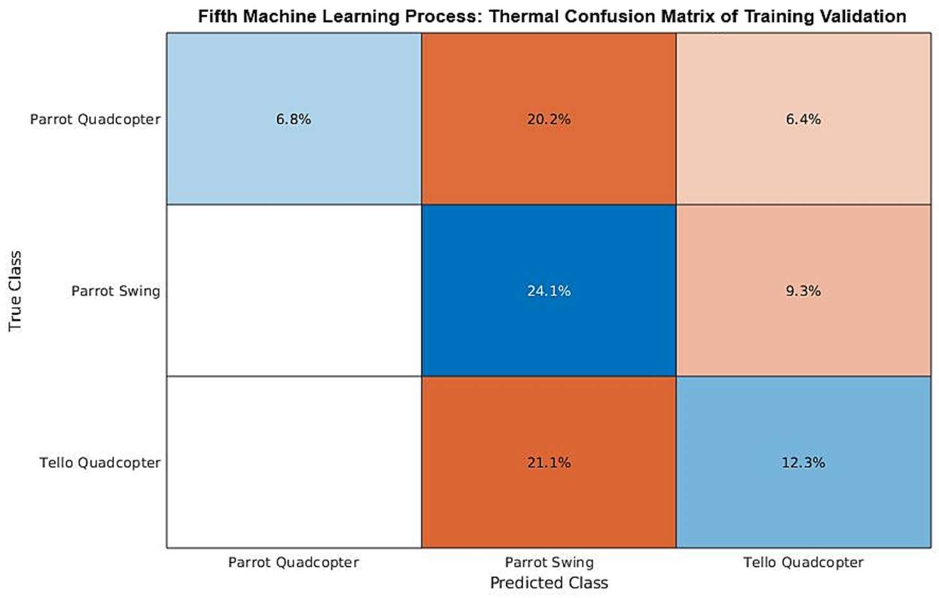


Figure 57: Process 5 Thermal Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

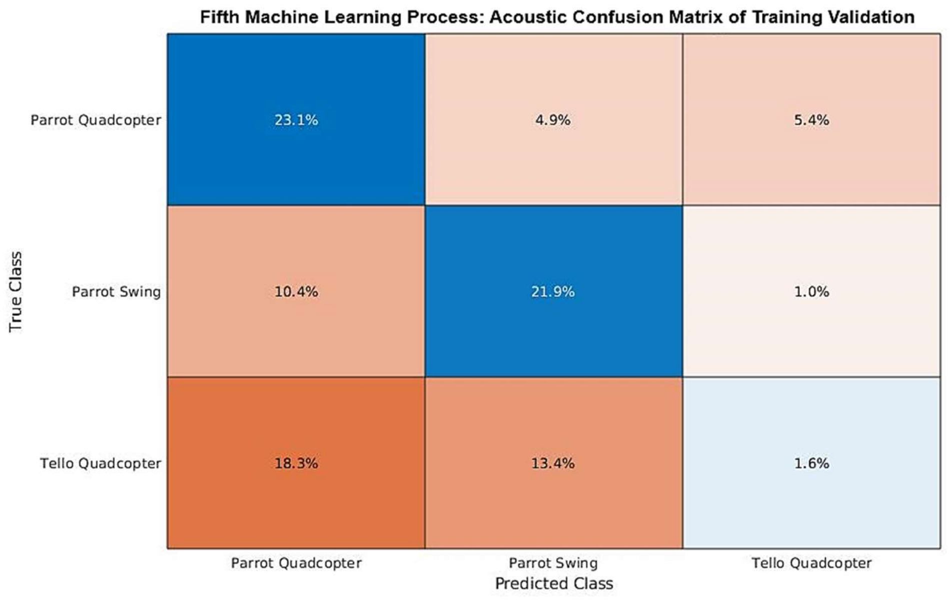


Figure 58: Process 5 Acoustic Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

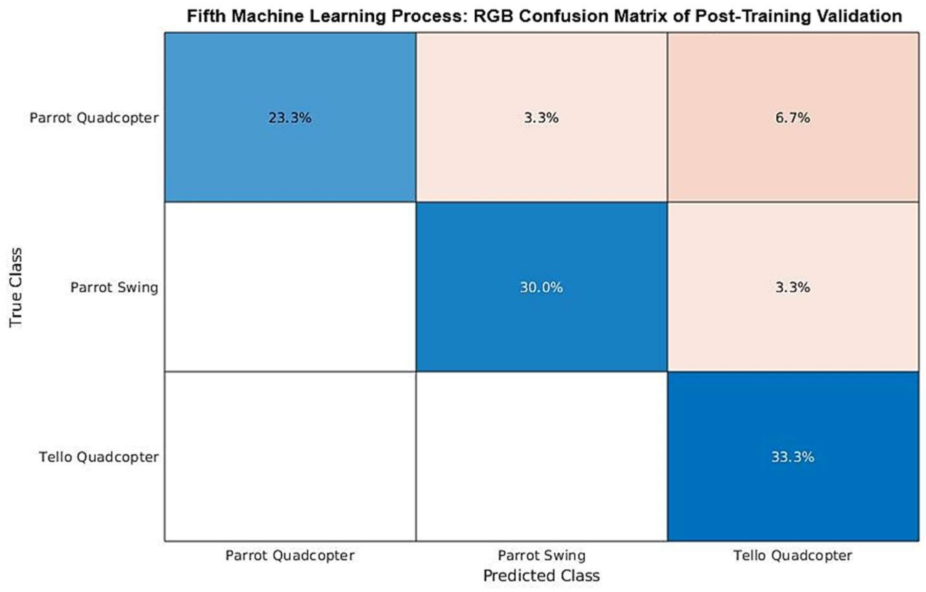


Figure 59: Process 5 RGB Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

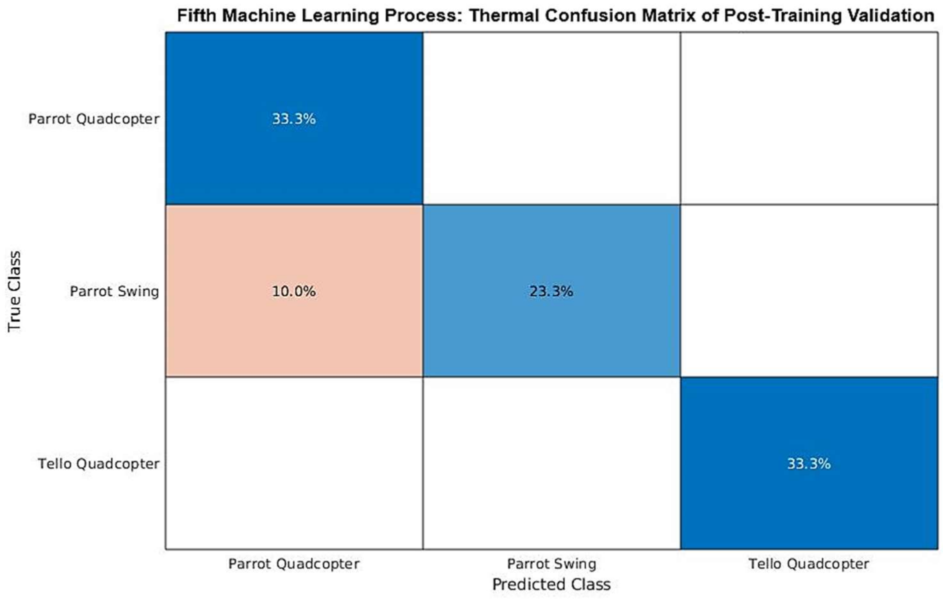


Figure 60: Process 5 Thermal Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

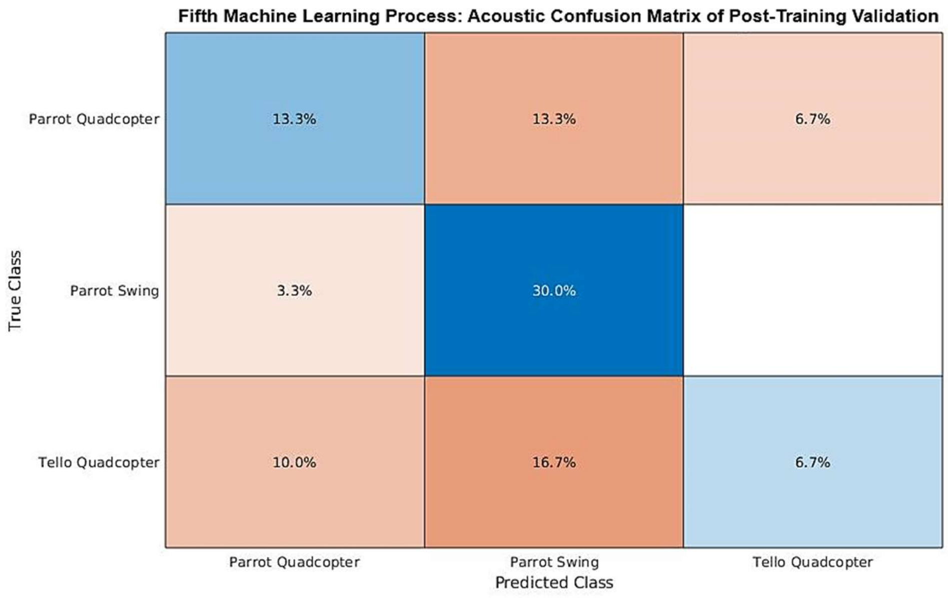


Figure 61: Process 5 Acoustic Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

Modified Machine Learning Accuracy and Loss

An adjustment was made to the overall program, and more machine learning processes were performed to validate that the limitations of the acoustic sensor were due to the low validation accuracy throughout the processes. The adjustment performed was narrowing the drone classes from the Parrot Swing, Parrot Quadcopter, and Tello Quadcopter to the Parrot Swing and Quadcopters. The Quadcopters were a combined data sample of the Parrot Quadcopter and Tello Quadcopter. Since there are only two classes, the random guess between classes increased from 33.33% to 50%. The acoustic category had a validation accuracy of 75.16% for Process 3, 79.83% for Process 4, and 64.56% for Process 5. The loss was approximately 0.49, 0.47, and 0.75 for Process 3, 4, and 5, respectively. These performances are shown in Figures G-1 - G-3. The confusion matrices for these are shown in Figures G-4 - G-9.

The results of the two-classes compared to the three-classes can be seen in Table 6. Since the random guess would differ greatly between the two systems, the validation accuracy should be by the system's random guess for comparison. When factoring in the random guess difference, the data shows that the best performance is three classes at Process 3 by 12.77 percentage points. However, with the increase in difference, the two-class system performs better than the three-class system.

Table 6: Acoustic Two-Classes VS Three-Classes. This table shows the comparison between the two classes of Parrot Swing and Quadcopters to the three classes Parrot Swing, Parrot Quadcopter, and Tello Quadcopter for Process 3, 4, and 5. The Process Random Difference is the validation accuracy subtracted by the random guess to compare the difference in accuracy realistically.

Machine Learning Inputs	**Two Classes**	**Three Classes**
Process 3 Validation Accuracy (%)	**75.16**	**71.26**
Process 4 Validation Accuracy (%)	**79.83**	**60.49**
Process 5 Validation Accuracy (%)	**64.56**	**46.58**
Random Guess (%)	**50.00**	**33.33**
Process 3 Random Difference (%)	**25.16**	**37.93**
Process 4 Random Difference (%)	**29.83**	**27.16**
Process 5 Random Difference (%)	**14.56**	**13.25**

Chapter 6. Conclusion

Project Overview

The beginning stage of this thesis involved the creation of a robot that would be able to collect data from the drones. The robot was designed in Solidworks, assembled, and wired to be able to be mobile and fully use the sensors. The graphical user interface, utilizing the RGB camera, the thermal camera, and the acoustic sensor, was programmed in Python and operational on the Minisforum minicomputer. The data collecting program was then compacted to this minicomputer to achieve as close to real-time data as achievable. An additional Python program was created to gather the data collected and safely transfer the data to the machine learning computer and perform the audio processing on the acoustic wav files as well. The program for the machine learning program was then created in Matlab to detect and classify the three drone classes, Parrot Swing, Parrot Quadcopter, and Tello Quadcopter using the data collected as inputs.

The main priority after the beginning stage of the project was to increase the number of samples of data collected that were inputted to each sensor's convolutional neural networks. This ideally gave the convolutional neural networks the flexibility to obtain a higher validation accuracy. The samples were collected in different locations, contained moving backgrounds, and obtained random noise. Some of these moving backgrounds and random noises included vehicles, clouds, people, lawnmowers, construction work, and people. The second priority was to determine the limitations of the sensors through the data collection and machine learning process. This was done by using the different samples and evaluating the results of the validation accuracy and loss.

Project Results

The convolutional neural network was successfully able to classify and detect the three classes of drones using the three categories RGB, thermal, and acoustic. The images that were obtained in the data collection were inputted into the machine learning program and revealed that the most accurate sensor was the RGB camera, followed by the acoustic sensor, and then the thermal camera. However, if the drone is beyond 50 ft., the most reliable prediction model must have the two quadcopters merge into a single class and rely on the acoustic and RGB sensors, followed by the thermal camera. When the majority of the data was within 50 ft. and had approximately 500 input data samples, the RGB camera had a maximum of 96.6% validation accuracy recorded with the three drone categories. In comparison, the thermal camera and acoustic sensor had a maximum validation accuracy of 82.9% and 71.3%, respectively, with the same conditions. When the CNN incorporated much more data of the drone over 50 ft. from the sensors and exposed to more limitations, the maximum validation accuracy recorded was 60.2% for the RGB category, 43.1% for the thermal category, and 46.58% for the acoustic category. However, when the consistent input data from the second database was used to perform the post-validation test, the post-training confusion matrix was very accurate. The input data from the primary database is significantly larger when compared to the second database and more data should be dedicated to the second database for reliability. Regardless, this post-training confusion matrix is an indication of sensors limitations being met due to the simi-selective nature of the images to incorporate into the second database. When the quadcopters merged classes, creating two overall classes in the machine learning process, the maximum validation accuracy of the acoustic category in the same conditions increased to 75.2% for Process 3, 79.8% for Process 4, and 64.6% for Process 5. In order to increase these accuracies more with the same conditions, the equipment would need to be upgraded, the sensors would need to stay inside the limitations, or a different machine learning process would need to be implemented.

Overall, the limitation of the RGB camera was due to range and background. The farther the drone is from the camera, the smaller and more blurry the drone becomes. This was noticeable at approximately 50 ft. for this RGB camera. The next factor to consider is the background. The drone went beyond 50 ft. multiple times in locations before the fifth location; however, the farthest distance occurred mostly with the sky as a background. When there is an object behind the drone, then the drone becomes hard to identify with the human eye. This issue became apparent at both the fourth and fifth locations.

The thermal camera is an advantageous sensor that will work well in conditions which are not suitable for the RGB camera. However, when the drone is in the hot intensity region of the image map, the drone is extremely hard to detect. The next limitation is the distance the drone is from the sensor. Like the RGB camera, the drone becomes smaller and more blurry at approximately 50 ft. With the combination of these two limitations, the thermal camera struggled to detect and classify the mini drones. Another limitation of this sensor is the background; if an object in the background has an extremely hot intensity compared to the drone, then the detection of the drone is incredibly difficult. These issues became very apparent at the fourth and fifth locations.

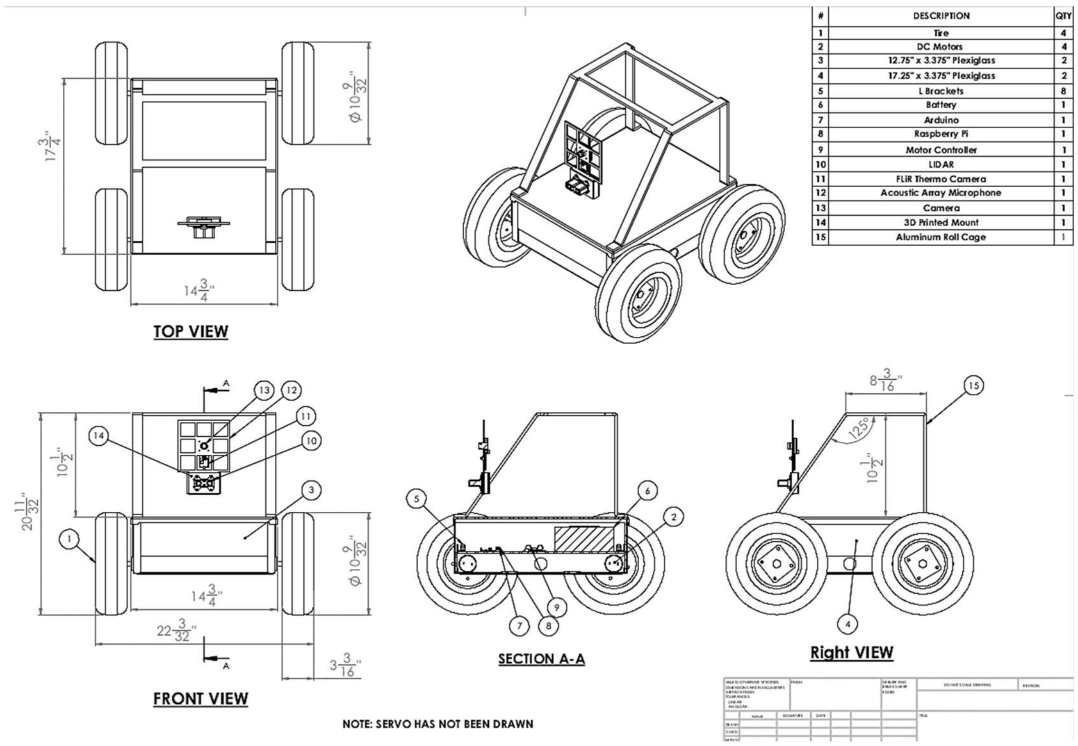
The acoustic sensor limitation was due to the distance and the mini quadcopters. The distance greatly affected the drone in the 50-ft.-to-90-ft. range. The accuracy dropped drastically when many of the input data from that range were incorporated. The mini quadcopters were a limitation due to the drone’s having difficulty differentiating them when the majority of the data was within 50 ft. However, when the mini quadcopters merged into one class, the validation accuracy in the classification was outstandingly higher.

Future Works

This project is capable of being modified in the future. One of the easy modifications to this project would be to use different sensors or to upgrade the sensors. Another possible modification would be to alter the type of input data, such as the Mel Frequency Cepstral Coefficient, or the machine learning process. The data could also be collected in a park or another location with different environmental settings or different drones. A modification code-wise could implement a convolutional neural network to determine which classification the drone is in using broad classes, and then perform another CNN on the specific type of drone inside that broad classification previously classified. As pointed out, there is a decent amount of potential to modify or expand upon this project to acquire additional data on the robotic detection of drones.

