NPS-IS-23-007

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

IMPROVED METHODS OF COMBAT IDENTIFICATION

by

Dr. Johnathan Mun & Ms. Jessica Kimball

October 2023

Approved for public release; distribution is unlimited.

Prepared for: N2/N6 – Information Warfare

This research is supported by funding from the Naval Postgraduate School, Naval Research Program (PE 0605853N/2098). NRP Project ID: NPS-23-N231-A

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
Oct 21, 2023	Technical Report	START DATE Oct 23, 2022	END DATE Oct 22, 2023

4. TITLE AND SUBTITLE
Improved Methods of Combat Identification

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
		0605853N/2098
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
NPS-23-N231; W2223/23-N231A		

6. AUTHOR(S)
Dr. Johnathan Mun, Professor of Research & Ms. Jessica Kimball, Ph.D. Student

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Naval Postgraduate School 1 University Circle, Monterey, CA 93943	NPS-IS-23-007

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Naval Postgraduate School, Naval Research Program; N2/N6 Information Warfare	NRP; N2/N6 – IW	NPS-23-N231-A

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

14. ABSTRACT
Complex sensor networks are an enormously large and expensive set of systems, software, sensors, and emitters that generate data contributing to battlespace awareness and combat identification. Multiple sensor network configurations can collect so much data that an ineffective "Data Rich, Information Poor" (DRIP) situation results. The desired solution for actionable information is coordinating high-value units (HVUs) to support actions such as identification and targeting. Timely detection of unknown signals is currently inadequate to maintain situational awareness at a tactical level. Few analysts have the experience and data access required to make effective use of the available data. Those who do are not close enough to the edge to support warfighting's tactical and operational levels. Automating the workflow and processes of well-seasoned analysts combined with new AI and modeling technologies would enable improved and more timely extraction of essential information from the data and support better situational awareness and tactical decision-making. The motivation for analysis of combat and target identification stems from the significant pressure usually applied to make rapid, effective, and informed decisions. The goal of target ID is to analyze the threat of a potential target in order to make informed decisions about engaging it. The benefits of leveraging these graph-based methodologies include greater situational awareness with automation tools to distill contextualized information at the tactical and operational levels.

15. SUBJECT TERMS
Combat ID, graph theory, artificial intelligence

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT U	b. ABSTRACT U	C. THIS PAGE U	UU	69

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
William Treadway	703-693-8008

STANDARD FORM 298 (REV. 5/2020)

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL

Monterey, California 93943-5000

Ann E. Rondeau	Scott Gartner
President	Provost

This report entitled “Improved Methods of Combat Identification" was prepared for N2/N6 – Information Warfare and funded by the Naval Postgraduate School, Naval Research Program (PE 0605853N/2098).

Further distribution of all or part of this report is authorized.

This report was prepared by:

MUN.JOHNATHAN.C.1282391216 Digitally signed by MUN.JOHNATHAN.C.1282391216 Date: 2023.10.04 09:02:20 -07'00'

Dr. Johnathan Mun

Professor of Research

KIMBALL.JESSICA.SPEIR.1367355510 Digitally signed by KIMBALL.JESSICA.SPEIR.1367355510 Date: 2023.10.04 10:49:30 -07'00'

Ms. Jessica Kimball

Ph.D. Student

Reviewed by:

COOK.GLENN.ROBERT.1040837619 Digitally signed by COOK.GLENN.ROBERT.1040837619 Date: 2023.10.04 10:56:09 -07'00'

Dr. Alex Bordetsky, Chairman

Department of Information Science

Released by:

SMITH.KEVIN.B.1230026366 Digitally signed by SMITH.KEVIN.B.1230026366 Date: 2023.10.04 16:28:07 -07'00'

Kevin B. Smith

Vice Provost for Research

THIS PAGE INTENTIONALLY LEFT BLANK

Improved Methods of Combat Identification

Johnathan Mun, Ph.D.
Professor of Research
Naval Postgraduate School

Jessica Kimball
Ph.D. Student
Naval Postgraduate School

Naval Research Program (NRP)

Topic Sponsor Organization: N2/N6 – Information Warfare
Topic Sponsor: Mr. William Treadway

TABLE OF CONTENTS

ABSTRACT

Complex sensor networks are an enormously large and expensive set of systems, software, sensors, and emitters that generate data contributing to battlespace awareness and combat identification. Multiple sensor network configurations can collect so much data that an ineffective "Data Rich, Information Poor" (DRIP) situation results. The desired solution for actionable information is coordinating high-value units (HVUs) to support actions such as identification and targeting. Timely detection of unknown signals is currently inadequate to maintain situational awareness at a tactical level. Few analysts have the experience and data access required to make effective use of the available data. Those who do are not close enough to the edge to support warfighting's tactical and operational levels. Automating the workflow and processes of well-seasoned analysts combined with new AI and modeling technologies would enable improved and more timely extraction of essential information from the data and support better situational awareness as well as support tactical decision making. The motivation for analysis of combat and target identification stems from the significant pressure usually applied to make rapid, effective, and informed decisions. The goal of target ID is to analyze the threat of a potential target in order to make informed decisions about engaging it. The benefits of leveraging these graph-based methodologies include greater situational awareness with automation tools to distill contextualized information at the tactical and operational levels.

EXECUTIVE SUMMARY

Project Summary

Complex sensor networks are an enormously large and expensive set of systems, software, sensors, and emitters that generate data contributing to battlespace awareness and combat identification. Multiple sensor network configurations can collect so much data that an ineffective "Data Rich, Information Poor" situation results. The desired solution for actionable information is coordinating high-value units to support actions such as identification and targeting. Timely detection of unknown signals is currently inadequate to maintain situational awareness at a tactical level. Few analysts have the experience and data access required to use the available data effectively. Those who do are not close enough to the edge to support warfighting's tactical and operational levels. Automating the workflow and processes of well-seasoned analysts combined with new artificial intelligence and analytical modeling technologies would enable improved and more timely extraction of essential information from the data and support better situational awareness and tactical decision-making. The motivation for analysis of combat and target identification stems from the significant pressure usually applied to make rapid, effective, and informed decisions. The goal of target identification is to analyze the threat of a potential target to make informed decisions about engaging it. The benefits of leveraging these graph-based methodologies include greater situational awareness with automation tools to distill contextualized information at the tactical and operational levels.

Using machine learning algorithms, artificial intelligence, knowledge graphs, and graphical identification techniques, the key results are encouraging: if we used pre-trained datasets applied to previously weighted neural network algorithms, the first iteration or epoch on a machine learning algorithm returns a 20.2% accuracy. Subsequent epochs show more significant accuracy increases to 94.9% after five epochs or more. Although the results show high accuracy, large datasets take a significant amount of time to run the deep convolution neural network algorithms, sometimes taking multiple hours to complete. If the timing of the required information is critical, dedicated high-powered computing will be required. Real-life testing may be needed to assess the actual efficacy of the methodologies described in this report. Other methods can also be applied in future research, such as base-band modulated in-phase and quadrature components of targets instead of visual representations, which, when applied to knowledge graphs, might increase the efficiency and accuracy of combat object identification.

Background

How do synergy and self-organization with neural networks impact complex netted-sensor efficiency and combat identification? Can we form weak and strong ties through the dynamic formation of information to support new knowledge? Graph machine learning and modeling techniques (probabilistic graphical models, neural networks, and clustering algorithms) can be applied to a knowledge graph of radio frequency (RF) signal data. Consequently, the main purpose of this study is to see if self-organization and complex adaptive systems will result in emergent properties for better signal detection, identification, and analysis of potential targets for combat

identification. The research hypotheses must account for the critical factors such as confidence, uncertainty, and accuracy of the new knowledge created. Methods for collecting data (e.g., time and location attributes, the strength of relationships, precision and recall of the knowledge graph) will be scrutinized. These metrics will be compared to the ground truth target of interest calibration signal to prove the enhancement made in combat identification through this research study.

There is significant relevance to the current research concerning Department of Defense operations. Appropriate and feasible data-centric approaches will be considered with proper fleet experiments that enable JADC2 warfighting concepts such as "Sense, Make Sense, and Act." Tactical Situational Awareness (TACSIT) considerations, Expeditionary Advanced Base Operations (EABO), Distributed Maritime Operations (DMO), and contested environments are all factors in providing the knowledge required for engagement decisions. It is essential to identify that the appropriate data required is provided in the right place at the right time to achieve decision superiority. This study will focus on joint combat identification using available sensors in theater, reference emitters, and the Multi-Int Orchestration Solution (MIOS) architecture. The JADC2 concept, data constructs, and information-sharing formats will be the primary informing program for alignment. In the U.S. military context, graph-based technologies enable a new way of approaching the problems of high-confidence over-the-horizon estimations to determine risk to mission and risk to force figures of merit.

Regarding methodology, the current research utilized known pictures of aircraft as a training dataset while machine learning algorithms were applied. Our tests used 14,806 validated images divided into bite-sized chunks called classes. The research illustrates the results of our tests of deep neural networks as they pertain to combat identification applications. The test used Python code that loads various analytical libraries such as NumPy, Pandas, Seaborn, Efficient Net, and Tensor Flow Keras. A large dataset of 14,806 pictures is loaded, and the dataset can be randomly segmented or split into a training set and a testing set. The two sets are further divided into smaller chunks of data for processing, for example, in batches of 40 figures. The accuracy and information losses are measured during training and validation. The training process is replicated and iterated multiple times over 20 neuron layers. Each iteration's or epoch's results are fed back into the next round of iterations. Specifically, the weights are saved and used in the next iteration. Predictions are then run using the best epoch result on a test dataset.

Findings and Conclusions

Starting with the assumption that complex sensor networks are an enormously large and expensive set of systems, software, sensors, and emitters that generate copious amounts of data, which, if analyzed quickly and efficiently, provides battlespace awareness and combat identification. The implications for an efficient solution that provides actionable information and timely detection of unknown signals are huge, as current methods are inadequate to maintain fluid and efficient situational awareness at a tactical level. This research looked at various algorithms using machine learning and knowledge graphs to evaluate if both the efficiency and accuracy can be increased. The results indicate that target identification is possible using graphic files, but

multiple epoch iterations are necessary to obtain higher accuracy levels. For example, running a visual dataset returns a 5.6% accuracy if a standard unweighted fast algorithm is applied. However, with a previously weighted neural network to kick-start the process, the first epoch returns a 20.2% accuracy. Using these initial weights, subsequent epochs show a more significant accuracy increase to 94.9% after just five epochs. The main problem of running large datasets is the time it takes for the deep convolution neural network to run. A moderate dataset of only 14,806 pictures on 5 epochs can take upwards of 3½ hours. Therefore, if the timing is critical, dedicated high-powered computing will be required. In addition, the limitation of deep neural networks, other than runtime, is the need for multiple iterations where any low-level accuracy and precision class types will need to be replaced with better pictures. It is recommended that the next step might be to perform real-life testing to ascertain the actual efficacy of the methodologies described in this report.

Recommendations for Further Research

Based on the research performed, the recommended next steps include using real-life continuous data streams of both graphical file formats as well as alternate types of identification data such as the use of In-phase and Quadrature (I&Q) data where the data has Pre-Detection and Pre-demodulation information preassigned. In communication systems, most signals are base-band modulated into an In-Phase (I) and its Quadrature (Q) subcomponents. These are then transmitted via the Radio Frequency (RF) spectrum. On the other end, receivers demodulate the I&Q subcomponents of the signal and using pre-trained data on previously identified combat identification, the speed, accuracy, and efficiency of detection might be further increased. Knowledge graphs and graphical distance error measures can be employed to quickly determine, with some levels of probability, if the target is a friend or foe.

I. INTRODUCTION

Sensors and capabilities have become increasingly complex as they are deployed in our world. Space, stratospheric balloons, buoys, and hilltops have all become ripe environments for transmitting or receiving signals. Sensors have proliferated into all domains from the seabed to space. Making productive use of them requires sophisticated knowledge of the operating environment, potentially exploitable behaviors of the targets, physical and technical limitations impacting performance, and the available communications links and data processing capacities to move and make use of the data. The application of knowledge graph techniques holds excellent potential for managing this complex tapestry of capabilities and extracting meaningful information from the vast data pools.

Knowledge graph construction and entity extraction automation are emerging, and this study will support the sponsor's combat identification methods. Understanding how machines can leverage graph theory and data sources in the dynamic formation of new knowledge entity creation is paramount. A complex network of national and tactical sensors serves as the data source for experimentation with knowledge graph creation.

A mesh network of national, tactical, and commercial sensors is being deployed in theater and along training ranges worldwide. This research will consider the optimal use of sharing sensor information to produce quality geolocation results and leverage automatic knowledge creation to support combat identification. Space, sea, land, and stratospheric sensor combinations from disparate services and partner nations can increase cooperative interoperability to improve geolocations in dense and sparsely populated radio frequency (RF) spectrum environments.

In this research, we propose utilizing multiple approaches focusing on graph theory methodologies. Methods such as machine learning, probabilistic modeling, and clustering techniques serve to formulate an experiment and execution to discover, hypothesize, and demonstrate through simulations a self-learning knowledge graph environment to explore the effectiveness of automatic knowledge creation for combat identification. These methodologies are an emerging enabler for rapid signal associations required for targeting and situational awareness. Combat identification procedures and planning can benefit from leveraging the power of graph machine learning and probabilistic modeling for automatic entity creation for unstructured and structured data sets.

Specifically, leveraging the OUSDI-proposed Common Data Fabric (CDF) construct data-centric approach in tandem with the Joint national and tactical netted-sensor architectures will be the primary data sources leveraged for this research. This study focuses on revealing the new methodologies based upon graph theory and the benefits to combat identification for targeteers.

Follow-on efforts may include automatic geolocation for signals of interest in contested environments supporting Joint All-Domain Command and Control (JADC2) and producing consumable information into the CDF. As requested, this information can inform automated tipping and queuing systems to aid planning for additional sensor coverage and the automated tasking of national and tactical assets.

The research methodology will adapt to include feedback from subject matter experts and access to any data sources that may become available. These methodologies will also leverage high-side cloud computing environments, data sources, and expertise available at NPS (Naval Postgraduate School) and the NIWC PAC (Naval Information Warfare Center Pacific) NITE (Net-centric Interoperability Test & Evaluation) Lab.

Research Objectives and Questions

How do synergy and self-organization in conjunction with neural networks impact complex netted-sensor efficiency and combat identification? Can we form weak and strong ties through the dynamic formation of information in support of new knowledge?

Graph machine learning and modeling techniques (probabilistic graphical models, neural networks, and clustering algorithms) can be applied to a knowledge graph of RF signal data. Consequently, self-organization and complex adaptive systems will result in emergent properties for better signal detection, identification, and analysis of potential targets for combat identification.

