COOPERATIVE PERCEPTION IN MULTI-AGENT SYSTEMS

by

Gautham Vinod

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

School of Aeronautics and Astronautics

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Shaoshuai Mou

School of Aeronautics and Astronautics

Dr. Inseok Hwang

School of Aeronautics and Astronautics

Dr. Dengfeng Sun

School of Aeronautics and Astronautics

Approved by:

Dr. Gregory Blaisdell

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Shaoshuai Mou who helped guide my work on this thesis and supported all of my ideas. I would also like to thank my commitee members Dr. Inseok Hwang and Dr. Dengfeng Sun for serving on my committee and helping my research. I would like to thank my lab-mates Taashi Kapoor, Zehui Lu, and Tianyu Zhou.

I would specially like to thank Dr. Daniel DeLaurentis whose continued support and encouragement has not only enhanced my graduate student experience at Purdue but has also vastly impacted and helped my research. Lastly, I would like to thank Dr. Gregory Shaver who took me onto his team of graduate students for a high-impact industry research project and guided me to do great work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
YOLO	You Only Look Once
SABA	Secure Broadcasting and Acceptance Algorithm
CNN	Convolutional Neural Network
FC	Fully Connected
AIDER	Aerial Image Database for Emergency Response
ROI	Region of Interest
FPS	Frames per Second

ABSTRACT

This thesis presents work and simulations containing the use of Artificial Intelligence for Unmanned Aerial Vehicles in search and rescue and/or surveillance operations. The goal is to create a vision system that leverages Artificial Intelligence, mainly Deep Learning techniques to build a pipeline that enables fast and accurate classification of the environment of the robot. Deep Neural Networks are trained and tested on 'emergency situational data.' Further, the power of this vision system is leveraged to extend the problem onto a multi-agent system to handle fault tolerance. The multi-agent system is also made resilient to Byzantine malicious attacks to help improve the reliability of the system.

This thesis also shows the use of Artificial Intelligence for effective surveillance for defense-related purposes. Tracking the GPS coordinates of a boat using only the video of the boat captured by a camera and the GPS coordinates of the camera itself is demonstrated. The solution was tested by the Department of Defense - Department of the Navy, Naval Information Warfare Center Pacific.

1. INTRODUCTION

Unlike human beings, computers see the world in 1s and 0s. What a human may see as a cat or dog is something entirely different to a computer. That being said, computers have come a long way in terms of what it can understand. In fact, modern AI scientists say that the Turing Test, a test developed by Alan Turing [1] that checks whether a computer and a human are indistinguishable, is somewhat simplistic because computer intelligence has surpassed imitation of human conversations and ventured into many other territories.

A vision system is one such venture of advanced computing. A vision system essentially is a system that allows a computer to observe the world around it and perform specific actions in response to what it understands. The challenge here is that computers do not see images like what humans do. For a computer, an image is a 2-D or 3-D array of numbers. Each cell in this array represents a specific value for the intensity of light at a specific pixel. Therefore, image processing is a very complicated yet vital area of research for the future of computers.

On a big scale, image processing is performing specific operations on the input images from the data source to get meaningful output. But on a technical level, image processing is feeding arrays of numbers into the computer and writing programs to ensure that the computer can make sense of these numbers and find specific patterns that characterize a visual feature. 1.1 shows different image processing outputs from a single input image. These outputs vary based on what information needs to be extracted from the image. A lot of operations must be performed on each image to extract such information such as filtering noise, masking the image to find the specific characteristic that the user requires and several other operations depending on the expected outcome. Here, we face the second problem - computing power.

For a standard video which has about 30 frames per second, 30 individual images are processed each second. Therefore, for a computer vision task, 30 images need to be pro-

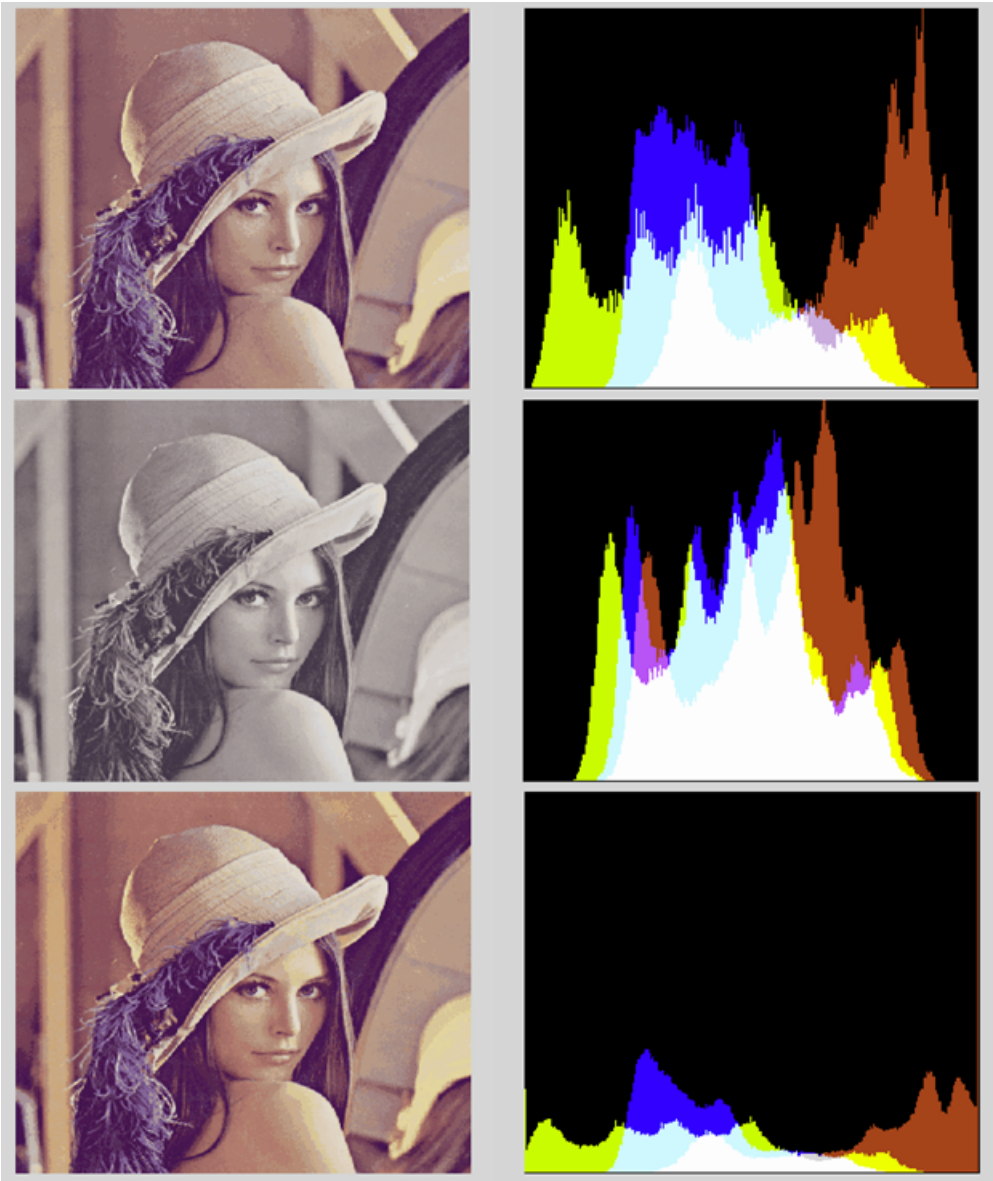


Figure 1.1. Different Image Processing Outputs. Adapted from [2]

cessed, information needs to be extracted and actions based on this information need to be performed 30 times in one second. For real-time video processing, the output or the processed images need to be similar to the frame-rate of the input, and hence needs to be around 30 images a second for a standard video. Hence, the processing time for each image needs to be minimal.

Here we make the trade-off between computing power and accuracy. Reducing the efficiency of a few of the operations would lessen the computational load. Hence each vision system is configured to a specific application, and the desirable characteristic, speed vs. accuracy is measured and chosen accordingly.

Now coming out of the world of image processing and into the world of computer vision [3], we find that the output required is entirely different. For image processing, the output required would be to to check whether a specific color exists in the input image or changing the filters in the image to highlight certain features. Fundamentally, for image processing, the input and output are both images. But computer vision faces a new set of problems.

Computer Vision is the ability of the computer to use visual input to perform certain tasks based on the qualitative and quantitative information obtained from the image. Taking the example of a robotic surface vehicle which would rely on a camera for visual input. The input from the camera is then used to understand the robot's surroundings and based on the obstacles detected in the image; the robot will have to change its trajectory and perform certain functions based on pattern recognition in the input.

Computer Vision has many applications in day to day life. A prevalent and useful application is to analyze the footage from a CCTV camera. A CCTV camera in a railway station or on the street or in apartments can be used to identify suspicious activities of people, monitoring proper law, finding people based on face recognition, counting the number of cars passing by in the street, identifying vehicles that jump a traffic signal, etc.

With the ubiquity of Unmanned Aerial Vehicles (UAVs) and the power of computer vision, we can combine them to build a vision system for UAVs that enables them to see and understand the world around them. A pressing issue that can be addressed with this combination is using UAVs as first responders in emergency situations and search and rescue operations.

The main task in such situations is creating a vision system that can be used by UAVs to understand its environment. Further, the usage of multiple UAVs instead of a single UAV in such situations is also studied in this thesis. This thesis shows that using multiple UAVs and having them secure to malicious attacks provides a complete, safe, and robust system that can be readily deployed in high-tension situations. Further, as this thesis looks at the software framework for such a system, the solution can be extrapolated to other type of robots in other tasks as well with minimal changes.

1.1 Motivation

Today, advanced and autonomous robots are shooting up in popularity thanks to three things: Sensors, Actuators, and AI [4]. The AI component is vital to the development of such robots as it increases the 'smartness' of modern robots. We have many applications for such robots even as first responders [5]. Mainly, UAVs and Unmanned Ground Vehicles (UGVs) are used as they can be completely autonomous, highly maneuverable, and can easily access difficult terrain. UAVs also provide the added advantage of an aerial view over uneven and obscured terrain. Therefore, UAVs have become popular in search-and-rescue situations [6], [7]

Figure 1.2. Drones in Search and Rescue Operations. Adapted from [7]

One of the key challenges that UAVs and other such autonomous robots face in emergency situations is the lack of infrastructure to communicate a cloud-based server. This

constraints these robots to rely solely on their on-board processing capabilities [8]. The small size of UAVs and the corresponding lack of processing power makes it very difficult to run computer vision models that use the power of deep learning on-board these UAVs. We cannot rely on other forms of image processing as the environment of UAVs is highly dynamic such that a 'one size fits all' solution is not feasible. Hence, there is a need to build a model that can analyze the images taken from the camera of the UAV and give certain parameters as the output that enables the drone to understand the content of the image.

Here, a Convolutional Neural Network needs to be employed that is both accurate and fast. Therefore, our first task is to create a model that matches these constraints and can easily be deployed on-board UAVs. One model will help a single UAV make sense of its surroundings. We develop this problem further to incorporate a multitude of UAVs for example a multi-agent system. A multi-agent system is defined as a group of interacting intelligent agents working to achieve a common goal.

We know that no model can accurately depict the realism of our environment but we can come close to it. Every agent that is deployed with our computer vision model is susceptible to inaccuracies because of the accuracy if the model, noise in the camera images, occlusions in the image, camera sensor problems, exposure problems and many other factors. For a single-agent these flaws are always amplified as there is no contingency to keep a check on the output of the network. However, in the case of multi-agent systems, each agent interacts with its neighbours and such inaccuracies in the output of an agent can confuse the system and as a result the system might not achieve its common goal. In order to avoid this we need to ensure that the agents work together to balance the inaccuracies of single agents and agree upon values as a group. Here, all agents in the system agreeing upon the same value is termed as 'consensus', so for the system to achieve consensus, every agent in the system needs to agree on the same value which in this case is the output of the computer vision model.

Taking our problem a step further, we want to introduce resilient consensus into the system. Resilient consensus is a systems ability to achieve consensus even when the system is under attack by malicious agents. In emergency situations it is possible that a system could malfunction and produce erogenous values or that agents in a system can be hacked/attacked by malicious entities to drive the system away from consensus. Therefore, we need algorithms to ensure that the system achieves consensus even under such attacks. Here we adopt different techniques to ensure that the output of our vision system achieves resilient consensus. There is a need to improve existing resilient algorithms to ensure compatibility of the values in our case which is the output of the vision system. The requirements of these algorithms and our system will be detailed in the later sections.

Finally, we extend this concept to create a new algorithm that is capable of tracking boats in the sea. This extension is part of a challenge put forth by the the United States Department of Defense - Department of the Navy, Naval Information Warfare Center Pacific [9]. The motivation behind this challenge is to autonomously track the GPS coordinates of a boat as observed from the camera of an Unmanned Marine Vehicle. Here, the only known inputs are the GPS coordinates of the camera and the video feed from the camera that contains the boat/vessel to be tracked. An algorithm must identify the boat in the video and then show the GPS coordinates of that boat. This would enable the Unmanned Marine Vehicle to stealthily gather and provide information about the GPS coordinates of another boat.

1.2 Literature Review

Image classification is a common problem with many state-of-the-art networks that are capable of classifying images with great accuracy. Starting with the structural foundation for modern convolutional neural networks laid by Alex [10] called AlexNet. The other famous higher accuracy networks include VGG [11], ResNet [12], Inception [13] and a few others. There are also high-speed networks that focus on performance by compromising on accuracy

such as the MobileNet [14].

Other methods for image classification exist that do not rely on neural networks for the entire classification and rather uses neural networks as a feature extractor and then classifies based on regression models [15] to classify images. There are also matrix ranked methods that project matrix data onto left and right vectors which upon further regression can be classified into one of many classes [16]. This section aims to highlight the work involved specifically in Computer Vision systems for UAVs and the multi-agent application of this problem. This section also shows the common algorithms in resilient consensus and how systems handle different adversarial attacks. Finally, it shows the work in tracking objects through computer vision.

1.2.1 Multi-agent Image classification

Consensus algorithms are used in multiple domains to ensure system security. For example, the Bitcoin relies on block chain technology that employs consensus algorithms to ensure the system is secure and stable [17]. Although blockchain technology uses very complicated algorithms to ensure system security, regular systems such as UAVs rely on algorithms that aren't as computationally intensive. One of the most common ways to achieve consensus is the average consensus algorithm introduced by Xiao et al [18]. Here, each agent updates its states based on a certain update rule that looks at the difference between the agent's state and its neighbours' states. For a multi-agent classification system, achieving consensus among all the agents in the system would be important to help the system give a single output than each agent giving its own output. Other methods to tackle this problem would involve combining the outputs of the agents in different ways to produce a single output or processing the input differently to obtain similar outputs.

The work of Mousavi et al [19] is a multi-agent classification problem that requires different agents to observe partial views of the image based on the pose. Then, a network is designed that helps the system form local beliefs for the purposes of reinforcement learn-

ing, take local actions, and extract relevant features from the agents' (partial) observations. Further, consensus protocols are run to update each agent's states based on its neighbours' states for the entire system to utilize reinforcement learning to achieve decentralized consensus on the classification of the image. Here, the main parameters are the area of observation of each agent (pixel area), the number of agents in the system, and the time passed since observation had begun. The limitations in this approach is that greater accuracy is achieved only when the number of agents are higher. Also, in our case of an emergency situation, it would be an unwise choice to make certain agents observe only a part of the environment and then rely on consensus. It would also be impractical in terms of coordinated path-planning for a system that will rely on high computations will also need high level of path-planning and coordination algorithms to run simultaneously.

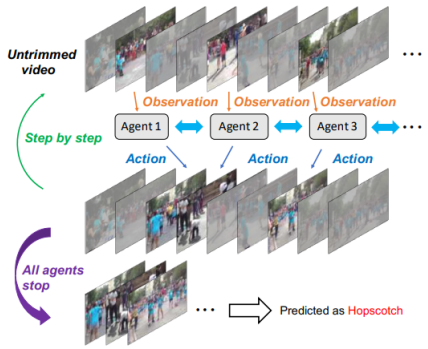


Figure 1.3. Sampling location adjustment of each agent based on context information. Adapted from [20]

Another such method is detailed by Pourpanah et al [21] where Q-learning [22] and a Bayesian formalism [23] is used for formulating trust-measurement. Video analysis is another method that is utilized to become a multi-agent image classification problem [20] where different agents process different frames in a video. Here, the frame sampling is based on multiple parallel Markov processes which helps the agents pick the frame to be analyzed. A reinforcement learning algorithm is formulated which models the information extracted by a single agent with that of its neighbours and also utilizes the historical states of agents, the action space, a policy network which ultimately gives the final network. A detailed schematic

of the logic in video analysis is shown in Figure 1.3. The limitations here are again that each agent views a different image which would mean that the information would have to extracted over time which is very costly in emergency situation where time is of the essence. Also, proper communication with each agent is a necessity in this case and more number of agents would mean more number of frames. Matching frames from different vantage points could also be a complicated task.

In general, multi-agent image classification usually relies on reinforcement learning algorithms to classify images based on certain conditions such as partial observation of the image space or different frames of videos. Further, accurate consensus often puts a constraint on higher number of agents in the system.

1.2.2 Resilient Consensus

There are many types of attacks that a system of agents can incur. Random attack targeting agents, spoof attacks, intelligent attacks, Denial of service attacks, etc. Having mentioned the consensus protocols in cryptocurrency in the previous section, blockchain technology relies on resilient consensus for its security [24]. These focus on a specific type of attack called Byzantine Attacks [25] where a single agent or multiple agents in a system of agents is attacked and faulty values/information is sent to the other agents in the system. The purpose of such attacks is to ensure that the system does not reach consensus. Bitcoin addresses this problem by using a consensus protocol called Proof of Work [26]. This algorithm and many similar algorithms are very computationally intensive and is also the reason that mining bitcoin consumes a lot of energy and requires a lot of time.

For the purposes of deploying resilient consensus protocols on-board drones such computationally expensive algorithms will not work and we hence we rely on smaller algorithms that model such adversarial attacks using certain assumptions. One such example is using an algorithm is proposed by Dibaji et al [27] where the authors develop a Secure Broadcasting

and Acceptance Algorithm (SABA) which employs a voting system that each agent accepts a value from its neighbour only if it receives an identical value for that node from other $f + 1$ agents where f is the total number of agents that are under attack in the system. The works in [27], [28] focus on achieving average consensus in time-varying networks using similar protocols.

Other forms of obtaining average consensus for multi-agent systems include assuming a few trustworthy agents in the system that cannot be compromised which dominate the connections in the network [29], a convex combination based resilient algorithm where the identity if the attacker is unknown [30], [31] where the intersection of convex hulls of the states of agents help drive the system towards the average of the states. Another work that focuses on reaching consensus without identifying the attacking agent(s) is described by Hou et al [32]. This work uses a reinforcement learning type method to learn the weights in the adjacency matrix of the system over time based on the differences in the states of the neighbours. Links between agents with a bad trust factor are given less weights and trusted links have more weights. These weights and the states of the agents are used to bring the system to consensus.

Different resilient consensus protocols have different assumptions and work under different conditions. Identification of the attacking agent(s) would lead to the exact average consensus of all initial states of agents whereas obtaining consensus without identification would lead to skewed values of the average but the system achieves consensus still. Therefore, we look at light consensus algorithms that protect the system from Byzantine attacks and drives the values of the system towards the average of the initial states.

1.2.3 Object Tracking

A survey of object tracking problems and methods is detailed in [33]. The common methods include object detection and/or video segmentation. Segmentation is an intensive task both for training and deployment. Brendel et al [34] describe a segmentation tracking

approach that tracks regions in the images. Our mission of finding lightweight solutions leads us to use object detection and corresponding actions based on the detection of objects. Also, the requirements for our tracking problem involves tracking boats at sea. The work of Kruger et al [35] uses thermal images from a thermal camera to track boats at sea. The only example of a full-functioning system for tracking boats is detailed in [36] although it is utilized for traffic monitoring rather than obtaining GPS coordinates of the boat as is required by our problem.

1.3 Contributions

The main contributions of this work are:

- A computer vision system that classifies emergency situations to be deployed on UAVs and achieves multi-agent consensus on the classification result for a system of agents to improve the accuracy of the system
- Extension of multi-agent image classification towards resilient consensus where the system achieves average consensus of the output of the image classification system even under Byzantine attacks. Here, a new algorithm is developed that reduces certain assumptions required by other works and significantly reduces the number of agents required to achieve consensus.
- A novel algorithm that combines object detection with GPS coordinate tracking for the purposes of tracking boats at sea solely through video analysis from a regular mono-ocular camera.

2. MULTI-AGENT IMAGE CLASSIFICATION

2.1 Introduction

In this chapter, the key idea and model behind image classification is addressed. Different techniques for image classification is introduced and the combination of multi-agent systems and image classification is addressed. The classification accuracy is compared between a single agent and a group of agents.