Appendices

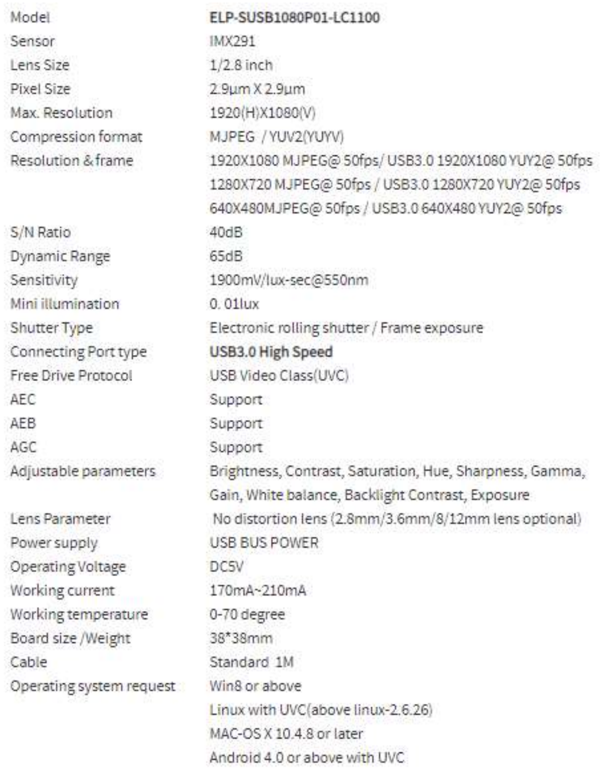
Appendix A: Robot Detailed Drawing



#	DESCRIPTION	QTY
1	Tire	4
2	DC Motors	4
3	12.75" x 3.375" Plexiglass	2
4	17.25" x 3.375" Plexiglass	2
5	L Brackets	8
6	Battery	1
7	Arduino	1
8	Raspberry Pi	1
9	Motor Controller	1
10	LIDAR	1
11	FLiR Thermo Camera	1
12	Acoustic Array Microphone	1
13	Camera	1
14	3D Printed Mount	1
15	Aluminum Roll Cage	1

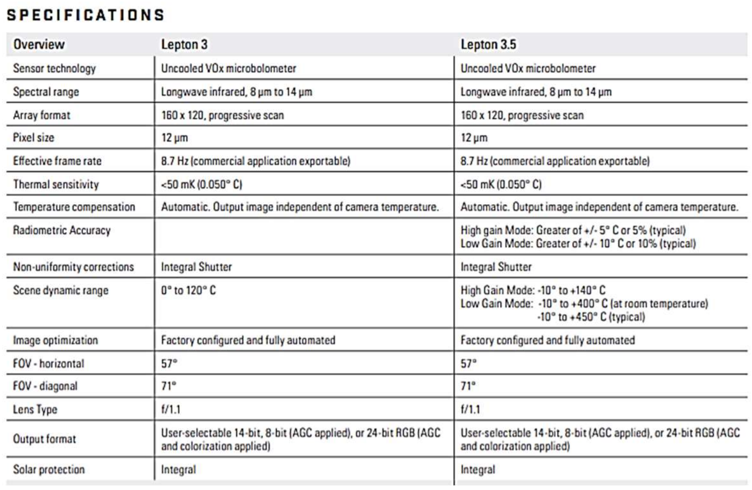
Figure A-1 Robot Design Detailed Drawing.

Appendix B: Sensor Technical Specifications



Model	**ELP-SUSB1080P01-LC1100**
Sensor	IMX291
Lens Size	1/2.8 inch
Pixel Size	2.9μm X 2.9μm
Max. Resolution	1920(H)X1080(V)
Compression format	MJPEG / YUV2(YUYV)
Resolution & frame	1920X1080 MJPEG@ 50fps/ USB3.0 1920X1080 YUY2@ 50fps 1280X720 MJPEG@ 50fps / USB3.0 1280X720 YUY2@ 50fps 640X480MJPEG@ 50fps / USB3.0 640X480 YUY2@ 50fps
S/N Ratio	40dB
Dynamic Range	65dB
Sensitivity	1900mV/lux-sec@550nm
Mini illumination	0. 01lux
Shutter Type	Electronic rolling shutter / Frame exposure
Connecting Port type	**USB3.0 High Speed**
Free Drive Protocol	USB Video Class(UVC)
AEC	Support
AEB	Support
AGC	Support
Adjustable parameters	Brightness, Contrast, Saturation, Hue, Sharpness, Gamma, Gain, White balance, Backlight Contrast, Exposure
Lens Parameter	No distortion lens (2.8mm/3.6mm/8/12mm lens optional)
Power supply	USB BUS POWER
Operating Voltage	DC5V
Working current	170mA~210mA
Working temperature	0-70 degree
Board size /Weight	38*38mm
Cable	Standard 1M
Operating system request	Win8 or above Linux with UVC(above linux-2.6.26) MAC-OS X 10.4.8 or later Android 4.0 or above with UVC

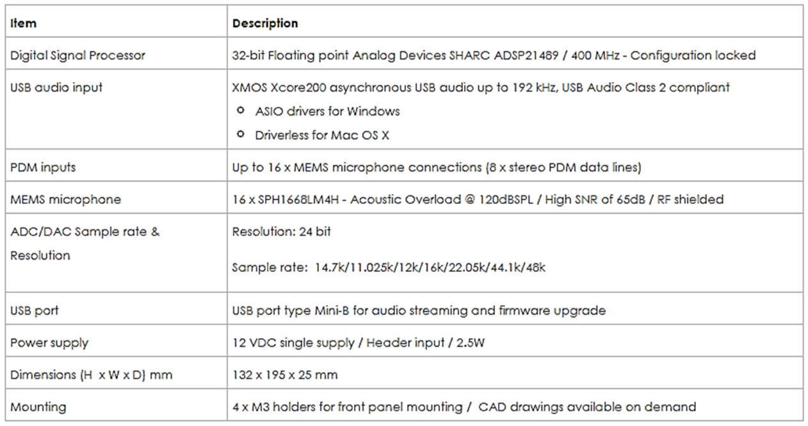
Table B-1: ELP USB 2.0 Webcam 2 Mega Pixels Specifications [11].



SPECIFICATIONS

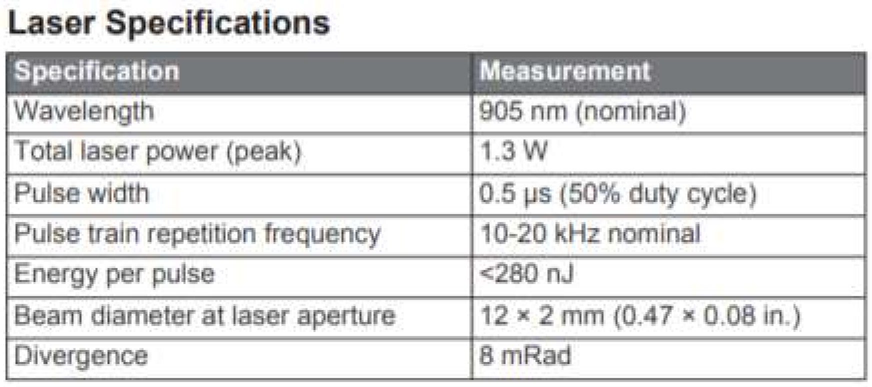
Overview	Lepton 3	Lepton 3.5
Sensor technology	Uncooled VOx microbolometer	Uncooled VOx microbolometer
Spectral range	Longwave infrared, 8 μm to 14 μm	Longwave infrared, 8 μm to 14 μm
Array format	160 x 120, progressive scan	160 x 120, progressive scan
Pixel size	12 μm	12 μm
Effective frame rate	8.7 Hz (commercial application exportable)	8.7 Hz (commercial application exportable)
Thermal sensitivity	<50 mK (0.050° C)	<50 mK (0.050° C)
Temperature compensation	Automatic. Output image independent of camera temperature.	Automatic. Output image independent of camera temperature.
Radiometric Accuracy		High gain Mode: Greater of +/- 5° C or 5% (typical) Low Gain Mode: Greater of +/- 10° C or 10% (typical)
Non-uniformity corrections	Integral Shutter	Integral Shutter
Scene dynamic range	0° to 120° C	High Gain Mode: -10° to +140° C Low Gain Mode: -10° to +400° C (at room temperature) -10° to +450° C (typical)
Image optimization	Factory configured and fully automated	Factory configured and fully automated
FOV - horizontal	57°	57°
FOV - diagonal	71°	71°
Lens Type	f/1.1	f/1.1
Output format	User-selectable 14-bit, 8-bit (AGC applied), or 24-bit RGB (AGC and colorization applied)	User-selectable 14-bit, 8-bit (AGC applied), or 24-bit RGB (AGC and colorization applied)
Solar protection	Integral	Integral

Table B-2: FLIR Lepton 3.0 Specifications [10].



Item	Description
Digital Signal Processor	32-bit Floating point Analog Devices SHARC ADSP21489 / 400 MHz - Configuration locked
USB audio input	XMOS Xcore200 asynchronous USB audio up to 192 kHz, USB Audio Class 2 compliant o ASIO drivers for Windows o Driverless for Mac OS X
PDM inputs	Up to 16 x MEMS microphone connections (8 x stereo PDM data lines)
MEMS microphone	16 x SPH1668LM4H - Acoustic Overload @ 120dBSPL / High SNR of 65dB / RF shielded
ADC/DAC Sample rate & Resolution	Resolution: 24 bit Sample rate: 14.7k/11.025k/12k/16k/22.05k/44.1k/48k
USB port	USB port type Mini-B for audio streaming and firmware upgrade
Power supply	12 VDC single supply / Header input / 2.5W
Dimensions (H x W x D) mm	132 x 195 x 25 mm
Mounting	4 x M3 holders for front panel mounting / CAD drawings available on demand

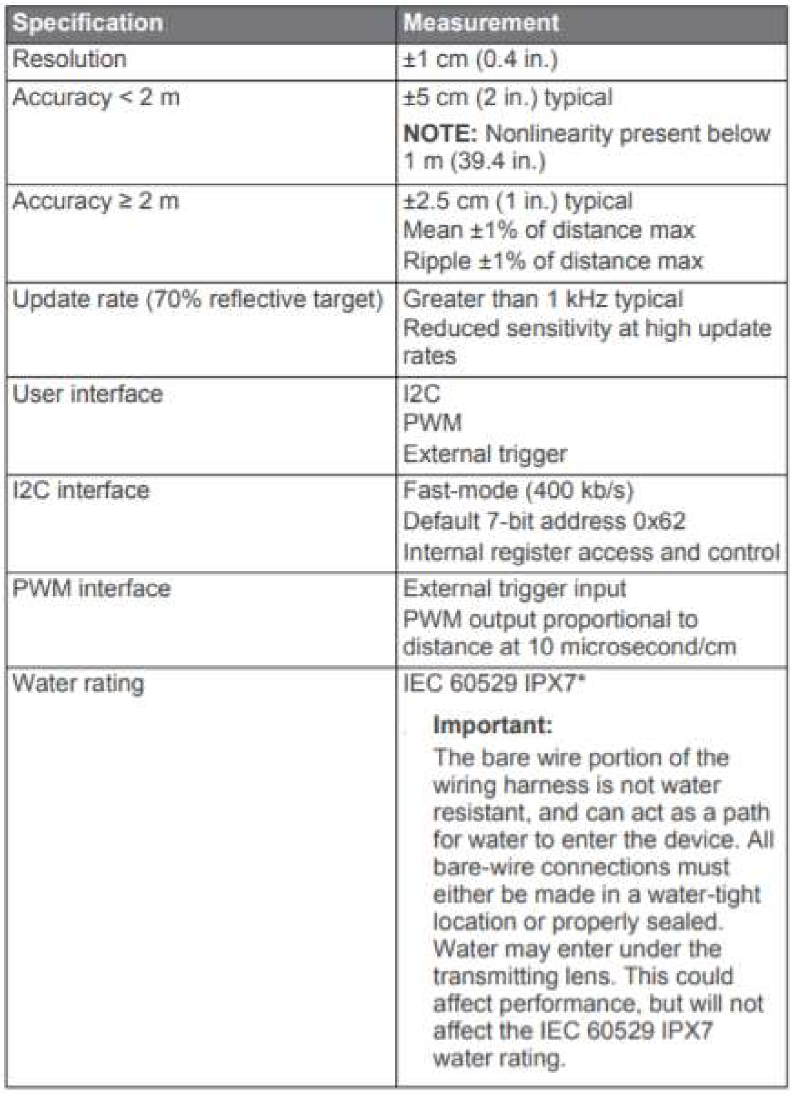
Table B-3: UMA-16 miniDSP Specifications [32].



Laser Specifications

Specification	Measurement
Wavelength	905 nm (nominal)
Total laser power (peak)	1.3 W
Pulse width	0.5 µs (50% duty cycle)
Pulse train repetition frequency	10-20 kHz nominal
Energy per pulse	<280 nJ
Beam diameter at laser aperture	12 × 2 mm (0.47 × 0.08 in.)
Divergence	8 mRad

Table B-4: LIDAR Lite v3 Performance Specifications 1 [31].



Specification	Measurement
Resolution	±1 cm (0.4 in.)
Accuracy < 2 m	±5 cm (2 in.) typical **NOTE:** Nonlinearity present below 1 m (39.4 in.)
Accuracy ≥ 2 m	±2.5 cm (1 in.) typical Mean ±1% of distance max Ripple ±1% of distance max
Update rate (70% reflective target)	Greater than 1 kHz typical Reduced sensitivity at high update rates
User interface	I2C PWM External trigger
I2C interface	Fast-mode (400 kb/s) Default 7-bit address 0x62 Internal register access and control
PWM interface	External trigger input PWM output proportional to distance at 10 microsecond/cm
Water rating	IEC 60529 IPX7* **Important:** The bare wire portion of the wiring harness is not water resistant, and can act as a path for water to enter the device. All bare-wire connections must either be made in a water-tight location or properly sealed. Water may enter under the transmitting lens. This could affect performance, but will not affect the IEC 60529 IPX7 water rating.

Table B-5: LIDAR Lite v3 Performance Specifications 2 [31].

Appendix C: Machine Learning Process Evaluation

C-1: First Input Set Evaluation. The convolutional neural network was performed three times on the data from Location 1 and the table shows the validation accuracy for each performance.

Validation Accuracy (%)	**RGB**	**Thermal**	**Acoustic**
Evaluation 1	**76.47**	**45.83**	**66.67**
Evaluation 2	**88.24**	**66.67**	**45.15**
Evaluation 3	**94.12**	**33.33**	**61.54**

Table C-2: Second Input Set Evaluation. The convolutional neural network was performed three times on the data from Locations 1-2 and the table shows the validation accuracy for each performance.

Validation Accuracy (%)	**RGB**	**Thermal**	**Acoustic**
Evaluation 1	**71.93 %**	**65.69 %**	**44.09 %**
Evaluation 2	**75.44 %**	**62.75 %**	**58.06 %**
Evaluation 3	**67.54 %**	**72.55 %**	**39.78 %**

C-3: Third Input Set Evaluation. The convolutional neural network was performed three times on the data from Locations 1-3 and the table shows the validation accuracy for each performance.

Validation Accuracy (%)	**RGB**	**Thermal**	**Acoustic**
Evaluation 1	**86.41 %**	**82.88 %**	**66.18 %**
Evaluation 2	**96.60 %**	**82.66 %**	**71.26 %**
Evaluation 3	**77.92 %**	**82.43 %**	**70.53 %**

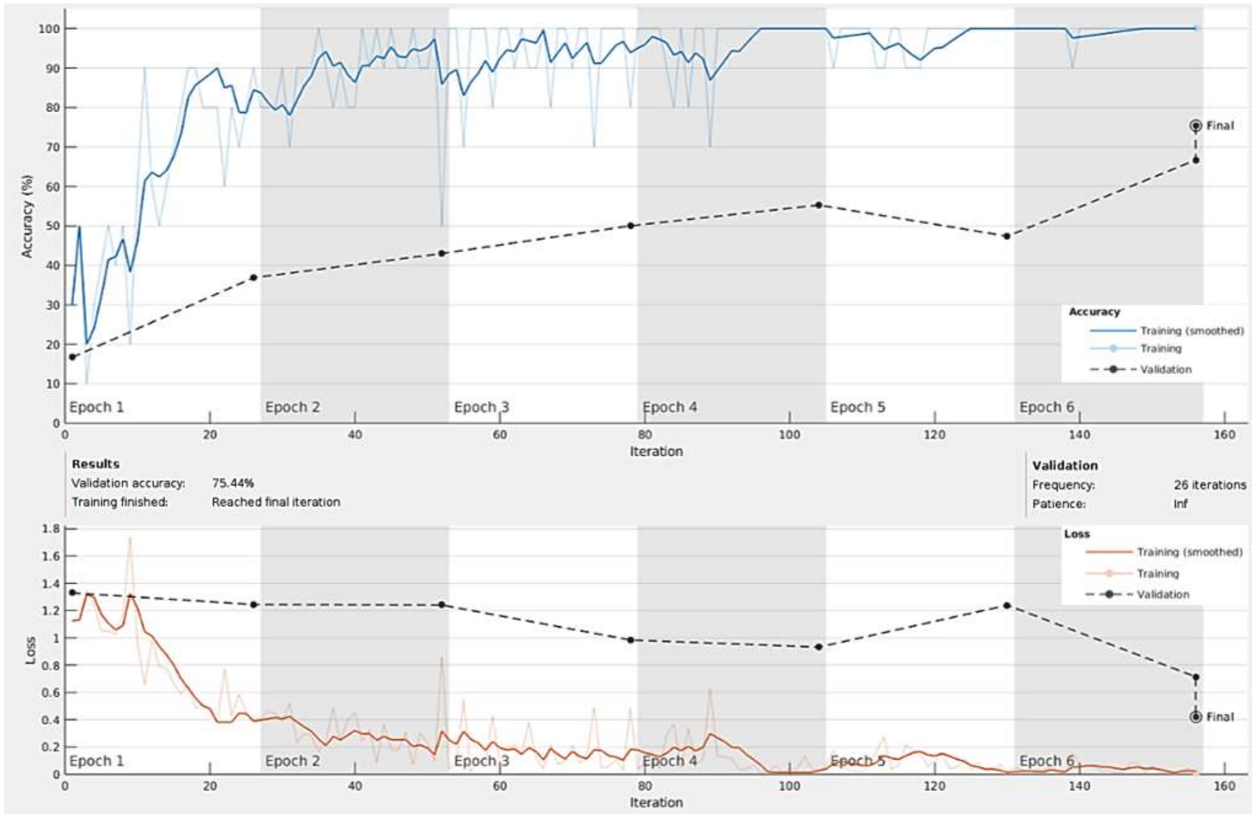
Table C-4: Fourth Input Set Evaluation. The convolutional neural network was performed three times on the data from Locations 1-4 and the table shows the validation accuracy for each performance.

Validation Accuracy (%)	**RGB**	**Thermal**	**Acoustic**
Evaluation 1	**75.43**	**53.92**	**60.49**
Evaluation 2	**74.57**	**62.66**	**56.58**
Evaluation 3	**78.35**	**58.47**	**56.79**

Table C-5: Fifth Input Set Evaluation. The convolutional neural network was performed three times on the data from Locations 1-5 and the table shows the validation accuracy for each performance.