This hypothesis must account for the critical factors such as confidence, uncertainty, and accuracy of the new knowledge created. Methods regarding how the data was collected (e.g., time and location attributes, the strength of relationships, precision and recall of the knowledge graph) will be scrutinized. These metrics will be compared to the ground truth target of interest calibration signal to prove the enhancement made in combat identification through this research study.

Research Methodology

Appropriate and feasible data-centric approaches will be considered with proper fleet experiments that enable JADC2 warfighting concepts such as “Sense, Make Sense, and Act.” Tactical Situational Awareness (TACSIT) considerations, Expeditionary Advanced Base Operations (EABO), Distributed Maritime Operations (DMO), and contested environments are all factors in providing the knowledge required for engagement decisions. Many ongoing experimentation efforts engaging in collaboration with CPF, C3F, USSF, and I-MEF (Commander Pacific Fleet, Commander Third Fleet, United States Space Force, 1st Marine Expeditionary Force) for training multi-domains of information warfare and providing independent verification and coordinating asset resource allocation.

It is essential to identify that the appropriate data required is provided in the right place at the right time to achieve decision superiority. This study will focus on Joint combat identification using available sensors in theater, reference emitters, and the Multi-Int Orchestration Solution (MIOS) architecture. The JADC2 concept, CDF data constructs, and information-sharing formats will be the primary informing program for alignment. Further, the study will highlight instances where the appropriate sensors, data, tools, and authority requirements were not met to remedy these deficiencies in future experimentation.

In the U.S. military context, graph-based technologies enable a new way of approaching the problems of high-confidence over-the-horizon estimations to determine risk to mission and risk to force figures of merit.

II. LITERATURE REVIEW

Graph Theory

Flovik (2020) provides a good history of graph theory and attributes the origin of graph theory to Leonhard Euler, a Swiss mathematician, who published the first treatise on the subject in 1736, "The Seven Bridges of Konigsberg." After that introduction, graph theory was applied to various mathematical puzzles, such as diagram tracing the four-color map problem (Wilson, 2013). Fleury gave a systematic method for "tracing an eulerian graph," while Tarry "showed how to escape from a maze."

The approach has remained dormant for decades (von Bell, 2015). Not until 1936, when Dénes Kőnig wrote "*Theorie der endlichen und unendlichen Graphen* (*Theory of finite and infinite graphs*)," was the method resurrected where this work was considered the first academic textbook relating to graph theory. Interest in graph theory began to grow shortly after (O'Connor and Robertson, 2014). Later, Frank Harary published *Graph Theory* (1969), which made the concept more accessible beyond the interests of mathematicians. The "mid-20th century saw algorithmic solutions to such problems as the minimum connector problem, the shortest and longest path problems, the Postman Problem, and various other issues arising in operational research" (Flovik, 2020).

Artificial Intelligence and Advanced Analytical Methods

Zhang (2021) found that artificial intelligence machine learning applications on graphs have been studied extensively in academia and industry. However, while the literature on graph machine learning has increased with vast volumes of emerging methods and techniques, it has become difficult to design so-called optimal machine learning algorithms for various graph related tasks. Graph machine learning applications such as Bayes' network, hyper-parameter optimization, and neural network architecture search have been studied to incorporate these elements into graph theory.

In an article by Johnson (2020), predictive analytics, one of the valuable outcomes of graph theory, was highlighted at the forefront of offering potentially game-changing capabilities for the U.S. Navy's tactical decision superiority. Tactical operations could take advantage of actionable intelligence by taking a significant leap with real-time and automated predictive analytics with predictions of second-order and third-order effects of different courses of action. Future capabilities would complement current developments in the use of artificial intelligence and data analytics to significantly help in improving battlespace knowledge by providing automated battle-management aids to the tactical warfighter. These include strategic options in tactical courses of action (COA) where predictive analytics could help forecast how adversaries might respond to various COA option. The predictive analytics could continue to wargame blue force vs. red force COA and proportionate responses, providing additional predictions of second and third-order

effects. These predictive modeling approaches can offer tactical warfighters strategic perspectives in making tactical COA decisions.

Johnson (2020) continues by identifying the use of Bayesian Networks (BN) and causal graph models as a potential solution to modeling probabilities of output events given input stimuli and observations. These models can be applied to build quantitative representations of more complex dynamic and interrelated situations. Dynamic BN models and BN-learning algorithms with artificial intelligence can learn from existing data to create adaptive capabilities that predict outcomes in a changing environment. These Bayesian approaches, such as Bayes' network, hypothetical belief network, decision network, Bayes' updating model, or probability acyclic graphical model, are quantitative statistical models representing a set of variables and their sequential conditional dependencies in the form of graphs.

For instance, various algorithms that calculate conditional probabilities can instantiate a Bayesian classifier. Algorithms can also be run in a Bayesian belief network. For example, a graphical model can represent a group of variables and their respective probabilistic independence. Thus, a Bayesian network can be created to potentially represent the probabilistic relationships among various threat classes (friend or foe) and predictive cues. Given a set of cues, the network can compute the probability of the target being a friend or a foe.

Vu and Thai (2020) applied Probabilistic Graphical Model (PGM) with Graph Neural Networks (GNNs), and their theoretical analysis showed that the PGM includes the Markov blanket of the target prediction, including all its statistical information. Their results contain the same set of independence statements in the perfect map using both synthetic and real-world datasets. It concluded that PGM performs better than existing methods in many benchmark tasks.

Larrañaga and Moral (2011) reviewed the role of probabilistic graphical models in artificial intelligence. Some of the techniques were applied in problem solving (abduction, classification, and decision-making) and showcased potential futures of relevant applications in forensic reasoning, genomics, and graphical models as a general optimization method.

III. GRAPH THEORY

A. Graph Theory

What is graph theory? A collection of data as vertices and edges that can be visualized using connected nodes in a graphical format. Large datasets are to be charted by their interconnections or interrelationships. Let's start with "graph" because when discussing graph theory, it doesn't mean what first comes to mind. In this context, it's not about pie charts or scatter plots. In this context, graphs are diagrams of points (nodes or vertices) and lines (links or edges) depicted like connect-the-dots structures (networks). Graph theory is "the study of the relationship between edges and vertices" (Javatpoint, n.d.). According to Flovik (2020), "given a set of nodes and connections, which can abstract anything from city layouts to computer data, graph theory provides a helpful tool to quantify and simplify the multiple moving parts of dynamic systems. Studying graphs through a framework provides answers to many networking, optimization, matching and operational problems."

A cursory search online will tell you that a graph is a pictorial and mathematical representation of objects where links connect some pairs of objects. Points represent the interconnected objects termed vertices or nodes, and the links that connect the vertices are called edges or arcs or lines.

Graph theory is a sub-specialization of both mathematics and computer science that handles graphs and diagrams containing lines and points that pictorially represent mathematical relationships. In short, graph theory studies the relationship between edges and vertices.

Graph theory can be applied in multiple areas, including but not limited to:

- Pattern recognition: PGMs can be used for image and speech recognition, natural language processing, and other forms of pattern recognition.
- Decision making: PGMs can be used to model decision-making problems, including decision trees and Markov decision processes.
- Natural language processing: PGMs can be used to model probabilistic relationships between words in a text corpus, allowing for tasks such as text classification and topic modeling.
- Healthcare: PGMs can be used to model complex relationships between patient attributes, medical records, and disease outcomes.
- Finance: PGMs can be used to model financial systems, such as stock market trends, and make predictions about future events.
- Artificial intelligence: PGMs can be used to develop intelligent agents that make decisions based on probabilistic models of the environment.

- Computer science: the study of algorithms, defining the flow of computation, representing networks of communication, representing data organization, and finding the shortest path in a road or a network.
- Gross (2003) additionally mentions the following applications:
    - Electrical engineering: designing circuit connections.
    - Linguistics: primarily used for parsing of language trees or the grammatical structure of a language tree.
    - Physics and chemistry: to study molecules.
    - Computer network: the relationships among computers connected within the network can sometimes follow the principles of graph theory, used in network security, vertex coloring algorithm (four colors for a map).
    - Social sciences: rumor spreading, measure prestige, acquaintanceship and friendship, influence, collaboration.
    - Biology: biological, transcriptional, metabolic, protein-protein interactions networks, and drug-target relationships.
    - Mathematics: operational research.
    - General: routing between cities, hierarchical ordered information (e.g., family tree).

Figures 1 to 3 illustrate some basic examples of graph theory (created with Neo4J), where publicly available data are applied to generate the relationships among various variables and their properties.

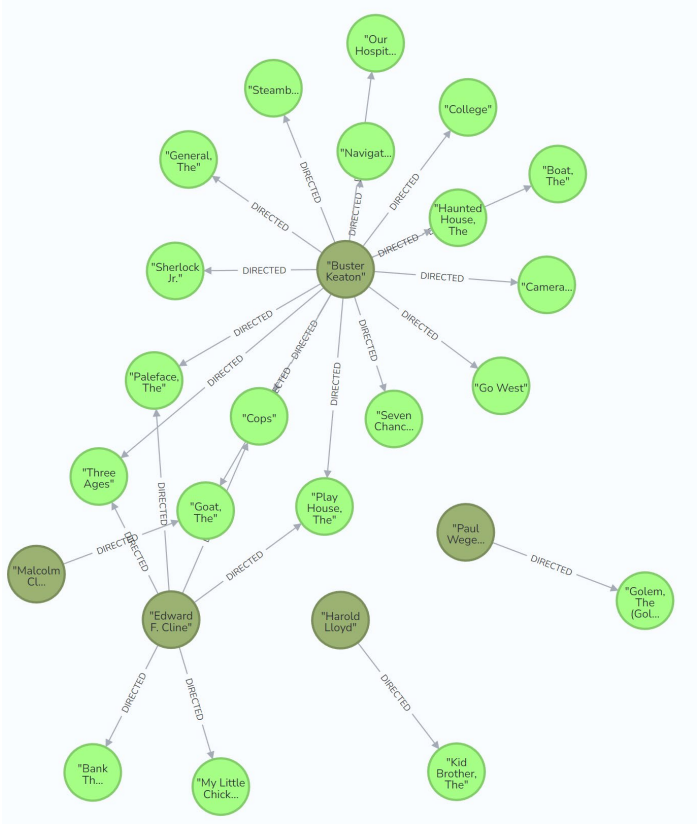


Figure 1. Knowledge Graph of Movies and Their Directors

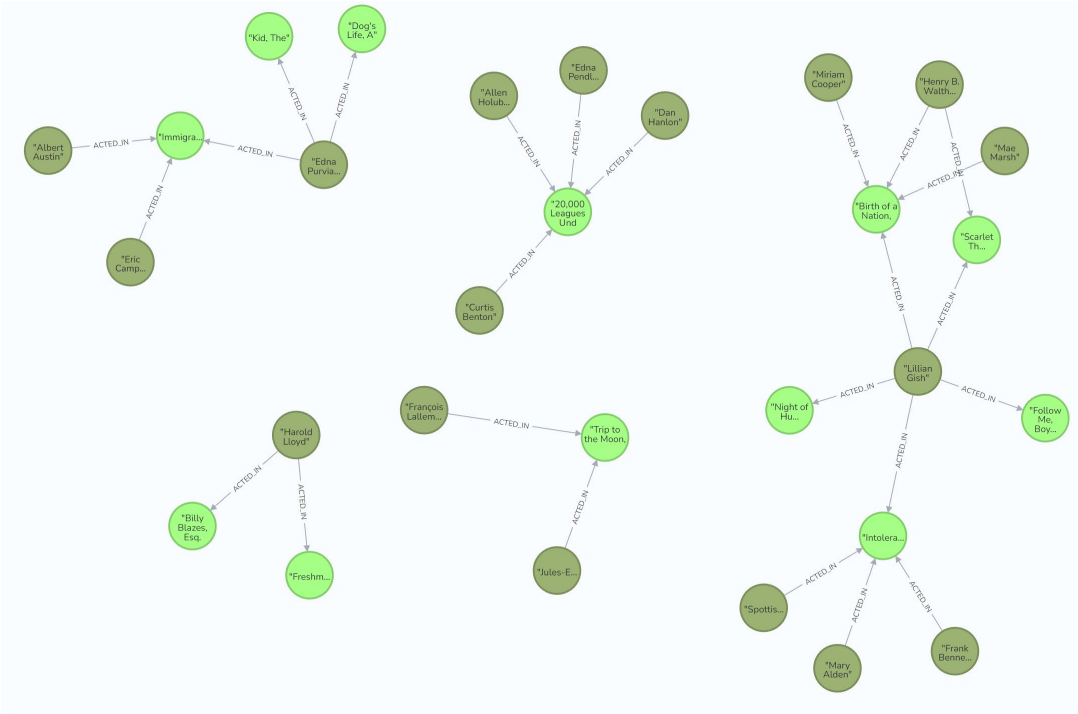


Figure 2. Standalone Knowledge Graphs of Actors and Movies

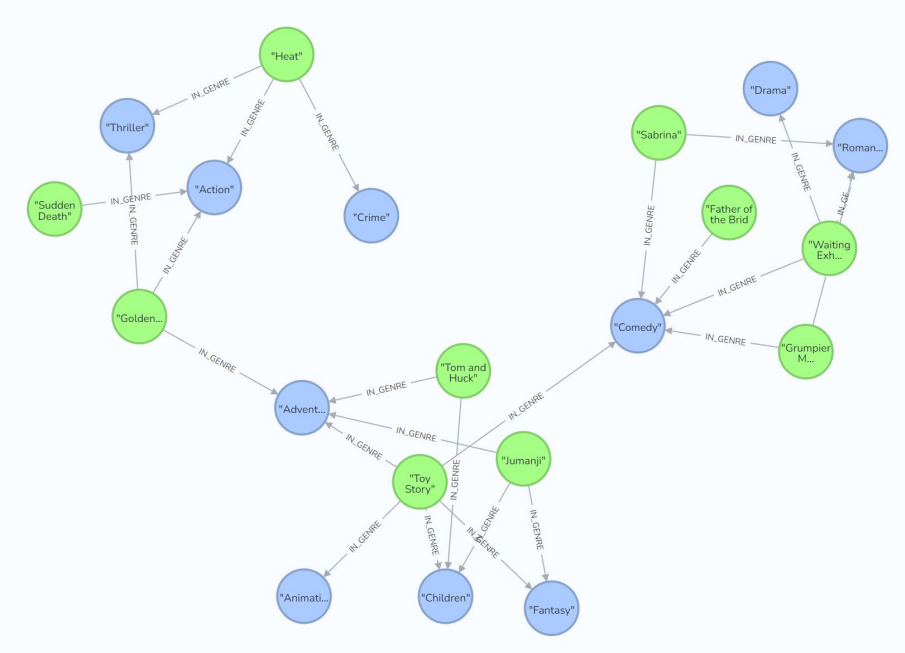


Figure 3. Example of Knowledge Graphs of Genres and Movies

B. Graph Machine Learning

Graph Machine Learning (GML) is a sub-specialization of Machine Learning (ML) that deals with the analysis and processing of graph-structured data. In graph-structured data, entities are oftentimes represented as nodes in a graph and their relationships are represented as edges. GML algorithms are designed to perform tasks such as node classification, link prediction, graph clustering, and graph generation. These algorithms often draw on techniques from graph theory, network science, and ML to develop models that can learn from graph-structured data and make predictions.