2.1.1 Introduction to Image Classification

Image classification through Deep Learning relies on using Convolutional Neural Networks that produce an output based on the image that is given as the input [37]. It helps to understand what neural networks are and how they help in image classification. A convolutional neural network takes an image input which is a 2D or a 3D matrix and classifies the image into different classes. The output is a set of probability values that give the probability of the image belonging to each class. Figure 2.1 shows a simple image classifier using different types of layers to classify beverages as either coffee, tea, or other.

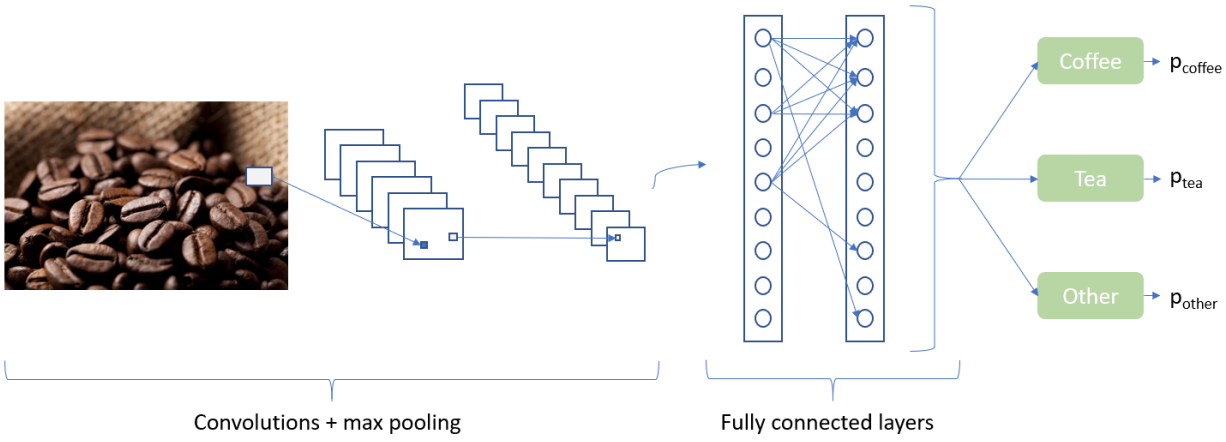


Figure 2.1. Convolutional Neural Network for Image Classification

Layers in a Convolutional Neural Network

The different layers in a typical convolutional neural network are shown in Figure 2.2 where each layer has different mechanisms for information flow.

Figure 2.2. Layers in a typical CNN

- **Input Layer**

  The input layer is image data that is represented as a 3D matrix for colored images and a 2D matrix for grayscale images. The dimensions of the matrix $l \times d \times c$ where l, d, and c are the length, width and channels of the image respectively. The channels for a normal image are RGB - Red, Green and Blue. These values are what determines the intensity and color of a certain pixel at a certain location on a screen.

- **Convolutional Layer**

  The convolutional operator is defined in mathematics as:

$$(f * g)(t) = \int_{\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

  In matrices, this corresponds to multiplication and addition where the kernel $g(t)$ is placed on the input $f(t)$ and the dot product of all corresponding values becomes a single value as shown in Figure 2.3a in the output. Then the kernel slides over the filter to the next location and the same operation is performed. The important parameters here are the size of the kernel, the stride which is the number of steps the kernel moves for each convolution.

  This operation behaves like a pattern feature extractor. Typically, the number of channels increases as we go deeper into the network. This is done by performing

the same convolution operation using n different kernels to obtain n channels as the output.

- **Pooling Operator**

  Pooling is usually down to downsample the data. As mentioned earlier, the number of channels typically increases across the network as we increase the number of convolutions. But we also decrease the length and width of the image across the network. This is done through downsampling which helps us pick out the essential features leaving aside the unimportant features. A common method of downsampling is done using the 'Max-pooling operator' which picks out the maximum value from the input for a given kernel size and then slides this kernel to the next set of values in the input to continue the same process as shown in Figure 2.3b. Another pooling operator is 'Average Pooling' where as the name suggests, the average of the

- **Fully-Connected Layers**

  In a fully connected layer, we have multiple neurons (values) in a single layer. Each of these neurons are connected to all the neurons in the next layer by a matrix of multipliers called *Weights*. Once these values are multiplied by the weights, a bias is added on each neuron and then passed through a non-linearity function. We will talk about the non-linearity functions in the next subsection. The formula that defines the flow of information in these layers are:

$$f\left(b + \sum_{\mathrm{i}=1}^{n} x_{\mathrm{i}} w_{\mathrm{i}}\right)$$

  where b - bias

  x - Value of input neurons

  w - weights

  n - Number of inputs from the incoming layer

  $f(.)$ - non-linearity function

  This output is the value of the neuron in the next layer.

- **Softmax**

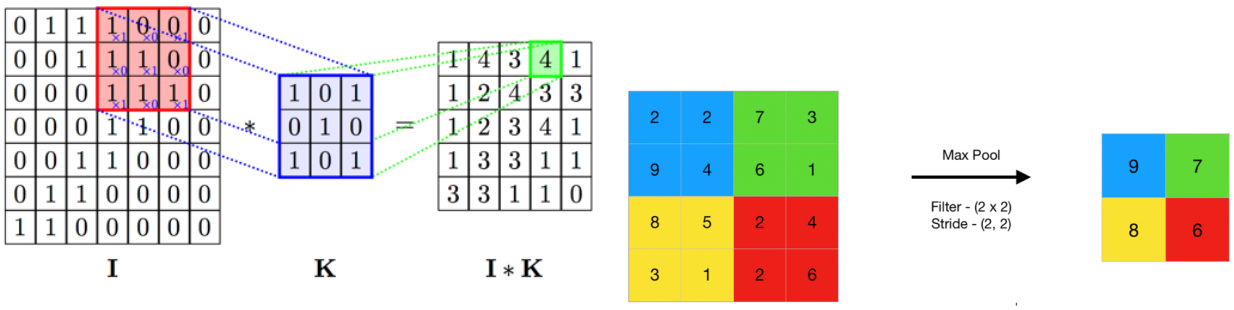
  The softmax function is usually present at the end of the last fully connected layer. These fully connected layers are defined in such a way that the last fully-connected layer will have the same number of neurons as the number of classes in the image dataset. The value from each of these neurons is passed through a softmax function given by the formula:

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{k=1}^{n} e^{y_j}}$$

  where y_i is each output from the last layer. The softmax outputs the probability values and the sum of all the outputs from the softmax function will equal 1. This function helps us in determining the probability of an image belonging to each class. The highest probability value is the class of the input image.

- **Output**

  It is the class of the image as predicted by the neural network. This is then used to backpropogate the error through the network.



(a) Convolutional Operator (b) Max-Pooling Operator

Figure 2.3. Operators in CNNs

Working of a Neural Network

- **Non-Linearity Functions**

A non-linear function decides the activation of a certain neuron. An output of 0 means that the neuron will be inactive whereas any other output will be passed to the next operation. A very common example of a non-linear function is the ReLU function (Rectified Linear Unit) [38]. The ReLU function is defined as $R(z) = \max(0, z)$ and the graph is shown in Figure 2.4.

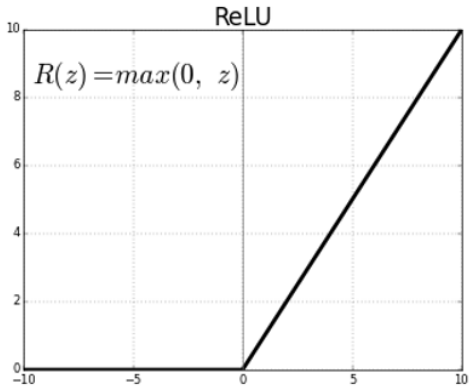


Figure 2.4. ReLU Function. Adopted from [39]

- **Loss Functions**

  Perhaps the most important part of a neural network is the loss function. Loss functions help us develop a measure between the values of the output and the all the trainable parameters in the network [40]. Typically, we train networks using the stochastic gradient descent algorithm. This optimization algorithm essentially calculates the negative of the gradient of the loss function with respect to the weights. This will lead the function towards the minima of the loss function (refer Figure 2.5). We represent the trainable hyper-parameters by θ. The pseudo-code that updates the weight w.r.t loss function represented by $J(\theta)$ is given by:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

  Here, α is the learning rate. The most common loss function is the Mean squared error loss and the Cross-entropy loss for classification problems.

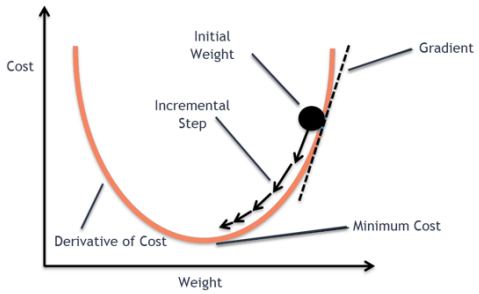


Figure 2.5. Gradient descent loss function

2.1.2 Transfer Learning

The structure of a typical neural network, the method of functioning, and the learning parameters were detailed in the previous subsection. This section will talk about a method to achieve image classification without having to go through the tedious process of building a network from scratch and further training the network to suit one's needs. Instead, this method lets you use existing networks that are state-of-the-art and have already been trained on large publicly available datasets [41]. The reason transfer learning is used is two fold:

- The more training data that is available to a network, the better it will learn. So if a particular application of a network only has a small dataset for training, it would be better to use the transfer learning method
- In a typical CNN, the first few convolutional layers perform the function of feature extractors such as detecting edges and shapes in images. The further layers will extract more complicated features such as patterns, faces, etc. It is the FC layers in the end of the network that is useful in classification. So for the large part, the first few layers can be kept constant for any CNN for image classification as it would have learned, through training on large datasets, how to extract the right features from images.

Essentially, transfer learning helps us modify existing pre-trained networks and then re-train it to suit any particular application. These networks are usually trained on the ImageNet dataset [42] which is a large dataset with over a million images in about 1000

classes. Obviously, training any network on this dataset will take a lot of time so transfer learning helps in reducing training time by using pre-trained networks whose weights are known.

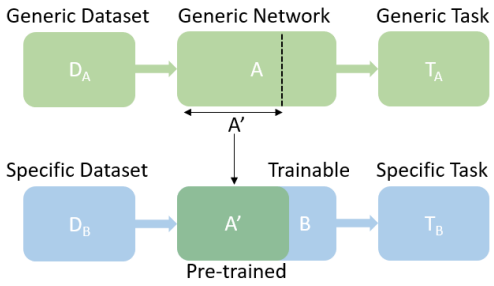


Figure 2.6. Transfer Learning

The illustration in Figure 2.6 shows how a network that has learned to perform a general task by being trained on a general dataset can be modified to use a few of the learned parameters on a new specific task by being trained on a specific dataset.

2.1.3 Multi-agent Consensus

We are fairly familiar with the concept of UAV swarms where multiple agents help out to perform a task. The reason this problem is extended to multi-agent systems is because:

- False Positive - A single UAV reliant on its vision system might falsely classify its given environment as an emergency situation because of the angle of incidence, color corrections, lighting, exposure, camera faults or just the accuracy of the vision system itself.
- Backup - In the case one UAV fails, sending multiple UAVs to such emergency situations definitely increases the chances of a successful mission.
- Accuracy - Using multiple UAVs instead of a single UAV could potentially increase the accuracy of the image classification as we will see in the later sections.

- Efficiency - UAVs can distribute the load of computing amongst themselves which could potentially increase the battery life of the system as a whole by lessening the load on the processors.

Therefore, it is hypothesized that introducing a multi-agent system will benefit the emergency response system and imporive the image classification.

2.2 Problem Formulation

2.2.1 CNN Image Classification Problem Formulation

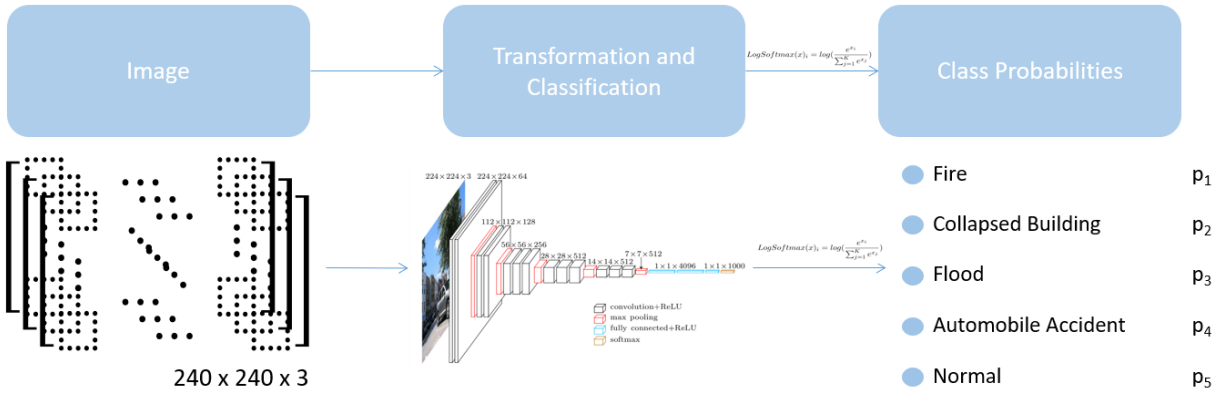


Figure 2.7. Image Classification Problem

The problem is founded in building a vision system that can be easily deployed on UAVs and accurately classify images based on emergency situations. Since the reliability of internet is not present in emergency situations we will rely on processing the images on-board the UAVs. This will mean that a completely new network will need to be build that can classify images in real-time through the camera present in the UAV.

Therefore the first part of our problem can be considered as transforming an image to class probabilities. Whatever the size of the input image is, the output must have 5 class probabilities - Fire, Collapsed Building, Flood, Automobile Accident, and Normal. This problem is detailed in Figure 2.7

The model will use a CNN to deliver the required image classes through a process of training the network. Training the network involves using training images on the network

that utilizes the loss function to correct the weights. This will be done repeatedly over all the training images until the loss is at a minima and the accuracy of classification is the highest. This process is shown in Figure 2.8

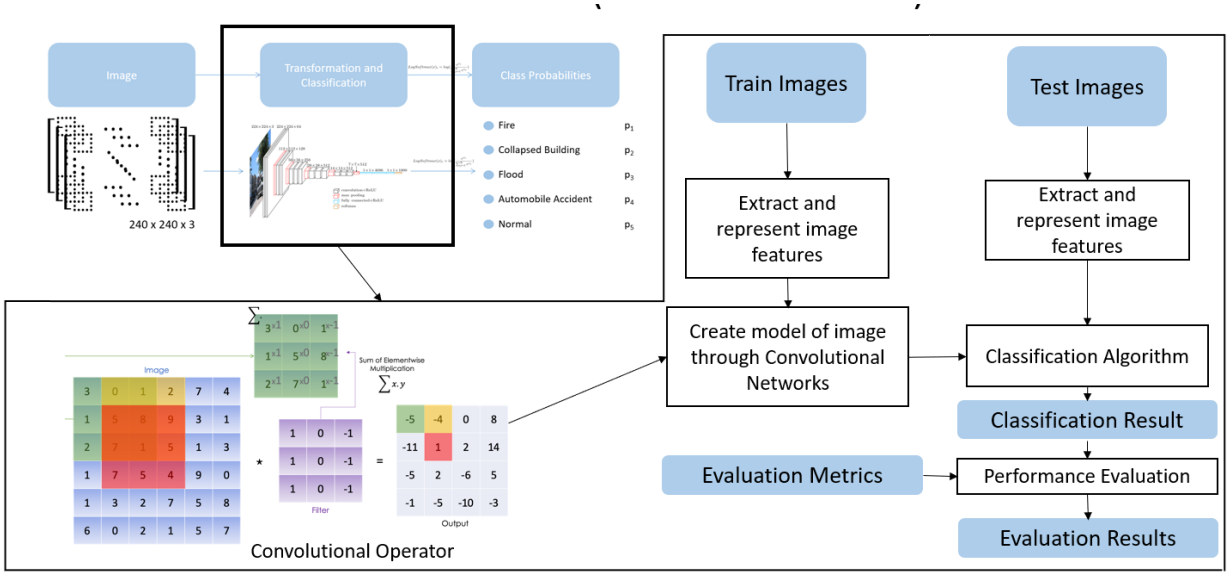


Figure 2.8. Training the network

For the sake of consistency and compatibility with a particular model, we will set the dimensions of the input image to 224×224 and the output will be a vector of dimensions 5×1.

This vector will be in the form $\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}$ where

p_1 - probability that image is fire

p_2 - probability that image is collapsed building

p_3 - probability that image is flood

p_4 - probability that image is automobile accident

p_5 - probability that image is normal

2.2.2 Transfer Learning Problem Formulation

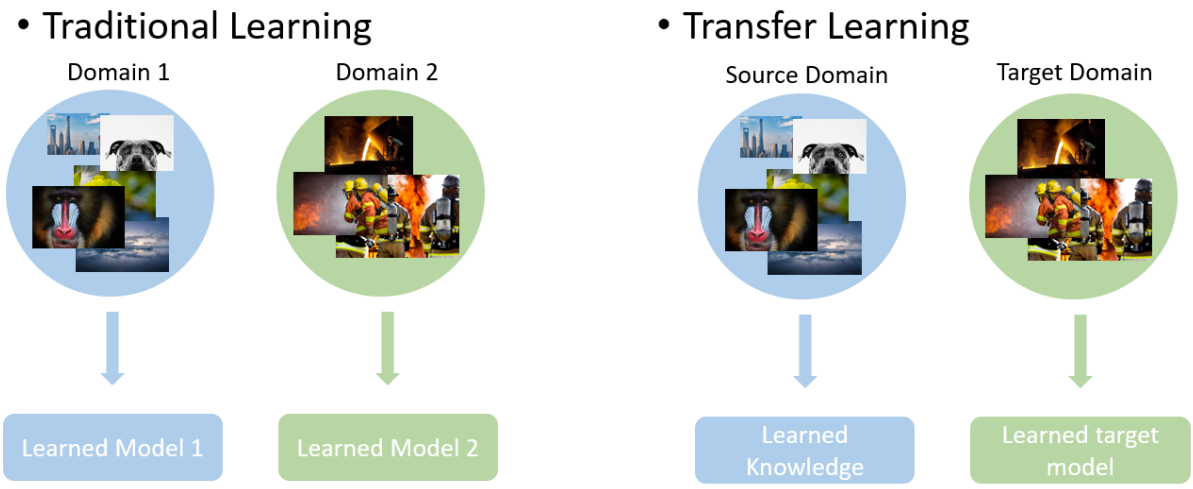


Figure 2.9. Training the network

The other method that is used here is transfer learning where the problem is still getting the same output from the same input as the regular CNN case. However, there is an additional problem in this case. In traditional learning approaches, when there are 2 different domains (data) for 2 different tasks, generally 2 different models are created and are made to learn on their respective domains. However, for transfer learning we need to use the learned knowledge of a source domain and transfer it to a target model.

To put it down mathematically, from [43], we define domains and tasks. Domains are represented by $\mathcal{D}$ and consists of a feature space $\mathcal{X}$ and a marginal probability distribution over this feature space $P(X)$ where $X = x_1, x_2, .., x_n \in \mathcal{X}$. For example, in an image classification problem, $\mathcal{X}$ is the space of all images, X are all the images samples that will be used for training and $x_{\text{i}} \in \mathcal{X}$ is each image in the training dataset.

Now, given a domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task $\mathcal{T}$ is defined which has its own space $\mathcal{Y}$ called a label space and a conditional probability distribution $P(Y|X)$. This probability distribution is usually learned from the training data and the label pair $x_{\text{i}} \in X$ and $y_{\text{i}} \in \mathcal{Y}$.

In the same example as above, $\mathcal{Y}$ is a set of classes for the image classification problem.

If we have a source domain $\mathcal{D}_s$ and a corresponding task $\mathcal{T}_s$, we also a define a target domain and task $\mathcal{D}_t$ and $\mathcal{T}_t$. The problem that transfer learning would need to solve is to learn the conditional probability distribution $P(Y_t|X_t)$ in the taget domain $\mathcal{D}_t$ with the information it learned from $\mathcal{D}_s$ and $\mathcal{T}_s$ where the constraint is that either $\mathcal{D}_s \neq \mathcal{D}_t$ or $\mathcal{T}_s \neq \mathcal{T}_t$.

After learning this probability distribution, the output of the network will still give us a 5×1 vector as the probabilities of an image belonging to each of the 5 classes that was defined earlier.

2.2.3 Consensus Problem Formulation

From the literature review section, we have identified the different works and methods used for achieving average consensus. In this section too, we will be focusing on achieving average consensus. We will define the consensus problem with respect to the UAV in emergency response situation.