Validation Accuracy (%)	**RGB**	**Thermal**	**Acoustic**
Evaluation 1	**60.23**	**31.84**	**39.58**
Evaluation 2	**49.15**	**41.92**	**36.31**
Evaluation 3	**59.55**	**43.07**	**46.58**

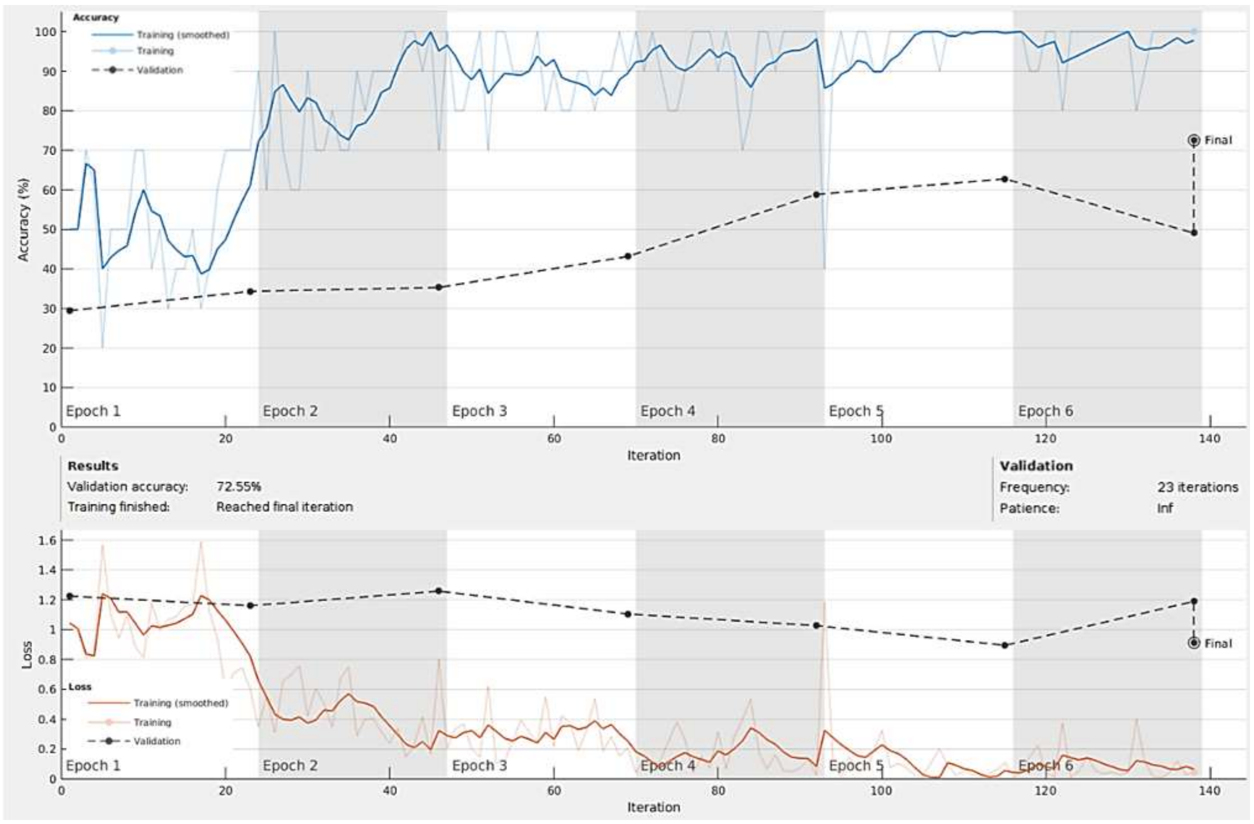
Appendix D: Machine Learning Process 2



Results
Validation accuracy: 75.44%
Training finished: Reached final iteration

Validation
Frequency: 26 iterations
Patience: Inf

Figure D-1: RGB Machine Learning Process Up to the Second Location.



Results

Validation accuracy: 72.55%

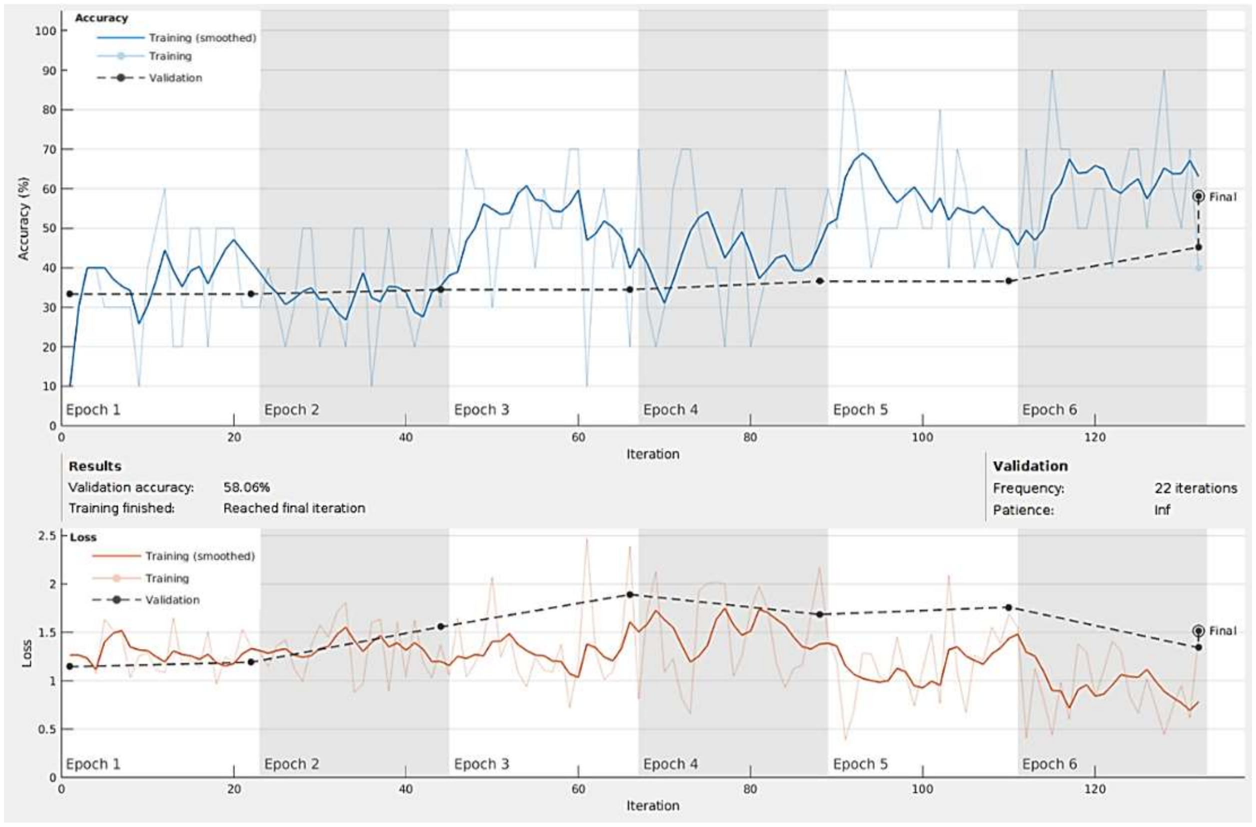
Training finished: Reached final iteration

Validation

Frequency: 23 iterations

Patience: Inf

Figure D-2: Thermal Machine Learning Process Up to the Second Location.



Results
Validation accuracy: 58.06%
Training finished: Reached final iteration

Validation
Frequency: 22 iterations
Patience: Inf

Figure D-3: Acoustic Machine Learning Process Up to the Second Location.

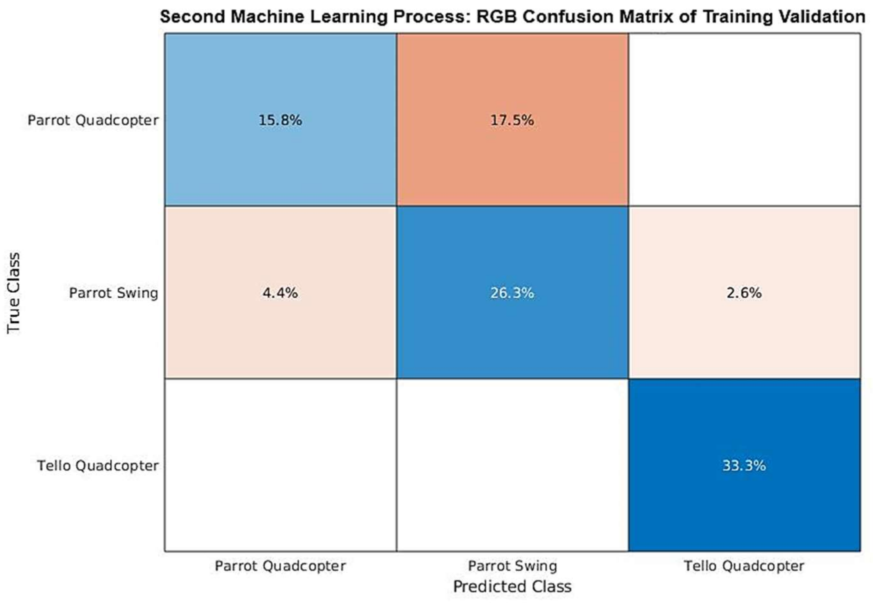


Figure D-4: Process 2 RGB Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

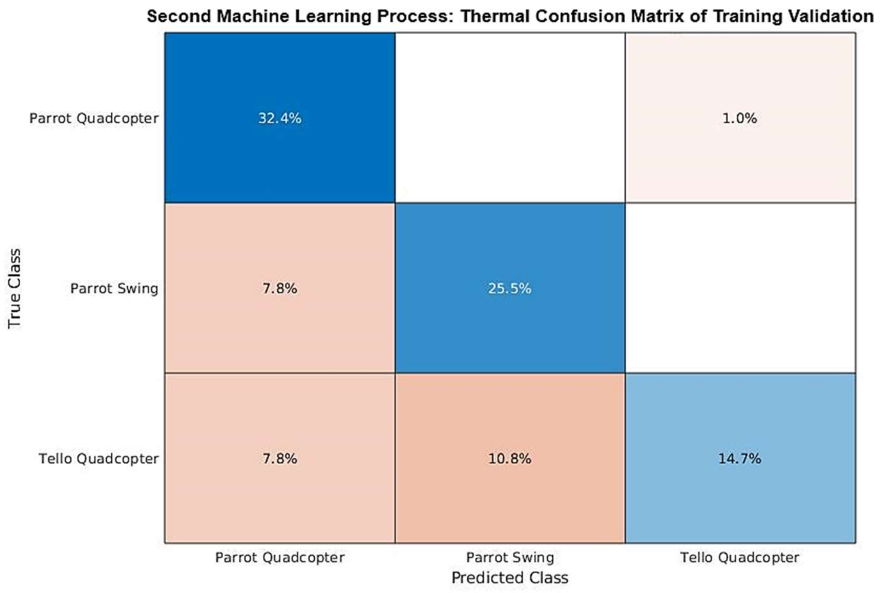


Figure D-5: Process 2 Thermal Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

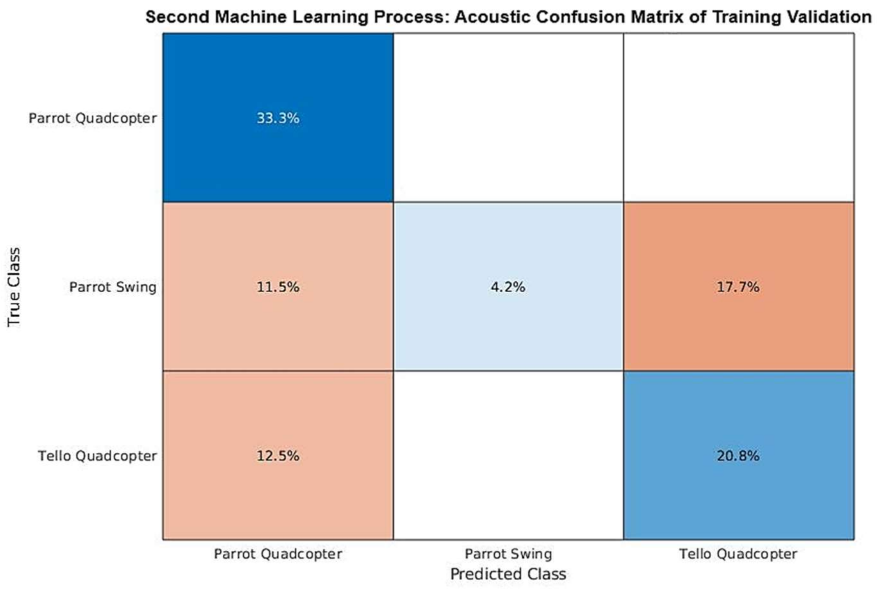


Figure D-6: Process 2 Acoustic Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

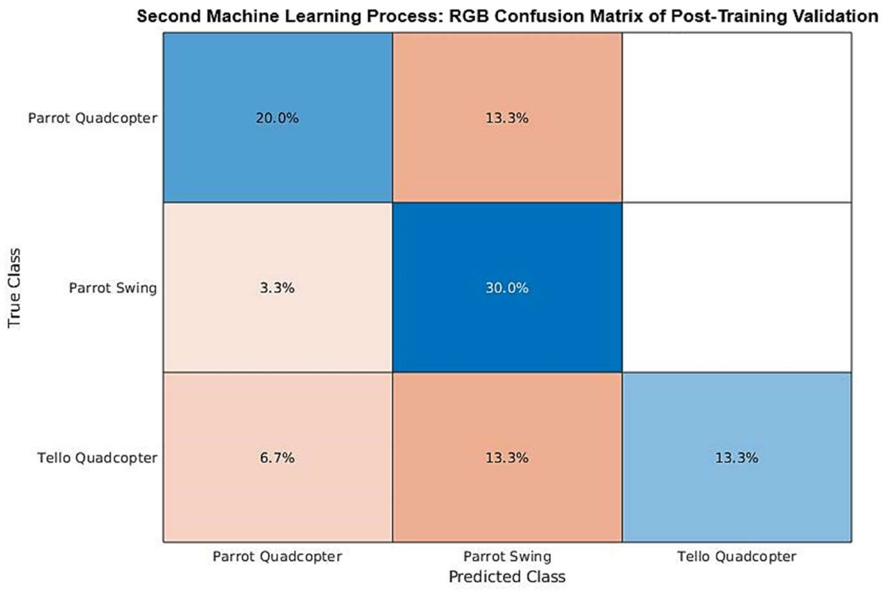


Figure D-7: Process 2 RGB Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

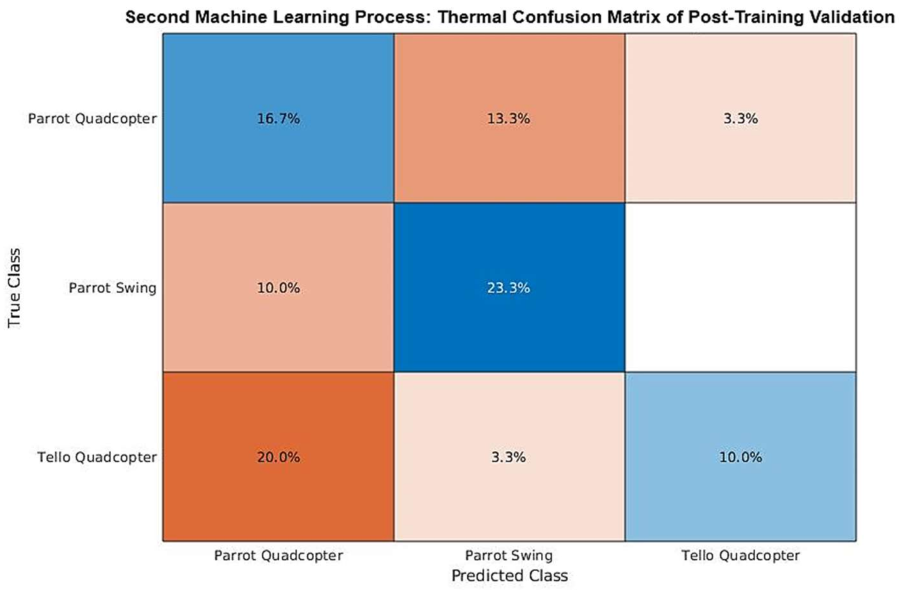


Figure D-8: Process 2 Thermal Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

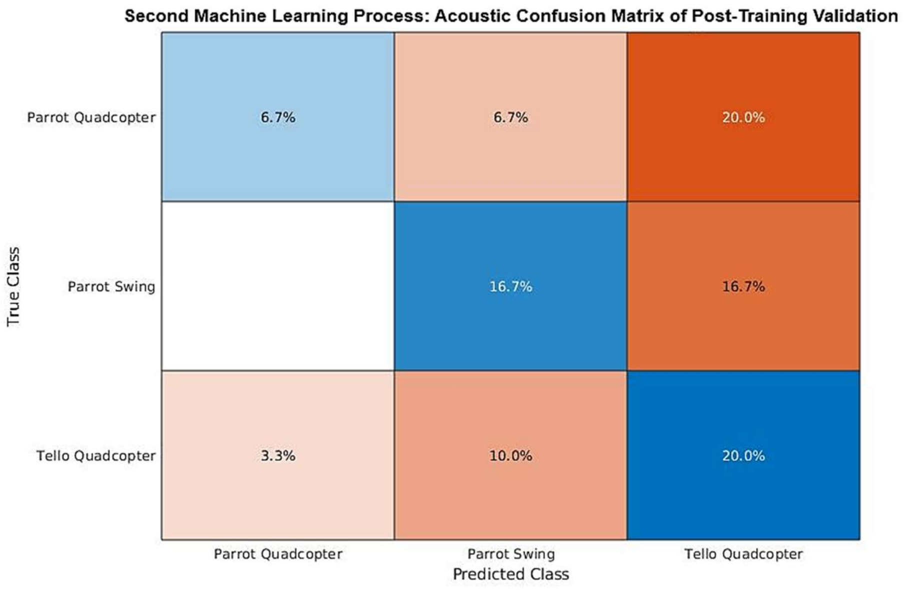
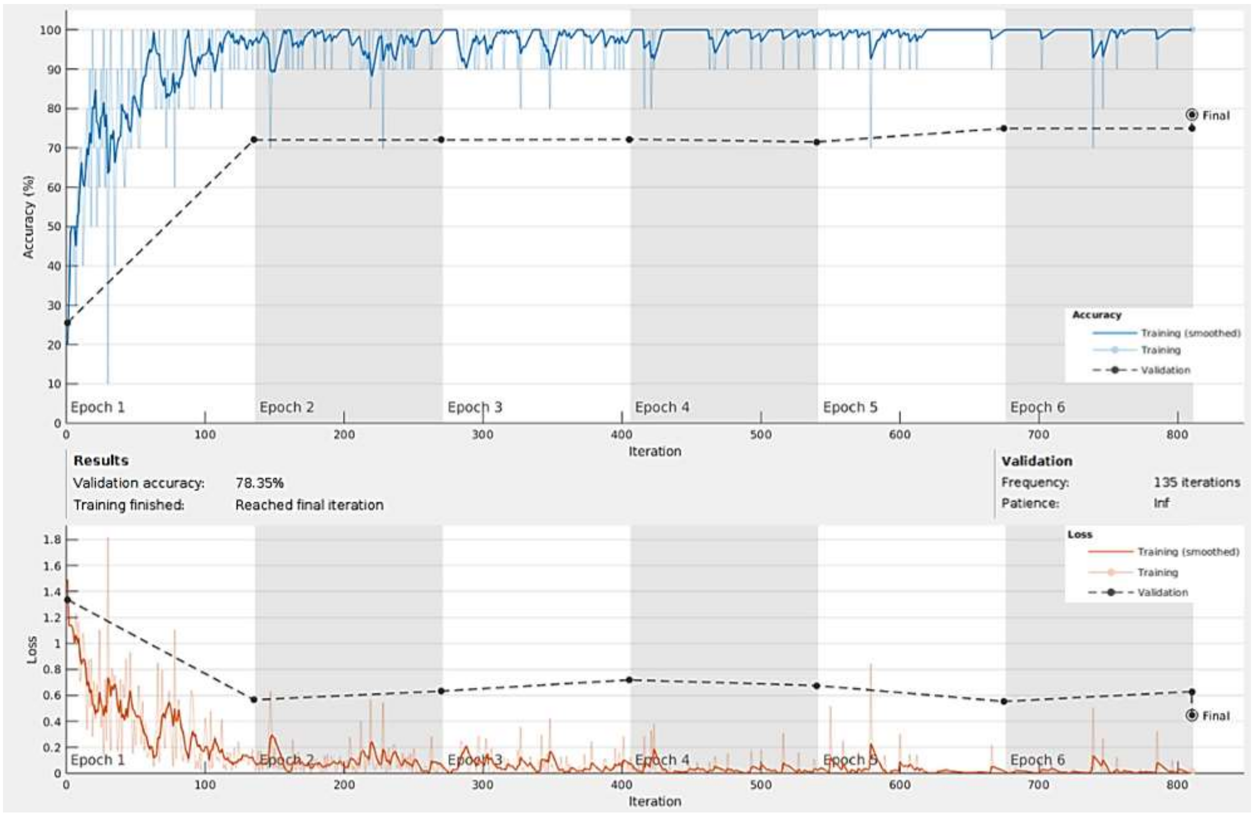