Examples of graph-structured data include social networks, protein-protein interaction networks in biology, and knowledge graphs in the Semantic Web. GML techniques can uncover hidden patterns and relationships in complex data structures, enabling new insights and applications in areas such as recommendation systems, fraud detection, and life sciences research.

C. Combat ID

Combat identification (Combat ID) refers to the process of accurately identifying friendly forces, enemy forces, and neutral entities during military operations. This helps to reduce the risk

of fratricide or friendly fire incidents, where friendly forces inadvertently attack each other. Combat ID can be accomplished through various means, including visual identification, electronic identification, and the use of identification friend or foe (IFF) systems.

D. Target ID

Target identification refers to the process of determining the nature, characteristics, and potential threat posed by a specific object or entity. This crucial step in military operations is used to prioritize and engage targets in support of mission objectives while minimizing unintended consequences. Target identification may involve the use of various sensors, such as radar, thermal imaging, and visible-light cameras, as well as human intelligence and analysis. The goal of target identification is to determine the most appropriate course of action, whether it be to engage the target, avoid it, or gather additional information.

E. ATR

Automatic Target Recognition (ATR) refers to the process of using computer algorithms to identify and classify objects or targets in real time without the need for human intervention. ATR is commonly used in military applications, where it can help to quickly and accurately identify potential threats, such as enemy vehicles, aircraft, or weapons systems. The technology behind ATR involves the use of sensors, such as radar or electro-optical cameras, to gather data on the target. This data is then processed and analyzed by the ATR system, which compares the data to a database of known objects and uses pattern recognition algorithms to determine the identity of the target. The goal of ATR is to provide a fast and reliable means of target identification, reducing the risk of friendly fire incidents and increasing the efficiency of military operations.

F. GML Uses for ATR

Graph machine learning (GML) can be used for automatic target recognition (ATR). GML techniques can be applied to the analysis of graph-structured data, which can be generated by various sensors, such as radar or electro-optical cameras, to identify targets in real time. The graph structure can capture the relationships and interactions between different features in the data, providing a more comprehensive representation of the target. By incorporating GML techniques, ATR systems can improve the accuracy and efficiency of target recognition, especially for complex targets or cluttered environments.

For example, GML algorithms can be used to model the relationships between different features in the data, such as the size, shape, and motion of a target, to more accurately determine its identity. This can help to reduce false alarms and improve the robustness of ATR systems, especially in challenging operational environments.

G. Differences Between Combat ID & Target ID

Combat ID is focused on positively identifying friendly forces, while Target ID is focused on determining the nature and threat of potential targets. It is noteworthy that ATR techniques are applicable to both. Combat ID refers to the identification of friendly forces in combat situations, typically achieved through visual or electronic means, in order to avoid friendly fire incidents. Target ID refers to the identification of potential targets, either friend or foe, in military operations. The goal of target ID is to analyze the threat of a potential target in order to make informed decisions about engaging it.

Automatic target recognition (ATR) can be applied to various forms of data, not just imagery. The technology can be applied to radar signals, acoustic signals, or even data from infrared, lidar or radio frequency sensors. ATR algorithms process the data from these sources and attempt to identify specific targets based on their unique characteristics. The specific type of data used for ATR will depend on the intended application and the sensors available.

H. Radar Signals and Graph-Based Methodologies

Several graph-based methodologies have been applied to interpret radar signals, including:

1. Graph Signal Processing (GSP): GSP is a framework that can be used to analyze signals on graphs, including radar signals. The approach involves representing the radar signal as a graph signal and processing it using graph-based filters and transforms.
2. Graph Neural Networks (GNNs): GNNs are a type of deep learning architecture that can be used to model the relationship between radar signals and their associated targets. In this context, the graph structure represents the relationships between different radar signals. The GNN is then trained to identify targets based on these relationships.
3. Spectral Graph Theory: Spectral graph theory is a branch of mathematics that uses graph theory and linear algebra to analyze signals on graphs. In the context of radar signals, spectral graph theory can be used to analyze the frequency and time components of the radar signal and to identify specific targets based on their unique signatures in the RF spectrum.
4. Graph Fourier Transform (GFT): The GFT is a graph-based counterpart to the classical Fourier transform that can be used to analyze signals on graphs, including radar signals. By decomposing the radar signal into its constituent frequency components, the GFT can be used to identify specific targets based on their unique frequency content.

I. Challenges and Problems with GML Methods

1. Scalability: Large-scale graph data can be computationally intensive to process, making it difficult to scale graph machine learning algorithms to handle large datasets.

2. Sparsity: Graphs are often sparse, meaning that many nodes have few connections to other nodes, which can make it difficult to accurately model complex relationships.
3. Node Feature Representation: In graph machine learning, nodes are typically represented as high-dimensional feature vectors. However, it can be challenging to effectively represent nodes in this way, especially when dealing with graph data that has a large number of nodes and complex relationships between nodes.
4. Label Propagation: Graph machine learning algorithms often rely on label propagation, which involves propagating labels from labeled nodes to unlabeled nodes in the graph. This process is challenging when dealing with graph data that has a large number of nodes and complex relationships between nodes.
5. Overfitting: Overfitting is a common issue in machine learning, and it can be particularly challenging in graph machine learning due to the complex relationships in graph data.
6. Transferability: Graph machine learning algorithms can be difficult to transfer from one problem or application to another, as the specific properties of the graph data can significantly impact the performance of the algorithms.

Despite challenges, graph machine learning is a growing field that combines graph theory and machine learning to analyze data represented as knowledge graphs. Graph methodologies can aid in predicting missing links, detecting meaningful structures, community detection, or activity tracking of targets of interest.

J. Graph Signal Processing (GSP)

Graph Signal Processing (GSP) is a framework for processing signals defined on graph domains. In GSP, a graph is used to represent the relationships between signals and their neighbors, and graph-based techniques are used to analyze and manipulate the signals.

GSP provides a mathematical framework for representing signals on graphs, and it has been applied to a wide range of applications, including image and video processing, network analysis, and recommendation systems. GSP algorithms can be used to analyze the frequency content of signals, filter and denoise signals, and perform dimensionality reduction and clustering.

GSP is becoming increasingly popular due to its ability to handle signals with non-Euclidean structures, such as signals that are defined on graphs or networks. By leveraging graph-based techniques, GSP enables the analysis of complex signals that are difficult to handle using traditional signal processing techniques.

K. Probabilistic Graphical Modeling

Probabilistic graphical modeling is a branch of machine learning that uses graphical models to represent probabilistic relationships between variables in a given problem. These models can be

used to make predictions, perform decision-making, and analyze complex systems with uncertainty.

Probabilistic graphical models are particularly useful in the following areas:

1. Pattern recognition: PGMs can be used for image and speech recognition, natural language processing, and other forms of pattern recognition.
2. Decision making: PGMs can be used to model decision-making problems, including decision trees and Markov decision processes.
3. Natural language processing: PGMs can be used to model probabilistic relationships between words in a text corpus, allowing for tasks such as text classification and topic modeling.
4. Healthcare: PGMs can be used to model complex relationships between patient attributes, medical records, and disease outcomes.
5. Finance: PGMs can be used to model financial systems, such as stock market trends, and make predictions about future events.
6. Artificial intelligence: PGMs can be used to develop intelligent agents that make decisions based on probabilistic models of the environment.

In summary, probabilistic graphical modeling is a powerful tool for modeling and analyzing complex systems with uncertainty. It is used in a wide range of applications, including pattern recognition, decision making, natural language processing, healthcare, finance, and artificial intelligence.

L. PGMs in AI

PGMs are powerful for modeling and reasoning about uncertain systems. In artificial intelligence, probabilistic graphical models (PGMs) can be used to develop intelligent agents that make decisions based on probabilistic models of an environment. An intelligent agent is a system that acts autonomously in an environment, perceiving its surroundings and making decisions based on its observations. PGMs provide a framework for modeling the environment in which an agent operates and for representing the uncertain relationships between the various components of the environment.

For example, a PGM can be used to model the probabilistic relationships between an agent's actions and its perceptions. Information can then be used by the agent to make decisions that are informed by the uncertainty in its environment to make inferences. The agent can use its probabilistic model of the environment to reason about the possible outcomes of different actions and choose the action that is most likely to achieve its goals.

PGMs can also be used to model the evolution of an environment over time, allowing an agent to make predictions about future events based on its current state and past experiences. This is useful in domains such as combat identification, where an agent must make decisions based on

incomplete or uncertain information about its surroundings. There is a wide range of applications in AI to include the development of intelligent agents that operate in uncertain environments, such as combat identification. This method can be employed within uncertain environments for combat identification.

M. I&Q definition

In-phase and Quadrature (I&Q) is just another term used for Pre-Detection or Pre-Demodulation (Pre-D). In communication systems, a signal is base-band modulated in its in-phase (I) as well as its quadrature (Q) components, transmitted to the Radio Frequency (RF) spectrum. Specialized receivers demodulate the I&Q components of the intercepted signal.

IV. AI ANALYTICAL APPROACHES

AI/ML Data Reduction and Classification and Logistic Predictive Modeling

When the dependent variable contains data that are constrained in scope and range, such as binary replies (0 or 1 for failures/successes), truncated, ordered, or censored data, the classification technique we'll apply is appropriate. For instance, using maximum likelihood estimation (MLE), we can estimate the likelihood of defaulting on mortgage payments given a collection of independent factors (such as age, income, and education level of credit card or mortgage loan holders). Binary makes up the response or dependent variable Y. In other words, it can only have two possible outcomes, which we denote as 1 and 0. For instance, Y may represent the presence or absence of a certain condition, defaulted/not defaulted on previous loans, success/failure of some device, answer yes/no on a survey, etc. We also have a vector of independent variable regressors, X, which are assumed to have an impact on the outcome Y. Because the regression errors are heteroskedastic and non-normal, a conventional ordinary least squares regression approach is inappropriate and the resulting estimated probability estimates will produce absurd values of above or below 1. When the dependent variables are constrained, MLE analysis solves these issues by utilizing an iterative optimization procedure to maximize a log-likelihood function. A similar approach was created in a previous research (Mun 2022b) and its theoretical foundations are available in more detail in Mun (2022a).

By fitting data to a logistic curve, a logit or logistic regression is used to estimate the likelihood that an event will occur. For binomial regression, a generalized linear model is employed, and like many other types of regression analysis, it employs a number of predictor variables that may be either numerical or categorical. The dependent variable is modeled using MLE performed in a binary multivariate logistic analysis to ascertain the expected success probability of belonging to a particular group. As logarithmic odds ratios, the computed coefficients for the Logit model cannot be taken literally as probabilities. First, a short calculation is necessary, and the process is straightforward.

The log odds ratios are the coefficients β_i in the Logit model, which is defined as Estimated $Y = LN[P_i/(1-P_i)]$ or, alternatively, $P_i = EXP(Estimated\ Y)/(1+EXP(Estimated\ Y))$. In order to get the odds ratio $P_i/(1-P_i)$, we take the antilog or $EXP(\beta_i)$. This implies that the log odds ratio rises by this amount for every unit increase in β_i. Lastly, the probability change rate is given by $dP/dX = \beta_i P_i(1-P_i)$. We can easily compute the Estimated Y value using the MLE coefficients and convert it into the inverse antilog of the odds ratio a discussed above in order to estimate the likelihood that a particular group will succeed (for example, predicting whether a program will experience problems and ultimately fail given a specific combination of lifecycle cost, ROI, FTE requirements, length of time, strategic value, etc.). Next, we can use statistically significant factors to categorize the programs into groups with a high likelihood of being accepted or rejected using a Gaussian Support Vector Machine (SVM).