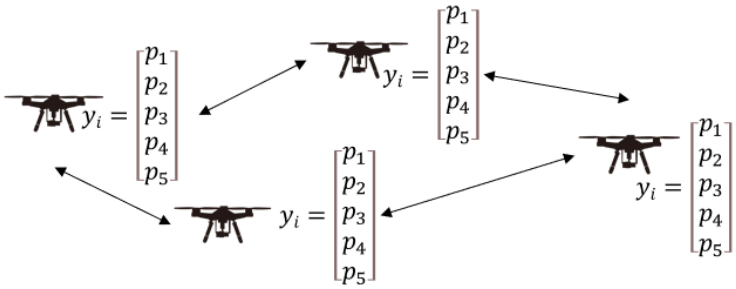


Figure 2.10. System with multiple agents

Here we define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of multiple agents where each vertex $\mathcal{V}$ represents a single UAV and they are connected to other UAVs through corresponding edges in $\mathcal{E}$. That is, for 2 agents i, j $\in \mathcal{V}$ can communicate with each other if the edge $e_{ij} \in \mathcal{E}$. Let this graph have N agents. For our case we define each of these agents will have its own vector denoted

by y_{i} and each of these vectors are an output from the agent's vision system. Essentially, the vectors that each agent will achieve consensus on is the image classification vector of dimension 5×1 that contain the probability of each input image belonging to each of the 5 different classes that was defined in the previous sections.

$$y_{\mathrm{i}}(k) = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}$$

Here, k represents the discrete time, at each step of which each agent will update its own vector to get closer to the average. The problem with average consensus is for every agent in $\mathcal{G}$ to agree upon the same vector y at any finite time k by making an update to its own vector at every time step. Therefore, we will define an average value of all vectors

$$\bar{y} = \frac{1}{N} \sum_{\mathrm{i}=1}^{N} y_{\mathrm{i}}(0)$$

Therefore, the algorithm to be used here to update the state of each variable must ensure that for each agent i,

$$\lim_{k \to \infty} y_{\mathrm{i}}(k) = \bar{y} \tag{2.1}$$

Once we find the average of the vectors of the entire system, we will have one vector for each image that the UAVs see which will be the average of all the vectors in the system and that will be what the system agrees to be the best classification of an image.

2.3 Method

2.3.1 Image Classification using CNN

In the previous sections, we discussed many aspects and the working of Convolutional Neural Networks. In this subsection, we will address the factors considered while construct-

ing our own network, inputs and outputs, architecture of the model, software and libraries used etc., that will enable UAVs to classify images. Most of the work in this section is derived from the work of Kyrkou et al. [44].

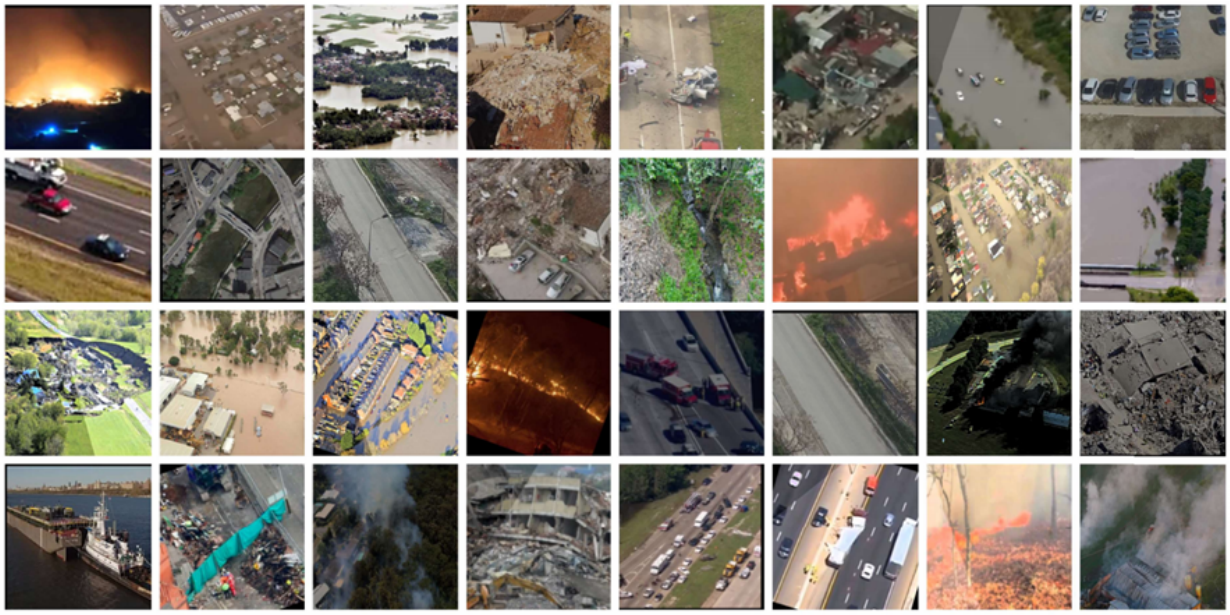


Figure 2.11. Sample images from AIDER dataset. Adopted from [44]

The first step is to gather the required data to train the network. From [44], we use the dataset called Aerial Image Database for Emergency Response (AIDER) which is a dataset consisting of the 5 classes that we have defined earlier i.e. Fire, Flood. Collapsed Building, Normal, and Automobile Accidents. Each of these classes contain 300-500 images each and the normal class contains 1200+ images. The normal class consists of images that does not belong to nay of the other classes and the vision system will always predict a class as normal unless it detects any of the other class. Essentially it is the class that the network falls back on in case it does not detect the other classes. All the classes in the dataset contain images of the corresponding category from an aerial vantage point which means that the pictures in these classes closely resemble what an actual UAV would see using its camera. Figure 2.11 shows a few samples of the different images in different classes in the AIDER dataset. The dataset also contains images with different viewpoints and different illuminations so that the

network can train on a vast variety of images. A reason why the 'normal' class has more images than any other class is to make sure that it is as close to real-life as possible where majority of what a UAV sees will be normal. However, this also makes it challenging to train the network. The reason is because the 'normal' class contains more than half the images in the entire dataset. Therefore, the network could falsely classify every image it sees as 'normal' and still produce a high classification accuracy. In order to avoid this problem, the dataset will be sampled in a balanced way while training which will be explained later.

Next we will talk about the factors considered that sets this network apart from existing networks. The factors considered by Kyrkou et al. [44] while building the network are:

- **First Layer cost** - The first layer of the network is usually a costly layer since it interacts with the entire image input. The larger the image the more computations will need to be done. From the concepts of convolutions mentioned earlier, we understand that the input usually has 3 channels (RGB). The number of channels increases as we move through the network while the dimensions (length and width) decreases. The more the number of channels, the more number of convolutions need to be performed. Also, the kernel size for convolutions play an important role. The bigger the kernel, the lesser number of convolutions are performed on the same input. For example, in a matrix input of dimensions 6×6, a kernel of size 3×3 will move around the input 16 times to cover the entire input assuming a stride (step size) of 1. However, if the kernel has a size of 5×5, it will move around 4 times. That means there will be 16 convolutions performed on the same input if a 3×3 filter is used and only 4 convolutions performed if the kernel used us 5×5. Using a larger kernel however is not always good as important features in the image can often be missed out when larger kernels are used. Hence we strike a balance between speed and accuracy. In this case the number of channels introduced in the first layer is only 16 and the size of the kernel used is 5×5. In other works, the number of channels introduced in the first layer is around 64. The imporved speed is attributed to this smaller channels and larger kernel size.

- **Early downsampling** - The process of downsampling was discussed in the previous section where the dimensions (length and width) are reduced as the information flows through the network. This is done with the help of pooling layers. Max-pooling effectively reduces the dimensions by half while emphasizing the important features in the image. Early downsampling also means that the dimensions reduces by half in each step of the network which decreases the number of computations required. Therefore, the max-pooling operation is performed after every convolutions but the last 2 convolutions.

- **Regular Architecture** - It was mentioned earlier that a typical CNN follows a certain architecture that enables the information to be decreased in dimensions as it flows through the network through pooling layers and increased in the number of channels through convolutional layers. Many networks have the number of channels in the last layers exceed 1000 but this will not be supported on UAVs as the computations become excess for the processors of UAVs to handle. Therefore, in this network, the maximum number of channels before classification will be only 256.

- **Regularization** - One of the many disadvantages of using small datasets is the risk of overfitting. Overfitting happens when the model learns the training data too well. This is to say that since there was not enough images to train, the model learns the output from the input perfectly for the training data and will predict with high accuracy the images in the training data. However, it will fail to successfully predict any other general input. This is not what we want from a network. A network should behave well given any data and should show a high accuracy. To avoid overfitting we use regularisation techniques such as *Batch Normalization* [6] and the dropout strategy [45] with a ratio of 0.5. Dropout strategy is a simple strategy where during training, certain neurons are made inactive with a probability of 0.5 so that the information will find other pathways and not rely on the same path all the time which might lead to over-fitting. This helps in more generalisation of the data. Another regularization technique used is the L2 regularization [46]. L2 regularization is an added loss to the existing loss function where the weights of the

layers are penalized. If the loss function for the network is defined as $L(\mathbf{w})$, where $\mathbf{w}$ represents the weights, then the regularized loss is given by:

$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda||\mathbf{w}||_2^2$$

where λ is the regularization parameter.

- **Depth of the network** - The network depth refers to the number of layers in the network. There are 2 factors to be considered here, 1 - The higher the network depth the better the classification but for small datasets, the number of features that can be learned are limited and hence a higher network depth will lead to a decreased accuracy. 2 - Greater network depth means more operations which means greater computational cost. The factors listed here show that the network we build should not have a high depth and so a network depth of 7 layers was chosen for this architecture.

- **Fully Convolutional Architecture** - As an alternative to a standard CNN where the last few layers are FC layers, a different network architecture is implemented where all the layers are convolutional layers and the last layer is a Global Average Pooling layer before it is passed to the softmax function. This reduces the computational cost effectively as FC layers are more computationally intensive than Convolutional layers. The performance between this architecture and the regular architecture is also compared.

Next, the architecture of the network is discussed. The architecture used by Kyrkou et al. [44] is shown in Figure 2.12. Here, the convolutions are mentioned and also, the alternative strategy is shown where instead of the FC layers before classification, convolutional layers are present instead which performs the classification. The canonical architecture where the last layers are the FC layers which performs the classification is named as *BaseNet* and the alternative where the last layers are convolutional layers is named as *SCFCNet*.

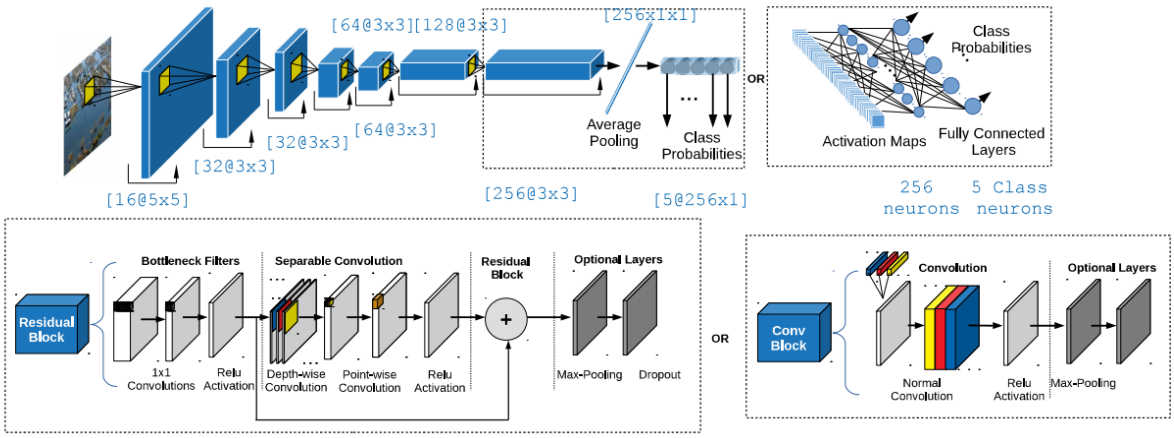


Figure 2.12. Network Architecture. Adopted from [44]

Table 2.1. Network Architecture

Layer	Channels	Dimension	Stride	Padding
Input	3			
Conv1	16	5×5	1	2
Pool1		2×2	2	0
Conv2	32	5×5	1	2
Pool2		2×2	2	0
Conv3	32	3×3	1	1
Pool3		2×2	2	0
Conv4	64	3×3	1	1
Pool4		2×2	2	0
Conv5	64	3×3	1	1
Pool5		2×2	2	0
Conv6	128	3×3	1	1
Pool6		2×2	2	0
FC-1		512		
FC-2		5		
Log Softmax				

In the implementation of this work in this thesis, the network parameters have been changed to further improve accuracy. The network that is modified from BaseNet and used in this work is described in Table 2.1 Also, more images have been added to the dataset in this thesis by using videos available on YouTube and extracting the frames from the videos.

There is another process called Data Augmentation [47] where in order to make-up for the small size of the dataset, the images are transformed to trick the network into thinking that it is receiving a new image. These transformations include very simple tricks as shown in Figures 2.13 and 2.14.

Figure 2.13. Example of Augmentation. Adopted from [48]

Figure 2.14. Another Example of Augmentation. Adopted from [49]

From these images we can observe that by applying simple transformations on images, we can generate a new image to fool the computer into thinking it is a different image. These transformations as seen from the examples above include translation, rotation, flipping, perspective shifts, etc., and help to enlarge the size of the dataset.

While training a neural network, the network goes through each and every image in the training dataset. The training dataset is divided into batches to reduce the training time where each batch goes through a network at one time and the loss of each batch is backpropogated through the network to update the weights. A network going through a single image in this situation is called an iteration. A normal batch size is about 32 to 64 which means the network goes through 32-64 iterations for each batch. Once the network goes through every image in the dataset, it is called one epoch. After every epoch, the neural network starts all over again with the data and learns from it. It takes quite a few epochs for the network to learn the data and give high accuracy results. The data augmentation is performed randomly at each epoch in our network training with the help of the DataLoaders in PyTorch [50]. This means that in each epoch, the training data is randomly augmented so that the network sees 'new' images every epoch. The augmentations applied in this work are:

- Resize - Resizing the image to 224×224.
- Random Rotation - Images are randomly picked to rotate by a random value between -40° to 50°
- Random Affine - Random affine transfomrations are performed on the image
- Random Perspective Shifts - The perspective of the image is shifted
- Random Horizontal Flip - Images are flipped horizontally at random
- Random Gaussian Noise - Random images are picked to which gaussian noise of mean 0 and standard deviation of 0.001 is added

Now, the dataset is split into 2 parts training and, validation with a 0.8, 0.2 ratio respectively. The training dataset is for training the neural network, the validation dataset is for checking the performance of the dataset after every epoch. This validation dataset accuracy and loss is what we look at to judge the performance of the network. We bear in mind that the network only learns from the training dataset. While validating and testing,

we turn the backpropogation off so the network can only predict from the data but never learn. This helps us understand the generalisation performance of the network. The testing dataset is what is used while reporting the results. For the work in this thesis, the testing dataset is not a part of the AIDER dataset but rather images compiled for this thesis to understand the true performance of the network. As mentioned earlier, the AIDER dataset has a large number of images in the 'normal' class. Since splitting the dataset will still not give the correct performance of the network as 'normal' can be predicted for most images to get high accuracy, the testing the dataset that the work in this thesis uses contains 800 images belonging to the class 'Fire' and 104 images belonging to the class normal'. This is a better metric to judge the classification accuracy of the network.

The trick that is used while training to make sure that all the classes are sampled equally to avoid under or over sampling where certain classes, in this case 'normal' is over-represented and hence the model might not learn properly, is called balanced sampling. In balanced sapling, it is ensured that for each batch that is sent to the network for training, the number of images from every class is the same. This would mean repeating a few images but it still ensures that every class is equally represented in each batch and hence the learning is better.

During the training process, there are important parameters that are paramount to proper training: learning rate, optimizer and loss function. For this work, many different learning rates, optimizers and loss functions were tried. The ones that yielded the best results were chosen to compare the accuracy. The finally selected learning is $\alpha = 0.001$. The optimizer selected is the Adam optimizer [51], and the loss function was the cross-entropy loss given by the equation:

$$L(x, y) = -\log \frac{\mathrm{e}^{x[y]}}{\sum_{\mathrm{j}} \mathrm{e}^{x[\mathrm{j}]}}$$

where x is the input to the loss function which are the predicted values from the model and y the actual class which is picked from a range $y = [0, , , C - 1]$ where C is the total number of classes. For example, if the output from the network after the softmax function

is $x = [0.1, 0.1, 0.7, 0.1, 0.0]$ for a 5 classes, and the ground-truth class of the image is the 3^{rd} class, then $y = 2$. These will be the inputs to the cross entropy loss function. Also, since we are using the cross-entropy loss function, we will use the log-softmax function at the output instead of the softmax function as it is more compatible with the Cross-Entropy loss. The log-softmax function is given by

$$\text{LogSoftmax}(x_{\text{i}}) = \log \frac{\text{e}^{x_{\text{i}}}}{\sum_{\text{j}} \text{e}^{x_{\text{j}}}}$$

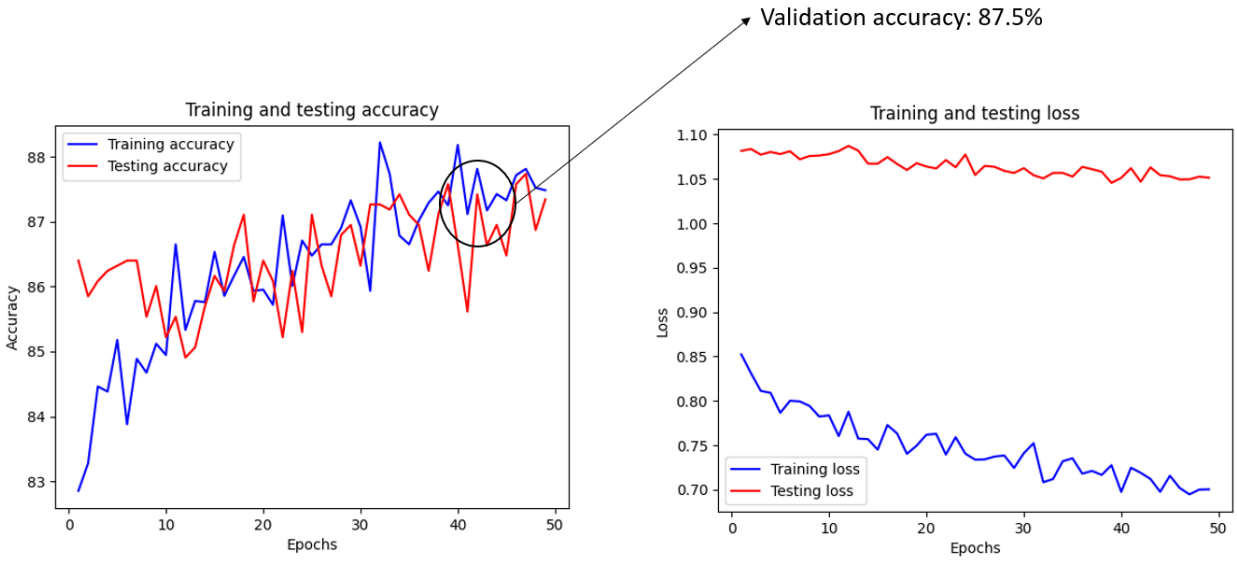


Figure 2.15. Training Statistics for BaseNet variant

The network was trained in parallel on 4 Nvidia GeGorce RTX 2080 GPUs with 128 GB of RAM and an Intel i9 processor. The speed performance discussed in the results section will be measured on this system however the GPU was not used for testing and we relied solely on the CPU.

Before moving to the main results, we will talk about the training for the network built in this work. The training graphs shown in Figure 2.15 are for the modified version of BaseNet constructed for this thesis. In these graphs, what is highlighted as the testing accuracy is the performance of the network on the validation dataset which is a part of the AIDER dataset. The graph shows the trend of increasing accuracy as the number of epochs increases and a decreasing loss as the number of epochs increases. The training is stopped at a little more

than 40 epochs when the loss did not decrease for about 5 epochs. Beyond this point, if the validation loss did not decrease, it would mean that the network has started overfitting which is why training was stopped at that point. At the stopping point as highlighted in Figure 2.15 the validation accuracy was 87.5%.