Figure D-9: Process 2 Acoustic Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

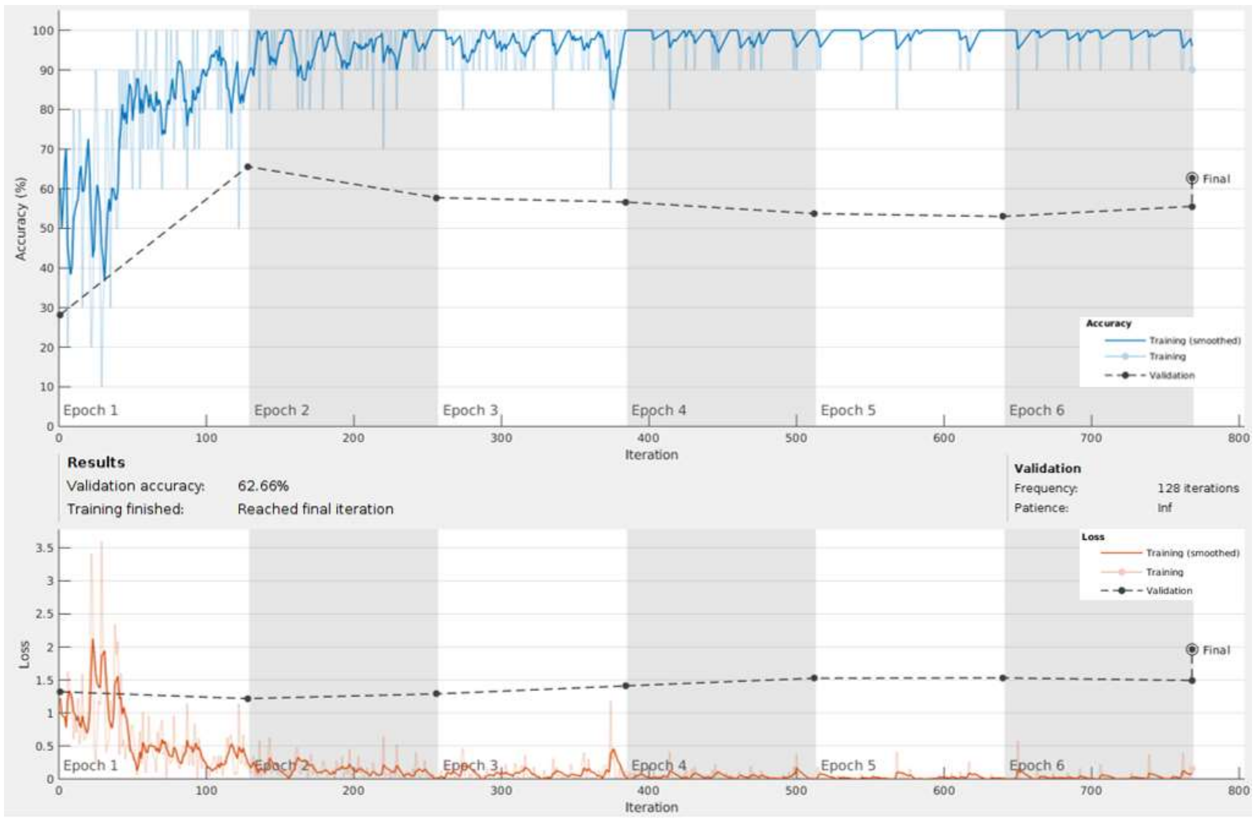
Appendix E: Machine Learning Process 4



Results
Validation accuracy: 78.35%
Training finished: Reached final iteration

Validation
Frequency: 135 iterations
Patience: Inf

Figure E-1: RGB Machine Learning Process Up to the Fourth Location.



Results

Validation accuracy: 62.66%

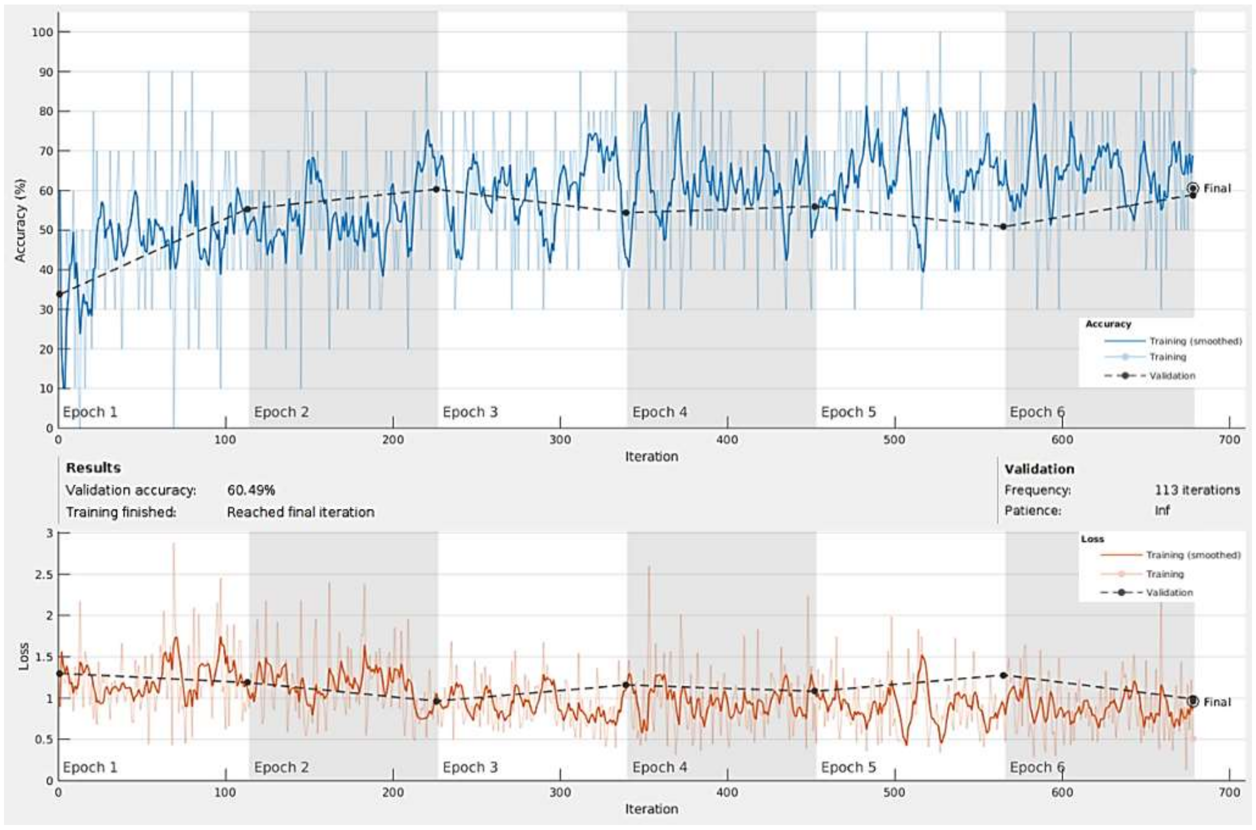
Training finished: Reached final iteration

Validation

Frequency: 128 iterations

Patience: Inf

Figure E-2: Thermal Machine Learning Process Up to the Fourth Location.



Results

Validation accuracy: 60.49%

Training finished: Reached final iteration

Validation

Frequency: 113 iterations

Patience: Inf

Figure E-3: Acoustic Machine Learning Process Up to the Fourth Location.

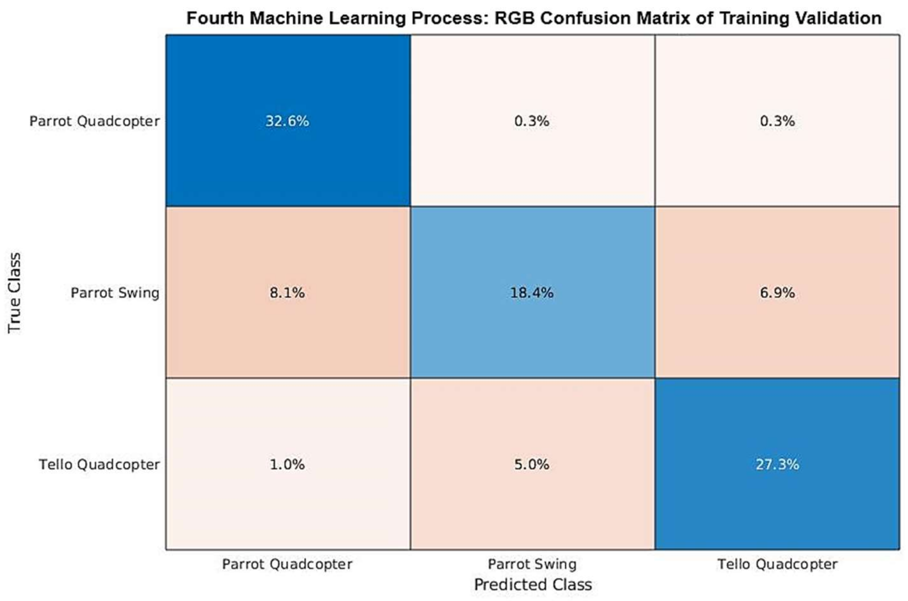


Figure E-4: Process 4 RGB Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

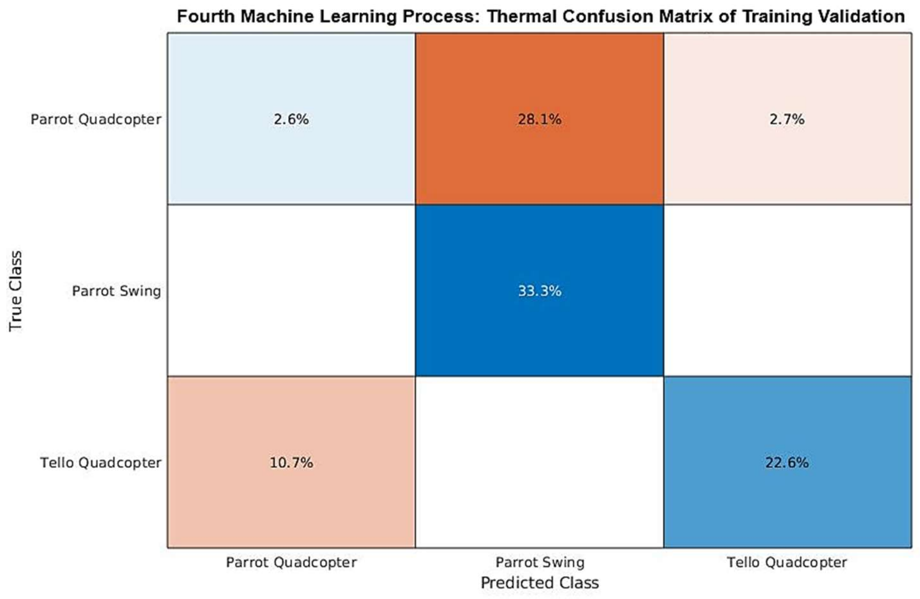


Figure E-5 Process 4 Thermal Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

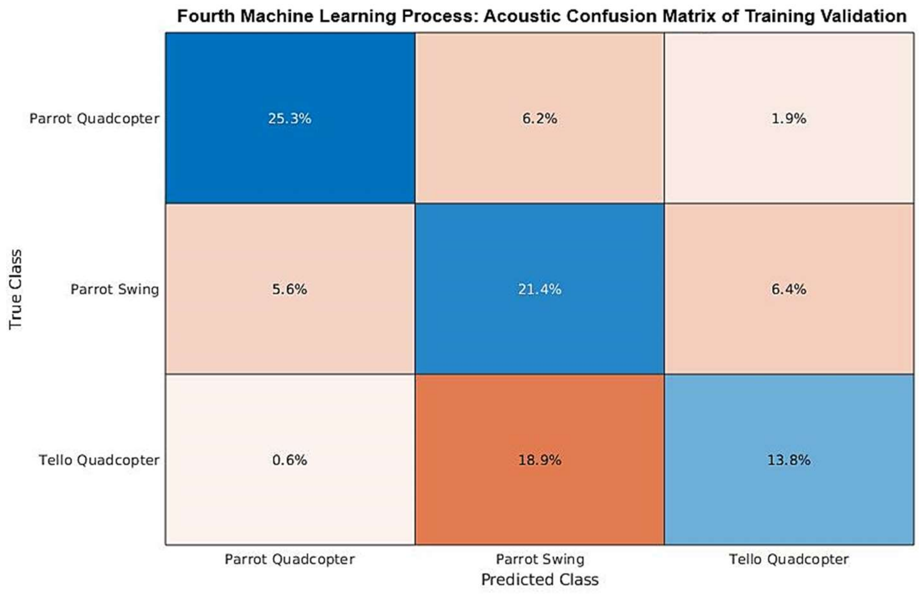


Figure E-6: Process 4 Acoustic Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter.

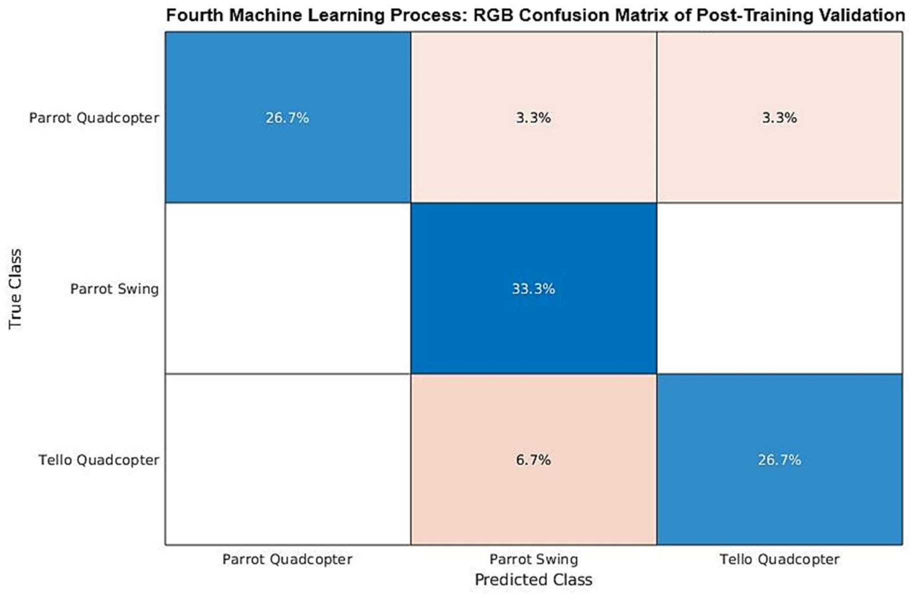


Figure E-7: Process 4 RGB Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

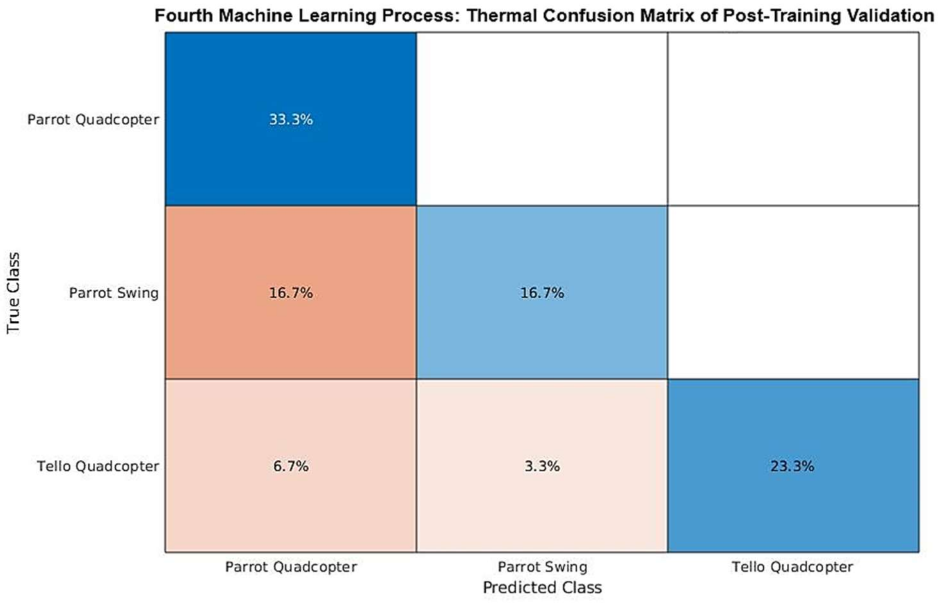


Figure E-8: Process 4 Thermal Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

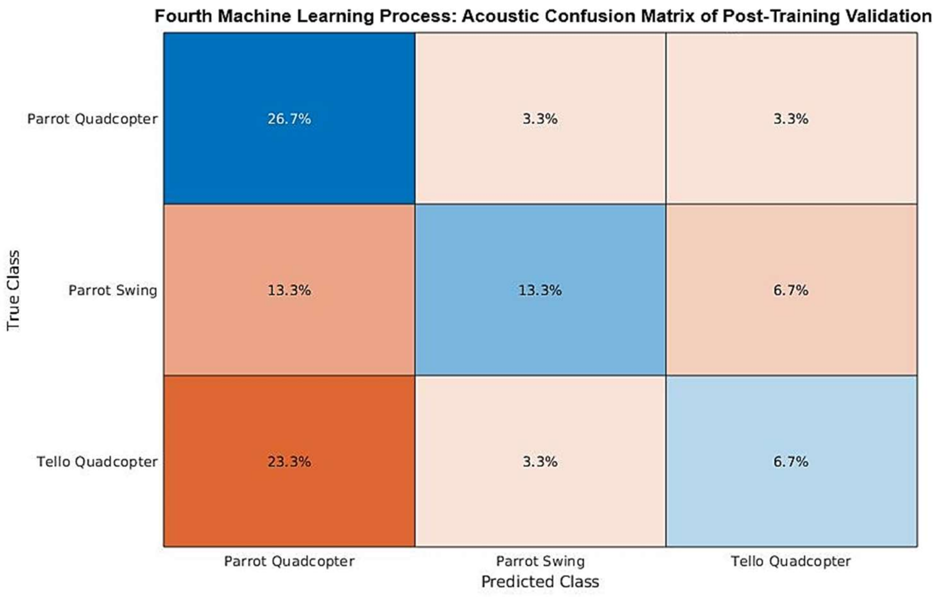


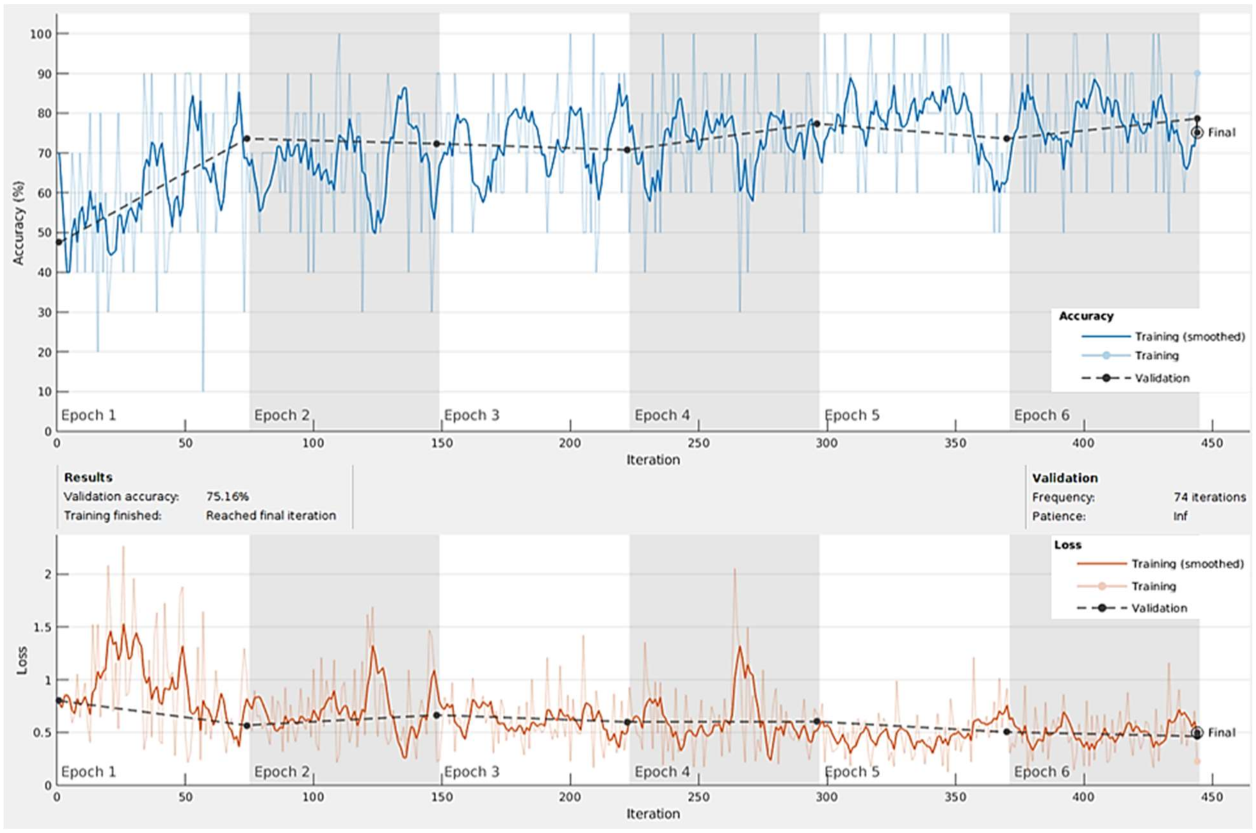
Figure E-9: Process 4 Acoustic Post-Training Confusion Matrix. The first column and row represent the Parrot Swing. The second column and row represent the Parrot Quadcopter, and the third column and row represent the Tello Quadcopter. This includes variations and the limitations from the fifth location that the network has not seen yet.