FIRST STEP

Model Inputs:
VAR1
VAR2; VAR3; VAR4; VAR5; VAR6; VAR7; VAR8; VAR9
Status (D)
Monthly FTE, Complexity Level, Strategic Value, Value to Command, Length in Months, Program Cost, Overrun Ratio, Annual Cost Savings

Generalized Linear Model (Logit with Binary Outcomes)

	Coefficient	Std. Error	Wald Test	P-value	Exp(B)	Lower	Upper
Intercept	-1.634198	0.754434	4.692098	0.030302	0.195109	0.000000	0.000000
VAR1	0.028625	0.020496	1.950585	0.162524	1.029039	0.988520	1.071218
VAR2	0.076812	0.144371	0.283071	0.594695	1.079839	0.813711	1.433004
VAR3	-0.262500	0.040630	41.7411	**0.000000**	0.769127	0.710254	0.832879
VAR4	-0.096195	0.027419	12.3083	**0.000451**	0.908287	0.860764	0.958434
VAR5	0.000823	0.012687	0.004210	0.948266	1.000824	0.976243	1.026022
VAR6	0.074324	0.039911	3.467833	**0.062573**	1.077155	0.996106	1.164799
VAR7	0.564136	0.134590	17.5689	**0.000028**	1.757929	1.350325	2.288569
VAR8	0.049994	0.101851	0.240943	0.623526	1.051265	0.861027	1.283535

Log-Likelihood	-199.9830
Restricted Log-Likelihood	-285.4773
McFadden's R-Squared	0.299479
Cox and Snell's R-Squared	0.289636
Nagelkerke's R-Squared	0.425440
Raw Akaike Info. Criterion	417.9659
Raw Bayes Criterion	455.8974

Log-Likelihood	-199.9830
Restricted Log-Likelihood	-285.4773
Chi-Square	170.9886
Degrees of Freedom	8
P-value	0.000000

SECOND STEP

Model Inputs:
VAR1
VAR4; VAR5; VAR7; VAR8
Status (D)
Strategic Value, Value to Command, Program Cost, Overrun Ratio

Generalized Linear Model (Logit with Binary Outcomes)

	Coefficient	Std. Error	Wald Test	P-value	Exp(B)	Lower	Upper
Intercept	-0.781188	0.305330	6.545958	0.010512	0.457862	0.000000	0.000000
VAR1	-0.239706	0.033215	52.0818	**0.000000**	0.786859	0.737266	0.839788
VAR2	-0.074519	0.023632	9.942889	**0.001615**	0.928190	0.886178	0.972194
VAR3	0.082202	0.022767	13.0359	**0.000306**	1.085675	1.038294	1.135218
VAR4	0.588673	0.108123	29.6424	**0.000000**	1.801597	1.457549	2.226855

Log-Likelihood	-201.7171
Restricted Log-Likelihood	-285.4773
McFadden's R-Squared	0.293404
Cox and Snell's R-Squared	0.284691
Nagelkerke R-Squared	**0.418177**
Raw Akaike Info. Criterion	413.4342
Raw Bayes Criterion	434.5072

Log-Likelihood	-201.7171
Restricted Log-Likelihood	-285.4773
Chi-Square	167.5204
Degrees of Freedom	4
P-value	0.000000

THIRD STEP

Model Inputs:
Status (D)
Strategic Value, Value to Command, Program Cost, Overrun Ratio
Sigma, Lambda, Omega, Calibration Level: 1.00, 1.00, 0.40, 1.00

AI Machine Learning: Classification with Gaussian SVM (Supervised)

Relax: 8.218332

Accuracy	68.20%	67.40%	68.20%	**69.40%**	67.80%	67.80%	66.60%	65.00%	63.40%	62.20%
Omega	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00

Forecast	Group
1.118101	1.00
0.971805	0.00
0.971803	0.00
0.971803	0.00
. . .	. .
. . .	. .
0.974960	1.00
0.975455	1.00
0.972106	1.00
0.971933	1.00
0.971805	0.00
0.971804	0.00
0.971985	1.00
0.971828	1.00

Stochastic Risk Simulation and Probabilistic Analysis

Stochastic distributional fitting, or how well the gathered historical data fit known probability distributions, is another suggested method. The input parameters for the variable can be taken from these fitted distributions (for example, a Fréchet or Weibull distribution with shape and scale parameters of 0.5 and 1.2). A case where historical program costs were fitted to ascertain their distributional characteristics is shown in Figure 4. These can be used as inputs into a Monte Carlo simulation model to anticipate and predict a new program's odds of success using the fitted distribution. The theoretical foundations of these approaches are available in more detail in Mun (2022a).

Status (D)	Monthly FTE	Complexity Level	Strategic Value	Value to Command	Length in Months	Program Cost	Overrun Ratio	Annual Cost Savings
1	41	3	17	12	176	9.3	11.36	5.01
0	27	1	11	6	31	17.3	1.36	4
0	40	1	15	14	55	5.5	0.86	2.17
0	41	1	15	14	120	2.9	2.66	0.82
1	24	2	2	1	28	17.3	1.79	3.06
0	41	2	5	5	25	10.2	0.39	2.16
0	39	1	21	9	67	30.6	3.83	16.67
0	43	1	12	11	38	3.6	0.13	1.24
1	2						1.36	3.28
0	3						2.78	2.15
0	2						0.18	0.09
0	2						0.25	0.94
0	5						3.93	2.47
0	3						1.72	3.01
0	4						3.7	5.4
1	3						0.82	3.4
1	3						2.92	3.81
0	4						1.18	4.29
0	3						0.56	2.91
0	4						0.1	0.34
0	3						1.15	0.51
0	4						0.59	1.82
0	2						0.43	2.17
0	2						0.4	2.24
1	2						0.24	2.64
0	2						2.14	3.49
0	4						0.71	3.92
0	4						0.95	9.74
0	3						3.08	7.59
0	2						0.2	5.05
0	4						0.11	0.32
0	3						1.14	1.17
0	2						0.72	0.84
0	2						1.02	2.87
0	2						0.08	0.94
0	2						6.05	5.65
0	3						1.41	2.13
0	3						3.2	4.58
1	2						1.34	2.77
0	45	1	23	5	50	4.2	0.56	1.54
0	23	1	7	2	31	6.6	0.34	1.71
0	34	1	17	3	59	8	1.81	2.91
0	42	2	7	23	41	4.6	0.94	0.94
0	39	1	19	5	48	13.1	1.93	4.36
1	26	1	1	1	14	7.5	0.3	0.75

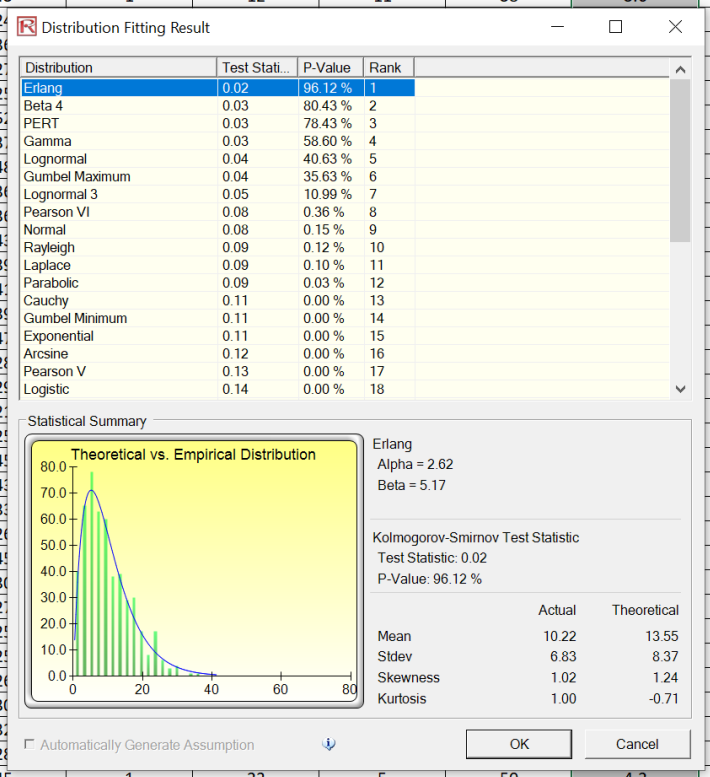


Figure 4. Distributional Fitting

Neural Network Pattern Recognition Prediction Methods

Staring with Box-Jenkins method, forward-looking predictive steps are run:

$$\hat{x}_{t+1} = f(x_t, x_{t-1}, \ldots, x_{t-n})$$

$$\hat{x}_{t+2} = f(x_{t+1}, x_t, \ldots, x_{t-n+1})$$

$$\ldots$$

$$\hat{x}_{t+k} = f(x_{t+k-1}, x_{t+k-2}, \ldots, x_{t-n+k-1})$$

Where x_t is the observation of x at time t. Using a k-step ahead predictive model:

$$x_{t+1} = f_1(x_t, x_{t-1}, \dots, x_{t-n})$$

$$x_{t+2} = f_2(x_t, x_{t-1}, \dots, x_{t-n})$$

$$\dots$$

$$x_{t+k} = f_k(x_t, x_{t-1}, \dots, x_{t-n})$$

Here, we see that f_i values are computed in the neural network paradigm.

Activation Transfer Functions for Neural Networks

Logistic sigmoidal function: $f(x) = (1 + e^{-x})^{-1}$

Hyperbolic tangent function: $f(x) = (e^x - e^{-x})(e^x + e^{-x})^{-1}$

Sine and cosine function: $f(x) = sin(x)\ or\ f(x) = cos(x)$

Linear function: $f(x) = x$

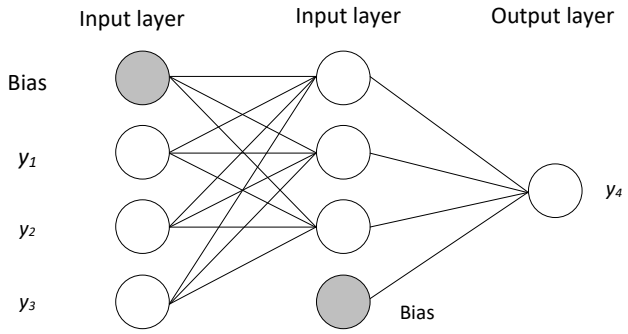


Figure 5. A Multiple Layered Perceptron Neural Network

The neural mapping in Figure 5 assumes that y_4 is the dependent variable, whereas the independent variables are y_1, y_2, y_3 and a constant. The neural network has one input layer, one or multiple hidden layers, and one output layer. In the example, we see 3 inputs in the input layer, a special neuron (colored) for the biases, with 4 neurons within the hidden layer, and 1 final neuron in the output layer.

Error Measurements and Error Correction for Parameter Calibration

Total Variables (Dependent and Independent): v

Mean Absolute Deviation: $MAD = \frac{\sum|e_t|}{n}$

Root Mean Squared Error: $RMSE = \sqrt{\frac{\sum(e_t)^2}{n}}$

Sums of Squared Errors: $SSE = \sum(e_t)^2$

Maximum Log-Likelihood: $MLL = \frac{n}{2} ln(2\pi) - \frac{n}{2} ln \frac{SSE}{2} - SSE \left[\frac{n}{2SSE}\right]$

Akaike Information Criterion: $AIC = \frac{-2MLL}{n} + \frac{2k}{n}$

Bayes Information Criterion (BIC): $BC = AIC + \frac{2(v+2)(k+3)}{n-k-3}$

Pesaran-Timmermann Test: $PT = \frac{p(xf)-p\prime}{\sqrt{v-w}}$ where $v = \frac{p\prime(1-p\prime)}{n}$, $p' = f^+x^+ + (1-f^+)(1-x^+)$, and where $w = \frac{(2f^+-1)^2x^+(1-x^+)}{n} + \frac{(2x^+-1)^2f^+(1-f^+)}{n} + \frac{(4x^+f^+)(1-x^+)(1-f^+)}{n^2}$, $x^+$is the proportion of positives on the data and $f^+$is the proportion of positives on the forecast

Hannan-Quinn Information Loss Criterion: $HC = \frac{-2MLE}{n} + \frac{2kln(ln(n))}{n}$

V. CONVOLUTION DEEP NEURAL NETWORK COMBAT ID

This section delves into the applications of convolution-deep neural networks. Starting with a quick introduction to how deep neural network algorithms work, is followed by some example applications of deep neural networks on combat identification.

Deep Neural Network

Figure 6 illustrates how a Deep Neural Network navigates its learned data to recognize various photos (Adapted from Golstein (2018); Parloff (2016)). AI software may be taught using photos, and relationships between these trained data can then be used in the neural network to classify inputs and ultimately arrive at a result. The Simple Neural Network uses a collection of input data that only passes through one hidden layer in order to determine the output layer. The Deep Learning Neural Network transfers the input data through several layers in order to more accurately identify the output data. In order to classify input data to determine whether the given picture is a dog, the Deep Learning Neural Network goes through basic to more intricate layers of trained data that correspond with dog features to make a 95% confidence classification that the picture is a dog and a 5% possibility that it is a wolf. Similar to this, images and photos of different fighter jets can be used to train the system so that it can be used to carry out combat recognition of fresh images with which it is supplied. A similar approach was created in previous research (Mun 2022b) and its theoretical foundations are available in more detail in Mun (2022a).

Combat Aircraft Identification

Figure *7* and Figure *8* show examples of some combat aircraft as well as civilian aircraft used to train the deep neural network system. In our test, we used 14,806 validated images divided into bite-sized chunks called classes. The next section illustrates the results of our tests of deep neural networks as they pertain to combat identification applications.

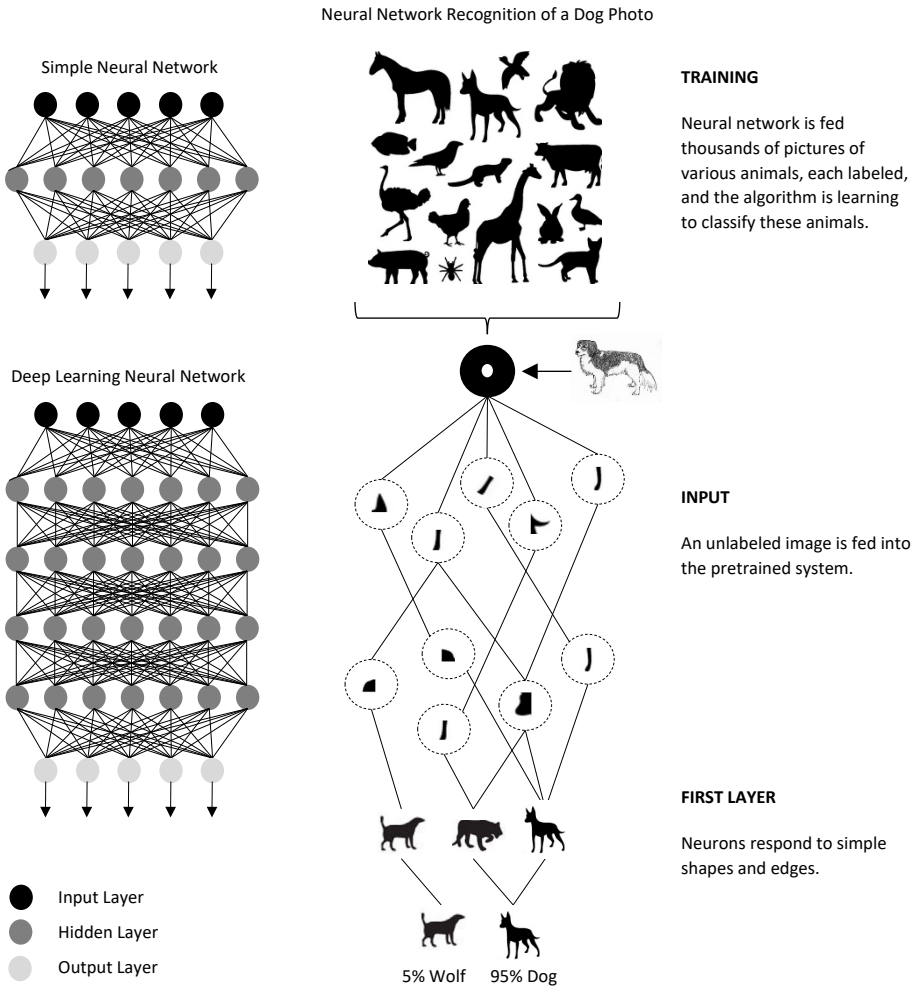


Figure 6. Neural network vs. deep learning network.

```
class_names = train_df.class_names

plt.figure(figsize = (20, 20))
for images, labels in train_df.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
```