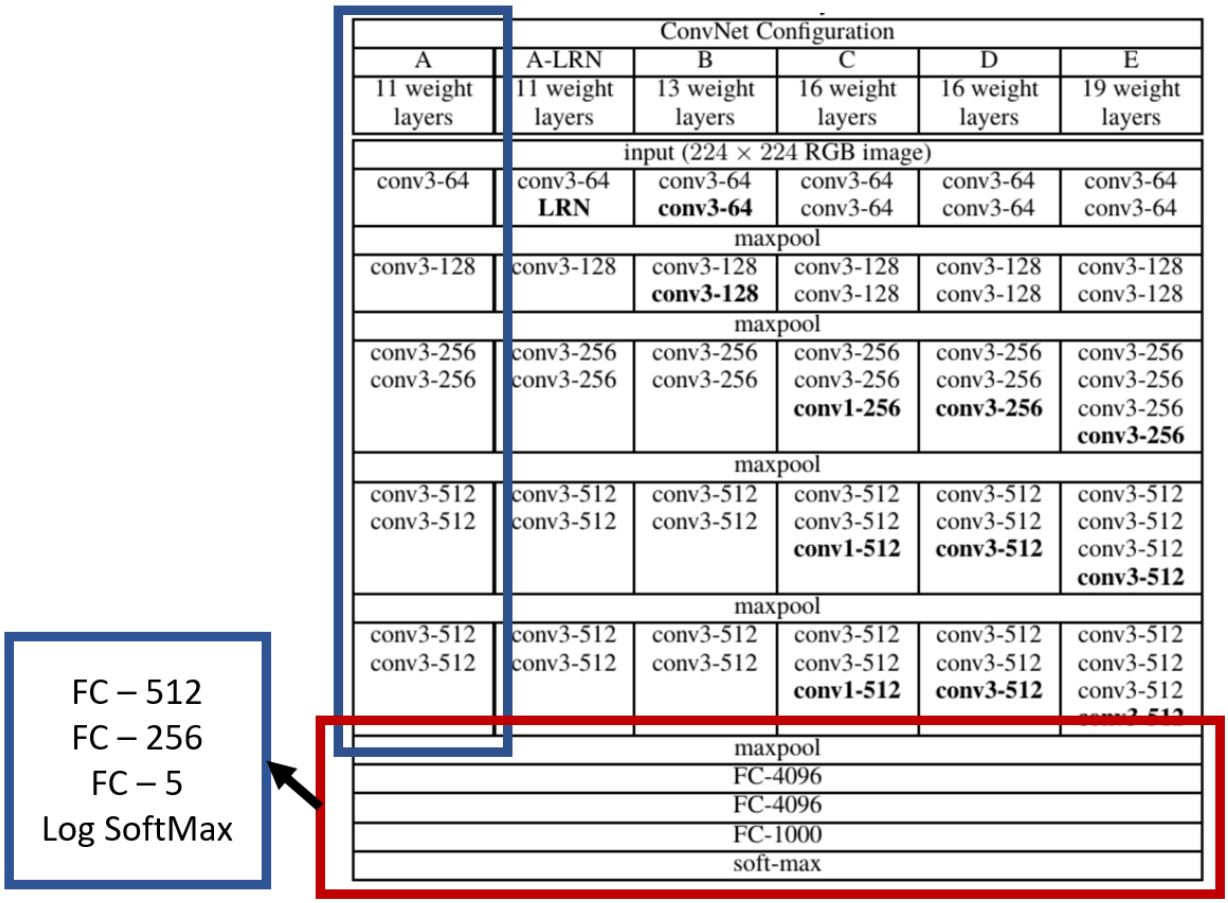
In the Section 2.4 we will look at the results of the performance from these networks, the author's original performance and the improvements in performance through this work in our new testing dataset.

2.3.2 Image Classification using Transfer Learning

In this section, we will address the method of training the networks for our tasks through transfer learning. As we described the process of transfer learning earlier, it involves using pre-trained networks to classify images in a specific dataset. It is important to keep in mind that the networks that will be used here are trained on the ImageNet dataset [52]. The ImageNet dataset contains over a million images in 1000 classes. It is a huge dataset and requires a lot of time to train. Training with the ImageNet dataset on the simple BaseNet architecture mentioned earlier takes about 2 hours for each epoch. Training for close to 40 epochs is a daunting task with a lot of computation 80 hours. Obviously, the time taken is considerably larger as different parameters are varied through trial and error to get the best performance in the network. Therefore, only BaseNet was trained from scratch with the ImageNet dataset and then made to transfer learn. Transfer learning works because the layers have learned to extract the important features from the image. All that needs to be changed is the classification layers which determine the final output. Hence, 2 variants of transfer learning was performed:

- The weights of the first few layers were locked and only the weights in the last layers were allowed to be changed while training with the specific AIDER dataset. Of course, all weights were allowed to change when being trained on the ImageNet dataset.

- While training on the AIDER dataset, all the weights in all layers were allowed to be trained.

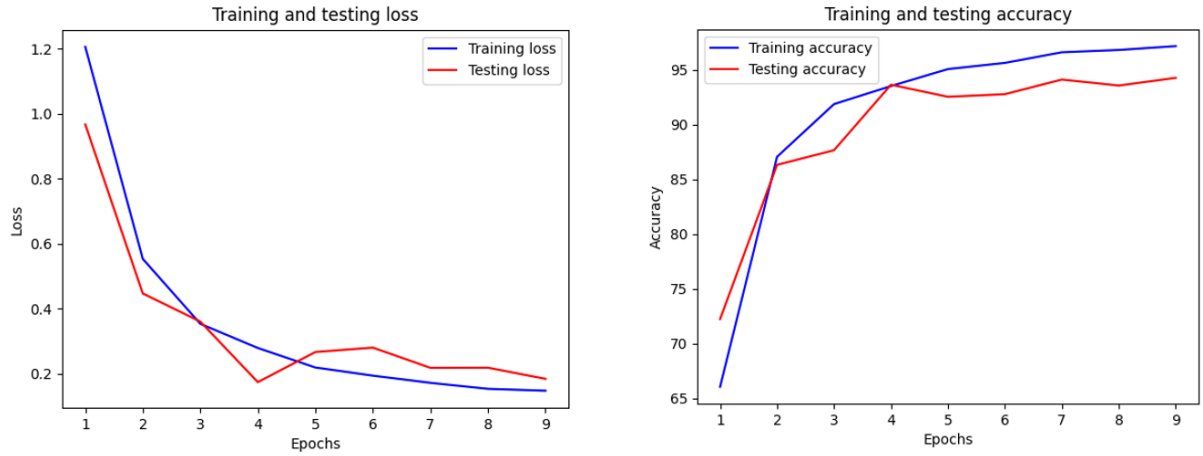


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

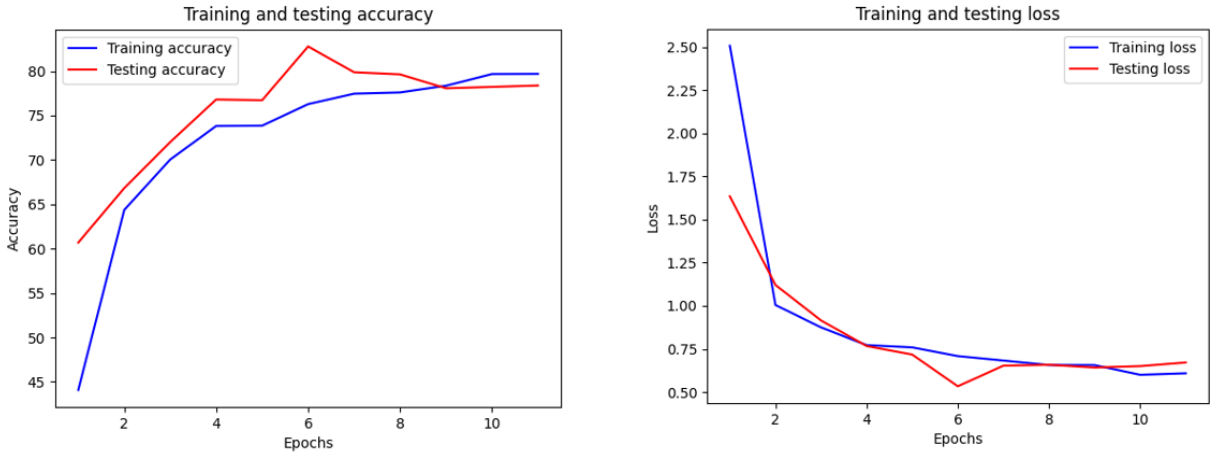
Figure 2.16. VGG11 used for Transfer Learning

The other network that is presented here is a network called VGG11 [11] which is a 11-layered version of the popular VGG16 network. The pre-trained model is readily available from PyTorch's pre-trained model library. In Figure 2.16 the network is shown, however, the network is modified by taking out the last few layers highlighted in red and instead replacing it with the new set of layers that starts with a FC layer of 512 and ends with a FC layers of size 5 as we have 5 classes. Therefore, the initial layers that were transferred already have weights which was learned from the ImageNet dataset but the new layers do not have any trained weights yet. Then, this network was trained with the AIDER dataset both, keeping

the weights of initial layers constant and allowing all weights to be learned again.



(a) VGG11 Transfer Learning training statistics



(b) BaseNet Transfer Learning training statistics

Figure 2.17. Transfer Learning Networks Training Statistics

The training and validation statistics for this network is shown in Figure 2.17a where it is evident that there is a high validation accuracy in about 7-8 epochs of about 94%. However, when we look at the performance for transfer learning in BaseNet, we see lower validation accuracy of about 78%. The drop in performance is expected as a shallow network like BaseNet is not built to handle training on the ImageNet dataset as it does not have enough learnable parameters to formulate the relationship between the input and the

output. However, both networks in Figures 2.17a and 2.17b the trend is for the accuracy to increase over the epochs and the loss to decrease over the epochs which shows that the networks are learning, The extent to which they would have learnt will become obvious in the main results section for this chapter.

Many training parameters were varied such as the learning rate, optimizer and the network architectures but it was found that the same settings that was used to train BaseNet as shown in the previous subsection provided the best results.

2.3.3 Multi-agent Image Classification

In all the subsections so far the method to obtain a 5×1 output vector from the vision system was discussed. This 5×1 vector is the output of a Log Softmax function from the last layer of the CNN. It is evident that the predicted class of the image by the neural network is the index of the highest value in this vector:

$$\text{Predicted class} = \arg\max_{p} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}$$

We have defined earlier that this output vector for each agent in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is $y_{\mathrm{i}}(k)$. From [18], we know that each agent must update their vector at each time step to move towards the average defined by equation 2.1. The update rule that each agent must follow to reach this average is given by:

$$y_{\mathrm{i}}(k+1) = y_{\mathrm{i}}(k) + \sum_{\mathrm{j} \in \mathcal{N}_{\mathrm{i}}} W_{\mathrm{ij}}(y_{\mathrm{j}}(k) - y_{\mathrm{i}}(k)) \tag{2.2}$$

where $\mathcal{N}_{\mathrm{i}}$ is a set of all agents that are neighbours of agent i. A neighbour of an agent means that the 2 agents are connected to each other through either a one way or two way

communication. W_{ij} is the i^{th} element of the j^{th} row in a weight matrix W. To achieve average consensus, this weight matrix must satisfy the conditions:

$$W = W^T \qquad W\mathbf{1} = \mathbf{1} \qquad W \in \mathcal{S}, \tag{2.3}$$

where $\mathbf{1}$ denotes a vector of all ones, and $\mathcal{S}$ denotes a sparse matrix that is compatible with the graph,

$$\mathcal{S} = \{W \in \mathbf{R}^{n \times n} | W_{ij} = 0 \text{ if } i \neq j \text{ and } (i, j) \notin \mathcal{E}\}$$

To achieve global average consensus, the *Metropolis-Hastings weights* is utilized that gives us the values of the W matrix so as to achieve average consensus. These are the values that are used for the work in this thesis.

There is also another method for achieving consensus that was tested in this thesis called Gossiping Consensus [53]. In this method, a node and its nearest neighbour communicate with each other at each time step to slowly move towards consensus. Here, the node sends $0.5 \times y_i$ to its neighbour and receives $0.5 \times y_j$ from its neighbour at each time step. The update rule followed here is:

$$y_i(k+1) = \frac{y_i(k) + y_j(k)}{2} \tag{2.4}$$

Although this method achieves average consensus, it is too slow as each agent communicates with only one other neighbour and our tests for a simple system with 4 agents show that gossiping consensus takes 17 time steps to achieve consensus. However, using the update rule in equation 2.2, we use 2 matrices to show the results for a system of 4 agents. The values that we test the algorithm with are:

$$y_1(0) = \begin{bmatrix} 0.0805 \\ 0.0805 \\ 0.6782 \\ 0.0805 \\ 0.0805 \end{bmatrix} \quad y_2(0) = \begin{bmatrix} 0.0805 \\ 0.0805 \\ 0.6782 \\ 0.0805 \\ 0.0805 \end{bmatrix} \quad y_3(0) = \begin{bmatrix} 0.0200 \\ 0.2000 \\ 0.2000 \\ 0.2000 \\ 0.2000 \end{bmatrix} \quad y_4(0) = \begin{bmatrix} 0.0805 \\ 0.0805 \\ 0.6782 \\ 0.0805 \\ 0.0805 \end{bmatrix}$$

It is evident that image belongs to class 3 for all but one agent. We will see whether the system can achieve this average. We define 2 weight matrices based on the topology in Figure 2.10:

$$W_1 = \begin{bmatrix} 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \end{bmatrix} \quad W_2 = \begin{bmatrix} 0.3 & 0.3 & 0 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0 \\ 0 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0 & 0.3 & 0.3 \end{bmatrix}$$

It is clear that both matrices follow the constraints in equation 2.3. W_2 is the Metropolis weights matrix. Both matrices used with equation 2.2 show convergence. When W_1 is used, the system converges in 5 time steps. When W_2 is used the system converges in 7 time steps. In both cases, the highest probability value is at index 3 with a value of 0.43.

2.4 Main Results

CNN Model	Type[1]	Average Accuracy (%)	Processing Time (ms) [2]	Frames-Rate[2]	Speedup[3]	Memory (MB)
ERNet	C	90.1	18.7	53	18.5	0.3
SCFCNet	C	87.7	13.1	76	26.4	0.2
SCNet	C	85.4	14.1	70	24.5	6.5
baseNet	C	88	21.2	47	16.3	7
VGG16	T	91.9	346	2	1	59.3
ResNet50	T	90.2	257	3	1.3	96.4
MobileNet	T	88.5	48.2	20	7.1	13.9

Figure 2.18. Accuracy comparison of the networks built by Kyrkou et al. [44]

In this section, we will start with the results of the works of Kyrkou et al. [44] which are shown in Figure 2.18 and then we will compare the network built for this thesis. We will start by comparing the CNNs classification accuracy and performance, then move onto the Transfer Learning networks and finally we will look into the Multi-agent image classification system.

From Figure 2.18 we will only look at the rows for baseNet which shows a classification accuracy of 88% and SCFCNet which shows a classification accuracy of 87.7%. It is important to note that the accuracy shown in Figure 2.18 are the network tested on the AIDER dataset. This thesis' results are tested on different images to check the generalization performance and to see if the network would not classify everything as 'normal'. While training the modified version of baseNet, we obtained the confusion matrix shown in Table 2.2. The confusion matrix is used to identify how many images in each category does the neural network classify into each category. Typically, we will see that the diagonal of confusion matrices should have the highest values which would show that the network has actually learned and not classifying images randomly. It also helps to determine if certain classes are over-represented or under-represented.

Table 2.2. Confusion matrix for modified baseNet

	Collapsed Building	Fire	Flood	Normal	Accident
Collapsed Building	73	4	0	15	5
Fire	1	90	0	9	1
Flood	6	0	60	20	14
Normal	9	40	0	827	2
Accident	9	5	2	15	64

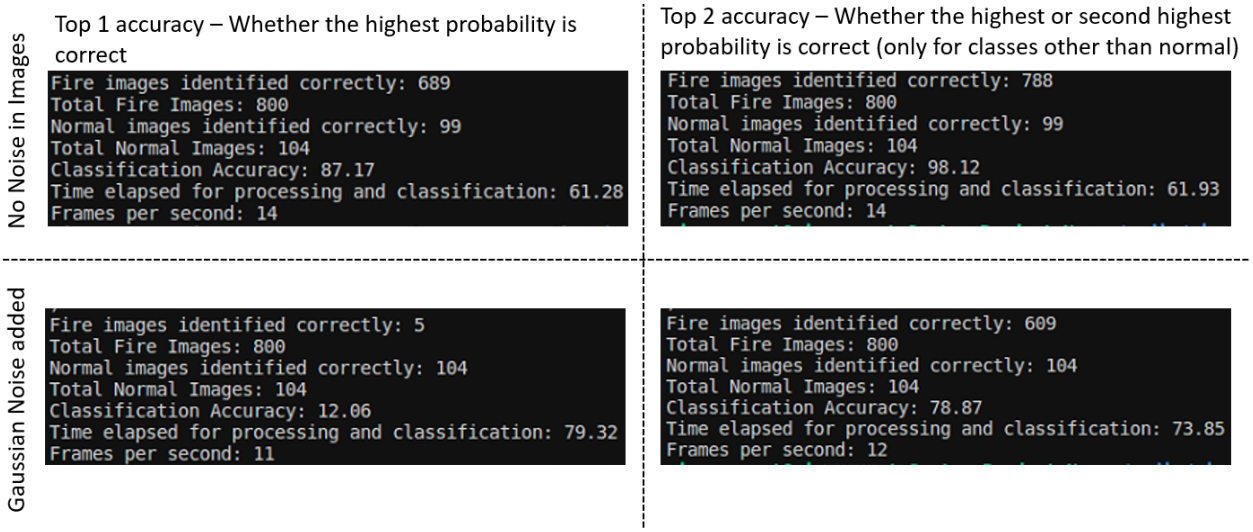
Our modified BaseNet shows a validation accuracy of **89.1%** which is higher than the **88%** accuracy as shown in the original results. Further, on testing it with the our testing dataset, we find that it achieves an accuracy of **91.81%** correctly classifying 726/800 images of 'fire' correctly and 104/104 images of 'normal'. We record an FPS (frames per second) of 51 but this value cannot be compared to the original results because the computing powers

are different. However, 51 fps is good enough to allow the network to be run on UAVs to process data in real-time. Looking at the performance of SCFCNet however, the validation accuracy showed 87.5% which is a marginal decrease from the reported 87.7%. However, running this on our testing dataset, the accuracy was very poor with an average accuracy of around 60%. This shows that the learning capability of SCFCNet is limited when data is more generalised. So here we conclude that the modified version of BaseNet is more suited towards classifying images.

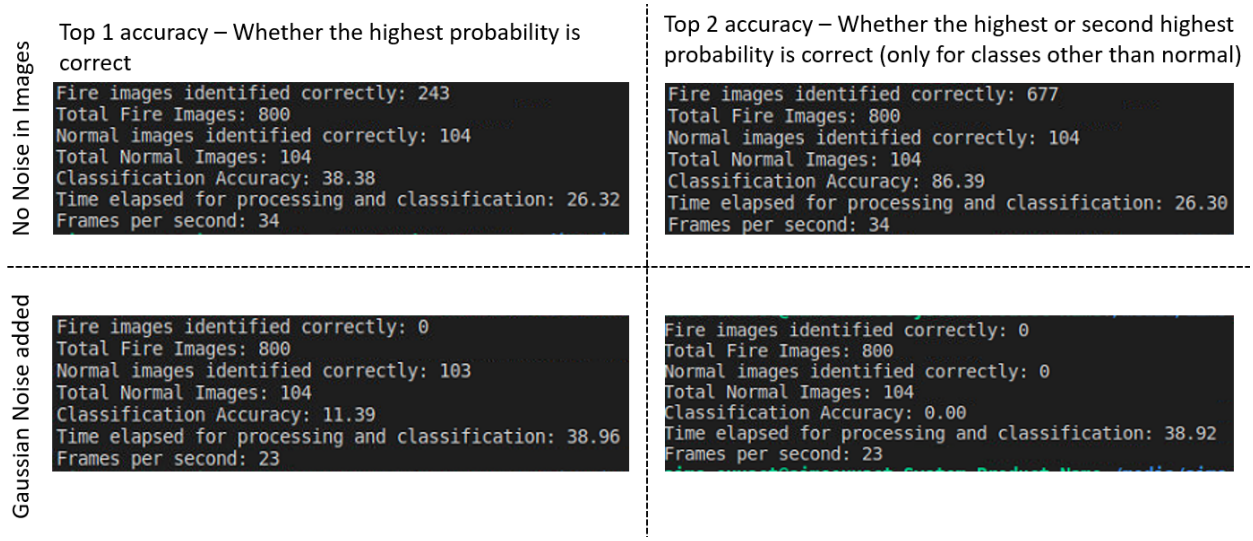
Equipped with this information we move onto the next stage of our results which include the transfer learning networks. As mentioned before, we use the VGG11 and the modified baseNet to be transfer learned. We will show the result for 2 scenarios the relevance of which will be evident later. One set of results are normal images passed through the network and the classification accuracy is calculated. The other set of results are for when Gaussian noise is added to the same images before passing it into the network. We obviously expect the images with Gaussian noise to have poor accuracy as the network was not trained to handle such noise in the image.