Appendix F: Overall Time Performances

Table G-1: Overall Time Performances. This table shows the comparison between the two classes of Parrot Swing and Quadcopters to the three classes Parrot Swing, Parrot Quadcopter, and Tello Quadcopter for Process 3, 4, and 5. The Process Random Difference is the validation accuracy subtracted by the random guess to compare the difference in accuracy realistically.

Machine Learning Inputs	**RGB**	**Thermal**	**Acoustic**
Process 1	**4min 34s**	**4min 11s**	**3min 26s**
Process 2	**9min 37s**	**8min 30s**	**8min 11s**
Process 3	**44min 13s**	**37min 51s**	**36min 58s**
Process 4	**50 min 11s**	**47min 28s**	**41min 47s**
Process 5	**78min 57s**	**77min 22s**	**60min 52s**

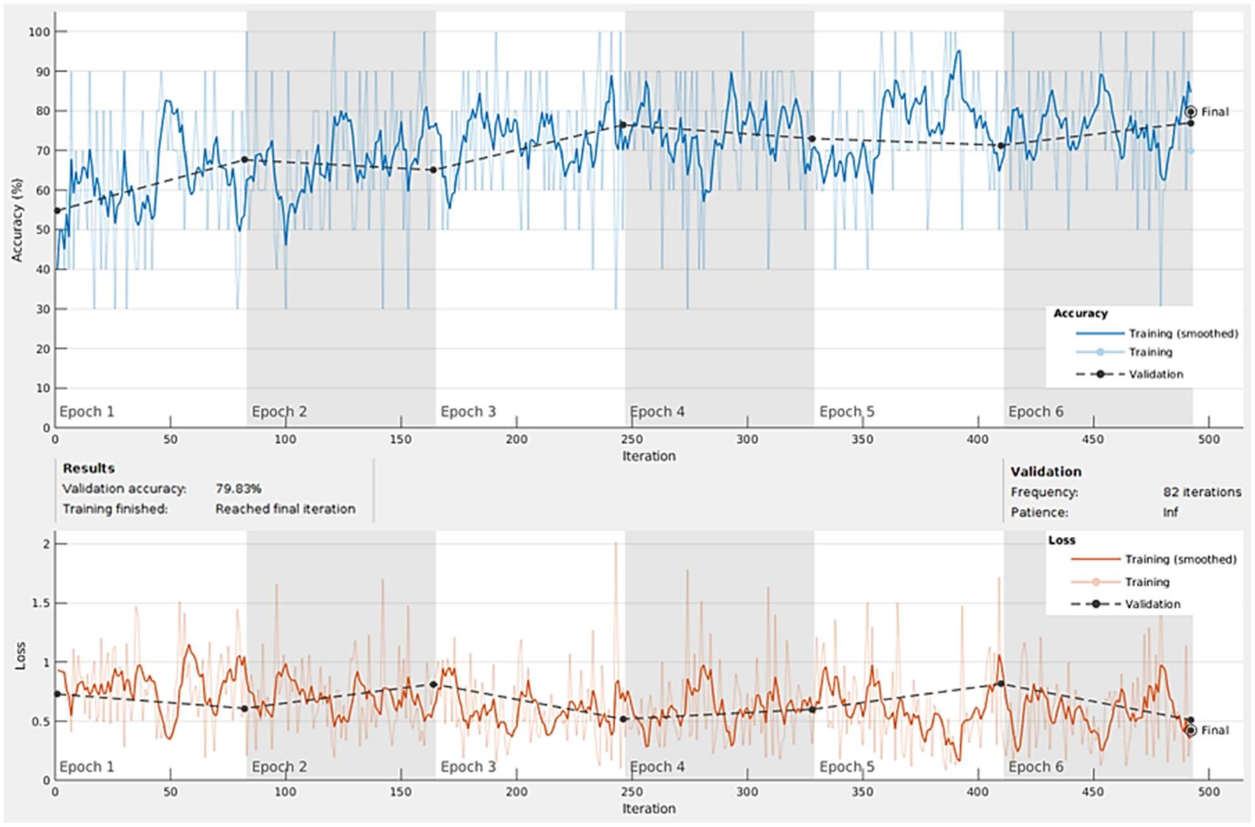
Appendix G: Modified Machine Learning Process



Results
Validation accuracy: 75.16%
Training finished: Reached final iteration

Validation
Frequency: 74 iterations
Patience: Inf

Figure G-1: Modified Acoustic Process 3. The machine learning process up to the third location was modified to contain two classes (Parrot Swing and Quadcopters).



Results

Validation accuracy: 79.83%

Training finished: Reached final iteration

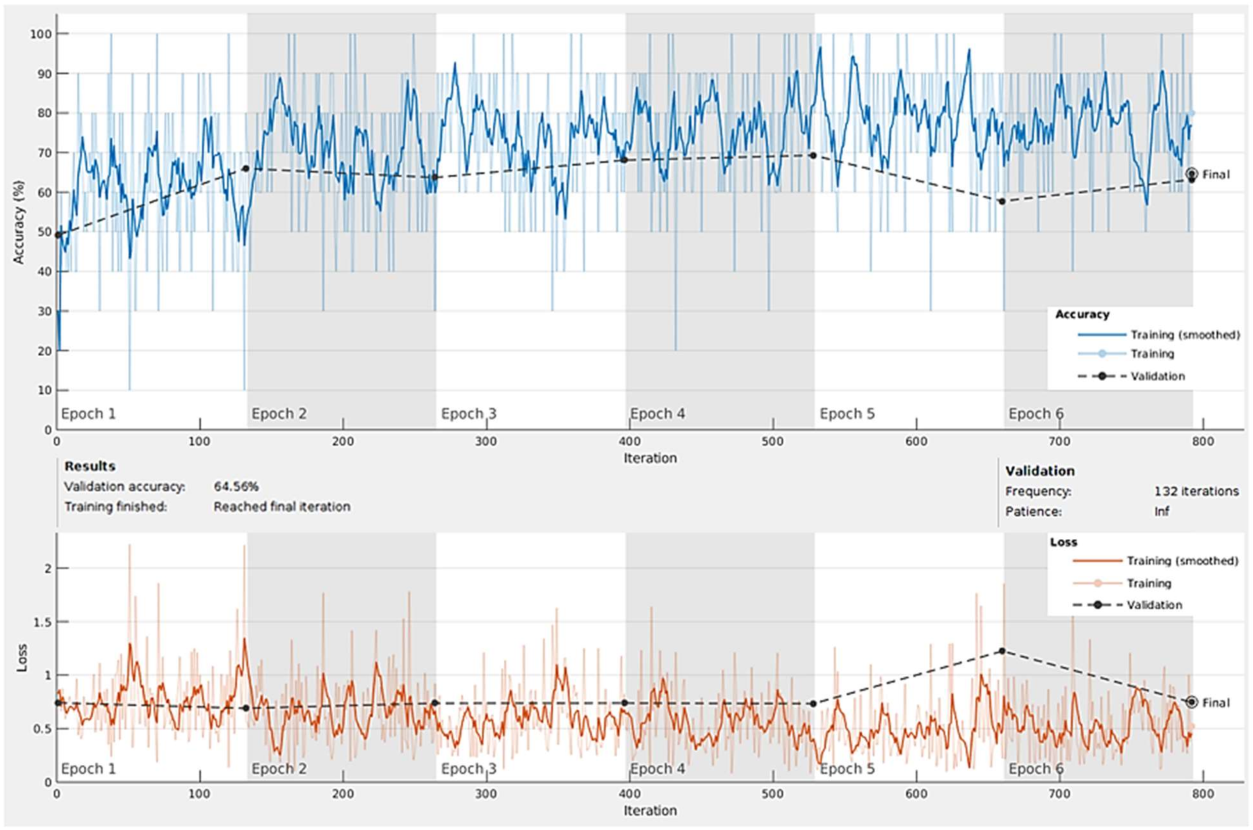
Validation

Frequency: 82 iterations

Patience: Inf

Figure G-2: Modified Acoustic Process 4. The machine learning process up to the fourth location was modified to contain two classes (Parrot Swing and Quadcopters).

`



Results

Validation accuracy: 64.56%

Training finished: Reached final iteration

Validation

Frequency: 132 iterations

Patience: Inf

Figure G-3: Modified Acoustic Process 5. The machine learning process up to the fifth location was modified to contain two classes (Parrot Swing and Quadcopters).

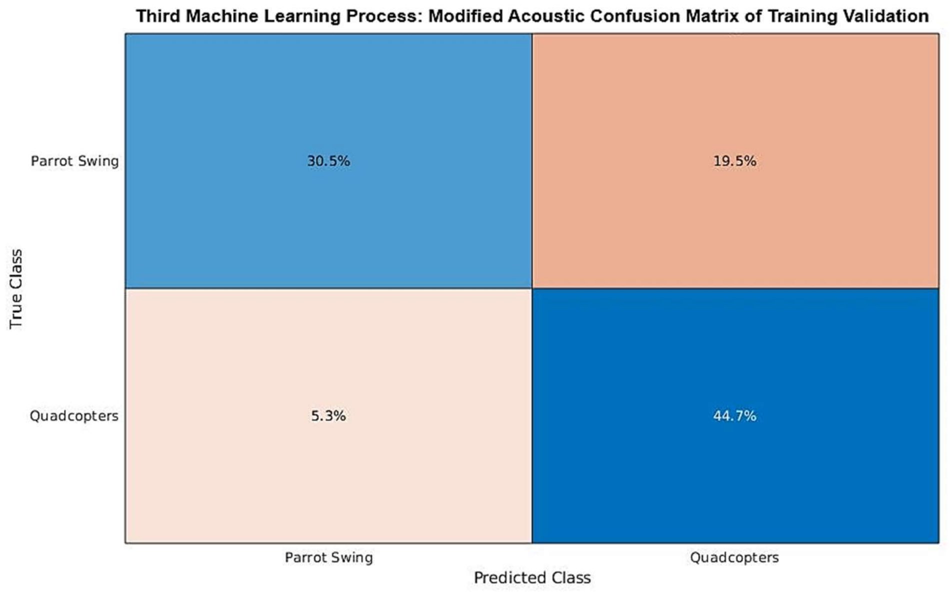


Figure G-4: Modified Process 3 Acoustic Training Confusion Matrix.

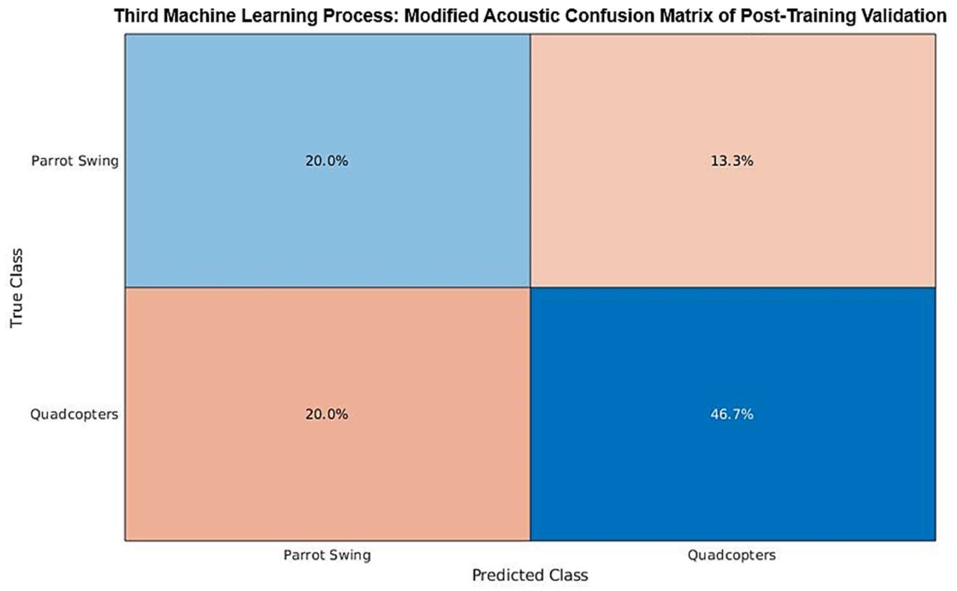


Figure G-4: Modified Process 3 Acoustic Post-Training Confusion Matrix.

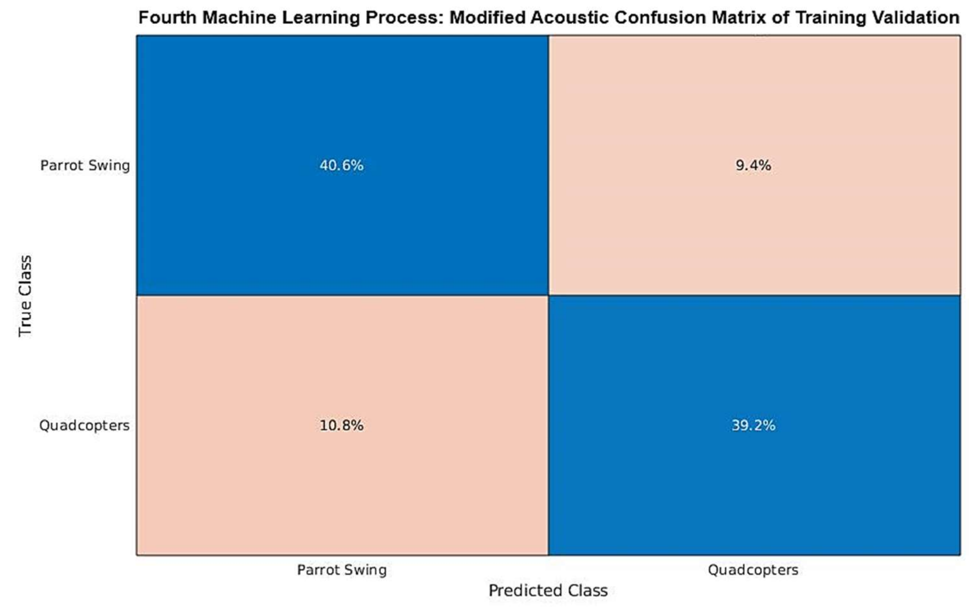


Figure G-5: Modified Process 4 Acoustic Training Confusion Matrix.

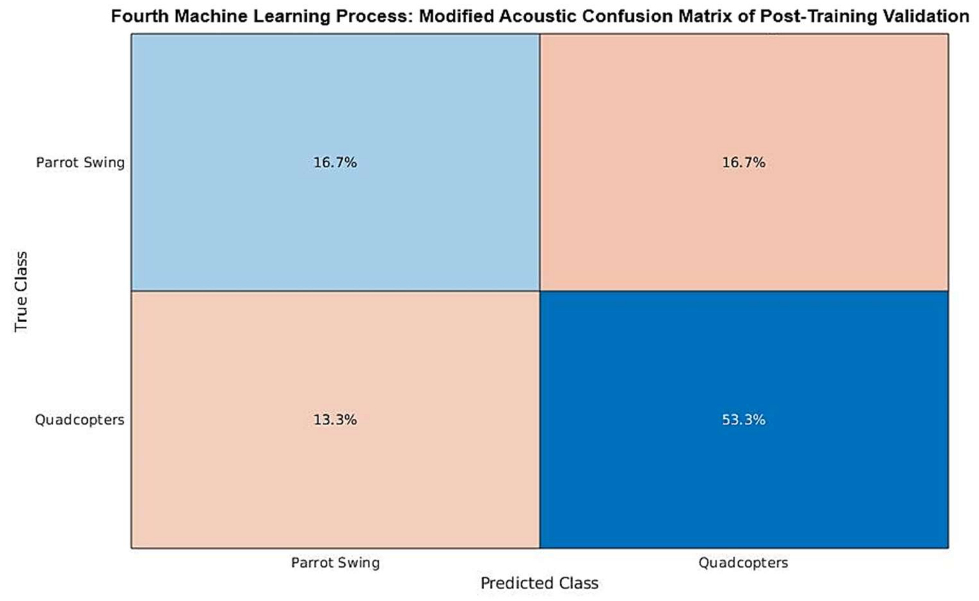


Figure G-6: Modified Process 4 Acoustic Post-Training Confusion Matrix.