Figure 7. Visual Identification Training Set

Display Image Sample

```
In [12]: show_images(train_gen)
```

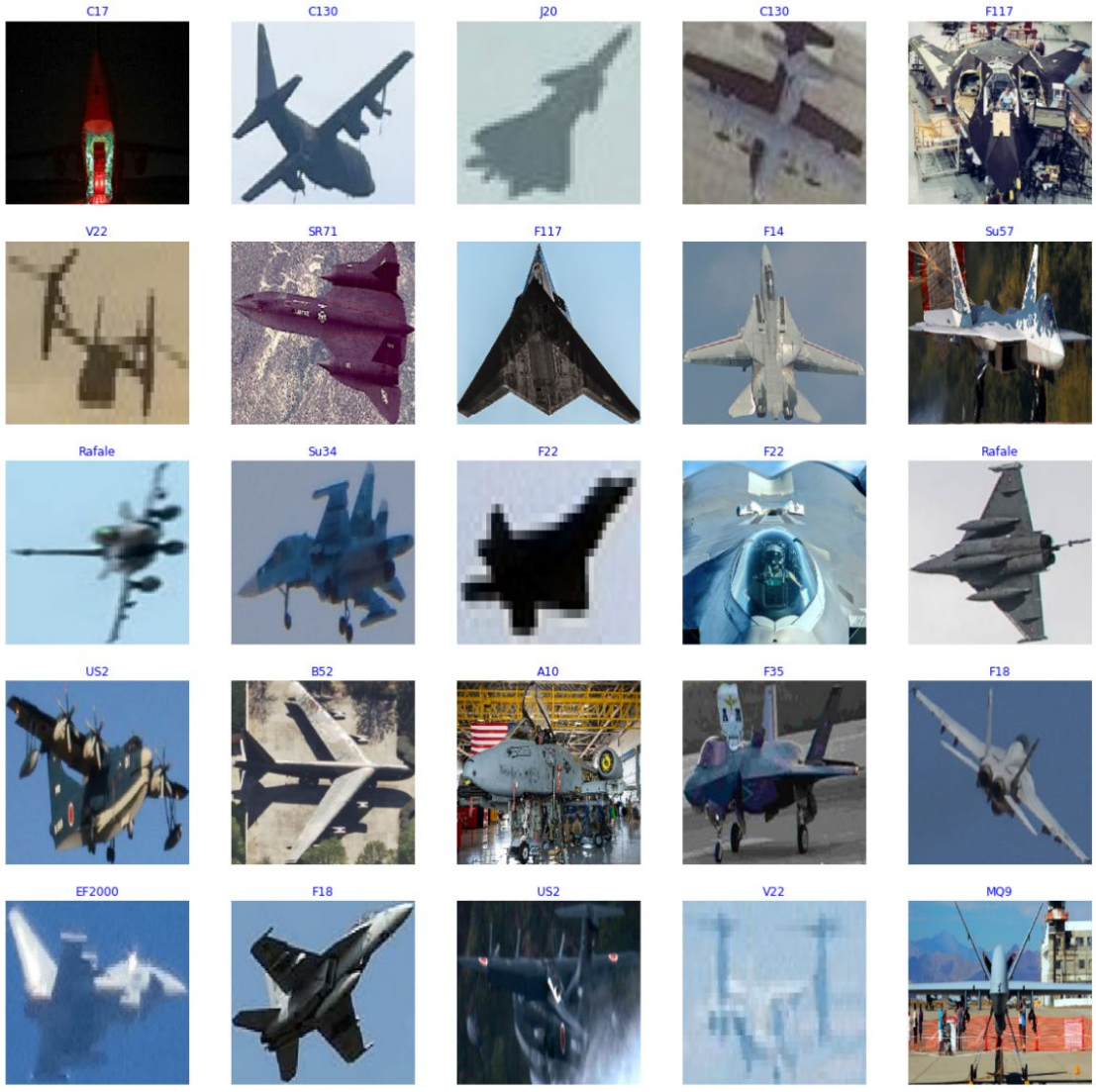


Figure 8. Visual Identification Training Set II

The test used Python code that loads various analytical libraries such as NumPy, Pandas, Seaborn, Efficient Net, and Tensor Flow Keras. The large dataset of 14,806 pictures is loaded, and the dataset can be randomly segmented or split into a training set and a testing set. The two sets are then further divided into smaller chunks of data for processing, for example, in batches of 40 figures. During training and validation, the levels of accuracy and information losses are measured. The training process is replicated and iterated multiple times over 20 neuron layers. Each iteration or epoch's results are fed back into the next round of iterations. Specifically, the weights are saved and used in the next iteration. Over time, the accuracy should increase if the weights are suboptimal. Predictions are then run using the best epoch result on a test dataset.

A. Moderate Dataset Test

```
Code File: Single Run Epoch - Codes and Results (Epoch 1 - Large Crop Dataset - Weights
None).ipynb

Found 14806 validated image filenames belonging to 43 classes.
Found 1851 validated image filenames belonging to 43 classes.
Found 1851 validated image filenames belonging to 43 classes.


Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnetb3 (Functional)  (None, 1536)              10783535
_________________________________________________________________
batch_normalization (BatchNo (None, 1536)              6144
_________________________________________________________________
dense (Dense)                (None, 256)               393472
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                11051
=================================================================
Total params: 11,194,202
Trainable params: 11,103,827
Non-trainable params: 90,375
_________________________________________________________________
```

B. Moderate Dataset Single Epoch Test

```
Epoch      Loss    Accuracy  V_loss   V_acc   LR       Next LR  Monitor    % Improv Duration
 1 /1      6.676    5.640   5.83630   5.348   0.00100  0.00100  accuracy     0.00   2128.45

training elapsed time was 0.0 hours, 35.0 minutes, 28.52 seconds)
```

C. Moderate Dataset Unweighted Multiple Epoch Test

```
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor   % Improv  Duration
 1 /10     9.324    4.870   7.75886   5.997   0.00100  0.00100  accuracy     0.00   2489.63
 2 /10     6.785    5.734   5.97386   4.970   0.00100  0.00100  accuracy    17.75   2475.55
 3 /10     5.295    5.694   4.99348   5.889   0.00100  0.00050  accuracy    -0.71   2889.53
```

4 /10	4.566	5.835	4.38049	6.159	0.00050	0.00050	accuracy	1.77	2441.16
5 /10	4.241	5.768	4.13966	6.159	0.00050	0.00025	accuracy	-1.16	2490.28
6 /10	4.049	6.274	4.06320	6.591	0.00025	0.00025	accuracy	7.52	2525.64
7 /10	3.957	5.727	3.92278	6.213	0.00025	0.00013	accuracy	-8.72	2485.77
8 /10	3.891	6.153	3.86949	6.159	0.00013	0.00006	accuracy	-1.94	2462.17
9 /10	3.858	6.099	3.84580	6.375	0.00006	0.00003	accuracy	-2.80	2469.47

training has been halted at epoch 9 after 3 adjustments of learning rate with no improvement

training elapsed time was 6.0 hours, 21.0 minutes, 29.59 seconds)

D. Moderate Dataset Weighted and Calibrated Multiple Epoch Test

Epoch	Loss	Accuracy	V_loss	V_acc	LR	Next LR	Monitor	% Improv	Duration
1 /5	8.702	20.208	6.13869	50.351	0.00100	0.00100	accuracy	0.00	2603.66
2 /5	4.788	59.949	3.35620	77.472	0.00100	0.00100	accuracy	196.66	2588.40
3 /5	2.629	81.690	1.99183	84.225	0.00100	0.00100	accuracy	36.27	2504.09
4 /5	1.512	90.977	1.30568	87.466	0.00100	0.00100	val_loss	34.45	2468.21
5 /5	0.984	94.914	1.01002	89.411	0.00100	0.00100	val_loss	22.64	2483.57

training elapsed time was 3.0 hours, 32.0 minutes, 0.86 seconds)

E. Training-Validation Loss vs. Accuracy

```
plot_training(history)
```

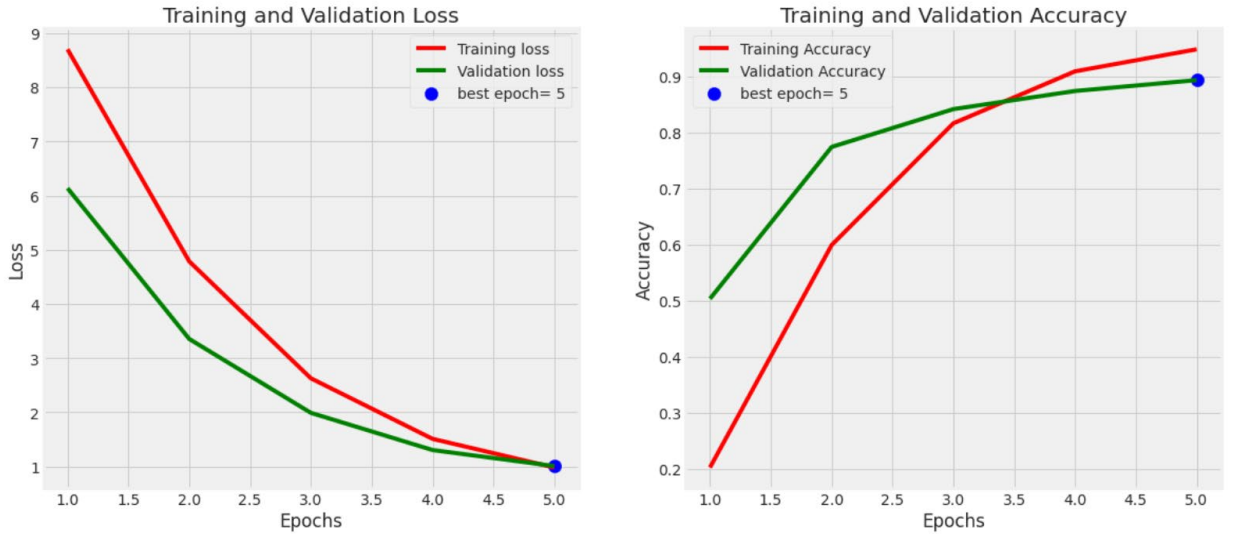


Figure 9. Training Accuracy and Results Validation

F. Confusion Matrix and Results Calibration

```
617/617 [] - 631s 1s/step - loss: 0.6556 - accuracy: 0.9930

617/617 [] - 77s 122ms/step - loss: 1.0100 - accuracy: 0.8941
617/617 [] - 124s 201ms/step - loss: 1.0450 - accuracy: 0.8833
Train Loss:  0.6555680632591248
Train Accuracy:  0.9930433630943298
--------------------
Validation Loss:  1.0100194215774536
Validation Accuracy:  0.8941112756729126
--------------------
Test Loss:  1.044999122619629
Test Accuracy:  0.8833063244819641

Confusion Matrix, Without Normalization
[[44  0  0 ...  0  0  0]
 [ 0 31  0 ...  0  0  0]
 [ 0  0 17 ...  0  0  0]
 ...
 [ 0  0  0 ... 28  0  0]
 [ 0  0  0 ...  0 10  0]
 [ 0  0  0 ...  0  1 11]]
```

Conclusion

The results indicate that multiple epoch iterations are a must for higher levels of accuracy. For example, running the same dataset returns a 5.6% accuracy. If a standard unweighted algorithm is applied, the accuracy increase is minimal at best. However, with a previously weighted neural network to kick start the process, the first epoch returns a 20.2% accuracy and using these initial weights, subsequent epochs show a more significant accuracy increase to as much as 94.9% accuracy after just 5 epochs. The main problem of running large datasets is the time it takes for the deep convolution neural network to run. With

a moderate dataset of only 14,806 pictures on 5 epochs, it can take upwards of 3½ hours. Therefore, if the timing is critical, and dedicated high-powered computing will be required. In addition, the limitations of deep neural networks, other than the time it takes to run, is the need for multiple iterations where any low-level accuracy and precision class types will need to be replaced with better pictures. Finally, real-life testing may be required to test the actual efficacy of the methodologies described in this report.

	precision	recall	f1-score	support
A10	0.88	0.79	0.83	56
A400M	0.79	0.94	0.86	33
AG600	1.00	0.89	0.94	19
AV8B	0.94	0.97	0.96	35
B1	0.94	0.85	0.89	52
B2	0.97	0.97	0.97	40
B52	1.00	0.91	0.95	43
Be200	0.82	0.96	0.88	24
C130	0.92	0.92	0.92	79
C17	0.91	0.73	0.81	44
C2	0.87	0.98	0.92	66
C5	0.91	0.94	0.93	33
E2	0.97	0.94	0.96	36
E7	1.00	0.82	0.90	11
EF2000	0.78	0.81	0.79	47
F117	0.97	0.97	0.97	29
F14	0.77	0.84	0.80	43
F15	0.86	0.91	0.88	107
F16	0.84	0.88	0.86	111
F18	0.89	0.87	0.88	107
F22	0.79	0.83	0.81	54
F35	0.87	0.91	0.89	92
F4	0.86	0.83	0.84	52
J20	0.91	0.89	0.90	44
JAS39	0.97	0.81	0.89	43
MQ9	0.83	0.86	0.84	28
Mig31	0.96	0.86	0.91	29
Mirage2000	0.77	0.94	0.85	35
P3	0.92	0.79	0.85	14
RQ4	0.92	0.89	0.91	27
Rafale	0.78	0.82	0.80	44
SR71	0.82	0.96	0.88	24
Su34	0.87	0.74	0.80	27
Su57	0.93	0.86	0.89	29
Tornado	1.00	0.72	0.84	36
Tu160	0.84	0.88	0.86	24
Tu95	1.00	1.00	1.00	24
U2	0.86	0.79	0.83	24
US2	1.00	0.96	0.98	56
V22	0.89	0.97	0.93	72
Vulcan	0.85	0.93	0.89	30
XB70	0.91	0.67	0.77	15
YF23	0.92	0.85	0.88	13
accuracy			0.88	1851
macro avg	0.90	0.88	0.88	1851
weighted avg	0.89	0.88	0.88	1851

VI. APPENDIX: COMBAT ID

Source: Kaggle (2023)

```
class_names = train_df.class_names

plt.figure(figsize = (20, 20))
for images, labels in train_df.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
```

Import needed modules

In [1]:

```
# import system libs
import os
import time
import shutil
import pathlib
import itertools

# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, Batch
Normalization
from tensorflow.keras import regularizers

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

print ('modules loaded')
```