In Figures 2.19a and 2.19b, we show 2 types of accuracy, Top 1 accuracy where the class with the highest probability as predicted by the network is the correct class of the image, and Top 2 accuracy where the classes with the highest and second highest probabilities as predicted by the network is the correct class of the image. The Top 2 accuracy is done only if the image is not 'normal' again because of over-representation of the 'normal' class. We see that the VGG11 has an amazing performance with a Top 2 accuracy of **98.12%** having transfer learnt but the BaseNet has dipped in performance as compared to the previous results. This shows that baseNet is incapable of transfer learning due to the sallow network size. As predicted we also see that the performance dipped significantly when the noise was added to the images before it was inputted into the neural network.

Now we will look at the final set of results for the multi-agent image classification system. Figure 2.20 shows what each of the 4 agents will see. It is the same image with perspective



(a) VGG11 Transfer Learning training results



(b) BaseNet Transfer Learning training results

Figure 2.19. Transfer Learnt Networks' Results

shift transformations to simulate the effect of UAVs viewing the same image from different angles. The first column are example images of what agent 1 sees, second column for what agent 2 sees and so on. Now we will look at the performance of the system as a whole. The agents will achieve consensus amongst themselves for the probability values of their vision system. The classification accuracy here is shown to be **91.48%**. This is despite the massive perspective shifts that is seen in Figure 2.20. It is evident that the actual accuracy if not

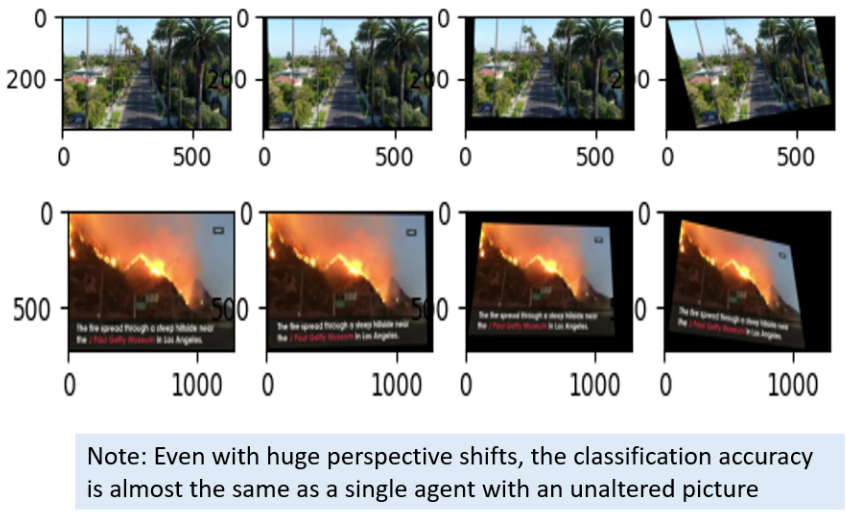


Figure 2.20. Image examples for the 4-agent system

for such perspective shits would definitely be higher. Unfortunately we did not perform this experimentally with multiple UAVs observing the same environment from different angles.

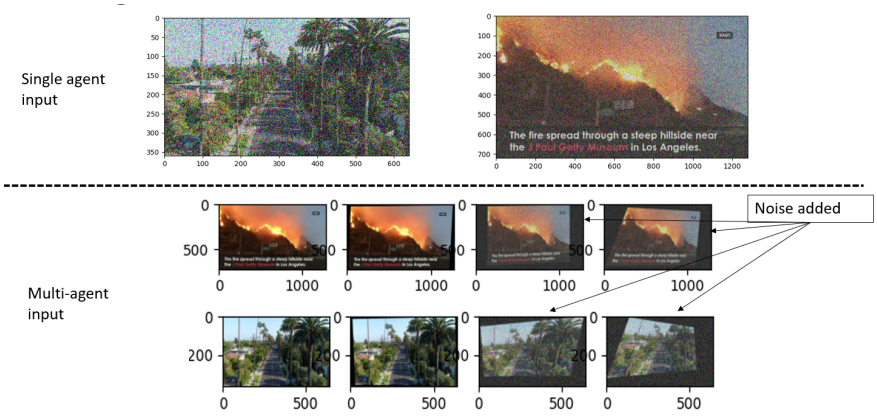


Figure 2.21. Image examples for the 4-agent system with noise added to 2 agents

Now we look at the same condition, but we will add Gaussian noise to 2 of the 4 UAVs' images. This means that half of the UAVs in a system are affected by Gaussian noise which we have seem before to cripple the accuracy of the system. The Gaussian noise added has a mean = 0 and standard deviation = 0.5. The example of the input is shown in Figure 2.21.

The results from this are very surprising and shown in table 2.3.

Table 2.3. Results for multi-agent image classification

Gaussian Noise (single agent)	Gaussian Noise on 2 of the 4 agents
Fire Images identified correctly: 0/800 Normal Images identified correctly: 104/104 Classification Accuracy: **11.5%**	Fire Images identified correctly: 723/800 Normal Images identified correctly: 104/104 Classification Accuracy: **91.48%**

The accuracy for the multi-agent system significantly increased as compared to the performance of a single agent. This means that a multi-agent image classifier is more resilient to variations in the input image to still produce the correct output. We also look at a case for the VGG 11 network where noise with a non-zero mean = 0.5 and a standard deviation = 0.5 is added to the images and the performance of single agent vs multiple agents is compared. Figure 2.22 shows this comparison and it is again evident that the classification accuracy greatly increased for multi-agent systems from 17.92% to 70.02%.

Figure 2.22. Single agent vs Multi-agent comparison for VGG11 with non-zero noise

In conclusion, we can say that the regular performance of single agent vs multi-agent is similar but multi-agent can prove to be better provided more experimental data. Regular events such as occlusion of ROI from one camera can be counteracted as the consensus of the graph will ensure that the common decision is the correct decision. The multi-agent system is tolerant to occasional irregularities such as noisy data, perspective shifts, poor lighting/sensors, out of focus images, too close or far from the object, etc,. among other

faults. Therefore, using a distributed vision system proves to be highly fault-tolerant and brings up the accuracy of the entire system.

3. RESILIENT AVERAGING CONSENSUS

3.1 Introduction

The results in the previous section shows a very promising approach to image classification. We are left with a 5×1 vector that is the output of the vision system of each UAV. Next, we applied this vector to a multi-agent system to achieve average consensus and saw how the accuracy can be increase in certain situations if the multi-agent approach is taken. Now, we will look at how to ensure that the system always achieves consensus even in the face of malicious attacks. A system that can achieve consensus even when the system is under-attack by entities that want to prevent consensus is called a resilient system. In this section, we will look at how our system is made to achieve consensus and the algorithms it uses to ensure that all benign agents agree upon the same value.

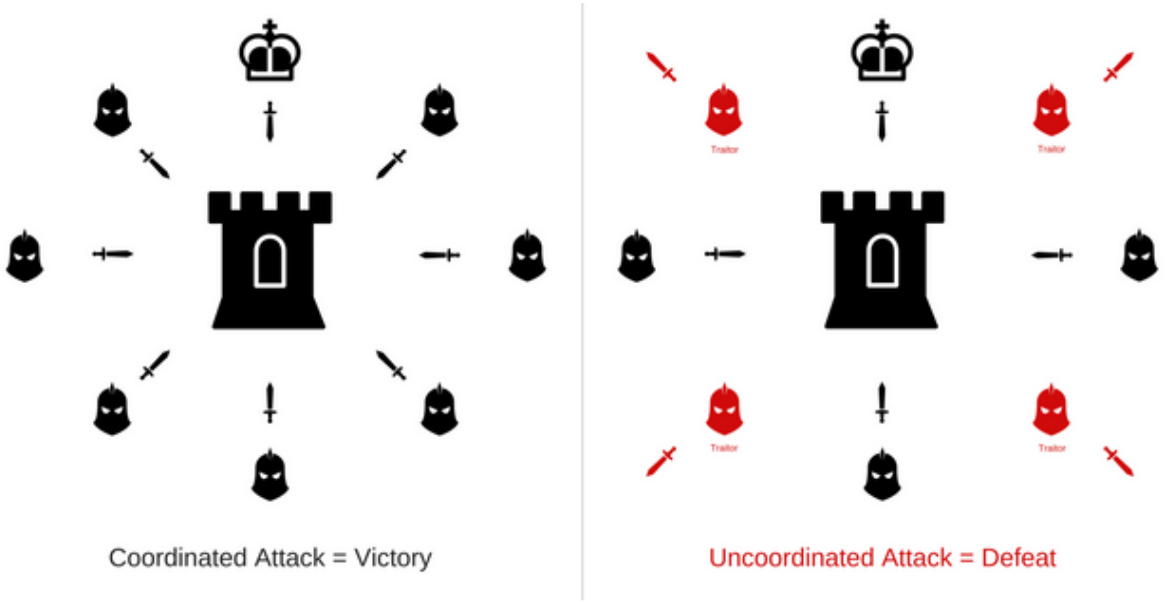


Figure 3.1. Byzantine Generals Problem. Adopted from [54]

We look at a special type of problem called Byzantine Generals problem. This is based on a though experiment that Byzantine generals outside a city are waiting to attack the city. However, all the generals must attack together to be victorious. Or all generals must retreat to live to fight another day. Either way, all the generals have to agree on the same

thing and execute it. Then, traitors are introduced into the system to confuse the generals. Traitors might tell one general that they are planning to attack and tell another general that they plan to retreat. This would cause an uncoordinated attack leading to the defeat of the generals. This is exactly the type of attack that can happen in our systems that leads the system away from consensus. Therefore, in our system, the UAVs need to be resilient to such attacks where the attacker will make some of the UAVs in the system produce erogenous values that will try and confuse the system. These values could be random or carefully crafted to try and fool the system. This section will talk about how a system of UAVs with each trying to achieve the average of a 5×1 vector can achieve resilient consensus. For the sake of convenience, we will call this 5×1 vector that state or state vector of the UAV, although in literature the state of UAVs refers to another vector.

3.2 Problem Formulation

For the problem, we have a multi-agent graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. In this we define a set of benign nodes $\mathcal{B}$ and a set of adversarial nodes $\mathcal{A} = \mathcal{V} \backslash \mathcal{R}$. Here, the benign nodes are the agents that are not attacked and will follow the required protocols and update rules whereas the adversarial agents are the ones that are attacked and will not follow protocols and update rules to try and move the system away from consensus.

We define 2 models of attacks [27] - The f-local adversarial model and the f-total adversarial model. In the f-total adversarial model, it is assumed that the upper bound of the number of adversarial agents in the system is f, which means at any given time, there will be at-most f adversarial agents in the system. In the f-local model, the number of adversarial agents in the vicinity of each agent is upper bounded by f. This means that each node will have at-most f adversarial agents as its neighbours. We assume that the value of f is known. We know that the neighbours of each agent i is denoted by $\mathcal{N}_\mathrm{i}$. Therefore, an f-local model can be described as

$$|\mathcal{N}_\mathrm{i} \cap \mathcal{A}| \leq f, \qquad \forall \mathrm{i} \in \mathcal{B}$$

Now, 2 different approaches to achieving resilient consensus will be shown and these approaches will have a different problem statement. The first one is a type of resilient consensus where we reach the exact average of the initial states of the agents. For this method, the problem statement is put-forth as for a multi-agent system $\mathcal{G}$ under an f-local adversarial attack model, resilient average consensus is achieved if for every initial condition, $x_{\mathrm{i}}[0], \quad \mathrm{i} = 1, 2, ..., N$ it holds that

$$\lim_{k\to\infty} x_{\mathrm{i}}[k] = x_a, \qquad \forall \mathrm{i} \in \mathcal{B}$$

where $x_a = \sum_{\mathrm{i}\in\mathcal{B}} x_{\mathrm{i}}[0]/|\mathcal{B}|$.

The second method does not involve finding the exact average of the state vectors rather it tries to get as close to the average as possible. However, since in many cases we cannot identify the identity of the adversarial nodes, we will have to rely on consensus with a margin of error. So in this case, the problem will be:

$$\limsup_{x\to\infty} \max_{\mathrm{i,j}\in\mathcal{B}} |x_{\mathrm{i}}(k) - x_{\mathrm{j}}(k)| < \epsilon$$

where ϵ is a very small number. In the next section, we will see the various methods that can be used to achieve resilient consensus on our vision system.

3.3 Method

The first method shown here is based on the first problem formulation where we will proceed to identify the adversarial nodes in the system. Here, we use an algorithm called Secure Acceptance and Broadcasting Algorithm (SABA) [27]. This algorithm is shown in Figure 3.2. Essentially this algorithm shows a simple way to accept the values of other agents. It employs a voting system to identify the correct values from the wrong ones. And provided, the graph has enough connections, each agent will have the states of all other agents after a finite number of time steps.

Initialization
The regular node i creates a persistent empty memory
$m^i[0] = [\,]_{1\times\bar{N}}$, where $\bar{N} \geq N$, and sets
$m^i_i[0] = x_i[0]$.
if $k = 1$ **then**
  for $j \in \mathcal{N}_i$
    The regular node i broadcasts $m^i[0]$ to the node j, receives $m^j[0]$ and updates its memory $m^i_j[1] = m^j_j[0]$, $j \in \mathcal{N}_i$.
  end
end
if $k > 1$ **then**
  The regular node i broadcasts its memory vector $m^i[k-1]$ to all its neighbors.
  for $n \in \{1, 2, \ldots, \bar{N}\}$
    If the regular node i received an identical value $m^j_n[k-1]$, $j \in \mathcal{N}_i$ from $f+1$ incoming neighbors, then it accepts that value and saves it in the memory $m^i_n[k]$.
  end
end
Result: $m^i[k] = [m^i_1[k], \ldots, m^i_{\bar{N}}[k]]$, $\bar{N} \geq N$

Figure 3.2. Secure Broadcasting and Acceptance Algorithm. Adopted from [27]

Essentially, the assumptions in this algorithm are that at $k = 0$, all agents send their true initial values and only from $k > 1$ are the adversarial agents allowed to send faulty values. So beginning at $k = 0$ all the benign agents sets its own memory vector to its initial value. At $k = 1$, each regular agent broadcasts its memory vector to its neighbours while receiving values from the neighbours to update its own vector. Again, it is assumed that there are no communication delays in the network and each agent receives and send its states simultaneously. Since from $k > 1$ the adversarial agents may send faulty values to its neighbours, the voting system is employed to ensure that the benign agents only accept the correct values. The memory vectors of each agent is only updated if it receives the identical value of the state of another agent from $f + 1$ agents. Since there are at-most f adversarial agents in a benign agents neighbourhood, identical values from $f + 1$ neighbours means that the value is trustworthy. Of course, the limitation here is that the graph $\mathcal{G}$ must be strongly $(2f + 1)$ robust for this algorithm to work. This would mean that each agent would need to be strongly connected to at-least $2f + 1$ other agents.

The end result of the SABA algorithm gives us a vector $m^{\mathrm{i}}[k]$ for each agent i that contains the state of all other agents which have been agreed upon by voting. Next, we will use this memory vector to find the average. At each time step, the node i updates its time step as:

$$\phi_{\mathrm{i}}[k] = \frac{\sum m_n^{\mathrm{i}}[k]}{\lambda[k]}, \;\; n \in \mathbb{M}^{\mathrm{i}}[k] \tag{3.1}$$

Here, $\mathbb{M}^{\mathrm{i}}[k[$ is the set of indices of $m^{\mathrm{i}}[k]$ that are non-empty and $\lambda[k] = |\mathbb{M}^{\mathrm{i}}[k]$ is the cardinality of this matrix. Further, to update the states of each agent, a low-pass filter is used which exponentially smooths the states of each agent which is given by the set of equation:

$$\begin{aligned} x_{\mathrm{i}}[0] &= \phi_{\mathrm{i}}[0], \\ x_{\mathrm{i}}[k] &= \epsilon_{\mathrm{i}} x_{\mathrm{i}}[k-1] + (1-\epsilon_{\mathrm{i}})\phi_{\mathrm{i}}[k], \;\; \forall k > 0, \end{aligned} \tag{3.2}$$

where $0 \leq \epsilon_{\mathrm{i}} < 1$ is the filter gain. In this case, since we don't really require exponential smoothing and we would like the system to reach consensus on the class probabilities, we set the filter gain $\epsilon_{\mathrm{i}} = 0$. This method helps in identification of the adversarial nodes and then finds the average of the states by averaging the states that is in the memory vector which ignores the random faulty values sent by adversarial agents. This algorithm in [27] can be applied to our problem although slight modifications have to be made. The algorithm was designed for each agent to have one value as its state. In our case, we have 5 values in the state vector. After different approaches using trial and error, the best method was found to be using SABA to identify the states of the agents. The dimensions of the memory vector will be different because now each memory vector will be turned into a matrix with the rows representing the different probability values. This means that that $m_n^{\mathrm{i}}[k]$ represents a 5×1 vector. The rest of the algorithm remains the same. Further, once the memory vectors have been updated with all of the states, we use equation 3.2 to find the average of each agent at each time step with $\epsilon_{\mathrm{i}} = 0$. Here, this step will need to be performed on vectors with element-wise addition and division. This will give us 5 values for $\phi_{\mathrm{i}}[k[$ in each time step and

each of these 5 values represents the respective average value of the probability values.

Implementing this algorithm for a system with 5 agents, we set that the adversarial agent is 'Agent 2'. Figure 3.3 shows the system executing SABA was able to determine the adversarial node as 'Agent 2' in the second time step. It identified that the values coming from 'Agent 2' through the edges highlighted in red are faulty values.

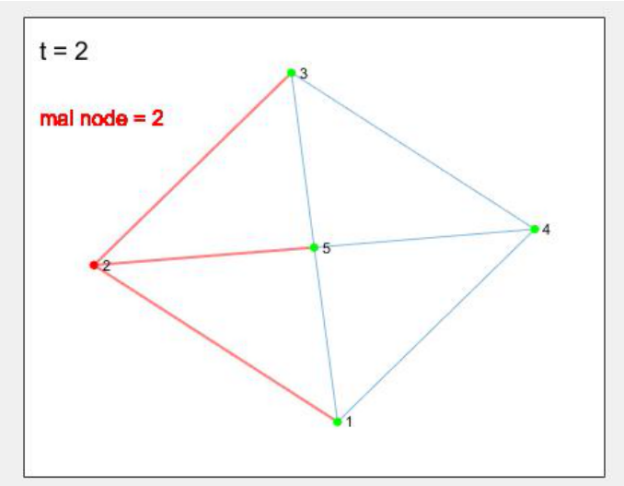


Figure 3.3. Finding the adversarial agent using SABA

Figure 3.4 shows the initial states of each agent. y_i denotes the initial state of agent i. The initial state vector is a 5×1 vector the reason for which is discussed in the previous sections. The graphs in Figure 3.4 show the convergence of average values for each node for each probability value. p_1 is the first probability value and is the first row of every state vector. We can see that at $k = 3$, all the values converge to a mean of 0.1044. Similarly, all the other probability values shows convergence at their respective mean values. In every graph, we can see the value of node 2, or agent 2 not converging as this node continues to produce a faulty value and will not converge. From the example, we can also see that p_3 has the highest probability which means that the predicted class will be the 'Class 3'.

This algorithm helps us identify the adversarial agent and then helps the system reach average consensus of the initial states. It may not always be possible to identify the attacker

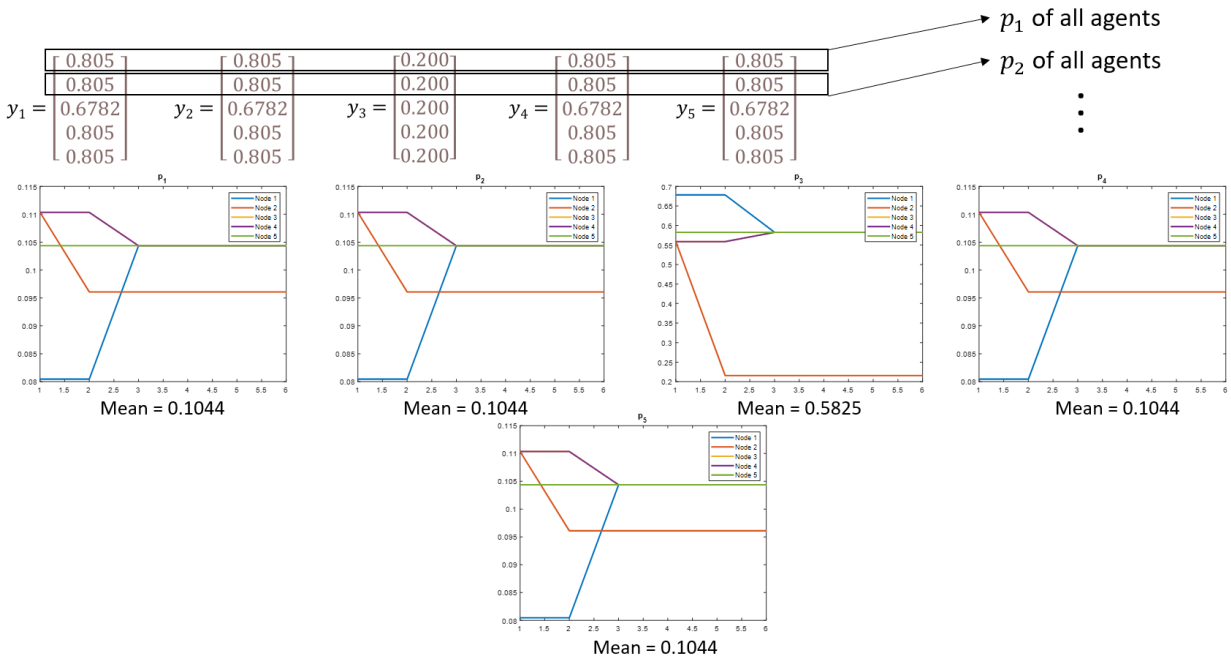


Figure 3.4. Convergence of states using SABA

in many scenarios so we look at methods where the adversarial agent is not identified yet the system can achieve consensus. In the next subsections, we will elaborate 2 methods [31], [32] that helps the system achieve consensus without identification of the adversarial node. It is evident that such a system can never reach the true average of the initial states and instead can only focus on coming as close as possible to the average of the states.