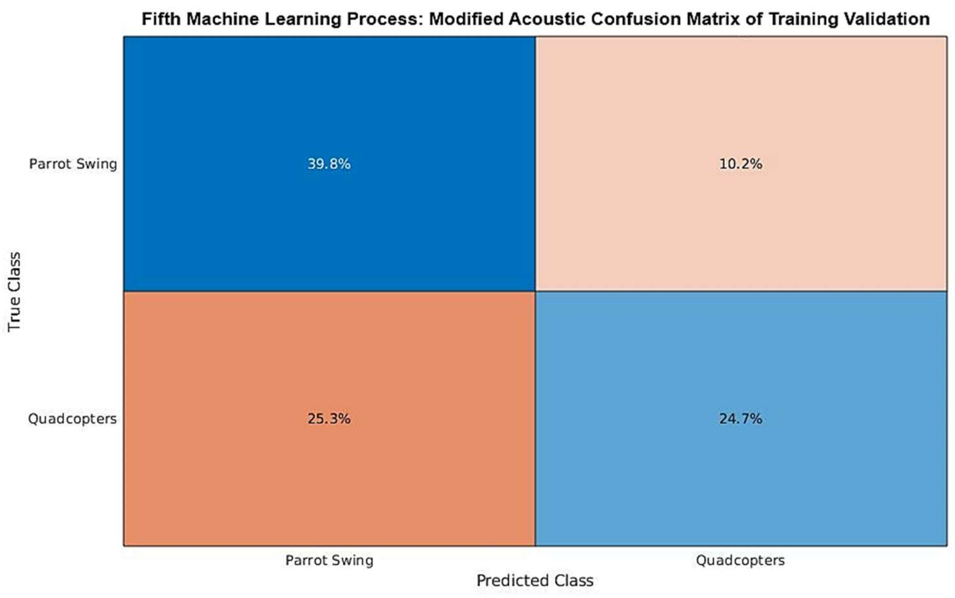


Figure G-7: Modified Process 5 Acoustic Post-Training Confusion Matrix.

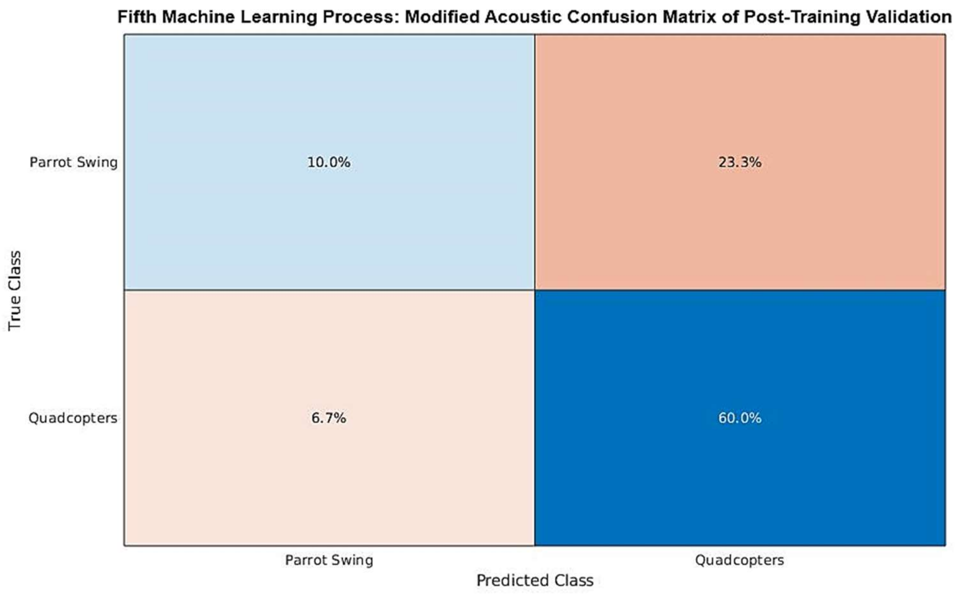


Figure G-8: Modified Process 5 Acoustic Post-Training Confusion Matrix.

Appendix H: Data Collecting Program

```
from collections import deque
import numpy as np
import time
import datetime
from PIL import Image, ImageTk
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib.figure import Figure
import wave

import cv2
import os
import pyaudio
#import csv
import tkinter as tk
import threading
import dill
import time
import shutil

######################################
########## Variable Declaration ##########
######################################
user_name='wasp'
usb_name='Flashy'
dirName='/media/%s/%s/Robot_Sensors' %(user_name, usb_name)

## Thermal and USB Camera Inputs ##
therm_input=1
usb_cam_input=0
cameras_process=0
acoustic_process=0

aud_prev=[]


counting=0

rec_setup=0
rec_data = False

### Acoustic Sensor ###
FORMAT = pyaudio.paInt16 # We use 16 bit format per sample
CHANNELS = 16
RATE = 44100
CHUNK = 1024 # 1024 bytes of data read from the buffer #44100
RECORD_SECONDS = 0.001
WAVE_OUTPUT_FILENAME = ("RobotAcoustic.wav")
Mic_Device_Number=3
audio = pyaudio.PyAudio()


stream = audio.open(format=FORMAT,
                                        channels=CHANNELS,
                                        rate=RATE,
                                        input=True,
                                        input_device_index = Mic_Device_Number,
                                        frames_per_buffer=CHUNK)
```

```
keep_going=True

proceed=1
##Exact positioning
cam_size=320
stn_font='18'
stb_font='15'

quit_x=.01
quit_y=.76875

record_x=.01
record_y=.6

stop_x=.1625
stop_y=.6

rgb_x=0.0885
rgb_y=0
rgb_lx=.011875
rgb_ly=.05125

hsv_x=.43
hsv_y=0
hsv_lx=.3425
hsv_ly=.05125

fps_x=.275
fps_y=.5125

thm_x=.74
thm_y=0
thm_lx=.67375
thm_ly=0.05125

act_x=.375
act_y=.56125
act_lx=.375
act_ly=.56125

####################################################################################
####################################################################################
class Application(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self,master)

       ###### Buttons ######
        quit_button = tk.Button(master=root, text='End Process', font='Helvetica %s
bold' %stb_font, bg='red', command=lambda: quit_(root))
        quit_button.place(relx=quit_x, rely=quit_y)

        record_button = tk.Button(master=root, text='Record Data', font='Helvetica %s
bold' %stb_font, bg='green', command=lambda: record_data())
        record_button.place(relx=record_x, rely=record_y)

        stop_button = tk.Button(master=root, text='Stop Record', font='Helvetica %s
bold' %stb_font, bg='green', command=lambda: stop_data())
        stop_button.place(relx=stop_x, rely=stop_y)

        self.guiSetup()
        self.main_setup()
        self.main()
        self.after(0,self.main)
```

```
#####################################################################################
    def main_setup(self):
            global acoustic_process
            daemonTc=True
            acous_thread=threading.Thread(target=self.audio_stream, daemon=daemonTc)
            acous_thread.start()

            while proceed==1:
                cam_thread=threading.Thread(target=update_image, daemon=daemonTc)
                cam_thread.start()

                try:
                    cam_thread.join()
                except:
                    pass
                self.convert_image(rgb_image_label)
                self.thermal_vid()

                if acoustic_process==0:
                    break
            try:
                acous_thread.join()
            except:
                pass

            plot_data_setup()
            acoustic_process=0
            daemonTc=False

#####################################################################################
    def main(self):
        print('main loop')
        global acoustic_process
        global aud_prev
        global counting
        daemonTc=True
        acous_thread=threading.Thread(target=self.audio_stream, daemon=daemonTc)
        acous_thread.start()

        while proceed==1:
            cam_thread=threading.Thread(target=update_image, daemon=daemonTc)
            cam_thread.start()


            try:
                cam_thread.join()
            except:
                pass

            self.convert_image(rgb_image_label)
            self.thermal_vid()

            if acoustic_process==0:
                break
        try:
            acous_thread.join()
        except:
            pass

        acoustic_process=0
        print('next')
        daemonTc=False
        ax.clear()
        print('plot begin')

        length=range(0,len(aud_prev))
        mlength=max(length)
        length2=range(mlength+1,mlength+len(audio_data)+1)
        mlength2=max(length2)
        ax.plot(length[0:int(mlength*.25)],aud_prev[0:int(mlength*.25)],
linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()

ax.plot(length[int(mlength*.25):int(mlength*.5)],aud_prev[int(mlength*.25):int(mlength
*.5)], linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()

ax.plot(length[int(mlength*.5):int(mlength*.75)],aud_prev[int(mlength*.5):int(mlength*
.75)], linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()
        ax.plot(length[int(mlength*.75):],aud_prev[int(mlength*.75):],
linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()
        ax.plot(length2[0:int(mlength2*.25)],audio_data[0:int(mlength2*.25)],
linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()

ax.plot(length2[int(mlength2*.25):int(mlength2*.5)],audio_data[int(mlength2*.25):int(m
length2*.5)], linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()

ax.plot(length2[int(mlength2*.5):int(mlength2*.75)],audio_data[int(mlength2*.5):int(ml
ength2*.75)], linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()
        ax.plot(length2[int(mlength2*.75):],audio_data[int(mlength2*.75):],
linestyle='solid', marker='.', color='b')
        update_image()
        self.convert_image(rgb_image_label)
        self.thermal_vid()
        canvas.draw()

        plt.close()
        aud_prev=audio_data
        print('plot end')
        counting=counting+1
        print(counting)
        self.after(0,self.main)

####################################################################################
    def guiSetup(self):
        global canvas
        global ax
        global cam
        global therm_cam
        global rgb_image_label
        global hsv_image_label
        global therm_image_label
        global fps_label
        global canvas
        global li_fig, ax

        ########################################################################
        ############################# Name Labels ##############################
        ########################################################################

       ###### Camera ######
       #RGB Image
        rgb_image_label_name=tk.Label(root, text="RGB Camera", font='Helvetica %s
bold' %stn_font)
        rgb_image_label_name.place(relx=rgb_x, rely=rgb_y)

        rgb_image_label = tk.Label(master=root)
        rgb_image_label.place(relx=rgb_lx, rely=rgb_ly)

       #HSV Image
        hsv_image_label_name=tk.Label(root, text="HSV Camera", font='Helvetica %s
bold' %stn_font)
        hsv_image_label_name.place(relx=hsv_x, rely=hsv_y)

        hsv_image_label = tk.Label(master=root)
        hsv_image_label.place(relx=hsv_lx, rely=hsv_ly)

       #FPS
        cam = cv2.VideoCapture(usb_cam_input)
        fps_label = tk.Label(master=root)
        fps_label._frame_times = deque([0]*5)
        fps_label.place(relx=fps_x, rely=fps_y)

       ####### Thermal Image ######
        thermal_label_name=tk.Label(root, text="Thermal Picture", font='Helvetica %s
bold' %stn_font)
        thermal_label_name.place(relx=thm_x, rely=thm_y)

       #Capture video frames
        therm_image_label = tk.Label(master=root)
        therm_image_label.place(relx=thm_lx, rely=thm_ly)
        therm_cam = cv2.VideoCapture(therm_input)
       ####### Acoustic Wave ######
        acoustic_label_name=tk.Label(root, text="Acoustic Waves", font='Helvetica %s
bold' %stn_font)
        acoustic_label_name.place(relx=act_x, rely=act_y)

        fig=plt.figure(figsize=(7,3))
        ax=fig.add_subplot(111)
        # Prepare the Plotting Environment with random starting values
        x = np.arange(10000)
        y = np.random.randn(10000)

        # Plot 0 is for raw audio data
        li, = ax.plot(x, y)
        ax.set_xlim(0,2*CHUNK)
        ax.set_ylim({-200,200})
        ax.set_title("Raw Audio Signal")
        canvas = FigureCanvasTkAgg(fig, master=root)
        canvas.get_tk_widget().place(relx=.3125, rely=act_ly)
####################################################################################
    def convert_image(self, rgb_image_label):
        global rgb_image
        global hsv_image
        rgb_im1 = Image.fromarray(rgb_image)
        rgb_im2 = ImageTk.PhotoImage(image=rgb_im1)
        rgb_image_label.configure(image=rgb_im2)
        rgb_image_label._image_cache = rgb_im2
        if rec_data:
            timearray1=time.strftime("%d_%m_%Y_%H_%M_%S")
            rgb_im3=rgb_im1.save('/media/wasp/Flashy/Robot_Sensors/RGB/'
+timearray1+'.jpeg')

        hsv_im1 = Image.fromarray(hsv_image)
        hsv_im2 = ImageTk.PhotoImage(image=hsv_im1)
        hsv_image_label.configure(image=hsv_im2)
        hsv_image_label._image_cache = hsv_im2
####################################################################################
    def thermal_vid(self):
        global therm_cam
        _, frame = therm_cam.read()
        cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
        cv2image = cv2.resize(cv2image, (cam_size,cam_size))
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)
        therm_image_label.imgtk = imgtk
        therm_image_label.configure(image=imgtk)
        imgtk._image_cache = imgtk
        if rec_data:
            timearray2=time.strftime("%d_%m_%Y_%H_%M_%S")
            img = Image.fromarray(cv2image)
            img2=img.convert('RGB')
            img2=img2.save('/media/wasp/Flashy/Robot_Sensors/Thermal/'
+timearray2+'.jpeg')
####################################################################################
    def convert_thermal(self):
        global thermal_image
        img = Image.fromarray(thermal_image)
        imgtk = ImageTk.PhotoImage(image=img)
        therm_image_label.imgtk = imgtk
        therm_image_label.configure(image=imgtk)
        imgtk._image_cache = imgtk
####################################################################################
    def update_fps(self, fps_label):
        frame_times = fps_label._frame_times
        frame_times.rotate()
        frame_times[0] = time.time()
        sum_of_deltas = frame_times[0] - frame_times[-1]
        count_of_deltas = len(frame_times) - 1
        try:
            fps = int(float(count_of_deltas) / sum_of_deltas)
        except ZeroDivisionError:
            fps = 0
        fps_label.configure(text='FPS: {}'.format(fps))
####################################################################################
    def audio_stream(self):
        global acoustic_process
        global audio_data
        acoustic_process=1
        print('stream begin')
        if keep_going:
            stream.start_stream()
            stream_data=stream.read(CHUNK, exception_on_overflow = False)
            stream.stop_stream()
            audio_data = np.fromstring(stream_data, np.int16)
            acoustic_process=0
            print('stream end')

            if rec_data:
                timearray3=time.strftime("%d_%m_%Y_%H_%M_%S")
                WAVE_OUTPUT_FILENAME='Acoustic'+timearray3+'.wav'
                wf =
wave.open('/media/wasp/Flashy/Robot_Sensors/Acoustics/'+WAVE_OUTPUT_FILENAME, 'wb')
                wf.setnchannels(CHANNELS)
                wf.setsampwidth(audio.get_sample_size(FORMAT))
                wf.setframerate(RATE)
                wf.writeframes(audio_data)
                wf.close()

######################################################################################
######################################################################################
def quit_(root):
    global proceed
    proceed=0
    print('Ending Python Code')
    stream.stop_stream()
    stream.close()
    audio.terminate()
    root.destroy()
    os.system("pkill python3")

def record_data():
    global rec_data
    global rec_setup
    rec_data = True

    if rec_setup==0:
        global user_name, usb_name, dirName
        try:
            os.makedirs('%s/RGB' %dirName)
            print("Directory " , dirName ,  " Created ")
        except FileExistsError:
            print("Directory %s/RGB"  %dirName ,  " already exists")

        try:
            os.makedirs('%s/Thermal'%dirName)
            print("Directory " , dirName ,  " Created ")
        except FileExistsError:
            print("Directory %s/Thermal"  %dirName ,  " already exists")

        try:
            os.makedirs('%s/Acoustics'%dirName)
            print("Directory " , dirName ,  " Created ")
        except FileExistsError:
            print("Directory %s/Acoustics"  %dirName ,  " already exists")

    print('Recording Turned ON')

def stop_data():
    global rec_data
    rec_data = False
    print('Recording Turned OFF')
```

```
def status():
    if rec_data==True:
            print('Record is On')
    if rec_data==False:
            print('Not Recording')

def update_image():
    global cam
    global rgb_image
    global hsv_image
    (readsuccessful, f) = cam.read(usb_cam_input)
    rgb_im = cv2.cvtColor(f, cv2.COLOR_BGR2RGB)
    rgb_image = cv2.resize(rgb_im, (cam_size,cam_size))
    hsv_im = cv2.cvtColor(f, cv2.COLOR_BGR2HSV)
    hsv_image = cv2.resize(hsv_im, (cam_size,cam_size))

def plot_data_setup():
    global audio_data
    global aud_prev
    print('plot begin S')
    length=range(0,len(audio_data))
    mlength=max(length)
    ax.plot(length,audio_data, linestyle='solid', marker='o', color='b')
    canvas.draw()
    plt.close()
    aud_prev=audio_data
    print('plot end S')

def plot_data():
    global audio_data
    global aud_prev
    ax.clear()
    print('plot begin')

    length=range(0,len(aud_prev))
    mlength=max(length)
    ax.plot(length,aud_prev, linestyle='solid', marker='o', color='b')
    ax.plot(range(mlength+1,mlength+len(audio_data)+1 ),audio_data, linestyle='solid',
marker='o', color='b')
    canvas.draw()

    plt.close()
    aud_prev=audio_data
    print('plot end')

########################################################################################
########################################################################################

######################################
############## PROGRAM ###############
######################################
root=tk.Tk()
root.title("Drone Data Collecting GUI")
root.geometry('1600x1250')
print('root begin')
app=Application(master=root)
app.mainloop()
```