Create needed functions

Functions to Create Data Frame from Dataset

Function to create data frame

In [2]:

```
# Generate data paths with labels
def define_paths(dir):
    filepaths = []
    labels = []

    folds = os.listdir(dir)
    for fold in folds:
        foldpath = os.path.join(dir, fold)
        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)
            filepaths.append(fpath)
            labels.append(fold)

    return filepaths, labels


# Concatenate data paths with labels into one dataframe ( to later be fitted into the model )
def define_df(files, classes):
    Fseries = pd.Series(files, name= 'filepaths')
    Lseries = pd.Series(classes, name='labels')
    return pd.concat([Fseries, Lseries], axis= 1)
```

Functions to check data splitting format

In [3]:

```
# Function that contain only a directory of data and it is not splitted
def tr_ts_data(tr_dir, ts_dir):
    # train and valid dataframe
    files, classes = define_paths(tr_dir)
    df = define_df(files, classes)
    strat = df['labels']
    train_df, valid_df = train_test_split(df, train_size= 0.8, shuffle= True, random_state= 123, str
atify= strat)

    # test dataframe
    files, classes = define_paths(tr_dir)
    test_df = define_df(files, classes)
    return train_df, valid_df, test_df

# Function that contain train and test directory of data.
def full_data(data_dir):
    # train dataframe
    files, classes = define_paths(data_dir)
    df = define_df(files, classes)
    strat = df['labels']
    train_df, dummy_df = train_test_split(df,  train_size= 0.8, shuffle= True, random_state= 123, st
ratify= strat)

    # valid and test dataframe
    strat = dummy_df['labels']
    valid_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle= True, random_state= 12
3, stratify= strat)

    return train_df, valid_df, test_df


# function that contain the three directory of data train, valid, and test
def tr_val_ts_data(tr_dir, val_dir, ts_dir):

    # train dataframe
    files, classes = define_paths(tr_dir)
    train_df = define_df(files, classes)

    # validation dataframe
    files, classes = define_paths(val_dir)
    valid_df = define_df(files, classes)

    # test dataframe
    files, classes = define_paths(ts_dir)
    test_df = define_df(files, classes)

    return train_df, valid_df, test_df
```

Function to split data into train, valid, test

In [4]:

```
def split_data(tr_dir, val_dir=None, ts_dir=None):
    '''
        This function split data into train, valid, and test after convert it to a dataframe.
        Dataset can be in several formats, it can contain train, valid, and test data, or it can conta
in only train and test data, etc.
        It depends on other needed function:
        - full_data function that contain only a directory of data and it is not splitted.
        - tr_ts_data function that contain train and test directory of data.
        - tr_val_ts_data function that contain the three directory of data train, valid, and test.
    '''

    # No Validation or Test data
    if val_dir == '' and ts_dir == '':
        train_df, valid_df, test_df = full_data(tr_dir)
        return train_df, valid_df, test_df

    # No Validation data
    elif val_dir == '' and ts_dir != '':
        train_df, valid_df, test_df = tr_ts_data(tr_dir, ts_dir)
        return train_df, valid_df, test_df

    # All data existed
    elif val_dir != '' and ts_dir != '':
        train_df, valid_df, test_df = tr_val_ts_data(tr_dir, val_dir, ts_dir)
        return train_df, valid_df, test_df
```

Function to generate images from dataframe

In [5]:

```
def create_model_data (train_df, valid_df, test_df, batch_size):
    '''
    This function takes train, validation, and test dataframe and fit them into image data generator,
because model takes data from image data generator.
    Image data generator converts images into tensors. '''


    # define model parameters
    img_size = (224, 224)
    channels = 3 # either BGR or Grayscale
    color = 'rgb'
    img_shape = (img_size[0], img_size[1], channels)

    # Recommended : use custom function for test data batch size, else we can use normal batch size.
    ts_length = len(test_df)
    test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n ==
0 and ts_length/n <= 80]))
    test_steps = ts_length // test_batch_size

    # This function which will be used in image data generator for data augmentation, it just take the
image and return it again.
    def scalar(img):
        return img

    tr_gen = ImageDataGenerator(preprocessing_function= scalar, horizontal_flip= True)
    ts_gen = ImageDataGenerator(preprocessing_function= scalar)

    train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'filepaths', y_col= 'labels', target_si
ze= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= True, batch_size= batch_size)

    valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'filepaths', y_col= 'labels', target_si
ze= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= True, batch_size= batch_size)

    # Note: we will use custom test_batch_size, and make shuffle= false
    test_gen = ts_gen.flow_from_dataframe( test_df, x_col= 'filepaths', y_col= 'labels', target_size
= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= False, batch_size= test_batch_si
ze)

    return train_gen, valid_gen, test_gen
```

Function to display data sample

In [6]:

```
def show_images(gen):
    '''
    This function take the data generator and show sample of the images
    '''

    # return classes , images to be displayed
    g_dict = gen.class_indices        # defines dictionary {'class': index}
    classes = list(g_dict.keys())     # defines list of dictionary's kays (classes), classes names :
string
    images, labels = next(gen)        # get a batch size samples from the generator

    # calculate number of displayed samples
    length = len(labels)        # length of batch size
    sample = min(length, 25)    # check if sample less than 25 images

    plt.figure(figsize= (20, 20))

    for i in range(sample):
        plt.subplot(5, 5, i + 1)
        image = images[i] / 255       # scales data to range (0 - 255)
        plt.imshow(image)
        index = np.argmax(labels[i])  # get image index
        class_name = classes[index]   # get class of image
        plt.title(class_name, color= 'blue', fontsize= 12)
        plt.axis('off')
    plt.show()
```

Function to plot value counts for a column in a dataframe

In [7]:

```
def plot_label_count(df, plot_title):
    '''
    This function take df and plot labels value counts
    '''

    # Define needed variables
    vcounts = df['labels'].value_counts()
    labels = vcounts.keys().tolist()
    values = vcounts.tolist()
    lcount = len(labels)

    if lcount > 55:
        print('The number of labels is > 55, no plot will be produced')

    else:
        plot_labels(lcount, labels, values, plot_title)

def plot_labels(lcount, labels, values, plot_title):
    width = lcount * 4
    width = np.min([width, 20])

    plt.figure(figsize= (width, 5))

    form = {'family': 'serif', 'color': 'blue', 'size': 25}
    sns.barplot(labels, values)
    plt.title(f'Images per Label in {plot_title} data', fontsize= 24, color= 'blue')
    plt.xticks(rotation= 90, fontsize= 18)
    plt.yticks(fontsize= 18)
    plt.xlabel('CLASS', fontdict= form)
    yaxis_label = 'IMAGE COUNT'
    plt.ylabel(yaxis_label, fontdict= form)

    rotation = 'vertical' if lcount >= 8 else 'horizontal'
    for i in range(lcount):
        plt.text(i, values[i] / 2, str(values[i]), fontsize= 12,
                rotation= rotation, color= 'yellow', ha= 'center')

    plt.show()
```

Callbacks

Callbacks : Helpful functions to help optimize model training
Examples: stop model training after specfic time, stop training if no improve in accuracy and so on.

In [8]:

```
class MyCallback(keras.callbacks.Callback):
    def __init__(self, model, patience, stop_patience, threshold, factor, batches, epochs, ask_epoc
h):
        super(MyCallback, self).__init__()
        self.model = model
        self.patience = patience # specifies how many epochs without improvement before learning rate
is adjusted
        self.stop_patience = stop_patience # specifies how many times to adjust lr without improvemen
t to stop training
        self.threshold = threshold # specifies training accuracy threshold when lr will be adjusted b
ased on validation loss
        self.factor = factor # factor by which to reduce the learning rate
        self.batches = batches # number of training batch to run per epoch
        self.epochs = epochs
        self.ask_epoch = ask_epoch
        self.ask_epoch_initial = ask_epoch # save this value to restore if restarting training

        # callback variables
        self.count = 0 # how many times lr has been reduced without improvement
        self.stop_count = 0
        self.best_epoch = 1   # epoch with the lowest loss
        self.initial_lr = float(tf.keras.backend.get_value(model.optimizer.lr)) # get the initial le
arning rate and save it
        self.highest_tracc = 0.0 # set highest training accuracy to 0 initially
        self.lowest_vloss = np.inf # set lowest validation loss to infinity initially
        self.best_weights = self.model.get_weights() # set best weights to model's initial weights
        self.initial_weights = self.model.get_weights()   # save initial weights if they have to get
restored

    # Define a function that will run when train begins
    def on_train_begin(self, logs= None):
        msg = 'Do you want model asks you to halt the training [y/n] ?'
        print(msg)
        ans = input('')
        if ans in ['Y', 'y']:
            self.ask_permission = 1
        elif ans in ['N', 'n']:
            self.ask_permission = 0

        msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s}{9:^8s}'.format('Epo
ch', 'Loss', 'Accuracy', 'V_loss', 'V_acc', 'LR', 'Next LR', 'Monitor','% Improv', 'Duration')
        print(msg)
        self.start_time = time.time()
```

```
    def on_train_end(self, logs= None):
        stop_time = time.time()
        tr_duration = stop_time - self.start_time
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))

        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f} minutes, {seconds:4.2f}
seconds)'
        print(msg)

        # set the weights of the model to the best weights
        self.model.set_weights(self.best_weights)


    def on_train_batch_end(self, batch, logs= None):
        # get batch accuracy and loss
        acc = logs.get('accuracy') * 100
        loss = logs.get('loss')

        # prints over on the same line to show running batch count
        msg = '{0:20s}processing batch {1:} of {2:5s}-   accuracy=  {3:5.3f}   -   loss: {4:8.5f}'.f
ormat(' ', str(batch), str(self.batches), acc, loss)
        print(msg, '\r', end= '')


    def on_epoch_begin(self, epoch, logs= None):
        self.ep_start = time.time()


    # Define method runs on the end of each epoch
    def on_epoch_end(self, epoch, logs= None):
        ep_end = time.time()
        duration = ep_end - self.ep_start

        lr = float(tf.keras.backend.get_value(self.model.optimizer.lr)) # get the current learning ra
te
        current_lr = lr
        acc = logs.get('accuracy')  # get training accuracy
        v_acc = logs.get('val_accuracy')  # get validation accuracy
        loss = logs.get('loss')  # get training loss for this epoch
        v_loss = logs.get('val_loss')  # get the validation loss for this epoch

        if acc < self.threshold: # if training accuracy is below threshold adjust lr based on trainin
g accuracy
            monitor = 'accuracy'
            if epoch == 0:
                pimprov = 0.0
            else:
                pimprov = (acc - self.highest_tracc ) * 100 / self.highest_tracc # define improvemen
t of model progres
```

```
            if acc > self.highest_tracc: # training accuracy improved in the epoch
                self.highest_tracc = acc # set new highest training accuracy
                self.best_weights = self.model.get_weights() # training accuracy improved so save the
weights
                self.count = 0 # set count to 0 since training accuracy improved
                self.stop_count = 0 # set stop counter to 0
                if v_loss < self.lowest_vloss:
                    self.lowest_vloss = v_loss
                self.best_epoch = epoch + 1  # set the value of best epoch for this epoch

            else:
                # training accuracy did not improve check if this has happened for patience number of
epochs
                # if so adjust learning rate
                if self.count >= self.patience - 1: # lr should be adjusted
                    lr = lr * self.factor # adjust the learning by factor
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate
in the optimizer
                    self.count = 0 # reset the count to 0
                    self.stop_count = self.stop_count + 1 # count the number of consecutive lr adjust
ments
                    self.count = 0 # reset counter
                    if v_loss < self.lowest_vloss:
                        self.lowest_vloss = v_loss
                else:
                    self.count = self.count + 1 # increment patience counter

        else: # training accuracy is above threshold so adjust learning rate based on validation loss
            monitor = 'val_loss'
            if epoch == 0:
                pimprov = 0.0

            else:
                pimprov = (self.lowest_vloss - v_loss ) * 100 / self.lowest_vloss

            if v_loss < self.lowest_vloss: # check if the validation loss improved
                self.lowest_vloss = v_loss # replace lowest validation loss with new validation loss
                self.best_weights = self.model.get_weights() # validation loss improved so save the w
eights
                self.count = 0 # reset count since validation loss improved
                self.stop_count = 0
                self.best_epoch = epoch + 1 # set the value of the best epoch to this epoch

            else: # validation loss did not improve
                if self.count >= self.patience - 1: # need to adjust lr
                    lr = lr * self.factor # adjust the learning rate
                    self.stop_count = self.stop_count + 1 # increment stop counter because lr was adj
usted
                    self.count = 0 # reset counter
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate
in the optimizer
```

```
                else:
                    self.count = self.count + 1 # increment the patience counter

                if acc > self.highest_tracc:
                    self.highest_tracc = acc

        msg = f'{str(epoch + 1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc * 100:^9.3f}{v_loss:^9.5
f}{v_acc * 100:^9.3f}{current_lr:^9.5f}{lr:^9.5f}{monitor:^11s}{pimprov:^10.2f}{duration:^8.2f}'
        print(msg)

        if self.stop_count > self.stop_patience - 1: # check if learning rate has been adjusted stop_
count times with no improvement
            msg = f' training has been halted at epoch {epoch + 1} after {self.stop_patience} adjust
ments of learning rate with no improvement'
            print(msg)
            self.model.stop_training = True # stop training

        else:
            if self.ask_epoch != None and self.ask_permission != 0:
                if epoch + 1 >= self.ask_epoch:
                    msg = 'enter H to halt training or an integer for number of epochs to run then a
sk again'
                    print(msg)

                    ans = input('')
                    if ans == 'H' or ans == 'h':
                        msg = f'training has been halted at epoch {epoch + 1} due to user input'
                        print(msg)
                        self.model.stop_training = True # stop training

                    else:
                        try:
                            ans = int(ans)
                            self.ask_epoch += ans
                            msg = f' training will continue until epoch {str(self.ask_epoch)}'
                            print(msg)
                            msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s}
{9:^8s}'.format('Epoch', 'Loss', 'Accuracy', 'V_loss', 'V_acc', 'LR', 'Next LR', 'Monitor', '% Impro
v', 'Duration')
                            print(msg)

                        except Exception:
                            print('Invalid')
```

Function to plot history of training

In [9]:

```
def plot_training(hist):
    '''
    This function take training model and plot history of accuracy and losses with the best epoch in b
oth of them.
    '''

    # Define needed variables
    tr_acc = hist.history['accuracy']
    tr_loss = hist.history['loss']
    val_acc = hist.history['val_accuracy']
    val_loss = hist.history['val_loss']
    index_loss = np.argmin(val_loss)
    val_lowest = val_loss[index_loss]
    index_acc = np.argmax(val_acc)
    acc_highest = val_acc[index_acc]
    Epochs = [i+1 for i in range(len(tr_acc))]
    loss_label = f'best epoch= {str(index_loss + 1)}'
    acc_label = f'best epoch= {str(index_acc + 1)}'

    # Plot training history
    plt.figure(figsize= (20, 8))
    plt.style.use('fivethirtyeight')

    plt.subplot(1, 2, 1)
    plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
    plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
    plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
    plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
    plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout
    plt.show()
```