The next subsections talk about the 2 methods that was used to achieve consensus without identification of the adversarial node.

3.3.1 Weight Learning Method

Similar to the weight matrix in 2.2, this method [32] uses a weight matrix denoted by A which is also called a weighted adjacency matrix to update the states of the agents. This matrix A represents the edges of the graph $\mathcal{G}$. $A = \{a_{ij} \geq 0\}$ but here we assume that there are no self-loops which means $a_{ii} = 0$. Since it is a weight learning method, the value of A changes at every time-step so we denote $a_{ij}(k)$ as the value of a_{ij} at time step k.

Previously, we defined a set $\mathcal{B}$ as the set of benign agents, and a set of adversarial agents $\mathcal{A}$. For this method, these definitions are taken a bit further, we define 3 sets of nodes:

$$\mathcal{V} = V^n \cup v^p \cup V^{\mathrm{i}}$$

where V^n are the regular/normal nodes, V^p are the persistent faulty nodes, V^{i} are the intermittent faulty nodes. Each of these sets of nodes follows the same update rule i.e.

$$x_{\mathrm{i}}(k+1) = x_{\mathrm{i}}(k) + u_{\mathrm{i}}(k)$$

.

The update rule by the different sets of nodes listed are given by:

- **Normal Nodes** V^n

$$u_{\mathrm{i}}(k) = \sum_{\mathrm{j} \in \mathcal{N}_{\mathrm{i}}} a_{\mathrm{ij}}(x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k)) + \omega_{\mathrm{i}}(k), \;\; \mathrm{i} \in V^n$$

  Here, $a_{\mathrm{ij}}(k)$ is the adjacency matrix weight which satisfies $\sum_{\mathrm{j} \in \mathcal{N}_{\mathrm{i}}} a_{\mathrm{ij}}(k) < 1$ and $\omega_{\mathrm{i}}(k)$ is a bounded noise bounded by $(|\omega_{\mathrm{i}}(k)| < \omega)$ introduced by transmission channel and environment.

- **Persistent Faulty Nodes** V^p

$$u_{\mathrm{i}}(k) = \text{Random}, \;\; \mathrm{i} \in V^p$$

- **Intermittent Faulty Nodes** V^{i} These kind of nodes show behaviour of both normal and faulty nodes,

$$u_{\mathrm{i}}(k) = \begin{cases} \sum_{\mathrm{j} \in \mathcal{N}_{\mathrm{i}}} a_{\mathrm{ij}}(x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k)) + \omega_{\mathrm{i}}(k), & \text{with probability } p, \\ \text{Random}, \quad \text{else} \end{cases} \quad , \mathrm{i} \in V^{\mathrm{i}}$$

Now that the nodes have been defined and the update rules for the nodes are also defined, the algorithm that makes this method work is defined. This method uses a learning method to change the weights of the adjacency matrix over time so that the links between regular and faulty nodes have very small weights (ideally 0) and the link between regular and regular nodes have high weights (ideally 1) . This is achieved through a reward function defined by:

$$r_{\mathrm{ij}}(k) = f(|x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k) + \omega_{\mathrm{ij}}(k)|, k), \quad \mathrm{j} \in \mathcal{N}_{\mathrm{i}}, \mathrm{i} \in V^n$$

where ω_{ij} is the transmission noise from node j to i. Usually, the function f is selected so that it is inversely proportional. The example given in [32] is the function that was selected for this work as well which is:

$$f(|x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k) + \omega_{\mathrm{ij}}(k)|, k) = \mathrm{e}^{-|x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k) + \omega_{\mathrm{ij}}(k)|\theta(k)}$$

where an arbitrary design parameter $\theta(k) > 1$. This function helps to restrict the value between 0 and 1 which means that if the states are different the reward moves closer to 0 while if the states are similar the reward moves closer to 1. Another function that is defined here is the credibility function which helps store the historical value of the reward function. This function essentially integrates the value of the reward function between 2 nodes over time thereby providing the reliability of a connection over time.

$$Q_{\mathrm{ij}}(k) = Q_{\mathrm{ij}}(k-1) r_{\mathrm{ij}}(k), \;\; Q_{\mathrm{ij}}(0) = 1, \;\; \mathrm{j} \in \mathcal{N}_{\mathrm{i}}, \mathrm{i} \in V^n$$

Finally, the weighted matrix is then updated using the formula

$$a_{\mathrm{ij}}(k) = \frac{Q_{\mathrm{ij}}(k)}{\sum_{\mathrm{j} \in \mathcal{N}_{\mathrm{i}}} Q_{\mathrm{ij}}(k)} (1 - \frac{1}{|\mathcal{N}_{\mathrm{i}}|})$$

The entire algorithm from [32] is shown in Figure 3.5. The authors in [32] also mention a special case for stochastic topology. The challenge here is that the function $f(|x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k) + \omega_{\mathrm{ij}}(k)|, k)$ is the reward function and that in-turn updates the weight of the adjacency matrix. The problem here is that since x_{i} is a 5×1 vector, the algorithm either needs to choose a

Algorithm 1: WLA-F for a normal node $i \in V^{\mathrm{n}}$

Initialize $Q_{ij}(0) = 1,\ j \in N_i$
for $k = 1;\ k++$ **do**
  for $j = 1;\ j < n+1;\ j++$ **do**
    if $j \in N_i$ **then**
      $r_{ij}(k) = f(|x_j(k) - x_i(k) + \omega_{ij}(k)|);$
      $Q_{ij}(k) = Q_{ij}(k-1) r_{ij}(k);$
      $a_{ij}(k) = \frac{Q_{ij}(k)}{\sum_{j \in N_i} Q_{ij}(k)} (1 - \frac{1}{|N_i|});$
    else
      $a_{ij}(k) = 0;$
    end if
  end for
  $x_i(k+1) = x_i(k) + \sum_{j \in N_i} a_{ij}(k)(x_j(k) - x_i(k)) + \omega_i(k).$
end for

Figure 3.5. Algorithm for Weight Learning Method. Adopted from [32]

single value or have 5 different weighted adjacency matrices for each of the probability values. Since having 5 different matrices does not make sense due to extra memory requirement and every adversarial agent will send faulty values for all the probability values therefore the communication links that are not trustworthy will be the same for all matrices. Therefore, for this work, the mean of the output of $f(|x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k) + \omega_{\mathrm{ij}}(k)|, k)$ is taken as the value of the reward function. Therefore, we rewrite the equations for this case.

$$\begin{aligned} R_{\mathrm{ij}}(k) &= \mathrm{e}^{-|x_{\mathrm{j}}(k) - x_{\mathrm{i}}(k) + \omega_{\mathrm{ij}}(k)|\theta(k)} \\ r_{\mathrm{ij}}(k) &= \frac{R_{\mathrm{ij}}(k) \cdot \mathbf{1}}{|R_{\mathrm{ij}}(k)|} \end{aligned} \tag{3.3}$$

where Rij is a vector output which in this case will be a 5×1 vector and $\mathbf{1}$ is a column vector of all ones with the same dimensions as $R_{\mathrm{ij}}(k)$. The results of this algorithm are really promising and will be shown in the next section. Using this method, a system of agents can achieve consensus without identification of the attacker. Also, another advantage here is that many underlying assumptions such as f-total or f-local attack model are not required. Further, the graph need not be $2f + 1$ robust, rather the only condition here is that the topology of the regular nodes need to be strongly rooted.

3.3.2 Intersection of Convex Combinations of States Method

The final method that is discussed is the method where the intersection of convex hulls of the states of the agents help define the new states of the agents. This method is described in [31] where the initial states of each agent are used to find the convex hulls of a set of states. The intersection of these hulls form a 'safe kernel' that the states of the agents move towards. To understand the method and this thesis' contributions to this method, we introduce a few notations.

- For a vector a, a_{i} denotes the i-th component of this vector
- For any matrix M, $M^{\mathrm{i}}_{\mathrm{j}}$ represents the element in the i^{th} column and j^{th} row
- For a set $\mathcal{S} \subset \mathbb{R}^d$, Conv($\mathcal{S}$) denotes its convex hull which is essentially a set of all convex combinations of the points in $\mathcal{S}$ Here, d is the dimensionality.
- We denote the state vector of an agent x^{i} with dimensionality d as:

$$x^{\mathrm{i}} = \begin{bmatrix} x^{\mathrm{i}}_1 \\ x^{\mathrm{i}}_2 \\ . \\ . \\ . \\ . \\ x^{\mathrm{i}}_d \end{bmatrix}$$

- For a set $\mathcal{S}$, $|\mathcal{S}|$ denotes the cardinality of the set
- Let Ver(Conv(S)) denote the vertices of the convex hull
- We denote c as an index variable with a maximum value of $c = d - 1$. Each value of c represents one of the intersections of convex hulls for a state vector with dimensionality d.
- We denote each intersection of convex hulls w.r.t agent i as $\mathcal{R}^{\mathrm{i}}_c$

- Let F denote the total number of adversarial agents in the system
- We denote a set $\mathcal{S}(\mathcal{A}, n)$ where $\mathcal{A} \subset \mathbb{R}^d$ with cardinality m, then the set $\mathcal{S}(\mathcal{A}, n)$ is the set of all subsets of $\mathcal{A}$ with cardinality $m - n$. It is clear that $\mathcal{S}(\mathcal{A}, n)$ has $\binom{m}{n}$ elements

A function is introduced for a set $\mathcal{A} \subset \mathbb{R}^d$ with cardinality m, then

$$\Psi(\mathcal{A}, n) = \bigcap_{S \in \mathcal{S}(\mathcal{A}, n)} \text{Conv}(S) \tag{3.4}$$

The algorithm follows the F-Local or F-Total attack model, the update that each benign agent follows is detailed in Figure 3.6. Here $\mathcal{R}^{\text{i}}(k)$ as the 'safe-kernel' which is guaranteed to be within the convex hull formed by the benign agents. An example of the 'safe-kernel' of 5 agents is shown in Figure 3.7. Therefore, we see that a healthy agent moves its state inside of this 'safe-kernel'. The states of each agent are update by the rule:

$$x^{\text{i}}(k+1) = a_{\text{i}}^{\text{i}}(k) x^{\text{i}}(k) + \sum_{\bar{x}^{\text{j}}(k) \in \text{Ver}(\mathcal{R}^{\text{i}}(k))} a_{\text{j}}^{\text{i}}(k) \bar{x}^{k}(k) \tag{3.5}$$

Algorithm 1 Resilient consensus algorithm

1: Receive the states from all neighboring agents $j \in \mathcal{N}_i$, and collect these values in $\mathcal{X}^i(k)$.

2: Define $\mathcal{R}^i(k) \triangleq \Psi(\mathcal{X}^i(k), F)$, and denote the vertices of this set to be $\text{Ver}(\mathcal{R}^i(k))$. Agent i updates its local state as:

$$x^i(k+1) = a_i^i(k) x^i(k) + \sum_{\bar{x}^j(k) \in \text{Ver}(\mathcal{R}^i(k))} a_j^i(k) \bar{x}^j(k), \tag{3}$$

satisfying that each weight is lower bounded by some $\alpha > 0$, and $a_i^i(k) + \sum_{\bar{x}^j(k) \in \text{Ver}(\mathcal{R}^i(k))} a_j^i(k) = 1$.

3: Transmit updated state $x^i(k+1)$ to all neighbors $j \in \mathcal{N}_i$.

Figure 3.6. Algorithm for the Safe Kernel Method. Adopted from [31]

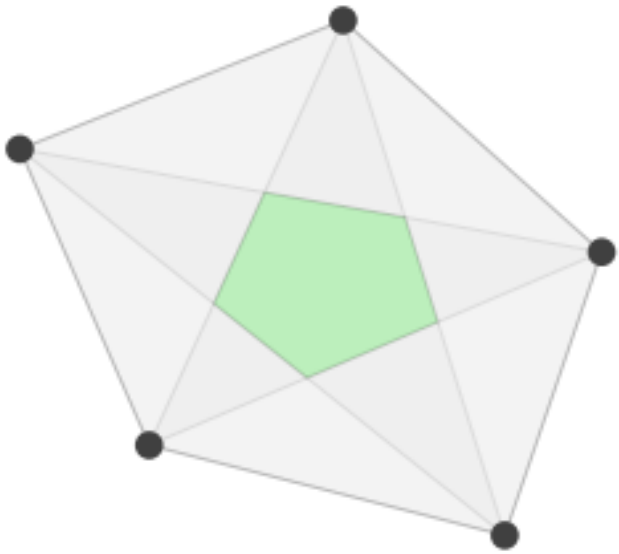


Figure 3.7. Safe kernel of 5 agents with F = 1. Adopted from [31]

The underlying assumption here is that for any i $\in \mathcal{V}$, $|\mathcal{N}_\text{i}| \geq (d+1)F+1$. Further, there is a limitation on the network topology. Here we define the topologies from [31]:

- (r-robust network) A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be r-robust if for any pair of disjoint and non empty subsets $\mathcal{V}_1, \mathcal{V}_2 \subsetneq \mathcal{V}$, then either there is more than one agent in $\mathcal{V}_1$ such that it has at least r neighbours outside $\mathcal{V}_1$ or there is more than one agent in $\mathcal{V}_2$ such that it has at least r neighbours outside $\mathcal{V}_2$.

- ((r,s)-robust network) the same conditions as the previous case hold except that there is an additional condition that the graph $\mathcal{G}$ may follow which is that there is no less than s agents in $\mathcal{V}_1 \cup \mathcal{V}_2$ such that each of them has at least r neighbours outside the set they belong to ($\mathcal{V}_1 or \mathcal{V}_2$)

This lets us understand the limitation on the network topology where if the F-Total attack model is used, then network must be $(dF+1, F+1)$-robust for the system to achieve consensus. And if the F-local attack model is followed, the graph needs to be $((d+1)F+1)$-robust for resilient consensus.

Now that we understand the algorithm in [31], we can quickly identify the limitation that the algorithm gets very complex as d increases since the dimensionality of the space for finding convex hulls increases. This means that it is harder to find convex hulls and their intersection in 4-D space rather than 2-D space. Also, we can see that the requirement for robustness in-terms of network topology is also directly proportional to d. In our case with $d = 5$ the network gets vast and it is harder to calculate the convex hulls in 5-D space. Looking at the assumptions, in the case of $d = 5$ and assuming $F = 1$, then each agent will need to have 7 neighbours and the graph will need to be $(6, 2)$-robust therefore the minimum number of agents is 12. A new algorithm is proposed that is an extension of the algorithm in [31] which significantly reduces the number of agents required for our problem by ensuring the dimensionality is always 2.

This proposed algorithm derives from the algorithm in Figure 3.6 and works by splitting each of the state vector into smaller pieces of size 2 and performing all the same steps as before. Essentially, the problem for this hypothesis is formulated in the next paragraph.

Let us now denote a variable c where $c = (1, 2, ..., d)$. Now, each component in agent i's state vector can be represented by $x_c^{\mathrm{i}}(k)$. Our proposed method follows that for every agent i, at each time step k, the vector $\begin{bmatrix} x_c^{\mathrm{i}} \\ x_{c+1}^{\mathrm{i}} \end{bmatrix}$ is made to follow the algorithm in Figure 3.6. If the algorithm is followed for each split vector $\begin{bmatrix} x_c^{\mathrm{i}} \\ x_{c+1}^{\mathrm{i}} \end{bmatrix}$, we hypothesize that at $k \to \infty$, each state vector will move towards the average of the initial values of all states.

Bearing this new problem, we come up with a the new proposed algorithm to incorporate this into the older algorithm. Algorithm 1 shows the process of updating its each state by each agent $\mathrm{i}, \forall \mathrm{i} \in \mathcal{B}$.

Algorithm 1: Resilience Algorithm

1. Receive the states from all neighbouring agents $\mathrm{j} \in \mathcal{N}_\mathrm{i}$ and each of these state vectors $x^\mathrm{j}(k)$ is appended as new column in a matrix $\mathcal{X}^\mathrm{i}(k)$ whose dimensions will be $d \times m$
2. **for** *each row* r *in* $\mathcal{X}^\mathrm{i}$ *till* $r = d-1$ **do**

   We define the r-th row in $\mathcal{X}^\mathrm{i}$ as s_r where s_r is a row vector with m elements ;

   Find the value of $\mathcal{R}^\mathrm{i}_r(k) = \Psi(\begin{bmatrix} s_r \\ s_{r+1} \end{bmatrix}, F)$.

end

3. Create 2 empty column vectors $p^\mathrm{i}(k)$ and $W^\mathrm{i}(k)$ both with dimensionality d.
4. **for** *each convex hull* $\mathcal{R}^\mathrm{i}_c(k)$ *in* $\mathcal{R}^\mathrm{i}(k)$ **do**

   Let c be the index of $\mathcal{R}^\mathrm{i}_c(k)$ in $\mathcal{R}^\mathrm{i}(k)$

   Let ρ be the number of vertices in $\mathcal{R}^\mathrm{i}_c(k)$

$$\begin{bmatrix} p^\mathrm{i}_c \\ p^\mathrm{i}_{c+1} \end{bmatrix} = \begin{bmatrix} p^\mathrm{i}_c \\ p^\mathrm{i}_{c+1} \end{bmatrix} + \sum_{\bar{x}_\theta \in \mathrm{Ver}(R^\mathrm{i}_c(k))} w^\mathrm{i}(\rho)\bar{x}_\theta$$

   where θ denotes the index of each vertex in $\mathcal{R}^\mathrm{i}_c(k)$

   In $W^2(k)$, assign $w^\mathrm{i}_c = w^\mathrm{i}(\rho)$

end

4. We update the values in the p vector as

$$\begin{bmatrix} p^\mathrm{i}_2 \\ . \\ . \\ p^\mathrm{i}_{d-1} \end{bmatrix} = 0.5 \times \begin{bmatrix} p^\mathrm{i}_2 \\ . \\ . \\ p^\mathrm{i}_{d-1} \end{bmatrix}$$

5. In $W^\mathrm{i}(k)$, assign $w^\mathrm{i}_d = w^\mathrm{i}_{d-1}$
6. Each agent update its state as:

$$x^\mathrm{i}(k+1) = W^\mathrm{i}(k) \odot x^\mathrm{i}(k) + p^\mathrm{i}(k)$$

The few functions required for this algorithm to run are:

- Each benign agent updates its states as

$$x^{\mathrm{i}}(k+1) = W^{\mathrm{i}} \odot x^{\mathrm{i}}(k) + p^{\mathrm{i}}(k) \tag{3.6}$$

  where $W^{\mathrm{i}}(k)$ is a column vector of weights. $p^{\mathrm{i}} \in \mathbb{R}^d$.