Appendix I: Data Transfer and Audio Processing Program

```
import platform
import os
import sys
import time
import datetime
from tqdm import tqdm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from python_speech_features import mfcc, logfbank
import librosa

#User Needs To Define
######################
usb_name='Flashy'
sample_rate=44100
######################
time=time.strftime("%H:%M:%S")
time_array=[str(time[0:])]
dirName = '/home/agent/Desktop/Robot/Data_%s' %datetime.datetime.now().date()+
'_'+time[0:]
dirname_acous='/home/agent/Desktop/Robot/Data_%s' %datetime.datetime.now().date()+
'_'+time[0:]+'/Acoustics'
dirname_sig='/home/agent/Desktop/Robot/Data_%s' %datetime.datetime.now().date()+
'_'+time[0:]+'/Signal_Image'
dirname_fft='/home/agent/Desktop/Robot/Data_%s' %datetime.datetime.now().date()+
'_'+time[0:]+'/FFT_Image'
dirname_fbank='/home/agent/Desktop/Robot/Data_%s' %datetime.datetime.now().date()+
'_'+time[0:]+'/FBank_Image'
dirname_mel='/home/agent/Desktop/Robot/Data_%s' %datetime.datetime.now().date()+
'_'+time[0:]+'/Mel_Image'

mfccs={}
fbank={}
signals={}
ffts={}

def move_files(usb_name, dirname):
    print(dirName)
    try:
        os.makedirs(dirname)
        print("Directory " , dirname ,  " Created ")

    except FileExistsError:
        print("Directory " , dirname ,  " already exists")
        print(dirname)

    os.system('mv /media/agent/%s/Robot_Sensors/Acoustics %s' %(usb_name,dirname))
    os.system('mv /media/agent/%s/Robot_Sensors/RGB %s' %(usb_name,dirname))
    os.system('mv /media/agent/%s/Robot_Sensors/Thermal %s' %(usb_name,dirname))
    print('Files moved')

def create_folders():
    try:
        os.makedirs('%s/Signal_Image'%dirName)
        print("Directory " , dirName ,  " Created ")
    except FileExistsError:
        print("Directory %s/Signal_Image" %dirName ,  " already exists")
    try:
        os.makedirs('%s/FFT_Image'%dirName)
        print("Directory " , dirName ,  " Created ")
    except FileExistsError:
        print("Directory %s/Mel_Image"  %dirName ,  " already exists")

    try:
        os.makedirs('%s/FBank_Image'%dirName)
        print("Directory " , dirName ,  " Created ")
    except FileExistsError:
        print("Directory %s/FBank_Image"  %dirName ,  " already exists")

    try:
        os.makedirs('%s/Mel_Image'%dirName)
        print("Directory " , dirName ,  " Created ")
    except FileExistsError:
        print("Directory %s/Mel_Image"  %dirName ,  " already exists")


def plot_signals(signal):
    plt.close()
    fig, axes = plt.subplots(nrows=1, ncols=1, sharex=False,
                             sharey=True, figsize=(15,5))
    fig.suptitle('Signal', size=16)
    axes.set_title('Frequency VS Time')
    axes.plot(signals)
    axes.get_xaxis().set_visible(False)
    axes.get_yaxis().set_visible(False)
    plt.savefig(dirname_sig+('/')+filename[:-4])
    plt.cla
    plt.close()

def plot_fft(Y, freq):
    plt.close()
    fig, axes = plt.subplots(nrows=1, ncols=1, sharex=False,
                             sharey=True, figsize=(15,5))
    fig.suptitle('Fourier Transform', size=16)
    axes.set_title('Fourier Transform')
    axes.plot(freq, Y)
    axes.get_xaxis().set_visible(False)
    axes.get_yaxis().set_visible(False)
    plt.savefig(dirname_fft+('/')+filename[:-4])
    plt.cla
    plt.close()

def plot_fbank(fbank):
    plt.close()
    fig, axes = plt.subplots(nrows=1, ncols=1, sharex=False,
                             sharey=True, figsize=(15,5))
    fig.suptitle('Filter Bank Coeffienents', size=16)
    axes.set_title('Filter Bank Coeffienents')
    axes.imshow(fbank,
            cmap='hot', interpolation='nearest')
    axes.get_xaxis().set_visible(False)
    axes.get_yaxis().set_visible(False)
    plt.savefig(dirname_fbank+('/')+filename[:-4])
    plt.cla
    plt.close()

def plot_mfccs(mfccs):
    plt.close()
    fig, axes = plt.subplots(nrows=1, ncols=1, sharex=False,
                             sharey=True, figsize=(15,5))
    fig.suptitle('Mel Frequency Cepstrum Coefficients', size=16)
    axes.set_title('Mel Frequency Cepstrum Coefficients')
    axes.imshow(mfccs,
            cmap='hot', interpolation='nearest')
    axes.get_xaxis().set_visible(False)
    axes.get_yaxis().set_visible(False)
    plt.savefig(dirname_mel+('/')+filename[:-4])
    plt.cla
    plt.close()


def calc_fft(y,rate):
    n = len(y)
    freq=np.fft.rfftfreq(n, d=1/rate)
    Y = abs(np.fft.rfft(y)/n)
    return(Y, freq)

if platform.system() == 'Linux':
    print('LINUX')
    print(usb_name)
    print(dirName)
    move_files(usb_name, dirName)
    create_folders()

    f=0
    total=len(os.listdir(dirname_acous))
    for filename in os.listdir(dirname_acous):
        if filename.endswith(".wav"):
            #print(filename)
            signal, rate = librosa.load(dirname_acous+'/'+filename, sr=sample_rate)
            rate=int(rate)
            ffts=calc_fft(signal, rate)
            windlength=.025/25
            windstep=.01/25
            bank=logfbank(signal[:rate],rate,winstep=windstep, winlen=windlength,
nfilt=26, nfft=1103).T #44100/40
            mfccs = mfcc(signal[:rate],rate,winstep=windstep, winlen=windlength,
numcep=13,nfilt=26,nfft=1103).T
            signals=signal
            fbank=bank
            plot_signals(signals)
            plot_fft(ffts[0],ffts[1])
            plot_fbank(fbank)
            plot_mfccs(mfccs)
            f=f+1
            percent=int(f/total*100)
            print('Pecrent Completed:', percent,'%\t(',f,'out of',total,'files)')

            continue
        else:
            continue
else:
    print('Not Linux')
```

Appendix J: Machine Learning Program

```
%https://www.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-
classify-new-images.html
clc
clear all
close all

addpath
'/home/agent/Documents/MATLAB/Examples/R2019b/nnet/TransferLearningUsingGoogLeNetExamp
le';
disp('Lets Begin')

% Choose which category to perform machine learning
% RGB | Thermal | Acoustic
category= "Acoustic";

audioFolder='/home/agent/Desktop/Robot/Machine_Learning/Acoustic_Database';
rgbFolder='/home/agent/Desktop/Robot/Machine_Learning/RGB_Database';
thermFolder='/home/agent/Desktop/Robot/Machine_Learning/Thermal_Database';

aud_categories = {'Swing_Parrot', 'Quad_Parrot', 'Tello'};
rgb_categories = {'Swing_Parrot', 'Quad_Parrot', 'Tello'};
therm_categories = {'Swing_Parrot', 'Quad_Parrot', 'Tello'};

aud_imds = imageDatastore(fullfile(audioFolder, aud_categories),'LabelSource',
'foldernames');
rgb_imds = imageDatastore(fullfile(rgbFolder, rgb_categories),'LabelSource',
'foldernames');
therm_imds = imageDatastore(fullfile(thermFolder, therm_categories),'LabelSource',
'foldernames');

aud_tbl = countEachLabel(aud_imds);
rgb_tbl = countEachLabel(rgb_imds);
therm_tbl = countEachLabel(therm_imds);

aud_minSetCount = min(aud_tbl{:,2});
rgb_minSetCount = min(rgb_tbl{:,2});
therm_minSetCount = min(therm_tbl{:,2});

aud_imds = splitEachLabel(aud_imds,aud_minSetCount,'randomize');
rgb_imds = splitEachLabel(rgb_imds,rgb_minSetCount,'randomize');
therm_imds = splitEachLabel(therm_imds,therm_minSetCount,'randomize');


countEachLabel(aud_imds);
countEachLabel(rgb_imds);
countEachLabel(therm_imds);


aud_black_parrot=find(aud_imds.Labels == 'Swing_Parrot',1);
aud_quad_parrot=find(aud_imds.Labels == 'Quad_Parrrot',1);
aud_tello=find(aud_imds.Labels == 'Tello',1);


rgb_black_parrot=find(rgb_imds.Labels == 'Swing_Parrot',1);
rgb_quad_parrot=find(rgb_imds.Labels == 'Quad_Parrot',1);
rgb_tello=find(rgb_imds.Labels == 'Tello',1);


therm_black_parrot=find(therm_imds.Labels == 'Swing_Parrot',1);
therm_quad_parrot=find(therm_imds.Labels == 'Quad_Parrot',1);
```

```
therm_tello=find(therm_imds.Labels == 'Tello',1);

[aud_imdsTrain,aud_imdsValidation] = splitEachLabel(aud_imds,0.7);
[rgb_imdsTrain,rgb_imdsValidation] = splitEachLabel(rgb_imds,0.7);
[therm_imdsTrain,therm_imdsValidation] = splitEachLabel(therm_imds,0.7);
net = resnet50();
analyzeNetwork(net)

net.Layers(1)
inputSize = net.Layers(1).InputSize;

if isa(net,'SeriesNetwork')
  lgraph = layerGraph(net.Layers);
else
  lgraph = layerGraph(net);
end

[learnableLayer,classLayer] = findLayersToReplace(lgraph);
[learnableLayer,classLayer]

numClasses = numel(categories(aud_imdsTrain.Labels));
rgb_numClasses = numel(categories(rgb_imdsTrain.Labels));
therm_numClasses = numel(categories(therm_imdsTrain.Labels));

if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(rgb_numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,rgb_numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end

lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);

newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);

figure('Units','normalized','Position',[0.3 0.3 0.4 0.4]);
plot(lgraph)
ylim([0,10])

layers = lgraph.Layers;
connections = lgraph.Connections;

layers(1:10) = freezeWeights(layers(1:10));
lgraph = createLgraphUsingConnections(layers,connections);


pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange);

aud_augimdsTrain = augmentedImageDatastore(inputSize(1:2),aud_imdsTrain, ...
    'DataAugmentation',imageAugmenter);
aud_augimdsValidation = augmentedImageDatastore(inputSize(1:2),aud_imdsValidation);


rgb_augimdsTrain = augmentedImageDatastore(inputSize(1:2),rgb_imdsTrain, ...
    'DataAugmentation',imageAugmenter);
rgb_augimdsValidation = augmentedImageDatastore(inputSize(1:2),rgb_imdsValidation);


therm_augimdsTrain = augmentedImageDatastore(inputSize(1:2),therm_imdsTrain, ...
    'DataAugmentation',imageAugmenter);
therm_augimdsValidation =
augmentedImageDatastore(inputSize(1:2),therm_imdsValidation);

miniBatchSize = 10;
imageSize= net.Layers(1).InputSize;

%__________________  ACOUSTICS _______________________________
if category=="Acoustic"
    disp("Acoustic CNN Starting to Train")
    aud_valFrequency = floor(numel(aud_augimdsTrain.Files)/miniBatchSize);
    aud_options = trainingOptions('sgdm', ...
        'MiniBatchSize',miniBatchSize, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',3e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',aud_augimdsValidation, ...
        'ValidationFrequency',aud_valFrequency, ...
        'Verbose',false, ...
        'Plots','training-progress');

    aud_net = trainNetwork(aud_augimdsTrain,lgraph,aud_options);

    disp("Acoustics Convolutional Neural Network Trained");

    [aud_YPred,aud_probs] = classify(aud_net,aud_augimdsValidation);
    aud_accuracy = mean(aud_YPred == aud_imdsValidation.Labels);

    sprintf('Acoustic Confusion Matrix: ')
    aud_confMat = confusionmat(aud_imdsValidation.Labels, aud_YPred);
    aud_confMat_mod = bsxfun(@rdivide, aud_confMat,sum(aud_confMat,2));


    aud_idx = randperm(numel(aud_imdsValidation.Files),4);
    figure
    for i = 1:4
        subplot(2,2,i)
        aud_I = readimage(aud_imdsValidation,aud_idx(i));
        imshow(aud_I)
        aud_label = aud_YPred(aud_idx(i));
        title(string(aud_label) + ", " + num2str(100*max(aud_probs(aud_idx(i),:)),3) +
"%");
    end

disp("Beginning Testing Images");
aud_image_array={};
aud_image_array{1}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c12_04_2020_15_28_04_SParrot_V1.png';
aud_image_array{2}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c12_04_2020_15_37_59_SParrot_V1.png';
aud_image_array{3}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c12_04_2020_16_52_40_SParrot_V2.png';
```

```
aud_image_array{4}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c12_04_2020_16_55_22_SParrot_V2.png';
aud_image_array{5}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c25_04_2020_15_44_50_SParrot_V3.png';
aud_image_array{6}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c25_04_2020_16_13_02_SParrot_V3.png';
aud_image_array{7}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c25_04_2020_17_22_10_SParrot_V4.png';
aud_image_array{8}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c25_04_2020_17_23_59_SParrot_V4.png';
aud_image_array{9}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acousti
c27_05_2020_15_03_36_SParrot_V5.png';
aud_image_array{10}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic27_05_2020_15_21_48_SParrot_V5.png';
aud_image_array{11}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_13_41_27_QParrot_V1.png';
aud_image_array{12}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_14_50_46_QParrot_V1.png';
aud_image_array{13}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_16_39_44_QParrot_V2.png';
aud_image_array{14}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_16_41_23_QParrot_V2.png';
aud_image_array{15}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_15_56_23_QParrot_V3.png';
aud_image_array{16}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_16_01_10_QParrot_V3.png';
aud_image_array{17}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_17_37_07_QParrot_V4.png';
aud_image_array{18}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_17_40_31_QParrot_V4.png';
aud_image_array{19}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic03_05_2020_13_06_07_QParrot_V5.png';
aud_image_array{20}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic03_05_2020_13_16_40_QParrot_V5.png';
aud_image_array{21}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_15_02_18_Tello_V1.png';
aud_image_array{22}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_15_08_28_Tello_V1.png';
aud_image_array{23}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_16_46_32_Tello_V2.png';
aud_image_array{24}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic12_04_2020_16_48_42_Tello_V2.png';
aud_image_array{25}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_14_33_48_Tello_V3.png';
aud_image_array{26}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_14_37_36_Tello_V3.png';
aud_image_array{27}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_16_37_46_Tello_V4.png';
aud_image_array{28}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic25_04_2020_16_38_17_Tello_V4.png';
aud_image_array{29}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic27_05_2020_15_51_30_Tello_V5.png';
aud_image_array{30}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/Acoust
ic27_05_2020_16_54_22_Tello_V5.png';

aud_valLabels={'Swing_Parrot'; 'Quad_Parrot';'Tello'};

i=1;
aud__Post_Pred={};
aud_Post_Val={'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';'Swing_Parro
t';'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';...
```

```
'Swing_Parrot';'Swing_Parrot';'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';
'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';...

'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';'Tello';'Tello';'Tello';'Tello';'Tello';'Tel
lo';'Tello';'Tello';'Tello';'Tello'};


while(i<31)
    aud_newImage=imread(aud_image_array{i});
    aud_ds = augmentedImageDatastore(inputSize, ...
        aud_newImage,'ColorPreprocessing','gray2rgb');
    [aud_YPred,aud_probs] = classify(aud_net,aud_ds);

    sprintf('The loaded acoustic image belongs to %s class', aud_YPred)

    if('Swing_Parrot'==aud_YPred)
        aud_Post_Pred{i,1}='Swing_Parrot';
    end

    if('Quad_Parrot'==aud_YPred)
        aud_Post_Pred{i,1}='Quad_Parrot';
    end

    if('Tello'==aud_YPred)
        aud_Post_Pred{i,1}='Tello';
    end

    i=i+1

end

aud_Test_confMat = confusionmat(aud_Post_Val, aud_Post_Pred);
aud_Test_confMat_mod = bsxfun(@rdivide,aud_Test_confMat,sum(aud_Test_confMat,2));

figure
aud_training_cm=confusionchart(aud_confMat,aud_valLabels);
aud_training_cm.ColumnSummary = 'column-normalized';
aud_training_cm.RowSummary = 'row-normalized';
aud_training_cm.Title = 'Fifth Machine Learning Process: Acoustic Confusion Matrix of
Training Validation';

figure
aud_post_training_cm=confusionchart(aud_Test_confMat,aud_valLabels);
aud_post_training_cm.ColumnSummary = 'column-normalized';
aud_post_training_cm.RowSummary = 'row-normalized';
aud_post_training_cm.Title = 'Fifth Machine Learning Process: Acoustic Confusion
Matrix of Post-Training Validation';

end
%_________________________________________________________________________

%_____________________  RGB __________________________
if category == "RGB"
    disp("RGB CNN Starting to Train")
    rgb_valFrequency = floor(numel(rgb_augimdsTrain.Files)/miniBatchSize);
    rgb_options = trainingOptions('sgdm', ...
        'MiniBatchSize',miniBatchSize, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',3e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',rgb_augimdsValidation, ...
        'ValidationFrequency',rgb_valFrequency, ...
        'Verbose',false, ...
        'Plots','training-progress');

    rgb_net = trainNetwork(rgb_augimdsTrain,lgraph,rgb_options);

    disp("RGB Convolutional Neural Network Trained");

    [rgb_YPred,rgb_probs] = classify(rgb_net,rgb_augimdsValidation);
    rgb_accuracy = mean(rgb_YPred == rgb_imdsValidation.Labels);


    sprintf('RGB Confusion Matrix: ')
    rgb_confMat = confusionmat(rgb_imdsValidation.Labels, rgb_YPred);
    rgb_confMat_mod = bsxfun(@rdivide, rgb_confMat,sum(rgb_confMat,2));


    rgb_idx = randperm(numel(rgb_imdsValidation.Files),4);
    figure
    for i = 1:4
        subplot(2,2,i)
        rgb_I = readimage(rgb_imdsValidation,rgb_idx(i));
        imshow(rgb_I)
        rgb_label = rgb_YPred(rgb_idx(i));
        title(string(rgb_label) + ", " + num2str(100*max(rgb_probs(rgb_idx(i),:)),3) +
"%");
    end

disp("Beginning Testing Images");
rgb_image_array={};
rgb_image_array{1}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_2
020_15_28_05_SParrot_rgb_V1.jpeg';
rgb_image_array{2}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_2
020_15_28_20_SParrot_rgb_V1.jpeg';
rgb_image_array{3}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_2
020_16_52_39_SParrot_rgb_V2.jpeg';
rgb_image_array{4}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_2
020_16_55_24_SParrot_rgb_V2.jpeg';
rgb_image_array{5}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_2
020_14_45_24_SParrot_rgb_V3.jpeg';
rgb_image_array{6}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_2
020_15_47_38_SParrot_rgb_V3.jpeg';
rgb_image_array{7}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_2
020_17_21_57_SParrot_rgb_V4.jpeg';
rgb_image_array{8}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_2
020_17_23_56_SParrot_rgb_V4.jpeg';
rgb_image_array{9}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_05_2
020_15_02_48_SParrot_rgb_V5.jpeg';
rgb_image_array{10}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_05_
2020_15_22_51_SParrot_rgb_V5.jpeg';
rgb_image_array{11}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_13_28_37_QParrot_rgb_V1.jpeg';
rgb_image_array{12}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_13_41_28_QParrot_rgb_V1.jpeg';
rgb_image_array{13}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_16_41_06_QParrot_rgb_V2.jpeg';
rgb_image_array{14}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_16_41_23_QParrot_rgb_V2.jpeg';
rgb_image_array{15}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_15_53_24_QParrot_rgb_V3.jpeg';
rgb_image_array{16}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_16_02_03_QParrot_rgb_V3.jpeg';
rgb_image_array{17}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_17_36_54_QParrot_rgb_V4.jpeg';
```

```
rgb_image_array{18}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_17_38_40_QParrot_rgb_V4.jpeg';
rgb_image_array{19}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/03_05_
2020_13_02_39_QParrot_rgb_V5.jpeg';
rgb_image_array{20}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/03_05_
2020_13_06_16_QParrot_rgb_V5.jpeg';
rgb_image_array{21}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_15_02_37_Tello_rgb_V1.jpeg';
rgb_image_array{22}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_15_09_17_Tello_rgb_V1.jpeg';
rgb_image_array{23}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_16_46_43_Tello_rgb_V2.jpeg';
rgb_image_array{24}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04_
2020_16_48_42_Tello_rgb_V2.jpeg';
rgb_image_array{25}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_14_35_29_Tello_rgb_V3.jpeg';
rgb_image_array{26}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_14_38_01_Tello_rgb_V3.jpeg';
rgb_image_array{27}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_16_37_20_Tello_rgb_V4.jpeg';
rgb_image_array{28}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04_
2020_16_38_10_Tello_rgb_V4.jpeg';
rgb_image_array{29}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_05_
2020_15_35_20_Tello_rgb_V5.jpeg';
rgb_image_array{30}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_05_
2020_16_50_28_Tello_rgb_V5.jpeg';

rgb_valLabels={'Swing_Parrot'; 'Quad_Parrot'; 'Tello'};

i=1;
rgb_Post_Pred={};
rgb_Post_Val={'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';'Swing_Parro
t';'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';...

'Swing_Parrot';'Swing_Parrot';'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';
'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';...

'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';'Tello';'Tello';'Tello';'Tello';'Tello';'Tel
lo';'Tello';'Tello';'Tello';'Tello'};

while(i<31)
    rgb_newImage=imread(rgb_image_array{i});
    rgb_ds = augmentedImageDatastore(inputSize, ...
        rgb_newImage,'ColorPreprocessing','gray2rgb');
    [rgb_YPred,rgb_probs] = classify(rgb_net,rgb_ds);

    sprintf('The loaded rgb image belongs to %s class', rgb_YPred)
%
    if('Swing_Parrot'==rgb_YPred)
        rgb_Post_Pred{i,1}='Swing_Parrot';
    end

    if('Quad_Parrot'==rgb_YPred)
        rgb_Post_Pred{i,1}='Quad_Parrot';
    end

    if('Tello'==rgb_YPred)
        rgb_Post_Pred{i,1}='Tello';
    end

    i=i+1
end
rgb_Test_confMat = confusionmat(rgb_Post_Val, rgb_Post_Pred);
```

```
rgb_Test_confMat_mod = bsxfun(@rdivide, rgb_Test_confMat,sum(rgb_Test_confMat,2));

figure
rgb_training_cm=confusionchart(rgb_confMat,rgb_valLabels);
rgb_training_cm.ColumnSummary = 'column-normalized';
rgb_training_cm.RowSummary = 'row-normalized';
rgb_training_cm.Title = 'Fifth Machine Learning Process: RGB Confusion Matrix of
Training Validation';

figure
rgb_post_training_cm=confusionchart(rgb_Test_confMat,rgb_valLabels);
rgb_post_training_cm.ColumnSummary = 'column-normalized';
rgb_post_training_cm.RowSummary = 'row-normalized';
rgb_post_training_cm.Title = 'Fifth Machine Learning Process: RGB Confusion Matrix of
Post-Training Validation';
end
% %_________________________________________________________________________

%______________________ Thermal __________________________
if category == "Thermal"
    disp("Thermal CNN Starting to Train")
    therm_valFrequency = floor(numel(therm_augimdsTrain.Files)/miniBatchSize);
    therm_options = trainingOptions('sgdm', ...
        'MiniBatchSize',miniBatchSize, ...
        'MaxEpochs',6, ...
        'InitialLearnRate',3e-4, ...
        'Shuffle','every-epoch', ...
        'ValidationData',therm_augimdsValidation, ...
        'ValidationFrequency',therm_valFrequency, ...
        'Verbose',false, ...
        'Plots','training-progress');

    therm_net = trainNetwork(therm_augimdsTrain,lgraph,therm_options);

    disp("Thermal Convolutional Neural Network Trained");

    [therm_YPred,therm_probs] = classify(therm_net,therm_augimdsValidation);
    therm_accuracy = mean(therm_YPred == therm_imdsValidation.Labels);

    sprintf('Thermal Confusion Matrix: ')
    therm_confMat = confusionmat(therm_imdsValidation.Labels, therm_YPred);
    therm_confMat_mod = bsxfun(@rdivide, therm_confMat,sum(therm_confMat,2));

    therm_idx = randperm(numel(therm_imdsValidation.Files),4);
    figure
    for i = 1:4
        subplot(2,2,i)
        therm_I = readimage(therm_imdsValidation,therm_idx(i));
        imshow(therm_I)
        therm_label = therm_YPred(therm_idx(i));
        title(string(therm_label) + ", " +
num2str(100*max(therm_probs(therm_idx(i),:)),3) + "%");
    end

disp("Beginning Testing Images");
therm_image_array={};
therm_image_array{1}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04
_2020_15_28_06_SParrot_therm_V1.jpeg';
therm_image_array{2}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04
_2020_15_28_20_SParrot_therm_V1.jpeg';
therm_image_array{3}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04
_2020_16_52_39_SParrot_therm_V2.jpeg';
```

```
therm_image_array{4}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_04
_2020_16_55_24_SParrot_therm_V2.jpeg';
therm_image_array{5}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04
_2020_14_45_24_SParrot_therm_V3.jpeg';
therm_image_array{6}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04
_2020_15_47_38_SParrot_therm_V3.jpeg';
therm_image_array{7}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04
_2020_17_21_57_SParrot_therm_V4.jpeg';
therm_image_array{8}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_04
_2020_17_23_56_SParrot_therm_V4.jpeg';
therm_image_array{9}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_05
_2020_15_02_48_SParrot_therm_V5.jpeg';
therm_image_array{10}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_0
5_2020_15_22_51_SParrot_therm_V5.jpeg';
therm_image_array{11}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_13_28_37_QParrot_therm_V1.jpeg';
therm_image_array{12}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_13_41_28_QParrot_therm_V1.jpeg';
therm_image_array{13}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_16_41_06_QParrot_therm_V2.jpeg';
therm_image_array{14}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_16_41_23_QParrot_therm_V2.jpeg';
therm_image_array{15}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_15_53_24_QParrot_therm_V3.jpeg';
therm_image_array{16}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_16_02_03_QParrot_therm_V3.jpeg';
therm_image_array{17}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_17_36_54_QParrot_therm_V4.jpeg';
therm_image_array{18}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_17_38_40_QParrot_therm_V4.jpeg';
therm_image_array{19}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/03_0
5_2020_13_02_39_QParrot_therm_V5.jpeg';
therm_image_array{20}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/03_0
5_2020_13_06_16_QParrot_therm_V5.jpeg';
therm_image_array{21}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_15_02_37_Tello_therm_V1.jpeg';
therm_image_array{22}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_15_09_17_Tello_therm_V1.jpeg';
therm_image_array{23}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_16_46_43_Tello_therm_V2.jpeg';
therm_image_array{24}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/12_0
4_2020_16_48_42_Tello_therm_V2.jpeg';
therm_image_array{25}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_14_35_29_Tello_therm_V3.jpeg';
therm_image_array{26}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_14_38_01_Tello_therm_V3.jpeg';
therm_image_array{27}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_16_37_20_Tello_therm_V4.jpeg';
therm_image_array{28}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/25_0
4_2020_16_38_10_Tello_therm_V4.jpeg';
therm_image_array{29}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_0
5_2020_15_35_20_Tello_therm_V5.jpeg';
therm_image_array{30}='/home/agent/Desktop/Robot/Machine_Learning/Classify_Images/27_0
5_2020_16_50_28_Tello_therm_V5.jpeg';

therm_valLabels={'Swing_Parrot'; 'Quad_Parrot'; 'Tello'};

i=1;
therm_Post_Pred={};
therm_Post_Val={'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';'Swing_Par
rot';'Swing_Parrot';'Swing_Parrot';'Swing_Parrot';...
'Swing_Parrot';'Swing_Parrot';'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';
'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';...

'Quad_Parrot';'Quad_Parrot';'Quad_Parrot';'Tello';'Tello';'Tello';'Tello';'Tello';'Tel
lo';'Tello';'Tello';'Tello';'Tello'};

while(i<31)
%
    therm_newImage=imread(therm_image_array{i});
    therm_ds = augmentedImageDatastore(inputSize, ...
        therm_newImage,'ColorPreprocessing','gray2rgb');
    [therm_YPred,therm_probs] = classify(therm_net,therm_ds);

    sprintf('The loaded thermal image belongs to %s class', therm_YPred)


   if('Swing_Parrot'==therm_YPred)
        therm_Post_Pred{i,1}='Swing_Parrot';
    end

    if('Quad_Parrot'==therm_YPred)
        therm_Post_Pred{i,1}='Quad_Parrot';
    end

    if('Tello'==therm_YPred)
        therm_Post_Pred{i,1}='Tello';
    end

    i=i+1

end
therm_Test_confMat = confusionmat(therm_Post_Val, therm_Post_Pred);
therm_Test_confMat_mod =
bsxfun(@rdivide,therm_Test_confMat,sum(therm_Test_confMat,2));

figure
therm_training_cm=confusionchart(therm_confMat,therm_valLabels);
therm_training_cm.ColumnSummary = 'column-normalized';
therm_training_cm.RowSummary = 'row-normalized';
therm_training_cm.Title = 'Fifth Machine Learning Process: Thermal Confusion Matrix of
Training Validation';

figure
therm_post_training_cm=confusionchart(therm_Test_confMat,therm_valLabels);
therm_post_training_cm.ColumnSummary = 'column-normalized';
therm_post_training_cm.RowSummary = 'row-normalized';
therm_post_training_cm.Title ='Fifth Machine Learning Process: Thermal Confusion
Matrix of Post-Training Validation';


end
```