Function to create Confusion Matrix

In [10]:

```
def plot_confusion_matrix(cm, classes, normalize= False, title= 'Confusion Matrix', cmap= plt.cm.Blu
es):
        '''
        This function plot confusion matrix method from sklearn package.
        '''

        plt.figure(figsize= (10, 10))
        plt.imshow(cm, interpolation= 'nearest', cmap= cmap)
        plt.title(title)
        plt.colorbar()

        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation= 45)
        plt.yticks(tick_marks, classes)

        if normalize:
                cm = cm.astype('float') / cm.sum(axis= 1)[:, np.newaxis]
                print('Normalized Confusion Matrix')

        else:
                print('Confusion Matrix, Without Normalization')

        print(cm)

        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                plt.text(j, i, cm[i, j], horizontalalignment= 'center', color= 'white' if cm[i, j] >
thresh else 'black')

        plt.tight_layout()
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
```

Model Structure

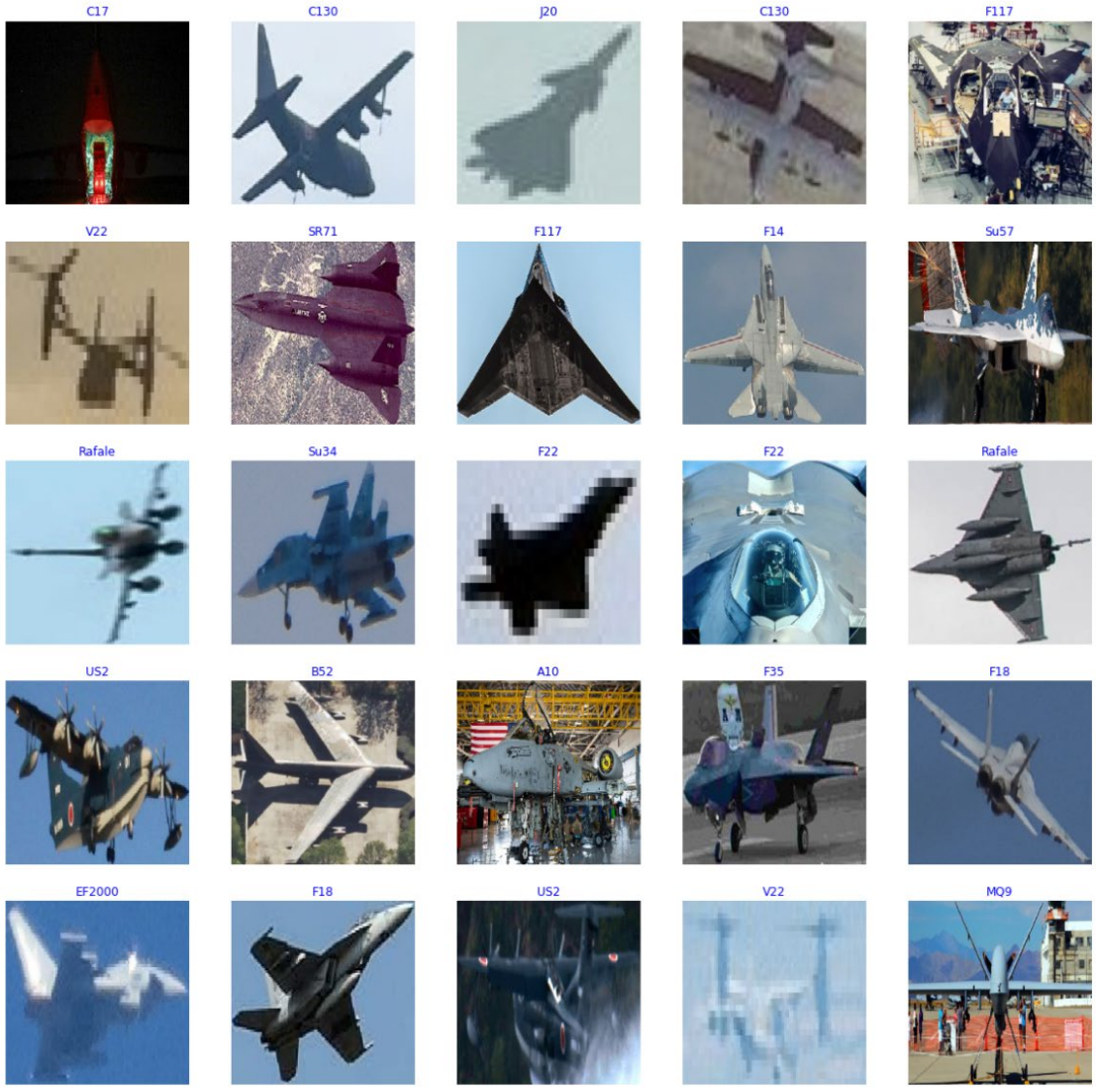
Start Reading Dataset

In [11]:

```
train_dir = input('Enter train data directory: ')
valid_dir = input('Enter validation data directory (if no valid dir press Enter): ')
test_dir = input('Enter test data directory (if no test dir press Enter): ')

try:
    # Get splitted data
    train_df, valid_df, test_df = split_data(train_dir, valid_dir, test_dir)

    # Get Generators
    batch_size = 40
    train_gen, valid_gen, test_gen = create_model_data(train_df, valid_df, test_df, batch_size)

except:
    print('Invalid Input')
```

Display Image Sample

In [12]:
```
show_images(train_gen)
```



Generic Model Creation

In [14]:

```
# Create Model Structure
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)
class_count = len(list(train_gen.class_indices.keys())) # to define number of classes in dense layer

# create pre-trained model (you can built on pretrained model such as :  efficientnet, VGG , Resnet )
# we will use efficientnetb3 from EfficientNet family.
base_model = tf.keras.applications.efficientnet.EfficientNetB3(include_top= False, weights= "imagene
t", input_shape= img_shape, pooling= 'max')

model = Sequential([
    base_model,
    BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
    Dense(256, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer= regularizers.l1
(0.006),
                bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
    Dropout(rate= 0.45, seed= 123),
    Dense(class_count, activation= 'softmax')
])

model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy', metrics= ['accuracy'])

model.summary()
```

Set Callback Parameters

In [15]:

```
batch_size = 40   # set batch size for training
epochs = 40   # number of all epochs in training
patience = 1   #number of epochs to wait to adjust lr if monitored value does not improve
stop_patience = 3   # number of epochs to wait before stopping training if monitored value does not im
prove
threshold = 0.9   # if train accuracy is < threshold adjust monitor accuracy, else monitor validation
loss
factor = 0.5   # factor to reduce lr by
ask_epoch = 5   # number of epochs to run before asking if you want to halt training
batches = int(np.ceil(len(train_gen.labels) / batch_size))    # number of training batch to run per e
poch

callbacks = [MyCallback(model= model, patience= patience, stop_patience= stop_patience, threshold= t
hreshold,
            factor= factor, batches= batches, epochs= epochs, ask_epoch= ask_epoch )]
```

Train model

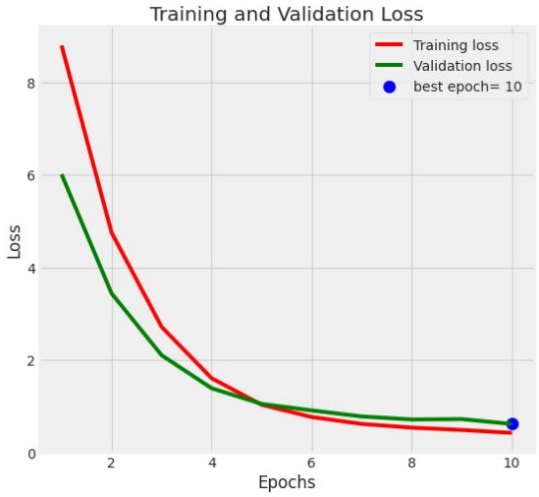
In [16]:

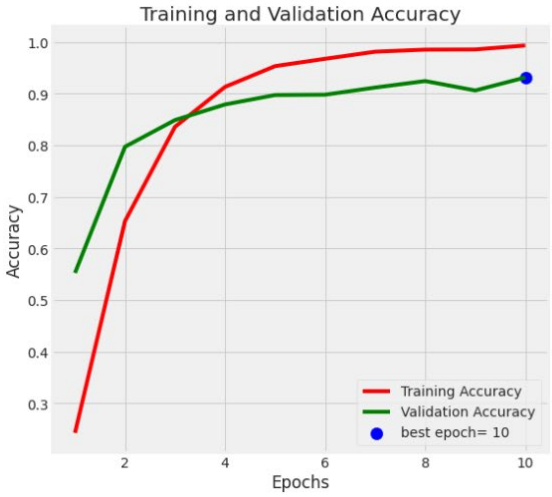
```
history = model.fit(x= train_gen, epochs= epochs, verbose= 0, callbacks= callbacks,
                    validation_data= valid_gen, validation_steps= None, shuffle= False)
```

```
Epoch     Loss   Accuracy  V_loss    V_acc     LR      Next LR  Monitor  % Improv  Duration
2023-01-29 23:31:33.216645:
2023-01-29 23:31:46.846608:
 1 /40     8.797   24.267   6.01173  55.261   0.00100  0.00100  accuracy     0.00    284.53
 2 /40     4.750   65.301   3.43436  79.710   0.00100  0.00100  accuracy    169.09   198.75
 3 /40     2.710   83.572   2.09777  84.877   0.00100  0.00100  accuracy    27.98    196.53
 4 /40     1.601   91.309   1.38350  87.902   0.00100  0.00100  val_loss    34.05    197.33
 5 /40     1.026   95.328   1.04514  89.729   0.00100  0.00100  val_loss    24.46    197.62
enter H to halt training or an integer for number of epochs to run then ask again
 training will continue until epoch 10
 Epoch     Loss   Accuracy  V_loss    V_acc     LR      Next LR  Monitor  % Improv  Duration
 6 /40     0.761   96.746   0.90429  89.792   0.00100  0.00100  val_loss    13.48    199.04
 7 /40     0.612   98.125   0.77389  91.178   0.00100  0.00100  val_loss    14.42    197.55
 8 /40     0.531   98.527   0.70789  92.439   0.00100  0.00100  val_loss     8.53    199.58
 9 /40     0.482   98.558   0.71645  90.611   0.00100  0.00050  val_loss    -1.21    199.12
10 /40     0.419   99.330   0.61102  93.132   0.00050  0.00050  val_loss    13.68    197.91
enter H to halt training or an integer for number of epochs to run then ask again
training has been halted at epoch 10 due to user input
training elapsed time was 0.0 hours, 35.0 minutes, 49.01 seconds)
```

Display model performance

In [17]:
```
plot_training(history)
```





Evaluate model

In [18]:
```
ts_length = len(test_df)
test_batch_size = test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if t
s_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = model.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

```
23/23 [==============================] - 10s 415ms/step - loss: 0.3272 - accuracy:
1.0000
23/23 [==============================] - 9s 369ms/step - loss: 0.5881 - accuracy:
0.9370
23/23 [==============================] - 27s 1s/step - loss: 0.6121 - accuracy: 0.9294
Train Loss:  0.3272090554237366
Train Accuracy:  1.0
--------------------
```

```
Validation Loss:  0.5881201028823853
Validation Accuracy:  0.936956524848938
--------------------
Test Loss:  0.6120601892471313
Test Accuracy:  0.9294266104698181
```

Get Predictions

In [19]:

```
preds = model.predict_generator(test_gen)
y_pred = np.argmax(preds, axis=1)
print(y_pred)
```

```
[21 16 11 ... 20 19 15]
```

Confusion Matrics and Classification Report

In [20]:

```
g_dict = test_gen.class_indices
classes = list(g_dict.keys())

# Confusion matrix
cm = confusion_matrix(test_gen.classes, y_pred)
plot_confusion_matrix(cm= cm, classes= classes, title = 'Confusion Matrix')

# Classification report
print(classification_report(test_gen.classes, y_pred, target_names= classes))
```

```
Confusion Matrix, Without Normalization
[[51  0  0 ...  0  0  0]
 [ 0 27  0 ...  0  0  0]
 [ 0  1 16 ...  0  0  0]
 ...
 [ 0  0  0 ... 27  0  0]
 [ 0  0  0 ...  0 12  0]
 [ 0  0  0 ...  0  1 11]]
```

	precision	recall	f1-score	support
A10	0.94	0.96	0.95	53
A400M	0.93	0.96	0.95	28
AG600	1.00	0.94	0.97	17
AV8B	0.97	0.97	0.97	35
B1	0.91	0.91	0.91	47
B2	0.92	0.95	0.94	38
B52	0.97	0.93	0.95	41
Be200	0.95	0.91	0.93	23
C130	0.94	0.93	0.93	68
C17	0.97	0.95	0.96	38
C5	0.83	0.86	0.84	28
E2	1.00	0.85	0.92	33
EF2000	0.84	0.86	0.85	43
F117	0.96	1.00	0.98	27
F14	0.95	0.98	0.96	41
F15	0.88	0.97	0.92	88
F16	0.90	0.92	0.91	100
F18	0.93	0.89	0.91	101
F22	0.94	0.92	0.93	48
F35	0.90	0.97	0.93	80
F4	0.93	0.91	0.92	43
J20	0.93	1.00	0.97	42
JAS39	0.87	0.95	0.91	43
MQ9	1.00	0.90	0.95	21
Mig31	0.96	0.93	0.95	28
Mirage2000	0.93	0.88	0.90	32
RQ4	0.96	1.00	0.98	25
Rafale	0.97	0.85	0.91	40
SR71	0.89	0.94	0.92	18
Su34	0.96	0.92	0.94	24
Su57	0.92	0.89	0.91	27
Tornado	0.89	0.78	0.83	32
Tu160	1.00	0.90	0.95	21
Tu95	0.80	0.95	0.87	21
U2	0.90	0.90	0.90	21
US2	1.00	0.96	0.98	52
V22	0.96	0.97	0.96	67
Vulcan	1.00	0.96	0.98	28
XB70	0.92	0.92	0.92	13
YF23	1.00	0.92	0.96	12
accuracy			0.93	1587
macro avg	0.94	0.93	0.93	1587
weighted avg	0.93	0.93	0.93	1587

Save model

In [21]:

```
model_name = model.input_names[0][:-6]
subject = input('Enter Project Subject')
acc = test_score[1] * 100
save_path = ''

# Save model
save_id = str(f'{model_name}-{subject}-{"%.2f" %round(acc, 2)}.h5')
model_save_loc = os.path.join(save_path, save_id)
model.save(model_save_loc)
print(f'model was saved as {model_save_loc}')

# Save weights
weight_save_id = str(f'{model_name}-{subject}-weights.h5')
weights_save_loc = os.path.join(save_path, weight_save_id)
model.save_weights(weights_save_loc)
print(f'weights were saved as {weights_save_loc}')
```

```
model was saved as efficientnetb3-Military Aircraft Detection-92.94.h5
weights were saved as efficientnetb3-Military Aircraft Detection-weights.h5
```

Generate CSV files containing classes indicies & image size

In [22]:

```
class_dict = train_gen.class_indices
img_size = train_gen.image_shape
height = []
width = []
for _ in range(len(class_dict)):
    height.append(img_size[0])
    width.append(img_size[1])

Index_series = pd.Series(list(class_dict.values()), name= 'class_index')
Class_series = pd.Series(list(class_dict.keys()), name= 'class')
Height_series = pd.Series(height, name= 'height')
Width_series = pd.Series(width, name= 'width')
class_df = pd.concat([Index_series, Class_series, Height_series, Width_series], axis= 1)
csv_name = f'{subject}-class_dict.csv'
csv_save_loc = os.path.join(save_path, csv_name)
class_df.to_csv(csv_save_loc, index= False)
print(f'class csv file was saved as {csv_save_loc}')
```

```
class csv file was saved as Military Aircraft Detection-class_dict.csv
```

Sample Computer Algorithm and Code Snippet

The neural network algorithm can be executed as shown below.