  $\odot$ is used to denote element-wise multiplication

- To find the weights w^{i} for updating the states, we use w as a function:

$$w^{\mathrm{i}}(\rho) = \frac{1}{\rho + 1} \tag{3.7}$$

Now that we have the algorithm, this method can be better explained through an example. We take an example of 6 interconnected agents with adjacency matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

‘ We generate random initial states for all the 6 agents. The states for each node is

$$x^1(0) = \begin{bmatrix} 0.0186 \\ 0.6748 \\ 0.4385 \\ 0.4378 \\ 0.1170 \end{bmatrix} x^2(0) = \begin{bmatrix} 0.8147 \\ 0.3249 \\ 0.2462 \\ 0.3427 \\ 0.3757 \end{bmatrix} x^3(0) = \begin{bmatrix} 0.5466 \\ 0.5619 \\ 0.3958 \\ 0.3981 \\ 0.5154 \end{bmatrix} x^4(0) = \begin{bmatrix} 0.6575 \\ 0.9509 \\ 0.7223 \\ 0.4001 \\ 0.8319 \end{bmatrix} x^5(0) = \begin{bmatrix} 0.1343 \\ 0.0605 \\ 0.0842 \\ 0.1639 \\ 0.3242 \end{bmatrix}$$

$$x^6(0) = \begin{bmatrix} 0.3017 \\ 0.0117 \\ 0.5399 \\ 0.0954 \\ 0.1465 \end{bmatrix}$$

The actual average of the states is avg = $[0.4122\ 0.4308\ 0.4045\ 0.3063\ 0.3851]^T$

We assume $F = 1$ and that the adversarial agent is node 1. Therefore, at every iteration, node 1 will update its state to a new set of random values. Therefore, we will look at the first update for agent 2.

The steps agent 2 takes to update its states are:

$$\mathcal{X}^2 = \begin{bmatrix} 0.0710 & 0.5466 & 0.6575 & 0.1343 & 0.3017 \\ 0.8877 & 0.5619 & 0.9509 & 0.0605 & 0.0117 \\ 0.0646 & 0.3958 & 0.7223 & 0.0842 & 0.5399 \\ 0.4362 & 0.3981 & 0.4001 & 0.1639 & 0.0954 \\ 0.8266 & 0.5154 & 0.8319 & 0.3242 & 0.1465 \end{bmatrix}$$

This matrix is built by making the state vectors of every neighbour of agent 2 as a column. Column 1 is denoted in red as it is the random states of malicious agent 1. The values in the column representing the states of agent 1 will change at every time step.

We define the r^th row in $\mathcal{X}^{\mathrm{i}}$ as s_r where s_r is a row vector with m elements. Hence, we take the first row of $\mathcal{X}^2$ as $s_1 = [0.0710\ 0.5466\ 0.6575\ 0.1343\ 0.3017]$. Note, the length of this row vector is $m = 5$. Since $F = 1$ we need to take every combination of $m - F = 4$ elements at a time from this row vector. We find the number of combinations by using the mathematical function $\binom{m}{n} = \frac{m!}{n!(m-n)!}$. In this case $\binom{5}{4}$ will give you 5 combinations of values taken 4 at a time from s_1. Each of these combinations is a row in the matrix below and

hence it has 4 elements in each row. Since there are 5 combinations, the total number of rows is 5.

$$\begin{bmatrix} 0.0710 & 0.5466 & 0.6575 & 0.1343 \\ 0.0710 & 0.5466 & 0.6575 & 0.3017 \\ 0.0710 & 0.5466 & 0.1343 & 0.3017 \\ 0.0710 & 0.6575 & 0.1343 & 0.3017 \\ 0.5466 & 0.6575 & 0.1343 & 0.3017 \end{bmatrix}$$

Similarly, we perform this operation for the second row of $\mathcal{X}^2$ i.e. s_2 and we find all the combinations of s_2 with $F = 1$ to get:

$$\begin{bmatrix} 0.8877 & 0.5619 & 0.9509 & 0.0605 \\ 0.8877 & 0.5619 & 0.9509 & 0.0117 \\ 0.8877 & 0.5619 & 0.0605 & 0.0117 \\ 0.8877 & 0.9509 & 0.0605 & 0.0117 \\ 0.5619 & 0.9509 & 0.0605 & 0.0117 \end{bmatrix}$$

To plot a convex hull in $2-D$ space, we need 2 coordinates for each point - x-coordinate and y-coordinate. This is the reason we perform the above operations on 2 vectors at a time - to obtain the x and y coordinates. Corresponding rows from the matrices above form the x and y coordinates respectively for each individual convex hull.

For example, the first rows of the matrices above are:

[0.0710 0.5466 0.6575 0.1343] and [0.8877 0.5619 0.9509 0.0605]

respectively. Therefore the coordinates of the vertices of the first convex hull will be:

$$(0.0710, 0.8877),\ (0.5466, 0.5619),\ (0.6575, 0.9509),\ \text{and},\ (0.1343, 0.0605)$$

These values are plotted in the Figure 3.8 as Convex Hull 1. Since the above matrices have 5 rows each, we will have a total of 5 convex hulls. Figure 3.8 shows the convex hulls being plotted together sequentially and the 6-th plot shows the intersection of all the 5 convex hulls.

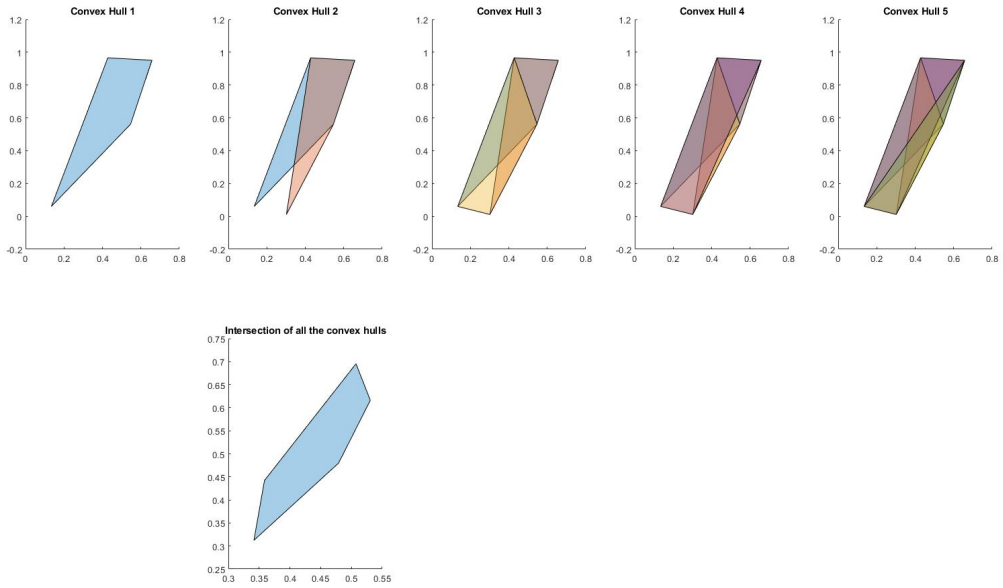


Figure 3.8. Intersection of Convex Hulls

As per the algorithm, this intersection of convex hulls will be called $\mathcal{R}_1^2(k)$. To find the intermediary states, we need to find all the convex hulls for $\mathcal{R}^2(k)$. Since there are 5 states for each agent i.e. 5 rows in $\mathcal{X}^2$ ($d = 5$), taking 2 rows at a time like how we did for s_1 and s_2 will yield 4 convex hull intersections. This means that taking s_1 and s_2 gave us $\mathcal{R}_1^2(k)$, taking s_2, and s_3 will give us $\mathcal{R}_2^2(k)$, and so on till $\mathcal{R}_4^2(k)$. We note that the variable $\mathcal{R}^2(k)$ contains all these convex hulls.

Now we calculate the p-vector as described hereafter. We first create an empty $p^2(k)$ vector which is a column vector with dimensionality d. The values of $p^2(k)$ are denoted by

$\begin{bmatrix} p_1^2 \\ p_2^2 \\ p_3^2 \\ p_4^2 \\ p_5^2 \end{bmatrix}$. We also create a $W^2(k)$ vector which is also a column vector with dimensionality d.

The values of $W^2(K)$ are denoted by $\begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \\ w_4^2 \\ w_5^2 \end{bmatrix}$

The formula that we use to calculate the p-vector is:

$$\begin{bmatrix} p_c^{\mathrm{i}} \\ p_{c+1}^{\mathrm{i}} \end{bmatrix} = \begin{bmatrix} p_c^{\mathrm{i}} \\ p_{c+1}^{\mathrm{i}} \end{bmatrix} + \sum_{\bar{x}_\theta \in \mathrm{Ver}(R_c^{\mathrm{i}}(k))} w^{\mathrm{i}}(\rho)\bar{x}_\theta \tag{3.8}$$

where θ denotes the index of each vertex in $\mathcal{R}_c^{\mathrm{i}}(k)$. Here, in the W^2 vector, we also equate $w_c^{\mathrm{i}} = w^{\mathrm{i}}(\mathrm{Ver}(R_c^{\mathrm{i}}(k)))$ from the above equation.

For this example, starting with $\mathcal{R}_1^2(k)$, this intersection has 5 vertices ($\rho = 5$) and each of the vertices has by 2 coordinates (2×1 vector). The formula for this case becomes:

$$\begin{bmatrix} p_1^2 \\ p_2^2 \end{bmatrix} = \begin{bmatrix} p_1^2 \\ p_2^2 \end{bmatrix} + \sum_{\bar{x}_\theta \in \mathrm{Ver}(R_1^2(k))} w^2(5)\bar{x}_\theta$$

This formula sums up the vertices of the intersection of convex hulls with a weight factor. Here $w_1^2 = w^2(\mathrm{Ver}(R_1^2(k)))$. In the above formula $\bar{x}_\theta$ is a 2×1 vector (x and y coordinate). All these 2×1 vectors are multiplied with w^2 and summed up to give $\begin{bmatrix} p_1^2 \\ p_2^2 \end{bmatrix}$. We get the values as $\begin{bmatrix} p_1^2 \\ p_2^2 \end{bmatrix} = \begin{bmatrix} 0.4297 \\ 0.4410 \end{bmatrix}$ and $w_1^2 = 0.1667$ because the number of vertices in $\mathcal{R}_1^2(k)$ is 5.

Now the algorithm looks at $\mathcal{R}_2^2(k)$ and uses equation 3.8 to find:

$$\begin{bmatrix} p_2^2 \\ p_3^2 \end{bmatrix} = \begin{bmatrix} 0.8499 \\ 0.3094 \end{bmatrix}$$

.

Here, $w_2^2 = w^2(\text{Ver}(R_2^2(k))) = 5$ because $\mathcal{R}_2^2(k)$ also has 5 vertices.
Note here that p_2^2 was updated by $\mathcal{R}_1^2(k)$ and $\mathcal{R}_2^2(k)$. After going through every intersection of convex hulls in $\mathcal{R}^2(k)$, we get our $p^2(k)$ as

$$p^2(k) = \begin{bmatrix} 0.4297 \\ 0.8499 \\ 0.6570 \\ 0.5040 \\ 0.3239 \end{bmatrix} W^2(k) = \begin{bmatrix} 0.1667 \\ 0.1667 \\ 0.1667 \\ 0.1667 \\ w_5^2 \end{bmatrix}$$

Now we note that p_2^2, p_3^2 and, p_4^2 were added updated twice in each time step so before we update the states of each agent, we divide the values of p_2^2, p_3^2 and, p_4^2 by 2. Therefore, we divide the value of every row in $p^2(k)$ except the first and the last row by 2. The first and the last rows were updated only once.

This gives us the value of the p-vector as:

$$p^2(k) = \begin{bmatrix} 0.4297 \\ 0.4249 \\ 0.3285 \\ 0.2520 \\ 0.3239 \end{bmatrix}$$

We also update the last value $w_5^2 = w_4^2$. This is because the last row (s_5) is updated with the same weight as s_4 with $\mathcal{R}_4^2(k)$ as:

$$\begin{bmatrix} p_4^2 \\ p_5^2 \end{bmatrix} = \begin{bmatrix} p_4^2 \\ p_5^2 \end{bmatrix} + \sum_{\bar{x}_\theta \in \text{Ver}(R_4^2(k))} w^2(\text{Ver}(R_4^2(k)))\bar{x}_\theta$$

and hence $w_4^2 = w_5^2$. With $p^2(k)$ and $W^2(k)$, we update the final state of agent 2 as $x^2(k+1) = W^2(k) \odot x^2(k) + p^2(k)$ and we get:

$$x^2(k+1) = \begin{bmatrix} 0.5654 \\ 0.4791 \\ 0.3695 \\ 0.3091 \\ 0.3865 \end{bmatrix}$$

This process is followed by every normal node at every time step. As a result, this system converges in 2 time steps. The resulting states of the agents at $k = 2$ are:

$$x^1(k) = \begin{bmatrix} 0.0186 \\ 0.6748 \\ 0.4385 \\ 0.4378 \\ 0.1170 \end{bmatrix} x^2(k) = \begin{bmatrix} 0.4920 \\ 0.2938 \\ 0.4156 \\ 0.3282 \\ 0.4309 \end{bmatrix} x^3(k) = \begin{bmatrix} 0.4907 \\ 0.2948 \\ 0.4113 \\ 0.3281 \\ 0.4318 \end{bmatrix} x^4(k) = \begin{bmatrix} 0.4917 \\ 0.2973 \\ 0.4088 \\ 0.3281 \\ 0.4314 \end{bmatrix} x^5(k) = \begin{bmatrix} 0.4921 \\ 0.2972 \\ 0.4106 \\ 0.3283 \\ 0.4317 \end{bmatrix}$$

$$x^6(k) = \begin{bmatrix} 0.4918 \\ 0.2926 \\ 0.4165 \\ 0.3219 \\ 0.4322 \end{bmatrix}$$

We calculate the new average of these updated states at $k = 2$ as $\text{avg}_{\text{new}} = [0.4917 \;\; 0.2951 \;\; 0.4125 \;\; 0.3269 \;\; 0.4316]^T$ which is nothing but the mean of the states of the benign agents at $k = 2$.

The calculated error between actual mean values and the algorithm obtained mean values is **0.97%**. As we can see this method is fairly effective in reducing the number of agents that are required. The results will be discussed in the next section.

3.4 Results

We will first look at the results of the weight learning algorithm and then the result of the convex combination of states algorithm.

The weight learning algorithm works really well and achieves consensus for all the tested cases. The number of agents and the number of faulty agents were varied but the algorithm still gave good results. The algorithm is not computationally intensive and on following equation 3.3, we get the average states very easily. The consensus values are really far from the actual mean where more than a third of the system is adversarial nodes although this is as expected. The number of steps time steps before consensus is achieved has been in the range of 7-9 time steps. This algorithm has many advantages and is exceptionally robust in dealing with adversarial agents in the system.

The intersection of convex hulls algorithm shows great promise. The modified version of this algorithm has been tested in many different scenarios with different number of agents and different number of adversarial nodes. Simulations show that this algorithm works well and significantly reduces the limitations posed by the original algorithm in [31]. It is evident that each split vector $\begin{bmatrix} x_c^{\mathrm{i}} \\ x_{c+1}^{\mathrm{i}} \end{bmatrix}$ which is a part of the whole state vector will achieve average consensus by following the algorithm in [31]. It is then clear, that combining all these spilt vectors after averaging out the state values in between will be inside the 'safe-kernel' of the entire state vector. Further, simulation results agree with the fact that the algorithm works and can reduce dimensionality of the state vector to perform the intersection of convex combinations method.

Thus we can safely say that our image classification system can now achieve resilient average consensus through either of the methods detailed in this chapter. The choice of

method would depend on the user's parameters but both methods are very effective and have been proven in this thesis to work well with our multi-agent image classification system.

4. BOAT TRACKING USING OBJECT DETECTION

4.1 Introduction

This section describes a novel method to tackle a challenge put forth by the United States Department of Defense - Department of the Navy, Naval Information Warfare Center Pacific [9]. The purpose of this challenge is to detect a specified target (the given boat) over the course of a video file recorded by a single electro-optical camera and predict its path or track by providing relevant GPS coordinates for each frame of the video file. The only data that was provided was the GPS coordinates of the camera which recorded the videos, the GPS coordinates of the boat that was supposed to be tracked (to provide ground-truth data), along with relevant video files recorded by the camera to help train the model.

We start our approach by understanding what a neural network can and cannot do. A neural network can identify mathematical relationships between the input and the output hence predicts outputs using given inputs by adjusting its weights. A neural network cannot be fed input that is seemingly random (not in a distribution) and expect an output without any meaningful relationship between the input and the output. Our approach then is to not only build a neural network that is capable of providing the output we need but also to choose the inputs and outputs so that our network actually solves the problem and gives the required solution.

4.2 Problem Formulation

The problem at hand is to model the relationship between the GPS coordinates of a source and the GPS coordinates of a target only through the source's view of the target. In this case, the source is a camera that record the boat in the sea. The target is the boat being recorded and the only information available is the camera's GPS coordinates and the video captured of the boat. An example of a frame of the video provided is shown in Figure 4.1

Figure 4.1. Frame captured from video of boat

The sub-parts in the problem would include identifying the position of the boat in the frame and then finding the GPS output from the position of the boat in the frame. The method used to perform this task is described in the section below.

4.3 Method

As described in the introduction section, we will try to find a relationship between the input and the output that makes mathematical sense for a neural network to compute. The problem at hand is to identify the GPS coordinates of a boat in video given only the video and GPS coordinates of the camera. An easy approach here would be to know to draw a line from the source(camera) to the target(boat in the image) and using the intrinsic and extrinsic properties of the image and the camera, we calibrate our algorithm to estimate the GPS coordinate of any point given the information for one particular point. However, since we do not possess any properties of the camera and image, we leave the entirety of the 'relationship building' between the input and the output to the neural network.

The method we developed is to solve the problem by training an object detection model using the initial video data provided so that we had a guaranteed match with our specified target in every video as there were several frames in which multiple boats were visible. Essentially, here an object detection model was trained to identify the specific boat required. Of course, a more general purpose version can identify any boat and give the GPS coordinates for all of them but that was not the purpose of this challenge. We used over 12000 frames with our target boat in them to train our model to improve its accuracy. We then created an algorithm that provided the height, width and x and y coordinates of each target detected in a frame. This data is sent to a neural network that compares the height, width, x and y coordinates of this target with data that was calculated initially and outputs an approximate GPS position of the boat. This process is detailed in the next paragraphs.

The first task was understanding the GPS data from the GPS logs and creating an organized and processed dataset that can be easily analysed. This section of the algorithm was written on MATLAB and helps us to retrieve relevant data from GPS logs. These GPS logs with our algorithm was matched to the respective frames in the video using the timestamp data to match the corresponding GPS data and frames. Figure 4.2 shows a snippet of the processed dataset with the corresponding information in each row.

Frame #	LatBoat	LonBoat	LatCam	LonCam	Distance	Bearing
91	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
92	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
93	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
94	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
95	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
96	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
97	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
98	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
99	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
100	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
101	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
102	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
103	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
104	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
105	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
106	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
107	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
108	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
109	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
110	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
111	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
112	32.7029	-117.233	32.703	-117.235	110.464	-95.0429
113	32.7029	-117.233	32.703	-117.235	110.464	-95.0429

Figure 4.2. Processed Dataset of GPS logs and Frame matching

In the processed dataset, there are also 2 other fields that are added which are the distance and bearing. This is essentially the ground-truth data which we calculated by using the GPS coordinates of the camera and the GPS coordinates of the boat. Again, the GPS coordinates of the boat are provided only to obtain the ground-truth data. We use the distance and bearing because as we mentioned before, it is imperative to give the neural network values it can find the relationship between. The location of a boat in the frame of a video can be shown with the distance and bearing to that region in the image from the camera. Figure 4.3 shows the relevancy of distance and bearing to the camera image. Distance is the length between the 2 GPS points and bearing is the angle between these 2 points with respect to the vertical axis.

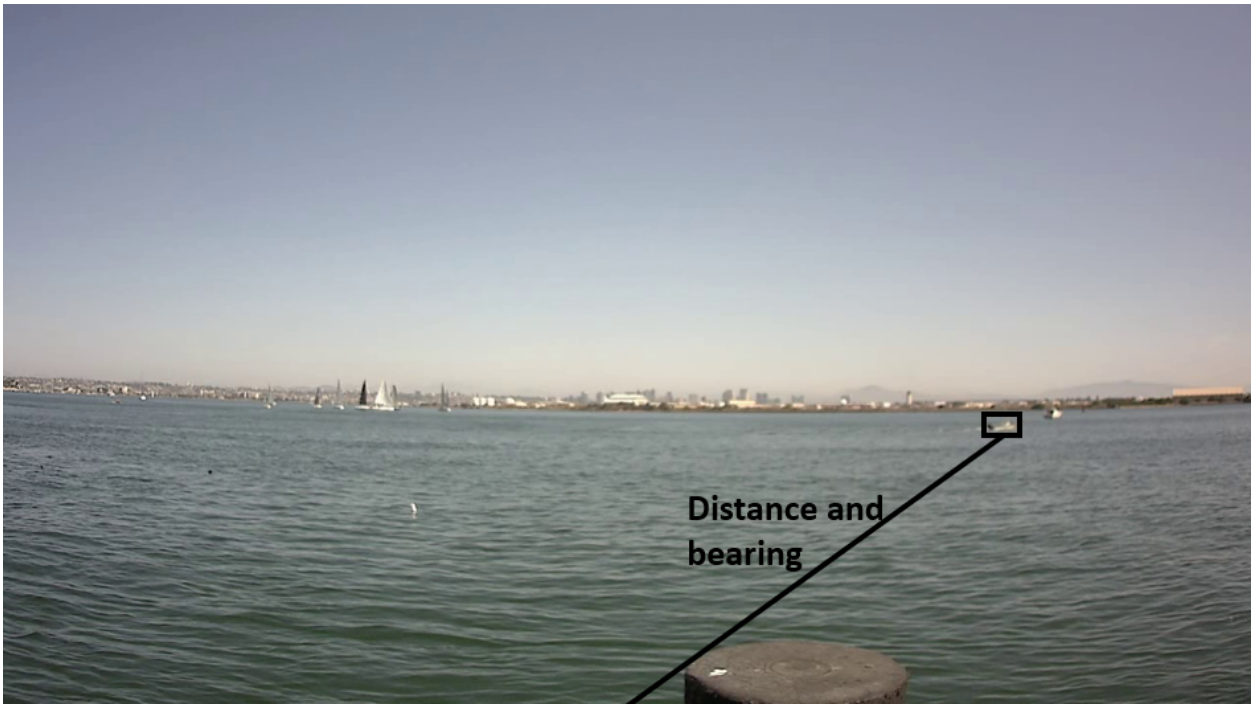


Figure 4.3. Relevance of distance and bearing of GPS coordinates to the camera image

From Figure 4.3 it is evident how distance and bearing can be seen as physical quantities on the image with respect to the camera and the position of the boat in the image. This is the reason that distance and bearing were chosen to be the representing factors for our model.