References

[1] Ortiz, Erik. “Newark Airport Drone Disruption Could Be Way of the Future.” *NBCNews, NBCUniversal News Group*. 23 January 2019. http://www.nbcnews.com/news/us-news/newark-airport-drone-disruption-could-be-way-future-n961761 (accessed 1 July 2020).

[2] Kesteloo, Haye. “The Ghost Drone That Shut down Barajas Airport in Madrid.” *DroneDJ*. 14 February 2020. https://dronedj.com/2020/02/14/the-ghost-drone-that-shut-down-barajas-airport-in-madrid/ (accessed 1 July 2020).

[3] Swales, Vanessa. “Drones Used in Crime Fly Under the Law's Radar.” *The New York Times*. 3 November 2019. https://www.nytimes.com/2019/11/03/us/drones-crime.html (accessed 1 July 2020).

[4] Harsha, Keagan. “Centennial Man Accused of Using Drone to Be a 'Peeping Tom'.” *FOX31 Denver*. 26 October 2019. https://kdvr.com/news/centennial-man-accused-of-using-drone-to-be-a-peeping-tom/ (accessed 1 July 2020).

[5] Captain, Sean. “Drones Try to Smuggle over $300K in Drugs across US Border.” *DroneDJ.* 5 May 2020. https://dronedj.com/2020/05/05/drones-try-to-smuggle-over-300k-in-drugs-across-us-border/ (accessed 1 July 2020).

[6] Kesteloo, Haye. “Two New Jersey Men Smuggled Drugs and Phones into Prison with Drones.” *DroneDJ*. 16 March 2020. https://dronedj.com/2020/03/16/two-new-jersey-men-smuggled-drugs-and-cell-phones-into-prison-with-drones/ (accessed 1 July 2020).

[7] “View All Products.” *DroneShield*. 2020. http://www.droneshield.com/view-all-products (accessed 2 July 2020).

[8] “Drone Detection & Defense Systems.” *DeTect, Inc*. https://detect-inc.com/drone-detection-defense-systems/ (accessed 25 June 2020)

[9] “ELP USB 3.0 2MP Sony IMX291 50fps High Speed Camera Module USB 3.0 Industrial with No Distortion Lens for Video Conference.” *ELP USB Webcam*. http://www.webcamerausb.com/elp-usb-30-2mp-sony-imx291-50fps-high-speed-camera-module-usb-30-industrial-with-no-distortion-lens-for-video-conference-p-249.html (accessed 25 September 2019).

[10] “FLIR LEPTON 3 & 3.5.” *FLIR*. 17 May 2018. https://www.flir.com/globalassets/imported-assets/document/lepton-3-3.5-datasheet.pdf (accessed 25 September 2019).

[11] “UMA-16 USB mic array.” *miniDSP*. 2019. https://www.minidsp.com/products/usb-audio-interface/uma-16-microphone-array (accessed 25 September 2019).

[12] “Garmin LIDAR-Lite v3HP: Distant Measurement Sensor.” *Garmin*. https://buy.garmin.com/en-US/US/p/578152 (accessed 25 September 2019).

[13] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition.”*2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV. 2016. pp. 770-778. doi: 10.1109/CVPR.2016.90 (accessed 15 February 2020).

[14] “resnet50.” ResNet-50 Convolutional Neural Network - MATLAB, The MathWorks, Inc. http://www.mathworks.com/help/deeplearning/ref/resnet50.html#:~:text=ResNet%2D50%20is%20a%20convolutional,%2C%20pencil%2C%20and%20many%20animals (accessed 15 February 2020).

[15] M. Ezuma, F. Erden, C. Kumar Anjinappa, O. Ozdemir, and I. Guvenc. "Detection and Classification of UAVs Using RF Fingerprints in the Presence of Wi-Fi and Bluetooth Interference." *IEEE Open Journal of the Communications Society*. vol. 1, pp. 60-76. 2020. doi: 10.1109/OJCOMS.2019.2955889 (accessed 3 August 2020).

[16] Ozturk, Ender, Fatih Erden, and Ismail Guvenc. “RF-Based Low-SNR Classification of UAVs Using Convolutional Neural Networks.” *ResearchGate*. 2020 (accessed 28 August 2020).

[17] Nguyen, Phuc, Mahesh Ravindranatha, Anh Nguyen, Richard Han, and Tam Vu. “Investigating Cost-effective RF-based Detection of Drones.” *ResearchGate*. 2020. doi: 17-22. 10.1145/2935620.2935632 (accessed 15 July 2020).

[18] Chen, V.C., Fayin Li, S. S. Ho, and Harry Wechsler. “Micro-Doppler Effect in Radar: Phenomenon, Model, and Simulation Study.” *IEEE Transactions on Aerospace and Electronic Systems*. 2006. doi: 10.1109/TAES.2006.1603402 (accessed 2 August 2020).

[19] Li, Chenchen and Hao Ling. “An Investigation on the Radar Signatures of Small Consumer Drones.” *IEEE Antennas and Wireless Propagation Letters*. vol. 16. pp. 649-652. 2017. doi: 10.1109/LAWP.2016.2594766 (accessed 2 August 2020).

[20] Rahman, Samiur, and Duncan Robertson. “Radar micro-Doppler signatures of drones and birds at K-band and W-band.” *Scientific Reports*. vol. 8. 26 November 2018. doi: 10.1038/s41598-018-35880-9. https://doi.org/10.1038/s41598-018-35880-9 (accessed 3 August 2020).

[21] Unlu, Eren, Emmanuel Zenou, Nicolas Riviere, and Paul-Edouard Dupuoy. "Deep learning-based strategies for the detection and tracking of drones using several cameras." *IPSJ Transactions on Computer Vision and Applications*. vol. 11. 24 July 2019. doi: 10.1186/s41074-019-0059-x (accessed 10 June 2020).

[22] Liu, Hao, Fangchao Qu, Yingjin Liu, Wei Zhao, and Yitong Chen. "A drone detection with aircraft classification based on a camera array." *IOP Conference Series: Materials Science and Engineering*. vol. 322. sp. 052005. 2018. doi: 10.1088/1757-899X/322/5/052005 (accessed 10 June 2020).

[23] Hengkang, Jin, and Yiwen Zhang. "Research on Feature Recognition of UAV Acoustic Signal Based on SVM." *Journal of Physics: Conference Series*. vol. 1302. sp. 022037. 2019. doi: 10.1088/1742-6596/1302/2/022037 (accessed 10 June 2020).

[24] Bernardini, Andrea, Federica Mangiatordi, Emiliano Pallotti, and Licia Capodiferro. "Drone detection by acoustic signature identification." *Electronic Imaging*. vol. 2017. pp. 60-64. 2017. doi: 10.2352/ISSN.2470-1173.2017.10.IMAWM-168 (accessed 12 June 2020).

[25] Polyzos, Konstantinos D., and E. Dermatas. "Real-Time detection, classification and DOA estimation of Unmanned Aerial Vehicle." *ResearchGate*. 2019. (accessed 28 August 2020).

[26] Vemula, Hari. "Multiple Drone Detection and Acoustic Scene Classification with Deep Learning." *Electronic Thesis or Dissertation*, Wright State University. 2018 (accessed 06 September 2020).

[27] Thompson, David John. "Maritime Object Detection, Tracking, and Classification Using Lidar and Vision-Based Sensor Fusion." *Dissertations and Theses*, Scholarly Commons. vol. 377. 2017. https://commons.erau.edu/edt/377 (accessed 7 July 2020).

[28] Kim, Byeong, Danish Khan, Cyril Bohak, Wonju Choi, Hyun Lee, and Min Kim. "V-RBNN based small drone detection in augmented datasets for 3D LADAR system." *Sensors*, MDPI. vol. 18. sp. 3825. 8 November 2018. doi: 10.3390/s18113825 (accessed 3 June 2020).

[29] Hammer, Marcus, Martin Laurenzis, and Michael Arens. "Lidar-based detection and tracking of small UAVs." *SPIE.* vol. 10799. 4 October 2018. doi: 10.1117/12.2325702 (accessed 17 September 2020).

[30] Svanström, Fredrik. "Drone Detection and Classification Using Machine Learning and Sensor Fusion." *DiVA*. 2020. http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-42141 (accessed 21 July 2020).

[31] "OPERATION MANUAL AND TECHNICAL SPECIFICATIONS." *LIDAR-LITE V3HP*, Garmin. http://static.garmin.com/pumac/LIDARLite_v3HP_Instructions_EN.pdf (accessed 25 September 2019).

[32] "UMA-16 Microphone Array." *miniDSP*. http://www.minidsp.com/images/documents/Product%20Brief-UMA16.pdf (accessed 25 September 2019).

[33] Miller, Zachariah W. "Streaming Audio with Python." *Zachariah W Miller, PHD*. 19 June 2017. http://zwmiller.com/projects/streamAudio.html (accessed 20 September 2019).

[34] Williams, Rohan. "Show Webcam Sequence TkInter." *Stackoverflow*. 6 June 2013. stackoverflow.com/questions/16366857/show-webcam-sequence-tkinter (accessed 10 September 2019).

[35] "Parrot Mambo Fly - Code, Pilot and Play." *Amazon*.https://www.amazon.com/Parrot-Mambo-Fly-Code-Pilot/dp/B074TGFML6?th=1 (accessed 1 December 2020).

[36] "Parrot Swing + Flypad." *Amazon*. http://www.amazon.com/Parrot-PF727003-SwingFlypad/dp/B01JYR44NS/ref=sr_1_4?dchild=1&keywords=parrot%2Bswing%2Bdrone&qid=1597002227&s=electronics&sr=1-4&th=1 (accessed 1 December 2020).

[37] "Parrot Minidrone Swing with Flypad Controller." *BH #PAIPF727003 • MFR #IPF727003*, B & H Foto & Electronics Corp. http://www.bhphotovideo.com/c/product/1274641REG/parrot_pf727003_minidrone_swing_with_flypad.html/specs (accessed 1 December 2020).

[38] "TELLO SPECS." *Tello*, RYZE. www.ryzerobotics.com/tello/specs (accessed 1 December 2020).

[39] Amini, Alexander, and Ava Soleimany. "Intro to Deep Learning." *6.S191: Introduction to Deep Learning*. 2020. http://introtodeeplearning.com/ (accessed 15 May 2020).

[40] Amini, Alexander, and Ava Soleimany. “Deep Computer Vision.” *6.S191: Introduction to Deep Learning*. 2020. http://introtodeeplearning.com/ (accessed 17 May 2020).

[41] Chatterjee, Chandra Churh. “Basics of the Classic CNN.” *Towards Data Science*, Medium. 31 July 2019. https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add (accessed 28 July 2020).

[42] Hasan, Md, Mustafa Jamil, Golam Rabbani, and Md Saifur Rahman. “Speaker Identification Using Mel Frequency Cepstral Coefficients.” *Proceedings of the 3rd International Conference on Electrical and Computer Engineering (ICECE 2004).* 2004 (accessed 29 May 2020).

[43] Fayek, Haytham. “Speech Processing for Machine Learning: Filter Banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between.” *Haytham Fayek*. 21 April 2016. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html (accessed 29 May 2020).

[44] Salomons, Etto, and Paul Havinga. “A Survey on the Feasibility of Sound Classification on Wireless Sensor Nodes.” *Sensors*, MDPI. vol. 15. pp. 7462-7498. 27 Februrary 2015. doi: 10.3390/s150407462 (accessed 29 May 2020).

[45] Adams, Seth. “Plotting & Cleaning - Deep Learning for Audio Classification p.3.” *Youtube*. 24 October 2018. https://www.youtube.com/watch?v=mUXkj1BKYk0&t=605s (accessed 15 February 2020)

[46] “Train Deep Learning Network to Classify New Images.” *MATLAB & Simulink*, The MathWorks, Inc. https://www.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html (accessed 16 January 2020).

[47] L. Fei-Fei, R. Fergus, and P. Perona. “Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories.” *IEEE*. CVPR 2004, Workshop on Generative-Model Based Vision. 2004 (accessed 28 February 2020).

[48] “Freesound General-Purpose Audio Tagging Challenge.” *Kaggle*. 2018. www.kaggle.com/c/freesound-audio-tagging (accessed 28 February 2020).