```
}

double CNeuralNetworkTANH::GetOptimizedPara(std::vector< std::vector< double > >& mHiddenLayerWeights, std::vector<
double >& vecHiddenLayerBias, std::vector< double >& vecOutputLayerWeights, double& dOutputBias)
{
}

double CNeuralNetworkTANH::GetOptimizedParaFinal(double& dMultiple, double& dAddition, double dMaxActual)
{
	grg2wrapper g;

	double* xx = new double[3];
	double* xlb = new double[3];
	double* xub = new double[3];

	double* glb = new double[2];
	double* gub = new double[2];
	glb[1] = -1.0e10;
	gub[1] = 1.0e10;

	remove( "NeuralNetworkTANHFinal.txt" );
	char *title = "Neural Network TANH Final", *report = "NeuralNetworkTANHFinal.txt";

	xx[1] = dMultiple;
	xlb[1] = 0.01;
	xub[1] = 10.0;

	xx[2] = dAddition;
	xlb[2] = 0.01 * dMaxActual;
	xub[2] = 10.0 * dMaxActual;

	g.nvars = 2;
	g.nfuns = 1;
	g.nobj = 1;
	g.xlb=xlb;
	g.xub=xub;
	g.glb=glb;
	g.gub=gub;
	g.title=title;
	g.report=report;
	g.xx=xx;
	g.GoptG3(g_COMP_NEURALNETWORKTANHFINAL);

	double dOptValue = std::numeric_limits<double>::quiet_NaN();
	g.RunGRG3(dOptValue, 0);

	dMultiple = xx[1];
	dAddition = xx[2];

	delete [] xx;
	delete [] xlb;
	delete [] xub;
	delete [] glb;
	delete [] gub;

	return dOptValue;
```

```
}

double CNeuralNetworkTANH::CalculateOptimizedSumSquares(double x[])
{
	std::vector< std::vector< double > > matrixHiddenLayWeights(*m_pNLayers);
	std::vector< double > arrayHiddenLayBias(*m_pNLayers);
	std::vector< double > arrayOutLayWeights(*m_pNLayers);
	double singleOutBias = Maths::DoubleNaN;

	int nXInd = 1;

	//Hidden Layer Weights
	for (int i=0; i<*m_pNLayers; i++)
	{
	…
		{
	…
	}
	}

	//Hidden Layer Bias
	for (int i=0; i<*m_pNLayers; i++)
	{
	…
	…
	…

	}

	//Output Layer Weights
	for (int i=0; i<*m_pNLayers; i++)
	{
		arrayOutLayWeights[i] = x[nXInd];
		nXInd++;
	}

	//Output Layer Bias
	singleOutBias = x[nXInd];

	…
	…

	double dSumSquare = 0.0;

	for (int i=0; i<nTrainingSet; i++)
	{
		std::vector< double> vecInputVars(0);
		for (int j=0; j<*m_pNLayers; j++)
		{
	…
	}

		//Response
		std::vector< double > vecResponse(0);
		for (int j=0; j<*m_pNLayers; j++)
		{
	…
	…
		}
```

```
                //Output
        …

                dSumSquare += pow(dOutput - dActual, 2);
        }

        return dSumSquare;
}

double CNeuralNetworkTANH::CalculateOptimizedSumSquaresFinal(double x[])
{
        double dMultiple = x[1];
        double dAddition = x[2];

        double dSumSquare = 0.0;
        for(size_t i=0; i<m_pActual->size(); i++)
        {
        …
        …
        …
        }

        return dSumSquare;
}

double CNeuralNetworkTANH::CalculateHiddenLayerResponse(double HiddenBias, std::vector<double>& InputVars,
std::vector<double>& HiddenLayerWeights)
{
        return CalculateOutput(HiddenBias, InputVars, HiddenLayerWeights);
}

double CNeuralNetworkTANH::CalculateOutput(double OutputBias, std::vector<double>& OutputLayerWeights,
std::vector<double>& HiddenLayerWeightsResponse)
{
        double dSum = 0.0;
        size_t OutCount = OutputLayerWeights.size();
        size_t HiddenCount = HiddenLayerWeightsResponse.size();
        size_t nCount = OutCount < HiddenCount ? OutCount : HiddenCount;

        for (size_t i=0; i<nCount; i++)
        {
        …
        …
        }

        //return ( 1.0 / ( 1.0 + exp( -(OutputBias + dSum) ) ) );
        …
        …
        return ( (dExpPlus - dExpMinus) / (dExpPlus + dExpMinus) );
}
```

VII. REFERENCES

Addanki, R., Battaglia, P. W., Budden, D., Deac, A., Godwin, J., Keck, T., Li, W. L. S., Sanchez-Gonzalez, A., Stott, J., Thakoor, S., Veličković, Petar. Large-scale graph representation learning with very deep GNNs and self-supervision. (2021). https://arxiv.org/abs/2107.09422

Anantharam, Pramod, Knowledge-Empowered Probabilistic Graphical Models for Physical-Cyber-Social Systems (2016). https://corescholar.libraries.wright.edu/etd_all/1517

Bryant, D. J. (2010). *Combat Identification Decision Making: Effect of a Secondary Task*. DEFENCE RESEARCH AND DEVELOPMENT TORONTO (CANADA). Accessed: https://apps.dtic.mil/sti/pdfs/ADA551146.pdf

Claudio Stamile, Aldo Marzullo, Enrico Deusebio. Graph Machine Learning https://www.perlego.com/book/2708215/graph-machine-learning-pdf

Daley, S. (2019, September 24). *19 examples of artificial intelligence shaking up business as usual*. https://builtin.com/artificial-intelligence/examples-ai-in-industry

Darken, R. (2019, October 21). *Human-machine teaming AI*. Naval Postgraduate School.

DARPA. (2019). *AI next campaign*. https://www.darpa.mil/work-with-us/ai-next-campaign

Denning, P. (2019, September). *Harnessing artificial intelligence*. Naval Postgraduate School.

Department of Defense. (2019, February 12). *Summary of the 2018 Department of Defense artificial intelligence strategy: Harnessing AI to advance our security and prosperity*. https://media.defense.gov/2019/Feb/12/2002088963/-1/-1/1/SUMMARY-OF-DOD-AI-STRATEGY.PDF

Eykholt, K., Evitimov, I., Fernandes, E., Li, B., Rahmati, A., Xia, C., & Song, D. (2018). *Robust physical-world attacks on deep learning visual classification*. Cornell University.

Flovik, V. (Aug. 12, 2020). What is graph theory, and why should you care? From graph theory to path optimization. Accessed: https://towardsdatascience.com/what-is-graph-theory-and-why-should-you-care-28d6a715a5c2

Gardner, H. (1993). *Multiple intelligences*. Basic Books.

Greenfield, D. (2019, June 19). *Artificial intelligence in medicine: Applications, implications, and limitations*. Harvard University. http://sitn.hms.harvard.edu/flash/2019/artificial-intelligence-in-medicine-applications-implications-and-limitations/

Gross, Jonathan L., and Jay Yellen. *Handbook of graph theory*. CRC Press, 2003.

Gunning, D. (2017, November). *Explainable artificial intelligence*. DARPA. https://www.darpa.mil/attachments/xaiprogramupdate.pdf

Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On past, present, and future of AI. *California Management Review*, *61*(4), 5–14.

Harary, F. (1969) *Graph theory*. Addison-Wesley Publishing Company.

Hu R. (2020) U.S. Air Force Target Knowledge Graph Construction Based on Multi-source Intelligence Analysis. In: Huang C., Chan YW., Yen N. (eds) Data Processing Techniques and Applications for Cyber-Physical Systems (DPTA 2019). Advances in Intelligent Systems and Computing, vol 1088. Springer, Singapore. https://doi.org/10.1007/978-981-15-1468-5_41 https://link.springer.com/chapter/10.1007/978-981-15-1468-5_41

Institute for Robotic Process Automation & Artificial Intelligence. (2019, November 16). *What is robotic process automation?* https://irpaai.com/what-is-robotic-process-automation/

Javatpoint (n.d.) Graph Theory Tutorial. Accessed: https://www.javatpoint.com/graph-theory-applications

Johnson, B. (2020). Predictive analytics in the naval maritime domain. In *Proceedings of the AAAI Symposium on the 2nd Workshop on Deep Models and Artificial Intelligence for Defense Applications: Potentials, Theories, Practices, Tools, and Risks*. Accessed: https://ceur-ws.org/Vol-2819/session2paper1.pdf

King, A. D. (2019, November 16). *Talk to a transformer*. https://talktotransformer.com/

Kipf, Thomas N and Welling, Max. (2016). Semi-Supervised Classification with Graph Convolutional Networks. https://arxiv.org/abs/1609.02907 https://tkipf.github.io/graph-convolutional-networks/

Knight, W. (2017, April 11). *The dark secret at the heart of AI*. Technology Review. https://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai/

Larrañaga, P., & Moral, S. (2011). Probabilistic graphical models in artificial intelligence. *Applied soft computing*, *11*(2), 1511-1528. Accessed: http://cig.fi.upm.es/wp-content/uploads/2022/03/Larranaga2011-AppliedSoftComputing-1.pdf

Liu, Zhiyuan, Jie Zhou: Tsinghua University. Introduction to Graph Neural Networks. Morgan & Claypool Publishers, 2020. ISBN: 9781681737669

Mun, J. (2015). *Modeling Risk: Applying Monte Carlo risk simulation, strategic real options, stochastic forecasting, portfolio optimization, data analytics, business intelligence, and decision modeling* (3rd ed.). CA: Thomson-Shore and ROV Press.

Mun, J. (2016). *Real options analysis: Tools and techniques for valuing strategic investments and decisions with integrated risk management and advanced quantitative decision analytics* (3rd ed.).CA: Thomson-Shore and ROV Press.

Mun, J. (2016a). *Real options analysis* (3rd ed.). Thomson-Shore and ROV Press.

Mun, J. (2022a). *Quantitative Research Methods* (5th ed.). ROV Press.

Mun, J. (2022b). Management and Business Knowledge Representation for Decision Making. Acquisition Innovation Research Center. Retrieved: https://apps.dtic.mil/sti/pdfs/AD1174813.pdf

Nayak, P. (2019, October 25). *Understanding searches better than ever before*. Google. https://blog.google/products/search/search-language-understanding-bert

O'Conner, J. J., and E. F. Robertson (2014). Dénes König. MacTutor History of Mathematics Archive. Accessed at https://mathshistory.st-andrews.ac.uk/Biographies/Konig_Denes/

Oppy, G., & Dowe, D. (2016, February 8). *The Turing Test*. In E. N. Zalta (ed.), *Stanford encyclopedia of philosophy.* https://plato.stanford.edu/entries/turing-test/

Parloff, R. (2016, September 28). From 2016: Why deep learning is suddenly changing your life. *Fortune.* https://fortune.com/longform/ai-artificial-intelligence-deep-machine-learning/

Raghaven, S., & Mooney, R. J. (2013). *Online inference-rule learning from natural-language extractions*. The University of Texas.

Shanahan, P. (2018). *DOD cloud strategy*. Department of Defense.

Shaw, M. (2019, October 15). *Why Google is the best search engine (and why businesses should care)*. Tower Marketing. https://www.towermarketing.net/blog/google-best-search-engine/

Shin, J., Wu, S., Wang, F., De Sa, C., Zhang, C., & Ré, C. (2015). Incremental Knowledge Base Construction Using DeepDive. Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases, 8(11), 1310–1321. https://doi.org/10.14778/2809974.2809991 http://sites.computer.org/debull/A14sept/p26.pdf

Sievo. (2019, November 16). *AI in procurement*. https://sievo.com/resources/ai-in-procurement

Von Bell, M. (2015). Highlights from the History of Graph Theory [pp. 7-8].

Vu, M., & Thai, M. T. (2020). Pgm-explainer: Probabilistic graphical model explanations for graph and neural networks. *Advances in neural information processing systems*, *33*, 12225-12235. Accessed: https://proceedings.neurips.cc/paper/2020/file/8fb134f258b1f7865a6ab2d935a897c9-Paper.pdf

Weikum, G., Dong, XL., Rzaniewski, S., & Suchanek, F. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. Foundations and Trends in Databases. NOW Publishers. 23 August 2021 https://arxiv.org/pdf/2009.11564.pdf

Wilson, Robin J. (Published online 17 Dec 2013). History of Graph Theory, from *Handbook of Graph Theory*, CRC Press.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T. & Weinberger, K. (2019). Simplifying Graph Convolutional Networks. Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research 97:6861-6871 https://proceedings.mlr.press/v97/wu19e.html

Zarkadakis, G. (2019, September 11). *The rise of the conscious machines: How far should we take AI?* Science Focus. https://www.sciencefocus.com/future-technology/the-rise-of-the-conscious-machines-how-far-should-we-take-ai/

Zhang, Z., Wang, X., & Zhu, W. (2021). Automated machine learning on graphs: A survey. arXiv preprint arXiv:2103.00742. Accessed: https://arxiv.org/pdf/2103.00742.pdf