The formulae for calculating the distance and bearing are given by the Haversine formula and the bearing formula:

$$\begin{aligned} a &= \sin^2(\Delta\phi/2) + \cos(\phi_1).\cos(\phi_2).\sin^2(\Delta\lambda/2) \\ c &= 2.\arctan(\sqrt{a}, \sqrt{1-a}) \\ d &= R.c \end{aligned} \tag{4.1}$$

Here, ϕ_1, λ_1 are the starting point's latitude and longitude respectively while ϕ_2, λ_2 are the target point's latitude and longitude respectively. $\Delta\phi$ and $\Delta\lambda$ are the differences in the latitudes and longitudes respectively. R is the radius of the earth (mean= $6,371$km). The distance is obtained through the haversine formula while the bearing is obtained through the formula:

$$\theta = \arctan(\sin\Delta\lambda.\cos\phi_2,\ \ \cos\phi_1.\sin\phi_2 - \sin\phi_1\cos\phi_2.\cos\Delta\lambda) \tag{4.2}$$

Now that we have the parameters d and θ that could identify a relationship between the input and the output, the method proposed here is to use an object detector to detect the boat in the image. With the help of this object detector, we will be able to extract the ROI (region of interest) in the image. In this case the region of interest will be a bounding box (a boundary drawn around the object) of the boat in the image. The parameters that the object detector will give us regarding this bounding box are the height, width, x and y pixel coordinates of the bounding box in the image. This x and y is the location of the centre of the bounding box with respect to the image.

We obtain these values of the height, width, x and y coordinates of the bounding box through an object detector. The object detector obviously needs to be light so that it can detect the object in the image with speed and accuracy. Hence, we chose an existing object detector called YOLOv4 [55]. This network is an improvement from the already popular and amazing object detection network YOLOv3 developed by Joseph Redmon [56]. The YOLO network which stands for You Only Look Once uses a neural network that divides the image into different regions and then the network predicts the bounding boxes and

probabilities for each region. The pre-trained version of the YOLOv4 network is available which has been trained on the COCO dataset [57]. The COCO dataset contains about 80 object categories. Although we identified that the COCO dataset had the category 'boats' in it, the YOLO network could not always detect the boat that we wanted from the frame. Also, it sometimes detected many boats in the image. Since the ground-truth data for only one boat existed, we had to use transfer learning again in order to ensure that the YOLO network could detect our boat. Therefore, the dataset had to be manually annotated to find the boat in the image. This annotation was done using software developed for this work. Finally, after training the YOLOv4 network to detect the boat in the image, we have successfully extracted the bounding box data of the boat in every frame of the video.

The bounding box data is fed into a custom-built neural network that takes in the height, width, x and y coordinates of the bounding box to give out the distance and bearing. Since the GPS coordinates of the source (camera) is known, we can easily calculate the target (boat) GPS coordinates from the distance and the bearing that is the output. This network was trained with the ground-truth distance and bearing. The full structure of our solution is shown in Figure **??**.

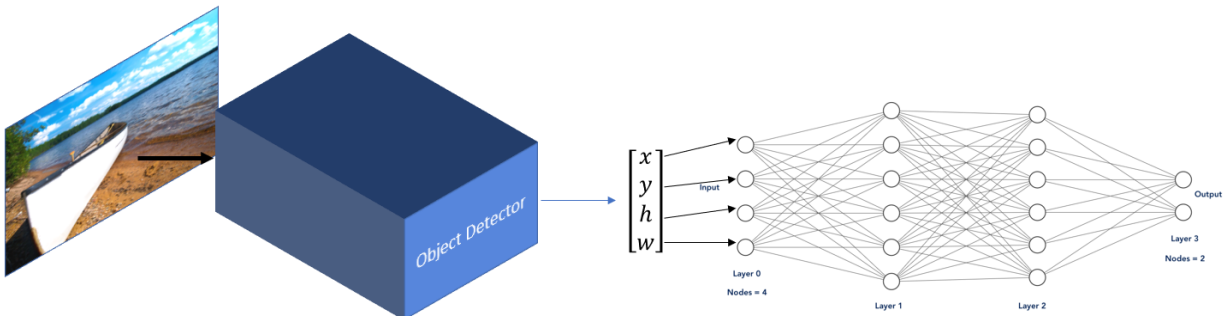


Figure 4.4. AI Tracks Solution Structure

With the ground-truth data, the custom network was trained. The structure of the network is shown in Table 4.1.The training statistics are shown in Figure 4.5. While training

the parameters include the initial learning rate $\alpha = 0.001$, the ADAM optimizer and the loss function used is the Mean Squared Error loss function which is given by the formula:

$$\text{MSE} = \frac{1}{n}\sum_{\text{i}=1}^{n}(Y_{\text{i}} - \hat{Y}_{\text{i}})^2$$

where n is the number of data points, Y_{i} is the actual value and $\hat{Y}_{\text{i}}$ is the predicted value.

Table 4.1. Custom network structure

Layer	Dimensions
Input	4 (height, width, x and y)
FC-1	8
FC-2	8
FC-3	4
FC-4	2 (Output: Distance and Bearing)

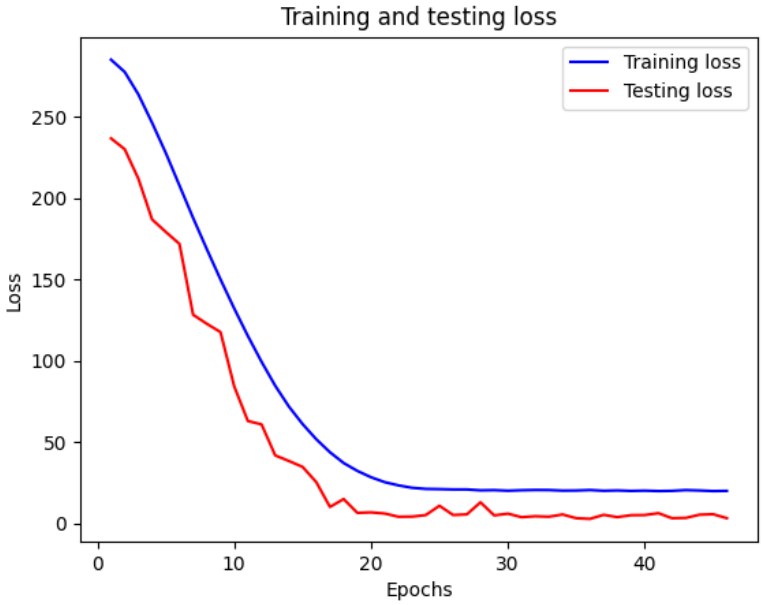


Figure 4.5. Training custom-built network

In the next section, the results of this method is discussed.

4.4 Results

The network was tested to output the GPS coordinates of the boat from the distance and bearing that the neural network calculates. The GPS coordinates of the camera was provided to the algorithm and the video of the boat was provided. Further, the GPS coordinates given from our model was compared to the ground-truth GPS coordinates and the error between them was calculated. The Root Mean Squared Error was calculated between the ground-truth and the model output. The values from this error calculation were:

Max RMSE: 0.0014303197713278148,
Min RMSE: 0.0007053464384854741,
Average RMSE: 0.0009926212150476418

This means that the average root mean squared error between the coordinates in terms of real world distance is less than a few inches which is more than the required level of accuracy typically expected from GPS coordinates. Further, the time taken to calculate the GPS coordinate from the input video was reported to be 11 seconds for 50 frames which gives us an average frame speed of 4.5 fps. Therefore, we conclude from the results that the algorithm designed for this work accurately gave the GPS coordinates of the target without compromising on speed.

5. SUMMARY

5.1 Conclusion

This thesis delved into image classification, application of image classification to emergency response UAVs, extension of image classification to multi-agent systems, resilience in multi-agent image classification systems and, object tracking in video to give GPS coordinates. We looked at an introduction to image classification and the working of neural networks was explained. once the image classifier for emergency situations was built using the AIDER dataset, the problem was extended to multi-agent systems. Different methods were also shown here to improve image classification accuracy in single agents which then later would be applied to multi-agent systems. The methods shown were to build a regular CNN that is both light and accurate using many design parameters and using transfer learning to use pre-trained models to suit the task at hand. The performance of the custom baseNet and the transfer learnt VGG11 are shown. The trade-off between speed and accuracy would cause oneself to use one of these networks over another. The trade-off will depend on the exact nature of the situation but both methods work really well and provide accurate results for image classification with the AIDER dataset.

It was shown that multi-agent systems would prove more efficient in emergency situations and that having multiple agents in such situations was a good contingency plan in case single agents malfunction. However, upon extending this problem to the multi agent system we saw that the accuracy of predictions of the image class significantly improved especially when noise or other errors/defects are present in the input images. It was shown that using a multi-agent system not only improves reliability and robustness of the system but also classification accuracy of the system. Therefore, it is implied that sending a swarm of UAVs running the algorithm discussed in chapter 2 of this thesis will show a high rate of success in completing the operation it set to do.

Next, the result of image classification for a multi-agent system was further extended to an attack resistant system. Here, the multi-agent system that classifies images was

assumed to be under attack from malicious agents that try to ensure that the system does not reach consensus. Different algorithms for moving past such attacks were shown along with their underlying assumptions. Every method has its limitations and advantages but all methods were shown to work with our multi-agent image classifier. One of the methods in the system which involved moving the states of each agent towards the intersection of convex hulls of the states of all agents in the system. This method had very high memory and minimum number of agents in the system requirements. A modified version of this algorithm was presented in this thesis that reduced the intensity of the computations and drastically reduced the minimum number of agents and connections the system needs to have to achieve resilient consensus. The specific method to be utilized in the multi-agent image classification system will again depend on the circumstance and which algorithm would work best for that particular circumstance, but all algorithms are shown to work well with our system and the work has been properly extended and executed on our multi-agent image classification system.

Finally, the method to track boats from a video using only the GPS coordinates of the camera used to take that video was presented in chapter 4. This novel approach to track the GPS coordinates of objects in the frame can be extended to any application. The work in this thesis was specific to boats as the challenge that the Purdue team in the AIMS lab participated in required the team to track boats in the image. The system used an object detector and then a custom-built network to obtain the outputs with high accuracy and speed.

5.2 Future Work

Extensions of this work in the future can be:

- Deploy the multi-agent image classifier on actual UAVs to gather experimental data and perform the necessary updates and corrections to the algorithms if need be. It

may be found that one particular algorithm is more suited to being deployed on actual UAVs than the other.

- Train more models with more data to improve the accuracy of the image classifier much further. Use different video processing techniques such as analyzing only ever i-th frame rather than every frame and then train the network accordingly so that the video processing becomes faster.

- Try a different approach to the weight learning algorithm for resilient consensus that can easily achieve accurate consensus no matter the dimensionality of the state vector similar to the modified version of the intersection of convex hulls of states algorithm.

- Train a custom object detector using similar principles as the YOLOv4 described in chapter 4 to make the object detection even faster and maybe more accurate.

REFERENCES

[1] G. Oppy and D. Dowe, "The Turing Test," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., Winter 2020, Metaphysics Research Lab, Stanford University, 2020.

[2] R. Wang. (). Color processing examples, [Online]. Available: http://fourier.eng.hmc.edu/e161/lectures/ColorProcessing/node4.html (visited on 10/10/2014).

[3] D. A. Forsyth and J. Ponce, *Computer Vision - A Modern Approach, Second Edition*. Pitman, 2012, pp. 1–791, ISBN: 978-0-273-76414-4.

[4] M. Simon. (). The wired guide to robots, [Online]. Available: https://www.wired.com/story/wired-guide-to-robots/ (visited on 04/16/2014).

[5] J. A. Adams, *Robotic technologies for first response: A review*. CRC Press, 2010, ISBN: 978-0-429-09852-9.

[6] M. B. Bejiga, A. Zeggada, A. Nouffidj, and F. Melgani, "A convolutional neural network approach for assisting avalanche search and rescue operations with uav imagery," *Remote Sensing*, vol. 9, no. 2, 2017, ISSN: 2072-4292. DOI: 10.3390/rs9020100. [Online]. Available: https://www.mdpi.com/2072-4292/9/2/100.

[7] J. Tucker. (). Drone use could soon soar in search and rescue operations, [Online]. Available: http://www.telemedmag.com/drone-use-soon-soar-search-rescue-operations/ (visited on 03/15/2018).

[8] S. Ehsan and K. D. McDonald-Maier, "On-board vision processing for small uavs: Time to rethink strategy," in *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, 2009, pp. 75–81. DOI: 10.1109/AHS.2009.6.

[9] N. I. W. C. P. Department of Defense - Department of the Navy. (). Ai tracks at sea, [Online]. Available: https://www.challenge.gov/challenge/AI-tracks-at-sea/.

[10] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *CoRR*, vol. abs/1404.5997, 2014. arXiv: 1404.5997. [Online]. Available: http://arxiv.org/abs/1404.5997.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015. arXiv: 1512.00567. [Online]. Available: http://arxiv.org/abs/1512.00567.

[14] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. arXiv: 1801.04381. [Online]. Available: http://arxiv.org/abs/1801.04381.

[15] L. X, Z. J, Y. Zhao, and M. J, "Saliency detection and deep learning-based wildfire identification in uav imagery," *PubMed Central*, vol. 18, no. 3, p. 712, Feb. 2018. DOI: 10.3390/s18030712. [Online]. Available: http://www.mdpi.com/1424-8220/18/3/712.

[16] C. Hou, F. Nie, D. Yi, and Y. Wu, "Efficient image classification via multiple rank regression," *IEEE Transactions on Image Processing*, vol. 22, no. 1, pp. 340–352, 2013. DOI: 10.1109/TIP.2012.2214044.

[17] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 2567–2572. DOI: 10.1109/SMC.2017.8123011.

[18] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 33–46, 2007, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2006.08.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731506001808.

[19] H. K. Mousavi, M. Nazari, M. Takáč, and N. Motee, "Multi-agent image classification via reinforcement learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5020–5027. DOI: 10.1109/IROS40897.2019.8968129.

[20] W. Wu, D. He, X. Tan, S. Chen, and S. Wen, "Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6222–6231.

[21] F. Pourpanah, C. J. Tan, C. P. Lim, and J. Mohamad-Saleh, "A q-learning-based multi-agent system for data classification," *Appl. Soft Comput.*, vol. 52, no. C, pp. 519–531, Mar. 2017, ISSN: 1568-4946. DOI: 10.1016/j.asoc.2016.10.016. [Online]. Available: https://doi.org/10.1016/j.asoc.2016.10.016.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] A. Quteishat, C. P. Lim, J. M. Saleh, J. Tweedale, and L. C. Jain, "A neural network-based multi-agent classifier system with a bayesian formalism for trust measurement," *Soft computing*, vol. 15, no. 2, pp. 221–231, 2011.

[24] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2017, pp. 1–5. DOI: 10.1109/ICACCS.2017.8014672.

[25] L. Zhang, G. Ding, Q. Wu, Y. Zou, Z. Han, and J. Wang, "Byzantine attack and defense in cognitive radio networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1342–1363, 2015.

[26] J. Mattila, "The blockchain phenomenon," *Berkeley Roundtable of the International Economy*, p. 16, 2016.

[27] S. M. Dibaji, M. Safi, and H. Ishii, "Resilient distributed averaging," in *2019 American Control Conference (ACC)*, 2019, pp. 96–101. DOI: 10.23919/ACC.2019.8814959.

[28] D. Saldaña, A. Prorok, S. Sundaram, M. F. M. Campos, and V. Kumar, "Resilient consensus for time-varying networks of dynamic agents," in *2017 American Control Conference (ACC)*, 2017, pp. 252–258. DOI: 10.23919/ACC.2017.7962962.

[29] C. Zhao, J. He, and Q.-G. Wang, "Resilient distributed optimization algorithm against adversary attacks," in *2017 13th IEEE International Conference on Control Automation (ICCA)*, 2017, pp. 473–478. DOI: 10.1109/ICCA.2017.8003106.

[30] X. Wang, S. Mou, and S. Sundaram, *A resilient convex combination for consensus-based distributed algorithms*, 2018. arXiv: 1806.10271 [math.OC].

[31] J. Yan, Y. Mo, X. Li, and C. Wen, "A "safe kernel" approach for resilient multi-dimensional consensus," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 2507–2512, 2020, 21th IFAC World Congress, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.224. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S240589632030495X.

[32] J. Hou, Z. Chen, Z. Lin, and M. Xiang, "Resilient consensus via weight learning and its application in fault-tolerant clock synchronization," *CoRR*, vol. abs/2002.03541, 2020. arXiv: 2002.03541. [Online]. Available: https://arxiv.org/abs/2002.03541.

[33] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, 13–es, Dec. 2006, ISSN: 0360-0300. DOI: 10.1145/1177352.1177355. [Online]. Available: https://doi.org/10.1145/1177352.1177355.

[34] W. Brendel and S. Todorovic, "Video object segmentation by tracking regions," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 833–840. DOI: 10.1109/ICCV.2009.5459242.

[35] W. Krüger and Z. Orlov, "Robust layer-based boat detection and multi-target-tracking in maritime environments," in *2010 International WaterSide Security Conference*, 2010, pp. 1–7. DOI: 10.1109/WSSC.2010.5730254.

[36] D. Bloisi and L. Iocchi, *Argos – a video surveillance system for boat traffic monitoring in venice*, 2009.

[37] F. Sultana, A. Sufian, and P. Dutta, "Advancements in image classification using convolutional neural network," in *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, IEEE, 2018, pp. 122–129.

[38] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[39] K. Sarkar. (). Relu : Not a differentiable function: Why used in gradient based optimization? and other generalizations of relu., [Online]. Available: https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec.

[40] J. Brownlee. (). Loss and loss functions for training deep learning neural networks, [Online]. Available: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[41] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*, Springer, 2018, pp. 270–279.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[43] S. Ruder. (). Transfer learning - machine learning's next frontier, [Online]. Available: https://ruder.io/transfer-learning/index.html#fn6.

[44] C. Kyrkou and T. Theocharides, "Deep-learning-based aerial image classification for emergency response applications using unmanned aerial vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019.

[45] C. Kyrkou, S. Timotheou, P. Kolios, T. Theocharides, and C. G. Panayiotou, "Optimized vision-directed deployment of uavs for rapid traffic monitoring," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2018, pp. 1–6.

[46] T. van Laarhoven, "L2 regularization versus batch and weight normalization," *CoRR*, vol. abs/1706.05350, 2017. arXiv: 1706.05350. [Online]. Available: http://arxiv.org/abs/1706.05350.

[47] D. A. Van Dyk and X.-L. Meng, "The art of data augmentation," *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001.

[48] A. Gandhi. (). Data augmentation | how to use deep learning when you have limited data — part 2, [Online]. Available: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[49] V. Alto. (). Data augmentation in deep learning, [Online]. Available: https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[52] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[53] R. Carli, F. Fagnani, P. Frasca, and S. Zampieri, "Gossip consensus algorithms via quantized communication," *Automatica*, vol. 46, no. 1, pp. 70–80, 2010, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2009.10.032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109809004932.

[54] J. Hunt. (). The byzantine generals' problem, [Online]. Available: https://steemit.com/bitcoin/@humanjets/the-byzantine-generals-problem.

[55] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: 2004.10934. [Online]. Available: https://arxiv.org/abs/2004.10934.

[56] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767.

[57] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.